Rendering Híbrido: Mapeamento de
Volumes sobre Superfícies

Danilo Medeiros Eler

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 25 de maio de 2006

Assinatura: _____

Rendering Híbrido: Mapeamento de Volumes sobre Superfícies

Danilo Medeiros Eler

Orientador: *Prof^a. Dr^a. Maria Cristina Ferreira de Oliveira*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC/USP como parte dos requisitos para obtenção do título de Mestre em Ciência de Computação e Matemática Computacional.

USP - São Carlos
Maio/2006

Agradecimentos

A partir de quando devo começar o agradecimento da conclusão desse mestrado? Algumas coisas do passado podem alterar o rumo de uma vida toda. É por isso que eu agradeço em primeiro lugar a Deus, por ter restaurado a minha comunhão com Ele por meio de Jesus, que se entregou por mim. Por isso estou hoje aqui onde estou.

Agradeço a Deus pela provisão nesses momentos difíceis, agradeço por me ajudar a descansar e não desistir, pois foi difícil saber se era para continuar. Agradeço por tudo, pois creio que tudo que está na vontade de Deus coopera para o meu bem (até minha operação de apêndice).

Agradeço a meus pais (Rubens e Glória) pela educação e amor que me dão até hoje. Pelo apoio e orientação constante em várias áreas da minha vida. São grandes exemplos que procuro seguir. Amo vocês.

Agradeço a meus amados irmãos, Daniela e Marcelo. Sempre se importando comigo. Marcelo, meu companheiro de quarto desde o dia que nasceu. Que bom não termos nos separado. Você é um grande companheiro. Admiro-te muito. E agradeço por todos os momentos.

Agradeço à minha namorada, Juliana. Que está lá em Martinópolis, esperando com paciência, saudades e torcendo sempre por mim. Sempre me dando amor e carinho por msn, telefone, email e muitos outros meios de comunicação. Acabou Ju!!!

Agradeço à Cristina por ter me aceitado como aluno, ter depositado sua confiança e ter me ensinado muitas coisas. Apertando nos momentos difíceis do começo e dando liberdade na hora que foi necessário. Agradeço pelo exemplo de competência, capacidade e seriedade que me foi passado.

Agradeço ao Professor Gustavo, um grande professor que nos ajudou a direcionar o projeto, apressar os prazos, aparar defeitos e a melhorar as idéias. E idéia é uma coisa que ele tem sobrando!!!

Agradeço ao Rogério, grande professor e amigo, que mais uma vez foi usado para contribuir na minha formação. A primeira foi na faculdade, a segunda me indicando para um emprego e agora me direcionando a tomar uma decisão para o mestrado.

Agradeço a todos meus familiares e amigos que torceram por mim e sempre deram apoio. E a toda a igreja lá de Martinópolis que sempre oraram por mim!!!

Agradeço ao Caio, amigo da época da faculdade e agora companheiro de apartamento. Valeu pela amizade, pelos momentos de diversão, pelas comidas, por correr atrás dos nossos compromissos com a imobiliária e por tudo mais. Grande amigo!!!

Agradeço a todos amigos que fiz aqui e em especial aos da turma PgCompUSP04. Amigos que ajudaram a dar alegria nos momentos de aperto das cruéis disciplinas, ajudaram a manter a saúde jogando um futebol e participando da Volta USP, ajudaram a me manter atualizado sobre os sabores de pizza e em muitos outros momentos divertidos. Foi muito bom conhecer todos vocês.

Agradeço a todos os funcionários do ICMC; os professores que mostraram ser grandes conhecedores da área em que atuam; às secretárias da pós, sempre pacientes e atenciosas; aos guardas da portaria, sempre mostrando companheirismo; ao pessoal da biblioteca; aos funcionários da limpeza, sem eles seria difícil; e a todos do bandeirão, comida muito boa.

Agradeço também ao CNPq pelo auxílio financeiro que foi de grande importância para o bom andamento do mestrado.

*Todas as coisas contribuem para o bem daqueles que amam a Deus,
daqueles que são chamados por seu decreto.*
(Romanos 8:28)

Algoritmos para *rendering* de superfícies são rápidos, mas não são compatíveis com situações em que é necessário investigar estruturas internas em volumes. Algoritmos de *rendering* volumétrico direto são adequados para a exploração de estruturas volumétricas, mas são lentos quando comparados a um *rendering de superfícies*. Várias soluções híbridas foram propostas na literatura, sendo que uma delas, conhecida como VoS (*Volume on Surface*), foi proposta recentemente com o objetivo de aumentar a capacidade de investigação do conteúdo de volumes por meio de superfícies. VoS é uma técnica híbrida que permite mapear o conteúdo de um volume em superfícies extraídas do mesmo. A técnica executa lançamento de raios para mapear as informações do volume na superfície, possibilitando a visualização das estruturas internas do volume utilizando *rendering* de superfícies convencional.

No presente trabalho estudamos a técnica VoS e propomos diversas modificações com o intuito de generalizar a técnica e tratar algumas de suas limitações. As novas soluções apresentadas permitem a utilização da técnica com volumes de voxels regulares, e geram imagens de melhor qualidade. Assim como a VoS, as duas novas versões implementadas, VoS^m e VoS^{m*}, têm o objetivo de melhorar o poder de investigação do *rendering* de superfícies, permitindo a exploração do conteúdo de volumes. A técnica VoS e suas variações oferecem uma ferramenta alternativa para aplicações em que a utilização de superfícies é uma solução natural.

Abstract

Surface rendering algorithms are fast, but are not suitable in situations where internal volume structures must be displayed for investigation. On the other hand, Direct Volume Rendering algorithms are effective to support exploration of internal volume structures, but software implementations are slow as compared to surface rendering solutions. Several hybrid solutions have been proposed in the literature. One of such hybrid solutions, named as VoS (Volume on Surface), has been recently introduced with the goal of using surfaces to enhance volume investigation capability. VoS is a hybrid technique that maps volume contents to surfaces extracted from this volume. The technique performs ray casting to map the volume information onto the surface, thus enabling the visualization of internal volume structures using standard surface rendering algorithms.

In this work we study the VoS technique and propose several modifications in order to generalize the technique and treat some of its limitations. The new solutions presented here enable applying the technique to volumes described as a regular grid of voxels and produce images of superior quality as compared to the original. As with VoS, the two novel implementations, VoS^m and VoS^{m*} , have the goal of improving the investigative power of a surface rendering display, supporting exploration of volumetric contents. VoS and its variations are alternative tools for applications where surface rendering is a natural visualization solution.

Sumário

Resumo	i
Abstract	ii
1 Introdução	1
2 Visualização Volumétrica	3
2.1 Ajuste de superfícies	5
2.2 <i>Rendering</i> Volumétrico Direto	7
2.3 <i>Rendering</i> Híbrido	10
3 A Técnica VoS^m	16
3.1 Visão Geral	17
3.2 Organização do Volume	19
3.3 Preparação do Volume	19
3.4 Preparação da Superfície	21
3.5 Pré-Visualização	22
3.5.1 Lançamento de Raios e Armazenamento das Amostras	23
3.5.2 Classificação das Amostras	23
3.5.3 Composição das Amostras	25
3.6 <i>Rendering</i> de Superfícies	26
4 Resultados do VoS^m	27
4.1 Comparação das Imagens com Imagens do Ray Casting Convencional	28
4.2 Tempos do Pré-Processamento e Atualização das Funções de Transferência	33
4.3 Comparação com <i>Rendering</i> de Superfícies Convencional	34
4.4 Número de Cones versus Qualidade das Imagens	36
4.5 Função de Transferência Baseada na Posição da Amostra no Raio	36
4.6 Efeitos do Limiar	39
4.7 Consumo de Memória	41
5 A Técnica VoS^{m*} e Resultados	42
5.1 Diferenças entre VoS ^m e VoS ^{m*}	42
5.2 Resultados	43
5.2.1 Tempos de Atualização das Funções de Transferência	43

5.2.2	Tempos de Interação com o Usuário	44
5.3	Consumo de Memória	44
5.3.1	Comparação com Imagens do VoS^m e do <i>Ray Casting</i> Convencional	45
6	Conclusão	51
6.1	Conclusão	51
6.2	Trabalhos Futuros	52
A	Utilizando as Classes Desenvolvidas	57
A.1	Classe SurfaceData	57
A.2	Classe ImageData	58
A.3	Classe ColorTF	60
A.4	Classe OpacityTF	61
A.5	Classe DistanceTF	61
A.6	Classe Ray Casting	62
A.7	Exemplo de Utilização das Classes	63

Lista de Figuras

2.1	<i>Marching Cubes</i> : configurações básicas para triangulação de uma célula	6
2.2	Footprints projetados no plano de imagem	8
2.3	Disparo de um raio no interior do volume	9
2.4	Distribuição de câmeras ao redor do volume	12
2.5	Cone de raios	13
2.6	Imagem de uma castanha gerada pelo VoS (Cateriano e Nonato, 2003)	14
3.1	Visualização de uma cabeça.	17
3.2	Idéia geral do VoS ^m	18
3.3	VoS ^m pipeline.	19
3.4	Malha Regular	20
3.5	Máscaras de convolução tridimensionais e cálculo do Sobel 3D	20
3.6	Formato padrão do arquivo com informações da superfície.	21
3.7	Superfície formada por pelo menos duas componentes: pulmões e o corpo.	22
3.8	Três cones virtuais com diferentes ângulos de abertura e as linhas de observação definindo as direções de visualização amostradas.	23
3.9	Amostras armazenadas	24
3.10	Especificação de uma função de transferência de cor.	24
3.11	Função de Transferência Baseada na Posição da Amostra	25
4.1	Volume do Mamão: Comparação entre DVR e VoS ^m	29
4.2	Configurações das funções de transferência de opacidade para o volume do mamão	29
4.3	Volume do Tórax: Comparação entre DVR e VoS ^m	30
4.4	Configurações das funções de transferência de opacidade para o volume do tórax	30
4.5	Volume do Tórax (pulmão)	31
4.6	Configuração da função de transferência de opacidade para o volume do tórax (pulmão)	31
4.7	Volume do Pé: comparação entre DVR e VoS ^m	32
4.8	Configuração da função de transferência de opacidade para o volume do pé	32
4.9	Imagens VoS ^m e <i>Rendering</i> de Superfície	35
4.10	Comparação de imagens de diferentes configurações de cones do VoS ^m	37
4.11	Função de transferência baseada na posição da amostra no raio: mamão	38
4.12	Configurações da função de transferência baseada na posição	38
4.13	Imagens 2D orientando escolha do limiar	39
4.14	Efeitos do limiar: tórax, pé e mamão	40

5.1	Imagens intermediárias geradas durante a interação do usuário (esquerda) e respectivas imagens finais (direita).	46
5.2	Volume do mamão: comparação entre as imagens do VOS^m (linha superior), do VOS^{m*} (linha intermediária) e <i>Ray Casting</i> (linha inferior).	47
5.3	Volume do tórax: comparação entre as imagens do VOS^m (linha superior), do VOS^{m*} (linha intermediária) e <i>Ray Casting</i> (linha inferior).	48
5.4	Volume do pé: comparação entre as imagens do VOS^m (linha superior), do VOS^{m*} (linha intermediária) e <i>Ray Casting</i> (linha inferior).	49
5.5	Volume da cabeça: comparação entre as imagens do VOS^{m*} (linha superior) e do <i>Ray Casting</i> (linha inferior).	50

Lista de Tabelas

4.1	Comparação entre os tempos de processamento para três configurações distintas do VOS^m aplicadas no volume do tórax	33
4.2	Comparação entre os tempos de processamento para três configurações distintas do VOS^m aplicadas no volume do pé	34
4.3	Comparação entre os tempos de processamento entre três configurações distintas do VOS^m aplicadas no volume do mamão	34
4.4	Comparação da interação de três configurações do VOS^m para os volumes do mamão, do tórax e do pé.	36
4.5	Quantidade de memória consumida pelo VOS^m e pelo volume.	41
5.1	Tempos de atualização de função de transferência no VOS^{m*}	44
5.2	Tempos de interação do usuário com a superfície no VOS^{m*}	44
5.3	Quantidade de memória consumida pelo VOS^{m*}	44

Introdução

A Visualização Volumétrica emergiu na década de 90 como um conjunto de técnicas para criar representações visuais do conteúdo de volumes de dados. As técnicas da Visualização Volumétrica viabilizam a exploração interativa do interior de volumes, possuindo inúmeras aplicações, já que muitas áreas de investigação de fenômenos físicos geram dados tridimensionais, e têm na visualização uma ferramenta de trabalho imprescindível. Modelos volumétricos representam explicitamente as estruturas internas de um objeto. Um modelo volumétrico é tipicamente representado como uma malha tridimensional de elementos primitivos, os quais são associados a um ou múltiplos valores de informação. Essas malhas podem ser regulares, formadas por unidades de volume cúbicas (os *voxels*); ou podem ser descritas por malhas irregulares em que os elementos básicos são células com características mais gerais do que os *voxels*.

Existem duas principais categorias de técnicas de visualização volumétrica de volumes de dados escalares, conhecidas como técnicas de ajuste de superfícies (*Surface-Fitting*) e técnicas de *rendering* volumétrico direto (DVR - *Direct Volume Rendering*). A primeira categoria inclui técnicas que buscam ajustar um modelo geométrico aos dados, a partir de parâmetros fornecidos pelo usuário. A segunda categoria inclui técnicas que geram imagens diretamente do volume de dados, também utilizando informações do usuário, mas sem criar uma representação geométrica intermediária. Nesse caso, um modelo óptico simula, de forma aproximada, o processo de interação da luz com o volume de dados, e é necessário estabelecer uma relação entre a intensidade/opacidade de elementos do volume ao interagir com a luz incidente, e a intensidade/opacidade de pixels na imagem resultante.

De modo geral, as maneiras de visualizar dados volumétricos variam de acordo com os requisitos do usuário e a complexidade dos dados. Isso leva a constantes atualizações das técnicas de visualização volumétrica empregadas em tais dados, que por sua vez motivam novas otimizações dos métodos de visualização, que resultam em algoritmos híbridos, que integram duas ou mais técnicas ou abordagens de *rendering*, buscando incorporar em uma única solução as vantagens de diferentes técnicas ou abordagens.

Uma solução híbrida, denominada VoS, que mescla as vantagens do ajuste de superfície e do *rendering* volumétrico direto foi apresentada recentemente por Cateriano e Nonato (Cateriano e Nonato, 2003). Essa solução mapeia o conteúdo de um volume, descrito por uma malha de tetraedros, sobre superfícies extraídas do mesmo. Para isso são criados cones de raios nas faces da superfície com o intuito de criar diferentes direções para os raios que serão lançados para dentro do volume, tendo como origem de lançamento as faces da superfície. Para cada raio disparado é armazenada a cor calculada ao fim do processo de composição das amostras colhidas ao longo de sua trajetória. Em uma etapa de *rendering* de superfície, a cor do raio cuja direção melhor aproxima a direção do raio vindo do observador é atribuída à face a qual o raio pertence. Com isso, não é necessário executar novo lançamento de raios quando o usuário interage com a superfície.

Essa solução é tema de investigação do presente trabalho, o qual relata as modificações realizadas na técnica VoS com o intuito de melhorar a qualidade das imagens geradas, bem como de flexibilizar o processo. Assim como VoS, pretendemos promover um *rendering* de superfícies melhorado, aumentando o poder de investigação do conteúdo do volume por meio de superfícies.

Iniciamos a apresentação dessa dissertação no Capítulo 2 com uma breve descrição das principais técnicas utilizadas na visualização volumétrica. Em seguida, no Capítulo 3, apresentamos uma versão modificada da técnica VoS, denominada VoS^m, e as razões para fazer tais modificações. No Capítulo 4 apresentamos os resultados da técnica VoS^m, apresentamos comparações das imagens geradas pelo VoS^m com imagens obtidas com um algoritmo de *Ray Casting* convencional, tempos de pré-processamento e de composição das amostras, e tempos de interação do usuário com a superfície. No Capítulo 5 apresentamos uma versão modificada da técnica VoS^m, utilizadas com o objetivo de melhorar a qualidade das imagens e tempos de interação do usuário. Também apresentamos tempos de pré-processamento e de composição das amostras, e comparações entre imagens obtidas com a técnica VoS^{m*}, *Ray Casting* e VoS^m. Finalmente, no Capítulo 6, são apresentadas as conclusões do trabalho e sugestões para futuras investigações.

Visualização Volumétrica

A Visualização Volumétrica engloba um conjunto de técnicas para criar representações visuais do conteúdo de um volume de dados. Muitas áreas de investigação de fenômenos físicos, que geram dados tridimensionais, têm na visualização volumétrica uma ferramenta de trabalho imprescindível, visto que suas técnicas viabilizam a exploração interativa do interior de volumes. Modelos volumétricos representam explicitamente as estruturas internas de um objeto. Um modelo volumétrico é tipicamente representado por uma malha tridimensional de elementos primitivos, os quais são associados a um ou mais valores escalares ou vetoriais. Essas malhas podem ser regulares, em geral formadas por unidades de volume cúbicas (os *voxels*); ou podem ser descritas por malhas irregulares em que os elementos básicos são células com características mais gerais do que os *voxels*. Células podem adotar diferentes formatos geométricos, como paralelepípedos ou tetraedros, sendo que elementos de dados são associados aos vértices da célula. Muitas técnicas tradicionais de *rendering* volumétrico, tanto direto como por extração de superfícies, operam sobre malhas 3D regulares constituídas de *voxels*. Entretanto, muitas técnicas vêm sendo criadas ou adaptadas para operar sobre malhas irregulares formadas por tetraedros, ou outros elementos, bem como nuvens de pontos.

Existem duas principais categorias de técnicas de visualização volumétrica de dados escalares, conhecidas como técnicas de ajuste de superfícies (*Surface-Fitting*) e técnicas de *rendering* volumétrico direto (DVR - *Direct Volume Rendering*). A primeira categoria inclui técnicas que buscam ajustar um modelo geométrico aos dados, a partir de parâmetros fornecidos pelo usuário. Por exemplo, na clássica técnica *marching cubes* (Lorensen e Cline, 1987) o usuário define um valor de interesse, e a técnica busca gerar uma malha de triângulos que aproxima a superfície associada

a esse valor no conjunto de dados. Imagens do volume serão criadas a partir dessa representação geométrica intermediária. A segunda categoria inclui técnicas que geram imagens diretamente a partir do volume de dados, também utilizando informações do usuário, mas sem criar uma representação geométrica intermediária. Nesse caso, um modelo óptico simula, de forma aproximada, o processo de interação da luz com o volume de dados, e é necessário estabelecer uma relação entre a intensidade/opacidade de elementos do volume ao interagir com a luz incidente, e a intensidade/opacidade de pixels na imagem resultante.

A vantagem mais importante das técnicas de ajuste de superfícies é a compactação do volume de dados, a facilidade de implementação e a rapidez com que esses algoritmos operam quando se utiliza placas gráficas convencionais. Como pontos fracos, no processo de ajuste podem ser introduzidos artefatos que não estão presentes nos dados; estruturas contidas nos dados podem não ser visualizadas com clareza, estruturas muito pequenas que estão presentes nos dados são difíceis de serem extraídas, e as estruturas internas a uma superfície selecionada são escondidas, obscurecendo informações importantes sobre os dados. Ao contrário das técnicas de ajuste de superfícies, as técnicas de *rendering* volumétrico direto buscam incorporar todas as estruturas contidas no volume em uma imagem do mesmo. Variando as funções de transferência, que estabelecem as variações entre os materiais do volume e a luz incidente, as estruturas do volume podem ser escondidas ou destacadas, tornando as imagens mais intuitivas e reais. Essas mesmas vantagens geram os principais pontos fracos do *rendering* volumétrico direto, são eles, a dificuldade de encontrar uma boa função de transferência, que geralmente é buscada por tentativa e erro, e o aumento do custo computacional, que muitas vezes prejudica ou inviabiliza a interatividade. No entanto, pesquisas vêm sendo desenvolvidas para amenizar esses pontos fracos, alguns deles são a criação de métodos automáticos e semi-automáticos para especificação de funções de transferência, a utilização de placas gráfica dedicadas para *rendering* volumétrico, otimizações dos algoritmos e utilização de estratégias de programação.

As formas de visualizar dados volumétricos variam de acordo com os requisitos do usuário e a complexidade dos dados, o que motiva constantes atualizações dos algoritmos e técnicas de visualização volumétrica empregados na análise e manipulação de tais dados. Uma nova tendência de otimização dos métodos de visualização tem focado na investigação de algoritmos híbridos. Tradicionalmente, o processo de visualização adota uma única abordagem de *rendering*, ou *rendering* de superfícies extraídas do volume, ou uma técnica de DVR. Embora a adoção de uma única técnica possa oferecer ao usuário uma boa compreensão do conteúdo do volume, isso pode não ser suficiente para muitos problemas. O *rendering* híbrido busca combinar duas ou mais técnicas ou abordagens de *rendering* para extrair mais informações e informações mais significativas (Hauser et al., 2000). Outra meta das abordagens híbridas é diminuir o custo do processo, permitindo executar *rendering* volumétrico em tempo real (Cateriano e Nonato, 2003).

A conceituação do termo “híbrido” é muito ampla, em um levantamento realizado por Cateriano e Nonato os algoritmos híbridos foram classificados em três tipos:

- Hibridização segundo o tipo de *rendering*: nessa categoria estão algoritmos que utilizam mais de uma técnica de *rendering* volumétrico;
- Hibridização segundo o domínio do volume: nessa categoria estão os algoritmos que hibridizam técnicas volumétricas em diferentes espaços de visualização, combinando técnicas de projeção;
- Hibridização segundo a arquitetura de computação: nessa categoria estão os algoritmos que utilizam arquiteturas especializadas para desenvolver técnicas tanto sequenciais quanto paralelas.

A seguir, é apresentada a técnica *Marching Cubes* de ajuste de superfícies, bem como técnicas clássicas de *rendering* volumétrico direto, e algumas técnicas de *rendering* híbrido. O presente trabalho é baseado na técnica proposta por Cateriano e Nonato que se encaixa na categoria hibridização segundo o tipo de *rendering*, portanto, apresentamos algumas abordagens que pertencem a essa categoria.

2.1 Ajuste de superfícies

Os algoritmos de ajuste de superfícies utilizam primitivas geométricas para gerar um modelo de objetos de interesse contidos no volume. Tipicamente, os algoritmos ajustam uma superfície, representada por uma malha de polígonos, sobre os elementos de interesse, analisando os elementos do volume em relação a um isovalor escolhido pelo usuário. O algoritmo determina como uma superfície intersecta um elemento, ou uma célula. Para encontrar a intersecção da superfície com a célula é associado o valor 1 para o vértice da célula se o valor do dado associado a esse vértice é maior ou igual ao valor escolhido pelo usuário. Os vértices com valores abaixo do valor escolhido pelo usuário recebem valor 0. A superfície intersecta as arestas nas quais um dos vertices está fora da superfície (valor 1) e o outro vértice está dentro da superfície (valor 0) (Lorensen e Cline, 1987). Após a definição da superfície são utilizados métodos tradicionais de *rendering* de polígonos para a visualização do volume, que são métodos rápidos bem conhecidos, já implementados em placas gráficas convencionais.

Um algoritmo clássico de ajuste de superfícies é o *Marching Cubes*, proposto por Lorensen e Cline (Lorensen e Cline, 1987), que percorre o volume pesquisando quais células contribuem na formação da superfície definida pelo valor limiar de interesse. O *Marching Cubes* trabalha

da seguinte maneira: a partir de dados posicionados em uma malha regular 3D (volumétrica), percorre-se cada célula da malha, formada por oito elementos do volume, identificando se uma parte da superfície passa pela célula. Triângulos são ajustados dentro de cada uma dessas células, dando origem a uma aproximação da superfície procurada. Finalmente, os triângulos são passados para um visualizador de polígonos que os projeta no espaço da imagem.

Este algoritmo considera o número finito de situações distintas em que uma superfície pode interceptar uma célula. Os autores analisaram 256 possíveis configurações de intersecção da superfície com uma célula cúbica, verificando os vértices da célula em relação ao valor do limiar. As 256 configurações possíveis foram reduzidas para 15, por simetria, utilizando reflexão e rotação, e são mostradas na Figura 2.1. Os pontos pretos representam os vértices com valores acima do limiar.

As células que não contém vértices com valores acima e abaixo do limiar são ignoradas, pois não são interceptadas pelas superfícies de interesse. Os oito vértices do cubo são numerados, de 1 a 8, e recebem o valor 1 se tem valores maiores ou iguais do que o limiar, e valor 0 se menores do que o limiar. A sequência de oito valores pode ser representada em um byte, que serve de índice para uma tabela pré-calculada de intersecções dos lados. A posição exata da intersecção da superfície com a célula é determinada mediante interpolação dos valores nos vértices. Assume-se que cada aresta é intersectada uma vez só e, então, no máximo quatro triângulos são suficientes para mostrar o caminho da superfície através da célula. Conjuntos de três pontos, resultantes da intersecção das arestas, da célula com a superfície, são agrupados para formar triângulos.

Figura 2.1: *Marching Cubes*: configurações básicas para triangulação de uma célula (Schroeder et al., 2002)

Situações ambíguas de conexões de pontos em triângulos, geradas pelo algoritmo *Marching Cubes*, podem ser reduzidas com a técnica *Marching Tetrahedra*, que é basicamente a versão do *Marching Cubes* para volumes compostos por tetraedros (cada cubo é subdividido em dois tetraedros). Nessa técnica o número de casos possíveis de intersecções é reduzido para seis. Essas duas técnicas apresentam as já citadas vantagens e desvantagens dos algoritmos de ajuste de superfícies.

2.2 *Rendering* Volumétrico Direto

Os algoritmos de *Rendering* Volumétrico Direto (DVR) são capazes de preservar as características do interior do volume na visualização. Eles criam uma imagem diretamente a partir dos dados volumétricos, sem a necessidade de introduzir primitivas geométricas (Kaufman, 1998). Para isso o volume de dados é varrido, acumulando os valores de opacidades calculados, até chegar no fim do volume ou atingir um valor de opacidade máximo. A forma como o volume é varrido permite agrupar os algoritmos de DVR em duas categorias principais: os que operam na ordem dos objetos e os que operam na ordem da imagem.

Na primeira categoria, a projeção é feita mapeando diretamente o conteúdo do volume sobre o plano da imagem. Isto significa que os *voxels* são projetados na tela em uma ordem determinada. Considerando que o volume de *voxels* já está ordenado, os algoritmos realizam uma varredura dos elementos em ordem crescente ou decrescente com relação à distância do observador. Logo, o algoritmo varre o volume, na ordem especificada, acumulando a contribuição de cor e opacidade de cada *voxel* percorrido para a imagem sendo formada.

Nesta categoria destaca-se o algoritmo *Splatting*, proposto originalmente por Westover (Westover, 1990), que recebe esse nome porque pode ser comparado ao arremesso de uma bola de neve ("o *voxel*") em uma vitrine ("a tela"). A contribuição da neve no centro do ponto de impacto será alta, diminuindo em pontos mais distantes do centro. De forma semelhante, a contribuição de cada *voxel* é maior perto do centro de sua projeção, e menor quanto mais distante do centro. Em um primeiro passo, a fatia mais próxima do plano da imagem é determinada. Os *voxels* dessa fatia serão "atirados" primeiro. O valor de cada *voxel* é classificado de acordo com as funções de transferência de cor e opacidade. Os *voxels* são representados por reconstrução de núcleos 3D, normalmente núcleos gaussianos com amplitudes escaladas pelos valores dos *voxels*. A integração destes núcleos ao longo da linha de observação resulta na construção de blocos chamados *footprints*. O mapeamento para o plano da imagem ocorre pela superposição dos *footprints* sobrecarregados pelos valores dos *voxels*, que formam a imagem no plano de observação, como ilustrado na Figura 2.2. O processo pode ser acelerado mantendo uma *lookup table* de *footprints* pré-computados. A técnica pode produzir imagens de alta qualidade, sendo que uma vantagem sobre outros algorit-

mos é que somente *voxels* relevantes para a imagem devem ser projetados e rasterizados, reduzindo consideravelmente o volume de dados a ser processado e armazenado (Mueller et al., 1999).

Na outra categoria, ordem da imagem, a projeção é realizada pixel a pixel. São lançados raios, partindo do observador para cada pixel da imagem, em direção ao volume. Quando o volume é atingido o raio o atravessa, acumulando cor e opacidade para cada *voxel* percorrido, determinando assim, a cor final do pixel. Nesta categoria destaca-se o algoritmo *Ray Casting* (Levoy, 1988).

Figura 2.2: Footprints projetados no plano de imagem (Westenberg e Roerdink, 1999)

O algoritmo *Ray Casting* permite a geração de imagens de alta qualidade. É uma técnica que permite a visualização de detalhes internos do volume com grande precisão. Por meio da especificação de funções de transferência adequadas o usuário pode destacar e esconder estruturas durante a visualização.

O algoritmo inicia com raios lançados a partir de cada pixel da imagem. Para projeções paralelas, todos os raios são paralelos à direção de observação. Para projeções perspectivas, os raios são lançados a partir do ponto do observador. Quando um raio atinge o volume é realizada uma amostragem em intervalos regulares ao longo do volume, formando o caminho do raio, como ilustrado na Figura 2.3. Para determinar o valor correspondente a cada ponto amostrado ao longo do caminho do raio, tipicamente é realizada uma interpolação tri-linear dos valores associados aos vértices do *voxel*. Em casos mais simples, é considerado um valor único para qualquer ponto do *voxel*. Em cada ponto ao longo do caminho do raio é calculada a sua contribuição de cor e opacidade. As amostras obtidas na interpolação são classificadas segundo as funções de transferência especificadas, as quais são responsáveis por atribuir cor e opacidade às amostras. Em seguida, para

cada amostra tomada no caminho do raio é computado o gradiente na direção normal e um modelo de iluminação é utilizado para criar a ilusão de profundidade, realçar bordas e características do volume na visualização.

Figura 2.3: Disparo de um raio no interior do volume (de Oliveira e Minghim, 1997)

O caminho de um raio é interrompido quando ele chega ao fim do volume ou quando a opacidade acumulada atinge um valor pré-estabelecido, fazendo com que os *voxels* mais distantes não contribuam para a cor final do raio. As contribuições das amostras calculadas ao longo do caminho do raio são compostas para determinar o valor final do pixel interceptado por ele. A composição das amostras pode ser feita pelo somatório da cor e opacidade, ou pelo produtório da cor e opacidade, entre outras maneiras. Uma abordagem, denominada Projeção de Intensidade Máxima (MIP - *Maximum Intensity Projection*), ao invés de compor as amostras encontradas ao longo do raio, utiliza a intensidade máxima encontrada no caminho do raio para determinar a cor do pixel correspondente ao raio lançado.

Uma etapa importante no algoritmo *Ray Casting* é a classificação dos dados que tem por finalidade identificar as estruturas internas do volume. As estruturas são identificadas de acordo com as funções de transferência, por meio das quais são definidas cor e opacidade para as estruturas do volume. As funções de transferência de cor são usadas para mapear valores de dados para cores significativas; as funções de transferência de opacidade permitem destacar as estruturas mais interessantes do volume para o usuário e tornar transparentes as estruturas de menor interesse. Um exemplo seria utilizar as funções de transferência em um conjunto de dados médicos para configurar os valores de dados que representam os ossos como branco e opaco, os valores de dados que representam os músculos como vermelho e semi-transparente, e os valores de dados que representam gordura como bege e mais transparente.

A tarefa de especificar funções de transferência que gerem imagens de qualidade e transmitam as informações requeridas não é trivial. A dificuldade em especificar funções de transferência motivou pesquisadores a criar métodos para facilitar essa etapa do *rendering* volumétrico direto.

Métodos de especificação de funções de transferência podem ser classificados como *tentativa e erro*, *orientados a dados* e *orientados a imagens*, segundo a classificação proposta por Pfister et al. (Pfister et al., 2001). Nos métodos por tentativa e erro o usuário especifica manualmente as tabelas de cor e opacidade. Isso pode ser feito de diversas maneiras, mas o mais usual é permitir que pontos de controle, para os quais o usuário especifica a cor e a opacidade, sejam inseridos e removidos das tabelas. Depois, uma função de interpolação é usada para preencher todos as posições intermediárias da tabela. Métodos orientados a dados (Bajaj et al., 1997), (Kindlmann e Durkin, 1998), (Kniss et al., 2001) são aqueles que extraem informações dos dados volumétricos, as quais restringem o espaço das funções de transferência ou guiam o usuário durante o processo de escolha dos parâmetros da função. Métodos orientados a imagens ((He et al., 1996), (Marks et al., 1997), (König e Gröller, 2001)) utilizam imagens geradas por *rendering* volumétrico direto para auxiliar o usuário na alteração dos parâmetros da função de transferência, até encontrar uma função satisfatória, i.e., que gere uma boa visualização para a tarefa do usuário.

O algoritmo *Ray Casting* gera imagens de boa qualidade, mas o custo computacional envolvido é muito alto, dependendo do tamanho da imagem final. A proposta inicial do *Ray Casting* gerou inúmeras variações e otimizações. As principais alterações adaptam o algoritmo para outros tipos de células que não são cúbicas, mudam a forma como o raio percorre o volume e caminha nas células, e as estruturas de dados utilizadas pelo algoritmo. Mas, mesmo com as otimizações já existentes a maioria dos algoritmos sequenciais não consegue gerar a visualização volumétrica em tempo real. Implementações mais rápidas desse método podem ser obtidas combinando diversas estratégias comuns em Computação Gráfica, como finalizar o traçado do raio após a opacidade atingir um determinado limiar (Gordon e Chen, 1991), decompor o volume em uma *octree* (Kaufman, 1997) e explorar a coerência espacial dos objetos do volume.

Embora estas técnicas de aceleração resultem em um ganho significativo no tempo de renderização, o *Ray Casting* ainda é uma técnica muito custosa o que motivou o desenvolvimento de estratégias de programação (Grimm et al., 2004a) (Grimm et al., 2004b) e a utilização de placas gráficas para garantir *rendering* a taxas interativas (Wilson et al., 2002).

2.3 *Rendering* Híbrido

O *rendering* híbrido não é um tema de estudo recente. Em 1990, Marc Levoy (Levoy, 1990) propôs um algoritmo que combinava um volume de dados com um conjunto de polígonos, de modo que eles aparecessem juntos em uma mesma imagem. O algoritmo, denominado *Hybrid Ray Tracing*, considera o problema de estender o *rendering* volumétrico para manipular objetos definidos por polígonos. Na solução proposta os raios são lançados simultaneamente no conjunto de polígonos e no volume de *voxels*. Amostras de cada conjunto são extraídas em intervalos igual-

mente espaçados no caminho do raio, as cores e opacidades resultantes são compostas de acordo com a ordem de profundidade. Para evitar erros na visibilidade das estruturas do volume, dependendo ponto de intersecção entre o polígono e o volume um tratamento especial é aplicado nas amostras do volume que estão imediatamente à frente ou atrás dos polígonos.

Alguns anos depois, Hauser et al. (Hauser et al., 2000) apresentaram uma técnica de *rendering* híbrido chamada *Two-Level Volume Rendering*, que combina técnicas de Projeção da Intensidade Máxima (*Maximum Intensity Projection*) e *Rendering* Volumétrico Direto. Em alguns casos, e.g., aplicações médicas, existe uma grande variação dos valores de dados presentes em uma mesma estrutura, dificultando a tarefa de classificação. Para facilitar a classificação das estruturas, técnicas de segmentação são utilizadas para sub-dividir o volume de dados em objetos. Após a segmentação, o volume é dividido automaticamente em objetos para que o usuário possa manipular cada um separadamente, exibindo ou escondendo um determinado objeto na cena. Para cada objeto, pode-se associar uma técnica de DVR ou MIP, que são processadas separadamente em um processo de *rendering* local, no nível do objeto. Por fim, todos os objetos são compostos para produzir uma imagem final. Esse método apresenta a vantagem de exibir as estruturas ocultas que foram escondidas pelo DVR; estruturas complexas, como veias, são melhor visualizadas com MIP do que com DVR; e, conseqüentemente, viabilizar imagens mais adequadas para uma melhor exploração dos dados, em comparação a uma única técnica de *rendering*.

Uma abordagem apresentada por Chen et al. (Chen et al., 2001) diminui o número de raios lançados na etapa de *rendering*. A idéia geral do algoritmo é mapear a textura de uma imagem renderizada do volume, ou *keyview*, sobre uma superfície extraída do volume para aumentar a velocidade de geração de novas visualizações. Durante uma etapa de pré-processamento é extraída uma superfície a partir do volume de entrada. É extraída a superfície visível mais externa do volume, determinada por uma função de transferência. O iso-valor da superfície é escolhido como o menor valor do voxel com opacidade diferente de zero. Uma vez que a superfície foi extraída do volume e as *keyviews* foram renderizadas, pode-se usá-las para acelerar a geração de novas visualizações. A idéia é mapear a textura das *keyviews* sobre a geometria da superfície.

Para manter a qualidade das imagens duas situações foram consideradas. A primeira é que quando o ponto de observação muda, partes originalmente sobrepostas ou partes ocultas na *keyview* atual poderiam se tornar visíveis. Para essas novas partes visíveis, nenhuma textura está disponível na *keyview* atual, portanto, é necessário disparar novos raios para amostrar as partes que antes estavam invisíveis. Para alcançar isso, o algoritmo detecta a visibilidade de cada polígono na *keyview* quando o mapeamento de textura é executado e os polígonos invisíveis são renderizados com uma cor especial especificada pelo usuário. A segunda situação é que a área de projeção de um polígono visível pode aumentar muito devido à mudança na posição de observação, o que leva a um aumento da textura, resultando em uma imagem borrada (*fuzzy*). Para garantir a qualidade da projeção de

cada polígono visível é calculada a fidelidade de textura, um parâmetro que verifica o aumento da textura. Primeiro, é testado se o triângulo está visível na *keyview*. Se sim, a sua fidelidade de textura é calculada e verifica-se se esta é maior que um certo limiar. Somente passando por esses dois testes, a imagem da *keyview* será mapeada no triângulo; caso contrário, ele estará desenhado com uma cor especial indicando um possível ‘buraco’. Em uma outra etapa, o frame buffer é verificado para localizar onde estão as cores especiais e são disparados novos raios para preencher os buracos.

Uma outra alternativa apresentada por Chen et al. foi pré-processar um grupo de *keyviews* durante o pré-processamento, assim qualquer nova visualização pode ser gerada de uma (ou de um subconjunto) dessas *keyviews*. Assim o sistema nunca tem que disparar raios em todo o volume durante a navegação. Um grande desafio no pré-processamento foi o de encontrar posições ideais para as câmeras das *keyviews*. No caso, foi contruída uma esfera ao redor do volume, e câmeras foram posicionadas na esfera, todas apontando para o centro do objeto, como ilustrado na Figura 2.4.

Figura 2.4: Distribuição de câmeras ao redor do volume, adaptado de Chen et al. (Chen et al., 2001)

Como mais de uma câmera poderia ser utilizada para gerar uma *keyview*, uma opção possível seria renderizar o volume múltiplas vezes utilizando cada câmera e sobrepor os resultados. Mas o método de sobreposição não somente cria uma imagem muito borrada, mas também é muito caro computacionalmente. Ao invés disso, decidiu-se utilizar a *keyview* mais próxima de cada polígono.

Para isso é realizada uma comparação do ângulo entre as normais do polígono e o vetor de observação de cada câmera ao redor do triângulo. Foram testadas somente as 4 câmeras mais próximas de um determinado triângulo, por razão de eficiência.

Um trabalho com idéia semelhante ao trabalho de Chen et al., também com o intuito de diminuir o lançamento de raios durante o *rendering* do volume, foi recentemente proposto por Cateriano e Nonato (Cateriano e Nonato, 2003). Eles propõem uma técnica que agrega as vantagens das abordagens de *rendering* volumétrico direto e ajuste de superfícies, manipulando malhas tetraedrais. A idéia é obter uma pré-visualização direta sobre o bordo do volume de modo a acelerar as transformações das projeções, viabilizando, assim, interações em tempo real. A técnica introduz várias aproximações com o propósito de simplificar o processo de visualização. Primeiramente os dados do volume são armazenados em uma estrutura de dados topológica, que representa explicitamente as informações de incidência e adjacência entre as células (tetraedros) que definem o volume. Isso é feito para tornar possível a identificação de faces e elementos vizinhos de cada célula de forma imediata. A segunda etapa do processo consiste em aplicar uma técnica de *rendering* volumétrico direto, sendo que é utilizada uma versão modificada do algoritmo de *ray casting*. Os raios são lançados para dentro do volume a partir das faces de bordo de uma superfície extraída do volume. Dessa forma, cada face de bordo da superfície é considerada como um plano de projeção. Para cada raio lançado, uma amostragem é feita com base nos valores das células. A cor final de cada face é obtida integrando as contribuições de cor e opacidade de cada célula interceptada pelo raio.

Figura 2.5: Cone de raios: a) os raios e suas direções b) os cones (Cateriano e Nonato, 2003)

Previamente ao lançamento dos raios, são definidos cones virtuais de raios nas faces da superfície, como ilustrado na Figura 2.5, que organizam o lançamento de raios em várias direções no volume. O cone de raios é definido por um conjunto discreto de raios que são gerados com base na normal da face. Utilizando, por exemplo, coordenadas esféricas, podem ser calculadas as direções que os raios seguirão, rotacionando a normal da face em diferentes ângulos. Cada raio lançado percorre o volume e, no seu término, armazena a cor resultante de seu percurso. O processo de

rendering volumétrico direto pode ser totalmente paralelizado, pois se baseia no lançamento de raios, o que pode ser feito em paralelo para múltiplos raios.

A última etapa consiste em aplicar um método de *rendering* de superfícies. Para cada face adota-se a cor associada a um dos raios do cone de luz. É escolhido o raio cuja direção mais se aproxima da direção do observador. Para isso, é realizado um cálculo de produto interno entre os raios e o vetor construído da posição do observador até o objeto. Nessa etapa de *rendering* de superfície não há mais lançamento de raios, somente as informações armazenadas nos raios são processadas. Um novo lançamento de raios é executado somente quando as funções de transferências são modificadas. Esse ambiente de visualização volumétrica híbrida facilita as transformações sobre as superfícies dos objetos, e aproveita melhor os diferentes recursos gráficos do computador, o que permite uma visualização em tempo real após o pré-processamento feito pelo *ray casting*. Com isso, o usuário pode facilmente navegar pela cena e interagir com o modelo. Uma imagem gerada com essa técnica pode ser observada na Figura 2.6.

Figura 2.6: Imagem de uma castanha gerada pelo VOS (Cateriano e Nonato, 2003)

Em outro trabalho recente, Ferre et al. (Ferre et al., 2004) propuseram uma abordagem baseada em Árvores de Decisão. A idéia é permitir um acesso eficiente às informações selecionadas em um modelo híbrido. O primeiro passo do método é construir uma *Rendering Decision Tree* (RDT) baseada na classificação dos *voxels* em regiões, *voxel* interno ou *voxel* de bordo, dividindo o volume em objetos, os quais são inseridos na RDT. Cada *voxel* classificado armazena um rótulo que indica a que folha da RDT ele pertence. Cada nó folha da RDT armazena o rótulo que identifica a qual classe de *rendering* ela está associada. Cada *voxel* pode se enquadrar em três categorias de *rendering*: *voxels* internos, que são renderizados por algoritmos de *rendering* volumétrico direto (no caso, *splatting*); *voxels* de bordo que são renderizados projetando as superfícies que passam por dentro deles; e *voxels* híbridos, em que primeiro são projetadas as superfícies que passam por eles e depois o interior de cada *voxel* é renderizado sobre as superfícies com o algoritmo *splatting*. Após a classificação do volume o modelo está pronto para a interação do usuário. O usuário define

um critério para o processo de *rendering* selecionando os nós da RDT. Após essa seleção a RDT é percorrida da seguinte forma: quando uma folha é selecionada os seus respectivos *voxels* são renderizados, e quando um nó não folha é selecionado todos os seus descendentes, e respectivos *voxels*, são renderizados.

Analisando estes trabalhos, observa-se que os pesquisadores usam a mesma proposta de Levoy (Levoy, 1990), utilizando duas ou mais técnicas de *rendering* em uma mesma visualização. A tendência é aumentar o número de técnicas usadas em uma única visualização e aumentar as possibilidades do usuário interagir com o processo de *rendering*, como feito por Hadwiger (Hadwiger et al., 2003), que estenderam a solução apresentada por Hauser et al. (Hauser et al., 2000) para incorporar técnicas de ajuste de superfícies e *rendering* não fotorealístico. Outra tendência é implantar estratégias de programação e estruturas de dados (e.g. *Octrees*, *Decision Trees*) para melhorar o processo do *rendering* híbrido e, também, a interação do usuário com o modelo visualizado.

Enfim, o *rendering* híbrido está melhorando o desempenho do *rendering* volumétrico, a qualidade das imagens resultantes e o controle e as possibilidades de interferência do usuário no processo de *rendering*. O *rendering* híbrido permite ao usuário destacar mais facilmente objetos em uma cena, aumentando sua intuição sobre os dados em questão. Assim, as técnicas de *rendering* baseadas em abordagens híbridas são promissoras no sentido de que podem gerar soluções que se adequam ainda mais aos objetivos da visualização volumétrica.

A Técnica VoS^m

Como discutido no Capítulo 2, a técnica VoS (Cateriano e Nonato, 2003) oferece uma solução híbrida para *rendering* volumétrico. A implementação da VoS desenvolvida por Cateriano e Nonato é aplicável a dados representados como malhas de tetraedros, tendo sido implementada na linguagem C++ utilizando a biblioteca gráfica OpenGL.

A implementação desenvolvida apresenta algumas deficiências nos recursos para exploração do volume e um dos motivos para essas deficiências é a pouca liberdade para o usuário especificar funções de transferência. O usuário pode selecionar somente três intervalos de intensidades e atribuir um valor de opacidade a cada intervalo. Além de dificultar a exploração do volume a qualidade das imagens é prejudicada. Como podemos observar na Figura 3.1 não é possível destacar com clareza o conteúdo do volume. No ajuste de superfícies há a mesma limitação, o usuário tem a opção de especificar três intervalos de iso-valores. Essa limitada liberdade de interação entre o usuário e os dados do volume não permite uma ampla investigação do seu conteúdo.

Nossa proposta foi de tratar essas deficiências da implementação anterior e aplicar a técnica VoS a volumes regulares, por ser um tipo mais geral de volume e mais utilizado pelos algoritmos de *rendering* volumétrico direto e de ajuste de superfícies. Com isso consegue-se realizar uma avaliação mais abrangente da técnica, utilizando volumes maiores, que não foram explorados por Cateriano e Nonato, e permitindo comparações com outras soluções de *rendering* volumétrico. Nomeamos a técnica VoS modificada de VoS^m. As diferenças do VoS^m em relação à técnica VoS são:

- Volume: na VoS^m os volumes de entrada são malhas regulares;

- Cones: os cones virtuais para o lançamento dos raios são posicionados nos vértices das faces, e não no centro das faces;
- Amostras dos raios: na VOS^m as amostras da trajetória de um raio são armazenadas se satisfazem um limiar pré-estabelecido pelo usuário.

Figura 3.1: Visualização de uma cabeça gerada pela técnica VOS (Cateriano e Nonato, 2003)

Mais detalhes sobre a implementação proposta e desenvolvida neste trabalho são descritos a seguir. Neste trabalho comparamos imagens resultantes da técnica VOS^m com imagens geradas pelo *Ray Casting* tradicional e por *Rendering* de Superfícies convencional, sendo que os resultados são apresentados no Capítulo 4.

3.1 Visão Geral

A idéia geral do algoritmo, que toma como entrada uma superfície S descrita por uma malha poligonal extraída de um volume, é apresentada a seguir: seja f uma face de S , e seja P o plano que contém f , como mostrado na Figura 3.2. A face f pode ser vista de qualquer ponto de P do lado externo de S (como na Figura 3.2(a)). Se S é um objeto transparente, parte do seu volume interno será visto através de f . Dependendo da posição do observador, diferentes estruturas internas podem ser observadas através de f . Evidentemente, o padrão de cores (ou textura) observável mudará de acordo com a mudança do ponto de observação.

Se a textura visível através de f pudesse ser armazenada para todos os possíveis pontos de observação seria possível tentar identificar e associar a textura mais apropriada para f , quando observada de qualquer ponto de observação. Um observador seria capaz de enxergar qualquer estrutura contida em S , processando somente a representação da superfície, desde que o conteúdo

Figura 3.2: a) Região de visualização de f ; b) Amostrando as direções de visualização de f . As setas em destaque ilustram a linha de observação que melhor aproxima a direção do ponto de observação.

adequado fosse mapeado sobre suas faces. Obviamente, não é computacionalmente factível calcular ou armazenar as cores dos vértices para todas as posições e direções de observação possíveis. Assim, o VOS^m opera em um subconjunto amostrado do conjunto de todas as possíveis posições de observação. Uma estratégia de amostragem permite obter um subconjunto das possíveis direções de visualização a partir das quais pode-se observar o interior de S através dos vértices de f .

Para cada face f o algoritmo calcula, para cada direção de visualização amostrada (também chamada de linha de observação), as cores que aproximam a textura em cada face quando observada de uma particular direção. De fato, as cores são pré-calculadas e armazenadas nos vértices, e a cor de uma face é calculada durante o *rendering* da superfície por meio da combinação das cores de seus vértices, como ilustrado na Figura 3.2(b). Quando as cores da face são calculadas o algoritmo de rendering identifica, para cada vértice da face, qual das linhas de observação pré-calculadas melhor se aproxima da linha de observação para aquele vértice. Então, é atribuída à face a média das cores associadas a cada vértice. Finalmente, o algoritmo de rendering exibe somente as faces visíveis.

Resumindo, a técnica opera em duas etapas:

- Pré-visualização (mapeia o conteúdo do volume em uma superfície extraída do mesmo, para um conjunto de direções de observação amostradas);
- Rendering da superfície (projeta e exibe a visualização da superfície).

Anteriormente a estas duas etapas existe uma etapa de preparação do volume e extração da superfície de interesse. O pipeline na Figura 3.3 mostra as etapas da técnica e as operações realizadas em cada etapa, as quais são descritas com mais detalhes a seguir. Também discutimos

alguns aspectos da implementação.

Figura 3.3: VOS^m pipeline.

3.2 Organização do Volume

O volume é formado por um conjunto de imagens 2D, sendo que cada qual representa uma fatia desse volume. As fatias são armazenadas em uma malha regular (Figura 3.4) de dimensões $X \times Y \times Z$, em que X é a resolução horizontal (largura), Y é a resolução vertical (altura) e Z é a quantidade de imagens. As dimensões da malha são fixas durante o processo e não podem ser alteradas pelo usuário.

Cada *pixel* da imagem é associado a um vértice da malha regular, de modo que a cada vértice está associada uma dupla de valores: um é o valor em tons de cinza do pixel e o outro é o valor resultante da aplicação de um filtro passa alta ao volume (discutido a seguir). O nível de cinza é calculado a partir dos valores RGB do *pixel*, para manipularmos um único valor escalar.

3.3 Preparação do Volume

Um filtro passa alta é aplicado ao volume armazenado na malha regular. No caso, foi implementado o algoritmo de Sobel 3D. O algoritmo é aplicado aos valores escalares (níveis de cinza

Figura 3.4: Malha Regular

dos *pixels*, gerados como discutido na Seção 3.2) associados aos vértices da malha. O objetivo é detectar as regiões de transição entre materiais, com base na verificação da vizinhança dos voxels nos quais a máscara de convolução é aplicada. Os valores resultantes do Sobel 3D são utilizados posteriormente no pré-processamento do volume, sendo que somente as amostras que pertencem a intervalos de interesse do usuário são armazenadas. O usuário seleciona o limiar interagindo com imagens do volume, como discutido na Seção 4.6. As máscaras de convolução tridimensionais utilizadas no algoritmo de Sobel 3D são as ilustradas na Figura 3.5.

G_x

-1	<u>0</u>	<u>1</u>		-3	<u>0</u>	<u>3</u>		-1	<u>0</u>	<u>1</u>
-3	<u>0</u>	<u>3</u>		-6	<u>0</u>	<u>6</u>		-3	<u>0</u>	<u>3</u>
-1	<u>0</u>	<u>1</u>		-3	<u>0</u>	<u>3</u>		-1	<u>0</u>	<u>1</u>

G_y

<u>1</u>	<u>3</u>	<u>1</u>		<u>3</u>	<u>6</u>	<u>3</u>		<u>1</u>	<u>3</u>	<u>1</u>
<u>0</u>	<u>0</u>	<u>0</u>		<u>0</u>	<u>0</u>	<u>0</u>		<u>0</u>	<u>0</u>	<u>0</u>
-1	-3	-1		-3	-6	-3		-1	-3	-1

G_z

-1	-3	-1		<u>0</u>	<u>0</u>	<u>0</u>		<u>1</u>	<u>3</u>	<u>1</u>
-3	-6	-3		<u>0</u>	<u>0</u>	<u>0</u>		<u>3</u>	<u>6</u>	<u>3</u>
-1	-3	-1		<u>0</u>	<u>0</u>	<u>0</u>		<u>1</u>	<u>3</u>	<u>1</u>

$$G = \sqrt{G_x * G_x + G_y * G_y + G_z * G_z}$$

Figura 3.5: Máscaras de convolução tridimensionais e cálculo do Sobel 3D

3.4 Preparação da Superfície

Na etapa posterior de pré-visualização, o VOS^m trabalha com o volume completo e com uma superfície extraída do mesmo. A extração da superfície pode ser realizada por qualquer algoritmo, mas o arquivo com as informações da superfície deve seguir o padrão ilustrado na Figura 3.6.

Figura 3.6: Formato padrão do arquivo com informações da superfície.

POINTS é a palavra reservada para indicar o início da seção que descreve as coordenadas dos vértices da superfície. É necessário indicar o número de vértices da superfície (no caso, 9802) e indicar o tipo de dado (no caso, *float*). Mais abaixo, a palavra POLYGONS é reservada para indicar o início da seção que descreve as faces. É necessário indicar quantas faces a superfície contém (no caso, 19600) e indicar o tamanho da matriz que descreve as faces (no caso, 78400). A descrição das faces segue o seguinte padrão: é indicado no início de cada bloco quantos vértices definem uma face (no caso 3, pois são triângulos) e logo após são dados os índices (IDs) dos vértices, cujas coordenadas foram descritas na seção acima. Os vértices são indexados sequencialmente a partir do primeiro, iniciando em zero.

A superfície extraída nem sempre é totalmente conexa, ou seja, pode ser formada por várias superfícies componentes. Na Figura 3.7 observamos uma superfície formada por pelo menos duas componentes desconexas (pulmões e o corpo). Quando o usuário carrega a superfície ele deve indicar qual componente será utilizada nas etapas subsequentes. Atualmente, a escolha da componente pelo usuário é realizada por um processo de tentativa e erro. A seleção de uma única componente permite diminuir o tamanho da superfície utilizada no processo de rendering e diminuir o número de cones gerados e, por consequência, o número de amostras armazenadas.

Figura 3.7: Superfície formada por pelo menos duas componentes: pulmões e o corpo.

Assim que a componente de interesse é indicada pelo usuário, é realizada a orientação das faces para garantir que todas as normais apontem para fora da superfície. Em seguida é iniciado o processo de criação dos cones de luz, que definem as direções de amostragem.

A superfície é carregada em uma estrutura de dados nomeada como OF, desenvolvida por pesquisadores do ICMC/USP. Essa estrutura de dados teve como base trabalhos realizados sobre a estrutura *Corner Table* (Rossignac, 2001) e o *cgal* (Boissonnat et al., 2002), uma biblioteca de algoritmos geométricos.

Com o intuito de amostrar o conjunto de todas as possíveis direções de observação e definir um conjunto de linhas de observação para cada vértice da superfície, o algoritmo VOS^m, assim como o VOS, define cones virtuais com diferentes ângulos de abertura, centrados em cada vértice e alinhados com o vetor normal no vértice, como ilustrado na Figura 3.8. As linhas de observação iniciam no vértice e suas direções são determinadas pela distribuição uniforme das linhas sobre a superfície do cone.

Na implementação atual a VOS^m usa, para cada vértice, um conjunto de cones com ângulos de abertura igualmente espaçados; por exemplo, se três cones são usados eles têm ângulos de abertura igual a 15°, 45° e 75° a partir do vetor normal. As linhas de observação também estão distribuídas uniformemente sobre a superfície dos cones. Por exemplo, se quatro linhas de observação são amostradas, o ângulo entre cada par delas é de 90°, medido no cone.

3.5 Pré-Visualização

A pré-visualização é a etapa responsável pelo lançamento dos raios. Esta etapa toma como entrada uma malha regular e uma superfície extraída do volume. Os passos da pré-visualização

Figura 3.8: Três cones virtuais com diferentes ângulos de abertura e as linhas de observação definindo as direções de visualização amostradas.

são discutidos em detalhes a seguir.

3.5.1 Lançamento de Raios e Armazenamento das Amostras

Esta etapa envolve o lançamento dos raios através do volume nas direções das linhas de observação amostradas. As trajetórias discretas dos raios são calculadas utilizando um algoritmo de *scanline* 3D. Raios são lançados a partir de cada vértice da superfície e atravessam o volume até o fim. Todos os raios de um vértice têm como origem o vértice da malha regular que mais se aproxima da posição do vértice da superfície. Dadas a origem e a direção de um raio a sua trajetória pode ser calculada. O algoritmo de *scanline* 3D calcula, baseado na inclinação do raio, os pontos por onde o raio passaria dentro do volume. Nesse caso, os pontos calculados são os vértices da malha regular. Cada vértice ao longo do caminho do raio, cujo valor resultante da aplicação do filtro passa alta está acima do limiar especificado pelo usuário é armazenado como uma amostra do raio, como ilustrado na Figura 3.9. Cada raio é associado a uma lista contendo as amostras de interesse.

Note que o valor da amostra em cada ponto por onde o raio passa não é obtido por interpolação, pois as amostras são colhidas nos vértices da malha regular que já têm o valor da amostra armazenado.

3.5.2 Classificação das Amostras

As funções de transferência de cor e opacidade são responsáveis pelo cálculo da cor e opacidade associadas às amostras do raio. É permitido ao usuário definir uma função de transferência de cor simples baseada na intensidade da amostra e dois tipos alternativos de função de transferência de opacidade, baseadas, respectivamente, na intensidade da amostra e em informação espacial

Figura 3.9: a) Limiar = 0, todas as amostras são armazenadas; b) Limiar = 500, algumas amostras são armazenadas.

relativa à amostra.

A interface oferecida para a definição das funções de transferência de cor é ilustrada na Figura 3.10, que mostra o histograma dos valores escalares do volume. O usuário especifica uma função de transferência de cor definindo um conjunto de pontos de controle e associando uma cor a cada ponto definido. Cada ponto de controle é associado a um valor escalar, e é definido diretamente sobre o histograma. Na Figura 3.10 observamos um ponto de controle *A* definido sobre o valor escalar 90, ao qual o usuário associou a cor C_A (neste caso, vermelho), e um ponto de controle *B* definido sobre o valor escalar 234, ao qual a cor C_B foi associada (amarelo na figura). A cor para qualquer valor escalar entre os pontos de controle *A* e *B* (ou seja, entre os valores escalares 90 e 234) é calculada por interpolação linear dos valores RGB correspondentes às cores C_A e C_B .

Figura 3.10: Especificação de uma função de transferência de cor.

A especificação de uma função de transferência de opacidade baseada na intensidade do valor escalar é similar. Agora o usuário associa valores de opacidade aos pontos de controle, e a opacidade para qualquer valor escalar entre os pontos já especificados é obtida por interpolação linear

dos valores de opacidades já definidos.

Uma função de transferência de opacidade alternativa, que considera a informação espacial das amostras, é fornecida ao usuário. Ela permite ao usuário associar valores de opacidade com base na posição da amostra ao longo do raio. A idéia é que amostras posicionadas próximas das extremidades do raio (no ponto inicial ou no ponto final) correspondem à interseção do raio com objetos mais internos ou mais externos no volume. O usuário pode então associar valores de opacidade com base na posição da amostras, como exemplificado na Figura 3.11, que mostra dois raios e suas amostras armazenadas (representadas pelos quadrados). Para favorecer a visualização de estruturas internas deve-se associar valores altos de opacidade para as amostras das regiões mais internas dos raios, como mostrado na Figura 3.11(a). Por outro lado, para destacar estruturas dos objetos mais externos deve-se associar valores altos de opacidade às amostras mais próximas das extremidades do raio, como ilustrado na Figura 3.11(b).

Figura 3.11: a) Função de opacidade para destacar estruturas internas; b) Função de opacidade para destacar as extremidades do objeto.

O tamanho do raio é parametrizado no intervalo $[0, 1]$, e a posição da amostra ao longo desse intervalo é determinada a medida em que o lançamento do raio procede. Durante o processo de classificação a posição normalizada da amostra é passada para a função de transferência, o usuário define alguns pontos de controle ao longo do intervalo $[0, 1]$ e define os valores de opacidade para aqueles pontos; opacidades para valores de amostras entre os pontos são obtidas por interpolação linear.

3.5.3 Composição das Amostras

As amostras armazenadas ao longo de um determinado raio são então compostas para determinar a cor do raio. A composição de amostras poderia adotar diversos modelos ópticos que fazem uso de diferentes parâmetros de fontes de luz, materiais e ambientes. Nós seguimos o modelo óptico simples proposto por Levoy (Levoy, 1990), que considera somente a cor e a opacidade das amostras. Este modelo pode ser definido como segue:

$$C = \sum_{i=1}^N \left[c(i) \alpha(i) \prod_{j=1}^i (1 - \alpha(j)) \right]$$

Na equação acima C é a cor final do raio e N é o numero de amostras. $c(i)$ e $\alpha(i)$ representam, respectivamente, a cor e a opacidade da i -ésima amostra e $\alpha(j)$ representa a opacidade da amostra j . Cada vértice da superfície armazenada o valor de C para cada raio disparado do vértice. Se as funções de transferência de cor ou opacidade são modificadas uma nova composição deve ser executada a partir dos vetores de amostras, mas nenhum novo lançamento de raio é realizado.

3.6 *Rendering* de Superfícies

No fim do passo de pré-visualização uma cor final já foi associada e armazenada para cada raio, em outras palavras, para cada linha de observação saindo de cada vértice da superfície. O problema inicial para o passo de *rendering* é decidir a cor apropriada para o vértice quando visto do ponto de observação atual.

Esse problema pode ser resolvido de maneira simples com um cálculo de produto interno. Dados v o vértice, e $R = \{r_1, \dots, r_k\}$ (k = número de cones * numero de raios disparados por cone) o conjunto de vetores unitários das linhas de observação. Seja r_o o vetor unitário construído da posição do observador, O , até o vértice. Suponha que r_o aponta em direção ao observador e $r_i \in R$ é o vetor cujo produto interno $r_i \cdot r_o$ produz o maior valor positivo entre os vetores em R . A cor associada à linha de observação de r_i é assim atribuída como a cor mais apropriada para v , quando observado de O .

Finalmente, após decidir a cor de cada vértice, o *rendering* da superfície é realizado pelo OpenGL. São passados para o OpenGL os pontos que formam cada face da superfície e a sua cor correspondente. A cor da face é determinada pela tonalização gerada pela média das cores de seus vértices. A cor de cada vértice é atualizada sempre que ocorre uma rotação no objeto ou uma mudança na posição do observador.

No Capítulo 4 são apresentados os resultados da técnica VOS^m, incluindo algumas comparações entre as imagens obtidas com o VOS^m e com o *Ray Casting*. São exibidos os tempos de pré-processamento e composição das amostras quando as funções de transferências são atualizadas, e tempos de interação do usuário com o objeto. Também são apresentados os efeitos da escolha de diferentes limiares pelo usuário.

Resultados do VoS^m

A técnica VoS^m foi submetida a diversos testes para verificar a qualidade das imagens e os tempos de geração das mesmas. Para constatar a qualidade das imagens fizemos comparações com imagens geradas pelo *Ray Casting* convencional, geradas utilizando o *VolView* da Kitware (disponível em <http://www.volview.com>). Para verificar os tempos de interação do usuário com o objeto comparamos com um algoritmo de *rendering* de superfícies convencional, utilizando o *rendering* de superfícies do VTK.

Nos testes foram utilizados três volumes de dados. O volume de um mamão, cedido pela EM-BRAPA (Empresa Brasileira de Pesquisa Agropecuária), (obtido com um tomógrafo desenvolvido para aplicações em agricultura) com dimensões 86x42x59, o volume de um tórax, disponível em <http://www9.cs.fau.de/Persons/Roettger/library>, com dimensões 120x120x241 e o volume de um pé, disponibilizado com o *VolView*, com dimensões 102x247x199.

O VoS^m foi codificado na linguagem C++, no ambiente de programação Borland Builder 5.0, e os testes realizados foram executados em um computador Intel Pentium 4 3.2 GHz com 1.0GB de RAM.

Neste capítulo são apresentados os testes realizados e a avaliação dos resultados obtidos. Conclusões sobre a técnica são apresentadas no Capítulo 6. No Capítulo 5 é apresentada uma variação da técnica VoS^m e os resultados associados.

4.1 Comparação das Imagens com Imagens do Ray Casting Convencional

Nesta seção apresentamos comparações entre imagens resultantes do *Ray Casting* convencional e imagens resultantes do VOS^m. A menos que seja explicitada outra configuração, as imagens resultantes do VOS^m foram geradas utilizando 8 cones por vértice e 8 raios por cone, e ao limiar de interesse foi atribuído o valor 0, ou seja, todas as amostras ao longo do caminho do raio são armazenadas. Isso é necessário para conseguir imagens com qualidade mais próxima a das imagens do *Ray Casting*, pois as imagens geradas com o *Ray Casting* utilizam todas as amostras ao longo do caminho do raio, não perdendo informações do volume.

As Figuras 4.1(a) e (c) mostram imagens do volume do mamão geradas pelo Ray Casting convencional e as Figuras 4.1(b) e (d) mostram imagens geradas pelo VOS^m. Observando a Figura 4.1 é possível notar que a qualidade das imagens obtidas com o VOS^m se aproxima da qualidade das imagens geradas pelo *Ray Casting*. Para gerar as imagens da Figura 4.1 a função de transferência de cor foi configurada com a cor amarela para os valores escalares 0 e 130, cor verde para os valores escalares 131 e 140, cor amarela para os valores escalares 141 e 217 e vermelho para os valores escalares 218 e 255. As configurações das funções de transferência de opacidade podem ser observadas na Figura 4.2. As funções de opacidade são diferentes porque no VOS^m o usuário já está na superfície de interesse, não havendo a necessidade de esconder outras estruturas que interferem na visualização. Já na visualização gerada pelo *Ray Casting* é necessário tornar transparentes as estruturas que interferem na visualização.

Os objetos apresentados nas imagens da Figura 4.1 são muito simples, não possuem muitas estruturas e nem grande variação dos valores escalares em uma mesma estrutura. Utilizando um volume mais complexo, como o do tórax (Figura 4.3), notamos que as imagens obtidas com o *Ray Casting*, no geral, possuem qualidade superior a das imagens geradas pelo VOS^m. Para as imagens da Figura 4.3 a função de transferência de cor foi configurada com a cor vermelha para os valores escalares 0 e 115 e com a cor branca para os valores escalares 116 e 255. As configurações das funções de transferência de opacidade podem ser observadas na Figura 4.4.

As superfícies geradas com o VOS^m dão uma boa noção do conteúdo do volume, mas artefatos podem aparecer nas imagens e detalhes pequenos podem ser perdidos. Isso ocorre em parte devido à qualidade da superfície. O resultado do VOS^m é totalmente dependente da qualidade da superfície. E, no caso da Figura 4.3, temos um exemplo em que ondulações na superfície prejudicam a criação dos cones e, por consequência, a qualidade da imagem final.

Figura 4.1: Volume do mamão: (a) e (c) são imagens criadas com Ray Casting convencional; (b) e (d) são imagens do VoS^m.

Figura 4.2: Configurações das funções de transferência de opacidade para o volume do mamão: a) Configuração para o VoS^m (Figura 4.1(b)), b) Configuração para o Ray Casting Convencional (Figura 4.1 (a)), c) Configuração para o Ray Casting Convencional e VoS^m (Figuras 4.1 (c) e (d)).

Figura 4.3: Volume do tórax: (a), (b) e (c) são imagens criadas com VOS^m ; (d), (e) e (f) são imagens criadas com o Ray Casting convencional.

Figura 4.4: Configurações das funções de transferência de opacidade para o volume do tórax: a) Configuração para o VOS^m b) Configuração para o Ray Casting Convencional.

Um exemplo da deficiência do VOS^m na exibição de pequenas estruturas do volume ou de estruturas que necessitam de muito detalhe é mostrado na Figura 4.5 para o mesmo conjunto de dados do tórax, na qual vemos em (a) uma imagem gerada pelo *Ray Casting* destacando os alvéolos do pulmão. Já na Figura 4.5(b) gerada com o VOS^m , os alvéolos não podem ser observados. Para gerar essas visualizações a função de transferência de cor foi configurada com a cor vermelha para os valores escalares 0 e 115 e com a cor branca para os valores escalares 116 e 255. A configuração da função de transferência de opacidade pode ser observada na Figura 4.6.

Figura 4.5: Volume do tórax (pulmão): a) imagem gerada com *Ray Casting* convencional; b) imagem gerada com VOS^m .

Figura 4.6: Configuração da função de transferência de opacidade para o volume do tórax (pulmão). A mesma configuração foi utilizada no VOS^m e no *Ray Casting* Convencional.

O terceiro volume de dados utilizado para testes foi o volume do pé. Na Figura 4.7 também notamos que a qualidade das imagens do *Ray Casting* convencional é bem superior a das imagens geradas pelo VOS^m . Percebe-se que, quando é exigido um nível de detalhe maior, as imagens do VOS não são totalmente confiáveis, apesar de darem uma idéia do conteúdo do volume. Para as imagens da Figura 4.7 a função de transferência de cor foi configurada com a cor vermelha para os

valores escalares 0 e 85 e com a cor branca para os valores escalares 86 e 255. A configuração da função de transferência de opacidade pode ser observada na Figura 4.8.

Figura 4.7: Imagens do volume do pé. Na linha superior estão as imagens do VOS^m e na linha inferior estão as imagens do *Ray Casting*.

Figura 4.8: Configuração da função de transferência de opacidade para o volume do pé. A mesma configuração foi utilizada no VOS^m e no *Ray Casting* Convencional.

A qualidade das imagens resultantes do *Ray Casting* é superior a das imagens geradas pelo VOS^m ; no entanto, o VOS^m atualiza a composição das amostras, quando uma função de transferência é modificada, muito mais rapidamente do que o *Ray Casting* convencional. VOS^m também é superior na interação do usuário com o objeto, o que favorece a interatividade em placas gráficas

convencionais, pois a cada rotação o *Ray Casting* convencional necessita refazer a composição de todas as amostras, tarefa que o VOS^m não precisa realizar, pois já a tem computada e permanentemente armazenada.

Uma outra desvantagem do VOS é na escala dos objetos. Quando o tamanho de um objeto é aumentado o tamanho dos triângulos da superfície também aumenta, o que prejudica a qualidade das imagens resultantes. Porém, o tempo para gerar as novas imagens é o mesmo tempo para gerar imagens com a superfície em seu tamanho original. Já no *Ray Casting* convencional, quando aumentamos o tamanho do objeto a qualidade também diminui e o tempo para gerar as imagens aumenta, pois mais raios são lançados.

4.2 Tempos do Pré-Processamento e Atualização das Funções de Transferência

O tempo para gerar as imagens da Figura 4.3 (volume do tórax) com o *Ray Casting* convencional (implementação do VolView) é de aproximadamente 20 segundos, tempo para o algoritmo compor as amostras dos raios disparados. A superfície extraída desse volume, com iso-valor 80, tem 83.949 vértices e 167.241 faces. Os tempos que VOS^m requer para fazer o pré-processamento (lançamento dos raios) e compor as amostras quando uma função de transferência é alterada são descritos na Tabela 4.1, na qual é feita uma comparação para três configurações distintas do VOS^m. O melhor caso indica que todos os valores escalares estão associados com o valor máximo de opacidade (1.0), logo o algoritmo irá compor somente uma amostra; o pior caso indica que todos os valores escalares têm valor de opacidade zero, logo o algoritmo irá compor todas as amostras armazenadas para o raio.

	Tempos		
	3 cones x 8 raios	6 cones x 8 raios	8 cones x 8 raios
Pré-Processamento	46s	1min 40s	2min 10s
Melhor Caso	0.38s	0.7s	0.95s
Pior Caso	4.6s	7.1s	9.3s

Tabela 4.1: Comparação entre os tempos de processamento para três configurações distintas do VOS^m aplicadas no volume do tórax

Utilizando o volume do pé, que é um volume maior do que o do tórax, obtivemos os tempos exibidos na Tabela 4.2, para as mesmas configurações de cones. A superfície extraída do volume do pé possui 115.942 vértices e 231.846 faces. O número médio de amostras armazenadas por raio no volume do pé é menor do que o número médio de amostras por raio no volume do tórax, por

isso a atualização das funções de transferência no volume do pé é mais rápida.

	Tempos		
	3 cones x 8 raios	6 cones x 8 raios	8 cones x 8 raios
Pré-Processamento	1min 8s	2min 20s	3min 10s
Melhor Caso	0.4s	0.7s	1.5s
Pior Caso	0.9s	1.7s	2.7s

Tabela 4.2: Comparação entre os tempos de processamento para três configurações distintas do VOS^m aplicadas no volume do pé

Comparando os tempos obtidos para um volume menor, mostramos na Tabela 4.3 os tempos dessas mesmas três configurações do VOS^m aplicadas ao volume do mamão. O volume do mamão tem dimensões 86x42x59 e uma superfície extraída desse volume, com iso-valor 135, possui 14.470 vértices e 28.926 faces.

	Tempos		
	3 cones x 8 raios	6 cones x 8 raios	8 cones x 8 raios
Pré-Processamento	2.6s	5.1s	7.4s
Melhor Caso	0.06s	0.1s	0.16s
Pior Caso	0.32s	0.47s	0.73s

Tabela 4.3: Comparação entre os tempos de processamento entre três configurações distintas do VOS^m aplicadas no volume do mamão

4.3 Comparação com *Rendering* de Superfícies Convencional

A comparação que podemos fazer entre o *rendering* de superfície convencional e o VOS^m é com relação aos tempos de interação do usuário com o objeto. Na Figura 4.9 são exibidas imagens geradas pelo VOS^m e suas correspondentes imagens exibidas com um *rendering* de superfície convencional, geradas utilizando o VTK.

A Tabela 4.4 mostra uma comparação do *rendering* da superfície gerada com o VOS^m, feita pelo OpenGL, para três configurações de cones distintas. A comparação é feita para os volumes do mamão, do tórax e do pé. A tabela também apresenta os tempos do *rendering* de superfície pelo VTK.

É notável a diferença entre a interação do *rendering* de superfície convencional e a interação do *rendering* de superfície do VOS^m. Entretanto, as imagens geradas pelo VOS^m são mais interessantes, do ponto de vista exploratório, do que imagens do *rendering* de superfície convencional.

Figura 4.9: Imagens de superfícies geradas pelo VOS^m (acima) e visualização das superfícies de interesse com *rendering* de superfície convencional (abaixo).

	Quadros por Segundo (VOS ^m e VTK)			
	3 cones x 8 raios	6 cones x 8 raios	8 cones x 8 raios	VTK
Mamão	10.6 fps	8 fps	6.6 fps	20 fps
Tórax	1.9 fps	1.3 fps	1.1 fps	3 fps
Pé	1.0 fps	0.6 fps	0.5 fps	2 fps

Tabela 4.4: Comparação da interação de três configurações do VOS^m para os volumes do mamão, do tórax e do pé.

O *rendering* de superfície do VOS^m é mais lento, pois precisa verificar qual dos raios amostrados mais se aproxima da direção de observação. Conseqüentemente, quanto mais raios por cone, ou quanto mais cones por vértice, mais lenta será a interação do usuário com o objeto.

4.4 Número de Cones versus Qualidade das Imagens

A Figura 4.10 mostra imagens do tórax geradas pelo VOS^m com três configurações diferentes. A coluna da esquerda foi gerada com 3 cones por vértice e 8 raios por cone, a coluna intermediária foi gerada com 6 cones por vértice e com 8 raios por cone, e a coluna da direita foi gerada com 8 cones por vértice e com 8 raios por cone. Claramente vemos que, quanto maior o número de cones, melhor a qualidade das imagens resultantes. Por outro lado, o custo do processo aumenta em termos dos tempos de processamento (já discutido) e do uso de memória (discutido na Seção 4.7).

4.5 Função de Transferência Baseada na Posição da Amostra no Raio

A Figura 4.11 ilustra o uso da função de transferência de opacidade baseada na posição da amostra, apresentada na Seção 3.5.2. Na imagem da Figura 4.11(a), obtida com a configuração de opacidade descrita na Figura 4.12(a), observa-se a parte de fora do mamão e a polpa, antes de chegar nas sementes. Na imagem da Figura 4.11(b), obtida com a configuração de opacidade descrita na Figura 4.12(b), a parte interna do volume é destacada.

Manipulando esse conjunto de dados observamos que foi mais fácil gerar imagens significativas das estruturas internas do volume utilizando essa abordagem para especificação de funções de transferência de opacidade, quando comparada à abordagem usual baseada apenas na intensidade da amostra. Embora os resultados mostrem imagens inferiores, acreditamos que esta é uma abordagem promissora para apoiar a definição de funções de transferência adequadas pelo usuário.

Figura 4.10: Efeito do número de cones na qualidade das imagens geradas com o VOS^m: coluna a esquerda: 3 cones com 8 raios, coluna intermediária: 6 cones com 8 raios e coluna a direita: 8 cones com 8 raios.

Figura 4.11: a) Opacidade mínima (zero) associada à posição de amostra 0 e máxima opacidade (1) associada à posição de amostra 1; b) Opacidade máxima associada às amostras posicionadas entre 0.39 e 0.47 e entre 0.53 e 0.64.

Figura 4.12: Função de transferência de opacidade baseada na posição da amostra: a) configuração para a Figura 4.11(a); b) configuração para a Figura 4.11(b).

4.6 Efeitos do Limiar

Como descrito no Capítulo 3 (Seção 3.5.1), VOS^m armazena somente as amostras que interessam para o usuário, que especifica um limiar de interesse. Na implementação atual o usuário seleciona o limiar por tentativa e erro observando o seu efeito nas imagens 2D do volume. Na Figura 4.13 é exibido o efeito do limiar nas imagens 2D para diferentes limiares, o número acima da imagem é o valor do limiar escolhido.

Figura 4.13: Imagens 2D orientando na escolha do limiar: a) Escolha do limiar de interesse no volume do Tórax; b) Escolha do limiar interesse no volume do Pé; c) Escolha do limiar interesse no volume do Mamão.

Na Figura 4.14 mostramos os efeitos da escolha do limiar de interesse em imagens geradas com o VOS^m. O número acima da imagem do volume é o valor do limiar escolhido pelo usuário e o número abaixo da imagem é o número médio de amostras armazenadas em cada raio. Nas Figuras 4.14 (a), (b) e (c) mantivemos a função de transferência de opacidade e modificamos somente o

valor do limiar.

Figura 4.14: a) Efeitos da escolha do limiar de interesse no volume do Tórax; b) Efeitos da escolha do limiar interesse no volume do Pé; c) Efeitos da escolha do limiar interesse no volume do Mamão.

Como era esperado, quanto maior o valor do limiar menor o número médio de amostras armazenadas por raio, e por consequência, a qualidade das imagens diminui, pois menos amostras são utilizadas para obter a imagem resultante. No entanto, o limiar permite focalizar somente os objetos de interesse.

4.7 Consumo de Memória

O consumo de memória do VOS^m depende de três fatores: número de vértices da superfície (NV), número de raios disparados por vértice (NRV) e o número médio de amostras armazenadas por raio (NMA). A multiplicação desses fatores ($NV * NRV * NMA$) leva à quantidade total de amostras que são armazenadas.

Na Tabela 4.5 é mostrada uma comparação entre a quantidade de memória necessária para o VOS^m armazenar todas as amostras, utilizando a configuração de 8 cones com 8 raios por cone. Também é mostrada a quantidade de memória necessária para armazenar somente o volume. O número total de amostras do volume é calculado multiplicando as suas dimensões. No exemplo consideramos que cada amostra ocupa 32 bits na memória.

	Amostras Armazenadas pelo VOS ^m	Total de Amostras do Volume
Pé (limiar 500)	$115942 * 65 * 12 = 90.434.760$ (345MB)	$102 * 247 * 199 = 5.898.360$ (22MB)
Tórax (limiar 500)	$83949 * 65 * 20 = 109.133.700$ (416MB)	$120 * 120 * 241 = 3.470.400$ (13MB)
Mamão (limiar 300)	$14470 * 65 * 6 = 5.643.300$ (21MB)	$86 * 42 * 59 = 213.108$ (800KB)

Tabela 4.5: Quantidade de memória consumida pelo VOS^m e pelo volume.

A quantidade de memória utilizada pelo VOS^m é muito superior à quantidade de memória requerida pelos algoritmos de DVR e SR convencionais. É importante ressaltar que a tabela só exibe a quantidade de memória gasta para armazenar as amostras do volume, sem levar em conta a memória utilizada para armazenar a superfície, as cores dos raios e outras informações necessárias para a execução do algoritmo.

A Técnica VoS^{m*} e Resultados

No Capítulo 4 foram apresentados os resultados da técnica VoS^m . Os tempos de atualização das funções de transferência são bons, quando comparados ao *Ray Casting* Convencional (implementação do VolView). Já o tempo de interação do usuário com a superfície permanecem inferiores ao tempo do *rendering* de superfície convencional. Além disso, mesmo dando uma noção do conteúdo do volume, as imagens são inferiores às imagens geradas por um algoritmo de *Ray Casting* Convencional.

Embora não tenhamos a intenção de competir com técnicas de *Rendering* Volumétrico Direto, investigamos mecanismos para melhorar a qualidade das imagens e os tempos de interação. Com o intuito de melhorar a qualidade das imagens, os tempos de atualização das funções de transferência e a interação do usuário com a superfície, algumas modificações foram introduzidas na técnica VoS^m . Agora as imagens finais são geradas por raios cuja direção coincide com a direção de observação. Assim como no VoS^m , os raios são disparados a partir dos vértices da superfície. Essa nova versão da técnica VoS^m , denominada VoS^{m*} , será descrita a seguir. Também apresentamos alguns resultados obtidos com essa nova versão. Conclusões sobre o trabalho são apresentadas no Capítulo 6.

5.1 Diferenças entre VoS^m e VoS^{m*}

A técnica VoS^{m*} segue o mesmo *pipeline* da técnica VoS^m apresentado no Capítulo 3 (Figura 3.3). É extraída uma superfície do volume, é criado um único cone de raios (no VoS^m são criados vários), o volume é carregado, a etapa de pré-processamento lança os raios amostrados sobre o

cone e armazena as amostras correspondentes para cada raio. É criado somente um cone de raios, que será utilizado para gerar as imagens da superfície com o volume mapeado a medida que o usuário interage com o objeto. Com a utilização de um único cone as rotações ficam mais rápidas, permitindo uma melhor interação do usuário com o objeto. Obviamente as imagens apresentadas durante a interação serão aproximações para orientar o usuário na escolha de uma posição interessante para explorar o volume. Quando o usuário pára a interação, a técnica gera a imagem final de melhor qualidade. Na Figura 5.1 são mostradas imagens geradas durante a interação e sua respectiva imagem final.

Assim, a diferença entre VOS^m e VOS^{m*} ocorre na etapa do *rendering* da superfície. A imagem final não é gerada a partir da amostragem armazenada dos raios disparados dos cones, mas sim pela amostragem de raios vindos do observador. A direção dos raios que amostram o volume coincide com a direção de observação. Esses raios têm como origem de lançamento os vértices da superfície, assim como na técnica VOS^m .

Na geração das imagens intermediárias, durante as rotações, são utilizadas as cores calculadas a partir da amostragem gerada com o cone de raios e o raio que melhor aproxima o vetor direção de observação é utilizado para atribuir cor ao vértice da superfície. Quando a interação do usuário com a superfície é interrompida o VOS^{m*} dispara raios de todos os vértices visíveis da superfície. Agora, os raios que formam a imagem final não são raios amostrados, como na técnica VOS^m , mas sim raios vindos do observador como no *Ray Casting* Convencional. Como na etapa de pré-processamento, os raios disparados para gerar a imagem final só compõem as amostras cujo valor é superior a um limiar definido pelo usuário.

5.2 Resultados

5.2.1 Tempos de Atualização das Funções de Transferência

A Tabela 5.1 mostra os tempos para executar o pré-processamento e a composição das cores quando as funções de transferência são atualizadas no VOS^{m*} . Comparando os tempos da Tabela 5.1 com os tempos do VOS^m (Seção 4.2) verificamos que, no geral, VOS^{m*} é mais rápido.

O pré-processamento é mais rápido porque são lançados somente 5 raios para cada vértice da superfície, enquanto que na configuração 8 cones com 8 raios do VOS^m são lançados 65 raios para cada vértice. Na atualização das funções de transferência é lançado apenas um único raio por vértice da superfície. No VOS^m os raios não são lançados novamente, pois as amostras já estão armazenadas. Mesmo assim o VOS^{m*} , no geral, executa o lançamento dos raios e a composição das amostras em um tempo menor.

	Tempos		
	Pré-Processamento	Melhor Caso	Pior Caso
Cabeça	12s	0,1s	3,7s
Pé	11s	0,9s	4,5s
Tórax	6s	0,07s	2,6s
Mamão	0,4s	0,01s	0,06s

Tabela 5.1: Tempos de atualização de função de transferência no VOS^{m*}.

5.2.2 Tempos de Interação com o Usuário

A interação com o usuário no VOS^{m*} é superior ao VOS^m, pois é necessário fazer a verificação com apenas 5 raios para atribuir a cor a um vértice, enquanto que no VOS^m com a configuração 8 cones e 8 raios por cone era necessário fazer a verificação com 65 raios. Entretanto, a interação ainda está lenta.

Na Tabela 5.2 são apresentados os tempos de interação do usuário (quadros por segundo) com o objeto para os volumes da cabeça (115.480 vértices e 231.215 faces), do pé (115.942 vértices e 231.846 faces), do tórax (83.949 vértices e 167.241 faces) e do mamão (14.470 vértices e 28.926 faces).

	Quadros por Segundo			
	Cabeça	Pé	Tórax	Mamão
fps	2	2	3	15

Tabela 5.2: Tempos de interação do usuário com a superfície no VOS^{m*}.

5.3 Consumo de Memória

O consumo de memória do VOS^{m*} é formado pela quantidade de amostras e pela quantidade ocupada para armazenar o volume. Na Tabela 5.3 é mostrada a quantidade de memória necessária para o VOS^{m*} armazenar todas as amostras e o volume. No exemplo consideramos que cada amostra ocupa 32 bits na memória.

	Comparação entre VOS ^{m*} e VOS ^m			
	Amostras Armazenadas	Volume	Total VOS ^{m*}	VOS ^m
Pé (limiar 500)	115942*5*12=6.956.520 (26MB)	22MB	48MB	345MB
Tórax (limiar 500)	83949*5*20=83.94.900 (32MB)	13MB	45MB	416MB
Mamão (limiar 300)	14470*5*6=434.100 (1,6MB)	800KB	2,4MB	21MB

Tabela 5.3: Quantidade de memória consumida pelo VOS^{m*}.

A quantidade de memória utilizada pelo VOS^{m*} é muito inferior à quantidade de memória requerida pelo VOS^m (Seção 4.7 Tabela 4.5). É importante ressaltar que a tabela só exibe a quan-

tidade de memória gasta para armazenar as amostras do volume e o volume, sem levar em conta a quantidade de memória utilizada para armazenar a superfície, as cores dos raios e outras informações necessárias para a execução do algoritmo.

5.3.1 Comparação com Imagens do VOS^m e do *Ray Casting* Convencional

Comparando as imagens obtidas com o VOS^m às imagens obtidas com o VOS^{m*} notamos que a qualidade das imagens do VOS^{m*} é superior e se aproxima das imagens obtidas com o *Ray Casting*. As Figuras 5.2, 5.3 e 5.4 mostram comparações entre as imagens obtidas com VOS^m , VOS^{m*} e *Ray Casting*. As imagens obtidas com o VOS^{m*} não apresentam as distorções e ruídos que o VOS^m apresenta. Além disso, algumas estruturas que não eram totalmente visíveis nas imagens obtidas com o VOS^m são agora visíveis.

Outro teste para comparar a qualidade das imagens com o *Ray Casting* Convencional foi realizado com o volume de uma cabeça, disponível em [http://www9.cs.fau.de/Persons/Roettger/library/](http://www9.cs.fau.de/Persons/Roettger/library/Visible%20Male/) (Visible Male). O volume possui dimensões 128x256x256.

Na Figura 5.5 são mostradas imagens do VOS^{m*} e imagens do *Ray Casting*. As imagens geradas pelo VOS^{m*} são bem próximas às imagens geradas pelo *Ray Casting*. Entretanto, estruturas pequenas, ou que requerem muito detalhe, ainda são perdidas pelo VOS^{m*} . Na coluna da direita a imagem obtida com o *Ray Casting* mostra uma marca (em forma de X) no crânio, já a imagem do VOS^{m*} não consegue exibir esse detalhe.

Figura 5.1: Imagens intermediárias geradas durante a interação do usuário (esquerda) e respectivas imagens finais (direita).

Figura 5.2: Volume do mamão: comparação entre as imagens do VoS^m (linha superior), do VoS^{m*} (linha intermediária) e *Ray Casting* (linha inferior).

Figura 5.3: Volume do tórax: comparação entre as imagens do VoS^m (linha superior), do VoS^{m*} (linha intermediária) e *Ray Casting* (linha inferior).

Figura 5.4: Volume do pé: comparação entre as imagens do VoS^m (linha superior), do VoS^{m*} (linha intermediária) e *Ray Casting* (linha inferior).

Figura 5.5: Volume da cabeça: comparação entre as imagens do VOS^{M*} (linha superior) e do *Ray Casting* (linha inferior).

Conclusão

6.1 Conclusão

Nesse trabalho fizemos um estudo e uma generalização da técnica $V\circ S$ (Cateriano e Nonato, 2003). Uma nova implementação da técnica foi desenvolvida, em que procuramos tratar diversas limitações da técnica $V\circ S$ original, tanto no que diz respeito ao tipo de volume manipulado, à qualidade das imagens geradas e aos recursos para especificação das funções de transferência. A nova versão da técnica é aplicável a volumes regulares, que são volumes mais frequentes, e isso permitiu comparar os resultados com outras técnicas. Além disso, a nova versão apresenta melhores tempos de execução e taxas de interação mais altas em relação à solução original.

A nova versão da técnica $V\circ S$ foi nomeada $V\circ S^m$, e como a técnica original, provê suporte para a visualização das estruturas internas contidas em um volume. Para isso uma superfície é extraída do volume e o seu conteúdo é mapeado nas faces da superfície extraída. A maior eficiência do $V\circ S^m$ em relação à proposta original é consequência do armazenamento das amostras obtidas ao longo dos raios lançados, acelerando a composição das amostras. O armazenamento das amostras também acelera a interação do usuário com o objeto, pois os raios não precisam ser novamente lançados a cada nova imagem gerada, visto que as cores dos vértices já estão pré-computadas para alguns pontos de observação amostrados.

A grande vantagem da técnica é na visualização do conteúdo de volumes em placas gráficas convencionais, oferecendo um *rendering* de superfícies melhorado. No entanto, as imagens geradas pelo $V\circ S^m$ não têm a mesma qualidade de um processo de *rendering* volumétrico direto,

apesar de darem uma boa noção do conteúdo do volume. Buscando gerar imagens de melhor qualidade foi desenvolvida uma versão alternativa do VOS^m , que foi denominada VOS^{m*} .

O VOS^{m*} segue os mesmos princípios do VOS^m , sendo que a diferença se encontra nas etapas do mapeamento do volume na superfície e de renderização da superfície final para o usuário. O mapeamento do volume é executado lançando raios disparados de um único cone (na implementação são lançados cinco raios). Essa amostragem de poucos raios resulta na exibição de uma aproximação grosseira da superfície de interesse enquanto o usuário está interagindo com o modelo. Quando a interação do usuário é interrompida, novos raios são disparados para o volume a partir dos vértices da superfície. Esses são raios originados no observador passando pelos vértices da superfície, como no *Ray Casting* convencional. Como não são raios com direções aproximadas, a qualidade das imagens geradas é superior a das imagens geradas pelo VOS^m . Além da melhor qualidade das imagens, as taxas de interação do usuário com a superfície, apesar de ainda serem inferiores as de um *rendering* de superfícies convencional, também é superior as do VOS^m . Finalmente, o tempo de composição das amostras, no geral, também é superior.

Em resumo, entre as duas novas implementações da técnica VOS , a VOS^{m*} apresenta melhores resultados em termos de qualidade das imagens, tempo de composição das amostras e interação do usuário com a superfície. VOS^{m*} mostra-se uma boa solução para *Rendering* de Superfícies quando a utilização de superfícies é uma solução natural, ou quando há a necessidade de conhecer o interior das superfícies. Além do mais, pode oferecer uma alternativa rápida para gerar imagens iniciais do conteúdo de volumes, pesquisar funções de transferência adequadas, ou disponibilizar visualização volumétrica em computadores sem placas gráficas dedicadas para *rendering* volumétrico.

6.2 Trabalhos Futuros

As técnicas desenvolvidas nesse trabalho mapeiam o conteúdo de um volume sobre as faces de superfícies definidas por malhas de triângulos. É evidente que a qualidade da malha que descreve a superfície afeta a qualidade das imagens geradas. Da mesma forma, o tamanho da malha afeta a eficiência da técnica, tanto em termos de tempo como de consumo de memória. Assim, seria interessante analisar com cuidado o efeito de algoritmos de suavização e simplificação de malhas na imagem final, bem como o efeito do tamanho dos polígonos na qualidade do mapeamento efetuado. Operações de *zooming in* exigiriam uma subdivisão local da superfície para manter a qualidade do mapeamento, e esse é outro aspecto que poderia ser tratado na implementação.

As soluções desenvolvidas utilizam a placa gráfica apenas na etapa de *rendering* das superfícies, que é feita via OpenGL. Soluções mais eficientes seriam possíveis explorando os recursos

disponíveis em aceleradores gráficos (GPU) para realizar o lançamento dos raios e implementar os cálculos de composição das amostras. Ademais, o lançamento dos raios na etapa de mapeamento do volume poderia ser realizado em paralelo, com ganhos significativos de tempos de processamento.

As funções de transferência são ferramentas essenciais na exploração de volumes, e a especificação de funções adequadas é uma etapa crítica para o usuário. Um aspecto que merece mais investigação é a aplicação de estratégias mais sofisticadas para a especificação de funções de transferência, que considerem não apenas o valor das amostras, mas informações como gradientes e mesmo outras de mais alta ordem.

Finalmente, também seria interessante investigar a aplicabilidade da técnica em aplicações de interesse que envolvam a exploração de volumes, como medicina e agricultura.

Referências Bibliográficas

- BAJAJ, C. L.; PASCUCCI, V.; SCHIKORE, D. R. The contour spectrum. *IEEE Visualization 97*, p. 167–173, 1997.
- BOISSONNAT, J. D.; DEVILLERS, O.; PION, S.; TEILLAUD, M.; YVINEC, M. Triangulations in cgal. *Computational Geometry – Theory and Applications*, v. 22, p. 5–19, 2002.
- CATERIANO, P. S. H.; NONATO, L. G. *Técnica híbrida de visualização para exploração de dados volumétricos não estruturados*. Dissertação de Mestrado, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, 2003.
- CHEN, B.; KAUFMAN, A.; TANG, Q. Image-based rendering of surfaces from volume data. In: *IEEE Workshop on Volume Graphics*, Stony Brook, NY, USA, 2001, p. 355–358.
- FERRE, M.; PUIG, A.; TOST, D. Using a classification tree to speed up rendering of hybrid surface and volume models. *Journal of Winter School of Computer Graphics*, v. 12, n. 1, p. 105–112, 2004.
- GORDON, D.; CHEN, S. Front-to-back display of BSP trees. *IEEE Comput. Graph. Appl.*, v. 11, n. 5, p. 79–85, 1991.
- GRIMM, S.; BRUCKNER, S.; KANITSAR, A.; GRÖLLER, E. A refined data addressing and processing scheme to accelerate volume raycasting. *Computers and Graphics*, v. 28, n. 5, p. 719–729, 2004a.
- GRIMM, S.; BRUCKNER, S.; KANITSAR, A.; GRÖLLER, M. E. Memory efficient acceleration structures and techniques for cpu-based volume raycasting of large data. In: D. SILVER, T. ERTL, C. S., ed. *Proceedings IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics*, 2004b, p. 1–8.
- HADWIGER, M.; BERGER, C.; HAUSER, H. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. *IEEE Visualization*, p. 301–308, 2003.

- HAUSER, H.; MROZ, L.; BISCHI, G.-I.; GRÖLLER, M. E. Two-level volume rendering – fusing mip and dvr. *Proceedings of the conference on Visualization '00*, p. 211–218, 2000.
- HE, T.; HONG, L.; KAUFMAN, A.; PFISTER, H. Generation of transfer functions with stochastic search techniques. *Proceedings of the 7th conference on Visualization '96*, p. 227–ff., 1996.
- KAUFMAN, A. Volume visualization: Principles and advances. *State University of New York*, 1997.
- KAUFMAN, A. Advances in volume visualization. *SIGGRAPH '98: Course Notes nº 24*, 1998.
- KINDLMANN, G.; DURKIN, J. W. Semi-automatic generation of transfer functions for direct volume rendering. *IEEE Symposium on Volume Visualization*, p. 79–86, 1998.
- KÖNIG, A. H.; GRÖLLER, E. M. Mastering transfer function specification by using volumepro technology. *Proceedings of the 17th Spring Conference on Computer Graphics*, 2001.
- KNISS, J.; KINDLMANN, G.; HANSEN, C. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. *Proceedings of the conference on Visualization '01*, p. 255–262, 2001.
- LEVOY, M. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, v. 8, n. 3, p. 29–37, 1988.
- LEVOY, M. A hybrid ray tracer for rendering polygon and volume data. *IEEE computer graphics and applications*, v. 10, n. 3, p. 33–40, 1990.
- LORENSEN, W. E.; CLINE, H. E. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, p. 163–169, 1987.
- MARKS, J.; ANDALMAN, B.; BEARDSLEY, P. A.; FREEMAN, W.; GIBSON, S.; HODGINS, J.; KANG, T.; MIRTICH, B.; PFISTER, H.; RUMMLER, W.; RYALL, K.; SEIMS, J.; SHIEBER, S. Design galleries: a general approach to setting parameters for computer graphics and animation. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, p. 389–400, 1997.
- MUELLER, K.; MÖLLER, T.; CRAWFIS, R. Splatting without the blur. *Proceedings of the conference on Visualization '99*, p. 363–370, 1999.
- OLIVEIRA, M. C. F.; MINGHIM, R. Uma introdução a visualização computacional. *Jornadas de Atualização em Informática*, p. 85–131, 1997.
- PFISTER, H.; LORENSEN, W. E.; BAJAJ, C.; KINDLMANN, G.; SCHROEDER, W.; AVILA, L. S.; MARTIN, K.; MACHIRAJU, R.; LEE, J. The transfer function bake-off. *IEEE Computer Graphics and Applications*, v. 21, n. 3, p. 16–22, 2001.

- ROSSIGNAC, J. 3d compression made simple: Edgebreaker with zip&wrap on a cornertable. *SMI 01: Proceedings of the International Conference on Shape Modeling & Applications*, p. 278, 2001.
- SCHROEDER, W.; MARTIN, K.; LORENSSEN, B. *The visualization toolkit: An object-oriented approach to 3D graphics*. 2 ed. Prentice Hall, 2002.
- WESTENBERG, M. A.; ROERDINK, J. B. T. M. Wavelet-based volume visualization. *Nieuw Archief voor Wiskunde*, v. 17, n. 2, p. 149–158, 1999.
- WESTOVER, L. Footprint evaluation for volume rendering. *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, p. 367–376, 1990.
- WILSON, B.; MA, K.-L.; MCCORMICK, P. S. A hardware-assisted hybrid rendering technique for interactive volume visualization. In: *VVS '02: Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, Piscataway, NJ, USA: IEEE Press, 2002, p. 123–130.

Utilizando as Classes Desenvolvidas

A.1 Classe SurfaceData

SurfaceData();

Construtor da classe SurfaceData.

void loadSurface(char *filename);

Método responsável por carregar as informações da superfície na estrutura de dados. Recebe como parâmetro o nome do arquivo que contém informações da superfície. Nesse método a superfície é separada, comente a componente de interesse é utilizada, as faces são orientadas e os cones são criados.

int getNumCel();

Retorna o número de triângulos que a superfície contém.

int getNumVert();

Retorna o número de vértices que a superfície contém.

void rotateX(int teta);

Rotaciona no eixo X os vértices da superfície e seus respectivos raios.

void rotateY(int teta);

Rotaciona no eixo Y os vértices da superfície e seus respectivos raios.

void rotateZ(int teta);

Rotaciona no eixo Z os vértices da superfície e seus respectivos raios.

double *getPoint(int vertice);

Retorna os pontos rotacionados de um determinado vértice.

double *getOPoint(int vertice);

Retorna os pontos originais de um determinado vértice.

void change(float X, float Y, float Z);

Percorre todos os raios e verifica qual é o mais próximo do vetor [X,Y,Z].

void deleteRays();

Deleta todos os raios

int ROM();

Retorna True se os raios ainda estão na memória, caso contrário retorna False.

void setCompo(int _compo);

Atribui qual componente da superfície será utilizada na visualização.

int NumMedioAmostras();

Retorna o número médio de amostras por raio.

A.2 Classe ImageData

ImageData();

Construtor da classe ImageData.

~ImageData();

Destrutor da classe ImageData.

void executeSobel();

Executa o algoritmo de Sobel volumétrico no volume armazenado.

float *findCell(float X, float Y, float Z);

Retorna os pontos (x1, x2, y1, y2, z1, z2) da célula onde se encontra ponto (X, Y, Z).

void readSlice(int slice, char *filename);

Lê a imagem do arquivo filename e armazena na posição indicada no parâmetro slice.

void setOrigin(float X, float Y, float Z);

Define os pontos de origem do volume.

void setExtension(float X, float Y, float Z);

Define as dimensões do volume.

void setSpacing(float X, float Y, float Z);

Define o espaçamento entre os pontos do volume.

void setScalars(int nscalars);

Define o número de escalares que o volume armazenará em cada vértice.

float getSample(float X, float Y, float Z);

Retorna o valor escalar correspondente ao ponto (X, Y, Z). O valor é calculado por interpolação.

int getIntSample(int X, int Y, int Z);

Retorna o valor escalar correspondente ao ponto (X, Y, Z). O valor não é calculado por interpolação, pega-se o vizinho mais próximo.

double getSobel(int X, int Y, int Z);

Retorna o valor calculado pelo algoritmo de Sobel na posição dos índices X, Y e Z.

double *getHistogram();

Obtém o histograma, caso ele não existe será criado.

int OnMemory();

Retorna True se o volume ainda está na memória, caso contrário retorna False.

A.3 Classe ColorTF

ColorTF(int max);

Construtor da classe ColorTF. Recebe como parâmetro o número máximo de pontos de controle que a função de transferência possuirá.

void setRange(float ini, float fim);

Define os intervalos iniciais e finais das intensidades do volume.

float *classify(float amostra);

Retorna os valores R, G e B para um valor de amostra passado como parâmetro. Os valores R, G e B são calculados por interpolação.

void generateValues();

Pré-calcula os valores R, G e B para todos os valores de intensidade do intervalo [ini,fim]. Os valores são armazenados em um vetor de 256 posições.

float *classify2(int amostra);

Retorna os valores R, G e B para um valor de amostra passado como parâmetro. Os valores R, G e B não são calculados por interpolação, são recuperados do vetor criado pelo método generateValues.

void addPoint(float point, float R, float G, float B);

Adiciona os valores R, G e B para uma intensidade passada como parâmetro.

float *getPoint(float point);

Retorna os valores R, G e B associados a uma determinada intensidade passada como parâmetro.

void removePoint(float point);

Remove o ponto de controle referente à intensidade passada como parâmetro.

void clearPoints();

Remove todos os pontos de controle.

A.4 Classe OpacityTF

OpacityTF(int max);

Construtor da classe OpacityTF. Recebe como parâmetro o número máximo de pontos de controle que a função de transferência possuirá.

void setRange(float ini, float fim);

Define os intervalos iniciais e finais das intensidades do volume.

void addPoint(float point, float op);

Adiciona os de opacidade op para uma intensidade passada como parâmetro.

void removePoint(float point);

Remove o ponto de controle referente à intensidade passada como parâmetro.

void clearPoints();

Remove todos os pontos de controle.

float classify(float amostra);

Retorna os valor de opacidade do valor de amostra passado como parâmetro. O valor da opacidade é calculado por interpolação.

void generateValues();

Pré-calcula os valores de opacidade para todos os valores de intensidade do intervalo [ini,fim]. Os valores são armazenados em um vetor de 256 posições.

float classify2(int amostra);

Retorna o valor de opacidade para um valor de amostra passado como parâmetro. O valor de opacidade não é calculado por interpolação, é recuperado do vetor criado pelo método generateValues.

A.5 Classe DistanceTF

DistanceTF(int max);

Construtor da classe DistanceTF. Recebe como parâmetro o número máximo de pontos de controle que a função de transferência possuirá.

float classify(float amostra);

Retorna os valor de opacidade da posição da amostra passado como parâmetro. O valor da opacidade é calculado por interpolação.

void addPoint(float point, float op);

Adiciona os de opacidade op para uma posição passada como parâmetro.

void removePoint(float point);

Remove o ponto de controle referente à intensidade passada como parâmetro.

void clearPoints();

Remove todos os pontos de controle.

A.6 Classe Ray Casting

RayCasting();

Construtor da classe RayCasting.

void setImageData(ImageData * ID);

Define o ImageData com informações do volume que será utilizado no ray casting.

void setSurfaceData(SurfaceData * SD);

Define o SurfaceData com informações da superfície que será utilizado no ray casting.

void setColorTF(ColorTF *TF);

Define a função de transferência de cor que será utilizada pelo ray casting.

void setOpacityTF(OpacityTF *TFO);

Define a função de transferência de opacidade que será utilizada pelo ray casting.

void setDistanceTF(DistanceTF *TFD);

Define a função de transferência baseada na posição da amostra que será utilizada pelo ray casting.

void executePre();

Executa o pré-processamento do ray casting. Executa o lançamento dos raios e armazenas as amostras para cada raio.

void execute();

Executa a composição das amostras que estão armazenadas para cada raio.

void setThreshold(float t);

Define um limiar que sera utilizado no lançamento dos raios.

int *castRay(double *origem, float *raio);

Lança um raio com direção 'raio' a partir da origem 'origem' para dentro do volume. Quando o raio atinge opacidade máxima ou sai do volume é retornado as amostras alcançadas por esse raio. São consideradas somente as amostras que estão acima de um determinado limiar.

float getThreshold();

Retorna o valor do limiar que está sendo utilizado.

void setTFOOn();

Define a utilização da função de transferência de opacidade baseada na intensidade.

void setTFPOn();

Define a utilização da função de transferência de opacidade baseada na posição da amostra.

A.7 Exemplo de Utilização das Classes

Carregando as Informações da Superfície

```
char *filename; //nome do arquivo com informações da malha
int componente; //componente da malha utilizada na visualização
int NumVert, //número de vértices
    NumFaces; //número de faces

SurfaceData *sd;
sd = new SurfaceData();
sd->setCompo(componente);
sd->loadSurface(filename);
NumVert = sd->getNumVert();
NumFaces = sd->getNumCel();
```

Carregando as Informações do Volume

```
int i;
int largura, //resolução horizontal da imagem
    altura, //resolução vertical da imagem
    fatias; //número de fatias do volume
float EspX, //espaçamento em X entre as células
    EspY, //espaçamento em Y entre as células
    EspZ; //espaçamento em Z entre as células
ImageData *id;
id = new ImageData();
id->setExtension(largura, altura, fatias);
id->setSpacing(EspX, EspY, EspZ);
id->setOrigin(0,0,0); //origem do volume é o ponto [0,0,0]
id->allocateVolume(); //aloca memória para o volume
for (i= 0; i < fatias; i++){
    //deve-se passar o nome do arquivo de cada fatia
    id->readSlice(i, filename + "i");
}
id->executeSobel(); //executa o Sobel 3D no volume carregado
```

Configurando uma Função de Transferência de Cor

```
ColorTF *tf = new ColorTF(20); //máximo de 20 pontos de controle
tf->clearPoints();
tf->addPoint(0, 1, 0, 0); //cor vermelha para o escalar 0
tf->addPoint(100, 0, 1, 0); //cor verde para o escalar 100
tf->addPoint(255, 0, 0, 1); //cor azul para o escalar 255
tf->generateValues(); //calcula e armazena a interpolação das cores
                        // para os escalares entre os pontos de
                        //controle previamente definidos
```

Recuperando as cores de uma Função de Transferência de Cor

```
Float *coram; //cor amostrada
coram = tf->classify2(0); //retorna a cor correspondente ao
                        //escalar 0
coram = tf->classify2(50); //retorna a cor correspondente ao
                        //escalar 50
```

Configurando uma Função de Transferência de Opacidade

```
OpacityTF *tfo = new OpacityTF(20); //máximo de 20 pontos de
tfo->clearPoints();                //controle
tfo->addPoint(0, 0); //opacidade 0 para o escalar 0
tfo->addPoint(100, 0.4); //opacidade 0.4 para o escalar 0
tfo->addPoint(255, 1); //opacidade 1 para o escalar 0
tfo->generateValues(); //calcula e armazena a interpolação das
                        //opacidades para os escalares entre os
                        //pontos de controle previamente definidos
```

Recuperando as opacidades de uma Função de Transferência de Opacidade

```
Float opam; //opacidade amostrada
opam = tfo->classify2(0); //retorna a opacidade correspondente
                        //ao escalar 0
opam = tfo->classify2(50); //retorna a opacidade correspondente
                        //ao escalar 50
```

Configurando uma Função de Transferência Baseada na Posição da Amostra

```
DistanceTF *tfd = new DistanceTF(20); //máximo de 20 pontos de controle
tfd->clearPoints();
tfd->addPoint(0, 0); //opacidade 0 para a distância
tfd->addPoint(0.5, 0.4); //opacidade 0.4 para o distância 0.5
tfd->addPoint(1, 1); //opacidade 1 para a distância 1
tfd->generateValues(); //calcula e armazena a interpolação das
                        //opacidades para as distâncias entre os
                        //pontos de controle previamente definidos
```

Recuperando as opacidades de uma Função de Transferência Baseada na Posição da Amostra

```
Float opam; //opacidade amostrada
opam = tfd->classify2(0); //retorna a opacidade correspondente
                        //à distância 0
opam = tfd->classify2(0.5); //retorna a opacidade correspondente
                        //à distância 0.5
```

Executando o Ray Casting

```
RayCasting *rc = new RayCasting();
rc->setImageData(id);
rc->setSurfaceData(sd);
rc->setThreshold(300);
rc->executePre();//executa o pré-processamento e armazena as amostras

//após o pré-processamento os raios e o
//volume podem ser descartados
sd->deleteRays();
id->\~ImageData();

rc->setColorTF(tf); \
rc->setOpacityTF(tfo); \
rc->setDistanceTF(tfd); \

//define qual função de opacidade será utilizada
if (funcao\_baseada\_na\_opacidade)
    rc->setTFOOn();
else rc->setTFPOn();

rc->execute(); //executa a composição das amostras
```

Visualizando a Superfície

```
Float VX, VY, VZ; //direção do observador
//seleciona o raio que mais se aproxima do observador
//o número do raio mais próximo é indicado pelo retorno
//do método getSelColor()
sd->change(VX, VY, VZ);

//No caso utilizaremos OpenGL
float cor[9], mR, mG, mB;
double x,y,z, pontos[9];
int c, v;
ofCell12D<MyTraits> *cell;
MyVertex *mv;
ofCellsIterator<MyTraits> itc(sd->malha);
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

for( itc.initialize(); itc.notFinish(); ++itc){
    c = &itc;
    cell = sd->malha.getCell(c);

    v = cell->getVertex(0);
    mv = sd->malha.getVertex(v);
    cor[0] = mv->getColor( mv->getSelColor() * 3);
    cor[1] = mv->getColor( mv->getSelColor() * 3 + 1);
    cor[2] = mv->getColor( mv->getSelColor() * 3 + 2);
    pontos[0] = mv->getRCoord(0);
    pontos[1] = mv->getRCoord(1);
    pontos[2] = mv->getRCoord(2);

    v = cell->getVertex(1);
    mv = sd->malha.getVertex(v);
    cor[3] = mv->getColor( mv->getSelColor() * 3);
    cor[4] = mv->getColor( mv->getSelColor() * 3 + 1);
    cor[5] = mv->getColor( mv->getSelColor() * 3 + 2);
    pontos[3] = mv->getRCoord(0);
    pontos[4] = mv->getRCoord(1);
    pontos[5] = mv->getRCoord(2);

    v = cell->getVertex(2);
    mv = sd->malha.getVertex(v);
    cor[6] = mv->getColor( mv->getSelColor() * 3);
    cor[7] = mv->getColor( mv->getSelColor() * 3 + 1);
    cor[8] = mv->getColor( mv->getSelColor() * 3 + 2);
    pontos[6] = mv->getRCoord(0);
    pontos[7] = mv->getRCoord(1);
    pontos[8] = mv->getRCoord(2);

    //média das cores
    mR = ( cor[0] + cor[3] + cor[6] ) / 3;
    mG = ( cor[1] + cor[4] + cor[7] ) / 3;
    mB = ( cor[2] + cor[5] + cor[8] ) / 3;
```

```
glColor3f(mR/255, mG/255, mB/255);
glBegin(GL_TRIANGLES);
    glVertex3f(pontos[0] * S + TX ,pontos[1] * S + TY ,pontos[2] );
    glVertex3f(pontos[3] * S + TX ,pontos[4] * S + TY ,pontos[5] );
    glVertex3f(pontos[6] * S + TX ,pontos[7] * S + TY ,pontos[8] );
glEnd();
}
```