IEEE/ EG Symposium on Volume and Point-Based Graphics (2008)
H.- C. Hege, D. Laidlaw, R. Pajarola, O. Staadt (Editors)

Layers for Effective Volume Rendering

S. Raman, O. Mishchenko, and R. Crawfis

Department of Computer Science, The Ohio State University

Abstract

A multi-layer volume rendering framework is presented. The final image is obtained by compositing a number of renderings, each being represented as a separate layer. This layer-centric framework provides a rich set of 2D operators and interactions, allowing both greater freedom and a more intuitive 2D-based user interaction. We extend the concept of compositing which is traditionally thought of as pertaining to the Porter and Duff compositing operators to a more general and flexible set of functions. In addition to developing new functional compositing operators, the user can control each individual layer's attributes, such as the opacity. They can also easily add or remove a layer from the composition set, change their order in the composition, and export and import the layers in a format readily utilized in a 2D paint package. This broad space of composition functions allows for a wide variety of effects and we present several in the context of volume rendering, including two-level volume rendering, masking, and magnification. We also discuss the integration of a 3D volume rendering engine with our 2.5D layer compositing engine.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Graphics Systems

1. Introduction

Volume rendering has proven to aid in the exploration and understanding of large volumes. While substantial research on transfer functions, volume segmentation, feature detection and rendering integrals has led to impressive results, very little research has gone into user interfaces for volume rendering. This paper builds on the previous research on segmenting and rendering volumes and examines a new system architecture that allows the power and ease of use found in typical 2D paint applications applied to volume rendering.

We segment the workflow process of visualization into two distinct phases. First, traditional volume rendering applied to small regions of interest are rendered. Then, similar to a sort-last scheme for parallel rendering, we apply a compositing phase to the resulting renderings. We expose this compositing phase and provide a rich set of tools and operations to allow unique and effective visualizations that would be difficult in a traditional volume rendering setting. We adopt the concept of layers in a paint package, such as Photoshop [Ado]. Digital artists and technical illustrators have leveraged the power of the layer concept to quickly provide compositions, to remove or highlight parts of an image, to limit the application of a filter or image adjustment and a variety of additional affects. Often a user will use a copy of a layer to safely experiment with different manipulations or processing of an image.

In this paper, we formalize our concept of layers to the field of volume rendering. We will present several basic examples of processes applied to layers for effective visualization, as well as some more advanced examples that would be difficult to achieve directly in a volume renderer. By allowing layers to be duplicated and processed, a rich set of effects can be achieved without the cost of redundant volume rendering. Additionally, we will present several techniques that mimic the results obtained in recent research reports, but achieved using a much simpler 2.5D paradigm. These techniques are not meant to replace previous work on volume rendering, but rather supplement them. We show examples resulting from our own 3D volume renderer, but the application of layers can be applied to other systems and visualization as a whole.

2. Related Work

There are many well known metaphors for visualization system architectures, from dataflows to spreadsheets. The Application Programming Environment, or [Dye90], system and the initial Application Visualization System (AVS)

[UTFK*89], combined a dataflow system with a visual programming language. Using a rich set of pre-built data filters and data mappers, the end-user could rapidly construct visualization applications using a computer assisted visual programming system. Even though AVS was intended as a tool for application developers to build turn-key applications for the developer's specific application area, it was adopted by the end-users and used directly for their applications. This was in part due to the clean separation of the visualization process into filters and mappers, the ease of use of the visual programming metaphor and the flexibility to extend the system with user developed modules. This was truly the first visualization system for the masses.

Other visualization systems have mimicked and extended AVS [AT95] [YAK95] [PJ95]. IBM Visualization Data Explorer enhances the data flow execution by minimizing redundant computations. The SciRun system allows for rapid prototyping of applications as is evident in their recent PowerApps [Pow]. Recently, VisTrails [BCS*05] [CFS*06] provides a systematic tracking of the workflow evolution evident in a complex visualization task. By tracking the history or provenance of a visualization session, it allows the user to experiment with possible paths and manipulate the version tree. Compared to other systems, such an approach frees the user from the need for memorizing the changes that led to a particular visualization.

Other unique metaphors for visualization have been few and far between. A spreadsheet approach was developed by Levoy [Lev94] and later extended by Jankun-Kelly and Ma [JKM00]. Levoy uses images and user controls as the values in the cells of the spreadsheet. A general purpose programming language allows complex manipulations with the data. Jankun-Kelly and Ma project the high dimensional visualization parameter space to two-dimensional spreadsheet, and also allow spreadsheet manipulations, such as rotations and translations, bringing additional flexibility to processing the data stored in the spreadsheet.

The specification of transfer functions using genetic algorithms were explored by Marks et al. [MAB*97]. With their design galleries approach, the tedious task of tweaking input parameters is done automatically by the system. The system varies the input parameter vector allowing the user to select from a collection of generated results. The system changes the parameter values in a way to make the output well-distributed. The Image Graphs [Ma99] methodology translates the process of data exploration into a graph representation, where each node of the graph consists of an image and the parameters used to produce it. This gives the user a clear representation of the dependencies between the parameter values and the visualization results. Semantic layers [RB07] use basic image compositing to produce the final image, though the major contribution is the linguistic specification of the mappings of volumetric attributes to visual styles. Attributes and styles are represented as fuzzy sets, and the mappings are calculated with fuzzy logic.

Our framework exploits the dataflow metaphor. However, we concentrate on the use of layers within the framework. The concept of layers can be seamlessly integrated into all the systems described above as a back-end system.

3. Our Approach

In this section we shall first look at the layers and then at the architecture of the system.

3.1. Layers

The layer concept is central to our framework. By using the term layer, we refer to a 2D container for storing an image, on which a number of operations are defined. The intuition for using layers come from the fact that 2D image manipulation is often easier than working with 3D volumes. A number of tasks for volume exploration are performed by processing the layers instead of applying volume rendering techniques.

User interaction with the system combines traditional tasks, but also allows for a rich set of manipulations. Hence, we use the term layer-centric to describe the framework. Manipulations include applying a number of operators (programmable shaders) to an individual layer, as well as specifying the order layers are combined. From the user point of view, the way layers are manipulated in our system is similar to the layers' manipulation in Adobe Photoshop. Users can create, delete, change the properties (for example, color and opacity), enable or disable the layer, as well as process the layers with the tools our system provides. The system provides a rich set of predefined shaders, including shaders for NPR [ER00] and focus+context techniques [BGKG05] [WZMK05].

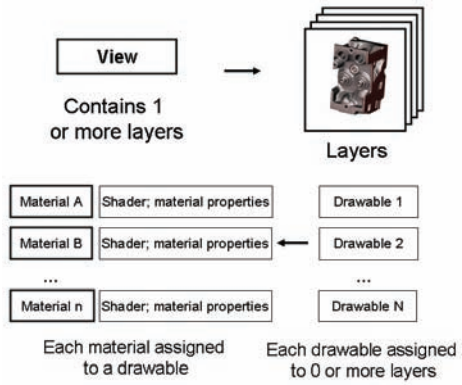

Figure 1: *System Architecture*

3.2. System Architecture

The central component responsible for the display of the final image is a view. A view is associated with a compositing camera and a viewport. A layer may be composited multiple times with different materials. A view also con-tains a list of layers. Each layer contains a list of drawables, which are the entities that should be rendered.

Each drawable has a material associated with it. The material contains attributes needed to render the drawable, for example, color and opacity and a GLSL shader program. The shader provides the implementation of the functionality for the material. All the entities in the hierarchy support one-to-many or many-to-one relationships.

Multiple layers can be rendered with the same drawable or one layer may render multiple drawables. If a layer contains zero drawables, we refer to it as a null layer. Null layers are generated using shaders which have access to the texture resources in the system. Each layer generates a texture, allowing null layers to either create new texture resources or process another layer's texture. The hierarchy of the major components is shown in the Figure 1.

Our system includes transfer function interface with a number of pre-loaded transfer functions. There is also an editor for creating, editing, compiling and linking shader programs. Compiling and linking are done on the fly. The user also controls creation and manipulation of regions of interest (which are three-dimensional boxes), materials and layers.

We employ the layer-based metaphor for the architecture of our visualization system. From the implementation point of view, our framework can be described as a dataflow. However, the difference from other dataflow systems is that we don't explicitly specify the dataflow nor present it to the user. Ease of use of our framework comes from providing a list of tracked assets and a rich set of user interfaces to build and edit each of the main components: drawables, materials, shaders, layers, and views. Additional assets, such as textures, shared numerical controls and cameras are maintained and tracked by the system for use as shader variables or manipulator controls. Figure 2 presents a screenshot of our system, showing the shader editor, material edit, 1D transfer-function editor and the layer editor.

The hierarchy of the entities shown in Figure 1 makes it convenient to make the system event based. Whenever any entity changes, the changes are propagated upwards to the entities that are above in the hierarchy, finally leading to the rendering of the view, if necessary. For example, if a material property is changed, it fires an event to all the drawables that share that material. All of these drawables fire events to their parent layers. These layers in turn, fire events to the view and the view fires an event to invalidate the screen. For detailed explanation of the system architecture we refer the reader to the Tech Report [MRC08].

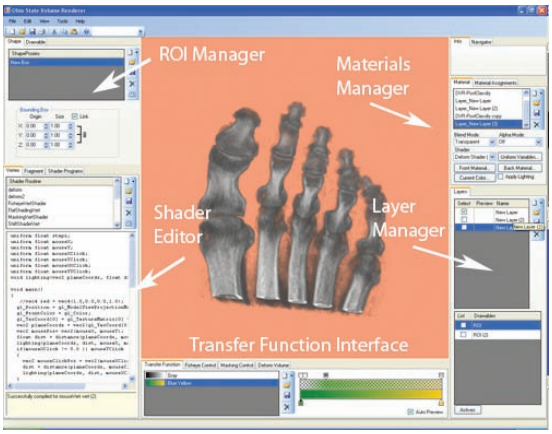

Figure 2: *User Interface Screenshot*

4. Application of Layers

In this section, we describe some of the applications of the layer based volume rendering. We simulate several recent research contributions using simple 2D operators. While not as general as the 3D contributions, they achieve similar results with lower computational complexity and often a simpler and more intuitive user interface. There are a number of properties that layers provide that are useful to list here before presenting the actual applications. Keeping in mind these properties not only provide better understanding, but also may lead to new ideas of how to use the layers for volume rendering.

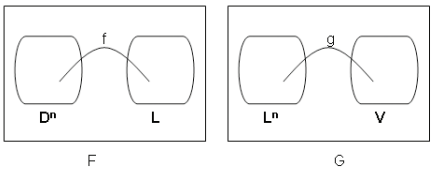

Figure 3: *The relationships between layers, drawables and views*

We specify the relationships between layers, views and drawables in the following way. Let V be the set of views, L the set of layers and D the set of drawables. We define the function space F as the set of all functions or operators that map a set of drawables to a layer, i.e. f: $D^n \rightarrow L$. Likewise, we define the function space G as the set of all functions that map a set of layers to a view, i.e. g: $L^n \rightarrow V$, as shown in Figure 3. Layer manipulation is specified by the compositing shaders (functions). Each layer is an entity to which functions are applied. We can also define the function space L_F as the set of all functions that map L^n to L. The composition of functions f and g, where f and g belong to L_F, is

denoted as $f \circ g = f(g)$; This operator is not commutative, i.e. $f(g) \neq g(f)$.

A simple set of layer manipulations is the set of image processing algorithms, such as blurring and desaturation. Geometric transformations, such as image warping and distortion, can also be achieved.

4.1. Basic Layer Manipulation

Layers can be seen as a control for interactivity. This is achieved by keeping the set of active layers small. While navigating in a huge dataset, only a low resolution layer may be turned on, providing interactive speed of exploration. Interactivity is especially important for effective volume navigation. By performing volume manipulations, such as rotations, the user often gets a better understanding of the volume. This is due to receiving necessary depth cues, as well as the fact that the human visual system is especially good at detecting movement. When the user zooms in on a selected region of interest, a layer with higher resolution is turned on, providing the necessary level of detail.

Layers are a way to provide the context to the user. Context and regions of interest may be rendered to separate layers and thus processed separately. Such a divide and conquer approach helps to focus on a single layer. This is achieved by turning off the context layers while manipulating the layer in focus. Bringing up the context is done by turning on the inactive layers. On the other hand, in some cases operators that provide focus + context functionality may be applied to a single layer.

Layers can be reused. Computationally expensive volume rendering can be avoided in some cases by reusing an already rendered image. When reusing several layers, the overall system performance benefits significantly. The user controls layer reuse. S/he does this by specifying the layer as a parameter to the shaders. For example, same rendered image can be used by two different shaders.

In the next section we provide more concrete examples of applications we have explored with our framework to provide an effective visualization.

4.2. Two Level Volume Rendering

Two level volume rendering [HMBG01] [HBH03] is an effective technique in volume rendering, combining a number of transfer functions and/or rendering types in the final image. Different transfer functions and/or rendering modes are better suited for different types of materials in a dataset. In a medical dataset, for example, bones may be rendered with a MIP, while skin and blood vessels with DVR, or vice versa, depending on the user preferences and the particular application. We achieve results similar to the ones of the two level approach by rendering each of the different styles in a separate layer and then compositing them into a final image.

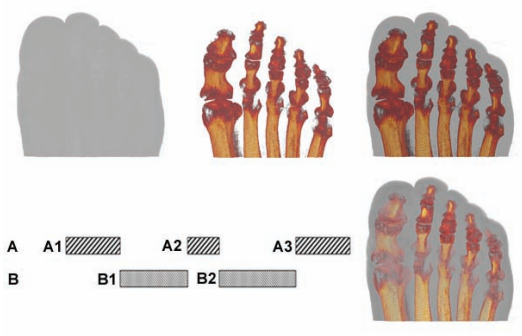

Figure 4: *Top: Two Level Volume Rendering with layers. Bottom Left: The path of a ray through the volume is divided into segments, corresponding to the different materials in the dataset. Bottom Right: original Two Level Volume Rendering of the same dataset.*

The example is shown in Figure 4. Two different transfer functions are used to render the foot dataset. On the top left, DVR is used to create a flat-looking representation of the foot; it is rendered to the first layer and given opacity 0.4 and to the third layer with opacity 0.1. The top center image is a DVR rendering of the bones; it is rendered to the second layer with opacity 0.5. In the top right is the resulting composited image. The second layer is "sandwiched" between the first layer and the third one. This example shows also layer reuse - the same image is rendered once and then used with two different opacities.

According to the notation introduced above, there are two functions, DVR1 and DVR2 that map D to L. The compositing function maps the resulting two layers to V. This is a rather simple example, using only one drawable and two layers. In the bottom right, we present the original two level volume rendering.

The approach described above provides results that slightly differ from the results obtained with the original two level volume rendering. In two level volume rendering, when the ray traverses through the volume, the type of local rendering is selected depending on the type of material encountered in the dataset. The accumulated color and opacity for all ray segments are then composited to produce the final result. Our approach differs in the order in which the segments are composited. In two level volume rendering, the order of compositing corresponds to the order in which the ray encounters the materials. With our approach, all the segments corresponding to the same material in the dataset are composited separately. Then these intermediate results are composited to produce the final image. This is illustrated by the scheme at the bottom left of Figure 4. Given two kinds of materials in the dataset, denoted as A and B, the compositing for two level volume rendering gives I = A1*B1*A2*B2*A3; while with our approach I =

A1*A2*A3*B1*B2. Compositing is not commutative, thus in general the results are different, though for some specific cases they may be the same.

4.3. 2D Magnification

Lenses are useful when text or images needs magnification. In volume rendering, 2D lens have been around for about a decade. It is probably the best and simplest example of a focus + context technique. Zooming in a specified region of interest gives the user the necessary detail for exploration, while the region outside the lens keeps the context necessary for volume navigation. This area has received much attention [BSP*93] [ZHT02]. Bier et al. [BSP97] came up with a user interface that could employ Magic lens filter to modify the presentation of an object. Viega et al. [VCWP96] came up with flat lenses and volumetric lenses. Recently, Wang et al. [WZMK05] developed a framework that provides a variety of GPU-accelerated volumetric lenses.

In our system we use a 2D lens. The zooming effect is achieved by distorting the texture representing the layer. Computationally this allows us to get magnification at almost no cost, since we do not need to re-render the drawables within the layer. Typically, a 2D lens would be applied to the final image. We apply the lens either to a single layer or to a user specified set of layers. Applying the zooming operator to a single layer and not to the final composited image, gives a number of interesting results. First, keeping the context non-zoomed can be useful. An example is shown in Figure 5. The kidney and the ribs are rendered to different layers. The kidneys are focused and magnified, while the ribs are not distorted. We allow for multiple lenses, as shown in Figure 5, where both kidneys are magnified.

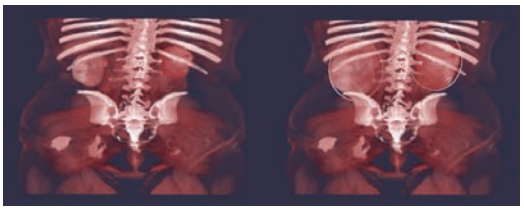

Figure 5: *2D magnification. Lenses applied to both kidneys. Occluding ribs are rendered in a separate layer and are not distorted.*

There are two ways to achieve 3D magnification; one way is to change the field of view and the other is to traverse through the volume, moving the camera closer to the region of interest. Moving the camera changes the occlusion or ray integration. When such changes are not desirable, 2D lens may be a better choice.

4.4. Masking

Masking is a technique used to remove occlusion from a specific area of interest. Given a number of input layers, a masking operator selects and/or blends them according to some specified criteria, such as the opacity. Consider the torso dataset in Figure 9. Keeping the skin visible while trying to explore the skeleton and inner structures, results in a cluttered image. We would like to get an image with the skeleton and other inner structures visible, but only selectively make the skin visible. This can be achieved in the following fashion.

First, skin and skeleton are rendered into two separate layers. The skeleton is rendered to a bottom layer and the skin to the top one. A masking operator makes a part of a top layer transparent, which in turn makes the previous layer visible. An example of this type of masking is shown in Figure 9. The image has two layers composited and the lens makes the enclosed area in the outer layer transparent, thereby displaying the previous layer. Second, any type of 2D texture could

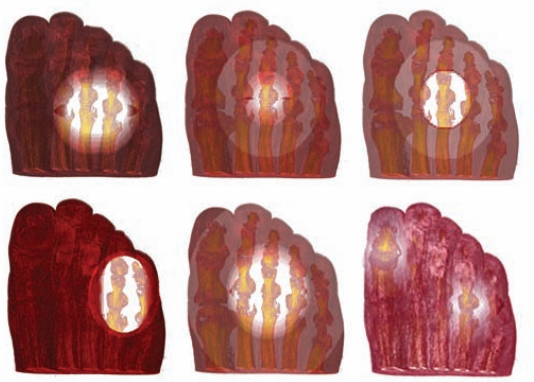

Figure 6: *Different masking styles.*

be used as a mask by having a threshold on the opacity values. This approach is used to get the results in Figure 9. The image with skin is rendered on top of the skeleton image. The masking operator checks the opacity value in the skeleton image, and if it is higher than the threshold, blends the skeleton image with the skin layer. Otherwise, only skin is rendered. The user specifies the blending level; in this case, the final image was generated with blending set to 0.8.

There are other flavours of masking functionality that our framework provides. Correa et al. [CSC06] came up with a technique for interactive manipulation of volumetric models like deformation or cuts. We simulate volumetric cuts using a 2D cut operator. We show the result in Figure 9(e). In Figure 6 we show different styles of masking available to the user. Notice that there may be more than one location to apply a masking operator to as well as it is possible to apply the operator to different layers.

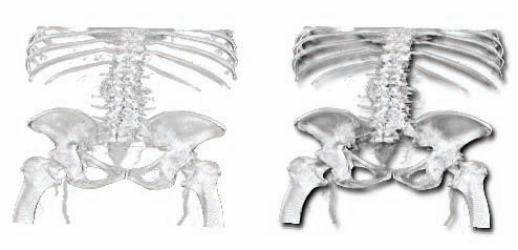

Figure 7: *Right: drop shadow. It is Gaussian blurred.*

4.5. Drop Shadows

Shadows greatly improve the perception of depth in an image. Figure 7 shows a compositing operation that produces a drop shadow. The volume is rendered to a layer and a copy of the layer is made. The shadow operator processes the layer and colors the parts with non-zero opacity and shadow color (e.g. black). A blur operator blurs the shadow, and a shift operator moves the shadow. Finally, the volume rendering and the shadow renderings are composited to get the result shown in the figure on the right. This example illustrates an important concept of layer reuse. We could render the volume twice with a different transfer function to achieve the same result; however, our layer approach is superior in terms of computational cost.

Figure 8: *Left: Gaussian blur. Center: Desaturation. Right: Hatch pattern.*

4.6. Blur and Desaturation

Keeping the amount of visual information presented to the user limited is crucial for effective volume exploration. Too many details can distract the user from the features he/she is interested in. One effective way of dealing with the problem is defocusing or desaturating the context, keeping high-level detail and/or colors only in the region of focus. Another option is to use a masking pattern, such as hatch pattern. Examples of these approaches are shown in Figure 8. This is an example of layer reuse. Instead of utilizing two transfer functions and re-rendering the volume twice, by processing a layer or a set of layers we can achieve the desired effect. (in the above example a single layer keeps both the context and the region of interest). This example along with the previous one illustrate the flexibility of layers: if necessary, multiple layers can be used to achieve the desired result; however, for some applications a relatively simple operator and only one layer are enough.

4.7. Creating Scientific Illustrations

One interesting application that our framework is suit-able for is creating scientific illustrations. An example is shown in Figure 9(g). To make this image, we used two features of our framework: exporting and importing the layers. Grid texture was loaded in a separate layer and the head rendered on top of it with opacity set to 0.8. The resulting image was exported to Photoshop, where the labels were added.

Texture import can be useful for loading any kind of ruler or some other background. This is a common practice in digital art, where the background may be a hand-drawn sketch to use as a template. It will not be used in the final composition, but provides a valuable reference. Likewise, during a volume rendering session, it may be useful to display an image of a reference model, allowing the user to determine the settings and shaders to best match the reference point of view and feature highlights.

5. Implementation

Our system is implemented in C# and OpenGL, with the use of the Tao framework [Tao] that provides OpenGL bindings for C#. Shader programs are written in GLSL. We used 3D texture-based volume rendering with back to front compositing for the attached volume renderer.

The system is event based. Any change in any of the entities in the architecture hierarchy forces the events to fire and propagate through the hierarchy. The components that are registered to receive a particular type of event perform the necessary actions to update their state upon receiving the corresponding event. This keeps the number of state changes minimal. For example, if the user changes a transfer function for one specific region of interest, only events corresponding to this change are fired. This finally leads to re-rendering of the specified region of interest. No other components are updated and unnecessary redraws are not performed.

While performance was not a concern when designing the system, we achieve a number of performance benefits. Eliminating unnecessary redraws as described above can bring significant performance improvements. This is especially noticeable with multiple ROIs and layers where the user is manipulating a transfer function or other material properties. Other performance advantages come directly from the layer properties and the way layers are utilized for particular tasks. As mentioned previously, we can control interactivity by turning off some expensive volume renderings while volume exploration. This is most applicable whenever there is a camera change event that makes all the layers invalid. Also, reuse of layers, as we saw in section 4, helps us avoid redundant volume renderings.

In Table 1 we show performance results for the volume rendering of engine dataset. We were using Dual Core Intel 3GHz 2GB RAM machine with Geforce 7900GTX video card with 512 MB of on-board memory. When only one region of interest is used for rendering, any modification of a transfer function or material enforces the whole dataset to be rendered. The first row in the table shows the result of using a shader with four lights, 4 Light DVR. It has 40 lines of code and is compiled into more than 300 instructions. We can estimate that with ≈ 300 slices and $\approx 400 \times 400$ ROI size, approximately 48 million fragments should be processed, each with more than 300 instructions. The measured results are slightly above 5 frames per second. A much higher frame rate (>60) is achieved with a short and simple shader, DVR.

As indicated above, during volume exploration the user may be interested only in adjusting parameters for certain parts of the dataset. In Figure 9(h) only the highlighted ROI is updated when the user modifies its transfer function. A DVR shader is used for the ROI. 4 Light DVR shader is used for the rest of the dataset (second ROI). The two ROIs are rendered to separate layers. The small size of the first region of interest ($\approx 75 \times 150$ with ≈ 50 slices) leads to processing of approximately 562500 fragments. Then the two layers are blended, resulting in processing of 1024×1024 fragments. The difference in number of fragments processed in the above two cases explains the difference in numbers for rows 1 and 3 in Table 1. Notice that in case of viewpoint changes, still the whole dataset has to be re-rendered.

Engine with 1 ROI, 4 Lights Shader	5.3 fps
Engine with 1 ROI, DVR Shader	63.0 fps
Engine with 2 ROIs, DVR and 4 Lights Shader, only ROI for DVR is updated	62.1 fps

Table 1: *Performance results for Engine dataset. When the user is updating parameters for part of the dataset, there is no need to re-render the other parts (row 3). This provides better performance.*

6. Conclusion

We have presented a layer-based volume rendering system. The ease of use and flexibility of layers allow the user to perform a variety of volume exploration tasks. In our future research work, we plan to automate the process of layer manipulation. With the current implementation of the system, the user is given full control over layer manipulation, including layer creation. This, however, implies that the user should carefully consider how to select layers and operators for a particular task. Often this is a tedious and trial-and-error process. Thus our goal is to make the process of translating the problem into layers automated or semi-automated. For example, the user may be suggested to select a number of volume exploration scenarios when loading datasets into the system.

References

[Ado] ADOBE: Photoshop. *http://www.adobe.com*.

[AT95] ABRAM G., TREINISH L.: An extended data-flow architecture for data analysis and visualization. In *VIS '95: Proceedings of the 6th conference on Visualization '95* (Washington, DC, USA, 1995), IEEE Computer Society, p. 263.

[BCS*05] BAVOIL L., CALLAHAN S. P., SCHEIDEGGER C. E., VO H. T., CROSSNO P. J., SILVA C. T., FREIRE J.: Vistrails: Enabling interactive multiple-view visualizations. *vis 00* (2005), 18.

[BGKG05] BRUCKNER S., GRIMM S., KANITSAR A., GRÖLLER M. E.: Illustrative context-preserving volume rendering, May 2005.

[BSP*93] BIER E. A., STONE M. C., PIER K., BUXTON W., DEROSE T. D.: Toolglass and magic lenses: the see-through interface. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), ACM, pp. 73–80.

[BSP97] BIER E., STONE M., PIER K.: Enhanced illustration using magic lens filters. *IEEE Comput. Graph. Appl. 17*, 6 (1997), 62–70.

[CFS*06] CALLAHAN S. P., FREIRE J., SANTOS E., SCHEIDEGGER C. E., SILVA C. T., VO H. T.: Vistrails: visualization meets data management. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2006), ACM, pp. 745–747.

[CSC06] CORREA C., SILVER D., CHEN M.: Feature aligned volume manipulation for illustration and visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1069–1076.

[Dye90] DYER D. S.: Visualization: A dataflow toolkit for visualization. *IEEE Comput. Graph. Appl. 10*, 4 (1990), 60–69.

[ER00] EBERT D., RHEINGANS P.: Volume illustration: non-photorealistic rendering of volume models. In *VIS '00: Proceedings of the conference on Visualization '00* (Los Alamitos, CA, USA, 2000), IEEE Computer Society Press, pp. 195–202.

[HBH03] HADWIGER M., BERGER C., HAUSER H.: High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 40.

[HMBG01] HAUSER H., MROZ L., BISCHI G. I., GRÖLLER M. E.: Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics 7*, 3 (2001), 242–252.

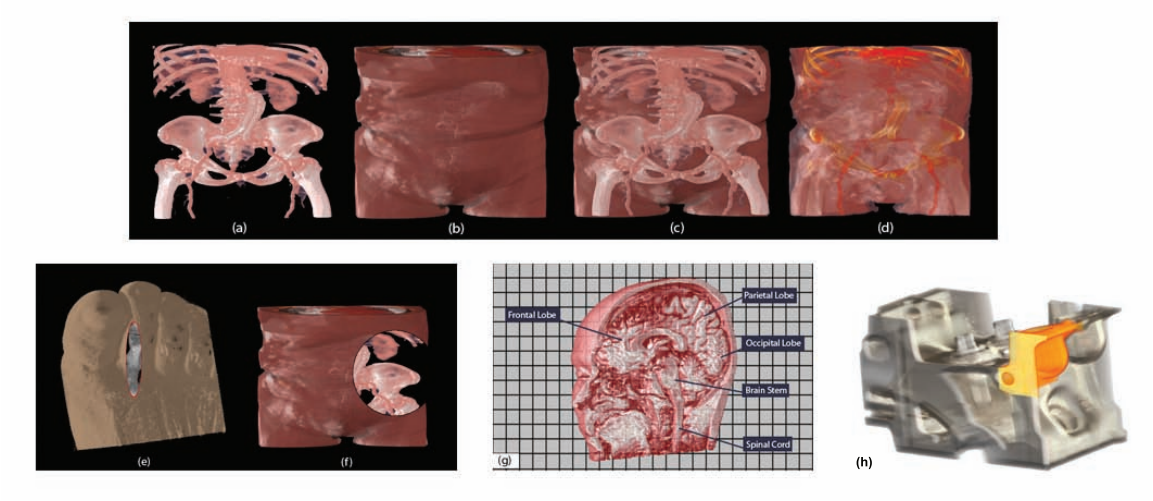

Figure 9: *Top Row: (a) skeleton and (b) skin rendered to separate layers. (c)The result of applying masking operator to (b) and (a). (d) Without masking, image is cluttered. Bottom Row: (e) Volumetric cut. (f) Masking with lens. (g) Scientific Illustration made with our system. (h) When the user is working only with the ROI that is highlighted orange, volume rendering for the rest of the dataset is not necessary. (only layers compositing is performed)*

[JKM00] JANKUN-KELLY T. J., MA K.-L.: A spreadsheet interface for visualization exploration. In *VIS '00: Proceedings of the conference on Visualization '00* (Los Alamitos, CA, USA, 2000), IEEE Computer Society Press, pp. 69–76.

[Lev94] LEVOY M.: Spreadsheets for images. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 139–146.

[Ma99] MA K.-L.: Image graphs—a novel approach to visual data exploration. In *VIS '99: Proceedings of the conference on Visualization '99* (Los Alamitos, CA, USA, 1999), IEEE Computer Society Press, pp. 81–88.

[MAB*97] MARKS J., ANDALMAN B., BEARDSLEY P. A., FREEMAN W., GIBSON S., HODGINS J., KANG T., MIRTICH B., PFISTER H., RUML W., RYALL K., SEIMS J., SHIEBER S.: Design galleries: a general approach to setting parameters for computer graphics and animation. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 389–400.

[MRC08] MISHCHENKO O., RAMAN S., CRAWFIS R.: *Distributed Visualization Framework Architecture*. Tech. rep., The Ohio State University, 2008.

[PJ95] PARKER S. G., JOHNSON C. R.: Scirun: a scientific programming environment for computational steering. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)* (New York, NY, USA, 1995), ACM, p. 52.

[Pow] POWERAPPS:. *www.software.sci.utah.edu/scirun.html*.

[RB07] RAUTEK P., BRUCKNER S.: Semantic layers for illustrative volume rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1336–1343. Member-Eduard Groller.

[Tao] TAO:. *http://www.taoframework.com*.

[UTFK*89] UPSON C., THOMAS FAULHABER J., KAMINS D., LAIDLAW D. H., SCHLEGEL D., VROOM J., GURWITZ R., VAN DAM A.: The application visualization system: A computational environment for scientific visualization. *IEEE Comput. Graph. Appl. 9*, 4 (1989), 30–42.

[VCWP96] VIEGA J., CONWAY M. J., WILLIAMS G., PAUSCH R.: 3d magic lenses. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology* (New York, NY, USA, 1996), ACM, pp. 51–58.

[WZMK05] WANG L., ZHAO Y., MUELLER K., KAUFMAN A.: The magic volume lens: An interactive focus+context technique for volume rendering. *vis 00* (2005), 47.

[YAK95] YOUNG M., ARGIRO D., KUBICA S.: Cantata: visual programming environment for the khoros system. *SIGGRAPH Comput. Graph. 29*, 2 (1995), 22–24.

[ZHT02] ZHOU J., HINZ M., TÖNNIES K. D.: Focal region-guided feature-based volume rendering. *3dpvt 0* (2002), 87.