A Focus and Context Interface for Interactive Volume Rendering

Milan Ikits Charles D. Hansen
Scientific Computing and Imaging Institute, University of Utah *

Abstract

Hardware-accelerated volume rendering techniques are limited by the available fragment processing capacity and on board texture memory of current GPUs. In this paper we investigate the utility of a focus and context interface for hardware-accelerated volume rendering by introducing an interactive lens that can provide finer detail within the region of interest through magnification and high-quality rendering, or can show different aspects of the data via a separate transfer function or rendering mode. In addition, the focus and context decomposition can be used as a user-controlled mechanism for multiresolution rendering in time-critical applications. We show through several examples that such an interface is a useful addition to existing volume rendering implementations and provides a way for improved interaction with volumetric datasets.

1. Introduction

The performance of state-of-the-art hardware-accelerated volume rendering techniques is seriously limited by the available fragment processing capacity and on board texture memory of current GPUs. To circumvent the fill-rate limitation, typical implementations reduce the sampling rate and viewport size during user interaction, both of which can hide details when designing transfer functions or probing the data. Advanced algorithms that use complex fragment shaders or multiple buffers can further increase the burden on the fragment processors and the texture memory caches [13, 10]. In addition, to avoid sampling and quantization artifacts, it is recommended to over-sample the volume and use high-precision buffers for compositing [22]. For these reasons, high-quality high-resolution real-time volume rendering has only been achievable for very small datasets, limiting its use in time-critical applications.

A simple approach for handling the fill-rate limitation that has not been explored completely in the past is to restrict high-quality rendering to a region of interest in the volume and use a lower quality but faster technique for the remaining portions. Such data-independent decomposition is advantageous, because it allows the user to explicitly balance the achievable quality and speed of rendering, and to more efficiently utilize the available screen area. In addition to providing a way to control the speed of rendering, the focus region can also be used to show more details or different aspects of the data. For example, when exploring isosurfaces of turbulent CFD simulation data, the focus can show a small selected area in detail without losing sight of the surrounding context (see Figure 1).

Even though volume rendering is a flexible approach for the simultaneous display of multiple volumetric features, understanding the spatial relationship between the features is challenging. For example, assigning higher opacity to a region of interest and a lower opacity to occluding areas makes the region of interest visible, but also results in a loss of context. In addition, depth cues provided by transparency and lighting work against each other. For example, when rendering multiple transparent isosurfaces, lighting effects are attenuated, making the resulting surfaces appear less distinct. Interactive exploration alleviates the ambiguity caused by transparency and allows users to more carefully and precisely examine the relationship between features and to incrementally build up a mental model of the data.

In this paper we investigate the utility of focus and context decomposition for texture-based interactive volume rendering applications. We extend the state of the art by adding an interactive lens that can provide finer detail within the region of interest through magnification and high-quality rendering and can show different aspects of the data via a separate transfer function or rendering mode. After briefly reviewing related work on interactive lenses and volume rendering acceleration techniques, we describe an efficient implementation of lenses based on depth clipping and the early-z test. We also present a solution to removing clipping artifacts when using pre-integrated classification with a low sampling rate. Next, we demonstrate the interface with several examples, including a multiresolution rendering technique for time-critical applications. We conclude the paper with a summary of our approach and point out possibilities for future research in this area.

* {ikits,hansen}@sci.utah.edu

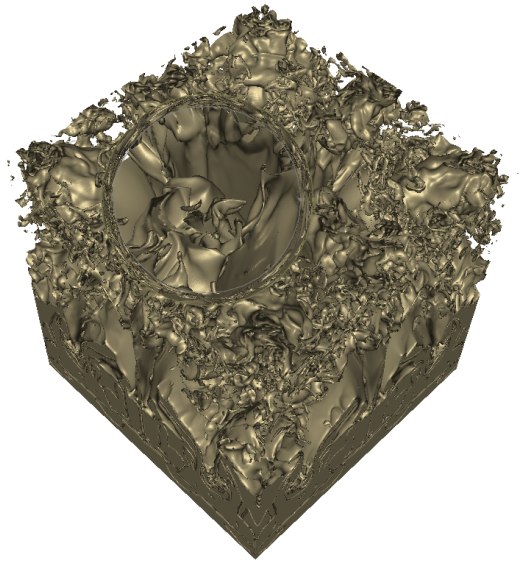


Figure 1. Using a magnifying lens to view details in turbulent fluid mixing simulation data.

2. Related Work

Detail in context viewing is a central user interface component of systems developed specifically for interactive visual inspection of complex abstract and image data. Recent examples include a system used for the comparison of large information trees [21] and the commercially available pliable display tools for detail in context viewing of multiple image attributes [9].

Interactive lenses were first introduced to computer graphics as part of the see-through interface [2]. Image space lenses can perform in-place editing and manipulation tasks and visualize multiple attributes of the image data in a manner that has a natural and intuitive analog. Later work presented a generalized formulation for the detail in context problem and developed several techniques for multi-level magnification and visual display of two-dimensional information [11].

In volume graphics and visualization, interactive lenses were first used for multiresolution reconstruction and time-critical rendering of isosurfaces [5, 31, 27]. Object space lenses proved useful for reducing clutter and occlusion in an immersive flow visualization application [7], and for using a different rendering mode to better show the internal structure of geometric models [28, 23]. VTK provides an example of how to use cutting spheres and planes for creating a combined view of features [24]. It is also possible to specify a region of interest with a high-dimensional transfer function [26] or from the output of interactive segmentation [18].

Recently, a focal region guided volume rendering technique was proposed that combines two different rendering modes to show continuous detail in the region of interest and to reduce occlusion by the surrounding context [33]. A similar idea is used in two-level volume rendering, where different transfer functions, and rendering and compositing modes are integrated to create a more informative display of segmented objects in medical datasets [8]. A 3D X-ray lens was used to combine volume rendering and icon visualization in a stereoscopic system for exploring the correlation between two scalar fields [25]. Lenses have also been proposed to magnify details in volumes without losing the surrounding context, which otherwise happens when zooming in the data [16].

Multiresolution volume rendering techniques build a hierarchical data structure to decompose the original volume into multiple levels of detail [17, 29]. During rendering, several factors can be taken into account to select the appropriate level for each subvolume, including viewing distance, projected screen area, average opacity contribution, and region of interest [19]. One problem with a hierarchical multiresolution approach is that as the user navigates through the data, the switching LODs result in popping and flickering artifacts. SGI's Volumizer tool provides a rendering mode that allows users to interactively explore very large datasets by moving a probe in the data [1]. Rendering and texture updates are accelerated by using clipped mip-mapped textures, such that the lower resolution levels are used during fast user motions. A hybrid multiresolution approach based on the combination of volume and point-based rendering has also been proposed to more efficiently utilize the limited amount of available texture memory [32].

Recently, it has become possible to incorporate the classical acceleration techniques of early ray termination and empty space skipping into hardware-assisted volume renderers [15]. These techniques do not effect the quality of rendering, but the degree of speedup depends on the applied transfer function, such that the worst case scenario incurs a 30% overhead. Similarly, it also has been demonstrated that it is possible to use programmable hardware for efficient volume clipping by arbitrary convex objects [30]. Our implementation of interactive lenses is based on the combination of these two techniques. We also use pre-integrated volume rendering to reduce the aliasing artifacts due to lower sampling rates [6, 20].

Clipping techniques have also been proposed to render only a subset of the data for interactively probing large volumes [1] and to increase rendering speed in immersive volume rendering applications [4, 3]. These approaches, however, do not allow the specification of arbitrary clip regions and provide only limited contextual information about which subset of the data is displayed to the user.

3. Interactive Lenses for Volume Rendering

There are three ways to add interactive lenses to a texture-based volume renderer: 1. tessellate the proxy geometry into focus and context polygons by slicing the clip object that represents the focus, 2. tag the volume to identify different regions, and 3. use depth-based clipping to cull away the unnecessary parts of the proxy geometry during rendering. We chose to implement the last approach, because it allows us to vary the sampling rate for different regions, and supports arbitrary convex clip objects and large bricked volumes without complications that would otherwise significantly influence rendering performance. Since we would like to exploit early-z rejection as much as possible, we render the volume in four passes, as illustrated in Figure 2. First, we render the front facing polygons of the clip object into the depth buffer, set the depth test to `GL_LESS` and draw all slices in front to back order. In the second pass, we render the back facing polygons of the clip object and copy the resulting depth values into a high resolution depth component clip texture. After rendering the front faces into the depth buffer, we change the depth test to `GL_GREATER` and draw slices from the front until the center of the clip object. Depth clipping is performed at the beginning of the fragment shader by reading the corresponding back face depth value from the clip texture and terminating execution of the shader if the fragment depth is greater than the fetched value. By using a clip texture, we ensure that fragments are discarded as early as possible in the fragment pipeline. In the third pass, the same process is repeated for slices from the center to the back of the clip object with the role of the front and back faces reversed, *i.e.* we use the back faces to initialize the depth buffer and the front faces to set the clip texture. In the final pass, we use the back faces of the clip object to initialize the depth buffer, set the depth test to `GL_GREATER`, and render slices from the front of the clip object to the back of the volume.

The implementation is simplified for screen space lenses, because the focus does not need to be divided and a clip texture is not required. However, to incorporate opaque geometry and to accelerate rendering using early ray termination and empty space skipping, two additional depth textures are needed: one to store the depth of opaque objects in the scene, and another to store the depth buffer for the current pass. We use the former to initialize the depth texture at the beginning of the passes, and the latter to initialize the depth buffer for the intermediate steps of the acceleration technique [15]. Also, additional passes are needed for adding a transition region for blending the results of rendering two different resolution versions of the dataset at each slice. We found that this transition region is often not necessary, makes the visualization more confusing, and decreases rendering performance due to its non-convex shape.

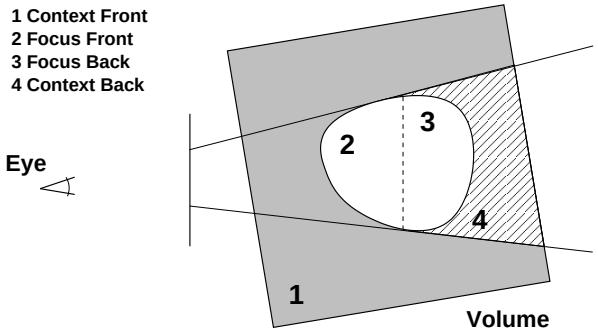


Figure 2. Depth-based clipping combined with the early-z test provides an efficient and flexible way to decompose the volume into focus and context regions.

Note that even though multiple passes are used to render the volume, only the required fragments pass through the fragment shader and compositing stages, because the early-z and clip tests discard the unnecessary fragments as early as possible in the pipeline. By decomposing the focus into front and back regions, we ensure that the majority of the fragments are rejected by the early-z test and do not enter the fragment processing stage at all. Also, clearing the depth buffer multiple times yields only minimal overhead on current GPUs.

One issue that is important to address is how to handle clip boundaries when using pre-integrated volume rendering and a low sampling rate (see Figure 4). An optical model for clipping based on the combination of volume and surface shading was proposed recently, which introduces a boundary layer of finite thickness around the clip object [30]. However, this approach requires interleaved rendering of the volume and the clip surface when using depth-based clipping. An alternative approach is to map an opaque texture onto the clip geometry, hiding rendering artifacts. In most cases, however, the goal of clipping is to show the relationship of features, *e.g.* the distances between isosurfaces. Thus, to correctly handle boundaries, we render the clipping surface before the clipped volume behind it and compute the contribution of the ray segment between the surface and the first slice in the clipped volume in a fragment program. First, we find the intersection Q of the viewing ray for point P on the surface with the first proxy polygon behind P, as shown in Figure 3. Next, we look up data values v_P at P and v_Q at Q and compute the corresponding sampling distance d'. Note that for consistency, we do not compute the actual length of the ray segment between P and O, but its projection to the viewing ray. The orthographic approximation of ray length is a commonly used technique

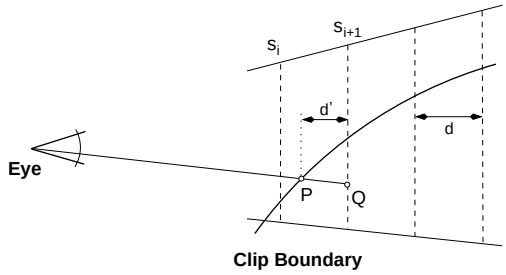
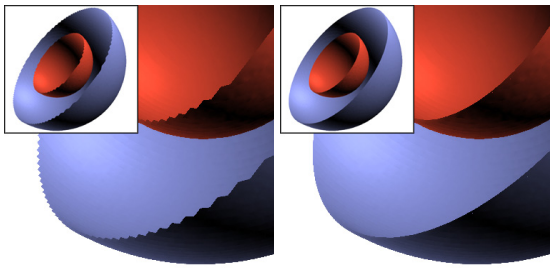


Figure 3. Pre-integrated volume rendering at clip boundaries. The boundary is rendered before the clipped region behind it. The contribution of the ray segment between the boundary and the first slice in the clipped volume is computed in a fragment program.



(a) (b)

Figure 4. Removing clipping artifacts when using pre-integrated volume rendering with a low sampling rate. The images show a clipped test dataset (a) without and (b) with correction.

that avoids the need for a 3D transfer function lookup table, since the same sampling distance can be assumed for the whole volume. The approximation usually does not cause serious rendering artifacts, because the sampling distance does not change abruptly across the volume when using view-aligned slicing. In contrast, at the clip boundaries the sampling distance d' can vary between 0 and d with abrupt changes across the slices. Thus, we use a 3D lookup table to find the corresponding color and opacity for scalar values v_P and v_Q and sampling distance d'. Fortunately, the 3D table is used only when rendering the clip geometry and only a few slices along the length dimension yields a good approximation to the original integral functions.

Note that the above formulation assumes that the clipped volume is behind the clipping surface and that the front slices are projected to the back slices to compute the texture coordinates for the data lookup. When the volume is in front of the clipping surface, the texture coordinates are computed by projecting the back slices to the front slices and the clipping surface is rendered after the volume. This scheme fits into the four-pass algorithm for rendering the focus and context regions and removes most of the artifacts that are the result of using pre-integrated classification at the boundaries.

There are many different uses of interactive lenses in volume rendering applications. In the following subsections we consider three possible scenarios: detail in context viewing of large datasets, assigning different transfer functions to provide more flexibility for interactive exploration, and achieving additional speedup through multiresolution rendering in time-critical applications.

3.1. Magnification

Volumetric magnifying lenses were proposed as a solution to the problem of viewing small details in large datasets together with their context [16]. Our implementation differs from previous work in two ways. First, to achieve a continuous transition between the magnified and normal views, we use a screen-space displacement texture for modifying the texture coordinates before the volume data lookup in the shader. The displacements are scaled by the distance of the fragment from the eyepoint along the view direction to create a uniform screen-space magnification effect for each slice in the focus. Second, to allow exploration of large datasets and to reduce aliasing problems, a high resolution version of the data is used in the focus and a downsampled version in the context. We keep only the required portion of the high-resolution version in texture memory and incrementally update it during user interaction. Instead of allocating separate texture objects for the bricks, we keep them in a single memory pool and use an index volume to locate them for the data lookup (see Figure 5). Adding a level of indirection forms the basis of the virtual texture memory scheme proposed recently [18]. In our case, however, the virtual to physical memory translation is slightly more complicated, because we have to take into account that the bricks overlap at their boundaries. Assuming a single voxel overlap, the address computation is composed of the following steps:

1. Compute index texture coordinate t_i from the virtual texture coordinate t_v, number of voxels n_v and number of bricks n_b along the axis:
$$t_i = \frac{t\, n_v - 0.5}{n_v - n_b} \tag{1}$$
2. Terminate execution if $t_i < 0$ or $t_i > 1$.
3. Look up the brick index i_b from the index texture and compute the physical brick address:
$$t_{pb} = i_b\, \frac{s_b\, 255}{n_p} \tag{2}$$

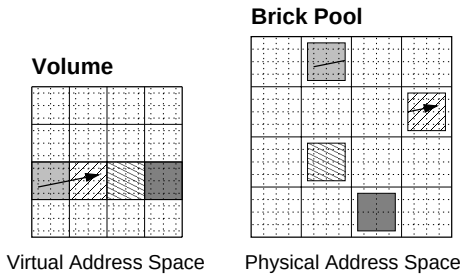


Figure 5. Keeping bricks in a single memory object and using virtual texture coordinates allows arbitrary coordinate transformations in the fragment shader.

where s_b is the size of the brick and n_p is the number of voxels along the corresponding axis in physical memory space.

4. Compute the virtual offset t_{vo} within the brick by taking the fractional part of $t_i\, n_b$. The physical offset is obtained from:

$$t_{po} = \frac{t_{vo}\,(s_b - 1) + 0.5}{n_p} \tag{3}$$

5. Add t_{po} to t_{pb} to obtain the physical address t_p.

Without the virtual to physical address translation, we would not be able to modify texture coordinates on the fly, because some of the modified coordinates would point into a different brick than the currently bound bricks in the shader. The index texture is usually small and the address translation adds a minimal overhead. When not using the lens for magnification, but only for high-resolution rendering, the address translation can further be optimized by modifying the slicing algorithm and computing the index texture coordinates on the CPU, which eliminates steps 1 and 2 of the algorithm. Note that virtual texture coordinates can also be used to implement volume deformation and perturbation for large datasets.

Our experience is that the magnification lens is useful for browsing very large datasets with limited resources, especially when only a small part of the original data can be placed in on-board texture memory and other multiresolution strategies fail.

3.2. Transfer Functions

The goal of using transfer functions in volume visualization is to assign colors and opacities to features of interest in the data. Transfer functions can be specified in terms of direct or derived measures of the data values and locations. Typically, the transfer function is based on a single scalar value. It has been shown that transfer functions based on multiple measures are more successful at capturing and isolating features from each other [14]. One problem of using multidimensional transfer functions is the complexity of their design. It is very difficult to intuitively explore transfer function domains with more than three dimensions. Thus, volume rendering is frequently combined with other visualization techniques and interactive tools to aid the exploration process and to create more insightful illustrations.

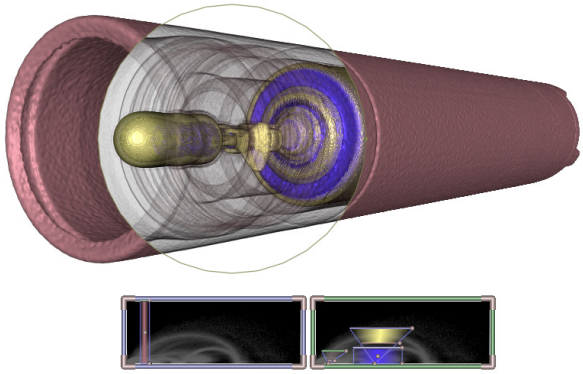


Figure 6. Using two transfer functions to show the internal and external parts of a CT scan of a flashlight.

Interactive lenses extend the existing set of exploration tools that are based on probing, clipping, and deforming the volume. By using different transfer functions in the focus and the context, it is possible to emphasize features that would be difficult to achieve with a single transfer function or with volume clipping alone. Screen space lenses are easier to manipulate than cutting and clipping objects and provide an alternative to switching transfer functions to show the external and internal details of objects, as illustrated in Figures 6 and 8(b).

Another application of lenses is querying different attributes in multidimensional datasets. An opaque detailed view in the focus within a transparent context is especially helpful for reducing clutter when interactively probing multidimensional data. One example is diffusion tensor field visualization, where the combination of a single measure in the context and multiple measures in the focus can be used to reduce occlusion (see Figure 8(a)). Interactive volume exploration is best combined with multiresolution rendering to improve interactivity of the visualization system.

3.3. Time-Critical Multiresolution Rendering

There are a number of factors that can influence the performance of hardware-accelerated volume rendering. Roughly speaking, performance depends on the number of fragments processed in the fragment pipeline, the

	Focus	Context
Data Size	4×256^3	1×128^3
Lookup Table Size	5×256^2	4×128^2
Rendering Model	Local lighting	Unshaded
Sampling Rate	4	1
Viewport Size	100%	50%

Table 1. Parameters for the experimental evaluation of the approach.

amount of work each fragment requires, and the utilization of the texture and framebuffer memory caches. Multiresolution volume rendering techniques use subsampled data blocks to increase cache coherency and draw fewer slices to reduce the sampling rate. We consider additional ways to more efficiently balance the available processing capacity between the focus and the context, such as choosing a simpler and faster rendering algorithm for the context, and reducing the size of the lookup table and the viewport. Drawing the context into a smaller viewport and enlarging and compositing the resulting image is an effective way to reduce the number of fragments generated when zooming into the volume. In this case, the context is rendered in sufficient detail in the smaller viewport. To reduce aliasing artifacts, a Gaussian filter can be used to slightly blur the resulting image. Note that our goal is not to create a seamless integration of the regions, but to achieve a better tradeoff between the achievable quality and speed of rendering to improve interactivity in time-critical applications.

We found two different cases where the proposed decomposition works well. In the first case, a transparent context is used, mostly for orienting the focus in the data. The context does not have to be drawn at full resolution and can be sampled sparsely if the transfer function contains only low frequency components. The focus is attached to a data probe, showing multiple attributes together, possibly using multidimensional classification requiring high-quality rendering and complex transfer functions. The probe may also be combined with polygonal visualization techniques, *e.g.* icons or streamlines, to provide further local information about the data. This kind of decomposition is useful for reducing clutter when exploring multidimensional or multifield datasets. In the second case, the context is opaque and provides an external view of an object, with which the focus is combined for examining the internal details of the object (see Figure 8(c)). In this case, it is possible to exploit acceleration techniques for the context, allowing higher quality detailed rendering of the focus.

To obtain a quantitative assessment of the tradeoff between quality and performance, we conducted an experiment consisting of rendering the same volume from the same view while varying the size of the focus (see Figure 7). The experimental platform was a 2.0GHz Pentium 4 with 2GB memory and an ATI Radeon 9800 Pro graphics card. The fixed parameters for the experiment are collected in Table 1; all textures had 8-bit components. We chose a single viewpoint instead of a path because of the high variability of the performance of slicing a 3D texture from different directions on ATI cards. Thus, the reported results reflect peak performance of the renderer. We turned off the acceleration techniques to obtain results that are the same for any dataset and transfer function combination when rendered under the same conditions. The focus was placed always closest to the user, mimicking typical usage of the interface.

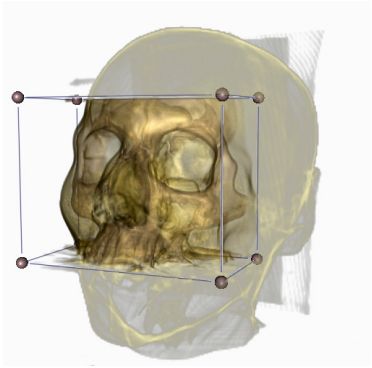


Figure 7. Focus and context rendering of the head CT scan used for the evaluation.

Focus Size		Viewport Size	
Axis	Volume	512^2	1024^2
100%	100%	3.39	1.06
79%	50%	4.22	1.28
63%	25%	7.55	2.31
50%	12.5%	12.99	3.97
40%	6.25%	20.87	6.33

Table 2. Volume rendering performance in frames per second as a function of focus and viewport size.

The results of the experiment for different focus and viewport sizes are shown in Table 2. Focus size is represented as percentage of the whole dataset size along a single axis and in volume. The results in the first row correspond to an implementation without the overhead of the decompo-

sition, which explains the small difference in performance when reducing the volume of the focus by half. The overhead mostly comes from rendering slices multiple times and setting up and using the clip textures and the depth buffer for each pass. By reducing the number of fragments required to render the focus more aggressively, it is possible to achieve 2-6 times faster rendering rates than when using the single high-resolution mode only. These results correspond to our empirical observations of the interactivity of the system.

4. Summary and Future Work

We presented a simple but generic approach for balancing the limited amount of resources available in hardware-accelerated volume rendering applications. The strength of our approach is that the attainable speedup does not depend on the transfer functions, only the rendering parameters used. We found that by incorporating early ray termination and empty space skipping acceleration, it is possible to further improve the interactivity of the system. However, the transfer functions need to be chosen carefully for the region in which the acceleration technique is used. In theory, the system could detect when the user switches to using transfer functions for which the overhead of the acceleration yields slower performance and adjust the rendering mode accordingly.

In the future, we plan to extend the approach to handle the combination of multiple lenses and cutting objects. The focus and context decomposition is particularly well suited for immersive environments and the center-of-workspace interaction paradigm. In these systems, the focus could be placed where the user pays the most attention, which is typically in the region with the best stereoscopic viewing conditions. It is also important to develop screen space interaction techniques for specifying the location and size of object space lenses. Our implementation can further be optimized by using vertex buffer objects instead of vertex arrays and paying more careful attention to streaming the proxy geometry from the CPU to the GPU. We also plan to apply interactive lenses to the exploration of large-scale multi-dimensional and multi-field datasets.

We are interested in conducting a more thorough analysis of the various tradeoffs between the parameters used in hardware-accelerated volume rendering. There are other factors we did not take into account that could be used to further lower the number of fragments generated, *e.g.* the sampling rate could be reduced for regions farther away from the viewpoint. Even though the performance of graphics accelerators will continue to increase rapidly in the future, so will dataset sizes and screen resolutions. With a quantification of how the various pieces affect quality and performance, it will be possible to specify rendering parameters in a semi-automatic fashion. Finally, we believe that in addition to the examples presented in this paper, interactive lenses have various other uses in volume graphics and visualization applications.

Acknowledgments

The authors thank Gordon Kindlmann for useful advice and the Teem toolkit that proved invaluable for data processing [12]. Milan Ikits is supported by NSF grant ACI-9978063 and the DOE Advanced Visualization Technology Center. Magic Lenses™ and Toolglasses™ are trademarks of the Xerox Corporation.

References

[1] P. Bhaniramka and Y. Demange. OpenGL Volumizer: A toolkit for high quality volume rendering of large data sets. In *Proc. IEEE Volume Visualization and Graphics Symp.*, pages 45–53, Boston, MA, Oct. 2002.

[2] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: The see-through interface. In *Proc. ACM SIGGRAPH*, pages 73–80, Anaheim, CA, Aug. 1993.

[3] M. Boyles and S. Fang. 3DIVE: An immersive environment for interactive volume data exploration. In *Proc. International Conference on Computer Aided Design and Computer Graphics*, pages 573–580, Yunnan, China, Aug. 2001.

[4] J. Bresnahan, J. Insley, and M. E. Papka. Interacting with scientific visualizations: User-interface tools within spatially immersive displays. Technical Report ANL/MCS-P789-0100, Argonne National Laboratory, Jan. 2000.

[5] P. Cignoni, C. Montani, and R. Scopigno. MagicSphere: an insight tool for 3D data visualization. *Computer Graphics Forum*, 13(3):317–328, 1994.

[6] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proc. ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 9–16, Los Angeles, CA, Aug. 2001.

[7] A. L. Fuhrmann and M. E. Gröller. Real-time techniques for 3D flow visualization. In *Proc. IEEE Visualization*, pages 305–312, Research Triangle Park, NC, Oct. 1998.

[8] M. Hadwiger, C. Berger, and H. Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proc. IEEE Visualization*, pages 301–308, Seattle, WA, Oct. 2003.

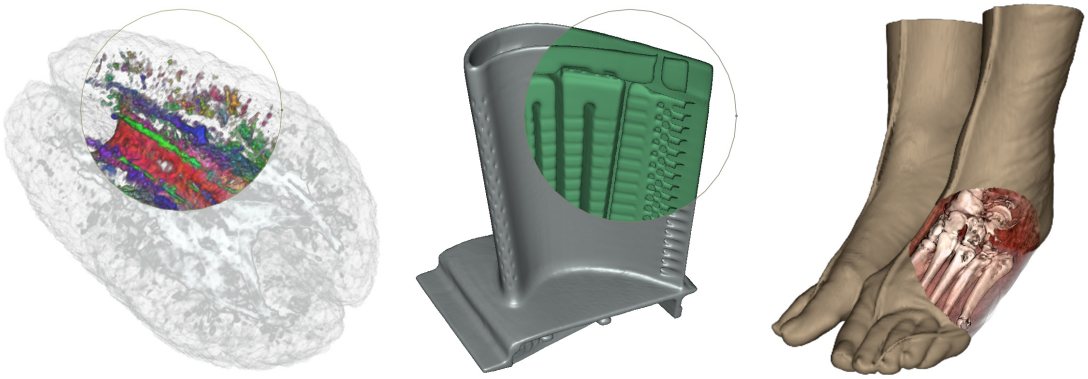
[9] IDELIX Software Inc. Pliable display technology, 2003.

[10] M. Ikits, J. Kniss, A. Lefohn, and C. Hansen. *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, chapter Volume Rendering Techniques. Addison Wesley, 2004. 667–692.

[11] T. A. Keahey. The generalized detail-in-context problem. In *Proc. IEEE Symp. on Information Visualization*, pages 44–51, Research Triangle Park, NC, Oct. 1998.

[12] G. L. Kindlmann. Teem, 2003. `teem.sourceforge.net`.

[13] J. Kniss, S. Premože, M. Ikits, A. Lefohn, C. Hansen, and E. Praun. Gaussian transfer functions for multi-field volume visualization. In *Proc. IEEE Visualization*, pages 497–504, Seattle, WA, Oct. 2003.



(a) (b) (c)

Figure 8. (a) Volume rendering of a DT-MRI dataset. The focus emphasizes the direction of the principal eigenvectors using an XYZ to RGB colormap. (b) View dependent illustration of external and internal isosurfaces in the turbine blade dataset based on gradient directions. (c) A surgical planning scenario using the feet dataset and a spherical focus region.

[14] J. M. Kniss, G. L. Kindlmann, and C. D. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Trans. Visualization and Computer Graphics*, 8(3):270–285, 2002.

[15] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proc. IEEE Visualization*, pages 287–292, Seattle, WA, Oct. 2003.

[16] E. LaMar, B. Hamann, and K. I. Joy. A magnification lens for interactive volume visualization. In *Proc. Pacific Conference on Computer Graphics and Applications*, pages 223–232, Tokyo, Japan, Oct. 2001.

[17] E. C. LaMar, B. Hamann, and K. I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proc. IEEE Visualization*, pages 355–362, San Francisco, CA, Oct. 1999.

[18] A. E. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker. A streaming narrow-band algorithm: Interactive deformation and visualization of level sets. *IEEE Trans. Visualization and Computer Graphics*, 10(4), 2004. 422–433.

[19] X. Li and H.-W. Shen. Time-critical multiresolution volume rendering using 3D texture mapping hardware. In *Proc. IEEE Volume Visualization and Graphics Symp.*, pages 29–36, Boston, MA, Oct. 2002.

[20] M. Meißner, S. Guthe, and W. Straßer. Interactive lighting models and pre-integration for volume rendering on pc graphics accelerators. In *Proc. Graphics Interface*, pages 209–218, Calgary, Alberta, May 2002.

[21] T. Munzner, F. Guimbretiere, S. Tasiran, L. Zhang, and Y. Zhou. TreeJuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility. *ACM Trans. Graphics*, 22(3):453–462, 2003.

[22] S. Roettger, S. Guethe, D. Weiskopf, T. Ertl, and W. Strasser. Smart hardware-accelerated volume rendering. In *Proc. Joint Eurographics/IEEE TVCG Symp. on Visualization*, pages 231–238, Grenoble, France, May 2003.

[23] T. Ropinksy and K. Hinrichs. Real-time rendering of 3D magic lenses having arbitrary convex shapes. In *Proc. International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, Plzen, Czech Republic, Feb. 2004.

[24] W. Schroeder, K. Martin, and W. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Prentice Hall, Upper Saddle River, New Jersey, 1996.

[25] C. D. Shaw, J. A. Hall, D. S. Ebert, and D. A. Roberts. Interactive lens visualization techniques. In *Proc. IEEE Visualization*, pages 155–159, San Francisco, CA, Oct. 1999.

[26] F.-Y. Tzeng, E. B. Lum, and K.-L. Ma. A novel interface for higher-dimensional classification of volume data. In *Proc. IEEE Visualization*, pages 505–512, Seattle, WA, Oct. 2003.

[27] T. Udeshi, R. Hudson, and M. E. Papka. Seamless multiresolution isosurfaces using wavelets. Technical Report ANL/MCS-P801-0300, Argonne National Laboratory, Mar. 2000.

[28] J. Viega, M. J. Conway, G. Williams, and R. Pausch. 3D magic lenses. In *Proc. ACM User Interface Software and Technology*, pages 51–58, Seattle, WA, Nov. 1996.

[29] M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, and T. Ertl. Level-of-detail volume rendering via 3D textures. In *Proc. IEEE Volume Visualization and Graphics Symp.*, pages 7–13, Salt Lake City, UT, Oct. 2000.

[30] D. Weiskopf, K. Engel, and T. Ertl. Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Trans. Visualization and Computer Graphics*, 9(3):298–312, 2003.

[31] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer*, 15(2):100–111, 1999.

[32] B. Wilson, K.-L. Ma, and P. S. McCormick. A hardware-assisted hybrid rendering technique for interactive volume visualization. In *Proc. IEEE Volume Visualization and Graphics Symp.*, pages 123–130, Boston, MA, Oct. 2002.

[33] J. Zhou, M. Hinz, and K. D. Tönnies. Focal region-guided feature-based volume rendering. In *Proc. International Symposium on 3D Data Processing, Visualization and Transmission*, pages 87–90, Padova, Italy, June 2002.