Eurographics/ IEEE-VGTC Symposium on Visualization 2008
A. Vilanova, A. Telea, G. Scheuermann, and T. Möller
(Guest Editors)

Volume 27 (2008), Number 3

Raycasting of Light Field Galleries from Volumetric Data

Christof Rezk Salama, Severin S. Todt, and Andreas Kolb

Abstract

The paper describes a technique to generate high-quality light field representations from volumetric data. We show how light field galleries can be created to give unexperienced audiences access to interactive high-quality volume renditions. The proposed light field representation is lightweight with respect to storage and bandwidth capacity and is thus ideal as exchange format for visualization results, especially for web galleries.

The approach expands an existing sphere-hemisphere parameterization for the light field with per-pixel depth. High-quality paraboloid maps from volumetric data are generated using GPU-based ray-casting or slicing approaches. Different layers, such as isosurfaces, but not restricted to, can be generated independently and composited in real time. This allows the user to interactively explore the model and to change visibility parameters at run-time.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

In recent years, public web archives have become popular, which give anonymous access to free volume data sets for a variety of different interests. A prominent example is the Digimorph data archive at the University of Texas [UTC07] providing volumetric scans of biological and geological specimen. Several other web sites allow us to access data sets for medical [Rad07] or various other purposes [Vol07, www07]. Most of these galleries, however, provide only the raw volume data set with pre-rendered thumbnail images. Some of the sites provide a free volume rendering software in addition. Other sites have integrated web-based slice viewers.

Although there are data sets open to the public, and we know from personal conversations that a wide audience is interested in viewing such data, the target audience for public data archives, however, still is restricted to computer scientists or technically versed people. The average visitor of such web sites would hardly accept the burden of installing and getting familiar with complex volume rendering software, only to view a few data sets that he finds interesting. Looking at pre-rendered images only, on the other hand, does not convey the fascination of truly volumetric objects. The information contained in volume data cannot be easily displayed in a static image or video sequence. User interaction is crucial in order to explore volumetric data.

To these ends we propose a solution, which combines ease-of-use with the necessary means of interaction, while hiding the complexity of most volume rendering software. Light fields are image-based representations of virtual or real scenes. The major benefit of light field rendering is that the rendering complexity is almost independent of the scene complexity. In this paper we propose a framework for generating and rendering of high quality light field galleries from volumetric data.

We show how existing light field representations can be enhanced to include means of interaction apart from simply changing the viewing position. We demonstrate that the use of light field data instead of volume data as format for archiving and exchanging visualization results has several advantages. A light field data set generated with our framework is *more lightweight* compared to the underlying volume data set with respect to storage capacity. Most volume ray-casters can easily be adapted to generate light fields. GPU-based ray-casting applications may generate light field representations fast and efficiently. The light field data does not contain the volume data itself, but the result of an entire visualization session. Hence, light field representations are ideal to store visualization results in a format that is easy to exchange. This is an important benefit, as it gives scientists at different places the ability to discuss data based on the same interactive visualization. It allows visualization results to be archived for documentation without the usual loss

of interactivity that comes with videos or still images. We also show that Monte-Carlo volume ray-casters, which are not capable of providing images in real-time, can be used to generate high-quality volume light fields. In this case the light field approach allows volumes including translucency and multiple-scattering effects to be displayed in real-time.

2. Related Work

In this section we will review related work on volume visualization and light field rendering. Accounting for the vast number of publications in both fields, we will restrict ourselves to those papers which are most relevant to our approach. A more detailed review of the particular light field representation which is fundamental to our approach is given in Section 3.

2.1. Volume Visualization

A variety of real-time volume rendering techniques have been presented in the past. GPU-based volume rendering techniques for uniform grids either use a slicing approach [WGW94, RSEB*00] or per-fragment raycasting [KW03, RGWE03]. An overview of GPU-based volume rendering can be found in the book by Engel et al. [EHK*06].

Westermann and Sevenich have presented a hybrid CPU/GPU raycasting system [WS01]. The first solely GPU-based implementations of volume raycasting have been published by Krüger and Westermann [KW03] and Roettger et al. [RGWE03]. Recent implementations like this take advantage of the support for loops and conditional branches of modern GPUs.

2.2. Light Field Rendering

Since its introduction by Levoy and Hanrahan in 1996 [LH96] *light field rendering* techniques have continuously improved. The *lumigraph* [GGSC96] employs a polygonal object approximation for depth correction of rays to achieve improved light field rendering results. The approach extends the *two plane* parametrization presented by Levoy and Hanrahan by employing six *light slabs* in a cubic arrangement. The virtual view can be chosen with six degrees of freedom. Discontinuity artifacts, however, appear at the edges of the cubic setup.

Unstructured lumigraph rendering [BBM*01] is targeted at the free-hand acquisition of light fields from arbitrary positions. Uniformity, which is important to avoid discontinuity artifacts, however, cannot be guaranteed with this approach. The polygonal approximation used for depth correction has to be generated and stored separately. *Free-form light fields* [SVSG01] replace the polygonal approximation by per-pixel depth information. Here, depth is evaluated on a per-vertex basis. A proxy mesh is successively subdivided at run-time based on the depth. Rendering performance, however, is limited by the subdivision process. Non-uniform sampling and insufficient depth information again lead to ghosting artifacts.

2.3. Image-Based Volume Rendering

Image-based technqiues has been used to accelerate the rendering of volumes [CS98, MSHC99, SLSM06] and surfaces from volume data [CKT01]. *Unstructured lumigraph rendering* has been applied to volume data by Meyer et.al. [MPHC05] for interactive view synthesis. In practice a high-tesselated polygonal representation and a high amount of sample cameras (>500) is necessary for high-quality image synthesis, resulting in increased memory consumption. Visibility parameters cannot be adjusted during light field rendering.

3. Spherical Light Fields

The light field rendering technique we utilize for building volume rendering galleries is based upon recent advances in image-based rendering technology [TRSK07]. In the following section we will explain the sphere-hemisphere parameterization and the employed raycasting technique for image-based rendering. This technique is expanded in Section 5 to account for volumetric data and different depth layers. We chose this particular technique due to the following requirements:

- Arbitrary viewpoint selection (outside the object) without any preferred viewing direction.
- Efficient light field storage based on parabolic maps with 8-bit per-pixel depth information.
- High-quality image reconstruction in real-time without noticeable ghosting artifacts.

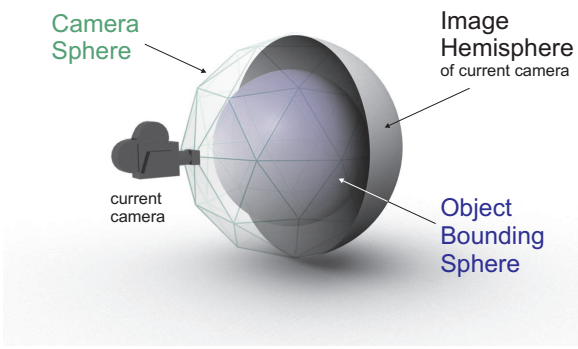

Figure 1: *Sphere-Hemisphere parameterization of the light field. The volume object is enclosed in the blue bounding sphere. Virtual cameras are positioned at the vertices of the camera sphere (green). Each camera is recording the opposing hemisphere.*

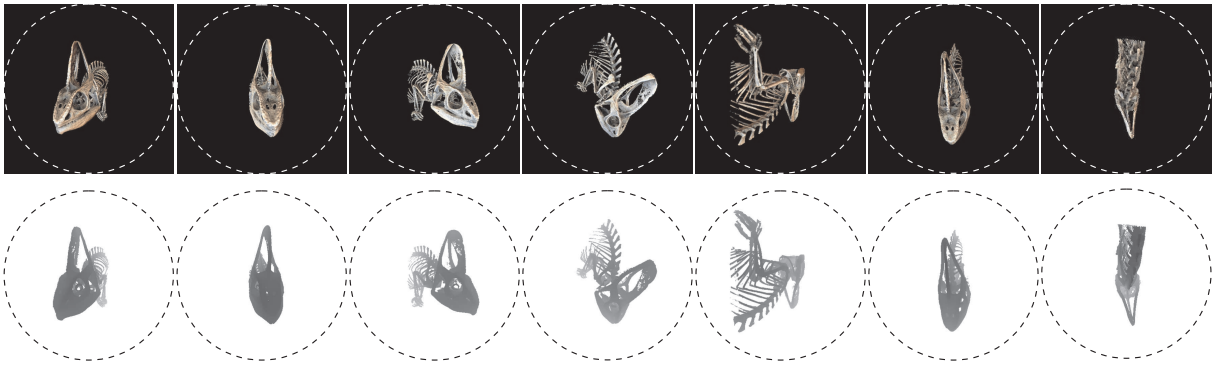

Figure 2: *Seven image samples taken for a spherical light field with 162 cameras. Each image represents a parabolic mapping of the hemisphere for color (top row) and depth (bottom row).*

3.1. Sphere Hemisphere Parameterization

In image-based rendering, the (4-dimensional) plenoptic function is reconstructed from a set of 2D images. Light field parameterizations differ in the *location* of the virtual cameras and the way the camera images are discretized. In our parameterization the cameras are located at the vertices of a uniform tessellation of a sphere around the volumetric object to be acquired. As displayed in Figure 1, the camera sphere must be larger than the bounding sphere of the volume by a factor of $\sqrt{2}$ to ensure that all rays from the current camera through the object boundary sphere will intersect the opposite hemisphere.

Examples of source images for the light field are displayed in Figure 2. For each camera we capture an image of the hemisphere centered about the vector between camera position and the center of the sphere. The image is parameterized as a parabolic map. For each image sample we additionally store the depth of the first intersection with the object (see Section 4). The depth is stored as fraction between the distance of the object intersection point from the camera and the length of the entire ray secant, i.e. the distance between camera and intersection point with the image hemisphere. This allows depth to be efficiently stored at 8-bit fixed point precision.

3.2. Real-Time Light Field Ray-Casting

The ray-casting algorithm presented in [TRSK07], directly uses this light field representation for image-based rendering. The presented approach renders the smooth shaded spherical approximation for light field reconstruction, with the vertices being equivalent to the model camera positions. The light field samples are bound as textures to their associated vertices. For each fragment a ray is cast into the scene. The intersection point with the surface of the captured object is established by subsequently evaluating the depth values stored in the adjacent light field samples for the current ray sample position. For the final intersection point the fragment color value is determined from the weighted sum of the light field samples. Using the ray-casting approach and per-pixel depth correction light fields are rendered without noticeable ghosting artifacts.

4. Generation of Volume Light Fields

In this section we discuss the basic generation of light fields from volumetric data. We can either employ a ray-casting approach or a slice-based solution. The ray-casting approach is easier to implement, in fact we only have to adjust the ray directions to the sphere hemisphere parameterization and render images for each vertex of the camera sphere. Ray-casting directly allows us to determine the depth value of the first hit with a voxel that exceeds a given opacity threshold. Depth information is more intricate to obtain using a slice-based solution.

4.1. Raycasting Approach

The source images for the light field data can be generated by almost any ray-casting algorithm. For each pixel of the final image, a ray is shot through the scene. In our case, however, the image is not a traditional image plane, but an image hemisphere, parameterized as a parabolic map to minimize distortion. For simplicity we assume that the camera sphere has unit radius and the object bounding sphere radius $1/\sqrt{2}$. We further define a camera coordinate system with a virtual camera located at position $\vec{c} = (0,0,-1)^T$ and the image hemisphere centered around $(0,0,1)^T$. A point $\vec{r} = (r_x, r_y, r_z)^T$ on the hemisphere with $(r_z \leq 0)$ is parameterized in parabolic coordinates (u,v), according to:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{1+r_z} \begin{pmatrix} r_x \\ r_y \end{pmatrix} \quad \text{with} \quad u,v \in [-1,1] \qquad (1)$$

To perform the ray-casting, we translate and scale the volume to entirely fit into the object bounding sphere centered about the origin and rotate it to account for the current

camera position. Each pixel (u,v) of the parabolic image is mapped back to the unit sphere, according to

$$r_z = \frac{1-u^2-v^2}{1+u^2+v^2}, \quad r_x = u(1+r_z), \quad r_y = v(1+r_z) \quad (2)$$

Finally, a ray is cast from the camera position $\vec{c}$ to the intersection point with the hemisphere $\vec{r}$. The depth for each viewing ray is determined at the first hit of the ray with the volume, i.e. at the point $\vec{p}$ where the accumulated opacity along the ray exceeds a small specified threshold. The depth value d is calculated as the distance of the point $\vec{p}$ from the camera divided by the length of the ray secant:

$$d = \frac{\|\vec{p}-\vec{c}\|}{\|\vec{r}-\vec{c}\|}, \qquad 0 \le d \le 1 \quad (3)$$

Most GPU-based ray-casting implementations can be adapted to the above parameterization. The basic idea of GPU-raycasting is to store the volume in a 3D texture and cast rays in the fragment program. The fragment program contains a large loop which successively samples the volume, applies the transfer function and computes the ray integral. A straight-forward approach for rendering parabolic maps directly is to rasterize a screen-spaced quad into a viewport mapped to $[-1,1]\times[-1,1]$. A fragment program interprets the screen coordinate as (u,v)-coordinate for the parabolic map, calculates the intersection point $\vec{r}$ according to Equation 2, and casts a ray from $\vec{c} = (0,0,-1)^T$ to $\vec{r}$ through the volume. While this approach is fairly easy to implement, it will calculate several rays at the outer rim of the map which do not intersect the volume at all.

Most GPU-based ray-casters rasterize the front faces of the volume's bounding box in order to generate valid starting points for the rays. This idea can be adapted for the parabolic map. We employ a vertex program which completely neglects the standard projection matrix. Instead, the vertex program does the following:

1. transform each vertex $\vec{v}$ to viewing coordinates (using the model/view-matrix)
2. calculate the intersection $\vec{r}$ of a ray from camera $\vec{c} = (0,0,-1)^T$ to the vertex $\vec{v}$ with the hemisphere. This involves the solving of a simple quadratic equation.
3. transform the resulting intersection point $\vec{r}$ to parabolic coordinates (u,v) as in Equation 1.
4. emit (u,v) as screen coordinates together with the 3D texture coordinate of the first hit and the ray direction from step 2 transformed into texture space. Those values are handed to the fragment program in available GPU registers.

From this information, the fragment program can calculate rays which start at the first intersection with the bounding box. Note, that the fragment program must calculate the correct depth value along the ray and replace the original fragment depth generated by primitive rasterization. Since the parabolic mapping does not preserve straight lines, however, the faces of the bounding box must be tessellated into smaller quadrangles for piecewise linear approximation. Such a task can efficiently be performed by a geometry shader, if supported by the graphics hardware. Otherwise, the tessellation should be pre-computed and stored as vertex buffer in local video memory for maximum performance.

After the ray-casting, the resulting color and depth values are read back from the frame buffer and stored in the light field data.

4.2. Slicing Approach

Adapting a slice-based GPU volume renderer to light field creation is a little bit more intricate. Slices are rendered usually in back-to-front order either with 3D textures [WGW94] or 2D Textures [RSEB*00]. Compositing, i.e the numerical approximation of the ray integral, is performed after the fragment program by the frame-buffer operations (alpha blending). For projecting the slice images onto the parabolic map, a similar vertex program described in the previous section can be applied with a slicing polygon tessellated into small quadrangles to account for the parabolic distortion of straight lines. The RGB part of the image is generated by projecting the tessellated slice images onto the parabolic map and blending the incoming fragments into the frame-buffer in back-to-front order using alpha blending.

Rendering correct depth images requires a separate pass with a modified vertex and fragment program. The vertex program has to calculate the depth of the slice vertex with respect to the sphere-hemisphere parameterization. This is achieved by inserting an additional computation step into the vertex program described in the previous section:

2a. calculate the depth value d of the vertex by dividing the distance $\|\vec{v}-\vec{c}\|$ by the ray secant length $\|\vec{r}-\vec{c}\|$ (see also Equation 3).

In step 4 the depth value is added to the per-vertex registers emitted to the fragment program. Depth values for each fragment of a slicing polygon are thus interpolated from values calculated at the vertices. This is in accordance with the piecewise linear approximation performed by the tessellation of slice images into small quadrangles.

For rendering the depth image, a fragment program is employed which calculates the opacity value of the fragment, by applying the transfer function to the scalar sample obtained from the volume via texture lookup as usual. If the opacity value exceeds a small threshold, the fragment emits the depth value d calculated by the vertex program as RGB triplet (luminance), and sets the alpha value (A) to 1. If the opacity is smaller than the threshold, the fragment program simply outputs an RGBA value of 0. For this depth pass, alpha blending can be replaced by a more efficient alpha test to discard all fragments with alpha less than 1. The back-to-front rendering of the slice images ensures that the color value in the final image is set to the depth value closest to the camera.

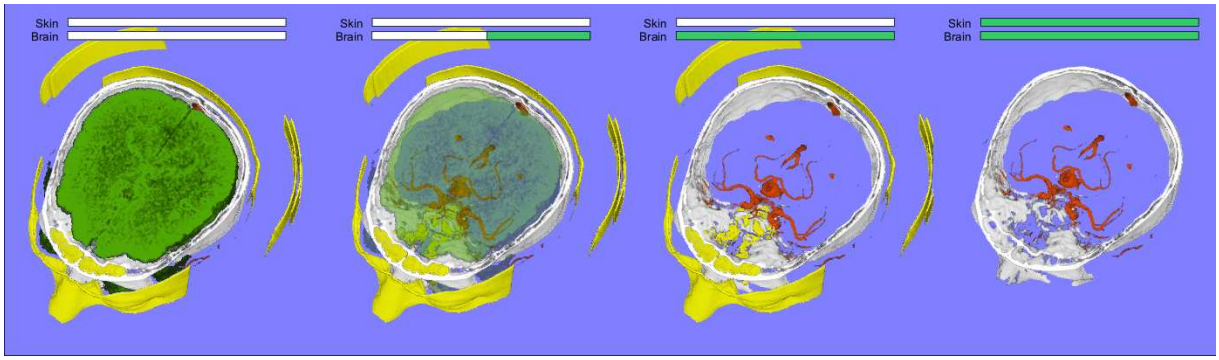

Figure 3: *Multi light field composed of different layers generated from the CT angiography data set: Separate light fields for bone/vessels, skin and brain were generated. The individual layers are depth sorted at pixel level during compositing. The user may interactively adjust the transparency of different layers. Light Fields generated fully automatically from a slice-based volume renderer using semantics described in [RSKK06]*

Compared to the ray-casting approach, the depth images obtained by slicing are less precise, since the depth values are not calculated per-fragment, but interpolated bi-linearly from values computed at the vertices. For a fine tessellation of the slice images this error may vanish, however, at the cost of an increased vertex load and a reduced performance.

5. Multi Light Fields

With the above mentioned approaches light field data of static volumetric objects can be efficiently generated using either slice-based rendering or ray-casting. Rendering of the resulting light field allows the user to view the volume from arbitrary directions outside the boundary sphere. All the interesting information contained in volumetric data, however, cannot be conveyed in a static image or video sequence, and the same holds true for a static light field representation. As the performance of light field rendering is high enough (see Section 8), and local video memory of current graphics hardware is large, we can render several light fields for each frame.

Simultaneous rendering of several light fields allows us to add means of interaction to the light field representation. The different possibilities will be outlined in the following sections.

6. Separate Light Fields for Different Structures

Volume data sets usually contain different structural, functional or anatomical entities. Depending on the purpose of the visualization, light fields are generated separately for each individual entity. If the different entities are not rendered completely opaque (like isosurfaces), the color source images must be extended to RGBA format and the accumulated opacity value must be stored in the alpha portion of the image. This will allow us to incorporate different entities contained in a volume data set, as long as these entities do not intersect each other.

Each light field is rendered in a separate pass into an off-screen render target including a separate depth buffer (render-to-texture). In a final pass a compositing fragment program reads the content of the off-screen targets and computes the final color. The results from each separate light field pass are handed to the compositing shader as color and depth textures for a screen-filling quadrangle. The compositing fragment program sorts the individual layers according to their depth value using odd-even merge sort [Bat68] as outlined in Figure 4. During light field rendering, the user can interactively modify the opacity values of the different layers. The compositing shader finally blends the different layers with the modified opacity value in correct depth order. Note, however, that if the volumetric structures contained in the individual light fields intersect each other in 3D, the compositing will lead to incorrect results.

In combination with sophisticated visualization approaches providing semantic information [RSKK06], the light fields for different structures can be generated in a fully automatic process, after an initial transfer function setup by the user. Figure 3 shows a multi-light field generated from

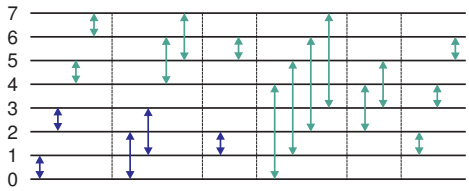

Figure 4: *Odd-even merge-sorting network for eight values. Arrows represent compare-and-swap operations. For sorting four values only the blue arrows are necessary.*

Figure 5: *Light field data sets are generated separately for different ray segments.*

CTA data using the semantic information about anatomical structures included in the transfer function model. The light field comprises three layers, skin, brain and a combined layer for bone and blood vessels. During light field rendering the user may interactively fade the different structures in and out. Note, however, that blending separate pre-rendered images does not provide the same results as a volume rendering of the same structures, since absorption is not additive. Nevertheless, this approach provides a valid means of interaction and exploration for otherwise static volume renditions.

7. Multiple Depth Layers

Another possibility which exploits the capability to simultaneously render multiple light fields is the use of depth layers. In this approach the ray is split into subsequent segments, each of which is rendered into a separate light field as outlined in Figure 5. The depth values d_i at which the ray is split, can either be fixed, or data dependent. Fixed ray segments may only be useful to improve the rendering of highly transparent volumes with varying color, where ghosting artifacts may occur due to depth discrepancies. In practice, however, volume data is rarely visualized this way. For data dependent splitting of ray segments, there are two alternative ways to generate the split points d_i. We can split the ray at a specified set of iso-surfaces or at the transition between segmented regions inside the volume data. This will allow the user to selectively modify the transparency of different layers at run-time. Alternatively, we may split the ray if the accumulated opacity reaches a certain threshold. Such a technique may be used to resolve occlusion problems along the ray, similar to opacity peeling [RSK06].

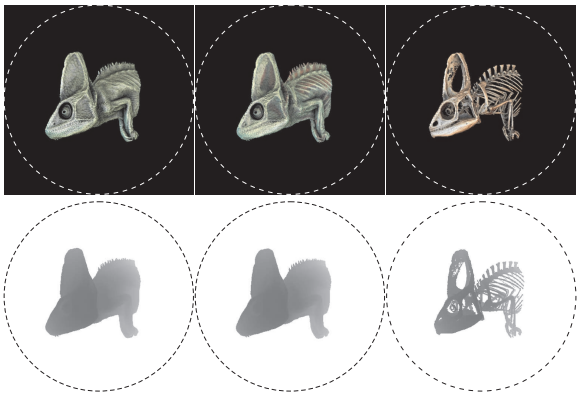

Figure 6: *Different layers generated from the chameleon data set (512^3,16 bit): High-quality isosurface of the skin (left), Transparent skin with soft tissue and scattering (middle), bone structures isosurface (right). All renderings were illuminated with an environment cube map and an ambient occlusion term for soft shadows.*

As in the previous section, an alpha component must be added to light field source images to store the accumulated opacity for the ray segment. If the rays are split consistently, the depth sorting step introduced in the previous section may be omitted, since the light field images already are in the correct depth order.

The result of a layered depth technique is shown in Figure 6 for the UTCT Veiled Chameleon data set. The first ray segment starts from the eye and ends at the first hit of the isosurface of the skin. The second layer comprises the region of soft tissue between the skin and the bone. The final layer represents the bone structures. The light field was generated with the Monte-Carlo volume ray-tracing algorithm described in [RS07], which took about 1 second per source image. The entire light fields consists of 162 cameras with 3 layers, resulting in an overall number of 486 images at 512×512 resolution. *This data set was submitted with an executable of the example viewer*

8. Results and Discussion

The image quality for light field reconstruction of surface data was thoroughly evaluated in [TRSK07]. These results are also valid for isosurfaces from volume data. For semi-transparent volume rendering, the depth value stored in the light field data does not necessarily correspond to the observed color value along the ray calculated by the integral along the ray. We have evaluated volume light field reconstruction using camera spheres computed from successive subdivison of an icosahedron, resulting in camera spheres with 12, 42, 162 camera positions.

Subjective evaluation of the images show that camera spheres with 162 cameras yield good light field reconstruction in practice. Lower resolutions may lead to visual artifacts due to the above mentioned depth discrepancies and to errors at the silhouette of the volume. Even with 42 images per light field, ghosting artifacts are not a major problem unless for highly transparent objects. However, in volume data sets with complex internal structures, there may be concavities that are not captured by any of the 42 cameras. Resolutions higher than 162 cameras, e.g. 642 cameras, can easily be captured, but do not improve the image quality significantly.

For an objective evaluation of the image quality, we have compared the result images from light field reconstruction to a ray-caster solution for a semi-transparent volume. The results are displayed in Figure 7 and Table 1, and support our subjective evaluation. The upper row of Figure 7 shows the

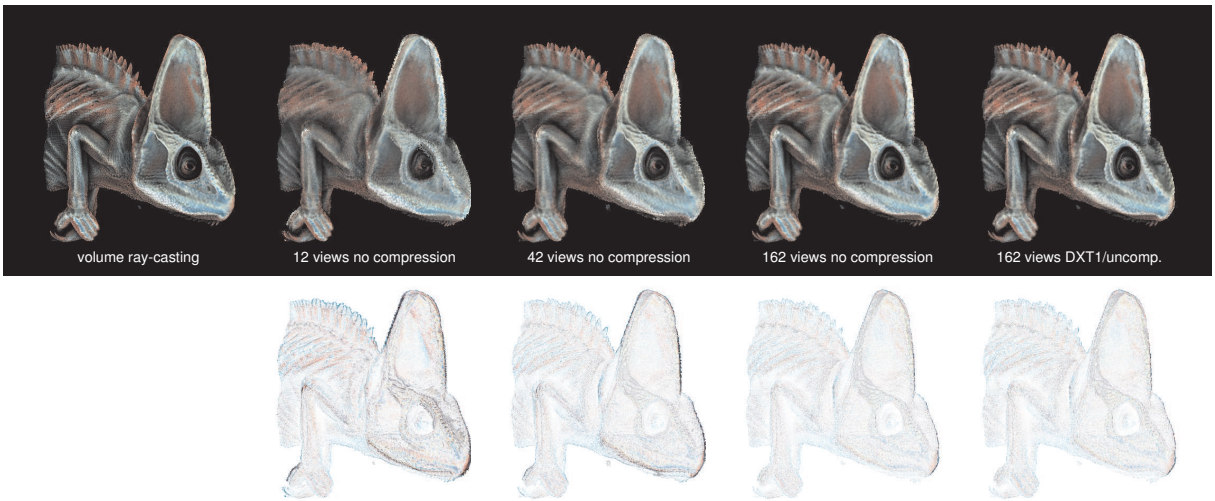

Figure 7: *Chameleon data set with a transparent skin. Light field reconstruction for different camera resolutions and texture compression schemes. The leftmost image represents the ground truth. The bottom row shows difference images from a ray-casting solution. (Difference images are inverted)*

num. cameras	RGB compression	depth compression	size	size zipped	pixel error	performance
12	uncompressed	uncompressed	12 MB	ca. 1.5 MB	10.41 [4.08%]	38.42 fps
42	uncompressed	uncompressed	43 MB	ca. 5 MB	7.84 [3.07%]	38.61 fps
162	uncompressed	uncompressed	166 MB	ca. 18 MB	6.34 [2.48%]	35.00 fps
162	S3TC RGB_DXT1	uncompressed	62 MB	ca. 6 MB	6.47 [2.53%]	35.28 fps
162	S3TC RGB DXT1	Luminance LATC1	41 MB	ca. 5 MB	6.48 [2.54%]	35.46 fps
162×3	S3TC RGB DXT1	uncompressed	186 MB	ca. 15 MB	na	12.56 fps

Table 1: *Evaluation of the light field data structure for different camera resolutions and texture compression schemes for source images of* 512×512.

images from light field reconstruction and difference images to a reference image obtained by ray-casting. The pixel error in Table 1 is the average error of one pixel color component calculated respectively from 30 difference images with randomized viewing directions. The remaining error of about 2.5% is not due to ghosting, but to a minor loss of high-frequency details caused by texture filtering of the source images.

The performance data displayed in Table 7 refers to rendering a single-layer light field into a viewport of size 512×512 on an NVidia Geforce 8800 GTX board with 768MB memory. The bottom row refers to a multi light field with 3 layers. Note that the performance does not significantly depend on the number of camera images, which indicates that light field rendering is not a bandwidth limited process, but clearly fragment-limited. Light field rendering greatly benefits from the unified shader model. A significantly reduced performance is observed on the previous generation of graphics boards. There is no technical limit for the number of light fields to be displayed. The only limit is the target performance. If three light fields are rendered instead of one, the frame rate will be about one third and the time required for compositing is negligible, as shown in the bottom row of Table 1.

The 3 bottom rows in Table 1 show that standard texture compression techniques, such as S3TC RGB_DXT1 for color and LATC1 for depth images, significantly reduce the file size of the light field data set with only very little influence on the image quality. The *size* entry refers to the light field data in a format that can directly be read by the graphics hardware. The current light field format still has some potential to reduce the memory footprint. The *zipped size* entry denotes the data size after standard zip compression to be used in a web gallery.

The light fields presented in this paper are meant as supplement, not as an alternative to volumetric data sets. A drawback of the presented light field rendering technique is that fly-throughs, such as virtual endoscopic views, are not supported due to the view point being restricted to lie outside

the convex hull of the object. Compared to direct volume rendering techniques the quality of the recorded light fields are limited by the resolution of the acquired images, not by the resolution of the original volume data. The individual choice of image resolution for providing a web gallery of volume data should this be determined with respect to the intended image quality, performance and the available bandwidth.

9. Conclusion

We have presented a versatile framework for the generation of light field representations for volume data. It allows interactive display of high-quality volumetric representations, which cannot be rendered in real-time. We have shown that light fields are suitable to present visualization results to the public, where fast access is more important than explicit control over all rendering parameters. They allow visualization results to be accessed and exchanged fast and easily, without the necessity to give access to the original (possibly restricted) volume data.

References

[Bat68] Sorting Networks and their Applications. In *Spring Joint Computer Conference, AFIPS Proceedings* (1968), vol. 4, pp. 307–314.

[BBM*01] BUEHLER C., BOSSE M., MCMILLAN L., S.GORTLER, COHEN M.: Unstructured lumigraph rendering. In *Proc. ACM SIGGRAPH* (2001), pp. 425–432.

[CKT01] CHEN B., KAUFMAN A. E., TANG Q.: Image-based rendering of surfaces from volume data. In *Volume Graphics* (2001), pp. 279–295.

[CS98] CHOI J., SHIN Y.: Efficient image-based rendering of volume data. In *Proc. Pacific Graphics* (1998).

[EHK*06] ENGEL K., HADWIGER M., KNISS J., REZK-SALAMA C., WEISKOPF D.: *Real-Time Volume Graphics*. AK Peters, Ltd., 2006.

[GGSC96] GORTLER S., GRZESZCZUK R., SZELISKI R., COHEN M.: The lumigraph. In *Proc. ACM SIGGRAPH* (1996), pp. 43–54.

[KW03] KRÜGER J., WESTERMANN R.: Acceleration Techniques for GPU-based Volume Rendering. In *Proc. IEEE Visualization* (2003), pp. 287–292.

[LH96] LEVOY M., HANRAHAN P.: Light Field Rendering. In *Proc. ACM SIGGRAPH* (1996), pp. 31–42.

[MPHC05] MEYER M., PFISTER H., HANSEN C., C.R.JOHNSON: *Image-Based Volume Rendering with Opacity Light Fields*. SCI Institute Technical Report UUSCI-2005-002, University of Utah, 2005.

[MSHC99] MUELLER K., SHAREEF N., HUANG J., CRAWFIS R.: IBR-Assisted Volume Rendering. In *Proc. IEEE Visualization, Late Breaking Hot Topics* (1999).

[Rad07] RADIOLOGY.UIOWA.EDU: Volume Archive at the Dept. of Radiology, University of Iowa, USA. http://radiology.uiowa.edu/downloads/, 2007. *last visited 12/01/2007*.

[RGWE03] RÖTTGER S., GUTHE S., WEISKOPF D., ERTL T.: Smart Hardware-Accelerated Volume Rendering. In *Procceedings of EG/IEEE TCVG Symposium on Visualization VisSym '03* (2003), pp. 231–238.

[RS07] REZK-SALAMA C.: GPU-Based Monte-Carlo Volume Raycasting. In *Proc. Pacific Graphics* (2007).

[RSEB*00] REZK-SALAMA C., ENGEL K., BAUER M., GREINER G., ERTL T.: Interactive Volume Rendering on Standard PC Graphics Hardware. In *Proc. Graphics Hardware* (2000).

[RSK06] REZK-SALAMA C., KOLB A.: Opacity Peeling for Direct Volume Rendering. *Computer Graphics Forum (Proc. Eurographics) 25*, 3 (2006), 597–606.

[RSKK06] REZK-SALAMA C., KELLER M., KOHLMANN P.: High-Level User Interfaces for Transfer Function Design with Semantics. In *Proceedings of IEEE Visualization* (2006).

[SLSM06] SHAREEF N., LEE T.-Y., SHEN H.-W., MUELLER K.: An image-based modelling approach to gpu-based rendering of unstructured grids. In *Volume Graphics* (2006), pp. 31–38.

[SVSG01] SCHIRMACHER H., VOGELGSANG C., SEIDEL H.-P., GREINER G.: Efficient Free Form Light Field Rendering. In *Proc. Vision Modeling and Visualization* (2001), pp. 249–256.

[TRSK07] TODT S., REZK-SALAMA C., KOLB A.: *Fast (Spherical) Light Field Rendering with Per-Pixel Depth*. Tech. rep., 2007. http://www.cg.informatik.uni-siegen.de/Publications.

[UTC07] UTCT: Volume Archive at the Texas Advanced Computing Center, University of Texas at Austin, USA. http://utct.tacc.utexas.edu/, 2007. *last visited 12/01/2007*.

[Vol07] VOLVIS.ORG: Volume Data Repository at the WSI/GRIS, University of Tübigen, Germany. `http://www.volvis.org/`, 2007. *last visited 12/01/2007*.

[WGW94] WILSON O., GELDER A. V., WILHELMS J.: *Direct Volume Rendering via 3D-textures*. Tech. Rep. UCSC-CRL-94-19, Univ. of California, Santa Cruz, 1994.

[WS01] WESTERMANN R., SEVENICH B.: Accelerated volume raycasting using texture mapping. In *Proc. IEEE Visualization* (2001).

[www07] WWW.CG.TUWIEN.AC.AT: Volume Archive at the Vienna University of Technology, Austria. http://www.cg.tuwien.ac.at/research/vis/datasets/, 2007. *last visited 12/01/2007*.