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 only 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

0-8186-2897-9/92 $03.00 © 1992 IEEE

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 $p \in R^3$, while a *Vector* defines 3D orientation or direction. A *Matrix* is a 4×4 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 R^3, and a *Color* is a (*red,green,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 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.

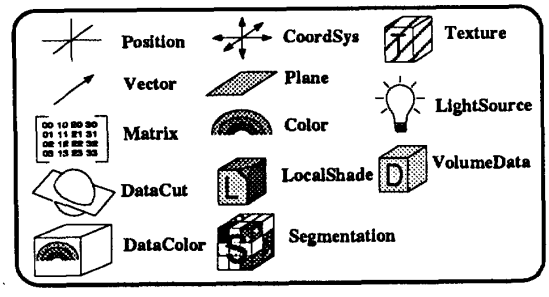

Figure 1: The basic building blocks of the abstract model.

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.

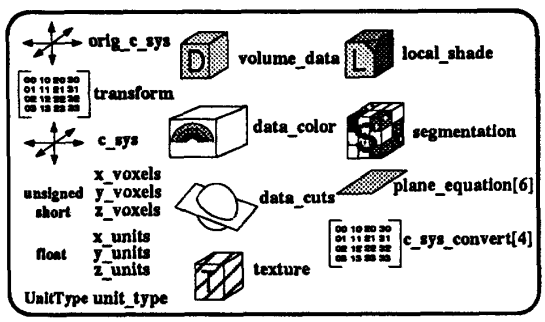

Figure 2: Iconic representation of a Volume in VolVis.

One of the major advantages of this abstract model is the ease with which *Volumes* can be manipulated. Translations and rotations are performed by simply concatenating the new transformation matrix to the one stored in the *Volume*, thereby defining a new current *CoordSys*, new *Plane* equations, and new conversion *Matrices*. Unlike many other systems that require data interpolation to be performed during the data input/reconstruction phase, the *VolVis* abstract model allows for arbitrary scaling of the data with no additional memory expense. This is accomplished by modifying the *x*, *y*, and *z* size in units of the data, which results in modified *Plane* equations and conversion *Matrices*.

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 can not 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.

Figure 3: A VolVis template-based projection of an MRI scanned head.

Figures 3-5 illustrate the differences between the template-based algorithm, PARC, and the volumetric ray tracer. Figure 3 shows an MRI head rendered using the template-based projection algorithm. As required by the algorithm, a parallel projection was performed from a *View* located outside of the volume. In contrast, Figure 4 shows several images of the same head rendered with PARC. The images illustrate the PARC algorithm's ability to handle an arbitrary *View* and cut geometries. Finally, Figure 5 shows the same MRI head, however this time rendered with the *VolVis* ray tracer. This figure illustrates the ray tracer's ability to handle both volumetric and geometric objects within a global illumination model.

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.

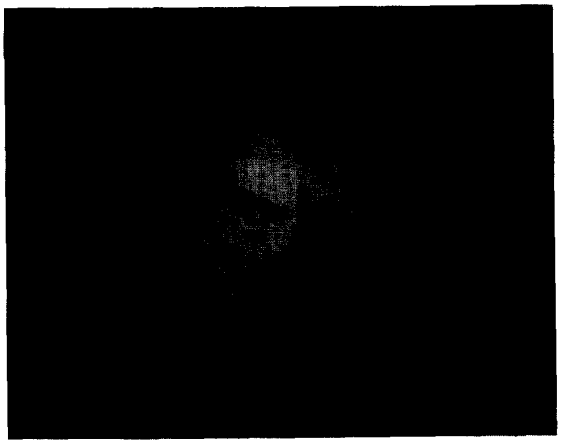

Figure 5: A VolVis ray traced projection of an MRI scanned head and a sphere.

The Image Control component facilitates the manipulation of images generated within the *VolVis*

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

system. One can create complex animations within the *VolVis* 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 *VolVis* 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 *VolVis* 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 *VolVis* 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 $V(x_v, y_v, z_v)$ is a rectilinear array of scalar values. The scalar value s of a point p located at $(x_p, y_p, z_p) \in R^3$ 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 $S \subset R^3$. 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_B, such that $Q(s)$ evaluates to 1 if $s \in S_O$, and $Q(s)$ evaluates to 0 if $s \in S_B$.

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 $Q(I(V, p)) = 1$, 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^l equal distance regions where l 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

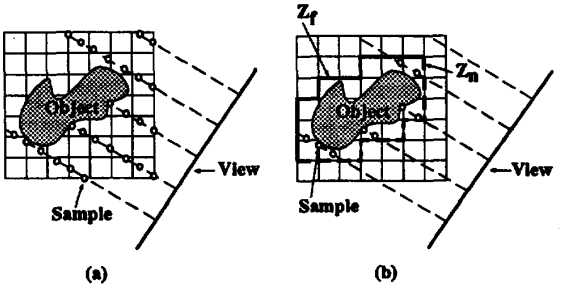

Figure 6: (a) A 2D illustration of brute-force ray casting. (b) A 2D illustration of the PARC algorithm, with Z_n shown as a dashed polyline, and Z_f shown as a solid polyline.

and S_B 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 *Volumes* and the possibility that the *View* may intersect with subvolumes in P. To handle multiple *Volumes*, we compute a Z_n and a Z_f for each *Volume* 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 *Volumes* 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_{Back} then the front facing polygons into Z_n with a color C_{Front}. Should the z-buffer Z_n contain the color C_{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 coursely 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_{PARC} is: $T_{PARC} = T_{Poly} + T_{Ray} + T_{Over}$ where T_{Poly} is the polygon projection time, T_{Ray} is the ray stepping time, and T_{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_{Poly}, T_{Ray}, and T_{Over} for the object S_O as a function of l. In general, as l increases, T_{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_{Ray} decreases exponentially depending on the surface characteristics of S_O and the floating point speed of the machine. The T_{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_{Poly}, T_{Ray}, T_{Over} and T_{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.

5. PARC results

We have performed the analysis of T_{Poly}, T_{Ray}, T_{Over} and T_{PARC} on two structurally different volumetric data sets using a single processor and the geometry engine of a Silicon Graphics 240GTX. An MRI scan of a human head and a confocal microscope scan of a rat hippocampal pyramidal neuron were reconstructed to a voxel resolution of 256×256×166 and 256×256×163, respectively. The MRI head is representative of a fairly solid, well connected, almost spherical structure while

(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 256×256 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.

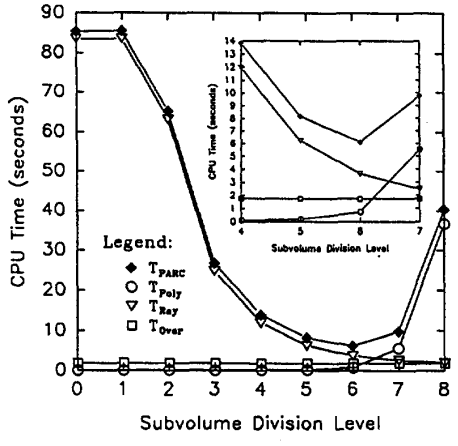

Figure 8: A plot of T_{PARC}, T_{Poly}, T_{Ray}, and T_{Over} as a function of subdivision level l when rendering the MRI human head. Inset: A magnified view of the plot.

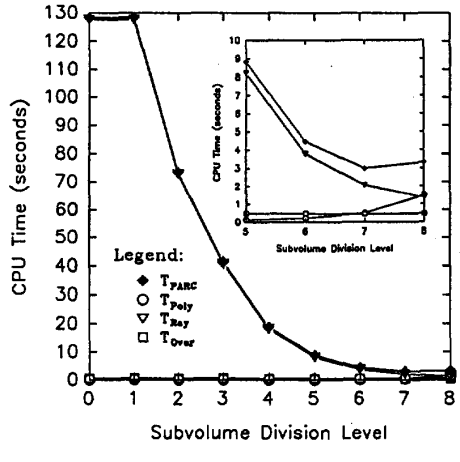

Figure 9: A plot of T_{PARC}, T_{Poly}, T_{Ray}, and T_{Over} as a function of subdivision level l when rendering the neuron. Inset: A magnified view of the plot.

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.

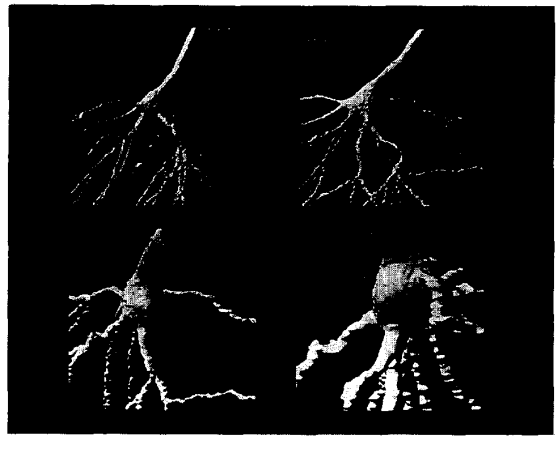

Figure 10: Four frames from a PARC animation depicting a flight through the confocal neuron data.

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 IRI-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

1. Drebin, R. A., Carpenter, L. and Hanrahan, P., "Volume Rendering", *Computer Graphics*, **22**, 4 (August 1988), 64-75.
2. Herman, G. T. and Udupa, J. K., "Display of 3D Digital Images: Computational Foundations and Medical Applications", *IEEE Computer Graphics and Applications*, **3**, 5 (August 1983), 39-46.
3. Hibbard, W. and Santek, D., "Visualizing Large Data Sets in the Earth Sciences", *IEEE Computer*, **22**, 8 (August 1989), 53-57.
4. Kaufman, A., *Volume Visualization*, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.
5. Levoy, M., "Display of Surfaces from Volume Data", *IEEE Computer Graphics and Applications*, **8**, 5 (May 1988), 29-37.
6. Lorensen, W. E. and Cline, H. E., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", *Computer Graphics*, **21**, 4 (July 1987), 163-169.
7. Montine, J. L., "A Procedural Interface for Volume Rendering", *Proceedings Visualization '90*, San Diego, CA, October 1991, 36-44 .
8. Robb, R. A., "A Software System for Interactive and Quantitative Analysis of Biomedical Images", in *3D Imaging in Medicine*, Springer-Verlag, Berlin, 1990.
9. Senft, S. L., Argiro, V. J. and Van Zandt, W. L., "Volume Microscopy of Biological Specimens Based on Non-Confocal Imaging Techniques", *Visualization '90*, San Francisco, CA, October 1990, 424-428.
10. Upson, C. and Keeler, M., "V-BUFFER: Visible Volume Rendering", *Computer Graphics*, **22**, 4 (August 1988), 59-64.
11. Van Zandt, W. and Argiro, V., "A New 'Inlook' on Life", *Unix Review*, **7**, 3 (1989), 52-57.
12. Westover, L., "Footprint Evaluation for Volume Rendering", *Computer Graphics*, **24**, 4 (August 1990), 367-376.
13. Wolfe, R. H. and Liu, C. N., "Interactive Visualization of 3D Seismic Data: A Volumetric Method", *IEEE Computer Graphics and Applications*, **8**, 7 (July 1988), 24-30.
14. Yagel, R., Kaufman, A. and Zhang, Q., "Realistic Volume Imaging", *Proceedings Visualization '90*, San Diego, CA, October 1991, 226-231.
15. Yagel, R. and Kaufman, A., "Template-Based Volume Viewing", *Proceedings Eurographics '92*, Cambridge, UK, September 1992.
16. Yagel, R., Cohen, D. and Kaufman, A., "Discrete Ray Tracing", *IEEE Computer Graphics and Applications*, 1992.

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