Otimização do Algoritmo de Ray Casting para Visualização de Tomografias

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

[1] Pontifícia Universidade Católica - PUC-RJ
Laboratório de CAD Inteligente - ICAD
Rua Marquês de São Vicente, 225
22453 Rio de Janeiro, RJ, Brasil
(tron,gattass,lhf,lfm)@icad.puc-rio.br

[2] Laboratório Nacional de Computação Científica - LNCC
Coodenação de Desenvolvimento de Software - CDS
Rua Lauro Müller, 455
22290 Rio de Janeiro, RJ, Brasil
tron@lncc.br

Abstract. Ray Casting is a useful volume visualization technique that has a high computational cost. This work proposes some optimization procedures of the Ray Casting algorithm, originally proposed by Levoy, to improve its efficiency when applied to medical images, mainly from computer tomography (CT). These procedures are currently being implemented for further verification.

Introdução

A aplicação de técnicas de Computação Gráfica em áreas que necessitam de um alto poder computacional e possuem uma grande massa de dados, tais como Engenharia, Metereologia, Medicina e Biologia, tem sido chamada de Visualização Científica.

Atualmente, uma das áreas que mais tem utilizado estas técnicas é a visualização de imagens médicas, como as obtidas por Tomografia por Raios X (CT), Ressonância Magnética (MRI), Emissão de Pósitron (PET) e Emissão de Fóton (SPECT).

No caso das tomografias, são geradas imagens de 256x256 *pixels*, denominadas *slices*, de toda a extensão a ser analisada. Estas fatias são, normalmente, dispostas paralelamente a um dos planos coordenados (Figura 1).

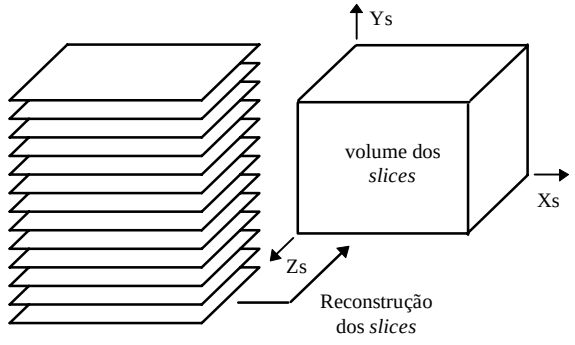

Figura 1: Reconstrução dos *slices.*

Para os algoritmos de visualização, o volume das fatias é representado por uma grade de paralelepípedos alinhados com os eixos, denominados *voxels*. Para cada *voxel*, é atribuído um valor de densidade do material correspondente - ar, gordura, pele, tecido, osso, etc. A esta densidade, é associada uma cor e uma opacidade.

Este trabalho propõe alguns procedimentos para otimizar o algoritmo de *Ray Casting* utilizado para visualização volumétrica. Estes procedimentos estão atualmente em fase de implementação para posterior verificação.

Algoritmo de Ray Casting

Uma das técnicas mais utilizadas para a visualização volumétrica é o *Ray Casting*. Esta técnica permite a visualização de pequenos detalhes internos ao volume, tem um bom controle de transparência removendo trivialmente as partes escondidas atrás de partes definidas como opacas, e visualiza o volume a partir de qualquer direção.

Ray Casting foi apresentado por Levoy (1988), por Debrin, Carpenter e Hanrahan (1988), e novamente por Levoy (1990), já com várias otimizações, descritas mais adiante.

A idéia básica do algoritmo é o "lançamento" de raios que partem da posição do observador, através de cada *pixel* do plano de visualização, para o volume

dos *slices*. Quando a projeção é paralela, a direção destes raios é a da normal a este plano (Figura 2). Para cada raio, integram-se as contribuições dos *voxels* interceptados por ele, determinando assim a cor do *pixel*.

O cálculo da contribuição da cor e da transparência de cada voxel baseia-se no valor da densidade e numa estimativa do gradiente da iso-superfície correspondente (∇f(i)); este gradiente corresponde a normal da iso-superfície (N(i)) que tem o valor do voxel. A estimativa do gradiente é geralmente feita antes do lançamento dos raios, por operadores locais do tipo diferenças finitas [Levoy (1988)]. Assim, a normal é calculada por:

$$N(i) = \frac{\nabla f(i)}{|\nabla f(i)|},$$

onde o gradiente é dado por:

$$\nabla f(i) = \nabla f(i,j,k) \approx \left(\frac{1}{2}\big(f(i+1,j,k) - f(i-1,j,k)\big), \frac{1}{2}\big(f(i,j+1,k) - f(i,j-1,k)\big), \frac{1}{2}\big(f(i,j,k+1) - f(i,j,k-1)\big) \right).$$

Para o cálculo da contribuição do *voxel* na cor do *pixel*, utilizam-se modelos de iluminação clássicos, como o de Phong [Foley (1990)]. Em outras palavras, o volume é tratado como um gel iluminado com iso-superfícies determinadas para cada *voxel* [Levoy (1990)].

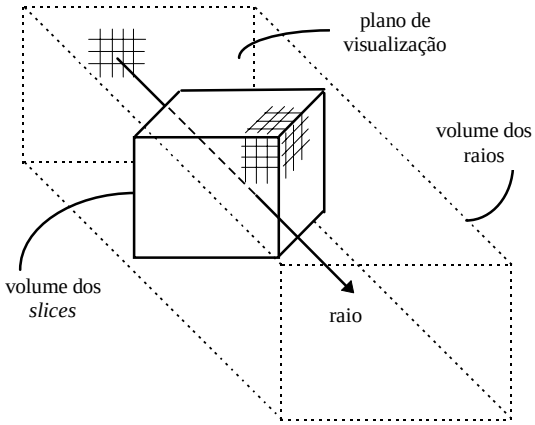

Figura 2: Técnica de *Ray Casting*.

O algoritmo assim descrito consome muita memória e tem alto custo computacional. Por isto, Levoy (1990) apresentou duas otimizações. A primeira substitui a enumeração exaustiva do volume por uma enumeração hierárquica espacial, onde *voxels* de valores semelhantes são agrupados em células. A segunda otimização refere-se à terminação do raio de uma forma adaptativa, onde interrompe-se o cálculo das contribuições quando estas não forem mais significativas.

Note-se que, tendo que armazenar densidade, cor (R, G, B) e opacidade para cada *voxel*, um volume de 256×256×256 requer quase 90 Mbytes de memória. Para contornar este problema, os cálculos podem ser feitos em tempo de lançamento dos raios, diretamente sobre o valor da densidade, resultando numa redução para 16.7 Mbytes. Para que este cálculo não fique muito lento, pode-se adotar um modelo simples que associa a cada valor de densidade uma cor e uma transparência através de uma tabela.

A partir destas simplificações, identificam-se alguns passos do algoritmo de *Ray Casting* passíveis de otimização:

- Lançamento dos raios;
- Determinação dos segmentos dos raios que interceptam o volume dos *slices*;
- Acumulação das contribuições de opacidade e cor ao longo do raio;
- Exibição do plano de visualização durante a sua construção.

A seguir, decreve-se as otimizações obtidas em cada um destes pontos.

Lançamento dos Raios

Apesar de a referência do raio ser o *pixel* (espaço da tela), é mais interessante a determinação de pontos de referência no espaço do objeto e se utilizar de incrementos, previamente calculados, para o caminhamento sobre a cena.

Este procedimento é feito pela utilização de um plano de visualização frontal e um traseiro, paralelos ao plano da tela e descritos no sistema de coordenadas do objeto. Os raios são lançados de um plano para o outro, em função dos seus pontos extremos, conforme a Figura 3.

Os incrementos são então calculados em função do tamanho do volume de raios, em *pixels*, de X, Y e Z, segundo as fórmulas a seguir, onde R_0 identifica a direção de lançamento do raio inicial. A partir deste, os incrementos para os demais raios são calculados de forma proporcional, referenciados como $\Delta_X R$, $\Delta_Y R$ e $\Delta_Z R$:

$$R_0 = P_{lbf} - P_{lbn},$$

$$\Delta_X R = \frac{\left(P_{rbn} - P_{lbn}\right)}{sizeX},$$

$$\Delta_Y R = \frac{\left(P_{ltn} - P_{lbn}\right)}{sizeY},$$

$$\Delta_Z R = \frac{\left(P_{lbf} - P_{lbn}\right)}{sizeZ}.$$

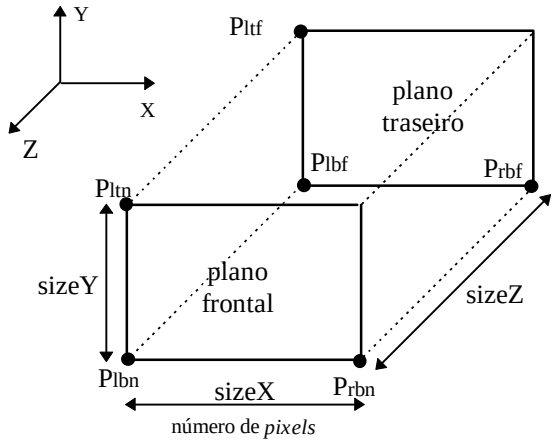

Figura 3: Caminhamento dos raios.

Cyrus-Beck

Para a determinação da interseção dos raios com o volume dos *slices*, é utilizado um algoritmo de *clipping* 3D. Como a maioria dos raios intercepta o volume dos *slices*, optou-se pela utilização do algoritmo de Cyrus-Beck [Foley (1990)], pois este é mais eficiente na determinação dos pontos de interseção.

Para permitir uma manipulação flexível pelo usuário, o volume de raios deve ser rotacionado, expandido e/ou contraído. Neste ambiente, torna-se fundamental descartar eficientemente as partes dos raios que não atingem o volume dos *slices*.

O algoritmo de Cyrus-Beck se baseia na determinação do parâmetro *t* da interseção do raio com a superfície do volume dos *slices*, e na classificação deste ponto como potencialmente entrando ou potencialmente saindo, de acordo com a superfície de interseção [Foley (1990)].

O valor de *t* é dado por

$$t = \frac{num}{den} = \frac{-N_i \cdot \left[P_0 - P_i\right]}{N_i \cdot \left[P_1 - P_0\right]}.$$

onde N_i é a normal da face, P_0 é o ponto inicial do raio, P_1 o ponto final do raio e P_i é um ponto arbitrário, a ser classificado como "potencialmente entrando" ou "potencialmente saindo".

Desta forma, se *den > 0*, então tem-se o caso "potencialmente saindo"; se *den < 0*, tem-se o caso "potencialmente entrando".

Com isso, consegue-se determinar todos os pontos de interseção, conforme a Tabela 1.

FACE	PONTO DE INTERSEÇÃO
LEFT	$t = \frac{x_0 - x_{min}}{-dx}$
RIGHT	$t = \frac{x_{max} - x_0}{dx}$
BOTTOM	$t = \frac{y_0 - y_{min}}{-dy}$
TOP	$t = \frac{y_{max} - y_0}{dy}$
NEAR	$t = \frac{z_0 - z_{min}}{-dz}$
FAR	$t = \frac{z_{max} - z_0}{dz}$

Tabela 1 : Tabela do cálculo de interseções.

Na Tabela acima, *Xmin*, *Xman*, *Ymin*, *Ymax*, *Zmin* e *Zmax* representam as dimensões, mínimas e máximas, do volume dos *slices*; X_0, Y_0 e Z_0, são as coordenadas do ponto de partida do raio; e *dx*, *dy* e *dz* são os tamanhos em *X*, *Y* e *Z* do ponto inicial ao ponto final do raio (do plano frontal para o traseiro).

Bresenham

Uma vez descobertos os pontos de interseção do raio com o volume dos *slices*, o algoritmo procede percorrendo o raio e acumulando a contribuição de cada *voxel* pelo qual o raio passa. A forma usual é percorrer o raio em pontos de amostragem equidistantes. No entanto, este procedimento possui como inconveniente a determinação do *voxel* correspondente a cada amostragem.

Para contornar tal inconveniente, propõe-se a utilização do algoritmo de Bresenham [Foley (1990)], originalmente desenvolvido para desenhar linhas retas em um dispositivo matricial, permitindo uma forma incremental de caminhamento no raio, além de usar somente aritmética inteira. Desta forma, evita-se o problema de localização e garante-se que, para um mesmo raio, não existem duas amostragens para um mesmo *voxel*.

Refinamento Progressivo

Como última proposta de otimização, a técnica de refinamento progressivo é utilizada neste caso para que o usuário possa ter rapidamente um esboço da

imagem final. Embora não seja estritamente uma técnica de otimização, ela propicia um *feed-back* imediato ao usuário, que pode interromper o processo para escolher outro ponto de vista [Hollasch (1992)].

Quando o processo de exibir um *pixel* é suficientemente rápido, a ordem dos *pixels* não é importante. Entretanto, se o processo de exibição for lento ou se a resolução da imagem for alta, o tempo de exibição pode ser muito alto, o que é tedioso para o usuário.

A idéia do algoritmo de refinamento progressivo é percorrer uma grade regular sobre a área da imagem, diminuindo a cada vez, o passo na grade, como ilustra a Figura 4.

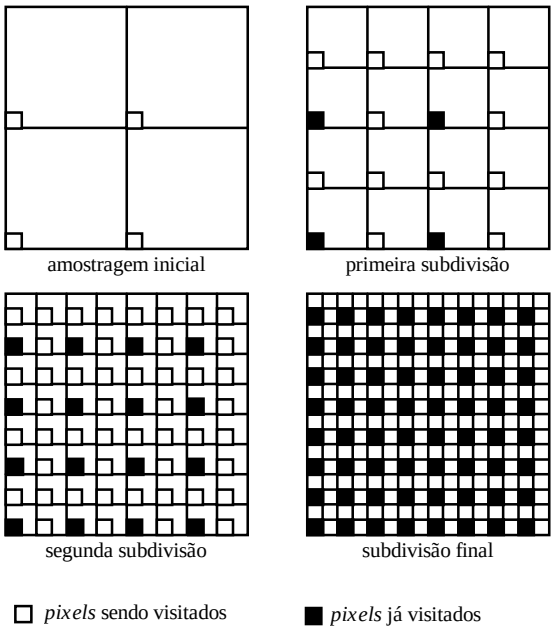

Figura 4: Refinamento progressivo da imagem.

A imagem é mostrada como se tivesse com baixa resolução, sendo refinada até a resolução final da imagem. Cada *pixel* é examinado uma única vez. O único problema desta técnica é quando a exibição de áreas preenchidas é muito mais lenta do que a exibição de um *pixel*, o que a tornaria inviável.

Conclusões

A utilização de métodos de otimização é extremamente importante para visualizar imagens médicas, devido à grande quantidade de dados com que se trabalha e à complexidade computacional dos algoritmos de visualização.

O algoritmo de *Ray Casting* tem-se mostrado bastante adequado para este tipo de visualização, mesmo quando implementado através de *scan-line* tradicional. A vantagem de se utilizar a técnica de refinamento progressivo é que o usuário já tem uma idéia da imagem final, podendo interromper o processo em caso de posicionamento incorreto do plano de visualização, ou outro tipo de erro.

A utilização do algoritmo de Cyrus-Beck deve diminuir consideravelmente o tempo necessário para a identificação dos pontos de interseção do raio com o volume dos *slices*. Como resultado parcial já se verificou também, que esta identificação depende muito pouco do posicionamento do volume dos *slices*.

O algoritmo de Bresenham é utilizado para diminuir o tempo gasto, quando tem-se que percorrer a interseção do raio com o volume dos *slices*, computando as contribuições dos *voxels*. Resultados parciais também indicam que o percentual de ganho é mínimo, uma vez que este procedimento é bastante afetado pelo posicionamento do volume dos *slices*, visto que uma maior extensão tem que ser computada para alguns ângulos.

Na fase atual de desenvolvimento desta pesquisa já se pode observar que os algoritmos ainda tem que ser bastante melhorados para permitir visualização de imagens médicas em tempo real, e a sua utilização em diagnósticos e tratamentos.

Referências

R. Drebin, L. Carpenter, P. Hanrahan, Volume Rendering, *Computer Graphics* **22** (1988) 65–74.

T. Elvins, A Survey of Algorithms for Volume Visualization, *ACM SIGGRAPH* (1992) 3.1–3.14 (course notes 1).

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

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

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, *ACM SIGGRAPH* (1990) 6–12 (course notes 11).

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

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

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