Stefan Diepenbrock

Rapid Development of Applications for the Interactive Visual Analysis of Multimodal Medical Data

Münster • 2013

Westfälische Wilhelms-Universität Münster

Informatik

Rapid Development of Applications for the Interactive Visual Analysis of Multimodal Medical Data

Inaugural-Dissertation

zur Erlangung des Doktorgrades der Naturwissenschaften
im Fachbereich Mathematik und Informatik
der Mathematisch-Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms-Universität Münster

vorgelegt von

Stefan Diepenbrock
aus Münster

2013

Dekan:	Prof. Dr. Martin Stein
Erster Gutacher:	Prof. Dr. Klaus Hinrichs
Zweiter Gutacher:	Prof. Dr. Klaus Schäfers
Tag der mündlichen Prüfung:	5.7.2013
Tag der Promotion:	5.7.2013

Abstract

With multimodal volumetric medical data sets becoming more common due to the increasing availability of scanning hardware, software for the visualization and analysis of such data sets needs to become more efficient as well in order to prevent overloading the user with data. This dissertation presents several interactive techniques for the visual analysis of medical volume data. All applications are based on extensions to the Voreen volume rendering framework, which we will discuss first. Since visual analysis applications are interactive by definition, we propose a general-purpose navigation technique for volume data. Next, we discuss our concepts for the interactive planning of brain tumor resections. The implemented applications support a wide range of modalities and exploit novel projection techniques. Finally, we present two systems designed to work with images of vasculature. First, we discuss an interactive vessel segmentation system giving the user full control over the result while enabling an efficient, visually supported workflow. Second, we propose an application for the visual analysis of PET tracer uptake along vessels. Novel visualization techniques are placed in a side-by-side layout with standard views to allow for an efficient comparison of two vessel segments.

Contents

Preface

This dissertation represents the results of research that has been carried out from July 2009 till March 2013 at the Department of Computer Science at the University of Münster. I would like to thank my supervisor, Prof. Dr. Klaus Hinrichs, for his guidance and for giving me this interesting research opportunity in his group. Furthermore, I would like to thank Prof. Dr. Timo Ropinski for support regarding the publications which form the basis of Chapters 6, 7 and 8.

I would also like to express my thanks to my past and present colleagues from the Visualization and Computer Graphics Research Group for a very nice working environment, especially the members of the Voreen development team, including Florian Lindemann, Dr. Jörg Mensmann, Dr. Jörg-Stefan Praßni and Tobias Brix, which worked hard to make Voreen the powerful and flexible framework it is today.

Implementations of the extensions to the Voreen framework described in Chapter 4 have been partially carried out by students working on the Voreen project: Linking, animation, plotting and ITK integration have been topics of seminars by the VisCG research group. The original implementation of DTI visualization and processing techniques for the brain tumor resection planning application described in Chapter 7 have been substantially extended by Christian Schulte zu Berge in his diploma thesis.

This work was partly supported by grants from the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) through the Collaborative Research Centre 656 Molecular Cardiovascular Imaging (project Z1). I would like to thank all my collaborators from the biomedical domain, including Prof. Dr. Cornelius Faber, Prof. Dr. Hans-Werner Bothe, Prof. Dr. Klaus Schäfers, Dr. Lydia Wachsmuth, Prof. Dr. Michael Schäfers and Dr. Sven Hermann, for insights into interesting problems that motivated the work carried out in this thesis.

I would also like to express my gratitude to my family who has supported me throughout my life and during my education leading up to this thesis. Finally, I would like to thank my wife Katharina-Maria who has always encouraged and supported me.

Münster, March 2013 *Stefan Diepenbrock*

Chapter 1

Introduction

A number of volumetric imaging modalities have emerged in the last decades and have become increasingly widespread in the medical field. Each of the modalities has its strengths and weaknesses: While computed tomography scans (CT) provide high resolution images of the anatomy, no functional data is gathered. In contrast, positron emission tomography (PET) can be used to capture the metabolic activity of tissue but lacks the high resolution. Therefore, a physician has to select the modality that is best suited to answer a medical question. Alternatively, a combination of modalities may provide even more insight: A PET-CT scan can be used to detect tumors in the PET image and localize them precisely in the CT image.

Volume visualization techniques have been a very active research topic for the last decade. Due to the increase of processing power in consumer graphics hardware during this time period it is now possible to render medical volume data on standard PCs instead of specialized workstations. Larger data sets which, due to improved scanning technology are more and more common, represent not so much a performance but an *information overload problem*. Keim et al. [KAF+08] define this problem as "the danger of getting lost in data which may be

- irrelevant to the current task at hand
- processed in an inappropriate way
- presented in an inappropriate way".

The suggested solution to the information overload problem is *visual analytics*:

> Visual analytics combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning and decision making on the basis of very large and complex data sets. [KAF+08]

In this dissertation, we discuss multiple applications for the visual analysis of medical volume data as well as modifications to the Voreen framework that enabled their development.

As expressed by the definitions above, we need to know what is relevant in a given data set and how to process and present it appropriately. Visualization is a discipline that studies the efficient presentation of data, but general-purpose solutions (e.g., standard medical workstation software) may not be sufficient for all tasks. Similar to Meyer-Spradow et al. [MS09] we recognize the need for close cooperation between computer scientists and domain experts to generate meaningful visualizations. Since time for such cooperations will usually be restricted it is desirable to use *rapid prototyping* tools in order to make trial and error with different alternatives during meetings as efficient as possible. We have therefore extended the Voreen framework in a number of ways:

Multi-View applications can now be developed in the VoreenVE rapid prototyping environment. Since one visualization technique is almost never sufficient this has been an important prerequisite for the development of all applications discussed here.

Rendering of multimodal data has been implemented in a number of processors, and Voreen has been redesigned for the *handling of generic data*. Developers can therefore resort to a selection of well-tested techniques or quickly develop techniques for new types of data.

A large number of *volume processing algorithms* have been integrated into Voreen to allow on the fly changes to the visualization pipeline.

Techniques for the *quantitative analysis* of data have been added in the form of a flexible and easily extensible region of interest (ROI) framework. Accordingly, several processors for the *display of non-spatial data* have been developed.

Aside from these extensions, we propose a number of novel approaches for the interactive visual analysis of medical volume data: First, we discuss a context-aware navigation technique for volume data, which is flexible enough to be integrated into most volume visualization pipelines. Second, we describe our concepts for the interactive planning of brain tumor resections. In this application we utilize a two-step workflow to allow the surgeon to efficiently analyze a multitude of modalities. Using different projection techniques and plots, we streamline the planning process for the user. Third, we propose a sketch-based vessel segmentation system which guides the user through the segmentation process using specific techniques for visualizing uncertainty. Finally, we developed an application for the comparative visualization of PET tracer uptake in the vicinity of vessels. The side-by-side view of an artificially modified vessel and a healthy control vessel allows the user to quickly compare the PET signal of both. We propose novel visualization techniques to quickly guide the user to areas of interest.

This thesis is structured as follows: We first introduce basic volume visualization concepts and techniques in Chapter 2. After giving an initial overview of the Voreen framework in Chapter 3 we will discuss the aforementioned extensions in Chapter 4. Before discussing our novel approaches, we will shortly discuss in Chapter 5 an application for the visual analysis of diffusion tensor imaging (DTI) data as an example for the rapid-prototyping capabilities provided by the Voreen framework. In Chapter 6 we will introduce our proposed volume navigation technique, followed by a presentation of our brain tumor resection planning-tool (Chapter 7). In Chapter 8 our novel vessel segmentation system will be described, which can also be used as a pre-processing step of an application for the visual analysis of tracer uptake discussed in Chapter 9. Chapter 10 concludes this thesis and provides a summary of our work as well as a discussion of future research directions.

The contributions in this thesis are based on the following publications: The application for the visual analysis of DTI data sets described in Chapter 5 has been presented as a poster at the congress of the European Society for Magnetic Resonance in Medicine and Biology (ESMRMB) [DSzBH+11]. The navigation technique described in Chapter 6 has been presented at the IEEE Pacific Visualization Symposium (PacificVis) [DRH11]. The application for the pre-operative planning of brain tumor resections discussed in Chapter 7 is based on the winning entry for the IEEE Visualization Contest 2010 [DPL+10] and the subsequent publication in the IEEE Computer Graphics and Applications journal [DPL+11]. The interactive vessel segmentation system discussed in Chapter 8 has been presented at the Eurographics Workshop on Visual Computing for Biology and Medicine [DR12]. The application for the comparative visualization of tracer uptake in PET/CT imaging described in Chapter 9 has been accepted at the Eurographics Conference on Visualization [DHS+13].

Additional research on volume visualization has been conducted by contributing to the work on dynamic ambient occlusion for volume rendering [RMSD+08] and texturing for volume rendering [RDB+12].

Chapter 2

Volume Visualization Techniques

This chapter introduces basic concepts of volume rendering and discusses commonly used volume visualization techniques, focussing on GPU-based approaches. Furthermore, intermixing techniques for multi-volume rendering algorithms are discussed.

Volume visualization is the process of generating an image from volumetric data. A volume is a stack of two-dimensional images, storing a scalar value at each element. The elements of a volume are called voxels (volume elements) instead of pixels and the distance (typically specified in millimeters) between voxels is called *spacing*. In contrast to the classic examination of individual slices, volume visualization techniques display the volume in its entirety [PB07].

2.1 Indirect Volume Rendering

Preim and Bartz [PB07] categorize indirect volume rendering techniques into plane-based and surface-based approaches:
Plane-based approaches display an arbitrary plane intersecting the volume. While data set-aligned planes are the most commonly used technique, planes aligned to anatomical structures can provide additional insight. For curved structures like vessels the *curved planar reformation (CPR)* has been proposed [KFW$^+$02]. The CPR renders a curved slice along the centerline of a vessel. Section 4.8 describes the implementation of the CPR in Voreen.

Surface-based approaches extract surfaces from the volume and then render the resulting geometry. Typically isosurfaces are extracted from the volume. The most well known technique to generate a geometric representation of an isosurface is the marching cubes algorithm by Lorensen and Cline [LC87]. Since surface-based rendering is not used in this thesis we will not go into further detail here.

2.2 Direct Volume Rendering

Direct volume rendering (DVR) techniques visualize the data without first generating an intermediate (polygonal) representation. We will first review the theoretical background of DVR before discussing slicing and ray casting as solutions to the volume rendering integral.

2.2.1 Theoretical Background

In this section, we will discuss the theoretical foundation of volume rendering [Eng06]. To render a volume, it is modelled as gaseous material interacting with light. The material can actively emit light, absorb light or scatter incoming light. Since no global illumination techniques are discussed in this thesis, we will not discuss models simulating scattering or shadowing. This model is called the *emission-absorption model*. Each scalar value in the volume needs to be mapped to optical properties using a *classification* method (see Secion 2.2.1). κ and q are the absorption and emission coefficients.

If we consider only a single ray, the light transport is modelled by the *volume-rendering equation* in its differential form:

$$\frac{\mathrm{d}I(s)}{\mathrm{d}s} = -\kappa(s)I(s) + q(s) \tag{2.1}$$

where s is the position along the ray. Through integration of the volume rendering equation along the ray from startpoint s_0 to endpoint D we get the *volume-rendering integral*:

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)\mathrm{d}t} + \int_{s_0}^{D} q(s) e^{-\int_{s}^{D} \kappa(t)\mathrm{d}t} \mathrm{d}s \tag{2.2}$$

I_0 is the background intensity, which is attenuated by the volume between s_0 and D. The second term describes the contribution of each point along the ray, attenuated by the volume between a point and s_0. With the transparency T between s_1 and s_2

$$T(s_1, s_2) = e^{-\int_{s_1}^{s_2} \kappa(t)\mathrm{d}t} \tag{2.3}$$

we can rewrite the volume-rendering integral as follows:

$$I(D) = I_0 T(s_0, D) + \int_{s_0}^{D} q(s) T(s, D) \mathrm{d}s \tag{2.4}$$

To solve this integral it is split into n intervals $[s_{i-1}, s_i]$ (with $0 < i \leq n$ and $s_n = D$).

The radiance at s_i is therefore:

$$I(s_i) = I(s_{i-1})\,\underbrace{T(s_{i-1}, s_i)}_{T_i} + \underbrace{\int_{s_{i-1}}^{s_i} q(s)T(s, s_i)\mathrm{d}s}_{c_i} \tag{2.5}$$

T_i and c_i are the transparency and color of segment i, they are commonly approximated by

$$T_i \approx e^{-\kappa(s_i)\Delta x},\ c_i \approx q(s_i)\Delta x. \tag{2.6}$$

where Δx is the lenght of the segment. We can now write $I(D)$ as

$$I(D) = \sum_{i=0}^{n} c_i \prod_{j=i+1}^{n} T_j \text{ , with } c_0 = I(s_0) \tag{2.7}$$

Compositing To solve equation 2.7 efficiently it is computed iteratively. The function that combines the current result with the next segment is called *compositing*. In the following equations, c is an RBG color and α is an alpha value ($\alpha = 1 - T$). c_{acc} and α_{acc} are the accumulated values which are updated. Compositing can be performed from front-to-back:

$$\begin{aligned} c_{acc} &= c_{acc} + (1 - \alpha_{acc})\alpha_i c_i \\ \alpha_{acc} &= \alpha_{acc} + (1 - \alpha_{acc})\alpha_i \end{aligned} \tag{2.8}$$

or back-to-front:

$$c_{acc} = (1 - \alpha_i)c_{acc} + \alpha_i c_i \tag{2.9}$$

Front-to-back compositing allows for *early ray termination*, an optimization technique that stops the calculation as soon as the accumulated alpha value is close to 1. Alternative compositing modes, which do not necessarily implement the emission-absorption model, include the commonly used *maximum intensity projection (MIP)*. The MIP simply displays the maximum intensity along each ray.

Local Illumination Illumination models have been shown to improve depth perception in images generated by direct volume rendering techniques [LR11]. Global illumination models need to simulate the interaction of all voxels in the volume and are therefore computationally expensive. As a trade-off, local illumination models are frequently used to render volumes. The Blinn-Phong shading model [Bli77] is frequently employed in computer graphics:

$$I = \underbrace{k_a i_a}_{\text{ambient term}} + \underbrace{k_d i_d (\vec{L} \cdot \vec{N})}_{\text{diffuse term}} + \underbrace{k_s i_s (\vec{H} \cdot \vec{V})^\alpha}_{\text{specular term}} \tag{2.10}$$

where k_a, k_d and k_s are the material coefficients, α is the shininess of the material, and i_a, i_d and i_s are properties of the light source. $\vec{L}$ is vector in direction of the light source, $\vec{N}$ is the surface normal, $\vec{V}$ is a vector in direction of the viewer and $\vec{H}$ is the halfway vector, determined by $\vec{V}$ and $\vec{L}$. The normal, usually given when rendering polygonal objects, is replaced by the gradient of the volume at the sampling position. Computation of the gradient is commonly performed using a gradient filter such as forward differences, central differences or the 3D Sobel filter. These filters require 3, 6, and 26 extra memory fetches and therefore have a different impact on performance as well as image quality.

Classification using Transfer Functions The assignment of optical properties to the volume is a critical problem in volume rendering. Apart from cryosection imaging (e.g., The Visible Human Project [*]), no optical properties can be directly derived from the volume data. A function assigning optical properties based on the data is called a *transfer function*. The standard approach is to assign a color and opacity based on the intensity at the current sampling position. In the graphical user interface, editing is commonly performed by editing a piecewise linear function. Figure 2.1 shows the transfer function editor implemented in Voreen and the images resulting from editing the transfer function. A number of higher-dimensional transfer functions have been proposed, the most commonly implemented one being 2D-transfer functions based on intensity and gradient magnitude [Lev88].

2.2.2 The Volume Rendering Pipeline

The volume rendering pipeline describes the typical steps found in a direct volume rendering algorithm (see Figure 2.2):

- **Traversal:** The current sampling position is updated.
- **Sampling & Interpolation:** The intensity at the sampling position is determined using an interpolation technique (usually trilinear interpolation).

[*] http://www.nlm.nih.gov/research/visible/visible_human.html

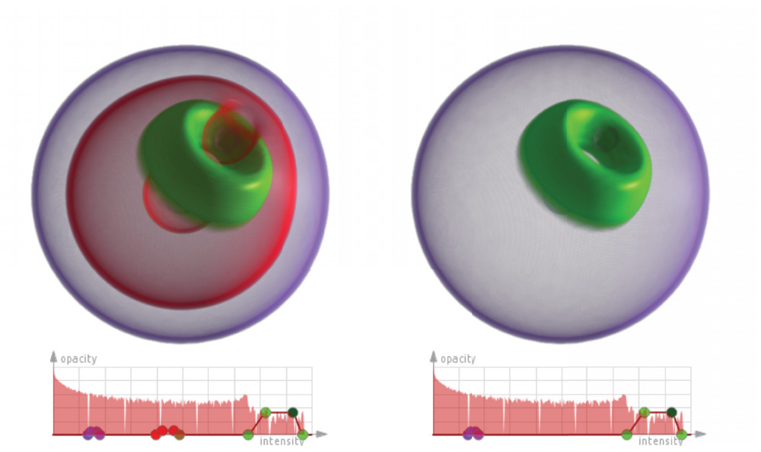


Figure 2.1: Effect of Transfer Functions on Rendering: For the right image, samples with medium intensity have been set to transparent.

- **Classification:** From the intensity the optical properties (typically an RGB-color and opacity) of the volume at the sampling position are determined, usually by application of a transfer function.
- **Illumination:** An illumination model modifies the color to simulate light interaction.
- **Accumulation:** The color is composited with the accumulated result, usually with front-to-back compositing.

The resulting pixel is then displayed on the screen.

2.2.3 Slicing

Slicing has originally been implemented for volume rendering on GPUs which did not support 3D texturing. The volume had to be converted into three stacks of 2D textures (one for each major axis). Depending on the position of the camera a slice stack is chosen and a set of data set-aligned quads are rendered in depth order (see Figure 2.3) using an appropriate blending function. Data set-aligned slices will produce artifacts for camera positions betweeen the major axes.

Therefore, view-aligned slicing has been implemented: Polygons aligned to the view plane are generated and clipped by the bounding box of the volume (see Figure 2.4). 3D texture coordinates are assigned to the vertices, and the polygons are rendered in depth order. Due to its limited flexibility, slicing has mostly been superseded by ray casting techniques.

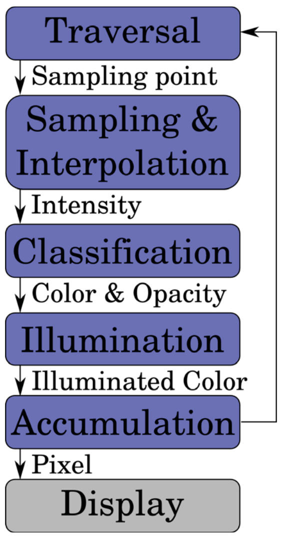


Figure 2.2: Volume Rendering Pipeline.

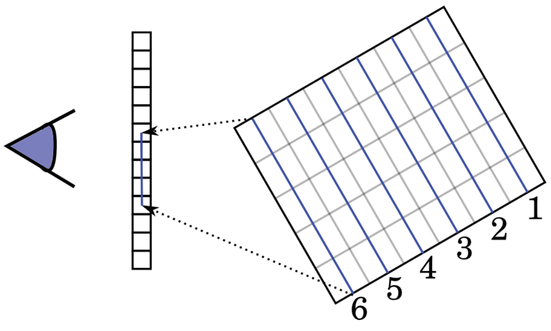


Figure 2.3: Data set-aligned slicing.

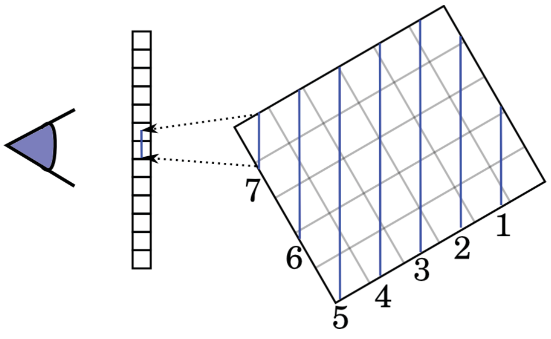


Figure 2.4: Screen-aligned slicing.

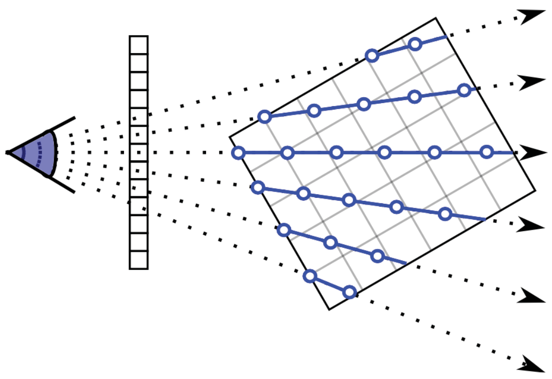


Figure 2.5: Volume raycasting.

2.2.4 Ray Casting

Ray casting is a commonly used direct volume rendering technique that allows for efficient implementation on modern GPUs. Ray casting is a simpler version of ray tracing that does not implement refractions and reflections. Rays are cast for every pixel of the screen and intersections with objects in the scene are computed in depth order. Volume ray casting has to compute the intersection of each ray with the volume and then sample it along the path (see Figure 2.5) as described in Section 2.2.2. The volume in the scene is commonly represented by a *proxy-geometry* and intersections are computed with this geometry. The trivial proxy-geometry is a cube matching the extent of the volume. Clipping of the proxy-geometry results in a clipped volume rendering. The use of a proxy-geometry also allows *empty-space skipping*: Empty parts of the volume (with respect to the transfer-function) are not included in the proxy-geometry and therefore optimize the distance to be sampled inside the volume. For GPU-based ray casting, an optimization technique proposed by Krüger and Westermann [KW03] is used to compute the ray entry and exit points: The proxy-geometry is rasterized by the GPU, which is highly optimized for this operation. First, all front-facing polygons are rendered, discarding all but the front-most fragments. Then, all back-facing polygons are rendered to compute the most distant fragments. The result is saved in two RGBA textures: The entry-point and exit-point textures. Coordinates in texture space are encoded in the RGB values of each pixel. In the actual raytracing step, two texture lookups are all that is needed to retrieve the ray parameters. Figure 3.3 illustrates these steps as implemented in Voreen.

2.2.5 Multi-Volume Ray Casting

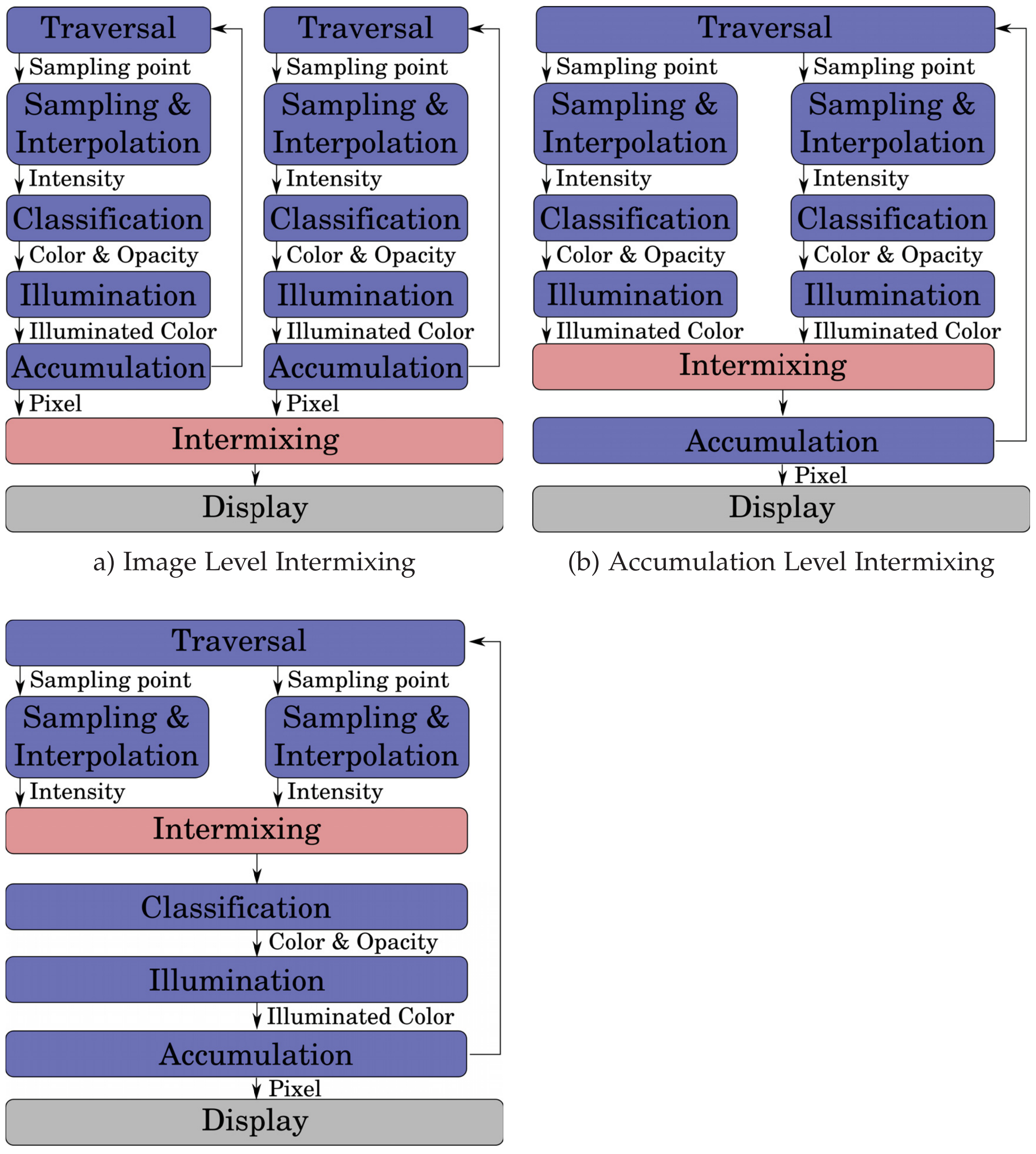
To render multiple volumes at once, the data has to be combined at some point in the rendering pipeline. Cai and Sakas [CS99] propose three data intermixing approaches: Image level intermixing, accumulation level intermixing and illumination model level intermixing. Schubert and Scholl [SS11] add classification level intermixing to this list.

Image Level Intermixing takes the resulting images of multiple single-volume ray castings and combines them into one image (see Figure 2.6 a). Several composing operators have been implemented, including depth-based blending. Rendering using image level intermixing generates unintuitive results, structures from different modalities do not occlude each other properly. Depth perception is therefore limited.

Accumulation Level Intermixing combines shaded colors from all volumes at each sampling point (see Figure 2.6 b). Structures from different volumes can therefore occlude each other in a manner consistent with the emission-absorption model.

Illumination Level Intermixing mixes the absorption coefficent (κ) of different volumes instead of blending the transparency values (T) and the color.

Classification Level Intermixing mixes the intensities of multiple volumes (see Figure 2.6 c). A meaningful result can only be achieved for volumes of the same modality.



a) Image Level Intermixing

(b) Accumulation Level Intermixing

(c) Classification Level Intermixing

Figure 2.6: Intermixing at different stages of the multi-volume rendering pipeline.

Chapter 3

The Voreen Framework

In this chapter we will describe the basic concepts of Voreen, the **Vo**lume **re**ndering **en**gine. A discussion of processors, ports and properties as well as their combination into rendering networks will form the basis for the extension of the platform towards a rapid application development framework described in the next chapter.

The general idea behind the Voreen project is to split up the steps necessary to render a volumetric data set into small building blocks in order to facilitate implementation of new visualization techniques. A `Processor` is the most important of these blocks, it encapsulates an algorithm with its inputs, outputs and parameters into a class with a well-defined interface. Input and output of data (e.g., volumes, images) are specified by `Ports`. Additional parameters (e.g., the size of a kernel for an image/volume filter) are specified as `Properties`. Execution of the algorithm encapsulated in the processor is triggered by a call of the `process()` method. A suitable combination of multiple processors with connections between in- and outports of the same type into a pipeline thus creates a desired visualization. Due to the common interface, parts of this rendering pipeline can easily be added or exchanged to experiment with new approaches. Examples for such modifications are pre-processing of volume data, the addition of image filters as post-processing or performance enhancements using an optimized proxy-geometry processor.
In early versions of Voreen the pipeline had to be constructed by program code, which turned out to be quite cumbersome and prone to errors and which did not facilitate on-the-fly experimentation by modifying the pipeline. The solution to this problem, as described by Meyer-Spradow et al. [MSRMH09], is to introduce visual programming concepts to construct rendering pipelines. Processors are arranged into *data-flow networks* (class `ProcessorNetwork`) which are sorted topologically by the `NetworkEvaluator`. The `NetworkEvaluator` evaluates a network by calling the `process()` method of each processor in the network.

Figure 3.1 shows an example network consisting of typical stages in a volume rendering pipeline: Input (`VolumeSource`), pre-processing (`VolumeResample`), rendering (`SliceViewer`), post-processing (`Background`) and output (`Canvas`).

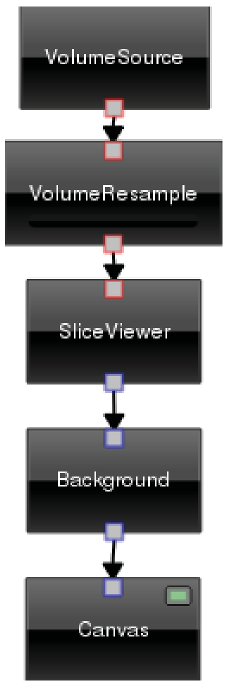


Figure 3.1: Processor network: A typical volume visualization pipeline, consisting of input, pre-processing, rendering, post-processing and output (top to bottom).

3.1 Graphical User Interface

Voreen has been designed to be independent of the utilized GUI-toolkit and can therefore be easily integrated into other applications. This is achieved by a strict separation into multiple layers: The `voreen_core` library handles all functionality described so far, GUI-toolkit libraries like the `voreen_qt` library provide commonly used widgets for specific toolkits, and applications utilize both libraries to integrate them into their GUI *. GUI-related functionality (e.g., events) in the core library is abstracted using the `tgt` (Tiny Graphics Toolbox) library.

* Pure console applications utilizing only the core library are also possible.

Developers can add a GUI to a processor by adding properties or by implementing a `ProcessorWidget`. Widgets for properties are automatically generated by the GUI-toolkit library and require no additional effort by the developer. Since a wide range of property-types (boolean, integer, float, vector and matrix types, transfer functions, shaders, file dialogs, etc.) is supported, this auto-generated GUI is sufficient for the vast majority of processors. To implement a customized user interface it is possible to implement a `ProcessorWidget` in a specific GUI-toolkit. The most prominent example for this is the output window displayed by the `CanvasRenderer`.

3.2 VoreenVE

The VoreenVE (Voreen **V**isualization **E**nvironment) provides the user with a graphical user interface (GUI) to build processor networks at runtime. Figure 3.2 shows the VoreenVE user interface with the example network discussed earlier. The user can drag processors from the processor-list (left) into the network-editor (center), properties of the currently selected processor are displayed on the right side of the screen. The resulting image is displayed on the canvas.
Networks built by the user are saved in an XML-based format using a serialization framework implemented for Voreen. Serialization captures the entire state of the network, including all properties (e.g., loaded data sets, transfer functions, etc.). Although these networks are constructed in VoreenVE, they can be loaded while retaining full functionality in other applications using the voreen core library.

VoreenVE can be switched into an *application mode* in which only a configurable subset of properties is shown to the user and the network editor is hidden.

3.3 Volume Ray Casting in Voreen

The standard renderer in Voreen (`SingleVolumeRaycaster`) implements Krüger-Westermann [KW03] ray casting using entry- and exit-point textures. The processors and rendering network have changed slightly from the ones described by Meyer-Spradow et al. [MSRMH09]. Instead of using a coprocessor port to connect proxy-geometry and entry-exit-point processors the proxy-geometry is now passed on using a `GeometryPort`. The rendering network is shown in Figure 3.3: A volume is loaded into the `VolumeSource` processor, which is used by the `CubeProxyGeometry` processor to create a proxy-geometry to be rendered by the `MeshEntryExitPoints` processor. The resulting entry- and exit-point textures in combination with the volume are used to perform the ray casting using GLSL in the `SingleVolumeRaycaster`.

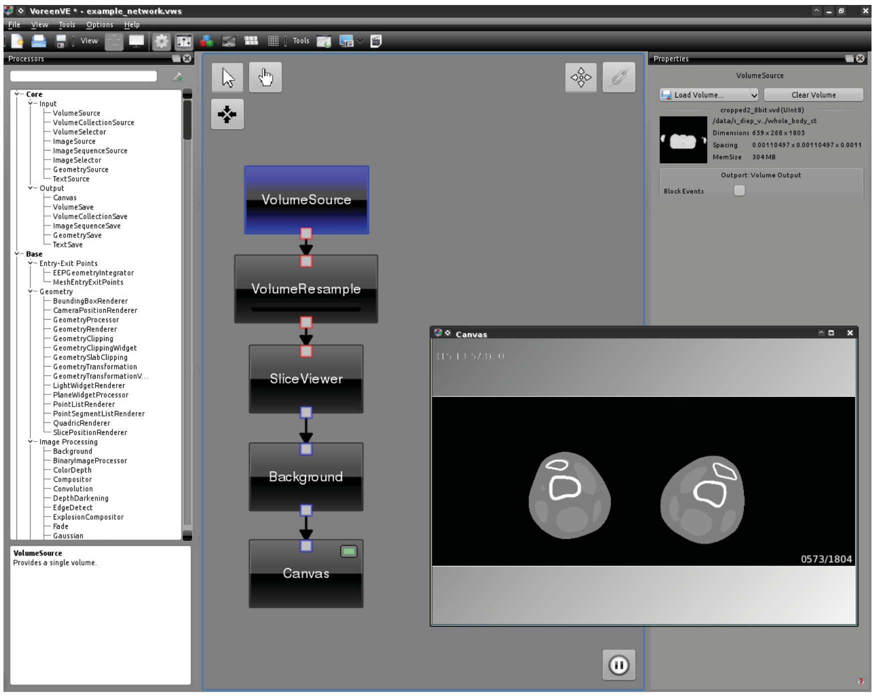


Figure 3.2: VoreenVE Application: The interface consists of the processor list (left), the network editor (center), the property list (right) and processor widgets, in this case a canvas (floating on top).

3.4 Developing Applications in Voreen

Development of novel visualization applications will usually require implementation of new components that work with the rest of the framework. This could include readers for new formats, new port types, processors, properties etc. Components designed for one application or for one purpose (e.g., DTI processing) are grouped into a *module*. Modules to be included in Voreen can be configured at compile time using the CMake build system *.

* http://www.cmake.org

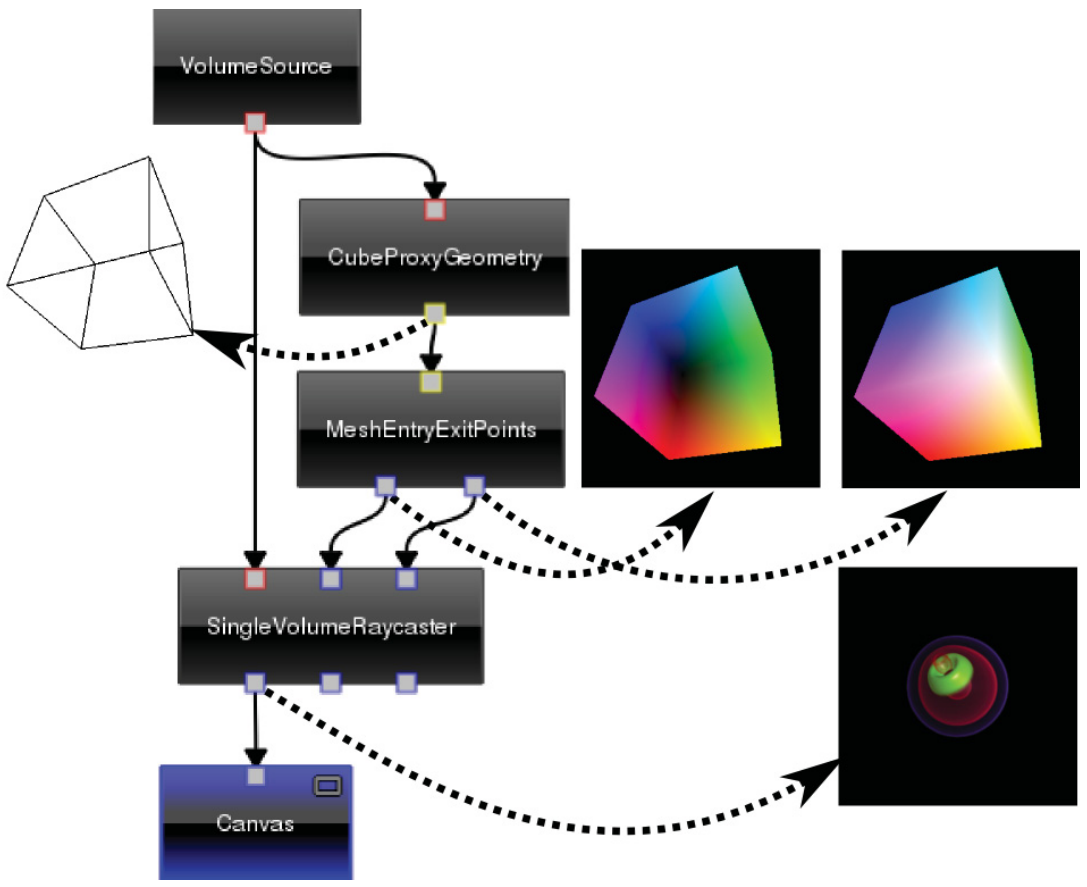


Figure 3.3: Standard Ray Casting Network consisting of `VolumeSource`, `CubeProxyGeometry`, `MeshEntryExitPoints` and `SingleVolumeRaycaster`.

Chapter 4

Extensions to the Voreen Framework

The focus of Voreen development has shifted from volume ray casting towards building a more fully featured and extendible framework for the visual analysis of volume data. This chapter will discuss the changes to the architecture that enabled the development of the applications discussed in the following chapters.

4.1 Changes to the Property Concept

While the types of properties used in Voreen have not changed much from the ones described by Meyer-Spradow [MS09, pp. 40-42], their usage throughout the framework has grown. A base-class `PropertyOwner` has been introduced to allow classes other than processors to have properties. Furthermore, singletons [*] are no longer used to store settings, a linking concept has been introduced to control synchronization between properties, and a property-based animation framework has been implemented. In combination with the network-based support for multi-view development this has made coding of new applications based on the Voreen libraries mostly obsolete, since most functionality can now be realized in VoreenVE by network-construction.

4.1.1 Implicit Caching

Explicit caching using special processors has been removed from Voreen in favor of an automatic mechanism to only call `process()` for processors that do not have a valid result. Other processors, however, might need to do more than what is normally done

[*] A design-pattern that restricts the number of instances of a class to one. [GHJV95, pp. 127-134]

in the `process()` method. One common example is the recompilation of shaders: Per-frame recompilation should be avoided for performance reasons, but some properties (e.g., shading model) require the processor to rebuild the shader after their value has changed. For this purpose the `PropertyOwner` class, from which `Processor` derives, has an *invalidation level*. The level is stored as integer, with a set of constants defined using the enumeration `Processor::InvalidationLevel`. A `VALID`(=0) processor does not need to be re-evaluated, a processor with the level `INVALID_RESULT` or greater will be evaluated by the `NetworkEvaluator`. Additional levels indicate a changed number of ports or the need to recompile shaders. A processor with the invalidation level `INVALID_PROGRAM` will first recompile its GLSL shader or OpenCL kernel before calling the usual rendering code. Properties and ports are assigned an invalidation level using their constructor and call the `invalidate(level)` method of the processor they belong to. The processor will set its invalidation level to the maximum of its current level and the level of the property or port. After the network evaluator has called `process()` on a processor its level is reset to `VALID`.

4.1.2 Linking

The messaging concept used to synchronize values of properties [MS09, p. 21] has been removed in favor of *property linking*. Messaging did not allow the user to control which properties to synchronize and also did not support linking of different types using conversion (e.g., integer to float). Synchronization between two properties now has to be established using a unidirectional `PropertyLink`. Each link has a *link evaluator* which reacts to changes of the source property. Evaluators implement the `LinkEvaluatorBase` interface:

```
class LinkEvaluatorBase : public VoreenSerializableObject {
    /// Called by PropertyLink to execute the link.
    virtual void eval(Property* src, Property* dst) throw (VoreenException) = 0;

    // Returns true if the LinkEvaluator can link the two properties.
    virtual bool arePropertiesLinkable(const Property* src, const Property* dst) const = 0;
};
```

Listing 4.1: `LinkEvaluatorBase` interface (abbreviated)

If there is a property link from property A to property B and the value of A changes, `eval(A, B)` of the corresponding linkevaluator will be called, which will cause a change of property B. Bidirectional links are created as two unidirectional links.

Evaluators performing an id-mapping between two properties of the same type can be implemented by deriving from the `LinkEvaluatorIdGeneric` template. For types that can be casted into each other (e.g., integer to float) the `LinkEvaluatorId-`

GenericConversion template has to be used. Custom evaluators derived from the base class allow the developer to create conversions between arbitrary types.

Figure 4.1 shows an example for property linking: Two VolumeMorphology processors which can compute a dilation or erosion on a volume are combined to perform a closing. By linking the kernel size properties of both processors we can make sure that they are always identical.

4.1.3 Animation

Early versions of Voreen supported animations only using scripting or camera rotations around the origin. Using properties a flexible animation framework has been implemented. All properties for which interpolation functions have been implemented can be animated using keyframes on parallel timelines. The animation dialog is shown in Figure 4.2.

4.2 Handling of Generic Data

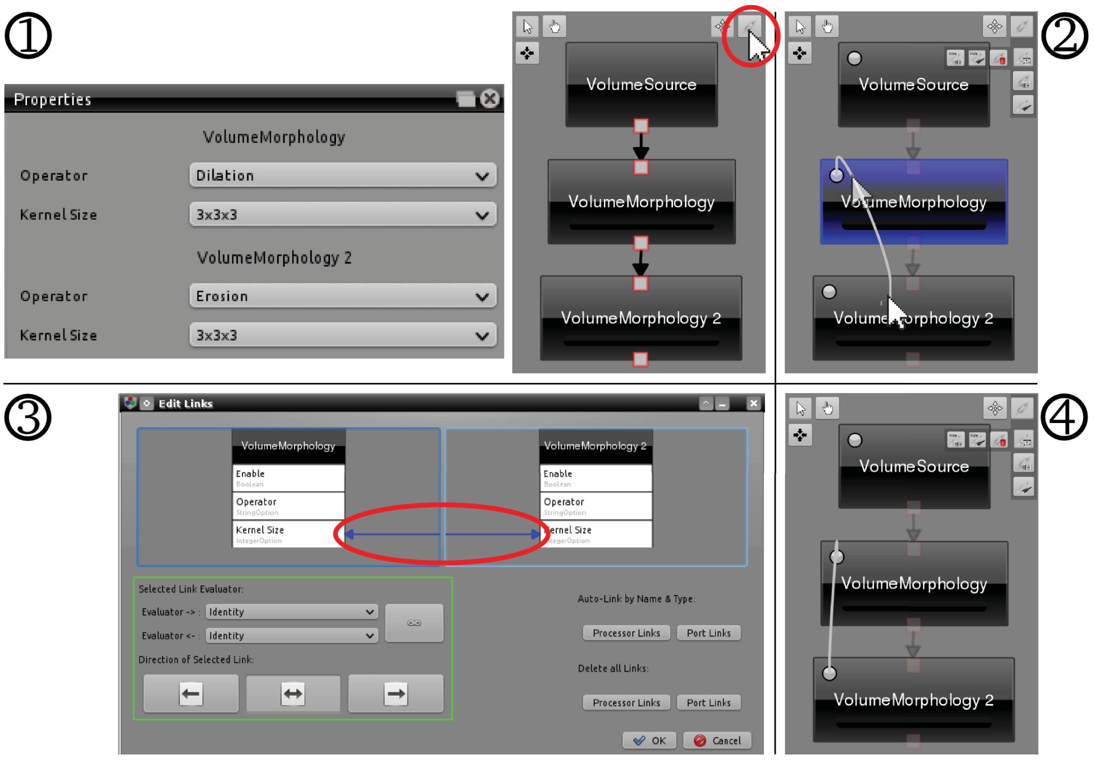
While early versions of Voreen were restricted to volume-, image- and geometry-ports, this limitation has been removed and arbitrary data can be exchanged via ports. Instead of specifying the type using a string (e.g., *"image.entrypoints"*), the different port types are now actual classes derived from the Port base class. Figure 4.3 shows a diagram of classes deriving from Port.

The following code is an excerpt of the definition of the Port class:

```
class Port : public PropertyOwner {
    enum PortDirection {
        OUTPORT,
        INPORT
    };

    Port(PortDirection direction, string& id, string& guiName,
         InvalidationLevel invalidationLevel = INVALID_RESULT);

    // Connection:
    virtual bool connect(Port* inport);
    virtual void disconnect(Port* other);
    virtual void disconnectAll();
    virtual bool testConnectivity(const Port* inport) const;
    const std::vector<const Port*> getConnected() const;
    virtual size_t getNumConnections() const;
    bool isConnected() const;
    bool isConnectedTo(const Port* port) const;
```


Properties

VolumeMorphology

Operator	Dilation
Kernel Size	3x3x3

VolumeMorphology 2

Operator	Erosion
Kernel Size	3x3x3

Edit Links

Selected Link Evaluator:

Evaluator -> : Identity

Evaluator <- : Identity

Figure 4.1: Property Linking: To link the kernel size property of both `VolumeMorphology` processors the user switches the network editor to linking mode (1) and drags a line from one processor to the other (2). A dialog showing properties of both processors opens (3). Properties can be linked by dragging arrows between them. The lower part of the dialog box provides widgets to modify direction and evaluator of the selected link (green box). Links between processors are indicated by a line between the circle in their upper left corners (4), which can be clicked to unfold a list of linked properties.

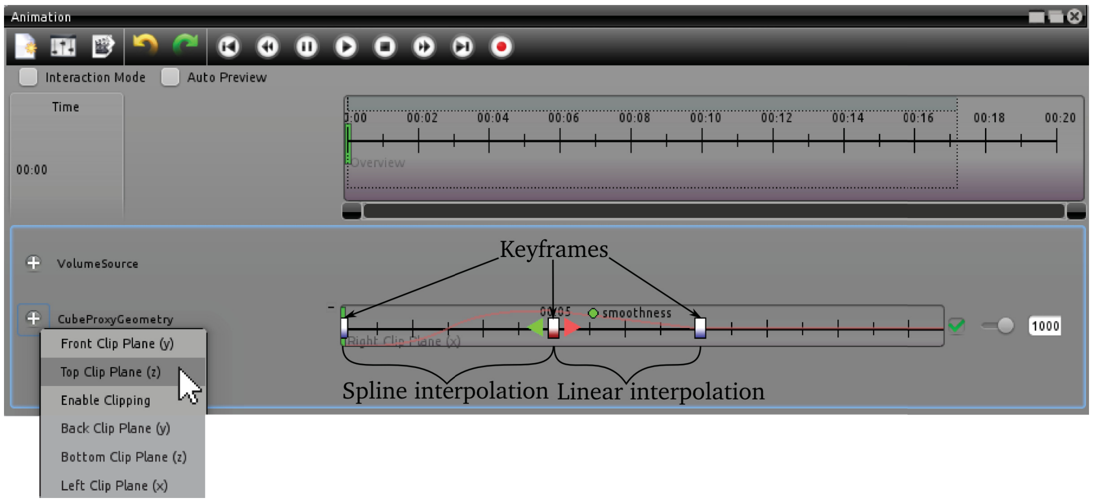


Figure 4.2: Property Animation: The animation dialog shows timelines of all animated properties. In this example, the right clipping plane of the proxy-geometry has been animated using three keyframes. Between the first two keyframes a spline animation has been selected while a linear interpolation function is used between keyframe two and three. To animate additional properties, the user can add click on the plus icon (left).

```
    // Data:
    virtual bool hasData() const;
    virtual void clear();

    virtual void invalidatePort();

    // Conditions:
    void addCondition(PortCondition* condition);
    virtual bool checkConditions() const;

    virtual bool isReady() const;

    // Caching:
    virtual bool supportsCaching() const;
    virtual std::string getHash() const;
    virtual void saveData(const std::string& path) const;
    virtual void loadData(const std::string& path);
};
```

Listing 4.2: The `Port` class (excerpt).

The constructor takes the direction, an id which has to be unique per processor, a name to be displayed in the GUI and an invalidation level which is used by inports to invalidate the processor when the data in the port changes. An inport can therefore

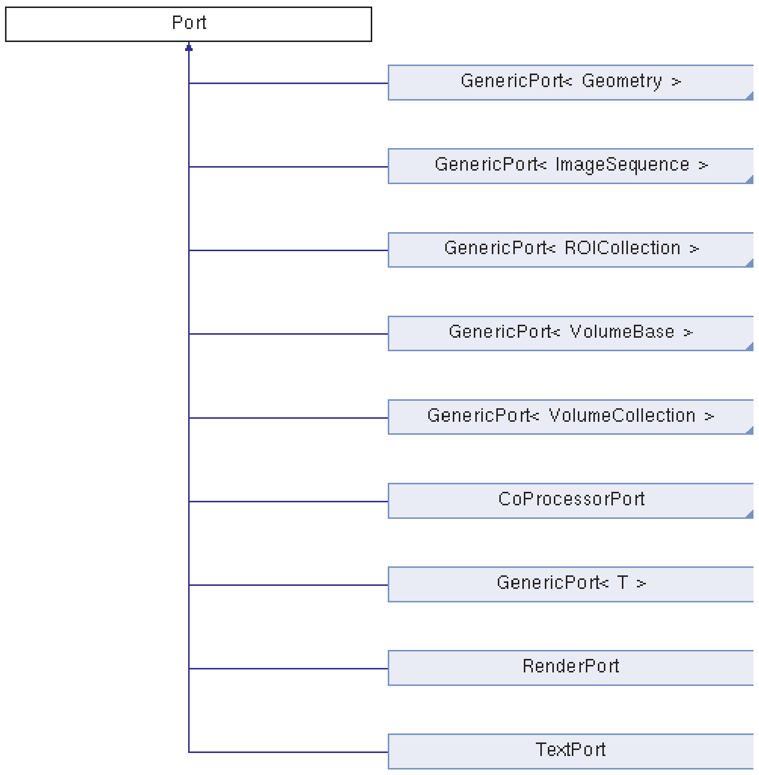

Port
GenericPort< Geometry >
GenericPort< ImageSequence >
GenericPort< ROICollection >
GenericPort< VolumeBase >
GenericPort< VolumeCollection >
CoProcessorPort
GenericPort< T >
RenderPort
TextPort

Figure 4.3: Inheritance Diagram of the class `Port`.

be configured to not invalidate the processor at all or to trigger a rebuild of a shader.

A number of methods allow connecting to and disconnecting from other ports. `testConnectivity()` only returns true if the type of both ports matches (i.e., `typeid(*portA) == typeid(*portB)`).

`PortConditions` can be added to an inport to restrict the type of data that is accepted on this inport. To only allow volumes with a floating-point voxel-type a `PortConditionVolumeTypeReal` can be added to a port.

The `isReady()` method returns `true` if a port is connected, has data and all conditions are fulfilled.

Adding ports to processors is straightforward:

```
class MyProcessor : public Processor {
    [...]
    VolumePort inport_;
    VolumePort outport_;
}

// Constructor:
MyProcessor::MyProcessor() : Processor(),
    inport_(Port::INPORT, "inport", "Volume␣Inport"),
    outport_(Port::OUTPORT, "outport", "Volume␣Outport")
{
    // register port at processor:
    addPort(inport_);
    addPort(outport_);
}
```

4.2.1 Caching

A port can support caching by implementing a hashing function (e.g., the MD5 message-digest algorithm) for its data-type and methods for saving/loading to a file. Processors that only use ports supporting caching can then cache results of their computations on disk. An entry in the cache is a path on disk that consists of the classname of the processor, the hashes of all inports and a hash of the current property-state:

$$\underbrace{data/cache/}_{\text{Cache-directory}}\ \underbrace{VolumeFilter}_{\text{Processor-classname}}\ /\ \underbrace{091d...e2}_{\text{hash(inport 1)}}\ /\ \underbrace{b4df...d1}_{\text{hash(inport 2)}}\ /\ \underbrace{3faf...90}_{\text{hash(property-state)}}\ /$$

Each outport writes its current content to this directory after a call to `process()` has finished and restores the result in case a matching cache entry for the current configuration is found.

4.2.2 GenericPort

The template class `GenericPort<T>` implements a port handling pointers of the class `T` and simplifies creation of new port types:

```
template<typename T>
class GenericPort : public Port {
public:
    GenericPort(PortDirection direction, string& id, const string& guiName = "",
                Processor::InvalidationLevel invalidationLevel = Processor::INVALID_RESULT);

    virtual void setData(const T* data, bool takeOwnership = true);
    virtual const T* getData() const;
    virtual T* getWritableData();

    virtual bool hasData() const;
    virtual bool isReady() const;
    virtual void clear();
protected:
    const T* portData_;
    bool ownsData_;
};
```

Listing 4.3: The `GenericPort` template class (abbreviated).

Data is stored only in outports and is set using `setData()`. In case the `takeOwnership` parameter is `true`, the port will delete the data in the destructor or when new data is set. `getData()` returns a const pointer to prevent processors from changing incoming data. `getWritableData()` only works on outports and allows the processor that computed the data to obtain a non-const pointer to the data.
The container/portmapping concept found in early versions of Voreen [MS09, p. 37] has been abandoned for all data types, data is now accessed directly:

```
void MyProcessor::process() {
    // Get input:
    const VolumeBase* inputVolume = inport_.getData();
    [...] // Perform calculations
    // Set output data:
    outport_.setData(result);
}
```

4.2.3 CoprocessorPort

Whereas processors normally do not communicate with other processors directly, coprocessor ports allow processors to utilize algorithms implemented in other processors. The coprocessor concept has been redesigned into the template class `CoprocessorPort<T>`. The template parameter `T` specifies an interface that is implemented by all processors adding a coprocessor-outport of this type.

Rendering of geometries in Voreen is bundled by the `GeometryProcessor` to reduce the number of used render targets. Therefore a number of processors implementing the `GeometryRendererBase` interface can be connected using coprocessor ports of type `CoprocessorPort<GeometryRendererBase>`. As shown in Figure 4.4, the `GeometryProcessor` has a `CoprocessorPort<GeometryRendererBase>`-inport whereas the processors performing the rendering have a `CoprocessorPort<GeometryRendererBase>`-outport. In the `process()` method of the `GeometryProcessor` connected coprocessors can easily be queried, and all methods implemented in the `GeometryRendererBase` class can be called:

```
void GeometryProcessor::process() {
    [...]
    vector<GeometryRendererBase*> portData = cpPort_.getConnectedProcessors();
    for (size_t i=0; i<portData.size(); i++) {
        GeometryRendererBase* geomRenderer = portData.at(i);
        if(geomRenderer->isReady()) {
            geomRenderer->setCamera(camera_.get());
            geomRenderer->setViewport(outport_.getSize());
            geomRenderer->render();
    }
    [...]
}
```

4.2.4 Handling of DTI Data

This section describes how DTI is handled in Voreen and also serves as an example on how to integrate new types of data into the Voreen architecture. For a detailed description of algorithms implemented in the processors discussed here we refer the reader to [SzB11]. An application utilizing the building blocks described here will be discussed in Chapter 5.
The following types of data occur in DTI workflows and are handled in Voreen:

- **Diffusion Weighed Images (DWI)** are the reconstructed images. They are read using a `VolumeReader` that is aware of the relevant metadata entries, and are represented by a `VolumeCollection` in Voreen. Each volume has metadata entries storing the gradient and b-value.
- **Diffusion Tensor Images (DTI)**, generated from a number of DWIs by tensor-estimation or read using a compatible `VolumeReader` are represented by a volume storing a second degree tensor at each voxel (with a representation of type `VolumeRAM_Tensor2Float`) in Voreen.
- **Fibers**, generated from DT-volumes using fiber-tracking algorithms are represented by the class `Fibers` (an array of `FiberLines`) in Voreen.

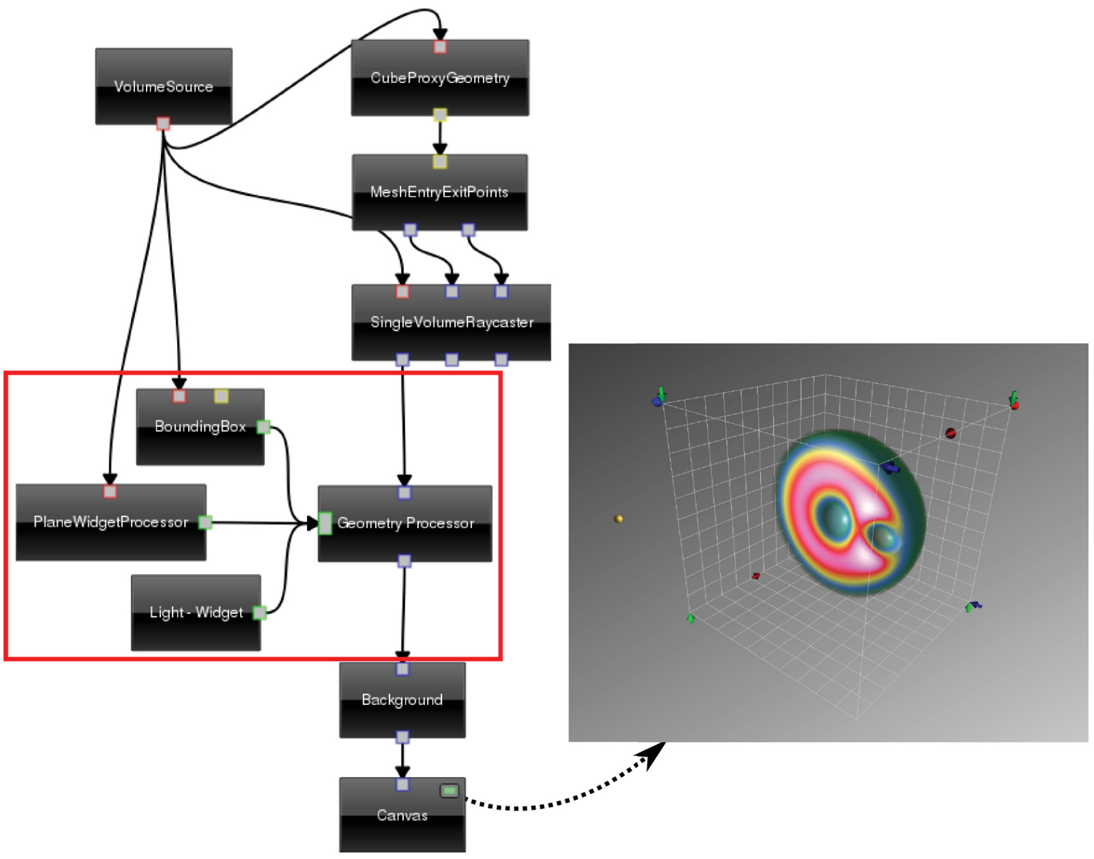


Figure 4.4: Coprocessor ports are used to call methods on other processors, in this case the `render()` method of three processors implementing the `GeometryRendererBase` class.

- Data derived from DTIs (e.g., eigenvectors, anisotropy, diffusivity etc.) are stored in scalar or vector volumes and can be processed or visualized by standard Voreen functionality.

While standard `VolumePorts` and `VolumeCollectionPorts` are used to transfer DWI and DTI data, we need to define a new `FiberPort` for the fibers. New ports are implemented by deriving a class from `Port` or, for most cases, from the template `GenericPort<T>`:

```
class FiberPort : public GenericPort<Fibers> {
public:
    FiberPort(PortDirection direction, string id, string guiName = "",
              InvalidationLevel invalidationLevel = INVALID_RESULT);

    virtual string getClassName() const {
        return "FiberPort";
    }

    [...] // optional methods to implement caching functionality
};
```

DTI Processors

After loading the DWIs in a `VolumeCollectionSource` processor the resulting `Volume-Collection` will usually be the input of a processor performing tensor estimation. This can either be handled by the `DiffusionTensorEstimator` which implements the algorithm by Westin et al. [WMM+02] or using the `ModelFitCamino` processor, which wraps the Camino toolkit [*] and thus supports a wide range of advanced algorithms. In most applications the DT-volume will be further processed by the `TensorAnalyzer`, which derives eigenvalues and eigenvectors as well as different anisotropy and diffusivity measures. The resulting scalar or vector volumes can be directly visualized or analyzed by existing Voreen processors (e.g., `SliceViewer`, ROI analysis). Alternatively, the `TensorGlyphRenderer` implements glyph visualization techniques. Finally, the `FiberTrackerFACT` and `FiberTrackTensorline` processors provide second order Runge-Kutta and tensorline [WKL99] based fiber tracking algorithms. Aside from the DTI data these processors also have a ROI inport to specify a seeding region. The resulting fibers are rendered by the `FiberRenderer` which implements rendering using illuminated streamlines [ZSH96], triangle strips [MSE+06] and streamtubes. Using compositing, fibers can be integrated with other rendering techniques. Figure 4.5 shows an example DTI rendering network containing the described processors.

4.3 Multi-View Support

Voreen supports creation of multi-view applications by rendering to multiple canvases and by subdividing canvases (see Figure 4.6). Each `CanvasRenderer` processor displays the image obtained on its inport in its `ProcessorWidget` (e.g., a `QGLWidget` for Qt-based applications). Additional processors to arrange multiple images in one canvas have been implemented. The `QuadView` and `TripleView` processors provide two commonly used multi-view configurations, dividing the view into four and three sub-views, respectively. For a more flexible layout, the `Splitter` processor divides the

[*] http://web4.cs.ucl.ac.uk/research/medic/camino/pmwiki/pmwiki.php?n=Main.HomePage

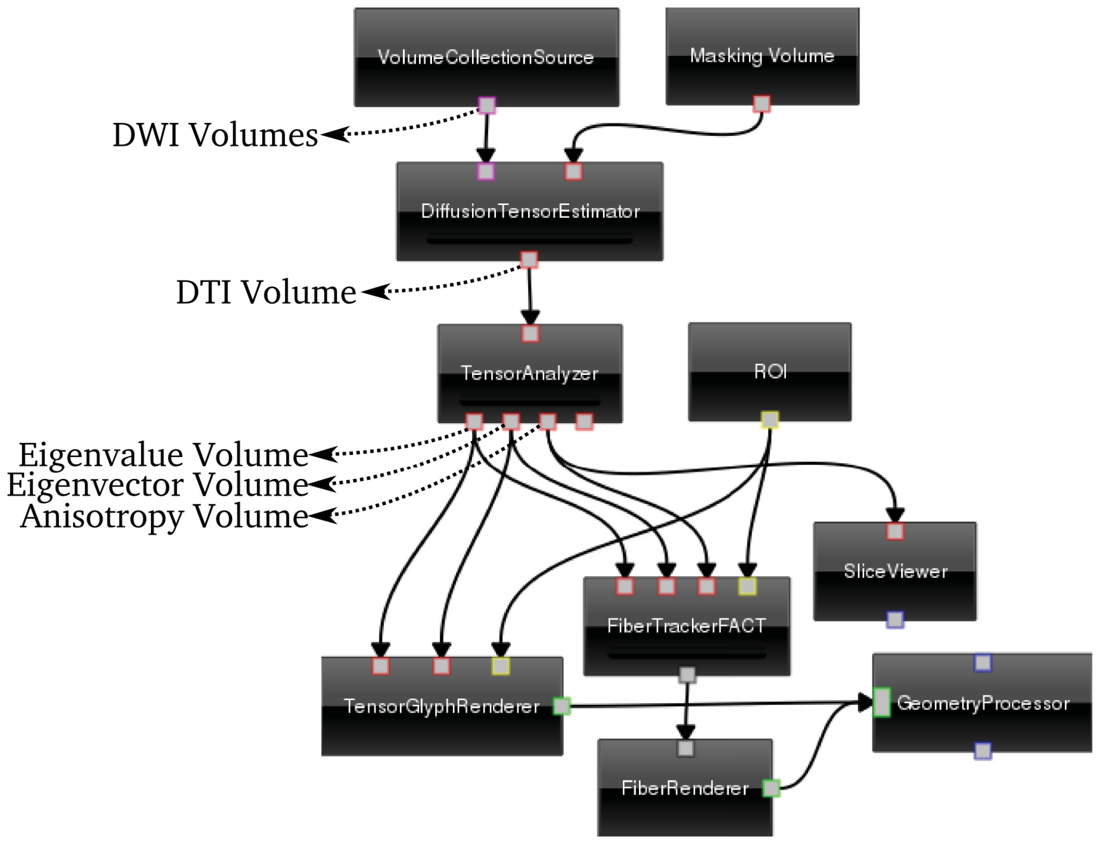


Figure 4.5: Example DTI Network containing the most important processors in a typical configuration.

view vertically or horizontally at a user-specified position. Several of these processors can be combined to create custom screen configurations.

4.3.1 Resizing and Events

While combining multiple input images into one is trivial, events (e.g., mouse events) and resizing of the output window need to be handled as well. Events on the `ProcessorWidget` of the `CanvasRenderer` are first converted from their native toolkit type to a corresponding `tgt::Event` subtype to keep the Voreen core library GUI toolkit independent. Next, the events are propagated through the network from the canvas upward. Since rendering processors in the network are not aware of the currently used multi-view configuration, the multi-view processors discussed in the previous section need to transform mouse events and forward them only to the

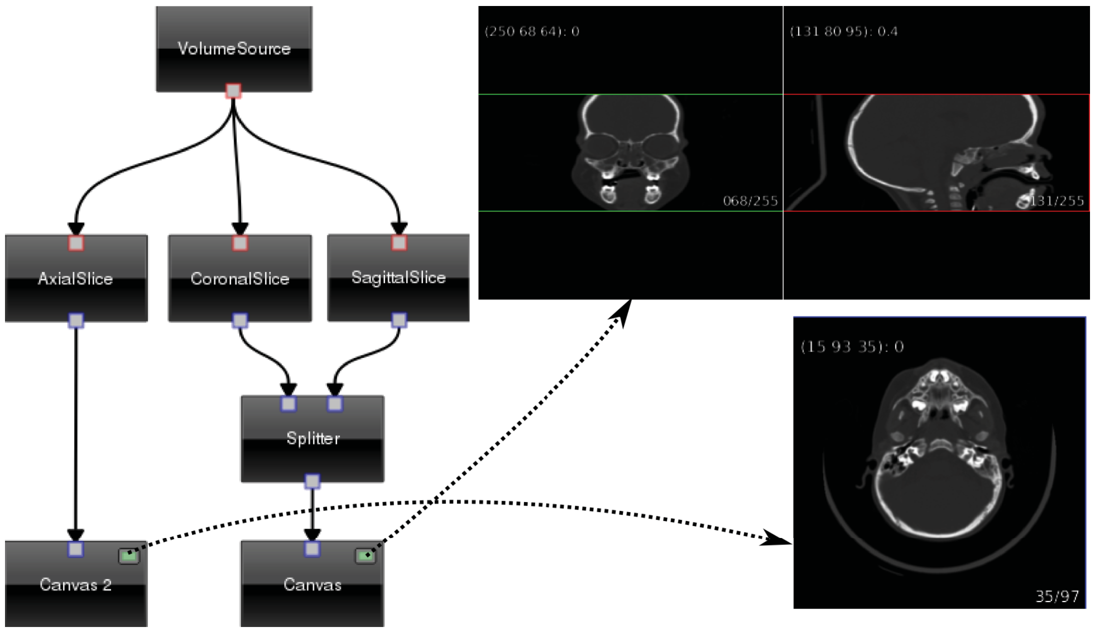


Figure 4.6: Multiple views: Voreen can handle multiple canvases and subdivision into sub-canvases using multi-view processors like the `Splitter` used in this network.

relevant inports (see Figure 4.7).

Images generated by rendering processors in the network obviously need to be computed in the size in which they are displayed on one of the canvases. A size-propagation similar to the event handling we just described proved to be problematic because images of constant size (e.g., slices) are also distributed through the network. Therefore, the linking mechanism of Voreen has been exploited to implement a render-size request system. Outports of processors that generate images have properties that allow to configure the size of the generated image, whereas the inport of the `CanvasRenderer` has a property that provides the current canvas size. Linking both properties results in matching image dimensions. Multi-view processors have size-properties on both inports and outports and compute the size to request on each inport based on the requested outport size. Size-links can be configured in a special mode (layer) inside the network-editor (see Figure 4.8), an automatic linking algorithm handles most cases for novice users.

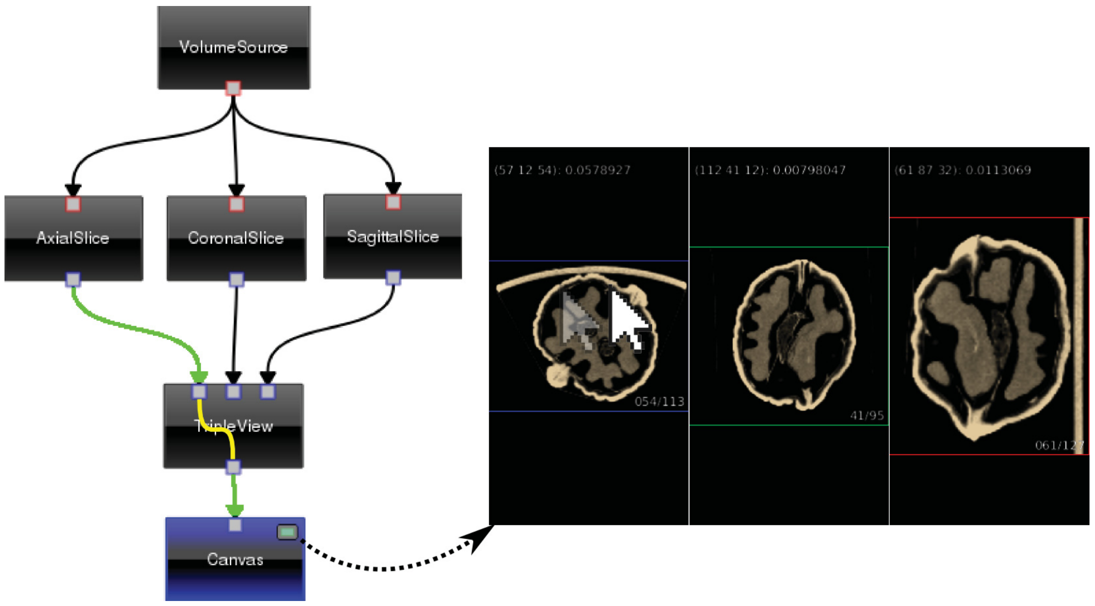


Figure 4.7: Event handling in a multi-view network: The mouse movement event is propagated form the Canvas processor through the network (green/yellow). The TripleView processor determines which of the inports to forward the event to and transforms the mouse position accordingly. The AxialSlice processor receives the event and updates the intensity under the mouse cursor. Therefore, it does not need to be aware of the multi-view configuration.

4.3.2 Cameras

In contrast to earlier versions of Voreen with only one rendering where one camera was stored in a singleton, it now makes sense to allow multiple cameras for multi-view networks. Cameras were therefore also made properties (`CameraProperty`) and are linkable between processors. This allows the user to construct networks that show the same data set from multiple perspectives or compare multiple data sets from the same perspective. Since most single-view networks require linked cameras to function properly and novice users might not be aware of this, an automatic camera-linking functionality has been added.

4.4 Redesign of Volume Data Structure

With the strong focus on its rendering capabilites, Voreen has been missing functionality to read and handle most metadata items commonly used. Therefore, the classes

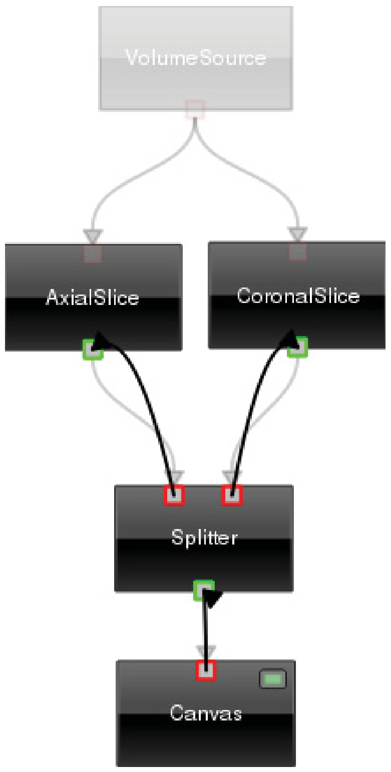


Figure 4.8: Size-linking: The current size of the output canvas is linked to the outport of the `Splitter` processor, which in turn requests images at half the size on its inports. This size is then linked to the `SliceViewers` which create the image.

handling volume data in Voreen have been redesigned to handle generic metadata, intensity values and transformations. Volumes in Voreen contain metadata (Spacing, Transformation, Patient ID, ...) and image data. The `Volume` class holds the metadata and manages one or more representations of the image. Representations store the raw image data and its dimensions. `VolumeRepresentation` is the base class, current subclasses are:

- `VolumeRAM`: The volume data in RAM.
- `VolumeGL`: The volume data as OpenGL texture.
- `VolumeDisk`: The volume on disk as filename, offset (in the file) and voxel data type. The `VolumeDisk` can be used to delay loading until the raw image data is needed in RAM (lazy loading).

The representation system is designed to be flexible with regard to new types which could be required by new APIs. `Volume::getRepresentation<T>()` returns the desired representation:

```
const Volume* vol = inport_.getData();
const VolumeRAM* vr = vol->getRepresentation<VolumeRAM>();
vr->getVoxelNormalized(10, 10, 10); // Read voxel at position (10, 10, 10)
```

Conversions between representations are handled transparently (i.e., if the data is currently only in the RAM and `vol->getRepresentation<VolumeGL>()` is called a `VolumeGL` will be created). These conversions are handled by representation converters (e.g., `RepresentationConverterUploadGL, RepresentationConverterDownloadGL, RepresentationConverterLoadFromDisk`). A new representation therefore also needs to implement at least one converter. If no converter can perform a direct conversion the representation is first converted to a `VolumeRAM` and then to the desired representation. `getRepresentation<T>()` returns a const-pointer to make sure the data is not modified (which would make other representations inconsistent). To get a non-const-pointer `getWritableRepresentation<T>()` has to be called. This will automatically delete all other representations because they are in an inconsistent state. Deleted representations are automatically re-created on demand:

```
Volume* vol = <Get the volume from somewhere>;
VolumeRAM* vr = vol->getWritableRepresentation<VolumeRAM>();
// All other rep. (e.g., VolumeGL) are deleted
vr->setVoxelNormalized(0.5, 10, 10, 10); // Set voxel at position (10, 10, 10) to 0.5
...
const VolumeGL* vgl = vol->getRepresentation<VolumeGL>();
// VolumeGL is re-created from changed image data
```

4.4.1 Metadata Handling

Metadata are stored as key-value pairs in the volume:

```
vol->setMetaDataValue<StringMetaData>("PatientName", "John␣Doe");
```

There are a number of shortcut-methods for frequently used metadata items: `getSpacing()`, `getOffset()`, ...

Decorating Volumes

In order to modify the metadata of a volume in a processor without copying the image data, Voreen exploits the decorator design pattern *. `VolumePorts` therefore

* A design pattern that allows the extension of objects at run-time. A decorator wraps the original class. [GHJV95, pp. 175-184]

return objects of type `VolumeBase` instead of **Volume**. This could either be an actual `Volume` or a decorated one. The following code replaces the transformation matrix for a volume and could be used in a co-registration processor:

```
const VolumeBase* vol = inport_.getData();
mat4 transformationMatrix = <compute registration>;
VolumeBase* outVol = new VolumeDecoratorReplaceTransformation(vol, transformationMatrix);
outport_.setData(outVol);
```

4.4.2 Derived Data

Data like the histogram of a volume could be used repeatedly in the processors of the Voreen network. One example for this case would be multiple slice viewers displaying different orientations of one volume. All of these utilize the histogram for the transfer function editor. We call data that is calculated from the volume *derived data* and also store it in the volume, allowing for transparent caching:

```
float min = vol->getDerivedData<VolumeMinMax>()->getMin();
```

If the min/max values inside the volume have been computed before, the stored `VolumeMinMax` object is returned, otherwise it is created. All derived data objects are deleted upon calling `getWritableRepresentation<T>()`.

4.4.3 Coordinate Systems

There are three similar, unit-less coordinate systems for a volume in Voreen (see Figure 4.9):

- **Voxel Indices** are used to access voxels using getVoxel();
- **Voxel Coordinates** come into play when filtering is involved (e.g., getVoxelLinear(1.5, 2.0, 3.1))
- **Texture Coordinates** are mainly used on the GPU (OpenGL/GLSL)

To give a volume a real, physical extent we introduce two additional coordinate systems (see Figure 4.10):

- To convert from Voxel Coordinates to **Physical Coordinates** we multiply by the spacing and add the offset (both in mm).
- The **World Coordinates** are defined by applying a 4x4 transformation matrix to the Physical Coordinates.

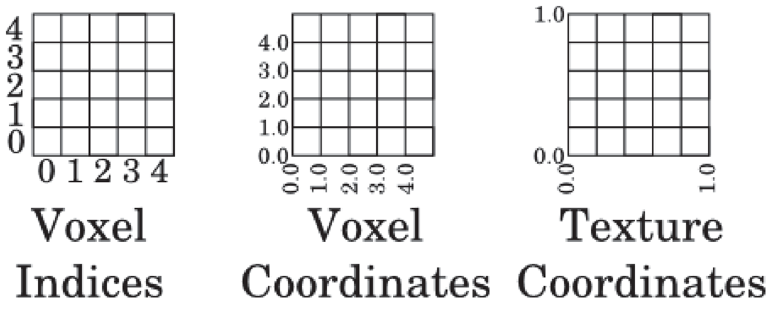


Figure 4.9: Voxel coordinate systems in Voreen.

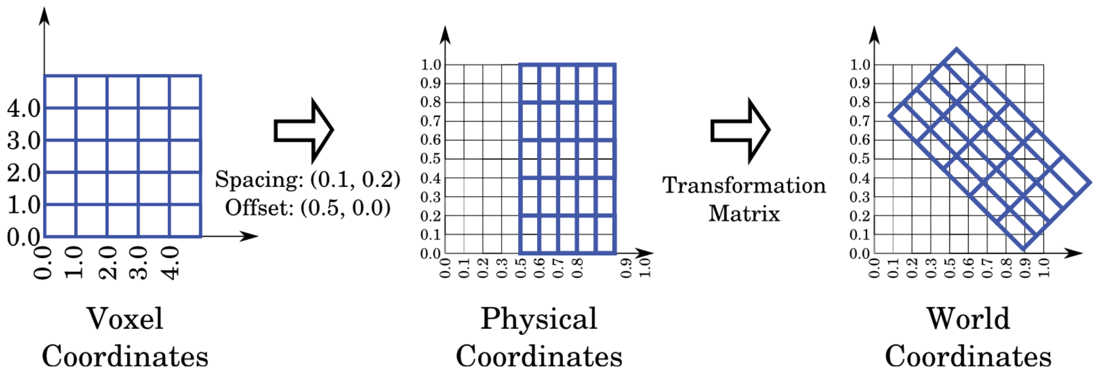


Figure 4.10: Transformation from voxel to physical and world space.

The offset is not strictly necessary since it could be integrated into the transformation matrix, but allows for easy cropping and gives us a definition of Physical Coordinates which is very similar to the one ITK uses. The following example calculates the world-space position of a voxel-center:

```
ivec3 pVoxel = ivec3(5, 4, 3);
vec3 pWorld = vol->getVoxelToWorldMatrix() * (vec3(pVoxel) + vec3(0.5));
```

Transformation of a point from physical coordinates of one volume to physical coordinates of another volume:

```
vec3 pPhys1 = vec3(10.0, 20.0, 10.0);
vec3 pPhys2 = vol2->getWorldToPhysicalMatrix() * vol1->getPhysicalToWorldMatrix() * pPhys1;
```

4.4.4 Values

The image data in commonly used formats like DICOM is mostly stored in integer format (e.g., 16 bit unsigned), whereas the actual values could be floating point numbers or lie outside of the type range. These formats therefore store a transformation inside the metadata to map voxel intensities to physical units. To implement this feature in Voreen we store a metadata item of type `RealWorldMapping` in each volume. This linear mapping also stores a string specifying the unit of the values (e.g., HU, °C, ...). Similar to the coordinate systems we just defined there are transformations for voxel values.

- **Unnormalized Values** are the values in the format of the volume. For a volume stored as unsigned int using 8 bits per voxel these are in the range $[0, 255]$, for signed integers using 16 bits the range is $[-32.768, 32.767]$. Floats are not limited (aside from the obvious technical limitations).
- **Normalized Values** map integer types to the range $[0, 1]$ for unsigned types and $[-1, 1]$ for signed types. Floats are not modified. getVoxelNormalized() returns normalized values.
- **Real-World Values** are Normalized Values after application of a `RealWorldMapping`.

The following code reads a value from a volume using linear interpolation and displays the result on the console:

```
const Volume* vol = <get the volume>;
const VolumeRam* volRam = vol->getRepresentation<VolumeRAM>();
RealWorldMapping rwm = vol->getRealWorldMapping();
float valNorm = volRam->getVoxelNormalizedLinear(3.141, 42.0, 666.0);
float valRW = rwm.normalizedToRealWorld(valNorm);
cout << "Value␣at␣(3.141,␣42.0,␣666.0):␣" << valRW << "␣" << rwm.getUnit();
```

The output could look like this:

```
Value at (3.141, 42.0, 666.0): 12.3 °C
```

4.5 Multimodal Volume Rendering

4.5.1 Multimodal Volume Ray Casting

Multimodal ray casting in Voreen has been implemented as extension of the technique proposed by Krüger and Westermann [KW03] which is also utilized for ray casting of single volumes in Voreen. As discussed in Chapter 3, Voreen uses a combination of the `CubeProxyGeometry, MeshEntryExitPoints and SingleVolumeRaycaster` to perform ray casting of single volumes. To render multiple volumes using accumulation level intermixing the MultiVolumeProxyGeometry and MultiVolumeRaycaster processors have been implemented. The MultiVolumeProxyGeometry accepts multiple volumes on the inport and generates a MeshListGeometry containing bounding boxes for all volumes. For ray casting of single volumes, the CubeProxyGeometry processor assigns each vertex of the proxy geometry a color corresponding to its texture coordinates. Multi-volume rendering in Voreen utilizes world-space entry/exit-point textures and performs the transformation to texture space in the raycaster, the vertices are therefore assigned their position in world space as color. The generated geometry is rendered using the existing MeshEntryExitPoints processor, which has been modified to output textures in GL_RGBA16F_ARB format to prevent clamping of colors to the $[0,1]^4$ range.

The MultiVolumeRaycaster performs the actual ray casting based on the generated ray parameters. It has one inport and a set of rendering parameter properties (transfer function, shading mode, filtering) for each volume. Up to four volumes are currently supported, which we believe is a reasonable limitation for the vast majority of application cases. Since all volumes are sampled using the same step size, the sampling rate is calculated based on the smallest spacing (smallest voxel edge lenght) of all volumes. In the ray casting shader the ray entry point and ray direction are transformed from world to texture space for each volume. During ray traversal each sampling position is advanced in texture space and the position is tested against the extents of the volume in texture space ($[0,1]^3$). If the sample is inside a volume, gradient calculation and classification as well as shading is performed and the resulting color is composited using standard DVR compositing. Similar to the SingleVolumeRaycaster, the MultiVolumeRaycaster can write first hit points to a secondary outport. These can be used for picking and are stored in world space coordinates. Figure 4.11 shows a network performing a fused rendering of two volumes.

4.5.2 Multimodal Slice Rendering

The multimodal slice rendering functionality in Voreen has been implemented similar to volume ray casting in that it uses a texture storing sampling positions. Networks performing multimodal slice rendering are therefore constructed using the same type of components: A proxy-geometry processor, a MeshEntryExitPoints processor and the actual renderer (see Figure 4.12). The AlignedSliceProxyGeometry processor generates a quad representing one axis-aligned slice in a volume. For rendering of arbitrarily oriented planes, the SliceProxyGeometry processor has been implemented. Rendering of the proxy-geometry can be performed using the existing

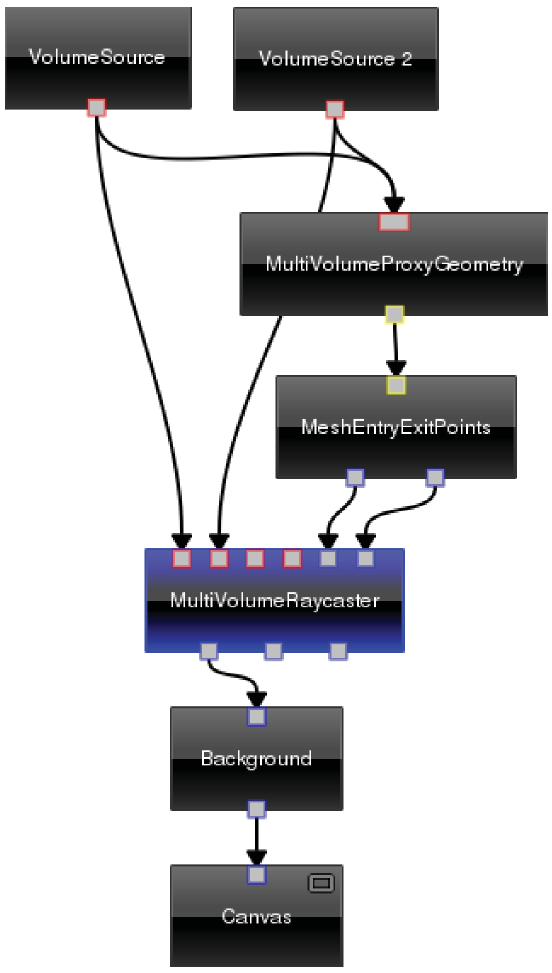


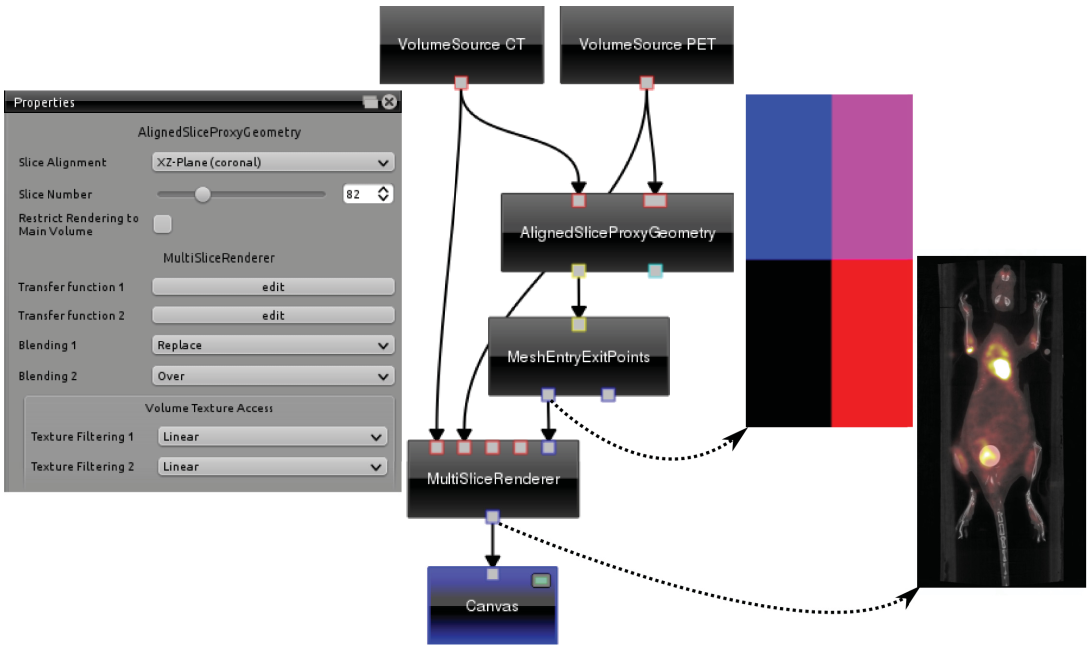
Figure 4.11: Simple multi-volume rendering network.

`MeshEntryExitPoints` processor. The resulting entry-points are then used by the `MultiSliceRenderer` which just fetches the sampling position for each voxel, transforms it from world to texture space and samples the volume at the position. Each volume has a set of properties (transfer function, compositing mode, filtering) to configure the rendering. The most commonly used compositing modes are user defined (using a slider in the GUI) mixing between two volumes and the over operator.
To create a standard slice rendering, the camera used by the `MeshEntryExitPoints` processor needs to be positioned orthogonally to the slice geometry with up and strafe vector aligned to the other main axes of the volume. The projection is configured to be orthogonal to prevent a zooming effect when the slice number is changed. To provide a suitable interaction we use the `CameraSliceInteractionHandler` instead of the trackball navigation. Dragging of the mouse is mapped to camera movement in direction of the up/strafe vectors, while zooming is mapped to a modification of the frustum instead of movement in view direction.
Because the slice is rendered in world space using a camera that is accessible as property, a combination with other renderings is especially easy. Example use cases are a combined rendering with glyphs or ROIs.

The `MultiSliceViewer` processor combines the three processors described above into one. In addition, it allows the use of 2D textures instead of the `VolumeGL` representation and can therefore display slices from volumes which would exceed the GPU memory when using 3D texturing. 2D slices are copied from the volume or resampled depending on the alignment with the selected main volume. A cache of slices is managed to improve performance while cycling through the volume. Figure 4.13 demonstrates how three `MultiSliceViewers` can be combined into a multimodal slice viewing tool. This does, however, not make the `MultiSliceRenderer` obsolete. Due to its greater flexibility it can be used for reformation techniques.

4.5.3 Coregistration

Coregistration of volumes in Voreen is performed in workspaces designed for this purpose, combining a standard rendering setup with specialized registration processors. One of the volumes is the *static volume*, to which the other volume, the *moving volume*, is registered by modifying its transformation matrix. Figure 4.14 shows a network performing interactive rigid-body coregistration of two data sets with support for automatic coregistration using the ITK library. Overall, the network is very similar to the multimodal slice rendering network discussed in the previous section. The following registration-specific processors have been added (highlighted in red):



Properties

AlignedSliceProxyGeometry

Slice Alignment	XZ-Plane (coronal)
Slice Number	82
Restrict Rendering to Main Volume	☐

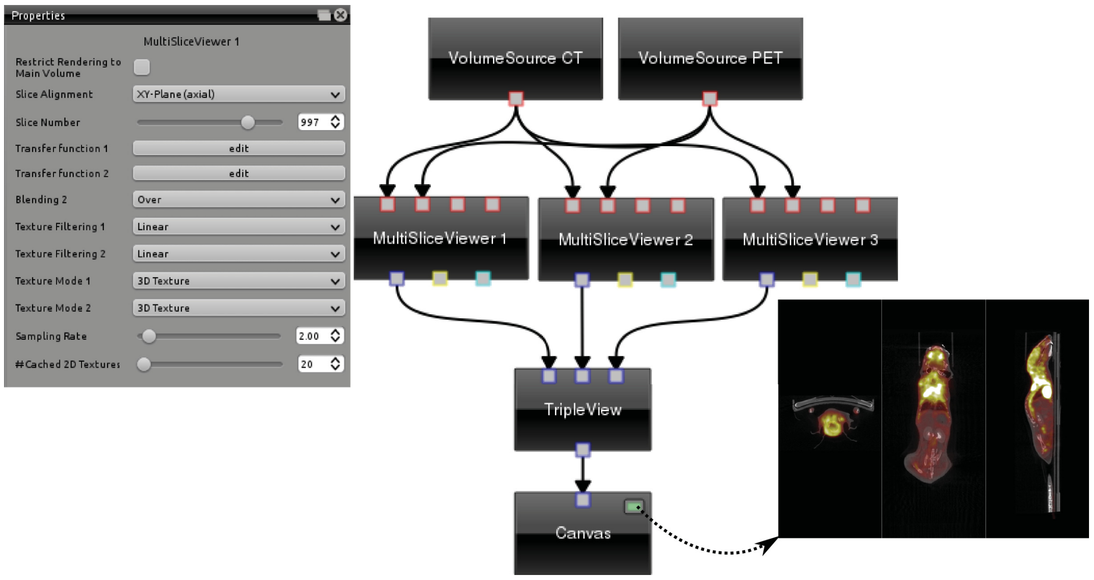
MultiSliceRenderer

Transfer function 1	edit
Transfer function 2	edit
Blending 1	Replace
Blending 2	Over

Volume Texture Access

Texture Filtering 1	Linear
Texture Filtering 2	Linear

Figure 4.12: Multimodal slice rendering using the `MultiSliceRenderer` in combination with `AlignedSliceProxyGeometry`.



Properties

MultiSliceViewer 1

Restrict Rendering to Main Volume	☐
Slice Alignment	XY-Plane (axial)
Slice Number	997
Transfer Function 1	edit
Transfer Function 2	edit
Blending 2	Over
Texture Filtering 1	Linear
Texture Filtering 2	Linear
Texture Mode 1	3D Texture
Texture Mode 2	3D Texture
Sampling Rate	2.00
#Cached 2D Textures	20

Figure 4.13: Multimodal slice viewer: This network renders a CT and a PET volume using multiple `MultiSliceViewer` processors. The main properties of one viewer are shown on the left.

- The `VolumeTransformation` processor modifies the 4*x*4 transformation matrix of the volume on its inport using the decorator concept discussed in Section 4.4.1.
- The `InteractiveRegistrationWidget` processor renders circular overlays that allow the user to manually coregister the data sets: Grabbing and dragging the circle rotates the moving volume around the center, dragging the center moves the widget (and therefore the center of rotation) and dragging the crossed-arrow symbol translates the moving volume accordingly.
- The `RegistrationInitializer` roughly aligns both volumes based on their bounding-boxes (triggered by a `ButtonProperty`).
- The `MutualInformationRegistration` processor uses the ITK library to perform an automatic mutual-information based coregistration of both data sets using the current matrix as starting point. The user can choose from a number of metrics.

The transformation matrix is stored in a `FloatMat4Property` which is linked between all of these processors. Saving the result can either be performed by using the property-widget to save just the matrix or by saving the transformed moving volume using the `VolumeSave` processor.

4.6 Region of Interest Rendering and Analysis

Analyzing the volume or intensity-distribution in a region of interest (ROI) is an essential tool for many medical applications [PB07]. Examples include the volume of a tumor, the distance of pathological structures to risk structures or a comparison of the PET activity in different organs. In this section, the design and implementation of a framework to edit and analyze 2D and 3D ROIs [*] in Voreen will be discussed.

4.6.1 Data Structures

The base class of all ROI objects is `ROIBase`, which defines the basic functionality of a ROI:

```
class ROIBase : public PropertyOwner {
public:
    ROIBase(Grid grid);
```

[*] 3D Regions of interest are sometimes called Volumes of Interest (VOI). We use ROI for both 2D and 3D since it is more commonly used.

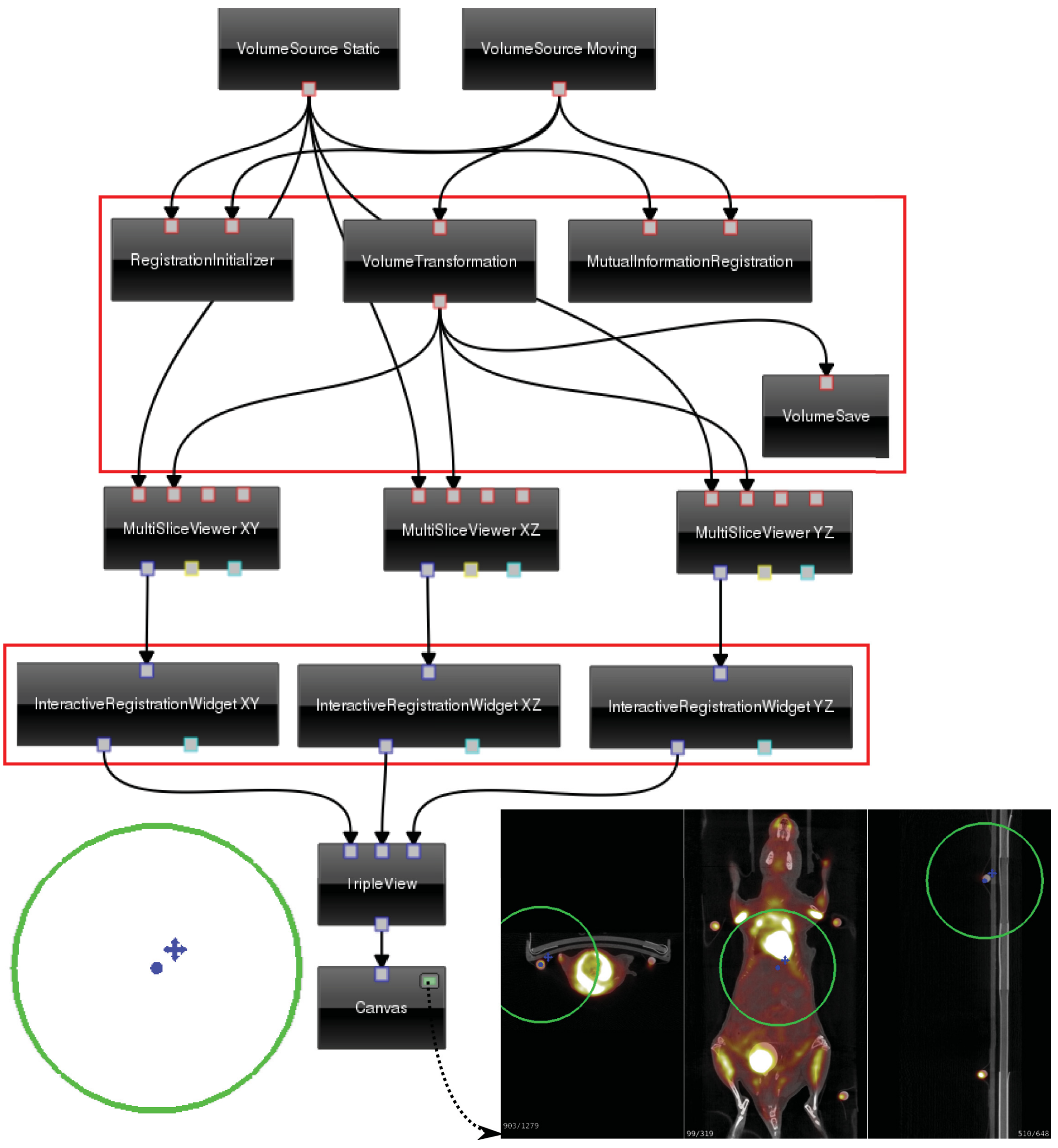


Figure 4.14: Interactive coregistration network: A combination of multimodal slice viewers with some specific registration processors (red) results in an interactive coregistration tool. The widget (bottom left), which is rendered by the `InteractiveRegistrationWidget` processors, allows the user to translate and rotate the moving data set in each view.

```
    // Get bounding box (in physical coordinates)
    Bounds getBoundingBox() const;

    // get sub-ROIs
    vector<const ROIBase*> getChildren() const;

    /// Test if p (in physical coordinates of the ROI) is inside this ROI.
    bool inROI(vec3 p) const;

    // Methods for 3D rendering:
    MeshListGeometry* getMesh() const;
    MeshListGeometry* getRasterMesh() const;

    // Methods for 2D rendering:
    MeshListGeometry* getMesh(plane pl) const;
    MeshListGeometry* getRasterMesh(plane pl) const;

    // Methods for user interaction through control points:
    vector<const ControlPoint*> getControlPoints() const; // for 3D rendering
    vector<const ControlPoint*> getControlPoints(plane pl) const; // for 2D rendering
    bool moveControlPoint(const ControlPoint* cp, vec3 offset);
private:
    StringProperty name_;
    StringProperty comment_;
    BoolProperty isVisible_;
    FloatVec4Property color_;

    Grid grid_;
};
```

Listing 4.4: `ROIBase` Interface (abbreviated)

Rendering of ROIs is performed by calling `getMesh()` and rendering the resulting `MeshListGeometry`. To visualize the extent of the ROI in its grid the `getRasterMesh()` method can be used. Generated meshes are cached internally for improved performance.

Figure 4.15 shows a hierarchy of ROI classes: Classes that combine several sub-ROIs are derived from the `ROIAggregation` class, whereas simple types are derived from `ROISingle`. These are `ROIRaster` (for rasterized ROIs generated by painting or segmentation), `ROIGraph` (for graphs like centerlines of vessel trees) and geometric ROIs `ROISphere, ROICube, ROICylinder`. `ROINormalizedGeomtry` is the super class for all geometric ROIs and simplifies the implementation because only methods for a geometry with unit size have to be implemented.
To store a number of ROIs the class `ROICollection` has been implemented.

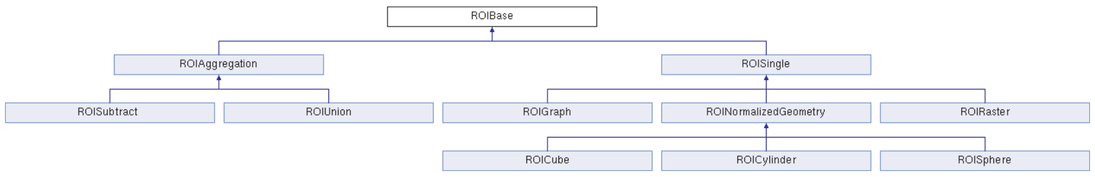


Figure 4.15: ROI classes derived from `ROIBase`.

4.6.2 Integration into Dataflow-Network

Integrating ROIs into Voreen is not as straightforward as adding a new port type. This is because multiple processors (e.g., three 2D renderers and one 3D renderer) need read and write access to a set of ROIs, but inports can only return `const` pointers. We therefore introduced the concept of a *storage processor*, which is a processor that stores data that needs to be modified by multiple processors but is too large to be handled as a property. A storage processor can be accessed by other processors using a coprocessor port and also serializes the data it stores.

The `ROIStorage` processor stores a `ROICollection` which can be accessed by processors connected to its coprocessor outport (type `GenericCoProcessorPort-<ROIStorage>`) using its `getROIs()` method. All modifications to the collection must use the `ROIStorage` interface because `getROIs()` returns a `const` pointer to the stored collection. This allows centralized undo-functionality and correct invalidation of connected processors.

Rendering Processors

Rendering of ROIs is performed using the `ROIRenderer2D` and `ROIRenderer3D` processors which combine a DVR or slice rendering with the currently visible ROIs. For 3D rendering, `getMesh()` is called and the mesh is rendered, followed by a compositing with the incoming rendering. The 2D renderer instead calls `getMesh(plane pl)`, where the plane is given by a property which is usually linked to a slice renderer connected to the inport. For each ROI the returned mesh is first rendered to a temporary buffer, followed by an enhancement of the outer edges and compositing with the input image.
In addition both renderers display a set of control points for each ROI, given by `getControlPoints()`. The user can then modify the ROIs by moving these points.
To create new ROIs a set of `ROITools` has been implemented:

- The `ROIToolGeometry` creates geometric objects in 2D and 3D.
- The `ROIToolGraph` allows the user to manually create `ROIGraph` objects by tracing a path.
- Using the `ROIToolPainter` the user can create `ROIRaster` objects. Different 2D and 3D brushes are supported and can be scaled in millimeter or voxel. The user can also modify existing ROIs using the paint or erase mode by selecting a ROI in the widget and then using the tool. In case the selected ROI is not a `ROIRaster`, a new `ROIUnion` or `ROISubtract` is automatically created, and the original ROI as well as a modifying `ROIRaster` are registered as sub-ROIs.

The active tool can be selected and configured (using properties) in the `ROIStorage` processor widget.

Statistics Processors

One important application of the framework is to compute statistics on ROIs. The most obvious application is the computation of basic statistics on a scalar volume inside a ROI, but the framework is designed to be extensible. We therefore separated the statistics algorithms from the storage processor using a coprocessor port (type `GenericCoProcessorPort<ROIStatistics>`). All attached processors therefore implement the `ROIStatistics` interface:

```cpp
class ROIStatistics : public Processor {
public:
    ROIStatistics();

    // Statistics provided by this processor:
    vector<string> getAvailableStatistics() const;
    // Calculate statistics for a given ROI:
    void calculateStatistics(const ROIBase* roi) const;
protected:
    GenericCoProcessorPort<ROIStatistics> outport_;
};
```

Listing 4.5: The `ROIStatistics` interface.

Computed statistics are stored in the ROI object and are automatically invalidated when the ROI changes. The data on which to compute the statistics is determined by the inports of the statistics processor. The `ROIStatisticsBasic` processor implements the most common statistics on the scalar volumes connected to its inport:

- Minimum, maximum, sum
- First quartile, median, third quartile, inter-quartile range

- Mean, standard deviation, variance
- Number of voxels, volume

An extension to other types of data (geometry, fibers) requires a new statistics processor with a matching inport.

Segmentation Processors

All processors that perform computations on a selection of ROIs implement the `ROISegmenter` interface and are connected to the `ROIStorage` processor similar to statistics processors.

```
class ROISegmenter : public Processor {
public:
    ROISegmenter();

    // Is this segmenter compatible with the currently selected ROIs?
    virtual bool isCompatible(set<ROIBase*> rois);
    // Set input data:
    virtual void setInput(set<ROIBase*> rois, ROIStorage* storage);
    // Perform the segmentation:
    virtual void segment();

    // Methods to display segmentation preview:
    virtual const MeshListGeometry* getMesh() const;
    virtual MeshListGeometry* getMesh(plane pl) const;
    virtual MeshListGeometry* getRasterMesh(plane pl) const;

    // Segmenter is considered active when its property dialog is shown:
    virtual void activate();
    virtual void deactivate();
    bool isActive() const;

protected:
    GenericCoProcessorPort<ROISegmenter> outport_;
};
```

Listing 4.6: The `ROISegmenter` interface

Segmentation processors implemented so far are:

- `ROIThresholder`: Adds a `ROIRaster` containing all voxels above or below a given threshold (absolute or relative to the maximum) in a connected volume (with an optional container ROI).
- `ROIRegionGrowing`: Performs a region growing from a seed ROI, which can be limited to a container ROI. Connectivity and strictness are configurable.

- `ROIFindExtrema`: Find the minimum or maximum in a volume (with optional container ROI). Places a sphere of configurable size (in millimeter or voxel) at this position.
- `ROIMorphology`: Applies erosion/dilation/open/close with a configurable structuring element to a ROI. The result is added as new `ROIRaster`.

Figure 4.16 shows an example workflow for region growing, while Figure 4.17 shows a network using a combination of all described ROI processor types with a 2D/3D rendering network.

4.7 Plotting

Voreen has been extended to display non-spatial data using the plotting module.

4.7.1 Data Structures

The base class underlying the data transferred via `PlotPorts` is `PlotBase`, which is sub-classed by `PlotFunction` and `PlotData`. `PlotData` is a table of scalars or strings, while `PlotFunction` stores functions to be plotted. Columns in both data structures can be assigned labels and a color hint so that display is consistent (e.g., a plot of a histogram matches the color of the ROI it was calculated for, see Figure 4.18).

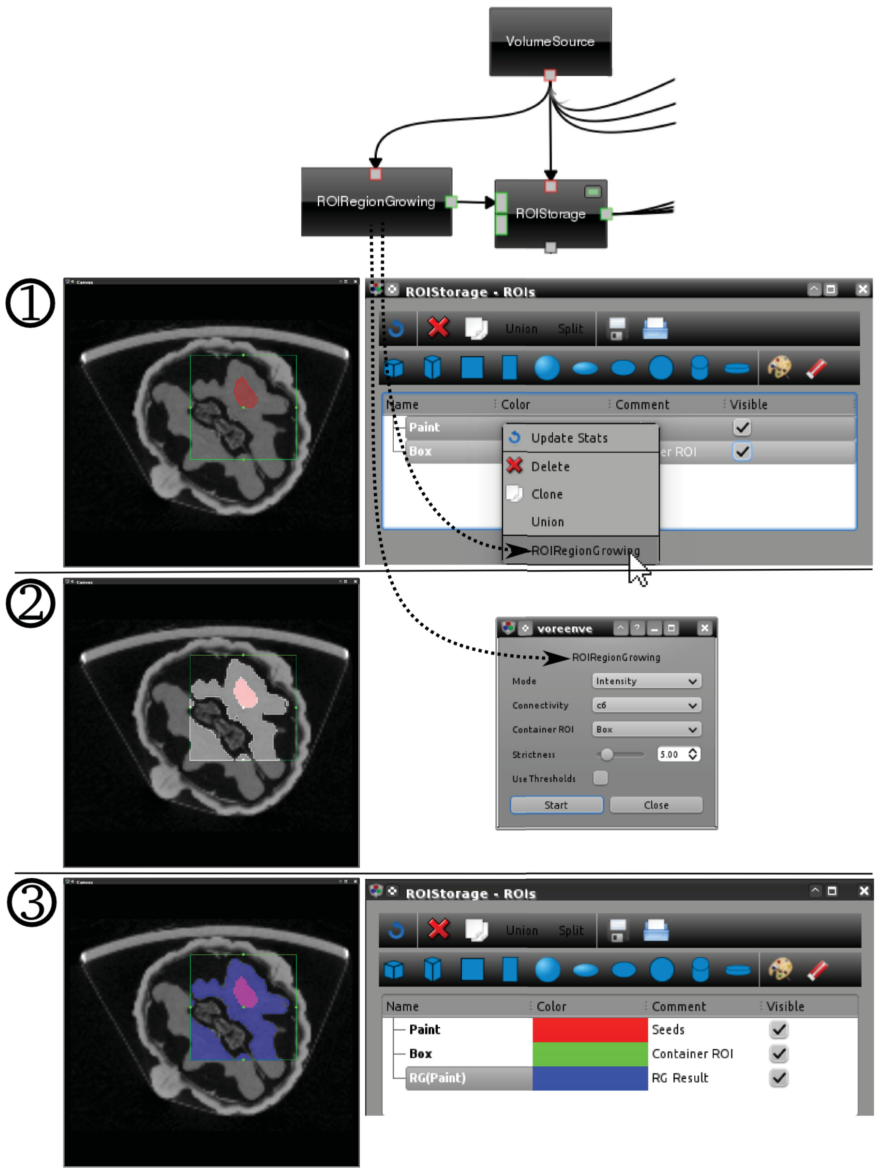
4.7.2 Plotting Processors

Processors in the `plotting` module can be categorized into three groups:

- **Input/Output:** `PlotDataSource` and `PlotDataExport` allow loading/saving from/to comma separated value (.csv) files.
- **Processing:** Several processors to process `PlotData` objects (select, merge, function-fitting, etc.) have been implemented.
- **Rendering:** `LinePlot`, `BarPlot`, `ScatterPlot` and `SurfacePlot` render plots of the respective type and allow exporting to SVG files.

4.7.3 Linking and Brushing

The concept of linking and brushing is to combine multiple visualization techniques using interaction. Selected items in one view are highlighted in other views and

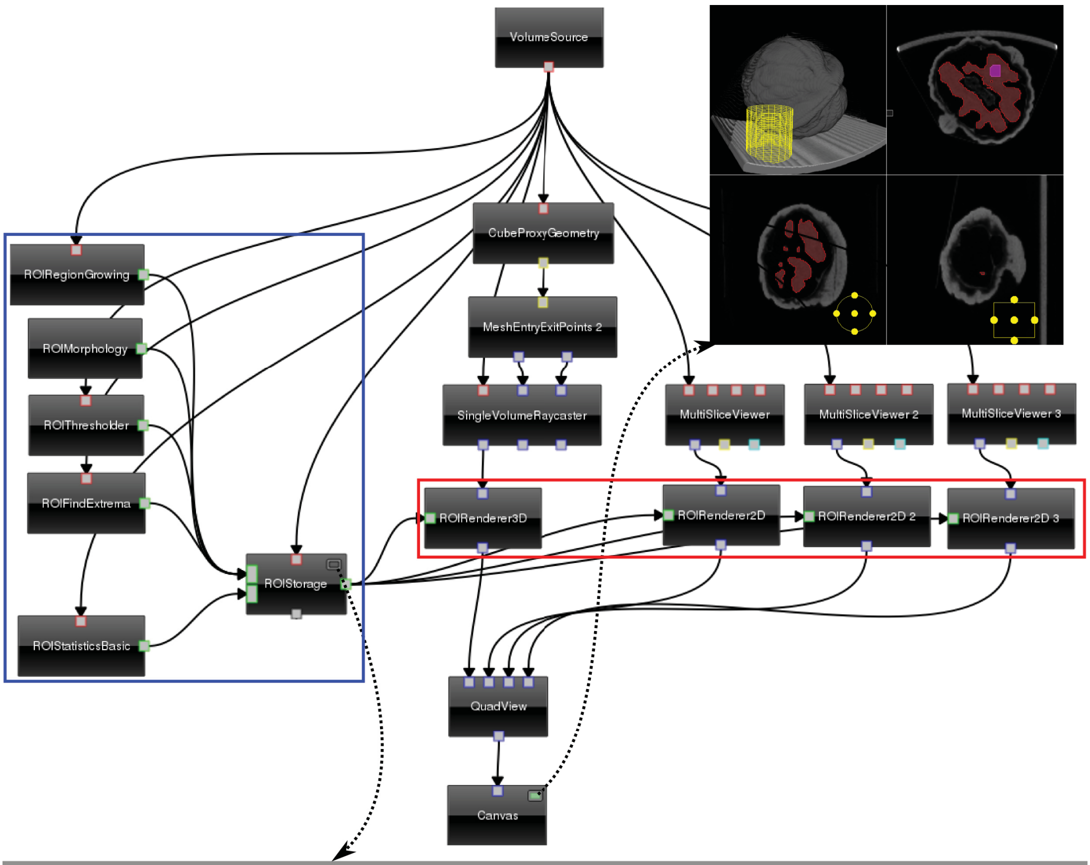


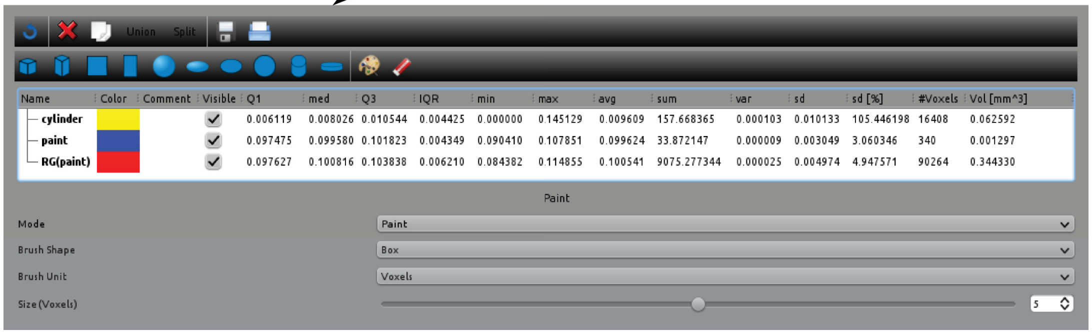
Name	Color	Comment	Visible
Paint			✓
Box		er ROI	✓

Mode	Intensity
Connectivity	c6
Container ROI	Box
Strictness	5.00
Use Thresholds	☐

Name	Color	Comment	Visible
Paint	🟥	Seeds	✓
Box	🟩	Container ROI	✓
RG(Paint)	🟦	RG Result	✓

Figure 4.16: Segmentation workflow using the ROI framework: The user selects a seed and container ROI and activates the context menu (1). Connected segmentation processors that are compatible with the current selection are displayed. Upon activation of a segmentation processor a window showing its properties is displayed and a preview of the result is overlayed (white) in the rendering(2). The resulting segmentation can be discarded or added to the current set of ROIs (3).



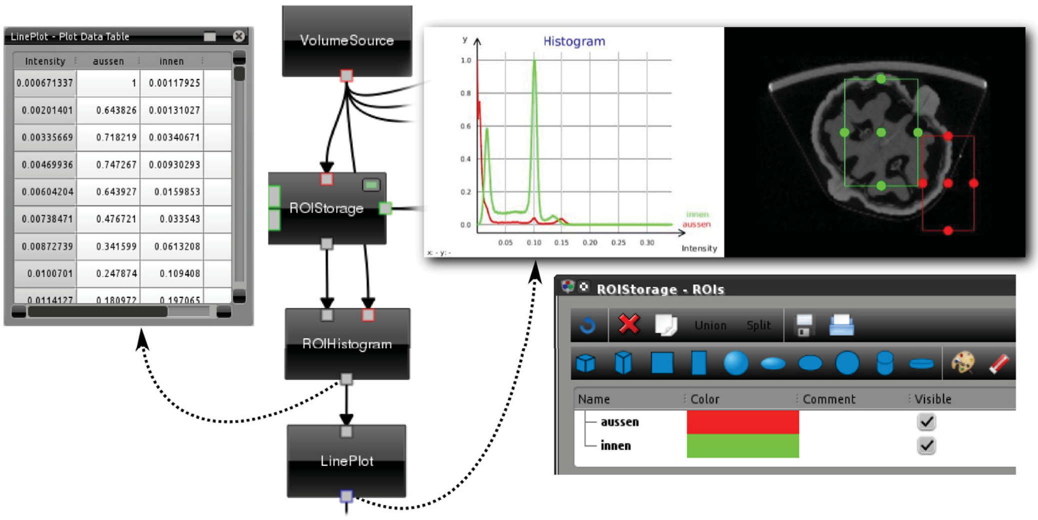


Name	Color	Comment	Visible	Q1	med	Q3	IQR	min	max	avg	sum	var	sd	sd [%]	#Voxels	Vol [mm^3]
cylinder	🟨		✓	0.006119	0.008026	0.010544	0.004425	0.000000	0.145129	0.009609	157.668365	0.000103	0.010133	105.446198	16408	0.062592
paint	🟦		✓	0.097475	0.099580	0.101823	0.004349	0.090410	0.107851	0.099624	33.872147	0.000009	0.003049	3.060346	340	0.001297
RG(paint)	🟥		✓	0.097627	0.100816	0.103838	0.006210	0.084382	0.114855	0.100541	9075.277344	0.000025	0.004974	4.947571	90264	0.344330

Paint

Mode	Paint
Brush Shape	Box
Brush Unit	Voxels
Size (Voxels)	5

Figure 4.17: Basic ROI network consisting of standard 2D/3D rendering processors in combination with ROI rendering processors (red) and ROI storage/segmentation/statistics processors (blue). The processor-widget of the ROIStorage processor provides the user with a selection of tools, a list of properties with their statistics and properties of the currently active tool (top to bottom).



Intensity	aussen	innen
0.000671337	1	0.00117925
0.00201401	0.643826	0.00131027
0.00335669	0.718219	0.00340671
0.00469936	0.747267	0.00930293
0.00604204	0.643927	0.0159853
0.00738471	0.476721	0.033543
0.00872739	0.341599	0.0613208
0.0100701	0.247874	0.109408
0.0114127	0.180972	0.197065

Name	Color	Comment	Visible
aussen	🟥		✓
innen	🟩		✓

Figure 4.18: Plotting in combination with ROIs: The `ROIHistogram` processor calculates histograms on ROIs and generates a `PlotData` table which is rendered by the `LinePlot` processor.

enable the user to detect dependencies and correlations [K$^+$02].

In the implemented plotting framework, selections of data through user interaction are saved in a `PlotSelectionProperty`. Linking and brushing functionality can be created by using property-linking as follows: The selection property of a `LinePlot` processor displaying a histogram could be linked to modify the domain of a transfer-function property elsewhere in the network.

4.8 Reformation Techniques

In this section we will discuss the implementation of the straigthened curved planar reformation (CPR, see Section 2.1); other variants of this technique and multiplanar reconstructions (MPR) can be implemented in a very similar way. The CPR needs a centerline of a vessel, which is specified by creating a `ROIGraph` object. This can be achieved by manually tracing the vessel using the `ROIToolGraph` or using the `ROISkeletonize` processor, which implements the `ROISegmenter` interface and performs a skeletonization of a ROI. To select the centerline to be used for the CPR we connect a `ROIGraphSelector` to the `ROIStorage`. The `CPRProcessor` uses the resulting point list to create a proxy-geometry with correct texture coordinates. The

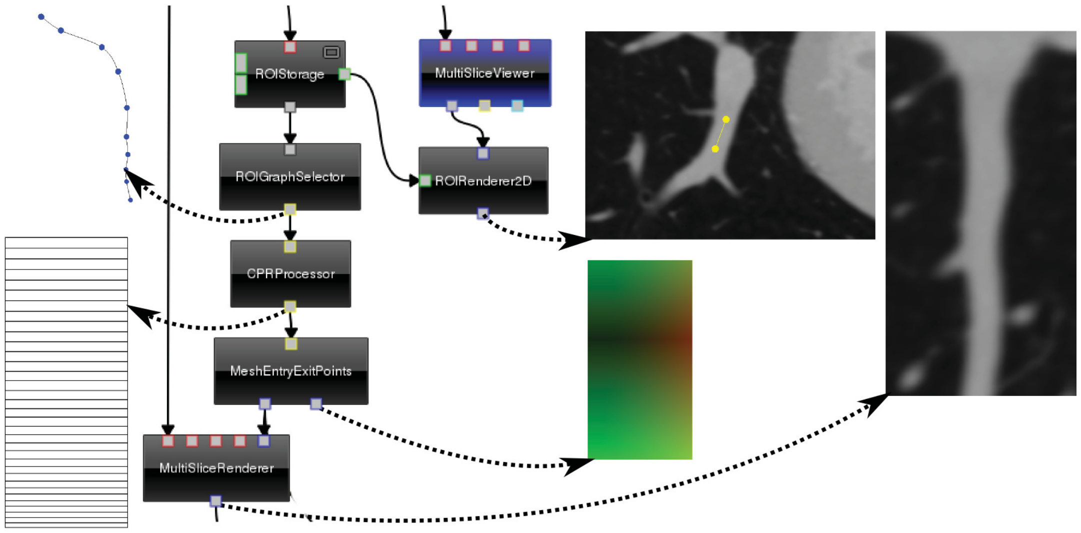


Figure 4.19: CPR in Voreen: A `ROIGraph` object is selected from the available ROIs using the `ROIGraphSelector` processor. The resulting point-list is used by the `CPRProcessor` to generate a proxy-geometry which is rendered by the `MeshEntryExitPoints` renderer. To create the final rendering, a `MultiSliceRenderer` is used.

type of CPR and the vector of interest can be specified using properties. Similar to multimodal slice rendering, the `MeshEntryExitPoints` and `MultiSliceRenderer` processors are used to perform the color mapping. A network performing CPR rendering is shown in Figure 4.19.

4.9 Volume Processing

Filtering and segmentation are parts of a typical volume visualization pipeline [PB07], but for a long time Voreen development has focussed mostly on the rendering part. Therefore, recently the Insight Segmentation and Registration Toolkit (ITK) has been integrated into Voreen. ITK is an open-source, cross-platform library that provides developers with an extensive suite of algorithms for image analysis [*]. Similar to Voreen's dataflow network consisting of processors, ITK uses pipelines of *filters*. Although no property-mechanism is implemented in ITK, the getters and setters for parameters of filters are named consistently (i.e., `GetLowerThreshold()` and `SetLowerThreshold()`). Algorithms and data structures in ITK are generally

[*] http://www.itk.org

templated to support all data types and dimensions [ISN+03]. Unlike the `Volume` or `Processor` classes in Voreen, there are no non-templated base classes, which means the type of data has to be known at compile-time. The following code reads a float volume from disk and applies a thresholding filter to it:

```
// Input and output are floating point volumes:
typedef itk::Image<float, 3> InputImageType;
typedef itk::Image<float, 3> OutputImageType;

// Configure the reader:
typedef itk::ImageFileReader<InputImageType>  ReaderType;
ReaderType::Pointer reader = ReaderType::New();
reader->SetFileName(argv[1]);

// Configure and execute the filter:
typedef itk::BinaryThresholdImageFilter<InputImageType, OutputImageType>  FilterType;
FilterType::Pointer filter = FilterType::New();
filter->SetInput(reader->GetOutput());
filter->SetLowerThreshold(atof(argv[3]));
filter->SetUpperThreshold(atof(argv[4]));
filter->Update();
```

Listing 4.7: Simplified excerpt from ITK source code (Examples/Filtering/BinaryThresholdImageFilter.cxx)

To wrap this functionality in Voreen we need a processor with a volume inport and outport as well as two properties for the thresholds. Since ports and processors in Voreen are not typed, the voxel-type of the input volume has to be determined, and the correct template is instantiated. This is the resulting processor:

```
class BinaryThresholdImageFilterITK : public ITKProcessor {
public:
    BinaryThresholdImageFilterITK();

    std::string getCategory() const   { return "Volume␣Processing/Filtering/Thresholding"; }
    std::string getClassName() const  { return "BinaryThresholdImageFilterITK";  }
protected:
    void process();

    // Templated function using ITK code to perform the thresholding:
    template<class T>
    void binaryThresholdImageFilterITK();
private:
    VolumePort inport1_;
    VolumePort outport1_;

    // Property adapting to the volume type (float, uint8, ...):
    VoxelTypeProperty lowerThreshold_;
    VoxelTypeProperty upperThreshold_;
};

BinaryThresholdImageFilterITK::BinaryThresholdImageFilterITK()
```

```
    : ITKProcessor(),
    inport1_(Port::INPORT, "InputImage"),
    outport1_(Port::OUTPORT, "OutputImage"),
    lowerThreshold_("lowerThreshold", "LowerThreshold"),
    upperThreshold_("upperThreshold", "UpperThreshold"),
{
    addPort(inport1_);
    // Allow only scalar volumes:
    PortConditionLogicalOr* orCondition1 = new PortConditionLogicalOr();
    orCondition1->addLinkedCondition(new PortConditionVolumeTypeUInt8());
    orCondition1->addLinkedCondition(new PortConditionVolumeTypeUInt16());
    [...]
    inport1_.addCondition(orCondition1);
    addPort(outport1_);

    addProperty(lowerThreshold_);
    addProperty(upperThreshold_);
}

void BinaryThresholdImageFilterITK::process() {
    const VolumeRAM* inputVolume1 = inport1_.getData()->getRepresentation<VolumeRAM>();

    // Type switch:
    if (dynamic_cast<const VolumeRAM_UInt8*>(inputVolume1))
        binaryThresholdImageFilterITK<uint8_t>();
    else if (dynamic_cast<const VolumeRAM_UInt16*>(inputVolume1))
        binaryThresholdImageFilterITK<uint16_t>();
    [...]
    else
        LERROR("Inputformat␣of␣Volume␣1␣is␣not␣supported!");
}

template<class T>
void BinaryThresholdImageFilterITK::binaryThresholdImageFilterITK() {
    typedef itk::Image<T, 3> InputImageType1;
    typedef itk::Image<T, 3> OutputImageType1;

    // Create image in ITK:
    typename InputImageType1::Pointer p1 = voreenToITK<T>(inport1_.getData());

    // Perform the filtering:
    typedef itk::BinaryThresholdImageFilter<InputImageType1, OutputImageType1> FilterType;
    typename FilterType::Pointer filter = FilterType::New();

    filter->SetInput(p1);
    filter->SetLowerThreshold(lowerThreshold_.getValue<T>());
    filter->SetUpperThreshold(upperThreshold_.getValue<T>());
    filter->Update();

    // Set result in Voreen:
    outport1_.setData(ITKToVoreenCopy<T>(filter->GetOutput()));
}
```

Listing 4.8: ITK's BinaryThresholdImageFilter wrapped in a Voreen processor.

Although this code is quite similar to the example discussed earlier it is certainly not desirable to write and maintain a large (and still growing) number of wrapper-processors for each filter in ITK. Therefore a semi-automatic wrapping process has been developed. To wrap a filter, only a few lines of XML-code which set the name, ports and properties have to be written. The *itkwrapper* application will generate the discussed wrapper-processor from the following XML code:

```
<filter name="BinaryThresholdImageFilter">
    <arguments>
        <argument name="LowerThreshold" argumenttype="PixelType"/>
        <argument name="UpperThreshold" argumenttype="PixelType"/>
    </arguments>
</filter>
```

Listing 4.9: XML-code needed to generate a Voreen processor wrapping ITK's Binary-ThresholdImageFilter.

Ports do not have to be configured in this case because the wrapper assumes one volume inport and one volume outport as default. itkwrapper generates a new module `itk_generated` which contains all auto-wrapped processors. Currently around 150 filtering and segmentation processors can be generated. For those filters for which the automatic process fails the resulting code can often be slightly modified to work. Since the coregistration framework in ITK is very different from the filters it has been manually wrapped in the `MutualInformationRegistration` processor. Manually developed processors are located in the `itk` module.

Chapter 5

Case Study: Analysis of DTI Data using Voreen

In this chapter we are going to discuss an application for the analysis of DTI data. The application features no new visualization techniques but illustrates how the rapid development capabilities provided by Voreen can be used to create a streamlined workflow for specific tasks.

5.1 Background and Data Acquisition

Fractional anisotropy, axial and radial diffusivity, and fiber reconstructions allow for better differentiation of several aspects of neuropathology: inflammation, demyelination, and axonal loss in small animal models, e.g. of Multiple Sclerosis. Images are usually acquired with scan times ranging from 2 - 3 h (in vivo) up to 28 h (ex vivo), precluding studies with severely impacted animals or large cohorts of animals. Here, we exploited the gain in signal-to-noise ratio (SNR) by using a cryogen-cooled, low-noise transmit/receive surface coil at 9.4 T to implement a high-resolution DTI-EPI protocol with scan times reasonable for in vivo imaging. A scan protocol of less than one hour is more likely to be tolerated by very sick animals and would be feasible for longitudinal imaging studies.

Two groups of mice, at 4 weeks of 0.2% Cuprizone diet (n=5) and age-matched controls (n=3) were scanned at 9.4 T (Bruker Biospec) with the Bruker MRI CryoProbe. Mice were anesthetized with 1-1.5% Isoflurane. A four-shot DTI-EPI sequence with respiratory gating was used: matrix = 256*256; field of view = $2x2cm^2$; slice thickness = $300\mu m$; 12 slices; TR/TE = 3500/58 ms; 2 averages. 5 B0 and 60 diffusion-weighted images (b=$1000s/mm^2$) in 30 directions were obtained over approximately 30 min [WOS$^+$11].

5.2 Requirements

The application should read diffusion weighed images as generated by the scanner, perform a tensor estimation followed by an analysis of the diffusivity in a user-specified ROI. Standard slice views and fiber tracking should be used to visualize the results. A special requirement for this application is the capability to exclude selected DWI volumes from the tensor estimation. Due to a problem with the scanner these volumes contain unusable slices which would decrease the quality of the DTI volume.

5.3 Implementation

Data Import

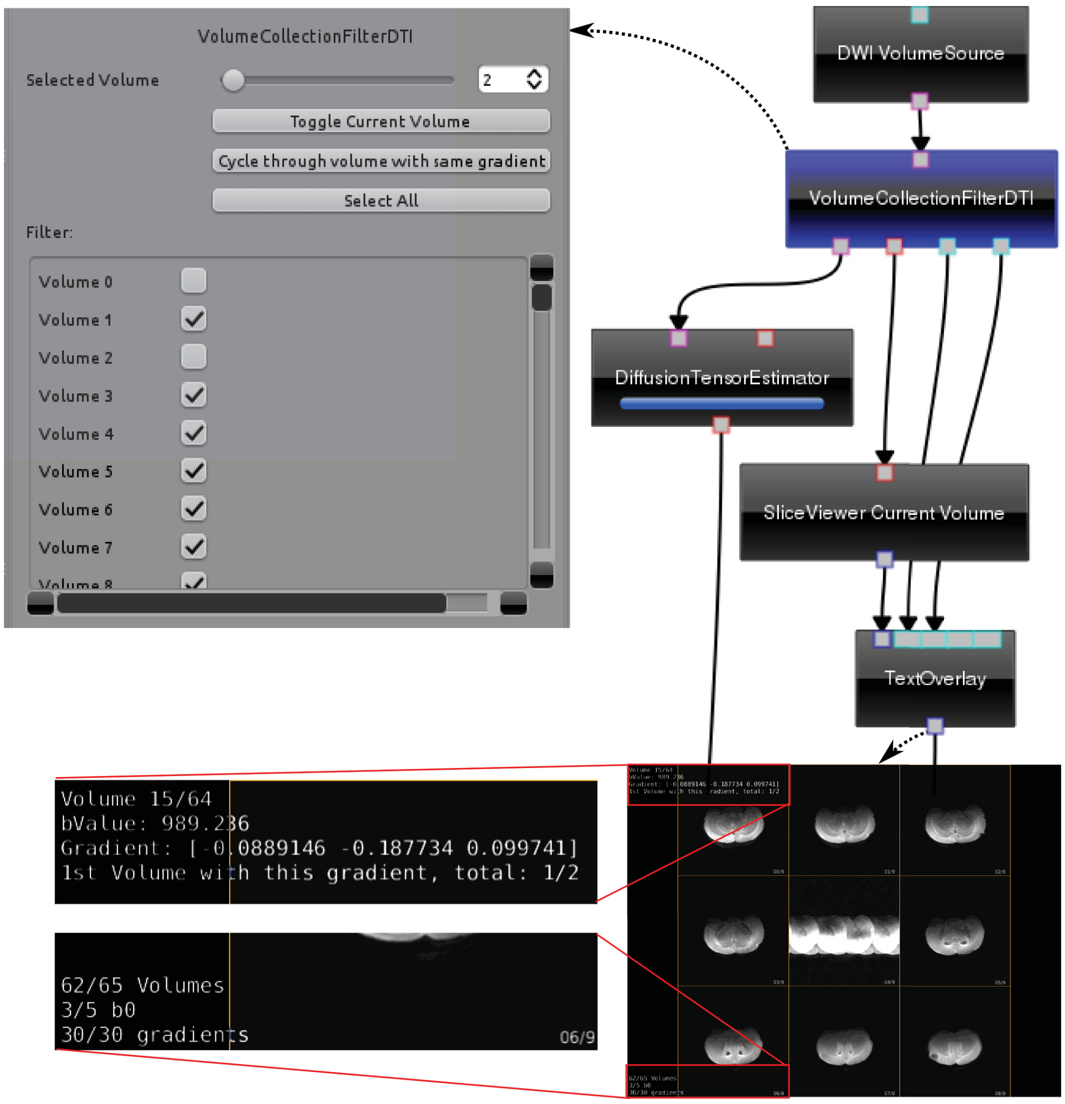
The `BrukerVolumeReader` is used to import the DWIs stored in Bruker ParaVision format, it supports reading of the necessary metadata for further processing.

Filtering of DWI Volumes

To filter corrupt DWI volumes the `VolumeCollectionFilterDTI` has been implemented (see Figure 5.1). The user can cycle through all DWI volumes using the mouse wheel, which modifies the "Selected Volume" property. The selected volume is forwarded on one of the outports and displayed using a standard sliceviewer, configured to display all slices at once. A vector of boolean properties controls which of the volumes in the collection should be forwarded. The state of the current volume can be toggled using a keyboard shortcut which activates a button property. The filtered volume collection is forwarded on an outport, which will usually be connected to a tensor estimation processor. One text outport contains information about the current volume and another about the number of selected volumes. Using property links between the `VolumeCollectionFilterDTI` and the slice viewer the color of the slice grid is modified to indicate the state of the current gradient volume: A green grid indicates that all volumes with this gradient are forwarded. An orange grid (see Figure 5.1) is displayed when the current volume is filtered but other volumes with this gradient are forwarded. If all volumes using one gradient are filtered the grid becomes red.

Tensor Estimation

Tensor estimation is performed using the `DiffusionTensorEstimator` or `ModelFitCamino` processors.


VolumeCollectionFilterDTI
Toggle Current Volume
Cycle through volume with same gradient
Select All
Filter:
DWI VolumeSource
VolumeCollectionFilterDTI
DiffusionTensorEstimator
SliceViewer Current Volume
TextOverlay
06/9

Selected Volume 2

- Volume 0 ☐
- Volume 1 ✓
- Volume 2 ☐
- Volume 3 ✓
- Volume 4 ✓
- Volume 5 ✓
- Volume 6 ✓
- Volume 7 ✓
- Volume 8 ✓

```
Volume 15/64
bValue: 989.236
Gradient: [-0.0889146 -0.187734 0.099741]
1st Volume with this gradient, total: 1/2
```

```
62/65 Volumes
3/5 b0
30/30 gradients
```

Figure 5.1: Filtering of DWI Volumes using the `VolumeCollectionFilterDTI`.

Masking of DTI Volume

The tensor volume is masked based on a threshold applied to a b0 volume. A closing operation can be applied to the mask, and a preview of the mask is displayed as outline on the b0 image.

ROI Analysis

Statistics for a user-specified ROI are computed for a set of scalar values (default: mean/axial/radial diffusivity). For further analysis the computed statistics are exported to a CSV file using the `PlotDataExport` processor.

Fibertracking

Fibertracking is seeded in the specified ROI and rendered using a `FiberRenderer` in combination with a slice to provide context.

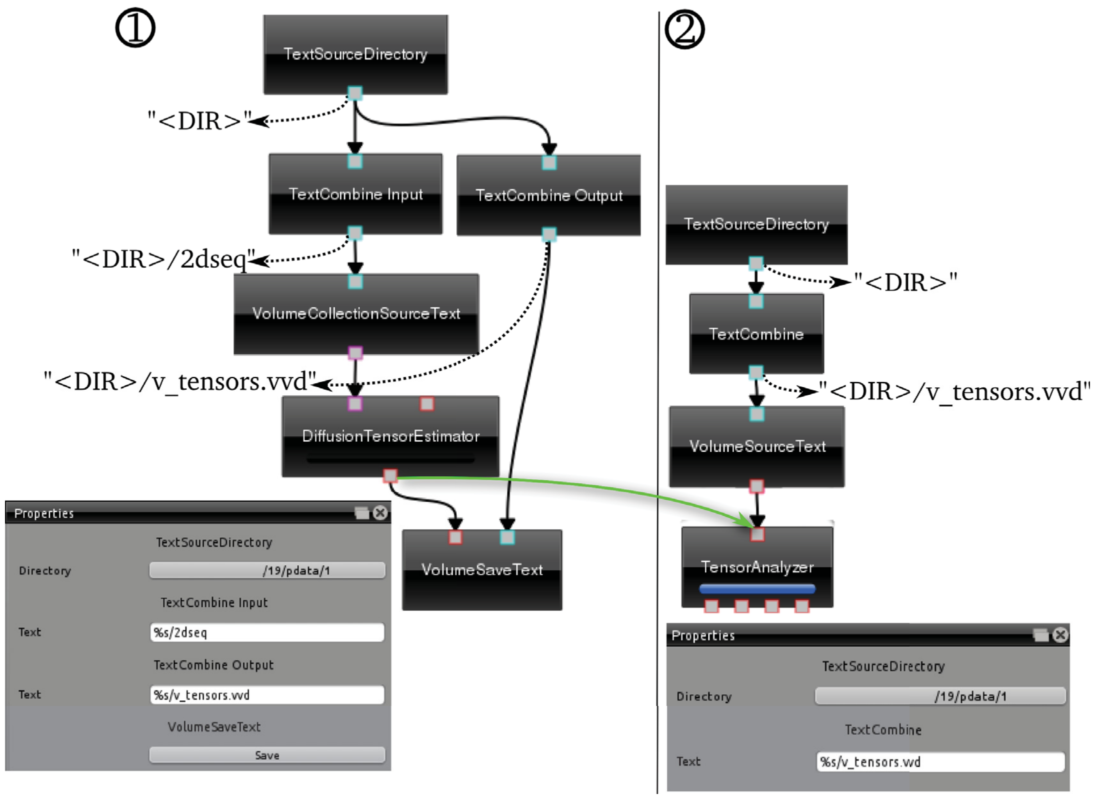
Workflow

The analysis-workflow is performed in two steps/networks: The DWI-filtering, tensor estimation and masking is performed in the first step (see Figure 5.3). ROI specification and analysis as well as exporting of results form the second step (see Figure 5.4).

Between both steps the masked DTI volumes have to be transfered, and analysis results have to be saved after step two. In order to achieve these objectives with minimal user interaction we utilize a set of processors from the `workflow` module. A sub-network showing the functionality is displayed in Figure 5.2: The user selects a directory containing one scan using the `TextSourceDirectory` processor. The directory is set in the text-outport and the following `TextCombine` processors concatenate filenames for input (the "2dseq" file contains the reconstructed scan in ParaVision format) and output files ("v_tensors.vvd"). Using a similar construction in step two, the tensor volume will be loaded, creating a connection between the tensor estimation in step one and the tensor analyzer in step two. Results of the analysis are saved to the same directory in the same way.

5.4 Conclusion

We have shown how to create a custom application performing a streamlined workflow by using the VoreenVE application. Due to the high degree of modularity, the



Properties

TextSourceDirectory

Directory /19/pdata/1

TextCombine Input

Text %s/2dseq

TextCombine Output

Text %s/v_tensors.vvd

VolumeSaveText

Save

Properties

TextSourceDirectory

Directory /19/pdata/1

TextCombine

Text %s/v_tensors.vvd

Figure 5.2: Data Transport across networks using processor from the `workflow` module.

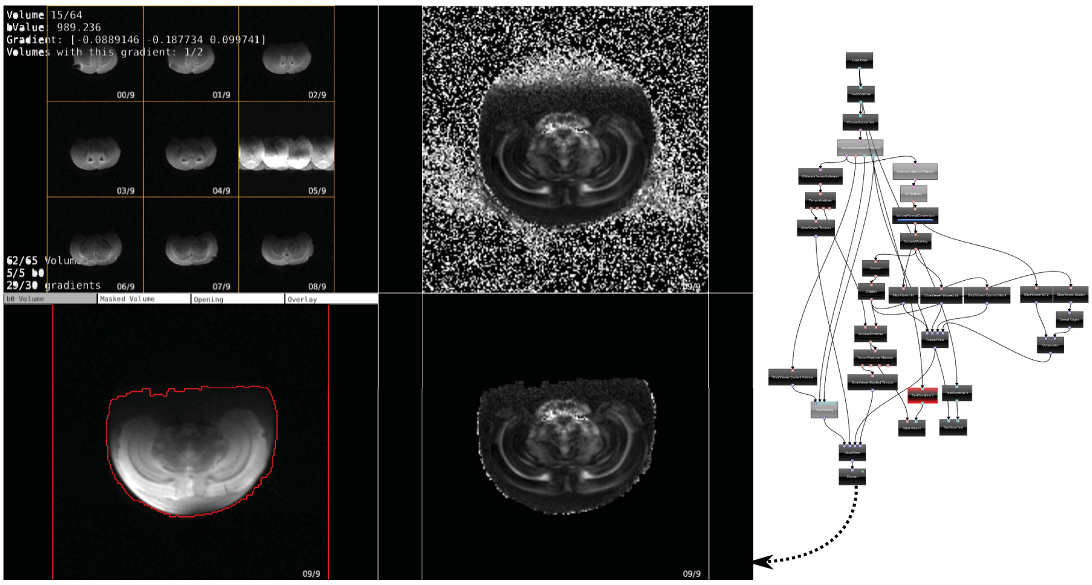

Volume 15/64
bValue: 989.236
Gradient: [-0.0889146 -0.187734 0.099741]
Volumes with this gradient: 1/2
62/65 Volumes
5/5 b0
29/30 gradients

Figure 5.3: First Workflow Step: DWI-filtering (top left), results of tensor estimation (fractional anisotropy (FA), top right), masking preview (bottom left), masked tensors (FA, bottom right).

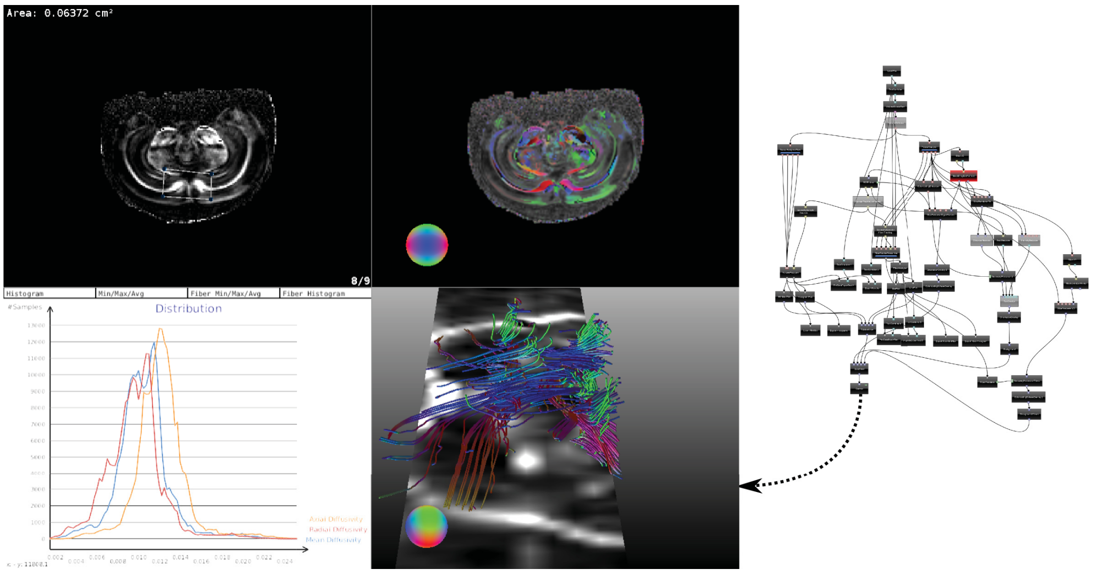

Area: 0.06372 cm²

Figure 5.4: Second Workflow Step: ROI specification on FA volume (top left), main eigenvector direction combined with FA (top right), plots showing the distribution of diffusivity values in the ROI (bottom left), fiber rendering combined with 3D-slice (bottom right).

only code specific to this application is the `VolumeCollectionFilterDTI` processor, which is easy to implement.
In agreement with published studies, increased radial diffusivity could be found in the corpus callosum of Cuprizone-fed mice, indicating demyelination [WOS$^+$11].

Chapter 6

Context-Aware Volume Navigation

The trackball metaphor is exploited in many applications where volumetric data needs to be explored. Although it provides an intuitive way to inspect the overall structure of objects of interest, an in-detail inspection can be tedious - or when cavities occur even impossible. Therefore we propose a context-aware navigation technique for the exploration of volumetric data. While navigation techniques for polygonal data require information about the rendered geometry, this strategy is not sufficient in the area of volume rendering. Since rendering parameters, e.g., the transfer function, have a strong influence on the visualized structures, they also affect the features to be explored. To compensate for this effect we propose a novel image-based navigation approach for volumetric data. While being intuitive to use, the proposed technique allows the user to perform complex navigation tasks, in particular to get an overview as well as to perform an in-detail inspection without any navigation mode switches. The technique can be easily integrated into ray casting based volume renderers like Voreen, needs no extra data structures and is independent of the data set as well as the rendering parameters. We will discuss the underlying concepts, explain how to enable the navigation at interactive frame rates using OpenCL, and evaluate its usability as well as its performance.

6.1 Introduction

In recent years several algorithms have been proposed which accelerate volume rendering and thus allow interactive frame rates. As a consequence, the user cannot only change rendering parameters interactively, but is also able to navigate within volumetric data. Although navigation is essential in order to get a deeper

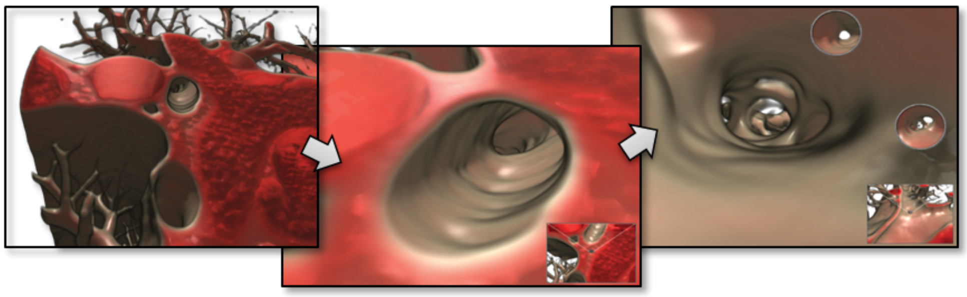


Figure 6.1: Three subsequent screenshots as made during the usage of our image-based volume navigation metaphor. Without changing the navigation mode, the user is able to inspect the data set in a behavior similar to a trackball and can also fly through internal structures when a collision detection is present. To support the spatial-awareness, appropriate thumbnails are displayed.

understanding of the visualized data sets and the need for navigation metaphors has been expressed [CON08], only little research has been dedicated to support this process within volume visualizations. In fact, in most general-purpose volume visualization systems the trackball metaphor is exploited [Sho92], since it is easy to use and allows predictable navigation [BRP05]. However, the trackball metaphor has several drawbacks. First, navigation is based on the assumption that the shape of the object to be explored is (approximately) spherical, which for instance makes it hard to explore longitudinal structures. Second, in-detail inspections are difficult, since the trackball is fixed to a given center. To deal with this shortcoming, it is often possible to reposition the trackball center, which, however, is tedious and in many cases counterintuitive. Third, the trackball metaphor is not location-aware and does not take visibility into account. This is especially problematic when diving into the volume, where it is hard to orient oneself, in particular in the absence of collision detection. Therefore specialized navigation techniques have been developed for application cases where these drawbacks are limiting factors, e.g., when navigating in virtual colonoscopy [HMK+97]. This chapter proposes a general purpose volume navigation metaphor, which can be integrated into existing volume visualization systems. Besides being intuitive to use, we believe that our navigation metaphor brings forward visualization-based research, since it does not have to be adapted for specific tasks and thus is also applicable to emerging application scenarios.

A navigation task can be understood as a combination of explorative navigation and directed navigation [CON08]. While explorative navigation is a rather undirected task, where the user interactively inspects the data to gather knowledge, directed navigation supports the user when intentionally visiting structures of interest. In most application scenarios, both navigation types are desirable, and they should therefore be integrated seamlessly. An often recurring pattern, where a seamless integration is required, could be as follows. The user first inspects the whole data set in an explorative manner in order to get an overview and identify potential structures of interest, to which s/he could navigate subsequently. This scenario is also in line with the often cited *overview, zoom, filter out, details-on-demand* concept, which has been introduced by Shneiderman to describe a general visual analysis process [Shn96]. By keeping these considerations in mind and avoiding the mentioned drawbacks of the trackball metaphor, we have developed an intuitive navigation metaphor for volumetric data, which supports explorative as well as directed navigation and is context-aware, thus allowing us to avoid manual navigation mode switches and to provide contextual information to the user (see Figure 6.1). Several challenges occur when developing such a context-aware navigation metaphor for volumetric data. Unlike in scenes consisting of polygonal data, structures are not clearly defined in volumetric data. The set of visible features can be changed easily by adjusting rendering parameters such as the transfer function, clipping planes or when switching between different rendering modes. Therefore existing navigation algorithms, which are based on the assumption that scenes consist of a set of separable features, e.g., McCrae et al. [MMGK09], cannot be applied when navigating volumetric data. As a consequence, precomputed data structures would be needed for each possible set of rendering parameters which affect the visualized features. Obviously, this is not a feasible option because of the vast number of possibilities for rendering a data set. An alternative would be to recompute the data structures for each frame by considering the current set of rendering parameters. However, this would require a thorough data analysis and thus result in a significant performance drop, especially for large volume data sets. Therefore many of the specialized volume navigation techniques discussed in Section 6.2 assume a preset of rendering parameters and analyze the data set to be explored during a preprocessing step. Hence navigating arbitrary data sets with arbitrary rendering parameters is not supported.

This chapter proposes a novel image-based navigation metaphor which meets all of the requirements discussed above. By exploiting the processing power of current GPUs, we are able to analyze the environment surrounding the camera in real-time and thus to extract knowledge to support context-aware navigation, which does not require any manual navigation mode switches. To achieve this goal, we make the following contributions:

- An interactive image analysis technique based on spherical volume ray-casting, which allows our navigation metaphor to adapt to any visible structure and virtually arbitrary rendering techniques.

- A seamless integration of strafing, panning, rotating and flying, which allows convenient proximal as well as distal object inspection.
- Context-aware overlays which support orientation of the user by providing an overview of the surrounding region.

6.2 Related Work

Since our navigation technique is of interactive nature, we do not cover automatic techniques in this section, and refer to the camera survey by Christie and Olivier [CON08].

Volumetric navigation. Most navigation techniques for volumetric data are specialized for navigation in tubular structures to support various virtual endoscopy applications: Virtual angioscopy [HBA$^+$04], virtual colonoscopy [HMK$^+$97], virtual sinus endoscopy [KKPS08] and virtual bronchoscopy [BMF$^+$03]. Most of these techniques attach the virtual endoscope (i.e., the camera) to a precomputed or manually determined centerline [SLC$^+$02]. Hence these techniques cannot provide an ad-hoc general-purpose navigation for arbitrary volumetric data. Nevertheless, these applications and algorithms have to be considered when designing a flexible navigation system. Especially relevant for our approach is the active virtual angioscopy navigation technique proposed by Haigron et al. [HBA$^+$04]. The authors avoid precomputation by using techniques from the field of mobile robot navigation to automatically steer the camera through vascular structures. They analyze the depth buffer of the current view and direct the camera towards the location of the maximal depth value. In contrast to our technique the authors constrain their analysis to the current field of view, which is sufficient for path planning in tubular structures but cannot provide enough information for a more general navigation metaphor. As a consequence, for instance no collision detection can be performed when moving sideways. Serlie et al. [SVVG$^+$01] describe a virtual colonoscopy application that uses a cube map to provide a full 360-degree view and speed up rendering. To inspect volumetric data various approaches for optimal viewpoint selection have been proposed. Kohlmann et al. [KBK07] describe a technique called LiveSync to link 2D slices with volume renderings. In contrast to our approach, LiveSync relies on 2D slice representations and does not support individual camera flights. Additionally, several more automatic and thus less relevant view point determination approaches exist [VFSH01, BS05, TFTN05, VMN08]. These techniques can be used as starting point for interaction exploration. Viola et al. [VFSG06] show how to determine expressive views on predefined features, which can be selected by the user during runtime.

Polygonal navigation. Besides these camera control techniques specifically developed for volumetric data, several relevant techniques for navigating polygonal data exist. Zeleznik and Forsberg [ZF99] describe a technique called UniCam, which allows the user to perform navigation by using a gesture alphabet. While this is a promising concept, the authors report that users initially required several hours of training to get used to the technique. Our technique is inspired by the HoverCam technique proposed by Khan et al. [KKS+05]. To support proximal object inspection, a mouse drag is translated into a translation of the camera position, followed by an adjustment of the focus to the point closest to the camera. However, there are several differences between our technique and the HoverCam. First, HoverCam requires an indexing structure called sphere-tree which has to be extracted from the mesh the user wants to inspect. Even the improved HoverCam metaphor [MMGK09], which exploits rasterization in order to find the closest point to the camera still requires the scene geometry to be accessible, in order to avoid sudden switching of the camera focus between the objects of the scene. In volume rendering this information is not accessible, since the visualized structures change frequently, based on the chosen transfer function or set clipping planes. Second, the improved HoverCam metaphor requires the use of Proxy Objects of different LoDs in order to speed up rendering, and it utilizes the surface normal at the cursor position. The latter may not be provided by all volume renderers or may not even be clearly defined for arbitrary volumetric data. Third, the HoverCam metaphor does not allow adaptation of the up vector, since the application programmer has to define its behavior for each object. This would be obviously not feasible in volume rendering, where a data set contains a multitude of different objects. Another technique developed for polygonal data is the user designed navigation assistance as proposed by Burtnyk et al. [BKF+02, BKFK06]. However, since it depends on predefined camera positions and/or paths, it is not suitable for the initial exploration of a volumetric data set, which we consider an important subtask of scientific visualization. Additionally, the predefined camera positions and paths may convey only little information after the transfer function has been changed, which is a common operation when exploring volume data. In general, 3D camera control as realized in computer games has only limited potential in the area of volume rendering. This is due to the assumption made by these techniques that the camera is linked to a character which is controlled by gravity and has a certain size. Hence we do not further consider these techniques and refer to the survey by Christie and Olivier [CON08]. Rendering additional camera control widgets like the Navidget developed by Hachet et al. [HDKG08] may not always be desirable since the widgets can block line of sight to the data and integration of geometry with volume rendering is not trivial.

6.3 Design Considerations

To realize an intuitive volume navigation metaphor, several design decisions have to be made. Since seven degrees of freedom come into play, camera control in general is a highly complex task. To allow full flexibility, the user must be able to change the three Cartesian coordinates describing the camera's position, the three Euler angles describing its orientation, and its focal distance [CON08]. Obviously, setting all these parameters manually would result in a cognitive overload. In contrast, since in many visualization scenarios exploration plays an important role, navigation constraints are only acceptable up to a certain degree. In particular, a fully automatic navigation system designed for a specific approach would not be able to adapt to the user's intentions. This is a very important prerequisite, since the user might change her intentions during a visual analysis process frequently [BRS00]. Therefore we have combined the strengths of an interactive approach with those of a reactive approach leading to a semi-automatic navigation metaphor. Especially, when a navigation mode change is required this semi-automatic proceeding is beneficial, since such a change should not be performed by using modifier keys or a graphical user interface [Ras00]. Accordingly, we have limited ourselves to use only a standard wheel-mouse, which also eases porting the technique to touch displays. To further improve the usability, we have integrated well-established concepts from existing navigation metaphors where applicable. We have included the three main metaphors described by Ware and Osborne [WO90]: Camera-in-hand, world-in-hand, and flying vehicle. While with the camera-in-hand metaphor the camera is manipulated as if held in the user's hand, the input mapping is inverted when using the world-in-hand metaphor, and thus the camera rotates around a location fixed in the world. When using the flying vehicle metaphor, the camera can be steered as when sitting in an airplane. By combining these metaphors in a seamless manner, we are able to support proximal object inspection and moving through cavities simultaneously, which we have identified as central tasks in scientific and medical volume visualization.

At this point we would like to emphasize the importance of the flexibility achieved by the semi-automatic nature of our metaphor, which makes it a general-purpose navigation metaphor. This flexibility does not only allow to support distal and proximal object inspection in a seamless manner, but it also supports the navigation of arbitrary data sets, which can be visualized with arbitrary rendering techniques. This decoupling of the data and the renderer is achieved by performing all computations in image space, and it allows to apply the presented metaphor also within yet unknown application scenarios. The benefit of this property becomes clear when considering other visualization systems. For instance, the virtual endoscopy system proposed by Krüger et al. [KKPS08] is a good example. To incorporate collision detection within their GPU-based rendering technique, an extra copy of the data set and knowledge about the renderer and its parameters are required on the CPU. Thus an additional development effort became necessary for navigation purposes only. By having a general purpose navigation metaphor, this interdependency during the development of visualization and navigation algorithms can be dissolved.

The example of the visualization system presented by Krüger et al. [KKPS08] highlights also another requirement for navigation algorithms: The need for collision detection and avoidance. This is in line with the work by Wojciechowski, who also emphasizes the importance of collision detection when dealing with navigation metaphors [Woj06]. We believe that collision detection is of even greater importance in the area of volume visualization, since the volumetric nature of the data to be visualized affords the user to dive into it. Allowing this behavior without providing collision avoidance results in a high degree of occlusion, and thus the reduced visibility quickly leads to disorientation. Integrating an occlusion avoidance algorithm improves the situation, but might still lead to orientation problems, for instance when traveling through complex vessel trees. Hence a general-purpose volume navigation metaphor should also provide some contextual information when diving into the data and thus support location-awareness.

The image-based volume navigation metaphor proposed in this chapter meets the following design criteria, which we have developed by having typical visualization scenarios in mind. It should:

- support semi-automatic camera control to allow flexible and adaptive navigation (e.g., when the user's intention, the data or the rendering style change),
- be intuitive and easy to learn, i.e., avoid navigation mode switches and integrate well-known concepts,
- allow distant and proximal object inspection, which are essential for most scientific visualization applications,
- integrate collision avoidance algorithms, and
- support location awareness.

Since most of these design criteria require some knowledge about the current visualization, we refer to the presented navigation metaphor as context-aware. To simplify the required knowledge extraction process, we assume that the navigation intended by the user depends on the visualized structures. For instance, if the user excludes structures from the rendering, e.g., by adapting the transfer function, s/he likely does not want to inspect these structures. To comply with this assumption, we have to extract knowledge regarding the visualized structures. This can be done by considering the opacity of each voxel or sampling point in the same way the renderer does it when rendering the data. However, a multitude of parameters can affect the opacity at each sampling point: Transfer functions, segmentation data, clipping, other modalities, derived data. Additionally, transfer functions have to be differentiated further since multiple types exist (e.g., 1D intensity, 2D intensity-gradient, 2D LH [SBSG06], 3D [KKH02]) and different segments may also have different transfer functions assigned to them. Incorporating all these parameters within the navigation algorithm would not be a trivial task, and if possible at all it would likely lead to code duplications, which would need to be updated to incorporate additional data sets or formats, new transfer function types, additional rendering parameters, etc. Even if the correct classification could be calculated in the navigation system, the calculated information needs to be stored (possibly at sub-voxel accuracy), processed and regenerated upon changes of rendering parameters. We therefore believe that choosing an image-based approach is not only an option to increase performance but imperative when implementing a flexible volume navigation technique.

6.4 Navigation Algorithm

In order to satisfy the design considerations introduced above, we have realized the image-based navigation metaphor, which follows the workflow depicted in Figure 6.2. To be able to achieve interactive frame rates, the workflow has been designed to work with a GPU-based volume raycaster [KW03], although with slight modifications arbitrary renderers could be used. Based on the data set and the current camera parameters, we generate two pairs of color-coded entry- and exit-points: a conventional one, which is used for the actual rendering, and a spherical one, which is used for the image-based analysis. After the spherical entry- and exit-points have been generated, the subsequent renderer passes its output to the image analyzer. This image analyzer receives incoming mouse events and extracts the required information from the spherical rendering. Based on this information, the current camera is modified, and when necessary, image overlays are generated in order to support spatial awareness. These images are then overlayed over the

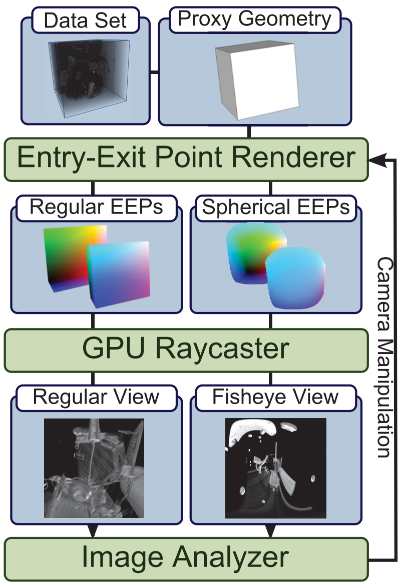


Figure 6.2: The workflow of our navigation system, integrated into a volume rendering system. The same raycaster is executed twice, one rendering is displayed on the canvas while the other is analyzed to map the user's input to camera movement.

standard rendering which is displayed on the screen.

By extracting knowledge about the regions surrounding the current camera position, we are able to flexibly react to mouse input to support the desired navigation tasks. In order to get a representation of these regions, we render a 360° fisheye view from the current camera position. All calculations necessary for the navigation are performed based on this image and the corresponding depth values of the first hit points. By exploiting a spherical mapping, we are able to get the information for all directions in only one rendering pass instead of six passes required for an alternative cube map approach. This is achieved by mapping each texel position of the entry- and exit-textures to $\theta \in [-\pi, \pi]$ and $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, which are translated to cartesian coordinates. The resulting positions, which are shown as color coded images in Figure 6.3 (left), are used to set up the rays for which an intersection test with the proxy geometry is performed. To simplify this process, we assume a convex proxy geometry, which should be no real limitation for most renderers and applications. We have to consider two different cases, based on the position of the camera (inside or outside the volume). In the first case when the camera is located inside the volume, the entry points are simply given by the camera position, and the exit points are the intersection points with the proxy geometry (see Figure 6.3 (right)). In the second

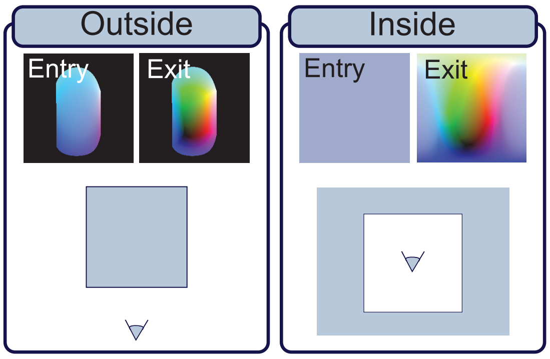


Figure 6.3: Spherical entry-exit points outside and inside the proxy geometry. The entry points for ray casting from inside the proxy geometry are constant because the rays start at the camera position.

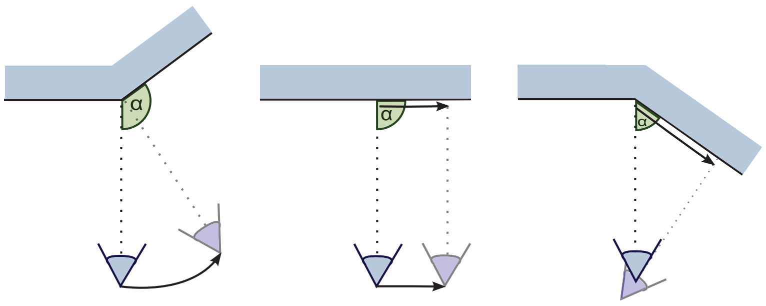


Figure 6.4: We distinguish between three basic camera movements when dragging the mouse: Rotating, Strafing and Panning. By analyzing the depth image we dynamically decide which movement the user wants to perform.

case when the camera is located outside the volume, the entry and exit points are set to the first and second intersection points, respectively, while the volume is visible.

This proceeding has the advantage that we can use a standard volume raycaster in order to process these entry- and exit-points. Hence, our technique does not depend on the volume ray-caster as long as it uses Krüger and Westermann style entry- and exit-points and writes first hit point depth values. Our approach also does not depend on the rendering parameters and the data sets. For the actual navigation we analyze the output, i.e., the color and depth image, of a ray casting performed by processing these entry- and exit-points. The depth image will be used to modulate flight speed and perform collision detection while the color image is used to generate image overlays providing contextual information.

6.4.1 Image-Based Volume Navigation

Since one of our design goals was to exploit well-established navigation concepts, we have initially used the generated image data to implement the original HoverCam algorithm [KKS$^+$05]. However, preliminary results have indicated that this approach suffers from the fact that the HoverCam metaphor always focuses onto the closest point in a certain search window. This often moves the focus in a direction not exactly matching the mouse movement, which is a known problem of the HoverCam metaphor that results in decreased productivity [BLP78]. Hence we have developed our own concepts, which mimic the behavior of the trackball metaphor, when rotations are desired.

When inspecting an object the camera either rotates around the object (rotating, world-in-hand), moves along the object (strafing) or it rotates around its current position (panning, camera-in-hand) (see Figure 6.4), while maintaining a constant distance to the object in all cases. While a common solution to enable the user to perform all these three camera movements is the use of modifier keys, our technique instead solves this problem by being context-aware.

We illustrate our approach by explaining two simple cases where the mapping between mouse input and camera reaction is straightforward: The first case is that of a camera positioned in front of a sphere, with the look-at vector pointing directly towards the sphere's center. In this case, the expected camera movement resulting from a mouse drag should be a rotation in the corresponding direction around the center of the sphere (see Figure 6.5 (a)). In the second considered case a camera is located at the center of a hollow sphere. When performing the same mouse interaction, the user would expect a rotation around the camera position (see Figure 6.5 (b)). Thus, in the first case the user perceives the camera rotating around the sphere, i.e., the world-in-hand metaphor, whereas in the second case s/he perceives the

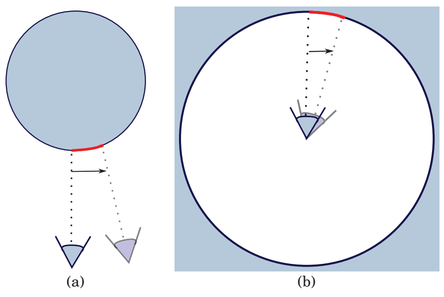


Figure 6.5: Expected camera movement for two simple cases: Rotating (a) and Panning (b).

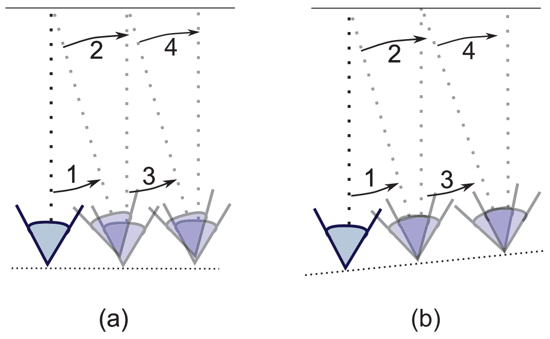


Figure 6.6: Strafing as combination of rotating and panning, with (a) and without (b) distance adjustment.

sphere rotating around the camera or the camera panning inside the sphere, i.e., the camera-in-hand metaphor.

To distinguish between these two basic cases, it is essential to have knowledge about the scene. Since the goal was to realize an image-based technique, the spherical depth image is our only information regarding the scene. Thus, we need to analyze it based on the current mouse input. We compare the depth value at the current screen center and at the current screen center plus the mouse offset. If the depth value in the center is smaller than the other one, we assume that the user intends a rotation, and otherwise a panning camera movement (recall Figure 6.5).

While panning requires no additional knowledge, for the rotating camera movement a center of rotation is required. Since we do not have such high level information in the depth image, we define the first hit point along the view vector as the center of rotation. While this proceeding does not explicitly integrate strafing, this behavior is inherently given. When for instance applying the presented approach in front of a plain wall, strafing behavior is achieved. In this case constant mouse drags along one direction would be mapped to rapid switches between small rotations and panning. The overall movement is a shaky strafing along the wall (see Figure 6.6 (a)). The fact that the combination of rotating and panning (roughly) results in strafing can be integrated into the algorithm to smooth the camera movement: Depending on the depth value difference, we interpolate rotating (movement of the camera position) with panning (movement of the focus), resulting in movement of camera position as well as focus, i.e., strafing. To correctly interpolate these two alternatives independent of the distance to the object, we calculate the surface angle α from the difference of depth values in relation to the depth. Based on this angle α the new camera position and focus are determined, α is depicted by the green arcs in Figure 6.4. If α is large

($> 100°$) the camera is rotated around the focus point. If α, on the other hand, is small ($< 80°$) the camera rotates around its current position, which results in a panning operation. For intermediate values both positions are interpolated, resulting in a translation along the surface, i.e., a strafing operation. However, this approximation of strafing will not maintain the initial distance between the camera and the object, as illustrated in Figure 6.6 (b). This can be fixed easily by setting the distance to the initial value.

In order to account for noise and to be able to move across smaller gaps, instead of just reading the depth values at two distinct locations, we analyze all depth values lying in-between these locations. By applying a Gaussian filter during this process, we are able to achieve smooth camera transitions. Further smoothing is obtained by the fact that the spherical depth image used for analysis has a limited resolution. This allows to integrate a LoD navigation approach, where object clusters are inherently detected. This works, since due to the fixed image resolution the camera distance directly influences the clustering. Objects further away are projected onto fewer pixels and thus small distances between them may dissolve. Thus it becomes possible to rotate around the whole cluster, when further away, while rotating around the individual objects when nearby.

Because rotations may result in an altered up vector, in contrast to the HoverCam metaphor [MMGK09] our technique allows to adapt the up vector interactively. This can be done by dragging with the mouse at the corners of the screen in order to initiate an appropriate rotation.

Since the used collision detection is based on well-known potential field approaches, we do not describe it within this Section, and provide a brief overview in Subsection 6.4.3.

6.4.2 Supporting Location Awareness

To support the user during the navigation, especially when diving into the volume, we augment the rendering by adding appropriate overlays.

Contextual Preview Images

To avoid disorientation, we add small preview windows that show parts of the surroundings currently not in the field of view (see Figure 6.7). These previews are generated by reprojecting the corresponding parts from the spherical rendering. We select the parts positioned on the left, the right, above, below and behind the camera. The latter case can be used for instance to realize a rear mirror metaphor. To render the previews, a view frustum is constructed, and rays are cast through all pixels into

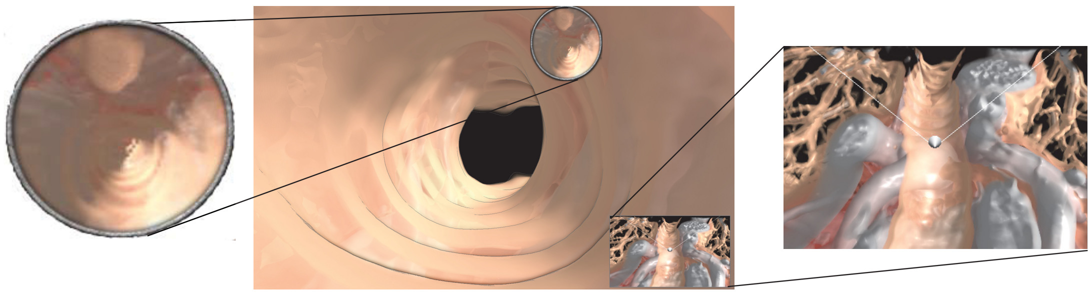


Figure 6.7: Based on saliency and the overall occlusion our technique overlays a rear preview and an overview map, respectively.

the spherical image surrounding the camera. The ray direction is then converted to spherical coordinates and used to look up the texels in the spherical image. Although one might suspect to see distortions in the top and bottom previews this is not the case as shown in Figure 6.8. Because this is essentially just an image processing and not a new volume rendering pass, the previews can be generated very efficiently.

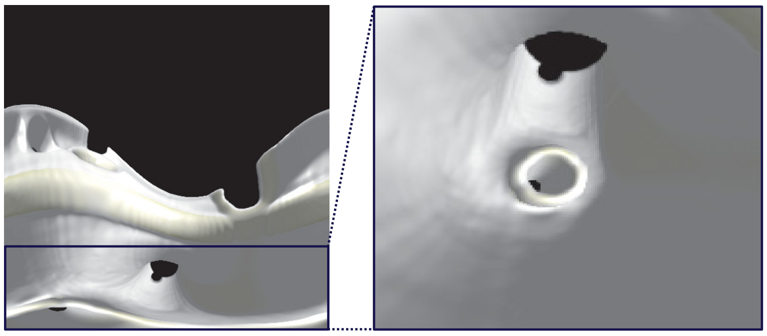


Figure 6.8: The previews are generated from the spherical image (left) without performing additional ray casting passes. The bottom preview (right) is reconstructed from the lower part of the spherical image without distortions.

Overview Map

While the contextual preview images help to get information regarding the surrounding, they provide only little support for estimating the own location. Thus, we also provide a map overlay, which shows the current position from a bird’s-eye perspective when diving into the volume. This metaphor is often used in map applications to support spatial awareness. The map is rendered in an additional pass, where we can use the existing volume raycaster again but with a clipped proxy geometry and a different perspective. To define the parts of the volume visible in this overview map, we exploit the plane defined by the camera position and the up vector to clip the proxy geometry and render it from the bird's-eye perspective, i.e., located above and behind the main camera. An example of this overview map is shown in Figure 6.7. As it can be seen, we have also added the position and the field of view of the camera to support the orientation.

Context-aware Overlays

Both the preview images and the overview map may be helpful to the user in certain situations, but may be useless and just consume valuable screen space in other situations. We therefore propose to make these overlays context-aware and use the information stored in the spherical image to dynamically decide which navigation overlays to display.

Saliency-based previews. Besides taking up screen space the content of the previews is not always necessarily helpful to the user - they might not even contain parts of the data set. The first heuristic we implemented just determined to what percentage each preview is filled. Although this correctly disabled most previews when exploring a data set from the outside, all five previews were active when flying for instance through a vessel. Most of the previews were not improving the exploration but simply showing a vessel wall of uniform color. Therefore we employ the notion of visual saliency for color images [IKN98] and depth images [OH00] to decide the relevance of each preview at the current position in the data set. This heuristic of course assumes that the user has configured the renderer in a way to discriminate features of interest. The more heterogeneous a preview is, the more interesting it is assumed to be. In a similar way, the depth image is considered interesting if the values are non-uniform. Hence previews appear for instance when the user flies by a branch when inside a vessel. If the overall saliency of a preview image exceeds the specified importance threshold, the preview is considered helpful to the user and therefore worth the screen space. Because the rear preview is particularly useful and can be used to reverse the camera direction with a double click, it has a lowered threshold and is therefore visible more often.

Occlusion-based overview. While it makes sense to overlay the map when exploring cavities, it is not very helpful when inspecting an object from the outside. We have therefore implemented a heuristic to decide when the camera is inside a cavity and only then display the map. We consider all preview images except the one representing the front view. If more than 20% of these preview images are occupied with data lying within a closeness threshold, the camera is considered to be within a cavity and thus the overview map is displayed. In our tests we found this heuristic to be sufficiently robust.

6.4.3 Integrated Camera Control

To unleash the full potential of our context-aware navigation technique, we have integrated it with other camera control metaphors. To combine our approach with a convenient traveling metaphor, we have implemented the flying vehicle metaphor. Thus, by pressing and holding the left mouse button for a short amount of time without moving it the user can start flying through the data set. During this flight, the direction is continuously modified by the offset between the mouse cursor and the center of the screen. The traveling speed is calculated based on the distance to the closest point in a region around the center of the screen. This proceeding is sufficient, since a user evaluation conducted by Ware and Fleet [WF97] has shown that averaging the depth values had no advantage over just using the closest depth value, when modulating flight velocity based on depth values. To prevent occlusions, we use a force field technique. In particular, we have implemented the force field approach proposed by Xiao and Hubbold [XH98] to prevent collisions while still allowing smooth flying. To allow mostly unrestricted movement, in our realization the camera is not constantly moved towards the center of the cavity, as it is done when applying navigation in virtual colonoscopy [HMK+97].

To support zooming, the user can move the camera forward and backward by using the mousewheel. The speed is again determined by the closest point, but no force field is used to provide a real zoom instead of an additional method to fly, which is consistent with the standard trackball. When moving backwards the closest point on the backside is used to check for collisions.

By double clicking the camera rotates to focus the point at the cursor position, which can be used to switch between separate objects. It should be pointed out that while it is of course possible to integrate this operation into a normal trackball navigation it would not be very helpful to the user. In contrast to our technique, which constantly updates the center of rotation, the trackball focus would stay fixed at the clicked point on the surface of the object, and thus the object would rotate around this point.

Finally, we have added a history function, which allows the user to fly back on the previous navigation path. Instead of adding an option to fly backwards or automatically push the camera out of an object, as proposed by McCrae et al. [MMGK09], we allow the user to rewind to previous positions by pressing the right mouse button. We believe that this solution, which has some similarity to forward and backward movement on a path generated for virtual endoscopy applications [HHCL01], is more robust and less confusing for the user.

6.5 Evaluation

6.5.1 Usability

Our usability evaluation is based on the results of a user study we performed. Additionally, we present our findings from an interview with a medical expert.

Theoretical Evaluation

Due to the integration of well-known concepts and the thus resulting similarities with the trackball metaphor, we can apply quality criteria developed for this metaphor. Bade et al. have developed such a measure [BRP05]. They have identified four general principles to which rotation techniques should adhere. In the following we briefly describe how our navigation metaphor complies with these criteria:

1. *Similar actions should provoke similar reactions.* Since the modification of the camera depends on the viewed object this principle cannot be generally fulfilled by our technique. However, when looking at the same or very similar objects the camera movement is comparable and predictable.

2. *Direction of rotation should match the direction of 2D pointing device movement.* This is guaranteed by our technique since the movement of camera position and focus are always in the plane defined by the mouse movement.

3. *3D rotation should be transitive.* This principle cannot be fulfilled because our technique jumps to different parts when the user rotates them directly into the focus. Bade et al. note that this principle is crucial to return to the initial viewing position, and we can substitute this functionality by allowing the user to rewind to previous camera positions.

4. *The control-to-display ratio should be customizable.* The ratio can be configured in the GUI.

To put this evaluation into perspective, it should be noted that none of the rotation techniques reviewed by Bade et al. [BRP05] fulfilled all four criteria, although the authors consider them as the state-of-the-art. Furthermore, the improved Hover-Cam [MMGK09], which is the technique most similar to ours, does not fulfill the second and the third criterion.

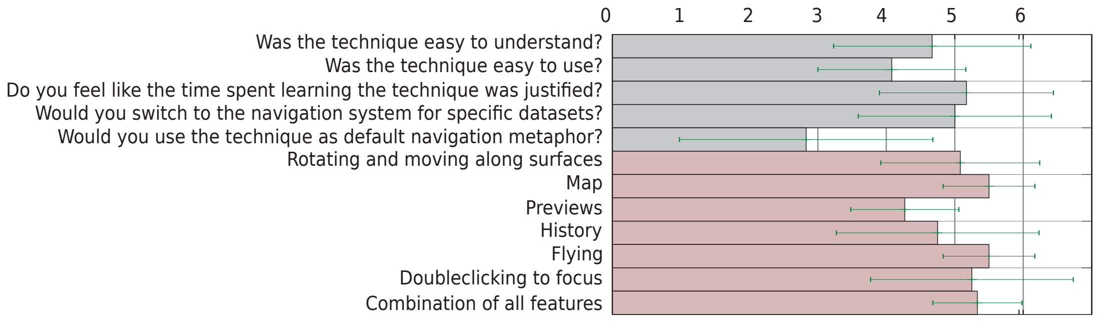


Figure 6.9: The results of the questionnaire of the conducted user study, which we have evaluated on a seven point Likert scale (0=strongly disagree, 7=strongly agree). Questions 6-12 (red) are about specific aspects of our technique and the participants were asked if they perceived these as useful or good solutions to the problem.

User Study

With a lot of mainstream applications like Google Earth using the trackball metaphor we assume that the vast majority of users, especially those that might be interested in navigating volumetric data, are familiar with this navigation approach. We therefore designed our study to investigate how users perceive the implemented navigation system and if they would be interested in using it - as an alternative to the trackball or even more ambitious as the default navigation technique. Another important aspect of our study was to find out how difficult it is for the average user to learn our technique and whether the user thought the time spent to learn the technique was well spent.

11 male and 2 female subjects participated in the study. Most subjects were students or members of the departments of computer science or medicine. All subjects were first-time users of our navigation technique. The total time per subject including instructions, training, experiment, breaks and debriefing took approximately five to ten minutes. The study itself consisted of a timed navigation task in which we compared our technique with the trackball (two-axis valuator) and a learning task in which the users had to navigate using our technique (roughly) as shown in a tutorial video. The second task was designed to include all aspects of our navigation system. A questionnaire based on a seven point Likert scale has been used to evaluate the user's experience.

The users were split up into two groups: The trackball group first performed the navigation task, was then instructed in our technique and performed the learning task. The other group was instructed how to use our technique, then performed the task to learn it, and performed the navigation task using our technique. Since the trackball is not designed to navigate through cavities, we compared only the ability to navigate around objects from the outside in the timed navigation task. The users had to read three letters placed on a sphere in a synthetic data set consisting of a set of objects. All users started at the same position, had to zoom in on the sphere and then navigate around it to read the letters.

The learning task was performed using the Baby data set (a CT scan of a head). The users were shown the following navigation subtasks in a video and had to replicate them:

1. Navigate around the head.
2. Use the transfer function to make skin and brain transparent and navigate inside the skull.
3. Use the rewind functionality to get back to the starting position.
4. Focus the tube, inspect it from the outside (rotate around it) and fly through it.
5. Use the rear preview to focus on the data set after leaving the tube at the neck.

Figure 6.9 shows the results of the questionnaire. For the users it was generally easy to learn the technique in a few minutes, and they perceived the navigation metaphor as easy to use. Many even fully agreed that they would like to use the metaphor for special data sets. However, the rather ambitious question, if the users would in general switch to the metaphor received only little positive resonance. We believe that this partially results from the fact that the users had some previous experience with the trackball metaphor. As seen in Figure 6.9, all additional navigation techniques were received very well. Besides these results, we found that users with the most experience using the trackball had greater difficulties learning the new technique. While the users were in general faster, when using our metaphor, this fact has to be evaluated carefully. However, since we only performed one test and just have evaluated 13 users, this may not be statistically significant.

Evaluation with Domain Experts

To get an expert opinion we interviewed a medical doctor and two medical PhD students, all specialized in cardiovascular medicine. During the interviews, we have demonstrated our technique by loading several different data sets and handed over control to the medical experts. The participating medical experts had less experience with the trackball metaphor and 3D graphics in general than the lay users participating in our study, because they almost exclusively work with slice viewers. Thus, they had more problems performing 3D navigation tasks, also when using the standard trackball metaphor. When using our technique, they appreciated the possibilty to move along surfaces and travel through the data set without experiencing collisions. While they also liked the option to strafe along a surface, the subjects expressed the desire to be also able to look ahead in the direction of movement instead of just towards the surface. Furthermore, they stated that medical doctors seldom need to inspect structures from the outside, and therefore the demand for trackball-like behavior was rather limited. As exceptions to this rule the inspection of bones, especially the pelvis with rather large and flat bone structures and the shoulder joint, were mentioned.

Furthermore, the non-transitive rotation (as already discussed) was found to be a bit non-intuitive. While the overview map was a favorite of the laymen users, the medical experts were irritated by the slab like view rotating with the camera. They explained that recognizing anatomical structures was difficult since they learned to recognize these structures on axis-aligned slices.

6.5.2 Performance

The proposed system has been implemented in C++, OpenGL, GLSL and OpenCL. To minimize downloads from the GPU and speed up computation we exploited OpenCLs ability to interoperate with OpenGL and perform calculations on the spherical image directly on the GPU. Thus, we were able to interweave the visualization and navigation technique in a manner that everything is done in-situ. This allows interactive frame rates, which would not have been possible with CPU realizations. In our implementation, we have parallelized the force field approach, and only a few floats need to be read back from the GPU. Finding the closest point in a direction and reading a line from the depth image is accelerated in a similar fashion. Reading only some scalar results or parts of a texture back from the GPU would have been more complicated using OpenGL exclusively.

The most expensive operations are obviously the additional rendering passes for the spherical view and the map. The spherical view is rendered at a relatively low resolution (we found 256×256 to be sufficient for all tested data sets) while the size of the map is a (configurable) fraction of the main canvas size. For a typical canvas resolution of 1024×1024 the rendering time for one frame increased by 5-10 % for each of these renderings. We found the impact of the actual calculations for the navigation to be negligible. Therefore, if an existing rendering system is able to deliver interactive performance this would very likely not be changed by the integration of our technique.

6.6 Integration in Voreen

The navigation technique discussed in this chapter has been implemented in a single Voreen processor handling the interaction as well as the overlay functionality. Figure 6.10 shows a minimal network integrating the navigation processor. The processors in the network correspond to the components in Figure 6.2, with an additional raycaster to render the overview map. We therefore have three ray casting pipelines in the network (from left to right) which all connect to the navigation processor:

- The spherical ray casting is performed using a standard raycaster in combination with a spherical entry-exit-point renderer, which performs an OpenCL based spherical ray casting of the incoming proxy-geometry.
- The overview map is rendered using a different camera position with a clipped proxy-geometry.
- The main 3D-view is generated using standard components (see Chapter 3).

The navigation processor controls the properties (camera, clipping plane) using linking and combines the incoming images.

6.6.1 Limitations

Although we developed our technique as general as possible we are aware that it will probably not be ideal for all data sets. Large semi-transparent regions (e.g., in data sets resulting from physics simulations) cannot be explored from the inside because the spherical image provides no usable depth values in this case. If these regions contain less transparent inner parts along which the user wants to navigate, the transfer function could automatically be adapted to make almost transparent voxels completely transparent. This transfer function would then be used to render the spherical image. Data sets where the objects of interest cannot be separated from the background visually (e.g., due to a low signal to noise ratio) can also be problematic. New rendering techniques that generate clearer images for these data sets can, however, be integrated easily and thus improve the navigation.

A general problem with semi-automatic navigation methods is the possible conflict with the user's intention. This can be overcome by adding possibilities (e.g., modifier keys) to force the navigation technique into a specific mode. Keeping in mind touchscreens as possible application we did not integrate these.

The continous switching between different navigation metaphors can produce occasional jittering. We have tested different approaches to eliminate this jittering. Using a hysteresis to limit switching between the different interaction metaphors resulted in increased jittering. For instance, when rotating the camera around a spherical object, the object would be moved out of focus which would then be corrected by rotating the camera. This results in an unsteady movement of the sphere on screen, which is perceived as irritating. Approximating the center of the object in focus from the depth image and rotating around it resulted in a smoother, more indirect movement around simpler objects (spheres, cubes), but turned out to be too unreliable for complex objects. If the centerline or medial surface for an object of interest is static and available it should obviously be used for navigation.

So far, we did not integrate support for time-varying data, because in the current form the navigation does not derive any knowledge from the data. When navigating with our technique while the current time step is changed, may lead to unintuitive behavior.

Besides these rather conceptual criticisms, we see also potential for providing technical improvements. We currently use a rather simple method to approximate the saliency of each preview. Although this heuristic provided reasonable results in all tested cases, it could certainly be extended to reposition the focus of the previews on features of maximal saliency, adjust the field of view to isolate interesting features or perform a sophisticated feature detection.

6.7 Conclusions

In this chapter we have introduced a general-purpose volume navigation metaphor. By exploiting image analysis of a spherical projection of the camera's environment, we are able to simulate different concepts known from well-established navigation metaphors without requiring navigation mode switches. We have shown how to support rotating, strafing and panning of the camera by exploiting an image-based analysis of its environment. Thus, we enable proximal as well as distal object inspection, which is an important combination in many scientific visualization applications. Furthermore, we have shown how to exploit the knowledge about the camera's context in order to generate preview images as well as overview maps. The latter are of particular importance in the area of volume visualization, since the volumetric nature of the data affords to dive into it, which easily leads to disorientation. Thus, our technique is able to support location-awareness. Since all required analysis tasks are performed based on the rendered image, the presented metaphor can be considered as a general-purpose metaphor, such that it can be used with most visualization techniques and data sets. To our knowledge the presented approach is the first general-purpose volume navigation metaphor. To evaluate the quality of the presented navigation approach, we have performed a usability evaluation and have conducted interviews with medical experts. Since we obtained positive feedback from these tests, we believe that the presented navigation metaphor can fill the gap between general purpose volume visualization systems using classic navigation metaphors and systems developed for specific application cases that exploit specialized navigation techniques. It therefore also represents a useful building block in rapid prototyping tools such as Voreen. In the future it would be worth to investigate how to address the limitations discussed in Section 6.6.1.

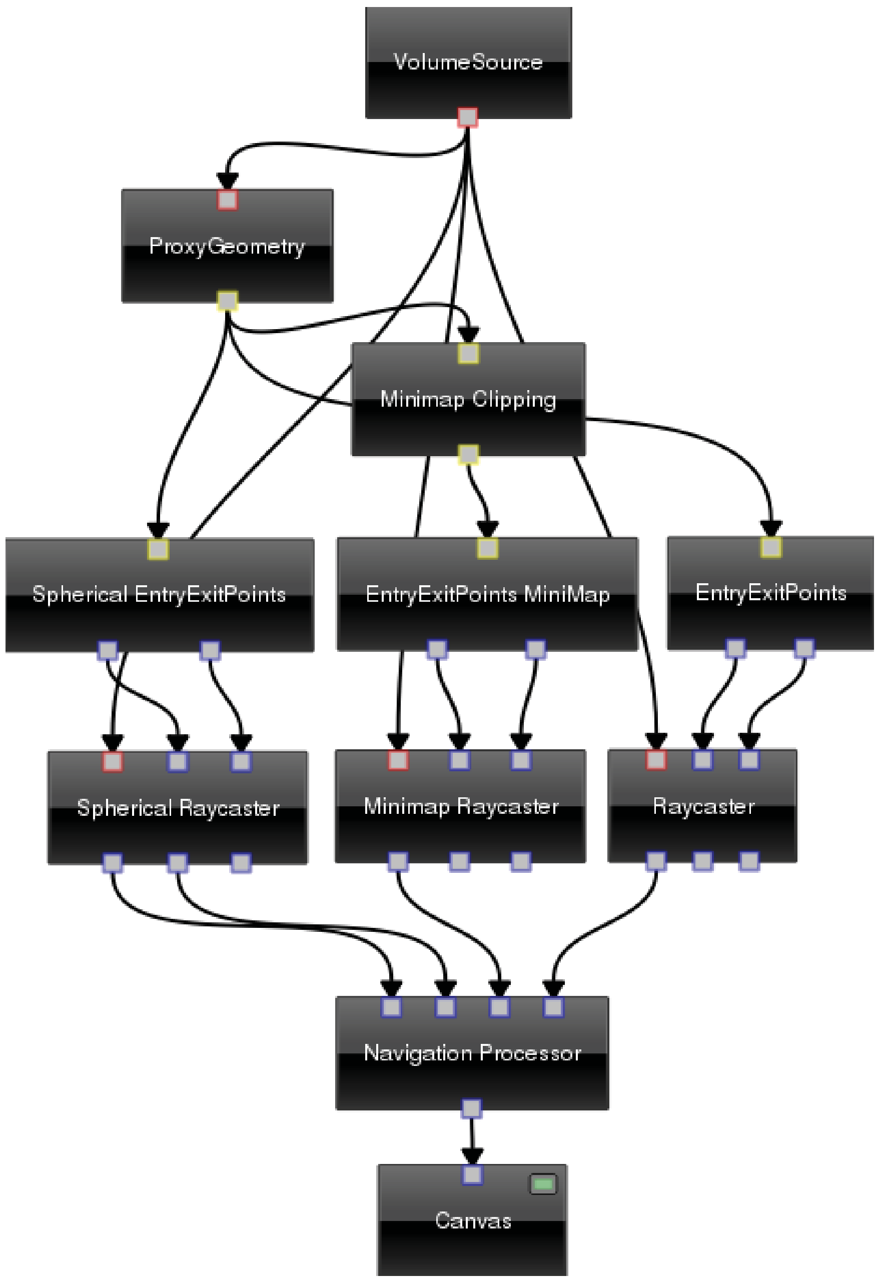


Figure 6.10: Integration of the navigation processor into a Voreen network.

Chapter 7

Interactive Planning for Brain Tumor Resections

This chapter presents concepts for pre-operative planning of brain tumor resections. The proposed system uses a combination of traditional and novel visualization techniques in order to support the neurosurgeon during the planning process. A set of multimodal 2D and 3D views conveys the relation between the lesion and the various structures at risk and also depicts data uncertainty. To facilitate efficient interactions while providing a comprehensible visualization, all views are linked. Furthermore, the system allows the surgeon to interactively define access paths by clicking in the 3D views as well as to perform distance measurements in 2D and 3D.

7.1 Introduction

We propose a planning system for neurosurgical procedures, using a variety of visualization techniques. In cooperation with our medical partners, we have developed a workflow during which the surgeon first inspects the tumor and surrounding tissue and then specifies and analyzes access paths. Our application prototype utilizes novel as well as best practice visualization techniques and integrates all the modalities provided for the Visualization Contest (see Section 7.2.1). We will discuss the workflow in our application, our preprocessing steps and visualization techniques as well as the feedback received from domain experts.

7.2 IEEE Visualization Contest 2010

7.2.1 Data Sets

Two state of the art data sets have been acquired on a Siemens 3T Verio scanner. They were provided courtesy of Prof. B. Terwey, Klinikum Mitte, Bremen, Germany. Each data set consisted of anatomical images (T1, T1 +contrast agent, T2, FLAIR, SWI) and functional/structural images (fMRI of a finger tapping task and DTI). Additionally, postprocessed data such as brain- and tumor masks, and a statistical parametric map of the fMRI data have been provided. For one of the data sets, high-resolution CT data was also provided.

7.2.2 Clinical Questions

The following questions were posed as problems to be addressed in the proposed visualization solutions for the contest.

- What is the relation between the lesion, functional areas and white matter tracts?
- How can the lesion be accessed most safely?
- How close is the tumor located to vital functional areas, such as the visual-, language- or motor-system?
- What is the distance between the tumor and important fiber bundles related to motor-, language- and vision tasks?
- Does the tumor infiltrate or displace any of these tracts?
- To what extent (how radical) may a resection be performed?
- Which arteries or veins lie on the chosen access path?
- Finally, being aware of the technical limitations of the underlying MR measures, an important aspect deals with the certainty with which algorithmically derived measures may be regarded. This is especially important for DTI and fMRI. How can the remaining uncertainty be visualized effectively?

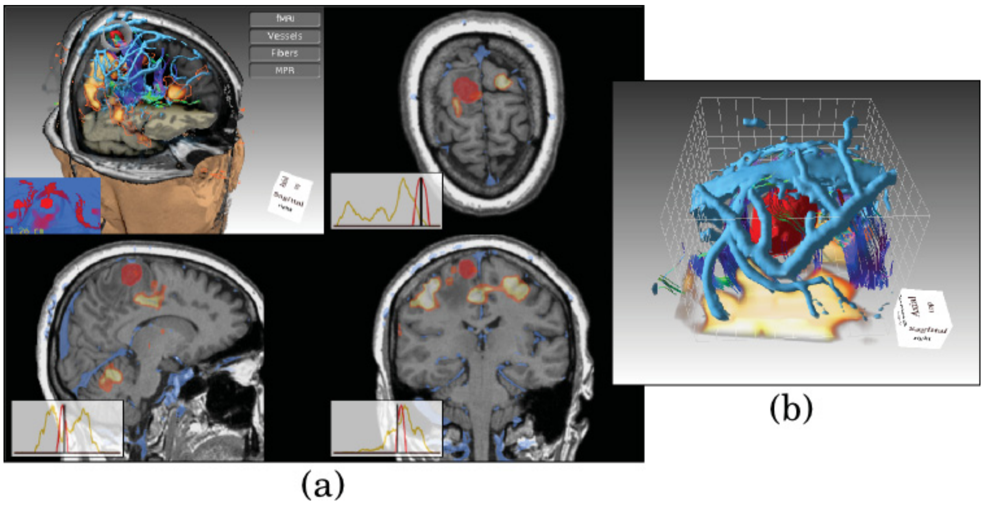


Figure 7.1: Workflow Step 1: Exploring the data and planning an initial access path can be done by exploiting multimodal 3D and 2D views (a). Furthermore, the tumor view allows a close-up inspection of the vicinity of the resection region (b).

7.3 Related Work

Neurosurgical planning software has been an active research topic for several years. Our discussion of the previous work in this area is focused on more recently proposed systems that utilize multi-volume 3D visualizations. Beyer et al. [BHWB07] propose an application that employs multi-volume ray casting and skull peeling, a technique to selectively remove structures obscuring the brain without segmentation. Rieder et al. [RRRP08] propose a neurosurgical planning tool that uses distance based transfer functions and visualizes the access path as a cylinder.

Multi-volume ray casting is an important part of a modern neurosurgical planning software. Examples for more recent implementations on modern GPUs are a BSP tree based technique proposed by Lindholm et al. [LLHY09] and a depth peeling based approach by Brecheisen et al. [BiBPtHR08]. Kainz et al. [KGB$^+$09] propose a CUDA-based renderer which can handle multiple volumes in combination with complex polyhedral objects.

Visualization of DTI uncertainty was an important question in the contest, but to the best of our knowledge the only solution to visualize fiber tracking uncertainty has been proposed by Brecheisen et al. [BVPtHR09].

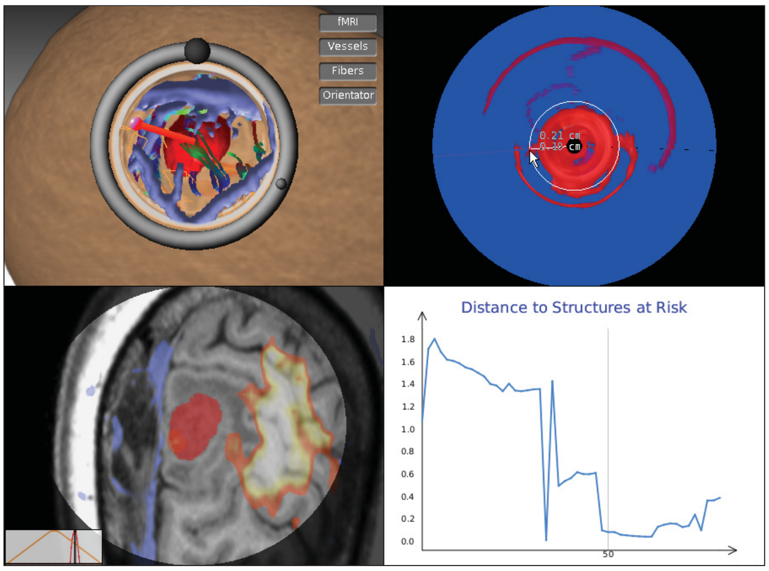


Figure 7.2: Workflow Step 2: In-detail inspection as well as modification of the access path and operation preparation by exploiting a probe view, a cylindrical access path projection, a surgeon microscope slice view and an access path distance plot.

7.4 Planning Workflow

We have identified a two step workflow used by our medical partner and developed our application prototype to support this workflow:

1. Initial investigation of the data in combination with an interactive access path specification
2. Deeper analysis of the chosen access path and the actual preparation for the surgery

Figures 7.1 and 7.2 show the views in workflow step 1 and workflow step 2. The surgeon can always switch back and forth between the steps to choose a new access path and analyze it.

7.4.1 Workflow Step 1

In step 1 we combine 2D slice views enhanced with lift charts with a 3D context view that integrates relevant information from the available modalities (see Figure 7.1 (a)). The slice views provide insight into structures and allow the neurosurgeon to identify structures inside the tumor and to diagnose its type. The location of the tumor as well as its relation to risk structures is depicted in the 3D view, which is further overlaid with a tumor map (see Section 7.6.2) showing a projection of relevant structures as seen from the tumor. To get a better view of the tumor and relevant adjacent structures we provide an additional close-up view (see Figure 7.1 (b)). Here, the surgeon is able to identify vessels and fibers that may be infiltrated or displaced by the tumor. Using these views the neurosurgeon can place one or more access paths before analyzing and comparing them in step 2.

7.4.2 Workflow Step 2

In step 2 we combine a classical probe view with a projected map of the access path, a plot showing minimum distances along the path and a slice view orthogonal to the path to allow a compehensive analysis of the chosen access path (see Figure 7.2). The probe view (top left) shows all structures inside the access path and provides a preview of how the access path would look during the operation. It also enables the surgeon to fine-tune the previously chosen access path. To locate structures close to the access path the top right view displays a projection of the structures surrounding the access path onto the surface of the access path cylinder (see Section 7.6.2). A plot showing the minimum distances of relevant structures along the access path is located at the bottom right to allow an easy comparison of several possible paths. Finally, in the bottom left the system offers a slice view that is centered around the access path and oriented perpendicular to it, thereby providing a view corresponding to the operation microscope focusing at a certain depth.

7.5 Preprocessing

7.5.1 Segmentation & Registration

We have segmented the vessels from the T1 data set by using a random walker based segmentation system presented by Praßni et al. [PRH10]. Brain and tumor mask were provided with the contest data sets, and all data sets provided by the contest were coregistered.

7.5.2 DTI Fiber Tracking

The DTK software [WBSW07] has been used to perform fiber tracking before importing the resulting fiberlines into our application. Tracking can be performed using either the FACT algorithm, second-order Runge-Kutta, interpolated streamlines (used for the images in this chapter) or tensorlines. To allow the extraction of relevant fiber

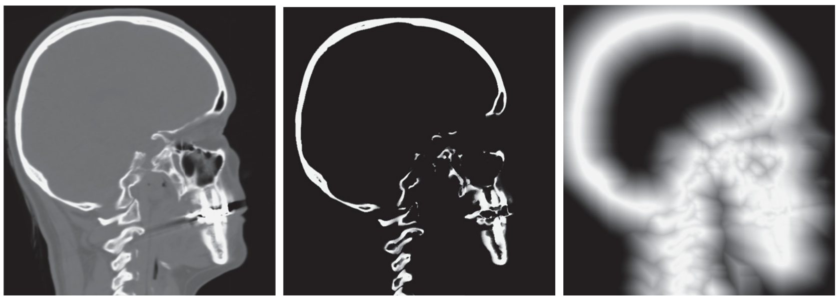


Figure 7.3: Contextual DTI uncertainty is based on the distance *d* to bones, masked from a CT scan.

tracts, e.g., the pyramidal tract and the arcuate fasciculus, we support an interactive ROI definition. Furthermore, the fibers can be filtered based on anisotropy, length and direction.

7.5.3 DTI Uncertainty Extraction

To deal with the uncertainty introduced through DTI, we incorporate the fiber context as well as the fiber anisotropy. Since DTI is less certain in regions near bone or air [Cha05], we have applied a volume analysis that first applies a threshold to extract bone and air structures. Based on the thresholded volume we perform a distance transformation that computes the distance d (see Figure 7.3) to these structures for each fiber segment. A user-defined security margin can then be defined around air and bone structures. We normalize the computed distance to obtain a structural uncertainty U_S. To get the final uncertainty for a fiber segment, we combine U_S with the anisotropy uncertainty U_A to obtain the overall uncertainty $U = max(U_S, U_A)$.

7.6 Visualization

In this section we discuss all visualizations we utilize in the application prototype, focusing on novel techniques like our proposed uncertainty visualizations and projection techniques.

7.6.1 2D and 3D Views

The 2D slice views integrated into our system are standard multimodal slice views, which have been extended by using enhanced lift charts, inspired by those proposed by Tietjen et al. [TMS+06]. Within the lift charts (see Figure 7.4), we depict the extent

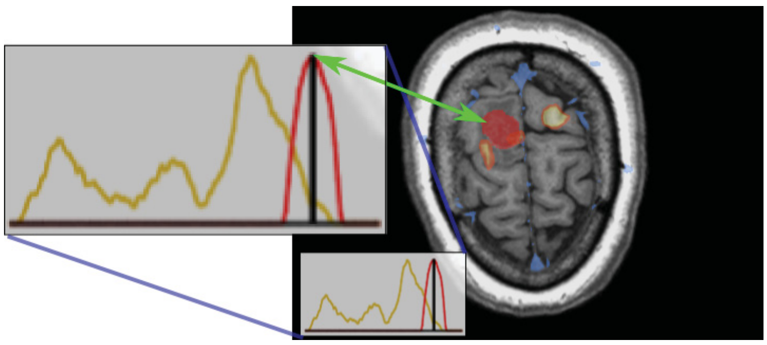


Figure 7.4: Enhanced lift charts indicate the current position in the slice stack and display the amount of malignant tissue (red) as well as fMRI signal (yellow).

of malignant tissue (red curve) as well as the fMRI signal (yellow curve) for each slice. The current slice in the stack is also indicated to help the user to navigate through the slices. We overlay the shown modality with the most important structures at risk, namely the tumor segmentation mask, the fMRI signal and the vessels.

There are three different 3D views provided by our system. The 3D *context view*, shown during the first step of the workflow, integrates all relevant modalities into a comprehensible rendering. It can be used to identify the various structures at risk and to understand how they relate and interact with each other. To generate high-quality 3D views, we exploit GPU-based volume ray-casting [KW03], which has been shown [SHC$^+$09] to generate images superior to other volume rendering techniques. In order to integrate the fiber geometry into the 3D views, we have modified the exit points used by our GPU-based ray-caster, as proposed by Scharsach [Sch05].

To prevent cluttering, all modalities can be easily deactivated through on-screen buttons (see Figure 7.1 (top right)). Additionally, the user can activate a region of interest for the vessels based on the distance to the tumor surface.

The 3D context view also allows the surgeon to efficiently define the access path by using the mouse. When the user clicks on the rendering the intersection point with the skull (which can be transparent) is set as new starting point for the access path.

The second 3D view in workflow step 1 is the *tumor view*, which focuses on the spatial relation between the tumor and nearby structures at risk (see Figure 7.1 (b)). As the context view, it also integrates all relevant modalities, but displays a close-up showing only the tumor and structures in the proximity of the tumor. Thus, the tumor view can be used to support the surgeon when analyzing how close the tumor is to vital functional areas. The surgeon can especially analyze which fibers infiltrate the tumor to what extent, or which fibers are displaced.

After the access path has been defined, a 3D *probe view* is exploited during the second step of the workflow in order to identify all related risk-structures lying along the chosen path. This probe view allows the physician to see the access path in a similar way as during the resection. The orientation of the patient's head during the operation depends on the type and location of the tumor. The surgeon may use the ring widget (see Figure 7.2 (top left)) to rotate the head around the fixed access path axis in order to match the actual orientation. The bigger ring marker can be used as rotation widget, while the smaller one indicates the direction pointing to the patient's nose.

7.6.2 Projection Techniques

We utilize two projection techniques to provide the surgeon with a quick overview of the most important structures at risk: The *tumor map* shows the structures close to the tumor, while the *access path projection* shows the structures close to the access path. We employ an intuitive red-blue color mapping for both views, where structures at risk nearby are displayed in red. The distance images needed to calculate these views are generated using a standard volume raycaster, parameterized by entry-exit-point textures. We pass spherical/cylindrical EEPs (see Figure 7.5(a)) to the raycaster and use the first hit points (see Figure 7.5(b)) to calculate the distance map (see Figure 7.5(c)). We raytrace the Proxy-Geometry with an OpenCL kernel to allow these types of projections.

Tumor Map

The tumor map, which is overlaid on the 3D context view in workflow step 1, is inspired by the projection type presented by Rieder et al. [RWS$^+$10]. However, we calculate the distance to nearby structures at risk and color the results using the aforementioned continous red-blue color mapping. The use of the tumor map is twofold: First, the surgeon can quickly identify directions with few critical structures by looking for large blue areas and directly set an access path in this direction by clicking on the map. Secondly, the map can be used to measure distances to structures, such as vital functional areas or the pyramidal tract. Measuring is performed by simply moving the mouse over the tumor map. The measured distances are also displayed in 3D in the context view (see Figure 7.2 (top left)). We generate the tumor map by performing two spherical ray castings from the center of the tumor. First, we render the tumor mask using an inverted transfer function (i.e., the inside of the tumor is transparent, the rest is opaque). We then use the first-hit points as entry points for a second ray casting of all risk structures. By calculating the distance

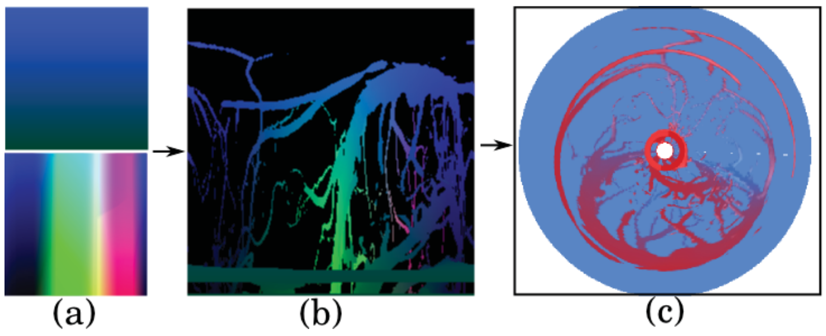


Figure 7.5: Cylindrical entry- and exit points (a), result of the cylindrical ray casting (first hit points) (b), resulting distances color coded and mapped to a disc (c).

between entry- and first-hit points for this ray casting we get a distance map to which we apply the red-blue color mapping.

Access Path Projection

Besides their distance to the tumor, the distance of structures at risk to the access path is also important. Therefore, the access path projection used in step 2 of the workflow shows the distance to all relevant structures as seen from the access path (see Figure 7.6 (right)). The result of this cylindrical projection is mapped to a disc, with the center of the projection representing the deeper end of the access path. As with the tumor map, moving the mouse cursor over a red region of the map will automatically measure the distance to the structure at risk and display it in the probe view (see Figure 7.6 (left)). For the access path projection we also display the distance along the access path, which is important to the surgeon.

Distance Plots

To further facilitate the assessment of the access path, we exploit an access path cache together with a plot depicting the minimal distance to structures at risk along the access path (see Figure 7.2 (bottom right)). The access path cache can be used like a bookmarking functionality, where the surgeon can cache access paths of interest. By selecting different access paths from the access path cache, they can be compared and modified easily.

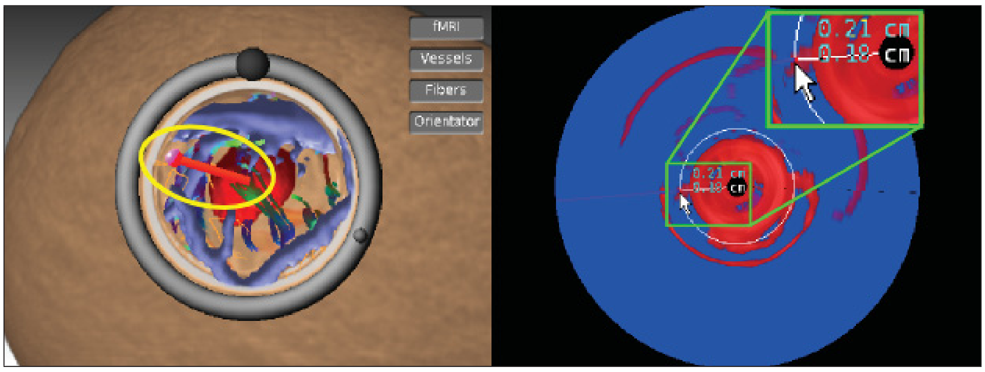


Figure 7.6: Access Path Projection: The measured distances are displayed in the probe view using linking.

7.6.3 Uncertainty Visualization

DTI Uncertainty

When visualizing the DTI fiber tracts, we incorporate the derived uncertainty information introduced in Subsection 7.5.3. We encode the uncertainty in the saturation and value of the displayed fiber color in the HSV color space. The hue is determined by the standardized directional fiber color-mapping physicians are used to. We lower the saturation and value in regions of high uncertainty (see Figure 7.7). Thus, uncertain fibers become less emphasized and their orientation, which can also be considered as less certain, is less prominent.

fMRI Uncertainty

Because of the low resolution of the fMRI scans and the possibility of partial highlighting of motor regions due to finger tapping we render rather larger regions of uncertainty around core fMRI regions. We have applied an approach inspired by the work of Nguyen et al. [NYE$^+$10]. We display the core of each fMRI region by exploiting a diffusely emitting light signal. Additionally, in order to express the uncertainty regarding the size of theses regions, we add an uncertainty margin, depicted by orange borders (see Figure 7.7). To generate this visualization we render the fMRI signal twice: In the first pass we render the core regions using a higher threshold, while in the second pass we use a lower threshold and apply an edge detection filter. We then composite this border-image with the result of the first pass.

7.6.4 Brain Rendering

Because gradients in MRI scans are unreliable due to noise, we use a distance based darkening (*dark means deep*) and depth darkening [LCD06] to render the brain and simulate the effects of a global illumination model with a minimal performance impact. We render the brain without shading (see Figure 7.8 (b)) and then apply the depth darkening to the image. The resulting image depicts the structures of the brain in a more comprehensible manner (compare Figure 7.8 (a) and (c)). We then integrate the rendering of the brain into our multi-volume ray casting by modifying the EEPs as proposed by Scharsach [Sch05].

7.7 Interaction Techniques

We have integrated several interaction techniques which support the mental linking of the different views as well as a deeper understanding of the data. The surgeon can intuitively measure distances between structures of the same or different modality in the image and specify or alter the access path. Interactive navigation in all 3D views is possible at interactive frame rates due to GPU acceleration.

7.8 Evaluation

The clinical value of our application prototype was rated as high (7 and 9 out of 9) by the two neurosurgeons who reviewed our entry in the contest. We have also demonstrated our application to our medical partners and received positive feedback. The enhanced lift charts were appreciated for providing a simple indication of the current position in the slice stack as well as indicating the amount of fMRI activity, while the 3D visualization was well received due to the intuitive integration of a wide range of modalities.

7.9 Future Work

The current evaluation by neurosurgeons is based on a video, a more practical hands-on session would certainly be desirable. Uncertainty visualizations for modalities other than DTI and fMRI should be investigated. The current DTI uncertainty visualization cannot be combined with shading techniques because the user cannot distinguish between low light and high uncertainty. Other sources of DTI uncertainty (e.g., by the fiber tracking algorithms) need to be integrated.

7.10 Implementation in Voreen

Multi-volume ray casting and slice rendering as well as DTI techniques for Voreen have been developed in the context of this application, they have already been discussed in Chapter 4. Slightly modified versions of the `MultiVolumeRaycaster` have been used to perform the visualization of the access path. Integration of opaque geometry has been realized by modifying the entry-exit-points (see Figure 7.9) using the `EEPGeometryIntegrator` processor.

The tumor map and access path projection are created using special entry-exit-point processors in combination with standard proxy-geometry and ray casting (see Figure 7.10).

7.11 Conclusion

We have presented an interactive system for brain tumor resection planning. By combining traditional with novel visualization and interaction techniques through linking, the system supports an intuitive analysis of multimodal brain data sets for pre-operative planning.

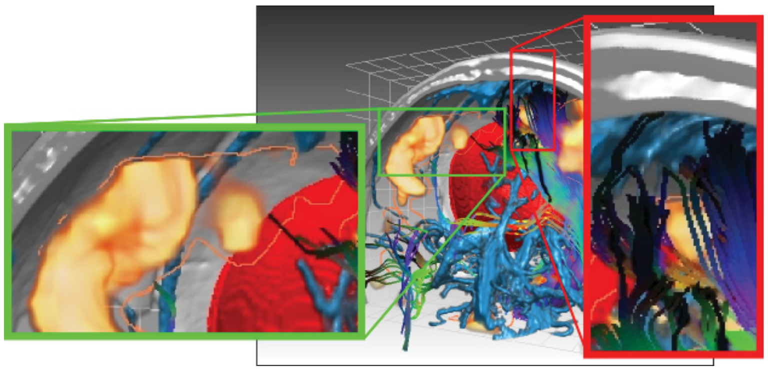


Figure 7.7: Uncertainty visualization. **fMRI (green inset):** Core regions are rendered using a diffusely emitting light signal, uncertainty borders are rendered in orange. **DTI (red inset):** Fibers close to bone and air are rendered with less saturation and brightness to mark them as uncertain.

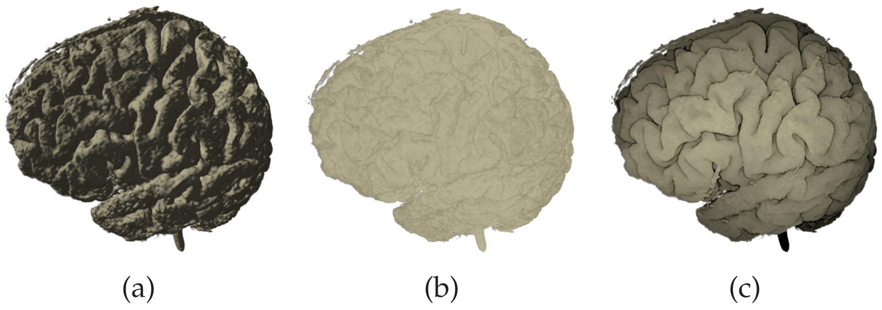


Figure 7.8: Comparing different techniques to shade the brain (using the same transfer-function): Gradient based shading (a), no shading (b), depth darkening and *dark means deep* (c).

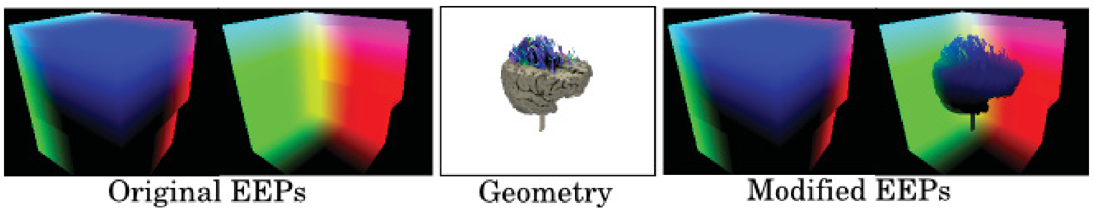


Figure 7.9: Integration of geometry (fiber) and single volume ray casting results (brain surface) into the EEPs of our multi volume raycaster.

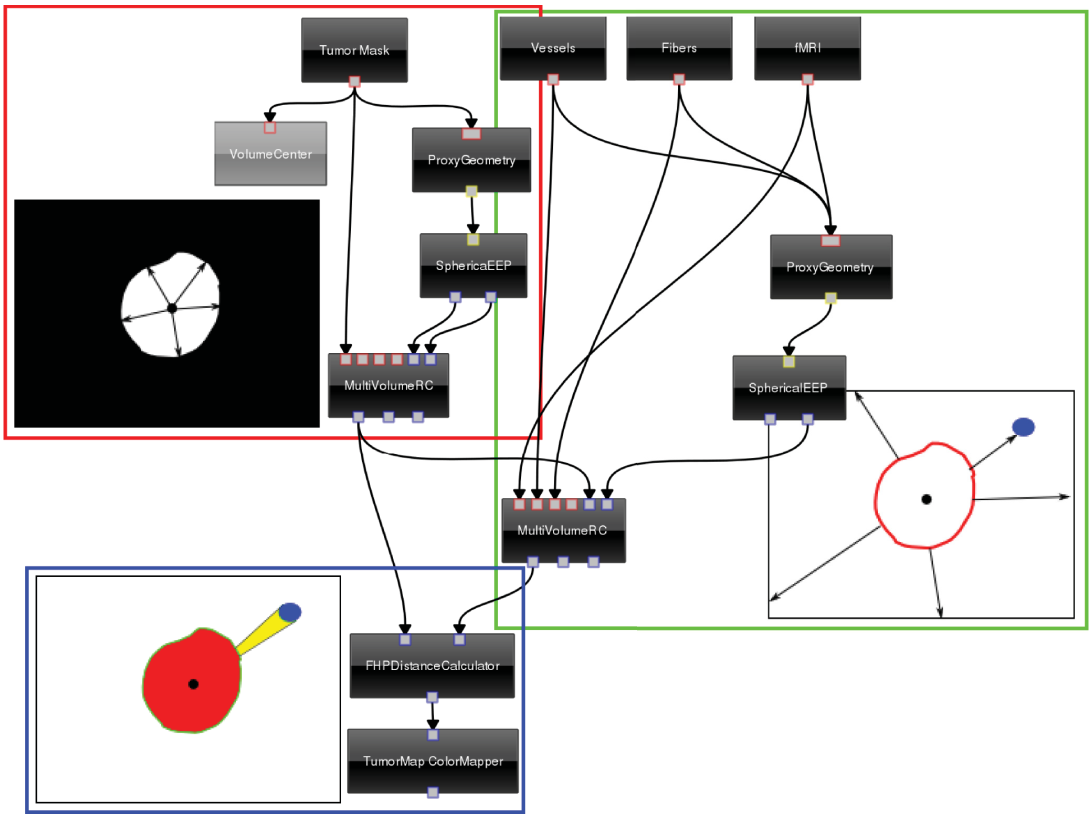


Figure 7.10: Computation of the tumor-map in Voreen: The center of the tumor is computed using the `VolumeCenter` processor and spherical projection is performed using the `SphereEEP` processor. Using an inverted transfer-function, the first-hit points with the tumor surface are computed (red box). In the second step (green box) the volumes containing critical structures (vessels, fibers, high fMRI activity) are raycasted from the surface onward. Finally, the distance of critical structures to the tumor is computed as the distance of first-hit-points from both ray casting passes and color-coded (blue box).

Chapter 8

Interactive Vessel Segmentation

Vessel segmentation is an important prerequisite for many medical applications. While automatic vessel segmentation is an active field of research, interaction and visualization techniques for semi-automatic solutions have gotten far less attention. Nevertheless, since automatic techniques do not generally achieve perfect results, interaction is necessary. Especially for tasks that require an in-detail inspection or analysis of the shape of vascular structures precise segmentations are essential. However, in many cases these can only be generated by incorporating expert knowledge. This chapter proposes a visual vessel segmentation system that allows the user to interactively generate vessel segmentations. Multiple linked views allow the user to assess different aspects of the segmentation and depict different quality metrics. Based on these quality metrics, the user is guided, can assess the segmentation quality in detail and modify the segmentation accordingly. One common modification is the editing of branches, for which we propose a semi-automatic sketch-based interaction metaphor. Additionally, the user can also control the shape of the vessel wall or the centerline through sketching. To assess the value of our system we discuss feedback from medical experts and report the results of a thorough evaluation we have performed.

8.1 Introduction

Vessel segmentation is an important preprocessing step in many medical applications, such as surgery planning or medical diagnosis [CCA$^+$05]. A crisp 3D visualization of vasculature can only be achieved when using high doses of contrast agent or long scanning procedures, which is often not possible or desirable. While line detection filters [SNS$^+$98] have been used to improve the visualization of vessel structures, they fail to detect deformed vessels which are often the parts most relevant for the user. Therefore, segmentation can still be considered as the gold standard for extracting vasculature prior to visualization as well as for performing a quantitative vessel analysis. The automatic segmentation of vascular structures has been an active field of research for over a decade. However, since a variety of factors make it such a challenging task, no fully automatic solutions exist. Lessage et al. [LABFL09] even state that aiming for a generic, flawless segmentation framework is probably illusory. They argue that a wide variability in shape, possibly caused by aneurysms, stenoses, calcifications or stents, together with the typical challenges in medical image segmentation, i. e., low resolution, noise, artifacts and contrast, make vessel segmentation such a tough problem. Nevertheless, segmentations for these difficult cases are still needed. While there are semi-automatic solutions (see Section 8.2) that aim to integrate the strengths of interaction and automation, we feel these do not fully utilize the power of visualization, which could make them potentially more intuitive and thus usable. Furthermore, some of these systems approximate the cross section using circles or ellipsoids (therefore not giving full control to the user) or require specialized hardware (see Section 8.2). This chapter proposes a visual vessel segmentation system that allows the user to edit a vessel's centerline and its surface in several linked views. These interactions are part of a two step workflow, where the user first focusses on the centerline, which is the basis for the subsequent segmentation. After an initial centerline has been defined through sketching, it can be modified and its quality can be assessed before an initial segmentation is computed. The system has been designed to generate surface segmentations of high precision, which are required in many application cases, such as vessel quantification as well as blood flow simulation based on real-world data [CCA+05]. Our system allows the user to improve the surface segmentation by sketching. To limit user interaction and promote efficiency we employ uncertainty visualization as a tool to convey the reliability of the intermediate results to the user and intuitively guide the input to introduce certainty in the system.

To the best of our knowledge this is the first semi-automatic vessel segmentation tool that gives the user full control over the centerline and the vessel surface. We propose new methods for visualization, interaction and segmentation to facilitate high efficiency and precision:

- To maximize relevant information for the user in one view without requiring further configuration of the rendering, we developed a new type of Curved Planar Reformation (CPR), the Importance Driven CPR.
- To add new centerline segments, we propose a novel sketch-based interaction technique, which resolves ambiguities and inaccuracies in the user input.

- We propose a seed volume generation technique to reduce computation and user interaction time required to perform the actual vessel segmentation.
- To segment the vessel surface we use a probabilistic random walker segmentation [Gra06], which would normally require a large number of user-placed seeds to segment vessels. Instead, we exploit the previously sketched centerline to automatically generate seed volumes.

8.2 Related Work

Since the literature on vessel segmentation is too extensive to discuss in its entirety we are focussing on interactive techniques and discuss the main classes of algorithms to motivate the choices we made while designing our system. We refer the reader to the survey by Kirbas and Quek [KQ04] for an overview of additionally existing techniques. Olabarriaga and Smeulders [OS01] point out in their survey on interaction in medical image segmentation that fully automatic methods sometimes fail and require intervention by the user. They also emphasize that interaction is usually only marginally discussed in segmentation papers, which is a tendency still valid today. However, there are exceptions which describe the most relevant techniques for our approach. Direct drag-and-drop modifications of the segmentation have been proposed by Timinger et al. [TPvB$^+$03], which have also been incorporated into our system. Kang et al. [KEK04] discuss editing tools for medical image segmentation in general. Their approach allows to modify segmentation masks, but does not allow an iterative processing such that the modifications are fed back to the segmentation algorithm. We therefore consider this approach a post-processing of the segmentation instead of a true integration of interaction. Saad et al. [SMH10] propose a system in which the user can manually post-process probabilistic segmentations. Poon et al. [PHA07] extend the livewire technique [FUS$^+$98] for vessel segmentation. However, their technique does not extend to 3D due to the too high computational cost.
Several approaches are focused on or build around the centerline specification of a vessel: Serra et al. [SHCP97] discuss a system in which the user can specify the centerline by manually tracing it with a 3D input device. The extent of the vessel can then be modelled as a tube by manually adjusting the diameter for each node of the centerline. Owada et al. [ONI$^+$08] present several approaches for sketch-based interactive segmentation. Similar to our vessel sketching approach, their techniques make use of the *sweep surface*, an extrusion of the user sketch into the view frustum. From the sketched line two adjacent strokes are automatically generated, which are used to compute the segmentation of the desired structures. However, their axis-

tracing tool requires a clear view of the structure to segment and follows an indirect approach to fit into the Volume Catcher pipeline [ONI05]. Abeysinghe and Ju [AJ09] discuss a tool to interactively determine skeletons of intensity volumes by clicking and painting on a 3D isosurface rendering. The proposed technique first precomputes a skeleton of an intensity volume and then uses the user input to correct topology errors. However, the resulting skeletons shown in the paper do not look smooth, and an isosurface rendering could be problematic for data sets with a low signal to noise ratio. Kohlmann et al. [KBKG09] propose a contextual picking approach based on ray profiles which is able to select structures after the user has clicked on a 3D rendering. The authors also propose interactive calculation of centerlines as application case, but the resulting line is not centered in the vessel and spatial coherency along the sketched line is not exploited. Jeong et al. [JBH$^+$09] present a system called NeuroTrace, which allows the segmentation and visualization of neural processes in large data sets. To make the system scalable the segmentation result is approximated by a set of elliptical cross sections. It should be further noted that branching is not supported by this technique. Helmstaedter et al. [HBD11] propose a system for neurite reconstruction that relies on the skeleton of a neurite to compare and merge reconstructions by different users. Wang and Smedby [WS10] discuss integration of automatic and interactive methods for coronary artery segmentation. Their system allows specification of additional seeds for the virtual contrast injection algorithm and modification of centerline end points. For analysis purposes the vessel is segmented using a level set approach in 2D in a CPR and a cross-sectional view. This segmentation can also be post-processed by the user with a tool called *repulsor*. Aside from the discussed shortcomings we feel that the mentioned systems only utilize visualization as far as necessary but do not explore how visualization can be exploited to actually improve segmentation results.
La Cruz [Cru03] evaluates the accuracy of centerline calculation techniques on cross-sectional slices. Based on these findings we chose a ray casting-based approach utilizing user specified thresholds.

8.3 Design Considerations

We have decided to come up with a semi-automatic, visually guided approach as a solution to the vessel segmentation problem. The goal was to exploit intuitive interaction metaphors in order to generate new or improve existing vessel segmentations and thus combine the strengths of humans and computers. This has been achieved by building a segmentation system that allows the user to intuitively and efficiently segment vessels. Although the system is built up on top of automatic segmentation

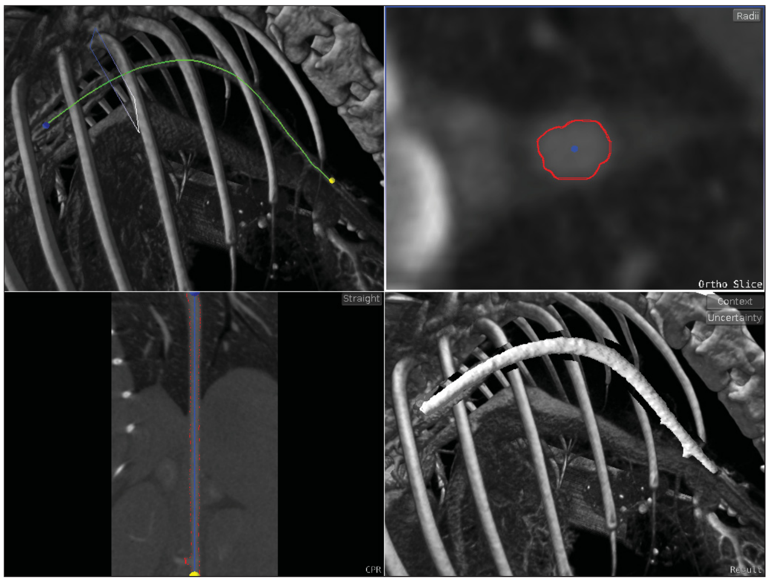


Figure 8.1: System Overview: Main 3D view (upper left), orthogonal slice view (upper right), CPR (lower left), 3D result view (lower right).

algorithms, we are still able to provide full flexibility to the user.
Since a vessel's centerline is often used in the vessel segmentation process [LABFL09], we have decided to center our system around this compact vessel representation. Additionally, centerlines are commonly used in vessel analysis software and for visualization purposes [PO08]. We therefore consider centerlines as a part of a complete segmentation and want the user to be able to modify them. In order to generate a centerline from segmentation results a skeletonization algorithm has to be applied. Besides introducing additional computations, these algorithms can introduce other problems [BFC04] like jagged lines or spurious branches, which may require additional smoothing or pruning. Furthermore, user modifications would be lost after recomputation of the centerline, e. g., when the segmentation has been modified through user input. As we will discuss in this chapter and as also postulated by Lesage et al. [LABFL09], going from centerline to segmentation is less problematic, and hence the centerline can be used as a starting point for an efficient and accurate segmentation. When segmenting the surface of a vessel, the user should have the ability to modify the result in an efficient and intuitive manner, without having to navigate through each slice intersecting the vessel. Furthermore, since we aim at high precision, we do not want to explicitly enforce smoothness constraints on the surface segmentation. The random walker algorithm [Gra06] is widely used for

image segmentation since it nicely complies with these criteria. It is a probabilistic segmentation algorithm that has been successfully utilized before in an interactive segmentation system by Praßni et al. [PRH10]. Top et al. [THA11] propose a very similar system. However, Praßni et al. point out that no thin structures, potentially intersecting multiple slices, could be segmented with their approach, since that would require the user to set an unmanageable number of seed points along the whole extent of the target structure. Because this would hamper the application of the random walker approach for vessel segmentation in general, we have overcome this limitation by exploiting a semi-automatic seed placement strategy. This has not only the benefit that it reduces user input but it also minimizes the computation time itself.

While the concepts introduced above allow the user to influence a segmentation by first defining the centerline and then modifying the segmentation result itself, they do not provide any clues about areas requiring interaction. Only by providing appropriate visual feedback, the user will be able to judge the current segmentation result and thus be able to assess the accuracy of the segmentation to be generated.

By definition, a centerline should run through the center of a vessel having potentially a circular cross section. However, in abnormal cases where the vessel shape is deformed, the situation might be less clear and further inspection by an expert user may be required. Therefore, we use the shape of the vessel cross section for each point along the centerline as an indicator for the centerline reliability. We define the *centerline uncertainty* $cl_u = 1 - \frac{r_{min}}{r_{max}}$ to be 0 for circular shapes and to increase with irregularity. In order to convey this information intuitively, we exploit a green (=certain) to red (=uncertain) color mapping. To depict the *surface uncertainty* s_{uc}, we exploit the probability field of the random walker algorithm, and display uncertain regions also in red. Thus, we are able to guide the user during the process when generating a precise segmentation (see Section 8.4).

Because an intuitive use of the system was one of the most important design goals, we decided to base the user input mostly on sketch-based interaction metaphors. The user can quickly sketch the vessel tree on a 3D rendering in order to extract an initial centerline. In our expert evaluation we were able to show that this is beneficial when compared to alternative techniques (see Section 8.6). As discussed in Section 8.2, sketching has been used before in the context of vessel segmentation. However, our approach is different in the sense that we directly transform the drawn strokes into a centerline. Furthermore, we support a semi-automatic linking of adjacent centerlines. When modifying the segmentation itself later on in our workflow, sketching can also be used, as it is a commonly used technique in segmentation tools. By initiating the surface segmentation technique based on these seeds and sketched centerlines, we

inherently achieve a coherent segmentation modification, since the algorithm propagates the seeds in three dimensions. Furthermore, it allows persistent segmentation editing, since the user modifications are still valid after changing the centerline or seeds and rerunning the segmentation process which is not the case for all interaction techniques for segmentation manipulation [KEK04].
We have selected four views (see Section 8.4) with the goal to provide all information which is necessary to perform a vessel segmentation efficiently and effectively. During the design of our system we have also considered the data structures used for representing the segmentation results. Since we want to achieve a precise segmentation, approximation approaches such as truncated cones, fitted ellipses [JBH+09] or convolution surfaces are not appropriate. Instead we have decided to directly store and visualize a segmentation volume.

8.3.1 Workflow

The first step in the workflow (see Figure 8.2) is the optional usage of an automatic vessel segmentation algorithm, which generates a centerline that could be used as a starting point for our system. Additionally, the user has to provide an intensity window for vessels, which is done interactively in a slice view of the volume. This intensity window can be rather broad, as it is only used to eliminate voxels clearly belonging to the background. We found that this window needs only little adjustment when using the same scanning protocol. The user can now use the sketching functionality to insert own centerline segments. To further modify the centerline, points of the centerline can be dragged in 3D. This dragging functionality is also available in the orthogonal slice view as well as the CPR. The induced movement is smoothly propagated to neighboring points on the centerline. Pruning of erroneously detected segments can also be performed easily in these views. When the user sketches new centerline segments these are automatically connected to the existing tree by considering their distance to the existing centerline segments.
When the user is satisfied with the centerline skeleton, which identifies the part of the volume to be segmented, the surface segmentation can be started. The user can then inspect the result and modify it by placing additional seeds in one of the 2D views. To guide this seed placement, segmentation parts with a high degree of uncertainty are highlighted. Finally, after the centerline and the surface have been confirmed by the user, the resulting segmentation mask can be passed on to other applications for further analysis.

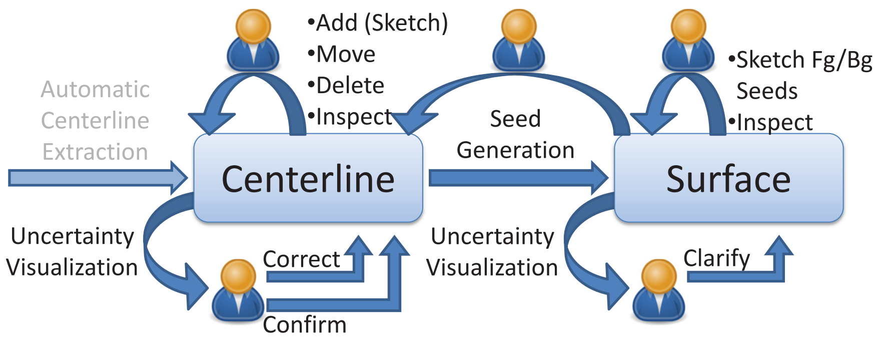


Figure 8.2: Workflow: The user can modify centerline and vessel surface. Editing of both aspects of the vessel segmentation is sketch-based and driven by uncertainty visualization. The centerline determines which parts of the vasculature are to be segmented. Because modification of the surface is not done in a post-processing step but by sketching seeds, the user can keep editing the centerline without losing any work done on the surface.

8.4 Visualization Techniques

Main 3D View The main 3D view (see Figure 8.1 (upper left)) shows a direct volume rendering of the data set, since standard 2D views are not sufficient in many cases, i. e., when vessels cannot be completely captured with a single slice. In addition to the centerline of the segmentation, the 3D view contains cues to support mental linking of the other views. The centerline geometry is considered as the main communication and interaction element in the main 3D view, and is thus overlayed over the vessel. Although the resulting depth ordering is not correct, we found it to be visually more expressive using a geometry integration approach like the one presented by Scharsach [Sch05]. We have tested this technique, and it seemed to result in a rather occluded geometry, which hampered the perception of the uncertainty encoding. Therefore we decided to render the geometry over the volume, ensuring compatibility with all data sets, and lessening the requirements on the transfer function while clearly displaying all color cues. As further discussed in Section 8.5, we consider centerline positions of irregular cross sections as potentially uncertain and color them red. A user certainty value based on the distance to the next user confirmed or corrected point along the centerline is added to this shape uncertainty.

Orthogonal Slice View While the 3D view provides a good overview of a vessel, it does not allow the inspection of the vessel's interior. Therefore, a cross section view is crucial. It further supports an in-detail inspection of the shape at a certain centerline point and provides an important basis for the interaction techniques allowing to

modify the centerline as well as the segmentation results (see Section 8.5).
After the segmentation has been calculated, we derive and visualize the uncertainty by using a simplified version of the approach proposed by Praßni et al. [PRH10] to guide the user interaction. Similarily to their approach, we exploit isolines for the depiction, but we limit the display of these lines to two, because the area of uncertainty turned out to be quite small in practice. We believe that this rather small uncertainty region is due to the large number of seeds that we generate automatically.
CPR The curved planar reformation [KFW+02] is another well established visualization technique that allows to inspect a vessel's interior and therefore also has been integrated into our system. Another important purpose of the CPR view is the navigation through the orthogonal slices by hovering the mouse over the centerline and thus causing change in the orthogonal slice view. As suggested by our medical partners during the expert evaluation of the system (see Section 8.6), we use colored spheres at the start and end positions for the currently active segment.
While for the other CPR techniques, their implementation is rather straightforward, rendering additional geometry (centerline, seeds) in the straightened CPR requires some extra effort. This is because the straightened CPR is not rendered in a regular cartesian coordinate system. We therefore render the proxy geometry for the CPR first and then use the resulting image containing texture coordinates to render seeds and centerline in an OpenCL kernel.
Importance Driven CPR The implemented classic CPR variants rely on a static vector of interest, which may not necessarily point to the parts of the vessel needing further attention. While the vector of interest can be rotated to point in a direction that allows the user to inspect and correct problematic parts, this requires user interaction; furthermore, there may be multiple directions of interest. There are variants of the CPR that display more than one slice of a vessel, like helical CPR [KWFG03] or thick-CPR [KFW+02]. However, we wanted the user to immediately see problematic sections and be able to react by centerline dragging or sketching of additional seed points, which would not be possible with these CPR variants because one pixel on the screen cannot be uniquely associated with one point in the data set. We therefore propose a new type of CPR in which the vector of interest is guided dynamically by importance. Importance is determined by the medialness of the centerline as well as the uncertainty of the surface segmentation. For each point on the centerline we cast rays in multiple directions in the cross-sectional plane (we found that 150 rays gave good results), these are possible vectors of interest. Tracing of a ray stops at the point where the segmentation probability is 0.5 and sets the surface uncertainty s_{uc} as the distance to the next certain voxel, normalized with the maximum distance for all points of the centerline. The centerline uncertainty for each ray is defined as described

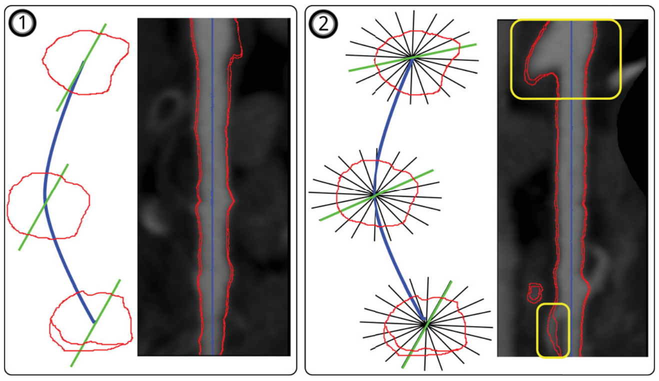


Figure 8.3: Importance Driven CPR: We maximize relevant information in the CPR rendering by making the vector of interest (green) importance driven. The vector of interest is static for a straightended CPR view (1). Based on the shape of the vessel (red) and the certainty of segmentation we dynamically rotate the vector of interest (2). This results in immediate visibility of possibly problematic centerline positions as well as surface segmentation uncertainties (yellow).

in Section 8.3. We then define the cost for each ray as $c_{ray} = max(cl_{uc}, s_{uc})$, compute a smooth sequence of vectors of interest that maximizes the overall importance to the user, and generate the CPR from these vectors instead from a static one (see Figure 8.3). The user can therefore immediately spot problems with both elements of the vessel segmentation without having to configure the vector of interest. Furthermore, a more efficient representation for providing correction through strokes is obtained since the importance driven CPR will rotate to display lengthy uncertain parts (e.g., other vessels running along the vessel to be segmented). After the uncertainty is removed the CPR will be re-evaluated to display the next most important parts.

3D Result View Inspired by the approach presented by Viola et al. [VKG04] and Straka et al. [SCLC$^+$04], we use a cutaway technique to show context while preserving an unobstructed view on the segmentation result. This is especially important because we visualize the segmentation uncertainty on the surface (see Section 8.5). To generate this rendering we first calculate a dilation on the binary segmentation result and render this volume, writing out the last hit points. We then render the context, replacing entry points with these last hit points where available. This gives us a configurable (through the threshold of the distance volume rendering) cutaway around the segmentation result (see Figure 8.4).

The main purpose of this view besides inspecting the shape of the resulting segmentation is to visualize the uncertainty information derived from the probabilistic segmentation. By masking the uncertain voxels (i.e., $[0.05, 0.95]$) from the probability

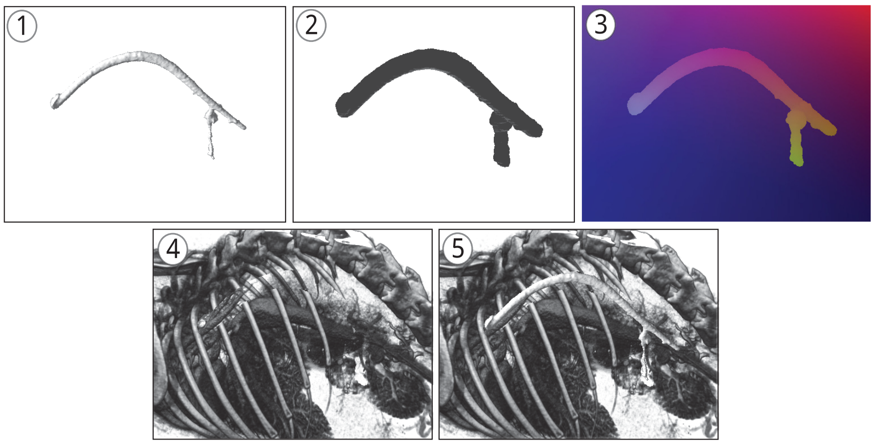


Figure 8.4: Cutaway Rendering: To render the segmentation result (1) with the context but without occlusion we calculate a distance transform (2) and render a last hit point image. This image is used to modify the entry points for the context ray casting (3), resulting in a cutaway (4). Image 5 shows the end result with compositing.

volume and applying a distance transform we generate an uncertainty volume. A large distance to regions of certainty is therefore considered uncertain (see Figure 8.8).

8.5 Interaction Metaphors

Centerline Interaction In order to add new centerline segments, the user draws a stroke in the main 3D view on top of the vessel to be segmented. To transform a user sketch into a centerline, we need to compute the correct depth value for the line drawn by the user (see Figure 8.5 (1)). In order to exploit the spatial coherence we use a technique related to the one proposed by Owada et al. [ONI05]. We generate an image from the line extruded into the view direction by reading entry and exit point for each point of the line. We can therefore also correctly handle clipping planes used in the 3D view. The intensities are based on the user provided vessel windowing (see Section 8.3.1). To find the correct depth we calculate the path with the lowest cost using dynamic programming (see Figure 8.5 (2)). As can be seen in Figure 8.5 (1), the stroke does not have to be precise, the minimal path algorithm corrects even deviations from the vessel (see Figure 8.5 (2)). To further increase the certainty that the computed centerline matches the user's intent, we incorporate the visibility based on the used transfer function. This is done by accumulating the alpha value in extrusion direction and modulating the cost with this visibility factor. Ambiguities where there is a second structure at the same position on the screen at a different

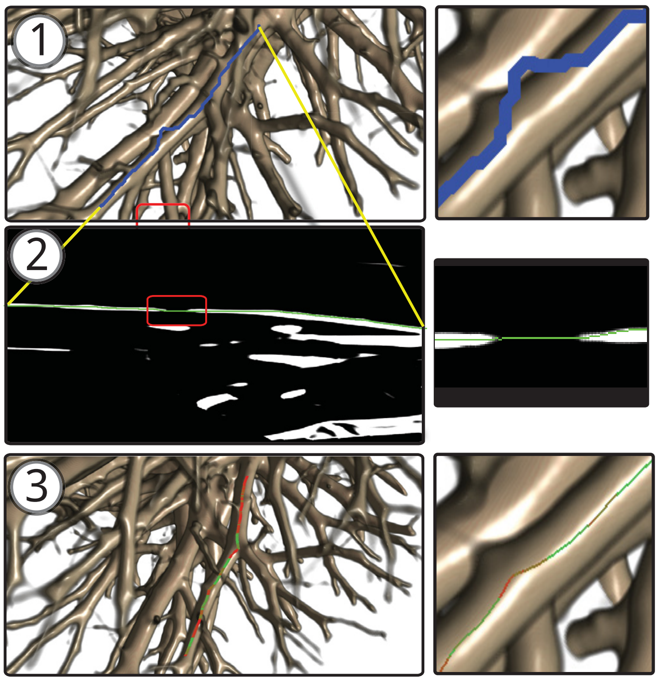


Figure 8.5: Adding new segments: 1) The user has sketched a part of the vessel tree to be segmented. The sketch does not have to be centered or even on the vessel all the time (see inlet) 2) The sketch is projected into the screen and the path with minimal cost is determined on the resulting image. Note how the gap introduced through the inaccuracy in the sketch is handled. 3) The line is centered inside the vessel using an active contour approach. This step also draws all points outside the vessel inside (see inlet).

depth are therefore resolved. Since vascular structures are usually a connected system we search for segments close to the ends of the depth-corrected line and attach them to the centerline-tree. To correct for imprecise user input and estimation of user's intended depth we center the line in the vessel using an active contour technique. In order to pull the line towards the center of the vessel we calculate the center for the current position and tangent using circular ray casting [Cru03] and pull the point in this direction (external force). The magnitude of this force is modulated by the centerline uncertainty cl_{uc}. For circular profiles the force is the strongest, while it is lower for irregular shapes. This external force is ignored for points that are outside the vessel, i. e., points with an intensity outside the user specified vessel window. Therefore, parts of the line where the user has not hit the vessel are pulled inside the vessel (see Figure 8.5 (3)).

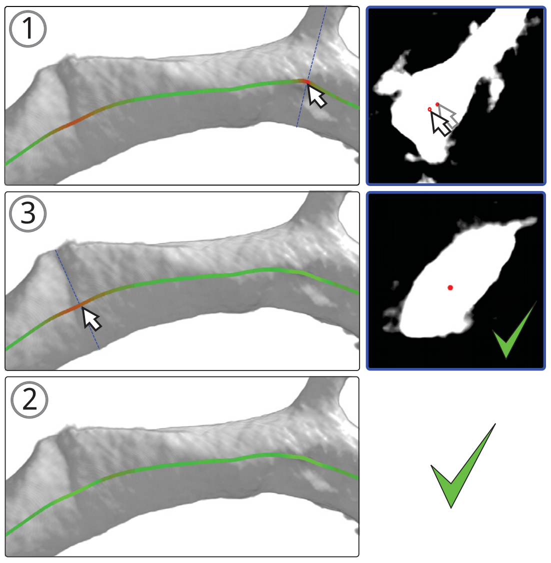


Figure 8.6: Centerline Uncertainty: Guided by red highlights on the centerline the user inspects parts of the centerline where the position is considered uncertain. 1) The user moves the mouse to the red section of the centerline to find it displaced due to the branch. By dragging the centerline on the orthogonal slice the position is corrected. This section of the centerline is therefore marked as user-corrected. 2) The user inspects another section of the centerline marked as uncertain, but is satisfied with the current position. By pressing a hotkey the position is accepted and marked as user-confirmed. 3) The centerline is now completely green, which means that it is either user-corrected, user-confirmed or has a regular shape and is therefore considered certain.

User Correction As we have just discussed we consider the position of the centerline where the circular ray casting results in circular shapes as certain. If the shape is irregular the user should confirm the position of the centerline or correct it in one of the views (see Figure 8.6).

Segmentation Modification The surface segmentation algorithm needs seed points to be set in order to segment the vessel structures. One major part of the random walker algorithm deals with solving a system of linear equations. The size of this system depends on the number of unseeded voxels. When segmenting volumes, this system becomes so large that GPU-based solvers need to be employed in order to integrate the algorithm in interactive applications. While this takes care of the computational cost, the memory requirements become the main problem. By auto-

matically generating a large number of seed points, we are able to reduce the size of the linear equation system and thus also allow the application of the technique to larger data sets. We generate foreground and background seed volumes based on conservative estimates of the inner and outer radius of the vessel. A voxel is considered as background voxel if it is not within the outer radius of any centerline segment. If a voxel is inside the inner radius of a centerline segment the voxel is marked as foreground voxel. Only the remaining voxels, usually a small fraction of the volume, need to be segmented by the random walker algorithm. This process is shown in Figure 8.7. From each point of the user confirmed centerline we cast circular rays. The inner radius is the minimum distance a ray can travel while hitting no background voxels. The outer radius is the minimum distance beyond which a ray will surely hit only background voxels. We use a safety margin in both calculations to account for a limited number of rays. This approach can degenerate in two ways: The inner radius could become zero and only the centerline itself would get rasterized into the foreground seed volume at this position. This is similar to the approach proposed by van Bemmel et al. [vBVN04] who initialize a level set segmentation on the centerline. Alternatively, the outer radius can become large for noisy or low contrast data sets, which may happen especially at branches. However, this only increases computational cost and does not lead to an unstable processing. Based on the calculated radii we now build the foreground and background seed volume (see Figure 8.7 (2)), and the random walker algorithm segments the layer between both. The resulting segmentation is displayed in Figure 8.7 (3). Because of the large number of seeds, the number of unseeded voxels is comparatively low and thus larger data sets (up to 440x300x1100) can be handled. Furthermore, times of user-inactivity are reduced compared to using only a small number of hand drawn seeds.
Guided by the highlighted areas shown on the segmentation, the user can move the mouse on these sections to inspect them in the linked views (see Figure 8.8).

8.6 Evaluation

In their survey on interaction in medical image segmentation Olabarriaga and Smeulders [OS01] list accuracy, repeatability and efficiency as the main criteria, which influence the quality as well as the usability of a segmentation system:

- *Accuracy* is the most common criterion and indicates the degree to which the delineation of the object corresponds to the truth.
- *Repeatability* evaluates to which extent the same result would be produced over different segmentation sessions.

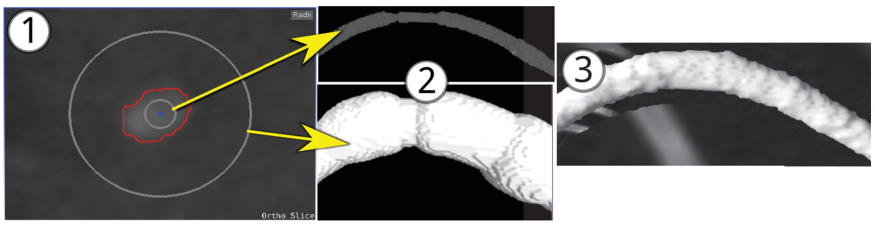


Figure 8.7: Seed Volume Generation: 1) Based on the centerline the algorithm calculates conservative estimates of the inner and outer radius (gray circles in slice view). 2) From these estimates seed volumes for foreground (upper) and background (lower, inverted) are generated. 3) The resulting segmentation, which is also visible as red line between the inner and outer circle in the slice view.

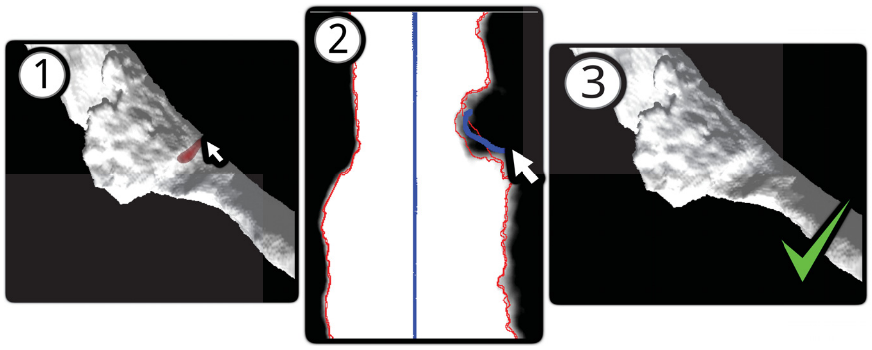


Figure 8.8: Surface Segmentation Uncertainty: 1) Regions where the segmentation of the surface is uncertain are highlighted in red. When hovering the mouse over the surface the orthogonal slice view and CPR are linked accordingly. 2) The user corrects the uncertainty by sketching additional seeds in the CPR. 3) The uncertainty is fixed.

- *Efficiency* of the computational (automatic) part of the application is measured in terms of the time need by the computer to generate the result. Efficiency of the interactive is determined by the amount (e.g., number of clicks) and nature (i.e., complexity) of the user interventions [OS01].

Accuracy can be more directly exploited as a quality criterion when dealing with automatic segmentation techniques. When using interactive approaches an expert user can modify the result until it best reflects his or her expert knowledge, which is called *potential accuracy* [OS01]. In our system, this can be done by sketching additional seed points, which given in a dense layout directly define borders. To investigate the degree of accuracy that can be achieved when using our system we evaluated the outcome of our system using the carotid bifurcation algorithm evaluation framework [HZF+11]. This framework is a good fit to evaluate our system because it focusses on a small part of the vasculature and provides a standardized evaluation system, comparing the submitted segmentation to averaged ground truth, provided by three experts. The data set consists of 56 computed tomography angiography (CTA) images (acquired on different scanners) of the carotid arteries, some with stenosis. We did not utilize the starting points provided with the data sets to perform our segmentation but instead sketched the centerline. We segmented all data sets, and our results were compared to the ground-truth using three metrics: Dice similarity measure [Dic45], mean surface distance (MSD) and maximum surface distance. Table 1 shows minimum, maximum, and average values for each metric. The average Dice measure was 90.0% — in only three cases the Dice similarity measure between our results and ground-truth was less than 85.0%. Table 2 shows the average grades of our system and of the three experts. Our average results are within a few percent of the inter-observer variability, which we achieved with one data set. We attribute this difference mostly to the approach used to generate the ground truth data: a manual, probably very time consuming process in which the cross-sectional contours were modeled using splines. Resulting segmentations are therefore smoother than the output of our system, which is more closely coupled to the data. This is most notable at secondary branches which were not to be included in the segmentation and account for the relatively large maximum surface distance. We argue that a more circular cross-section at these branching points is not more true to reality than the protrusion which remains when removing the branch with a few sketched background seeds and therefore did not spend too much time modeling these sections to match the ground truth. Overall, we conclude that our technique enables users with no medical training to create precise vessel segmentations.

Repeatability is mainly influenced by difference in user input or judgment which may lead to a high variability of the results [OS01]. To test for the influence of differences in sketching, we used a ground truth segmentation of a vessel segment in a contrast enhanced CT scan, and asked three users to perform a segmentation with our system. We instructed the three novice users having no medical background in the use of our system, and explained the task to be performed. The result was compared to the ground truth segmentation using the dice metric. The obtained dice scores (93.7%, 94.4%, 94.2%) confirm a good repeatability of segmentation results.

Efficiency of semi-automatic systems can be estimated by considering two key components: usability of the interactive parts and performance of the automatic parts. When judging the amount of user input one should keep in mind that it is not the aim to construct large vessel trees manually but to successively add or correct a reasonable amount of specific vessels. Like other authors working on sketching techniques, we believe that the sketching directly communicates that no precise interaction (i.e., exact tracing of contours) is required. We argue that sketching vessels on a 3D rendering is as intuitive and predictable as possible, which our domain experts confirmed. The placement of additional seeds for the surface segmentation and movement of the centerline are only necessary where the automatic approach fails and have not to be performed for each slice. Regarding the performance of the automatic parts, we have implemented the surface segmentation and sketch-extrusion on the GPU to facilitate a smooth interaction.

Expert Evaluation To evaluate the practical value of our approach we have organized a feedback session with our medical partners. The group consisted of four researchers interested in the vascular system and working with scans from different modalities on a daily basis. One possible application case of our system is the quick segmentation of parts of the vasculature in which the user is interested. The need for this type of segmentation was confirmed for research as well as clinical use. Due to the intuitive and quick sketching of vessels the approach was seen as a viable alternative to selection of parts of a full segmentation. A significant number of the scans our partners use are small animal CT scans, which make vessel segmentation even more challenging due to the small size of the species under observation. In addition, abnormalities of the shape of vessels are common in these scans. Therefore automatic approaches which rely on strong contrast or shape-based approaches fail and manual segmentation needs to be performed. They appreciated the segmentation directly on the basis of a 3D rendering, especially for cases where one vessel cannot be cut by a single slice. Exact, user-controlled placement of the vessel wall as well as the centerline was also confirmed as an important criterion for a vessel segmentation system by our medical partners. Considering parts of the vessel with irregular shape as uncertain was perceived as a sensible approach. The red-green color scheme was intuitively understood and our technique of user-confirmed and modified points used to guide the user towards confidence about the segmentation was appreciated. We also discussed the visualization techniques used in our system and the applicability of all views was confirmed. However, the direction of the CPR was unclear and the inclusion of start/end markers in the 3D view was suggested. The incorporation of a multiple label segmentations approach was also suggested to handle plaques and similar structures.

Table 8.1: Summary lumen (generated using [HZF+11])

Measure	% / mm			rank		
	min.	**max.**	**avg.**	**min.**	**max.**	**avg.**
L_dice	81.3%	95.5%	90.0%	3	4	3.98
L_msd	0.21mm	1.31mm	0.54mm	4	4	4.00
L_max	0.78mm	8.04mm	5.96mm	3	4	3.98
Total (lumen)				**3**	**4**	**3.98**

Table 8.2: Averages lumen (generated using [HZF+11])

Team	Total	dice		msd		max		Total
name	**success**	**%**	**rank**	**mm**	**rank**	**mm**	**rank**	**rank**
Our technique	41	90.0	4.0	0.54	4.0	5.96	4.0	4.0
ObserverA	41	95.1	1.6	0.10	1.7	0.54	1.8	1.7
ObserverB	41	94.6	2.2	0.11	2.2	0.62	1.8	2.1
ObserverC	41	94.4	2.2	0.12	2.1	0.72	2.4	2.3

8.7 Limitations

Although the most important algorithms we utilize in our system are well established, no singular algorithm can guarantee success in all cases. While we integrate semi-automatic techniques to allow for easy corrections one can certainly imagine worst case scenarios (e.g., leaking contrast agent) where the segmentation process would degrade into something quite like slice by slice painting. Although adding new vessel segments proved to be robust using relatively simple methods the algortithm can fail to detect the correct vessel if the viewing angle is bad.

8.8 Implementation

The implementations of most views used in this system (DVR, CPR, slice rendering) have already been discussed in Chapter 3. For a discussion of the random walker implementation in Voreen we refer the reader to the work by Praßni [Pra11]. Figure 8.9 shows the integration of sketching into a minimal Voreen network.

8.9 Conclusion

In this chapter, we discussed the design, implementation and evaluation of an interactive, visually guided vessel segmentation system. The system has been designed for two usage scenarios: the correction of the results of automatic segmentation algorithms and the fast, user guided segmentation of parts of the vasculature that the user wants to inspect or analyze. To provide intuitive solutions for these scenarios, we have exploited a semi-automatic approach. On the interaction side, the key concept of the system is a sketch-based user interface, which allows to generate precise vessel segmentations from rough strokes quickly drawn on a 3D view. While the actual segmentation is performed using the random walker algorithm, it can be also modified through the proposed sketch-based interface. In combination with these interaction techniques, we exploit visualization techniques arranged in linked views, which help the user to assess the quality of the current segmentation and thus support its modification. In this context, a key concept is the uncertainty visualization, which conveys the reliability of the intermediate results in a concise way. We have thoroughly evaluated our system using the carotid bifurcation lumen segmentation framework and discussed opinions by a group of expert users from the medical domain. Their feedback has shown that there is a high demand for such a system in the medical/biological community. In the future it is planned to improve the system based on the suggestions by these experts, e. g., by integrating the multi-label segmentation approach. However, beyond these extensions we see also directions for future research. When extracting whole vessel trees, our system is currently limited to a sequential processing of the branches. One reason for this is the CPR technique, which allows to display only one branch. In the future we would like to investigate how multipath CPR techniques could be exploited in order to investigate multiple branches in parallel. On top of that, a formal evaluation with a larger user group should be performed, to prove the practicability of our system when dealing with everyday tasks.

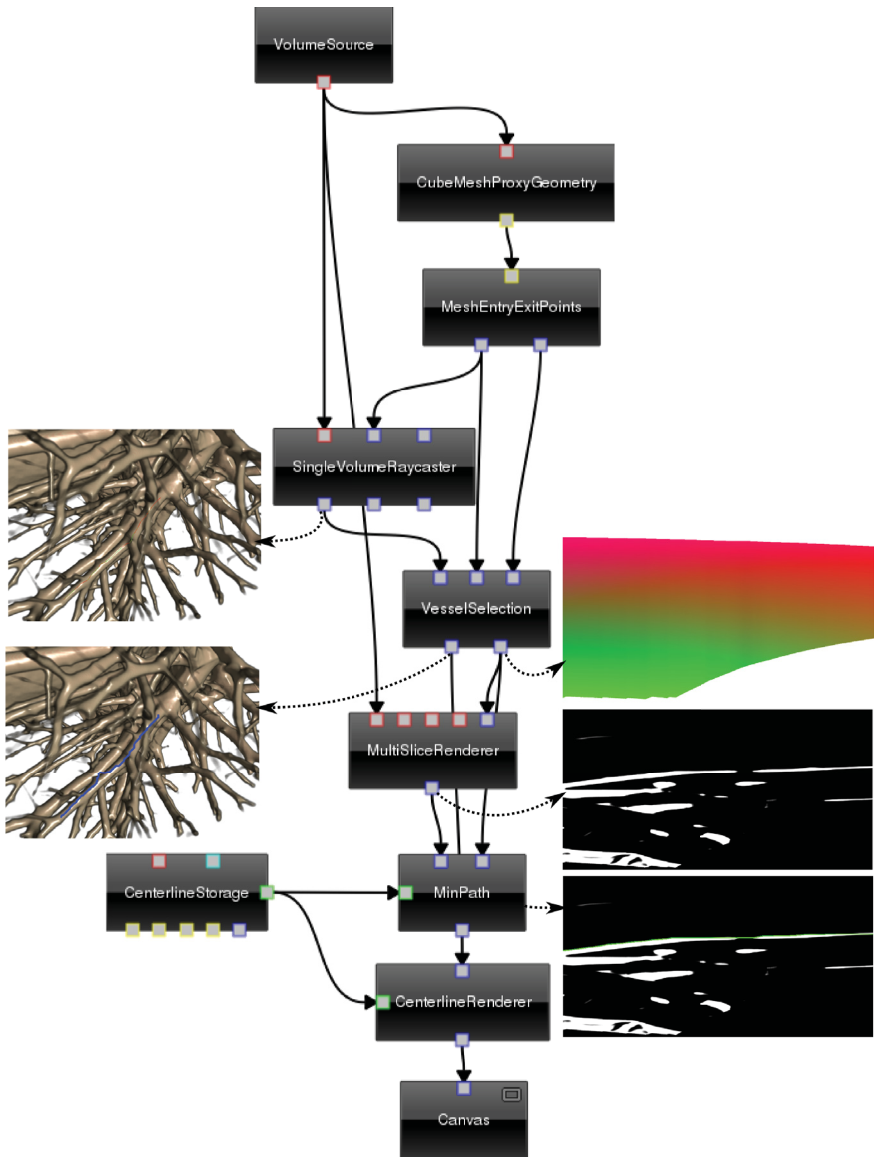


Figure 8.9: Implementation of vessel-sketching in Voreen (simplified): The `VesselSelection` processor uses the entry-exit-points textures to generate a texture containing sampling positions on the sweep-surface (top right). Next, the intensities are mapped to a cost determined by the user-specified intensity window using the `MultiSliceRenderer`. Finally, the `MinPath` processor computes a minimal-cost path across the sweep-surface and adds the centerline to the storage processor.

Comparative Visualization of Tracer Uptake in In Vivo Small Animal PET/CT Imaging of the Carotid Arteries

Cardiovascular diseases are the main cause of death in the western world. Medical research on atherosclerosis is therefore of great interest and a very active research topic. This chapter presents a visualization system that supports scientists in exploring plaque development and evaluating the applicability of PET tracers for early diagnosis of cardiovascular diseases. In our application case a cone shaped cuff has been implanted around the carotid artery of ApoE knockout mice, fed with a high cholesterol western type diet. As a result, vascular lesions develop upstream and downstream from the cuff. Tracer uptake induced by these lesions needs to be analyzed in order to evaluate the effectiveness of different PET tracers. We discuss the approach previously utilized to perform this kind of analysis, the problems arising from in vivo image acquisition (in contrast to ex vivo) and the design process of our application. In close cooperation with domain experts we have developed new visualization techniques that display PET activity in the vessel wall and surrounding tissue in a single image. We use the vessel wall detected in the CT image to perform a normalized circular projection which allows the user to judge PET signal distribution in relation to the deformed vessel. Based on this projection a quantitative analysis of a defined region adjacent to the vessel wall can be performed and compared to the artery without the cuff.

9.1 Introduction

The majority of both vascular pathologies and cardiovascular events result from atherosclerosis. A hallmark of atherosclerosis is accumulation of cholesterol in the arterial vessel wall in conjunction with infiltration by inflammatory cells forming a vascular inflammatory lesion (atherosclerotic plaque). While the thickening of the vascular wall due to the inflammatory lesion primarily leads to the constriction of

the vessel (stenosis) and impaired perfusion, the individual prognosis (myocardial infarction or stroke) is rather dependent on the morphology and inflammatory activity of the plaque [SSH10]. Therefore, an integrated approach combining morphological, functional and molecular imaging modalities to multiparametrically characterize atherosclerotic lesions seems needed to predict and prevent future cardiovascular events. This challenge is multi-facetted: it involves new imaging probes and devices, animal models and such. Even with successfully established approaches new explorative and interactive analysis strategies are needed for visualization of the resulting complex multiparametric images. This chapter describes a new visualization application to explore imaging findings in a mouse model of atherosclerosis.
This model is specifically designed to produce an atherosclerotic plaque in one carotid artery by a constrictive cuff whereas the opposite untreated carotid artery serves as a control. Here, both carotid arteries are studied simultaneously by positron emission tomography (PET) for imaging of inflammatory plaque activity and contrast-enhanced computed tomography (CT) highlighting the carotid vessel lumen and the cuff. This approach uniquely enables evaluation of the vessel lumen and inflammatory activity in different sections of operated arteries as well as in comparison to the untreated contralateral artery at all times.

In the current analysis protocol (see Section 9.3) for this type of study standard medical workstations are used. One of the more trivial shortcomings of this approach is the missing ability to view both the left and right carotid artery of a subject at a sufficient zoom level simultaneously. Standard software also does not provide a standardized visualization for tracer uptake in the vicinity of the vessel and therefore requires the user to form a mental picture of the data. Analysis needs to be performed by manually placing regions of interest and exporting results to external plotting software. Similar to other authors [RHR$^+$09, KBH$^+$10] we acknowledge the need for specialized visual analysis software to support researchers in forming or proving their hypothesis. Designing such applications can obviously only be done in collaboration with potential users. The application presented in this chapter is therefore the result of an iterative design process between computer scientists and medical researchers.

Our contributions are as follows:

- We describe the design of an application (see Figure 9.1) to compare the PET tracer uptake between two vessel segments for in-vivo PET/CT scans.
- We propose the normalized circular projection (NCP), a novel visualization technique that gives the user a standardized overview of the data surrounding a tubular structure, in our case a maximum intensity projection (MIP) of the PET signal. According to the needs of PET analysis, the NCP preserves distance

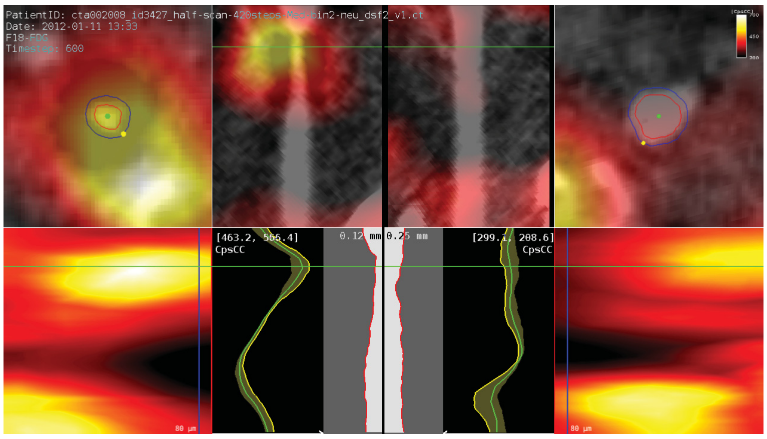


Figure 9.1: Overview: Our Application consists of a number of linked views, arranged in a symmetric layout (right screen half is control). The top half of the screen is dedicated to display the unprocessed data using orthogonal slices (outside) and CPR (inside) while the bottom half shows our proposed visualization techniques, the normalized circular projection (outside) and vessel wall analysis (inside).

to the lumen and maximum intensity.

- Based on the NCP we sample the PET data in a user-defined neighborhood of the vessel wall and generate side by side plots to facilitate an efficient visual comparison and quantitative analysis.

Our application case is the analysis of tracer uptake in the left and right carotid artery of mice. However, the system could be used for other vessel segments as well as for human subjects.

We will first discuss related work and motivate the need for our proposed techniques. Then we describe the needed medical background and our resulting design considerations, before we give an overview of the application followed by an in detail discussion of the proposed visualization techniques and interaction concepts. Before concluding the chapter we present resulting images, their interpretations by domain experts and a comparison of our techniques to previous work.

9.2 Related Work

Visualization of vessels can broadly be categorized in 3D techniques, flattening approaches and reformation techniques. 2D slice views are not well suited for vessel visualization because of the longitudinal structure that is generally not aligned with the slice directions. We will give an overview of the most relevant techniques from these categories and discuss applicability in our application case. Techniques that focus on the layout of an entire vessel tree are not included in this discussion, since we are dealing with linear segments exclusively. We will also not compare segmentation and centerline detection techniques, since both are not the focus of our work. For a survey of such techniques see Lesage et al. [LABFL09] and Kirbas and Quek [KQ03].

3D Vessel Visualization can be sub-categorized into approaches that extract geometry [BFC04, OP05, SOB+07] and direct rendering approaches [JQD+08, KGNP12]. Alternatively, approaches can be divided into model-based and model-free. Wink et al. [WNV00] segment a vessel segment after the user specifies start and end point. We evaluated a similar gradient magnitue based approach to detect the vessel wall. Yang et al. [YZH+05] discuss a 3D visualization technique to analyze stenosis based on harmonic skeletons. In our case, the vessel itself is not the main object of interest, it rather provides a context for the PET visualization. While it would be easily possible to generate a 3D rendering of the vessel extracted from the CT image and combine it with the PET signal in a volume rendering [LLHY09, KGB+09], we refrained from doing so. The reasons for this are threefold: (1) With 3D visualizations there is always an occlusion problem (e.g., it would be hard to visualize the PET signal inside the vessel), (2) correctly assessing the spatial relationship between the diffuse and unstructured PET signal and the vessel would be difficult, and (3) evaluation of intensities is not possible because of the necessary compositing. While a MIP could remedy the third shortcoming, spatial understanding would become even worse.
Recently van Pelt et al. [vPBB+10] have proposed techniques for stylized visualization of blood flow in the carotid arteries. Unfortunately, PET data is quite unstructured and of low frequency and therefore not a good basis for stylized techniques.
Gasteiger et al. [GNBP11] discuss the *FlowLens*, a focus and context technique to visualize blood flow in cerebral aneurysms. Their approach is, similar to most vessel visualization approaches, concerned with data in or on the vessel.

Flattening Approaches allow the user to inspect the wall of curvilinear structures (vessels, colon) in its entirety, and a wide range of techniques have been published [ZHT+02, ZHT05, LLS05, Mar11]. Borkin et al. [BGP+11] recently evaluated 2D (flattened) versus 3D artery representations for the visualization of endothelial sheer stress (ESS). They found the 2D representation to be superior with regards to accuracy and efficiency.
Flattening is an efficient approach to visualize scalar data like ESS, wall thickness or diameter. In our application case however, the PET signal is not only present on the

vessel wall but inside and outside of the vessel as well.

Reformation Techniques A commonly used 2D visualization technique is the Curved Planar Reformation (CPR) [KFW$^+$02], which slices a vessel along its centerline. The main drawback of CPR techniques is that not the entire vessel can be visualized at once. Kanitsar et al. [KWFG03] have extended the CPR technique to support multiple branches and introduced the helical CPR, which displays the interior of a vessel in a single image. While this unconvential representation seems to have its uses in finding calcifications, the authors unfortunately do not present feedback from potential users. Our medical partners understood what the technique did, but found the resulting images to be confusing.
Straka et al. [SCLC$^+$04] introduce the VesselGlyph, a combination of Direct Volume Rendering (DVR) for the context and CPR for the vessels. Williams et al. [WGC$^+$08] propose a technique that does the opposite, their application case is colonoscopy. Daae Lampe et al. [LCMH09] introduce the Curve-Centric Volume Reformation, which deforms a volume with regards to a curve. They also propose an inside-out projection based on this data. The images rendered using this technique are somewhat similar to flattening but preserve distance instead of shapes and angles. Angelelli and Hauser [AH11] extend this approach for flow data sets and incorporate a side-by-side view to give an overview of different timesteps. Aortic flow visualization is one of their application cases.

Applications Gerhards et al. [GRH$^+$04] discuss the *VascuVision* system, which provides multiple linked views (MIP, orthogonal slice, CPR, lumen diameter plot) to assess stenosis.
Kok et al. [KBH$^+$10] propose the Articulated Planar Reformation to visualize change in small animals. The reformation registers all bones to a segmented mouse atlas and allows the user to inspect different timesteps for each bone using comparative visualization techniques. According to the authors, their system is currently not capable to register soft tissue. They employ side by side visualizations, comparison by switching, overlays and a checkerboard pattern in their focus view.
Ropinski et al. [RHR$^+$09] discuss a visualization system for mouse aorta PET/CT scans, integrating flattening and CPR techniques. While their system might seem quite similar to ours there is one crucial difference: The described system can only handle data sets of ex-vivo mouse aortic arches embedded in parafin. This has several implications, with the most obvious being that users cannot monitor a subject over time. Scanning a parafin block also allows the usage of much higher radiation doses during the CT scan, resulting in higher quality images. In addition, the vessel wall

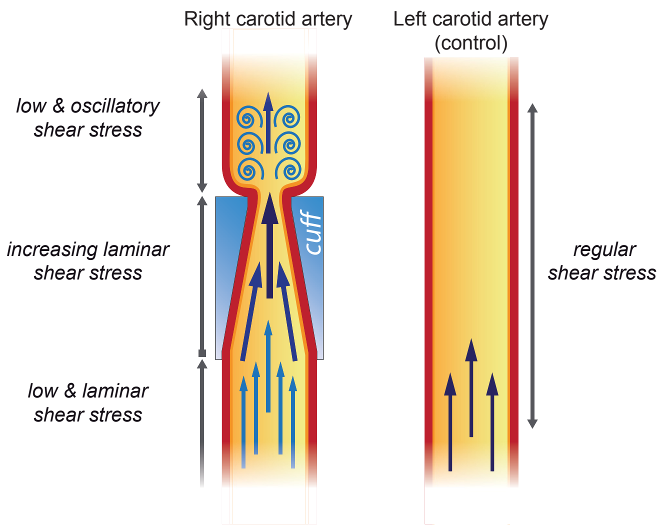

Right carotid artery
Left carotid artery (control)
low & oscillatory shear stress
increasing laminar shear stress
low & laminar shear stress
cuff
regular shear stress

Figure 9.2: Carotid Cuff: The surgically implanted cuff modifies the shear stress along the right carotid artery. The left side is used for control. (Image courtesy of Michael Kuhlmann)

has a unique density in the CT scan, whereas in in-vivo scans the vessel wall cannot be distinguished from the surrounding tissue. With the vessel removed from the body and filled with contrast agent, the PET signal is guaranteed to originate from the vessel wall. This makes flattening a viable approach. In in-vivo scans the use of a maximum intensity projection (MIP) would display hot spots farther away from the vessel, with no way for the user to determine the distance.

9.3 Medical Background

Carotid Cuff Modifications in shear stress are known to alter the expression of inflammatory genes in endothelial cells and by this means induce atherosclerosis. Therefore, surgical models where the shear-stress is modified are of interest for both basic research as well as for evaluating new imaging approaches. A well-established model in mice is based on the implantation of a cone-shaped cuff around the carotid artery creating defined regions of low, high and oscillatory shear stress within the common carotid artery (see Figure 9.2). Applying this model in ApoE knockout mice, which are fed a high-cholesterol diet, two atherosclerotic plaques are forming with a more stable phenotype downstream of the cuff and highly inflamed unstable phenotypes upstream of the cuff [KCH$^+$12].

Data Acquisition PET experiments were carried out using the high resolution (0.7 mm full width at half maximum) quadHIDAC small animal PET-scanner (Oxford Positron Systems, Weston-on-the-Green, U.K.). ApoE$^{-/-}$ mice (25-30 g) were anesthetized with isoflurane (1.5%/0.3 L min^{-1}) and placed on a heating pad to maintain body temperature for insertion of tail vein catheters. One hour after intravenous injection of 10 MBq F-18-FDG in 100 μl 0.9% saline list-mode data was acquired for 15 minutes. Subsequently, the scanning bed was transferred to the computed tomography (CT) scanner (Inveon, Siemens Medical Solutions, U.S.) and a CT acquisition with a spatial resolution of 80 μm was performed for each mouse. Data acquisition changes slightly when using different tracers, F-18-FDG was used for all images shown in this chapter.

Current Analysis Protocol The current analysis protocol for this study is performed using standard medical workstation software in combination with spreadsheets. After co-registration the data set is aligned and resampled so that the right carotid artery (with cuff) corresponds to one of the primary axes of the volume. Following this resampling, nine cylindrical regions of interest (ROIs) of identical dimensions are manually placed along the arteries (three upstream from the cuff, three over the cuff and three downstream from the cuff), corresponding to the different ways shear stress is modified by the cuff (see Figure 9.2). Statistics computed for the regions are then transferred to a spreadsheet application, which is also used for visualization purposes (line plots).

Aside from requiring a reorientation and resampling of the data and the manual placement of ROIs we felt the main drawback of this protocol is the missing explorative aspect. The user has to define the ROIs for analysis, transfer the results to a separate application, look at the plots and then has to come back to the original application to look for sources of anomalities in the results. Changing parameters (e.g., the extent of the ROIs) is also tedious due to the amount of work required. Comparing both arteries side by side is not easily possible at sufficient zoom levels, and to find the source of increased PET activity in one of the regions requires scrolling through the slices. Furthermore, the modelling of the vessel wall using a stack of (solid) cylinders leaves room for improvement, which is hardly possible with standard medical workstation software.

9.4 Design Considerations

The protocol described in the previous section led to the design of the current application, with the following aims:

- Provide the user with linked views for an easy in detail inspection and comparison of the co-registered PET/CT data set.
- Support a quick location of spots of high activity in the PET data, with immediate display of the most relevant information (intensity value, distance to vessel wall).
- Generate analysis results comparable to the previous protocol, but include more a priori information about the vessel shape in the process.
- Support easy comparative visualization of the analysis results by standardized renderings.
- Facilitate experimentation and allow calibration with histology results by easy configuration of analysis parameters.
- Support calculation and display of the vessel diameter.

When looking at the categories of comparative visualization by Verma and Pang [VP04], feature level comparison certainly seems to be the most desirable approach. Unfortunately, feature detection in PET data is extremely challenging due to the low spatial resolution and low frequency data. Data level comparison (e.g., by calculating the difference between volumes) is also not meaningful in our case because of the biological variance and resulting spill-in from other vessels in the region to be analyzed. This leaves us with image level comparison, which can be sub-categorized in side by side views and image differences. Image differences are unsuitable for the same reasons we cannot exploit data level comparison. Since PET data is low frequency and we are looking at a relatively small region we found techniques like switching or checkerboards [KBH$^+$10] unneccesary. Fortunately, side by side views can be enhanced by suitable reformation and/or projection of the data [LCMH09, AH11, KBH$^+$10]. As discussed in Section 9.2, the techniques currently available to display a vessel in one image are not well suited for our purpose, which led us to the development of our proposed techniques.

9.5 Application Overview

This section gives a general overview of the application, the utilized visualization techniques will be discussed in detail in section 9.6.

Preprocessing Reconstructed image data sets are co-registered based on extrinsic markers attached to the multimodal scanning bed using a rigid body registration. For best results our application requires a segmentation of the vessels to be analyzed (see Section 9.6.2). Depending on the data set, this can be accomplished by thresholding, manual segmentation, using a standard semi-automatic medical segmentation approach or the application described in Chapter 8. Centerlines can be generated from the segmentation using a potential field based technique [CSYB05]. Because of the small vessel diameter inside the cuff the algorithm can fail and we therefore provide capabilites to correct the automatic detection or specify a centerline manually. We utilize Catmull-Rom splines to smooth the centerline, which is important for the CPR as well as our visualization techniques since both depend on smoothly changing tangents.

Screen Layout and Views Our application consists of a number of linked views (see Figure 9.1), which can be conceptually organized in two ways: Left-Right and Top-Down. The left half of the screen displays the right carotid artery and vice versa, as a radiologist would expect from a standard 2D slice view. The original (albeit reformated) data is displayed in the upper half and the advanced visualizations and analysis results in the lower half. By mirroring the views vertically we generate a layout that facilitates an easy comparison between the corresponding visualization on cuff and control side.

Curved Planar Reformations together with an orthogonal slice view is a combination that is frequently used in vessel visualization tools since vessels are usually not aligned with the slice directions. These techniques are also accepted by our medical partners and were therefore the first to be integrated in our tool. The position of the orthogonal slices is linked between both sides and we indicate the current position in the CPR views. In these views we provide a set of standard tools to perform intensity and distance measuring. To display the entire PET data within the vicinity of the vessel in one image we integrated our novel *Normalized Circular Projection*, described in the next section. In addition, we automatically perform an analysis of the PET activity, integrating knowledge about the vessel shape. The activity plots are rotated by 90 degrees to enable side by side display with the other visualizations.

9.6 Visualization Techniques

9.6.1 Color Mapping

In search for a color mapping that is accepted by our potential users and perceptually advantageous we consulted the literature. For CT data a grayscale mapping is the de-facto standard and is able to reproduce the high frequencies found in these images [RT98]. For the PET image, the heated body color map and the rainbow color map were favored by our partners because they can be scaled in such a way that the regions of maximum PET intensity can be recognized easily. We chose the heated body color map because of its perceptual ordering [BTI07]. The combination of both modalities is quite similar to color mapping on a surface, since luminance needs to be combined with color. Borland et al. [BTI07] suggest using an isoluminant color map in this case. We found that the PET signal would disappear in a substantial portion of the data set (i.e., in regions of low CT intensity), which was unacceptable for us. Instead a simple blending was implemented, which slightly reduces CT perception in favour of visibility of the PET signal (see Figure 9.3). With the PET data being the focus of this analysis we found this tradeoff to be justified. The user can blend between PET and CT in case the combination of both obstructs the clarity of the rendering.

For our medical partners to rely on the findings of our analysis it was of great importance to see where the data came from that generated the images. We therefore clearly indicate the inner vessel wall (red), analysis region (blue) and centerline (green) in a consistent color scheme over all views.

9.6.2 Normalized Circular Projection

To judge whether an area of high PET activity indicates a plaque in the vessel wall or just a spill-in from another structure close by, the position of maximum intensity as well as overall spread are important. We therefore introduce the *Normalized Circular Projection* (NCP). Each pixel in the resulting image represents a ring around the vessel, orthogonal to the tangent of the centerline. Because of the substantially varying diameter of the vessel we normalize the radius to the inner vessel wall. This is equivalent to a straightened reformation of the vessel, followed by an outward shift of the vessel wall to form a cylinder (see Figure 9.4). We then project in concentric circles around this cylinder.

Figure 9.5 illustrates the technique: For each row of the resulting image (step on the centerline) a radial ray casting is performed to first find the inner vessel wall in the

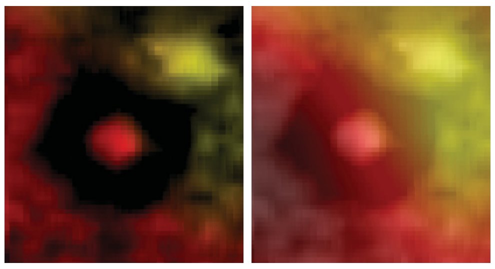


Figure 9.3: Compositing of PET and CT: While a modification of hue by the PET transfer function renders the CT data clearer (a), the PET signal vanishes inside the cuff. Since the PET data is more important in our case we use a simple blending between PET and CT (b), which can be configured by the user in case one modality should be inspected on its own.

CT data set. After hitting the vessel wall we then sample the PET data set from that point onward. All n-th PET samples have therefore the same distance to the inner vessel wall. In contrast to other reformation techniques [AH11, LCMH09] we do not need to concern ourselves with a consistent computation of normal and binormal. This is because the following MIP makes our technique rotationally invariant.

For the detection of the inner vessel wall we have evaluated several approaches: Setting a global threshold produces unacceptable results because the intensity of the vessel inside the cuff is lower due to spillout and/or partial volume effect. Using an adaptive threshold relative to the intensity at the centerline did not significantly improve the results. We therefore implemented a maximum gradient based approach, which has been succesfully employed to detect the vessel wall in order to compute the centerline position [La 04]. In combination with anisotropic diffusion filtering this did improve results but still produced occasional misclassifications. While an active contour could have helped to make the detection more robust by including more than just local information on the current ray, we have decided to use a pre-segmentation of the vessels (see Section 9.5). Segmenting the two vessel takes only a short time and in return gives the expert user the possibility to correct errors. Results are coherent over all angles in the sampling and slices along the centerline, which would not be achieved by active contours. This coherence is important to prevent unwanted effects in the projection. We are, however, optimistic that one of the simpler techniques might work when using (small-animal) MRI scans or human subjects (which would not contain a cuff).

From the inner vessel wall outward we continue sampling along the ray, but read

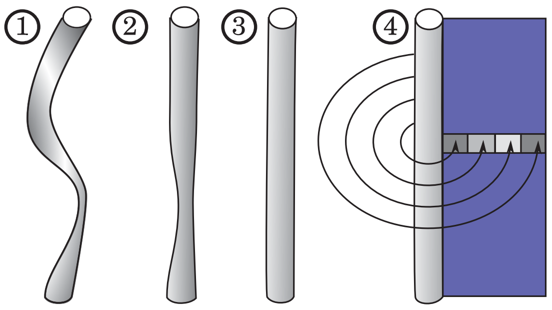


Figure 9.4: Normalization: In comparison to other reformation techniques (see Section 9.2) our approach does not only straighten the vessel (1-2) but also extrudes the vessel wall to a cylinder (3). We then switch modalities from CT to PET and sample in concentric circles (4).

values from the PET data set for the projection and analysis (see Figure 9.5(3)).

To render the final image, we project all n-th PET samples in one slice to one pixel (see Figure 9.5(4)). For compositing we utilize a MIP, which is commonly used for 3D renderings of PET data sets and corresponds with the need to locate the center of high activity spots. For the visualization we link the transfer function from the CPR and orthogonal slice views to give a consistent and comparable result.

Because we are sampling in radial direction we need to make sure that we do not undersample our data set. The number of rays needed to generate an accurate image is calculated as follows:

$$num_{rays} \geq \frac{2 * \pi * d_{max}}{0.5 * min(SP_{PET})}$$

with $d_{max} = r_{max} + d_{maxProj}$ where $min(SP_{PET})$ is the minimal voxel side length in the PET data set, r_{max} the maximal lumen radius and $d_{maxProj}$ the extent to which the data should be sampled from there. One should keep in mind that $min(SP_{PET})$ is significantly larger than CT spacing and d_{max} will be quite limited since distant PET activities would only distract the user. We do not factor in the CT sampling rate here because rays are a lot more dense inside the lumen (the CT is sampled only there). Accordingly, to make sure the number of (equidistant) slices along the centerline is sufficient to satisfy the sampling theorem, the following relations need to be satisfied:

$$d_{slices} + 2 * d_{max} \sin \frac{\alpha_{max}}{2} \leq 0.5 * min(SP_{PET})$$

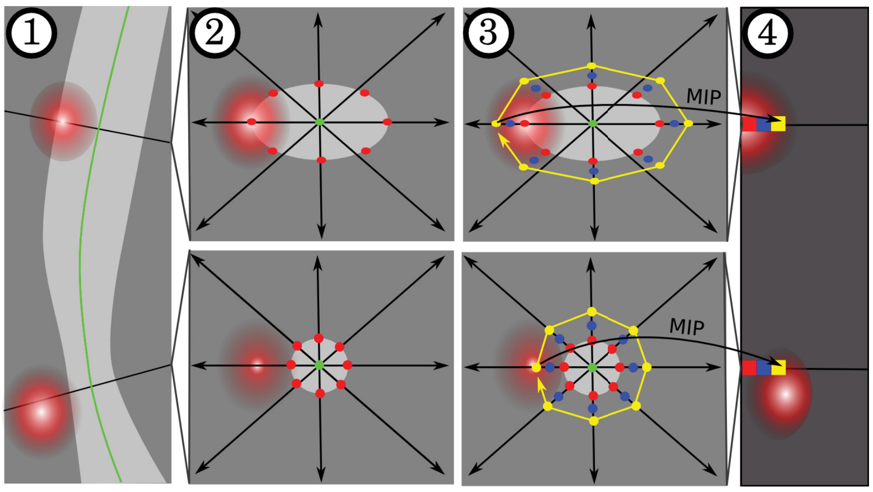


Figure 9.5: Normalized Circular Projection: We visualize the tracer uptake (red highlights) near a vessel segment (1) in a standardized fashion: From the centerline (green) we cast rays in an orthogonal plane (2) to detect the inner vessel wall (red). We then sample the PET signal further along each ray (3). Samples of the same color have the same distance to the vessel wall, regardless of the vessel diameter and shape (compare top-bottom of 2,3). Applying a maximum intensity projection (MIP) of all equidistant PET samples to one pixel (4) we generate the final image, which facilitates comparison to control.

and

$$d_{slices} + \frac{2 * \sin \alpha_{max}}{r_{max}} \leq 0.5 * min(SP_{CT})$$

where d_{slices} is the distance between orthogonal slices and α_{max} is the maximum angle between two slice planes. In this case we also make sure the CT image is sampled at a sufficient rate to get an accurate calculation of the vessel diameter at all positions. Since the carotid arteries are fairly straight, α_{max} will be small.

9.6.3 Vessel Wall Analysis

To support the domain experts in testing their hypotheses we perform an analysis of the PET signal in the vessel wall and plot the results. While the maximum activity can theoretically be read from the NCP in the form of color mapping, this plotting provides a more quantitative visualization by reducing perceptual impact.

Since the actual vessel wall is not detectable in our microCT scan we analyze a user-defined area surrounding the inner vessel wall (contrast border). This layer of

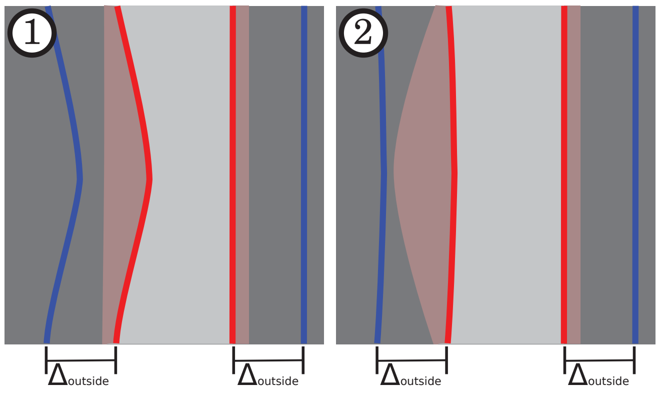


Figure 9.6: Extent of PET analysis: Plaques can remodel the vessel wall (light red, not visible in actual CT scan) inward (1) or outward (2). By analyzing a region of about four times the width of the vessel wall, starting at the lumen (light gray), our application can handle both types.

width $\Delta_{outside}$ is defined in the NCP view (see Figure 9.8, blue line). In our application case we set $\Delta_{outside} = 120\mu m$, determined by histology. This value corresponds to about four times the typical wall thickness for the carotid arteries in mice to include potential plaques. Because our analysis region starts at the inner vessel wall we are able to include internal as well as external plaques (see Figure 9.6).

The generated plot is illustrated in Figure 9.7: From the layer of width $\Delta_{outside}$ around the vessel lumen (blue) we compute minimal and maximal activity. Maximal activity is the most important value for each slice and is therefore highlighted in solid yellow. To provide information about the range of values we render a light yellow band extending to the minimal value. For comparison, the value at the centerline is plotted to give the user the activity in the bloodstream. To provide context for the activity plot we render the average radius along the length of the vessel next to it. We chose a colorscheme matching a typical grayscale CT slice rendering, highlighting the inner vessel wall in red to provide a visual link to the other views. Since our screenlayout is mirrored, the plots are next to the control side, clearly separated by a thick line. This enables a convenient comparison and highlights differences through asymmetry.

9.6.4 Interaction

Our system extensively uses linked cursors to support the visualizations. Clicking and dragging one of the views will also synchronize the others accordingly to give the user different perspectives on the data. In the NCP, intensity as well as distance

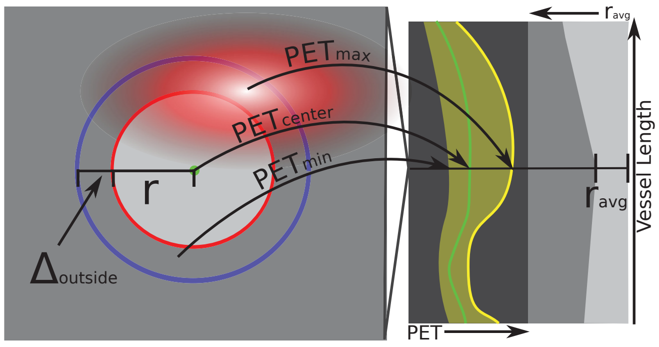


Figure 9.7: Vessel Wall Analysis: The maximal and minimal PET activity inside a user-defined area (blue) around the inner vessel wall is plotted for each orthogonal slice along the centerline. Maximum activity is highlighted in solid yellow, while the area between maximum and minimum is rendered semi-transparent. For comparison, the activity along the centerline is also displayed (green). The average radius provides a context for the user.

to the vessel is automatically displayed on mouse movement. We mirror the cursor to the opposing side and also display the position along the vessel in the plot views. To restore the spatial relation that is lost in the maximum intensity projection, we also display the location of the maximum at the distance to the vessel the mouse cursor is over. The mouse-over behavior in the plots is very similar, highlighting the corresponding section of the analysis region.

9.7 Results

In this section we are discussing images generated using our application and their interpretation by domain experts.
As discussed before, one of the aspects making analysis of PET images difficult is judging from where the signal originates. We therefore show how our proposed techniques visualize different situations, starting with a center of activity inside the vessel wall (see Figure 9.8). Our NCP technique (left) clearly shows how the PET signal is distributed in relation to the vessel, with distance to the vessel along the x-Axis and position along the centerline on the y-Axis. The center of activity is inside the defined analysis region (blue line) and is therefore very likely coming from the vessel wall, which can be verified in the orthogonal slice view (right). Since a MIP is used the color mapping is identical to the slice view and mental linking is easily possible. To be able to rely on the results of our analysis it was of major importance

for the domain experts to also visualize the detected inner vessel wall (red) as well as the region analyzed for the plots (blue) in the original data.

Figure 9.9 illustrates how a signal spill-in is handled by the different views. In the intensity plot we can see that the spread of values is quite wide at the current position. The NCP nicely demonstrates the reason for this effect: A region of high activity is well outside our analysis region but still affects intensities through spill-in and can potentially be misinterpreted as vessel wall related tracer uptake if just looking at spreadsheet analysis. Since the vector of interest used for the CPR is configured so that a rendering comparable to a standard coronal slice view is obtained (as requested by our medical partners) the center of activity is not visible. Since rotation of CPR views has been proposed [KWFG03] we suggested an integration in our tool, but our domain experts found that this would make interpretation too difficult.

Figure 9.10 shows the effect of the cuff on lumen diameter and PET signal in comparison with control. As expected the uptake in the cuff region is increased over control. From the vessel-shape plot we can see that the lumen diameter does not abruptly change with start and end of the implanted cuff. Instead, the lumen diameter decreases continously towards the cuff and also increases gradually downstream from the cuff. This corresponds to the plaque formation expected due to low and laminar shear stress upstream and low and oscillatory shear stress immediately downstream from the cuff (see Figure 9.2). Histology of different cross sections along the vessel of this mouse have confirmed our findings.

In summary, the application enables the user to analyze atherosclerotic lesions much faster than conventional analysis, with direct visual comparison to the control side, and immediate visual feedback of potentially false positive results in the vessel wall. Since the carotid cuff model is well established we expect our application to see continued use in building or testing hypotheses about new PET tracers over the next years.

9.7.1 Comparison

We have created a software phantom to compare our proposed visualizations to existing techniques. The phantom consists of two volumes: A CT-volume with vessel, cuff and background, and a PET-volume with a number of PET activities modeled using gaussians. Since the carotid arteries are relatively straight at the section where the cuff is implanted we modeled the vessel in the CT volume as a cylinder with varying diameter. This also simplified the implementation of existing techniques as well as the phantom. The activities in the PET volume represent the most relevant

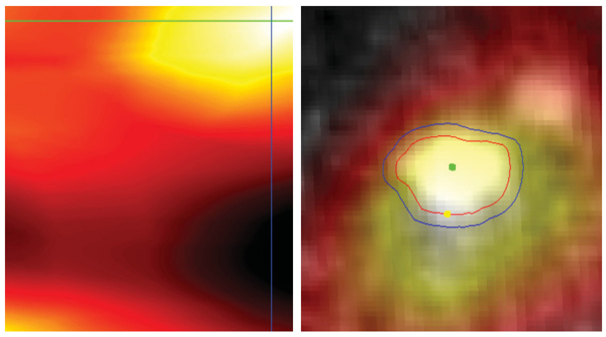


Figure 9.8: PET Activity inside the vessel wall is indicated by maxima inside the analysis region (blue line). By clicking on a position to be examined in any of the views the orthogonal slice view (right) is set and shows the analysis region between red (inner vessel wall) and blue line, the centerline position (green dot) and the position of the maximum (yellow dot).

configurations (see Figure 9.11, from top to bottom):

- Activity inside the vessel wall.
- Activity outside (spill-in).
- Activity inside the vessel wall plus spill-in on the opposite side.
- Same as above but spill-in is closer to the vessel.
- Activity inside the vessel wall plus overlapping spill-in on the same side.

We chose the CPR (due to the straight phantom equivalent to a slice rendering), helical CPR and flattening (cylindrical MIP) as comparison to our proposed NCP plus activity plot. We believe these techniques represent the previous work (see Section 9.2) quite well since most systems utilize some of these visualization techniques. The CPR/slice rendering in combination with the orthogonal view also represents what the user is seeing in the current analysis protocol. The usefulness of the CPR (top left) heavily depends on the vector of interest. In the best case it generates an intuitive visualization of the vessel and PET activity, in the worst case the user misses important features. Our proposed visualizations (top right) allow the user to locate PET activity close to the vessel wall immediately by looking at the right side of the NCP. Distance to other activities can easily be judged and the source of features in the plot can be traced back using the NCP. Secondary activities at each pixel are not represented but we would like to emphasize that we do not see these visualizations as a standalone

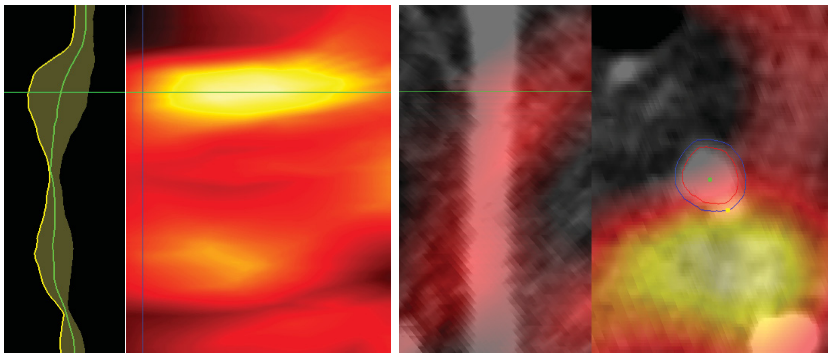


Figure 9.9: Spill-In on the control side: The wide spread of intensites in the activity plot (left) hints at a possible spill-in. The NCP (center-left) confirms this indication by visualizing position of the intensity distribution in one image. Note how the situation is not visualized by the CPR (center-right) due to an unsuitable vector of interest and would therefore require rotation by the user.

solution but rather as an overview/navigation tool. A high activity close to the vessel in the NCP will (in our application case) result in inspection of the orthogonal slice by the user. Activities of lesser magnitude will therefore still be detected. The linking, especially to the orthogonal slice, is crucial to get a complete mental image of the data. The helical CPR (middle row) looks like a (stretched) CPR rendering and provides good context but interpretation of the PET data is extremely difficult. Maxima are hard to locate, and it is very difficult to determine the spatial relation to the vessel, especially with constant arc-length sampling. The results of the flattening (bottom row) are dependent on the extent of the sampling outside the vessel. A narrow layer around the vessel can generate an overview of the activity, but the distance to the vessel, which is critical in our case, cannot be determined.

9.7.2 Learning Curve

We realize that our proposed NCP technique is not quite as easy to understand and interpret as a normal slice view or a CPR, and therefore we performed an informal evaluation of the learning curve associated with our application. Since the current analysis is done by a medical laboratory scientist this also represents our most likely user group. It took us just a few minutes to explain the concepts (using the figures in this chapter) and the actual usage of our application, during which we encountered no problems. Given that a large number of mice need to be analyzed we argue that the time needed to learn our application is insignificant.

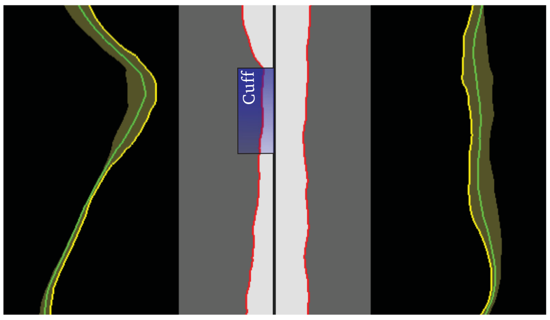


Figure 9.10: Activity increase along the cuff: Confirming the hypothesis of medical researchers, the PET uptake has increased along the cuff. The plot of the lumen shape shows that diameter shrinks steadily upstream and does not immediately increase downstream from the cuff. This finding fits well with the way shear stress is modified by the cuff (compare with Figure 9.2) and was confirmed by histology.

9.7.3 Implementation and Performance

The application has been implemented as a set of processors in Voreen, using OpenGL and GLSL. Preprocessing (i.e., segmentation and centerline computation) can be performed using the ROI framework in combination with the segmentation system discussed by Praßni et al [PRH10] or with the application described in Chapter 8. A somewhat simplified (excluding text overlays etc.) rendering network is shown in Figure 9.12: The topmost part of the network (cyan) handles the loading of CT, PET, segmentation and centerline. Rendering is performed for the cuff- and control-side separately and split up into slice- (red), CPR- (blue) and NCP-rendering (green). The lower part of the network (orange) composes the images and displays them on the screen.

The only operation which is not real-time is the calculation of the NCP. Taking only a few seconds on an Intel i7 930 (2.80 GHz), we do not see this as limiting usability. Other authors have implemented similar techniques on the GPU [LCMH09], and we believe an implementation in OpenCL would be straightforward.

9.7.4 Limitations

Our application currently analyzes only one timestep from a PET data set, the user can, however, easily switch between the different steps triggering an automatic NCP update. As with most reformation techniques our approach requires a fairly accurate placement of the centerline and approximately convex vessel cross sections. For the current application case we do not see this as problematic.

9.8 Conclusion and Future Work

In this chapter we have presented a visualization application which enables the medical expert to explore and analyze in-vivo PET activity around vessels. We have proposed a novel projection technique that renders data around tubular structures in one image. This allows the user to quickly locate activity maxima and accurately judge intensity as well as spread. Activity inside the vessel wall is analyzed by looking at a defined layer surrounding the lumen. For an easy quantitative visualization we plot activity along the vessel and provide the vessel shape as context. We discuss how we combine well known vessel visualization approaches with our proposed techniques in one linked multi-view application. Our screen layout is symmetric to facilitate comparison of the right carotid artery (implanted cuff) with control.
We present resulting images, discuss interpretation of our visualizations and compare our techniques to previous work.

We see several directions in which this work could be extended. An extension to multiple branches using one of the many published layout techniques while preserving clarity of display and comparative capabilites needs to be explored. We are investigating possible interaction techniques to exclude spill-in detected by the user from the analysis. Furthermore, a series of MRI scans is planned and we will test our system using this data. Preliminary tests with MRI scans indicate that an on-the-fly detection of the vessel wall might be possible using this modality. Application to human subjects should just require a modification of $\Delta_{outside}$. Preliminary comparisons with histology have found our approximation of $\Delta_{outside}$ to be fairly accurate, but we are still looking to improve our approach by incorporating a specialized lumen segmentation technique to compensate for partial volume effect and spill-out.
Finally, other application cases for the proposed visualization techniques should be explored.

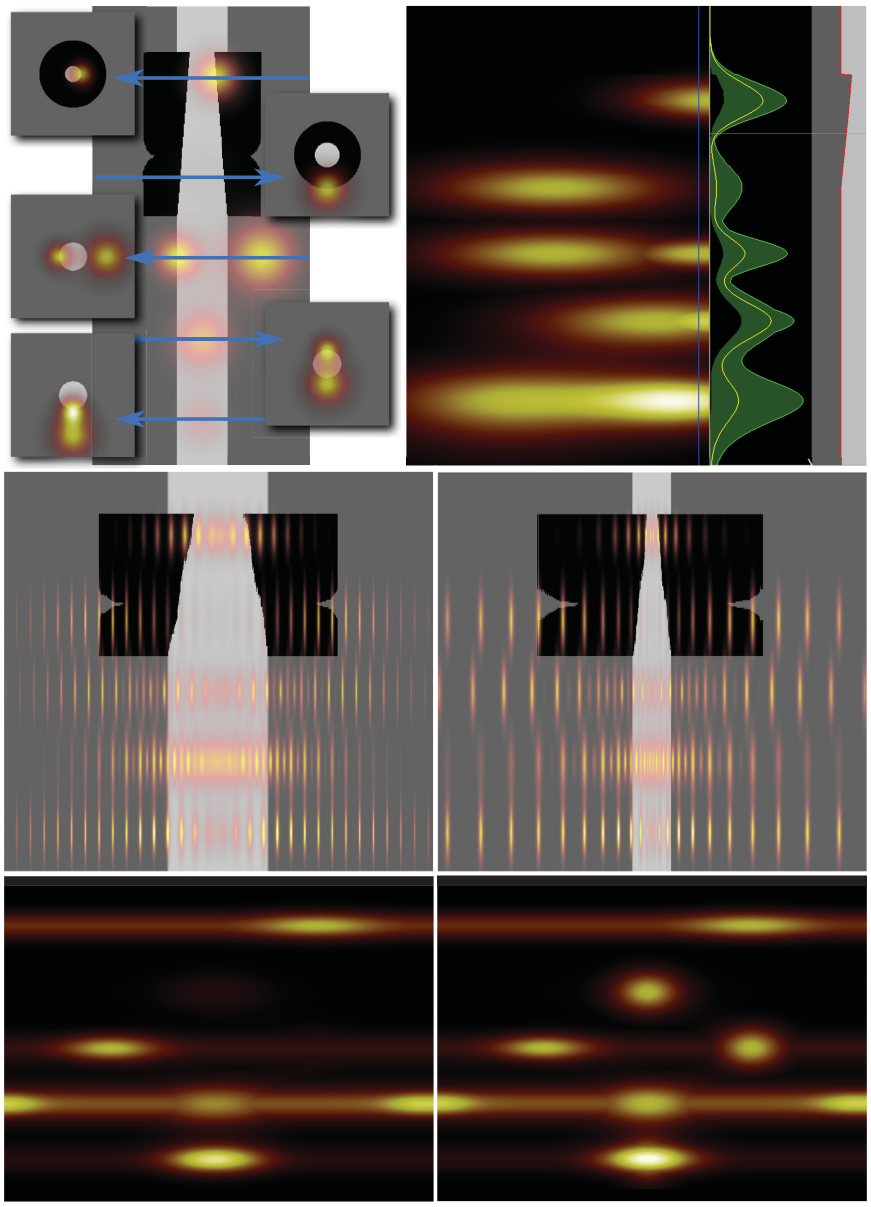


Figure 9.11: Comparison Using a Software Phantom First Row: CPR plus orthogonal slices (left) and our proposed techniques (right). Second Row: Helical CPR using constant angle (left) and constant arc-length (right). Third Row: Flattening using a cylindrical MIP with shorter (left) and longer (right) rays.

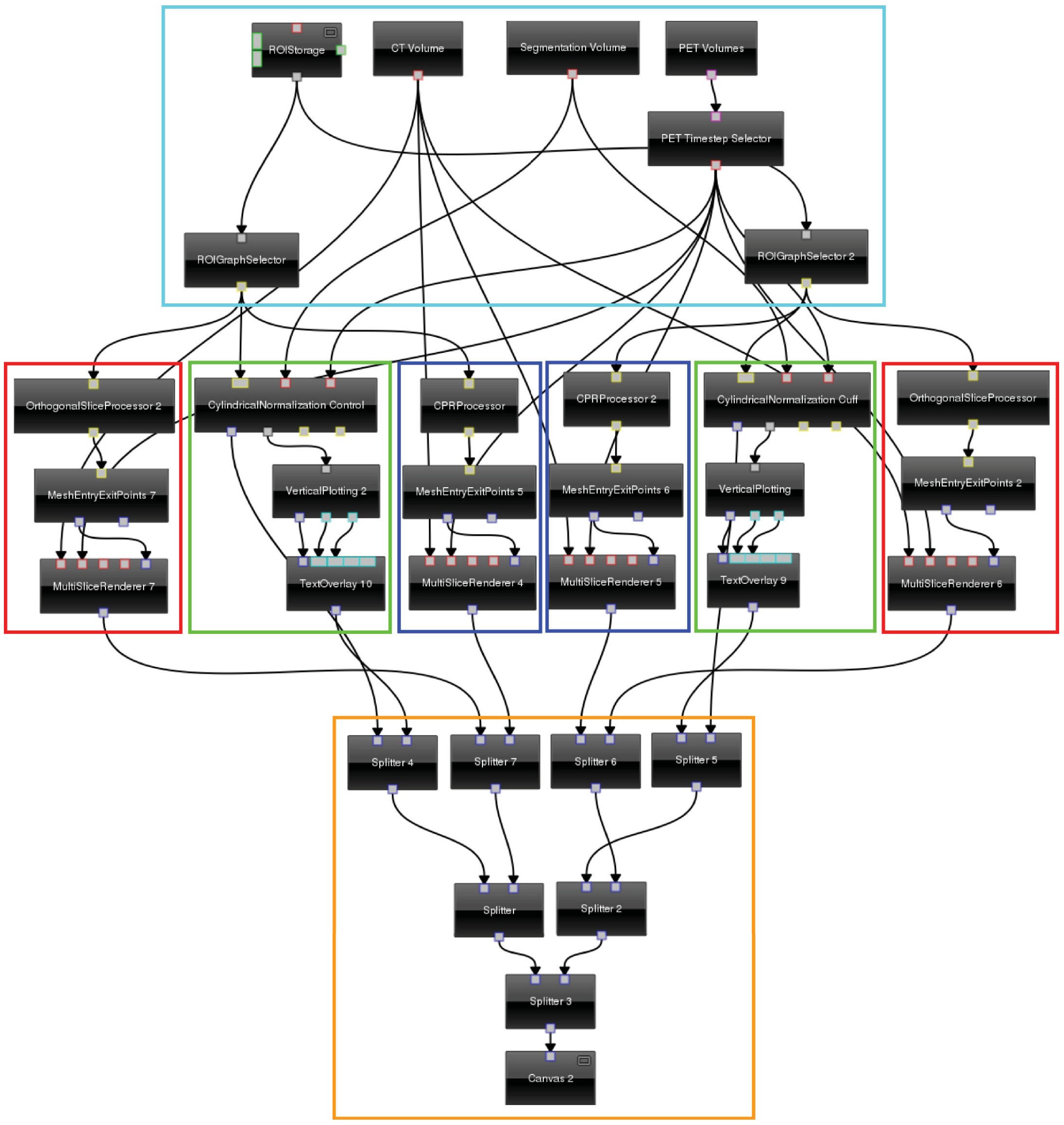


Figure 9.12: Rendering network: The processors in the (simplified) rendering network can be grouped into five categories: Input of volumes (CT, PET and segmentation) and centerlines (cyan), rendering of orthogonal slices (red), NCP and plots (green) and CPR (blue), followed by a composting of all rendering using `Splitter` processors (orange).

Chapter 10

Conclusions

In this dissertation we have discussed several applications for the visual analysis of volumetric medical data. All of these applications have in common that they are designed for a specific task and provide an optimized workflow to the user. Through the use of specialized visualization techniques, like reformations or projections, the data sets are processed to present important aspects at a glance. Standard medical workstation software cannot provide techniques for each application case but usually provides a set of standard views and interaction techniques. These standard techniques are part of all applications discussed in this dissertation for several reasons: Medical users are often trained to view volumes in 2D slice views and new visualization techniques oftentimes impede the ability to navigate the data set and locate anatomical structures. Viewing (relatively) unprocessed data can also provide details that are lost in advanced rendering techniques. Finally, standard measuring techniques for distances, volumes or intensities often can not be used in these views.

A combination of both standard and novel techniques can, however, only reach its full potential when linking between views is implemented. A recurring pattern in the applications discussed here is the linking of a projection technique, which is used to highlight areas needing attention by the viewer, with slice views to allow in-detail inspection. This approach is also in line with the visual information seeking mantra [Shn96]:

> Overview first, zoom and filter, then details-on-demand.

Having specialized applications for each task obviously requires a high development effort, which may be prohibitive in some cases like medical research. We have shown how the Voreen framework has been extended to minimize the amount of code that has to be written in order to develop a new application. One key aspect is the fine granularity of components: Instead of providing 2D and 3D viewer modules, the implementation is broken down into multiple processors to enable a very flexible development environment and high reusability of code. The number of processors providing standard functionality has increased over the last years to a point where the development of novel techniques will usually only require the implementation of a few processors. Using the editor provided by the VoreenVE application, a set of processors can now easily be connected into a linked multi-view application. With the implementation of analysis and plotting frameworks Voreen has developed from a pure volume rendering engine to a rapid application development framework for the visual analysis of multimodal volumetric data sets.

Of the opportunities for future research already discussed in this dissertation we believe the following ones are the most promising: As stated in the discussion of our context-aware navigation technique, the trackball metaphor is still the standard interaction technique for 3D navigation and we believe a multitude of applications could benefit from work on advanced approaches. An extension of our approach for the analysis of PET tracer uptake along vessels to multiple branching vessels should be investigated. As for the future development of Voreen, an improved support for workflows is currently being implemented and the application mode will be redesigned to improve usability for larger networks.

Bibliography

[AH11] P. Angelelli and H. Hauser, *Straightening Tubular Flow for Side-by-Side Visualization*, IEEE TVCG **17** (2011), no. 12, 2063–2070.

[AJ09] S.S. Abeysinghe and T. Ju, *Interactive skeletonization of intensity volumes*, The Visual Computer **25** (2009), 627–635.

[BFC04] Katja Bühler, Petr Felkel, and Alexandra La Cruz, *Geometric methods for vessel visualization and quantification- a survey*, Geometric Modelling for Scientific Visualization, Springer, 2004, pp. 399–419.

[BGP$^+$11] M. Borkin, K. Gajos, A. Peters, D. Mitsouras, S. Melchionna, F. Rybicki, C. Feldman, and H. Pfister, *Evaluation of artery visualizations for heart disease diagnosis*, IEEE TVCG **17** (2011), no. 12, 2479–2488.

[BHWB07] J. Beyer, M. Hadwiger, S. Wolfsberger, and K. Buhler, *High-quality multimodal volume rendering for preoperative planning of neurosurgical interventions*, IEEE Vis **13** (2007), no. 6, 1696–1703.

[BiBPtHR08] R. Brecheisen, A.V. i Bartroli, B. Platel, and B. ter Haar Romeny, *Flexible GPU-based multi-volume ray-casting*, Vision, modeling, and visualization (2008), 303–312.

[BKF$^+$02] N. Burtnyk, A. Khan, G. Fitzmaurice, R. Balakrishnan, and G. Kurtenbach, *StyleCam: interactive stylized 3D navigation using integrated spatial & temporal controls*, ACM UIST, 2002, pp. 101–110.

[BKFK06] Nicolas Burtnyk, Azam Khan, George Fitzmaurice, and Gordon Kurtenbach, *ShowMotion: camera motion based 3D design review*, ACM I3D, 2006, pp. 167–174.

[Bli77] James F. Blinn, *Models of light reflection for computer synthesized pictures*, SIGGRAPH Comput. Graph. **11** (1977), no. 2, 192–198.

[BLP78] Edward G. Britton, James S. Lipscomb, and Michael E. Pique, *Making nested rotations convenient for the user*, SIGGRAPH **12** (1978), no. 3, 222–227.

[BMF$^+$03] Dirk Bartz, Dirk Mayer, Jan Fischer, Sebastian Ley, Anxo del Rio, Steffi Thust, Claus Peter Heussel, Hans-Ulrich Kauczor, and Wolfgang Strasser, *Hybrid Segmentation and Exploration of the Human Lungs*, IEEE Visualization, 2003, pp. 177–184.

[BRP05] R. Bade, F. Ritter, and B. Preim, *Usability comparison of mouse-based interaction techniques for predictable 3d rotation*, Smart Graphics, 2005, pp. 138–150.

[BRS00] Steffi Beckhaus, Felix Ritter, and Thomas Strothotte, *CubicalPath - Dynamic Potential Fields for Guided Exploration in Virtual Environments*, Pacific Graphics, 2000, pp. 387–459.

[BS05] Udeepta D. Bordoloi and Han-Wei Shen, *View Selection for Volume Rendering*, Visualization, IEEE, 2005, pp. 487–494.

[BTI07] D. Borland and R.M. Taylor II, *Rainbow color map (still) considered harmful*, IEEE CG & A (2007), 14–17.

[BVPtHR09] R. Brecheisen, A. Vilanova, B. Platel, and B. ter Haar Romeny, *Parameter sensitivity visualization for dti fiber tracking*, Visualization and Computer Graphics, IEEE Transactions on **15** (2009), no. 6, 1441–1448.

[CCA$^+$05] J.R. Cebral, M.A. Castro, S. Appanaboyina, C.M. Putman, D. Millan, and A.F. Frangi, *Efficient pipeline for image-based patient-specific analysis of cerebral aneurysm hemodynamics: technique and sensitivity*, Medical Imaging **24** (2005), no. 4, 457–467.

[Cha05] Soonmee Cha, *Update on brain tumor imaging*, Current Neurology and Neuroscience Reports **5** (2005), 169–177.

[CON08] M. Christie, P. Olivier, and J.M. Normand, *Camera control in computer graphics*, Computer Graphics Forum, vol. 27, 2008, pp. 2197–2218.

[Cru03] A. La Cruz, *Accuracy Evaluation of Different Centerline Approximations of Blood Vessels*, Tech. report, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2003.

[CS99] Wenli Cai and Georgios Sakas, *Data intermixing and multi-volume rendering*, Computer Graphics Forum **18** (1999), no. 3, 359–368.

[CSYB05] N.D. Cornea, D. Silver, X. Yuan, and R. Balasubramanian, *Computing hierarchical curve-skeletons of 3d objects*, The Visual Computer **21** (2005), no. 11, 945–955.

[DHS$^+$13] Stefan Diepenbrock, Sven Hermann, Michael Schäfers, Michael Kuhlmann, and Klaus Hinrichs, *Comparative Visualization of Tracer Uptake in In Vivo Small Animal PET/CT Imaging of the Carotid Arteries*, EG EuroVis, 2013, accepted.

[Dic45] L.R. Dice, *Measures of the amount of ecologic association between species*, Ecology **26** (1945), no. 3, 297–302.

[DPL$^+$10] Stefan Diepenbrock, Jörg-Stefan Praßni, Florian Lindemann, Hans-Werner Bothe, and Timo Ropinski, *Pre-Operative Planning of Brain Tumor Resections*, IEEE VisWeek Electronic Proceedings, 2010.

[DPL$^+$11] S. Diepenbrock, J.S. Praßni, F. Lindemann, H.W. Bothe, and T. Ropinski, *2010 IEEE Visualization Contest Winner: Interactive Planning for Brain Tumor Resections*, Computer Graphics and Applications, IEEE **31** (2011), no. 5, 6–13.

[DR12] Stefan Diepenbrock and Timo Ropinski, *From Imprecise User Input to Precise Vessel Segmentations*, EG Visual Computing for Biology and Medicine, 2012, pp. 65–72.

[DRH11] Stefan Diepenbrock, Timo Ropinski, and Klaus H. Hinrichs, *Context-Aware Volume Navigation*, IEEE Pacific Visualization Symposium (PacificVis 2011), 2011, pp. 11–18.

[DSzBH$^+$11] Stefan Diepenbrock, Christian Schulte zu Berge, Klaus H. Hinrichs, Lydia Wachsmuth, and Cornelius Faber, *DTI Visualization using the Voreen framework*, ESMRMB Congress, 2011, Poster.

[Eng06] K. Engel, *Real-time volume graphics*, Ak Peters Series, A K Peters, Limited, 2006.

[FUS$^+$98] A.X. Falcão, J.K. Udupa, S. Samarasekera, S. Sharma, B.E. Hirsch, and R.A. Lotufo, *User-Steered Image Segmentation Paradigms: Live Wire and Live Lane*, Graphical Models and Image Processing **60** (1998), no. 4, 233–260.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: Elements of reusable object-oriented software*, Reading: Addison-Wesley, 1995.

[GNBP11] R. Gasteiger, M. Neugebauer, O. Beuing, and B. Preim, *The FLOWLENS: A Focus-and-Context Visualization Approach for Exploration of Blood Flow in Cerebral Aneurysms*, IEEE TVCG **17** (2011), no. 12, 2183–2192.

[Gra06] L. Grady, *Random walks for image segmentation*, IEEE Pattern Analysis and Machine Intelligence (2006), 1768–1783.

[GRH$^+$04] A. Gerhards, P. Raab, S. Herber, K. F. Kreitner, T. Boskamp, and P. Mildenberger, *Software-assisted CT-postprocessing of the carotid arteries*, Rofo **176** (2004), no. 6, 870–874.

[HBA$^+$04] P. Haigron, M.E. Bellemare, O. Acosta, C. Göksu, C. Kulik, K. Rioual, and A. Lucas, *Depth-map-based scene analysis for active navigation in virtual angioscopy*, IEEE Transactions on Medical Imaging **23** (2004), no. 11, 1380–1390.

[HBD11] M. Helmstaedter, K.L. Briggman, and W. Denk, *High-accuracy neurite reconstruction for high-throughput neuroanatomy*, Nature neuroscience **14** (2011), no. 8, 1081–1088.

[HDKG08] M. Hachet, F. Decle, S. Kn
"odel, and P. Guitton, *Navidget for easy 3D camera positioning from 2d inputs*, IEEE 3DUI, 2008, pp. 83–89.

[HHCL01] T. He, L. Hong, D. Chen, and Z. Liang, *Reliable path for virtual endoscopy: ensuring complete examination of human organs*, IEEE TVCG (2001), 333–342.

[HMK$^+$97] Lichan Hong, Shigeru Muraki, Arie Kaufman, Dirk Bartz, and Taosong He, *Virtual voyage: interactive navigation in the human colon*, SIGGRAPH, 1997, pp. 27–34.

[HZF$^+$11] K. Hameeteman, M.A. Zuluaga, M. Freiman, L. Joskowicz, O Cuisenaire, L. Florez Valencia, M.A. Gulsun, K. Krissian, J. Mille, W.C.K. Wong, M. Orkisz, H. Tek, M. Hernandez Hoyos, F. Benmansour, A.C.S. Chung, S. Rozie, M.J. van Gils, L. van den Borne, J. Sosna, P. Berman, N. Cohen, P. Douek, I. Sánchez, M. Aissat, M. Schaap, C.T. Metz, G. P. Krestin, A van der Lugt, W.J. Niessen, and T. van Walsum, *Evaluation framework* *for carotid bifurcation lumen segmentation and stenosis grading*, Medical Image Analysis **15** (2011), no. 4, 477–488.

[IKN98] L. Itti, C. Koch, and E. Niebur, *A model of saliency-based visual attention for rapid scene analysis*, IEEE TPAMI **20** (1998), no. 11, 1254–1259.

[ISN$^+$03] L. Ibanez, W. Schroeder, L. Ng, J. Cates, et al., *The ITK software guide*, vol. 8, Kitware, 2003.

[JBH$^+$09] W.K. Jeong, J. Beyer, M. Hadwiger, A. Vazquez, H. Pfister, and R.T. Whitaker, *Scalable and interactive segmentation and visualization of neural processes in EM datasets*, Vis 09 (2009), 1505–1514.

[JQD$^+$08] Alark Joshi, Xiaoning Qian, Donald Dione, Ketan Bulsara, Christopher Breuer, Albert Sinusas, and Xenophon Papademetris, *Effective visualization of complex vascular structures using a non-parametric vessel detection method*, IEEE TVCG **14** (2008), no. 6, 1603–1610.

[K$^+$02] D.A. Keim et al., *Information visualization and visual data mining*, IEEE transactions on Visualization and Computer Graphics **8** (2002), no. 1, 1–8.

[KAF$^+$08] D. Keim, G. Andrienko, J.D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon, *Visual analytics: Definition, process, and challenges*, Information Visualization (2008), 154–175.

[KBH$^+$10] P. Kok, M. Baiker, E.A. Hendriks, F.H. Post, J. Dijkstra, C.W.G.M. Lowik, B.P.F. Lelieveldt, and C.P. Botha, *Articulated planar reformation for change visualization in small animal imaging*, IEEE TVCG **16** (2010), no. 6, 1396–1404.

[KBK07] P. Kohlmann, S. Bruckner, and A. Kanitsar, *LiveSync: Deformed viewing spheres for knowledge-based navigation*, IEEE TVCG **13** (2007), no. 6, 1544–1551.

[KBKG09] Peter Kohlmann, Stefan Bruckner, Armin Kanitsar, and Meister Eduard Gröller, *Contextual Picking of Volumetric Structures*, Pacific Vis (2009), 185–192.

[KCH$^+$12] MT Kuhlmann, S. Cuhlmann, I. Hoppe, R. Krams, PC Evans, GJ Strijkers, K. Nicolay, S. Hermann, and M. Schäfers, *Implantation of a Carotid Cuff for Triggering Shear-stress Induced Atherosclerosis in Mice*, Journal of Visualized Experiments (2012), no. 59, Video Publication.

[KEK04] Y. Kang, K. Engelke, and W.A. Kalender, *Interactive 3D editing tools for image segmentation*, Medical Image Analysis **8** (2004), no. 1, 35–46.

[KFW$^+$02] A. Kanitsar, D. Fleischmann, R. Wegenkittl, P. Felkel, and M.E. Gröller, *CPR: curved planar reformation*, Vis 02, 2002, pp. 37–44.

[KGB$^+$09] B. Kainz, M. Grabner, A. Bornik, S. Hauswiesner, J. Muehl, and D. Schmalstieg, *Ray casting of multiple volumetric datasets with polyhedral boundaries on manycore GPUs*, ACM Trans. on Graphics (TOG) **28** (2009), no. 5, 152:1–152:9.

[KGNP12] Christoph Kubisch, Sylvia Glaßer, Mathias Neugebauer, and Bernhard Preim, *Vessel Visualization with Volume Rendering*, Visualization in Medicine and Life Sciences II, Mathematics and Visualization, Springer Berlin Heidelberg, 2012, pp. 109–134.

[KKH02] J. Kniss, G. Kindlmann, and C. Hansen, *Multidimensional transfer functions for interactive volume rendering*, IEEE TVCG (2002), 270–285.

[KKPS08] Arno Krueger, Christoph Kubisch, Bernhard Preim, and Gero Strauss, *Sinus Endoscopy - Application of Advanced GPU Volume Rendering for Virtual Endoscopy*, IEEE TVCG **14** (2008), 1491–1498.

[KKS$^+$05] A. Khan, B. Komalo, J. Stam, G. Fitzmaurice, and G. Kurtenbach, *HoverCam: interactive 3D navigation for proximal object inspection*, ACM I3D, 2005, pp. 73–80.

[KQ03] C. Kirbas and F.K.H. Quek, *Vessel extraction techniques and algorithms: A survey*, IEEE Bioinformatics and Bioengineering, IEEE, 2003, pp. 238–245.

[KQ04] C. Kirbas and F. Quek, *A review of vessel extraction techniques and algorithms*, ACM Surveys **36** (2004), no. 2, 81–121.

[KW03] J. Krüger and R. Westermann, *Acceleration techniques for GPU-based volume rendering*, IEEE Visualization (2003), 287–292.

[KWFG03] A. Kanitsar, R. Wegenkittl, D. Fleischmann, and M.E. Gröller, *Advanced curved planar reformation: Flattening of vascular structures*, IEEE VisWeek, 2003, pp. 43–50.

[La 04] A. La Cruz, *Accuracy Evaluation of Different Centerline Approximations of Blood Vessels*, EG/IEEE Visualization Symposium, 2004, pp. 115–120.

[LABFL09] David Lesage, Elsa D Angelini, Isabelle Bloch, and Gareth Funka-Lea, *A review of 3D vessel lumen segmentation techniques: models, features and extraction schemes.*, Medical image analysis **13** (2009), no. 6, 819–45.

[LC87] W.E. Lorensen and H.E. Cline, *Marching cubes: A high resolution 3d surface construction algorithm*, ACM Siggraph Computer Graphics, vol. 21, ACM, 1987, pp. 163–169.

[LCD06] Thomas Luft, Carsten Colditz, and Oliver Deussen, *Image enhancement by unsharp masking the depth buffer*, ACM Transactions on Graphics **25** (2006), no. 3, 1206–1213.

[LCMH09] O.D. Lampe, C. Correa, K.L. Ma, and H. Hauser, *Curve-centric volume reformation for comparative visualization*, IEEE TVCG **15** (2009), no. 6, 1235–1242.

[Lev88] M. Levoy, *Display of surfaces from volume data*, Computer Graphics and Applications, IEEE **8** (1988), no. 3, 29–37.

[LLHY09] S. Lindholm, P. Ljung, M. Hadwiger, and A. Ynnerman, *Fused Multi-Volume DVR using Binary Space Partitioning*, Computer Graphics Forum, vol. 28, John Wiley & Sons, 2009, pp. 847–854.

[LLS05] Hye-Jin Lee, Sukhyun Lim, and Byeong-Seok Shin, *Unfolding of Virtual Endoscopy Using Ray-Template*, Bio. and Medical Data Analysis, 2005, pp. 69–77.

[LR11] Florian Lindemann and Timo Ropinski, *About the influence of illumination models on image comprehension in direct volume rendering*, IEEE TVCG (Vis Proceedings) **17** (2011), no. 12, 1922–1931.

[Mar11] J. Marino, *Context Preserving Maps of Tubular Structures*, IEEE TVCG **17** (2011), no. 12, 1997–2004.

[MMGK09] J. McCrae, I. Mordatch, M. Glueck, and A. Khan, *Multiscale 3D navigation*, ACM I3D, 2009, pp. 7–14.

[MS09] Jennis Meyer-Spradow, *Interaktive Entwicklung Raycasting-basierter Visualisierungs-Techniken für medizinische Volumen-Daten mit Hilfe von Datenflussnetzwerken*, Ph.D. thesis, Westfälische Wilhelms-Universität Münster, 2009.

[MSE$^+$06] D. Merhof, M. Sonntag, F. Enders, C. Nimsky, P. Hastreiter, and G. Greiner, *Hybrid visualization for white matter tracts using triangle strips and point sprites*, Visualization and Computer Graphics, IEEE Transactions on **12** (2006), no. 5, 1181–1188.

[MSRMH09] Jennis Meyer-Spradow, Timo Ropinski, Jörg Mensmann, and Klaus Hinrichs, *Voreen: A Rapid-Prototyping Environment for Ray-Casting-Based Volume Visualizations*, IEEE Computer Graphics and Applications **29** (2009), no. 6, 6–13.

[NYE$^+$10] Tan Khoa Nguyen, Anders Ynnerman, Anders Eklund, Patric Ljung, Frida Hernell, Henrik Ohlsson, Mats Andersson, Hans Knutsson, and Camilla Forsell, *Concurrent Volume Visualization of Real-Time fMRI*, IEEE/EG Volume Graphics, 2010, pp. 53–60.

[OH00] Nabil Ouerhani and Heinz Hügli, *Computing visual attention from scene depth*, Pattern Recognition **1** (2000), 375–378.

[ONI05] Shigeru Owada, Frank Nielsen, and Takeo Igarashi, *Volume catcher*, SI3D **C** (2005), 111–116.

[ONI$^+$08] S. Owada, F. Nielsen, T. Igarashi, R. Haraguchi, and K. Nakazawa, *Projection plane processing for sketch-based volume segmentation*, Biomedical Imaging (2008), 117–120.

[OP05] Steffen Oeltze and Bernhard Preim, *Visualization of vasculature with convolution surfaces: method, validation and evaluation*, IEEE Med. Imaging **24** (2005), no. 4, 540–548.

[OS01] S.D. Olabarriaga and AWM Smeulders, *Interaction in the segmentation of medical images: A survey*, Medical image analysis **5** (2001), no. 2, 127–142.

[PB07] B. Preim and D. Bartz, *Visualization in Medicine*, Morgan Kaufmann, 2007.

[PHA07] K. Poon, G. Hamarneh, and R. Abugharbieh, *Live-vessel: Extending livewire for simultaneous extraction of optimal medial and boundary paths in vascular images*, MICCAI (2007), 444–451.

[PO08] B. Preim and S. Oeltze, *3D visualization of vasculature: an overview*, Visualization in Medicine and Life Sciences (2008), 39–59.

[Pra11] Jörg-Stefan Praßni, *Interactive Feature Detection in Volumetric Data*, Ph.D. thesis, Westfälische Wilhelms-Universität Münster, 2011.

[PRH10] Jörg-Stefan Praßni, Timo Ropinski, and Klaus H. Hinrichs, *Uncertainty-Aware Guided Volume Segmentation*, Vis 10 **16** (2010), no. 6, 1358–1365.

[Ras00] Jef Raskin, *The humane interface: new directions for designing interactive systems*, ACM Press/Addison-Wesley, 2000.

[RDB$^+$12] Timo Ropinski, Stefan Diepenbrock, Stefan Bruckner, Klaus H. Hinrichs, and Eduard Gröller, *Unified Boundary-Aware Texturing for Interactive Volume Rendering*, Visualization and Computer Graphics, IEEE Transactions on **18** (2012), no. 11, 1942–1955.

[RHR$^+$09] Timo Ropinski, Sven Hermann, Rainer Reich, Michael Schäfers, and Klaus H. Hinrichs, *Multimodal Vessel Visualization of Mouse Aorta PET/CT Scans*, IEEE Vis 09 (2009), 1515–1522.

[RMSD$^+$08] Timo Ropinski, Jennis Meyer-Spradow, Stefan Diepenbrock, Jörg Mensmann, and Klaus H. Hinrichs, *Interactive Volume Rendering with Dynamic Ambient Occlusion and Color Bleeding*, Computer Graphics Forum (Eurographics 2008) **27** (2008), no. 2, 567–576.

[RRRP08] C. Rieder, F. Ritter, M. Raspe, and H.O. Peitgen, *Interactive visualization of multimodal volume data for neurosurgical tumor treatment*, Computer Graphics Forum, vol. 27, John Wiley & Sons, 2008, pp. 1055–1062.

[RT98] B.E. Rogowitz and L.A. Treinish, *Data visualization: the end of the rainbow*, Spectrum, IEEE **35** (1998), no. 12, 52–59.

[RWS$^+$10] Christian Rieder, Andreas Weihusen, Christian Schumann, Stephan Zidowitz, and Heinz-Otto Peitgen, *Visual Support for Interactive Post-Interventional Assessment of Radiofrequency Ablation Therapy*, Computer graphics forum, vol. 29, Wiley Online Library, 2010, pp. 1093–1102.

[SBSG06] P. Sereda, AV Bartroli, IWO Serlie, and FA Gerritsen, *Visualization of boundaries in volumetric data sets using LH histograms*, IEEE TVCG **12** (2006), no. 2, 208–218.

[Sch05] H. Scharsach, *Advanced GPU raycasting*, Proceedings of CESCG **5** (2005), 67–76.

[SCLC$^+$04] M. Straka, M. Cervenansky, A. La Cruz, A. Kochl, M. Sramek, E. Groller, and D. Fleischmann, *The VesselGlyph: Focus & Context Visualization in CT-Angiography*, IEEE Vis 04, 2004, pp. 385–392.

[SHC$^+$09] M. Smelyanskiy, D. Holmes, J. Chhugani, A. Larson, D.M. Carmean, D. Hanson, P. Dubey, K. Augustine, D. Kim, A. Kyker, et al., *Mapping high-fidelity volume rendering for medical imaging to cpu, gpu and many-core architectures*, IEEE Vis (2009), 1563–1570.

[SHCP97] L. Serra, N. Hern, C.B. Choon, and T. Poston, *Interactive vessel tracing in volume data*, Proceedings of the 1997 symposium on Interactive 3D graphics, 1997, pp. 131–ff.

[Shn96] B. Shneiderman, *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*, IEEE Symposium on Visual Languages, 1996, pp. 336–343.

[Sho92] Ken Shoemake, *ARCBALL: a user interface for specifying three-dimensional orientation using a mouse*, Graphics Interface, 1992, pp. 151–156.

[SLC$^+$02] T. Schlathoelter, C. Lorenz, I.C. Carlsen, S. Renisch, and T. Deschamps, *Simultaneous segmentation and tree reconstruction of the airways for virtual bronchoscopy*, SPIE, vol. 4684, 2002, pp. 103–113.

[SMH10] A. Saad, T. Möller, and G. Hamarneh, *ProbExplorer: Uncertainty-guided Exploration and Editing of Probabilistic Medical Image Segmentation*, EuroVis 2010, 2010, pp. 1113–1122.

[SNS$^+$98] Y. Sato, S. Nakajima, N. Shiraga, H. Atsumi, S. Yoshida, T. Koller, G. Gerig, and R. Kikinis, *Three-dimensional multi-scale line filter for segmentation and visualization of curvilinear structures in medical images*, Medical Image Analysis **2** (1998), no. 2, 143–168.

[SOB$^+$07] Christian Schumann, Steffen Oeltze, Ragnar Bade, Bernhard Preim, and Heinz-Otto Peitgen, *Model-free Surface Visualization of Vascular Trees*, IEEE/EG EuroVis, 2007, pp. 283–290.

[SS11] N. Schubert and I. Scholl, *Comparing gpu-based multi-volume ray casting techniques*, Computer Science-Research and Development **26** (2011), no. 1, 39–50.

[SSH10] Michael Schäfers, Otmar Schober, and Sven Hermann, *Matrix-Metalloproteinases as Imaging Targets for Inflammatory Activity in Atherosclerotic Plaques*, Journal of Nuclear Medicine **51** (May 2010), no. 5, 663–666.

[SVVG$^+$01] IWO Serlie, F. Vos, R. Van Gelder, J. Stoker, R. Truyen, F. Gerritsen, Y. Nio, and F. Post, *Improved visualization in virtual colonoscopy using image-based rendering*, IEEE/EG Eurovis, Springer Verlag Wien, 2001, pp. 137–146.

[SzB11] Christian Schulte zu Berge, *Visualization Techniques for Diffusion Tensor Data*, Diploma thesis, Westfälische Wilhelms-Universität Münster, 2011.

[TFTN05] S. Takahashi, I. Fujishiro, Y. Takeshima, and T. Nishita, *A feature-driven approach to locating optimal viewpoints for volume visualization*, IEEE Visualization (2005), 495–502.

[THA11] A. Top, G. Hamarneh, and R. Abugharbieh, *Active learning for interactive 3d image segmentation*, MICCAI (2011), 603–610.

[TMS$^+$06] Christian Tietjen, Björn Meyer, Stefan Schlechtweg, Bernhard Preim, Ilka Hertel, and Gero Strauss, *Enhancing Slice-based Visuaizations of Medical Volume Data*, EuroVis, 2006, pp. 123–130.

[TPvB$^+$03] H. Timinger, V. Pekar, J. von Berg, K. Dietmayer, and M. Kaus, *Integration of interactive corrections to model-based segmentation algorithms*, Bildverarbeitung für die Medizin (2003), 171–175.

[vBVN04] C.M. van Bemmel, M.A. Viergever, and W.J. Niessen, *Semiautomatic segmentation and stenosis quantification of 3D contrast-enhanced MR angiograms of the internal carotid artery*, Magnetic resonance in medicine **51** (2004), no. 4, 753–760.

[VFSG06] Ivan Viola, Miquel Feixas, Mateu Sbert, and Meister Eduard Groller, *Importance-Driven Focus of Attention*, IEEE TVCG **12** (2006), no. 5, 933–940.

[VFSH01] P.P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich, *Viewpoint selection using viewpoint entropy*, Vision Modeling and Visualization, 2001, pp. 273–280.

[VKG04] Ivan Viola, Armin Kanitsar, and Meister Eduard Groller, *Importance-Driven Volume Rendering*, IEEE Vis '04, 2004, pp. 139–146.

[VMN08] P.P. Vázquez, E. Monclús, and I. Navazo, *Representative views and paths for volume models*, Smart Graphics, 2008, pp. 106–117.

[VP04] Vivek Verma and A. Pang, *Comparative flow visualization*, IEEE TVCG **10** (2004), no. 6, 609 – 624.

[vPBB$^+$10] R. van Pelt, J.O. Bescós, M. Breeuwer, R.E. Clough, M.E. Groller, B. ter Haar Romenij, and A. Vilanova, *Exploration of 4D MRI blood flow using stylistic visualization*, IEEE TVCG **16** (2010), no. 6, 1339–1347.

[WBSW07] R. Wang, T. Benner, AG Sorensen, and VJ Wedeen, *Diffusion Toolkit: A Software Package for Diffusion Imaging Data Processing and Tractography*, Proc. Intl. Soc. Mag. Reson. Med, vol. 15, 2007, p. 3720.

[WF97] C. Ware and D. Fleet, *Context sensitive flying interface*, ACM I3D, 1997, pp. 127–ff.

[WGC$^+$08] David Williams, Soren Grimm, Ernesto Coto, Abdul Roudsari, and Haralambos Hatzakis, *Volumetric Curved Planar Reformation for Virtual Endoscopy*, IEEE TVCG **14** (2008), no. 1, 109–119.

[WKL99] D. Weinstein, G. Kindlmann, and E. Lundberg, *Tensorlines: Advection-diffusion based propagation through diffusion tensor fields*, Proceedings of the conference on Visualization'99: celebrating ten years, IEEE Computer Society Press, 1999, pp. 249–253.

[WMM$^+$02] C.F. Westin, SE Maier, H. Mamata, A. Nabavi, FA Jolesz, R. Kikinis, et al., *Processing and visualization for diffusion tensor MRI*, Medical image analysis **6** (2002), no. 2, 93–108.

[WNV00] O. Wink, W.J. Niessen, and M. A. Viergever, *Fast Delineation and Visualization of Vessels in 3D Angiographic Images*, IEEE Med. Imaging **19** (2000), no. 4, 337–346.

[WO90] Colin Ware and Steven Osborne, *Exploration and virtual camera control in virtual three dimensional environments*, ACM I3D, 1990, pp. 175–183.

[Woj06] Adam Wojciechowski, *Potential field based camera collisions detection in a static 3D environment*, MG&V **15** (2006), no. 3, 665–672.

[WOS$^+$11] L. Wachsmuth, K Obrusnik, F. Schmid, S. Diepenbrock, C. Schulte zu Berge, S. Albrecht, T. Kuhlmann, and C. Faber, *High-resolution separation* *of adjacent fiber bundles by fast in vivo DTI-EPI of the mouse brain*, ESMRMB Congress, oct 2011, Poster.

[WS10] C. Wang and Ö. Smedby, *Integrating automatic and interactive methods for coronary artery segmentation: let the PACS workstation think ahead*, International journal of computer assisted radiology and surgery **5** (2010), no. 3, 275–285.

[XH98] D. Xiao and R. Hubbold, *Navigation guided by artificial force fields*, ACM CHI, 1998, pp. 179–186.

[YZH$^+$05] Y. Yang, L. Zhu, S. Haker, A. Tannenbaum, and D. Giddens, *Harmonic skeleton guided evaluation of stenoses in human coronary arteries*, Med. Image Comput. Comput. Assist. Interv., 2005, pp. 490–497.

[ZF99] R. Zeleznik and A. Forsberg, *UniCam-2D gestural camera controls for 3D environments*, ACM I3D, 1999, pp. 169–173.

[ZHT$^+$02] Lei Zhu, S. Haker, A. Tannenbaum, S. Bouix, and K. Siddiqi, *Angle-preserving mappings for the visualization of multi-branched vessels*, Image Processing 2002, vol. 2, 2002, pp. 945–948.

[ZHT05] Lei Zhu, S. Haker, and A. Tannenbaum, *Flattening maps for the visualization of multibranched vessels*, IEEE Med. Imaging **24** (2005), no. 2, 191–198.

[ZSH96] M. Zockler, D. Stalling, and H.C. Hege, *Interactive visualization of 3D-vector fields using illuminated stream lines*, Visualization'96. Proceedings., IEEE, 1996, pp. 107–113.

Acronyms

CPU	central processing unit
CT	computed tomography
DTI	diffusion tensor imaging
DVR	direct volume rendering
EEP	entry-exit points (ray parameters)
FPS	frames per second
GLSL	OpenGL shading language
GPU	graphics processing unit
GUI	graphical user interface
MIP	maximum intensity projection
MRI	magnetic resonance imaging
OpenCL	open computing language
OpenGL	open graphics library
PET	positron emission tomography
ROI	region of interest
TF	transfer function