Flexible Direct Multi-Volume Rendering in Dynamic Scenes

Sören Grimm[†], Stefan Bruckner[†], Armin Kanitsar[‡], Eduard Gröller[†]

[†] Vienna University of Technology, Austria
[‡]Tiani Medgraph AG, Austria
Email: {grimm, bruckner, gröller}@cg.tuwien.ac.at, kanitsar@tiani.com

Abstract

In this paper we describe methods to efficiently visualize multiple intersecting volumetric objects. We introduce the concept of V-Objects. V-Objects represent abstract properties of an object connected to a volumetric data source. We present a method to perform direct volume rendering of a scene comprised of an arbitrary number of possibly intersecting V-Objects. The idea of our approach is to distinguish between regions of intersection, which need costly multi-volume processing, and regions containing only one V-Object, which can be processed using a highly efficient brick-wise volume traversal scheme. Using this method, we achieve significant performance gains for multi-volume rendering. We show possible medical applications, such as surgical planning, diagnosis, and education.

1 Introduction

Direct volume rendering is an important and flexible technique for visualizing 3D data. It allows the generation of high quality images without a need of an intermediate interpretation. Traditionally, medical volume visualization systems feature only simple scenes consisting of a single volumetric data set. It has been proposed to extend these scenes to a more complex description [13]. In this paper we introduce a flexible data structure called V-Objects for representing scenes containing multiple volumetric objects. We present an efficient approach to render a scene composed of V-Objects. Furthermore, by presenting practical examples for possible applications we demonstrate that medical visualization systems can take advantage of V-Objects. The main contributions of this paper are an efficient technique to render V-Objects and to show that common medical volume visualization systems can greatly extend their flexibility by supporting concurrent display of multiple volumetric objects.

The presentation of the paper is subdivided as follows: Section 2 surveys related work. In Section 3 we present as a new data-structure V-Objects and an approach to efficiently render a scene composed of multiple V-Objects. In Section 4 we present possible medical applications. Finally, in Section 5 we give ideas for future work and conclude our work.

2 Related Work

The past decade has seen significant progress in volume visualization, driven by applications such as medical imaging. A number of different volume rendering algorithms have been developed, improved, and extended [8, 6, 12]. Today, it is possible to perform interactive high-quality volume rendering on commodity hardware [15, 2]. Hybrid algorithms have been designed, which allow concurrent display of intersecting volumetric and polygonal objects [9, 5]. Direct rendering of scenes consisting of multiple volumetric objects, however, has received less attention. With the increasing performance of modern hardware, we feel that this topic will become more important in the future. Our work was inspired by Leu and Chen, who introduced a two-level hierarchy for complex scenes of non-intersecting volumes [7]. While their approach allows multi-volume rendering, the individual volumes cannot intersect. However, the display of intersecting semi-transparent objects can be a powerful visualization technique. In the work of Nadeau [13] intersecting volumes are possible, however, the whole scene description has to be re-sampled every time a volume's transformation changes. The idea behind our approach is to allow multiple intersecting volumetric objects to be rendered directly, without requiring costly re-sampling. Thus, our algorithm is well-suited for animations.

3 V-Objects

A V-Object is an element of a scene description which is connected to one volumetric data source. The V-Object comprises the following visual properties:

- *Illumination:* This includes the selected Illumination model and its parameters. For example, for the Phong-Blinn Illumination model the ambient, diffuse, specular, and emissive material coefficients are specified.
- *Transfer Functions:* For each defined region in the attached volumetric data source a mapping function between scalar values and colors as well as opacities is stored.
- *Region of Interest:* An arbitrary number of planes define a convex region of interest.
- *Transformation:* An affine transformation defining position, orientation, and scaling of the V-Object.

The separation of visual properties and volumetric data sources allows an arbitrary number of varying representations of the same data source within one scene, as illustrated in Figure 1. This is achieved by assigned several V-Objects to the same volumetric data source. To take full advantage of the V-Objects we apply direct volume rendering to simultaneously visualize a scene consisting of several V-Objects. The properties of a V-Object influence the points in a scene at which it is defined. For example, regions of the volume which are classified as transparent due to the transfer function specification are not part of a V-Object.

The following section briefly describes how mono-volume raycasting can be greatly accelerated by a bricked volume layout and an appropriate brick-wise processing scheme. This performance benefit is then exploited in Section 3.2 to accelerate multi-volume raycasting.

3.1 Mono-Volume Rendering

The discrepancy between processor and memory performance is rapidly increasing, making memory access a potential bottleneck for applications which have to process large amounts of data. Volume raycasting is prone to cause problems, as it generally

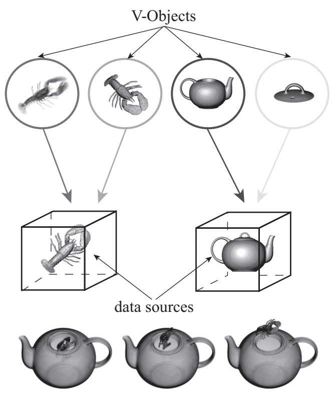

Figure 1: Different representations of the same data source using V-Objects.

leads to irregular memory access patterns. There are approaches which address this issue, such as Knittel et al. [4] or Mora et al. [11]. The basic idea is to employ a spread memory layout. They achieve high frame-rates. As we are interested in rendering of multiple volumes these approaches are not well suited, due to their immense memory requirements. Therefore we first take a look at an appropriate memory layout.

The most common way of storing volumetric data is a linear volume layout, see Figure 2(a). Volumes are typically thought of as a number of

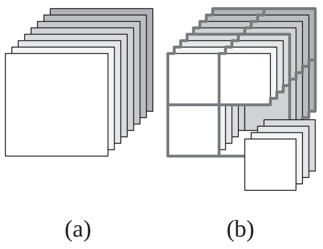

Figure 2: (a) Linear memory layout. (b) Bricked memory layout.

two-dimensional images (slices) which are kept in an array. This three-dimensional array has the advantage of a simple address calculation. It has disadvantages when used in raycasting, due to its view-dependent memory access patterns. A much more suitable memory layout for raycasting is based on the concept of bricking. It has been shown that bricking is one way to achieve high cache coherency, without increasing memory usage. Bricking supposes the decomposition of data into small fixed-size data blocks. Each block is stored in linear order, see Figure 2(b). The basic idea is to choose the block size according to the cache size of the architecture so that an entire block fits in a fast cache of the system. To take full advantage of a bricked memory layout, especially of its very good cache coherency, a suitable addressing and processing scheme must be used. We apply the one proposed in [3].

They propose a brick-wise processing scheme, as shown in Figure 3 and present an approach to take full advantage of commodity multi- and hyper-threading technology. Different bricks are processed in parallel on different physical CPUs, while single bricks are processed in parallel on different logical CPUs on the corresponding physical CPU. By this processing scheme, the main memory to CPU transfer is considerably reduced.

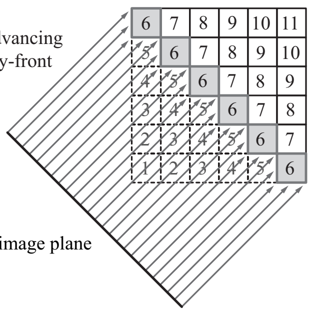

Figure 3: Brick-wise processing scheme. The numbers correspond to processing order of the bricks, with respect to the visibility order.

The other important issue is the access of data in a bricked volume layout. Accessing data in a bricked volume layout is very costly. One way to avoid the costly address computation is to inflate each brick by an additional layer. However, this considerably increases the memory consumption. A solution to this issue is the proposed addressing scheme in [3]. They propose to use offset lookup-tables to address the direct neighbors of one sample. The key to efficiency are a small lookup table and a sophisticated indexing of this look-up table. Basically, they differentiate the possible sample positions, within a brick, by the locations of the needed neighboring samples. This is illustrated in 2D in Figure 4 for a eight- and a 26-neighborhood. The principle can be mapped straightforwardly to 3D. Each shaded area defines a subset. For each subset, the offsets to access the corresponding eight or 26 neighbors are constant, assuming that one brick is stored one after the other in memory. With the aid of these subsets, a very small offset lookup table can be created. This table can then be very efficiently indexed, as shown in [3]. By using this addressing scheme, the neighbors can be accessed in constant time, similar to the addressing in a linear volume layout.

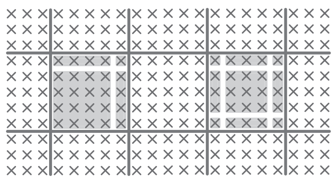

Figure 4: Different brick addressing patterns: Solid lines represent brick boundaries and crosses correspond to samples. Left side: 8-neighbor-addressing pattern (4 subsets). Right side: 26-neighbor-addressing pattern (9 subsets).

The next important issue to accelerate volume raycasting is to efficiently skip empty regions. As we are interested in rendering multiple V-Objects, the memory consumption as described earlier is of great importance. We therefore apply the approach presented in [2], which is based on a hybrid transparent region removal and skipping technique. The work-flow is shown in Figure 5. At first transparent regions are removed on a brick basis (Figure 5a → Figure 5b). Then to support even more refined removal of smaller transparent regions they perform octree projection (Figure 5b → Figure 5c). Due to efficiency reasons the octree subdivision, contained in each brick, does not fully go down to individual cells. This granular resolution of the octree leads to approximate rays-volume intersections. To overcome the resulting performance penalty they use a Cell Invisibility Cache to skip the remaining transparent cells (Figure 5c → Figure 5d). We use the complete pipeline of the hybrid transparent region removal and skipping technique. However, our main focus is on the octree projection which we will extend and exploit for multi-volume rendering in Section 3.2.

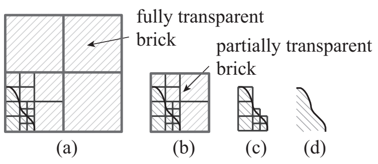

Figure 5: General work-flow of our hybrid transparent region removal and skipping technique: Thick solid lines are brick boundaries, thin solid lines are octree boundaries, / are transparent regions and \ is visible volume data. (a) $\rightarrow$ (b): transparent region removal of bricks. (b) $\rightarrow$ (c): removal based on an octree projection. (c) $\rightarrow$ (d): removal using a cell invisibility cache.

3.2 Multi-Volume Rendering

Before describing our approach to accelerate multi-volume rendering, we briefly discuss how we perform compositing of multiple volumes [1]. The basic idea of volume raycasting is to cast for each pixel of the image plane a ray through the volume. For a single object we determine the final color and opacity of the image pixel by the over-operator [14] in front-to-back order. That is, at each re-sample location, the current color and alpha values for a ray are computed in the following way:

$$\begin{aligned} c_{out} &= c_{in} + c(x)\alpha(x)(1-\alpha_{in}) \\ \alpha_{out} &= \alpha_{in} + \alpha(x)(1-\alpha_{in}) \end{aligned} \quad (1)$$

c_{in} and α_{in} are the color and opacity the ray has accumulated so far. x is the reconstructed function value and $c(x)$ and $\alpha(x)$ are the classified and shaded color and opacity for this value.

For the simultaneous processing of multiple volumes it must be decided how they should be combined. We consider volumes as clouds of particles and take into account all volume simultaneously at the corresponding sample position. The simultaneously compositing of multiple volumes is achieved by sequentially applying Equation 1 for each individual volume. Hereby we obtain an approximation for compositing multiple volumes at the same location.

As described in Section 3.1, the key to high performance for mono-volume rendering is to optimize the memory access pattern. This can be achieved by using a bricked volume layout. However, to find such a suitable memory layout for multi-volume rendering is very difficult, due to the unpredictable memory access pattern. Every time multiple volumes have to be simultaneously processed, several data entities from memory regions far apart have to be accessed. This leads to enormous cache trashing. To keep this cache trashing penalty as low as possible, we propose to separate multi-volume rendering from mono-volume rendering within a scene composed of multiple volumes. While the processing of the intersection between multiple volumes requires costly computations, the non-intersecting regions could be efficiently computed by using a mono-volume rendering technique. In the following we will address this issue and present a solution to efficiently decompose the scene in mono- and multi-volume rendering regions.

We start by distinguishing between the concepts of a mono- and a multi-volume renderer. Both types of renderers are initially supplied with a list of rays. Each entry in the ray data structure contains, among other data, a start and an end depth. A mono-volume renderer processes one V-Object using the brick-wise processing scheme described in the previous section. It operates efficiently by using the cache coherent volume traversal presented in [3]. A multi-volume renderer performs compositing in multiple V-Objects. As the V-Objects can have different volumetric data sources and transformations, efficient brick-wise traversal is in general not possible. Thus, the multi-volume renderer performs classical ray traversal. As re-sample positions along a ray are likely to lie within totally different memory locations for different V-Objects, lacking cache coherence results in considerable performance penalties. In our approach, a mono-volume renderer is set up for every V-Object in the scene, while there is only one global multi-volume renderer.

The main idea is to first identify different segments along a ray's path in the initialization phase. There are two basic types of segments:

- *Mono-object segment:* A mono-object segment is a continuous interval along a ray lying within the same V-Object.
- *Multi-object segment:* A multi-object segment is a continuous interval along a ray lying

within one or more V-Objects. Thus, each mono-volume segment is also a multi-volume segment.

In our approach, when performing this kind of segmentation, the goal is to maximize the combined length of mono-object segments, as mono-object segments can be processed more efficiently. In general, for this the intersections between a ray and all V-Objects need to be known. We apply a conservative approach by using the octree projection explained in the previous section. For each V-Object, the entry and exit points of all rays can be obtained by projecting its octree and setting the depth test to either less or greater. By a depth sort of these entry and exit points mono- and multi-object segments can be determined.

All mono-object segments are added to the mono-volume renderer which is responsible for this V-Object. These segments can then be processed efficiently, as brick-wise traversal is possible. All other segments are added to one global multi-volume renderer. This renderer traverses every ray segment and performs multi-volume compositing in every step for all V-Objects which are defined at one re-sample location. This distribution of ray segments between mono- and multi-volume rendering is done in the initialization phase. After all ray segments have been assigned to their corresponding renderer, the actual raycasting process is started. All renderers operate independently. After they have finished, a final compositing step is required which combines the values accumulated in each segment of a ray.

The distribution of ray segments between mono- and multi-volume renderers is illustrated in Figure 6. Ray A consists only of one mono-object segment (A1). Ray B consists of the mono-object segments B1 and B3 which pass through V-Object II, and the multi-object segment B2 which passes through the intersection of the two V-Objects. Ray C consists of the mono-object segments C1 and C3 which pass through V-Object II, and the multi-object segment C2 which passes through the intersection of the two V-Objects. Thus, A1 is the only ray segment added to the mono-volume renderer for V-Object I. The segments B1, B3, C1, and C3 are added to the mono-volume renderer for V-Object II. The multi-volume segments B2 and C2 are added to the global multi-volume renderer. Through this kind of distribution only a fraction of the ray segments have to undergo costly multi-volume processing.

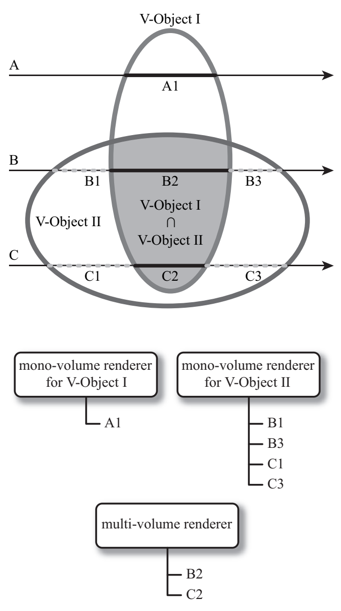

Figure 6: **Top:** Segmentation of three rays, A, B, and C, in a scene consisting of two V-Objects. **Bottom:** Distribution of ray segments among mono- and multi-volume renderers.

The performance gains achieved through our multi-volume rendering approach depend on the size of intersections between the individual V-Objects. Typically, intersections are small, which allows us to exploit the high performance of mono-volume processing to accelerate the rendering. Moreover, if there is no intersection between any V-Object, our algorithm evaluates to a pure mono-volume renderer.

We have compared our approach to a standard multi-volume raycaster where rays are processed sequentially, performing re-sampling and compositing in each of the V-Objects defined at every point along a ray. Figure 7 shows the speed-ups we achieve for different degrees of intersection. As can be seen from the figure, the performance gains due to our algorithm depend on the size of the intersection between the V-Objects. For typical scenes, where size of intersections is normally not excessively large, we achieve speed-ups of about 2.5.

Scene 1	Scene 2	Scene 3
43.5%	21.7%	0.0%
1.93	2.43	2.78

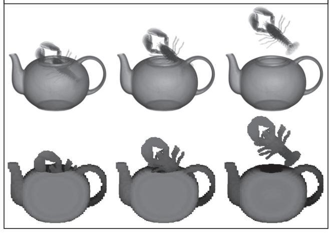

Figure 7: Obtained speed-ups for different degrees of intersection. The first row displays the ratio between the combined lengths of all ray segments and the multi-object ray segments, row two shows the achieved speed-up of our approach compared to brute-force multi-volume rendering. The images (512×512) in row three show the rendering results. The last row displays the corresponding octree projections of both V-Objects. Test system specifications: Intel Pentium 4, 2.4 GHz, 1 GB RAM.

4 Exploring the capabilities of V-Objects

In general, in a volume rendering system there exist several means to enhance visual perception. Such means are for example transfer function specification, segmentation and clipping. Since their introduction to volume visualization by Levoy [8], piecewise linear transfer functions mapping scalar values to colors and opacities are featured in virtually every volume visualization system. Many researchers have proposed to extend transfer functions to higher dimensions to increase their usability, as their specification is a non trivial task. Some approaches for semi-automatic transfer function definition have been presented, however fully automatic transfer function selection remains a widely unsolved problem. Still, much research needs to be done to produce results automatically as shown in Figure 8 (color plate), first row, image (a).

One way to simplify the transfer function specification is segmentation. Many methods exists to identify certain structures within the data. Most of these approaches produce a labelling of the data. Different transfer functions can be assigned to identify regions or objects. Figure 8 (color plate), first row, image (b) shows an example of segmentation based on region growing. The main vascular structures and the kidneys have been segmented.

One problem in the visualization of volumetric data is occlusion. While transparency can be useful to simultaneously display different structures of interest, it can lead to cluttering when used extensively. It is therefore common to cut away opaque objects to reveal occluded features. Cutting can be performed using axis aligned or arbitrarily oriented planes, convex regions or arbitrarily shaped objects. However complex cutting shapes are often difficult to interpret by the user. Figure 8 (color plate), first row, image (c) shows the use of cutting planes to reveal an aneurism. In the following we show how such basic tools can be greatly enhanced by the use of V-Objects.

Though many systems allow modification of transfer functions, and illumination properties can be specified on a per object basis, certain limitations apply: objects typically can not intersect, are unique and static. By assigning V-Objects to components of a data set we can overcome these limitations in a natural way. For example, with V-Objects it is possible to move an object while keeping a virtual copy in place. These two objects can have a totally different appearance. This capability can be used in applications such as surgical planning, education, illustration, and for investigation or exploration of data. In the following we give several examples for the use of V-Objects in combination with the previous described basic tools. We show examples for moving objects, simultaneously changing the appearance of objects, time varying, and multi-modal data. As some of these concepts are very hard to illustrate using still images, we have produced several animation sequences. Our application features an interactive tool for the generation of animation sequences using key-framing. V-Objects can be interactively positioned in the scene and their properties can be modified. Between the key-frames, V-Object states are interpolated. This enables specification of animation paths, transfer function fades, light movement, control of clipping planes, change of data sources, enabling and disabling of objects, etc. For each of these properties, different interpolation schemes can be applied. In the following we present our application examples.

4.1 Advanced Browsing Techniques

Traditional techniques for inspecting volumetric data like cutting, involve removing portions of the data. This has the disadvantage of potentially hiding important contextual information. McGuffin et al. [10] have presented novel tools for browsing volume data which employ transformations and deformations of semantic layers contained in the data set. In the future, we feel that multi-volume rendering will be an important tool to improve the visual quality of such interaction techniques. In Figure 8 (color plate), second row, we demonstrate that V-Objects can be used to realize this kind of visualization. Although we support only affine transformations currently, we are confident that we can extent our concept to more complex deformations in the future. One the other hand V-Objects can be used to indicate the positions of displaced structures. Multiple V-Objects can be assigned to the same structure in the data, only one of these V-Objects is deformed, the other objects remain in place to indicate the original position in space. Transparency is especially useful to indicate the position while not hiding surrounding important information. In Figure 8 (color plate), third row, we show an example of such an application of V-Objects. The images represent stills of an animation we made. In the animation you can see, for example, that a kidney is relocated to the left to reveal an occluded tumor. Other interesting features are shown, such as transfer function fading, moving, and object specific cutting planes.

4.2 Time Varying Data

Visualization of time varying volume data is a very complex task. Animations due to their dynamic nature often do not allow an in depth analysis of certain data characteristics. Static images on the other hand, often suffer from cluttering when many time-steps are visualized. Therefore, it has been proposed to apply more advanced projection and mapping techniques to aid the understanding of such data. For example, Woodring at al. [16] apply hyper-slicing to a 4D dataset. The flexibility of V-Objects allows to use a variety of different mappings by combining transfer functions and varying the spatial arrangement of objects. We show an example of a time varying data set, see Figure 8 (color plate), fourth row. It is a electrocardiogram triggered CT scan of beating heart. Time is mapped to color and opacity. This type of visualization allows the three dimensional examination of several time steps simultaneously. The fanning in time allows to convey similarities and differences in the progress of time. Furthermore, a topological relationship between different time steps is visualized. In general, V-objects support the explorations of useful layouts and mappings. In the future, we therefore seek to further investigate the application of multi-volume rendering to 4D data visualization.

4.3 Multi-Modal Imaging

Multi-modal imaging is a method for combining disparate sets of 3D imaging data that contain complementary information on overlapping length scales. In medicine, for instance, morphological modalities are combined with functional modalities in order to increase the information content of the resulting image. The concept of V-Objects inherently includes the ability to perform multi-modal visualization of registered data sets. In Figure 8 (color plate), first row, image (d) we show an example of a combination of computed tomography (CT) and positron emission tomography (PET). While the CT method supplies high precision, it is difficult to distinguish between tumors and healthy tissue. PET, on the other hand, is a functionally oriented method that allows to identify tumors, but only provides lower resolutions.

5 Conclusion and Future Work

We presented V-Objects, a concept of modelling scenes consisting of multiple volumetric objects. We have shown that an efficient volume renderer based on a brick-wise traversal scheme can be extended to handle scenes comprised of multiple possibly intersecting V-Objects. As regions of intersection typically do not cover all the objects, we achieve significant performance gains by identifying mono-volume regions and performing efficient brick-wise traversal of these regions. The advantage of our approach is that it allows to efficiently render multiple intersecting volumetric objects directly from their grid representation. Multi-volume rendering is a promising technique for visualizing medical data. We showed three examples for its application: browsing, time varying data, and multi-modal imaging. Each of them emphasized that the concept of V-Objects can provide advanced means to explore and investigate data. In the future, we will further investigate the topic of multi-volume visualization as we believe it has great potential to improve medical applications.

The animation sequences described in this paper and additional material are available at:
http://www.cg.tuwien.ac.at/research/vis/adapt/2004_vobjects

6 Acknowledgements

The work presented in this publication has been funded by the ADAPT project (FFF-804544). ADAPT is supported by *Tiani Medgraph*, Vienna (http://www.tiani.com), and the *Forschungsförderungsfonds für die gewerbliche Wirtschaft*, Austria. See http://www.cg.tuwien.ac.at/research/vis/adapt for further information on this project. The used data sets are courtesy of AKH Vienna and Univ.-Klinikum Freiburg.

References

[1] W. Cai and G. Sakas. Data intermixing and multi-volume rendering. *Computer Graphics Forum*, 18(3):359–368, 1999.

[2] S. Grimm, S. Bruckner, A. Kanitsar, and E. Gröller. Memory efficient acceleration structures and techniques for cpu-based volume raycasting of large data. In *IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics*, 2004. To appear.

[3] S. Grimm, S. Bruckner, A. Kanitsar, and E. Gröller. A refined data addressing and processing scheme to accelerate volume raycasting. *Computers and Graphics*, 28(5), 2004. To appear.

[4] G. Knittel. The Ultravis system. In *IEEE Symposium on Volume visualization*, pages 71–79, 2000.

[5] K. A. Kreeger and A. E. Kaufman. Mixing translucent polygons with volumes. In *Proceedings of IEEE Visualization*, pages 191–198, 1999.

[6] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. *Computer Graphics*, 28(Annual Conference Series):451–458, 1994.

[7] A. Leu and M. Chen. Modelling and rendering graphics scenes composed of multiple volumetric datasets. *Computer Graphics Forum*, 18(2):159–171, 1999.

[8] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

[9] M. Levoy. A hybrid ray tracer for rendering polygon and volume data. *IEEE Computer Graphics and Applications*, 10(2):33–40, 1990.

[10] M. McGuffin, L. Tancau, and R. Balakrischnan. Using deformations for browsing volumetric data. In *Proceedings of IEEE Visualization*, pages 401–408, 2003.

[11] B. Mora, J. Jessel, and R. Caubet. A new object order ray-casting algorithm. In *Proceedings of IEEE Visualization*, pages 107–113, 2002.

[12] K. Mueller, N. Shareef, J. Huang, and R. Crawfis. High-quality splatting on rectilinear grids with efficient culling of occluded voxels. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):116–134, 1999.

[13] D. R. Nadeau. Volume scene graphs. In *Proceedings of the IEEE symposium on Volume visualization*, pages 49–56, 2000.

[14] T. Porter and T. Duff. Compositing digital images. *Computer Graphics*, 18(3):253–259, 1984.

[15] S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser. Smart hardware-accelerated volume rendering. In *Proceedings of the Symposium on Data Visualisation*, pages 231–238, 2003.

[16] J. Woodring, C. Wang, and H. Shen. High dimensional direct rendering of time-varying volumetric data. In *Proceedings of IEEE Visualization*, pages 417–414, 2003.

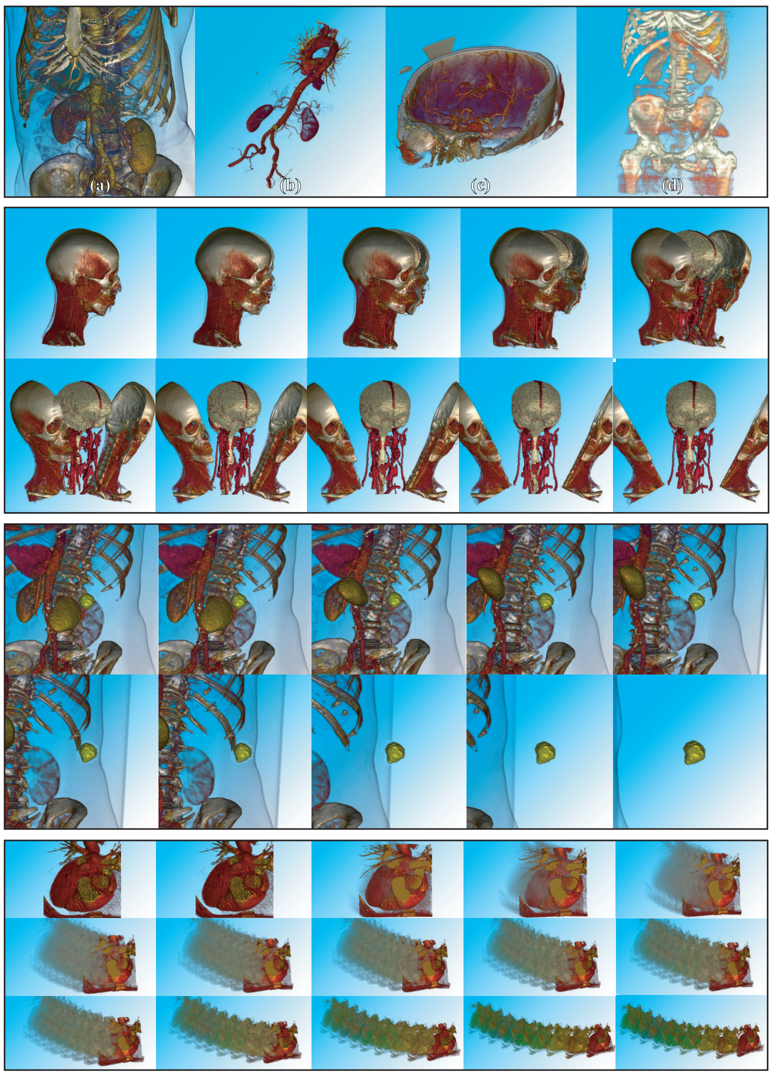

Figure 8: **First row:** (a) Transfer function example. (b) Segmentation example. (c) Clipping example. (d) Fusion of CT and PET scan based on V-Objects. **Second row:** Stills of animation to illustrate advanced browsing of volumetric data based on V-Objects. Virtual dissection of the human skull, uncovering the nervous and vascular system. **Third row:** Stills of animation to illustrate advanced browsing of volumetric data based on V-Objects. Enhancing anatomical features by spatial displacement. **Fourth row:** Stills of animation to illustrate 4D visualization based on V-Objects. Fanning in time allows to convey similarities and differences in the progress of time.