Int J CARS
DOI 10.1007/s11548-014-1069-x

ORIGINAL ARTICLE

GPU-based multi-volume ray casting within VTK for medical applications

Mohammadmehdi Bozorgi · Frank Lindseth

Received: 10 January 2014 / Accepted: 28 April 2014
© CARS 2014

Abstract
Purpose Multi-volume visualization is important for displaying relevant information in multimodal or multitemporal medical imaging studies. The main objective with the current study was to develop an efficient GPU-based multi-volume ray caster (MVRC) and validate the proposed visualization system in the context of image-guided surgical navigation.
Methods Ray casting can produce high-quality 2D images from 3D volume data but the method is computationally demanding, especially when multiple volumes are involved, so a parallel GPU version has been implemented. In the proposed MVRC, imaginary rays are sent through the volumes (one ray for each pixel in the view), and at equal and short intervals along the rays, samples are collected from each volume. Samples from all the volumes are composited using front to back α-blending. Since all the rays can be processed simultaneously, the MVRC was implemented in parallel on the GPU to achieve acceptable interactive frame rates. The method is fully integrated within the visualization toolkit (VTK) pipeline with the ability to apply different operations (e.g., transformations, clipping, and cropping) on each volume separately. The implemented method is cross-platform (Windows, Linux and Mac OSX) and runs on different graphics card (NVidia and AMD). The speed of the MVRC was tested with one to five volumes of varying sizes: 128^3, 256^3, and 512^3. A Tesla C2070 GPU was used, and the output image size was 600×600 pixels. The original VTK single-volume ray caster and the MVRC were compared when rendering only one volume.
Results The multi-volume rendering system achieved an interactive frame rate (>15 fps) when rendering five small volumes (128^3 voxels), four medium-sized volumes (256^3 voxels), and two large volumes (512^3 voxels). When rendering single volumes, the frame rate of the MVRC was comparable to the original VTK ray caster for small and medium-sized datasets but was approximately 3 frames per second slower for large datasets. The MVRC was successfully integrated in an existing surgical navigation system and was shown to be clinically useful during an ultrasound-guided neurosurgical tumor resection.
Conclusions A GPU-based MVRC for VTK is a useful tool in medical visualization. The proposed multi-volume GPU-based ray caster for VTK provided high-quality images at reasonable frame rates. The MVRC was effective when used in a neurosurgical navigation application.

Keywords Multi-volume ray casting · Visualization toolkit · GPU · Volume rendering

M. Bozorgi (✉) · F. Lindseth
Department of Computer and Information Science, Norwegian University of Science and Technology, Sem Saelandsvei 7-9, 7491 Trondheim, Norway
e-mail: mohammeb@idi.ntnu.no

F. Lindseth
SINTEF Medical Technology, Trondheim, Norway

Introduction

Medical image analysis and visualization have been an important part of modern medicine for many years. Advanced imaging and processing methods assist medical personal in interpreting the available data and investigate the inner parts of the human body. CT, MRI, and ultrasound scanners are a few examples of well-known modalities in the medical field that can generate 3D scalar data (volumes) of human anatomy and pathology [1]. To directly extract a comprehensive 2D image from the produced 3D volumes, several visualization techniques commonly known as *volume rendering* have been proposed; Ray casting, texture slicing, shear–warp, splatting, and cell projection are some examples of different volume rendering techniques. Each volume rendering technique has its own advantages and disadvantages; however, ray casting is the most widely used volume rendering technique for medical purposes today. Ray casting follows a few well-defined stages and can produce high-quality images. However, the method is computationally expensive.

CPU-based ray casting has been used for almost three decades in numerous applications, especially in the medical domain [1]. Nevertheless, GPU-based ray casting was introduced only few years ago to increase the speed of this approach by exploiting the parallel nature of the ray casting algorithm as well as taking advantage of the programmability of modern GPUs, also known as GPGPU.

Advances in both hardware and software made also *multi-volume rendering* possible. Multi-volume rendering refers to a volume rendering technique that creates a 2D image using several volumes that have overlapping regions. Using multiple volumes aids the analysis of the region of interest imaged and decreases the chance of misinterpretation by medical doctors.

Visualization of multiple volumes with overlapping regions can be useful in many scientific applications, especially in the medical domain. Subsequently, as the demand and need for multi-volume rendering has grown, various methods have been proposed. For instance, Zuiderveld and Viergever [2] developed an optimized and flexible framework for multi-volume rendering so that the different visualization methods can be selected and applied on different pre-segmented volumes (objects).

Later, Jacq and Roux [3,4] presented a more complete model based on Density Emitter Radiation Transport for direct multimodal volume rendering. In this method, the final image was made based on the intermixing of different volumes during the ray casting pipeline. The proposed multimodal volume rendering was used to evaluate 3D image registration. As the intermixing of volumes can happen during different stages in multi-volume rendering, Cai and Sakas [5] precisely explained three different strategies for volume intermixing and then proposed a basic approach for multi-volume rendering.

To increase the speed of CPU-based multi-volume rendering, a GPU-based implementation was presented by Hadwiger et al. [6]. The input volumes used in their method were a set of pre-segmented data sets that allowed the method to distingue between overlapping areas and accepting different transfer function for the different volumes.

Krissian and Falcón-Torres [7] proposed a more practical GPU-based two-volume ray caster which is implemented inside the visualization toolkit (VTK), and it follows, although not complete, the VTK pipeline.

Visualization toolkit is a widely used library, supporting many visualization and image processing methods. The library is open source and cross-platform which makes this library very popular for both open source and commercial visualization applications. Both CPU- and GPU- based ray casting are implemented within VTK. Although VTK has support for ray casting, it lacks support of multi-volume rendering.

In this paper, a general GPU-based multi-volume ray casting technique integrated in VTK is proposed. The method is based on the work of Krissian and Falcón-Torres [7] and follows the VTK pipeline. Additionally, the ability to apply different operations (e.g., transformations, transfer functions, clipping, and cropping) on each volume separately is also added to the method.

The proposed method is explained in the methods section, and both performance and functionality are evaluated in the result section. At the end, the multi-volume renderer is discussed, and some concluding remarks are given.

Methods

Ray casting is capable of producing high-quality images at acceptable frame rates. In addition, the method is parallel in nature and has the potential to be optimized for the GPU. For these reasons, the method proposed in this paper is based on ray casting. Furthermore, VTK is one of the most widely used visualization toolkits and hence chosen as the target platform for implementing the multi-volume ray caster (MVRC). In this section, first, the proposed method is described, and subsequently, a brief description of the VTK implementation is given.

Multi-volume rendering basics

Ray casting, like most of other volume rendering methods, is based on the *rendering equation* [8,9]:

$$\begin{aligned} I(x, x') \\ = g(x, x')\left[\epsilon(x, x') + \int_S \rho(x, x', x'')I(x', x'')\mathrm{d}x''\right] \end{aligned}$$

The rendering equation tries to model the physical phenomenon of light emission and scattering for realistic rendering in computer graphics. The rendering equation basically demonstrates that the transported intensity of light (I) from point x' to x can be calculated by summing the emitted light (ϵ) from point x' to x and the total light intensity that is scattered (ρ) toward the point x from all other surfaces (x'') reflected at point x'. The term g is a geometry term which defines the mutual visibility between point x and x'.

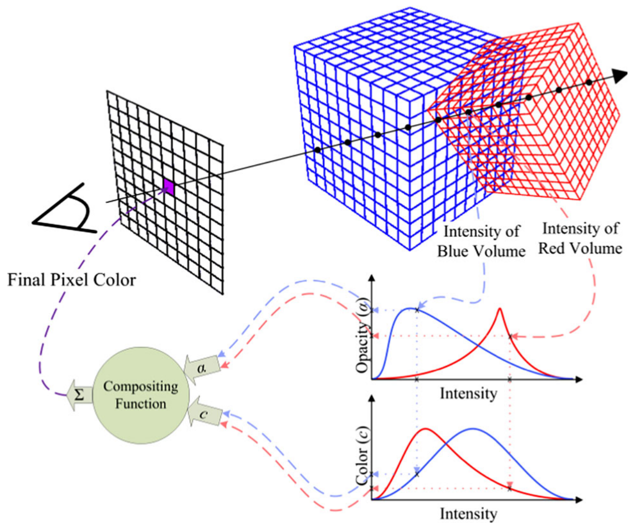

Fig. 1 An overview of the multi-volume ray casting method

A volume rendering integral is derived from the rendering equation which emphasis and states the relation between the volume intensities at point t along the ray, the color (C) and opacity (α) transfer functions, and the final image plane intensity (I) [10]:

$$I = \int_{t_1}^{t_2} C(t) e^{-\int_{t_1}^{t} \alpha(s)\mathrm{d}s} \mathrm{d}t$$

$C(t)$ contains all the necessary light information, including emitted, scattered, and reflected light [10]. The volume rendering integral must be solved with a good approximation of the integrals and be solvable within the computational constraints. Applying the zero-order quadrature for approximating the inner integral is the most common approach, and a first-order approximation is usually used for the exponential. The outer integral is mostly solved by a finite sum of volume samples. Therefore, the volume rendering integral can be reformulated as below:

$$I = \sum_{k=1}^{M} C_k \alpha_k \prod_{i=1}^{k-1} (1 - \alpha_i)$$

where C_k and α_k are the color and opacity samples collected along the ray. This is the equation implemented for front to back ray casting and has been expanded to achieve the proposed method for multi-volume rendering.

Figure 1 shows the entire process of multi-volume ray casting. The process starts by casting an imaginary ray in the view direction for each pixel in the image plan. Next, the samples are collected along the ray at small and equivalent intervals. Then, the collected samples (intensity values) are converted into color and opacity using transfer functions. In multi-volume rendering, during ray traversal, a sampling position might lie in more than one volume; so, the sampling is done for all the volumes at the sampling position, and each of the samples follow the transfer function that belongs to the volume that the sample was taken from. Finally, all the converted samples from all the volumes are accumulated together based on the composition function in order to generate the final pixel value in the image (color and opacity). Consequently, the original ray casting equation is generalized for multi-volume rendering as shown below:

$$I = \sum_{k=1}^{M} \sum_{v=1}^{N} \omega(k, v) \cdot C_k^v \alpha_k^v \prod_{i=1}^{k-1} (1 - \alpha_i^v).$$

$$\omega(k, v) = \begin{cases} 1 \text{ if sample } \boldsymbol{k} \text{ is } \textbf{\textit{inside}} \text{ volume } \boldsymbol{v} \\ 0 \text{ if sample } \boldsymbol{k} \text{ is } \textbf{\textit{outside}} \text{ volume } \boldsymbol{v} \end{cases}$$

where the index v runs over all the N volumes that are rendered, and $\omega(k, v)$ is the member function indicating that a sample will participate in the calculation if the sample k is located inside volume v.

Independent transformations for each volume

One of the features that the proposed MVRC offers is the support for independent transformations (translation, rotation, and scaling) for each volume that are rendered. To achieve this, we need to calculate the texture coordinate matrix ($\mathrm{TC}_{1\mathrm{to}V}$) for each volume:

$$\mathrm{TC}_{1\mathrm{to}V} = M_V.UT_V^{-1}.M_1^{-1}$$

where UT_V is a user-defined transformation matrix for the Vth volume, and M_1 and M_V are the world to texture coordinate matrices of the first and Vth volumes, respectively. As Fig. 2 demonstrates, an initial ray generates and casts toward the first volume based on the viewport and viewer position. Afterward, an imaginary (inverse) transformed ray is created from the initial ray for each additional volume in the rendering scene.

Independent clipping for each volume

Clipping is an important feature in volume rendering that can clip out unnecessary parts of the volume based on user-defined clipping planes. In single-volume rendering, if the sampling position goes beyond the clipping plane, the ray traversal can be immediately stopped. However, in the proposed multi-volume renderer, a single clipping plane is individually associated with each volume. As a result, the ray might need to continue beyond the encountered clipping plane in order to draw the rest of volumes (see Fig. 3).

The independent clipping functionality is achieved by a ray (line) to plane intersection test when clipping is requested and a clipping plane is defined by the user. If an intersection occurs, in each sampling step, the distance between the intersection point and the sampling point is tested, if the distance

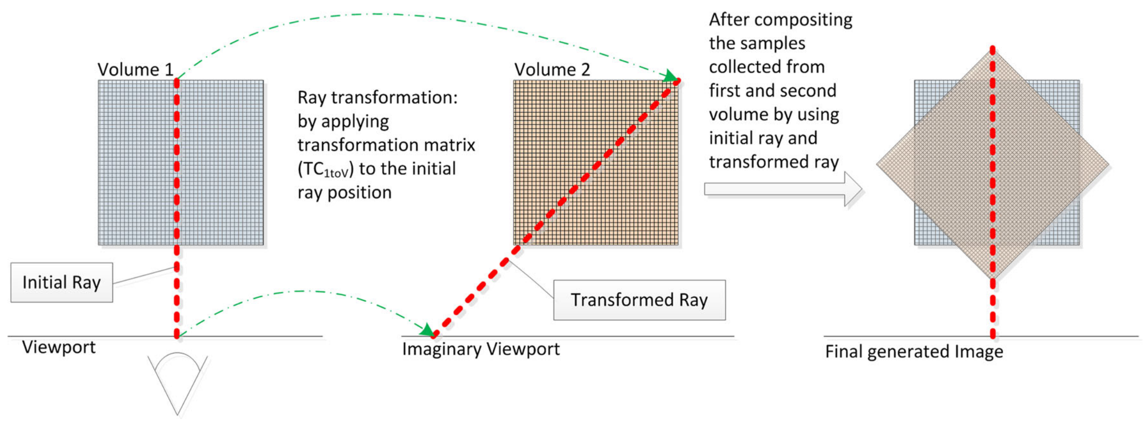
Volume 1
Volume 2
Ray transformation: by applying transformation matrix (TC_{1toV}) to the initial ray position
After compositing the samples collected from first and second volume by using initial ray and transformed ray
Initial Ray
Transformed Ray
Viewport
Imaginary Viewport
Final generated Image

Fig. 2 Ray transformation by applying the ray transformation matrix to the initial ray

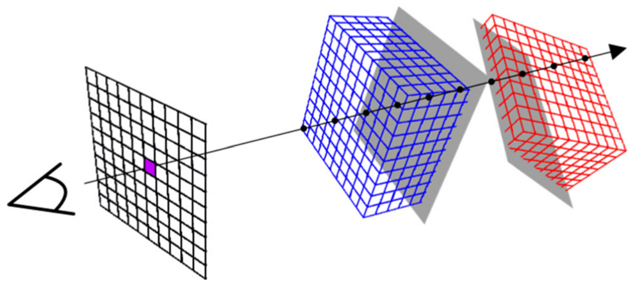

Fig. 3 Clipping in the proposed multi-volume renderer

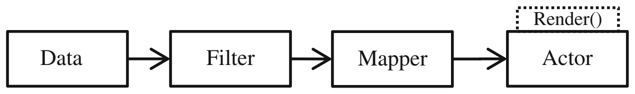

Fig. 4 A generic overview of the VTK pipeline

has a negative value, the sampling point will be discarded from the volume of interest.

Integration into the visualization toolkit

[1]The VTK architecture is briefly described in this section to help recognizing the VTK classes contributing to volume rendering and consequently to ray casting. VTK uses a data flow concept for visualization of 3D scalar data. The general overview of the VTK pipeline is illustrated in Fig. 4. As can be seen, the data are read by VTK, and then, it can be transformed using a VTK filter; filtered data are mapped to a visual representation via a mapper. Finally, the data can be rendered using an actor [11,12].

The class diagram for the proposed multi-volume renderer is illustrated in Fig. 5. *vtkGPUMultiVolumeRayCastMapper* and *vtkOpenGLGPUMultiVolumeRayCastMapper* are the two main classes where most of the code for integrat-

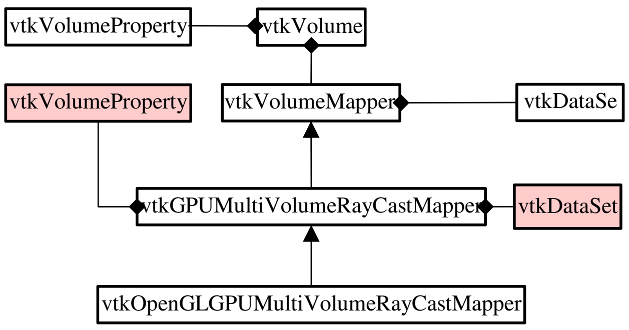

Fig. 5 The class diagram of the proposed multi-volume rendering within VTK

ing the new method into VTK is developed. These classes use OpenGL Shading Language (GLSL) [13] for running the computationally expensive parts of the code on the GPU. The class diagram is very similar to the original VTK single-volume ray caster, the only difference being that the *vtkGPUMultiVolumeRayCastMapper* needs direct access to *vtkVolumeProperty* and *vtkDataSet* for the additional volumes provided.

Results

Performance

This section depicts the performance of the proposed VTK-based MVRC. The machine that was used for all experiments was equipped by an NVIDIA Tesla C2070 Graphic Card and an Intel Core i7-3770 @ 3.40 GHz Processor. All the rendering experiments were performed using the same window size (600×600 pixels).

To assess the performance of the proposed work, first the speed of the original VTK volume ray caster was tested by rendering a single volume of the head (see Fig. 6) resampled to different sizes (128^3, 256^3, and 512^3). Afterward, the pro-

[1] The source code is available online at http://github.com/mohammeb/VTKMultiVolumeRayCaster/.

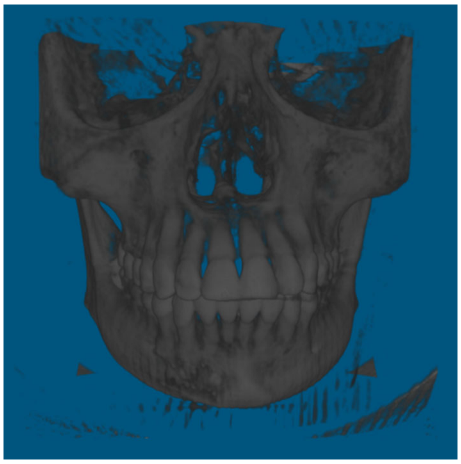

Fig. 6 The head volume dataset that was used for performance testing

Table 1 Performance of proposed multi-volume ray casting

	128^3 (fps)	256^3 (fps)	512^3 (fps)
Single volume *with original VTK*	60.4	59.2	30.1
Single volume *with proposed multi-volume*	60.3	58.4	27.1
2 Volumes	58.4	30.1	14.5
3 Volumes	30.1	20.0	9.5
4 Volumes	28.8	15.1	7.0
5 Volumes	20.5	12.1	5.5

posed VTK- based MVRC was tested using the same volume sizes. As it is illustrated in Table 1, performance when rendering small and medium-sized volumes remained almost identical for the original and the proposed ray casting methods. Furthermore, the performance dropped only by 3 frames per second for the large volume (512^3), from 30.1 frames per second in the original VTK ray caster to 27.1 frames per second for the proposed multi-volume ray caster.

One of the most important issues affecting the performance of the proposed MVRC (even for rendering a single volume) is the added branching conditions and loops in the GLSL shader codes. Having branches and loops in the GPU code is relatively costly and has high impact on performance [14].

The performance of the MVRC was also tested for rendering two to five volumes of the same sizes (128^3, 256^3, and 512^3). Table 1 shows all the accomplished results from the testing process. The results indicate that an interactive frame rate (>15 fps) can be achieved for up to five small volumes (size of 128^3), four medium-sized volumes (size of 256^3), or two large volumes (size of 512^3).

Functionality

To evaluate the functionality offered by the new MVRC and compare it to existing methods for volume rendering in a practical setting, the proposed method was integrated in an in-house research system for ultrasound-based image-guided surgery called CustusX [15] and tested during a neurosurgical tumor resection (see Fig. 7). As the new mapper behaves like an ordinary VTK mapper, the integration process was relatively straightforward, the challenge being to refactor the system to take advantage of the new multimodal functionality (one mapper showing multiple volumes with individual control of the various rendering properties instead of multiple mappers showing a single volume). CustusX offers various visualization techniques like slicing (presented in both 2D and 3D views) and surface rendering of structures extracted from the available volumes, in addition to the volume rendering methods made available in VTK, e.g., GPU-based single-volume ray casting and 3D texture-based methods. The system is built for multimodal visualization, and the idea is to use preoperative CT and MR in combination with intraoperative ultrasound. Preoperative data are mainly used for planning in the start of the procedure; intraoperative ultrasound can be used as an updated navigation map during the procedure as well as for resection control (verify that all tumor tissues are removed) toward the end of the operation. During the procedure, preoperative MR (CT) can provide an overview of the surgical field and ease the interpretation of the ultrasound images. Outdated preoperative data cannot be used for navigation as the correspondence to the patient will gradually decrease due to surgical manipulation and resection. However, ultrasound data can be used to shift correct the preoperative data (move into the correct position) greatly improving its usability during the resection. That way, important gray (fMRI) and white matter (DTI) structures can be avoided during a neurosurgical procedure for example.

Figure 7 shows two views (different rows) of the neurosurgical case where each of the six 3D scenes contains the same structures. The only difference is that the volume rendering method used has been changed between the three columns. In the first column, the proposed multi-volume ray caster has been used to render the three partly overlapping volumes at the same time, and we see that the rendering quality is good and that the depth ordering of the structures is correct. In the middle column, multiple single-volume ray casters have been used (one for each volume), and we see that the apparent order of the structures is wrong; the green tumor seems to be in front of the white matter tracts for example. In fact, the order at which the volumes are added to the 3D scene matters as each renderer generates a separate image that is combined and shown to the surgeon. The volume added last appears to be on top when there is overlap between the volumes relative to the viewing direction. In a clinical setting, loosing

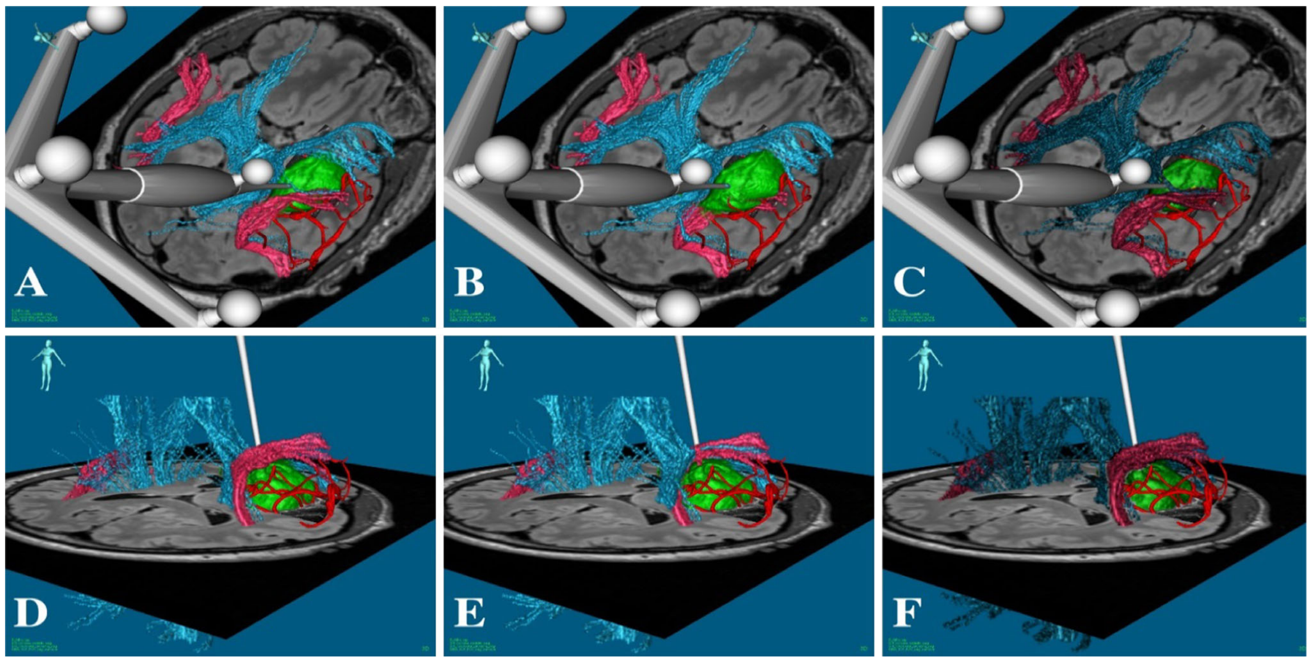

Fig. 7 Multi-volume ray casting versus single-volume and texture-based volume rendering (column one, two, and three, respectively). The images contain the following structures: MR and ultrasound slices through the tumor overlaid each other and show in *gray*. Volume rendered MR tumor in *green* and important *white matter* tracts in *light blue* (radiata) and *pink* (arcuata). In addition, surface-rendered MRAngio vessels are shown in *red*, and the tracked surgical pointer is *gray*. The multi-volume ray caster presents the volume rendered structures correctly in terms of which object is in front, the single-volume ray caster shows the last volume added to the scene on top making interpretation of the scene very challenging. The texture-based volume renderer presents the data semi-correct but with reduced image quality. The *top and bottom rows* show two difference views of the navigation scene

the depth information like this can of course have dramatic consequences. In addition, texture-based volume rendering is shown in the last column, and we observe reduced image quality relative to ray casting. On the other hand, the ordering of the structures is for the most part correct.

As previously stated preoperative MR can be shift corrected using intraoperative ultrasound, this is done in Fig. 8 where we have both ultrasounds tissue (b-mode) and vessel (Doppler) data as well as shift corrected MR tissue (Flair) and vessel (MRAngio) data. As all data are in the correct position relative to each other and to the patient, the effect of the order at which the spatially overlapping volumes are added to the scene becomes very apparent. In the first column, the MVRC is used and the views are correct. In the last two columns, the single-volume renderer is used and different volumes are added to the scene last. As can be seen, the visualizations are not correct any more.

Clipping is another useful feature in medical imaging as it can exclude parts that are not interesting making potentially interesting parts more visible. Figure 9 illustrates this feature in the proposed MVRC. As Fig. 9 shows, a clipping plane is assigned to a MR volume of the head while other volumes allocated to the MVRC remain unclipped (tumor in green and white matter tracts in light blue). In Fig. 9, the clipping plane is defined by the pointing instrument and by moving the pointer the clipping plane is changing and applied to the chosen volume (MR volume of the head).

Discussion

In this paper, we have presented a multi-volume rendering method. The multi-volume renderer is successfully implemented using ray casting. The method is fully integrated with the VTK pipeline, and users can feed the new mapper with multiple volumes regardless of their modality, size, spacing, and voxel type, with the freedom of applying different opacity and color transfer function for each volume. Furthermore, common features such as transformation (translation, scaling, and rotation), clipping, and cropping can be independently and interactively applied to any of the volumes. Experiments confirmed an acceptable frame rate for real-time rendering of multiple volumes, and the new mapper was relatively straightforward to integrate in an existing image-guided surgery system.

As part of VTK, an open source GPU-based mapper for ray casting is available; however, it only renders one volume

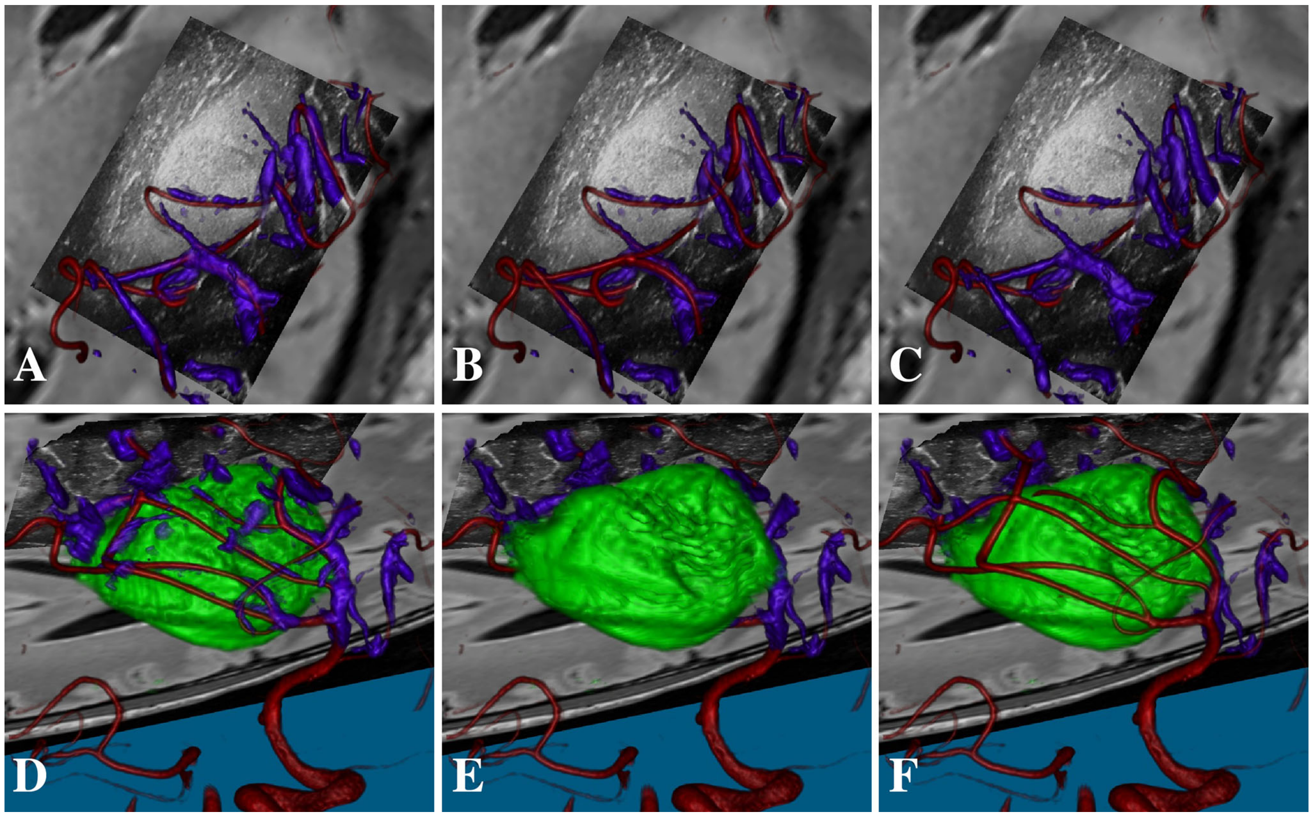

Fig. 8 Multi-volume versus single-volume ray casting. The images contain the following structures: MR and ultrasound slices through the tumor overlaid each other and show in *gray*. Volume rendered vessels (MRAngio in *red* and ultrasound Doppler in *blue*) and MR tumor (in *green*). In the first column, the overlapping volumes are volume rendered using the MVRC, and the data are presented correctly in terms of what parts are in front, independently of the order in which the volumes were added to the scene. The single-volume ray caster is used in the following two columns, in the middle, the *red* MRAngio vessels (**b**) and the *green* MR tumor (**e**) have been added to the scene last and is wrongly placed in front of the other volumes. In **c** and **f**, the *blue* ultrasound Doppler vessels and the *red* MRAngio vessels have been added last respectively, and we see the same effect making the interpretation of the scene very difficult. The *top and bottom rows* show two difference views of the navigation scene

per mapper. Therefore, if it is desirable to render more than one volume, one mapper per volume has to be initialized and render each volume independently and at the end combine the generated images. Loosing depth information is the main side effect of this process; this can cause one volume to always be on top of the other volumes (see Figs. 7, 8). The consequence is that the viewer can easily misinterpret the generated scene.

There are many practical applications for multi-volume rendering, especially in a clinical setting. One of the most useful applications of ray casting is in image-guided surgery (IGS). IGS system allows surgeons to visualize preoperative (e.g., MRI, CT) and intraoperative images during surgery and interactively track the surgical instruments used by employing a tracking system (e.g., optical or magnetic). To increase the practicality of the IGS system and giving more detail to the surgeon, it is common to visualize several volumes from the region of the interest.

Surface rendering is an alternative to multi-volume rendering with clipping, i.e., relevant structures are extracted from the available volumes and the generated surface models are rendered together. The clinical evaluation of the proposed method showed quite clearly the benefits with multi-volume rendering; it is faster and closer to the raw data, meaning that 3D visualizations can be shown with a minimum of preparations and that the surgeon is more responsible and aware of the interpretation involved (i.e., making decisions based on tumor borders that are often semi-automatically segmented by a technician).

It can be concluded that the proposed GPU-based MVRC, which is developed for VTK, is a useful tool for medical visualization. It works under different platforms (Windows, Linux, and Mac OS) and graphics card (NVidia and AMD). The proposed method revealed reasonable performance and produced correct high-quality images. Lastly, the new multi-volume renderer was relatively easy to integrate in an excis-

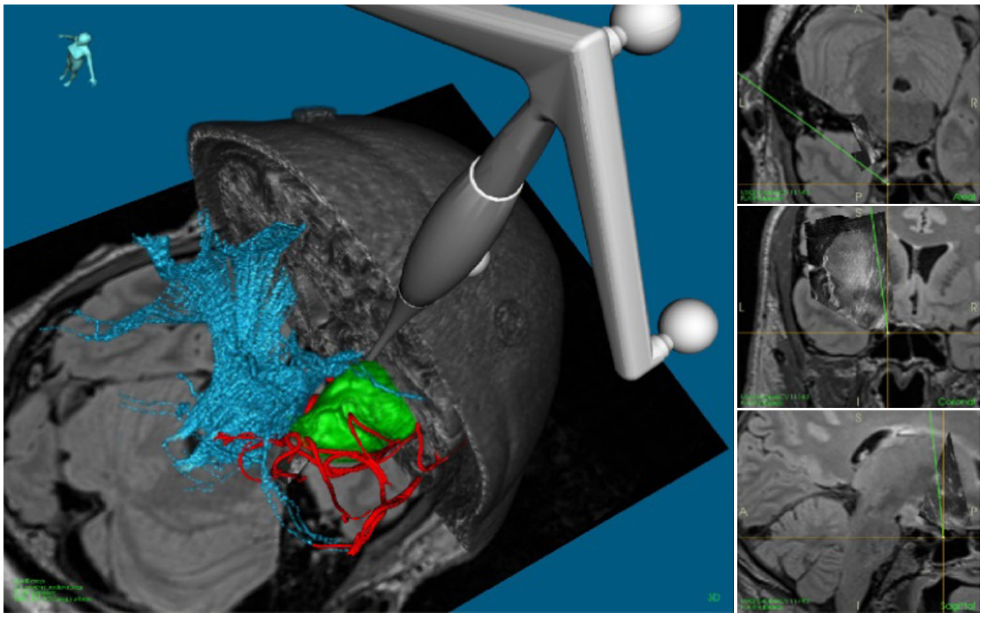

Fig. 9 Clipping and 2D slice views. The 3D view contains the following structures: A sliced MR dataset in *gray*, volume rendered MR tumor in *green*, important *white matter* tracts in *light blue*, and a tracked pointer clipped MR dataset in *gray*. In addition, surface-rendered MRAngio vessels are shown in *red,* and the tracked surgical pointer is shown in *gray*. Notice that the multi-volume ray caster is able to clip each of the volumes independently, i.e., just the complete MR volume of the head is sliced not the *green* tumor that the pointer slices through. The column to the *right* shows three orthogonal 2D slices extracted from both MR and ultrasound using the tip of the tracked pointer

ing navigation application and was found to be clinically useful during a neurosurgical tumor resection.

Conflict of interest Mohammadmehdi Bozorgi and Frank Lindseth declare that they have no conflict of interest.

References

1. Zhang Q, Eagleson R, Peters TM (2011) Volume visualization: a technical overview with a focus on medical applications. J Digit Imaging 24(4):640–664
2. Zuiderveld KJ, Viergever MA (1994) Multi-modal volume visualization using object-oriented methods. In: Proceedings of the 1994 symposium on, volume visualization 1994, ACM: Tysons Corner, Virginia, USA, pp 59–66
3. Jacq JJ, Roux C (1996) A direct multi-volume rendering method. Application to visual assessment of 3-D image registration algorithms. Vis Biomed Comput 1131:53–62
4. Jacq JJ, Roux CJ (1997) A direct multi-volume rendering method aiming at comparisons of 3-D images and models. IEEE Trans Inf Technol Biomed 1(1):30–43
5. Cai WL, Sakas G (1999) Data intermixing and multi-volume rendering. Comput Graph Forum 18(3):359–368
6. Hadwiger M, Berger C, Hauser H (2003) High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In: Proceedings of the IEEE Visualization 2003, pp 301–308
7. Krissian K, Falcón-Torres C (2012) GPU volume ray casting of two volumes within VTK. VTK J
8. Kajiya JT (1986) The rendering equation. SIGGRAPH Comput Graph 20(4):143–150
9. Immel DS, Cohen MF, Greenberg DP (1986) A radiosity method for non-diffuse environments. SIGGRAPH Comput Graph 20(4):133–142
10. Gross MH et al (1995) A new method to approximate the volume-rendering equation using wavelet bases and piecewise polynomials. Comput Graph 19(1):47–62
11. Caban JJ, Joshi A, Nagy P (2007) Rapid development of medical imaging tools with open-source libraries. J Digit Imaging 20(Suppl 1):83–93
12. Schroeder WJ, Avila LS, Hoffman W (2000) Visualizing with VTK: a tutorial. IEEE Comput Graph Appl 20(5):20–27
13. Kessenich J, Baldwin D, Rost R (2004) The OpenGL shading language (language version 1.10)
14. Owens JD et al (2008) GPU computing. Proc IEEE 96(5):879–899
15. Tangen GA et al (2005) CustusX—a platform for navigated simulation and improved computer-assisted education. CARS 2005 Comput Assist Radiol Surg 1281:1374–1374