VISUALIZAÇÃO TRIDIMENSIONAL COMBINADA DE DADOS VOLUMÉTRICOS E MODELOS POLIGONAIS USANDO O ALGORITMO SHEAR-WARP

ANA ELISA FERREIRA SCHMIDT

TESE DE DOUTORADO

Tecgraf
DEPARTAMENTO DE INFORMÁTICA
PUC-Rio

Rio de Janeiro, 28 de abril de 2000

VISUALIZAÇÃO TRIDIMENSIONAL COMBINADA DE DADOS VOLUMÉTRICOS E MODELOS POLIGONAIS USANDO O ALGORITMO SHEAR-WARP

ANA ELISA FERREIRA SCHMIDT

TESE DE DOUTORADO

Tecgraf
DEPARTAMENTO DE INFORMÁTICA
PUC-Rio

Rio de Janeiro, 28 de Abril de 2000

Ana Elisa Ferreira Schmidt

VISUALIZAÇÃO TRIDIMENSIONAL COMBINADA DE DADOS VOLUMÉTRICOS E MODELOS POLIGONAIS USANDO O ALGORITMO SHEAR-WARP

Tese apresentada ao Departamento de Informática como parte dos requisitos necessários para a obtenção do título de Doutor em Ciências em Informática.

Orientador: Marcelo Gattass

Co-orientador: Paulo Cezar Pinto Carvalho

Tecgraf
DEPARTAMENTO DE INFORMÁTICA
PUC-Rio

Rio de Janeiro, 28 de Abril de 2000

AGRADECIMENTOS

Ao Prof. Marcelo e Prof. Paulo Cezar, pelo apoio, orientação e amizade, fundamentais para o desenvolvimento e conclusão desta tese.

A todos os amigos e colegas do Tecgraf que de alguma forma contribuíram para a realização deste trabalho. Em especial, gostaria de agradecer ao Scuri, a Paulinha, ao Alexandre Ferreira, ao Dario, ao Flávio Szenberg, ao Luiz Henrique, a Carla e a Bia, pelas inúmeras horas que compartilhamos durante estes anos, quer fossem pra resolver problemas relacionados a tese, quer fossem numa conversa oferecendo o ombro amigo.

Aos meus pais e amigos distantes que não faltaram nunca com o suporte carinhoso nas horas mais difíceis.

Ao CNPq pelo auxílio financeiro.

Este trabalho foi desenvolvido no Tecgraf/PUC-Rio, Grupo de Tecnologia em Computação Gráfica da PUC-Rio.

RESUMO

Esta de tese de doutorado apresenta soluções para a integração entre os algoritmos de *Shear-Warp* e de *Z-Buffer*, visando a criação de cenas compostas de dados volumétricos e modelos poligonais opacos e/ou transparentes.

A renderização volumétrica baseada na fatoração *Shear-Warp* é eficiente para a realização das etapas de projeção e composição dos *voxels*. Ela trabalha de forma coerente com a ordenação das fatias do volume e pode ser naturalmente integrada ao algoritmo de *Z-Buffer* para a combinação de modelos poligonais.

São propostos vários algoritmos para a combinação de dados volumétricos e modelos poligonais visando reduzir o efeito de *aliasing* introduzido durante o processo de reamostragem do modelo poligonal. São apresentados testes e resultados que dão suporte às conclusões apresentadas sob o ponto de vista de qualidade de imagem e tempo de processamento.

Palavras-chave: Shear-Warp, Z-Buffer, Modelo Poligonal, Renderização Volumétrica Híbrida.

ABSTRACT

This thesis presents a study on the integration between Shear-Warp and Z-Buffer algorithms, and extends the Shear-Warp algorithm to handle scenes composed of both volume and polygonal data.

Shear-Warp techniques provide an efficient way to perform the projection and blending stages of the volume-rendering process. They treat volume-data slices in a coherent order and can be naturally integrated with the Z-Buffer algorithm.

We present methods to handle opaque/translucent volumes combined with opaque/translucent polygonal models. As volume data usually has a different resolution from that of the final image, in which Z-Buffer renders the polygonal data, several variants for this integration that try to reduce aliasing problems are analyzed. Results are shown to support some conclusions on the trade-off quality-versus-time that can be expected.

Keyword: Shear-Warp, Z-Buffer, Polygonal Models, Hybrid Volume Rendering.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

1 VISUALIZAÇÃO VOLUMÉTRICA

A Visualização Científica tem por objetivo criar imagens de conjuntos de dados que representam fenômenos contendo informações científicas sobre as quais se deseja realizar uma análise e/ou exploração. Essas imagens auxiliam na extração de informações relevantes e permitem uma melhor compreensão dos fenômenos estudados.

Uma subárea da Visualização Científica é a **Visualização Volumétrica,** cujo objetivo é a exibição do interior de objetos volumétricos, a fim de explorar sua estrutura e facilitar seu entendimento [McCormick,1987].

Na maioria das vezes, os objetos volumétricos são definidos através de uma grade tridimensional, com um ou mais valores escalares e/ou vetoriais associados a cada ponto da grade. Os dados volumétricos são normalmente tratados como uma matriz de elementos de volume, denominados *voxels* (análogo 3D de *pixels*).

O *voxel* é a representação de uma vizinhança espacial, sobre a qual determinados atributos e características estão associados e podem ser tratados e inferidos. Restringindo-se o conceito de *voxel* tem-se as células-*voxels*, que são paralelepípedos formados pela divisão do espaço do objeto através de um conjunto de planos paralelos aos eixos principais desse espaço. As células-*voxels* realizam uma decomposição celular do espaço, com células suficientemente pequenas se comparadas às características representadas dentro deste espaço de dados volumétricos [Paiva,1999].

No contexto desta tese, vamos nos referir a célula-*voxel* através do termo *voxel*, uma vez que os dados que serão trabalhados aqui representam seu espaço de dados através de *voxels* do tipo células-*voxels*. Partindo de uma analogia com o conceito de *pixel*, entende-se como *voxel* uma tupla $\langle i,j,k,E \rangle$ que define um ponto amostrado do campo escalar na posição (i,j,k) e com valor E associado. O valor escalar E pode estar

associado ao centro do *voxel* ou a cada um dos 8 vértices que o compõe. A Figura 1.1 mostra a representação de um dado volumétrico como uma grade tridimensional (isotrópica e regular), onde o valor escalar associado a cada *voxel* é obtido a partir da interpolação dos valores escalares contidos em cada vértice do mesmo.

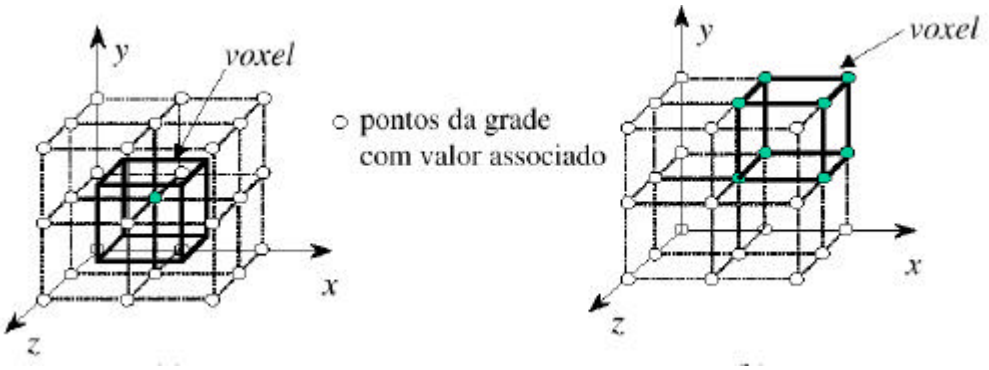


Figura 1.1 – Elementos de um dado volumétrico representado na grade 3D [Paiva.1999]

Os dados volumétricos são obtidos, geralmente, de duas formas: por simulação, na qual os dados são construídos a partir de algum modelo matemático, e por aquisição, em que os dados são resultantes da conversão de objetos existentes na natureza.

Um exemplo de dado volumétrico obtido por simulação ocorre na análise de fenômenos físicos através de elementos finitos ou por dinâmica de fluidos. A Figura 1.2 mostra uma imagem criada a partir de dados obtidos pela simulação do fluxo de ar em torno de um míssil, através de dinâmica de fluidos.

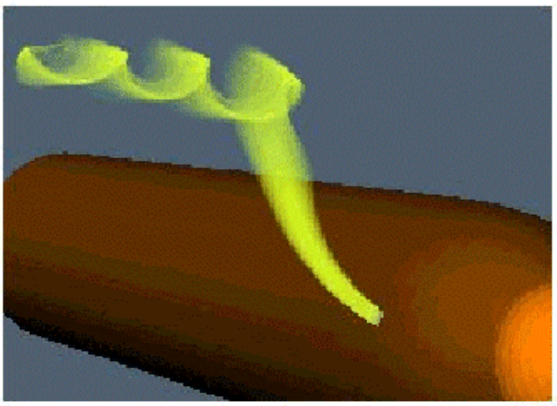
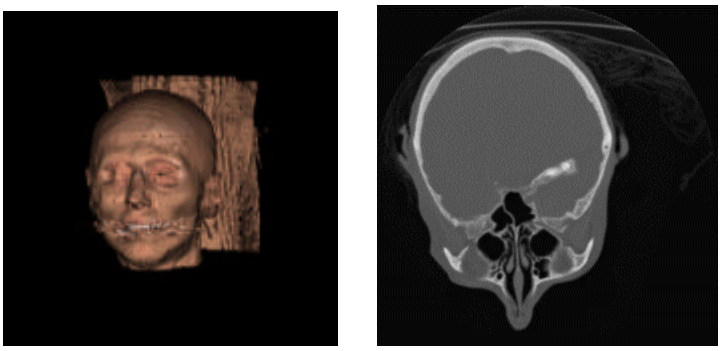


Figura 1.2 – Simulação do fluxo de ar através de dinâmica de fluidos [Paiva,1999]

Exames médicos como os de Ressonância Magnética (RM), Tomografia Computadorizada (TC) e Ultra-sonografia geram conjuntos de dados volumétricos adquiridos. Este volumes representam as estruturas internas de regiões do corpo do paciente, permitindo que os médicos façam diagnósticos sem a necessidade de procedimentos invasivos. A Figura 1.3 mostra um exemplo de dado volumétrico

construído a partir das fatias extraídas de um exame de Tomografia Computadorizada de uma cabeça.



(a) (b)

Figura 1.3 – (a) Reconstrução 3D a partir de (b) fatias de TC de uma cabeça

Uma outra área de aplicação que trabalha com dados volumétricos adquiridos é a de exploração de petróleo, em que estruturas geológicas sob a superfície da terra devem ser detectadas através de um processo denominado *interpretação sísmica*. Os dados podem ser obtidos ou através da perfuração de um poço e da extração de material para a inspeção das camadas geológicas existentes na área, ou através da aquisição indireta, na qual são detonados explosivos e medidas as ondas refletidas pelas descontinuidades existentes na subsuperfície. No segundo caso, os dados adquiridos são representados através de uma grade 3D de amostras, contendo a amplitude das ondas sonoras refletidas. A Figura 1.4 exibe um exemplo de volume sísmico visualizado através de técnicas de visualização volumétrica.

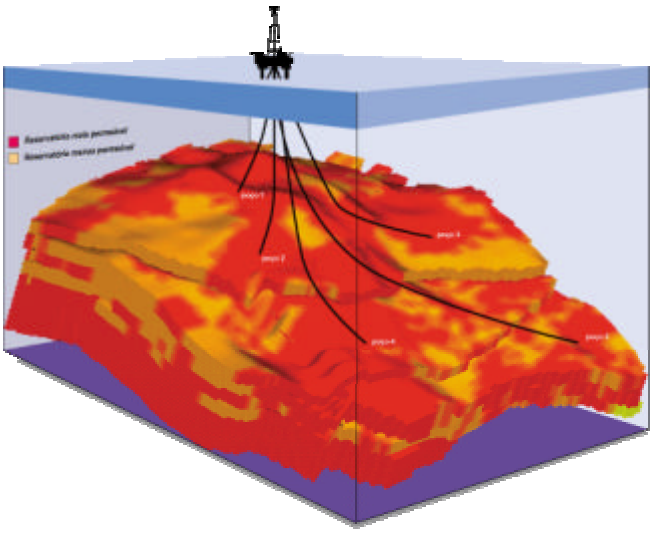


Figura 1.4 – Representação volumétrica de um dado sísmico (cedida pela Petrobrás)

Esta diversidade de formas de aquisição de dados volumétricos vem impulsionando pesquisas na busca de novas técnicas de visualização, a serem empregadas em áreas como medicina, geociências, sensoriamento remoto, meteorologia, dentre outras.

Tais formas de aquisição de dados reais e técnicas de simulação podem produzir grandes quantidades de dados volumétricos, o que dificulta a sua compreensão e manipulação. É necessário, então, que se empreguem técnicas específicas de visualização de dados volumétricos para produzir imagens que auxiliem na exploração e interpretação dos mesmos. As grandezas mais comuns a serem interpretadas nestes conjuntos de dados incluem densidade de tecidos, pressão, temperatura, amplitude de onda, carga eletrostática e velocidade.

Pode-se classificar as técnicas de visualização de dados volumétricos em dois grandes grupos: métodos que realizam a extração de superfícies *(surface fitting*) e métodos que implementam a renderização direta de volumes *(volume rendering*). Estas abordagens diferem, basicamente, pela utilização ou não de representações intermediárias dos dados volumétricos para a criação de imagens adequadas à aplicação.

Enquanto na renderização direta de volumes a criação da imagem é realizada diretamente a partir dos dados volumétricos, na extração de superfícies os dados volumétricos são convertidos para uma representação geométrica (polígonos) a partir da qual são empregadas técnicas tradicionais de renderização de polígonos para criar a imagem final [Paiva,1999]. O algoritmo de *Marching Cubes* [Lorensen,1987] é uma das mais conhecidas técnicas de extração de superfícies a partir de dados volumétricos, enquanto que o *Ray-Casting* [Levoy,1988] é o algoritmo clássico de renderização direta de volumes. Nesta última categoria, o algoritmo de renderização volumétrica através da fatoração *Shear-Warp*, proposto por Lacroute [Lacroute,1994], é um dos métodos mais eficientes para a visualização direta de volumes, especialmente quando estes podem ser pré-classificados e não são extremamente grandes. Um dos primeiros trabalhos na área de volumetria direta foi realizado por Debrin e outros [Debrin,1988] e apresenta as bases para a manipulação de dados volumétricos compostos por diferente tipos de materiais, empregando o algoritmo de *Ray-Casting*.

Uma descrição detalhada sobre os algoritmos de visualização volumétrica, bem como o histórico e uma introdução a esta área, podem ser encontrados na monografia

Introdução à Visualização Volumétrica [Paiva,1999].

1.1 Visualização Volumétrica Híbrida

Em diversas aplicações é importante visualizar modelos híbridos contendo dados volumétricos e objetos cuja representação natural seja através das superfícies que os delimitam. Por exemplo, inúmeras aplicações médicas analisam imagens criadas a partir da combinação de exames médicos com modelos de objetos externos. No planejamento radioterápico, os feixes de radiação podem ser modelados geometricamente, simulando sua atuação sobre o corpo do paciente através da combinação do modelo virtual com o dado médico da região ou do órgão a ser tratado [Levoy,1990a]. Na área de ortopedia, próteses de ossos das pernas e dos pés, representadas através de modelos poligonais, podem ser virtualmente posicionadas dentro do corpo do paciente. Desta maneira, o cirurgião pode planejar de forma não-invasiva os procedimentos de colocação e adequação da prótese no paciente, antes que o ato cirúrgico aconteça [Kreeger,1999; Kreeger,1999a].

Nas aplicações em que o médico interage com o modelo virtual do órgão do paciente juntamente com modelos geométricos, é necessário que o processo de criação das imagens seja rápido o suficiente para permitir a manipulação interativa. Tal interatividade é necessária para que se tenha uma resposta imediata quando o usuário modificar parâmetros, como, por exemplo, o ponto de vista em relação aos modelos virtuais, o posicionamento do modelo geométrico em relação ao dado volumétrico, entre outros. A resposta interativa é essencial para a exploração e análise dos dados médicos de forma eficiente.

Existem diferentes estratégias para a criação de imagens híbridas envolvendo a visualização do dado volumétrico e do modelo poligonal. Podemos classificar estas estratégias de acordo com a necessidade ou não de conversão dos dados para uma única representação, a fim de facilitar a etapa de combinação:

?? Estratégia baseada em superfícies: conversão da representação volumétrica para a geométrica, através da criação de isosuperfícies poligonais que correspondem às estruturas presentes no dado volumétrico. Algoritmos como o *Marching Cubes* [Lorensen,1987] são utilizados para realizar a extração das isosuperfícies a partir do dado volumétrico. Assim, isosuperfícies e

modelo poligonal são renderizados através do *pipeline* de visualização geométrica. Empregar somente a descrição por polígonos parece uma estratégia promissora, uma vez que o *pipeline* de visualização geométrica pode ser acelerado por *hardware*. No entanto, a extração de isosuperfícies tende a criar centenas de polígonos, alguns muito finos ou degenerados, exigindo que se realize uma etapa de decimação antes da visualização, o que introduz uma perda na qualidade e na precisão do modelo e um acréscimo no tempo de processamento.

?? Estratégia baseada em elementos de volume: transformação do modelo poligonal para a representação volumétrica através de algoritmos de voxelização [Kaufman,1986; Kaufman,1987; Kaufman,1988]. A representação volumétrica do dado poligonal é combinada diretamente com o dado volumétrico, empregando algoritmos como, por exemplo, o *Ray-Casting* [Levoy,1988]. Também nesta abordagem pode existir perda de precisão do modelo poligonal que está sendo convertido.

?? Estratégia baseada em primitivas de pontos: transformação do volume e do dado poligonal em primitivas do tipo ponto [Yagel, 1998]. Estas primitivas são compostas de informações como posição 3D, valor de cor; vetor normal, opacidade e outros dados necessários durante o processo de cálculo de cor. A primitiva de ponto pode ser renderizada como um ponto no espaço 3D ou através de técnicas como *Splatting* [Westover,1990], que calcula a contribuição da projeção do ponto para uma determinada vizinhança de *pixels*.

?? Estratégia híbrida: renderização de cada tipo de dado separadamente, combinando as imagens do modelo poligonal e do volume em uma única imagem final. Nesta abordagem, podem-se empregar as melhores técnicas para cada tipo de representação, pois não há a necessidade de conversão de formatos, evitando perda de precisão. Trabalhos como o *Z-Merging* [Kaufman, 1990] e o *Ray-Merging* [Levoy,1990a] apresentam algoritmos de visualização híbrida para a combinação de volumes e modelos poligonais.

Algumas aplicações específicas exigem que o tempo de resposta para a geração dos modelos e das imagens virtuais aconteça em tempo real. Por exemplo, o uso de

técnicas de realidade aumentada [Brigham,2000], visando ressaltar as estruturas de interesse no transcorrer de uma cirurgia, só é factível se a geração das imagens virtuais ocorrer em tempo real. Os dados médicos, neste caso, são adquiridos constantemente durante o procedimento cirúrgico e o modelo virtual deve ser atualizado imediatamente para que esse reflita as alterações que estão ocorrendo nas estruturas de interesse.

Diante da necessidade de interatividade e geração de imagens de qualidade, que preservem a precisão dos dados, a estratégia de renderização híbrida apresenta-se a mais adequada para o desenvolvimento de soluções para a visualização de dados volumétricos combinados com modelos poligonais. Então, nada mais natural que buscar as melhores técnicas de renderização em cada um dos tipos de dados e tentar conjugar suas vantagens em um único método híbrido de visualização volumétrica.

Algoritmos de visualização volumétrica baseados na fatoração *Shear-Warp* são capazes de tratar o volume de dados como uma seqüência de texturas 2D, sem implicar nos erros de reamostragem que ocorrem quando estas texturas são projetadas diretamente sobre a imagem, como é o caso do mapeamento de texturas 2D por *hardware* [Westermann,1998]. Para volumes de pequeno e médio porte, estes algoritmos apresentam tempos interativos para a criação de uma imagem final [Lacroute, 1994].

O algoritmo de *Z-Buffer* [Foley,1990] é um dos mais flexíveis para o tratamento de visibilidade de modelos poligonais. Diferentes modelos de iluminação podem ser integrados a ele, possibilitando a criação de imagens de qualidade e sem erros de oclusão. Por tratar-se de um algoritmo clássico para a visualização de polígonos, bibliotecas gráficas como a OpenGL [Woo,1999] e Direct3D [Kovach,2000] o implementam. Várias placas gráficas de uso comercial, que possuem preços acessíveis, aceleram em *hardware*, as funções destas bibliotecas, inclusive o *pipeline* de renderização de polígonos.

Este trabalho de tese tem por objetivo apresentar um conjunto de algoritmos que realizam a integração entre o *Shear-Warp* e o *Z-Buffer* visando a criação de imagens compostas de dados volumétricos combinados com modelos poligonais. Uma vez que o *Shear-Warp* e o *Z-Buffer* são eficientes para a visualização de dados volumétricos e de dados poligonais, respectivamente, escolheu-se estes dois algoritmos como base para o desenvolvimento de algoritmos híbridos, capazes de integrar as vantagens de cada uma

das técnicas escolhidas. Será mostrado que, estendendo o algoritmo de visualização *Shear-Warp* proposto por Lacroute [Lacroute,1995] e integrando a este o algoritmo de *Z-Buffer* [Foley,1990], pode-se combinar dados volumétricos opacos e/ou transparentes com modelos poligonais também opacos e/ou transparentes, empregando somente algoritmos implementados via *software*, criando imagens corretas e de qualidade.

Um dos problemas que devem ser tratados pelos algoritmos que combinam dados volumétricos e modelos poligonais é o *aliasing*. O problema de *aliasing* é introduzido durante o processo de reamostragem do modelo poligonal, especialmente para os casos em que a resolução do dado volumétrico é baixa e o modelo poligonal é reamostrado para esta mesma resolução.

O problema de *aliasing* é mais significativo em relação ao modelo poligonal do que ao dado volumétrico especialmente se a geometria do modelo poligonal possuir contornos ou partes detalhadas. Já no dado volumétrico, normalmente os contornos dos objetos presentes no volume possuem uma natureza mais difusa, não evidenciando o efeito de *aliasing*, mesmo que o dado seja amostrado para uma imagem de resolução inferior à resolução do dado. Levoy discute este problema de *aliasing* de dados volumétricos e modelos poligonais no contexto do algoritmo de *Ray-Casting* [Levoy,1990a].

Neste trabalho serão apresentadas e discutidas estratégias para minimizar os efeitos de *aliasing* introduzidos durante a reamostragem dos polígonos, buscando um equilíbrio entre qualidade de imagem *versus* tempo de processamento. Estas soluções são independentes de *hardware*, permitindo maior flexibilidade quanto ao tratamento de volumes de dimensões maiores e também quanto à implementação de modelos de iluminação mais complexos. Estas características, aliadas à possibilidade de uso de funções gráficas aceleradas por *hardware*, são o diferencial apresentado por este trabalho em relação às propostas já existentes para a combinação de dados volumétricos e modelos poligonais.

1.2 Trabalhos Correlatos

O algoritmo de *Z-merging* [Kaufman,1990] propõe o uso de dois *pipelines* separados para realizar a combinação de dados volumétricos e modelos poligonais: o primeiro deles trata o dado volumétrico através de um (qualquer) algoritmo de

visualização direta de volumes, enquanto o segundo *pipeline* trabalha sobre o modelo poligonal, empregando uma (qualquer) técnica de renderização de polígonos. Cada um destes processos de renderização produz duas saídas, que correspondem ao mapa de cor e opacidade e o mapa de profundidade da imagem produzida. As imagens resultantes da renderização do volume e dos polígonos são combinadas de acordo com os valores de *Z* dos mapas de profundidade correspondentes a cada um dos dados. Nesse trabalho Kaufman apresenta as linhas básicas da construção de um algoritmo híbrido que utilize a informação de profundidade para a combinação dos dados volumétricos com os poligonais. No entanto, não são propostos algoritmos específicos para cada um dos *pipelines* de renderização envolvidos.

Visando a interatividade, Lacroute e Levoy [Lacroute,1995] propuseram uma nova família de algoritmos de visualização volumétrica baseados na fatoração *Shear-Warp* capaz de aumentar a eficiência até então alcançada para esta classe de algoritmos. O método *Shear-Warp* transforma geometricamente o dado volumétrico, simplificando as etapas de projeção e composição das contribuições de cada fatia dentro do processo de criação da imagem final. Uma versão paralelizada deste algoritmo produziu, para um dado volumétrico de dimensão 256^3, uma taxa de exibição de 15 quadros/segundo, em uma Silicon Graphics Challenge com 32 processadores [Lacroute,1996].

Mais recentemente, surgiram novas abordagens para a questão da interatividade relativa aos algoritmos de visualização volumétrica. As novas técnicas fazem uso, principalmente, de placas gráficas especializadas e *hardware* para mapeamento de textura. Quando o *hardware* gráfico suporta apenas texturas 2D, cada fatia do volume pode ser processada como uma simples textura bidimensional, sendo composta na imagem final através do operador *over* [Porter,1984]. No entanto, este método produz erros de reamostragem, principalmente quando a direção de visualização não está alinhada com um dos eixos principais do volume de dados [McReynolds,1997]. Wilson, Gelder e Wilhelms [Wilson,1994] apresentaram um método que utiliza texturas 3D e é implementado com recursos da biblioteca gráfica OpenGL [Woo,1999]. Com este método podem-se obter tempos interativos na criação de imagens compostas de volumes opacos e/ou transparentes, desde que exista aceleração por *hardware* de textura 3D, como o encontrado em estações SGI *InfiniteReality*.

Osborne e outros [Osborne,1997] apresentaram o EM-Cube, uma arquitetura para placa gráfica especializada para PCs que implementa os conceitos do Cube-4

[Pfister,1996]. A placa gráfica VolumePro, apresentada no SIGGRAPH'99 [Pfister,1999], baseia-se na arquitetura Cube-4 de renderização volumétrica, incluindo as otimizações de memória presentes na arquitetura EM-Cube. O sistema é capaz de renderizar volumes com até 256^3 *voxels* com uma taxa de 30 quadros por segundo. Este novo *hardware* permite a mudança das funções de transferência de cor e opacidade em tempo real, incluindo ainda facilidades como a visualização de volumes cuja forma varia no decorrer do tempo. No entanto, a combinação com modelos poligonais não faz parte das facilidades implementadas no *hardware* da VolumePro.

Várias estratégias têm sido propostas para combinar dados volumétricos com modelos poligonais. Algoritmos de renderização volumétrica bem conhecidos, como *Ray-Casting* [Levoy,1988] e *Splatting* [Westover,1990], são combinados com algoritmos de renderização de polígonos, como *Ray-Tracing*, *Z-Buffer* e *Scan-conversion* [Foley,1990], visando criar imagens do modelo híbrido em tempo interativo e sem perda de qualidade [Kaufman,1990; Levoy,1990a; Miyazawa,1992; Tost,1993; Walsum,1993]. Estes sistemas possuem a vantagem de ultrapassar a limitação do tamanho do volume a ser tratado, presente nas propostas dependentes de *hardware* como o mapeamento de texturas e as placas gráficas especializadas. Entretanto, os sistemas implementados via *software* apresentam menores taxas de exibição de quadros do que aqueles que se utilizam das facilidades de *hardware*.

Atualmente, a maioria das placas gráficas pode eficientemente visualizar cenas baseadas em objetos poligonais empregando o algoritmo de *Z-Buffer*. A principal diferença entre estas placas diz respeito ao número de polígonos que podem ser renderizados a cada quadro, ao tamanho da memória de textura e à quantidade de mapas de texturas que podem ser manipulados pela placa. A maioria das placas gráficas atuais suportam mapeamento de texturas 2D e somente algumas possuem mapeamento de texturas 3D. Uma séria limitação destas placas diz respeito ao tamanho da memória de textura que, para a visualização de dados volumétricos de médio e grande porte, normalmente é insuficiente.

À medida que a tecnologia computacional avança, o tamanho dos dados volumétricos a serem tratados também aumenta. Assim, as melhores soluções para a manipulação destes volumes são aquelas que gerenciam eficientemente os recursos de *hardware* juntamente com implementações via *software*.

Uma solução para combinar polígonos e dados volumétricos foi proposta no HP Voxelator [Lichtenbelt,1997]. Entretanto, esta proposta trata somente volumes e polígonos opacos, empregando *pipelines* separados para renderizar cada um dos objetos. As imagens resultantes dos diferentes *pipelines* são combinadas numa única imagem final, usando a informação de profundidade do *Z-Buffer* para realizar a oclusão. O problema de combinação com volume e polígono semitransparente não foi abordado pelo HP Voxelator.

Outro sistema de visualização volumétrica, o SGI Volumizer [Eckel,1998], utiliza o *hardware* de textura 3D juntamente com o *pipeline* de renderização dos polígonos, através do *Z-Buffer*, para combinar volumes transparentes e/ou opacos com modelos poligonais somente opacos.

Kreeger e Kaufman apresentaram dois métodos de visualização volumétrica híbrida que combinam volumes e modelos poligonais através do uso do *hardware* gráfico dedicado à renderização de polígonos e do *hardware* do Cube-5 [Kreeger,1999]. Estes métodos podem combinar volumes e modelos poligonais opacos e/ou transparentes, obtendo tempos interativos para a criação das imagens finais. O primeiro método proposto compartilha um *buffer* externo DRAM entre o *hardware* do Cube-5 e a placa gráfica. O *buffer* de composição SRAM do Cube-5 não é utilizado, sendo substituído pelo *buffer* externo DRAM compartilhado. Assim, a etapa de combinação dos polígonos com cada fatia do volume é realizada diretamente na memória DRAM, à qual a placa gráfica e o Cube-5 têm acesso, agilizando este passo. O segundo método proposto aplica a técnica de *run-length encode* (RLE) [Barenholtz,1996] ao modelo poligonal, que está organizado em pequenos intervalos, utilizando um *hardware* específico juntamente com o *buffer* DRAM externo. Após esta etapa, o RLE do dado poligonal é combinado com o dado volumétrico no *buffer* de composição do Cube-5.

Outro trabalho também apresentado por Kreeger e Kaufman utiliza funções 3D da biblioteca OpenGL, tais como mapas de texturas e *pipelines* de renderização dos polígonos, para combinar volumes e polígonos que podem ser transparentes e/ou opacos [Kreeger,1999a]. São alcançados tempos interativos na criação das imagens finais através do uso de mapeamento de texturas 3D, acelerado por *hardware*.

Zakaria apresenta um algoritmo de visualização *Shear-Warp* híbrido que implementa um *Zlist-Buffer* para criar imagens combinando volumes e modelos

poligonais opacos e/ou transparentes [Zakaria,1999]. Sua estratégia para o tratamento de transparência é similar ao que abordamos nesta tese; entretanto, problemas como *aliasing* não foram discutidos, faltando também parâmetros para a comparação de resultados em termos de tempo de processamento e qualidade de imagem.

Este trabalho de tese apresenta soluções para a combinação de dados volumétricos e modelos poligonais que não necessitam de *hardware* gráfico especializado para a criação de imagens corretas, em tempos aceitáveis. Os algoritmos propostos são implementados via *software*, oferecendo maior flexibilidade, por exemplo, quanto aos modelos de iluminação empregados, ao tamanho dos volumes a serem tratados e ao uso de técnicas para aumentar a qualidade da imagem final gerada. A característica de extensibilidade também é inerente às implementações realizadas em *software* e, neste caso, ferramentas especiais de manipulação, exploração e análise dos dados podem ser integradas, sem restrições, aos algoritmos propostos.

Alguns dos trabalhos correlatos descritos acima [Kreeger,1999; Kreeger,1999a] também implementam soluções para o problema da combinação, abordando temas como transparência e *aliasing*. Entretanto, eles fazem uso de *hardware* específico para a criação das imagens, especialmente para obter tempos interativos e imagens corretas e de qualidade.

1.3 Organização da Tese

O Capítulo 2 desta tese faz uma revisão do método *Shear-Warp* proposto por Lacroute [Lacroute,1995], detalhando sua formulação algébrica e emprego de estruturas de dados que permitem a aceleração do método.

No Capítulo 3 são apresentados os algoritmos híbridos propostos, baseados nos algoritmos de *Shear-Warp* e de *Z-Buffer*, que realizam a combinação dos dados volumétricos e poligonais, enfatizando a redução dos efeitos de *aliasing*.

O Capítulo 4 apresenta os testes realizados e resultados obtidos, em termos de tempos de processamento e qualidade de imagem. Os algoritmos foram implementados com diferentes bibliotecas gráficas e executados em diferentes plataformas de *hardware*, visando realizar uma comparação do desempenho dos mesmos nestes diferentes contextos. Uma implementação utiliza a biblioteca G3D, desenvolvida pelo Tecgraf [Tecgraf,1996], onde todos os passos são realizados em *software*, ou seja, não é

necessária a presença de *hardware* gráfico especializado para que a combinação de dados volumétricos e modelos poligonais se realize corretamente. Na versão dos algoritmos que faz uso das funções da biblioteca gráfica OpenGL [Woo,1999], as funções de renderização do modelo poligonal são aceleradas por *hardware*, através da utilização de placas gráficas de uso geral. Os resultados obtidos nestes diferentes contextos são apresentados e comparados.

Por fim, o Capítulo 5 apresenta as conclusões e propostas de trabalhos a serem desenvolvidos como extensão desta tese.

2 ALGORITMO *SHEAR-WARP*

Os algoritmos de visualização volumétrica podem ser classificados, segundo Kaufman [Kaufman,1999], de acordo com a ordem com que atravessam o volume e com o método empregado para projetar os *voxels* na imagem.

Seguindo esta classificação, duas classes merecem destaque dentro do contexto desta tese:

- visualização a partir da imagem (*image order*);
- visualização a partir do objeto (*object order*).

Os algoritmos de visualização a partir da imagem realizam uma reamostragem do volume a partir dos *pixels* da imagem a ser gerada. O algoritmo clássico desta categoria é o *Ray-Casting* [Levoy,1988; Sabella,1988; Upson,1988, Sobierajski,1994], no qual, para cada *pixel* da imagem que está sendo criada, é lançado um raio que atravessa o volume, acumulando as contribuições de cor e opacidade de cada *voxel* percorrido.

Uma técnica de aceleração empregada para algoritmos de visualização a partir da imagem é o *early ray termination* [Levoy,1990], no qual o raio pára de percorrer o volume quando o *pixel* ao qual o raio está associado estiver completamente opaco. A idéia desta otimização é reduzir ou eliminar amostragens em regiões "ocultas" do volume. Assim, os raios não atravessam necessariamente todas as *n* fatias para compor a imagem final. Técnicas como a da tela dinâmica [Reynolds,1987] propõem o uso de *run-length encode* da imagem que está sendo criada para "saltar" trechos de *pixels* que já estão completamente opacos, acelerando ainda mais o processo de criação da imagem final.

São classificados como algoritmos de visualização a partir do objeto aqueles em que os *voxels* são projetados diretamente sobre a imagem que está sendo gerada. Como representante desta classe, tem-se o algoritmo de *Splatting* [Westover,1989; Westover,1990], no qual cada *voxel* é projetado sobre a imagem, contribuindo para uma certa vizinhança de *pixels*, determinada através de um filtro de "espalhamento". Podem ser empregadas estruturas de dados espaciais para acelerar os algoritmos de visualização a partir do objeto. Estas estruturas fazem uso da coerência dos dados e do fato de que o volume é processado na sua ordem de armazenagem. Por exemplo, estruturas como *run-length encode* do volume [Montani,1990] podem ser empregadas para armazenar somente os *voxels* mapeados como não transparentes, saltando trechos transparentes, reduzindo assim o tempo de renderização.

Os algoritmos baseados na fatoração *Shear-Warp*, propostos por Lacroute [Lacroute,1995], podem ser enquadrados numa terceira categoria, a de algoritmos híbridos [Paiva,1999]. Nestes algoritmos é realizada a fatoração da matriz de visualização, operação que simplifica o processo de projeção dos *voxels* sobre a imagem que está sendo criada. Esta simplificação permite empregar técnicas de aceleração intrínsecas aos métodos baseados no objeto, bem como técnicas de aceleração associadas aos algoritmos baseados na imagem. Assim, podem ser implementados algoritmos eficientes a partir da fatoração *Shear-Warp* combinada com técnicas como *run-length encode* e *early ray termination*. Devido a estas características, nesta tese optou-se por implementar um algoritmo baseado na fatoração *Shear-Warp* para realizar a renderização dos dados volumétricos.

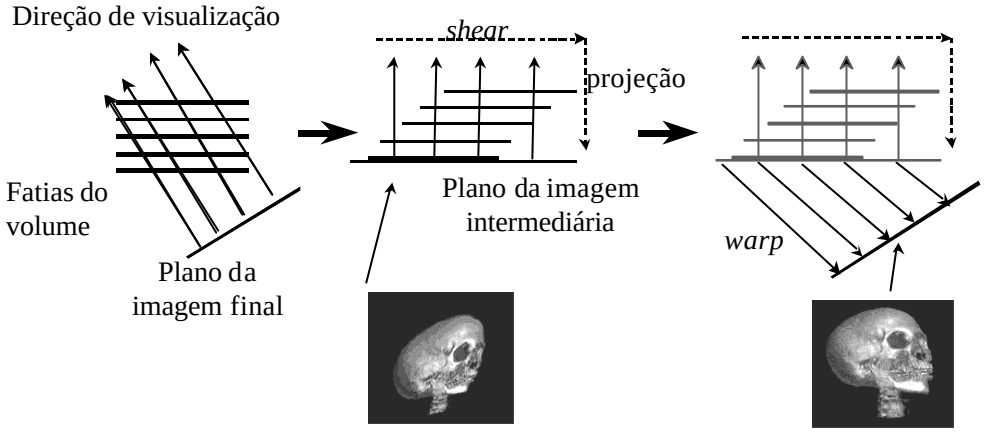
A seguir, a fatoração *Shear-Warp* é descrita em detalhes, incluindo sua derivação matemática e passos básicos para o desenvolvimento de algoritmos de renderização de volumes. Também são detalhadas técnicas de aceleração que podem ser incorporadas ao algoritmo de renderização.

2.1 Fatoração *Shear-Warp*

O método *Shear-Warp* baseia-se na fatoração da matriz de visualização em uma operação de *shear 3D*, que realiza um cisalhamento (*shear*) paralelo às fatias do volume; uma projeção, que forma uma imagem intermediária distorcida; e uma operação de *warp 2D*, que transforma a imagem distorcida, produzindo a imagem final.

A matriz de visualização é derivada a partir dos parâmetros de posição do observador, do tipo de projeção a ser empregada, das dimensões do volume e do tamanho da imagem final a ser gerada. Cabe ressaltar que os algoritmos propostos neste trabalho empregam somente projeções paralelas, motivo pelo qual as derivações apresentadas a seguir fazem uso de matrizes de transformação e projeção adequadas a este contexto. No entanto, no trabalho de Lacroute [Lacroute,1995] são apresentadas as derivações e matrizes empregadas para a fatoração *Shear-Warp* com projeção perspectiva, cujo emprego é discutido na seção de trabalhos futuros desta tese.

A Figura 2.1 mostra o esquema genérico do algoritmo de *Shear-Warp* para projeções paralelas. As linhas horizontais representam as fatias do volume.



$$M_{view} = M_{Warp2D} * M_{shear3D}$$

Figura 2.1 – Esquema da fatoração *Shear-Warp* [Lacroute,1995]

O algoritmo de *Shear-Warp* trabalha reduzindo o problema de renderizar um determinado volume, através de uma projeção paralela genérica, para o caso especial de projeção ao longo de um dos eixos de orientação do volume. Nesse caso específico, pode-se obter a imagem final projetada através de um procedimento que consiste em dois passos distintos.

No primeiro passo, as contribuições das fatias que compõem o volume são acumuladas, de frente para trás (usando o operador *over* [Porter,1984]), valendo-se do fato de que *voxels* da mesma posição (*x*,*y*) dentro do volume permanecem sempre sobre o mesmo raio de projeção. O resultado deste passo é a criação de uma imagem intermediária que possui as mesmas dimensões que as das fatias do volume.

No segundo passo, a imagem intermediária é apropriadamente transformada, através de uma operação de *warp*, para produzir a imagem final. Neste caso, em que se

projeta ao longo de um dos eixos do volume, o passo de *warp* trata, simplesmente, de aplicar uma escala sobre a imagem intermediária para alcançar a resolução da imagem final.

Para que se possa reduzir o caso da projeção paralela genérica para a situação descrita acima, o volume deve sofrer transformações de acordo com os quatro sistemas de coordenadas apresentados na Figura 2.2: sistema de coordenadas do objeto, sistema de coordenadas *standard*, sistema de coordenadas de *shear*, e sistema de coordenadas da imagem final.

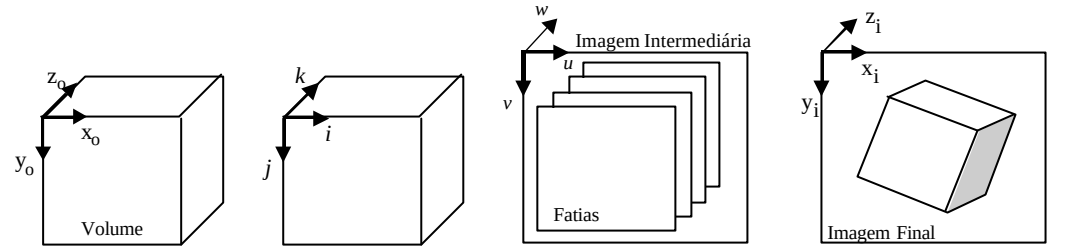


Figura 2.2 – Sistemas de coordenadas empregados na fatoração *Shear-Warp* [Lacroute,1995]

O sistema de coordenadas do objeto é o sistema natural do dado volumétrico. A origem está localizada em um dos cantos do volume (normalmente, no canto superior esquerdo da primeira fatia). A unidade de distância ao longo de cada um dos eixos é igual ao comprimento do *voxel* ao longo daquele eixo. Os eixos são chamados de x_o, y_o, z_o.

Para formar o sistema de coordenadas *standard*, os eixos do sistema de coordenadas do objeto são permutados de forma que o eixo principal de visão torne-se o terceiro eixo coordenado. O eixo principal de visão é o eixo do espaço do objeto que está mais paralelo à direção de visualização. Os eixos do sistema de coordenadas *standard* são chamados *i, j, k*, onde *k* corresponde ao eixo principal de visão.

O sistema de *shear* é formado através da transformação do sistema de coordenadas *standard* pela matriz de *shear*, obtida da fatoração da matriz de visualização. Este sistema de coordenadas é também o sistema de coordenadas da imagem intermediária distorcida. Sua origem está localizada no canto superior esquerdo da imagem intermediária e seus eixos são chamados *u, v, w*.

Finalmente, o sistema de coordenadas da imagem final é obtido a partir da transformação do sistema de coordenadas de *shear* pela matriz de *warp*, obtida através

da fatoração da matriz de visualização. A origem do sistema de coordenadas da imagem final está localizada no canto superior esquerdo desta imagem. Os eixos são chamados x_i, y_i e z_i.

Descrevendo estas transformações em termos de matrizes, elas equivalem à fatoração da matriz de visualização M_{view} conforme a expressão abaixo:

$$M_{view} = M_{view} \; M_{shear}^{-1} \; M_{shear} = M_{warp2D} \; M_{shear} \; [P]. \qquad (2.1)$$

A matriz de visualização M_{view} é uma matriz 4x4 que transforma pontos do espaço do objeto para o espaço da imagem final, como apresentado a seguir:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \\ w_i \end{bmatrix} = M_{view} \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix}.$$

Nas próximas seções, serão detalhadas as derivações necessárias para que se chegue à equação (2.1). Nestas derivações, $\vec{v}$ representa um vetor e v_x representa uma componente de $\vec{v}$. A formulação matemática e os conceitos apresentados a seguir foram extraídos da tese elaborada por Lacroute sobre a fatoração *Shear-Warp* [Lacroute,1995].

2.1.1 Determinando o eixo principal de visão

Para que se possa encontrar a matriz de permutação [P] correta, que realize a transformação do espaço do objeto para o espaço *standard*, é necessário, primeiramente, que se determine o eixo principal de visão. Este é o eixo do espaço do objeto que forma o menor ângulo com o vetor de direção de visualização. No espaço da imagem, o vetor de direção de visualização é dado por

$$\vec{v}_i = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Fazendo $\vec{v}_o$ ser o vetor de direção de visualização transformado para o espaço do objeto, pode-se montar o seguinte sistema linear:

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} v_{o,x} \\ v_{o,y} \\ v_{o,z} \end{bmatrix},$$

onde m_{ij} são elementos da matriz de visualização M_{view}. Assim, pela regra de Cramer, a solução para este sistema linear é dada através do vetor $\vec{v}_o$, onde cada uma das suas componentes é dividida por um determinante. Contudo, como o determinante é o mesmo para todas as componentes, este pode ser eliminado, resultando na seguinte expressão:

$$\vec{v}_o = \begin{bmatrix} m_{12}m_{23} - m_{22}m_{13} \\ m_{21}m_{13} - m_{11}m_{23} \\ m_{31}m_{22} - m_{21}m_{12} \end{bmatrix} .$$

Os cosenos dos ângulos entre o vetor de direção de visualização e os eixos do espaço do objeto são proporcionais ao produto escalar de $\vec{v}_o$ com cada um dos vetores unitários no espaço do objeto. O maior produto escalar corresponde ao menor ângulo. Então, encontra-se o eixo principal de visão através da seguinte expressão:

$$c = \max(|v_{o,x}|, |v_{o,y}|, |v_{o,z}|) \quad .$$

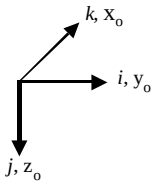
Se $c=|v_{o,x}|$ for o maior, então o eixo principal de visão é o eixo x_o do sistema de coordenadas do objeto. Se $c=|v_{o,y}|$, então y_o é eixo principal de visão; caso contrário z_o é o eixo escolhido.

2.1.2 Transformação para o sistema de coordenadas *standard*

O algoritmo de *Shear-Warp* trabalha compondo o conjunto das fatias do volume que é perpendicular ao eixo principal de visão. Para evitar o tratamento de casos especiais, para cada um dos três eixos coordenados do espaço do objeto, é realizada a transformação destes para o sistema de coordenadas *standard*. Neste sistema, o eixo de composição, equivalente ao eixo principal de visão, é sempre o eixo *k*.

Esta transformação é feita através da permutação dos eixos do sistema do objeto. A matriz de permutação [P] é escolhida em função do eixo principal de visão. Se o eixo for x_o, então [P] é dada por:

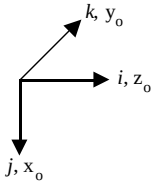
$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



O diagrama ao lado da matriz mostra a correspondência entre os eixos do sistema de coordenadas do objeto e o sistema de coordenadas *standard*.

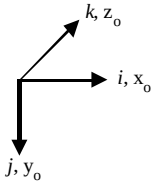
Se o eixo for y_o, então [P] é dada por:

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Finalmente, se o eixo for z_o, então [P] é a matriz identidade:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Em qualquer caso, a transformação do sistema de coordenadas do objeto para o sistema de coordenadas *standard* é expressa por:

$$\begin{bmatrix} i \\ j \\ k \\ 0 \end{bmatrix} = P \begin{bmatrix} x_o \\ y_o \\ z_o \\ 0 \end{bmatrix}$$

A matriz que expressa a transformação de visualização a partir do sistema de coordenadas *standard* será denotada por M'_{view}. Assim, pode-se expressar a matriz de visualização permutada M'_{view} como sendo:

$$M'_{view} = M_{view}\, P^{-1} \qquad (2.2)$$

Esta matriz transforma pontos do sistema de coordenadas *standard* para o sistema de coordenadas da imagem final.

2.1.3 Cálculo dos fatores de *shear* e *warp*

O próximo passo na derivação trata da fatoração da matriz de visualização permutada, M'_{view}, em uma operação de cisalhamento (*shear*) nas direções i e j, seguida da operação de *warp*. A fatoração deve satisfazer a seguinte condição: depois da transformação de *shear*, o vetor $\vec{v}_{so}$, que representa a direção de visualização no sistema de coordenadas *standard*, deve estar perpendicular ao plano (i, j). No sistema de coordenadas *standard*, o vetor de direção de visualização, denotado por $\vec{v}_{so}$, é dado por

$$\vec{v}'_{so} = P\vec{v}_{so} = \begin{bmatrix} m'_{12}m'_{23} - m'_{22}m'_{13} \\ m'_{21}m'_{13} - m'_{11}m'_{23} \\ m'_{31}m'_{22} - m'_{21}m'_{12} \end{bmatrix},$$

onde m'_{ij} são elementos da matriz de visualização permutada M'_{view}.

Assim, buscamos uma transformação de cisalhamento perpendicular ao eixo k, cuja matriz é da forma:

$$S = \begin{bmatrix} 1 & 0 & s_i \\ 0 & 1 & s_j \\ 0 & 0 & 1 \end{bmatrix},$$

onde os fatores de *shear* s_i e s_j devem ser calculados de modo que

$$S\vec{v}'_{so} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Resolvendo este sistema, encontramos que os fatores s_i e s_j são calculados através das seguintes expressões:

$$s_i = -\frac{v_{so,i}}{v_{so,k}} = \frac{m'_{22}m'_{13} - m'_{12}m'_{23}}{m'_{11}m'_{22} - m'_{21}m'_{12}} \quad \text{e}$$

$$s_j = -\frac{v_{so,j}}{v_{so,k}} = \frac{m'_{11}m'_{23} - m'_{21}m'_{13}}{m'_{11}m'_{22} - m'_{21}m'_{12}}. \tag{2.3}$$

A interpretação geométrica dos fatores de cisalhamento nas direções (i, j) é

apresentada na Figura 2.3.

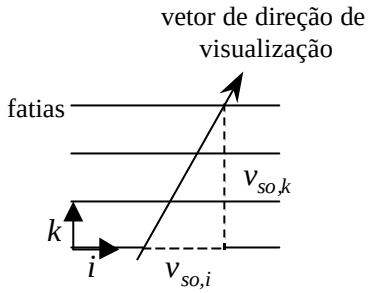


Figura 2.3 – Projeção do vetor de direção de visualização sobre o plano (*i*, *k*) [Lacroute,1995]

A projeção do vetor de direção de visualização sobre o plano (*i*, *k*) tem uma inclinação dada por $\frac{v_{so,i}}{v_{so,k}}$, conforme mostra a Figura 2.3. O fator de *shear*, na direção *i*, necessário para tornar o vetor de direção perpendicular aos planos das fatias, fixos em *k*, é o oposto do valor da inclinação da projeção deste vetor sobre o plano (*i*, *k*). Analogamente, pode-se encontrar o fator de *shear* para a direção *j*, determinando-se o valor da inclinação da projeção do vetor sobre o plano (*j*, *k*).

Pode-se, então, escrever a fatoração *Shear-Warp* da matriz de visualização permutada da seguinte forma:

$$M'_{view} = \begin{bmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} m'_{11} & m'_{12} & (m'_{13} + s_i m'_{11} + s_j m'_{12}) & m'_{14} \\ m'_{21} & m'_{22} & (m'_{23} + s_i m'_{21} + s_j m'_{22}) & m'_{24} \\ m'_{31} & m'_{32} & (m'_{33} + s_i m'_{31} + s_j m'_{32}) & m'_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

O operando à esquerda é a matriz de *warp* e o da direita é a matriz de *shear*. Estas matrizes têm a propriedade de que, depois de aplicar a transformação de *shear* ao volume, é possível criar a imagem intermediária distorcida através da projeção das *k*-fatias sobre a imagem posicionada no plano *k*=0. Após a criação desta imagem, pode-se aplicar o *warp*2D sobre a mesma, gerando a imagem final.

2.1.4 Criação da imagem intermediária

Aplicando-se a transformação de *shear*, cujos fatores foram derivados em (2.3), o volume é transformado do sistema de coordenadas *standard* para o espaço de *shear*. No entanto, este espaço não é conveniente para realizar a etapa de projeção, porque sua origem não está localizada em um dos cantos da imagem intermediária. Assim, é necessário transladar o sistema de coordenadas de *shear* para reposicionar a origem para o canto superior esquerdo da imagem intermediária. A transformação de *shear*, seguida da operação de translação, produz o sistema de coordenadas da imagem intermediária.

A Figura 2.4 ilustra os quatro casos possíveis e a formulação correspondente, para calcular os fatores de translação. Cada caso é identificado através do sinal dos fatores de *shear*.

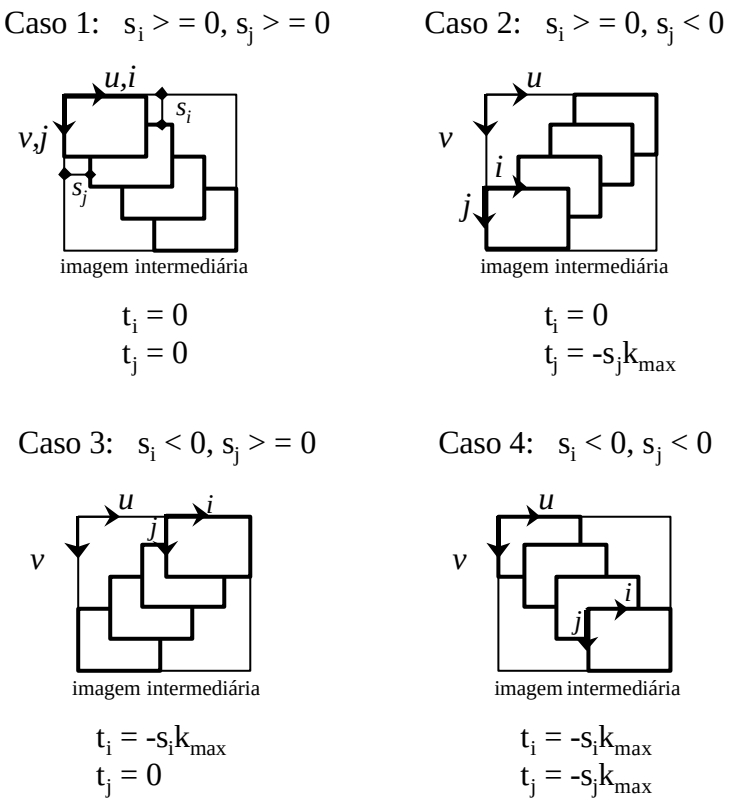


Figura 2.4 – Possíveis casos para o cálculo dos fatores de translação [Lacroute,1995]

Deve-se especificar, ainda, em que ordem as fatias serão percorridas durante o processo de composição da imagem intermediária. As fatias estão ordenadas através da sua coordenada *k* no espaço *standard*. Compor as fatias de frente para trás pode corresponder a percorrê-las numa ordem crescente ou decrescente, dependendo da direção de visualização. Para encontrar a ordem segundo a qual as fatias devem ser

percorridas, deve-se examinar a componente do vetor de direção de visualização que corresponde ao eixo principal de visualização, ou seja, $v_{so,k}$. Se $v_{so,k}$ for positivo, então a fatia frontal é a que está no plano k=0; caso contrário, é a fatia do plano $k=k_{max}$.

Uma vez determinados os fatores de translação, podem-se rescrever os fatores das matrizes de *shear* e *warp*. A matriz de *shear*, que transforma do sistema de coordenadas *standard* para o sistema de coordenadas da imagem intermediária, é dada por:

$$M_{shear} = \begin{bmatrix} 1 & 0 & 0 & t_i \\ 0 & 1 & 0 & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & s_i & t_i \\ 0 & 1 & s_j & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.4)$$

A matriz de *warp*, que transforma o sistema de coordenadas da imagem intermediária para o da imagem final, é dada por:

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

Uma vez que a transformação de *warp* é aplicada sobre uma imagem 2D localizada no plano w=0, podem-se remover a terceira linha e a terceira coluna da matriz de *warp*:

$$M_{warp2D} = \begin{bmatrix} m'_{11} & m'_{12} & (m'_{14} + t_i m'_{11} + t_j m'_{12}) \\ m'_{21} & m'_{22} & (m'_{24} + t_i m'_{21} + t_j m'_{22}) \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.5)$$

A transformação entre o sistema da imagem intermediária e o da final pode, então, ser expressa por:

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = M_{warp2D} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

2.1.5 Passos básicos da fatoração

Resumindo, os seguintes passos são necessários para calcular a fatoração *Shear-Warp*, para o caso de projeções paralelas:

1. Encontrar o eixo principal de visão e escolher a matriz de permutação correspondente para transformar o volume do espaço do objeto para o *standard.*
2. Calcular a matriz de visualização permutada M'_{view}, que representa o espaço *standard*, a partir das matrizes M_{view} e P.
3. Calcular os coeficientes de *shear*, s_i e s_j, a partir da matriz de visualização permutada M'_{view}.
4. Calcular os coeficientes t_i e t_j, que transladam a origem do sistema de coordenadas *standard* para o sistema de coordenadas da imagem intermediária.
5. Calcular a matriz de *shear*, M_{shear}, e a matriz de *warp*, M_{warp}.

2.1.6 Propriedades da fatoração *Shear-Warp*

O volume representado no espaço de *shear* possui várias propriedades geométricas fundamentais para a simplificação da implementação dos algoritmos de renderização volumétrica:

Propriedade 1: as linhas de *voxels* do volume são paralelas às linhas de *pixels* da imagem intermediária.

Propriedade 2: todos os *voxels* de uma dada fatia do volume são submetidos aos mesmos fatores ponderadores, quando da reamostragem para o cálculo dos valores de cor e opacidade a serem acumulados na imagem intermediária.

Propriedade 3: cada fatia do volume possui o mesmo fator de escala que as demais fatias, quando projetada na imagem intermediária. Para o algoritmo de *Shear-Warp* escolheu-se o fator de escala unitário, de forma que, para uma dada linha de *voxels*, existe um mapeamento de um-para-um entre os *voxels* e os *pixels* da imagem intermediária.

A implicação mais importante destas propriedades para projeções paralelas é que todos os *voxels* de uma dada fatia possuem os mesmos ponderadores de reamostragem,

que serão empregados na etapa de projeção. Como cada fatia é simplesmente transladada pela operação de cisalhamento (*shear*), o conjunto de ponderadores pode ser pré-calculado e reutilizado durante a reamostragem e projeção de cada *voxel* de uma dada fatia. Esta propriedade elimina o problema de realização de uma reamostragem eficiente, existente nos algoritmos de renderização de volumes a partir do objeto [Lacroute,1995].

2.2 Imagem Intermediária

É importante ressaltar que nos algoritmos que implementam a fatoração *Shear-Warp*, a resolução da imagem intermediária depende da dimensão do volume e da direção de projeção; ela não depende da dimensão da imagem final a ser criada. Esta propriedade é fundamental para a etapa de composição do algoritmo *Shear-Warp*, pois a complexidade desta fase é proporcional à maior dimensão do volume, e não à dimensão da imagem final.

As dimensões da imagem intermediária, em *pixels*, são calculadas da seguinte forma:

$$\begin{matrix} w_{Low} ? w_{Slice} ? d_{Slice} ? s_i \\ h_{Low} ? h_{Slice} ? d_{Slice} ? s_j \end{matrix} , \qquad (2.6)$$

onde w_{Slice} e h_{Slice} são, respectivamente, a largura e a altura de cada fatia; d_{Slice} é o número de fatias e s_i e s_j são os coeficientes de *shear* calculados a partir da matriz de permutação do eixo de visualização (vide seção 2.1.3).

O alinhamento das linhas do volume com as da imagem intermediária, representado na Figura 2.5, ilustra o emprego da Propriedade 1 (vide seção 2.1.6) para evitar cálculos desnecessários durante o processo de composição das fatias na imagem intermediária.

Quando um filtro de reconstrução bilinear é utilizado durante a etapa de reamostragem do volume, um *voxel* pode contribuir para até quatro *pixels* da imagem intermediária e, conseqüentemente, um *pixel* recebe, usualmente, a contribuição de quatro *voxels*. Assim, *pixels* vizinhos em uma linha da imagem recebem a contribuição de *voxels* vizinhos de uma linha do volume.

Além disto, se as fatias são compostas na ordem de frente para trás, à medida que os *pixels* da imagem intermediária tornam-se opacos (? =1), eles param de receber

novas contribuições de *voxels* (otimização de *early ray termination*). Por outro lado, *voxels* transparentes (? =0) podem ser ignorados, uma vez que eles não têm qualquer contribuição a acrescentar (otimização de *run-length encode* do volume). Na Figura 2.5, somente dois *pixels*, chamados *pixels modificados*, irão receber contribuições dos *voxels* da fatia.

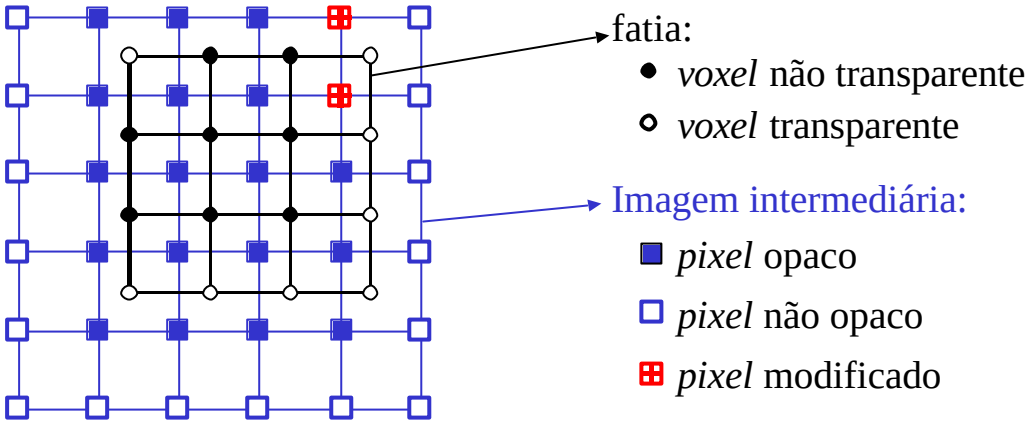


Figura 2.5 – Composição das fatias no algoritmo de *Shear-Warp*

Podem-se, ainda, empregar técnicas de *run-length encode* para a imagem intermediária. Nas estruturas do *run-length encode* da imagem são armazenadas, para cada *pixel*, informações sobre quantos *pixels* opacos vizinhos podem ser "pulados" até que o próximo *pixel* não opaco seja encontrado [Lacroute,1995].

Quando a dimensão das fatias do volume for menor que a da imagem final, a transformação de *warp* não somente corrige as distorções da imagem intermediária, mas também reamostra esta imagem para a resolução da imagem final.

Quando a dimensão das fatias do volume for maior que a da imagem final, a imagem intermediária criada também terá uma resolução maior que a da imagem final. Para estes casos, a transformação de *warp* deve igualmente corrigir as distorções introduzidas pela operação de *shear* durante a criação da imagem intermediária. No entanto, a etapa de reamostragem para a resolução da imagem final deve realizar uma superamostragem na imagem intermediária, usando uma transformação que contabilize de forma inteligente as contribuições dos diversos *pixels* da intermediária para um único *pixel* da imagem final. Esse tipo de problema é comum em dados volumétricos muito grandes, como por exemplo, dados sísmicos, onde a dimensão no eixo Y de cada fatia normalmente assume valores acima de 2000 amostras.

Neste trabalho, a transformação de *warp* realiza uma reamostragem simples da

imagem intermediária para a imagem final. Ou seja, para cada *pixel* da imagem final, é contabilizada a contribuição de somente um *pixel* da imagem intermediária. Assim, para dados volumétricos cujas fatias possuem dimensões maiores que a imagem final haverá uma perda de informação na hora da transformação da imagem intermediária em imagem final. O aperfeiçoamento da transformação *warp* para o tratamento de dados volumétricos de dimensões grandes está fora do escopo deste trabalho.

2.3 Estruturas de *Run-length Encod*e

A Propriedade 1 da fatoração *Shear-Warp*, apresentada na seção 2.1.6, ressalta que as linhas dos *voxels* do volume, descrito no espaço de *shear*, estão alinhadas com as linhas dos *pixels* da imagem intermediária. Isso implica que as estruturas do volume e da imagem intermediária possam ser percorridas simultaneamente.

Para o desenvolvimento de algoritmos de renderização otimizados, é natural que se empreguem estruturas de dados que se beneficiem desta correspondência, aproveitando também a coerência dos dados. O *run-length encode* apresenta um conjunto de estruturas que se enquadram neste contexto [Foley,1990].

O RLE do dado volumétrico [Montani,1990; Lacroute,1995] é construído a partir do volume classificado através da função de transferência de opacidade fornecida pelo usuário. A partir desta classificação, podem-se distinguir os *voxels* transparentes daqueles que possuem alguma contribuição a ser acumulada na imagem. O RLE do volume é construído em uma etapa de pré-processamento, de forma a armazenar somente os *voxels* não transparentes, permitindo que se "saltem" as porções transparentes dentro de uma dada linha do volume. Esta estrutura tira vantagens da coerência do dado volumétrico para descartar as porções transparentes do mesmo.

Uma linha do volume representada através do RLE consiste de uma seqüência de grupos, transparentes e não transparentes, organizados em forma de lista, e de um valor de dado associado a cada grupo não transparente. O algoritmo de renderização atravessa o RLE do volume, fatia por fatia e para cada fatia, o algoritmo emprega os fatores de *shear* para calcular o deslocamento da mesma em relação a origem da imagem intermediária. Então, o algoritmo percorre os grupos de cada linha, transladando e reamostrando somente os *voxels* presentes nos grupos não transparentes, compondo-os na imagem intermediária.

Empregando o algoritmo descrito acima, o *Shear-Warp* realiza a etapa de composição sem a decodificação do RLE do volume e, ainda, elimina o trabalho nas porções transparentes do volume.

Na seção 4.4.1, as estruturas de dados empregadas para a implementação do RLE do volume são detalhadas, juntamente com a apresentação dos tempos de processamento gastos para a construção das mesmas.

Também é construída uma estrutura RLE para a imagem intermediária [Reynolds,1987; Lacroute,1995; Barenholzt,1996], tirando proveito da coerência entre os *pixels*. O RLE da imagem intermediária armazena, para cada *pixel*, um elo para o próximo *pixel* não opaco. Este elo sempre aponta para um *pixel* não opaco dentro de uma mesma linha da imagem. Um *pixel* é considerado opaco quando o seu valor de opacidade acumulado alcança um limite especificado. Assim, através do RLE da imagem é possível "saltar" regiões de *pixels* já completamente opacos, processando novas contribuições somente para aqueles que ainda podem ser influenciados. Esta estrutura de RLE é montada e atualizada em tempo de processamento, à medida que a imagem intermediária está sendo gerada.

Associando-se estas duas estruturas de RLE do volume e da imagem intermediária, desenvolveu-se um algoritmo rápido para processamento das linhas do volume, conforme mostra a Figura 2.6.

Empregando-se o RLE do volume, pode-se caminhar através das linhas do volume e da imagem simultaneamente, evitando-se o desperdício de trabalho. Esta simultaneidade possibilita "saltar" *voxels* transparentes e, ao mesmo tempo, "saltar" *pixels* já opacos. Desta forma, é realizado o trabalho de cálculo somente para aqueles *voxels* que são, simultaneamente, não transparentes e visíveis. Como existe a correspondência de um-para-um entre os *voxels* de uma linha do volume e os *pixels* de uma linha da imagem intermediária (vide Propriedade 3, seção 2.1.6), se um *voxel* for transparente, então o *pixel* correspondente pode ser "saltado"; da mesma forma, se um *pixel* já estiver opaco, o *voxel* correspondente não precisa ser processado.

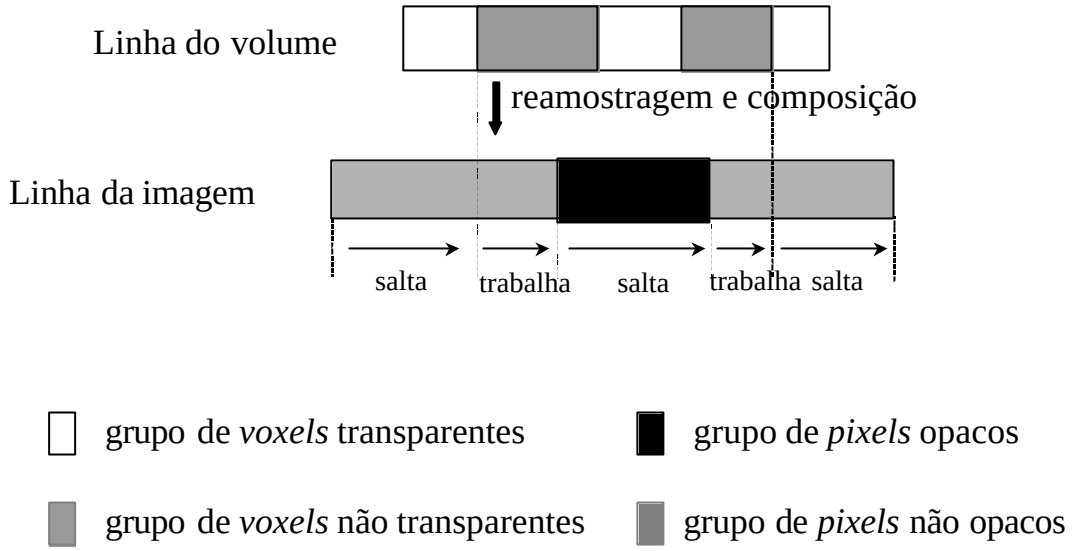


Figura 2.6 –Utilização de RLE do volume e da imagem intermediária [Lacroute,1995]

2.4 Quantização do Gradiente

A informação do vetor gradiente é utilizada durante a etapa de tonalização do dado volumétrico, representando o vetor normal à superfície definida em cada *voxel*. O vetor normal em cada ponto do volume é definido como sendo o vetor unitário paralelo ao gradiente local, determinado em função do valor escalar associado ao *voxel*, de forma que:

$$n(x, y, z) \; ? \; \frac{? \, d(x, y, z)}{\left| ? \, d(x, y, z) \right|} \; .$$

O gradiente é estimado empregando-se a regra de diferenças centrais:

$$\begin{aligned} ? \, d(x, y, z) \; ? \; & \frac{1}{2} \; ?d(x ? 1, y, z) - d(x - 1, y, z ?i \\ ? \; & \frac{1}{2} \; ?d(x, y ? 1, z) - d(x, y - 1, z) ? j \\ ? \; & \frac{1}{2} \; ?d(x, y, z ? 1) - d(x, y, z - 1) ?k \end{aligned}$$

Cada componente do vetor gradiente é representada por um número em ponto flutuante, o que, em termos de implementação, significa alocar 4 *bytes* para cada componente do gradiente. Dependendo das dimensões do dado volumétrico, o volume de gradientes pode atingir proporções inviáveis para ser mantido em memória durante o processo de tonalização dos *voxels*.

Visando reduzir a quantidade de memória ocupada com o volume de gradientes,

realiza-se uma quantização dos vetores originalmente calculados. O processo de quantização consiste em mapear os vetores gradientes originais para um conjunto menor de vetores normais, que foram uniformemente distribuídos sobre a superfície de uma esfera.

Desta forma, uma tabela de vetores normais é previamente calculada e, durante o processo de quantização, encontra-se o representante no conjunto de normais do qual o vetor gradiente mais se aproxima. Esta verificação é feita através do cálculo do ângulo entre o vetor gradiente e o vetor normal; o que apresentar o menor ângulo é o vetor normal que melhor representa o gradiente original.

O índice da tabela de vetores normais é armazenado no lugar do vetor gradiente original, reduzindo assim, sua representação. Para os testes apresentados no Capítulo 4, empregou-se uma quantização para uma tabela com 256 vetores normais pré-calculados. Com isto, é necessário somente 1 *byte* para representar o índice do vetor normal correspondente a cada gradiente, ao invés dos 4 *bytes* por componente do gradiente original.

Para se computar previamente a tabela de vetores normais, empregou-se o programa *distribute* [Leech,1992], que apresenta uma solução para o problema de distribuir uniformemente *N* pontos sobre a superfície de uma esfera. A solução simula um sistema de pontos mutuamente repulsivos, onde uma determinada distância deve ser mantida em relação a cada um dos pontos vizinhos, criando uma distribuição uniforme dos pontos.

Outras estratégias de quantização podem ser empregadas para a redução da representação do vetor gradiente. No trabalho apresentado por Lacroute [Lacroute,1995], por exemplo, os vetores gradientes são quantizados através de uma tabela de vetores normais com 8192 entradas; neste caso, são necessários 13 *bits* para representar o índice associado a cada vetor normal.

A quantização dos vetores gradientes associados a um determinado dado volumétrico é realizada em uma etapa de pré-processamento, uma única vez para cada volume, sendo salvos em disco os valores dos índices da tabela dos vetores normais encontrados para representar cada gradiente. A cada nova seção de visualização, os gradientes quantizados são carregados em memória e empregados durante o tonalização dos *voxels*.

No Capítulo 4, seção 4.4.2, são apresentados os tempos gastos com a quantização dos gradientes para os dados volumétricos empregados nos testes. Também são apresentadas considerações a respeito da qualidade da imagem gerada a partir dos gradientes quantizados.

3 *SHEAR-WARP* E MODELOS POLIGONAIS

A renderização volumétrica baseada na fatoração *Shear-Warp* compõe as contribuições das fatias do volume numa imagem intermediária distorcida que depois será transformada na imagem final através da operação de *warp*, conforme detalhado no Capítulo 2.

Durante o processo de composição, as fatias do volume são projetadas no plano da imagem intermediária, uma a uma, obedecendo sua ordenação em profundidade. Durante a composição das fatias, um filtro de interpolação bilinear é empregado para obter o peso correto de cada *voxel* para seu *pixel* correspondente na imagem intermediária (vide seção 2.2).

Para que se possam combinar as contribuições do modelo poligonal junto às do volume, é necessário que o modelo seja integrado ao processo de criação da imagem intermediária. Para tanto, o modelo deve ser rasterizado e a profundidade de cada *pixel* correspondente aos polígonos deve ser armazenada. A partir desta informação de profundidade, cada ponto pertencente ao polígono pode ser corretamente posicionado em relação às fatias do intervalo [*Z-1, Z*], que contém o referido ponto.

O algoritmo *Z-Buffer* [Foley,1990] renderiza modelos poligonais, gerando informações de cor, opacidade e profundidade (*Z*) para cada *pixel* da imagem sobre a qual os polígonos são projetados. A matriz de profundidade é empregada como uma máscara para determinar quais *voxels* serão combinados com os polígonos. Assim, as matrizes de cor e opacidade resultantes do *Z-Buffer* são combinadas com as informações de cor e opacidade em cada *voxel*. A Figura 3.1 mostra o esquema genérico da combinação de dados volumétricos com modelos poligonais, utilizando os algoritmos de *Shear-Warp* e *Z-Buffer*.

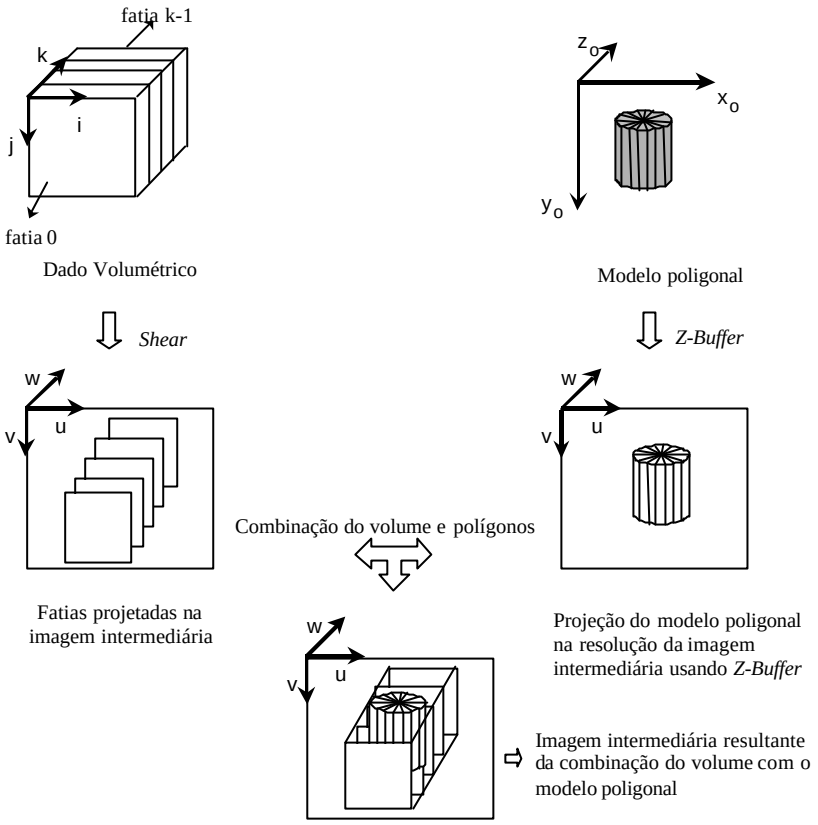


Figura 3.1 – Esquema para a combinação de volumes e modelos poligonais

O algoritmo híbrido básico, que integra o *Shear-Warp* e o *Z-Buffer*, combina as contribuições dos fragmentos de polígonos que estão entre duas fatias do volume na imagem intermediária à medida que a composição das fatias é efetuada. O passo de *warp* permanece inalterado.

Esta idéia de acumular a contribuição do modelo poligonal a medida que as fatias do volume vão sendo compostas não é uma idéia nova. No trabalho de Lacroute [Lacroute,1994] já existe um esboço desta proposta de combinação dos dados volumétricos e modelos poligonais durante a etapa de criação da imagem intermediária. No trabalho de Zakaria [Zakaria,1999] esta idéia também é discutida e implementada dentro do escopo do algoritmo de *Shear-Warp*.

Para que se possa realizar a combinação com o volume, o modelo poligonal deve, então, ser reamostrado para a mesma resolução da imagem intermediária. Vale lembrar que a resolução da imagem intermediária é dada em função das dimensões do volume e dos fatores de *shear* (vide seção 2.2), e não em função das dimensões da imagem final. Se a resolução da imagem intermediária for baixa, comparada à da imagem final, então, dependendo da geometria do modelo poligonal, artefatos e efeitos

de *aliasing* podem ser introduzidos durante o processo de reamostragem do modelo.

Quando o modelo poligonal é rasterizado para uma resolução superior à originalmente calculada para a imagem intermediária visando reduzir os efeitos de *aliasing*, o passo de composição das contribuições das fatias e dos polígonos torna-se mais complexo. É necessário, neste caso, tratar com duas resoluções diferentes: a das fatias e a da imagem do modelo poligonal.

Uma forma simples de evitar que a etapa de composição torne-se complexa consiste em reamostrar totalmente as fatias para a mesma resolução da imagem do modelo, e, conseqüentemente, criar uma imagem intermediária em alta resolução. Contudo, reamostrar completamente as fatias do volume, depois de classificadas, é uma solução ineficiente, dado o número de fatias a serem processadas. Construir uma imagem intermediária cujas dimensões estão relacionadas com as dimensões originais das fatias do volume é uma propriedade fundamental para a implementação de um passo de composição eficiente (vide seção 2.2). Aumentar a resolução das fatias implica em incrementar, significativamente, o tempo gasto para a criação da imagem intermediária.

Uma estratégia alternativa à proposta de reamostragem total das fatias consiste em trabalhar com duas imagens intermediárias, uma em baixa e outra em alta resolução. Nas áreas da imagem sujeitas ao problema de *aliasing*, a imagem intermediária de alta resolução é utilizada durante a etapa de composição; nas demais áreas, utiliza-se a imagem em baixa resolução.

Nos algoritmos propostos, criar uma imagem em baixa resolução implica que as dimensões desta imagem serão calculadas de acordo com a dimensão das fatias do volume e fatores de *shear*, conforme apresentado na equação (2.6). Esta é a resolução originalmente calculada pelo algoritmo de *Shear-Warp* para a imagem intermediária.

Nas imagens em alta resolução, as dimensões são inicialmente calculadas através da equação (2.6). Porém, cada *pixel* desta imagem inicial é subdividido em MxN *subpixels*, gerando uma imagem de alta resolução, conforme será detalhado na seção 3.1 deste capítulo.

Os algoritmos propostos são classificados em três grupos, segundo a etapa de composição da imagem intermediária:

?? **Composição em baixa resolução**: imagem intermediária e modelo poligonal são compostos em baixa resolução.

?? **Composição em alta resolução**: todos os *pixels* da imagem intermediária e o modelo poligonal são compostos em alta resolução (método força bruta); cor e opacidade de todas as fatias do volume são reamostradas para alta resolução durante o processo de composição.

?? **Composição em resolução dual**: cria imagens intermediárias em alta e baixa resolução e o modelo poligonal é renderizado para alta resolução.

A principal diferença na qualidade da imagem final produzida pelos três métodos reside na forma de tratamento do *aliasing* que ocorre quando o modelo poligonal é renderizado. Para ilustrar como os métodos propostos tratam o problema de *aliasing*, escolheu-se como exemplo o modelo sintético *Syn64*, com dimensões de 64^3 *voxels*, combinado com o modelo de cone poligonal, composto de 300 quadriláteros, como mostra a Figura 3.2.

A função de transferência de opacidade criada para este exemplo considera, tanto no volume como no modelo poligonal, que todos os valores diferentes do valor de ruído (no caso zero) são totalmente opacos, ou seja, com opacidade ? =1.0.

Figura 3.2 – Volume sintético e cone poligonal utilizados para detectar *aliasing*

Nos casos onde o modelo poligonal a ser combinado é completamente opaco, emprega-se a implementação tradicional do algoritmo de *Z-Buffer*, que associa somente 1 (um) valor de cor, opacidade e profundidade a cada *pixel* da imagem rasterizada. Para os casos em que o modelo poligonal possui transparência, o *Z-Buffer* tradicional foi modificado para construir um *Zlist-buffer*, onde, para cada *pixel*, serão armazenados *n* valores de cor, opacidade e profundidade, sendo *n* o nível de transparência escolhido pelo usuário.

Nas seções 3.3, 3.4 e 3.5, serão apresentadas as soluções propostas para o tratamento de modelos poligonais opacos, abordando as diferentes estratégias utilizadas

para a redução do efeito de *aliasing* versus tempo de processamento. Na seção 3.6 são colocadas as alterações necessárias para o tratamento de transparência dos polígonos.

A seguir, serão apresentadas considerações importantes a respeito do cálculo das dimensões da imagem intermediária e da criação da matriz de projeção do modelo poligonal, empregada no algoritmo de *Z-Buffer*.

3.1 Resolução da Imagem Intermediária

Para contornar problemas de *aliasing* e *blur* relacionados com a rasterização para uma baixa resolução, os polígonos devem ser renderizados para uma imagem intermediária de resolução mais alta. Esta resolução é determinada a partir dos valores da matriz de *warp* apresentada na equação (2.5). Se a resolução das fatias do volume for inferior à da imagem final, a matriz de *warp* não somente corrige as distorções da imagem intermediária, mas também reamostra esta imagem para a resolução da imagem final.

Para minimizar os problemas de *aliasing*, a resolução da imagem intermediária deve ser redefinida de forma que a matriz de *warp* mapeie vetores da imagem intermediária em vetores de mesmo tamanho na imagem final. Isto é, idealmente, as dimensões de um *pixel* da imagem intermediária não devem mudar significativamente quando este *pixel* for transformado, através do *warp*, para a imagem final. Quando a resolução do volume é menor que a da imagem final, um *pixel* da imagem intermediária é transformado em *MxN pixels* da imagem final.

Os fatores *M* e *N* podem ser calculados a partir da matriz de *warp* $M_{warp} = [w_{ij}]$, como ilustrado na Figura 3.3. Um passo unitário nas direções *u* e *v* no espaço da imagem intermediária implica em incrementos (w_{00}, w_{10}) e (w_{01}, w_{11}), respectivamente, no sistemas de coordenadas da imagem final.

Assim, *M* e *N* podem ser determinados da seguinte forma:

$$\begin{aligned} M &= ceil(\max(|w_{00}|,|w_{10}|,1)) \\ N &= ceil(\max(|w_{01}|,|w_{11}|,1)) \end{aligned} \quad . \tag{3.1}$$

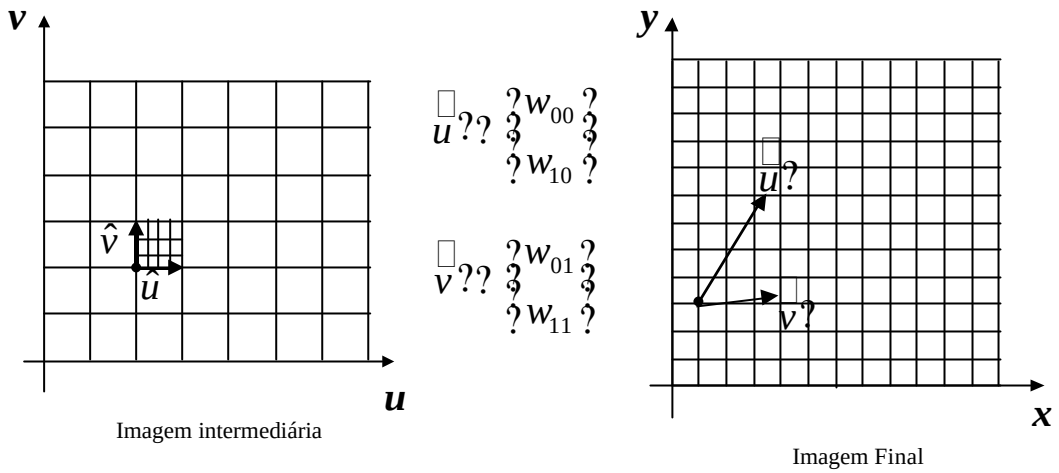
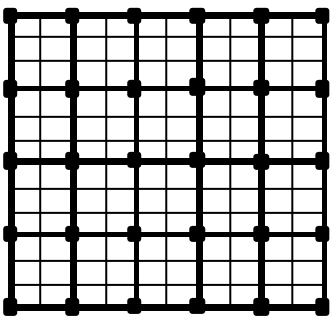


Figura 3.3 – Transformação do espaço da imagem intermediária para o da imagem final

Se as dimensões da imagem intermediária de baixa resolução, calculadas através da equação (2.6), são multiplicadas pelos fatores M e N, então um incremento unitário em coordenadas da imagem final corresponde a um incremento aproximadamente unitário na nova imagem intermediária de alta resolução, cujo tamanho (w_{High}, h_{High}) é dado por:

$$\begin{matrix} w_{High} ? M(w_{Low} ? 1) ? 1 \\ h_{High} ? N(h_{Low} ? 1) ? 1 \end{matrix} . \qquad (3.2)$$

Isto corresponde a dividir cada intervalo de *pixel* da imagem intermediária de baixa resolução em *M*x*N subpixels*, como ilustrado na Figura 3.4. Esta subdivisão aumenta a resolução, preservando as posições dos *pixels* da imagem de baixa resolução. A característica de se preservar as posições dos *pixels* em baixa resolução é importante para as imagens de resolução dual, criadas pelos algoritmos de *footprint* e alta freqüência. Nestes algoritmos, regiões compostas em alta resolução devem ser combinadas com outras compostas em baixa e, para tanto, os *pixels* de ambas as imagens devem estar alinhados.



$w_{Low} = 6,\ M = 2$

$w_{High} = 11$

$h_{Low} = 5,\ \ N = 3$

$h_{High} = 13$

Figura 3.4 – Exemplo de reamostragem de imagem de baixa para alta resolução

3.2 Matriz de Projeção do *Z-Buffer*

Existe um passo que é comum a todos os algoritmos propostos: trata-se da etapa de projeção dos polígonos para o plano da imagem intermediária. A matriz de projeção utilizada pelo algoritmo de *Z-Buffer* mapeia o modelo poligonal do espaço do objeto para o espaço da imagem intermediária. Assim, esta projeção está baseada na resolução da imagem intermediária e nos parâmetros de *shear*.

A matriz que realiza esta transformação é dada por:

$$M_{shearModel} = R\,C\,M_{shear}\,P\,. \tag{3.3}$$

Em (3.3), P é a matriz de permutação escolhida conforme descrito na seção 2.1.2, e M_{shear} é a matriz de *shear*, dada pela equação (2.4). A matriz C, que determina a ordem de composição das fatias, é dada por:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & k_{incr} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.4}$$

O coeficiente k_{incr} pode ser +1 ou –1, conforme a transformação de permutação que mapeia o eixo de visualização do espaço do objeto para o eixo k, positivo ou negativo, do espaço de *standard* (vide seção 2.1.4).

Finalmente, a matriz de resolução R mapeia o modelo poligonal para a resolução da imagem intermediária. Se os polígonos forem mapeados para uma imagem intermediária de alta resolução, R é dada por:

$$R = \begin{bmatrix} \frac{w_{High}}{w_{Low}} & 0 & 0 & \frac{w_{High}}{w_{Low}} - 1 \\ 0 & \frac{h_{High}}{h_{Low}} & 0 & \frac{h_{High}}{h_{Low}} - 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.5}$$

Quando a imagem intermediária tem baixa resolução, R é simplesmente a matriz de identidade.

O resultado da projeção dos polígonos são as matrizes de cor, opacidade e profundidade produzidas pelo *Z-Buffer*. Para acelerar o passo de acumulação da

contribuição do modelo poligonal na imagem intermediária, uma lista auxiliar é construída com os *pixels* ordenados pela sua profundidade. Desta forma, no momento de encontrar a contribuição dos polígonos entre as fatias k e $k+1$, os algoritmos não precisam percorrer toda a matriz de profundidade para encontrar as contribuições para este intervalo.

3.3 Algoritmo de Composição em Baixa Resolução

O algoritmo que efetua a composição da imagem intermediária em baixa resolução é o mais simples dentre os métodos propostos. A resolução da imagem intermediária, neste algoritmo, é definida com os valores calculados pelo algoritmo padrão de *Shear-Warp*, como apresentado pela equação (2.6), e o modelo poligonal é rasterizado, através do *Z-Buffer*, para esta mesma resolução. Os passos básicos deste algoritmo são:

```
Calcular a matriz de projeção para o Z-buffer (equação (3.3));
Renderizar o modelo poligonal, obtendo mapas de cor, opacidade e profundidade;
Para cada k-fatia do volume
        Calcular os deslocamentos ( slice_u, slice_v ) da fatia dentro da imagem intermediária;
        Calcular os pesos ponderadores para cada um dos 4 voxels que contribuem para um
          pixel da imagem intermediária (Figura 3.5);
        Para cada j-linha da fatia
            Calcular a v-linha correspondente dentro da imagem intermediária;
            Enquanto houver voxels não processados nesta j-linha
                Buscar o primeiro pixel não-opaco ( u, v );
                Encontrar os 4 voxels (TL, TR, BL e BR) que contribuem para o pixel (u, v );
                Se um dos 4 voxels correntes for não-transparente, então
                    Interpolar as contribuições dos 4 voxels;
                    Acumular as contribuições ao pixel (u, v );
                senão
                    Saltar os voxels transparentes nas linhas j e j+1, atualizando u-pixel
                      correspondente;
Combinar na imagem intermediária as contribuições dos fragmentos dos polígonos
  que estão entre as fatias k e k+1;
Aplicar a operação de warp na imagem intermediária, transformando-a na imagem
  final.
```

A Figura 3.5 ilustra um aspecto importante do processo de combinação das fatias na imagem intermediária de baixa resolução. Uma vez que seja empregado um filtro de interpolação bilinear para ponderar as contribuições de cada *voxel* (vide seção 2.2), quatro *voxel*s podem contribuir para um dado *pixel* **p**.

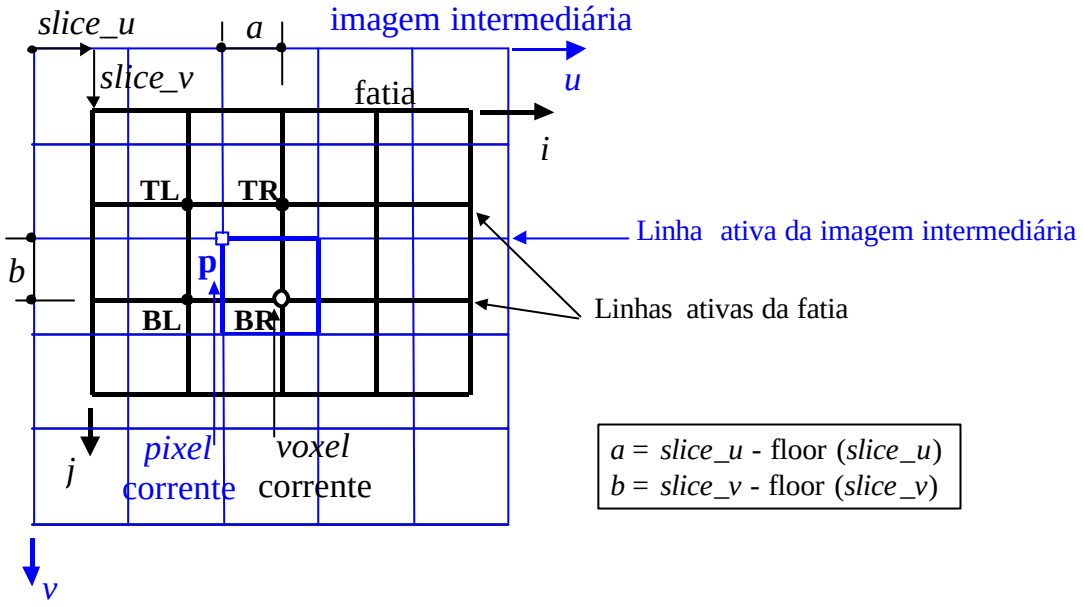


Figura 3.5 – Composição das fatias no *Shear-Warp* padrão

A contribuição dos *voxels* **TL**, **TR**, **BL** e **BR** para o *pixel* **p** é dada pela interpolação bilinear

$$C'_v = (1-a)(1-b)C'_{BR} + a(1-b)C'_{BL} + (1-a)bC'_{TR} + abC'_{TL}$$
$$\alpha_v = (1-a)(1-b)\alpha_{BR} + a(1-b)\alpha_{BL} + (1-a)b\alpha_{TR} + ab\alpha_{TL} \quad , \qquad (3.6)$$

onde α é a opacidade e $C' = \alpha C$ é o canal de cor ponderado pela opacidade (αR, αG, αB) para cada um dos quatro *voxels* [Wittenbrink,1998]. As cores e opacidades dos *voxels* são computadas a partir de funções de transferência fornecidas pelo usuário, sendo empregadas pelo modelo de iluminação para o cálculo da cor final associada a cada *pixel*. O modelo iluminação de Gouraud [Gouraud,1971] é utilizado para a realização da iluminação dos *voxels* em todos os algoritmos implementados nesta tese.

Cabe ressaltar que a expressão de iluminação que está sendo implementada neste trabalho é extremamente simples, não considerando efeitos como o de refração, que poderia ser de especial interesse quando da iluminação de modelos poligonais semi-transparentes. Outro efeito importante que pode ser acrescido ao modelo atual é o tratamento de sombras, quer seja em relação modelo poligonal quer seja em relação ao dado volumétrico.

A cor e a opacidade interpoladas são combinadas com aquelas já presentes no *pixel* **p** através das seguintes expressões :

$$C^{c}_{p,new} \;?\; C^{c}_{p,old} \;?\; C^{c}_{v}(1 ?\; ?\; {}_{p,old}) \qquad (3.7)$$
$$?\; {}_{p,new} \;?\; ?\; {}_{p,old} \;?\; ?\; {}_{v}(1 ?\; ?\; {}_{p,old})$$

Estas expressões podem ser escritas com a utilização do operador *over* de composição digital [Porter,1984], o que resulta em:

$$C^{c}_{p,new} \;?\; C^{c}_{p,old} \; over \; C^{c}_{v} \qquad (3.8)$$
$$?\; {}_{p,new} \;?\; ?\; {}_{p,old} \; over \; ?\; {}_{v}$$

À medida que a opacidade do *pixel* se aproxima de 1.0, a contribuição dos *voxel*s subseqüentes torna-se menos importante para a cor final. Quando a opacidade do *pixel* for igual a 1.0, então cessa-se o acúmulo de contribuições de *voxels* para aquele *pixel*, dizendo-se que o *pixel* está completamente opaco. A equação (3.8) também é empregada para combinar a cor e a opacidade dos fragmentos dos polígonos nos *pixels* correspondentes.

A Figura 3.6 mostra a imagem final do volume sintético combinado com o cone poligonal criada através do algoritmo de composição em baixa resolução.

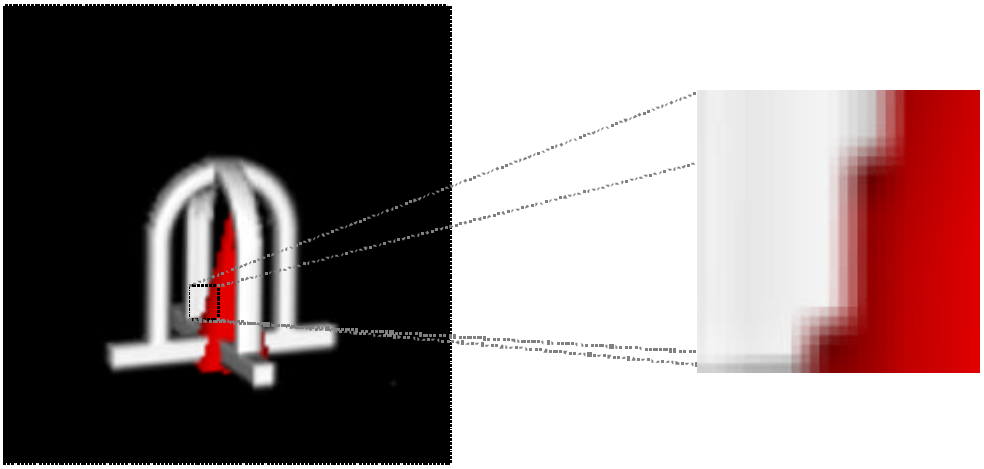


Figura 3.6 – Imagem criada com o algoritmo de baixa resolução, bordas do modelo poligonal em detalhe

O detalhe da Figura 3.6 ilustra o problema de *aliasing* introduzido pela reamostragem do modelo poligonal para a resolução da imagem intermediária. Este problema é mais pronunciado nas bordas do modelo poligonal e pode ser mais ou menos sério, de acordo com o ponto de vista do observador, a geometria do modelo poligonal e, especialmente, a resolução da imagem intermediária.

Visando reduzir o efeito de *aliasing*, investigamos outros algoritmos, apresentados nas próximas seções, para aumentar a resolução da imagem intermediária.

3.4 Algoritmo de Composição em Alta Resolução

Neste algoritmo, todos os *pixels* da imagem intermediária e os mapas de cor, opacidade e profundidade produzidos pelo *Z-Buffer* são calculados em alta resolução (vide equação (3.2)). As fatias do volume são compostas na imagem intermediária de alta resolução com as cores e as opacidades interpoladas, como ilustrado pela Figura 3.7, onde cada *pixel* de alta resolução é considerado como uma subdivisão de um *pixel* de baixa resolução, conforme apresentado na seção 3.1.

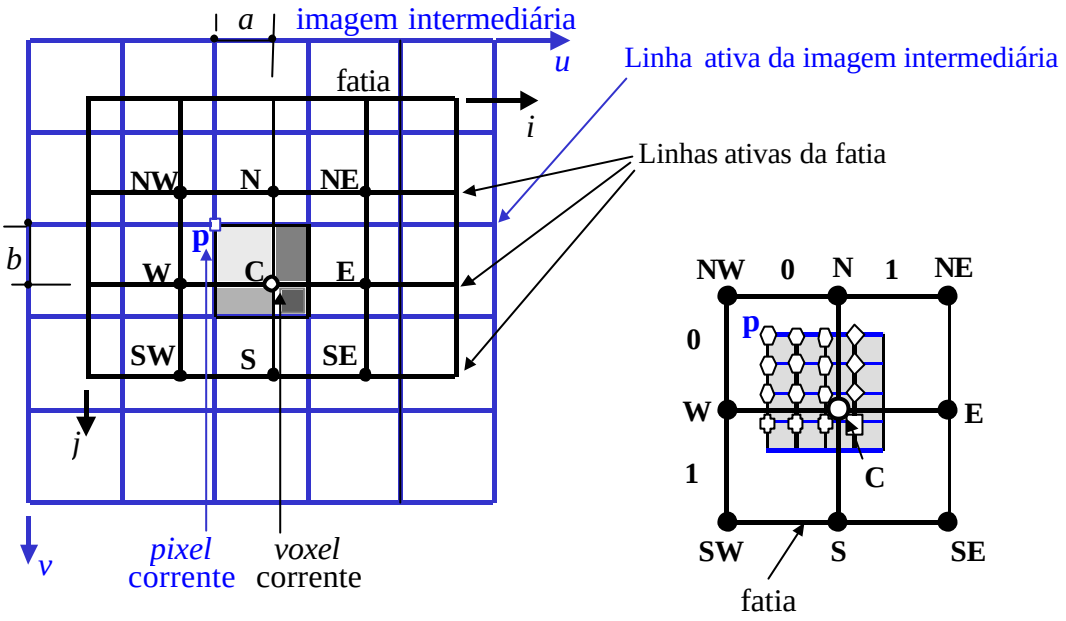


Figura 3.7 – Combinando fatias em baixa resolução em uma imagem intermediária de alta resolução

Como a Figura 3.7 mostra, o *pixel* **p** é subdividido em *MxN subpixels*. Seguindo o mesmo esquema de interpolação bilinear apresentado na seção anterior, para cada *subpixel* de **p,** somente quatro *voxels* contribuem para determinar sua cor e sua opacidade. Entretanto, devido ao desalinhamento, indicado pelas distâncias *a* e *b* na Figura 3.7, entre a fatia e a imagem intermediária, este conjunto de quatro *voxels* pode variar para diferentes *subpixels* de um mesmo *pixel*. Os quatro *voxels* que contribuem para cada *subpixel* são escolhidos dentre uma vizinhança de nove *voxels*, identificados a seguir:

?? C: *voxel* corrente (*i*,*j*)

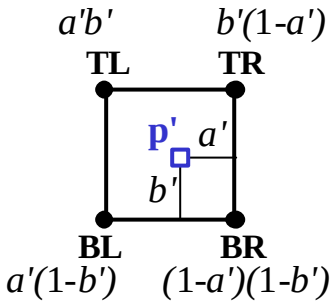
?? W, E: *voxels* *i*-1 e *i*+1 na linha *j*

?? NW, N, NE: *voxels* *i*-1, *i* e *i*+1 na linha *j*-1

?? SW, S, SE: *voxels* *i*-1, *i* e *i*+1 na linha *j*+1

A Figura 3.8 mostra o subconjunto de quatro *voxels* que são escolhidos para cada grupo de *subpixels* de **p,** de acordo com o setor em que os *subpixels* estão posicionados. Note, nesta figura, que os setores marcados pelos símbolos (hexágono, losango, etc.) são aqueles mostrados na Figura 3.7. Os *voxels* **TL**, **TR**, **BL** e **BR** desempenham a mesma função que aquela apresentada na Figura 3.5 e na equação (3.6).

Na Figura 3.8, para cada *subpixel* **p'**, as distâncias *a'* e *b'* têm papel análogo, para a definição dos ponderadores de contribuição de cada *voxel*, ao de *a* e *b* na equação (3.6).



(a)

Setor		TL	TR	BL	BR
00	⬡	NW	N	W	C
01	◇	N	NE	C	E
10	⯃	W	C	SW	S
11	□	C	E	S	SE

(b)

Figura 3.8 – (a) *Voxels* que contribuem para um *subpixel* e seus ponderadores
(b) Tabela de setores e o conjunto de 4 *voxels* associados a cada setor

Os limites de cada um dos quatro setores, que determinam a posição relativa dos *subpixels* dentro do *pixel*, são definidos conforme mostra a Tabela 3.1:

Setor	m_{min}	m_{max}	n_{min}	n_{max}
00	*0*	*floor(a∗M)*	*0*	*floor(b∗N)*
01	*floor(a∗M)*+1	*M*-1	*0*	*floor(b∗N)*
10	*0*	*floor(a∗M)*	*floor(b∗N)*+1	*N*-1
11	*floor(a∗M)*+1	*M*-1	*floor(b∗N)*+1	*N*-1

Tabela 3.1 – Limites dos setores que definem diferentes conjuntos de quatro *voxels*

Considera-se que o índice dos *subpixels* indexados por *m* varia de 0 até *M*-1 na direção *u*, e o dos indexados por *n* varia de 0 até *N*-1 na direção *v*, sendo este índice uma posição relativa dentro de cada *pixel*. Assim, os limites em que ocorre a troca entre os setores são calculados levando-se em conta o deslocamento de cada fatia, indicado através das distâncias *a* e *b*, e os fatores *M* e *N*, que determinam o número de *subpixels*

reamostrados para cada *pixel*.

A tabela de limites dos setores é utilizada pelo algoritmo de alta resolução (e também pelos algoritmos de resolução dual) para determinar quais os *subpixels* que pertencem a cada setor. Por exemplo, se os índices do *subpixel* **p'**, apresentado na Figura 3.8, estiverem dentro dos intervalos ([0, *floor(a,M)*], [0,*floor(b,N)*], nas direções *u* e *v*, respectivamente, então este *subpixel* encontra-se no setor **00**, conforme mostra a tabela acima. Assim, os quatro *voxels* que são utilizados para realizar a interpolação bilinear de cor e opacidade deste *subpixel* são os que estão nas posições NW, N, W, C.

As mesmas informações, presentes em cada fatia do volume e empregadas na composição em baixa resolução (vide Figuras 3.5 e 3.7 e equação (3.6)), são utilizadas para a etapa de composição em alta resolução. No entanto, para este último caso, existem algumas diferenças em relação à manipulação destas informações. O número de linhas ativas da fatia, acessadas durante o processo de acumulação das contribuições dos *voxels*, é maior para a composição em alta resolução. Neste último algoritmo, acessam-se três linhas ativas e consecutivas do volume para obter-se os nove *voxels* vizinhos que podem contribuir para o (*sub*)*pixel* corrente da imagem intermediária, enquanto que no algoritmo de baixa resolução é necessário acessar somente duas linhas ativas da fatia para buscar os quatro *voxels* vizinhos.

Outra diferença diz respeito à determinação de quando um *pixel* torna-se completamente opaco. Na composição em baixa resolução, um *pixel* é considerado opaco quando seu valor de opacidade ? é igual a 1.0 (vide seção 3.3 e equação (3.7)). Já em alta resolução, um *pixel* só é considerado opaco quando todos os seus *subpixels* estiverem completamente opacos.

Finalmente, o passo de *warp* deve ser ajustado para transformar esta imagem intermediária de alta resolução na imagem final. O processo de *warp* é semelhante ao aplicado às imagens intermediárias de baixa resolução. A matriz de *warp* deve ser modificada para que se introduzam os fatores que determinaram o aumento da resolução da imagem intermediária. Esta alteração consiste em multiplicar as duas primeiras linhas da matriz de *warp*, apresentada na equação (2.5), pelos fatores *M* e *N*, definidos na equação (3.1), respectivamente.

A Figura 3.9 mostra uma imagem final do volume sintético combinado com o modelo de cone poligonal criada pelo algoritmo de alta resolução.

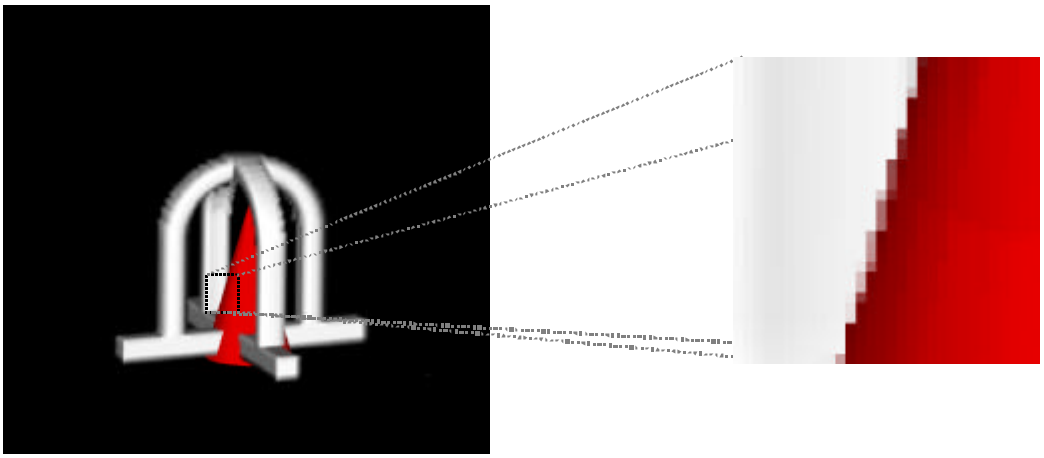


Figura 3.9 – Imagem final do modelo híbrido criada com o algoritmo de alta resolução

Como podemos observar na Figura 3.9, o algoritmo de alta resolução suaviza as bordas do cone, produzindo uma imagem final de boa qualidade. Contudo, este método é muito lento e pode ser otimizado através das propostas apresentadas nos algoritmos de resolução dual, como veremos a seguir. Uma comparação completa entre este e os outros algoritmos é apresentada no Capítulo 4.

3.5 Algoritmos de Composição em Resolução Dual

A idéia básica nesta classe de algoritmos é aumentar a resolução da imagem intermediária somente nas regiões que são influenciadas pela presença do modelo poligonal, conforme mostra a Figura 3.10. Para identificar estas regiões, dois critérios são propostos: (a) *footprint* do modelo poligonal, e (b) regiões de alta freqüência. Destes diferentes critérios resultam dois algoritmos de resolução dual. No primeiro, chamado *footprint*, todos os *pixels* na imagem intermediária de baixa resolução que recebem a projeção do modelo poligonal são marcados para serem processados em alta resolução. No segundo, chamado “alta freqüência”, aplica-se um filtro para identificar as regiões de alta freqüência da imagem resultante da projeção do modelo poligonal, que são aquelas regiões que potencialmente podem apresentar efeitos de *aliasing*. Após a identificação, os *pixels* que correspondem a estas regiões serão marcados e processados em alta resolução. Estes dois métodos serão detalhados a seguir.

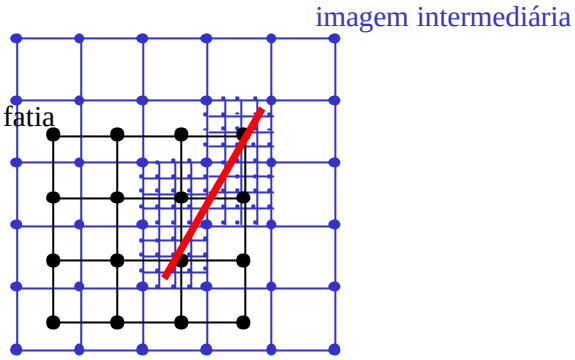


Figura 3.10 – Regiões influenciadas pelo polígono são reamostradas em alta resolução

3.5.1 Algoritmo de *Footprint*

No algoritmo de *footprint* de resolução dual são compostos em alta resolução todos os *pixels* que recebem a contribuição do modelo poligonal. Este algoritmo constrói duas imagens intermediárias: uma de alta resolução e outra de baixa resolução. A imagem intermediária de alta resolução recebe a contribuição somente nas regiões marcadas pelo critério de *footprint*. Esta imagem é utilizada para combinar os *voxels* e polígonos para os *pixels* marcados para serem superamostrados. A imagem de baixa resolução é utilizada para compor somente as contribuições do volume, para todos os *pixels* em baixa resolução. A Figura 3.11 mostra um exemplo das imagens intermediárias de alta e de baixa resolução criadas pelo algoritmo de *footprint*.

(a) (b)

Figura 3.11 – Imagens intermediárias de (a) alta e (b) de baixa resolução criadas com *footprint*

Estas duas imagens, de resoluções diferentes, são construídas em separado para facilitar o processo de acumulação das contribuições dos *voxels* e dos polígonos. Toda contribuição de polígonos é computada somente na imagem intermediária de alta resolução, juntamente com a contribuição dos *voxels* afetados. Devido às propriedades da fatoração *Shear-Warp*, apresentadas na seção 2.1.6, pode-se fazer o mapeamento

entre os *pixels* marcados para serem processados em alta resolução e os *voxels* que irão contribuir para estes *pixels*, devendo ser superamostrados.

Já a imagem de baixa resolução é totalmente preenchida, recebendo as contribuições de todos os *voxels*, inclusive daqueles que serão superamostrados e combinados com os polígonos na imagem de alta resolução. Apesar de serem compostas em ambas as imagens intermediárias, as contribuições dos *voxels* para regiões de alta resolução são calculadas somente uma vez, sendo os valores de cor e opacidade acumulados tanto na imagem em alta resolução como na imagem em baixa resolução (nesta última, os valores acumulados são uma média dos valores calculados para os *subpixels*).

Ao final da etapa de composição, a imagem intermediária de alta resolução é "preenchida" com as contribuições do volume que foram computadas em baixa resolução. Para tanto, as contribuições acumuladas na imagem intermediária de baixa resolução são reamostradas e copiadas para a imagem de alta resolução, criando uma imagem intermediária final, em alta resolução. A Figura 3.12 mostra a imagem intermediária final, em alta resolução, resultante da combinação das imagens intermediárias de baixa e alta resolução, apresentadas na Figura 3.11.

Figura 3.12 – Imagem intermediária final criada pelo algoritmo de *footprint*

Esta imagem intermediária final em alta resolução é transformada pelo passo de *warp* na imagem final. Para tanto, assim como no algoritmo de alta resolução apresentado na seção 3.4, a matriz de *warp* deve ser modificada, multiplicando-se as duas primeiras linhas desta matriz pelos fatores M e N, respectivamente.

A Figura 3.13 mostra a imagem final do volume sintético combinado com o cone poligonal utilizando o algoritmo de *footprint*.

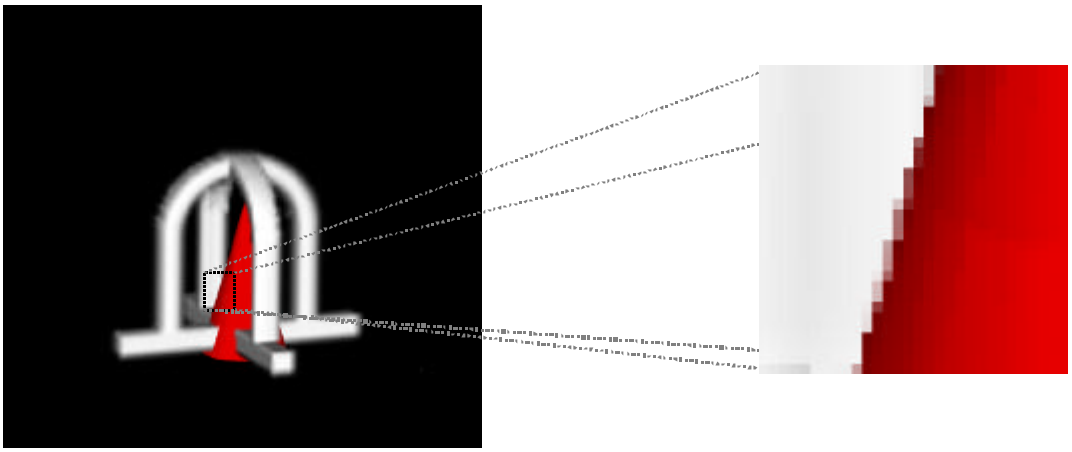


Figura 3.13 – Imagem final criada através do algoritmo de *footprint*, região em detalhe.

3.5.2 Algoritmo de Alta Freqüência

Neste algoritmo, aplica-se um filtro passa-alta à imagem resultante da rasterização do modelo poligonal. O intuito é detectar as regiões de borda ou irregularmente iluminadas do modelo, pois estas são as mais propensas a apresentarem problemas de *aliasing*. Uma vez identificadas essas regiões, os *pixels* correspondentes às mesmas na imagem intermediária de baixa resolução são marcados para serem processados em alta resolução. Desta forma, um número menor de *pixels* será tratado em alta resolução, comparando-se com o algoritmo de *footprint*, visto na seção anterior.

O filtro de altas freqüências utilizado neste algoritmo é o filtro Laplaciano 3x3, dado pela seguinte matriz:

$$L = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}.$$

Esta convolução produz uma nova imagem com valores no intervalo [0-255] para cada um dos canais de cor. Nesta nova imagem convolucionada, as regiões de alta freqüência da imagem rasterizada do modelo poligonal correspondem a valores altos em pelo menos um dos canais RGB. Estes valores de alta freqüência eqüivalem às bordas e às regiões irregularmente iluminadas do modelo poligonal.

A partir da nova imagem criada pela convolução, determinar o que é considerado "valor de alta freqüência" depende de um *limite de freqüência* fornecido pelo usuário. Quanto menor o valor do *limite de freqüência*, que varia no intervalo [0-255], maior é o número de *pixels* que serão computados em alta resolução. A Figura

3.14 mostra o exemplo do modelo do cone reamostrado para a resolução da imagem intermediária de alta resolução e a imagem resultante da convolução com o filtro Laplaciano.

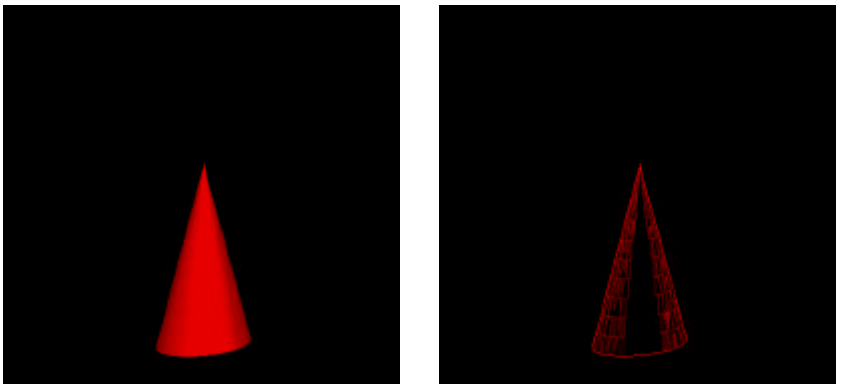


Figura 3.14 – Cone poligonal e sua imagem filtrada em alta freqüência correspondente

O algoritmo de alta freqüência constrói as mesmas duas imagens intermediárias que o de *footprint*, além de uma imagem auxiliar adicional, de baixa resolução, para compor contribuições dos *voxels* e dos polígonos que não pertencem às regiões de borda ou irregulares. A Figura 3.15 mostra as três imagens intermediárias criadas através do algoritmo de alta freqüência, utilizando o valor 120 como *limite de freqüência*.

(a) (b) (c)

Figura 3.15 – Imagens intermediárias em (a) alta resolução;
(b) baixa resolução somente com contribuição do volume;
(c) baixa resolução com contribuições do volume e dos polígonos

Da mesma forma que no *footprint*, ao final do processo de composição a imagem intermediária de alta resolução deve ser "preenchida" com as contribuições dos *voxels* e polígonos que foram compostos nas imagens de baixa resolução (vide Figura 3.15(a)). Visando este "preenchimento", são criadas duas imagens intermediárias de baixa resolução: a primeira que contém somente contribuições do volume, Figura 3.15(b), para preencher as regiões da imagem de alta resolução que não possuem

influência do polígono; e a segunda, Figura 3.15(c), que contém informações do volume mais polígonos, para preencher as regiões de baixa frequência na imagem de alta resolução.

A Figura 3.16 ilustra a necessidade de se construir estas duas imagens em baixa resolução para este método. Nesta figura, somente os *pixels* marcados com um losango são calculados em alta resolução; os demais *pixels* (quadrados e hexágonos) são calculados através da reamostragem das imagens de baixa resolução.

Para compor corretamente os *pixels* marcados com hexágonos na imagem intermediária de alta resolução, é preciso considerar a imagem intermediária de baixa resolução que contenha informações tanto do volume como do modelo poligonal. No entanto, se esta mesma imagem auxiliar (volume+polígono) for considerada para calcular os *pixels* marcados com um quadrado, as bordas do modelo poligonal, que foram compostas em alta resolução, poderiam ficar borradas (*blur*) pela cópia indevida de contribuições desta imagem de baixa resolução (volume+polígono), como ilustrado pela Figura 3.16 e exemplificado com imagens na Figura 3.17.

Para compor corretamente os *pixels* da imagem em alta resolução que não recebem contribuição dos polígonos (quadrados), o algoritmo usa a outra imagem intermediária auxiliar de baixa resolução que contém somente as contribuições dos *voxels*.

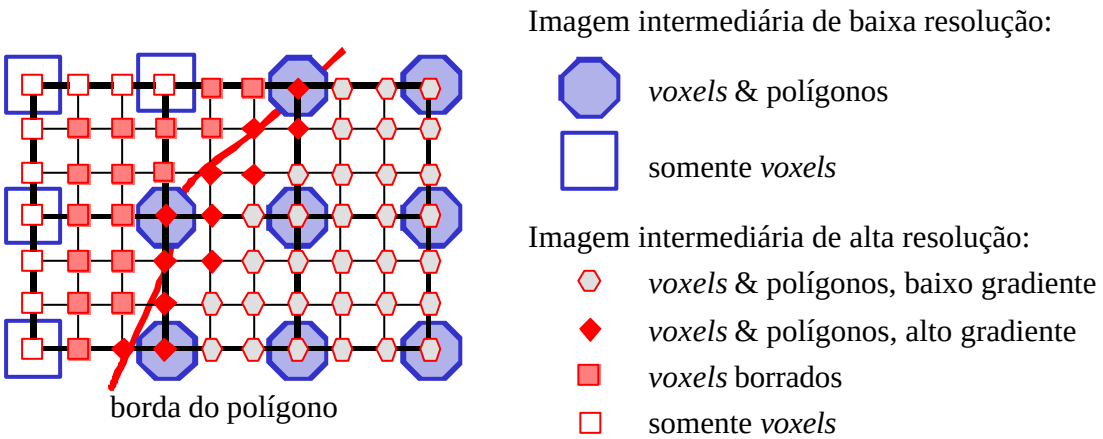


Figura 3.16 – *Blur* produzido nas bordas do polígono usando a imagem em baixa resolução do volume+polígono

A Figura 3.17 mostra através de imagens do dado sintético uma situação onde ocorre o problema de *blur*. Neste exemplo, a imagem intermediária de alta resolução, Figura 3.17(a), é preenchida somente com as contribuições da imagem intermediária de

baixa resolução que contém volume+polígonos (Figura 3.17(b)). Como conseqüência, a imagem intermediária final em alta resolução (Figura 3.17(c)), apresenta erros de *blur*, especialmente nas regiões próximas a borda do modelo poligonal, que deveriam ter sido preenchidas somente com contribuições do volume.

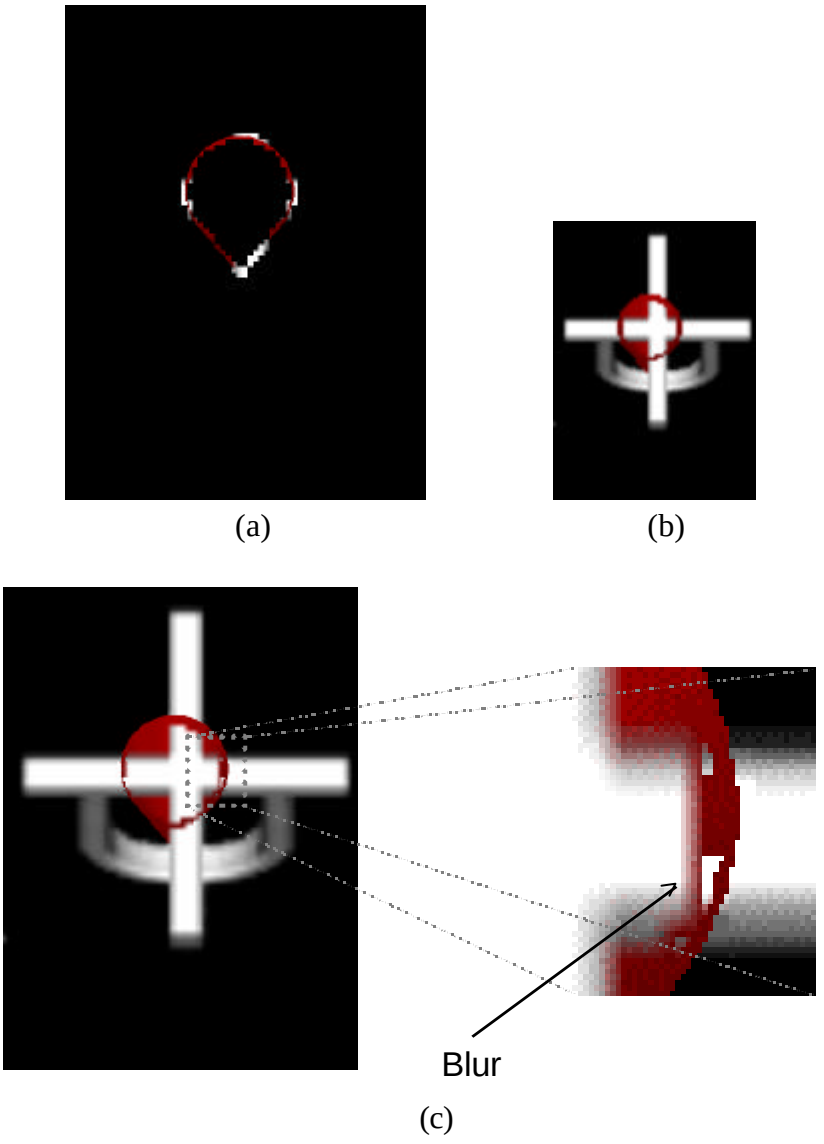


Figura 3.17 – (a) Imagem intermediária composta em alta resolução. (b) Imagem intermediária em baixa resolução do volume+poligono. (c) Imagem intermediária final preenchida apresentando *blur*.

A matriz de opacidade produzida pelo *Z-Buffer* é empregada para identificar quais os *pixels* que receberam contribuição do modelo poligonal (todo *pixel* com ? diferente de zero), distinguindo assim qual das imagens de baixa resolução deve ser utilizada para evitar o efeito de *blur*. A Figura 3.18 mostra a imagem intermediária final, em alta resolução, criada pelo algoritmo de alta freqüência.

Figura 3.18 – Imagem intermediária final em alta resolução criada pelo algoritmo de alta freqüência

Após a construção da imagem intermediária final em alta resolução, o passo de *warp* é o mesmo realizado pelo algoritmo *footprint*, onde a matriz de *warp* deve ser modificada, multiplicando-se pelos fatores *M* e *N* as suas duas primeiras linhas, respectivamente.

A Figura 3.19 mostra a imagem final do volume sintético combinado com o cone poligonal utilizando o algoritmo de alta freqüência. Esta imagem foi criada empregando-se um *limite de freqüência* de valor 120.

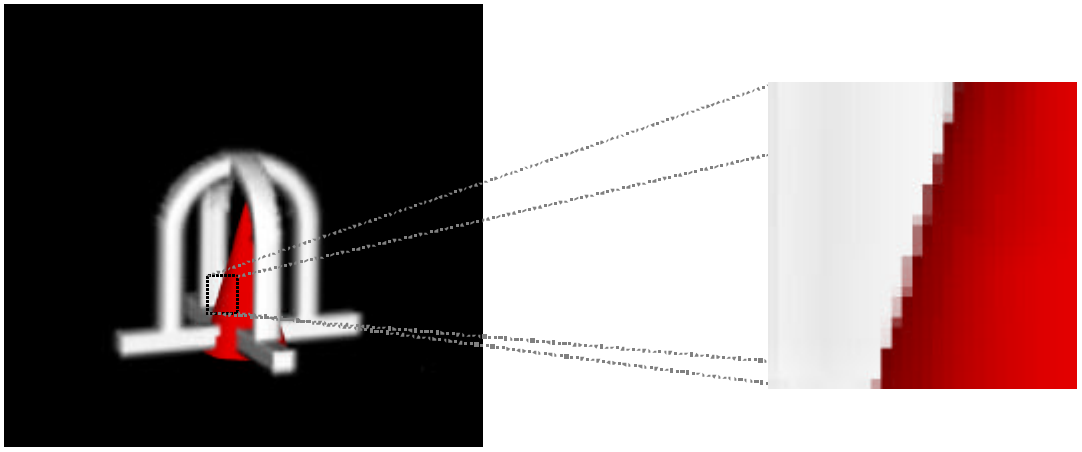


Figura 3.19 – Imagem final do exemplo híbrido criada pelo algoritmo de alta freqüência

3.6 Composição com Modelos Poligonais com Transparência

Os algoritmos de combinação apresentados nas seções 3.3, 3.4 e 3.5 podem ser estendidos para tratar modelos poligonais transparentes. A transparência associada ao dado volumétrico é naturalmente tratada pelo algoritmo de *Shear-Warp*, uma vez que ela é mapeada através da função de transferência de opacidade associada ao dado volumétrico e fornecida pelo usuário. A necessidade de um tratamento especial para a transparência diz respeito somente ao modelo poligonal, uma vez que este é renderizado

através do algoritmo de *Z-buffer*, que não realiza o tratamento de transparência de forma intrínseca.

Basicamente, o algoritmo de *Z-Buffer* deve ser alterado para que mais de um valor de profundidade, cor e opacidade seja armazenado para cada *pixel* da imagem rasterizada dos polígonos.

O algoritmo de *Z-Buffer,* implementado pela biblioteca G3D [Tecgraf,1996], foi alterado para criar um *zlist* para cada *pixel* da imagem. Cada nó desta lista armazena valores de cor, opacidade e profundidade, possibilitando que vários polígonos contribuam para um mesmo *pixel*. O número de níveis acumulado no *zlist* é controlado pelo usuário, através do valor denominado *níveis de transparência*. No G3D, a alocação do *zlist-buffer* é feita logo após se conhecer quantos níveis de transparência o usuário deseja armazenar. Para todos os *pixels* é alocada uma lista de tamanho igual ao número de níveis escolhido pelo usuário.

O processo de composição dos polígonos transparentes continua acontecendo da mesma forma que para os polígonos opacos. Logo após o *Z-Buffer* ter feito a rasterização e criado o *zlist-buffer*, as contribuições dos fragmentos de polígonos que possuem valor de profundidade entre duas fatias do volume são acumuladas nos *pixels* correspondentes da imagem intermediária.

Dentre os algoritmos propostos, o único que necessita de uma etapa a mais, em função da presença de modelos poligonais transparentes, é o algoritmo de alta freqüência. Ele aplica um filtro de passa-alta sobre a imagem resultante do *Z-Buffer* para determinar as regiões propícias à presença de *aliasing* na imagem do modelo (vide seção 3.5.2). Agora, porém, o *Z-Buffer* devolve *n* planos de imagem do modelo, onde *n* é o número de níveis de transparência escolhido pelo usuário. Assim, é necessária a inclusão de uma nova etapa, na qual estas *n* imagens, com valores RGB? *Z*, resultantes da renderização do modelo transparente, serão combinadas em uma única imagem, desprezando o valor da componente de profundidade *Z*. Esta nova imagem será, então, convolucionada com o filtro de passa-alta, identificando-se as regiões que devem ser compostas em alta resolução. Deste ponto em diante, o algoritmo de alta freqüência procede da mesma forma que a descrita para os modelos opacos na seção 3.5.2.

O passo de *warp* não sofre nenhuma alteração em função da presença de modelos transparentes.

A Figura 3.20 mostra o volume sintético opaco combinado com o cone poligonal renderizado com um valor de 0.5 de opacidade, com um nível de transparência, utilizando o algoritmo de alta freqüência com um *limite de freqüência* com valor igual a 120.

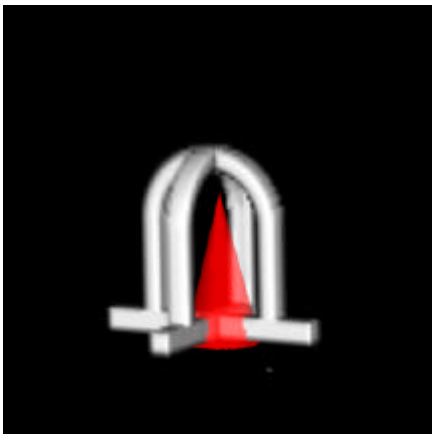


Figura 3.20 – Modelo sintético opaco combinado com o cone poligonal com um nível de transparência

A proposta de criação de um *zlist-buffer* não é nova, e já foi empregada com o algoritmo de *Shear-Warp* no trabalho de Zakaria [Zakaria,1999], além de trabalhos anteriores como o do *A-Buffer* [Carpenter,1984]. A idéia principal no trabalho de Zakaria é renderizar os polígonos criando um *zlist-buffer*, onde cada *pixel* do *buffer* pode armazenar um ou mais valores de profundidade *Z*, juntamente com sua opacidade e cor associada, construindo uma lista de fragmentos translúcidos. Durante o processo de composição com o volume, para cada nova fatia a ser processada, o algoritmo checa a lista e acumula os valores de cor e opacidade dos fragmentos de polígonos que estiverem associados àquela profundidade.

A diferença básica da implementação do *zlist-buffer* desta tese e do trabalho de Zakaria é quanto à forma de alocação do *zlist*. No trabalho de Zakaria, os nós do *zlist* são alocados dinamicamente, até que a opacidade acumulada nos mesmos, para um dado *pixel*, esteja completa (? =1.0). Com isto, diferentes *pixels* podem ter *zlists* de diferentes tamanhos. A implementação realizada nesta tese aloca espaço para *zlists* de mesmo tamanho, para todos os *pixels*, de acordo com o nível de transparência escolhido pelo usuário.

Uma vantagem da implementação de Zakaria é garantir a contribuição de todos os fragmentos de polígono para cada *pixel*, independente da sua profundidade. Entretanto, tempo de processamento pode ser desperdiçados criando-se um *zlist* com um

número de nós maior do que aquele que será realmente aproveitado durante o processo de composição com as fatias. Por exemplo, se o volume a ser composto for quase opaco, poucos níveis de contribuição dos polígonos serão utilizados, pois a opacidade de cada *pixel* será rapidamente completada com as informações do volume somada à de mais alguns poucos fragmentos de polígono.

Já na implementação desta tese, o usuário tem controle sobre o número de níveis de transparência, podendo evitar desperdício de tempo de processamento com a montagem de um *zlist* maior que o necessário. Entretanto, como será discutido no Capítulo 4, podem ser criadas imagens incorretas nos casos onde o grau de transparência do volume é alto e o nível de transparência escolhido para os polígonos não é suficiente para representar todas as contribuições que devem ser combinadas com as do volume.

Cabe ressaltar que o modelo de iluminação que está sendo implementado neste trabalho não está considerando efeitos de refração, especialmente presentes quando da iluminação de objetos semitransparentes. Para que esta característica seja incorporada, o algoritmo de iluminação deve considerar os raios refratados, combinando o algoritmo atual a um método baseado no *Ray-tracing* [Foley,1990], que calcula corretamente os efeitos da refração a cada ponto dos objetos semitransparentes.

4 RESULTADOS EXPERIMENTAIS

Neste capítulo, são apresentados e discutidos resultados obtidos a partir das implementações dos algoritmos propostos. Os testes buscam evidenciar diferenças em termos de tempo de processamento e qualidade das imagens. Diferentes plataformas são testadas, variando-se as bibliotecas gráficas que implementam o algoritmo de *Z-Buffer* e o *hardware* empregado. Os dados volumétricos e modelos poligonais também variam, de acordo com as características que se desejam evidenciar.

Primeiramente, serão apresentados os testes realizados para comparar a eficiência de cada um dos algoritmos, combinando volumes e modelos poligonais opacos, empregando a biblioteca G3D [Tecgraf,1996] para a renderização do modelo poligonal. Estes resultados são discutidos na seção 4.3.1

Ainda empregando volumes e modelos poligonais opacos, novos testes foram realizados para comparar a implementação que só utiliza recursos de *software versus* aquela que faz uso de recursos de aceleração da placa gráfica para a etapa de renderização dos modelos. Na implementação por *software* foi empregado o algoritmo de Z-*buffer* da biblioteca gráfica G3D [Tecgraf,1996]. Na implementação por *hardware*, foi utilizado o algoritmo de *Z-Buffer* da biblioteca OpenGL [Woo,1999] acelerado pelo *hardware* da placa gráfica. Estes resultados são apresentados na seção 4.3.2.

Em uma outra bateria de testes, buscou-se avaliar a eficiência dos algoritmos quando da presença de modelos poligonais com transparência. Os algoritmos foram implementados empregando método de *Zlist-Buffer*, a partir do Z-*Buffer* modificado da biblioteca G3D (vide seção 3.6). Os resultados são apresentados na seção 4.3.3.

A seção 4.4 apresenta os tempos gastos com os procedimentos realizados em fase de pré-processamento, como a quantização dos gradientes e a construção das

estruturas de RLE do volume. Ainda dentro dessa seção, são traçadas considerações sobre a qualidade da imagem gerada com e sem a quantização do gradiente.

Além dos resultados que avaliam a eficiência dos algoritmos, compararam-se os mesmos em termos de qualidade da imagem final gerada. Para tanto, várias imagens de um mesmo volume e modelo poligonal e com os mesmos parâmetros de visualização foram criadas a partir dos diferentes algoritmos. Foram avaliadas e discutidas características como tratamento de *aliasing*, presença de *blur* e precisão quanto ao posicionamento do modelo poligonal, que são apresentadas na seção 4.5.

4.1 Dados Volumétricos e Modelos Poligonais

Para comparar os algoritmos propostos em termos de eficiência e qualidade de imagem, foram empregados três dados volumétricos; dois deles foram obtidos do conjunto de dados de Tomografia Computadorizada do *Visible Human Project* [Visible] e um terceiro dado geométrico foi construído sinteticamente.

Dado de tomografia da cabeça da *Visible Woman*

O dado volumétrico utilizado para a avaliação do tempo de processamento das diferentes versões do S*hear-Warp* híbrido é um exame de Tomografia Computadorizada de cabeça, extraída do conjunto de dados *Visible Woman* [Visible], ilustrado na Figura 4.1(a). O dado volumétrico da cabeça foi montado a partir das fatias 1001 a 1209 extraídas de todo o conjunto da *Visible Woman*. Cada fatia possui 512x512 valores de densidade e a dimensão do volume é, portanto, de 512x512x209 *voxels*. Para estudar a influência do tamanho do volume sobre os resultados obtidos com os algoritmos propostos, filtrou-se o dado original, produzindo um volume com dimensões menores, com 127x127x51 *voxels*. Os valores originais de densidade, que são representados em 12 *bits*, foram quantizados para uma representação em um *byte*, mapeando os valores de densidade para o intervalo entre 0 e 255.

O tempo de processamento gasto pelo algoritmo de visualização volumétrica é especialmente dependente das funções de transferência associadas ao volume. Para visualizar o crânio do dado volumétrico da cabeça, mostrado na Figura 4.1(b), a função de transferência ? fornece os valores de opacidade e a função *C* fornece os valores da tripla RGB de cor. Estas duas funções são definidas da seguinte forma:

$$\lambda(v) = \begin{cases} 0.0, & \text{se } v \in [0,100]; \\ 1.0, & \text{se } v \in [101,255]. \end{cases}$$

$$C(v) = \begin{cases} (0\ 0\ 0), & \text{se } v \in [0,49]; \\ (215\ 150\ 115), & \text{se } v \in [50,66]; \\ (245\ 100\ 100), & \text{se } v \in [67,99]; \\ (255\ 255\ 255), & \text{se } v \in [100,255]. \end{cases},$$

onde v é o valor de densidade do *voxel*.

A escolha da função de transferência de opacidade λ acima implica que ambos, volume e modelo poligonal, sejam mapeados como opacos. Escolheu-se trabalhar com um exemplo sem transparência, neste caso, para avaliar melhor o tempo gasto com o cálculo e a renderização em nível de *subpixel* pelos algoritmos que produzem imagens em alta resolução.

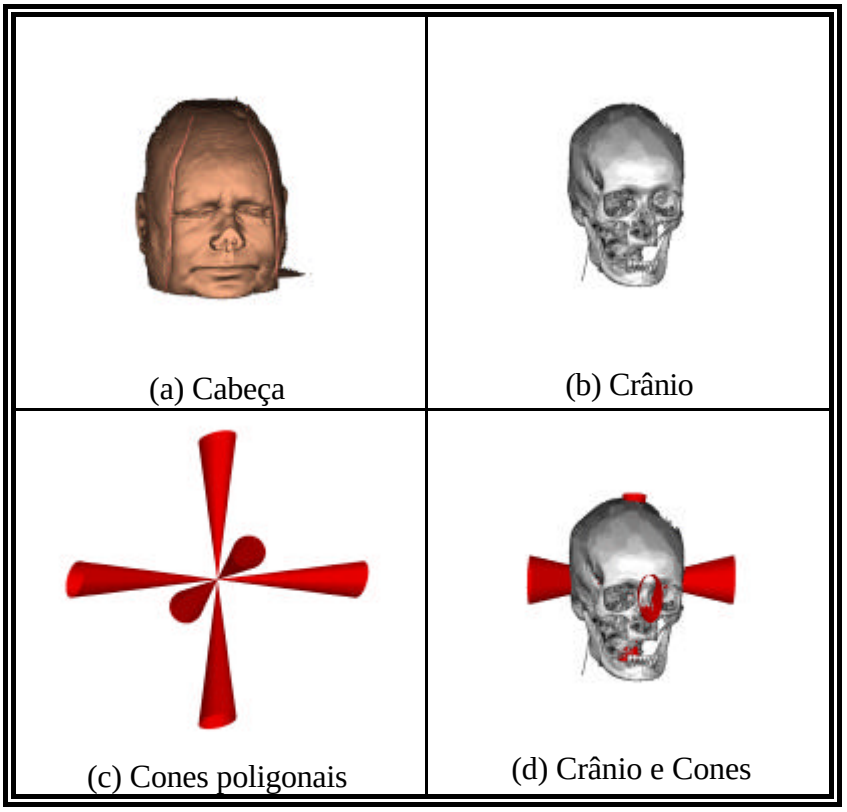


Figura 4.1 – Cabeça da *Visible Woman* e modelo poligonal de cones

O modelo poligonal é composto de seis cones paralelos a cada um dos eixos (*X*, *Y*, *Z*) do espaço do dado volumétrico, como mostra a Figura 4.1(c). Cerca de 300 quadriláteros são usados para modelar cada um dos cones. Todos os cones são mapeados como completamente opacos (λ = 1.0). A Figura 4.1(d) mostra a imagem final, usando o RLE do volume, onde o dado volumétrico e o modelo poligonal

aparecem combinados.

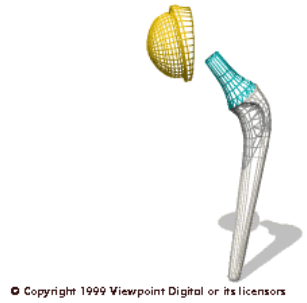
Outros parâmetros importantes empregados nos testes são apresentados na Tabela 4.1. A *opacidade mínima* indica o menor valor considerado como não transparente. Se o valor de opacidade associado a um *voxel* é menor que o valor mínimo, então o *voxel* é considerado transparente. A *opacidade máxima* indica o maior valor de opacidade considerado como não opaco. Durante o processo de composição, os *pixels* podem receber a contribuição dos *voxels* até que eles se tornem opacos, isto é, até que o valor de opacidade de cada pixel seja maior que o valor máximo de opacidade. A *luz ambiente* indica a porcentagem de luz que vem do ambiente. O *limite de ruído* indica o menor valor considerado como um material válido no dado volumétrico. O *limite de freqüência* é um valor definido pelo usuário que indica o limite inferior do valor considerado como alta freqüência para a imagem filtrada do modelo poligonal (válido somente para o método de composição híbrida em alta freqüência).

Opacidade mínima	0.05	Limite de ruído	10
Opacidade máxima	0.95	Limite de freqüência	120
Luz ambiente	10%	Resolução imagem final	500^2

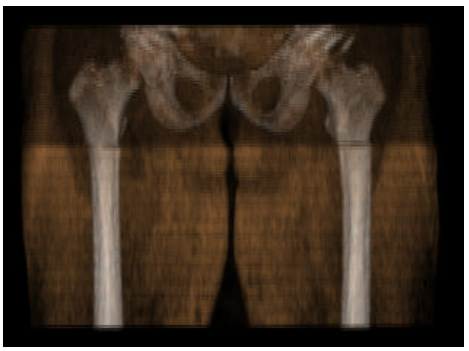
Tabela 4.1 – Parâmetros utilizados nos testes com o volume da cabeça

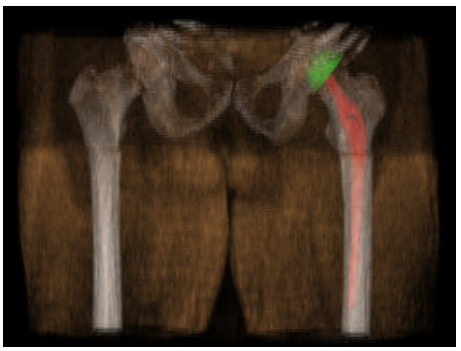
Dado de tomografia da Pélvis do *Visible Man*

O dado volumétrico da Pélvis, extraída do conjunto de exames de Tomografia Computadorizada do corpo do *Visible Man* [Visible], é utilizado para a realização de testes com o modelo poligonal com transparência. O conjunto de dados da pélvis foi montado com as fatias do intervalo de 1843 a 2258 do conjunto do corpo completo. Cada fatia tem a resolução 512x512, e o volume possui 512x512x100 *voxels*. Ele é combinado com um modelo poligonal de prótese de fêmur, composto por 3758 triângulos, que foi manualmente instanciado dentro do osso do fêmur. As imagens apresentadas nas Figuras 4.2(a), 4.2(b) e 4.2(c) mostram o modelo poligonal, o dado da Pélvis e a imagem dos dados combinados, respectivamente.



(a)





(b) (c)

Figura 4.2 – (a) Modelo poligonal da prótese; (b) dado volumétrico da Pélvis e (c) dados combinados

Da mesma forma que no caso do dado do crânio da *Visible Woman*, os valores originais de densidade, que são representados em 12 bits, foram quantizados para uma representação em um *byte*, sendo mapeados para o intervalo entre 0 e 255.

A função de transferência ? mapeia os valores de opacidade e a função *C* mapeia os valores da tripla RGB de cor, sendo aplicadas ao volume da Pélvis. Estas funções fornecem valores para pontos chave de densidade. Os valores das densidades intermediárias, entre dois pontos chave, são interpolados linearmente. As funções são definidas da seguinte forma:

$$? ? \begin{cases} ?0.0, & se\ v\ ?\ 0 \\ ?0.0, & se\ v\ ?\ 41 \\ ?0.03, & se\ v\ ?\ 42 \\ ?0.03, & se\ v\ ?\ 57 \\ ?0.0, & se\ v\ ?\ 58 \\ ?0.0, & se\ v\ ?\ 80 \\ ?0.15 & se\ v\ ?\ 100 \\ ?0.2, & se\ v\ ?\ 101 \\ ?0.2, & se\ v\ ?\ 255 \end{cases} \qquad C ? \begin{cases} ?(255\ 173\ 94), & se\ v\ ?\ 0 \\ ?(255\ 172\ 95), & se\ v\ ?\ 80 \\ ?(215\ 150\ 115), & se\ v\ ?\ 81 \\ ?(255\ 171\ 100), & se\ v\ ?\ 90 \\ ?(255\ 195\ 150), & se\ v\ ?\ 91 \\ ?(255\ 223\ 201), & se\ v\ ?\ 110 \\ ?(255\ 255\ 255), & se\ v\ ?\ 111 \\ ?(255\ 255\ 255), & se\ v\ ?\ 255 \end{cases} ,$$

onde *v* é o valor de densidade do *voxel*.

Os parâmetros de visualização utilizados com o dado da Pélvis são os

apresentados pela Tabela 4.2:

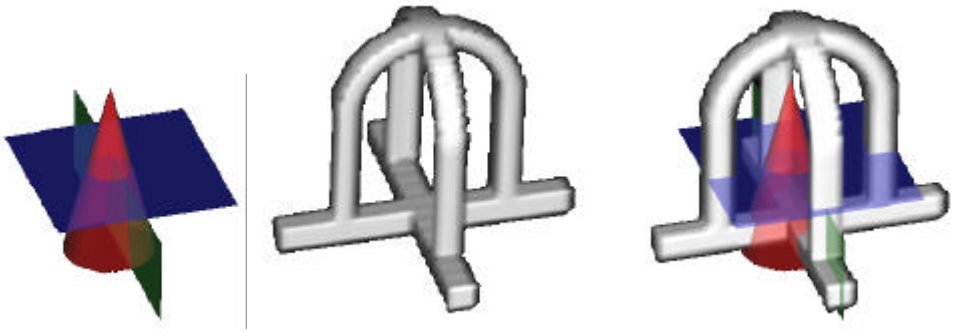
Opacidade mínima	0.01	Limite de ruído	10
Opacidade máxima	0.95	Limite de freqüência	120
Luz ambiente	20%	Resolução imagem final	500^2

Tabela 4.2 – Parâmetros utilizados nos testes com o volume da Pélvis

Dado sintético *Syn64*

O dado sintético *Syn64* também foi empregado nos testes onde o modelo poligonal é renderizado com transparência. As dimensões deste dado são 64^3 *voxels*, com valores de densidades no intervalo 0 a 255.

O modelo poligonal empregado nos testes com transparência também foi montado sinteticamente, sendo composto por dois planos ortogonais, um de cor verde e outro azul, e um cone, de cor vermelha, descrito por 300 quadriláteros. A Figura 4.3 mostra o modelo poligonal dos planos com o cone centrado, o volume *Syn64* e os dois dados combinados.



(a) (b) (c)

Figura 4.3 – (a) Modelo poligonal, (b) volume *Syn64* e (c) combinação

A função de transferência ? fornece os valores de opacidade e a função *C* fornece os valores da tripla RGB de cor, sendo aplicadas ao volume *Syn64*. Elas são definidas da seguinte forma:

$$? \; ? \; ? \begin{cases} ?0.0, & \text{se } v \; ? \; ?0,29?; \\ ?1.0, & \text{se } v \; ? \; ?30,255?. \end{cases}$$

$$C \; ? \; ? \begin{cases} ?(0 \;\; 0 \;\; 0), & \text{se } v \; ? \; ?0,29?; \\ ?(255 \;\; 255 \;\; 255), & \text{se } v \; ? \; ?30,255?. \end{cases} ,$$

onde *v* é o valor de densidade do *voxel*.

Os parâmetros de visualização foram os mesmos utilizados nos testes com

volumes e modelos opacos da cabeça da *Visible Woman*, conforme apresenta Tabela 4.3.

Opacidade mínima	0.05	Limite de ruído	10
Opacidade máxima	0.95	Limite de freqüência	120
Luz ambiente	10%	Resolução imagem final	500^2

Tabela 4.3 – Parâmetros utilizados nos testes com *Syn64*.

4.2 Contexto das Implementações

Os algoritmos propostos nesta tese foram implementados em um programa próprio de visualização volumétrica, usando como ferramenta de programação a linguagem C. O programa de visualização volumétrica baseia-se na implementação do algoritmo de *Shear-Warp* feita por Lacroute em sua biblioteca Volpack [Lacroute,1995].

Os algoritmos desenvolvidos empregam somente projeção paralela e, para efeito dos testes realizados, utilizam as estruturas de dados RLE do dado volumétrico e da imagem intermediária (vide seção 2.3), visando acelerar o processo de composição dos *voxels*.

Para minimizar a quantidade de memória alocada pelas estruturas do RLE do dado volumétrico, o programa quantiza o vetor gradiente de cada *voxel* através de uma tabela pré-calculada com um número fixo de vetores normais uniformemente distribuídos sobre a superfície de uma esfera (vide seção 2.4). Nos testes descritos neste capítulo, foi realizada uma quantização para uma tabela de vetores normais com 256 entradas, reduzindo a representação do gradiente para um *byte* por *voxel*.

Os algoritmos foram testados na seguinte plataforma de *hardware*:

- Intergraph RealiZm II [Graphics Workstations], dual Pentium 333MHz, 512Mb RAM, placa gráfica RealiZm II L3 com 32Mb, rodando sistema Windows NT 4.0.

Além deste ambiente, alguns testes foram realizados em uma segunda plataforma, descrita a seguir:

- Pentium III 500 MHz, com 1Gb RAM, placa gráfica Viper 770 [Diamond], rodando sistema Windows NT 4.0.

Para identificar melhor o contexto de cada bateria de testes, elaborou-se uma

ficha técnica, como a apresentada na Tabela 4.4, contendo dados a respeito do volume, do modelo poligonal e dos demais parâmetros de visualização.

4.3 Eficiência

Foram consideradas quatro versões do algoritmo de *Shear-Warp*, propostas no Capítulo 3, mais a versão padrão, para gerar as medidas de tempo apresentadas nesta seção: (1) *Shear-Warp* padrão (sem combinação com polígonos); (2) composição em baixa resolução; (3) composição em alta resolução; (4) *footprint* em alta resolução; e (5) alta freqüência em alta resolução.

Diferentes tabelas e gráficos apresentam os tempos obtidos para cada um dos algoritmos propostos, variando-se o dado volumétrico processado, o modelo poligonal combinado, a biblioteca gráfica utilizada para a renderização do modelo poligonal e a presença ou não de modelos transparentes.

4.3.1 Tempos de processamento de volume e modelo opacos

Contexto do Teste	
Dado Volumétrico	cabeça *Woman*
Dimensão do Volume	512x512x209 (original) 127x127x51 (reduzido)
Opacidade do Volume	Opaco
Imagem Final	500x500
Modelo Poligonal	6 cones
Número de Polígonos	300
Opacidade do Modelo Poligonal	Opaco
Níveis de Transparência do Modelo	0
Biblioteca Gráfica	G3D (*software*)
Plataforma de *Hardware*	Intergraph

Tabela 4.4 – Teste com a TC de cabeça e o modelo de cones

Esta bateria de testes foi realizada com os dados e parâmetros mostrados na Tabela 4.4.

Os resultados, apresentados na Tabela 4.5, são os tempos de CPU, em segundos, gastos nos principais passos dos algoritmos implementados. Uma vez que o tempo de processamento dos algoritmos também é dependente da direção de projeção dos dados, os tempos apresentados nesta tabela foram computados como uma média de 50 direções de projeção aleatoriamente distribuídas.

Crânio da *Visible Woman*	Tamanho do Volume	Sem Polígonos	Composição em Baixa resolução	Composição em Alta resolução	Composição Dual	
					Footprint	Alta Freqüência
Z-Buffer do Modelo Poligonal	reduzido	0.000	0.111	0.221	0.222	0.220
	original	0.000	0.291	0.353	0.357	0.357
Marcação das Regiões de Alta resolução	reduzido	0.000	0.000	0.029	0.029	0.060
	original	0.000	0.000	0.066	0.041	0.128
Criação da Imagem Intermediária	reduzido	0.068	0.065	0.958	0.524	0.511
	original	1.687	1.711	3.940	3.297	2.459
Composição da Contribuição dos Polígonos entre as Fatias	reduzido	0.000	0.000	0.058	0.053	0.055
	original	0.000	0.024	0.125	0.135	0.131
Composição da Contribuição das Fatias	reduzido	0.068	0.065	0.900	0.356	0.275
	original	1.687	1.687	3.815	2.908	1.922
Combinação das Imagens Intermediárias	reduzido	0.000	0.000	0.000	0.115	0.182
	original	0.000	0.000	0.000	0.254	0.405
Warp	reduzido	0.167	0.175	0.190	0.195	0.199
	original	0.195	0.208	0.209	0.224	0.238
Tempo Total	reduzido	**0.235**	**0.351**	**1.399**	**0.970**	**0.990**
	original	**1.882**	**2.210**	**4.569**	**3.919**	**3.182**

Tabela 4.5 – Tempos de processamento do Crânio da *Visible Woman*, usando G3D

Várias observações podem ser feitas a partir da Tabela 4.5. O tempo gasto pelo algoritmo de *Z-Buffer* para renderizar o dado poligonal é significativo, se comparado com o tempo total, conforme evidencia a Tabela 4.6, especialmente para o volume reduzido, onde o tempo de criação da imagem final é menor, dado o menor número de *voxels* a serem compostos na imagem intermediária. Mesmo para o volume original, este percentual está, em média, acima dos 10% do tempo total.

É natural, então, buscar alternativas mais eficientes para esta etapa, como por exemplo o uso de bibliotecas gráficas que tenham suas funções aceleradas por *hardware*. Esta solução foi implementada e é avaliada na seção 4.3.2.

Retomando a análise sobre a Tabela 4.5, o tempo gasto pelo algoritmo de baixa resolução para combinar as fatias do volume e do modelo poligonal é praticamente igual ao tempo gasto para combinar somente as fatias do volume na versão do *Shear-Warp* sem polígonos. Isto acontece porque o modelo poligonal faz com que *pixels* da imagem intermediária tornem-se opacos mais rapidamente, em função do acúmulo da contribuição dos polígonos. Assim, com o uso das otimizações de RLE do volume e da imagem intermediária, a etapa de composição do volume com polígonos realiza uma quantidade menor de processamento que a composição somente das fatias do volume.

Crânio da *Visible Woman*	Tamanho	Baixa resolução	Alta resolução	Dual	
				Footprint	Alta Freqüência
Z-Buffer do Modelo Poligonal	reduzido	0.111	0.221	0.222	0.220
Tempo Total		0.351	1.399	0.970	0.990
Porcentagem (%)		**32**	**16**	**23**	**22**
Z-Buffer do Modelo Poligonal	original	0.291	0.353	0.357	0.357
Tempo Total		2.210	4.569	3.919	3.182
Porcentagem (%)		**13**	**8**	**9**	**11**

Tabela 4.6 – Percentual de tempo gasto com o passo de *Z-Buffer* no volume reduzido e original

Em todos os algoritmos propostos e na maioria dos casos testados, o tempo gasto para a criação da imagem intermediária é maior que o tempo gasto em qualquer dos outros passos dos algoritmos. Esta relação só não se observa no algoritmo de baixa resolução, com o volume reduzido. À medida que o número de *pixels* da imagem intermediária aumenta, o tempo gasto no passo de composição também aumenta. Entretanto, cabe lembrar que o número de *pixels* a serem processados em alta resolução depende da versão do algoritmo escolhida, da geometria do modelo poligonal e da direção de projeção.

Considerando a etapa de criação da imagem intermediária, o subpasso que mais consome tempo de processamento é a composição das fatias. Este passo inclui o *shading* do *voxel*, a interpolação de cor e opacidade para cada *pixel*, e a acumulação na imagem intermediária. Os resultados mostrados na Tabela 4.5 indicam que os algoritmos híbridos podem ser ordenados da seguinte forma, do mais para o menos eficiente nesta etapa de processamento: baixa resolução, alta freqüência, *footprint* e alta resolução.

O passo de combinação das imagens intermediárias aplica-se somente aos algoritmos de resolução dual, onde parte da imagem intermediária está representada em baixa resolução e parte em alta resolução. O algoritmo de *footprint* obteve melhores tempos para este passo do que o algoritmo de alta freqüência. O primeiro cria somente uma imagem intermediária de baixa resolução que acumula apenas contribuições do volume, enquanto o último cria duas imagens intermediárias em baixa resolução, sendo que uma acumula somente volume e a outra acumula volume mais polígonos. Assim, o método de *footprint* precisa tratar somente uma imagem em baixa resolução, enquanto que o de alta freqüência trabalha sobre duas destas imagens (vide seção 3.5). Além disto, o *footprint* precisa "copiar" um número menor de *pixels* da imagem intermediária

de baixa resolução para a versão em alta resolução. Os resultados apresentados na Tabela 4.5 confirmam esta análise.

A respeito do tempo gasto no passo de *warp*, pode-se notar que este depende da resolução da imagem intermediária e da final. No algoritmo de baixa resolução, o tempo gasto pelo *warp* é próximo ao do algoritmo padrão, sem composição de polígonos. Nos algoritmos que criam uma imagem intermediária em alta resolução, o tempo gasto com o *warp* aumenta com o aumento da resolução da imagem intermediária. No entanto, para volumes com um maior número de *voxels*, o tempo gasto com a operação de *warp* torna-se mínimo se comparado com o tempo de criação da imagem intermediária.

Quando a resolução das fatias do volume que estão paralelas ao plano de projeção é semelhante à da imagem final, os fatores M e N são iguais a 1 (um). Neste caso, o algoritmo de baixa resolução cria uma imagem intermediária com a mesma resolução da criada pelos algoritmos de alta resolução e resolução dual; portanto, a imagem final criada por todos estes algoritmos possui a mesma qualidade. Neste caso, a melhor escolha é o algoritmo de baixa resolução, por ser o mais rápido.

4.3.2 Comparação entre as bibliotecas G3D e OpenGL

Para comparar o desempenho dos algoritmos implementados com a biblioteca G3D [Tecgraf,1996], implementada totalmente por *software*, e a OpenGL [Woo,1999], com aceleração de *hardware*, o contexto da Tabela 4.7 foi utilizado.

Contexto do Teste	
Dado Volumétrico	cabeça *Woman*
Dimensão do Volume	127x127x51 (reduzido)
Opacidade do Volume	opaco
Imagem Final	500x500
Modelo Poligonal	6 cones
Número de polígonos	300
Opacidade do Modelo Poligonal	opaco
Níveis de Transparência do Modelo	0
Biblioteca Gráfica	G3D e OpenGL
Plataforma de *Hardware*	Intergraph

Tabela 4.7 – Contexto empregado na comparação G3D com OpenGL

Na implementação em OpenGL dos algoritmos propostos, somente a etapa de renderização do modelo poligonal faz uso das funções aceleradas por *hardware.* As demais etapas não são relevantes para a comparação de desempenho entre as bibliotecas, uma vez que são todas implementadas pelo mesmo código, em *software*.

Na implementação em OpenGL, a etapa de extração dos valores do *Z-Buffer* e de mapas de cor é a mais custosa, dentro do passo de renderização de polígonos. Esta afirmação só não é válida para o algoritmo de baixa resolução aplicado ao volume reduzido, em que o tamanho da imagem intermediária é pequeno, devido às dimensões do volume, e o tempo gasto com a extração é menor que o do passo de rasterização. Na etapa de extração, a versão OpenGL é inclusive mais lenta que a versão G3D, para todos os algoritmos, sendo que a diferença se dá na ordem de centésimos de segundo. A etapa de extração no OpenGL é mais lenta porque os valores de profundidade são extraídos do *z-buffer* um a um. Esta operação não é uma operação realizada normalmente durante o *pipeline* de renderização de polígonos e, por este motivo, o OpenGL não possui uma função otimizada para a recuperação dos valores de profundidade.

No entanto, a etapa de rasterização da versão OpenGL é sempre mais eficiente que a implementação na versão G3D. Esta diferença se reflete no tempo total gasto com o passo de *Z-Buffer* do modelo poligonal, sendo que o ganho é da ordem de décimos de segundo. O gráfico da Figura 4.4 ilustra as diferenças de tempo de processamento entre as versões OpenGL e G3D para os passos de rasterização e extração do *Z-Buffer*, para o algoritmo de *footprint* empregando o volume reduzido.

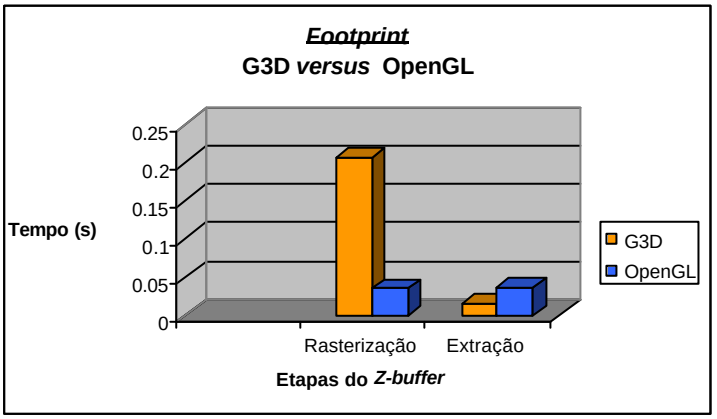


Figura 4.4 – Gráfico de comparação entre tempos de *Z-Buffer* das versões G3D e OpenGL

Espera-se que a pequena desvantagem na etapa de extração apresentada pela versão que se utiliza da aceleração de *hardware* possa ser superada quando empregada uma biblioteca gráfica como a Direct3D [Kovach,2000], que possibilite um acesso eficiente aos valores do Z-*Buffer*.

4.3.3 Tempos de processamento para modelos com transparência

Para o tratamento de transparência, o algoritmo de *Z-Buffer* da biblioteca G3D foi alterado para implementar um Z*list-buffer*, conforme descrito na seção 3.6. O parâmetro do nível de transparência é utilizado pelo *Z-Buffer* para determinar o número de quíntuplas RGB? ? que serão armazenadas para cada *pixel* da imagem do modelo poligonal renderizado.

No primeiro conjunto de testes para modelos com transparência foi empregado o contexto da Tabela 4.8:

Contexto do Teste	
Dado Volumétrico	*Syn64*
Dimensão do Volume	64^3
Opacidade do Volume	Opaco
Imagem Final	500x500
Modelo Poligonal	planos + cone
Número de Polígonos	302
Opacidade do Modelo Poligonal	Transparente
Níveis de Transparência do Modelo	0, 1, 3, 5
Biblioteca Gráfica	G3D
Plataforma de *Hardware*	Intergraph

Tabela 4.8 – Contexto do teste com transparência para *Syn64*

Diferentes níveis de transparência foram testados, visando estabelecer a influência dos mesmos no tempo final de processamento. A Tabela 4.9 mostra os tempos de CPU, em segundos, gastos pelos cinco algoritmos implementados, para as situações onde o *nível de transparência* assume os valores 0, 1, 3 e 5. O *coeficiente de transparência* associado ao modelo poligonal é 0.5, onde 0 equivale a totalmente transparente e 1 a totalmente opaco.

Como nos testes anteriores, uma vez que o tempo de processamento dos algoritmos também é dependente da direção de projeção dos dados, os tempos apresentados nesta tabela foram computados como uma média de 50 direções de projeção aleatoriamente distribuídas.

Analisando a Tabela 4.9, pode-se verificar que o algoritmo de baixa resolução continua sendo o mais eficiente. Observa-se em relação ao passo de *Z-Buffer* do modelo poligonal que, naturalmente, o número de níveis de transparência tem impacto sobre o tempo de processamento desta etapa. Nos testes realizados, esta diferença ocorreu na ordem de décimos de segundo. Quanto maior o número de níveis a serem considerados, maior o tempo gasto para a criação e obtenção dos valores armazenados no *Zlist-buffer*.

Syn64	Nível	Sem Polígonos	Composição em Baixa resolução	Composição em Alta resolução	Composição Dual	
					Footprint	Alta Freqüência
Z-Buffer do Modelo Poligonal	*0*	0.000	0.022	0.110	0.108	0.111
	1	0.000	0.025	0.109	0.111	0.107
	3	0.000	0.027	0.210	0.208	0.211
	5	0.000	0.030	0.294	0.292	0.292
Marcação das Regiões de Alta resolução	*0*	0.000	0.000	0.026	0.053	0.071
	1	0.000	0.000	0.026	0.052	0.070
	3	0.000	0.000	0.025	0.098	0.456
	5	0.000	0.000	0.025	0.101	0.610
Criação da Imagem Intermediária	*0*	0.020	0.017	0.439	0.360	0.330
	1	0.020	0.018	0.548	0.474	0.368
	3	0.020	0.021	0.603	0.525	0.362
	5	0.020	0.022	0.610	0.532	0.373
Warp	*0*	0.169	0.169	0.188	0.190	0.193
	1	0.169	0.169	0.185	0.189	0.192
	3	0.169	0.169	0.186	0.189	0.193
	5	0.169	0.170	0.185	0.190	0.193
Tempo Total	*0*	**0.189**	**0.208**	**0.763**	**0.711**	**0.704**
	1	**0.189**	**0.212**	**0.869**	**0.826**	**0.738**
	3	**0.189**	**0.217**	**1.024**	**1.021**	**1.221**
	5	**0.189**	**0.221**	**1.113**	**1.115**	**1.468**

Tabela 4.9 – Tempos para dado *Syn64* com níveis de transparência para modelo poligonal

Outra observação interessante, ainda sobre a Tabela 4.9, é que o algoritmo de composição em alta resolução obteve tempos próximos aos demais algoritmos de resolução dual, apesar do primeiro reamostrar para alta resolução todos os *voxels* do volume. Este fato deve-se à geometria do modelo poligonal testado, pois, devido às dimensões dos planos ortogonais presentes no modelo, quase todos os *pixels* da imagem intermediária são marcados para serem compostos em alta resolução. Assim, os tempos dos algoritmos duais, especialmente o de *footprint*, se aproximam do tempo obtido pelo algoritmo de alta resolução.

Uma segunda bateria de testes foi realizada, visando avaliar a influência do tamanho do volume nos tempos obtidos. Assim, o contexto da Tabela 4.10 apresenta os dados e parâmetros empregados nesta segunda bateria:

Contexto do Teste	
Dado Volumétrico	Pélvis
Dimensão do Volume	512x512x100
Opacidade do Volume	transparente
Imagem Final	500x500
Modelo Poligonal	prótese
Número de Polígonos	3758
Opacidade do Modelo Poligonal	Transparente
Níveis de Transparência do Modelo	0, 1, 3, 5
Biblioteca Gráfica	G3D
Plataforma de *Hardware*	Intergraph

Tabela 4.10 – Contexto para teste de transparência com o volume da Pélvis mais prótese

Da mesma forma que no teste com o *Syn64*, empregou-se o algoritmo de *Zlist-buffer* da biblioteca G3D para suportar os diferentes níveis de transparência testados.

A Tabela 4.11 mostra os tempos de CPU, em segundos, gastos pelos algoritmos implementados, para as situações em que o *nível de transparência* assume valores 0, 1, 3 e 5.

Como nos testes anteriores, uma vez que o tempo de processamento dos algoritmos também é dependente da direção de projeção dos dados, os tempos apresentados nesta tabela foram computados como uma média de 50 direções de projeção aleatoriamente distribuídas.

Pélvis+Prótese	**Nível**	**Sem Polígonos**	**Composição em Baixa resolução**	**Composição em Alta resolução**	**Composição Dual**	
					Footprint	**Alta Freqüência**
Z-Buffer do Modelo Poligonal	*0*	0.000	0.214	0.260	0.261	0.259
	1	0.000	0.213	0.263	0.216	0.264
	3	0.000	0.297	0.456	0.418	0.427
	5	0.000	0.389	0.560	0.584	0.565
Marcação das Regiões de Alta resolução	*0*	0.000	0.000	0.053	0.004	0.033
	1	0.000	0.000	0.054	0.003	0.034
	3	0.000	0.000	0.054	0.009	0.748
	5	0.000	0.000	0.051	0.011	1.051
Criação da Imagem Intermediária	*0*	13.536	13.681	49.537	24.971	24.533
	1	13.536	13.592	50.562	25.617	25.575
	3	13.536	13.376	50.990	25.152	24.580
	5	13.536	13.816	48.865	25.634	24.672
Warp	*0*	0.187	0.196	0.199	0.207	0.211
	1	0.187	0.190	0.203	0.203	0.225
	3	0.187	0.188	0.205	0.203	0.211
	5	0.187	0.196	0.194	0.211	0.213
Tempo Total	*0*	**13.723**	**14.091**	**50.048**	**25.443**	**25.036**
	1	**13.723**	**13.994**	**51.083**	**26.039**	**26.099**
	3	**13.723**	**13.862**	**51.705**	**25.781**	**25.967**
	5	**13.723**	**14.401**	**49.671**	**26.440**	**26.501**

Tabela 4.11 – Tempo da Pélvis combinada com prótese com diferentes níveis de transparência

O algoritmo de baixa resolução continua sendo a opção mais eficiente para os casos onde a qualidade da imagem não é crucial.

Neste exemplo também pode-se observar que o tempo gasto no passo de renderização do modelo poligonal aumenta, na ordem de décimos de segundo, à medida que o número de níveis de transparência é incrementado. O mesmo acontece com o passo de determinação de altas freqüências, para o algoritmo de alta freqüência, onde o tempo gasto para realizar a convolução sobre a imagem do modelo poligonal aumenta à medida que torna-se necessária a combinação dos *n* planos de imagem resultantes do *Z-*

Buffer.

Uma observação que pode ser feita a partir dos dois conjuntos testados com modelos transparentes, *Syn64* e Pélvis, diz respeito à influência da dimensão do volume frente ao uso de mais de um nível de transparência. Para o volume *Syn64*, o tempo gasto na etapa de *Z-Buffer* e de determinação de alta freqüência é significativo em relação ao tempo gasto com a criação da imagem intermediária. Isto ocorre porque, como as dimensões do volume são pequenas, o tempo de projeção dos *voxels* na imagem intermediária é da mesma ordem de grandeza que o tempo gasto na renderização dos polígonos, ou seja, décimos de segundo. Por isto, para o dado *Syn64*, a escolha de um número maior de níveis de transparência reflete diretamente no tempo total gasto para a criação da imagem final.

Esta mesma análise feita com o dado da Pélvis nos mostra que, como as dimensões deste volume são bem maiores, o tempo gasto com a renderização dos polígonos com diferentes níveis de transparência não é tão significativo em relação ao tempo total. Enquanto o tempo de renderização do modelo fica na ordem de centésimos de segundo, o tempo de composição dos *voxels* na imagem intermediária fica na ordem de segundos. Entretanto, o uso de mais de um nível de transparência pode, inclusive, contribuir para a redução do tempo de composição dos *voxels*, como é o caso do uso de 3 níveis de transparência, já discutido anteriormente nesta seção.

4.4 Tempos de Pré-processamento

Para que se obtenha uma maior eficiência em termos de tempo de processamento do algoritmo de *Shear-Warp*, é necessário empregar estruturas de dados que armazenam informações já pré-processadas a respeito do dado volumétrico (vide seções 2.3 e 2.4).

A etapa de composição dos *voxel*s pode ser acelerada através do uso de *run-length encode* do volume. São construídas estruturas onde somente os *voxels* não transparentes, já classificados através da função de transferência de opacidade, são armazenados (vide seção 2.4). Para cada função de transferência de opacidade associada a um determinado volume, uma estrutura RLE deve ser construída, uma única vez, sendo salva em disco. Na seção 4.4.1 são apresentados os tempos gastos com a construção do RLE para cada um dos dados volumétricos empregados nos testes já apresentados.

Ainda visando acelerar o passo de renderização dos *voxels*, os vetores gradientes associados aos mesmos são calculados em fase de pré-processamento. Além disso, para que se reduza a quantidade de memória necessária para seu armazenamento, os gradientes são quantizados para uma tabela de 256 vetores normais, conforme descrito na seção 2.4.

Da mesma forma que o RLE, o volume de gradientes quantizado é calculado uma única vez para cada dado volumétrico, sendo armazenado em disco e recuperado no início de cada sessão de visualização. A seção 4.4.2 apresenta os tempos gastos para o cálculo e a quantização dos vetores gradientes associados a cada um dos dados volumétricos empregados nos testes. São apresentadas, também, considerações a respeito da qualidade das imagens produzidas empregando-se os gradientes quantizados.

4.4.1 Construção do RLE do volume

O algoritmo de construção do RLE percorre o volume na sua ordem de armazenagem, fatia a fatia, empregando a função de transferência de opacidade para classificar os *voxels* como transparentes ou não transparentes. O algoritmo utiliza a função de transferência de opacidade apenas para verificar quais *voxels* irão contribuir para a imagem; o processo de tonalização propriamente dito acontece durante a etapa de composição.

O algoritmo percorre as linhas de uma fatia, criando uma lista de grupos de *voxels*. Um grupo consiste em uma seqüência de *voxels* contíguos, transparentes ou não. O algoritmo armazena o número de *voxels* presentes em cada seqüência, juntamente com os valores do dado associado aos *voxels* não transparentes.

As estruturas de dados criadas para a representação do RLE do volume são similares as propostas por Lacroute [Lacroute, 1995], sendo apresentadas na Figura 4.5: um vetor de grupos; um vetor de *voxels* não transparentes e um vetor de ponteiros para os outros dois vetores, que é utilizado para encontrar o início de cada linha de uma dada fatia do volume. No vetor *Run*, as entradas pares representam uma seqüência de *voxels* transparentes; as ímpares, representam as seqüências de *voxels* não transparentes. Cada nova linha inicia com um novo grupo, que pode ser transparente ou não.

O vetor *Voxel* contém as informações de valor e gradiente associadas a cada *voxel* não transparente. Apesar de não armazenar informações a respeito dos *voxels*

transparentes, o volume original pode ser reconstruído através da representação das seqüências de *voxels* de cada linha do volume, armazenadas no vetor *Grupo*.

O vetor *Pointer* faz a relação entre o vetor *Run* e o vetor *Voxel*. Cada entrada do vetor *Pointer* aponta para o primeiro grupo e para o primeiro *voxel* não transparente das linhas de uma dada fatia do volume.

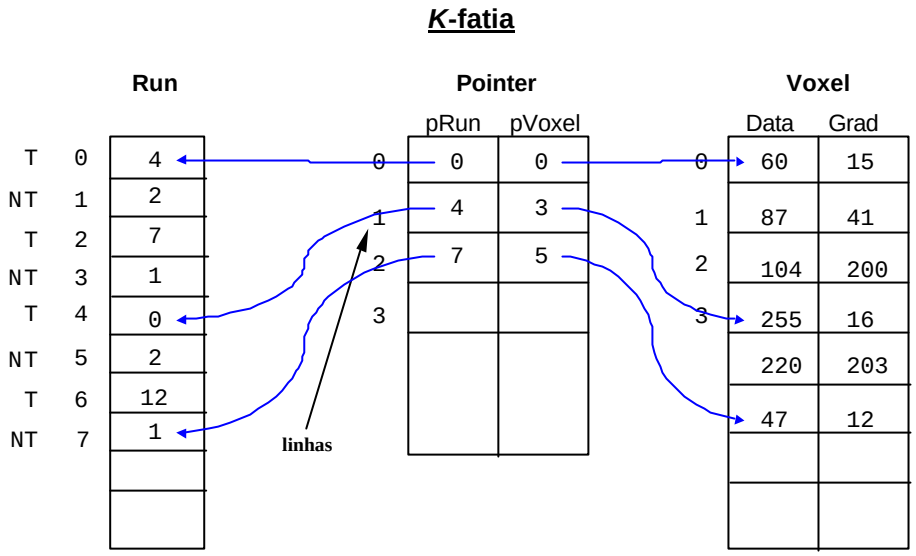


Figura 4.5 – Estruturas de dados empregadas no *run-length encode* do dado volumétrico

O algoritmo descrito acima percorre o volume na sua ordem de armazenagem, criando estruturas RLE que só podem ser percorridas segundo esta ordem. No entanto, durante o processo de composição dos *voxels*, o algoritmo de renderização pode necessitar percorrer o volume em uma ordem transposta, de acordo com o eixo principal de visualização (vide seção 2.1.1). A solução para este problema consiste em criar três representações de RLE para o volume, uma para cada um dos três eixos principais de visualização (X, Y, Z). O algoritmo de renderização seleciona qual representação a utilizar de acordo com a direção de visualização escolhida.

Estas três versões do RLE devem estar presentes em memória durante o processo de composição, evitando *page faults.* Esse também é um dos motivos de se buscar uma representação mais compacta para os vetores gradientes (vide seção 2.4), uma vez que os gradientes associados aos *voxels* não transparentes estão presentes nas três representações do RLE, carregadas em memória durante a renderização.

O trecho de código apresentado a seguir mostra a definição dos tipos de dados que representam o RLE do volume.

```
typedef unsigned char Data;
typedef unsigned char Grad;

typedef struct {
              Data d;
              Grad g;
} VoxelData;

typedef struct {
              unsigned int p_runs;
              unsigned int p_voxels;
} Pointers;

typedef struct {
  int ilen, jlen, klen;       /* tamanho de cada dimensao */
  unsigned  short int *Run;   /* vetor dos grupos */
  unsigned  int nRun;         /* numero total de grupos*/
  VoxelData *Voxel;           /* vetor de voxels nao-transp. */
  unsigned  int nVoxel;       /* numero entradas vetor voxels */
  Pointers  *Pointer;         /* vetor de ponteiros por linha */
  unsigned  int nPointer;     /* numero entradas vetor ponteiro */
}RLEVoxels;
```

O tempo de processamento gasto com a construção das três versões do RLE do volume não permite que esta operação seja realizada em tempo de renderização. A Tabela 4.12 apresenta estes tempos, em segundos, de construção das três versões do RLE para dois dos dados volumétricos apresentados na seção 4.1. O algoritmo de construção do RLE foi executado na plataforma Pentium III 500, descrita na seção 4.2.

Tempos de Construção do RLE do Volume		
Crânio da *Visible Woman*	127x127x51	30.59
	512x512x209	562.97
Pélvis	512x512x100	160.69

Tabela 4.12 – Tempos, em segundos, de construção do RLE do volume

Apesar do alto tempo de processamento para a criação do RLE do volume, o uso destas estruturas durante o processo de renderização é fundamental para a obtenção de tempos interativos. O ganho é dependente da coerência dos dados volumétricos e também da função de transferência de opacidade escolhida. A Tabela 4.13 apresenta os tempos de processamento, com e sem o uso de estruturas RLE, para o volume da Pélvis combinada com a prótese, com um nível de transparência para os polígonos, obtidos através do algoritmo de baixa resolução na versão OpenGL, empregando a plataforma Intergraph (vide seção 4.2).

Pélvis+Prótese - Baixa resolução		
Tempo total (s)	**Sem RLE**	**Com RLE**
	91.025	14.365

Tabela 4.13 – Tempos de renderização da Pélvis com e sem o uso do RLE do volume

4.4.2 Quantização do vetor gradiente

Os vetores gradientes calculados a partir dos valores escalares do dado volumétrico são quantizados para um conjunto menor de vetores normais (vide seção 2.4), visando reduzir a quantidade de memória alocada para a representação do gradiente dentro das estruturas de RLE do volume.

Esta quantização é realizada em uma etapa de pré-processamento, sendo armazenados em disco os índices dos gradientes quantizados para que sejam posteriormente utilizados no passo de renderização dos *voxels*. Realizando o cálculo dos gradientes numa etapa de pré-processamento, diminui-se o tempo do passo de renderização, uma vez que não é necessário calcular os gradientes em tempo de execução, apenas consultá-los.

A Tabela 4.14 mostra o tempo de processamento gasto, em segundos, para a criação dos gradientes quantizados do volume para uma tabela de 256 entradas de vetores normais. Estes tempos foram obtidos a partir da plataforma Pentium III 500, detalhada na seção 4.2.

Tempos de Quantização do Gradiente		
Cabeça da *Visible Woman*	127x127x51	17.19
	512x512x209	1158.97
Pélvis	512x512x100	525.36

Tabela 4.14 – Tempos de quantização do gradiente

Quantizar o vetor gradiente pode introduzir uma redução em termos de qualidade da imagem final gerada. As Figuras 4.6(a) e 4.6(b) mostram duas imagens de crânio, uma criada utilizando os vetores gradientes originais e outra usando os gradientes quantizados para uma tabela de 256 vetores normais. Nas áreas das imagens que correspondem às superfícies que variam sua curvatura suavemente, as diferenças entre as imagens 4.6(a) e 4.6(b) são notáveis; nas outras áreas, contudo, há diferenças

muito pequenas. Os outros dois pares de imagens nas Figuras 4.6(c), 4.6(d), 4.6(e) e 4.6(f) ilustram situações onde a quantização do gradiente não diminui significativamente a qualidade da imagem.

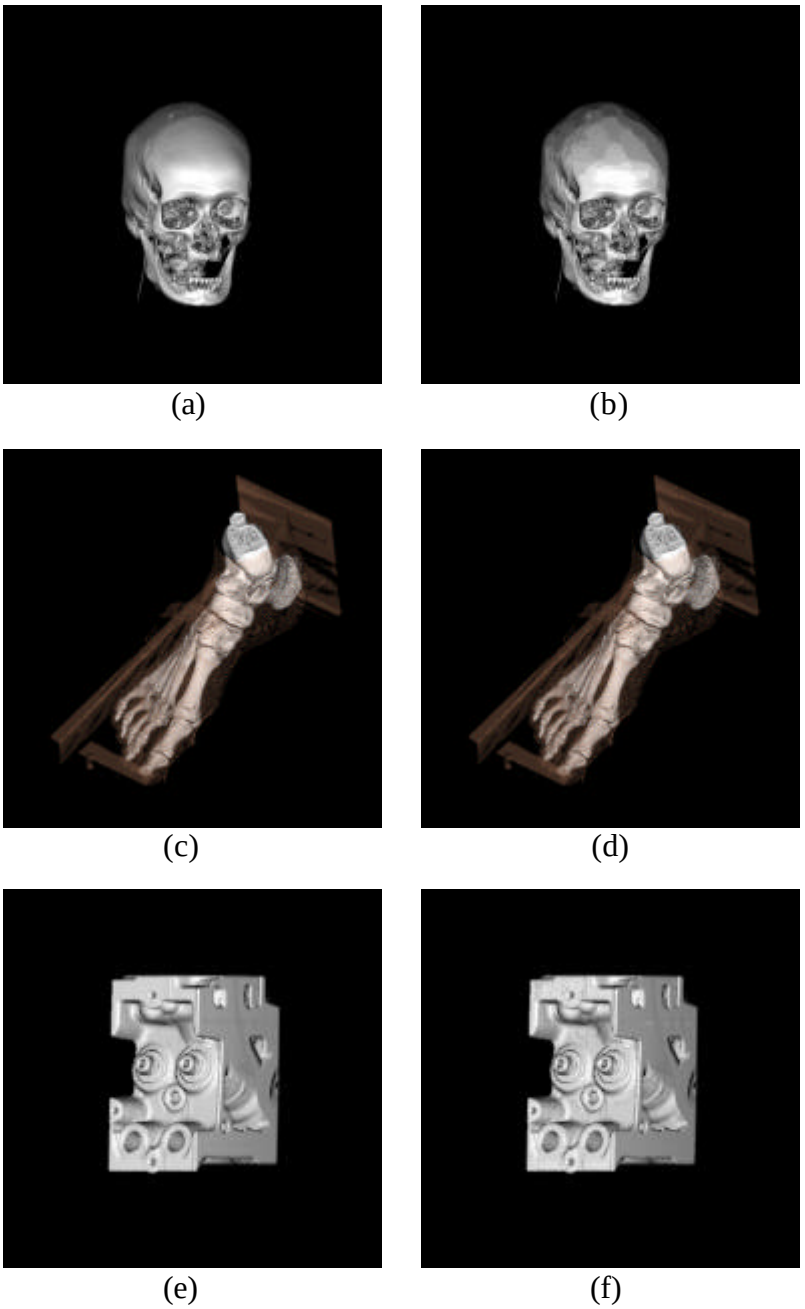


Figura 4.6 – (a) Crânio renderizado com gradiente originais e (b) com quantização.
(c) Pé renderizado com transparência e gradientes originais e (d) com quantização.
(e) Motor renderizado com gradientes originais e (f) com quantização.

Para os casos onde existem regiões de curvatura suave no dado volumétrico, melhores resultados podem ser obtidos quantizando-se o gradiente para um conjunto maior de vetores normais pré-calculados. No entanto, isto implica num aumento na

quantidade de memória alocada para representar o índice da tabela de normais, fator que deve ser considerado antes da escolha desta opção.

4.5 Qualidade de Imagem

Para que se tenha a noção correta de custo *versus* benefício de cada um dos algoritmos propostos, deve-se, também, considerar a qualidade da imagem final gerada.

Os resultados da avaliação de qualidade de imagem são discutidos em 3 seções: na seção 4.5.1, ressaltam-se diferenças no tratamento de *aliasing* e presença de *blur* em imagens resultantes de cada um dos algoritmos propostos; na seção 4.5.2, discute-se a necessidade do uso de modelos poligonais com transparência para que a imagem final seja criada corretamente; na seção 4.5.3, apresentam-se imagens onde o emprego de mais de um nível de transparência para o modelo poligonal é fundamental para a criação de imagens finais corretas, especialmente quando o dado volumétrico também possui certo nível de transparência.

4.5.1 Imagens produzidas pelos algoritmos híbridos

Para comparar a qualidade das imagens criadas a partir dos algoritmos propostos, foram geradas quatro versões da imagem do volume da Pélvis, combinada com o modelo poligonal da prótese, empregando 1 nível de transparência, conforme apresentado na Figura 4.7. O dado volumétrico da Pélvis está mapeado como semitransparente, através de uma função de transferência de opacidade que atribui valores semitransparentes para a maioria das densidades presentes no volume. Como será visto na seção 4.5.2, a combinação de volumes semitransparentes com modelos poligonais com transparência é uma das contribuições importantes deste trabalho.

Estas imagens foram geradas utilizando a versão OpenGL dos algoritmos propostos, uma vez que o modelo de iluminação desta biblioteca cria imagens mais suaves que as obtidas a partir da biblioteca G3D. A plataforma Intergraph (vide seção 4.2) foi empregada na criação destas imagens. Os tempos de processamento obtidos para a criação da imagem final (cuja dimensão é 1024^2), para cada um dos algoritmos, são apresentados na Tabela 4.15.

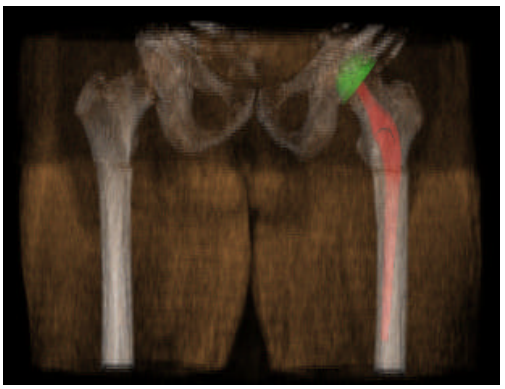


Figura 4.7 – Volume da Pélvis combinada com o modelo de prótese, com 1 nível de transparência

Pélvis+prótese (imagem 1024^2)	Tempo (s)
Composição em Baixa Resolução	14,07
Composição em Alta Resolução	202,26
Footprint	33,11
Alta Freqüência	30,22

Tabela 4.15 – Tempos de processamento para a criação da imagem da Pélvis+prótese

As imagens apresentadas na Figura 4.8 são ampliações de uma mesma região de interesse obtida das imagens originais criadas por cada um dos algoritmos. Esta região foi escolhida por evidenciar características que serão analisadas a seguir.

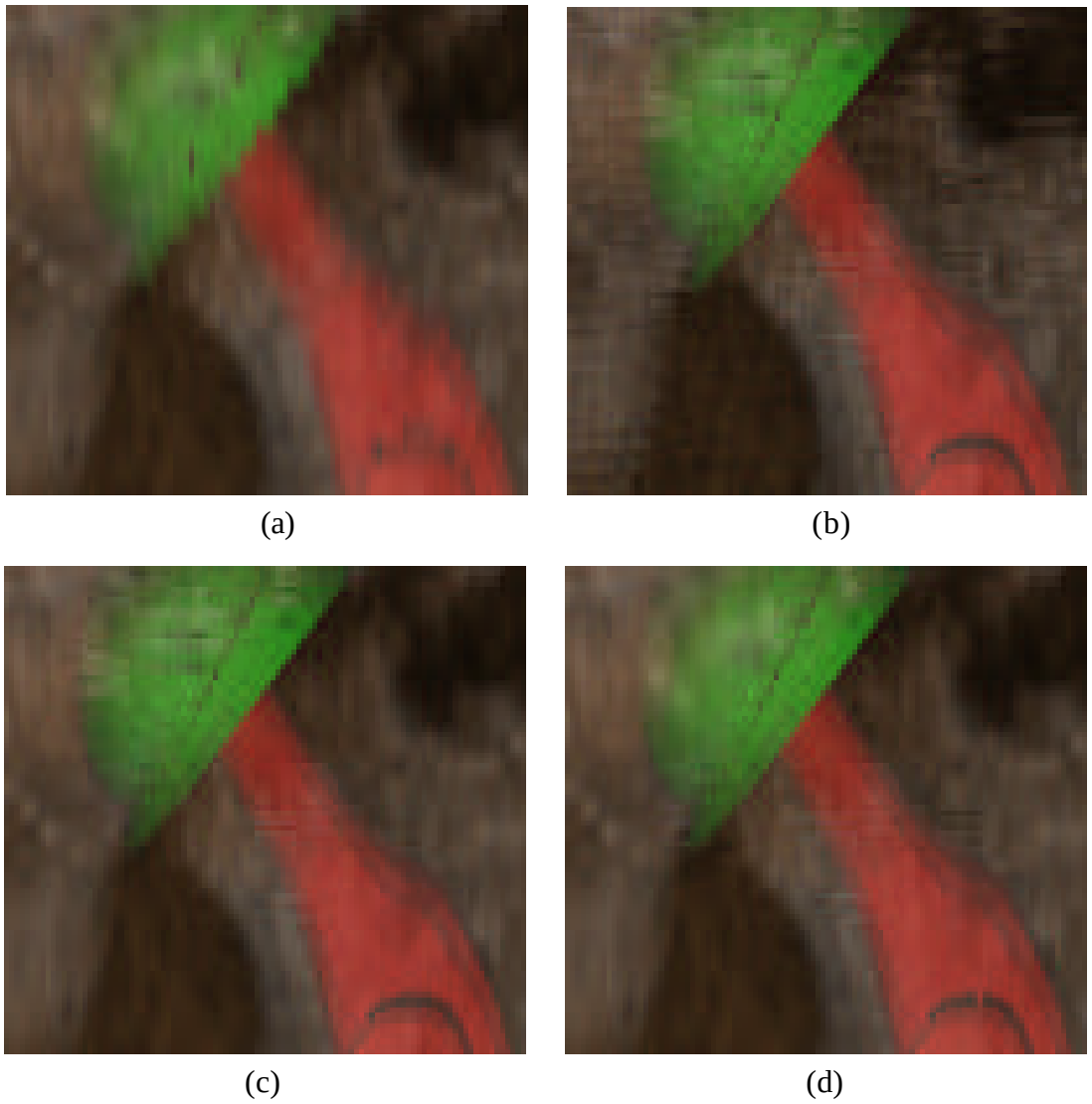
A Figura 4.8(a) mostra a imagem criada com o algoritmo de baixa resolução. Como pode-se observar, ela está borrada (*blur*) e com forte presença de *aliasing* junto às bordas do modelo poligonal. No entanto, este algoritmo apresenta o menor tempo de processamento para a criação da imagem final, como já foi analisado na seção 4.3 e pode ser observado através da Tabela 4.15.

A Figura 4.8(b) apresenta a imagem produzida pelo algoritmo de alta resolução. Esta é a imagem de melhor qualidade, sem borrões e com o efeito de *aliasing* junto às bordas do modelo poligonal bastante reduzido. No entanto, este algoritmo é o mais lento na geração da imagem final, conforme mostra Tabela 4.15.

A Figura 4.8(c) mostra os resultados do algoritmo de *footprint*. Aqui, também tem-se uma imagem de boa qualidade, sem a presença de borrões nas áreas onde a prótese está combinada com o volume, com *aliasing* reduzido nas regiões de borda e com o segundo menor tempo de processamento dentre os algoritmos que trabalham com a criação de uma imagem intermediária em alta resolução.

A Figura 4.8(d) apresenta a imagem criada com o algoritmo de alta freqüência,

usando 120 como *valor limite de freqüência*. Como pode-se observar, algumas regiões da imagem contidas dentro da prótese estão borradas porque foram compostas em baixa resolução (região superior esquerda da concha). Entretanto, as bordas do modelo estão bem definidas e com reduzida presença de *aliasing*.



(a) (b)

(c) (d)

Figura 4.8 – Região ampliada das imagens criadas pela composição em (a) baixa resolução; (b) alta resolução; (c) *footprint* e (d) alta freqüência

Como observação final, para este teste, pode-se ressaltar que a diferença em relação à qualidade de imagem entre os algoritmos de *footprint* e alta freqüência é muito pequena, ocorrendo próximas às regiões de baixa freqüência e em alguns *pixels* da borda que recebem a contribuição do modelo poligonal. Esta diferença é apresentada pela Figura 4.9, onde foi aplicado um fator de correção *gamma* de 2.5.

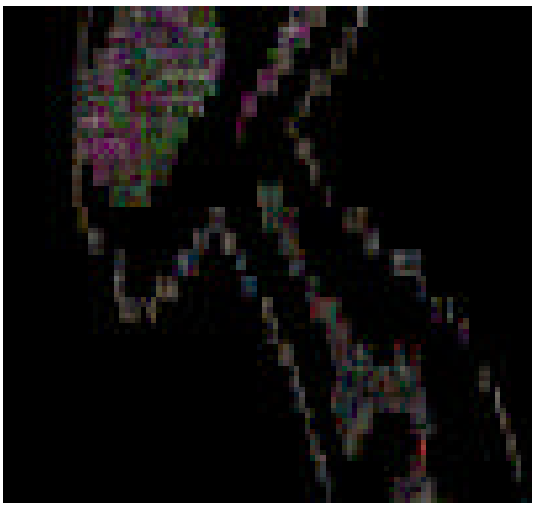
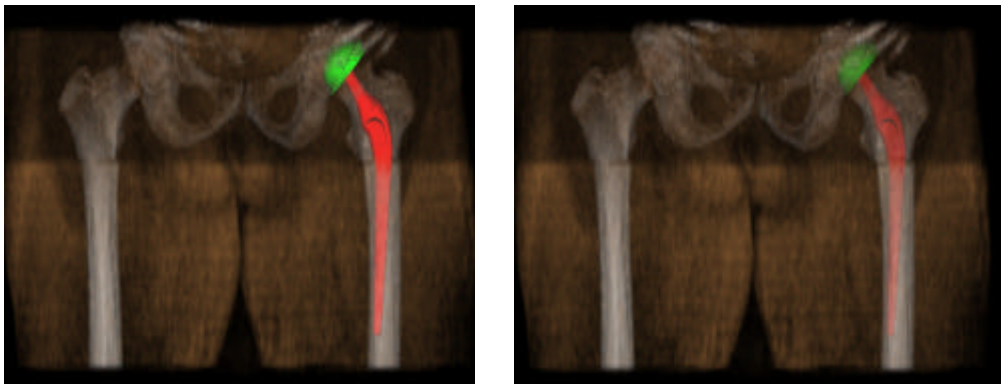


Figura 4.9 – Diferença entre as imagens do *footprint* e alta freqüência (*gamma-2.5)*

4.5.2 Uso de transparência no modelo poligonal

A Figura 4.10 apresenta duas imagens do volume da Pélvis com semitransparência combinada com a prótese, criadas com o algoritmo de baixa resolução. Na Figura 4.10(a), pele, músculos e ossos são transparentes e a prótese é opaca. Na Figura 4.10(b), o volume continua semitransparente e a prótese é renderizada com um valor de opacidade de 0.3, considerando um nível de transparência.



(a) (b)

Figura 4.10 – Imagens da Pélvis semitransparente combinada com a prótese poligonal. (a) Prótese é opaca. (b) Prótese com 1 nível de transparência

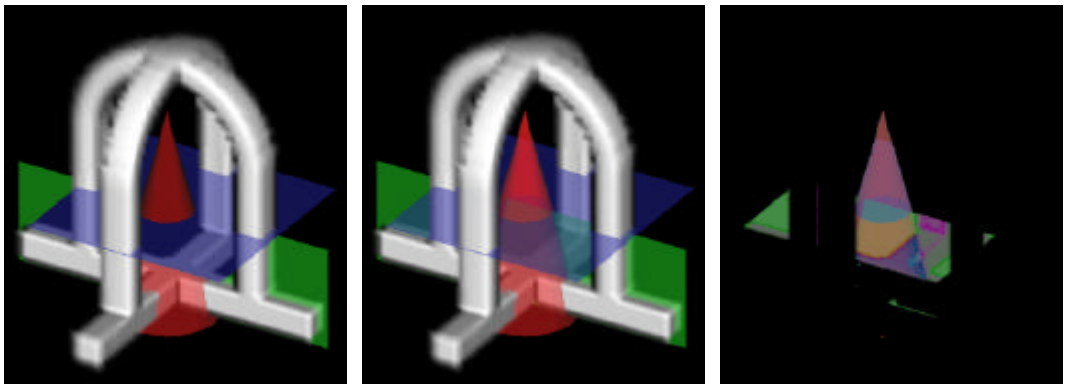
Como podemos ver na Figura 4.10(a), a imagem criada com a prótese opaca parece “errada”, como se a prótese estivesse posicionada fora do osso do fêmur. Por outro lado, a Figura 4.10(b), criada com um nível de transparência para a prótese, dá uma noção correta da posição da prótese dentro do osso do fêmur. Os resultados apresentados nas imagens acima enfatizam a necessidade de associação de transparência ao modelo poligonal, especialmente quando o volume também é renderizado com um

certo grau de transparência.

Neste exemplo, na Figura 4.10(a), a imagem aparece “errada” porque a camada de *voxels* que antecede o modelo poligonal é muito delgada, fazendo com que, no caso em que o modelo é opaco, prevaleça a cor associada aos polígonos, dando a sensação de que não existe “osso” rodeando a prótese. A partir do momento em que o modelo poligonal passa a ser transparente, o peso da contribuição dos *voxels* que circundam a prótese passa a ser maior, acumulando corretamente cor e opacidade e criando uma imagem final correta.

4.5.3 Vários níveis de transparência do modelo poligonal

Dependendo do dado volumétrico e do modelo poligonal combinados, a associação de diferentes níveis de transparência aos polígonos pode ou não apresentar resultados relevantes na imagem final. As Figuras 4.11(a), 4.11(b) e 4.11(c) apresentam imagens do dado *Syn64* combinado com o modelo de planos e cone, onde fica evidente que o emprego de diferentes níveis de transparência é fundamental para a composição correta da imagem final.

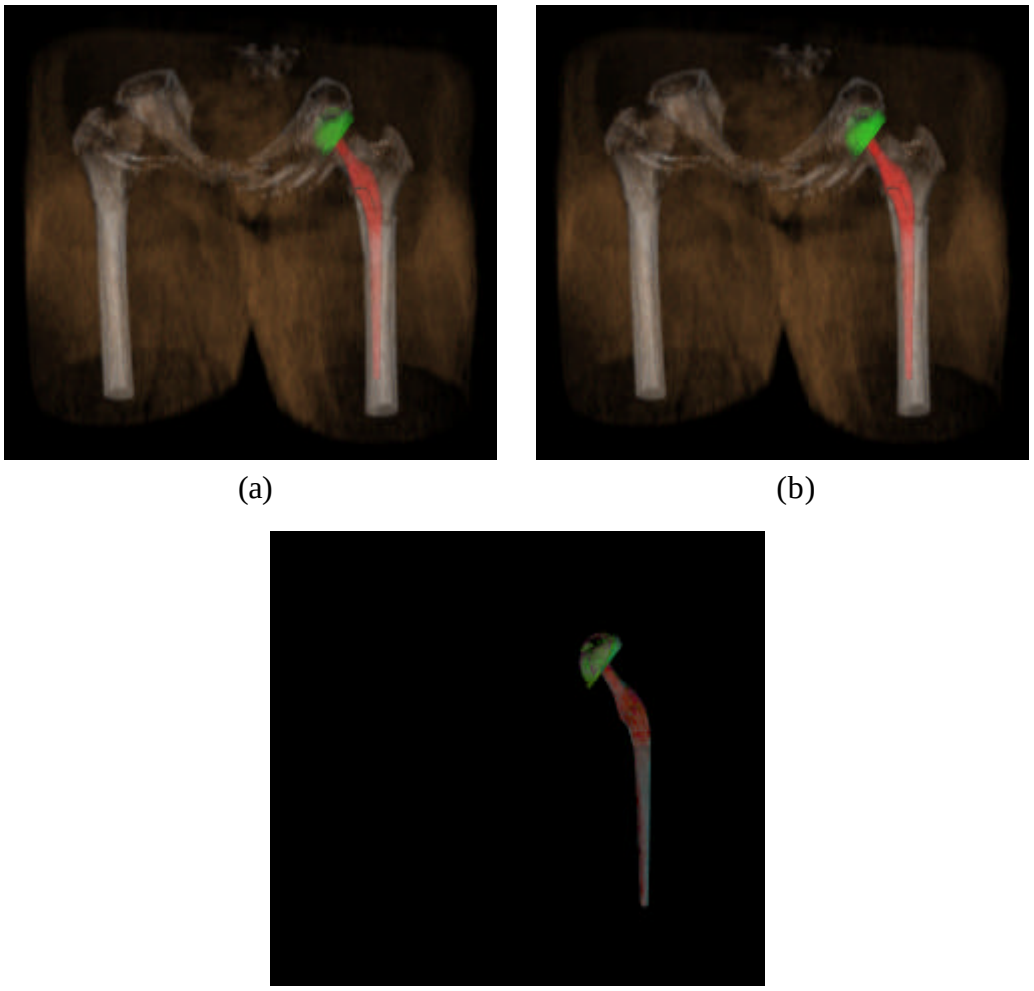


(a) (b) (c)

Figura 4.11 – Dado *Syn64* e modelo poligonal com (a) 1 nível de transparência; (b) com 3 níveis de transparência e (c) diferença entre (a) e (b), com fator *gamma*=2.5

A Figura 4.11(a) foi criada com apenas 1 nível de transparência para os polígonos e, por este motivo, a parte central do cone não está presente na imagem. Já na Figura 4.11(b), que foi criada empregando 3 níveis de transparência para os polígonos, todas as porções do modelo estão representadas na imagem. A Figura 4.11(c) mostra a diferença entre as imagens de (a) e (b), aplicando um fator de correção *gamma* igual a 2.5.

A Figura 4.12 mostra imagens do volume da Pélvis combinada com a prótese, onde a associação de mais de um nível de transparência aos polígonos não apresenta efeitos sensíveis em termos de qualidade da imagem final gerada. A Figura 4.12(a) foi criada empregando somente 1 nível de transparência, enquanto que a Figura 4.12(b) foi criada com 3 níveis de transparência. Na Figura 4.12(c) é apresentada a diferença entre as imagens 4.12(a) e 4.12(b), com fator de correção *gamma* igual a 2.5, evidenciando que, para este caso, o uso de mais de um nível de transparência acrescenta pouca informação para a melhora da qualidade da imagem final.



(a) (b)

(c)

Figura 4.12 – Pélvis combinada com a prótese com (a) 1 nível de transparência e (b) 3 níveis de transparência. Em (c) tem-se a imagem da diferença entre as duas imagens anteriores.

5 CONCLUSÕES E TRABALHOS FUTUROS

Nesta tese foram apresentados quatro algoritmos que integram a fatoração *Shear-Warp* ao algoritmo de *Z-Buffer* para a combinação de dados volumétricos e modelos poligonais: composição em baixa resolução, em alta resolução, *footprint* e alta freqüência.

Os resultados obtidos empregando-se estes algoritmos mostram que eles combinam visualmente de forma correta dados volumétricos com modelos poligonais, com ou sem a presença de transparência. Foram implementadas diferentes estratégias de redução de *aliasing*, visando alcançar a melhor relação entre tempo de processamento e qualidade da imagem gerada.

É importante ressaltar que os algoritmos propostos combinam dados volumétricos opacos e/ou semitransparentes com modelos poligonais também opacos e/ou semitransparentes. A capacidade de visualizar corretamente volume e modelo poligonal semitransparentes é um dos diferenciais entre este trabalho em relação aos demais trabalhos encontrados na bibliografia e que também tratam modelos poligonais. Os resultados apresentados nesta tese, através das imagens de volume e polígonos semitransparentes, enfatizam a necessidade de associação de transparência ao modelo poligonal para a criação de imagens finais visualmente corretas, especialmente quando o volume também é renderizado com um certo grau de transparência.

O algoritmo de composição em baixa resolução apresenta o menor tempo para realizar a combinação de volume e polígonos; entretanto, a imagem final gerada pode apresentar problemas de *aliasing*, principalmente nas bordas do modelo poligonal.

Por outro lado, o algoritmo de alta resolução gera a imagem final de melhor qualidade, mas também apresenta o tempo de processamento mais elevado. Este alto

tempo é obtido mesmo utilizando-se as otimizações de RLE das fatias do volume e da imagem intermediária. Sem o uso destas otimizações, o tempo de processamento torna-se totalmente inaceitável.

Os algoritmos de resolução dual geram imagens finais de qualidade comparável àquelas criadas pelo algoritmo de alta resolução, apresentando melhor performance em termos de tempos de processamento. A partir dos testes realizados, pode-se indicar o uso do algoritmo de alta freqüência para os casos em que não há a presença de transparência, e o de *footprint* para os casos com volume e/ou modelo poligonal transparente.

O emprego das estruturas de *run-length encode* do volume e da imagem intermediária é imprescindível para a obtenção de tempos interativos na renderização do dado volumétrico. Entretanto, o tempo de construção da estrutura de RLE do volume torna proibitiva a sua criação durante o processo de geração das imagens. Assim, para manter os tempos interativos durante o processo de visualização, as funções de transferência de opacidade são ajustadas em uma etapa de pré-processamento. Uma vez definida a função de transferência de opacidade, é construído o RLE do volume, classificando os *voxels* segundo esta função.

O emprego das técnicas de RLE do volume e *early ray termination*, que aceleram o algoritmo de *Shear-Warp*, é eficiente quando a função de transferência de opacidade escolhida é aplicada a volumes com um alto grau de coerência. Volumes classificados que contenham grandes regiões transparentes ou grandes regiões opacas resultam em menores tempos de processamento. Para dados volumétricos ruidosos, como, por exemplo, dados sísmicos, se a função de transferência de opacidade não classificar apropriadamente os *voxels*, criando uma coerência aparente, o emprego de estruturas RLE para o volume oferece pouco ganho no tempo total de processamento [Gerhardt,1998; Lacroute,1995].

A etapa de renderização dos *voxels*, subdividida em tonalização, reamostragem e composição, é a mais custosa dentro do processo de criação da imagem final, mesmo empregando-se otimizações de RLE do volume e da imagem intermediária. Por este motivo, a obtenção de tempos interativos depende do tamanho do dado volumétrico e da função de transferência associada ao mesmo.

Visando-se melhorar o desempenho da etapa de renderização dos *voxels*, podem-

se produzir imagens de qualidade mais baixa através da diminuição do número de *voxels* processados, reduzindo potencialmente o custo das três etapas associadas à renderização do volume. Técnicas como a de amostragem adaptativa [Levoy,1990a] e a subamostragem de volumes [Laur,1991] podem ser empregadas conjuntamente com o *Shear-Warp*, visando reduzir adequadamente o número de *voxels* a serem tratados.

Soluções que implementem em *hardware* as etapas de classificação, iluminação e composição dos *voxels* podem manipular dados volumétricos em tempo real. A placa VolumePro [Mitsubishi] é a primeira placa gráfica capaz de realizar visualização em tempo real de dados volumétricos da ordem de 512^3 *voxels*. Contudo, esta solução ainda não incorpora a visualização de modelos poligonais combinados com os dados volumétricos.

Os algoritmos desenvolvidos nesta tese são independentes da presença de *hardware* especializado para a criação das imagens combinadas, oferecendo maior flexibilidade quanto às dimensões dos volumes a serem renderizados. O fato de não se utilizar *hardware* especializado faz com que os tempos de processamento obtidos pelos algoritmos propostos possam ser aprimorados à medida que estações com processadores mais rápidos estejam disponíveis no mercado. Além disso, os algoritmos propostos podem ser portados para máquinas de uso geral, respeitando a relação memória e espaço em disco *versus* tamanho dos dados volumétricos a serem tratados.

Pode-se tirar mais proveito de bibliotecas gráficas, como a OpenGL, que possuem as funções de rendering de polígonos aceleradas em *hardware* em diversas placas gráficas, desde que seja oferecido um controle eficiente do *Z-Buffer*. Por controle eficiente do *Z-Buffer* nós entendemos a habilidade de decodificar facilmente os valores de profundidade e prover um acesso eficiente a estes valores, copiando-os para estruturas alocadas na memória principal. A biblioteca OpenGL oferece o acesso à matriz de valores *Zs*, mas nos testes realizados este acesso mostrou-se lento quando comparado com o da implementação do *Z-Buffer* em *software*.

O acesso eficiente ao *Z-Buffer* não é uma preocupação dos fabricantes de placas gráficas, uma vez que o *pipeline* de renderização de polígonos é, quase que exclusivamente, empregado para a visualização de cenas compostas somente de polígonos, onde não é necessário o acesso a este tipo de informação, já que toda a etapa de composição e exibição da cena fica por conta da placa gráfica.

Pode-se dizer, então, que a estação de trabalho ideal para a visualização de dados volumétricos combinados com modelos poligonais é aquela que reuna numa única placa gráfica especializada a renderização do dado volumétrico e a renderização do modelo poligonal. Hoje em dia, estas duas tecnologias já existem, porém implementadas em *hardwares* distintos. Assim, a implementação em *software* de algoritmos híbridos que realizem a combinação de dados volumétricos com modelos poligonais continua sendo uma solução que viabiliza este tipo de visualização.

Com base nos resultados obtidos nesta tese pode-se estabelecer que para a visualização híbrida de dados volumétricos não extremamente grandes (ordem de 512^3 *voxels*), a estação de trabalho desejável deve possuir as seguintes características:

- a quantidade de memória RAM disponível deve ser da ordem de 512Mb a 1Gb;
- deve ser utilizado um processador com bom desempenho, lembrando que quanto maior a capacidade de processamento disponível menor será o tempo gasto na etapa de renderização do volume;
- deve ser empregada uma placa gráfica que viabilize um controle eficiente do *Z-Buffer*, reduzindo o tempo de acesso e retirada dos mapas de cor, opacidade e profundidade.

5.1 Trabalhos Futuros

Vários pontos apresentados neste trabalho podem ser estendidos, visando acrescentar novas facilidades e aumentar o desempenho dos algoritmos propostos.

Focando a obtenção de melhores tempos de processamento para os algoritmos híbridos, podem-se implementar etapas como a de *warp* da imagem intermediária e a de convolução da imagens oriundas do *Z-Buffer* do modelo poligonal, empregada no algoritmo de alta freqüência, através de funções oferecidas pelas bibliotecas gráficas aceleradas por *hardware*. Com isso, espera-se reduzir o tempo gasto nessas etapas, que estão sendo implementadas somente via *software*, aumentando a performance dos algoritmos quando do emprego de aceleração através do *hardware* das placas gráficas.

Pode ser implementada uma versão dos algoritmos empregando funções da biblioteca gráfica Direct3D [Kovach,2000], visando comparar a eficiência no controle do *Z-Buffer* com a implementação já testada da biblioteca OpenGL. Espera-se conseguir

melhores resultados, uma vez que o acesso a estes mapas na implementação Direct3D tende a ser mais rápido, já que eles estão disponíveis em memória, sem que seja necessária uma cópia destes valores para estruturas internas dos algoritmos.

É recomendável o desenvolvimento de uma interface com o usuário que permita o instanciamento dos modelos poligonais em relação ao dado volumétrico. Esta interface deve prover funções para translação, rotação e escala do modelo como um todo ou de partes dele, para que ele possa ser corretamente posicionado dentro do espaço em que o dado volumétrico está definido.

A importação de modelos poligonais criados através de ferramentas especializadas para a modelagem geométrica também pode ser incorporada a este trabalho. A importação de modelos vetoriais descritos em formatos como DXF, DGN, etc., permite que o usuário faça uso de ferramentas já existentes para a criação de modelos poligonais que serão combinados com o dado volumétrico.

Pode-se desenvolver uma ferramenta mais complexa e ambiciosa para a realização da modelagem e do posicionamento do modelo poligonal em relação ao dado volumétrico. Note-se que as superfícies dos objetos poliedrais devem se conformar com os objetos que estão implícitos dentro do volume. Assim, os objetos poligonais podem ser criados (ou adaptados) em função da geometria dos objetos volumétricos.

Para tanto, deve-se integrar a modelagem do objeto poligonal ao processo de segmentação de objetos presentes no dado volumétrico. Ou seja, as superfícies devem estar onde a segmentação do objeto implícito indicar. No caso de uma prótese já existente (com forma e geometria já conhecida), este ajuste se faz através de translações e rotações. No caso geral, tem-se um ajuste de forma e geometria de acordo com os objetos presentes no dado volumétrico.

É interessante, também, prover uma interface para a associação de cor e opacidade ao modelo poligonal, quer seja como um todo, quer através da definição de diferentes funções para grupos de polígonos. No protótipo atual, a função de transferência de cor e opacidade do modelo poligonal é especificada de forma textual, sendo incluída no arquivo que possui a definição de cada face que compõe o modelo.

O desenvolvimento de ferramentas para exploração e análise do dado volumétrico combinado com o modelo poligonal também é um tema importante, especialmente visando direcionar estas técnicas de visualização híbrida para dados

obtidos através de exames médicos. Facilidades como o estabelecimento de planos de corte, a realização de medidas quantitativas, a manipulação das fatias do volume com ferramentas de ampliação, mudança de brilho e contraste, a visualização do histograma de cada fatia, dentre outras, são indispensáveis para a construção de um ambiente de exploração eficiente de dados médicos.

Visando aplicações de navegação 3D e exploração dos modelos híbridos é interessante implementar os algoritmos de renderização do volume e modelos poligonais empregando projeção perspectiva. Este é o modelo natural da nossa visão e de todos os sistemas de captura de imagens através de câmeras. Assim, para inúmeras aplicações onde se deseja simular a presença de uma câmera ou observador virtual em movimento, a projeção perspectiva deve ser utilizada. Como exemplo de aplicações que necessitam do emprego de projeção perspectiva pode-se citar a simulação de exames de laparoscópicos, como a colonoscopia virtual, a bronqueoscopia e a laringoscopia, dentre outros.

6 BIBLIOGRAFIA

Barenholtz, J. e Dewhurst, C. A Run Length Encoding Scheme for Real Time Video Animation. *Proceedings of Northwest'96*, Washington, EUA, 63-69, **1996**.

Brigham and Women's Hospital. *Surgical Planning Laboratory – SPL*. [online] Disponível na Internet via WWW. URL:http://splweb.bwh.harvard.edu:8000/, Boston, EUA, **2000**.

Carpenter, L. The A-Buffer, an Antialiased Hidden Surface Method. *Computer Graphics Proceedings, Annual Conference Series - SIGGRAPH'84*, 103-109, **1984**.

Drebin, R. A., Carpenter, L., and Hanrahan, P., Volume Rendering, *Computer Graphics*, **22**, 4, 65-74, **1988**.

Diamond Video Cards. *Graphics Viper 770*. [online] Disponível na Internet via WWW. URL: http://www.diamondvideocards.com/.

Eckel, G. e Grzeszczuk, R. *OpenGL Volumizer Programmer's Guide*. Silicon Graphics, Inc, Mountain View, CA, EUA, **1998**.

Foley, J. D.; van Dam, A.; Feiner, S. K. e Hughes, J. F. *Computer Graphics: Principles and Practice*. Addison-Wesley, New York, 2nd Edn, 92-100, **1990**.

Gerhardt, A. e outros. Aspects of 3D Seismic Data Volume Rendering. *3D Modeling of Natural Objects: A Challenge for the 2000's - Proceedings***,** ENSG - Ecole Nationale Supérieure de Géologie**,** Nancy, França, Junho, **1998**.

Gouraud, H. Continuous Shading of Curved Surfaces, *IEEE Transactions of Computers*, **20**, 6, 623-629, **1971**.

Graphics Workstations. Intergraph TDZ 2000 3D Graphics Workstation. *PC-Magazine Online*. [online] Disponível na Internet via WWW. URL: http://www.zdnet.com/pcmag/edchoice/1704/1704p1.html, Fevereiro, **1998**.

Kaufman, A. e Shimony, E. 3D Scan-Conversion Algorithms for Voxel-Based Graphics. *Proceedings ACM Workshop on Interactive 3D Graphics*, Chapel Hill, NC, EUA, 45-75, **1986**.

Kaufman, A. Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes. *Computer Graphics*, **21**, 4, 171-179, **1987**.

Kaufman, A. Efficient Algorithms for 3D Scan-Converting Polygons. *Computers & Graphics*, **12**, 2, 213-219, **1988**.

Kaufman, A.; Yagel, R. e Cohen, D. Intermixing Surface and Volume Rendering. *3D Imaging in Medicine: Algorithms, Systems and Applications*. K. H. Hoehne, H. Fuchs, and S. M. Pizer (Eds.), Springer-Verlag, 217-228, **1990**.

Kaufman, A. Volume Visualization: Principles and Advances. *SIGGRAPH'99 Course Notes - Advances in Volume Visualization*, Los Angeles, EUA, **1999**.

Kovach, P. J. e Richter, J. *Inside Direct3D: The Developers Guide to Real-Time 3D Power and Performance for Microsoft*. Microsoft Press, Março, **2000**.

Kreeger, K. e Kaufman, A. Hybrid Volume and Polygon Rendering with Cube Hardware. *Proceedings of the 1999 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 14-24, **1999**.

Kreeger, K. e Kaufman, A. Mixing Translucent Polygons with Volumes. *Proceedings of the IEEE Visualization'99*, San Francisco, EUA, **1999a**.

Lacroute, P. e Levoy, M. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. *Computer Graphics Proceedings, Annual Conference Series - SIGGRAPH'94*, Orlando, EUA, 451-458, **1994**.

Lacroute, P. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. Technical Report: CSL-TR-95-678, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, EUA, **1995**.

Lacroute, P. Analysis of a Parallel Volume Rendering System Based on the Shear-Warp Factorization. *IEEE Transactions on Visualization and Computer Graphics*, **2**, 3, 218-231, **1996**.

Laur, D. e Hanrahan, P. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. *Computer Graphics*, **25**, 4, 285-288, **1991**.

Leech, J. P. Evenly distributed points on a sphere. [online] Disponível na Internet via WWW. URL: http://www.math.niu.edu/~rusin/known-math/96/repulsion, **1992**.

Levoy, M. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, **5**, 3, 29-37, **1988**.

Levoy, M. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, **9**, 3, 245-261, **1990**.

Levoy, M. A Hybrid Ray-Tracer for Rendering Polygon and Volume Data. *IEEE Computer Graphics and Applications*, **10**, 3, 33-40, **1990a**.

Lichtenbelt, B. Design of A High Performance Volume Visualization System. *Proceedings of the 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 111-120, **1997**.

Lorensen, W. E. e Cline, H. E. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, **21**, 4, 163-170, **1987**.

McCormick, B.; Defanti, T. e Brown, M. Visualization in Scientific Computing. *Computer Graphics*, **21**, 6, Novembro, **1987**.

McReynolds, T.; Blythe, D.; Fowle, C.; Grantham, B.; Hui, S. e Womack, P. Programming with OpenGL: Advanced Rendering. *SIGGRAPH ´97 Lecture Notes, Course 11*, 144-153, **1997**.

Mitsubishi. *VolumePro, 3D Real Time Volume Rendering PCI Hardware Board*. [online] Disponível na Internet via WWW. URL: http://www.rtviz.com/, **1999-2000**.

Miyazawa, T. e Koyamada, K. A High-Speed Integrated Renderer for Interpreting Multiple 3D Volume Data. *The Journal of Visualization and Computer Animation*, **3**, 65-83, **1992**.

Montani, C. e Scopigno R. Rendering Volumetric Data using the STICKS Representation Scheme. *Computer Graphics*, **24**, 5, 87-93, Novembro, **1990**.

Osborne, R.; Pfister, H.; Lauer, H.; McKenzie, H.; Gibson, S.; Hiatt, W. e Ohkami, T. EM-Cube: An Architecture for Low-Cost Real-Time Volume Rendering. *Proceedings of the 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 131-138, **1997**.

Paiva, A. C.; Seixas, R. B. e Gattass, M. Introdução à Visualização Volumétrica. *Monografias da Ciência da Computação - PUC-Rio*, Carlos José Pereira de Lucena (ed.), Inf MCC03/99, 145 pp., Rio de Janeiro, Brasil, **1999**.

Pfister, H. e Kaufman, A. Cube-4 – A Scalable Architecture for Real-Time Volume Rendering. *ACM/IEEE Symposium on Volume Visualization*, 47-54, **1996**.

Pfister, H. e outros. The VolumePro Real-Time Ray-Casting System. *Computer Graphics Proceedings, Annual Conference Series - SIGGRAPH'84*, Los Angeles, EUA, 251-260, **1999**.

Porter, T. e Duff, T. Compositing Digital Images. *Computer Graphics*, **18**, 3, 253-259, Julho, **1984**

Reynolds, R. A.; Gordon, D. e Chen, L. S. A Dynamic Screen Technique for Shaded Graphics Display of Slice-represented Objects. *Computer Vision, Graphics and Image Processing*, **38**, 3, 275-298, **1987**.

Sabella, P. A Rendering Algorithm for Visualizing 3D Scalar Fields, *Computer Graphics*, **22**, 4, 51-58, **1988**.

Sobierajski, L. e Kaufman, A. Volumetric Ray Tracing. *Symposium on Volume Visualization*. 11-19, **1994**.

Tecgraf. *G3D - Uma Biblioteca Gráfica para Visualização Tridimensional Baseada em OpenGL Versão 3.0*. [online] Disponível na Internet via WWW. URL: http://www.tecgraf.puc-rio.br/~paula/g3d.html, Rio de Janeiro, Brasil, **1996**.

Tost, D.; Puig, A. e Navazo, I. Visualization of Mixed Scenes Based on Volumes and Surfaces. *Proceedings of Fourth Eurographics Workshop on Rendering*, Paris, França, 281-292, **1993**.

Upson, C. & Keeler, M. V-Buffer - Visible Volume Rendering. *Computer Graphics*, **22**, 4, 59-65, **1988**.

***Visible** Human Project*. [online] Disponível na Internet via WWW. URL: www.nlm.nih.gov/research/visible/visible_human.htm, U.S. National Library of Medicine - National Institute of Health, EUA.

Walsum, van T.; Hin, A.; Versloot, J. e Post, F. H. Efficient Hybrid Rendering of Volume Data and Polygons. *Advances in Scientific Visualization*, Springer-Verlag, 83-96, **1993**.

Westermann, R. e Ertl, T. Efficiently Using Graphics Hardware in Volume Rendering Applications. *Computer Graphics Proceedings, Annual Conference Series - SIGGRAPH'98*, 169-178, **1998**.

Westover, L. Interactive Volume Rendering. *Proceedings of the Chapel Hill Workshop on Volume Visualization*, Chapel Hill, North Carolina, EUA, 9-16, **1989**.

Westover, L. Footprint Evaluation for Volume Rendering. *Computer Graphics*, **4**, 24, 144-153, **1990**.

Wilson, O.; Gelder, A. V. e Wilhelms, J. *Direct Volume Rendering via 3D Textures*. Technical Report UCSC-CRL-94-19, Baskin Center for Computer Engineering and Information Sciences, University of California, Santa Cruz, EUA, **1994**.

Wittenbrink, C. M.; Malzbender, T. e Goss, M. E. *Opacity-Weighted Color Interpolation for Volume Sampling*. HP Labs Technical Report: HPL-97-31R2, Hewlett-Packard Laboratories, **1998**.

Woo, M.; Neider, J.; Davis, T. e Shreiner, D. *OpenGL Programming Guide – The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley, 3nd Edn, **1999**.

Yagel, R. Volume Viewing Algorithms: Survey. *SIGGRAPH'98 Course Notes - Advanced Geometric Techniques for Ray Casting Volumes*, Orlando, EUA, **1998**.

Zakaria, M. N. e Saman, M. Hybrid Shear-Warp Rendering. *Proceedings of the ICVC*, Goa, Índia, **1999**.

APÊNDICE 1 – IMAGENS COLORIDAS

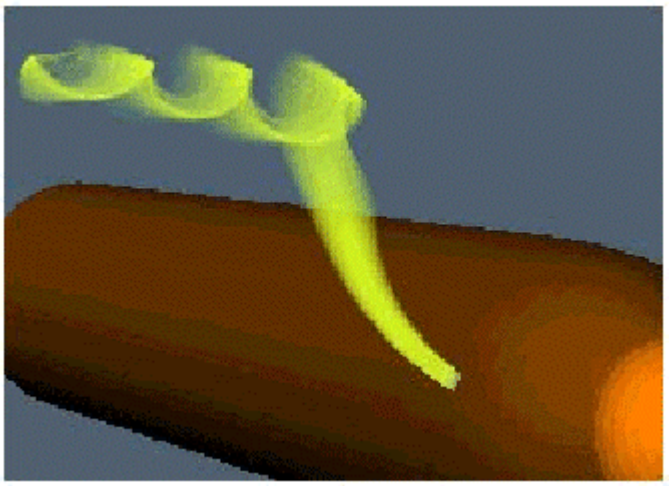
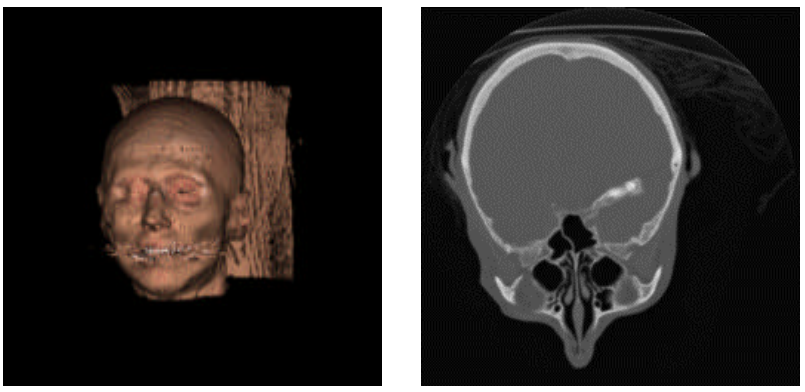


Figura 1.2 – Simulação do fluxo de ar através de dinâmica de fluidos [Paiva,1999]



(a) (b)

Figura 1.3 – (a) Reconstrução 3D a partir de (b) fatias de TC de uma cabeça

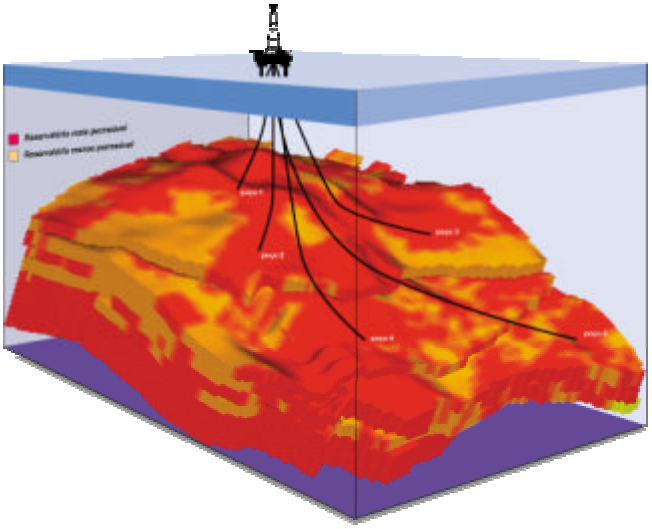


Figura 1.4 – Representação volumétrica de um dado sísmico (cedida pela Petrobrás)

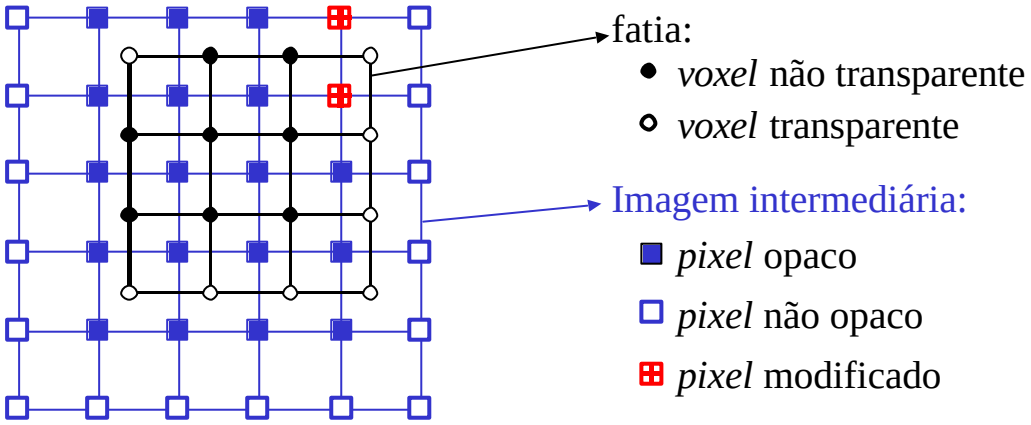


Figura 2.5 – Composição das fatias no algoritmo de *Shear-Warp*

Figura 3.2 – Volume sintético e cone poligonal utilizados para detectar *aliasing*

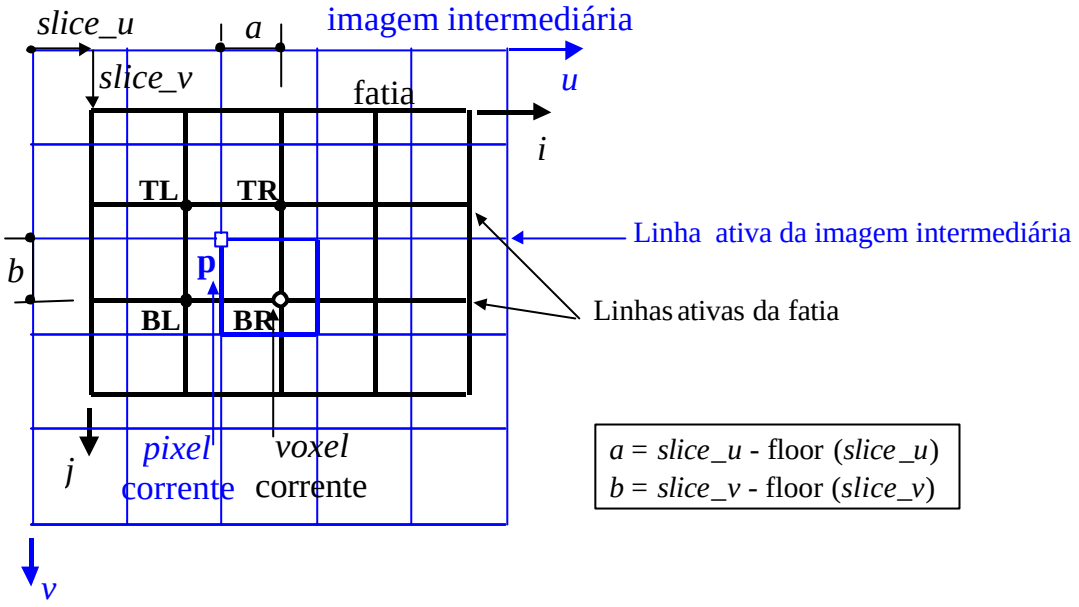


Figura 3.5 – Composição das fatias no *Shear-Warp* padrão

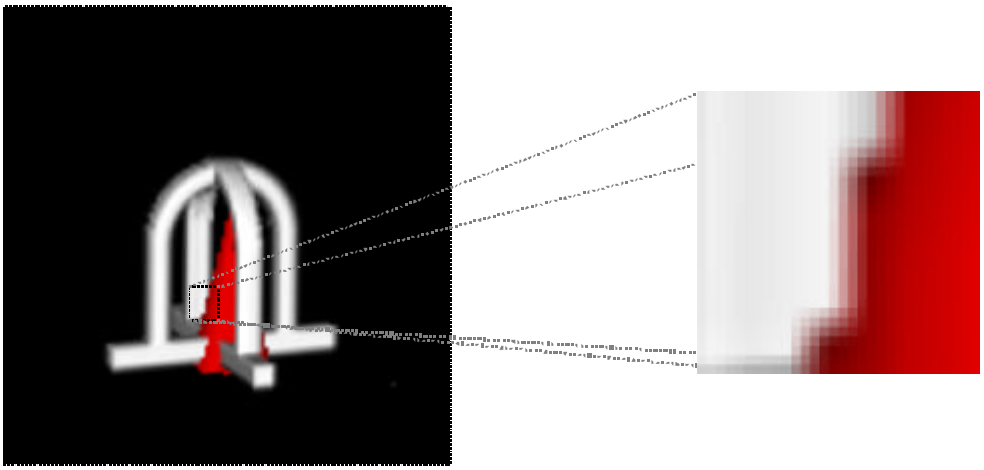


Figura 3.6 – Imagem criada com o algoritmo de baixa resolução, bordas do modelo poligonal em detalhe

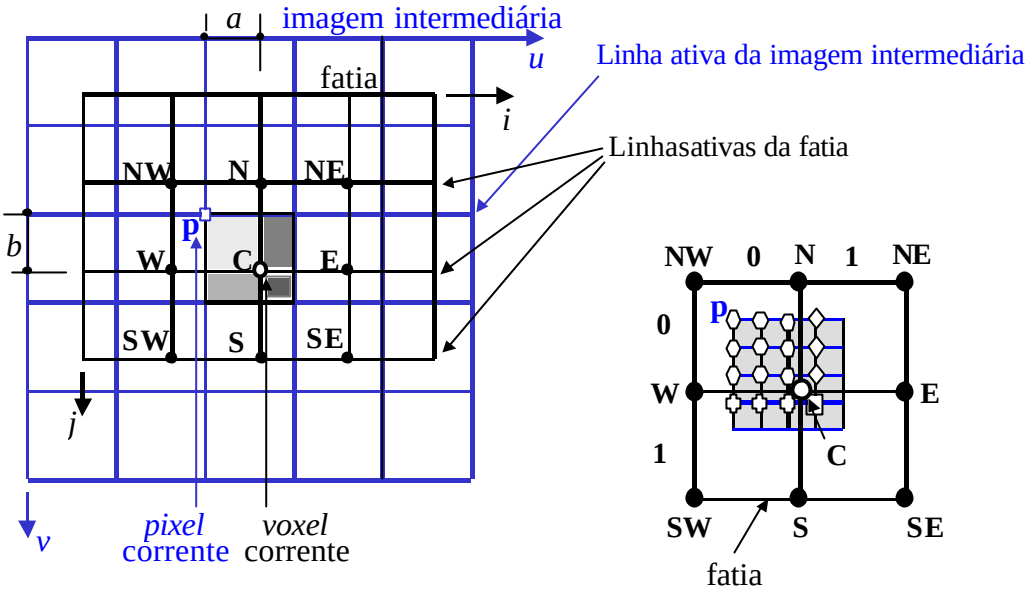


Figura 3.7 – Combinando fatias em baixa resolução em uma imagem intermediária de alta resolução

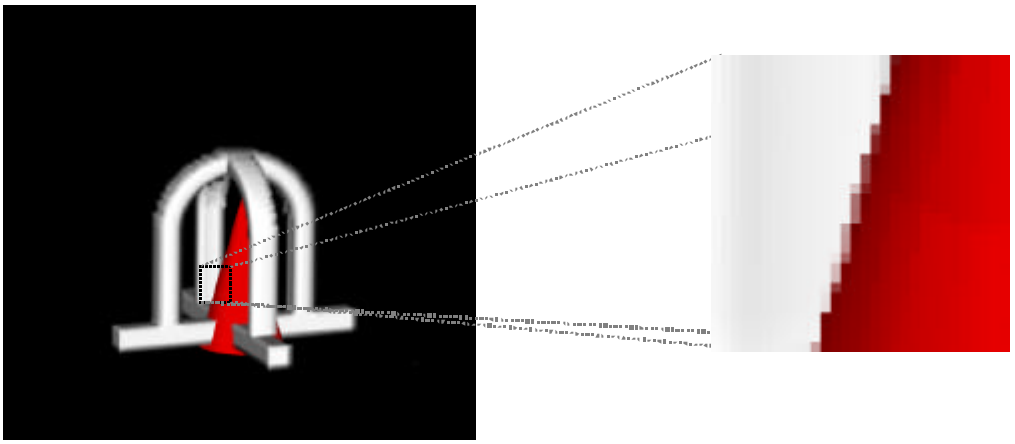


Figura 3.9 – Imagem final do modelo híbrido criada com o algoritmo de alta resolução

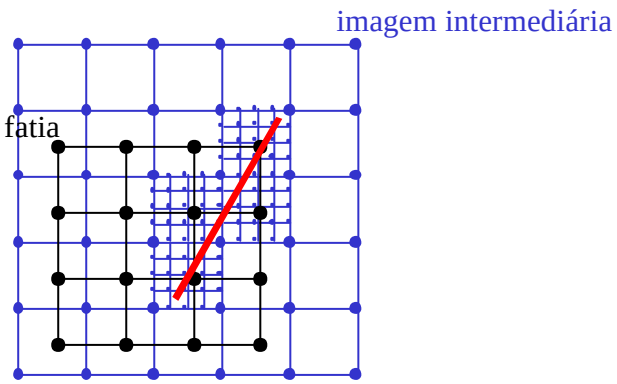


Figura 3.10 – Regiões influenciadas pelo polígono são reamostradas em alta resolução

(a) (b)

Figura 3.11 – Imagens intermediárias de (a) alta e (b) de baixa resolução criadas com *footprint*

Figura 3.12 – Imagem intermediária final criada pelo algoritmo de *footprint*

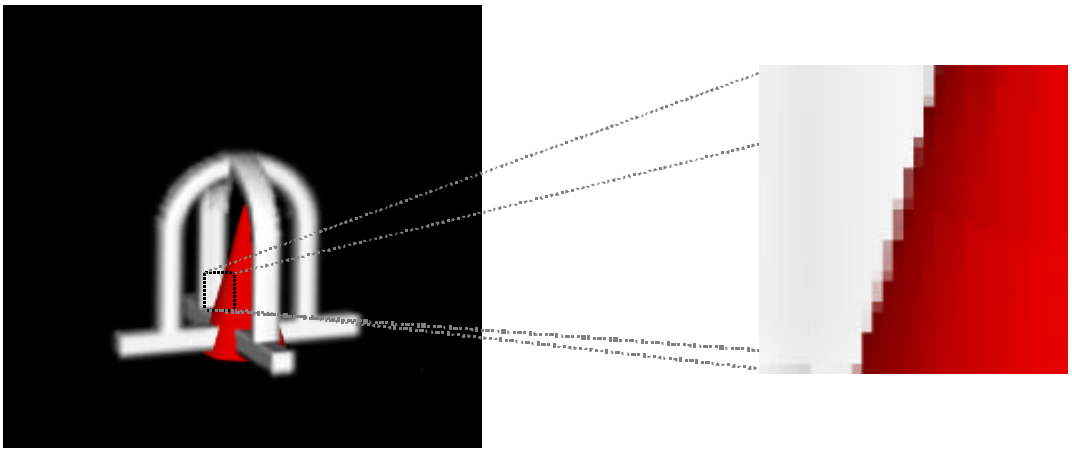


Figura 3.13 – Imagem final criada através do algoritmo de *footprint*, região em detalhe.

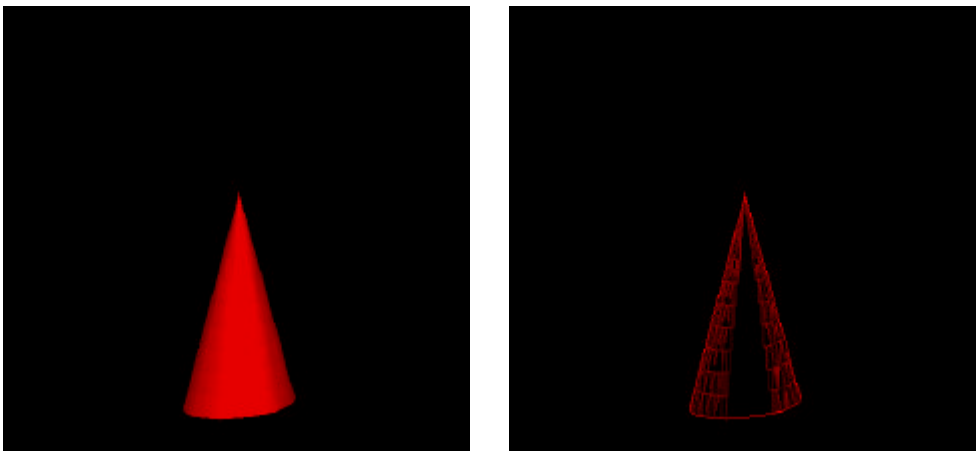


Figura 3.14 – Cone poligonal e sua imagem filtrada em alta freqüência correspondente

Figura 3.15 – Imagens intermediárias em (a) alta resolução;
(b) baixa resolução somente com contribuição do volume;
(c) baixa resolução com contribuições do volume e dos polígonos

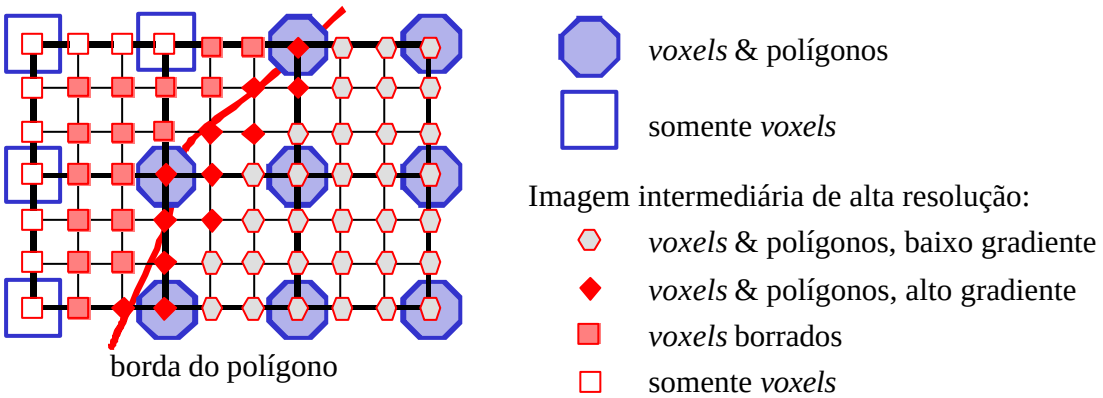
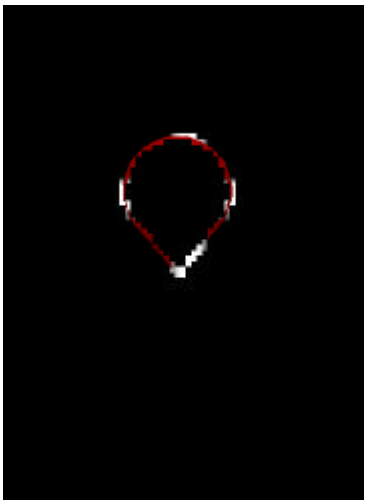
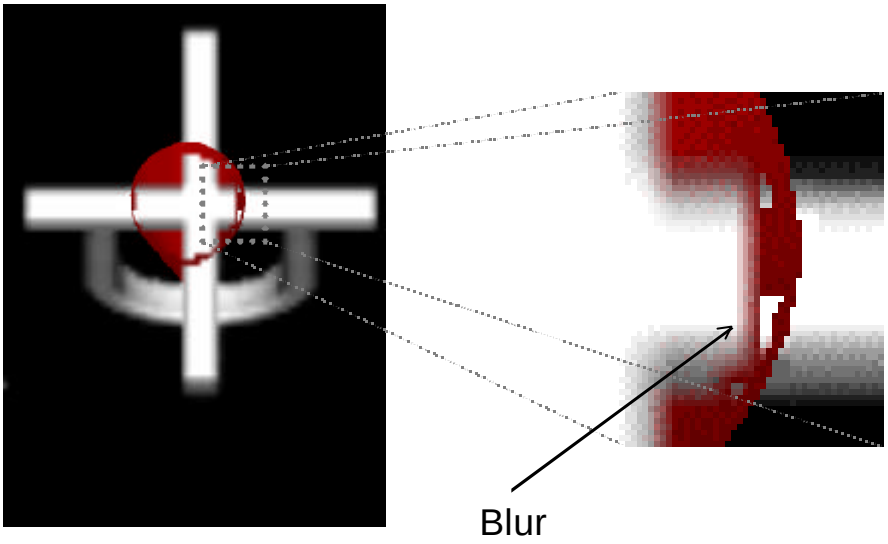


Figura 3.16 – *Blur* produzido nas bordas do polígono usando a imagem em baixa resolução do volume+polígono



(a)

(b)



(c)

Figura 3.17 – (a) Imagem intermediária composta em alta resolução. (b) Imagem intermediária em baixa resolução do volume+poligono. (c) Imagem intermediária final preenchida apresentando *blur*.

Figura 3.18 – Imagem intermediária final em alta resolução criada pelo algoritmo de alta freqüência

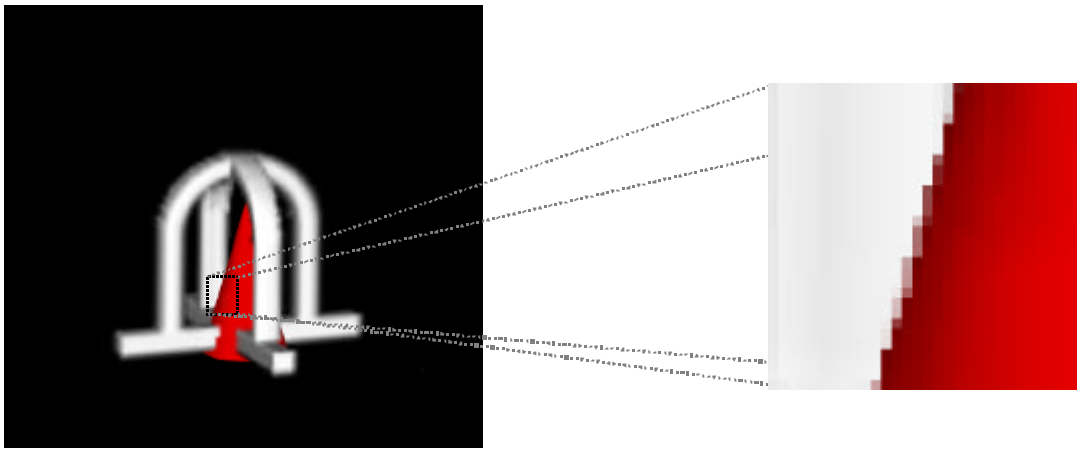


Figura 3.19 – Imagem final do exemplo híbrido criada pelo algoritmo de alta freqüência

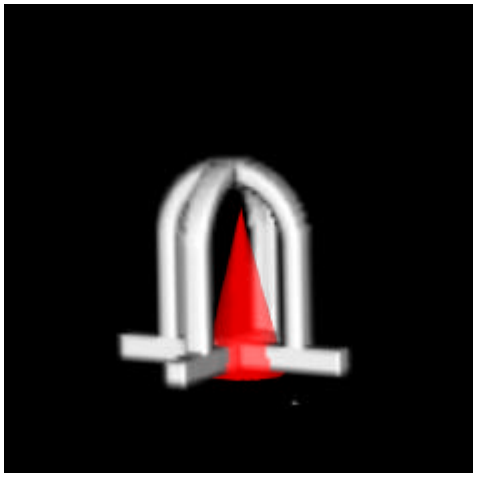


Figura 3.20 – Modelo sintético opaco combinado com o cone poligonal com um nível de transparência

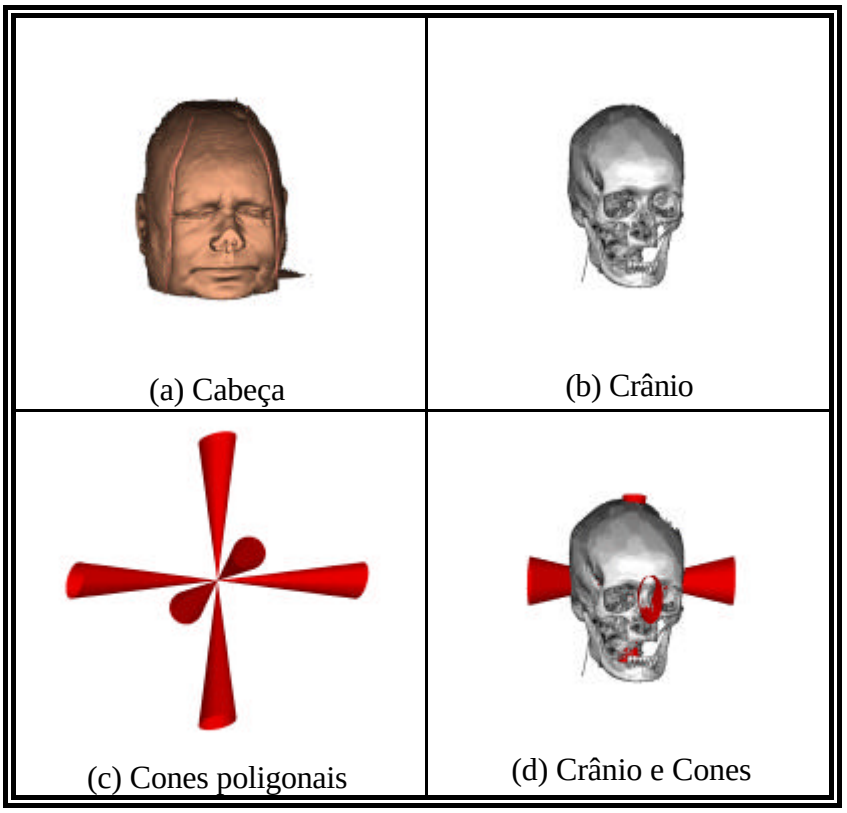
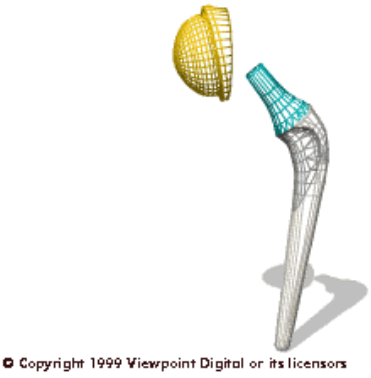
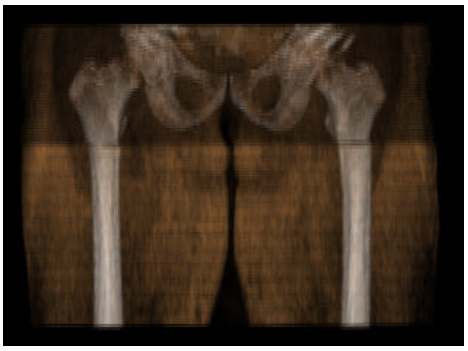


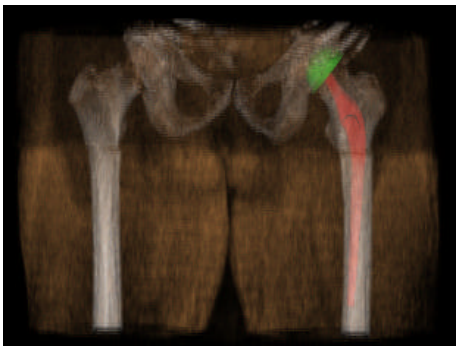
Figura 4.1 – Cabeça da *Visible Woman* e modelo poligonal de cones



(a)

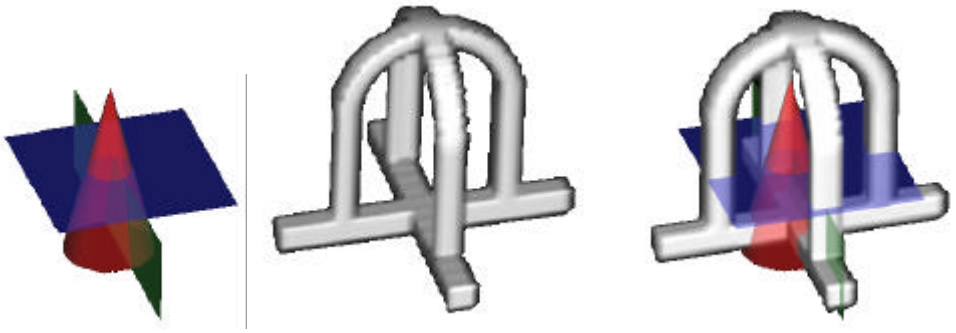


(b)



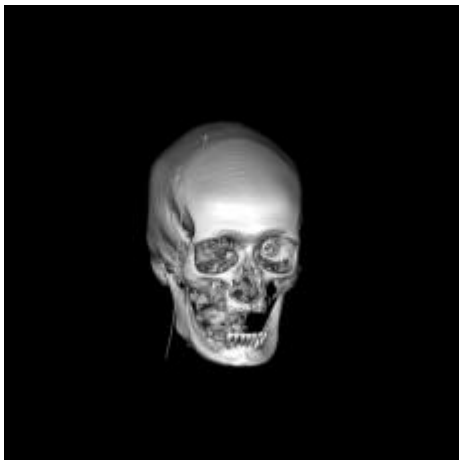
(c)

Figura 4.2 – (a) Modelo poligonal da prótese; (b) dado volumétrico da Pélvis e (c) dados combinados

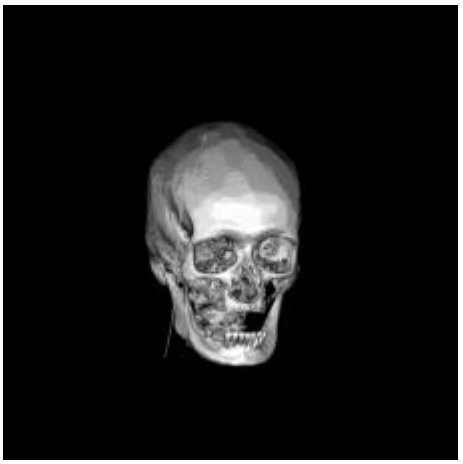


(a) (b) (c)

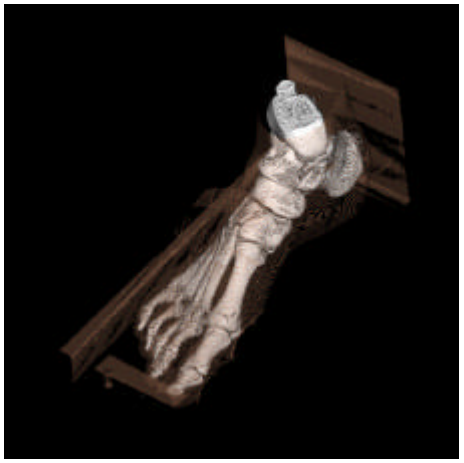
Figura 4.3 – (a) Modelo poligonal, (b) volume *Syn64* e (c) combinação



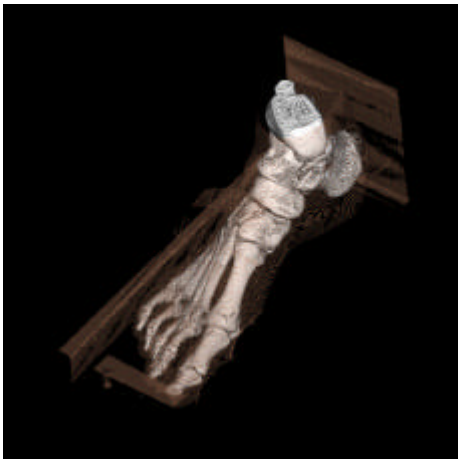
(a)



(b)



(c)



(d)

(e) (f)

Figura 4.6 – (a) Crânio renderizado com gradiente originais e (b) com quantização.
(c) Pé renderizado com transparência e gradientes originais e (d) com quantização.
(e) Motor renderizado com gradientes originais e (f) com quantização.

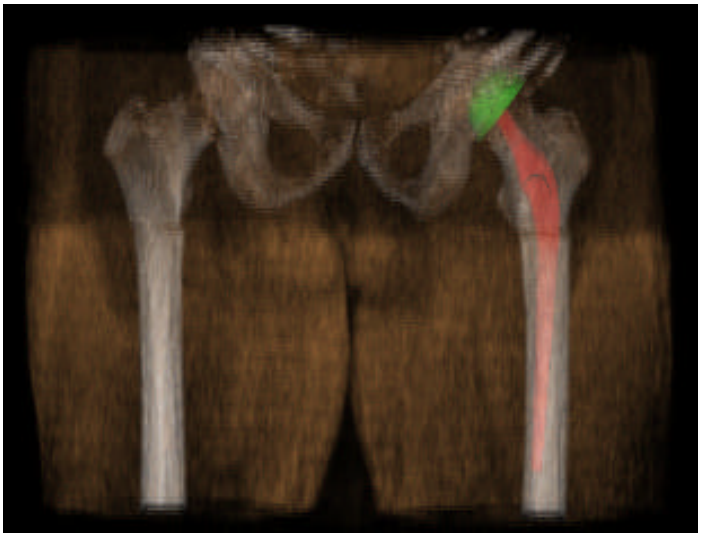
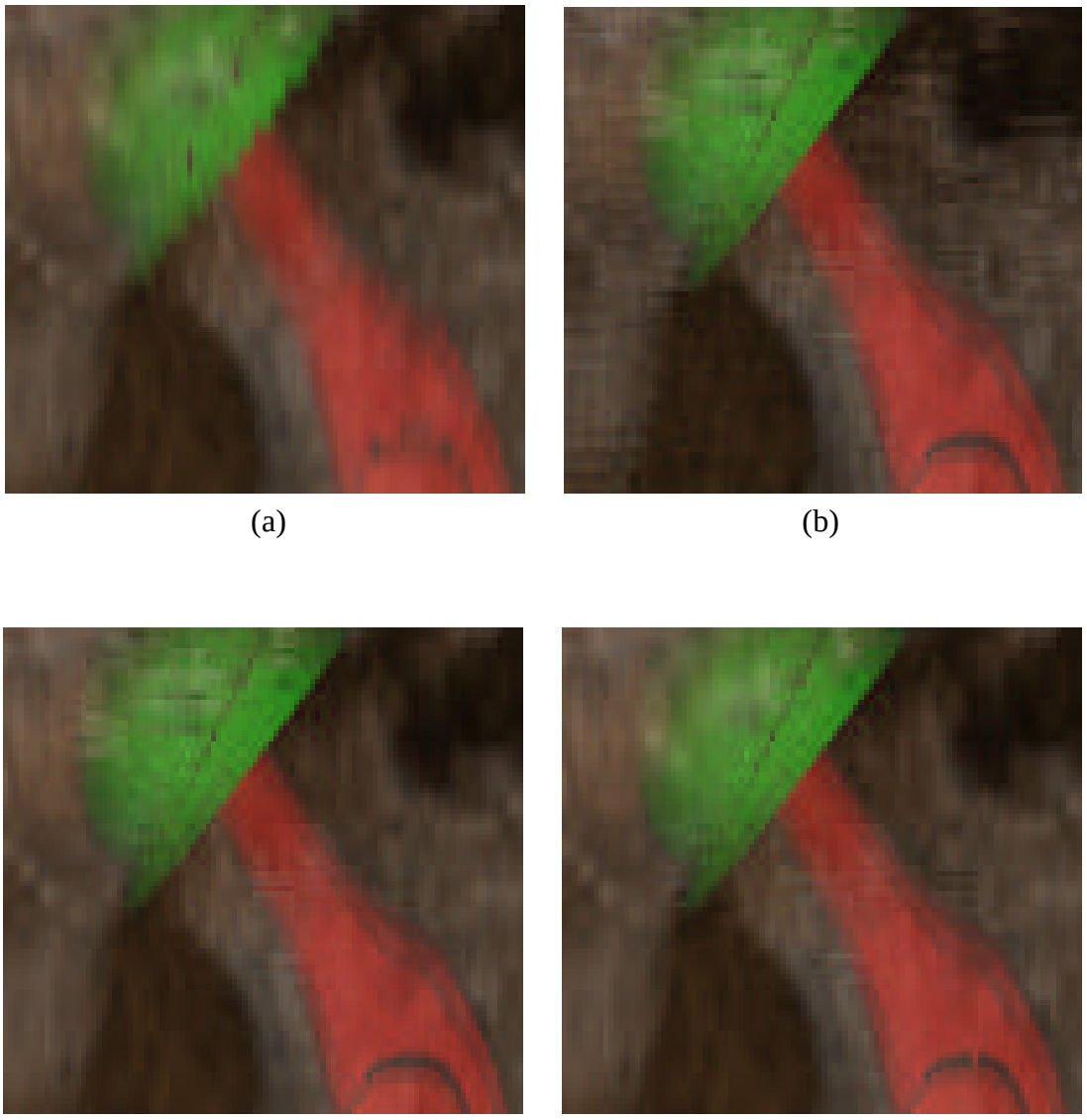


Figura 4.7 – Volume da Pélvis combinada com o modelo de prótese, com 1 nível de transparência



(a) (b)

(c) (d)

Figura 4.8 – Região ampliada das imagens criadas pela composição em (a) baixa resolução; (b) alta resolução; (c) *footprint* e (d) alta freqüência

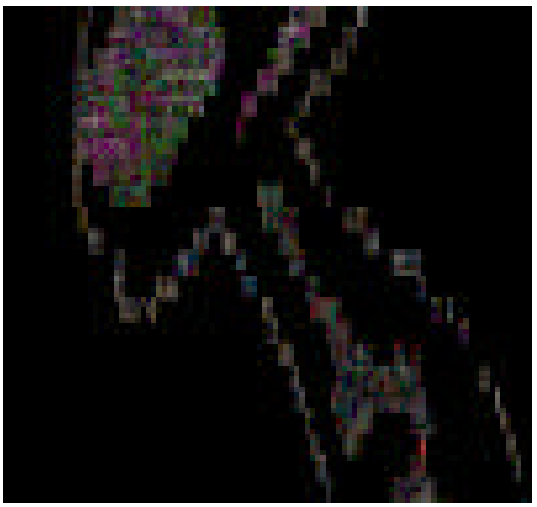
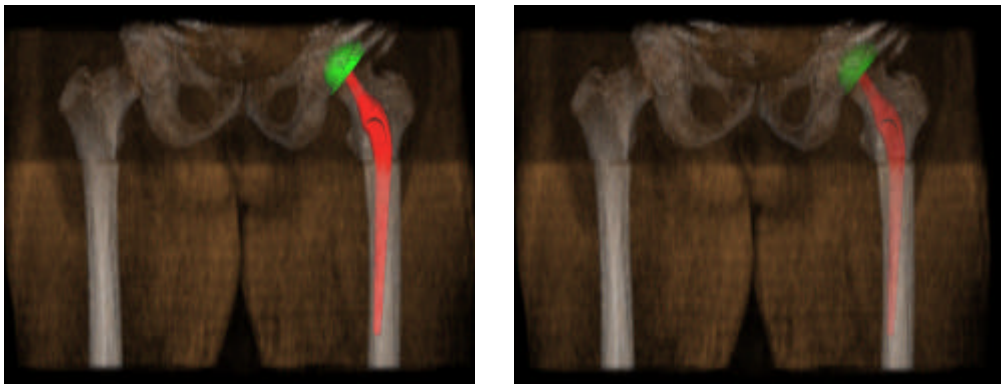
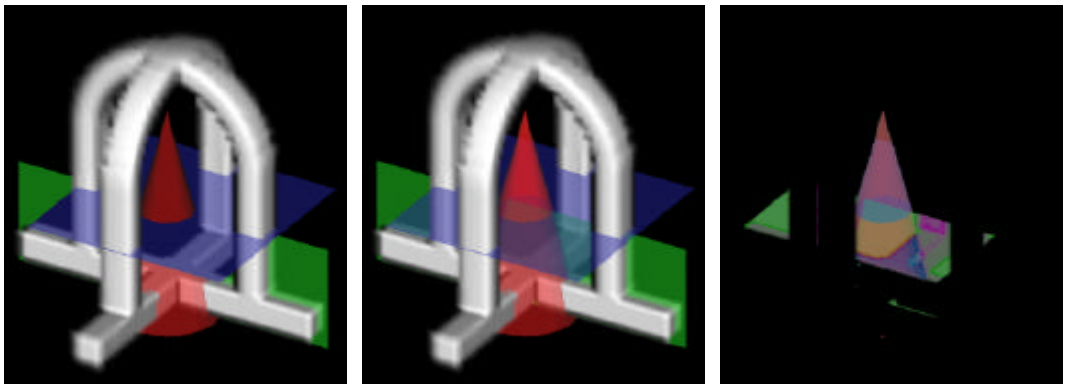


Figura 4.9 – Diferença entre as imagens do *footprint* e alta freqüência (*gamma-2.5)*



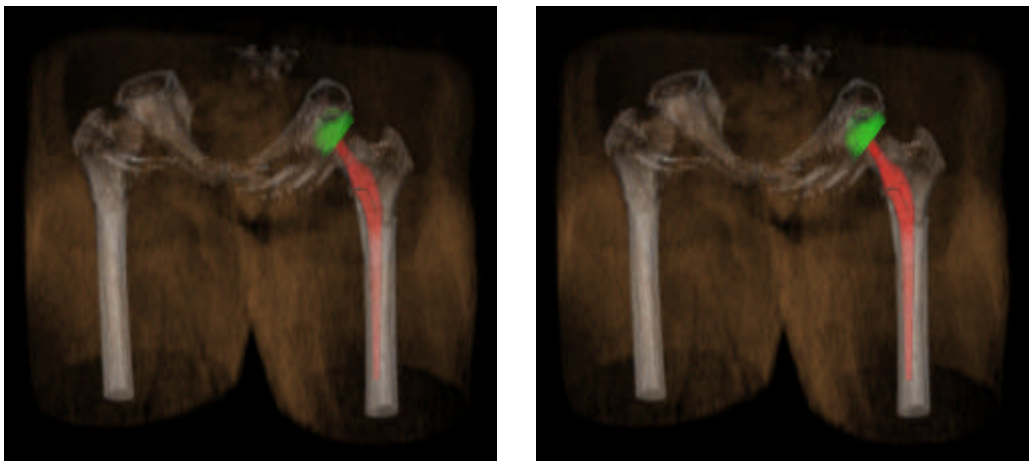
(a) (b)

Figura 4.10 – Imagens da Pélvis semitransparente combinada com a prótese poligonal. (a) Prótese é opaca. (b) Prótese com 1 nível de transparência

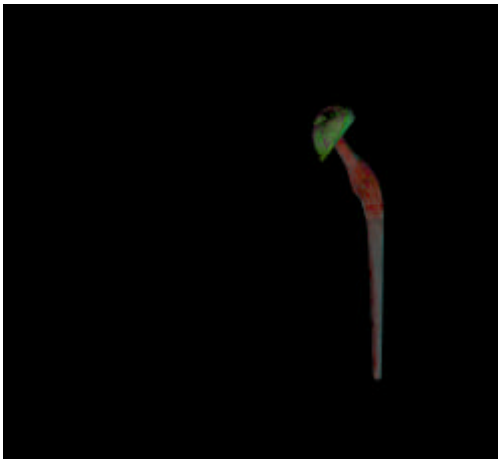


(a) (b) (c)

Figura 4.11 – Dado *Syn64* e modelo poligonal com (a) 1 nível de transparência; (b) com 3 níveis de transparência e (c) diferença entre (a) e (b), com fator *gamma*=2.5



(a) (b)



(c)

Figura 4.12 – Pélvis combinada com a prótese com (a) 1 nível de transparência e (b) 3 níveis de transparência. Em (c) tem-se a imagem da diferença entre as duas imagens anteriores.