Towards a Comprehensive Volume Visualization System

Ricardo S. Avila*, Lisa M. Sobierajski†, and Arie E. Kaufman‡

*Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400

†Howard Hughes Medical Institute
State University of New York at Stony Brook
Stony Brook, NY 11794-5230

Abstract

The VolVis system has been developed to satisfy the diverse requirements of the volume visualization community by comfortably housing numerous visualization algorithms and methods within a consistent and well organized framework. The VolVis system is supported by a generalized abstract model which provides for both geometric and volumetric constructs. VolVis contains several rendering algorithms that span the speed versus accuracy continuum. A fast volume rendering algorithm has been developed, which is capable of exploiting existing graphics hardware without placing any viewing restrictions or compromising accuracy. In addition, VolVis includes a volumetric navigation facility, key-frame animation generator, quantitative analysis tools, and a generalized protocol for communicating with 3D input devices.

1. Introduction

Volume visualization is quickly becoming an integral part of many scientific fields [4]. Visualization techniques are used in areas as diverse as aiding physicians in surgical and radiation treatment planning [2], helping meteorologists understand weather patterns [3], and assisting geologists in identifying potential oil reservoirs [13]. The software used by the geologist would not help a meteorologist predict the weather, and would be useless for surgical planning. In the past, new visualization systems have been developed for each visualization application. In each system, assumptions are made about the type of data, the rendering algorithm, or the input device employed, creating a software package with limited applications.

A few visualization systems have been developed which achieve some degree of independence. For example, a procedural interface was developed [7] which provides a library of C routines for data visualization. This is useful as a basis on which a computer scientist can build a new visualization application, but is not useful to a scientist or an engineer without C programming experience. A more complete system, ANALYZE [8], was developed which provides the scientist with a rich set of tools for the manipulation, analysis, and display of 3D and 4D biomedical data. Interaction in ANALYZE is achieved through keyboard and mouse input, which does not always provide an intuitive method for performing three-dimensional data manipulation and analysis. Another commercial system is VoxelView of Vital Images [9, 11]; however, VoxelView is running only on the Silicon Graphics graphics engine and cannot be used on other platforms.

In this paper we present the VolVis system, a comprehensive volume visualization system. One goal of the VolVis system is to create a powerful volume visualization environment, while avoiding any windowing, input device, algorithm or data dependencies. This is achieved by first creating an abstract model for the system environment, which defines world, volume, and light source properties. Algorithms are then developed for data manipulation, navigation, quantitative analysis, and rendering in this environment. These algorithms rely on the abstract model, and are independent of machine types, windowing systems, and input devices. Finally, the user interface, and any device-dependent routines (e.g., for rendering geometric primitives) are incorporated into the system. The abstract model and functionality of the VolVis system are further described in Sections 2 and 3.

Another goal of the VolVis system is to provide the user with several rendering algorithms spanning the speed versus accuracy continuum. At one extreme is the low-accuracy rendering algorithm employed by the navigator. At the other end is a volumetric ray tracer (e.g., [14, 16]), which provides realistic images at a relatively high cost. By placing restrictions on the viewing parameters, a template-based ray caster [15] can be employed which produces more accurate images than the navigator at lower rendering times than the volumetric ray tracer.

Many algorithms for surface projections of volumetric data currently exist. One method is to approximate the surface with a polygon mesh [6], then utilize standard graphics hardware for projection. To produce accurate images of complex surfaces with this method typically requires more polygons than can be projected interactively on even a high-end graphics workstation. Another method for surface projection...
from volume data is described by Levoy [5]. This method produces accurate images, but has a high rendering time. Other visualization methods exist for rendering complex surfaces [1, 10, 12], but none of these methods allow arbitrary viewing parameters while maintaining interactive rates.

The VolVis system contains a new volume rendering algorithm for complex objects, called Polygon Assisted Ray Casting, or PARC, which can accurately project complex surfaces at interactive rates. This algorithm provides for accurate rendering at interactive rates by exploiting the hardware of existing graphics workstations. The PARC algorithm is described in detail in Section 4.

2. Abstract model

The VolVis system is designed to meet several key objectives. One important objective is to be independent of the window system, input device, algorithm, and data. Another important objective is flexibility, thereby insuring that a new concept or algorithm can be incorporated without constraining existing or future algorithms. To achieve these goals, an expandable abstract model for volume visualization was developed.

The first step in creating the abstract model is to define some basic building blocks. A Position defines a point \( p \in \mathbb{R}^3 \), while a Vector defines 3D orientation or direction. A Matrix is a \( 4 \times 4 \) array of floating point values which represents a transformation in a three-dimensional homogeneous coordinate system. A Plane is a plane equation defining an infinite plane in \( \mathbb{R}^3 \), and a Color is a \((\text{red}, \text{green}, \text{blue})\) triple corresponding to a 24-bit color. Built upon these basic building blocks are some higher level constructs such as a CoordSys, which is a Position indicating the origin, and three Vectors representing the \( x \), \( y \), and \( z \) axes of a cartesian coordinate system in \( \mathbb{R}^3 \).

Some of the basic building blocks in the VolVis abstract model can assume one of multiple representations. For example, a Segmentation can be a simple threshold, a binary segmentation function, or a transfer function defining opacity based on scalar value and gradient. Likewise, a LocalShade defines a shading model for a volume which can contain constant coefficients, or complex transfer functions defining shading coefficients based on scalar value, gradient, data color, or texture color. A DataColor can define a single color for the volume, or a transfer function mapping scalar value and gradient to color. A Texture may contain a 2D texture, a 3D solid texture, or a function which is used to generate the texture values. A DataCut contains a cut geometry which is to be applied to a volume. For example, a DataCut may be a plane, a hexahedron, or a sphere. The VolumeData can be either geometric, scalar data, or vector data, and a LightSource can be a point light source, a directional light source, a geometric light source or a volumetric light source. These basic abstract model building blocks are illustrated in Figure 1.

![Figure 1: The basic building blocks of the abstract model.](image)

The three top-level components of the VolVis abstract model are the Image, the View, and the World. An Image is simply an object which contains a width and height (in pixels), a depth (in bytes), and raw pixel values. A View is an object which defines the current specification of the view plane. This is done by storing an original CoordSys, a current CoordSys, and a Matrix defining a transformation between the two. The \( x \) and \( y \) axes of the current CoordSys lie on the view plane, while the \( z \) axis defines the general viewing direction. Also stored in the View are the field of view, and the width and height of the View in both pixels and units, where a unit is considered to be a distance of 1.0 in the World coordinate system. When a projection is performed, an Image is associated with the View.

The World contains all Volumes and Lights, as well as the WorldShade shading model which includes global information such as ambient light and background color. The CoordSys for the World is a standard left-handed cartesian coordinate system. All other coordinate systems defined in the abstract model are relative to this standard coordinate system. The World may contain multiple DataCuts defining cut geometries which are to be applied to all Volumes. Multiple Lights are defined within the World, where each Light contains a LightSource. Multiple Volumes are also supported, with each Volume containing an original CoordSys, a current CoordSys, and a transformation Matrix, similar to the View. Also contained in the Volume are the Plane equations defining the faces of the bounding hexahedron of the Volume, as well as the \( x \), \( y \), and \( z \) resolution of this.
hexahedron in both voxels and units. The resolution in voxels indicates the number of sample points in the volume data, while the resolution in units specifies the size of the object in World space. Each Volume contains a unit type, which indicates what one unit in this volume represents. For example, one unit in a volume may represent a micron, an inch, or a kilometer. This information is used during measurement, and is also used to convert multiple volumes into a common unit type, thereby showing the relative sizes of these volumes. In order to facilitate transformations between the World coordinate system and the Volume coordinate system, four conversion Matrices are stored within each Volume. These Matrices are used to transform a Position or a Vector from the World coordinate system to the Volume coordinate system (and vice-versa) in both units and voxels. A Volume may contain multiple DataCuts which are applied to the data in addition to the ones specified in the World. Each Volume also contains a LocalShade, a DataColor, a Texture, and the actual VolumeData. If the VolumeData is scalar data, it contains a Segmentation, and the actual sampled data. If the VolumeData is geometric data, it contains a geometric object description. A representation of a Volume built from the icons of Figure 1 is shown in Figure 2.

Another advantage of the VolVis abstract model is its expandability. The model is designed so that, for example, the addition of a new Texture type, Segmentation type, or geometric object requires a minimal amount of work, and does not compromise the flexibility of the model. Incorporating a new projection algorithm or measurement tool is also made simple by this model.

3. System functionality

The VolVis system contains several primary components which are intended to meet the various needs of volume visualization users. These include file input and output, filters, object control, image control, rendering, navigation, animation, quantitative analysis, and input devices. Each of these components is described below.

The File I/O component handles all input and output within the VolVis system. This includes the ability to save and retrieve several volumetric data file types, geometric data files, multi-dimensional texture files, 24-bit image and animation files, world environment files, and interaction log files. It is often desirable to apply a filter to a volumetric data set during or after input in order to enhance features, smooth data, or reduce noise. For this purpose several standard filters are provided within the Filters component of the VolVis system.

Every object within the VolVis system contains properties which can be modified within the Object Control component. The VolVis system contains several basic object types including World, View, Volume, and Light. The number and types of properties vary depending on the object in question. For example, some of the more relevant properties of a Volume are position and orientation, segmentation, color, texture, local shading model, cut geometries, and flags indicating whether this object is currently visible and/or modifiable. Modifiability allows the user to apply transformations to several Volumes and Lights simultaneously in order to perform complex object manipulations easily. Visibility simply determines whether a Volume is visible or a Light is active. The global properties stored within the World such as ambient lighting, light attenuation factor, background color, and global cut geometries may also be modified in this component. The Object Control component provides a window based method for modifying these properties. However, some properties may be modified utilizing more interactive methods within the Navigation component.

The Rendering component of VolVis is perhaps the most powerful component within the system. The
user may select from a variety of rendering algorithms, each having its own unique set of advantages and disadvantages. Generally speaking, these algorithms span the continuum between fast and inaccurate to slow and accurate. The Rendering component includes a template-based ray casting algorithm for quick discrete parallel projections [15]. Despite the speed of the algorithm, it unfortunately cannot perform perspective projections and is also unable to handle a generalized viewing window. The PARC algorithm can perform parallel and perspective projections as well as handle a fully generalized viewing window. This algorithm provides highly accurate projections with relatively fast projection times. The PARC algorithm is discussed in more detail in Section 4. Finally, a powerful volumetric ray tracer is included within the Rendering component to provide high quality projections using a global illumination model, similar to [14, 16]. This is particularly useful when global effects such as shadows and reflections can provide more clues about the structure of the data.

The Navigation component is an interactive volumetric navigation aid which allows the user to specify complex flight paths through volumetric data sets as well as directly manipulate all objects defined within the World. The user navigates through data sets much like a flight simulator in which movement can take place with respect to the View coordinate system. The navigator is controlled through either the navigation window or the 3D input device specified in the Input Devices component. Interactive projection rates are achieved by utilizing a quick projection algorithm which operates on a reduced representation of the Volumes. The user may change the level of detail displayed by the navigator to suit the operation being performed.

Figure 4: Various VolVis PARC projections of an MRI scanned head.

Figure 5: A VolVis ray traced projection of an MRI scanned head and a sphere.

The Image Control component facilitates the manipulation of images generated within the VolVis.
system. One can create complex animations within the Vol/Vis system through the use of the Animation component. The Animator is a key-frame animation system in which animations are specified by a series of key-frames or states of the system and the number of steps to create between each state. The navigator is generally used to position Volumes, Lights, and the View, and the animator is used to store each key-frame. Once all key-frames have been specified, the animator then interpolates between the key-frames to produce an animation sequence.

The Quantitative Analysis component of Vol/Vis extracts quantitative measurements from volumetric data. Measurement capabilities include 2D and 3D distance, histogram, surface area, and volume measurements. These measurements are performed either on an entire volume or on a subregion selected within the Navigator component.

The Input Devices component allows the user to choose the type of input device currently in use within the Vol/Vis system. The user may choose between a standard mouse, a flying mouse, buttons and dials, an Isotrak, a Spaceball, or even a DataGlove. This component is also responsible for initializing the device and any other parameters associated with its operation.

4. PARC - polygon assisted ray casting

In an effort to attain interactive projection rates, the Vol/Vis system utilizes the PARC algorithm. PARC is a ray casting algorithm capable of computing a fast and accurate surface projection without placing restrictions on the projection type, and the position, orientation or size of the viewing window.

The original volumetric data \( V(x_0, y_0, z_0) \) is a rectilinear array of scalar values. The scalar value \( s \) of a point \( p \) located at \((x_0, y_0, z_0) \in \mathbb{R}^3\) can be approximated with an interpolation function \( I(V, p) \). The interpolation function \( I \) should be piecewise polynomial such as trilinear or tricubic interpolation. The volumetric data \( V \) combined with the interpolation function \( I \) define the scalar field \( S \subset \mathbb{R}^3 \). An object \( S_o \) in the scalar field \( S \) is defined by a segmentation function \( Q \). The segmentation function \( Q \) partitions \( S \) into two sets, either object \( S_o \) or background \( S_g \), such that \( Q(s) \) evaluates to 1 if \( s \in S_o \), and \( Q(s) \) evaluates to 0 if \( s \in S_g \).

The algorithm casts rays through the scalar field \( S \) from each pixel of the viewing window. Figure 6(a) shows the rays as they traverse \( S \) and intersect with an object \( S_o \). Each ray begins from a position on the viewing window and a sample \( p \) is performed at regular intervals along the ray until \( Q(Q(V, p)) = 1 \), indicating that the ray has entered the object \( S_o \). The color of the pixel is then computed by applying a shading function at point \( p \). Although the basic approach does not place any restrictions on the viewing window, rendering times are high due to the large number of samples performed within \( S \). The cost of performing these samples can be quite high given the expense of the interpolation function \( I \). Secondly, an increase in accuracy is accomplished by reducing the distance between sample points, thereby causing a proportional increase in rendering times.

This algorithm can be improved dramatically by reducing the number of samples required during projection. Let's assume that prior to ray casting we know whether a ray originating at some pixel in the View intersects \( S_o \), and if so, the distance along the ray at which the closest intersection occurs. Given this information we need not cast rays which do not intersect \( S_o \). Additionally, for those rays which do intersect \( S_o \), we can effectively skip over all the samples along the ray to the intersection. If we combine casting rays only at viable pixels with sampling \( S \) only on the projected surface of the object, the number of samples required during projection is equal to the number of viable pixels in the View.

Although obtaining the viable pixel and intersection distance information prior to projection appears too costly for consideration, this information can be approximated quickly through the use of polygon projection into two z-buffers. The object \( S_o \) must first be approximated with a set of polygons that completely contain it. We have chosen to represent the polygons as the faces of a set of subvolumes \( P \) within \( S \) which contain a surface of \( S_o \). \( P \) is determined by first subdividing each axis of \( S \) into \( 2^n \) equal distance regions where \( n \) is the subdivision level. A scalar field \( S \) subdivided with level \( l \) then contains \( 2^{3l} \) subvolumes. Every subvolume that contains a boundary between \( S_o \)

![Figure 6: (a) A 2D illustration of brute-force ray casting. (b) A 2D illustration of the PARC algorithm, with \( Z_v \) shown as a dashed polyline, and \( Z_f \) shown as a solid polyline.](image)
and \( S_p \) according to \( Q \) is then placed in the set \( P \).

Once the set of subvolumes \( P \) has been calculated, the six faces of each subvolume in \( P \) are then projected into two z-buffers. The three front facing polygons of each subvolume are projected into the near z-buffer \( Z_n \) while the three back facing polygons are projected into the far z-buffer \( Z_f \). The near z-buffer \( Z_n \) now contains a distance value for every viable pixel in the View and a flag for a pixel that cannot send a ray which intersects \( S_o \). Due to the polygonal approximation, rays sent from viable pixels do not necessarily intersect \( S_o \). However, the higher the subdivision level \( l \), the greater the likelihood that a viable pixel spawns a ray that actually intersects \( S_o \). The far z-buffer \( Z_f \) contains the farthest distance from a pixel to an intersection with \( S_o \). Therefore, a ray from a viable pixel need only step a distance of \( Z_f - Z_n \) starting at a distance of \( Z_n \) along the ray. Provided that graphics hardware is available, projection of the set \( P \) can be performed quickly since shading of the subvolumes is not necessary. Figure 6(b) shows the approximation of the object and the substantial reduction in the number of samples performed along the rays.

The algorithm thus far does not take into account multiple Views and the possibility that the View may intersect with subvolumes in \( P \). To handle multiple Views, we compute a \( Z_n \) and a \( Z_f \) for each View then step along each ray from the closest distance in all \( Z_n \) buffers to the farthest distance in all \( Z_f \) buffers. At each step along the ray we use the information within the z-buffers to determine which Views to sample. To handle a View intersecting subvolumes in \( P \), we project first the back facing polygons of each subvolume into \( Z_n \) with a color \( C_{\text{Back}} \) then the front facing polygons into \( Z_f \) with a color \( C_{\text{Front}} \). Should the z-buffer \( Z_n \) contain the color \( C_{\text{Back}} \), the ray must begin sampling at the pixel position on the View.

The performance of this algorithm now rests on the subdivision level \( l \) chosen. When \( l \) is large, the high number of polygons causes a large polygon projection time but a small ray casting time since the polygons closely approximate the surface of \( S_o \). When \( l \) is small, the low number of polygons causes a low polygon projection time but a large ray casting time since the surface of \( S_o \) is only coarsely approximated. The challenge then is to choose the appropriate subdivision level \( l \) which minimizes the total projection time. Unfortunately, the optimal subdivision level \( l \) is dependent on the floating point speed of the machine, the polygon projection time of the machine, and even the characteristics of the object \( S_o \) within \( S \). In general, the PARC algorithm total time \( T_{\text{PARC}} \) is:

\[
T_{\text{PARC}} = T_{\text{Poly}} + T_{\text{Ray}} + T_{\text{Over}}
\]

where \( T_{\text{Poly}} \) is the polygon projection time, \( T_{\text{Ray}} \) is the ray stepping time, and \( T_{\text{Over}} \) is the projection overhead time.

The rendering of an object \( S_o \) requires the selection of the appropriate subdivision level \( l \). This entails the analysis of the functions \( T_{\text{Poly}}, T_{\text{Ray}}, \) and \( T_{\text{Over}} \) for the object \( S_o \) as a function of \( l \). In general, as \( l \) increases, \( T_{\text{Poly}} \) grows exponentially depending on the surface characteristics of \( S_o \) and the polygon projection rate of the machine. In contrast, as \( l \) increases, \( T_{\text{Ray}} \) decreases exponentially depending on the surface characteristics of \( S_o \) and the floating point speed of the machine. The \( T_{\text{Over}} \) function, which includes looping through the View pixels and shading at ray-object intersections, remains relatively constant.

The strength of this approach is that PARC can adapt to different machine abilities to provide the best possible PARC rendering time. For instance, a machine with exceptional floating point performance but satisfactory polygon projection performance is likely to cause PARC to use a relatively low subdivision level \( l \) in an effort to obtain the lowest possible PARC rendering time. The computation of \( T_{\text{Poly}}, T_{\text{Ray}}, T_{\text{Over}} \) and \( T_{\text{PARC}} \) can be performed as a preprocessing step and stored for later rendering sessions on a specific machine.

![Figure 7: The MRI head and confocal cell images computed while determining the subdivision level \( l \).](See color plates, p. CP-2.)
the hippocampal pyramidal neuron is a representative of a sparse, disconnected branch-like structure. The data sets were projected onto a View with a resolution of 256x256 pixels. The View was rotated 12 times about the y axis with 30 degree angular displacements yielding the functions $T_{Poly}$, $T_{Ray}$, $T_{Over}$ and $T_{PARC}$ for an average image. Figure 7 shows a view of the MRI head and the corresponding view of the hippocampal pyramidal neuron which were computed during this analysis.

The functions for both the MRI head and the hippocampal pyramidal neuron are shown in Figures 8 and 9, respectively. Both the MRI head and the hippocampal pyramidal neuron exhibit a substantial decrease in ray stepping time as the subdivision level $l$ increases. However, the polygon projection time of the MRI head begins to grow quickly when $l > 6$, while the polygon projection time of the hippocampal pyramidal cell shows a gradual increase when $l > 7$. The total projection time $T_{PARC}$ indicates that the MRI head should be subdivided at $l = 6$ resulting in an average PARC projection time of 6.139 seconds. However, the hippocampal pyramidal neuron obtains an optimal projection time $T_{PARC}$ of 3.004 seconds at $l = 7$. The brute-force rendering time is roughly equivalent to PARC level 0 projection. Comparing PARC level 0 and the optimal levels for the MRI head and the hippocampal pyramidal neuron shows that PARC, at the optimal level, achieves a 13.9 and a 42.7 speedup, respectively.

6. Implementation and future directions

VolVis is currently running under X/Motif on HP and Silicon Graphics platforms, and under X/Openwindows on the Sun platform, taking advantage of the graphics hardware as is available on the HP 400S, SGI 240GTX, and the Sun SPARCstation 2 GS. Interactive control of VolVis is accomplished through either a standard mouse, buttons and dials, a Spaceball, or a VPL DataGlove.

Figure 10 shows four images created with the PARC algorithm which are part of an animation of a flight through a hippocampal pyramidal neuron. The VolVis ray tracer was used to render the CT lobster data shown in figure 11. Two blue mirrors were placed in the

(See color plates, p. CP-2.)
scene to reflect hidden parts of the lobster.

The VolVis system is currently being expanded to work in higher dimensions. This includes rendering and analyzing higher dimensional data, as well as navigation in higher dimensions. Additionally, the abstract model is being extended to incorporate irregular grids. The VolVis system is also being ported to other platforms.

 Improvements on the PARC algorithm are also planned. Methods for reducing the number of projected polygons and an adaptive subdivision level are being investigated. The PARC algorithm is also being extended to handle rendering with transparency and more complex segmentation techniques.

Figure 11: A VolVis ray traced projection of a CT scanned lobster.

7. Acknowledgments

This project has been supported in part by a grant from the Center for Biotechnology, SUNY at Stony Brook, which is sponsored by the New York State Science and Technology Foundation, by the National Science Foundation under grant 1RI-9008109, by Howard Hughes Medical Institute, and by a grant from Hewlett-Packard Company. The confocal microscopy is courtesy of Paul R. Adams, Barry J. Burbach, and David Printzenhoff of the Howard Hughes Medical Institute at Stony Brook, New York. The CT lobster data set is courtesy of AVS. Special thanks to Han-Wei Shen, Hui Chen and Richard Calmbach for their help with the implementation of VolVis.

8. References


(See color plates, p. CP-2.)