OPTIMIZING COMBINED VOLUME AND SURFACE DATA RAY CASTING

Marcelo Rodrigo Maciel Silva, Isabel Harb Manssour, Carla Maria Dal Sasso Freitas

Instituto de Informática, UFRGS
Caixa Postal 15064 - 91501-970 Porto Alegre, RS
Brazil
{mrms, manssour, carla}@inf.ufrgs.br http://www.inf.ufrgs.br/cg

ABSTRACT

Techniques for simultaneous display of volume data and geometric models have been reported in the literature. These techniques either require conversion from one representation to the other or are based on a hybrid data model. This paper reports on the optimization of the classical ray casting technique for direct volume rendering applied to both volume data and geometric model visualization. The algorithm was implemented as part of an interactive volume visualization tool named RenderVox currently under validation. Images and performance data are presented and discussed.

Keywords: Volume visualization, Hybrid Visualization

1. INTRODUCTION

Volume visualization deals with the problem of generating images from volumetric data with the purpose of revealing and exploring the inner structures of these data. Volumetric data can be obtained either by scanning real objects using a sampling device, as for example CT and MRI scanners, or generated by a simulation program. Moreover, volume data can be produced by a scan conversion technique applied to a 3D geometric model [Kaufm87]. Such 3D voxels are usually stored in a 3D array representing a set of values of discrete sample points obtained from the domain space. In case of volumetric data obtained from 2D images, image processing and reconstruction techniques should be employed in order to obtain the regular set of data.

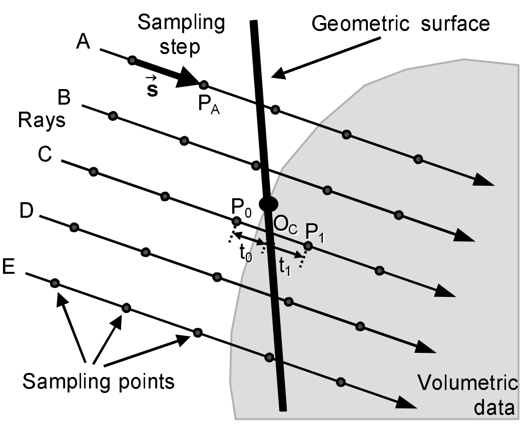
Surface rendering and direct volume rendering are two main categories for visualizing volumetric data. Techniques following these two methodologies have been thoroughly described and employed. Surface rendering techniques rely on segmentation and reconstruction processes to obtain geometric primitives that correspond to the structures of interest inside the volume. Since these geometric primitives can be rendered with traditional computer graphics techniques [Fuchs77; Loren87], they can take advantage of common graphics acceleration hardware to be employed in real-time applications. Direct volume rendering algorithms transform, shade and project 3D voxels into 2D pixel space (see, for example, the work of Levoy [Levoy90a], and Drebin [Drebi88]). Moreover, specific hardware for supporting real-time visualization has also been developed [Günth94; Knitt94; Brady97] and parallel versions of the algorithms have been reported [Ma94].

Although surface-based methods and direct volume rendering algorithms are suited for displaying volumetric data in different situations, or rendering different types of data, there are many applications where volume data should be either displayed, or manipulated along with geometric modeled objects. For example, in surgical planning, surgery simulation, and radiation therapy planning, volume data obtained from 2D images taken with imaging devices need to be rendered with geometric models of prosthesis, surgical instruments, and radiation beams.

Techniques for simultaneous display of volume data and geometric models have been described in the literature [Kaufm90; Levoy90b]. They either require conversion from one representation to the other or are based on a hybrid data model. This paper reports on the optimization of the ray casting technique for direct volume rendering applied to both volume data and geometric model visualization. The technique was implemented as part of an interactive volume visualization tool named RenderVox.

2. BASIC ALGORITHM

In our technique for rendering volume data and geometric objects the first ray is fired into the scene from the central point of a projection plane. The other rays start at adjacent points and are fired following a center-border direction, so we can stop generating rays when the borders of the volume (or the borders of the objects' bounding rectangle) are reached. The sampling points in the volume along a ray are calculated as usual, considering a fixed step on the path determined by the ray. Intersection points between the ray and the geometric models are calculated following Badouel's technique [Badou90], and inserted in the same path so the algorithm can accumulate both color and opacity values along the ray. Both sampling and intersection points are parametrically represented so it's easier to obtain the ordered set of samples in a ray. Color and opacity values for each sample point are obtained by sampling the voxels values or the geometric surface, which is considered as flat-shaded. The parametric representation of sampling and intersection points also allows to correct color and opacity values when an intersection point lies between two volume sampling points.



Sampling and intersection points along rays.
Figure 1

As can be seen in Fig. 1, the ray labeled as C enters the volume and then intercepts the geometric surface at the point O_c. Voxel values are sampled and mapped to colors and opacities considering P_0 as outside the inner structure. In this case, the intersection point with the geometric surface would be considered in front of the volumetric data, and then its color and opacity would superimpose those of the volumetric data. Colors are accumulated in the correct sequence and the opacity is calculated as

$$O_{corrected} = 1 - (1 - O_{sample})^{t/|s|}$$

where O means opacity, $|s|$ is the sample step, $t=t_0$ for point P_0, and t_1 for point P_1.

3. OPTIMIZING THE COMPUTATION OF INTERSECTIONS

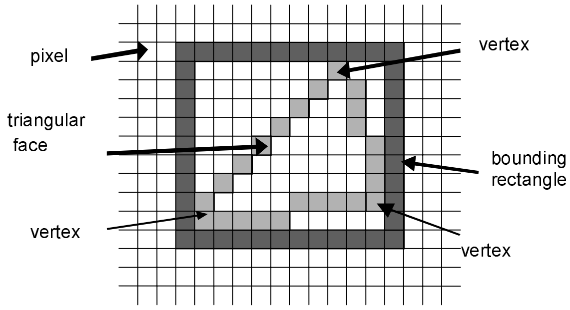
Using bounding boxes, or bounding spheres for each object, or indexing the space with an octree usually do optimize the computation of intersections in classical ray tracing. When using bounding boxes or bounding spheres, each ray is firstly tested against the bounding volumes of objects [Whitt80]. Intersections are then calculated only for objects whose bounding volumes are intercepted by the rays. The use of octrees to index the object space avoids intersection calculation for objects that are in a subspace out of the path of the ray [Glass84].

These techniques proved to be useful in reducing the total processing time in ray tracing. Since in volume rendering projection can be parallel and there are no second-generation rays, we adopt similar approaches: instead of bounding volumes, bounding rectangles are computed for each triangular face of the geometric objects, and instead of subdividing the space in octants, our algorithm subdivide the viewport in cells.

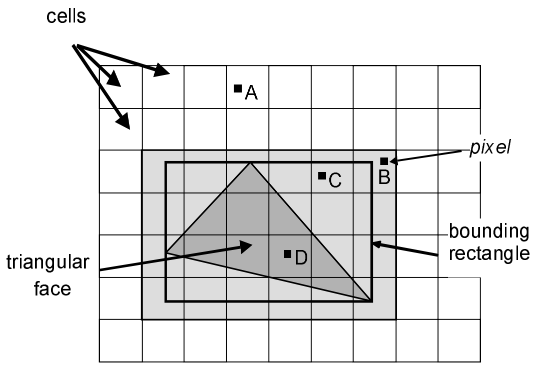
In the first approach, triangular faces are projected into the viewport, their bounding rectangles are determined (Fig. 2), and then intersection points between a ray and each face are calculated only for those rays that are fired from pixels inside the bounding rectangles.

We choose to use bounding rectangles because these elements provide better performance for the point containment test than projected triangles. Also, we do not need to order the bounding rectangles because the parametric representation of intersection and sample points gives their relative position.

In the latter optimization approach, the viewport is subdivided into square cells (Fig. 3) and a list stores pointers to the bounding rectangles, which lie inside each cell. When a ray R is fired from a pixel inside cell C, only the bounding rectangles belonging to the cell C list are tested, in order to determine which faces are likely to be intercepted by the ray. The size of the cells is calculated based on the average size of the bounding rectangles of each face as well as the total size of objects' projection onto the viewport. This approach corresponds to an optimization of the previous one.



Triangular face and its bounding rectangle.
Figure 2



Viewport subdivision.
Figure 3

4. RESULTS AND DISCUSSION

The basic, brute force algorithm (1), the bounding rectangle technique (2) and the subdivided viewport method (3) were executed in an Intel Pentium II 300 MHz, 64 Mb RAM computer, using the geometric models shown in fig. 4a-c. Each head was rendered combined with a brain manually segmented from a 102x161x175 voxels MRI volume (Fig 4d). All the images for these tests were generated with neither flat shading, no ambient light nor specular reflection.

Table 1 presents processing time and other results obtained with the 9,184 faces geometric model (Fig. 4c), while Table 2 shows total processing time for each algorithm executed with each geometric model.

Analysis of Table 1 shows that for these tests the algorithm with bounding rectangles testing was 17 times faster than the brute force algorithm, while that with viewport subdivision was 119 times faster, generating the same image in only 14.7 seconds. From Table 2 we can conclude that processing time of the brute-force algorithm increases faster with the number of faces than the optimized viewport subdivision method. Since the procedures associated with the volumetric part of the hybrid ray casting are the same for all algorithms, the total processing time reduction is only due to the intersection computation optimization. This is easily observed in the second line of Table 1 (processing time without volume sampling).

Table 2 also allows observing that the optimized viewport subdivision algorithm takes only 9% more time to render the combined brain volume and 9,184-faces model, than it takes to render the combination of volume with the simpler model, although complexity (measured in number of faces) increases 1,528%. The same analysis for the brute force algorithm shows that the processing time for this algorithm increases 2,187% to render the more complex combination.

Pre-processing time is very short in both approaches. For example, using the 2,296-faces model, pre-processing time was 0.01 seconds in the bounding rectangle method and 0.08 seconds when subdividing the viewport in 64 cells.

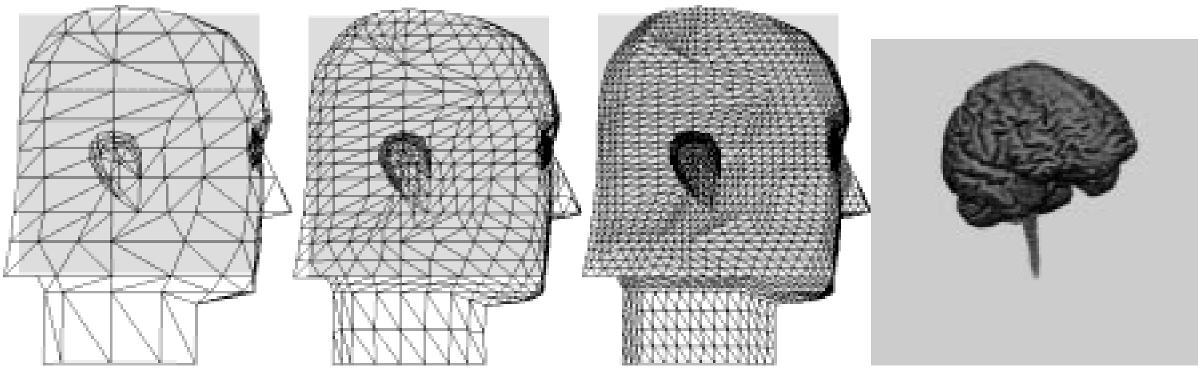
Figure 5 shows three images produced with the hybrid ray casting in RenderVox. The first one corresponds to the volume and 564-faces model used in the above tests. The others were obtained by processing a 113x256x256 CT volume of a head with geometric objects representing schematic radiation beams and a (manually inserted) sphere representing a tumor. In the third example, the parts corresponding to the intersection of each radiation beam with the volume itself are shown in green (see color plate of Figure 5), thus identifying those regions affected by each radiation beam. These images were produced using non-segmented data, i.e., based only on selective classification tables. For better results one can use segmented volumes.

5. FINAL REMARKS

Considering the well-known proposals for simultaneous rendering of volume and surface data [Kaufm90; Levoy90b], the technique reported here is based on optimization approaches for hybrid models different from those already described in the literature. Hybrid ray casting has the disadvantage of repeatedly render volume data even if changes occur only in the position or orientation of geometric objects. For rendering static volume data and dynamic geometric objects, the Z-merging technique [Kaufm90] could be interpreted as a better solution. However, this technique produces two separate images, one for the volumetric part and the other for the geometric objects, and so it does not correctly combine color and opacity data for non-opaque objects. Our approach has not been tested in real-time visualization but a parallel ray casting implementation certainly allow this possibility.

It should be noticed that our basic algorithm is based on parallel projection, as mentioned in section 2. However, perspective projection is needed for navigation techniques employed in virtual endoscopy, for example. For sure, this is easy to incorporate and does not affect the proposed optimizations.

This work is part of a project that intends to provide a framework for surgery simulation based on virtual patients. RenderVox provides many functions for volume visualization based on ray casting, including a simple one for visualizing volumes from different modalities.



(a) (b) (c) (d)

Geometric object modeled with (a) 564 faces, (b) 2,296 faces, (c) 9,184 faces, and (d) manually segmented brain from a MRI volume.

Figure 4

	Brute force	With bounding rectangles	With viewport subdivision[(1)]
Total rendering time	1754.6 s (≅ 29 min)	103.3 s	14.7 s
Time without volume sampling [(2)]	1742.2 s	91.0 s	3.1 s
#pixels x rectangles tests [(3)]	-	1,043,449,344	3,632,535 (0.35%)
#ray x faces tests [(4)]	1,043,449,344	873,061 (0.08%)	873,061 (24.0%)
# intercepted faces [(5)]	166,642 (0.016%)	166,642 (19.1%)	166,642 (19.1%)

(1) The 360x360 pixels viewport was divided in 1024 square cells.
(2) This is the processing time for the geometric model rendering only.
(3) The percentage indicates the number of rectangles tested against a cell relative to the total number of tests in second column.
(4) and (5) Percentage indicates the number of tests returning true relative to the total number of tests.

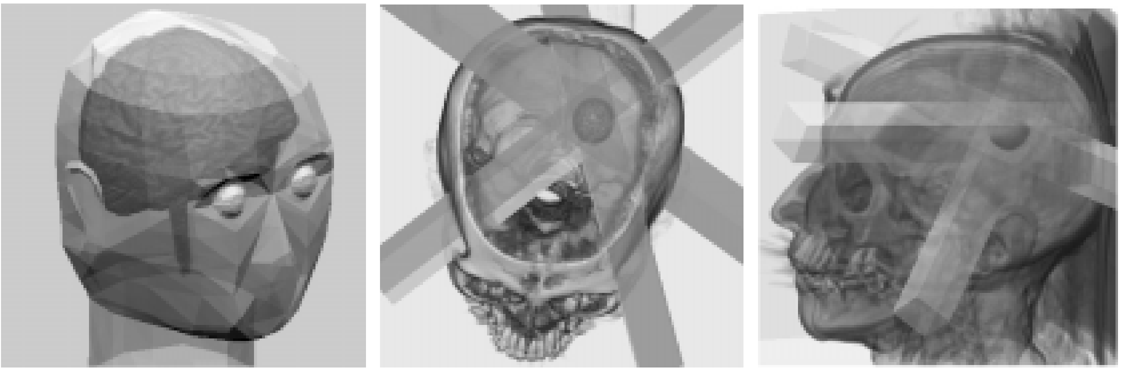
Hybrid ray casting with the 9,184 faces geometric model.

Table 1

Number of faces in the geometric model	Brute force	With bounding rectangles	With viewport subdivision
564 faces	76.7 s	19.0 s	13.5 s
2,296 faces	297.8 s (≅ 5 min)	30.6 s	14.0 s
9,184 faces	1,754.6 s (≅ 29 min)	103.3 s	14.7 s

Comparison of rendering time obtained with each technique when rendering the geometric objects combined with the brain volume.

Table 2



Images obtained with hybrid ray casting in RenderVox.

Figure 5

REFERENCES

[Badou90] Badouel, D. An efficient ray-polygon intersection. In: *Graphics Gems*, A.S. Glassner (ed.). pp. 390-396, Academic Press, New York, 1990.

[Brady97] Brady, M.; Jung, K.; Nguyen, H. Two-phase perspective ray casting for interactive volume navigation. *Visualization'97*, Phoenix, AZ, 1997. pp. 183-189.

[Drebi88] Drebin, R.A.; Carpenter, L.; Hanrahan, P. Volume rendering. *Computer Graphics*, 22(4):64-75, 1988.

[Fuchs77] Fuchs H.; Kedem, Z.M., Uselton, S.P. Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693-702, October, 1977.

[Glassn84] Glassner, A.S. Space subdivision for fast ray tracing. *IEEE Comp. Graphics and Applications.*, 4(10): 15-22, 1984.

[Günth94] Günther, T.; Poliwoda, C.; Reinhart, C. Hesser, J., Manner, R.; Meinzer H.P., Baur, H.J. VIRIM: a massively parallel processor for real-time volume visualization in medicine. *9th Eurographics Workshop on Graphics Hardware*, Oslo, Norway, 1994. pp. 103-108.

[Kaufm87] Kaufman, A. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes. *Computer Graphics*, 21:4(171-179, July 1987.

[Kaufm90] Kaufman, A.; Yagel, R.; Cohen, D. Intermixing surface and volume rendering. In: *3D Imaging in Medicine, Algorithms, Systems, Applications*, K. H. Hoehne, H. Fuchs; S. M. Pizer, (eds.), Springer-Verlag, 1990.

[Knitt94] Knittel, G. and Strasser, W. A compact volume rendering accelerator. *1994 Symposium on Volume Visualization*, 1994. pp. 67-74

[Levoy90a] Levoy, M. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245-261, July 1990.

[Levoy90b] Levoy, M. A hybrid ray tracer for rendering polygons and volume data. *IEEE Computer Graphics and Applications*, 10(3): 33-40, 1990.

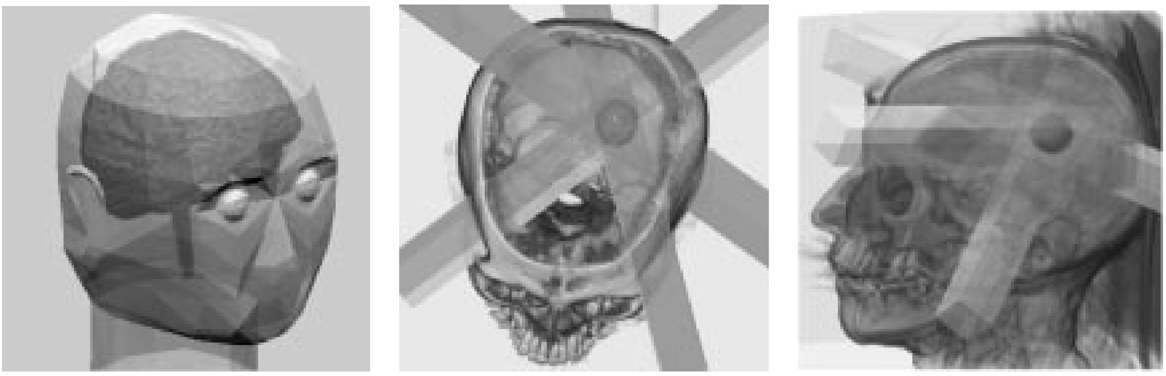
[Loren87] Lorensen, W. and Cline, H. Marching cubes: a high-resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163-170, July 1987.

[Ma94] Ma, K.-L., Painter, J.S., Hansen, C.D., and Krogh, M.F. Parallel volume rendering using binary-swap image composition. *IEEE Computer Graphics and Applications*, 14(4): 59-68, 1994.

[Whitt80] Whitted, T. An improved illumination model for shaded displays. *Communications of the ACM*, 23(6):342-349, 1980.

Credits - Data volumes were obtained from Chapel Hill Volume Rendering Test Data Set, Volume II, provided by the North Carolina Memorial Hospital.

Acknowledgments - Thanks are due to Luciana Porcher Nedel, from UFRGS, for helpful comments on a previous version of this paper. We also acknowledge the financial support from CAPES and CNPq.



Images obtained with hybrid ray casting in RenderVox.
Color plate of Figure 5