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. In this work we are looking for optimization methods to minimize computational time when applied in visualization of medical images, mainly computer tomography (CT).

Introdução

Uma das técnicas mais utilizadas para a visualização volumétrica é o *Ray Casting*. No entanto, esta técnica tem um custo computacional muito alto. Este trabalho procura otimizar o algoritmo na sua aplicação em visualização de imagens médicas, principalmente Tomografias.

Este algoritmo foi apresentado em 1988 por [Levoy (1988)] e [Drebin (1988)] e, depois com já com várias otimizações em [Levoy (1990)].

Algoritmo de Ray Casting

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, Metereológia, 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, tais como 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. Após isso, é utilizada uma técnica de visualização volumétrica para a construção tri-dimensional da imagem.

Cada *pixel* possui uma cor e uma opacidade relacionadas com a densidade do material - ar, gordura, pele, tecido, osso, etc.

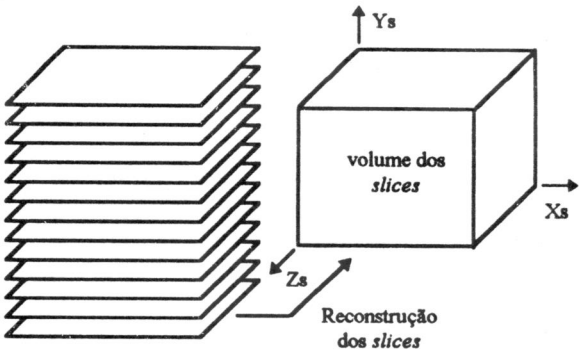

Figura 1: Reconstrução dos *slices*

O algoritmo de *Ray Casting* é bastante utilizado para a visualização destas imagens devido às suas vantagens :

- O volume pode ser visualizado em qualquer direção;
- A remoção de superfícies escondidas é feita de forma trivial;
- Obtem-se um maior detalhamento com uso de cor e transparência.

O algoritmo "lança" raios paralelos de cada *pixel* do plano de visualização para o volume dos *slices* (figura 2). Computa-se ao longo do raio os valores de

cor e opacidade.

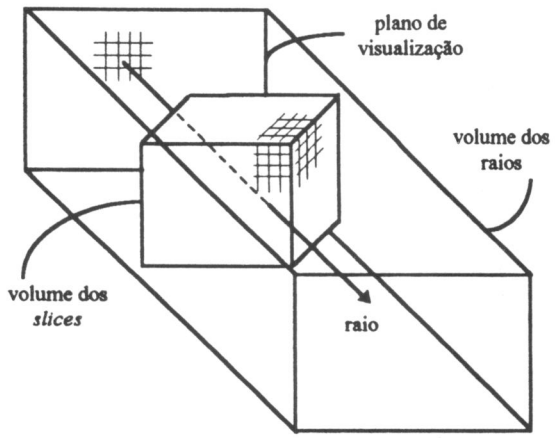

Figura 2: Técnica de *Ray Casting*

Assim, podemos identificar os seguintes pontos do algoritmo de *Ray Casting* passíveis de otimização:

1.Lançamento dos raios;
2.Determinação dos raios que interseptam o volume dos *slices*;
3.Acumulação das contribuições de cor e opacidade ao longo do raio;
4.Exibição do plano de visualização.

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

Lançamento dos Raios

Apesar da 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.

Este procedimento é feito pela utilização de um plano de visualização frontal e um traseiro. 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.

$$\vec{R}_0 = \vec{P}_{lbf} - \vec{P}_{lbn}$$

$$\Delta_X \vec{R} = \frac{\left(\vec{P}_{rbn} - \vec{P}_{lbn}\right)}{sizeX}$$

$$\Delta_Y \vec{R} = \frac{\left(\vec{P}_{rtn} - \vec{P}_{ltn}\right)}{sizeY}$$

$$\Delta_Z \vec{R} = \frac{\left(\vec{P}_{lbf} - \vec{P}_{lbn}\right)}{sizeZ}$$

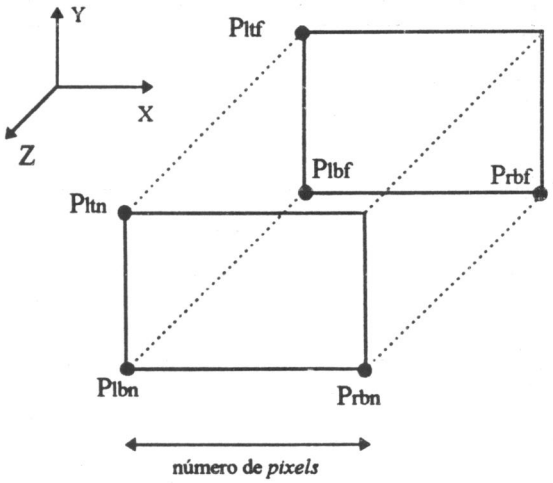

Figura 3: Caminhamento dos raios

Cyrrus-Beck

Para a determinação da interseção dos raios com o volume dos *slices* é utilizado um algoritmo de *clipping* 3D.

Uma primeira opção seria a de se utilizar o algoritmo de Cohen-Sutherland (extensão para 3D). No entanto, ele se mostra mais eficiente para determinar raios que não interseptam o volume dos *slices*.

Por isso, utilizamos o algoritmo de Cyrus-Beck pois se mostra mais eficiente na determinação dos pontos interseção.

Para se permitir uma manipulação flexível pelo usuário, o volume de raios deve ser rotacionado, expandido e/ou contraido. Neste ambiente, torna-se fundamental tratar eficientemente os raios que não atingem o volume dos *slices* e dos trechos de raios que estão fora.

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

$$t = \frac{num}{den}$$

onde :

$$num = -N_i \cdot \left[P_0 - P_i\right]$$

$$den = N_i \cdot \left[P_1 - P_0\right]$$

Desta forma, se $den > 0$ então temos *o potentially leaving* e se $den < 0$ temos o *potentially entering*.

Com isso, conseguimos determinar todos os pontos de interseção, mostrada na tabela a seguir.

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

Tabela 1 : Tabela do cálculo de interseções

Bresenham

Uma vez descoberto os pontos de interseção do raio com o volume dos *slices*, precisamos percorrê-lo, acumulando a contribuição de cada *voxel* pelo qual o raio passar. A forma usual é percorrê-lo utilizando o algoritmo conhecido como *Digital Differential Analyzer* (DDA). No entanto, este algoritmo possui como inconveniente os arredondamentos, o que o torna lento para os propósitos que desejamos.

Para contornar tal inconveniente, utilizamos o algoritmo de Bresenham que usa somente aritmética inteira, evitando funções de arredondamento, permitindo uma forma incremental de caminhamento no raio. Desta forma consegue-se um melhor desempenho quando temos que percorrer um raio que tem uma interseção de grande extensão, de forma que conseguimos usar a contribuição de cada *voxel* uma única vez.

Refinamento Progressivo

A técnica de refinamento progressivo não é na verdade uma técnica de otimização, mas é utilizada neste caso pois mostrou que é uma excelente técnica de visualização para o usuário, que rapidamente já tem um esboço da imagem final.

Quando o processo de exibir um *pixel* é rápido o suficiente, 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.

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.

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 porém desta técnica é quando a exibição de áreas preenchidas é muito mais lenta do que a exibição de um *pixel*.

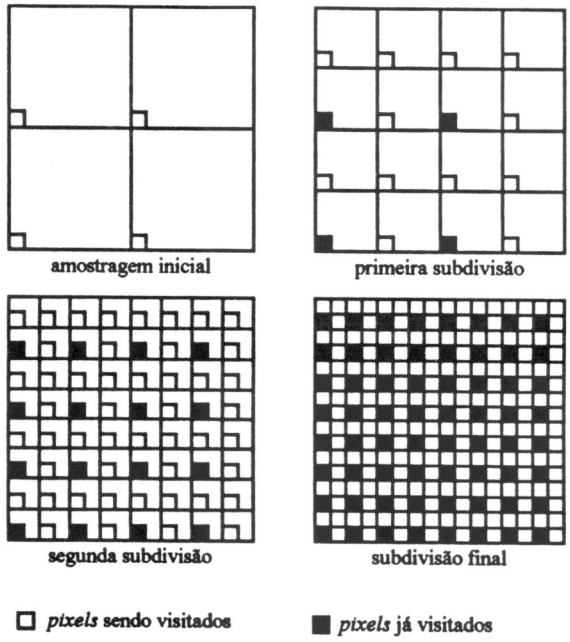

Figura 4: Refinamento Progressivo dos *pixels*

Conclusões

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

O algoritmo de *Ray Casting* se mostrou bastante adequado para este tipo de visualização, até se implementado através de *scan-line* tradicional. A vantagem 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 Cyrrus-Beck diminuiu consideravelmente o tempo necessário para a identificação dos pontos de interseção do raio com o volume dos *slices*. Verificou-se, também, que depende muito pouco do posicionamento do volume dos *slices*.

O algoritmo de Bresenham é utilizado para diminuir o tempo gasto, quando temos que percorrer a interseção do raio com o volume dos *slices*, computando as contribuições dos *voxels*. O percentual de ganho foi 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.

Pode-se notar que os algoritmos ainda tem que ser bastante melhorados para se conseguir uma visualização em tempo real de imagens médicas, permitindo a sua utilização em diagnósticos e tratamentos.

Referências

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

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

D. Kirk (ed.), *Graphics GEM*, Academic Press, 1992.

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 (curse notes - numer 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).