Advanced Engineering Forum *Online: 2011-12-22*
ISSN: 2234-991X, Vols. 2-3, pp 567-572
doi:10.4028/www.scientific.net/AEF.2-3.567
© 2012 The Author(s). Published by Trans Tech Publications Ltd, Switzerland.

Fast Volume Rendering Using a Spherical Index Method for Shading

Yu Wang[1, a], Hong Wang[2,b] and Lu Huang[3]

[1]Sino-Dutch Biomedical and Information Engineering School, Northeastern University, Shenyang 110819, PRC

[2] Northeastern University, POB 319, 110004 Shenyang, China

[3]School of Information Engineering, Dalian Ocean University, Dalian 116023, PRC

[a]w_y@neusoft.com, [b]hongwang@mail.neu.edu.cn

Keywords: Volume rendering, shading model, surface normal

Abstract. Volume rendering can be used to exhibit the shape and volumetric properties of 3-D objects. However, it requires a considerable amount of time to process the large volume of data. In this article we present a speed-up method by pre-computing some data of the shading model. We index voxel surface normal by θ and ϕ in spherical coordinate system. Each voxel surface normal is pre-computed and stored in array of values [θ, ϕ]. Some values of the shading model related to voxel surface normal and light vector are also stored. Each rendering we only need updating these values for shading calculation. We found that our speed-up method can reduce about one fourth of computing time but need less additional memory to store surface normal using this spherical index method.

Introduction

Volume rendering [1] is a flexible technique for visualizing scalar fields with widespread applicability in medical imaging and scientific visualization. It can be used to analyze the shape and volumetric property of three-dimensional objects for medical imaging and computational fluid dynamics. It can display semi-opaque objects and provide better visualization of the surface of an object. Volume rendering is a popular technique for medical imaging used to understand objects by analyzing the large amount of empirical data obtained from measurements or simulations [2].

However, most volume rendering methods that produce effective visualizations are computation intensive [3]. It is very difficult for them to achieve interactive rendering rates for the large amount of volume data. One way to solve the above problems is to parallelize the serial volume rendering techniques onto hardware, for example, distributed memory multicomputers [4]. But hardware speed-up method will make a higher cost and make the system more complicated. Hardware speed-up method will also reduce the expansibility of the system. So software/algorithms speed-up techniques are always required in this field.

Research work of the speed-up techniques based on ray casting has been being done since it was put forward. According to a generally accepted criterion, a "good" ray-shooting algorithm runs in sub-linear time after subquadratic preprocessing and uses linear memory space. Thus instead of implementing the algorithms invented in computational geometry, computer graphics practitioners prefer heuristic ray-shooting speed-up techniques, including, for example,

_ bounding box [5],
_ uniform space subdivision,
_ octree [6,7],
_ BSP or kd-tree [8],
_ ray coherence methods [9,10],
_ ray classification [6,7],
_ Voronoi diagram based space partitioning [11].

These algorithms try to minimize ray-object intersection calculations by building a space partitioning data structure, which has two purposes. On the one hand, this data structure can select only those objects that are in the direction of the ray and can ignore those that are not in this direction and thus can have no intersection with it. From another point of view, it means that the space partition selected for a given ray encapsulates the points of object locations that can be intersected, but is usually larger than that.

The other main feature of these algorithms is that they sort the objects along the ray. It means that candidate objects that are in the ray direction are reported in such an order that if we find an intersection, then we can stop the calculations, because all other intersections are surely behind the found one.

Even if we use these method mentioned above, it is not fast enough to achieve interactive rendering rates on a PC, even on a workstation. In this paper we proposed a spherical index method by pre-computing much of the shading model required data, which can reduce computing time based on the algorithms mentioned above.

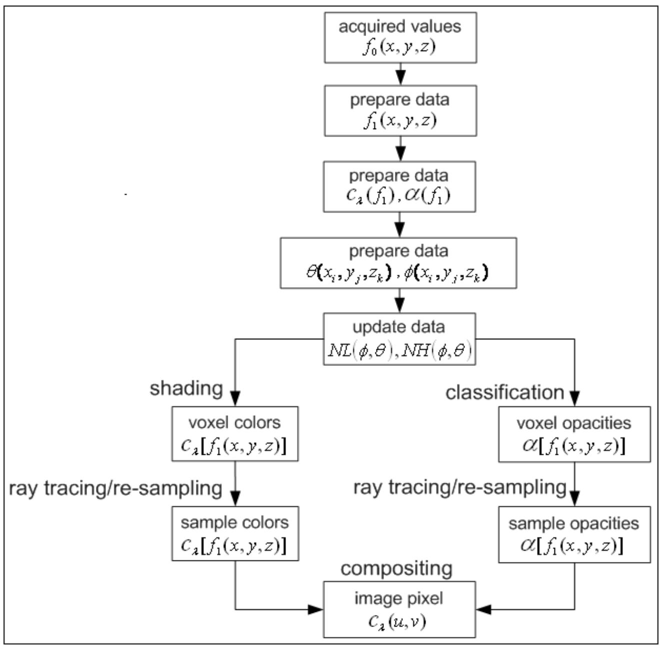
Pre-integrated volume rendering [MHC90, EKE01] is a commonly used technique for improving the quality of volume renderings. Because much of the necessary computation is done in advance, this method can generate high quality images with better performance than heavily super sampling the volume. Unfortunately, the pre-integrated lookup table can take a long time to compute and can not incorporate lighting due to space constraints.

The pre-calculated integral in the lookup table is based only on pairs of scalar values, not normals. Integrating three-component normals into the pre-integrated lookup table requires four values each for the front and back samples(scalar, *Nx*, *Ny*, *Nz*) giving an eight-dimensional lookup table, far too large for a practical implementation.

Using the method proposed in this paper, to store the normals, the additional memory is only the same size of the volume data. So it is possible to integrated normals into the pre-integrated lookup table.

Rendering Pipeline

The pipeline of volume rending used in this paper is summarized in Fig. 1. We begin with an array of acquired values $f_0(x_i, y_j, z_k)$ at voxel location [x_i, y_j, z_k]. In order to render some information we want from the volume data, some data preparation may be done, including correction for nonorthogonal sampling grids in electron density maps, correction for patient motion in CT data, contrast enhancement, interpolation of additional samples, etc..


acquired values $f_0(x,y,z)$
prepare data $f_1(x,y,z)$
prepare data $c_\lambda(f_1), \alpha(f_1)$
prepare data $\theta(x_i,y_j,z_k), \phi(x_i,y_j,z_k)$
update data $NL(\phi,\theta), NH(\phi,\theta)$
shading
classification
voxel colors $c_\lambda[f_1(x,y,z)]$
voxel opacities $\alpha[f_1(x,y,z)]$
ray tracing/re-sampling
ray tracing/re-sampling
sample colors $c_\lambda[f_1(x,y,z)]$
sample opacities $\alpha[f_1(x,y,z)]$
compositing
image pixel $C_\lambda(u,v)$
Fig. 1. Pipeline Overview

The output of the data preparation is an array of prepared values $f_1(x_i, y_j, z_k)$. This array is used as input to the shading model. We also prepare another two array. One is voxel colors $c_\lambda(f_1)$, $\lambda = r, g, b$, the other is voxel opacities $\alpha(f_1)$. The array of prepared values is used as input to one of the classification procedures, yielding an array of voxel opacities $\alpha(f_1)$. In order to reduce shading time we also need preparing the array of voxel normal $\theta(x_i, y_j, z_k)$ and $\phi(x_i, y_j, z_k)$.

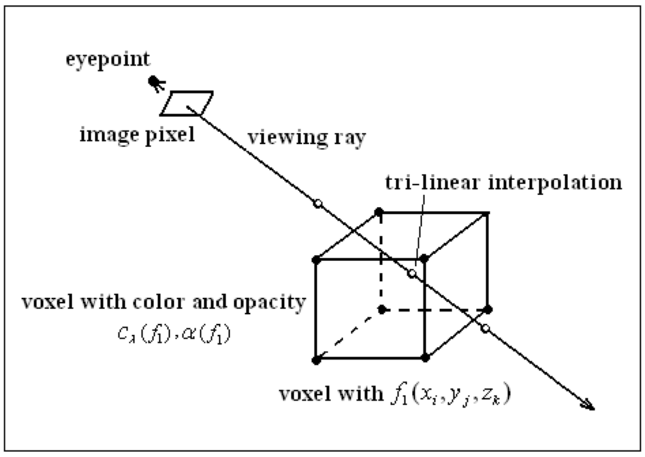

eyepoint
image pixel
viewing ray
tri-linear interpolation
voxel with color and opacity
$c_\lambda(f_1), \alpha(f_1)$
voxel with $f_1(x_i, y_j, z_k)$

Fig. 2. Ray tracing/Resembling steps

Rays are then cast into these two arrays from the observer eye point. For each ray value f_1 of sample voxel at location $[x_i, y_j, z_k]$ is computed by trilinearly interpolating from values f_1 in the eight voxels closest to the sample voxel, as shown in Fig. 2. Then voxel colors $c_\lambda(x_i, y_j, z_k)$ is acquired by looking up array table of prepared values $c_\lambda(f_{1k})$, voxel opacities $\alpha(x_i, y_j, z_k)$ is acquired by looking up array table of prepared values $\alpha(f_{1k})$. Finally, a full opaque background of color $c_{bkg,\lambda}$ is draped behind the dataset and the resample colors and opacities are merged with each other and with the background by compositing in back-to- front order to yield a single color $C_\lambda(u,v)$ for the ray, and since only one ray is cast per image pixel, for the pixel location $[u_m, v_n]$ as well.

The compositing calculations referred to above are simply linear interpolations. Specifically, the color $C_{out,\lambda}(u,v)$ of the ray as it leaves each sample location is related to the color $C_{in,\lambda}(u,v)$ of the ray as it enters and the color $c_\lambda(x_i, y_j, z_k)$ and opacity $\alpha(x_i, y_j, z_k)$ at that sample location by the transparency formula

$$C_{out,\lambda}(u,v) = C_{in,\lambda}(u,v)(1-\alpha(x_i, y_j, z_k)) + c_\lambda(x_i, y_j, z_k)\alpha(x_i, y_j, z_k) \qquad (1)$$

Solving for pixel color $C_{out,\lambda}(u,v)$ in terms of the vector of sample colors $c_\lambda(x_i, y_j, z_k)$ and opacity $\alpha(x_i, y_j, z_k)$ along the associated viewing ray is given by Eq. 2

$$C_\lambda(u_m, v_n) = \sum_{l=0}^{L}\left[c_\lambda(x_i, y_j, z_k)\alpha(x_i, y_j, z_k)\prod_{in=l+1}^{L}(1-\alpha_{in}(x_i, y_j, z_k))\right] \qquad (2)$$

where $c_{0,\lambda}(x_i, y_j, z_k) = c_{bkg,\lambda}$ and $\alpha_0(x_i, y_j, z_k) = 1$.

Shading and Classification

The mapping from acquired data to color provides 3D shape cues but dose not participate in the classification operation using the rendering pipeline presented above. In order to provide a satisfactory illusion of smooth surfaces, a shading model must be selected. The model chosen was developed by Phong: [12]

$$c_\lambda(s_L) = c_{p\lambda}k_{\alpha\lambda} + \frac{c_{p\lambda}}{k_1 + k_2 d(s_L)}\left[k_{d\lambda}(N(s_L)\bullet L) + k_{s\lambda}(N(s_L)\bullet H)^n\right] \qquad (3)$$

where, $c_\lambda(s_L)$: λ'th component of color at voxel location s_L $[x_i, y_j, z_k]$, $\lambda = r, g, b$;

$c_{p\lambda}$: λ'th component of color of parallel light source;

$k_{\alpha\lambda}$: Ambient reflection coefficient for λ'th color component;

$k_{d\lambda}$: diffuse reflection coefficient for λ'th color component;

$k_{s\lambda}$: Specular reflection coefficient for λ'th color component;

n : Exponent used to approximate highlight;

k_1, k_2 : Constants used in linear approximation of depth-cueing;
$d(s_L)$: Perpendicular distance from picture plane to voxel location $s_L[x_i, y_j, z_k]$;
$N(s_L)$: Surface normal at voxel location $s_L[x_i, y_j, z_k]$;
L: Normalized vector in direction of light source;
H: Normalized vector in direction of maximum highlight.

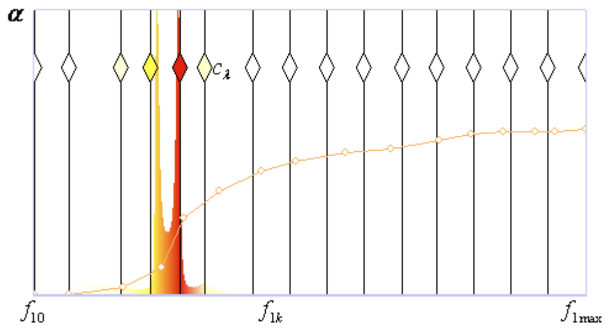


Fig. 3. Mapping arrays definition

The mapping from prepared values f_1 to opacity α and color c_λ performs the essential task of classification. It is not the main point of this paper. So the mapping arrays $c_\lambda(f_{1k})$ and $\alpha(f_{1k})$ are simply defined by user, as shown in Fig. 3.

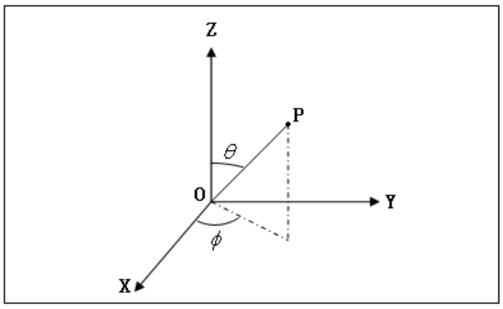
Speeding-up Method

In Eq. 3 presented on section shading and classification, the surface normal at voxel location $s_L[x_i, y_j, z_k]$ is given by Eq. 4

$$N(s_L) = N(x_i, y_j, z_k) = \frac{\nabla f(x_i, y_j, z_k)}{\left|\nabla f(x_i, y_j, z_k)\right|} \tag{4}$$

where the gradient vector $\nabla f(x_i, y_j, z_k)$ is approximated using the operator

$$\nabla f(x_i,y_j,z_k) = [f(x_{i+1},y_j,z_k) - f(x_{i-1},y_j,z_{k-1}), f(x_i,y_{j+1},z_k) - f(x_i,y_{j-1},z_k), \\ f(x_i,y_{j+1},z_k) - f(x_i,y_{j-1},z_k)] \tag{5}$$



Since an array of values $f_1(x_i, y_j, z_k)$ is prepared, we can pre-compute the voxel surface normals $N(s_L)$ at location $s_L[x_i, y_j, z_k]$. Here $N(s_L)$ is indexed by θ and ϕ, which are are defined in Fig. 4. We store voxel surface normals in an array of voxel normal $\theta(x_i, y_j, z_k)$ and $\phi(x_i, y_j, z_k)$. Since a parallel light is used, L is a constant during rendering each time. Furthermore.

Fig. 4. θ and ϕ definition

$$H = \frac{V + L}{|V + L|} \tag{6}$$

where, V: normalized vector in direction of observer. Since an orthographic projection is used, V and H are also constants.

Considering voxel surface normals $N(s_L)$ of the volume data are constants, which are computed from prepared values $f_1(x_i, y_j, z_k)$, $N(s_L) \bullet L$ and $N(s_L) \bullet H$ in Eq. 3 are also constants. So we can pre-compute the values of $N(s_L) \bullet L$ and. The values of $N(s_L) \bullet L$ is stored in an array of $NL(\theta_I,\phi_J)$. The values of $N(s_L)\bullet H$ is stored in an array of $NH(\theta_I,\phi_J)$. So we only need updating array of $NL(\theta_I,\phi_J)$ and array of $NH(\theta_I,\phi_J)$ instead of recomputing the values of $N(s_L)\bullet L$ and $(N(s_L)\bullet H)^n$ at each sampling voxel each time rendering.

Results and discussion

Five patient CT series image data were selected to make a test. The patient volume data info and the result are shown in table 1. The timings were measured on a PC with Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00Hz. From the result we found it can reduce about one fourth of computing time with this method.

Table 1 Result and data info

Patient ID	Volume Data	Computing Time(ms) Normal	Computing Time(ms) Spherical Index
94396	512x512x56	1422	1032
98296	512x512x101	1641	1250
V2462	512x512x61	1593	1203
V1923	512x512x56	1422	1063
85919	512x512x73	1641	1140

Considering the patient volume data type is short, which is 16 bit in C++ language, if integer $I_{\max}$ and $J_{\max}$ of the discrete θ_I and ϕ_J are both less than 255, it needs only a short type array with the same size of the volume data to store voxel normals $\theta(x_i,y_j,z_k)$ and $\phi(x_i,y_j,z_k)$. The first eight bits of a short type value is used to store value of θ_I and the second eight bits is used to store value of ϕ_J. Less additional memory is needed to store surface normal using this kind of spherical index method since three times memory of the volume data is need to store voxel normal with three-component (*Vx*, *Vy*, *Vz*)..

Acknowledgments

The authors would like to thank WangHao for providing CT data used for this research. This work was supported by the National Natural Science Foundation of China under Grant No. 61071057.

References

[1] A. Kaufman (eds.). Volume visualization. IEEE Computer Society Press, 1991.
[2] T. S. Yoo, U. Neumann, H. Fuchs, S. M. Pizer, T. Cullip, J. Rhoades, and R. Whitaker. Direct visualization of volume data. IEEE Computer Graphics & Applications, 12:63–71, 1992.
[3] J. P. Singh, A. Gupta, and M. Levoy. Parallel visualization algorithms: Performance and architectural implications. Computer, 27:45–55, 1994.
[4] C. M. Wittenbrink and A. K. Somani. Permutation warping for data parallel volume rendering. InProceedings of the 1993 Parallel Rendering Symposium, pp. 57–60. San Jose, October 1993.
[5] Hu Ying, Hou Yue，Xu Xin-he. Fast Volume Rendering for Medical Image[C], Proceedings of XI International Congress for Stereology, Beijin, Nov 2003.
[6] A. S. Glassner. Space subdivision for fast ray tracing. IEEE Computer Graphics and pplications, 4(10):15–22, 1984.
[7] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In Andrew S. Glassner, ditor, An Introduction to Ray Tracing, pages 201–262. Academic Press, London,1989.
[8] V. Havran. Heuristic Ray Shooting Algorithms. Czech Technical University, Ph.D. dissertation, 2001.
[9] M. Ohta and M. Maekawa. Ray coherence theorem and constant time ray tracing algorithm. In T. L. Kunii, editor, Computer Graphics 1987. Proc. CG International '87, pages 303–314, 1987.

[10] T. Horv´ath, P. M´arton, G. Risztics, and L. Szirmay-Kalos. Ray coherence between sphere and a convex polyhedron. Computer Graphics Forum, 2(2):163–172, 1992.

[11] G. M´arton. Acceleration of ray tracing via voronoidiagrams. In Alan W. Paeth, editor, Graphics Gems V, pages 268–284. Academic Press, Boston, 1995.

[12] P.Bui-Tuong. Illumination for Computer-Generated Pictures. CACM, June 1975, pages 311-317.