1998 International Computer Symposium
Workshop on Computer Graphics and Virtual Reality
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

A Java-Based Testbed for Volume Visualization

Kwan-Liu Ma
Institute for Computer Applications in Science and Engineering
Mail Stop 403, NASA Langley Research Center
Hampton, Virginia 23681-2199, U.S.A.
Email: kma@icase.edu

Abstract

This paper describes a Java-based visualization software testbed for experimenting with new ideas and rendering codes, as well as approaches for remote and collaborative visualization. This system allows visualization across the Internet from a web browser supporting the Java language. We also address issues in designing more intuitive interfaces and better user interaction. The system's modular design allows parts of the system to be easily replaced to compare the performance of differing approaches to a given design problem in the system. Particularly, we have designed a graph-based interface which helps users better understand how certain rendering parameter changes affect the visualization results, keep track of decisions toward some visualization discovery, and communicate their findings with others.

1. Introduction

Research in data visualization technology encompasses a wide variety of topics including data management, abstraction and representation of the object to be visualized, design of rendering algorithms, interaction techniques, the process of visualization, user interface and I/O devices, visualization system hardware and software, validation of visualization results, visual perception problems, and application specific problems. It would be advantageous to visualization researchers if they can study either a single topic or multiple topics and their relationship in the same framework. This paper describes such a framework, a Java-based testbed designed for volume visualization applications. Java supports network computing and 2D (and 3D) graphics which allow us to focus on the development of high level concepts of our design rather than low level implementation.

One important feature of this testbed is its modular design. Any component can be replaced with another component which communicates in the same manner as the original component to the other parts of the system, but which implements its functionality differently. For example, the rendering program could be replaced with another renderer which handles a different type of data, such as data on a curvilinear or unstructured grid, or a renderer which produces very high quality images but which is more expensive than the algorithms we currently use. This modularity is also relevant to the user interface because it allows different approaches to the user interface to be used and tested at the same time. Because the rendering algorithm can be changed, our user interface can be used in concert with any volume rendering algorithm as long as the algorithm is able to use the rendering parameters our system handles in order to render an image. This capability is similar to the popular data flow model [20] adopted by many commercial visualization systems.

The other important feature of this testbed is a graph based user interface that we have been developing for representing not only the results but also the process of a volume visualization session. A preliminary design of the graph approach is reported in [16]. In essence, as images are rendered, they are connected to other images in a graph based on their rendering parameters. The user can take advantage of the information in this graph to understand how certain rendering parameter changes affect a dataset, making the visualization process more efficient. Because the graph contains more information than is contained in an unstructured history of images, the image graph is also helpful for collaborative visualization. Finally, we should point out that using the Java programming language allows us to implement the aforementioned and other features of the testbed in a more straightforward manner.

This paper is organized as follows. Section 2. gives an introduction of volume rendering. Section 3. presents the system architecture of the testbed, which consists of an applet written in the Java language which runs in any web browser which supports Java, a render "server" process which manages communication with active web clients, and a volume rendering process, currently implemented using two different volume renderers, which are used depending on rendering requirements. Section 4. discusses design issues for the key steps in the typical volume visualization process. Section 5. describes the graph based interface and its impact on the overall visualization process. In Section6., we discuss how our overall design supports collaborative visualization. The final section offers some concluding remarks and suggests directions for future work.

2. Volume Rendering

Current computing and sensing technologies allow scientists to study physical phenomena in three spatial dimensions at high resolution. Appropriate techniques are needed for visualizing the resulting *volume data* with which usually a grid structure is associated. Volume rendering refers to techniques directly rendering sampled scalar fields of three dimensions without constructing intermediate polygonal representations. Generally, it can display more information in a single visualization than techniques such as isosurface or slicing. It is especially effective for displaying features that are either very fine or difficult to define analytically.

Several volume rendering algorithms have been invented including the ray casting [8], projection [4], splatting [22] and shear warp [7] methods. Various optimization and acceleration techniques for volume rendering have also been developed including encoding object space coherence [9], encoding image space coherence [23], hardware assisted [17, 1] and parallel [12, 6] methods. These advances make volume rendering a practical visualization solution.

Volume rendering is very computationally expensive. While previous research work on volume visualization has mainly focused on improving rendering performance, the study of the entire process of volume visualization is equally important to make volume rendering a useful data analysis technique. This is particularly true since volume data exploration often involves a trial and error process of parameter specification. The key elements that constitute an efficient visualization process thus include not only fast rendering rates but also an intuitive user interface, capabilities to incorporate domain knowledge into the data filtering step, and a mechanism for tracking (and sharing) the visualization process and results. This paper also reports our research results in these directions.

The objective of our work is to optimize the overall volume visualization process and support remote and collaborative visualization. Several other systems have addressed the problem of remote volume visualization. For example, the VizWiz system [14] performs isosurface and slice rendering over the Internet. MPIRE [15] is a parallel rendering system that offers a web-based interface which allows the user to select from three different platforms and two different volume rendering methods to accomplish a rendering task. An on-going research is VolRenCV [21] which is web-based volume rendering comparison tool. Its web interface provides volume rendering with multiple algorithms and a set of metrics for comparing different rendering parameters.

3. System Architecture

The main components of our system are the render server, the communication server, and clients. Figure 1 presents the system architecture of the testbed. In this section each of these components is discussed in further detail.

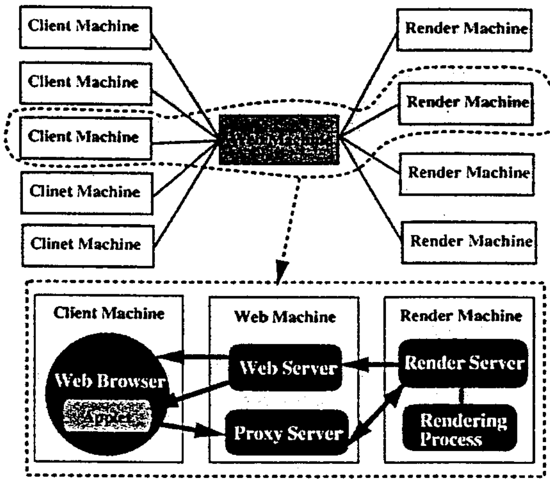

Figure 1: System Architecture of the testbed.

3.1 Render Server

The render server is the process which actually performs the rendering of the image to be displayed on the client. This server is started by the proxy server when needed in order to fulfill a rendering request from a client. Because the system is modular, different implementations of the render server can be used interchangeably. The current implementation uses both the shear warp algorithm [7] and a ray casting algorithm [10] to render data on Cartesian grids. The system selects which algorithm to use based on the size of the image requested and our performance data. With our current configuration, the ray casting algorithm performs better on smaller images and the shear warp algorithm performs better on larger images. In addition, the ray casting algorithm is used to render zoomed images as the shear warp implementation does not support supersampling.

Both rendering algorithms can benefit from preprocessing. To speed up subsequent rendering calculations, the shear warp algorithm as used in our system uses two stages of preprocessing: classification and octree encoding. Octree encoding is dependent on the opacity transfer function used. This means that if the render server's preprocessing is up to date, a new image of the dataset can be rendered very quickly if only the color transfer function or the view transformation is changed. On the other hand, the ray casting renderer uses a view-dependent ray cache to achieve very fast rendering rates independent of changes of transfer functions.

3.2 Proxy Server

The proxy server handles the communication of the clients and the render server. It provides mechanisms for access control, tracking the state of a client's session, load balancing, and caching. When a client is started, it connects to the

2. Volume Rendering

Current computing and sensing technologies allow scientists to study physical phenomena in three spatial dimensions at high resolution. Appropriate techniques are needed for visualizing the resulting *volume data* with which usually a grid structure is associated. Volume rendering refers to techniques directly rendering sampled scalar fields of three dimensions without constructing intermediate polygonal representations. Generally, it can display more information in a single visualization than techniques such as isosurface or slicing. It is especially effective for displaying features that are either very fine or difficult to define analytically.

Several volume rendering algorithms have been invented including the ray casting [8], projection [4], splatting [22] and shear warp [7] methods. Various optimization and acceleration techniques for volume rendering have also been developed including encoding object space coherence [9], encoding image space coherence [23], hardware assisted [17, 1] and parallel [12, 6] methods. These advances make volume rendering a practical visualization solution.

Volume rendering is very computationally expensive. While previous research work on volume visualization has mainly focused on improving rendering performance, the study of the entire process of volume visualization is equally important to make volume rendering a useful data analysis technique. This is particularly true since volume data exploration often involves a trial and error process of parameter specification. The key elements that constitute an efficient visualization process thus include not only fast rendering rates but also an intuitive user interface, capabilities to incorporate domain knowledge into the data filtering step, and a mechanism for tracking (and sharing) the visualization process and results. This paper also reports our research results in these directions.

The objective of our work is to optimize the overall volume visualization process and support remote and collaborative visualization. Several other systems have addressed the problem of remote volume visualization. For example, the VizWiz system [14] performs isosurface and slice rendering over the Internet. MPIRE [15] is a parallel rendering system that offers a web-based interface which allows the user to select from three different platforms and two different volume rendering methods to accomplish a rendering task. An on-going research is VolRenCV [21] which is web-based volume rendering comparison tool. Its web interface provides volume rendering with multiple algorithms and a set of metrics for comparing different rendering parameters.

3. System Architecture

The main components of our system are the render server, the communication server, and clients. Figure 1 presents the system architecture of the testbed. In this section each of these components is discussed in further detail.

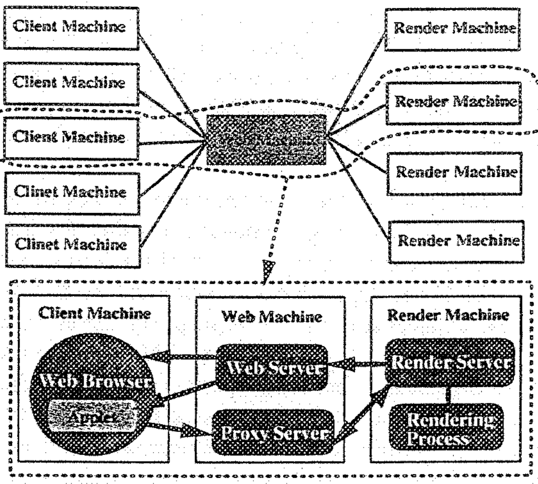

Figure 1: System Architecture of the testbed.

3.1 Render Server

The render server is the process which actually performs the rendering of the image to be displayed on the client. This server is started by the proxy server when needed in order to fulfill a rendering request from a client. Because the system is modular, different implementations of the render server can be used interchangeably. The current implementation uses both the shear warp algorithm [7] and a ray casting algorithm [10] to render data on Cartesian grids. The system selects which algorithm to use based on the size of the image requested and our performance data. With our current configuration, the ray casting algorithm performs better on smaller images and the shear warp algorithm performs better on larger images. In addition, the ray casting algorithm is used to render zoomed images as the shear warp implementation does not support supersampling.

Both rendering algorithms can benefit from preprocessing. To speed up subsequent rendering calculations, the shear warp algorithm as used in our system uses two stages of preprocessing: classification and octree encoding. Octree encoding is dependent on the opacity transfer function used. This means that if the render server's preprocessing is up to date, a new image of the dataset can be rendered very quickly if only the color transfer function or the view transformation is changed. On the other hand, the ray casting renderer uses a view-dependent ray cache to achieve very fast rendering rates independent of changes of transfer functions.

3.2 Proxy Server

The proxy server handles the communication of the clients and the render server. It provides mechanisms for access control, tracking the state of a client's session, load balancing, and caching. When a client is started, it connects to the proxy server. The server can choose to allow the client to connect or not, based on any desired set of rules. For example, the proxy server might want to limit access to only ten users at a time. The server could also only accept connections from within a certain network, or according to some other metric.

If a client is allowed to connect, the server begins keeping track of the rendering parameters associated with the current client session. A separate set of rendering parameters is maintained for each client connected, so that many clients can connect at a time. A single proxy server can also connect to a variety of render servers. This allows a proxy server to distribute a rendering load by assigning separate rendering tasks to rendering processes running on different machines. While our current implementation does not support the division of the task of rendering a single image among multiple servers, this type of separation is a possibility for future exploration.

The proxy server also implements a caching mechanism to try to reduce the number of images that need to be rendered. When an image is generated, the rendering parameters used to generate that image are stored along with the image. If another image is requested later using the same rendering parameters, the initial image is used.

The shear warp algorithm can take advantage of preprocessing in order to reduce rendering time. The proxy server determines what type of preprocessing to do based on changes in the state of the clients. As new images are produced, the proxy server tells the render server to do the correct preprocessing.

3.3 Client

The Java client displays a user interface for the system. The interface allows the user to upload data, filter the data, set and adjust rendering parameters, displays the images produced by the render server, and organizes a concise history of the visualization process conducted by a user. The rest of the paper is devoted to the design of user interfaces, and an mechanism for organizing and presenting the visualization process conducted.

4. The Volume Visualization Process

Data exploration is inherently an iterative process which includes multiple steps: loading data, filtering, mapping, rendering, and evaluating visualization results. Each step is relevant to the goal of achieving efficient visualization. In this section, we focus our discussion more on those topics neglected by previous research.

4.1 Loading Data

Presently, the system allows the render server to upload volume data from any publicly accessible URL. The user must specify the dimensions of the data and the data must be on a regular grid. These restrictions are due to the renderer which the testbed currently uses, however, the renderer could be replaced with another one which does not have these requirements in order to improve the flexibility of the system. Once the user has entered the required information using the client, the server downloads and preprocesses the dataset. The data is then ready for the user to view.

4.2 Filtering

Filtering can involve a variety of operations: interpolating from scattered data to a regular grid, smoothing, segmentation, and others. For example, presently the two renderers installed in the testbed were optimized for byte data. If the volume dataset contains 16-bit or 32-bit values, a quantization step is required. Quantization should be done according to the characteristics of the data to bring out the most important features in the data. Quantization techniques and other filtering techniques will not be discussed in this paper. An interactive user interface has been implemented for specifying the quantization function.

4.3 Mapping

A visual representation of volume data is made by mapping raw or filtered data values to colors and different levels of opacity. A projection of these color and opacity values of the volume elements based on their depth orders produces an image. This mapping step is the so called classification step in medical imaging.

4.3.1 Color transfer function

In the context of volume visualization, the color transfer function specifies a mapping from values in the volumetric dataset to color values used when rendering a representation of the dataset. Manipulating the color transfer function lets one change the color of specific ranges of values in the dataset. This manipulation is useful for making certain features of the dataset more prominent or less prominent during the process of data exploration.

Some systems approach this problem by letting the user specify three distinct mappings. The user defines a function which maps the data value to the intensity of red color, as well as two other functions for the green and blue intensity. The mapping of data value to color is computed by producing an RGB value from the values of the red, green, and blue functions at each data value. Existing volume visualization systems are equipped with more or less the same interface as ICol's [5] for making color and opacity transfer functions.

While most previous systems use the RGB model for color selection, it has been shown that the HSV (hue, saturation, value) model [19] is more intuitive. However, it has also been shown that color model has no effect on the speed and accuracy of color selection [2]; rather, it is visual feedback which has a more significant effect. Further experiments have also been done to qualitatively analyze the influence of visual feedback on the color selection process [3].

Our approach to color selection is based on the HSV model. The color selection process involves linear interpolation between user specified points on the color transfer function. After each interpolation point is added to the color transfer function, the system automatically interpolates in RGB space between the points of differing colors to produce the color transfer function. In order to edit the color transfer function, the user can remove interpolation points, or drag them to another location on the transfer function with the mouse. The system automatically updates the color transfer function in real time in response to these operations.

We also allow the user to select from a collection of predefined color transfer functions such as rainbow and gray scale maps. In addition, we have implemented two automatic color map generation programs which use the histogram of the dataset. These are useful as standard starting points to explore unfamiliar datasets. With the testbed, it becomes fairly easy to test these different approaches.

4.3.2 Opacity transfer function

The opacity transfer function is used by the renderer to determine the importance (and thus visibility) of a certain voxel of the dataset according to the values of the data adjacent to or inside of that voxel. The opacity function maps values in the dataset to values between 0 (completely transparent) and 1 (completely opaque). We explored several different user interface approaches to the task of specifying an opacity transfer function. We tried an approach, also used by a variety of other systems, where the user described the desired opacity mapping by adding or subtracting the value of a linear, step, or sigmoid function from the current value of the opacity function in order to incrementally build the function. This approach was difficult for new users to understand, perhaps because it required the user to interact with the opacity mapping tool in a variety of modes.

We implemented a simpler interface, which allowed the user to define the opacity function by sketching it with the mouse. Users did not have trouble understanding how to use this interface, but there were still a few difficulties. In order to render a useful image using a volume renderer, the opacity transfer function usually must map most data values to low opacities, or a very dark and uninteresting image will be produced. In specifying an opacity function, mapping a small range of values to an opacity of 0.010 instead or 0.005 can make a large difference in the resultant image. However, mapping a range of values to 0.75 instead of 0.80 rarely produces a noticeable change in the output image. Thus a good interface for specifying opacity transfer functions should provide more precision in the more transparent domains of the function, even if this is at the expense of precision in the more opaque regions. However, it is important not to give the user an inaccurate or imprecise idea of the selected opacity mapping.

To try to reach these goals, we implemented an interface where the opacity function is displayed on both a linear graph and a logarithmic graph. While the user sketches the opacity function on either graph, the function is updated on both graphs. Using this interface, the user can sketch the function initially using the linear graph, and adjust the areas of low opacity using the logarithmic graph.

4.4 Selecting Views

Our system allows users to set a view by specifying zoom and rotation. The problem of selecting a zoom parameter is not unique to volume visualization. Our system uses the common approach in which the user selects a rectangle of interest in a rendered image, and a new image is rendered which shows the region of interest in as much detail as is possible while still showing the entire region. While the zoom is represented to the user with respect to the picture plane, the zoom parameters the user specifies are used to change a view transformation to render the zoomed image.

An interesting issue relating to zoom is whether zoom should be presented to the user in the two dimensional space of the image plane, or the three dimensional space of the dataset. An image rendered after a zoom operation could be an enlargement and enhancement of a section of the previous image, or an image produced after moving the viewpoint from which the dataset was rendered such that from the new viewpoint only the portion of the dataset seen in the newly rendered image is visible.

While the task of specifying rotational parameters in three dimensional space is not unique to volume visualization there are some important considerations in this type of visualization which do not apply in other problem domains.

Typically, in order to visually represent the rotation of an object in three dimensional space, one renders the object as it might appear from a certain fixed perspective, given the said rotation. There are two additional considerations that must be dealt with in the area of volume visualization.

First, it is not clear what sort of representation of the dataset should be used. If the a visualization system could automatically generate a useful image of a dataset without fail, there would be no need to study user interfaces for volume visualization in the first place. Second, in order for a representation of rotation to be useful, a system must be able to update it at interactive rates. When dealing with large datasets, it is usually not feasible to generate renderings at interactive rates without some sort of subsampling.

4.4.1 Some plausible approaches

Saito [18] presents a technique which involves drawing voxels with higher priority values as line segments to achieve real-time previewing for volume visualization. Ma and Interrante [11] extract feature lines from geometry embedded in the volume data. The feature lines can then be displayed in a highly interactive manner to assist the selection of viewing position for high resolution rendering. In this work, we have explored a few of the following possibilities for representing and specifying rotation.

One possibility is to represent the data using an identifying geometrical shape, preferably asymmetrical to avoid confusion between rotations. One could use the bounding box of the dataset and a set of 3D axes, for example. While this approach has small computational expense, it conveys little information about the dataset at hand.

Another approach is to render a severely subsampled version of the dataset using a traditional volume rendering approach to produce a small image of the dataset which can be updated at interactive rates. With this approach, we has the same question for general volume rendering; that is, which color, opacity and zoom parameters to use when rendering the image used to represent rotation. One might use the current rendering parameters from the last image rendered of the dataset. This approach would make the mapping between the image used to specify rotation and the full size image readily apparent.

With graphics hardware support, isosurface rendering is also a good way to represent the dataset. The question with this approach is what isovalue to select for the surface. Ideally, this value would be determined without user intervention in order to allow the user to focus on the task of rendering useful images instead of intermediate ones.

Finally, if the system had a method for determining which ranges of data values were of interest, the system could render these interesting values as opaque spots in the image. While one could produce this type of image rapidly, the images produced would not be very useful unless the range of interesting values was chosen carefully. We are currently developing ways of automatically generating and evaluating characteristic views of volume datasets based on certain data statistics. This approach makes view selection a possible task for visualizing very large datasets.

5. The Image Graph

The process of adjusting rendering parameters while rendering images of a dataset is a search process. The target of the search is an image which tells the user something interesting about the dataset, and the search space itself is a multidimensional space. We propose a graph based interface which effectively represents the user's search pattern [16]. The topology of the graph is dependent on the type of modifications the user makes to the rendering parameters. That is, we add each newly rendered image to a graph which represents the relationships of all the images which the user has rendered so far, and believe the graph aids in the process of finding a satisfactory image within the design space. To simplify our discussion, the space is limited to four dimensions in which each image is represented by a color map, an opacity map, a zoom, and a rotation. The goal of the image graph is to make searching for a desirable image more

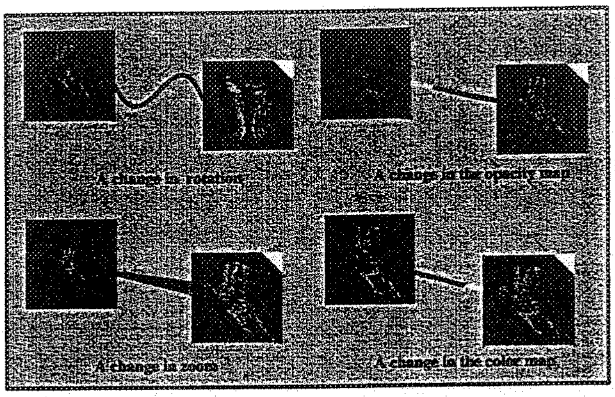

Figure 2: Edge representations for different rendering parameters.

effective by showing how changes in parameters affect the output for a given dataset.

The Design Galleries system [13] treats volume rendering as the process of exploring a multidimensional space. In a preprocessing phase, the system renders images based on parameters in different regions of the search space and can look for the desired image among the group of rendered images. This is an interesting approach because it recognizes that volume rendering should be treated as a process of searching a design space rather than a process of trial and error. Our approach avoids preprocessing in favor of adding newly rendered images to an image graph. The graph keeps track of the relationships between images to make the search of the design space more efficient and effective.

5.1 How the Graph Works

With our graph approach, each newly rendered image is associated with a 4-tuple of rendering parameters (color, opacity, zoom, rotation). A notion of equality is defined for each of these rendering parameters. Two nodes on a graph are considered to be equal if all of their rendering parameters are equal. Two nodes are considered to be similar if all but one of their rendering parameters are equal. After each image is rendered, it is added to the graph. Then the node is attached to similar nodes in the graph. The similar nodes are connected with an edge that represents how they are related. Because similar images can differ in one of four aspects, there are four types of edges that can exist between nodes as shown in Figure 2. An edge represents the change in rendering parameters between the two nodes it connects. When a new node is added to the graph, at most one new edge of each type is drawn to prevent the graph from becoming cluttered.

If a user changes the values of two or more rendering parameters and then renders a new image, a node will be added to the graph which is not similar to any existing node. In the rare case that a new node does not have more than one rendering parameter in common with a preexisting node, the

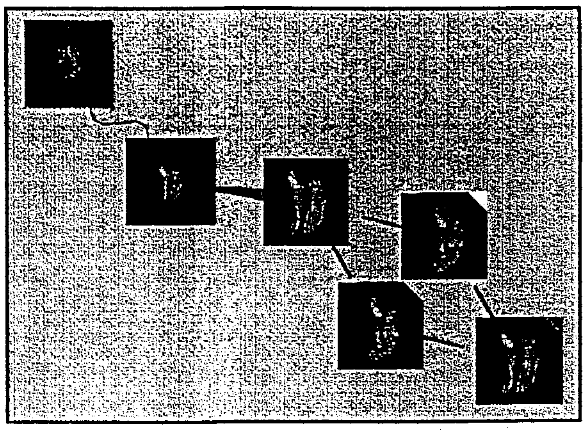

Figure 3: A small graph of some images of a foot dataset. The image in the top left corner is the initial image, and the right most image shows the result of applying rotation, zooming, and different opacity and color maps.

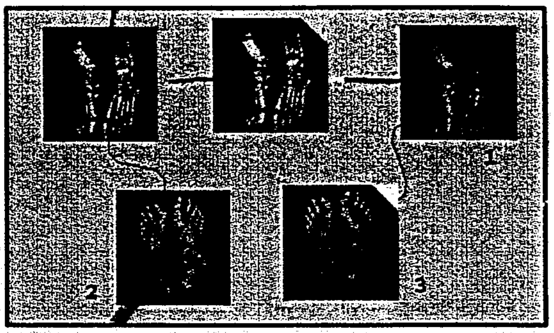

Figure 4: A portion of a graph representing the exploration of the foot dataset. The user combines the color and opacity maps of node 1 in the top right corner with the zoom and rotation of node 2 in the bottom left corner to produce node 3 the image in the bottom right corner.

node is added to the graph without creating any new edges. However, if there is a node in the graph which has exactly two rendering parameters in common with an existing node, the system joins these nodes by creating two nodes which are similar to each of the nodes to be joined. For example, if a user rendered one image, and then changed the color and opacity transfer functions, then rendered a new image, the system would add two intermediate nodes to the graph. As shown in Figure 3, one of these nodes would have the color mapping of the first node on the graph and the opacity mapping of the second node on the graph. The other of these two intermediate nodes would have the opacity mapping of the first node and the color mapping of the second. These two automatically added nodes establish the relationship between the two previously rendered images. To display these intermediate nodes on the graph, the system generates a thumbnail image for each of these nodes. These images are generated using the ray-tracer algorithm which performs well for very small images.

This process of automatically generating graph nodes with thumbnail images of intermediate steps in the rendering process is especially useful when a series of changes in rendering parameters results in an image which is not what the user expected. In this case, the user can look at the intermediate images and determine which of the changes in rendering parameters are responsible for the undesirable aspects of the resulting image.

Note that in Figure 3, the red mark in the corner of the two intermediate images indicates that they are thumbnails. The intermediate images are rendered at low resolution to minimize rendering time. The user can click on a thumbnail to render a full size image with the rendering parameters of that graph node. Avoiding the production of full size images of intermediate nodes saves time, preserving the interactivity of the user's session.

Another feature the graph provides is the ability to combine the attributes of two existing nodes to produce a new node. During the process of searching for the rendering parameters which will produce a useful image, a user may find several images which have some qualities of the desired image, but are not perfect. In this case, the user can drag one node on top of another node on the graph to produce an image which shares selected rendering parameters of the two parent nodes. Figure 4 presents an example. A dialog box lets the user specify which rendering parameters of each parent image will be used for the child image. The new image is then rendered and added to the graph, showing the relationship between the rendering parameters of the child and its parents.

More sophisticated manipulation, such as set operations can also be achieved with the graph approach. For example, a new image may be generated based on a new opacity transfer function which is the union (or difference) of two previously defined opacity transfer functions. This powerful capability enables an even more intuitive and efficient visualization process. Figure 5 presents an example in which the two top images exhibit similar structures which in fact represent two different value ranges. In this particular case, scientists want to see both structures and their interaction as well as relationship in a single visualization which can be produced with the union operation.

The use of a graph to represent the exploration of the dataset provides several improvements over a listing approach. "listing approach," we mean a strategy where each image that is rendered is stored in a list in chronological order, and images can be reviewed by selecting them from this list. Essentially, a graph representation provides a better model of the search pattern, indicates relationships between images, and reserves the ordering information present in a history list. However, a graph representation suffers from the scalability problem as the number of nodes in the graph increases; that is, the graph may become cluttered. We have been experimenting with a node collapsing approach, using a scrolling window, and an automatic node pruning ap-

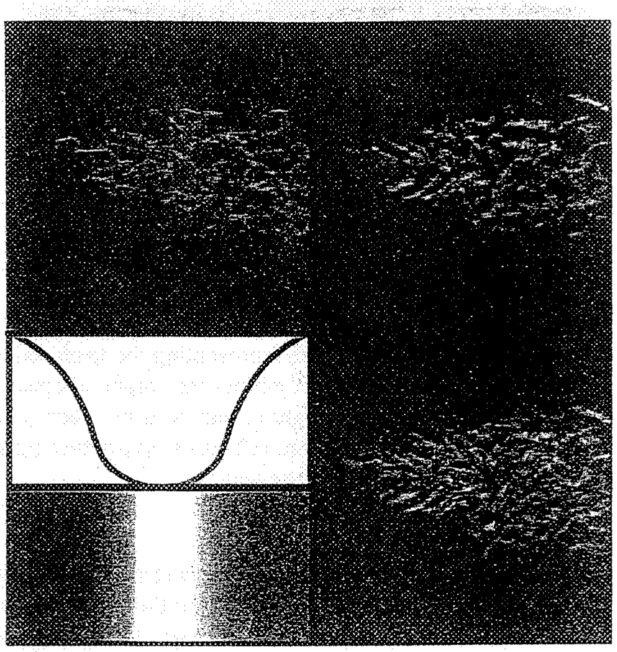

Figure 5: A desirable visualization result (bottom right) was produced using the union of two opacity transfer functions defined by the red and blue curves respectively. The top left image (negative, blue vortices) corresponds to the blue curve. The top right image (positive, red vortices) corresponds to the red curve.

proach to control the scalability problem.

6. Collaborative Visualization

Our testbed system also provides features for collaborative visualization. Users can share, understand, and build upon each others results by sharing annotated graphs. With the graph-based approach, the user can annotate images, both by drawing on the actual images, and by writing comments about the images. These comments are stored in the visualization graph along with the nodes to which they correspond. As well, these graphs can be saved to the local file system, if the web browser gives the Java applet access to it. The annotated graphs can then be exchanged among users of the system.

The exchange of image graphs among users is more useful than the exchange of just image data. If a group of images is used, the user has no clear idea of the relationship between them. If users want to work together to explore a dataset, it is important to minimize the amount of a user's work which is lost when that work is communicated to another user. By expressing the data exploration process in terms of a graph as opposed to a list of images, the system can communicate more information to other users. When a user explores a new dataset, the first step is to locate a reasonable set of rendering parameters which produce an intelligible image. Once this starting point is reached, the user can begin to refine the image. During this process of refinement, a lot of information about the dataset is discovered which can not be captured by images alone. The user may learn, for example, that for a given dataset, changes in the color map do little to change the resulting image compared to the change caused by changes in the opacity map. In a collaborative scenario, it would be useful to communicate this information to other users so they would not have to rediscover it. The image graph accomplishes this goal.

7. Conclusions and Future Directions

This paper gives an overview of a Jave-based volume visualization system which serves as a testbed for our volume visualization research in the areas of user interfaces, collaborative visualization, rendering, etc. We describe the overall system architecture of the testbed and its modular design which allows us to experiment with different approaches to visualization problem solving.

The proxy server we have developed controls the communication between rendering clients and servers in order to optimize the interactivity of the client processes, even if they are running on relatively low powered machines. This has been accomplished by using preprocessing and by distributing rendering requests between several rendering servers.

We also discuss the fundamental problem of parameter specification, the user interface designs that we have derived, and the graph approach to a structured representation of the visualization results. The image graph helps guide the user's data exploration process and assist collaborative work. We feel that our work addresses many problems in volume visualization that are as important as rendering performance but have been neglected by previous research.

So far, we have only plugged in two renderers, both of which are for data on Cartesian grids. We plan to experiment with a renderer for visualizing data on unstructured grids. Most of existing user interfaces can be shared except the ones for loading and preprocessing the raw data.

There are many other visualization/rendering parameters which we can include in our graph approach such as filtering functions, sampling functions, interpolation functions, lighting, data modalities, quantization functions and rendering algorithms. In addition, we are extending the power of the graph approach by adding automatic graph pruning and property propagation capabilities.

Finally, a small user study has been done to evaluate our present user interface design. We plan to perform a more comprehensive user study to evaluate the overall visualization system design, in particular, the effectiveness of the graph approach.