EMPOWERING ISO-SURFACES WITH VOLUME DATA

D.M. Eler  P.S.H. Cateriano  L.G. Nonato  M.C.F. de Oliveira

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Av. Trabalhador São-Carlense 400, POBox 668, 13560-970, Brasil
Email: {eler,patricia,gnonato,cristina}@icmc.usp.br

H. Levkowitz
Department of Computer Science
University of Massachusetts Lowell
Lowell, MA 01854, USA
Email: haim@cs.uml.edu

Keywords: Hybrid rendering, Iso-Surface, Volume Rendering, Ray-Tracing, Volume on Surface.

Abstract: Surface rendering algorithms are fast, but not suited to applications that demand exploration of internal volume structures, as such information is lost in conventional surface rendering. In this article we introduce an enhanced surface rendering algorithm - named VoS, Volume on Surface - that supports visualization of internal volume structures. VoS integrates surface and volume rendering approaches into an efficient framework for interactive visualization of volume information. A ray casting is performed to map volume information onto a boundary surface extracted from the volume grid, enabling the display of structures internal to the surface using conventional surface rendering. As such, the technique exploits the advantages of surface rendering while keeping the volumetric information. VoS thus offers a low-cost alternative to volume rendering in some practical situations, as its resulting surfaces can be rendered on commodity graphics hardware at interactive rates. Moreover, a user can easily fine tune the color and opacity transfer function definitions, because changes in the transfer functions are handled at the rendering step, rather than at the costly ray-casting operation.

1 INTRODUCTION

Volume visualization has enjoyed significant evolution in recent years, contributing to major advances in important applications, such as fluid dynamics, analysis of biological structures and medicine. In medicine, for example, volume visualization has played an essential role in the empowerment of imaging technologies, such as ultra-sound, as well as in the development of new methods, including computer-guided surgery. It continues to play a major role in computer-assisted diagnosis, motivating further research into new methods capable of generating improved visualization of internal body structures at faster rates.

Despite the fast evolution and the availability of specific hardware and effective algorithms, volume visualization is still slow and computationally expensive as compared to surface rendering. It is not an option if the goal is fast image generation on standard graphics cards. Surface rendering is fast, but has limited applicability. Conventional surface rendering approaches completely dismiss the volumetric information, manipulating only surfaces extracted from the volume data using some geometric approach. Therefore, surface rendering is not suited to applications that demand exploration of internal volume structures.

In this article we introduce an enhanced surface rendering algorithm – named VoS, Volume on Surface – that supports visualization of internal volume structures. VoS targets the limitations of surface rendering by integrating surface and volume rendering approaches in order to provide an efficient framework for interactive visualization on standard graphics cards. A pre-processing step uses ray casting to map the volumetric information contained in the domain grid onto a boundary surface extracted from this grid, enabling the display of structures internal to the surface using conventional surface rendering. As such, the technique exploits the advantages of surface rendering while keeping the volumetric information. VoS thus offers a low-cost alternative to volume rendering in some practical situations, as its resulting surfaces can be rendered on commodity graphics hardware at interactive rates. Moreover, changes in the color and opacity transfer functions are handled at the rendering step, with no need to repeat the costly ray-casting operation.

This paper is organized as follows: Section 2 presents an overview of related work. Section 3 de-
The term hybrid volume rendering techniques is not consolidated, however, and has been employed in different contexts. One such context refers to a wide class of algorithms characterized by combining SR and DVR strategies into a single visualization environment. Examples include the combination of ray-tracing with surface rendering (Levoy, 1990b) and of splatting with surface rendering (Tost et al., 1993). The goal of such approaches is to enable simultaneous visualization of volumetric data and objects modeled from geometric primitives. A typical application for such hybrid approaches is computer-aided surgery, where surgical instruments must be displayed by an SR technique and patient’s data is shown with direct volume rendering (Gross, 1998). Other hybrid approaches, such as the one proposed by (Zakaria and Saman, 1999), integrate surface rendering, direct volume rendering, and domain transform into a single environment.

Another use of the term refers to Image-Based Hybrid Rendering techniques, which take the overall approach of mapping a set of images generated from a volume onto surfaces. These can then be rendered using conventional graphics hardware (Wilson et al., 2002). The mapping is typically performed using texture maps that are available in graphics cards. Chen et al.’s work (Chen et al., 2001) is a good representative of this class. In their method, the main idea is to pre-compute, using conventional volume rendering, a set of *keyviews*, which, depending on the viewer’s position, are texture-mapped onto a surface that bounds the volume of interest (their solution uses a sphere as the bounding surface). When a viewer moves away from a *keyview*, the texture is kept in the regions still visible and rays are cast for pixels in newly visible regions. Where to place the cameras to generate the *keyviews* is an important issue in this approach. Another major problem is that undesired holes appear in the image when the viewer moves away from the *keyviews*.

Still another meaning for the term hybrid rendering is adopted by (Samanta et al., 2000), who name as “hybrid” a parallel volume visualization algorithm that sub-divides both the volumetric and the image domains in order to improve the load balance among processors. Sometimes the term is also used to describe optimization approaches introduced in traditional rendering algorithms. Levoy and Whitaker’s (Levoy and Whitaker, 1990) and Laur and Hanraham’s (Laur and Hanraham, 1991), for example, are considered hybrid approaches, though they are essentially DVR algorithms highly optimized through progressive mesh refinement and hierarchical representations.

The VoS technique, proposed in this work, can also be considered a hybrid algorithm, in the sense that it combines both surface rendering and ray casting into a unified algorithm. However, its goal is not to enable simultaneous visualization of geometrically defined surface models and volume data. In fact it uses ray-casting as a mechanism to enrich the surface rendering with volume information, in an approach that bears similarity with image-based approaches such as the one by (Chen et al., 2001). However, some differences are distinguishable:

1) The volumetric information can be transferred to any user-specified iso-surface of interest extracted from the data volume, rather than to an external surface bounding the volume; 2) VoS does not use tex-
ture mapping, i.e., rays are cast directly from the faces of the boundary surface; 3) ray casting is executed only in a pre-processing step, thus enabling efficient surface rendering; 4) changes in the color and opacity transfer functions are handled in the rendering step, with no need to redo the ray casting process; 5) VoS presents no problems with the position of cameras.

VOS can be summarized as follows: given a volume stored in a regular grid of voxels, a ray-casting algorithm, analogous to the one typically employed in direct volume rendering, maps the volumetric information onto a surface of interest extracted from the volume. Note that only content internal to the extracted surface will be shown. A surface rendering algorithm is then applied to produce a volumetric visualization of the domain.

3 THE VOS TECHNIQUE

The pipeline in Figure 1 shows the stages of the technique and the main operations undertaken at each stage. Given a regular volume grid, the surface of interest must be extracted using any appropriate algorithm. Isosurface data is then stored in a topological data structure capable of keeping the information required by the mapping process. A high-pass filter is applied to the volume and the resulting information is stored in order to allow identifying transitions between materials.

The overall rationale of the algorithm, which takes as input an iso-surface \( S \), is as follows. Let \( f \) be a face of \( S \) and let \( P \) be the plane containing \( f \), as shown in Figure 2. Face \( f \) can be seen from any point \( P \) on the opposite side of \( S \) (as in Figure 2 (a)). If \( S \) is a transparent object, part of its internal volume will be observable through \( f \). Depending on the viewer’s position, different internal structures can be observed through \( f \). Evidently, the pattern of colors (or ‘texture’) observable will change as the viewpoint changes.

Figure 1: VoS pipeline.

Figure 2: a) Viewing region of \( f \); b) Sampling the viewing directions of \( f \). The bold arrows illustrate the observation line that best approximates the viewing direction.

If one could compute and store the texture of \( f \) for every possible viewing direction it would be possible to identify and assign the appropriate texture to \( f \) for any viewpoint. Therefore, a viewer would be able to observe any structures contained in \( S \) processing only the surface representation. Obviously, it is not feasible to compute or even store the colors of the vertices for all possible viewing positions and directions. Thus, VoS operates on a sampled sub-set of the set of all possible viewer’s positions: a sampling approach obtains a sub-set of viewing directions from which one might observe the interior of \( S \) through the vertices of \( f \).

For each face \( f \) the algorithm computes, for each sampled viewing direction (also called ‘observation lines’), the colors that approximate the textures in the face when observed from that particular direction. In fact, colors are pre-computed and stored at the vertices, and the color of a face is computed in the rendering step by combining the colors of its vertices, as illustrated in Figure 2 (b). When computing face colors the rendering algorithm will identify, for each face vertex, which of the pre-computed observation lines
approximates best the viewer’s line of sight for that vertex. Then, it assigns to the face a color computed from the average of the appropriate colors assigned to each of its vertices. Finally, the rendering algorithm just renders all visible faces.

The technique is actually a two-step method:

- Pre-visualization (maps the volume onto the extracted surface)
- Surface rendering (renders and projects the surface for visualization)

In the following we describe both steps in detail and discuss some implementation issues.

### 3.1 Pre-visualization

Pre-visualization, responsible for the ray-casting process, concentrates the core of the processing. This step takes as input a volume grid of voxels and an isosurface extracted from it.

#### 3.1.1 Sampling the Viewing Directions

In order to sample the set of all possible viewing directions and define a set of observation lines for each vertex in the surface, the VoS algorithm defines virtual cones with different opening angles, centered at the vertex and aligned with the normal vector at the vertex, as illustrated in Figure 3. The observation lines start from the vertex and their directions are determined by uniformly distributing the lines over the surface of each cone, as shown in Figure 3.

![Figure 3: Three virtual cones with different opening angles and observation lines defining the viewing directions to be sampled.](image)

Our current implementation of VoS uses, for each vertex, a set of cones with equally spaced opening angles, for example, if three cones are used they have opening angles equal to 15°, 45° and 75° from the normal vector. The observation lines are also uniformly distributed on the surface of the cones. For example if four observation lines are cast, the angle between each two of them is 90°, measured on the cone.

#### 3.1.2 Storing Samples

This step involves casting rays through the volume in the directions of the observation lines, similarly to traditional Ray Casting for volume rendering. Discrete ray trajectories are computed with a 3D scanline algorithm. Each voxel along the scanline whose value of the high-pass filter is within a user-specified threshold is stored as a ray sample. Each ray is associated with a list of its samples.

#### 3.1.3 Transfer Function Specification

The color and opacity transfer functions are responsible for computing the color and opacity associated with a ray sample. VoS uses a simple user-defined intensity-based color transfer function, and two alternative types of opacity transfer functions, based on sample intensity information and on sample spatial information.

The user specifies a color transfer function by defining a set of control points and assigning a color to each defined point. A user interface for color transfer function definition is illustrated in Figure 4, which shows the volume scalar histogram. Each control point is associated with a scalar value, and is defined directly over the histogram. In Figure 4 we observe a control point $A$ defined over the scalar value 90, to which the user assigned a color $C_A$ (in this case, Red), and a control point $B$ defined over the scalar value 234 for which a color $C_B$ was assigned (Yellow in the figure). The color of any scalar value between control points $A$ and $B$ (that is, between scalars 90 and 234) is linearly interpolated from the corresponding RGB values of $C_A$ and $C_B$.

![Figure 4: Specification of a color transfer function.](image)

Specification of an opacity transfer function based on scalar intensity is similar: now the user assigns opacity values to the control points, and the opacity
for any scalar value in-between is obtained with linear interpolation from the known opacity values.

An alternative opacity transfer function that embeds spatial information is provided: it allows users to assign opacity values based on the position of a sample along the ray. The rationale is that samples positioned closer to the extreme ends of the ray (its starting point or its final intersection point) correspond to ray intersections with the most external or the most internal objects in the volume. The user can then assign opacity values based on sample position, as exemplified in Figure 5, which shows two rays and their associated sample points (represented by the squares).

To favor visualization of internal structures one must assign higher opacity values to samples in the internal regions of the ray, as shown in Figure 5 (a). On the other hand, to highlight more external object structures one must assign higher opacities to the samples closer to the ray extremes, as illustrated in 5 (b).

The samples stored on a given ray are then composed to determine the ray color and opacity values. Sample composition might use several optical models that take into account different light, material and ambient parameters. We follow the simple optical model proposed by Levoy (Levoy, 1990a), which considers only sample color and opacities and can be stated as follows:

\[
C = \sum_{i=1}^{N} c(i) \alpha(i) \prod_{j=1}^{i} (1 - \alpha(j))
\]

In the above equation \(C\) is the final color and \(N\) is the number of samples. \(c(i)\) and \(\alpha(i)\) represent, respectively, color and opacity of sample \(i\) and \(\alpha(j)\) represents opacity of sample \(j\). Each surface vertex stores the values of computed \(C\) for each ray cast from it. If the color or opacity transfer functions are modified a new composition must be performed, but no further ray casting is required.

### 3.2 Surface Rendering

At the end of the pre-visualization step a final color has already been assigned and stored for each ray, or equivalently, for each observation line leaving each surface vertex. The initial problem for the rendering stage is to decide the appropriate color for the vertex as observed from the viewer’s current position.

The problem of finding the appropriate color for a vertex may be solved with a dot product computation: Let \(v\) be the vertex and \(R = \{r_1, \ldots, r_k\} (k = \text{number of cones} \times \text{number of rays cast per cone})\) the set of unit vectors on the observation lines (pointing outwards from \(S\)). Let \(r_o\) be the unit vector constructed from the viewer position to the object, \(O\). Suppose that \(r_o\) points towards the viewer and \(r_i \in R\) is the vector whose dot product \(r_i \cdot r_o\) produces the largest positive value amongst the vectors in \(R\). The color associated with the observation line of \(r_i\) is thus attributed to \(v\) as the most appropriate color for \(v\) when observed from \(O\).

It is worth mentioning that this dot product computation can be performed in hardware, and the same happens for the computation of hidden elements and light effects on the surface. These operations are supported by a considerable number of commodity graphics cards, ensuring the efficiency of the rendering step.

### 4 RESULTS

In this section we present some results of applying VoS to visualize internal volume structures.

Figure 6 shows two visualizations of the chest data set (available at http://www9.cs.fau.de/Persons/Roettger/library/).

This data set has dimensions 120x120x241. The visualization in Figure 6 (a) was generated using Kitware’s Volview (http://www.volview.com) using conventional ray casting. Figure 6 (b) shows a shaded surface rendering of the same data, for an isosurface value of 80, created with the `vtkContourFilter` method of VTK (Schroeder et al., 2004). The resulting mesh has 83,949 vertices and 167,241 faces. The VolView visualization took nearly 30 seconds to compute (on a Pentium 4, CPU 3.2 GHz with 1 GB RAM).
Isosurface visualizations have a rate of nearly 2 frames per second on VTK.

![Figure 6: a) Ray casting image; b) Surface rendering image with isosurface = 80.](image)

Figure 6: a) Ray casting image; b) Surface rendering image with isosurface = 80.

Figure 7 shows visualizations of the same data created with VoS. Figures 7 (a), (b) and (c) were created with 3, 6 and 8 cones per vertex, respectively, and all of them casting 8 observation lines per cone. The quality of the VoS visualizations is comparable to that of the ray casting image shown in Figure 6 (a), even for the one generated with less cones - clearly, image quality is better for higher numbers of cones.

![Figure 7: VoS visualizations using 8 observation lines per cone a) three cones; b) 6 cones; c) 8 cones; d) 8 cones, from a different viewpoint.](image)

Figure 7: VoS visualizations using 8 observation lines per cone a) three cones; b) 6 cones; c) 8 cones; d) 8 cones, from a different viewpoint.

Computational times to generate these visualizations are shown in Table 1. Pre-visualization time (first row) is the time to cast all rays (observation lines) from all surface vertices and to store their associated samples. The best and worst case entries indicate the time to re-compute color composition when transfer functions are modified. In the best case all scalar values are associated with the maximum opacity value, and therefore ray casting stops at the first sample; in the worst case all samples along the ray must be taken into account in the color composition. Note that the time required to update the visualization once the transfer functions change is much shorter than in a conventional DVR ray casting, as the pre-processing step stores all the necessary information, avoiding the need of casting rays again. Note also that the number of frames rendered per second with VoS is lower than, but on the same order of magnitude, that of a conventional surface rendering implementation, such as the VTK one. These tests have been performed on a Pentium 4, CPU 3.2 GHz and with 1.0 GB RAM.

The time required for the VoS pre-visualization stage is slightly greater than that of a DVR visualization, when using three cones with 8 observation lines). Pre-visualization time increases if more cones are used, however, once pre-processing is done surface renderings can be displayed in much lower times, as shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>3x8</th>
<th>6x8</th>
<th>8x8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-visual</td>
<td>41s</td>
<td>1min 36s</td>
<td>2min 10s</td>
</tr>
<tr>
<td>Best Case</td>
<td>0.25s</td>
<td>0.49s</td>
<td>0.64s</td>
</tr>
<tr>
<td>Worst Case</td>
<td>2.3s</td>
<td>3.85s</td>
<td>4.88s</td>
</tr>
<tr>
<td>fps</td>
<td>1.9</td>
<td>1.3</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Table 1: VoS measurements.

Only the lungs and their internal structures are visible in the visualizations depicted in Figure 8. Figure 8(a) was created with conventional ray-casting, while Figure 8(b) was generated with VoS using the intensity based opacity function.

![Figure 8: Lungs in the chest data set: (a) Visualization created with conventional ray-casting; (b) VoS visualization with 8 cones and 8 observation lines per cone.](image)

Figure 8: Lungs in the chest data set: (a) Visualization created with conventional ray-casting; (b) VoS visualization with 8 cones and 8 observation lines per cone.
Figure 9 shows still another comparison between visualizations created with conventional ray casting and with VoS, now for a papaya fruit data set, with dimensions 86x42x59. This example reinforces that VoS can be a reasonable alternative to conventional DVR in some applications, with the advantage of allowing more interactive display rates.

![Figure 9: Papaya data set](image)

Figure 9: (a) and (c) Images created with conventional ray-casting; (b) and (d) VoS images (8 cones with 8 observation lines per cone).

Figure 10 illustrates the use of the opacity transfer function based on sample position. In the visualization shown in Figure 10 (a), obtained with the opacity function configuration depicted in Figure 11 (a), one observes the outer parts and the pulp that hides the seeds. The internal hole is highlighted in Figure 10(b), obtained configuring the opacity function as shown in Figure 11 (b).

While manipulating these data sets we observed that it was quicker to generate meaningful images of internal volume structures using this transfer function as compared to using the usual intensity based one – though it sometimes results in inferior image quality. We believe this is a promising approach for transfer function definition that requires further tuning.

Regarding memory usage, the amount of memory required depends on three factors: the number of vertices in the surface mesh, the number of observation lines cast per vertex, and the number of samples stored for each observation line. For the VoS visualizations of the papaya data set shown in Figure 9(b) and (d), produced using 8 cones per vertex and 8 observation lines per cone, memory consumption was close to 32 MB.

5 CONCLUSIONS

In this paper we have introduced VoS, an image-based surface rendering approach that supports fast visualization of internal volume structures. VoS maps the volume information contained within a surface on its faces in a way that enables efficient rendering, as the information required at the rendering step is pre-computed and stored. A user still has the flexibility of modifying the transfer functions in the rendering step in order to hide or highlight information.

VoS has a major advantage in its ability to image internal volume contents quickly on conventional graphics hardware, thus offering enhanced surface rendering. It provides an additional tool in situations in which surface rendering is a natural visualization solution. Moreover, it can be an alternative to quickly generate initial views of volume contents, or to enable volume visualizations on devices where specific volume graphics hardware is not available, such as...
'portable' devices.

The algorithm is also applicable to the visualization of tetrahedral meshes, in which case the pre-visualization stage could be incorporated into the mesh generation and numerical simulation software, enabling straightforward visualization of results.

One aspect that deserves further investigation is how the quality of the resulting visualizations is affected by the quality of the surface mesh. Mesh simplification and smoothing algorithms could be applied prior to generating the visualizations, and their effect on the quality of the VoS outcome must be analyzed.

Some improvements that can be realized in VoS deserves attention, they are: use of GPU to speed up the calculations, parallelism in pre-processing (ray casting) and surface rendering, and the mechanisms for specifying color and opacity transfer functions. In the intensity based transfer function specification we are interpolating in the RGB space, which is obviously not the best choice. The sample based opacity transfer function was not explored to its full potential. For example, in assigning opacities we could use both the sample position and its value in more sophisticated interpolation schemes.

6 ACKNOWLEDGMENTS

The papaya data set has been provided by EMBRAPA – Empresa Brasileira de Pesquisa Agropecuária. This work has been supported by FAPESP Grants 03/02815-0 and 04/01756-2, and CNPq Grants 300531/99-0 and 521931/95-5. Haim Levkowitz’s research was partially conducted while he was a Fulbright US Scholar to Brazil, August 2004 - January 2005.

REFERENCES


