NASA/CR-1998-208468
ICASE Report No. 98-35

A Graph Based Interface for Representing Volume Visualization Results

James M. Patten
University of Virginia, Charlottesville, Virginia

Kwan-Liu Ma
ICASE, Hampton, Virginia

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center under
Contracts NAS1-19480 and NAS1-97046

August 1998

A GRAPH BASED INTERFACE FOR REPRESENTING VOLUME VISUALIZATION RESULTS

JAMES M. PATTEN[1] AND KWAN-LIU MA[2]

Abstract. This paper discusses a graph based user interface for representing the results of the volume visualization process. 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 and animation.

Key words. user interface, volume rendering, scientific visualization, graph drawing, knowledge representation.

Subject classification. Computer Science

1. Introduction. Direct volume rendering has received considerable attention because it is effective for visualizing features in the data that are either very fine or difficult to define analytically. Several volume rendering algorithms have been introduced including the ray casting [7], projection [3], splatting [15] and shear warp [6] methods. Various optimization and acceleration techniques for volume rendering have also been developed including encoding object space coherence [8], encoding image space coherence [17], hardware assisted [13, 2] and parallel [10, 5] methods. To take advantage of these improvements in visualization algorithms, we need an effective user interface for volume visualization. Because volume data exploration often involves a trial and error process of parameter specification, an important part of a user interface is a structured representation of the rendering results. We have implemented an image graph which provides such a representation. This paper gives a summary of the volume visualization process and discusses the fundamental problem of parameter specification. The paper focuses on our graph approach to this parameter specification problem as it applies to volume rendering, and explains the benefits of our approach for data exploration, collaborative visualization, and animation.

1.1. Background. We have implemented a visualization system for volumetric data called DiVision. It serves as a testbed for our visualization research in the areas of user interfaces and collaborative visualization. This system allows visualization across the Internet. Several other systems have addressed the problem of remote visualization. The VizWiz system [12] performs some types of visualization over the Internet, including isosurface rendering and cutting planes, but the system does not support direct volume rendering. The PermWeb system [16] uses a client server approach to volume rendering, but this system does not address the specification of rendering parameters such as color and opacity transfer functions.

DiVision consists of an applet written in the Java language which runs in any web browser which supports Java 1.1, a render "server" process which manages communication with active web clients, and a

[1]Department of Computer Science, University of Virginia, Charlottesville, VA 22906, jmp7d@virginia.edu.

[2]Institute for Computer Applications in Science and Engineering, Mail Stop 403, NASA Langley Research Center, Hampton, VA 23681-2199, kma@icase.edu.

This research was supported by the National Aeronautics and Space Administration under NASA Contract Nos. NAS1-19480 and NAS1-97046 while the authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-2199.

volume rendering process, currently implemented using either of two renderers, one based on the ray-casting algorithm and another based on the shear warp algorithm.

2. The Volume Rendering Process. In a typical volume rendering application, the user specifies some rendering parameters, renders an image, and repeats the process with different parameters based on the results. In our system, the user can select a view and define color and opacity transfer functions before rendering an image of a dataset. A view is described by a zoom parameter and a rotation parameter. These parameters are described briefly here.

Color Map - In volume visualization, the color transfer function specifies a mapping from values in the volumetric dataset to color values used when rendering an image of the dataset. Manipulating the color transfer function changes 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. However, aesthetic considerations may also play a role in colormap selection.

Opacity Map - The opacity transfer function is used by the renderer to determine the opacity (i.e. the importance) of a certain voxel of the dataset according to the values of the data adjacent to or inside of that voxel. Thus the opacity function maps values in the dataset to values between 0 (completely transparent, i.e. not important) and 1 (completely opaque, i.e. very important).

Zoom - The problem of specifying a zoom parameter is not unique to volume visualization. An interface may allow the user to specify a certain magnification factor, or a region of interest. This region of interest may be specified in the 2D coordinates of the rendered image or in the 3D coordinates of the dataset to be rendered.

Rotation - The issue of specifying 3D rotation is important in a variety of problem domains as well. Clearly, an interface must allow manipulation of the dataset's rotation to give the user a variety of perspectives on the data.

There are other parameters which can be included in our system such as filtering functions, sampling frequencies, interpolation functions, and lighting. This paper, however, only addresses transfer functions and view parameters, which are sufficient to illustrate and justify the graph-based approach we have proposed.

3. The Parameter Specification Problem. A variety of volume visualization systems have been developed which include the ability to graphically specify rendering parameters. While we need to develop a good interface for rendering parameter specification, the selection of the parameters is only part of the problem. The remainder of the problem is that even if the user can easily specify a given parameter, the user may not understand exactly how that parameter will affect the resulting image. If the user can specify the rendering parameters he or she wants in an intuitive manner, this does not guarantee good images. Because a user may not be able to predict the rendered image output, volume visualization is an inherently iterative process. One system [4] uses stochastic search techniques in concert with user defined fitness functions to help the user pick good transfer functions. However, under most current systems this iteration is a process of trial and error. The user simply tries combinations of rendering parameters, and stops if he finds a combination which produces a useful image. This rendering process can be time consuming, depending on the algorithm and hardware being used.

The Design Galleries system [11] treats volume rendering as the process of exploring a multidimensional space. The dimensions of the space are the rendering parameters. The image the user is looking for exists in this space, but the user does not know the appropriate combination of rendering parameters to produce that image. In a preprocessing phase, the system renders images based on parameters in different regions of the search space. When the preprocessing is complete, the user can view a 3D representation of the design space

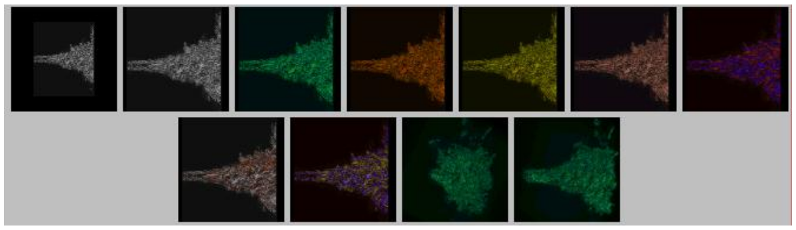

Fig. 1. *A sequence of images produced from a Computational Fluid Dynamics (CFD) dataset. The images are listed in order of creation from the top left to the bottom right.*

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. This efficiency is important regardless of the speed of the volume rendering process. If rendering is a time consuming process, it is clear that we should reduce the number of times it needs to be done to arrive at a given result. But even if rendering happens very quickly, the user's search for the appropriate rendering parameters will still take time.

Independent of the issue of rendering time, a method to represent the data exploration process is useful because it aids in the process of reviewing and recording the interesting structures found in the dataset. A graph which shows the relationships between the images of a dataset provides the user with more information than just a final image or group of images of the dataset. While current systems do not provide a structured representation of the rendering results, some systems have explored structured visual representations of the image production process. The SI system [9] extends the spreadsheet paradigm by incorporating images, data and widgets into spreadsheets. This extension allows the user to manipulate data according to formulae in the same way that numbers are manipulated in a traditional spreadsheet. The Khoros system provides a visual programming environment called Cantata [18] which allows the user to construct directed graphs which represent the flow of data through the system. It is important to stress that our approach to volume visualization differs from standard flowchart based data analysis in that most flow chart systems use a graph to control the analysis of data, while our system uses a graph to help the user understand the results of the process.

4. The Image Graph. Instead of using a history list of rendered images as shown in Figure 1, we add each newly rendered image to a graph which represents the relationships of all the images which the user has rendered so far as shown in Figure 2. We represent the images as a graph to aid in the process of finding a satisfactory image within the design space. 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 with our current system the search space itself is the four dimensional space 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 effective by showing how changes in parameters are affecting the output for a given dataset.

4.1. How the Graph Works. Since each newly rendered image is associated with a 4-tuple of rendering parameters, a notion of equality is defined for each of the four rendering parameters. Two nodes on

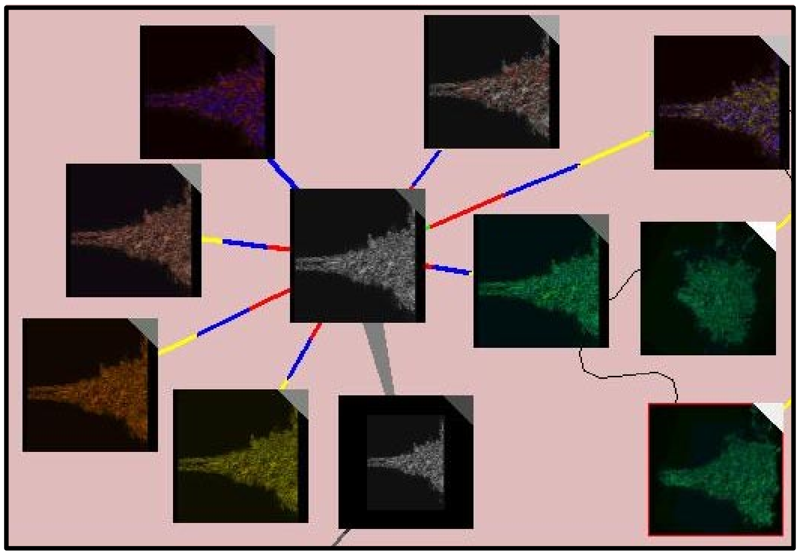

Fig. 2. *A graph representation of the same images produced from the CFD dataset. The graph makes it clear that the user was experimenting with a variety of color maps. After the user had created a few maps, he produced images by changing the rotation of the node with the greenish color mapping. Note that these nodes are close to each other in the graph even though they were not created in sequence.*

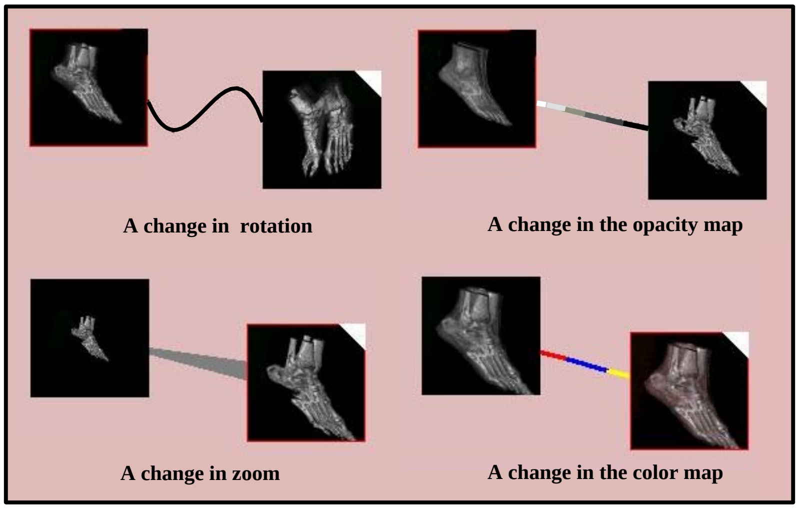

Fig. 3. *Edge representations for different rendering parameters.*

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 and attached to similar nodes. 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 3. 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

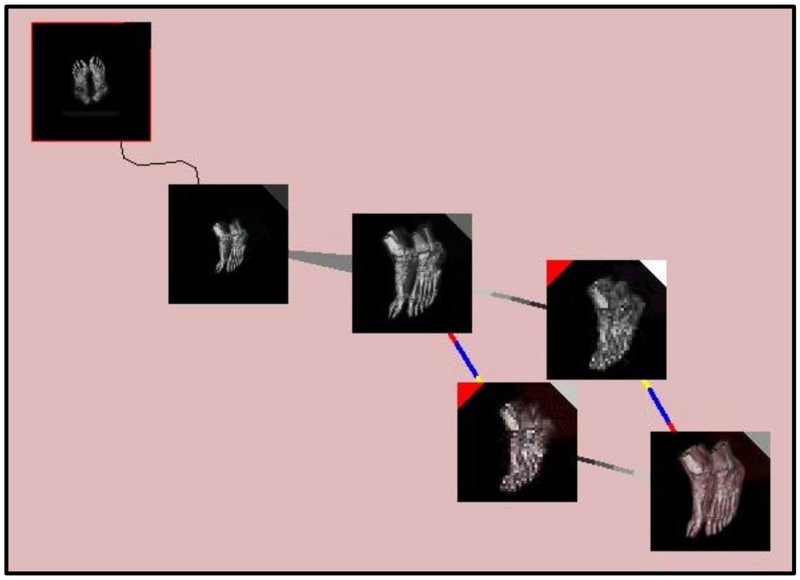

Fig. 4. *A small graph of some images of a foot data. The image in the top left corner is the initial image. The dataset is rotated to produce the second image. The third image is produced by zooming in on the second one. The final image comes from a change in both the color and opacity maps. The system displays this change by estimating the visual effects of both the color and opacity map changes, and rendering separate thumbnails of each. The red mark in the corner of these images indicates that they are thumbnails. The user can click on a thumbnail to render a full size image with the rendering parameters of that graph node. Note that the intermediate images are rendered at low resolution to minimize rendering time.*

has less than two rendering parameters in common with a preexisting node, the 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 renders one image, and then changes the color and opacity transfer functions, then renders a new image, the system would add two intermediate nodes to the graph. As shown in Figure 4, one of these nodes would have the color mapping of the first user 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.

Note that while we could connect nodes with only one parameter in common using a number of intermediate nodes, we do not connect them for two reasons. First, a relationship between two nodes which have only one parameter in common is tenuous compared to the relationships which are emphasized by the graph. Second, by not populating the graph with the intermediate nodes needed for these connections, we keep graph clutter to a minimum.

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 is responsible for the undesirable aspects of the resulting image. Figure 5 shows an example.

A user can view a full size representation of any image by clicking on the image's icon in the graph.

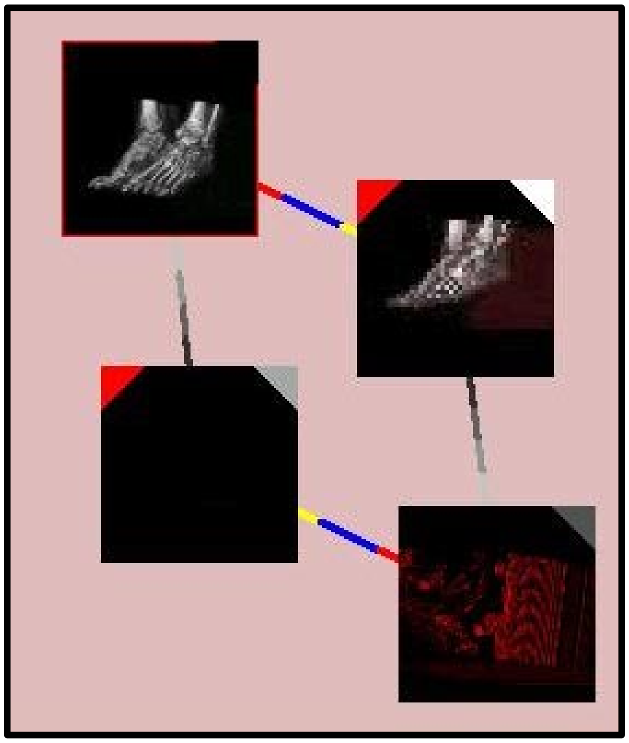

Fig. 5. *Here the cause of an unwanted rendering result is found using the intermediate nodes in the graph. The problem with the lower right image is the opacity mapping. The intermediate image with the same opacity mapping also conveys no useful information. However, the intermediate image with a different colormap than the top image still shows the feet.*

Nodes which do not have a corresponding full size image are marked with a red triangle in the top left corner. The user can render full size versions of these thumbnail images by clicking on them. These full size images are not rendered automatically because the intermediate nodes are mainly intended to show the relationships between two other nodes. In the case that the user can not get the information needed out of the thumbnail image, he can explicitly request that the full size image be rendered. 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 the rendering parameters of the two parent nodes. Figure 6 presents an example. A dialog box as shown in Figure 7 lets the user specify which rendering parameters of each parent image which 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.

4.2. Why Use the Graph Approach?. The use of a graph to represent the exploration of the dataset provides several improvements over a simple listing approach. By a "simple 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.

4.2.1. Search pattern. Considering that the user's task is essentially a search for desirable images within a space defined by the rendering parameters, we need an interface which effectively represents the

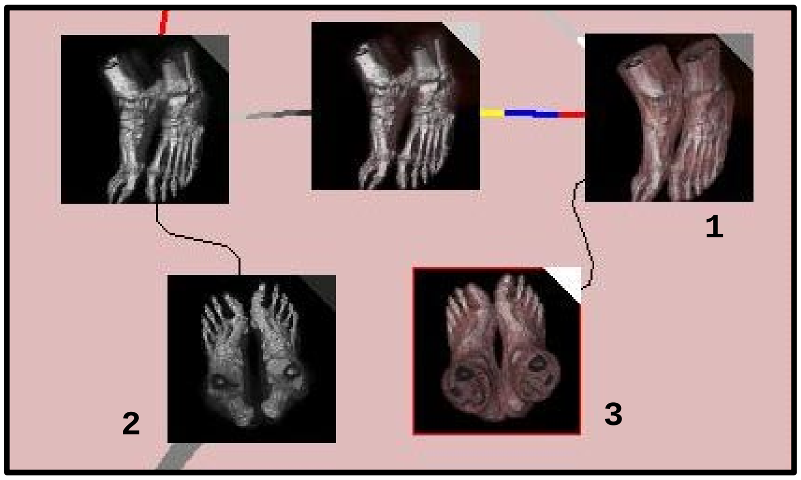

FIG. 6. *A portion of a graph representing the exploration of the foot dataset. The user combines the color and opacity maps of Node 1 in the top right corner with the zoom and rotation of Node 2 in the bottom left corner to produce Node 3 the image in the bottom right corner.*

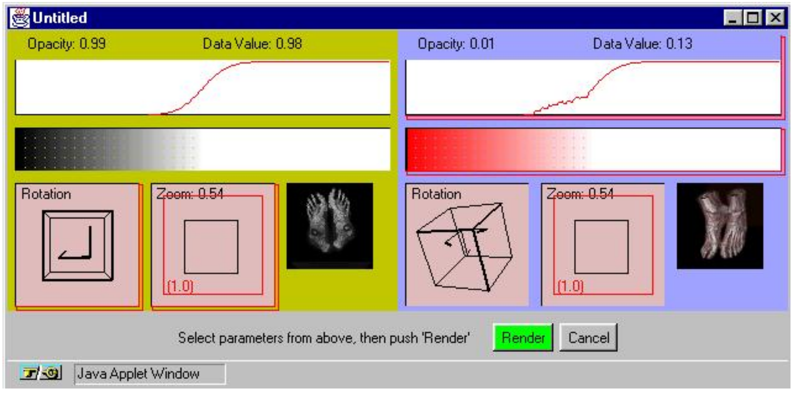

FIG. 7. *The interface used to combine the rendering parameters of existing graph nodes to create new nodes.*

user's search pattern. The graph representation is good at this because the topology of the graph is dependent on the type of modifications the user makes to the rendering parameters. For example, if after rendering an image using the initial default parameters, the user wants to fine tune the rotation of the dataset to best display a certain small structure in the data, the user might render a series of images with differing rotations to search for the best rotation. This process would be represented on the graph by a group of images surrounding the initial image, with each of the surrounding images connected to the original image with a curved line, used to represent a change in rotation. Once the user had found the correct rotation, he might continue his exploration by experimenting with different color and opacity values. Whatever images he rendered after finding the correct rotation would be attached to the image with the desired rotation. The graph would allow the user to quickly locate images of interest by looking at the relationship between images. A serial list of images does not provide this sort of information as shown in Figure 1. A better representation using the graph-based approach has been shown in Figure 2.

The graph allows the user to easily switch back and forth between different points in the image search space. A user could explore different rotations to make a structure visible as described above, and later try using different opacity mappings to make the same structure visible independent of rotation. The user could

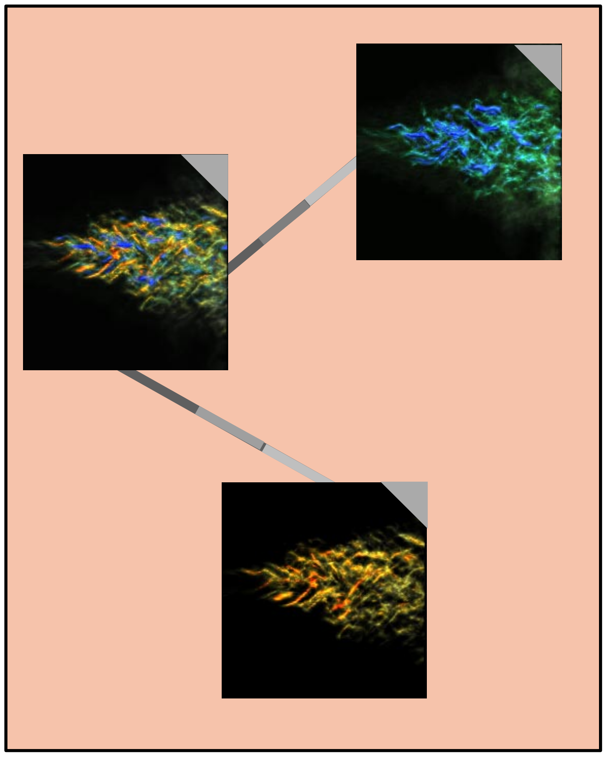

FIG. 8. *The change in rendering parameters is not always apparent from the images themselves. The three nodes in this graph vary only in their opacity maps, yet the images differ greatly in color. The change in opacity maps exposes different portions of the data, each of which are mapped to a different set of colors.*

switch back and forth between these approaches, and the graph would keep the nodes relating to the two approaches separate from each other.

4.2.2. Image relationships. Edges on the graph vary in appearance according to the type of relationship they represent. The reason for this distinction is to depict the changes made during the data exploration to get from one image on the graph to another. It is especially important to know the relationships between the images that have been rendered in case the types of the changes are not readily apparent from the images. This can happen when a color or opacity mapping is not effective for a given dataset. For example, if a lot of contrasting colors are assigned to a range of data values which are also assigned low opacity values, a change in the color map will not necessarily affect a change in the colors of the resultant image. Figure 8 shows an example.

4.2.3. Ordering information. To convey the order in which the images on the graph were generated, each image on the graph has a mark in the corner which represents its relative age. The color of the marks range from black (oldest) to white (newest). Ideally we would use a graph layout algorithm which tried to apply the constraint that newer nodes on the graph were towards the right, while older nodes were located towards the left. We provide "forward" and "back" buttons to select nodes in the graph relative to the current node according to the order of creation. Drawing ordered graphs has been investigated elsewhere [14, 1].

Figure 9 displays a more complete data exploration process using the graph-based approach. The initial image is in the top left corner of the figure. The images grow newer towards the right side of the graph. The node displaying a mix of blue and red vortices shows the dataset with extreme values emphasized. Low

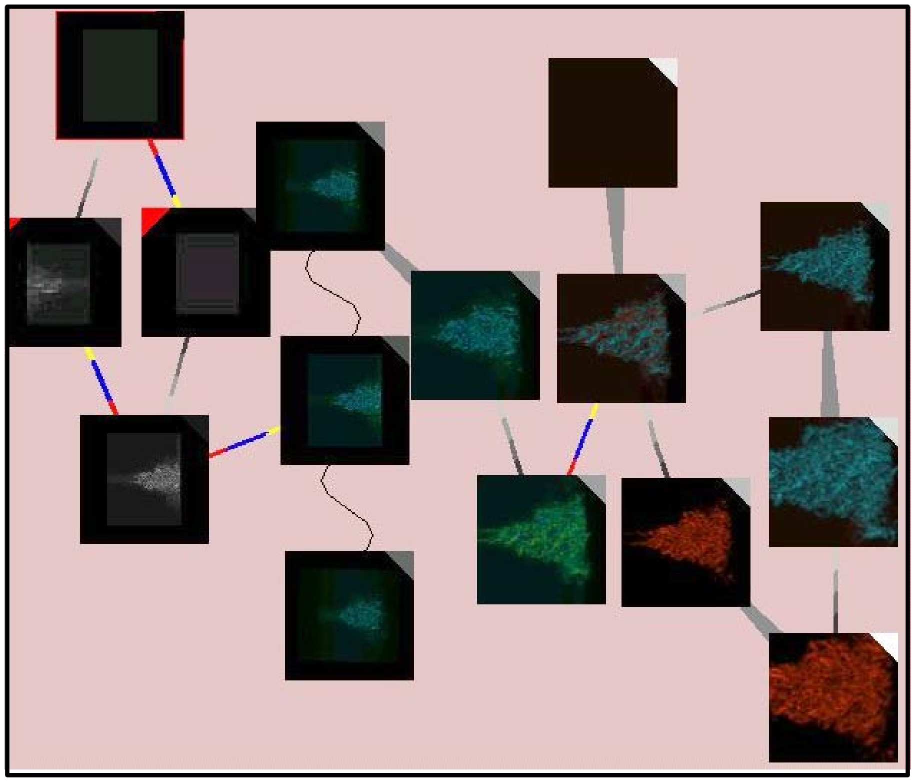

FIG. 9. *The graph constructed during the exploration of the CFD dataset.*

values have a blue-green color, and high values are orange-red. We make two opacity map changes to this image to produce an image of the high values and an image of the low values. Next we zoom in on both of these images. The graph shows that the two zoomed images differ only in terms of their opacity map. Figure 10 displays a desirable visualization result for the CFD dataset. Both negative and positive vortices are captured in a single visualization.

4.2.4. Support for animation and collaboration. In addition, DiVision provides the ability to produce animations from images of the dataset. The user selects the series of images to use for the production of the animation, and the system performs interpolation between them to produce an animation. This interpolation is done in the space of rotation, color, opacity and zoom. Animation works well with the image graph because the user can understand the interpolation process better with the aid of the graph.

The system also provides features for collaborative visualization which allow users to share, understand, and build upon each others' results by sharing annotated graphs. Using DiVision, 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 images to which they correspond. As well, these graphs can be saved to the local file system, if the web browser gives the 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.

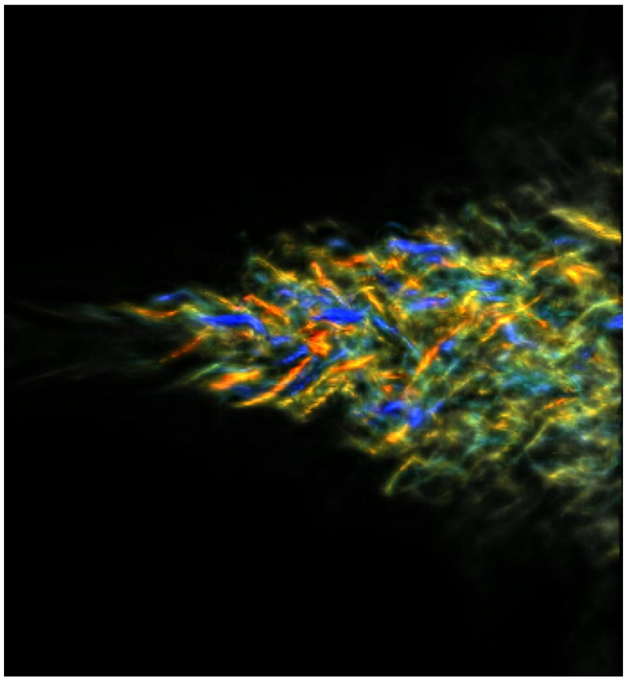

FIG. 10. *The graph-based approach helps derive the desirable visualization which captures both the negative (blue) and positive (red) vortices.*

4.3. Graph Scalability. Currently, the graph occupies a fixed amount of screen real estate. As the number of nodes in the graph increases, the graph may become cluttered. To handle this, we allow the user to collapse nodes on the graph. The user can specify a group of nodes to collapse, and a node to represent these nodes on the graph. The user can expand the representative node later if he wishes to explore the collapsed nodes. With this approach, nodes on the graph which have proven to be less useful can be hidden. For example, if a user was looking for an opacity map which would reveal a certain structure in the dataset, he might produce several images with less useful opacity maps. After he was satisfied with the opacity map, he could collapse the nodes with less useful images into the node with the good opacity map, or he could simply delete the preliminary nodes from the graph if he was certainly not interested in looking at them later on.

Another concern related to scalability is the number of edges present in the graph. If nodes have a lot of connections to each other, it may be difficult to represent the graph clearly in a small amount of screen space. One solution to this problem is to minimize the number of edges connected to the average node on the graph. To this end, we only draw one edge of each type for each node in the graph when that node is created.

5. Conclusions. The graph approach to representing the volume visualization process aids the user by providing a structured representation of the results of the process. This representation aids in collaboration and animation by illustrating the relationships between rendered images. The image graph is especially useful for understanding how certain types of rendering parameter changes affect a given dataset, and preserving information learned about the dataset which can not be expressed in a single image or group of images.

6. Future Work. We would like to use a graph layout algorithm which can provide a notion of the age of graph nodes. Currently, the age of a node is represented by the color of a triangle in a corner of the node's icon. We would like to try to improve on this approach with a graph layout algorithm which establishes a left to right flow of nodes in the graph. With certain graph topologies it would be difficult to enforce this constraint, so a graph algorithm which could consider the left to right chronological constraint along with others would be needed.

Another area for possible future work involves ensuring the scalability of the image graph to large numbers of nodes. While the ability to collapse and remove nodes serves this purpose, it requires user intervention. An approach involving a scrollable graph larger than the display area would be interesting to explore. The system could provide the user with a full view of the graph and a larger view of the area surrounding the current node in the graph to provide scalability without user intervention. Finally we plan to perform a more comprehensive user study to evaluate the effectiveness of the graph approach.

Acknowledgments. The foot dataset was provided by the Visible Human Project and the CFD dataset by Robert Wilson.

REFERENCES

[1] G. D. Battista, P. Eades, R. Tamassia, and I. Tollis, *Annotated Bibliography on Graph Drawing Algorithms*, Computational Geometry: Theory and Applications, 4 (1994), pp. 235–282.

[2] B. Cabral, N. Cam, and J. Foran, *Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware*, in Proceedings of 1994 Symposium on Volume Visualization, October 1994, pp. 91–98.

[3] R. A. Drebin, L. Carpenter, and P. Hanrahan, *Volume Rendering*, in Proceedings of SIGGRAPH '88, August 1988, pp. 65–74.

[4] T. He, L. Hong, A. Kaufman, and H. Pfister, *Generation of Transfer Functions with Stochastic Search Techniques*, in Proceedings of Visualization '96, October 1996, pp. 227–234.

[5] P. Lacroute, *Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization*, in Proceedings of the 1995 Parallel Rendering Symposium, 1995, pp. 15–22.

[6] P. Lacroute and M. Levoy, *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*, in Proceedings of SIGGRAPH '94, July 1994, pp. 451–458.

[7] M. Levoy, *Display Surfaces from Volume Data*, IEEE Computer Graphics and Applications, 25 (1988), pp. 29–37.

[8] ——, *Efficient Ray Tracing of Volume Data*, ACM Transactions on Graphics, 9 (1990), pp. 245–261.

[9] ——, *Spreadsheets for Images*, in Proceedings of SIGGRAPH '94, July 1994, pp. 139–146.

[10] K.-L. Ma, J. S. Painter, C. Hansen, and M. Krogh, *Parallel Volume Rendering Using Binary-Swap Compositing*, IEEE Computer Graphics & Applications, 14 (1994), pp. 59–67.

[11] J. Marks, B. Andalman, P. Beardsley, W. Freeman, S. Gibson, J. Hodings, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber, *Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation*, in Proceedings of SIGGRAPH '97, August 1997, pp. 389–400.

[12] C. K. Michaels and M. J. Bailey, *A Java Applet for Interactive 3D Scientific Visualization on the Web*, in Proceedings of Visualization '97, October 1997.

[13] H. Pfister and A. Kaufman, *Cube4 A Scalable Architecture for Real-Time Volume Rendering*, in Proceedings of 1996 Symposium on Volume Visualization, 1996, pp. 47–54.

[14] I. Rival, *The Diagram*, Reidel Publishing, 1985, pp. 103–133.

[15] L. Westover, *Footprint Evaluation for Volume Rendering*, in Proceedings of SIGGRAPH '90, August 1990, pp. 367–376.

[16] C. Wittenbrink, K. Kim, J. Story, and A. Pang, *PermWeb: Remote Parallel and Distributed Volume Visualization*, in Proceedings of SPIE Visual Data Exploration and Analysis IV, SPIE's Electronic Imaging '97, February 1997, pp. 100–110.

[17] R. Yagel and Z. Shi, *Accelerating Volume Animation by Space-Leaping*, in Proceedings of Visualization '93, 1993, pp. 63–69.

[18] M. Young and D. Argiro, *Cantata: Visual Programming Environment for the Khoros System*, Computer Graphics, 29 (1995), pp. 22–24.