VolVis: A Diversified Volume Visualization System

Ricardo Avila[‡], Taosong He[*], Lichan Hong[*], Arie Kaufman[*],
Hanspeter Pfister[*], Claudio Silva[*], Lisa Sobierajski[*], Sidney Wang[*]

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

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

Abstract

VolVis is a diversified, easy to use, extensible, high performance, and portable volume visualization system for scientists and engineers as well as for visualization developers and researchers. VolVis accepts as input 3D scalar volumetric data as well as 3D volume-sampled and classical geometric models. Interaction with the data is controlled by a variety of 3D input devices in an input device-independent environment. VolVis output includes navigation preview, static images, and animation sequences. A variety of volume rendering algorithms are supported, ranging from fast rough approximations, to compression-domain rendering, to accurate volumetric ray tracing and radiosity, and irregular grid rendering.

1. Introduction

The visualization of volumetric data has aided many scientific disciplines ranging from geophysics to the biomedical sciences. The diversity of these fields coupled with a growing reliance on visualization has spawned the creation of a number of specialized visualization systems. These systems are usually limited by machine and data dependencies and are typically not flexible or extensible. A few visualization systems have attempted to overcome these dependencies (e.g., AVS, SGI Explorer, Khoros) by taking a data-flow approach. However, the added computational costs associated with data-flow systems results in poor performance. In addition, these systems require that the scientist or engineer invest a large amount of time understanding the capabilities of each of the computational modules and how to effectively link them together.

VolVis is a volume visualization system that unites numerous visualization methods within a comprehensive visualization system, providing a flexible tool for the scientist and engineer as well as the visualization developer and researcher. The *VolVis* system has been designed to meet the following key objectives:

Diversity: *VolVis* supplies a wide range of functionality with numerous methods provided within each functional component. For example, *VolVis* provides various projection methods including ray casting, ray tracing, radiosity, Marching Cubes, and splatting.

Ease of use: The *VolVis* user interface is organized into functional components, providing an easy to use visualization system. One advantage of this approach over data-flow systems is that the user does not have to learn how to link numerous modules in order to perform a task.

Extensibility: The structure of the *VolVis* system is designed to allow a visualization programmer to easily add new representations and algorithms. For this purpose, an extensible and hierarchical abstract model was developed [1] which contains definitions for all objects in the system.

Portability: The *VolVis* system, written in C, is highly portable, running on most Unix workstations supporting X/Motif. The system has been tested on Silicon Graphics, Sun, Hewlett-Packard, Digital Equipment Corporation, and IBM workstations and PCs.

Freely available: The high cost of most visualization systems and difficulties in obtaining their source code often lead researchers to write their own tools for specific visualization tasks. *VolVis* is freely available as source code.

2. System Overview

Figure 1 shows the *VolVis* pipeline, indicating some paths that input data could take through the *VolVis* system in order to produce visualization output. Two of the basic input data classes of *VolVis* are volumetric data and 3D geometric data. The input data is processed by the Modeling and Filtering components of the system to produce either a 3D volume model or a 3D geometric surface model of the data. For example, geometric data can be converted into a volume model by the Modeling component of the system, as described in Section 3, to allow for volumetric graphic operations. A geometric surface model can be created from a volume model by the process of surface extraction.

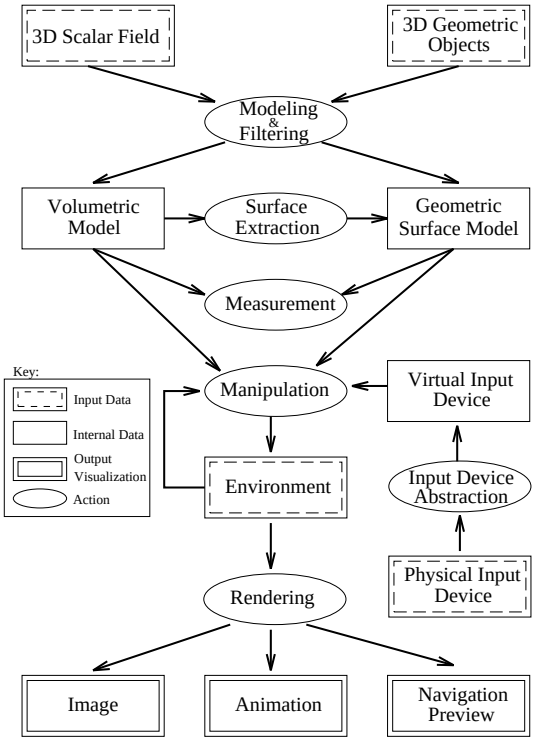

Figure 1: The VolVis pipeline.

The Measurement component can be used to obtain quantitative information from the data models. Surface area, volume, histogram and distance information can be extracted from volumes using one of several methods. Isosurface volume and surface area measurements can be taken either on an entire volume or on a surface-tracked section. Additionally, surface areas and volumes can be computed using either a simple non-interpolated voxel counting method or a Marching Cubes [8] based measurement method. For geometric surface models, surface area, volume, and distance measurements can be performed.

Most of the interaction in *VolVis* occurs within the Manipulation component of the system. This part of the system allows the user to modify object parameters such as color, texture, and segmentation, and viewing parameters such as image size and field of view. Within the Navigation section of the Manipulation component, the user can interactively modify the position and orientation of the volumes, the light sources, and the view. This is closely connected to the Animation section of the Manipulation component, which allows the user to specify animation sequences either interactively or with a set of transformations to be applied to objects in the scene. The Manipulation component is described in Section 4.

The Rendering component encompasses several different rendering algorithms, including geometry-based techniques such as Marching Cubes, global illumination methods such as ray tracing and radiosity, and direct volume rendering algorithms such as splatting. The Rendering component is described in Section 5.

The Input Device component of the system maps physical input device data into a device independent representation that is used by various algorithms requiring user interaction. As a result, the *VolVis* system is input device independent, as described in Section 6.

3. Modeling

A primary responsibility of the Modeling component is the voxelization of geometric data into volumetric model representations. Voxelizing a continuous model into a volume raster of voxels requires a geometrical sampling process which determines the values to be assigned to voxels of the volume raster. To reduce object space aliasing, we adopt a volume sampling technique [14] that estimates the density contribution of the geometric objects to the voxels. The density of a voxel is determined by a filter weight function which is proportional to the distance between the center of the voxel and the geometric primitive. In our implementation, precomputed tables of densities for a predefined set of geometric primitives are used to assign the density value of each voxel. For each voxel visited by the voxelization algorithm, the distance to the predefined primitive is used as an index into the tables.

Figure 2: A volumetric ray traced image of a volume-sampled geometric wine bottle and glasses.

Since the voxelized geometric objects are represented as volume rasters of density values, we can essentially treat them as sampled or simulated volume data sets, such as 3D medical imaging data sets, and employ one of many volume rendering techniques for image generation. One advantage of this approach is that volume rendering carries the smoothness of the volume-sampled objects from object space over into image space. Hence, the silhouette of the objects, reflections, and shadows are smooth. Furthermore, by not performing any geometric ray-object intersections or geometric surface normal calculations, a large amount of rendering time is saved. In addition, CSG operations between two volume-sampled geometric models are accomplished at the voxel level during voxelization, thereby reducing the original problem of evaluating a CSG tree of such operations down to a Boolean operation between pairs of voxels. Figure 2 shows a ray traced image of a wine bottle and glasses that were modeled by CSG operations on volume-sampled geometric objects. The upper right window in Figure 3 shows a ray traced image of a nut and bolt that were also modeled by CSG operations.

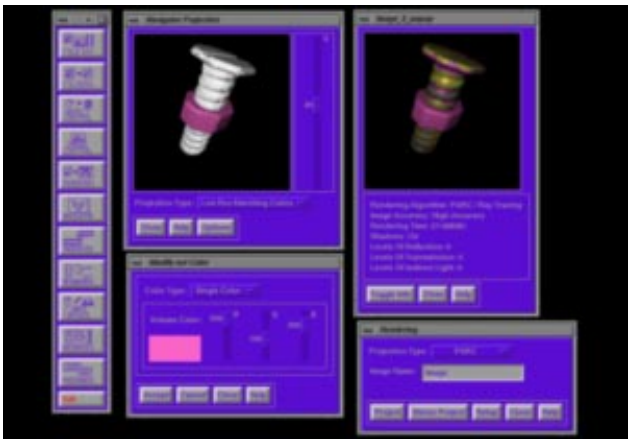

Figure 3: An example VolVis session. The nut and bolt are volume-sampled geometric models.

4. Manipulation

The Manipulation component of *VolVis* consists of three sections: the Object Control section, the Navigation section, and the Animation section. The Navigation and Animation sections are also referred to as the Navigator and Animator, respectively. Both the Navigator and Animator produce output visualization, shown in Figure 1 as Navigation Preview and Animation, respectively.

The Object Control section of the system is extensive, allowing the user to manipulate parameters of the objects in the scene. This includes modifications to the color, texture, and shading parameters of each volume, as well as more complex operations such as positioning of cut planes and data segmentation. The color and position of all light sources can be interactively manipulated by the user. Also, viewing parameters, such as the final image size, and global parameters, such as ambient lighting and the background color, can be modified.

The Navigator allows the user to interactively manipulate objects within the system. The user can translate, scale and rotate all volumes and light sources, as well as the view itself. The Navigator can also be used to interactively manipulate the view in a manner similar to a flight simulator. To provide interactive navigation speed, a fast rendering algorithm was developed which involves projecting reduced resolution representations of all objects in the scene. This task is relatively simple for geometric objects, where calculating, storing, and projecting a polygonal approximation requires little overhead. However, when considering a volumetric isosurface the cost of an additional representation increases considerably. A simple and memory efficient method available within the Navigator creates a reduced resolution representation of an isosurface by uniformly subdividing the volume into boxes and projecting the outer faces of all the boxes that contain a portion of the isosurface. These subvolumes serve a dual purpose in that they are also used by the PARC (Polygon Assisted Ray Casting) acceleration method [1] during ray casting and ray tracing.

Although the PARC subvolume representation can be stored as a compact list of subvolume indices, the resulting images are boxy and uninformative for many data sets. To overcome this problem, another method is provided which utilizes a reduced resolution Marching Cubes representation of an isosurface. In order to reduce the amount of data required for this representation, edge intersections used to compute triangle vertices are restricted to one of four possible locations. This results in much smoother images which are typically more informative than the uniform subdivision method. The Navigator also supports the other *VolVis* rendering techniques that are described in Section 5, although interactive projection rates with these methods can be achieved only on high-end workstations.

The Animator also allows the user to specify transformations to be applied to objects within the scene, but as opposed to the Navigator which is used to apply a single transformation at a time, the Animator can be used to specify a sequence of transformations to produce an animation. The user can preview the animation using one of the fast rendering techniques within the Navigator. The user can then select a more accurate and time consuming rendering technique, such as volumetric ray tracing, to

create a high quality animation. In addition to simple rotation, translation and scaling animations, the Navigator can be used to interactively specify a ‘‘flight path’’, which can then be passed to the Animator, and rendered to create an animation.

An example session of the *VolVis* system is shown in Figure 3. The long window on the left is the main *VolVis* interface window, with buttons for each of the major components of the system. The current scene is displayed in the Navigator window on the left, and in the Rendering image window on the right. A low resolution Marching Cubes technique was used in the Navigator, while a ray casting technique using the PARC acceleration method was employed during rendering.

5. Rendering

Rendering is one of the most important and extensive components of the *VolVis* system. For the user, speed and accuracy are both important, yet often conflicting aspects of the rendering process. For this reason, a variety of rendering techniques have been implemented within the *VolVis* system, ranging from the fast, rough approximation of the final image, to the comparatively slow, accurate rendering within a global illumination model. Also, each rendering algorithm itself supports several levels of accuracy, giving the user an even greater amount of control. In this section, a few of the rendering techniques developed for the *VolVis* system are discussed.

Two of the *VolVis* rendering techniques, volumetric ray tracing, and volumetric radiosity, are built upon global illumination models. Standard volume rendering techniques, which are also supported by *VolVis*, typically employ only a local illumination model for shading, and therefore produce images without global effects. Including a global illumination model within a visualization system has several advantages. First, global effects can often be desirable in scientific applications. For example, by placing mirrors in the scene, a single image can show several views of an object in a natural, intuitive manner leading to a better understanding of the 3D nature of the scene. Also, complex surfaces are often easier to render when represented volumetrically than when represented by high-order functions or geometric primitives, as described in Section 3. Volumetric ray tracing is described in Section 5.1 and volumetric radiosity is discussed in Section 5.2.

In order to reduce the large storage and transmission overhead as well as the volume rendering time for volumetric data sets, a data compression technique is incorporated into the *VolVis* system. This technique allows volume rendering to be directly performed on the compressed data and is described in Section 5.3.

Although many scanning devices create data sets that are inherently rectilinear, this restriction poses problems for fields in which an irregular data representation is necessary. These fields include computational fluid dynamics, finite element analysis, and meteorology. Therefore, support was added for irregularly gridded data formats in the *VolVis* system, as discussed in Section 5.4.

5.1. Volumetric Ray Tracing

The volumetric ray tracer provided within the *VolVis* system is intended to produce accurate, informative images [11]. In classical ray tracing, the rendering algorithm is designed to generate images that are accurate according to the laws of optics. In *VolVis*, the ray tracer must handle classical geometric objects as well as volumetric data, and strict adherence to the laws of optics is not always desirable. For example, a scientist may wish to view the maximum value along the segment of a ray passing through a volume, instead of the optically-correct composited value. Figure 4 illustrates the importance of including global effects in a maximum-value projection of a hippocampal pyramidal neuron data set which was obtained using a laser-scanning confocal microscope. Since maximum-value projections do not give depth information, a floor is placed below the cell, and a light source above the cell. This results in a shadow of the cell on the floor, adding back the depth information lost by the maximum-value projection.

In order to incorporate both geometric and volumetric objects into one scene, the classical ray tracing intensity equation, which is evaluated only at surface locations, must be extended to include volumetric effects. The intensity of light, $I_\lambda(x, \vec{\omega})$, for a given wavelength λ, arriving at a position x, from the direction $\vec{\omega}$, can be computed by:

$$I_\lambda(x, \vec{\omega}) = I_{v\lambda}(x, x') + \tau_\lambda(x, x') I_{s\lambda}(x', \vec{\omega}) \qquad (1)$$

where x' is the first surface intersection point encountered along the ray $\vec{\omega}$ originating at x. $I_{s\lambda}(x', \vec{\omega})$ is the intensity of light at this surface location, and can be computed with a standard ray tracing illumination equation [15]. $I_{v\lambda}(x, x')$ is the volumetric contribution to the intensity along the ray from x to x', and $\tau_\lambda(x, x')$ is the attenuation of $I_{s\lambda}(x', \vec{\omega})$ by any intervening volumes. These values are determined using volume rendering techniques, based on a transport theory model of light propagation [7]. The basic idea is similar to classical ray tracing, in that rays are cast from the eye into the scene, and surface shading is performed on the closest surface intersection point. The

difference is that shading must be performed for all volumetric data that are encountered along the ray while traveling to the closest surface intersection point.

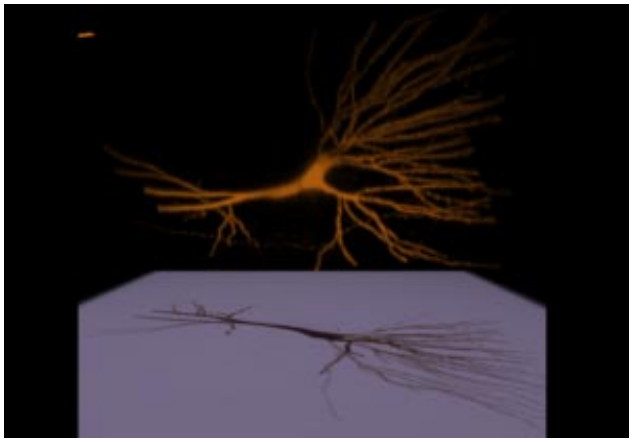

Figure 4: A volumetric ray traced image of a cell using a maximum-value projection.

For photo-realistic rendering, the user typically wants to include all of the shading effects that can be calculated within a given time limit. However, visualization users may find it necessary to view volumetric data with no shading effects, such as when using a maximum-value projection. In *VolVis*, the user has control over the illumination equations for both volumetric and geometric objects, and can specify, for each object in the scene, which shading effects should be computed. For example, in Figure 4 no shading effects were included for the maximum-value projection of the cell, while all parts of the illumination equation were considered when shading the geometric polygon. In another example, the user may place a mirror behind a volumetric object in a scene in order to capture two views in one image, but may not want the volumetric object to cast a shadow on the mirror, as shown in Figure 5. The head was obtained using magnetic resonance imaging, with the brain segmented from the same data set. The mirror is a volume-sampled polygon that was created using the modeling technique described in Section 3.

5.2. Volumetric Radiosity

The ray tracing algorithm described in the previous section can be used to capture specular interactions between objects in a scene. In reality, most scenes are dominated by diffuse interactions, which are not accounted for in the standard ray tracing illumination model. For this reason, *VolVis* also contains a radiosity algorithm for volumetric data. Volumetric radiosity includes the classical surface ‘‘patch’’ element as well as a ‘‘voxel’’ element. As opposed to previous methods that use participating media to augment geometric scenes [10], this method is intended to render scenes that may solely consist of volumetric data. Each patch or voxel element can emit, absorb, scatter, and transmit light. Both isotropic and diffuse emission and scattering of light are allowed, where ‘‘isotropic’’ implies directional independence, and ‘‘diffuse’’ implies Lambertian reflection (i.e., dependent on normal or gradient). Light entering an element that is not absorbed or scattered by the element is transmitted unchanged.

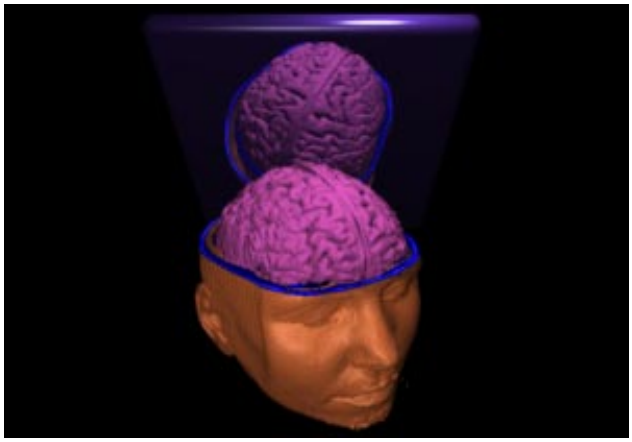

Figure 5: A volumetric ray traced image of a human head.

In order to cope with the high number of voxel interactions required, a hierarchical technique similar to [5] is used. An iterative algorithm [2] is then used to shoot voxel radiosities, where several factors govern the highest level in the hierarchy at which two voxels can interact. These factors include the distance between the two voxels, the radiosity of the shooting voxel, and the reflectance and scattering coefficients of the voxel receiving the radiosity. This hierarchical technique can reduce the number of interactions required to converge on a solution by more than four orders of magnitude.

After the view-independent radiosities have been calculated, a view-dependent image is generated using a ray casting technique, where the final pixel value is determined by compositing radiosity values along the ray. Figure 6 shows a scene containing a volumetric sphere, polygon, and light source. The light source isotropically emits light, and both the sphere and the polygon diffusely reflect light. The light source is above the sphere and directly illuminates the top half of the sphere. The bottom half of the sphere is indirectly illuminated by light diffusely reflected from the red polygon.

5.3. Compression Domain Volume Rendering

Another rendering method incorporated in *VolVis* is a data compression technique for volume rendering. Our volume compression technique is a 3D generalization of the JPEG still image compression algorithm [13] , with one important exception: the transform is a discrete Fourier transform rather than a discrete cosine transform. The original 3D data is subdivided into $M \times M \times M$ subcubes, where each subcube is Fourier transformed to the frequency domain through a 3D discrete Fourier transform. Each of the 3D Fourier coefficients in each subcube is then quantized, and the resulting 3D quantized frequency coefficients are organized as a linear sequence through a 3D zig-zag order. The resulting sequence of Fourier transform coefficients is then fed into an entropy encoder that consists of run-length coding and Huffman coding.

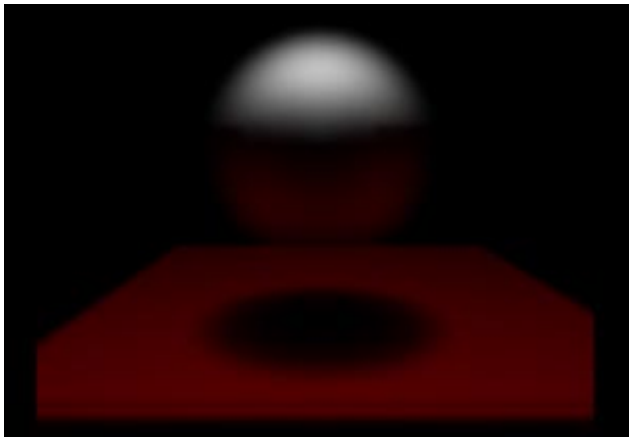

Figure 6: A volumetric radiosity projection of a voxelized sphere and polygon.

To render in the compressed domain, we use a new class of volume rendering algorithms [3, 9, 12] that are based on the Fourier projection slice theorem. It states that a projection of the 3D data volume from a certain direction can be obtained by extracting a 2D slice perpendicular to the view direction out of the 3D Fourier spectrum and then applying an inverse Fourier transform. In our approach we apply the Fourier projection slice theorem to each subcube in the Fourier domain, which results in a set of 2D planes in the spatial domain called subimages that are composited using spatial compositing to get the final projection of the original 3D data set.

Using our compression-domain rendering approach, we were able to achieve high compression ratios while maintaining image quality. Figure 7 shows a CT scan of a lobster that was rendered in the compressed domain.

We are currently investigating the adaptation of subcube sizes to various spatial or frequency domain criteria, such as subcube AC coefficient energy, which is a measure of subcube activity, sample density, and coefficient distribution.

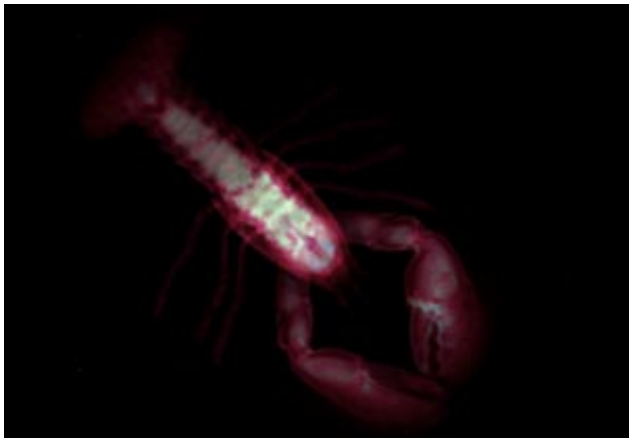

Figure 7: Compression domain volume rendering of a lobster.

5.4. Irregular Grid Rendering

An intuitive way to visualize irregularly gridded data sets is to resample the data into a regular grid format. Unfortunately, it is quite difficult to find a resampling method that preserves details yet does not require a large amount of memory. Consequently, we chose to extend the traditional volume rendering algorithms to process the irregularly gridded data directly. For example, we have extended the ray tracing algorithms in *VolVis* to visualize data represented in a spherical coordinate system, with grids that are unevenly spaced in r, evenly spaced in θ, and unevenly spaced in ϕ. When rendering, we could cast rays into the scene, uniformly stepping and compositing along each ray. A problem with uniform stepping is that it inevitably misses detailed information. To avoid this problem, we traverse the ray cell by cell in the volume, in a method similar to Garrity [4].

6. Input Devices

The Input Device component of the *VolVis* system allows the user to control a variety of input devices in an input device independent environment. For example, to control the Navigator, the user can utilize a variety of physical input devices such as a keyboard, a mouse, a Spaceball, and a DataGlove. To achieve this, we have developed the *device unified interface* (DUI) [6], which is a generalized and easily expandable protocol for communication between applications and input devices.

The key idea of the DUI is to convert raw data received from different input sources into unified format parameters of a ‘‘virtual input device’’. Depending on the requirements of the application, the parameters may include a number of 3D positions and orientations as well as abstract actions. The abstract actions include direct and simple actions like mouse or Spaceball button clicks , and complex dynamic actions like two hand gestures or ‘‘snap-dragging’’. The conversion from the real device operations to abstract actions is performed by the selected simulation methods which are incorporated into the DUI.

The most important advantage of employing the virtual input device paradigm is input device independence. In the DUI, each application is interactively assigned a virtual input device, whose configuration is also interactively decided. Modification of either the input device component or the application does not affect other parts of the system. The simulation methods used to convert different kinds of raw information into the unified format are often difficult to design. For example, the recognition of dynamic gestures of a DataGlove is fairly difficult. By using the DUI, new simulation methods can be easily incorporated and tested with no adverse effect on the application or the other parts of the Input Device component.

However, in order to fully utilize the capability of different devices, a virtual input device should not totally hide the device dependent information since different devices are suitable for different applications. For example, it is harder to control the Navigator with the Spaceball than with the DataGlove, since the six degrees of freedom provided by the Spaceball are not entirely independent as they are in the DataGlove. In the DUI, a *device information-base* is associated with each virtual input device. All of the device dependent information related to a virtual device is classified and stored in an abstract form, which is then queried by an application when necessary [6].

We are currently working on the expansion of the DUI into a general-purpose interaction model. The model is created based on lightweight threads and is designed to handle simultaneous high bandwidth, multimodal, and complex input from multiple users, even through the network. A general abstract action and input device description language is also being studied.

7. Implementation

Two major concerns during the implementation of *VolVis* have been to ensure that the system could be expanded to include new functionality and techniques, and that the system would be relatively easy to port to new platforms. Therefore, the development of the *VolVis* system required the creation of a comprehensive, flexible, and extensible abstract model [1]. The model is organized hierarchically, beginning with low-level building blocks which are then used to construct higher-level structures. For example, low-level objects such as vectors and points can be combined to create a coordinate system, while at the highest level the World structure contains the state of every object in the system. The World structure includes Lights, Volumes, Views, global cut planes, and global shading parameters. Each Volume structure includes color and texture information, local shading parameters, local cut planes, and data which may be either geometric descriptions, or rectilinear or irregularly gridded data.

The abstract model is flexible in that a structure can assume one of many representations. For instance, a segmentation structure can consist of either a threshold or opacity and color transfer functions. A natural consequence of flexibility is expandability. Since the objects in the abstract model already provide for numerous representations, the addition of a new segmentation type, shading type, or even data type is fairly simple.

The *VolVis* system requires only Unix and X/Motif to run. Unfortunately, only simple two-dimensional graphics operations are supported in X. Therefore, all viewing transformations, shading, and hidden surface removal must be done in software. This greatly reduces the rendering speed for the geometry-based projection routines used in the Navigation section, and therefore also reduces the overall interactivity of the system. Since many Unix workstations now include graphics hardware, interactivity can be maintained by utilizing the graphics language of the workstation. To avoid rewriting large sections of the code, we have developed a library of basic graphics functions that are used throughout the *VolVis* code. This simplifies the process of porting the system to a new workstation that has a different graphics language, since only the graphics function library must be rewritten.

8. Conclusions

The *VolVis* system for volume visualization has been used for many tasks in diverse applications and situations. First, *VolVis* has been used to test new algorithms for rendering, modeling, animation generation, and computer-human interaction. Due to the flexible nature of the abstract model, testing new ideas within the system is much easier and less time consuming than writing a new application for each new algorithm. *VolVis* has also been used by scientists and researchers in many different areas. For example, neurobiologists have used *VolVis* to navigate through the complex dendritic paths of nerve cells, which

is extremely useful since the function of nerve cells is closely tied to their structure (see Figure 4).

VolVis is a rapidly growing system, with new plans for future development continually being considered. Since the system is currently being used by many research labs and visualization developers, feedback from these