Métodos de Aceleração de Rendering de Volumes

Marcelo Medeiros Carneiro
PUC-Rio/Departamento de Informática
mmc@inf.puc-rio.br

Marcelo Gattass
TeCGraf – Grupo de Tecnologia em Computação Gráfica/PUC-Rio
gattass@tecgraf.puc-rio.br

PUC-RioInf.MCC16/00 Março/2000

ABSTRACT: Volume Rendering is a method of creating bidimensional images from a data set in the tridimensional space. However, when the 3D data set is too large, this method has several limitations, especially requiring too much computation time to generate good quality images. This work presents some volume rendering acceleration techniques in order to allow interactive volume visualization.

KEYWORDS: Volume Rendering, Interactive Volume Rendering, Scientific Visualization, Rendering Systems.

RESUMO: Visualização Volumétrica é um método utilizado para criar uma imagem bidimensional a partir um conjunto de dados no espaço tridimensional. Entretanto, quando o conjunto de dados é muito grande, esse método possui várias limitações, principalmente com relação ao tempo necessário para a geração de imagens de boa qualidade. Este trabalho estuda algumas técnicas de aceleração do processo de *rendering* de volumes, permitindo a visualização de dados volumétricos com tempos interativos.

PALAVRAS-CHAVE: Visualização Volumétrica, Visualização Volumétrica Interativa, Visualização Científica, Sistemas de *Rendering.*

Sumário

1 INTRODUÇÃO 4

2 ALGORITMOS DE RENDERING DE VOLUMES 6

2.1 Ray Casting 8

2.2 Shear-Warp 12

2.3 Splatting 13

3 OTIMIZAÇÕES DE ALGORITMOS 15

3.1 Ray Casting 15

3.1.1 Término Adaptativo de Raios 15

3.1.2 Refinamento Adaptativo da Imagem 16

3.1.3 Templates 18

3.1.4 Decomposição Hierárquica do Espaço 19

3.1.5 Polygon Assisted Ray Casting (PARC) 21

3.2 Shear-Warp 22

3.3 Splatting 24

4 HARDWARES ESPECIAIS 27

4.1 VOGUE 29

4.2 VIRIM 30

4.3 Cube-4 30

4.4 VolumePro 30

5 CONCLUSÕES 32

6 REFERÊNCIAS 33

Lista de Figuras

Figura 2.1: Pipeline de visualização volumétrica. 7
Figura 2.2: Algoritmo *Ray Casting* 9
Figura 2.3: Cálculo da iluminação em um *voxel* 9
Figura 2.4: Composição de cor e opacidade 11
Figura 2.5: Algoritmo *Shear-Warp* 12
Figura 3.1: Refinamento progressivo [Paiva e outros, 99]. 17
Figura 3.2: Refinamento adaptativo da imagem [Levoy, 90d] 18
Figura 3.3: *Template-based volume viewing* [Kaufman e Sobierajski, 94]. 19
Figura 3.4: Construção de uma *octree* [Levoy, 90d]. 20
Figura 3.5: Caminhamento em uma *octree* 2D [Levoy, 90b]. 21
Figura 3.6: *Polygon Assisted Ray Casting* [Kaufman e Sobierajski, 94]. 22
Figura 3.7: Alinhamento das *scanlines* do algoritmo *Shear-Warp* [Lacroute, 95] 23
Figura 3.8: Composição utilizando as estruturas RLE [Lacroute, 95]. 24

1 INTRODUÇÃO

A Visualização Volumétrica é um campo da computação gráfica em crescente expansão. Pode-se citar muitas aplicações de técnicas de visualização volumétrica, por exemplo, na medicina (imagens médicas). Nesse campo, o dado volumétrico é usualmente obtido através de algum processo de aquisição, seja por tomografia computadorizada (CT), ressonância magnética (MRI), microscopia confocal, entre outros [Paiva e outros, 99].

Além de larga aplicação na medicina, técnicas de visualização volumétrica também são muito empregadas em outras áreas da ciência, por exemplo, meteorologia, geologia, engenharia mecânica etc. Entre essas áreas, destaca-se a utililização de visualização volumétrica na indentificação de estruturas geológicas sob a superfície do solo. Isso permite, através da interpretação de dados sísmicos, inferir propriedades do terreno, auxiliando o geólogo na identificação de bacias de petróleo [Gerhardt e outros, 98].

Um de seus principais objetivos é permitir que o usuário extraia informações relevantes dos dados. Muitos algoritmos foram desenvolvidos e alguns aperfeiçoados visando melhorar a qualidade das imagens ou diminuir seu tempo de geração, permitindo o desenvolvimento de sistemas interativos.

Entretando, apesar do crescente uso de técnicas de visualização volumétrica, ainda há uma grande dificuldade para sua efetiva popularização, tendo em vista seu alto custo computacional, a crescente demanda por visualização exploratória e o aumento do tamanho do dado volumétrico. Um exame médico composto por 1024 fatias de dimensão 1024x1024, utilizando 16 bits para representar o dado volumétrico ocupa 2Gb de memória, o que é relativamente grande para os recursos computacionais atuais.

Por isso, torna-se necessário estudar algumas técnicas que permitam contornar este sério problema. Este trabalho tem o objetivo principal de estudar vários métodos de aceleração do processo de visualização volumétrica. Isso pode ser obtido de duas formas diferentes.

A primeira forma utiliza estruturas de dados eficientes que permitem identificar que partes do volume devem ser realmente processadas e que partes podem ser ignoradas

por não contruibuirem o suficiente para influenciar na imagem final do volume. Basicamente, essas estruturas permitem explorar a coerência espacial do dado volumétrico ou da imagem do volume. Com isso, pode-se obter um ganho expressivo de velocidade nos algoritmos.

A segunda forma de acelerar o processo de visualização volumétrica é procurar implementar em *hardware* os principais gargalos apresentados pelos algoritmos de visualização, em especial o problema de acesso à memória, classificação do dado, iluminação e composição dos *voxels*. Assim, algumas arquiteturas especiais surgiram com a intenção de oferecer visualização de volumes em tempo real.

Este trabalho procura, então, abordar as duas formas de aceleração de *rendering* de volume. Primeiro, os principais conceitos de visualização volumétrica são apresentados, principalmente o *pipeline* de visualização. A seguir, os principais algoritmos de *rendering* direto de volume são discutidos, em especial *Ray Casting*, *Shear-Warp* e *Splatting*. Vários métodos de otimização dos algoritmos também são analisados. Finalmente, os conceitos mais importantes de arquiteturas para visualização volumétrica são apresentados juntamente com algumas soluções obtidas nos últimos anos, em especial *Cube-4* e *VolumePro*.

2 ALGORITMOS DE RENDERING DE VOLUMES

Este capítulo tem o objetivo de apresentar os algoritmos básicos de *rendering* direto. Esses algoritmos têm em comum a não utilização de nenhuma representação intermediária para gerar a imagem do volume. Porém, têm a desvantagem de serem, em geral, computacionalmente mais caros.

Como exemplo de algoritmos de *rendering* direto, pode-se citar *Ray Casting* [Levoy, 88], *Splatting* [Westover, 90] e *Shear-Warp* [Lacroute, 95], entre outros. Esses algoritmos realizam a projeção do volume diretamente no plano da imagem e são os mais indicados para visualizar volumes que representam objetos amorfos.

Pode-se ainda subdividir os algoritmos de *rendering* direto em duas categorias principais: espaço dos objetos e espaço da imagem. Na primeira situação (espaço dos objetos), é utilizado o mapeamento direto (*forward mapping*) do volume sobre o plano da imagem, ou seja, os *voxels* são projetados na tela em uma determinada ordem. Como exemplo, pode-se citar os algoritmos *Splatting* [Westover, 90] e *V-Buffer* [Upson e Keeler, 88]. No outro caso (espaço da imagem), é utilizado o mapeamento reverso (*backward mapping*), onde são lançados raios para cada pixel da imagem em direção ao volume determinando, assim, a cor final do *pixel*. Como exemplo, pode-se citar o algoritmo *Ray Casting* [Levoy, 88].

Os algoritmos de visualização volumétrica requerem, em sua maioria, os passos apresentados na Figura 2.1. O primeiro passo consiste na aquisição do dado volumétrico, que normalmente é realizado através do uso de um equipamento de sensoriamento apropriado. A seguir, o dado volumétrico bruto é classificado, normalmente com o auxílio do usuário. Essa etapa pode ser vista como uma forma de segmentação, pois sua finalidade principal é identificar estruturas internas do volume, através do uso adequado de cores e opacidades. A seguir, o volume classificado é iluminado, facilitando sua interpretação pelo usuário, pois a forma tridimensional do volume é realçada. O último passo consiste na projeção do volume classificado e iluminado em um plano de visualização, gerando assim uma imagem bidimensional que será visualizada pelo usuário.

Essas etapas são muito comuns na grande maioria dos algoritmos de visualização. Nessa análise, está sendo considerado que o dado volumétrico é apresentado sob a forma de valores escalares em um reticulado retilínio. Esta etapa já foi previamente realizada através de alguma simulação numérica ou através de algum processo de aquisição.

Tipicamente, esses dados podem ser pré-processados, visando reduzir o ruído, realçar algumas características do objeto em estudo. Em algumas situações, os dados precisam ser reconstruídos, visando estimar valores perdidos durante o processo de aquisição, reamostrar o volume, através de interpolação, para adequar sua topologia (convertendo um reticulado irregular para um reticulado regular, por exemplo), novas fatias podem ser criadas e outras podem ser descartadas etc.

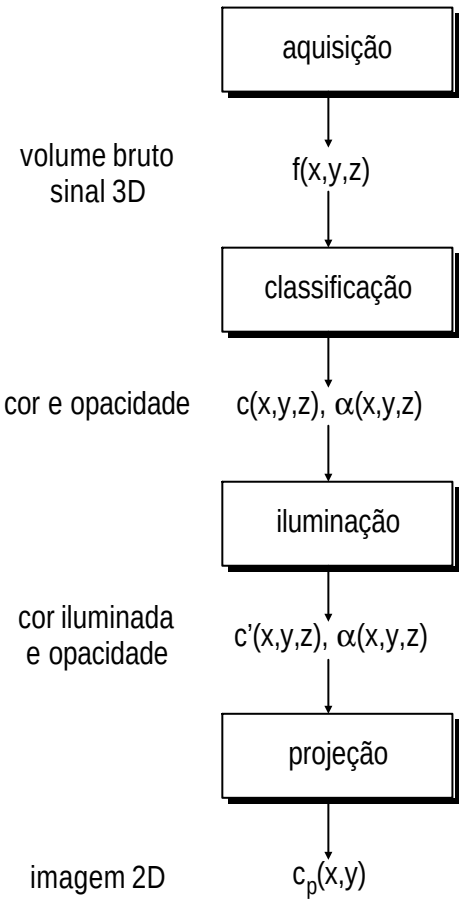

Figura 2.1: Pipeline de visualização volumétrica.

Pressupondo que o dado volumétrico já esteja armazenado em uma estrutura adequada, o processo de visualização volumétrica consiste em, a partir de uma função escalar discreta $f(x_i, y_j, z_k)$ que define o volume, gerar uma imagem bidimensional

$p(x_i, y_j)$ que represente a projeção desse volume em um determinado plano, de acordo com os parâmetros de visualização.

O interessante ao estudar o *pipeline* de visualização é perceber que em cada etapa uma novidade é introduzida. Após a aquisição dos dados, tem-se o volume em seu estado bruto. Ao separar as estruturas que serão visualizadas, através da definição das funções de transferências, tem-se o dado classificado. Após o posicionamento das luzes, tem-se o dado iluminado. Finalmente, após o posicionamento da câmera, tem-se o dado projetado em uma imagem bidimensional. Fica claro, então, que qualquer alteração nesse processo é necessário revisitar as etapas seguintes. Por exemplo, se a posição da câmera for alterada, apenas a etapa de projeção precisa ser executada. Entretanto, se o usuário mudar uma das funções de transferência, as etapas de classificação, iluminação e projeção precisam ser novamente executadas.

A seguir serão apresentadas as principais características dos algoritmos mais importantes de *rendering* direto: o traçado de raios (*Ray Casting*), que opera no espaço da imagem, o *Splatting*, que opera no espaço do objeto e o algoritmo *Shear-Warp* que mescla as duas técnicas.

2.1 Ray Casting

O algoritmo *Ray Casting* [Levoy, 88] permite a geração de imagens de alta qualidade através de *rendering* direto, operando no espaço da imagem. A idéia básica do algoritmo é lançar raios na direção de visão atravessando cada *pixel* da tela (plano de visualização ou plano de projeção), conforme mostrado na Figura 2.2.

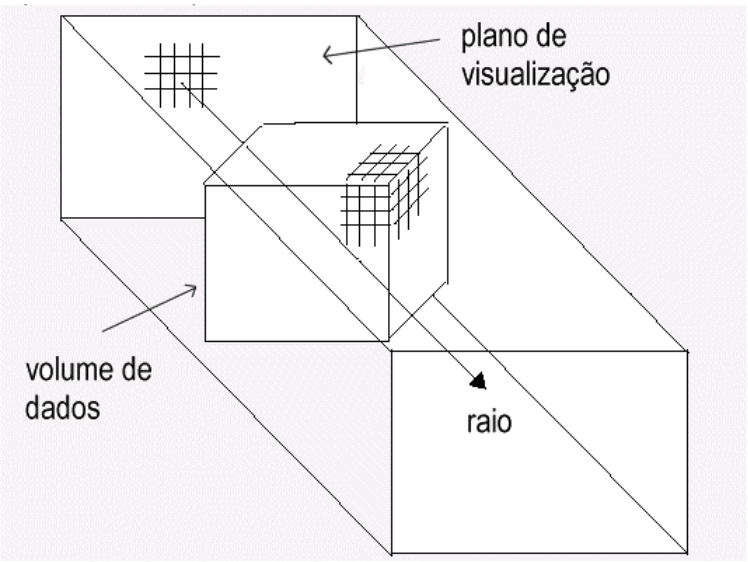

Figura 2.2: Algoritmo *Ray Casting*

Durante a etapa de iluminação do *pipeline* de visualização volumétrica, a cor de cada *voxel* é modificada utilizando algum modelo de iluminação local, tipicamente *Phong* [Phong, 75]. Para isto, o vetor normal em cada *voxel* é calculado através de uma estimativa local do gradiente da função volumétrica, tipicamente utilizando diferenças finitas. Além disto, algumas luzes podem ser posicionadas na cena. A Figura 2.3 ilustra esse processo.

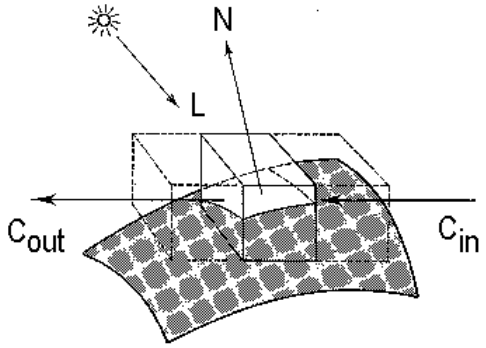

Figura 2.3: Cálculo da iluminação em um *voxel*

Pela figura acima, a luz C_{out} que sai de um *voxel* é calculada em função de três fatores fundamentais:

- luz refletida na direção de visão por todas as fontes de luz presentes na cena;
- luz C_{in} filtrada pelo *voxel*;
- qualquer luz emitida pelo *voxel*.

Tipicamente, a luz emitida pelas fontes de luz presentes na cena não é atenuada pelo volume. Esse efeito é usualmente ignorado durante o cálculo da iluminação pois, além de simplificar bastante o algoritmo (não é necessário calcular uma integral de linha), muitas vezes é até indesejado. Se as fontes de luz fossem atenuadas pelo volume, áreas ao redor de regiões muito densas (por exemplo, osso) poderiam ficar totalmente obscurecidas. Note, entretanto, que o efeito da atenuação da luz ao longo do raio de visão não pode ser desprezado. Isso é levado em consideração durante a etapa de projeção do *pipeline* de visualização.

Para cada raio lançado no volume (Figura 2.1), é realizada uma amostragem do volume em intervalos regulares ao longo do raio. Através de interpolação, é obtida a cor e a opacidade em cada ponto amostrado, segundo as funções de transferências empregadas durante o processo de classificação. A seguir, todas as contribuições obtidas (cores e opacidades) são compostas calculando assim a cor final $C(R)$ do *pixel* (Figura 2.4).

A cor $C_{out}(R,k)$ e a opacidade $\alpha_{out}(R,k)$ do raio após passar pela amostra k ao longo do raio é calculada em função da cor $C_{in}(R,k)$ e a opacidade $\alpha_{in}(R,k)$ antes de passar pelo *voxel* e pela cor $C(R,k)$ e opacidade $\alpha(R,k)$ da amostra:

$$C'_{out}(R,k) = C'_{in}(R,k) + C'(R,k)*(1-\alpha_{in}(R,k))$$

$$\alpha_{out}(R,k) = \alpha_{in}(R,k) + \alpha(R,k)*(1-\alpha_{in}(R,k))$$

Sendo que,

$$C'_{in}(R,k) = C_{in}(R,k)*\alpha_{in}(R,k)$$

$$C'_{out}(R,k) = C_{out}(R,k)*\alpha_{out}(R,k)$$

Depois de computar todas as N contribuições de $k=1..N$, a cor final $C(R)$ do *pixel* é dada por:

$$C(R) = C'_{out}(R,N)/ \alpha_{out}(R,N)$$

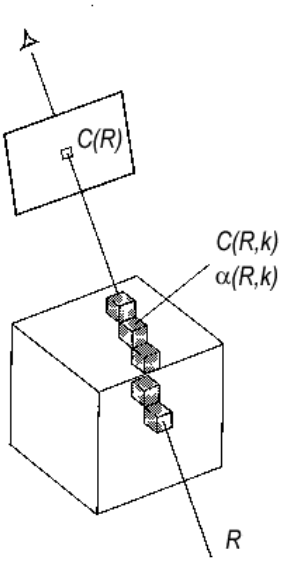

Figura 2.4: Composição de cor e opacidade

Essencialmente, esse cálculo pode ser expresso através de:

$$
\begin{aligned}
C(R) = {} & C(R,1) + \\
& C(R,2)*(1-\alpha(R,1)) + \\
& C(R,3)*(1-\alpha(R,1))*(1-\alpha(R,2)) + \\
& \ldots + \\
& C(R,N)*(1-\alpha(R,1))\ldots(1-\alpha(R,N-1)) \\
= {} & \sum_{k=1}^{N} \left(C(R,k) * \prod_{i=1}^{k-1} (1-\alpha(R,i)) \right)
\end{aligned}
$$

Pode-se também utilizar o operador *over* de composição digital [Porter e Duff, 84] para expressar a cor final do *pixel*:

$$C(R) = C(R,1) \textit{ over } C(R,2) \textit{ over } C(R,3) \ldots \textit{ over } C(R,N)$$

O algoritmo *Ray Casting* gera imagens de alta qualidade, entretanto seu custo computacional pode ser muito grande, dependendo da resolução final da imagem, i.é., da quantidade de raios a serem lançados pelo volume. Uma outra desvantagem é que os raios lançados tipicamente atravessam o volume fora da ordem de armazenamento dos

voxels, sendo necessário, portanto, calcular em que *voxel* está cada ponto amostrado ao longo do raio.

2.2 Shear-Warp

A idéia principal do algoritmo *Shear-Warp* [Lacroute, 95] é transformar o volume de dados de modo a simplificar a etapa de projeção do *pipeline* de visualização. Isto é obtido através da aplicação de um cisalhamento nas fatias do volume, transformando os dados para um sistema de coordenadas intermediário cuja principal caracterísica é que os raios de visão são perpendiculares as fatias do volume (Figura 2.5). Isso facilita bastante a projeção das fatias, pois os dados volumétricos são acessados na ordem de armazenamento. Note que isso não acontece no algoritmo *Ray Casting*. O cisalhamento é uma transformação geométrica afim que simplesmente translada as fatias do volume, não sendo, portanto, computacionalmente cara.

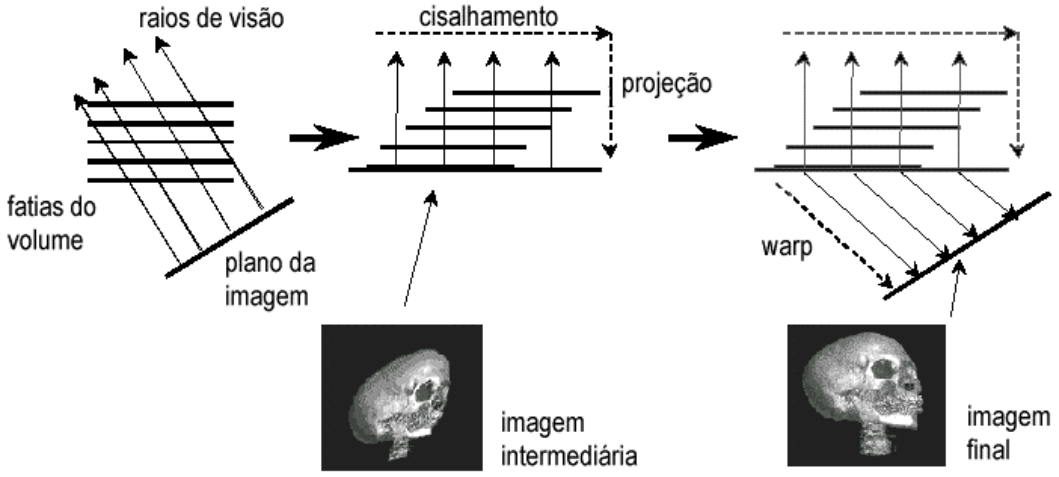

Figura 2.5: Algoritmo *Shear-Warp*

A composição e projeção das fatias nesse novo sistema de coordenadas (realizada através do operador *over*) gera uma imagem intermediária distorcida, que precisa ser corrigida posteriormente (através da transformação de *warp*). A principal vantagem desse processo é que as dimensões da imagem intermediária dependem apenas do tamanho do volume de dados e do fator de cisalhamento, não dependendo, portanto, da resolução final da imagem. Como as dimensões do volume de dados são tipicamente menores que a

resolução final da imagem, o algoritmo *Shear-Warp* leva uma grande vantagem em relação ao algoritmo *Ray Casting*.

A principal razão para utilizar essa transformação no volume é que as *scanlines* de *voxels* em cada fatia ficam paralelas as *scanlines* de *pixels* na imagem intermediária, facilitando bastante a projeção das fatias. Após a geração da imagem intermediária distorcida, esta é transformada na imagem final através da aplicação do *warp*. Esta transformação é realizada em 2D e restaura as dimensões verdadeiras da imagem que será visualizada pelo usuário. De uma forma geral, o ganho de velocidade do algoritmo *Shear-Warp* pode chegar a ser de 5 a 10 vezes em relação ao algoritmo *Ray Casting*.

2.3 Splatting

O algoritmo *Splatting* [Westover, 90] trabalha no espaço dos objetos e procura projetar cada *voxel* do volume no plano da imagem. Essa projeção normalmente é realizada a partir dos *voxels* mais próximos do observador até o mais distante. A idéia central é "arremessar" cada *voxel* sobre o plano de imagem, o que deixa uma "marca" característica do *voxel*, daí o nome do algoritmo.

O primeiro passo do algoritmo é determinar em que ordem o volume será percorrido. Isto é essencial para o correto cálculo da visibilidade pois, assim como no algoritmo *Ray Casting*, a ordem correta de projeção permite acumular adequadamente as opacidades dos *voxels*. Para isso, escolhe-se os dois eixos do volume mais paralelos ao plano da imagem para formar o *loop* mais interno. Com isso, o plano formado por esses dois eixos serão rapidamente projetados e o usuário poderá perceber mais rapidamente a formação da imagem final. A projeção é realizada fatia por fatia, ou seja, todos os *voxels* de uma determinada fatia são projetados antes que a próxima fatia seja processada. Como ocorre nos demais algoritmos, o valor de densidade de cada *voxel* é classificado de acordo com as funções de transferências de cor e opacidade e, a seguir, é iluminado utilizando a técnica de estimativa da normal através do gradiente.

A seguir vem a parte mais importante do algoritmo, onde é calculada a contribuição de cada *voxel* no plano da imagem. O algoritmo procura reconstruir um sinal contínuo a partir de um conjunto discreto de dados. Para isto, utiliza-se um filtro de reconstrução (*kernel*) para calcular a extensão da projeção do *voxel* sobre o plano da imagem. A projeção do *kernel* é chamada de *footprint* e, no caso de projeções

ortográficas, o *footprint* é o mesmo para todos os *voxels*. Isto significa que ele pode ser gerado em uma etapa de pré-processamento. A extensão do *footprint* está relacionada com o tamanho do volume e a resolução da imagem, ou seja, se a resolução da imagem for maior que o tamanho do volume, a projeção de um único *voxel* pode ocupar vários *pixels*.

A cor e opacidade de cada *voxel* é composto com os *pixels* já presentes no *buffer*, levando em consideração a atenuação provocada pela aplicação do *footprint*. Na verdade, o *footprint* é uma tabela que determina como o *voxel* será "arremessado" sobre o plano da imagem. Isso significa que a contribuição do *voxel* é maior no centro de projeção sobre o plano da imagem (*pixel* central) e menor nos *pixels* mais afastados. Quando um determinado *pixel* do plano de projeção acumula opacidade próxima de 1.0, este *pixel* não precisa mais ser processado.

O algoritmo *Splatting* permite gerar imagens de alta qualidade, porém é muito sensível ao tamanho da tabela de *footprint*. Tabelas pequenas geram imagens com muitos artefatos enquanto tabelas grandes suavizam demais o volume. Como a projeção é realizada de frente para atrás, uma vantagem do algoritmo é permitir que o usuário acompanhe o processo de geração da imagem final. Isso é mais eficiente que, por exemplo, no algoritmo *Ray Casting*, pois é projetada uma fatia do volume de cada vez, e não um *pixel* por vez. Além disso, o algoritmo *Splatting* é facilmente paralelizável, pois a projeção de cada *voxel* é realizada de modo independente, ou seja, não é necessário considerar o restante do volume durante a projeção. Entretanto, a ordem correta de projeção deve ser respeitada.

3 OTIMIZAÇÕES DE ALGORITMOS

Este capítulo procura abordar algumas otimizações empregadas nos algoritmos de visualização volumétrica. As otimizações apresentadas aqui utilizam estruturas de dados mais elaboradas com a finalidade de reduzir a complexidade dos algoritmos e, com isso, o tempo necessário para construir uma imagem do volume. São estudadas algumas modificações nos algoritmos apresentados no capítulo anterior, ou seja, *Ray Casting*, *Shear-Warp* e *Splatting*.

3.1 Ray Casting

O algoritmo *Ray Casting* permite gerar imagens de alta qualidade, porém seu custo computacional é bastante grande caso nenhuma otimização seja utilizada. Isto ocorre devido ao processo de lançamento de raios, que deve ser realizada para cada *pixel* do plano da imagem, e do cálculo das contribuições dos *voxels* ao longo de cada raio.

Esta seção procura abordar as formas mais comuns de otimização do algoritmo *Ray Casting*, em especial, o término adaptativo de raios (*early ray termination*), refinamento adaptativo da imagem, uso de *templates* para agilizar o processo de discretização dos raios (*template-based volume viewing*), uso de estruturas de dados hierárquica para decompor o espaço volumétrico (*octrees*) e PARC (*polygon-assisted ray casting*). Essas técnicas basicamente procuram reduzir o número de raios lançados e o número de amostras realizadas ao longo dos raios. Para isto, as técnicas buscam explorar a coerência espacial dos dados.

3.1.1 Término Adaptativo de Raios

Essa técnica de otimização é bastante simples e pode ser utilizada quando os raios são traçados de frente para trás. Conforme visto no capítulo anterior, a opacidade α_{out} de um raio após atravessar uma amostra do volume semitransparente é dada em função da opacidade α_{in} antes de atravessar a amostra e da opacidade α da amostra em questão:

$$\alpha_{out} = \alpha_{in} + \alpha*(1-\alpha_{in})$$

Com isto, nota-se que a opacidade é acumulada enquanto o raio atravessa o volume. Nessa situação, quando a opacidade aumenta, a cor de cada amostra tem cada vez menos influência na cor final do *pixel*, ou seja, C_{out}-C_{in} torna-se cada vez menor. No caso limite, quando a opacidade acumulada atinge 1, a cor do *pixel* não sofre mais alteração, ou seja, $C_{out}=C_{in}$. Nesse caso, o traçado do raio pode ser interrompido.

Tipicamente, utiliza-se um valor limite máximo, especificado pelo usuário, para a opacidade acumulada. Quanto menor esse valor, mais rápido é o traçado de cada raio. Entretanto, quanto menor, melhor é a qualidade final da imagem.

3.1.2 Refinamento Adaptativo da Imagem

No algoritmo *Ray Casting*, o traçado dos raios são totalmente independentes entre si, podendo ser realizado em qualquer ordem. Com isso, pode-se visualizar progressivamente a construção da imagem, permitindo que o usuário tenha uma noção do resultado final sem ter que esperar pelo traçado de todos os raios.

A Figura 3.1 ilustra uma técnica simples de refinamento progressivo. Neste caso, a imagem é construída gradualmente e nenhuma consideração é feita com relação a complexidade de cada região da imagem, ou seja, a seqüência de construção é exatamente a mesma em todas as situações.

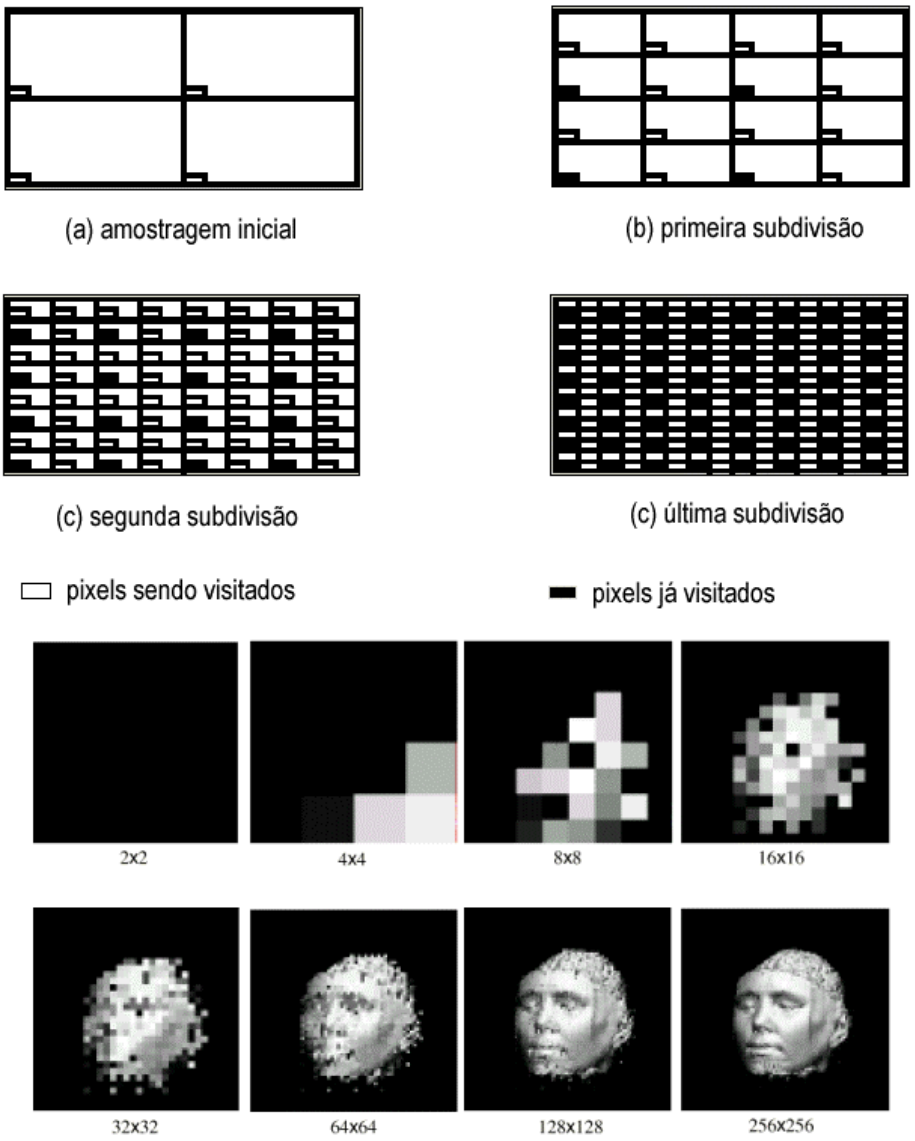

Figura 3.1: Refinamento progressivo [Paiva e outros, 99].

Um método mais eficiente, baseado no refinamento adaptativo da imagem, é descrito em [Levoy, 90c]. Através desse método, o plano de imagem é dividido em subregiões quadradas com w_{max} *pixels* em cada lado e raios são lançados em cada um dos quatro vértices e a cor de cada raio é calculada normalmente. Se a variação de cor nos quatro vértices for menor que um certo ε então nenhuma subdivisão é realiza. Caso contrário, a região é subdividida em quadrantes, conforme mostrado na Figura 3.2, e outros quatro raios são lançados. O processo continua até que a variação de cor seja menor que ε ou a região alcance a dimensão w_{min} *pixels*.

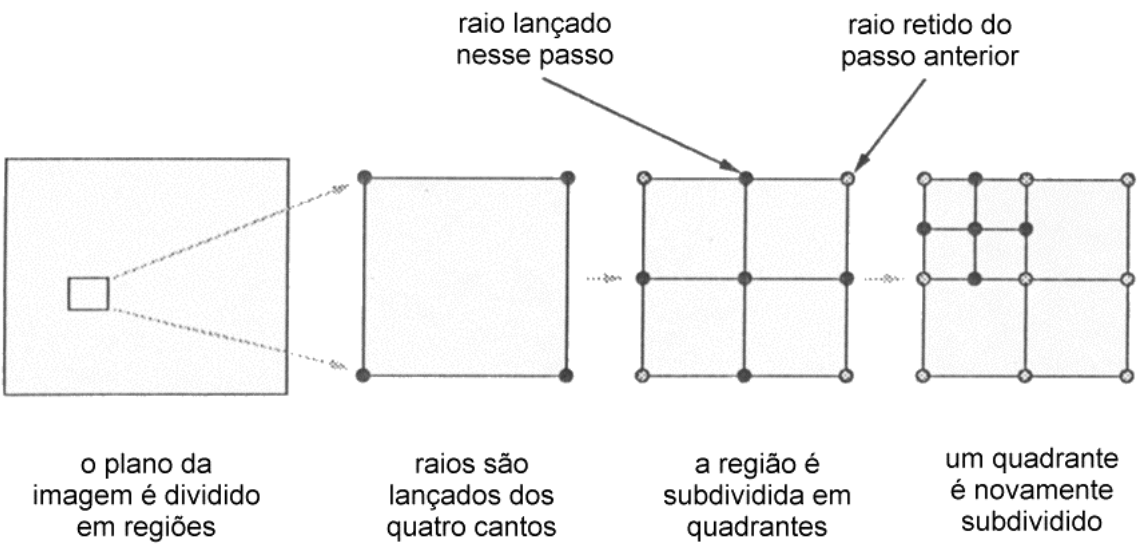

Figura 3.2: Refinamento adaptativo da imagem [Levoy, 90d]

O método apresentado anteriormente tem a vantagem de aumentar adptativamente a taxa de amostragem nas regiões de maior complexidade da imagem. Como isso, obtem-se mais rapidamente uma aproximação da imagem final. Segundo o autor, a eficiência desse método depende da correta escolha dos parâmetros w_{max}, w_{min} e ε. Isso deve ser feito manualmente pelo usuário. De uma forma geral, valores grandes de w_{max} fazem com que os detalhes da imagem sejam perdidos. Valores grandes de ε fazem com que os detalhes sejam ignorados e valores grandes de w_{min} fazem com que os detalhes sejam pobremente visualizados, mesmo que eles não sejam perdidos ou ignorados.

3.1.3 Templates

Para calcular a cor final de um *pixel* utilizando o algoritmo *Ray Casting* é necessário discretizar cada raio lançado e acumular a contribuição de cor e opacidade dos *voxels* ao longo do raio Essa operação consome bastante tempo de computação pois precisa ser realizada em todos os raios lançados no volume. Entretanto, para o caso de lançamento de raios paralelos, a discretização precisa ser realizada apenas uma única vez.

Yagel e Kaufman desenvolveram uma técnica chamada *Template-based volume viewing* [Yagel e Kaufman, 92] que utiliza um padrão de discretização que é utilizado para todos os raios lançados no volume. A Figura 3.3 ilustra o funcionamento deste método. Para garantir que cada *voxel* contribua apenas uma vez para a imagem e evitar

que outros não sejam considerados (Figura 3.3a) é necessário que o raio seja do tipo 26-*connected*, ou seja, caso seja retirado qualquer ponto o *template* perde a continuidade. Além disso, é necessário que os raios sejam lançados a partir do volume de dados e não do plano de visualização (Figura 3.3b).

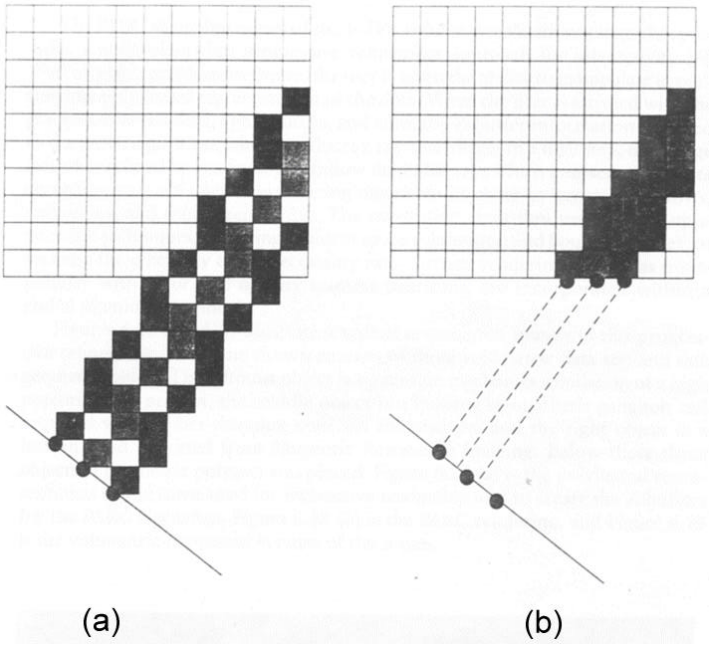

Figura 3.3: *Template-based volume viewing* [Kaufman e Sobierajski, 94].

3.1.4 Decomposição Hierárquica do Espaço

Esta técnica de otimização utiliza *octrees* para decompor o volume em regiões transparentes e não transparentes. A *octree* é uma representação hierárquica baseada na decomposição recursiva do espaço tridimensional em cubos (octantes). Cada octante pode ser totalmente transparente (contendo somente *voxels* transparentes) ou não transparente (contendo pelo menos um *voxel* não transparente). A idéia principal da técnica é procurar tirar o máximo proveito da coerência espacial do volume. Com isso, durante a integração dos *voxels* ao longo dos raios, pode-se descartar rapidamente regiões com *voxels* transparentes.

A técnica consiste em construir um modelo hierárquico do volume. Inicialmente, tem-se todo o volume de dados constituindo o nível mais alto da hierarquia. Caso exista algum *voxel* não transparente, o volume é subdividido em oito subvolumes (octantes ou

células), constituindo um nível imediamente inferior. Se todos os *voxels* são transparentes não é necessário subdividir. Para cada um dos oito subvolumes, o mesmo processo é aplicado até que se atinja o nível 0 da hierarquia, ou seja, não é mais possível subdividir a célula, pois ela contém exatamente oito *voxels* (nesse caso, um *voxel* é uma amostra do dado volumétrico).

De uma outra forma, uma célula de nível 0 é transparente (possui o valor 0) caso seus oito *voxels* dos vértices tenham opacidade igual a zero. Caso contrário, tem valor 1. Uma célula de qualquer outro nível tem valor 0 caso as oito células do nível exatamente inferior tenham valor 0. Caso contrário, essa célula tem valor 1. A Figura 3.4 ilustra a construção da *octree* de um volume 5x5x5.

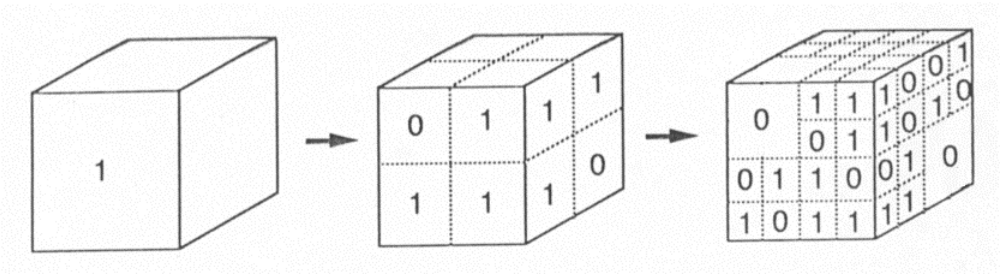

Figura 3.4: Construção de uma *octree* [Levoy, 90d].

Uma vez construída a *octree*, deve-se modificar o processo de lançamento de raios do algoritmo *Ray Casting*. Para cada raio traçado, primeiro é calculada a interseção do raio com a célula de nível máximo (todo o volume). Se ela contém 0, pode-se avançar para a próxima célula do mesmo nível. Se o pai desta nova célula não for o mesmo da célula anterior, deve-se subir um nível na hierarquia. Isso é muito importante caso essa nova célula contenha 0. Nessa situação, o algoritmo economiza tempo pois avança rapidamente pelo raio, ignorando todos os *voxels* transparentes.

Essencialmente, a idéia é avançar até atingir uma célula contendo 1. Nesse caso, deve-se descer um nível na hierarquia. Se atingir o nível 0, sabe-se que pelo menos um dos oito *voxels* da célula possui opacidade maior que 0. Nessa situação, deve-se amostrar o raio ao longo da célula e calcular a contruibuição das amostras para a cor final do *pixel*, como acontece no algoritmo padrão. A Figura 3.5 ilustra o caminhamento em uma *octree* para o caso bidimensional.

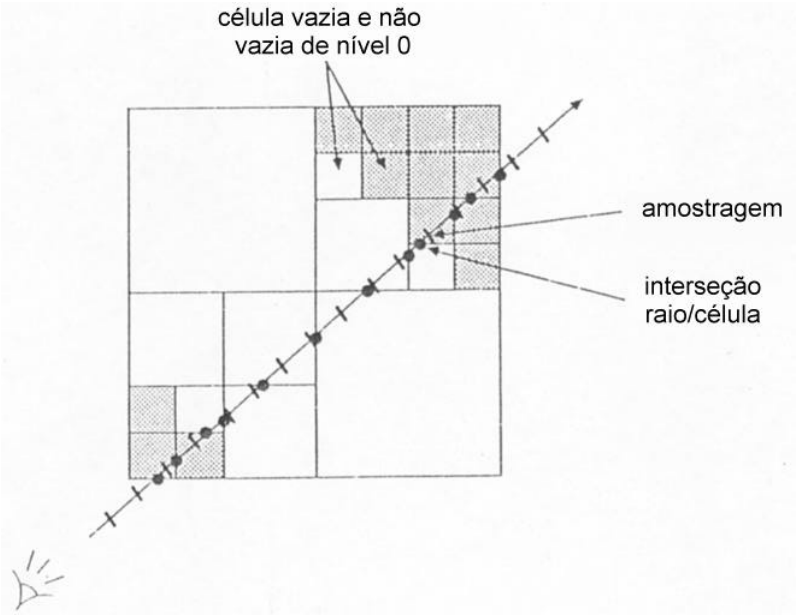

Figura 3.5: Caminhamento em uma *octree* 2D [Levoy, 90b].

É importante notar que a eficiência desta técnica de aceleração depende muito da coerência espacial do volume, pois só se consegue avançar sobre regiões transparentes caso o volume contenha muitos *voxels* transparentes contíguos. Vale notar também que a construção da *octree* pode ser realizada como uma etapa de pré-processamento. Uma vez construída, pode-se visualizar o volume a partir de diversas posições de câmera, pois a construção da *octree* é baseada apenas nas opacidades dos *voxels*.

3.1.5 Polygon Assisted Ray Casting (PARC)

Esta técnica de otimização consiste basicamente em reduzir ao máximo os limites para discretizar cada raio lançado no volume. Foi inicialmente implementada no sistema de visualização VolVis [Avila e outros, 92; Avila e outros, 94]. Para isto, é necessário desprezar os *voxels* transparentes no começo e no fim do traçado de cada raio. A Figura 3.6a ilustra os pontos amostrados ao longo do raio para o algoritmo *Ray Casting* de força bruta. O algoritmo otimizado utilizando a técnica PARC primeiro encontra uma representação poliédrica que contenha todos os *voxels* não transparentes (Figura 3.6b). Fora desta representação os demais *voxels* tem opacidade inferior a um valor limite

especificado pelo usuário. Pela figura, pode-se observar que um número muito menor de pontos são amostrados ao longo de cada raio.

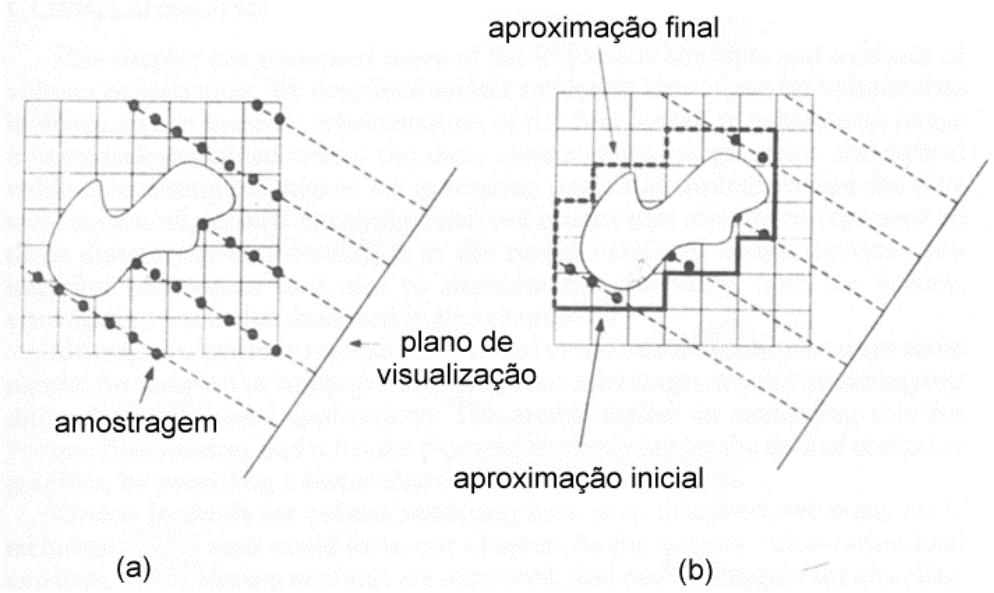

Figura 3.6: *Polygon Assisted Ray Casting* [Kaufman e Sobierajski, 94].

A implementação desta técnica envolve a projeção da representação poliédrica que contém o volume em dois *z-buffers*: o primeiro, convencional, guarda a menor distância ao observador ao volume enquanto o outro guarda a maior distância. Com isso, tem-se o limite inferior e superior para a discretização de cada raio lançado dos *pixels* do plano da imagem.

3.2 Shear-Warp

Uma das características mais importantes do algoritmo *Shear-Warp*, motivo principal de sua rapidez em relação aos demais, é o fato das *scanlines* no volume cisalhado serem alinhadas com as *scanlines* dos *pixels* na imagem intermediária (Figura 3.7). Isso significa que tanto o volume como a imagem intermediária podem ser percorridos simultaneamente. Além disso, pode-se tirar proveito da coerência espacial no volume e na imagem, utilizando estruturas de dados eficientes para evitar ao máximo o processamento desnecessário de *voxels* e *pixels*.

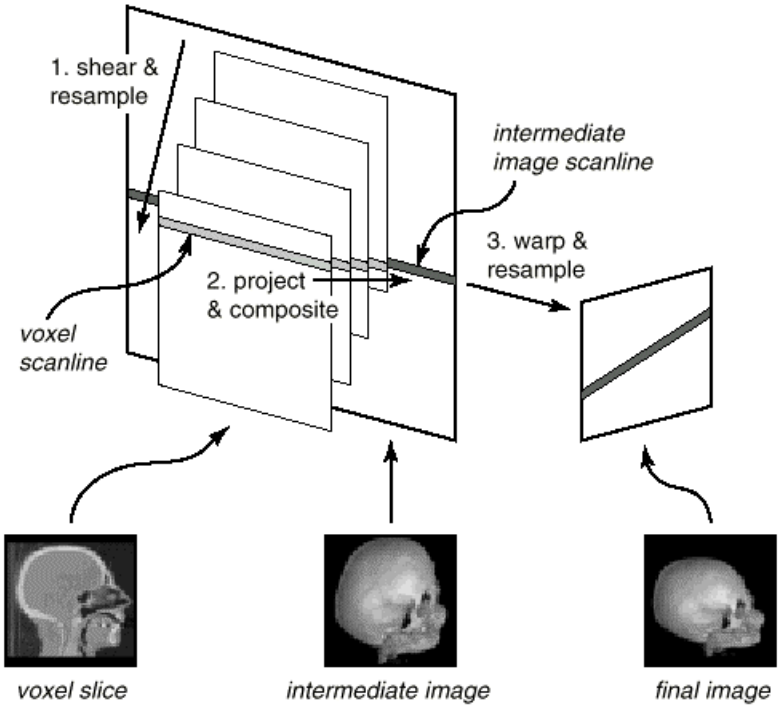

Figura 3.7: Alinhamento das *scanlines* do algoritmo *Shear-Warp* [Lacroute, 95].

Lacroute [Lacroute, 95] propõe a utilização de estruturas do tipo RLE (*run-length encoding*) para armazenar de maneira eficiente os *voxels* transparentes do volume cisalhado e os *pixels* opacos na imagem intermediária de maneira que, durante a construção da imagem intermediária, esses *voxels* e *pixels* possam ser ignorados.

As estruturas RLE permitem eliminar repetitivas especificações de dados com o mesmo valor. Para isso, as *scanlines* de *voxels* são divididas em seqüências ou grupos (chamados de *runs*) de *voxels* transparentes e não transparentes (utilizando um valor limite especificado pelo usuário). Da mesma forma, as *scanlines* de *pixels* são divididas em seqüências de *pixels* opacos e não opacos. A Figura 3.8 ilustra o uso de estruturas RLE no algoritmo *Shear-Warp*.

Com isso, obtém-se uma representação compacta do volume e da imagem, acelerando o processo de visualização do volume, pois as *scanlines* são percorridas simultaneamente e de maneira síncrona, evitando perder tempo processando *voxels* transparentes e *pixels* já opacos. Ou seja, apenas *voxels* não transparentes e visíveis são efetivamente processados.

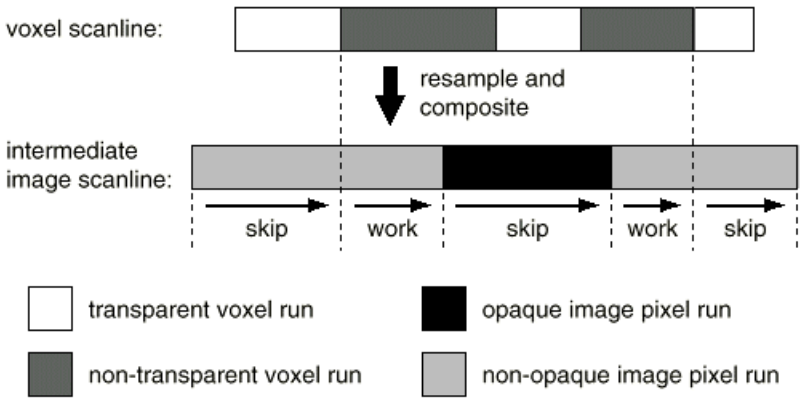

Figura 3.8: Composição utilizando as estruturas RLE [Lacroute, 95].

A figura anterior mostra também de forma o algoritmo *Shear-Warp* é acelerado pela simples utilização de estruturas RLE. Observa-se claramente que, durante a composição dos *voxels scanline* a *scanline*, os grupos de *voxels* transparentes são ignorados. Isto também acontece com os grupos de *pixels* opacos. Obviamente, este tipo de otimização é muito dependente do volume. Entretanto, em muitas situações, principalmente com imagens médicas, o número de *voxels* transparentes é muito grande, tornando o dado volumétrico bastante coerente. Com isso, o algoritmo é bastante acelerado, com a vantagem de não haver perda de qualidade na imagem final.

A construção da estrutura RLE do volume pode ser realizada em uma etapa de pré-processamento, pois depende apenas do resultado da classificação dos *voxels* do dado volumétrico (atribuição de opacidade para cada *voxel*). Entretanto, a construção da estrutura RLE da imagem intermediária deve ser realizada simultaneamente com a construção da própria imagem intermediária, pois os *pixels* da imagem intermediária ficam opacos a medida em que os *voxels* são compostos para a geração a imagem intermediária do volume.

3.3 Splatting

Laur e Hanrahan [Laur e Hanrahan, 91] apresentam algumas melhorias no algoritmo *Splatting* tradicional de Westover [Westover, 90]. Os autores procuram explorar a coerência do volume e utilizam refinamento progressivo da imagem. Para isto, é

construída uma representação piramidal do volume. Essa estrutura de dados permite a representação do dado original em multiresolução.

A construção da pirâmide é realizada de forma hierárquica, i.é., o primeiro nível da hierarquia consiste na representação de todo o volume original de dados. O nível seguinte é criado através do cálculo da média de um grupo de 2x2x2 *voxels* do volume original, representando, assim, o mesmo dado com 1/8 da resolução inicial. O processo se repete até que todos os níveis sejam criados. Além de armazenar os valores médios de cor e opacidade, cada nó da pirâmide registra um erro estimado associado ao nó. Este erro é uma medida da aproximação de uma região do espaço utilizando valores médios de um grupo de *voxels* contidos na região.

A estrutra piramidal aproxima o volume original em múltiplas resoluções, permitindo implementar facilmente a técnica de refinamente progressivo da imagem. O algoritmo *Splatting* ordena as células de trás para frente e então compõe a projeção de cada célula (chamada de *footprint*) acumulando cores e opacidades no plano de imagem. Durante a composição, a pirâmide é percorrida e em cada nível é verificado se o erro desejado por unidade de volume (especificado pelo usuário) é menor que o erro médio armazenado no nó da pirâmide. Se for, o percorrimento da pirâmide se encerra neste nível. Caso contrário, é necessário descer um nível na hierarquia e repetir o processo recursivamente.

Utilizando essa técnica, regiões onde ocorrem mudanças mais rápidas no dado volumétrico são aproximadas utilizando um número maior de pequenos nós da pirâmide, enquanto nas regiões onde o dado varia mais lentamente, uma aproximação é rapidamente atingida utilizando um número pequeno de nós, porém de dimensões maiores. Isso significa que o número de nós necessários para aproximar o volume é dependente do erro: quanto maior o erro, maior o número de nós e vice-versa. Com isso, regiões mais simples são representadas em uma resolução menor, enquanto regiões mais complexas são representadas em uma resolução maior. Assim, a imagem do volume é construída com diferentes níveis de detalhe.

É importante notar que a técnica de refinamento progressivo apresentada anteriormente requer o uso de *splats* de diferentes tamanhos. Para isso, os autores não utilizam gaussianas para aproximar o *footprint*, como faz Westover. Ao invés disso, utilizam uma aproximação baseada em uma coleção de polígonos RGBA (*Gouraud-shaded polygons*). Esses polígonos são facilmente desenhados pelo *hardware* gráfico em

diferentes tamanhos sem muito esforço adicional, além do tempo necessário para a *rasterização*. Com isso, o tempo necessário para desenhar cada *splat* é praticamente o mesmo, dependendo apenas da área. Nesse ponto o algoritmo é realmente acelerado, pois sempre que a aproximação não comprometer muito a qualidade da imagem, *splats* maiores são utilizado. Além disso, utilizando polígonos RGBA, o algoritmo consegue implementar projeções em perspectiva com mais facilidade que o algoritmo tradicional de Westover.

De uma maneira geral, os autores constataram que o uso da estrutura piramidal permitiu explorar a coerência do volume, implementar refinamento progressivo e, com isso, possibilitou a obtenção de tempos interativos para a geração da imagem do volume. Isso é importante para a animação e visualização exploratória do dado volumétrico, pois pode-se refinar a imagem quando houver tempo disponível (por exemplo, quando o usuário observa um detalhe ou quando o processador estiver em *idle*).

Obviamente, segundo os autores, a qualidade da imagem obtida na maior resolução utilizando essas técnicas é ainda inferior a qualidade obtida utilizando o algoritmo original, sem otimizações. Entretanto, as aproximações permitem tempos interativos, sacrificando a qualidade da imagem. Por outro lado, os autores se mostraram surpresos com o desempenho do algoritmo quando as imagens de baixa qualidade são visualizadas em movimento. Obviamente, isso não ocorre quando elas são visualizadas paradas.

4 HARDWARES ESPECIAIS

Embora o uso da visualização volumétrica em diversas aplicações científicas tem crescido constantemente, alguns fatores têm contribuído para restringir sua efetiva popularização. Um dos mais importantes é, sem dúvida, a dificuldade de se obter tempos interativos que permitam a visualização exploratória, tendo em visto o crescente aumento no tamanho do dado volumétrico.

Algumas tentativas foram realizadas para contornar este problema. Uma delas é a otimização dos algoritmos de visualização volumétrica. Muitas delas foram apresentadas no capítulo anterior e talvez o algoritmo *Shear-Warp* tem apresentado os melhores resultados. Entretanto, algumas otimizações requerem o uso de estruturas de dados complexas e por isso podem consumir bastante tempo de pré-processamento. Isso pode comprometer a visualização caso essas estruturas precisem ser atualizadas constantemente como, por exemplo, se as funções de transferências forem alteradas. Isso acontece no algoritmo *Shear-Warp*. Além disso, muitas otimizações são dependentes do dado, não sendo possível, portanto, garantir uma taxa de atualização (*frame rate*) fixa.

Uma segunda alternativa foi utilizar mapeamento de texturas 3D em *hardware*. Apesar de ter sido obtido tempos interativos utilizando essa técnica, o mapeamento de texturas não permite o uso de estimativa de gradientes, necessário para o cálculo da iluminação. Além disso, ainda há sérias limitações de memória para armazenar texturas na placa gráfica.

A terceira alternativa foi utilizar supercomputadores e bastante processamento paralelo. Entretanto, o alto custo de supercomputadores impediu o efetivo uso dessa solução. Além disso, o maior problema enfrentado pelos algoritmos paralelos é a necessidade de transferir uma quantidade cada vez maior de dados entre as estações e o problema da latência e largura de banda apresentada pelas memória convencionais. Isso dificultou a utilização de computação paralela para implentar algoritmos eficientes de visualização volumétrica [Kaufman, 97].

Uma nova alternativa que vem apresentado alguns resultados importantes é a utilização de *hardwares* especiais para visualização volumétrica. Entretanto, poucas

soluções foram realmente colocadas a disposição no mercado. Este capítulo pretende dar uma visão geral de algumas arquiteturas dedicadas para visualização volumétrica

A necessidade de *hardware* específico para visualização volumétrica pode ser claramente justificada pela exigência de desempenho, principalmente com relação ao acesso a memória. Para obter-se tempos interativos (30 quadros por segundo) com dados de 16 bits de dimensão 128^3 é necessário taxa de transferência (largura de banda) de 120Mb/s. Para dados de dimensão 256^3 é necessário 960 Mb/s. Para dados de dimensão 512^3 é necessário 7.5 Gb/s. Finalmente, para dados de dimensão 1024^3 é necessário 60 Gb/s [Ray e outros, 99].

A grande maioria das arquiteturas disponíveis para visualização volumétrica utilizam o algoritmo *Ray Casting* para realizar o *rendering* de volumes. Esta foi a escolha mais imediata devido a facilidade de implementá-lo em *hardware* e a existência de muitas otimizações. Entretanto, o algoritmo *Ray Casting* tradicional é difícil de ser paralelizado, devido ao acesso aleatório ao volume de dados. Por isso, existem algumas variações do algoritmo básico (que trabalha no espaço da imagem) para trabalhar no espaço dos objetos ou então uma solução híbrida, tal como a utilizada no algoritmo *Shear-Warp*. Essas duas alternativas têm a grande vantagem de acessar o volume de dados de uma maneira previsível, sendo, portanto, facilmente paralelizável. Entretanto, pode ser mais difícil de ser otimizado (por exemplo, término adaptativo de raios) ou pode requerer alguma nova etapa de processamento (por exemplo, *warp*).

Por uma questão de desempenho, todas as arquiteturas estudadas são baseadas em dados volumétricos escalares e distribuídos em uma grade regular 3D. Os componentes mais importantes, implementados em *hardware*, de uma arquitetura de visualização volumétrica baseada no algoritmo *Ray Casting* são [Ray e outros, 99]:

- *Sistema de Memória*. É um dos gargalos existentes nas arquiteturas atuais. Deve armazenar o dado volumétrico e prover acesso rápido a ele. Deve ter baixa latência, alta largura de banda e permitir acesso simultâneo à memória;
- *Traçado de Raios*. Deve ser capaz de calcular um grande número de endereços de memória em paralelo. Para isso, é razoável implementar *templates* (veja seção 3.1.3) para reduzir o tempo de lançamento de raios pelo volume;
- *Interpolação*. Tipicamente utiliza interpolação trilinear, requerendo, portanto, bastante esforço computacional devido ao grande número de multiplicações e somas

envolvidas. Interpolações de ordem mais elevadas, apesar de melhorar a qualidade da imagem, não é tipicamente utilizada, devida às dificuldades de implementação em *hardware*;

- *Estimativa de Gradiente.* Utilizado para aproximar a normal de uma superfície. Tipicamente utiliza diferenças centrais, porém alguns algoritmos utilizam uma vizinhança maior de *voxels* para reduzir o *aliasing*. Esta componente é fundamental para garantir a geração de imagens de qualidade;
- *Classificação*. Mapeia os dados escalares em cores e opacidades. Tipicamente é implementado utilizando *lookup tables* (LUTs). É desejável que essas tabelas possam ser modificadas em tempo real;
- *Iluminação*. Tipicamente utiliza-se o algoritmo *Phong* [Phong, 75] ou seus variantes. Esse componente é obrigatório ser implementado em *hardware*, devido ao seu alto custo computacional;
- *Composição*. É responsável por integrar as contribuições de cor e opacidade de cada *voxel* amostrado durante o traçado de raios. Pode ser realizada de frente para trás ou de trás para frente. No primeiro caso (frente para trás) pode-se utilizar a otimização de término adaptativo do raio, enquanto no segundo é mais fácil de ser implementado e o usuário pode acompanhar a construção da imagem. Entretando, não permite implementar o término adaptativo do raio.

A seguir será apresentado algumas das arquiteturas mais importantes de visualização volumétrica. Entretanto, um estudo mais aprofundado pode ser encontrado em diversos trabalhos, tais como [Ray e outros, 99], [Pfister e Kaufman, 96], [Pfister e outros, 99] e [Yagel, 96].

4.1 VOGUE

Esta arquitetura [Knittel, 93; Knittel, 95] foi desenvolvido na Universidade de Tübingen na Alemanha e permite obter 2.5 quadros/s para dados com dimensão de 256^3. É capaz de produzir imagens em perspectiva e implementa a otimização de término adaptativo de raios. Um dos objetivos desta arquitetura é a flexibilidade, permitindo combinar vários módulos VOGUE e com isso melhorar o desempenho. Entretanto, utilizando a melhor

qualidade de imagem (através da seleção do melhor algoritmo de estimativa de gradiente), não se obtém tempos interativos.

4.2 VIRIM

Esta arquitetura [Günther e outros, 95] foi desenvolvida na Universidade de Mannhein e permite obter 10 quadros/s para dados com dimensão de 256x256x128 com imagens de alta qualidade. Utiliza uma variação do algoritmo *Ray Casting* no espaço dos objetos [Meinzer e outros, 91] e suporta sombras e projeções em perspectiva.

4.3 Cube-4

A arquitetura Cube-4 [Pfister e Kaufman, 96] foi desenvolvida na Universidade de New York em Stony Brook. Foi a primeira arquitetura que possibilitou visualização de alta qualidade em tempo real (30 quadros/s) com dados de dimensão 1024^3. Esta arquitetura é altamente paralelizada e utiliza o algoritmo *Ray Casting* híbrido (os autores chamam de *slice parallel ray-casting*).

Este algoritmo é bastante semelhante ao *Shear-Warp*. A utilização de *Ray Casting* híbrido deve-se a facilidade de implementá-lo em paralelo. Os *voxels* são processados por fatias do volume e, em cada fatia, raios são lançados simultaneamente, resultando em um algoritmo no espaço dos objetos. Além disso, sua principal característica é que cada *voxel* é acessado apenas uma única vez por projeção. Além disso, a organização da mémoria permite que o acesso dos *voxels* sem conflitos na direção dos três eixos principais de visão (*linear skewing*). Esta arquitetura possui múltiplas *pipelines* de *rendering* e cada uma implementa o algoritmo *Ray Casting* híbrido.

4.4 VolumePro

Esta arquitetura foi a primeira implementada comercialmente para a plataforma PC [Pfister e outros, 99]. Foi desenvolvida pela Mitsubishi Electric e a primeira versão foi lançada em junho de 1999. Ela permite a visualização de imagens de alta qualidade em tempo real (30 quadros/s) com dados volumétricos de dimensão 256^3. Atualmente já existe uma nova versão capaz de visualizar 1024^3 a 30Hz. O sistema é composto por uma

placa gráfica PCI , com 128Mb de memória para armazenar o volume e uma API em C++ para acessar placa (*The Volume Library Interface*).

VolumePro é baseada na arquitetura Cube-4 [Pfister e Kaufman, 96] porém, devido ao custo elevado desta arquitetura, que dificultava a produção comercial a um preço aceitável, os autores desenvolveram uma versão de custo mais baixo chamada de *Enhanced Memory Cube-4*, ou simplesmente EM-Cube [Osborne e outros, 97].

Esta arquitetura implementa em *hardware* estimativa de gradiente, classificação e iluminação *Phong*. Além disso, a placa não realiza nenhum tipo de pré-processamento. Isso significa que o usuário pode alterar livremente as funções de transferências de cor e opacidade que o dado é imediatamente visualizado. Assim, esta arquitetura é apropriada para visualização exploratória de dados volumétricos.

Para atingir os objetivos comerciais (tempo e custo), algumas simplificações foram necessárias. A arquitetura suporta apenas projeções paralelas e dados escalares de 8 e 12 bits somente. Não é possível visualizar polígonos junto com os dados volumétricos. Além disso, os dados devem estar organizados em uma grade regular 3D. Entretanto os autores garantem que essas limitações serão vencidas nas próximas versões da placa.

VolumePro implementa o mesmo algoritmo *Ray Casting* híbrido da arquitetura Cube-4. Entretanto, vários recursos adicionais foram introduzidos nessa nova arquitetura. A qualidade da imagem pode ser bastante melhorada através da utilização de superamostragem (*supersamplimg*). Entretanto, a placa implementa em *hardware supersampling* apenas na direção z. Segundo os autores, o impacto no tempo de processamento é mínimo e a melhoria da qualidade é bastante acentuada.

Uma outra melhoria é a implementação de supervolumes e subvolumes. Supervolumes permitem visualizar um volume de dados maior que a capacidade da placa. O *software* automaticamente cuida do particionamento do volume de forma transparente. Subvolumes,por outro lado, permitem aumentar a velocidade de processamento.

Por fim, a arquitetura VolumePro implementa *clipping* de volume, permitindo visualizar fatias com espessuras e orientações arbitrárias. Além disso, pode-se utilizar múltiplos planos de *clipping* paralelo às faces do volume. Com isto, pode-se fazer *cropping* de forma bastante flexível.

5 CONCLUSÕES

Este trabalho inicialmente apresenta uma visão geral de *rendering* de volumes e discute o *pipeline* de visualização volumétrica. A seguir, apresenta os aspectos mais importantes dos principais algoritmos de visualização volumétrica direta, tais como *Ray Casting*, *Shear*-Warp e *Splatting*.

Depois são estudadas diversas técnicas de aceleração dos algoritmos apresentados. A grande maioria dessas otimizações procuram tirar proveito da coerência espacial dos dados. Nota-se que o algoritmo *Ray Casting* pode ser otimizado de diversas maneiras, tais como o término adaptativo de raios, refinamento adaptativo da imagem, *templates*, decomposição hierárquica do espaço e PARC. O algoritmo *Shear-Warp* pode ser otimizada utilizando *run-length encoding* do volume e dos *pixels* da imagem intermediária. Já o *Splatting* pode ser otimizado utilizando representações em multiresolução (através do uso de estruturas piramidais), o que permite também implementar refinamento progressivo da imagem. Também é discutido o uso de polígonos RGBA ao invés de gaussianas para aproximar o *footprint*. Isto foi importante devido a necessidade de *splats* de vários tamanhos.

A seguir, o trabalho apresenta rapidamente algumas arquiteturas especiais de visualização volumétrica. Essas arquiteturas permitem visualização em tempo real (30 quadros/s) para dados volumétricos de dimensão 1024^3. Essas placas tipicamente implementam o algoritmo *Ray Casting* ou algumas de suas variações, tendo em vista a facilidade de implementá-lo em *hardware* e a possibilidade de otimizá-lo de diversas maneiras diferentes.

O desenvolvimento dessas arquiteturas, em especial *Cube-4* e *VolumePro*, foi uma clara conseqüência das dificuldades de se obter visualização volumétrica em tempo real utilizando algoritmos mais eficientes ou otimizações inteligentes de *software*. O surgimento de *hardwares* dedicados para visualização volumétrica é uma tendência natural, já que a demanda por visualização exploratória é cada vez mais crescente. A direção atual de pesquisa deve ser, sem dúvida, voltada para o desenvolvimento de arquiteturas dedicadas, pois os resultados obtidos estão superando os avanços dos algoritmos de *rendering*.

6 REFERÊNCIAS

[Avila e outros, 92] Avila, R., Sobierajski, L., Kaufman, A., *Towards a Comprehensive Volume Visualization System*, In Visualization'92 Proceedings, pp. 13-20, October, 1992.

[Avila e outros, 94] Avila, R., He, T., Hong, L., Kaufman, A., Pfister, H., Silva, C., Sobierajski, L., Wang, S., *VolVis: A Diversifiied Volume Visualization System*, In Visualization'94 Proceedings, pp. 31-38, October, 1994.

[Gerhardt e outros, 98] Gerhardt, A., Paiva, A., Schmidt, A.E., Martha, L.F.; Carvalho, P.C., Gattass, M., *Aspects of 3-D Seismic Data Volume Rendering*, Proceedings of the GOCAD ENSG Conference – 3D Modeling of Natural Objects: A Challenge for the 2000's, Nancy, França, 04 a 05 de junho de 1998.

[Günther e outros, 95] Günther, T., Poliwoda, C., Reinhart, C., Hesser, J., Männer, R., Meinzer, H. P., Baur, H. J., *VRIM: A Massively Parallel Processor for Real-Time Volume Visualization in Medicine*, Proceedings of the 9th Eurographics Hardware Workshop, Vol. 19, No. 5, pp. 705-710, 1995.

[Kaufman e Sobierajski, 94] Kaufman, A. E., Sobierajski, L. M., *Continuum Volume Display*. Computer Visualization: Graphics Techniques for Scientific and Engineering Analysis, Chapter 6, Edited by Richard S. Gallagher, pp171-202, 1994.

[Kaufman, 97] Kaufman, A. E., *Volume Visualization: Principles and Advances*. SIGGRAPH'97 Course Notes, 1997.

[Knittel, 93] Knittel, G., *VERVE: Voxel Engine for Real-Time Visualization and Examination*, Computer Graphics Forum, Vol. 19, No. 3, pp. 37-48, September, 1993.

[Knittel, 95] Knittel, G., *A Scalable Architecture for Volume Rendering*, Computer and Graphics, Vol. 19, No. 5, pp. 653-665, 1995.

[Lacroute, 95] Lacroute, P., *Fast Volume Rendering Using a Shear-Warp Factorization of theViewing Transformation*", Ph.D. Thesis, Stanford University, 1995.

[Laur e Hanrahan, 91] Laur, D., Hanrahan, P., *Hierarchical Spaltting: A Progressive Refinement Algorithm for Volume Rendering*, Computer Graphics (Proc. SIGGRAPH'91), Volume 25, Number 4, July, 1991.

[Levoy, 88] Levoy, M., *Display of Surfaces from Volume Data*", IEEE Computer Graphics and Applications, Vol. 5, No. 3, pp 29-37, 1988.

[Levoy, 90a] Levoy, M., *Ray Tracing of Volume Data,* SIGGRAPH'90 Course Notes, Volume Visualization Algorithms and Archtectures, August, 1990, pp. 120-147.

[Levoy, 90b] Levoy, M., *Efficient Ray Tracing of Volume Data,* Computer Graphics (Proceedings of SIGGRAPH'90), pp. 157-167, 1990.

[Levoy, 90c] Levoy, M., *Volume Rendering by Adaptative Refinement*, The Visual Computer (1990) 6:2-7, pp. 175-180, 1990.

[Levoy, 90d] Levoy, M., *Ray Tracing of Volume Data,* First Conference on Visualization in Biomedical Computing, Atlanta, Georgia, May 22-25, pp. 120-147, 1990.

[Meinzer e outros, 91] Meinzer, H. P., Meetz, K., Scheppelmann, D., Engelmann, U., Baur, H. J., *The eidelberg Rayy Tracing Model*, IEEE Computer Graphics and Applications, pp. 34-43, November, 1991.

[Osborne e outros, 97] Osborne, R., Pfister, H., Lauer, H., McKenzie, N., Gibson, S., Hiatt, W., Ohkami, T., *EM-Cube: An Archtecture for Low-Cost Real-Time Volume Rendering*. In Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware, pp. 131-138, August, 1997.

[Paiva e outros, 99] Paiva, A. C., Seixas, R. B., Gattass, M., *Introdução à Visualização Cientifica*, Monografia em Ciência da Computação, nº 3/99, Departamento de Informática, PUC-Rio, 1999.

[Pfister e Kaufman, 96] Pfister, H., Kaufman, A., *Cube-4 – A Scalable Architecture for Real-Time Volume Rendering*, 1996 Symposium on Volume Visualization Proceedings, 1996.

[Pfister e outros, 99] Pfister, H., Hardenbergh, J., Knittel, J., Lauer, H., Seiler, L., *The VolumePro Real-Time Ray-Casting System*, Computer Graphics (Proceedings of SIGGRAPH'99), pp. 251-260, 1999.

[Porter e Duff, 84] Porter, T., Duff, T., *Compositing Digital Images,* Computer Graphics, Vol. 18, No. 3., July, 1984.

[Phong, 75] Phong, B. T., *Illumination for Computer Generated Pictures,* Communications of ACM, Vol. 18, No. 6, pp. 311-317, 1975.

[Ray e outros, 99] Ray, H., Pfister, H., Silver, D., Cook, T. A., *Ray Casting Archtectures for Volume Visualization*, IEEE Transactions on Visualization & Computer Graphics, Vol. 5, No,. 3, pp. 210-223, July-September, 1999.

[Upson e Keeler, 88] Upson, C., Keeler, M., *V-Buffer – Visible Volume Rendering*, Computer Graphics, Vol. 22, No. 4, August, 1988.

[Yagel e Kaufman, 92] Yagel, R., Kaufman, A., *Template-Based Volume Viewing*, In Proceedings of Eurographics'92, Cambridge, UK, pp. 153-167, September, 1992.

[Yagel, 96] Yagel, R., *Towards Real Time Volume Rendering*, Proc. GRAPHICON'96, Vol. 1, pp. 230-241, July, 1996.

[Westover, 90] Westover, L., *Footprint Evaluation for Volume Rendering*, Computer Graphics (Proc. SIGGRAPH), 24 (4), pp. 144-153, August, 1990.