GPU-based Multi-Volume Rendering for the Visualization of Functional Brain Images

Friedemann Rößler* Eduardo Tejada* Thomas Fangmeier†
Thomas Ertl* Markus Knauff†

Abstract

Nowadays, medical imaging procedures provide a great variety of data obtained with different modalities. To offer a deeper insight into this data the simultaneous visualization of these images, not regarding their source, must be provided. One example is the field of functional brain imaging, where the activation of brain regions during cognitive tasks is measured. This data has to be visualized whitin its anatomical context - the brain - in order to have an spatial impression of the position of the activation. To that end, multi-volume visualization must be supported. In this paper we describe a flexible framework for GPU-based multi-volume rendering, which provides a correct overlaying of an arbitrary number of volumes and allows the visual output for each volume to be controlled independently. We also present, a visualization tool specific for the rendering of functional brain images, we built on top of this framework. This tool includes different GPU-based volume rendering techniques, on the one hand for the interactive visual exploration of the data, and on the other hand for the generation of high-quality visual representations. We demonstrate the effectiveness of our tool with data gathered from a cognitive study, in regard to performance and quality results.

1 Introduction

The field of cognitive neuroscience seeks to understand the links between human thoughts, feelings, and actions, and the functions of our brains. Its main belief is that all facets of our psychic life have a neuronal basis. Early research in this field primarily explored which psychic functions are distorted if parts of the brain have been damaged by accidents, tumors, or strokes. Today, however, the *via regia* to explore the neural basis of mental activities are the so called neuroimaging methods, like functional Magnetic Resonance Imaging (fMRI), of which the main goal is to make visible the activities of the brain.

The three dimensional visualization of these functional brain images in relation to their anatomical context would help cognitive scientists gaining a deeper insight into the data. In this sense, Direct Volume Rendering (DVR) techniques have been proven helpful in many medical areas, due to the fact that they provide a three dimensional impression of the shape of the data, whilst showing the inner structures. Thus, its use for the visualization of fMRI data is a clear option. However, in contrast to other applications, fMRI visualization must

*Institute for Visualization and Interactive Systems, University of Stuttgart, D-70569 Stuttgart, Germany, {friedemann.roessler, eduardo.tejada, thomas.ertl}@vis.uni-stuttgart.de

†Center for Cognitive Science, University of Freiburg, D-79098 Freiburg, Germany, {fangmeier, knauff}@cognition.iig.uni-freiburg.de

deal with multiple volumes, one anatomical reference and one or more functional measurements. Hence, special rendering techniques for the geometrically correct overlaying of these volumes must be used.
Motivated by this particular problem, we developed a flexible visualization framework for multi-volume rendering. This framework is based on hardware-accelerated single volume rendering [CCF94] and overcomes the occlusion problem, that always arises when multiple data is presented in a single 3D view, by allowing to control the visual attributes of each of the volumes independently. For instance, the transparency of the anatomical brain could be increased to provide a clearer look at the activation inside. Even completely different rendering modes, like direct volume rendering for the functional data and illuminated isosurfaces for the brain, could be combined.
On top of this framework we built an application for the visualization of fMRI data, which takes the output of SPM [SPM05], a well-established software to analyze brain activities, and supports the cognitive scientists in all phases of their scientific work, starting from the first inspection of rough data and ending with the presentation of results at conferences and in scientific publications.
The remaining of this paper is organized as follows. After having a short look at related work in the field of fMRI visualization and multi-volume rendering we give a brief description of the neuroimaging methods that provide the basis of the paper. In the main part of the paper we present our flexible framework for GPU-based multi-volume rendering, followed by the description of the rendering methods we integrated to counteract the specific problems of fMRI visualization. We conclude with a brief presentation of the use of our fMRI visualization tool with functional data obtained from a cognitive study carried out at the University of Freiburg and some performance measurements for the implemented rendering techniques.

2 Related Work

Since hardware-accelerated volume rendering is nowadays widely known among the visualization community, in this section we will focus on previous work aimed at rendering multiple volumes and generating visual representations of functional brain data. However, we refer the reader to the notes of the excellent SIGGRAPH'04 course on real-time volume graphics [EHK$^+$04] for a comprehensive review of the state-of-the-art of GPU-based volume visualization algorithms.
Kreeger and Kaufman [KK99] present an algorithm which renders opaque and/or translucent polygons embedded within volumetric data. The processing occurs such that all objects are composited in the correct order, by rendering thin slabs of the translucent polygons between volume slices using slice-order volume rendering.
In [SZV01] Stokking et al. describe a generic method, called normal fusion, for integrated three-dimensional (3D) visualization of functional data with surfaces extracted from anatomical image data. The first part of the normal fusion method derives quantitative values from functional input data by sampling the latter along a path determined by the (inward) normal of a surface extracted from anatomical data; the functional information is thereby projected onto the anatomical surface independently of the viewpoint. Fusion of the anatomical and functional information is then performed with a color-encoding scheme

based on the HSV model.
Grim et al. [GBKG04] developed methods to efficiently visualize multiple intersecting volumetric objects. They introduced the concept of V-Objects, which represent abstract properties of an object connected to a volumetric data source. Also a method to perform direct volume rendering of a scene comprised of an arbitrary number of possibly intersecting V-Objects was presented.
Commercial [Sie05] and non-commercial software [Ror05, Sin05] with similar features are also available. As in [SZV01], these software packages project the functional data onto the surface of the anatomical data. This projection is usually made by following the normal to the surface or the viewing vector. Both approaches however are incorrect since the location of the functional data either would appear to move when the viewpoint changes [Ror05] or deep objects will appear greatly magnified [SZV01, Sin05].
Leu and Chen [LC98, LC99] presented CPU methods for modelling and rendering complex multi-volume scenes. They proposed a two-level hierarchy to reduce the storage consumption and extended single-volume rendering methods to multi-volume environments.
Preim et al. [PSHOP00] describe the use of GPU-based volume rendering technics in clinical applications. The most important aspect they deal with, is the direct visualization of multiple-source medical image data by means of image-based, object-based and texture-based rendering approaches. For this, they use registration of the data gathered from different sources (e.g. CT and MRT) to assure a correct overlapping during the merging of the data.
Cai and Sakas [CS99] proposed three levels of data intermixing and their rendering pipelines in direct multi-volume rendering, which discriminate image level intensity intermixing, accumulation level opacity intermixing, and illumination model level parameter intermixing.

3 Neuroimaging in Cognitive Neuroscience

Contemporary research in the field of cognitive neuroscience is to a great extent performed by using fMRI. This method takes advantage of the fact that cognitive processes lead to a local increase in oxygen delivery in the activated cerebral tissue [FR85]. Physically, the fMRI technique relies on the understanding that deoxy-hemoglobin is paramagnetic relative to oxy-hemoglobin and the surrounding brain tissue. Increased presence of oxy-hemoglobin leads to changes in the local magnetic field homogeneity which are commonly referred to as the Blood-Oxygen-Level-Dependent (BOLD) effect [OLKT90, Rai01]. A local increase in oxygen delivery is thought to be correlated with brain activation.
To measure these changes in blood flow a number of people are placed one after the other in a magnetic resonance tomograph. The principle of such fMRI experiments is to measure brain activation of quickly repeated intervals and to explore differences among them. Typically, the baseline activity is measured when the volunteer is at rest, and other measurements are taken when the participant performs certain cognitive tasks. In the simplest experimental design, the activity in the baseline condition is then subtracted from the activity measured during the performance of the cognitive tasks.
The resulting data can be statistically analyzed. Areas in which statistically significant differences were measured are presumed to have been activated by the cognitive task. In more

sophisticated experiments, combinations of experimental conditions are compared to other combined conditions. To illustrate the results, the patterns of activation are usually transferred into so-called fMRI images, in which the most visible regions correspond to the areas activated by the cognitive task. A great majority of cognitive neuroscientists simply use the output of the SPM (Statistical Parametric Mapping) software [SPM05] in which the brain activities are statically analyzed. It has been developed by members of the Wellcome Department of Imaging in London and allows the analysis of whole sequences of brain imaging data. The sequences can be a series of images from different groups of people, or time-series from the same subject.

4 The Visualization Framework

As already mentioned in section 1, we developed a visualization framework for multi-volume rendering which allows to control the visual attributes and the rendering mode for each of the rendered volumes independently. To achieve this high degree of flexibility we built the framework completely object-oriented and defined abstract interfaces for all the main components. By providing a new concrete implementation for some of these abstract components, the framework could easily be adapted to new rendering tasks. This way the framework is not restricted to fMRI visualization, but could be used for any other multi-volume visualization problem.

4.1 Basic Structure

The framework basically consist of three layers (Figure 1) with strictly separated responsibilities. In the lowest *model layer* the data models which should be visualized are defined. At the moment, the only supported model type is a volume, but other types, like polygon meshes, could easily be integrated. A volume is defined by an array of scalar values which are arranged on a regular three-dimensional grid. The scalar values are stored in the original domain of the volume, e.g. the statistical activation maps provided by the SPM software are given with 32-bit floating point accuracy.

On top of the model layer, the *scene layer* is built which combines a couple of volumes (models) to a multi-volume scene and enriches the models with additional visual and geometrical information. The basic element of the scene layer is the actor, which puts a model together with a transformation matrix that defines the model's position and orientation in the scene. Additionally any other attributes needed for visualization could be attached. For example an actor for volumes contains an extra transfer function, which describes the mapping of the volume's scalar values to color values.

The interpretation of the scene is done in the *render layer*. Primary the model data is mapped to the graphics domain (usually 8-bit per channel) and stored in the memory of the GPU. Then a visual representation of the scene is rendered. All these tasks of mapping and rendering are managed by a central *renderer* component which knows the scene by storing a pointer to its root actor group. A 3D renderer additionally contains a camera object, where all the information needed for 3D projection is stored.

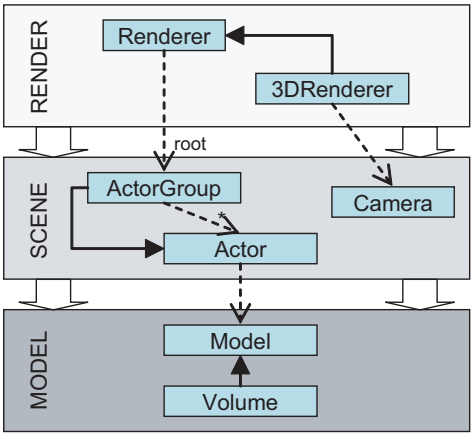


Figure 1: The three layers of the visualization framework with the basic classes and interfaces. The relations between the classes are given in a UML-like style. Solid arrows show inheritance, dashed arrows show, that the pointed element is part of the element the arrow comes from

4.2 Multi-Volume Rendering

With the advent of graphics hardware with support for 3D textures, rendering techniques based on view aligned slicing became popular [CCF94]. These techniques store the volume data as 3D textures on the GPU. The data in these textures is mapped to a number of slices used as proxy geometry, which are alpha blended in a back-to-front order. These slices are calculated by intersecting the bounding box of the volume with planes perpendicular to the viewing direction. Thus, the texels to be mapped to each slice are computed using trilinear interpolation whitin the 3D texture. Additionally, the distance between the slices could be chosen arbitrarily and thus the rendering quality could be directly manipulated.

Multi-volume rendering based on view aligned texture slicing is straightforward to implement. Each volume is sliced, as done for view-aligned single volume rendering, and the slices are then intermixed in the correct geometrical order (depth sort) on a shared slice stack (Figure 2). The slices on the stack are rendered in a back-to-front order and blended to the framebuffer. If the next slice to be rendered belongs to a volume that is not the current one, the render state has to be changed. For instance, a new volume texture and perhaps another vertex or fragment program must be bound.

We integrated a special multi-volume renderer to our framework (Figure 3). The essential feature of this renderer is that the rendering of a single slice is not done by the renderer itself but delegated to so-called *shaders*. For each volume (actor) an individual shader could be defined and stored in the shader map. The advantage of this method is that each of the shaders only has to deal with the rendering of one single volume and that it is possible to combine arbitrary rendering modes without changes of the basic rendering algoritm. All the multi-volume rendering tasks are done by the central multi-volume renderer. Even the slicing is decentralized and done by individual slicers, although at the moment we provide only an ordinary view aligned slicer. However, by implementing a new slicer other model types,

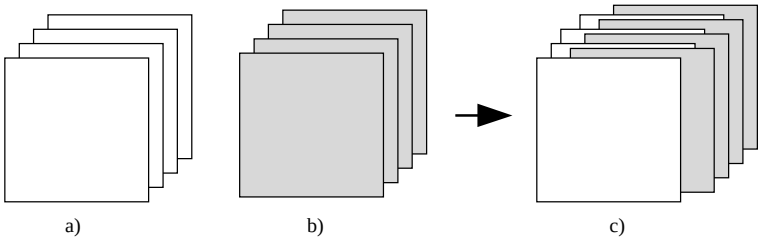


Figure 2: Multi-volume rendering by independently slicing each volume (a and b) and intermixing the slices in the correct order on the slice stack (c).

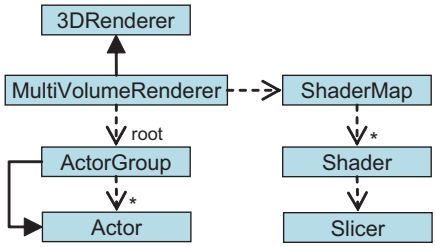


Figure 3: Class diagram of the multi-volume renderer

for example polygonal models, could be easily integrated into the multi-volume scene. Based on the architecture of the multi-volume renderer we get the following basic rendering algorithm (pseudocode):

```
void renderMultiModelScene()
{
    for each actor in the scene {
        shader = getShader(actor);
        shader.prepare();
        shader.slicer.sliceModelAndAddToStack(actor.model, stack);
    }
    for each slice on the stack {
        previousSlice = currentSlice;
        currentSlice = slice;
        if (currentSlice.actor != previousSlice.actor)
           changeRenderMode(previousSlice, currentSlice);
        currentSlice.shader.render(currentSlice);
    }
}
```

As it can be seen, the rendering is divided into two parts. In the first part, for each actor in the scene the assigned shader is prepared for the rendering. This means that, for instance, the corresponding volume texture and fragment program are loaded to the GPU. Then the model (volume) is sliced and the slices are sorted to the slice stack. To avoid artifacts, especially arising in combination with clipping, the slice distance is not choosen idependently

for each volume, but is fixed to the minimal requested sampling distance of all volumes, which is computed in a preliminary step.
In the second part the slices are rendered. Only if the actor of the actual slice differs from that of the previous one, the render mode, like bounded textures and fragment programs, is changed. This way, we assure that state changes only happen if they are really necessary. This fact comes into account, if volumes of different sizes or with different positions are rendered.

4.3 Resource Management

The decoupling of models, scene description and rendering itself, allows an arbitrary number of visual representations to be generated, e.g. two- and three-dimensional views, of the same scene at the same time. On the other hand, this flexibility avoids, that the different render components could share the same hardware resources, like the volume textures on the GPU, directly. To nevertheless support the sharing of those resources and by this save valuable memory and computing time, we integrated a global resource management system into our visualization framework. This layer manages all kind of resources which different components of the framework may want to share and takes care of their creation, distribution and deletion. Figure 4 shows the class diagram of this module.

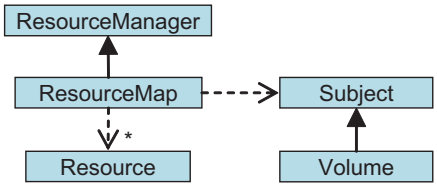


Figure 4: Class diagram of the resource managing module

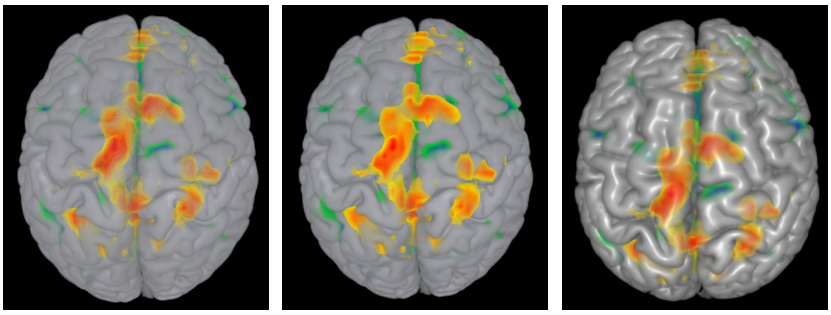
The basic component of the resource management is the *resource*. This is a class, which could control any kind of additional data needed for the visualization of a scene. This could either be the data stored on the GPU, like textures, fragment programs etc., or data computed and stored on the CPU, for example volume gradients which are needed for illumination. A resource not just keeps the data but also cares for its creation.
Related to the resource management there are two types of objects in the framework: those which are possible sources of a resource, like volumes and transfer functions, and those which are consuming the resources, like renderers and shaders. The sources are all derived from a common class, called *subject*. For each subject the global resource manager holds a resource map, where all related resources are stored and can be accessed by an unique ID.
Subjects are usually undergoing several changes during their life time. To keep the resources always up-to-date, they have to be informed about those changes. Because of this, each subject keeps a log of its changes. This log is checked each time a consumer wants to access a resource, and in case the state of the resource differs with that of the subject, the resource updates itself. At the end of a subject's life cycle, it automatically informs the resource manager of this event, so that it could perform the deletion of all the related resources, which are no longer needed.

5 Visualization of fMRI Data

The visualization of functional data involves solving specific problems not found in traditional volume rendering. The need for an anatomical context in the form of a template brain to offer a frame of reference for the graphical representation of the functional data, arises some considerations related to the efficient visualization of multiple volumes. To that effect, we implemented different known volume visualization techniques on the basis of the special overlaying method we developed to provide the user with a set of tools that would facilitate the exploration of the data. It is important to note, that we use preprocessed functional data, which is already aligned with the template brain. By this, there is no need of registration.

5.1 Rendering

One consideration in the development of these tools was the hardware on which the application could be used. To support a wider range of hardware we implemented the fast and widely known viewport-aligned slice-based volume rendering algorithm [CCF94] (Figure 5a).
However, current graphics hardware supports more complex algorithms that achieve better visual results, such as pre-integrated volume rendering [EKE01]. This algorithm was also included as an additional rendering mode, together with oversampling as an alternative to the user for generating high-quality volume renderings as can be seen in Figure 5b. Additionally, lighting is supported in our application for both rendering modes [WE98]. This provides a further clue for the 3D perception of the volume in space as shown in Figure 5c.



(a) (b) (c)

Figure 5: Render modes for simultaneous visualization of a template volume and a functional dataset. In (a) simple direct volume rendering (DVR) is used, whilst pre-integrated volume rendering is used in (b). Lighting in direct volume rendering is shown in (c).

Rendering non-polygonal isosurface [EKE01] also offers a different perspective for the visualization of functional data. By means of transparent isosurfaces for the rendering of the template model, it is possible to gain a clearer insight into the functional data rendered

with direct volume rendering (DVR), whilst still having a context that provides a reference in space (Figure 6).

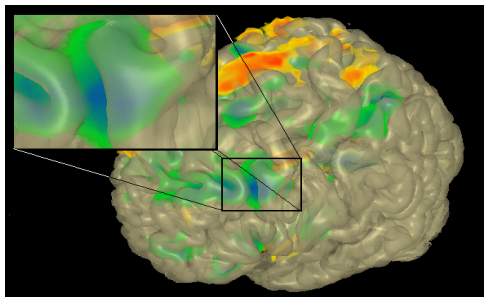
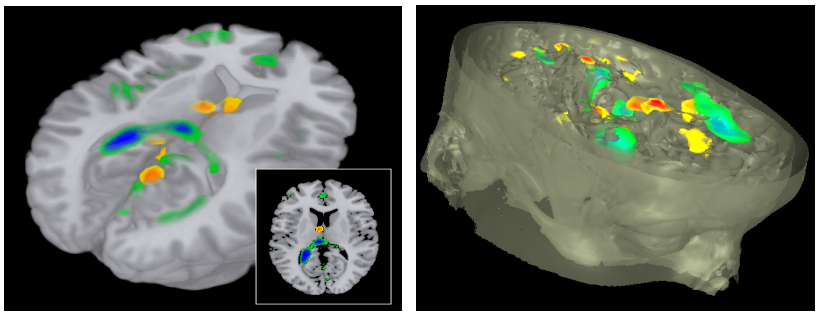


Figure 6: Non-polygonal isosurface with a detail on a selected area.

5.2 Clipping

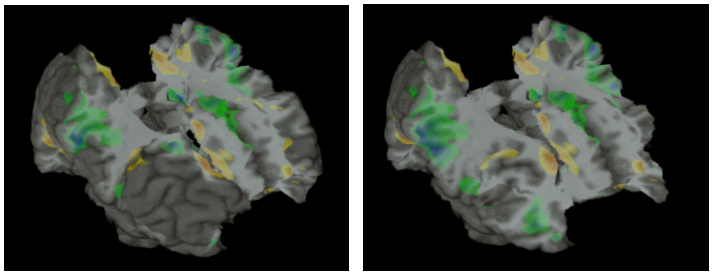
To enhance the exploration capabilities of the user, we provide two clipping modes to avoid the rendering of non interesting features. The basic clipping planes offered by OpenGL are used to implement simple clipping of the geometry (slices) as seen in Figure 7. This clipping mode is fast and useful for simple exploration tasks where interactivity is the key issue.



(a) (b)

Figure 7: Use of OpenGL clipping planes to explore the data. Our overlapping scheme allows us to see the correct depth sorting of the volumes in (a), where the slice corresponding to the clipping plane is shown as a detail. Clipping isosurfaces helps focus on important features of the data by removing occluding areas of the surface in (b).

The second clipping mode implemented is based upon the work of Weiskopf et al. [WEE02] to perform per-fragment clipping operations using an atlas of the human brain such as the Brodmann atlas [Bro09]. An example of such functionality is shown in Figure 8. This allows the independent exploration of different areas of the brain.



(a) (b)

Figure 8: Volume clipping using per-fragment operations using the Brodmann human brain atlas. 24 out of 48 areas are shown in (a), whilst 19 are shown in (b).

5.3 Special Analysis Tasks

To offer further visual aids during the exploration we exploit the support provided by our framework for multiple renderings of a single volume (see Section 4) to offer three 2D views (frontal, temporal and occipital) and a 3D view of the activation (functional data) and the template volume. These views are independent but share common states, such as the current chosen voxel or the transfer function, by means of our framework architecture based on the observer design pattern [GHJV95]. This pattern was used to provide efficient update of render views and interacting widgets.
As the focus of the application built upon our framework was specific to functional MRI data, we built special features to manipulate this functional data. One of these features is the inclusion of a threshold/boundary-based transfer function. This transfer function allows the setting of the value and color of upper and lower boundaries both for the positive and negative activations. This information is interpolated to generate a smooth transfer function. One further specific feature, is the possibility of loading a series of functional data, to exploit the time dependency of our data, in order to gain a deeper insight into the human brain activity. Within a series, we use a single transfer function and the same states are common to the activation volumes. This means that, among other features, we are able to provide reference curves for the value of a selected voxel as well as an animation of the activation along time.

6 Results

To aid the cognitive scientist in their effort of investigating the brain activity during cognitive tasks, we built a visualization tool implemented using OpenGL as graphics API and Qt[1] for the GUI. Many parameters can be set to control the rendering results to fit the needs of the specific analysis task being performed. For instance, during the first stages of the exploration simple volume rendering with no oversampling could be used to assure maximal

[1] http://www.trolltech.com

interactivity, whilst pre-integrated volume rendering with oversampling and illumination would fit best the goals pursued during presentation of results.
These features have been successfully used to visualize data gathered from a study that has been conducted at the University of Freiburg [FKRS05]. In these experiments, the participants performed logical reasoning problems while the brain activity was measured. During the logical reasoning problem, the participants are asked to draw conclusions from given premises and later their responses are evaluated for logical validity. For instance, they saw two premises:

Premise 1 :	*V X*	*(V is on the left of X).*
Premise 2 :	*X Z*	*(X is on the left of Z).*

and they had to decide afterwards and indicate by a key press whether the following statement logically follows from the premise:

Conclusion:	*V Z*	*(V is on the left of Z)?.*

Figure 9 shows two different stages in the reasoning process (event-related design, 12 participants) for different rendering modes. In the top row it is possible to see activations in the occipito-parietal cortex and the anterior prefrontal cortex, whilst the bottom row depicts the activation during the validation in the parietal and the prefrontal cortex (more details of the study can be found in [FKRS05]).

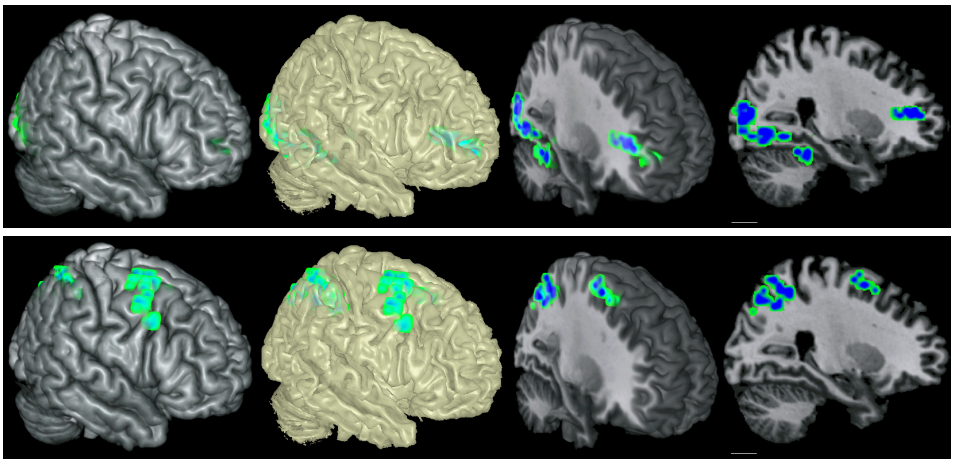


Figure 9: Two different stages in the reasoning process. From left to right: pre-integration with illumination, transparent isosurfaces, pre-integration with one clipping plane and a corresponding two-dimensional slice.

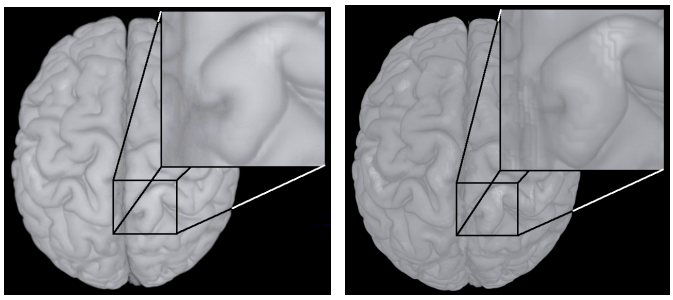
In Table 1 we show performance measurements in frames per second for the different rendering techniques we combined to visualize functional data within the context of a template brain. Our tests were performed on a standard PC equipped with an NVidia 6800 GT graphics card, using a viewport of size 510×820 with the volumes covering the entire viewport.

We carried out the measurments shown in the table using the same rendering technique for both volumes. However, as mentioned before, these render modes can be chosen independently for each volume. For instance, with isosurface rendering for the template brain and simple DVR for the map, we achieve a performance of 3.12 fps, whilst using simple DVR for the template brain and pre-integration for the map the performance obtained raises to 12.86 fps.

	Template Brain 256^3 [fps]	Functional Map 64^3 [fps]	Both [fps]
Simple DVR	27.20	153.50	16.20
Pre-integrated	15.62	94.60	9.50
Oversampling 2	7.98	47.97	4.80
Oversampling 4	4.02	24.30	2.41
Lighting	5.40	32.40	3.23
Isosurface	4.34	25.65	2.58
Volume Clipping	11.31	68.30	6.86

Table 1: Performance in frames per second for a template brain and a functional map. Results when rendering both volumes simultaneously are also shown.

Although more expensive, as shown in Figure 10 the quality improvement due to pre-integrated rendering with oversampling is noticeable. It is important to note however that, as reported by Röttger et al. [RGWE03], oversampling factors larger than 4 do not improve the quality further.



(a) (b)

Figure 10: Comparison between pre-integrated with an oversampling factor of 4 (a) and simple volume rendering (b).

7 Conclusion

We described a flexible framework for GPU-based multi-volume rendering, which served as basis to our visualization tool for functional brain imaging data. We addressed the problem of occluding volumes by offering a number of different rendering techniques and the

possibility for setting independently the visual characteristics, like transparency, of each volume in the scene.
This tool is thought to support cognitive neuroscientists in experimental studies. It facilitates the early phases of the research as it helps to get a quick glance of what might be in the data that is worth to be analyzed more carefully. Another advantage is that the software is very helpful because non-experts often have serious problems to understand results of fMRI experiments. With our methods the presentation is much more intuitive and thus also supports the knowledge transfer within the scientific community and with the public.
One important feature of our application is the support for using different atlases of the human brain to perform volume clipping. This, together with the automatic transformation between different normalized coordinates system, will provide the user with the necessary tools to work with various data sources.
At present, we implement a correct blending of the slices by means of a temporary buffer, in order to compare it with the simple direct blending to the framebuffer we currently use. Furthemore we want to test if we could avoid the state changes between the rendering of slices of different volumes without the loss of the high flexibility of our framework and by this gain any performance improvements.
Since interactivity is a key issue when exploring functional MRI data, the automatic setting of the render mode (pre-integration, oversampling, etc.), to better fit the platform on which the application is being used, is a necessary feature to be included in our application. Furthemore it should be tested if the expensive state changes between the rendering of slices of different volumes could be avoided without any loss of the high flexibility of our framework.
Finally, it is important to note, that our framework is well suited to the implementation of a wide range of visualization techniques and to the use in diverse application areas. Specially, in medical imaging where data is often obtained from different modalities, the integrated visualization of multiple volumes offers useful new insights.

References

[Bro09] Korbinian Brodmann. *Vergleichende Lokalisationslehre der Grosshirnrinde in ihren Prinzipien dargestellt auf Grund des Zellenbaues*. Leipzig, . Translated by Laurence Garey as Localisation in the Cerebral Cortex (1994), London: Smith-Gordon, new edition 1999, London: Imperial College Press. edition, 1909.

[CCF94] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization*, pages 91–98, New York, NY, USA, 1994. ACM Press.

[CS99] Wenli Cai and Georgios Sakas. Data intermixing and multi-volume rendering. *Comput. Graph. Forum*, 18(3):359–368, 1999.

[EHK+04] Klaus Engel, Markus Hadwiger, Joe Kniss, Aaron Lefohn, Christof Rezk-Salama, and Daniel Weiskopf. Real-time volume graphics. course notes for course #28 at siggraph 2004, 2004.

[EKE01] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '01*, pages 9–16, 2001.

[FKRS05] Thomas Fangmeier, Markus Knauff, Christian C. Ruff, and Vladimir Sloutsky. fMRI evidence for a three-stage model of deductive reasoning. *Journal of Cognitive Neuroscience (in press)*, 2005.

[FR85] Peter T. Fox and Marcus E. Raichle. Stimulus rate determines regional brain blood flow in striate cortex. *Annals of Neurology*, 17:303–305, 1985.

[GBKG04] Sören Grimm, Stefan Bruckner, Armin Kanitsar, and Eduard Gröller. Flexible direct multi-volume rendering in interactive scenes. In *Vision, Modeling, and Visualization*, pages 386–379, 2004.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlisside. *Design Patterns: elements of resusable object-oriented software*. Addison-Wesley, 1995.

[KK99] Kevin Kreeger and Arie Kaufman. Mixing translucent polygons with volumes. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 191–198, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[LC98] Adrian Leu and Min Chen. Direct rendering algorithms for complex volumetric scenes. In *Proceedings of the 16th Eurographics UK Conference*, pages 1–15, 1998.

[LC99] Adrian Leu and Min Chen. Modeling and rendering graphics scenes composed of multiple volumetric datasets. *Computer Graphics Forum*, 18(2):159–171, 1999.

[OLKT90] S. Ogawa, T.M Lee, A. R. Kay, and D. W. Tank. Brain magnetic resonance imaging with contrast dependent on blood oxygenation. In *Proceedings of the National Academy of Sciences*, pages 9868–9872, 1990.

[PSHOP00] Bernhard Preim, Wolf Spindler, and Heinz-Otto-Peitgen. Interaktive medizinische volumenvisualisierung - ein überblick'. In *Simulation und Visualisierung*, pages 69–88. SCS-Verlag, 2000.

[Rai01] Marcus E. Raichle. *Handbook of Functional Neuroimaging of Cognition*, chapter Functional neuroimaging: A historical and physiological perspective, pages 3–26. MIT Press, 2001.

[RGWE03] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, and Thomas Ertl. Smart Hardware-Accelerated Volume Rendering. In *Procceedings of EG/IEEE TCVG Symposium on Visualization VisSym '03*, pages 231–238, 2003.

[Ror05] Chris Rorden. MRIcro. http://www.sph.sc.edu/comd/rorden/mricro.html, 2005.

[Sie05] Siemens. Medical Solutions. Syngo 3D Offline fMRI. http://www.medical.siemens.com, 2005.

[Sin05] Krish Singh. MRI3DX. http://www.aston.ac.uk/lhs/staff/singhkd/mri3dX/, 2005.

[SPM05] SPM. Statistical Parametric Mapping . http://www.fil.ion.ucl.ac.uk/spm/, 2005.

[SZV01] Rik Stokking, Karel Zuiderveld, and Max Viewgever. Integrated volume visualization of functional image data and anatomical surfaces using normal fusion. *Human Brain Mapping*, 12(4):203–218, 2001.

[WE98] Rüdiger Westermann and Thomas Ertl. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 169–177, New York, NY, USA, 1998. ACM Press.

[WEE02] Daniel Weiskopf, Klaus Engel, and Thomas Ertl. Volume Clipping via Per-Fragment Operations in Texture-Based Volume Visualization. In *Procceedings of IEEE Visualization '02*, pages 93–100, 2002.