SCI INSTITUTE
TECHNICAL REPORT

Image-Based Volume Rendering with Opacity Light Fields

Miriah Meyer, Hanspeter Pfister, Charles Hansen, Chris Johnson

UUSCI-2005-002

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

March 24, 2005

Abstract:

While low cost PC graphics hardware has proven valuable for volume rendering, large datasets continue to overwhelm the capabilities of the graphics cards, reducing the interactivity of volume rendering utilizing such hardware. We present a novel, image-based approach to volume rendering that can render arbitrarily large datasets interactively on current graphics hardware. Our method is independent of the volume rendering system and the dataset representation, and allows for exploration of the interior structure of the volume. The process consists of three main steps, each of which can be run independently. In the first step, a set of *ray slices* of the data are produced from multiple viewpoints using a volume render, and a geometric proxy surface bounding the volume is defined. Next, the ray slices and geometric proxy are processed to compute a set of *key views*. Finally, the key views and proxy surface are rendered interactively as *opacity light fields* on current graphics hardware. The user can change the proxy surface to reveal the interior structure of the volume data. Our method has been tested on a variety of volume datasets and these results are presented in this paper.

THE UNIVERSITY OF UTAH

Image-Based Volume Rendering with Opacity Light Fields

Miriah Meyer[1] Hanspeter Pfister[2] Charles Hansen[1] Chris Johnson[1]

[1]Scientific Computing and Imaging Institute
University of Utah

[2]Mitsubishi Electric Research Labs
Cambridge, MA

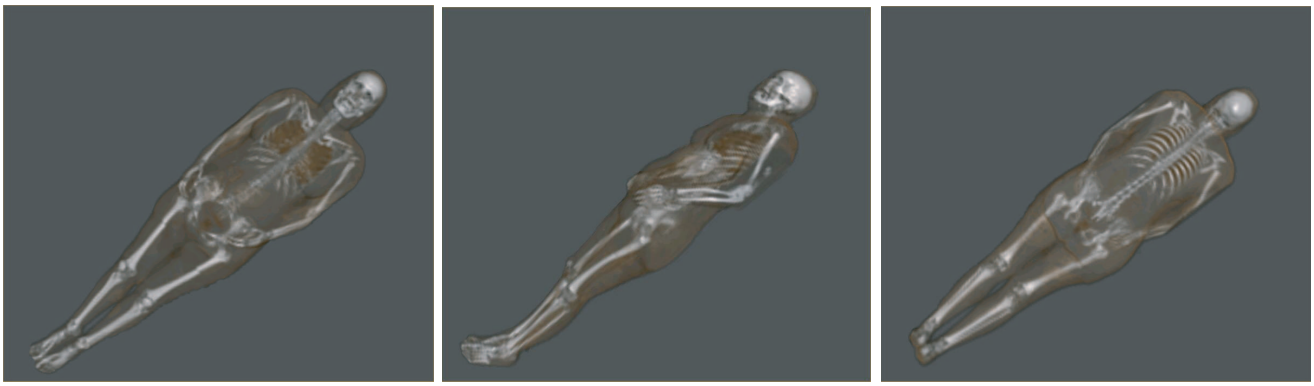

Figure 1: *IBVR of the visible woman.*

Abstract

While low cost PC graphics hardware has proven valuable for volume rendering, large datasets continue to overwhelm the capabilities of the graphics cards, reducing the interactivity of volume rendering utilizing such hardware. We present a novel, image-based approach to volume rendering that can render arbitrarily large datasets interactively on current graphics hardware. Our method is independent of the volume rendering system and the dataset representation, and allows for exploration of the interior structure of the volume. The process consists of three main steps, each of which can be run independently. In the first step, a set of *ray slices* of the data are produced from multiple viewpoints using a volume renderer, and a geometric proxy surface bounding the volume is defined. Next, the ray slices and geometric proxy are processed to compute a set of *key views*. Finally, the key views and proxy surface are rendered interactively as *opacity light fields* on current graphics hardware. The user can change the proxy surface to reveal the interior structure of the volume data. Our method has been tested on a variety of volume datasets and these results are presented in this paper.

1 Introduction

With today's rapid increase in computing power also comes a drastic expansion in dataset size. This phenomenon is especially prevalent in scientific, medical, and engineering visualization, where the availability of increasingly powerful computers and high resolution scanners have resulted in highly accurate and detailed data. For example, CT scanners now capture thousands of images with 512×512 resolution, supercomputers are producing terabytes of simulation data, and seismic scans for the oil and gas industry contain gigabytes or terabytes of data [32].

Visualization of immense volume datasets remains a challenge despite technological advancements, and interactive visualizations of these datasets on a commodity desktop machine is currently not possible. The main challenges are the massive computational power necessary for interactive visualization, moving huge amounts of data from memory to the GPU, and providing insightful, high quality images. Interactive volume rendering of large datasets is presently achieved using large multiprocessor machines or PC clusters [26, 12, 23], but these systems are expensive and not widely available.

On the desktop, graphics hardware is widely used for interactive volume rendering [10, 29, 27]. But special-purpose hardware is not commonly available, and on-board memory of commodity graphics cards is limited (currently 256 MB). One solution to rendering large datasets on memory restricted graphics hardware is to downsample the volume such that it fits in graphics memory, while another is to aggressively compress the data [14]. However, both approaches come with a significant loss of fidelity. There is also work on multiresolution and out-of-core visualization on the desktop [33, 6, 7]. None of these methods allow the user to interactively render arbitrarily large datasets.

We present a new, image-based approach to volume rendering that can render arbitrarily large datasets on current graphics hardware. Our method is independent of the volume rendering method and the dataset representation. For example, image-based volume rendering (IBVR) works for irregular grids as well as for rectilinear volumes. In contrast to previous IBVR methods [22, 4], our technique places no restriction on the viewpoints. It allows the user to place arbitrary clipping surfaces through the volume, revealing the interior structure of the volume data [34] without the need to volumetrically re-render the data. We split the volume visualization pipeline by separating the task of volume rendering (which can run as a preprocessing step) from volume viewing (which runs interactively on the desktop).

The overall process and main data structures of our system are shown in Figure 2.

During volume rendering (step one), the volume data is

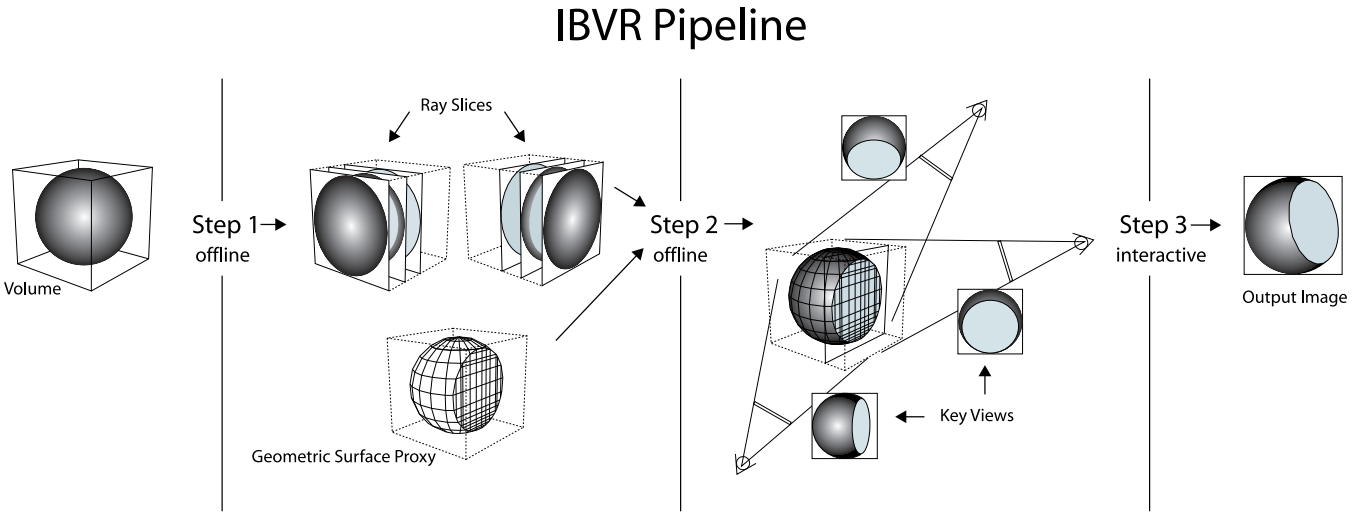

Figure 2: *Processing pipeline of the IBVR system. Step 1 renders a volume and outputs ray slices from multiple viewpoints. Step 2 takes the ray slices and a proxy surface to generate key views from the ray slice viewpoints. Step 3 interactively renders the key views using an opacity light field.*

rendered from multiple viewpoints using an arbitrary volume rendering system. The output of the volume rendering system is a set of *ray slices* from each viewpoint. A ray slice stores information about the volume for a specific viewing location. As we will show in Section 3, ray slices can be easily computed using any standard volume rendering application on any type or size of data. Independently, a geometric *proxy* surface is computed for the volume. The initial proxy can be a bounding sphere or an isosurface.

In step two, new *key views* are generated using the ray slices and geometric proxy for each ray slice viewpoint, as described in Section 4. Each key view shows a composited view of the volume intersected by the proxy surface from a specific viewing location. The user can edit the proxy surface, e.g., by adding new clipping planes that modify the original proxy surface, to reveal the internal structure of the data. Finally, during volume viewing (step three), key views are rendered from arbitrary, novel viewpoints using opacity light field rendering, as discussed in Section 5.

We present our results in Section 6, followed by a discussion of our findings in Section 7. Future work is explored in Section 8 along with our final conclusions.

2 Previous Work

Many techniques have been developed to interactively visualize large datasets. Utilizing the parallelism of ray tracing on multiprocessor machines, Parker et al. [26] allow rendering of volumes up to several gigabytes in size at multiple frames per second. Similarly, Muraki et al. [23] allow rendering of volumes up to 1024^3 interactively on PC Clusters. However, these techniques require high-end parallel systems to render the data interactively.

On PCs, most interactive volume rendering techniques utilize the capabilities of modern graphics cards. The most popular methods use 2D [29] and 3D texture mapping [35, 3]. Many extensions have been proposed, such as pre-integration for more accurate reconstructions of the volume rendering integral [10], advanced shading [20, 19], and cell-projection rendering for irregular data [30]. Despite these improvements, visualization of large datasets is still fundamentally limited by the amount of texture memory available on the graphics cards.

Some approaches have focused on compressing large datasets to allow the data to fit into texture memory before rendering the volume. Guthe et al. [14] propose a very effective method using wavelet compression. However, their method results in an inherent loss of detail prior to rendering. Out-of-core visualization techniques [8] use some form of virtual memory to access data that is larger than the size of memory. Other out-of-core methods [6, 7] utilize hierarchical data indexing schemes and representations to enable faster data retrieval. Nevertheless, none of these methods achieve real-time frame rates on PCs for arbitrarily large data, and most assume a rectilinear dataset.

Our method separates the task of volume rendering from the task of interactive viewing, which eliminates the loss of data before rendering, as well as any dependency on the size or representation of the data. Volume rendering can be run on massively parallel machines or on a cluster of PCs. Our volume viewing application uses image-based rendering with 2D texture mapping and can be run on any modern PC graphics card in real-time, as described in Section 5.

Image-based representations have the advantage of capturing and representing objects regardless of the complexity of their geometry. Pure image-based methods [16, 13] do not use any geometry, but require a large number of images. Gortler et al. [13] include a visual hull of the object for improved ray interpolation. View-dependent texture mapping [28, 9] combines simple geometry and sparse texture data to accurately interpolate between the images. Surface light fields [21, 36, 24, 5] are a more efficient parameterization of view-dependent texture maps. Matusik et al. [18] combine view-dependent opacity for every surface point with surface light fields into *opacity light fields* to render objects with arbitrarily complex shape and materials from arbitrary viewpoints. As we will describe in Section 5, our system uses interactive opacity light field rendering [31] with unstructured lumigraph interpolation [2] on graphics hardware for volume viewing.

Two recent methods have used image-based rendering for volumes. Chen et al. [4] store preprocessed volume images

for on-the-fly reconstruction of surfaces, but do not allow for opacity in the final rendering. Another method proposed by Mueller et al. [22] blends preprocessed images including opacity values to render external views of a volume within a limited viewing cone. Neither of these recent techniques store information about the volume integral for each viewpoint. Our method stores information about the volume integral, allowing the user to view the internal structure of the volume without having to recompute the input images. Furthermore, our method places no restrictions on viewpoints external to the proxy surface.

We will now describe each stage of our IBVR approach that is shown in Figure 2.

3 Step 1: Volume Rendering

The volume rendering step can be accomplished with any arbitrary volume rendering system, on any type of hardware. For example, an oil and gas industry seismic dataset of several terabytes in size could be rendered on a supercomputer or large parallel system, or an irregular dataset from Computational Fluid Dynamics (CFD) could be rendered on a cluster of PCs. The volume data can be stored in an arbitrary grid representation, although we focus on rectilinear volumes in this paper.

The volume renderer produces ray slices for a set of original viewpoints. Ray slices are created by progressively moving an image-aligned clipping plane through the volume back-to-front, clipping away the volume between the clipping plane and the viewpoint (see Figure 3).

Each pixel within a ray slice image represents a partial ray projected from the slice location through the volume along the corresponding viewing ray. Hence, the color and opacity values at each pixel correspond to the accumulated colors and opacities generated by the volume integral along a viewing ray starting at the slice through the remainder of the volumetric data. For a given viewpoint, multiple ray slices can be stored for key view generation, the next step in our viewing pipeline.

Ray slices are easily produced by any volume rendering application that computes RGBA images with back-to-front compositing for arbitrary image-aligned clipping planes. This requirement is easily fulfilled by any traditional volume rendering method, including irregular volume rendering and splatting.

Ray slices can be compressed via standard image compression schemes. We use the Portable Network Graphics (PNG) [1] format for lossless compression. JPEG 2000 or other methods could be used for lossy compression with higher compression ratios.

The number of ray slices per viewpoint is one of the parameters that can be adjusted by the user. In general, the number of slices is dataset dependent. For example, anisotropic data can have more slices along one axis. Furthermore, the distance between slices can be adapted to skip empty space or acquire more detail at certain depths within the volume. Each viewpoint can have as few as one ray slice, or as many as memory restrictions will allow. We will show results in Section 6 and discuss these trade-offs in Section 7.

4 Step 2: Key View Generation

The key view generation step takes the ray slices and associated viewpoint parameters as input, along with a geometric proxy surface, to calculate key views for each ray

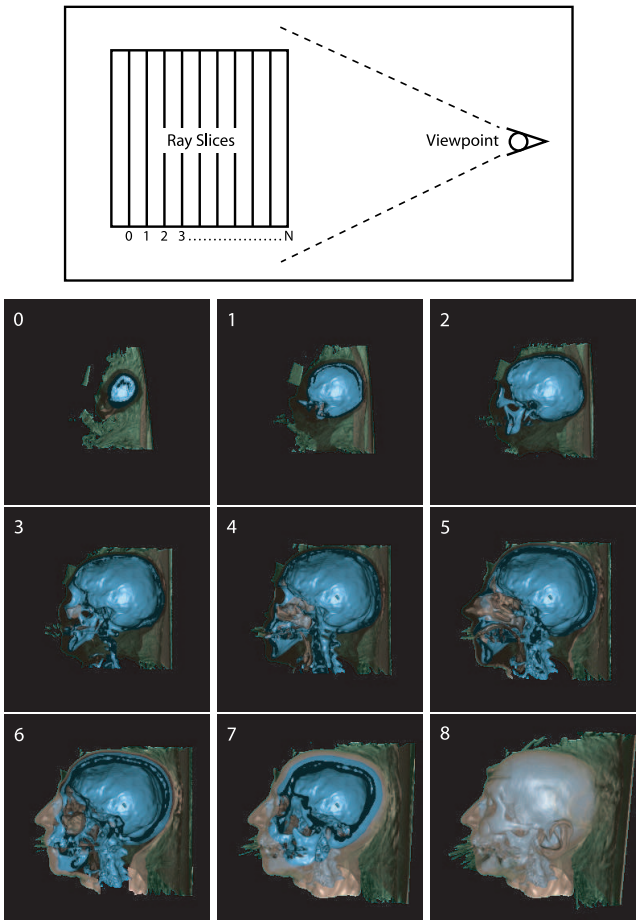

Figure 3: *Ray slices for a specific viewpoint. The top illustration indicates the relative position of each slice with respect to the viewpoint, and following images are the example ray slices from the head dataset.*

slice viewpoint. Each key view represents a view of the volume bounded by the proxy surface from a specific viewpoint. Similar to lumigraph rendering [13], the proxy surface is used for ray interpolation to generate key views of the volume. These key views are then rendered during volume viewing as described Section 5.

The proxy surface can be a simplified isosurface, extracted using the Marching Cubes algorithm [17], corresponding to the exterior material exposed through the transfer function. The choice of proxy surface has a significant impact on the final image quality due to interpolation errors. Figure 4 illustrates this point. The image on the left contains ghosting artifacts around the skin isosurface due to an inaccurate proxy surface and inadequate number of key views. A better proxy surface allows fewer key views for smooth output images, while a more coarse proxy surface will require more key views.

Typically, the proxy surface is computed only once per volume. To reveal the interior structure of the original volume, the proxy can be modified by the user, e.g., by allowing clipping planes to modify the initial proxy surface. This modification effectively changes the bounding surface of the volume, and thus reveals new views of the data. After deformation of the proxy surface, key views must be recomputed using the new proxy geometry.

The key views are interpolated from the ray slices for each

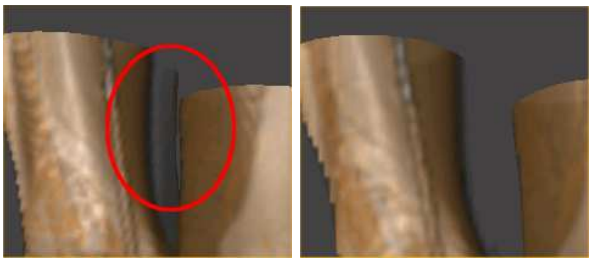

Figure 4: *Comparison of two proxy surfaces for the same number of key views. The image on the left has visible artifacts around the isosurface boundary due to poor isosurface extraction, denoted by the red ellipse, while a more carefully picked isovalue for the proxy surface on the right produces better final images.*

viewpoint as follows: First, the proxy surface is rendered to the frame buffer for a specific viewpoint. Next, the depth values are queried from the depth buffer to determine the distance from the proxy surface to the viewpoint for each pixel in the key view. These depth values are used to index into the ray slices to determine which pixels of the ray slices are required for interpolation of the resulting key view (see Figure 5). We use linear interpolation between ray slice pix-

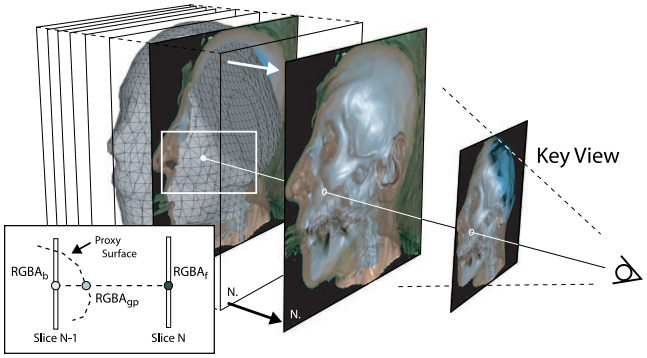

Figure 5: *Key view generation from ray slices and the proxy surface. Once the back and front ray slices for interpolation are determined from the proxy depth value, the appropriate pixel in each ray slice is weighted and summed to determine the resulting key view pixel color for the geometric proxy depth.*

els, although higher order or nearest-neighbor interpolation could be used. We interpolate RGB and alpha. The result is a new key view, which is output in PNG RGBA image format. The same procedure is repeated for every original viewpoint.

The ray slices could also consist of only one slice per viewpoint. In this case no interpolation is required for generating the key views. It suffices to render the proxy surface to the depth buffer and color key view pixels black where the depth of the pixel equals the far plane distance. This single ray slice technique removes scanning artifacts, as well as surfaces external to the proxy surface. We used this single ray slice technique for the visible female and visible female feet datasets presented in Section 6.

The number of key views can be adjusted, as can the viewpoint locations (see Figure 6). These adjustments are highly data, application, and system dependent. For example, a surgical simulation may require highly accurate renderings within a viewing cone (Figure 6a); seismic data may need viewing along 360 degrees in longitude but only for a limited range in latitude (Figure 6b); simulation data exploration may require unlimited freedom for the viewing location (Figure 6c). As we will discuss in Section 6, each viewpoint arrangement requires a minimal number of key views to achieve high image quality.

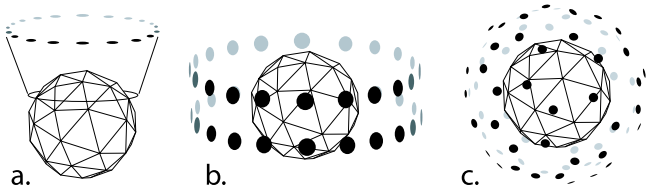

Figure 6: *Viewpoint configurations used in our experiments.*

5 Step 3: Volume Viewing

The key views and the proxy surface are used for image-based rendering. Since each key view contains opacity and color, the set of key views constitutes an opacity light field, parameterized on the proxy surface. We use the system by Vlasic et al. [31] and unstructured lumigraph rendering (ULR) [2] to visualize the opacity light field. This method is interactive on PCs using current graphics hardware.

The input to the algorithm is a triangular mesh (i.e., the geometric proxy) and a set of RGBA key views with corresponding camera parameters. First, we find the set of visible key views for each mesh vertex v. A key view is visible if all the neighboring triangles around v are completely unoccluded in that view. For each vertex v, we then find the k ($k \leq 3$) visible views closest to the new viewing direction d. We call these the closest views for vertex v. Next we compute the blending weights for the closest views using unstructured lumigraph interpolation. Higher blending weights are assigned to views closer to d. Finally, we render each triangle by blending the key views from the closest views of its three vertices. This amounts to blending up to nine textures per triangle, which is carried out solely by the graphics hardware.

To ensure correct alpha blending, the triangles must be rendered in back-to-front order. We pre-compute a BSP tree for the geometric proxy and traverse the tree while rendering. For geometric proxies with relatively small occlusions, we consider all camera views for blending at each vertex, resulting in a slight drop in performance but higher image quality. The details of the implementation, including optimizations and pseudo-code, can be found in [31].

6 Results

The volume rendered images of the ray slices were produced with Kindlmann's volume renderer `miter` [15]. While this volume render produced highly detailed images, it is implemented completely in software, resulting in a slower rendering phase than other implementations. The key view generation program was implemented in OpenGL, and the opacity light field was rendered using DirectX 9.0 and ATI's Radeon 9700 card with 128 MB of texture memory.

We used three datasets for our comparisons – the UNC Chapel Hill CT head, the visible female [25], and the visible female feet. The visible female was cropped from the original

Figure 7: *Example key views for each of our datasets: UNC CT head (left) generated from a clipped proxy surface, the visible female feet (center), the and visible female (right).*

$512 \times 512 \times 1734$ dataset to eliminate empty voxels along the sides of the volume. The size of each dataset is listed in Table 1. Although our experiments focused on regular grids, the IBVR method is easily extendible to irregular grids, as well as dataset sizes of several gigabytes or terabytes.

Dataset	Resolution	Size
UNC CT Head	184 x 247 x 225	10 MB
Visible Female Feet	300 x 350 x 256	51 MB
Visible Female	483 x 275 x 1734	440 MB

Table 1: *Volume data used in our experiments.*

We found that using lossless PNG encoding for the ray slices provided adequate compression, even for a large number of viewpoints. The volume viewing system uses BMP images for the input key views. Due to texture memory limitations on current graphics cards, key views are limited to a resolution of 256×256 pixels. Consequently, the ray slice images were produced with the same resolution. Increased memory capabilities on future graphics cards will allow for higher resolution key views, and thus, higher resolution ray slice images. For the visible female dataset, images were initially rendered with a resolution of 1024×1024 and resampled with a Blackman windowed sinc kernel to reduce undersampling artifacts. Although resampling the images to a smaller resolution resulted in a loss of detail, it is important to note that this reduction occurs after the rendering phase, ensuring a more complete evaluation of the volume integral.

The arrangement of viewpoints for the CT head was six rings (see Figure 6b) spaced four degrees apart. For the visible female feet we used 545 views spanning the hemisphere (see Figure 6c), and for the visible woman we used seven rings spaced four degrees apart. Table 2 lists the number of viewpoints, the number of ray slices per viewpoint, the total memory size of the ray slices for each dataset, and the total memory size of the key views for each dataset.

One set of 20 ray slices for a specific viewpoint for the head dataset required approximately 831 KB of memory. The same number of ray slices for the visible female required 426 KB. From these results it is clear that the memory requirement for ray slice storage is independent of the original dataset size. Instead, memory requirements depend on the number of ray slices and the number of empty or coherent pixels per ray slice, which affects the compression efficiency of PNG. The visible female's non-uniform dimensions required fewer ray slices for most viewpoints, which meant the total ray slice storage was less than for the CT head.

Dataset	Nbr of Views	Ray Slices per View	Total Ray Slices	Total Key Views
Head	600	20	529 MB	154 MB
Feet	545	1	16 MB	105 MB
Female	630	1	18 MB	162 MB

Table 2: *The number of viewpoints and slices per viewpoint for the datasets used in our experiments, along with the memory required for storage of the ray slices and key views.*

The volume rendering was computed on a 64 processor SGI Origin 3800 with 32GB of RAM. Volume rendering of all ray slices took between 1 and 12 hours, depending on the dataset. Key view creation was done on a 2 GHz Pentium 4 PC with 1 GB of RAM. Table 3 lists the processing time for key view generation for each dataset in our experiments, and Figure 7 shows sample key views.

Dataset	Viewpoints	Slices	Processing Time
Head	600	20	495 sec.
Feet	545	1	118 sec.
Female	630	1	135 sec.

Table 3: *The number of viewpoints and slices per viewpoint for the datasets used in our experiments, along with the number of seconds required to generate the key views.*

Proxy surfaces for the datasets were generated using the Marching Cubes algorithm [17]. The results of our experiments indicate that the choice of isovalue (or more specifically, of the resulting isosurface) can greatly effect the quality of the final volume viewing image (see Figure 4). For our datasets, isovalues that closely matched the outermost surface specified by the transfer function produced the best results. Table 4 lists the proxy surface mesh sizes for each dataset in our experiments.

Figure 8 presents images of each dataset from novel view-

Dataset	Mesh Size
Head	5827
Feet	9292
Female	11,143

Table 4: *The sizes of the proxy surface mesh for each of the datasets used in our experiments, given in number of triangles.*

points rendered as opacity light fields, along with their proxy surfaces. The UNC CT head is shown twice with different proxy surfaces to illustrate our method's ability to reveal the internal structure of a volume. All images where captured during an interactive session on a PC. The performance varied from 4.5 frames per second to 20.2 frames per second, depending on the viewpoint.

As with any surface light field rendering method, our technique has problems in areas of concavities of the proxy geometry. These problems are especially apparent when introducing a clipping plane due to the poor geometric approximation of the interior surface. The lack of accurate geometry can lead to jumping of features, also known as texture popping. A solution to texture popping is to use more key views if clipping planes are used or if the geometry has pronounced concavities.

For example, we found that doubling the number of key views minimizes popping artifacts for the UNC CT head when using proxy geometry with a clipping plane. The top row of the head dataset in Figure 8 was rendered using six rings of key views spaced four degrees apart, while the second row was rendered with the key views placed within a viewing cone at two degrees apart. For good surface proxies we found that a key view displacement of four degrees eliminated most popping artifacts.

7 Discussion

Our image-based volume rendering (IBVR) approach offers several advantages over traditional volume visualization methods. An important advantage stems from the independence of each step in the IBVR pipeline. Each of the three stages could be run on separate machines, allowing the user to take full advantage of access to multiprocessor systems or high end workstations. For example, volume rendering could be run on a parallel machine, key views could be generated on a workstation, and the volume viewing could take place on a laptop. If high end systems are not available, all three steps could run on the same desktop PC.

Another advantage is the flexibility to control the amount of data generated by a specific stage of the pipeline. The output of one stage can be restricted based on the capabilities of the system that runs the next stage. For example, the number of ray slices, the amount of compression, and the number of viewpoints in the volume rendering step could be constrained based on the main memory available for key view generation. Or, the number and resolution of key views could depend on the texture memory available on the PC graphics card used for volume viewing.

In contrast to other IBVR methods, our system allows the user to view internal structures of the volume by editing the proxy surface. Although this step is currently non-interactive, the key view generation time is several orders of magnitude faster than re-rendering the volume. This speed-up factor becomes greater as the volume dataset size increases. For example, rendering 630 images of the visible female on the system described in Section 6 required ∼12.25 hours using 32 processors. Generating key views for the same number of viewpoints required only several hundred seconds (see Table 3) on the desktop PC as described also in Section 6. We believe that sufficient optimizations can lead to interactive key view generation.

Presently, our system does not allow for interactive changing of the transfer function, which is a disadvantage for many applications. However, there are currently no systems with this capability that can render very large datasets interactively on a desktop PC. One solution is to use a downsampled version of the data to determine an acceptable transfer function prior to the volume rendering step. The addition of interactive transfer function editing is the subject of future work.

8 Conclusions and Future Work

We have presented the first steps toward integrating opacity light field rendering with volume rendering. Our IBVR pipeline has three distinct stages that can be run on different machines and with different parameters. This offers a lot of flexibility in terms of image quality, speed, and required memory footprint.

An important improvement will be the ability to interactively change the transfer functions for the volume. One possibility is to use pre-integration techniques with information stored in the key views. Alternatively, recent work on Plenoptic Opacity Functions (POFs) [11] as basis functions in a multiprocessor volume rendering pipeline, could be integrated into the IBVR pipeline. Also, optimizing the key view generation stage of the pipeline will allow interactive updates of the proxy surface to reveal the internal structures of the volume.

Acknowledgments

This material is based upon work supported in part by the National Science Foundation under Grants: 9977218 and 9978099, and awards from DOE and NIH. We would like to thank Gordon Kindlmann for the many discussions and assistance with producing the volume renderings in this work. We used his `miter` volume renderer from the `teem` toolkit [15]. We would also like to thank Wojciech Matusik for the inspiring discussions that initiated this work, and Daniel Vlasic for his opacity light field viewer. All figures in the paper were graciously produced by Nathan Galli (`nathang@sci.utah.edu`). Many thanks also to Stefan Guthe who provided us with aligned versions of the visible human datasets. The visible female dataset is courtesy of the visible human project sponsored by the NIH National Library of Medicine [25].

References

[1] Portable network graphics. http://www.libpng.org/pub/png/.

[2] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured lumigraph rendering. In *Computer Graphics*, SIGGRAPH 2001 Proceedings, pages 425–432, Los Angeles, CA, 2001.

[3] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *1994 Workshop on Volume Visualization*, pages 91–98, Washington, DC, October 1994.

[4] B. Chen, A. Kaufman, and Q. Tang. Image-based rendering of surfaces from volume data. In K. Mueller and A. Kaufman, editors, *Volume Graphics 2001*, pages 279–295, Stony Brook, NY, June 2001.

[5] W-C. Chen, J-Y. Bouguet, M. H. Chu, and R. Grzeszczuk. Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields. *ACM Transactions on Graphics*, 21(3):447–456, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[6] Yi-Jen Chiang, Ricardo Farias, Cludio T. Silva, and Bin Wei. A unified infrastructure for parallel out-of-core isosurface extraction and volume rendering of unstructured grids. In *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 59–66. IEEE Press, 2001.

[7] Wagner T. Corra, James T. Klosowski, and Cludio T. Silva. Out-of-core sort-first parallel rendering for cluster-based tiled displays. In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pages 89–96. Eurographics Association, 2002.

[8] M. Cox and D. Ellsworth. Application-controlled demand paging for out-of-core visualization. In *IEEE Visualization*, pages 235–244, 1997.

[9] P. Debevec, Y. Yu, and G. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Proceedings of the 9th Eurographics Workshop on Rendering*, pages 105–116, Vienna, Austria, June 1998.

[10] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics hardware*, pages 9–16. ACM Press, 2001.

[11] J. Gao, J. Huang, H. Shen, and J. Kohl. Visibility culling using plenoptic opacity functions for large volume visualization. In *Proceedings of the IEEE Visualization '03 Conference*, pages 341–348, Seattle, WA, October 2003.

[12] Antonio Garcia and Han-Wei Shen. An interleaved parallel volume renderer with pc-clusters. In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pages 51–59. Eurographics Association, 2002.

[13] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumigraph. In *Computer Graphics*, SIGGRAPH 96 Proceedings, pages 43–54, New Orleans, LS, August 1996.

[14] S. Guthe, M. Wand, J. Gonser, and W. Strasser. Interactive rendering of large volume data sets. In *Proceedings of IEEE Visualization '02*, pages 53–60. IEEE Press, 2002.

[15] Gordon Kindlmann. The teem toolkit. http://www.cs.utah.edu/∼gk/teem.

[16] M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics*, SIGGRAPH 96 Proceedings, pages 31–42, New Orleans, LS, August 1996.

[17] W. E. Lorensen and H. E. Cline. Marching–cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics*, Proceedings of SIGGRAPH 87, pages 163–169, 1987.

[18] W. Matusik, H. Pfister, A. Ngan, P. Beardsley, R. Ziegler, and L. McMillan. Image-based 3d photography using opacity hulls. *ACM Transaction on Graphics*, 21(3):427–437, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[19] M. Meissner, S. Guthe, and W. Strasser. Interactive lighting models and pre-integration for volume rendering on pc graphics accelerators. In *Proceedings of Graphics Interface 2002*, pages 209–218, 2002.

[20] M. Meissner, U. Hoffmann, and W. Strasser. Enabling classification and shading for 3d texture mapping based volume rendering using opengl and extensions. In *Proceedings of the 1999 IEEE Visualization Conference*, pages 207–214, San Francisco, CA, October 1999.

[21] G. Miller, S. Rubin, and D. Ponceleon. Lazy decompression of surface light fields for precomputed global illumination. In *Proceedings of the 9th Eurographics Workshop on Rendering*, pages 281–292, Vienna, Austria, June 1998.

[22] K. Mueller, N. Shareef, J. Huang, and R. Crawfis. Ibr-assisted volume rendering. In *Proceedings of IEEE Visualization Late Breaking Hot Topics*, pages 5–8, October 1999.

[23] Shigeru Muraki, Masato Ogata, Kwan-Liu Ma, Kenji Koshizuka, Kagenori Kajihara, Xuezhen Liu, Yasutada Nagano, and Kazuro Shimokawa. Next-generation visual supercomputing using pc clusters with volume graphics hardware devices. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 51–51. ACM Press, 2001.

[24] K. Nishino, Y. Sato, and K. Ikeuchi. Appearance compression and synthesis based on 3d model for mixed reality. In *Proceedings of IEEE ICCV '99*, pages 38–45, September 1999.

[25] NIH National Library of Medicine. The visible human project. http://www.nlm.nih.gov/research/visible.

[26] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, pages 238–255, July – September 1999.

[27] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The volumepro real-time ray-casting system. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 99)*, pages 251–260. ACM Press/Addison-Wesley Publishing Co., 1999.

[28] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In *Eurographics Rendering Workshop 1997*, pages 23–34, June 1997.

[29] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 109–118, Interlaken, Switzerland, August 2000.

[30] S. Röttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings of IEEE Visualization '00*, pages 109–116. IEEE Computer Society Press, 2000.

[31] D. Vlasic, H. Pfister, S. Molinov, R. Grzeszczuk, and W. Matusik. Opacity light fields: Interactive rendering of surface light fields with view-dependent opacity. In *Proceedings of the Interactive 3D Graphics Symposium 2003*, 2003.

[32] W. Volz. Gigabyte volume viewing using split software/hardware interpolation. In *Volume Visualization and Graphics Symposium 2000*, pages 15–22, Salt Lake City, UT, October 2000.

[33] M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, and T. Ertl. Level-of-detail volume rendering via 3d textures. In *Volume Visualization and Graphics Symposium 2000*, pages 7–13, Salt Lake City, UT, October 2000.

[34] D. Weiskopf, K. Engel, and T. Ertl. Interactive Clipping Techniques for Texture-Based Volume Visualization and Volume Shading. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):298–312, 2003.

[35] O. Wilson, A. Van Gelder, and J. Wilhelms. Direct volume rendering via 3D textures. Ucsc-crl-94-19, Jack Baskin School of Eng., University of California at Santa Cruz, 1994.

[36] D. Wood, D. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. Salesin, and W. Stuetzle. Surface light fields for 3d photography. In *Computer Graphics*, SIGGRAPH 2000 Proceedings, pages 287–296, Los Angeles, CA, July 2000.

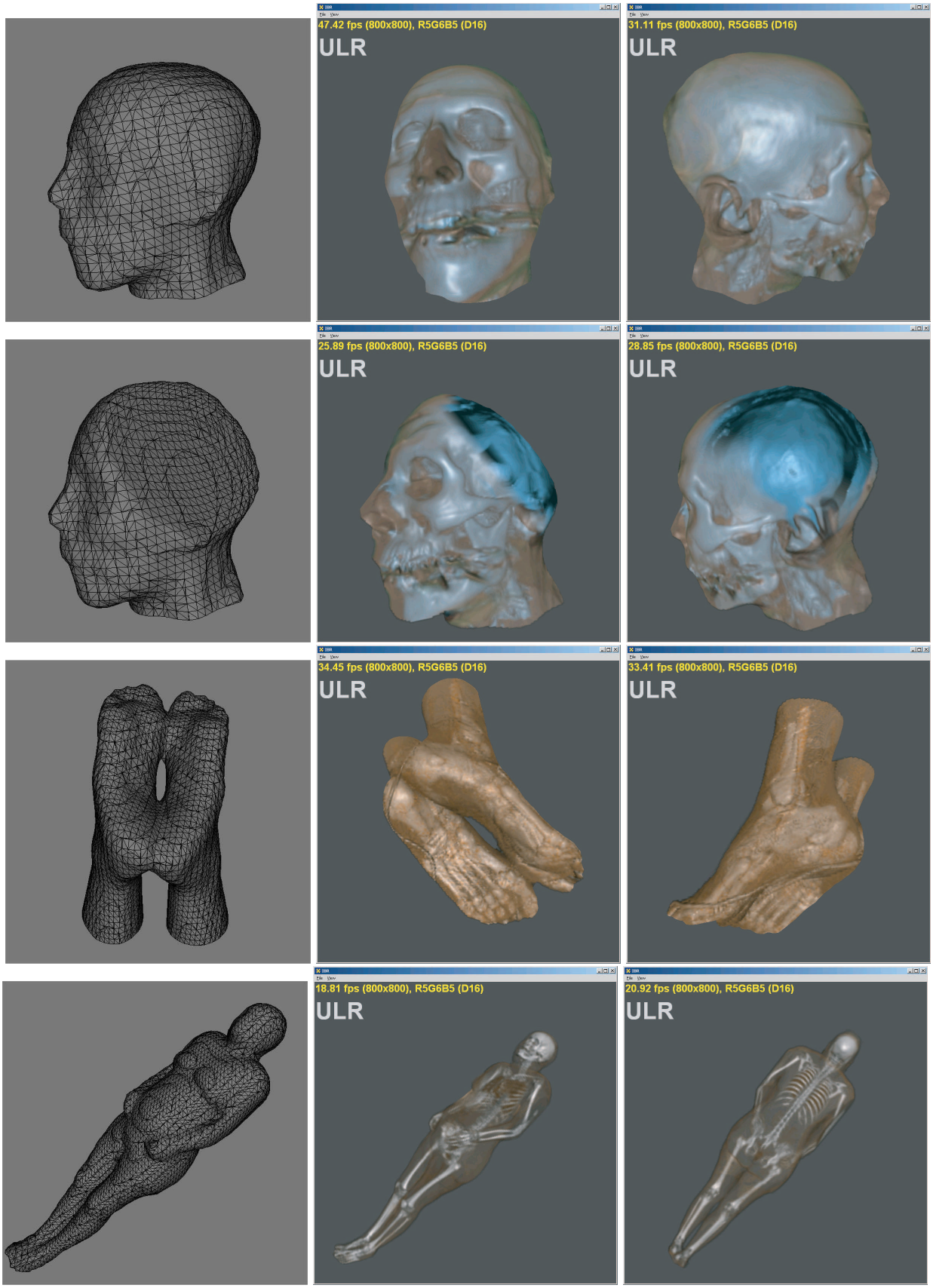

Figure 8: *Results of IBVR using three datasets along with their proxy surface mesh. The first two rows show the UNC CT head, the third row the Visible Female feet, and the fourth row is the Visible Female. The left column images are the proxy surfaces used to generate the key views. All IBVR images were captured during an interactive volume viewing session.*