Visualização Volumétrica com Otimizações de Ray Casting e Detecção de Bordas

ROBERTO DE BEAUCLAIR SEIXAS[1,2]
MARCELO GATTASS[1]
LUIZ HENRIQUE DE FIGUEIREDO[1]
LUIZ FERNANDO MARTHA[1]

[1] ICAD – Laboratório de CAD Inteligente, PUC–Rio
Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, RJ, Brasil
{tron,gattass,lhf,lfm}@icad.puc-rio.br

[2] Laboratório Nacional de Computação Científica
Rua Lauro Müller, 455, 22290–160 Rio de Janeiro, RJ, Brasil
tron@lncc.br

Abstract. Ray Casting is a useful volume visualization technique that has a high computational cost. We propose some optimizations to the ray casting algorithm for previewing volume data. We also propose a simple classification technique, based on a 3D edge detection algorithm, that produces good images. We compare our results with the latest algorithm presented by Lacroute and Levoy.

Keywords: Computer Graphics, Volume Visualization, Volume Rendering.

1 Introdução

Técnicas de visualização volumétrica direta têm sido desenvolvidas para obter uma melhor compreensão de conjuntos de dados tridimensionais, obtidos de amostragens (CT, MRI), simulações (elementos finitos, diferenças finitas) e técnicas de modelagem. Originalmente, dados volumétricos eram visualizados através de aproximações poligonais das iso-superfícies, obtidas por interpolação dentro dos elementos de volume (*voxels*).

O principal problema das técnicas de interpolação é detectar quando uma iso-superfície passa por um *voxel*. Uma detecção imprecisa pode produzir inconsistências geométricas e topológicas, apresentadas visualmente como falhas ou superfícies espúrias [Elvins (1992)].

A visualização direta não usa representações geométricas intermediárias, como os polígonos usados nas técnicas de iso-superfícies, evitando os problemas de detecção imprecisa mencionados. Além disso, como todo o conjunto de dados é usado, preserva-se a noção global das características dos dados.

A principal desvantagem da visualização volumétrica direta é o alto tempo de processamento quando comparado à visualização por iso-superfícies. Por isso, é extremamente importante identificar otimizações e simplificações nos algoritmos disponíveis na literatura, de forma a possibilitar a sua utilização em diversas áreas de aplicação.

Um outro aspecto importante é a questão de modelos de representação (*voxel* × célula) e de armazenamento dos dados. Para serem úteis, técnicas de visualização volumétrica têm que tirar proveito de representações eficientes que permitem uma manipulação adequada para os dados. Finalmente, usuários têm necessidade de interagir com os parâmetros do modelo e ver a imagem resultante instantaneamente. Poucos sistemas atuais podem oferecer este tipo de desempenho.

Neste trabalho, descrevemos uma implementação otimizada do algoritmo de *ray casting* para visualização volumétrica, possibilitando uma visualização rápida (*preview*). Descrevemos também uma extensão 3D do algoritmo de detecção de bordas para uma visualização final de qualidade.

2 O Algoritmo de *Ray Casting* original

O algoritmo de *ray casting* foi originalmente proposto por Levoy (1988) e Drebin *et al*. (1988), como uma técnica que permitia a visualização de pequenos detalhes internos ao volume, através do controle de transparência dos *voxels*, removendo trivialmente as partes escondidas atrás de partes definidas como opacas, e visualizando o volume a partir de qualquer direção. O algoritmo efetua um lançamento de raios a partir do observador em direção ao volume. A cor final de cada *pixel* da imagem é obtida integrando as contribuições de cor $C(x)$ e opacidade $\alpha(x)$ de cada *voxel* x interceptado pelo raio.

Na Figura 1, a cor do *pixel* correspondente ao raio antes do cálculo da contribuição do *voxel* em questão é c_{in}. Computada a contribuição de um *voxel x*, a cor passa a ser c_{out}:

$$c_{out} = c_{in} (1 - \alpha(x)) + C(x)\, \alpha(x).$$

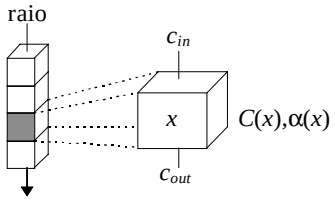

Figura 1: Composição da cor no *voxel*.

Levoy (1988) implementou este algoritmo através de dois *pipelines* independentes: um para iluminação e um para classificação do material, concluindo com uma fase final de composição dos dois *pipelines*.

Na etapa de iluminação, as componentes RGB da cor $C(x)$ para cada *voxel x* são calculadas a partir de uma estimativa do gradiente da função densidade $D(x)$ e da intensidade de luz, usando o algoritmo de Phong [Foley (1990)]. Na etapa de classificação, uma opacidade $\alpha(x)$, baseada na densidade dos materiais, é associada a cada *voxel*. A composição final é realçada enfatizando-se as bordas das regiões de densidade quase uniforme, e desenfatizando-se seus interiores, multiplicando-se o valor da opacidade pelo gradiente:

$$\alpha'(x) = \alpha(x) \mid \nabla D(x) \mid .$$

A visualização final da imagem é feita pela projeção bi-dimensional de $C(x)$ e $\alpha(x)$ no plano de visualização.

3 Trabalhos Correlatos

Sabella (1988) modificou o algoritmo original de Levoy (1988) considerando cada *voxel* como um emissor de densidade variável. Neste método, quatro valores são calculados para cada raio lançado através do volume: o maior valor encontrado ao longo do raio, a distância para este valor máximo, a intensidade da atenuação, e o centro de gravidade. A imagem é gerada usando o modelo de cor HSV (*Hue*, *Saturation* e *Value*) da seguinte forma: *hue* é o maior valor encontrado, *saturation* é a distância para o valor máximo ou para o centro de gravidade, e *value* é a intensidade da atenuação. No algoritmo original, Levoy utiliza os próprios valores no *voxels* para a determinação da cor e opacidade.

Westover (1990) propôs uma técnica, denominada s*platting*, que realiza um caminhamento ordenado pelos elementos. As contribuições dos *voxels* são calculadas e compostas usando-se tabelas de mapeamento (*look-up tables*). O primeiro passo do algoritmo é determinar em qual ordem o volume deve ser percorrido, ou seja, qual o *voxel* mais próximo do plano de visualização. No segundo passo, os *voxels* são "atirados" de acordo com as suas distâncias; a seguir, é calculada a projeção dos *voxels* no plano de visualização. Um filtro de arredondamento (*reconstruction kernel*) é usado para determinar a extensão da contribuição. A projeção do filtro no plano de visualização, denominada *footprint*, é proporcional ao tamanho do volume e ao tamanho da imagem a ser gerada.

Uma otimização para o algoritmo de Westover, chamada *hierarquical splatting*, foi proposta por Laur e Hanrahan (1991) e utiliza um caminhamento hierárquico nos dados e refinamento progressivo da imagem. Um algoritmo similar, denominado *V-buffer*, foi apresentado por Upson e Keeler (1988) e tem como principal característica o fato de se basear em células ao invés de *voxels*. O algoritmo de *V-buffer* percorre o interior das células, interpolando os valores dos vértices e projetando cada valor interpolado no plano de visualização.

Posteriormente, começaram a surgir otimizações no algoritmo de *ray casting* de forma a reduzir o consumo de memória e o alto custo computacional. As primeiras otimizações foram propostas pelo próprio Levoy (1990), substituindo a enumeração exaustiva do volume por uma enumeração espacial hierárquica, onde *voxels* de valores semelhantes são agrupados em células, e utilizando a terminação do raio de uma forma adaptativa, interrompendo o cálculo das contribuições quando estas não forem mais significativas.

Lacroute e Levoy (1994) apresentaram um algoritmo eficiente que combina as vantagens de outros algoritmos, utilizando a coerência no volume e na imagem, obtendo um acesso sincronizado às estruturas de dados, mantendo sempre o alinhamento entre o caminhamento no volume e no plano de visualização.

Zuffo e Lopes (1994) apresentaram uma otimização significativa do *pipeline* original de Levoy, antecipando a reamostragem e eliminando um volume intermediário com as componentes RGBO (cor e opacidade). Além disso, as imagens finais têm uma melhor qualidade devido a uma melhor aproximação do gradiente, a partir dos dados reamostrados, e não a partir dos dados originais como propôs Levoy.

4 O Algoritmo Básico

O algoritmo que propomos aqui inclui várias otimizações nos passos básicos do algoritmo de *ray casting*

original [Seixas et al (1994)].

A primeira otimização faz o lançamento dos raios através da determinação de pontos de referência no espaço do objeto. O caminhamento na imagem é efetuado através de incrementos previamente calculados neste plano de referência. Esta técnica é mais simples do que a proposta por Yagel e Kaufman (1992), pois utiliza dois planos auxiliares de visualização.

As outras otimizações dizem respeito à detecção da interseção e ao caminhamento pelo raio através do volume. Para tal, foram utilizadas, respectivamente, extensões dos algoritmos clássicos de Cyrus–Beck para *clipping* e de Bresenham para discretização de retas. Além disso, o caminhamento pelo raio é interrompido quando se atinge uma superfície opaca, ou quando as contribuições deixam de ser significativas em relação aos valores já calculados.

Para obter uma visualização rápida, são utilizadas estruturas de dados hierárquicas, do tipo "pirâmides". Neste tipo de estrutura, decompõe-se o volume de N^3 *voxels* em *log N* volumes, onde o primeiro é o volume original, o segundo é uma média de 2×2×2 *voxels* do original, resultando num volume com 1/8 da resolução. Este processo se repete até serem definidos *log N* volumes [Laur–Hanrahan (1991)]. Uma vez construída a pirâmide, a visualização é feita percorrendo a pirâmide no nível correspondente à resolução desejada.

Para evitar o desperdício de memória e obter uma maior interatividade com o usuário, apresentamos a imagem em refinamentos sucessivos [Hollasch (1992)]. A idéia é percorrer uma grade regular sobre o plano de visualização, diminuindo a cada etapa o passo na grade. Assim, a imagem é mostrada rapidamente em baixa resolução, sendo refinada até a resolução final. Como ilustra a Figura 2, cada *pixel* da imagem é calculado uma única vez. Portanto, o tempo de cálculo da cor dos *pixels* permance inalterado. Assim, obtemos o mesmo efeito pretendido quando na utilização de "pirâmides", sem entretanto, nenhuma memória adicional. Este *feedback* imediato permite que o usuário interrompa o processo de visualização para, por exemplo, escolher outro ponto de vista.

Essas otimizações contribuem para diminuir o tempo da parte básica do algoritmo de *ray casting*; através de um mapeamento simples entre os valores escalares do volume e cor e opacidade (Figuras 4 e 5), já é possível obter imagens razoáveis, como a da Figura 3. Temos, assim, um bom método de *preview*. Entretanto, essas otimizações não contribuem para aumentar a qualidade da imagem. Para isso, usamos uma variação 3D do algoritmo de detecção de bordas em imagens, descrita a seguir.

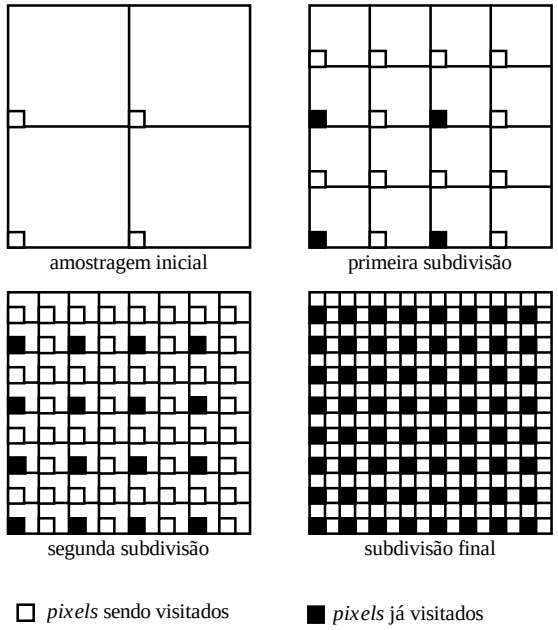

Figura 2: Refinamento progressivo da imagem.

5 Detecção de Bordas em Volumes

Tradicionalmente, as "bordas" em imagens bi-dimensionais correspondem a discontinuidades no gradiente [Gonzalez–Wintz (1987)]. Os algoritmos tradicionais para detecção de bordas em imagens usam aproximações discretas do gradiente, como a por diferenças centrais, também usada em visualização volumétrica. As bordas ocorrem nos *pixels* aonde o módulo do gradiente ultrapassa um dado limiar.

A inclusão de detecção de bordas num algoritmo de *ray casting* para visualização volumétrica é feita associando a cada *voxel* uma cor proporcional ao ângulo entre o gradiente e a direção da luz. Como no caso de imagens, as bordas ocorrem nos *voxels* aonde o módulo do gradiente ultrapassa um limiar. Entretanto, como veremos a seguir, é necessário combinar essa detecção de bordas com uma seleção de *voxels* por densidade, a fim de evitar ruídos nos dados originais.

Nos testes descritos a seguir, consideramos também as duas alternativas clássicas no cálculo do módulo do gradiente: em vez da "norma 2" (raiz quadrada da soma dos quadrados das componentes), pode-se usar a "norma 1" (soma dos valores absolutos das componentes) ou a "norma infinito" (maior valor absoluto das componentes) [Gonzalez–Wintz (1987)].

6 Resultados Comparativos

Para verificar a viabilidade do método proposto, os resultados obtidos foram comparados com os resultados gerados pela biblioteca *VolPack*, que é baseada no

algoritmo de Lacroute–Levoy (1994). As comparações foram feitas em uma estação de trabalho IBM RS/6000 modelo 520 utilizando um conjunto de dados MRI de uma cabeça humana com 128×128×84 *voxels*, distribuído juntamente com *VolPack*.

Para obter os valores característicos dos algoritmos, não foi utilizado nenhum sistema gráfico ou *hardware* específico. Assim, ao invés de exibir diretamente a imagem resultante, geramos o resultado na resolução final (128×128) num arquivo no formato do *VolPack*, de modo que a exibição da imagem não interfira na comparação.

Foram efetuadas as medições de tempo para a geração rápida da imagem de baixa qualidade (*preview*) e para a geração da imagem final com alta qualidade. Embora a técnica de refinamento sucessivo seja adequada para *preview* interativo, ela não foi utilizada nestes testes para não prejudicar a comparação de tempos.

Os resultados obtidos são apresentados a seguir na forma de tabelas, acompanhadas das respectivas imagens. A imagem gerada com a biblioteca *VolPack* foi feita por geração e renderização direta do volume de dados, sem o uso de *octree* ou pré-classificação do volume (Figura 9).

A Tabela 1 mostra os tempos obtidos com o algoritmo básico, sem a detecção de bordas, para a sua utilização como método de *preview*.

leitura dos dados	3 s
rendering	16 s
gravação da imagem	<1 s
tempo total	19 s

Tabela 1: Sem detecção de borda.

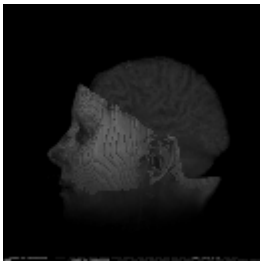

Figura 3: Sem detecção de borda (*preview*).

A imagem da Figura 3 foi obtida com um mapeamento direto para cor e opacidade a partir do valor das densidades, sem uso de algoritmos de iluminação ou qualquer outro método para melhoria da imagem. Os mapeamentos de cor e opacidade utilizados para a geração dessa imagem são ilustrados nas Figuras 4 e 5.

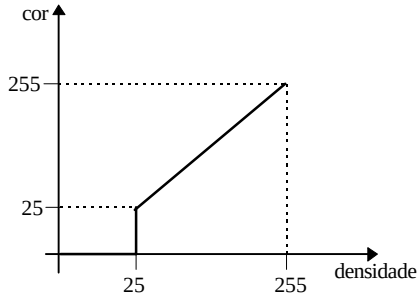

Figura 4: Mapeamento densidade×cor.

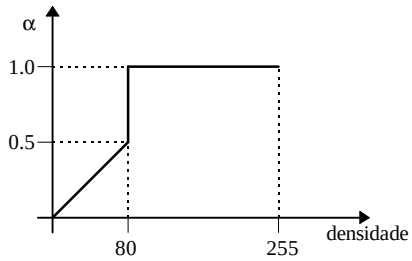

Figura 5: Mapeamento opacidade×cor.

A escolha dos mapeamentos de cor e opacidade depende fortemente do conjunto de dados e da necessidade de observação específica de alguma faixa de valores. Assim, a determinação dos mapeamentos, requer um grande conhecimento das características dos dados, uma vez que um mapeamento errado pode acarretar numa grande alteração da imagem, produzindo artefatos ou omitindo detalhes importantes.

A Figura 6 mostra uma imagem obtida utilizando o próprio algoritmo de detecção de bordas para determinar a cor nos *voxels*, sem usar a densidade em nenhum cálculo ou mapeamento de cor e opacidade.

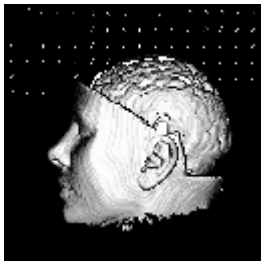

Figura 6: Com detecção de borda e sem mapeamento de densidade para cor e opacidade.

Os tempos obtidos para esta forma de visualização são mostrados na Tabela 2.

leitura dos dados	3 s
cálculo do gradiente	16 s
rendering	24 s
gravação da imagem	<1 s
tempo total	43 s

Tabela 2: Com detecção de borda e sem mapeamento de densidade para cor e opacidade.

Note que apareceram alguns pontos espúrios na imagem devido a altos gradientes em áreas de ruído, apesar de utilizarmos um valor de *threshold* mínimo para o módulo do gradiente. Para limiares maiores, partes da imagem também iriam ser desprezadas. Assim, é necessário que utilizar conjuntamente as informações de densidade e gradiente, e seus respectivos mapeamentos em cor e opacidade, como mostra a Figura 7.

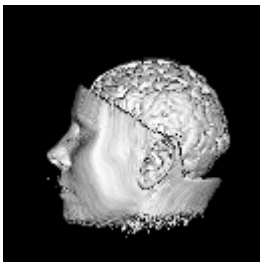

Figura 7: Com detecção de borda e com mapeamento de densidade para cor e opacidade.

Nota-se, agora, uma melhoria considerável da qualidade da imagem. As contribuições provenientes do ruído, com um alto gradiente, foram agora desprezadas pelo mapeamento da sua densidade baixa em uma cor e opacidade desprezíveis em relação à contribuição mínima de cor dos *voxels*. Os tempos obtidos com esta técnica são mostrados na Tabela 3.

leitura dos dados	3 s
cálculo do gradiente	16 s
rendering	25 s
gravação da imagem	<1 s
tempo total	44 s

Tabela 3: Com detecção de borda e mapeamento de densidade para cor e opacidade.

A Tabela 4 mostra os tempos finais obtidos pela utilização do método proposto com *depth-cueing* na visualização.

leitura dos dados	3 s
cálculo do gradiente	16 s
rendering	26 s
gravação da imagem	<1 s
tempo total	45 s

Tabela 4: Com detecção de borda, mapeamento de densidade para cor e opacidade e *depth-cueing*.

Para comparação visual, apresentamos as duas imagens finais geradas pelo nosso método (Figura 8) e pelo algoritmo de Lacroute–Levoy implementado na biblioteca *VolPack* (Figura 9). Como mostra a Tabela 5, o custo computacional dos dois métodos é praticamente o mesmo.

	tempo total
detecção de bordas	45 s
biblioteca VolPack	44 s

Tabela 5: Comparação final.

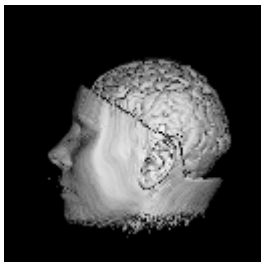

Figura 8: Com detecção de borda e *depth-cueing*.

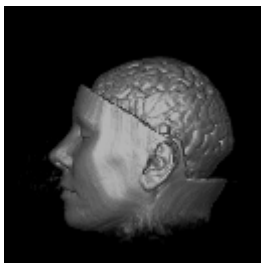

Figura 9: Imagem gerada pela bilioteca *VolPack.*

Os resultados obtidos mostram que o tempo gasto no cálculo do gradiente é bastante alto. Infelizmente, para todos os métodos citados, não é possível obter imagens com qualidade sem o cálculo do gradiente. Para diminuir este tempo, experimentamos as duas alternativas clássicas à norma 2: a "norma 1" e a "norma infinito". No entanto, apesar da diminuição do tempo de 40% na primeira aproximação e de 36% na segunda,

houve uma perda da qualidade da imagem (Figuras 10 e 11).

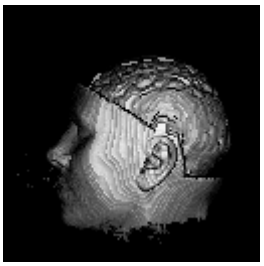

Figura 10: Aproximação com a "norma 1".

Figura 11: Aproximação com a "norma infinito".

7 Conclusões

A utilização de algoritmos de visualização volumétrica tem se mostrado uma ótima técnica de exploração de dados, principalmente dados não estruturados sobre modelos geométricos. No entanto, os resultados de métodos novos, ou otimizações nos métodos antigos, ainda são necessárias para que permitir uma manipulação interativa de dados volumétricos.

O método apresentado aqui é simples de implementar e mostra-se apropriado para a visualização volumétrica, como comprovam os tempos e as imagens obtidas. Além disso, a presença de uma forma rápida de visualização (*preview*) possibilita que a exploração dos dados volumétricos seja feita de uma forma mais interativa. Esta característica, aliada à técnica de refinamento sucessivo para exibição, permite uma forma quase interativa de manipulação e visualização de dados volumétricos.

Agradecimentos

A motivação para este estudo teve origem no âmbito do convênio CENPES/Petrobrás e teve suporte parcial do projeto temático do CNPq GEOTEC - Geo-processamento: Sistema e Técnicas.

Referências

R. Drebin, L. Carpenter, P. Hanrahan, "Volume Rendering", Computer Graphics 22 (1988) 65-74 (Proceedings of SIGGRAPH'88).

T. Elvins, "A Survey of Algorithms for Volume Visualization", *SIGGRAPH'92 course notes 1*, 3.1–3.14.

J. Foley, A. van Dam, S. Feiner, J. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, 1990.

R. C. Gonzalez, P. Wintz, "Digital Image Processing", Addison-Wesley, 1987.

S. Hollasch, "Progressive Image Refinement via Gridded Sampling", in: D. Kirk (ed.), *Graphics Gems III*, Academic Press, 1992, 353–361.

A. Kaufman, W. Lorensen, R. Yagel, "Volume Visualization: Algorithms and Applications", *Visualization'94 course notes 1*.

P. Lacroute, M. Levoy, "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation", *Proceedings of SIGGRAPH'94*, 451–458.

D. Laur, P. Hanrahan, "Hierarquical Splatting: A Progressive Refinement Algorithm for Volume Rendering", *Computer Graphics* **25** (1991) 285–288 (Proceedings of SIGGRAPH'91).

M. Levoy, "Volume Rendering: Display of Surfaces from Volume Data", *IEEE Computer Graphics and Applications* **8** (1988) 29–37.

M. Levoy, "Efficient Ray Tracing of Volume Data", *ACM Transactions on Graphics* **9** (1990) 245–261.

M. Levoy, "A Taxonomy of Volume Visualization Algoritms", *SIGGRAPH'90 course notes 11*, 6–12.

P. Sabella, "A Rendering Algorithm for Visualizing 3D Scalar Fields", *Computer Graphics* **18** (1988) 51–58 (Proceedings of SIGGRAPH'88).

R. B. Seixas, M. Gattass, L. H. de Figueiredo, L. F. Martha, "Otimização do Algoritmo de Ray-Casting para Visualização de Tomografias", *Caderno de Comunicações do VII SIBGRAPI* (1994) 5–8.

L. Sobierajski, A. Kaufman, "Volumetric Ray Tracing", *Proceedings of Visualization'94*, 11–18.

C. Upson, M. Keeler, "V-Buffer: Visible Volume Rendering", *Computer Graphics* **22** (1988) 59–64 (Proceedings of SIGGRAPH'88).

L. Westover, "Footprint Evaluation for Volume Rendering", *Computer Graphics* **24** (1990) 367–376.

R. Yagel, A. Kaufman, "Template-Based Volume Viewing", *Proceedings of Eurographics'92*, 153-167.

M. Zuffo, R. Lopes, "A High Performance Direct Volume Rendering Pipeline", *Anais do VII SIBGRAPI* (1994) 241–248.

"VolPack User's Guide", Laboratório de Computação Gráica, Universidade de Stanford, `ftp://www-graphics.stanford.edu:/software/volpack`