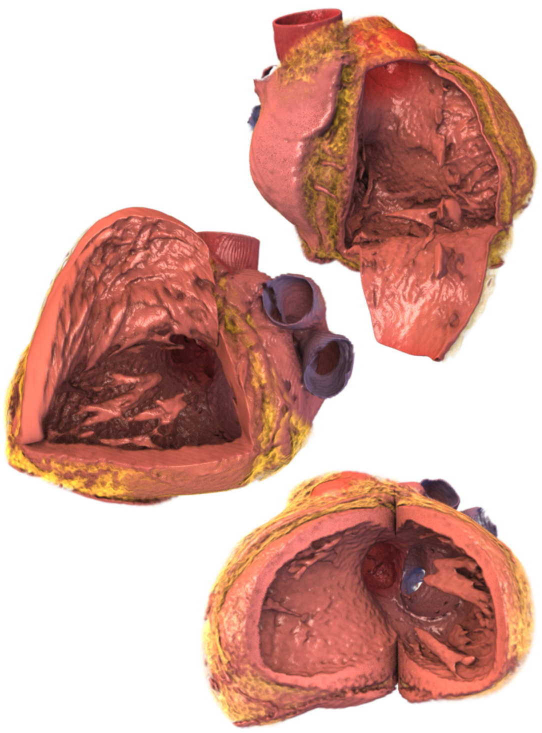
Dissertation

A Dataflow-based Shader Framework for Visualizing Dissections of the Heart Using Individual Patient Data

by
Stephan Arens



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Institut für Informatik
Fachgebiet Computergrafik, Visualisierung und Bildverarbeitung

Abstract

This PhD Thesis addresses the technical demands to create illustrations as in hand-drawn anatomy atlases but based on real volume datasets of the human heart.

Hand-drawn illustrations in anatomy atlases serve to understand the human morphology or functioning. These illustrations use a wide variety of visualization techniques for emphases, e.g. coloring, cuts, deformations, shading, and shadows. In the case of the human heart, the anatomy varies considerably between individual patients and it would be desirable to have such illustrations for individual patients to understand the distinct patients' anatomy.

High flexibility and low effort is of major importance when prototyping such visualizations in volume rendering. This thesis proposes a dataflow-based approach for shader generation to meet the requirement for high flexibility in multi-volume rendering while providing the user with reusable building blocks. The approach allows for any composition of transfer functions, logical combinations, and deformations in multi-volume rendering with correct shading and shadowing without introducing special case handling. Hence, it combines the GPU-power of modern approaches with the flexibility and reusability of dataflow-based programs.

Many GPU-based volume visualization techniques have been presented that imitate the techniques in traditional anatomy atlases. Nevertheless, the quality of such atlases has not been achieved yet. One reason is, of course, the reliance on real clinic volume datasets which do not exhibit enough information and suffer from noise or too low resolution. To overcome this issue, the visualizations in this thesis will be enriched with a priori known information of the human heart. It will be shown that the aforementioned dataflow-based approach promotes the use of this information. However, another reason why the quality of anatomy atlases has not yet been achieved is the effort to combine the needed techniques. The dataflow-design, that allows arbitrary graph compositions without reprogramming, reduces this effort to a minimum.

The applicability of the proposed methods will be demonstrated on a set of CT volume data of human hearts that are visualized as in anatomy atlases.

Zusammenfassung

In dieser Dissertation werden die technischen Anforderungen ergründet, die nötig sind, um basierend auf Volumendaten individueller Patienten Illustrationen wie im Anatomieatlas zu erzeugen.

Die handgezeichneten Illustrationen in Anatomieatlanten dienen dem Verständnis der Morphologie und der Funktionsweise des menschlichen Körpers. Für eine bessere Verständlichkeit gebrauchen sie verschiedene Visualisierungstechniken wie Einfärbungen, Schnitte, Deformationen, Schattierung und Schatten. Die Anatomie des menschlichen Herzens aber unterliegt einer breiten interindividuellen Variabilität, weshalb es vorteilhaft wäre, solche Illustrationen für jeden einzelnen Patienten vorliegen zu haben, um dessen individuelle Anatomie zu verstehen.

Für die computerbasierte Erzeugung von neuen Visualisierungen aus Volumendaten ist folglich vor allem eine hohe Flexibilität bei gleichzeitig möglichst geringem Aufwand erstrebenswert. In dieser Dissertation wird deshalb die Erzeugung von Shaderprogrammen durch ein datenflussbasiertes Verfahren vorgeschlagen, welches dadurch eine hohe Flexibilität bei gleichzeitig großer Wiederverwendbarkeit von Programmbausteinen gewährleistet. Es erlaubt beliebige Zusammenstellungen mehrerer Volumendatensätze, Transferfunktionen, logischer Operatoren, Deformationen und Beleuchtungsalgorithmen, wobei es stets deren korrekte Zusammenarbeit garantiert. Die massive Rechenleistung moderner GPUs kann somit flexibel in dem datenflussbasierten Verfahren verwendet werden.

Es wurden viele GPU-basierte Visualisierungstechniken vorgestellt, die die traditionellen Illustrationstechniken imitieren. Trotzdem konnte die Bildqualität und Aussagekraft von Anatomieatlanten nicht erreicht werden. Einer der Gründe ist sicherlich die Abhängigkeit von klinischen Volumendaten, die nicht genügend Informationen aufweisen oder stark rauschen. Es wird gezeigt, dass vorab bekannte Informationen in dem vorgeschlagenen datenflussbasierten Verfahren auf einfache Weise zu der Visualisierung hinzugefügt werden können, um dem Informationsmangel entgegenzuwirken. Ein anderer Grund für die bisher nicht erreichte Qualität ist aber sicherlich der Aufwand, der damit verbunden ist, verschiedene benötigte Techniken zu kombinieren. Das entworfene datenflussbasierte Verfahren hingegen erlaubt das beliebige Kombinieren verschiedener Verfahren, ohne etwas programmieren zu müssen, wodurch der Aufwand auf ein Mimimum reduziert wird.

Die Anwendbarkeit des Verfahren wird mit CT Volumendaten mehrerer menschlicher Herzen überprüft, indem diese so visualisiert werden, wie im Anatomieatlas.

Acknowledgments

This thesis would not have been possible without the support of others. First, I would like to thank my advisor Prof. Dr. Gitta Domik for her time, intensive discussions about visualizations, and the collaboration and instruction in many other areas including education, controlled experiments and UX-testing in games. I would also like to thank Prof. Dr. Horst Karl Hahn and Prof. Dr. Gerd Szwillus. Prof. Hahn offered me the great opportunity to have insightful one-on-one discussions and presentations at the Fraunhofer Institute MEVIS. Prof. Gerd Szwillus showed me different conceptional views to my topic. I would also like to thank Dr. Matthias Fischer for his piece of good advice and calming words. Special thanks go to Prof. Dr. med. Wolfgang Burchert and his team from the Heart and Diabetes Centre NRW, for collaboration, many fascinating datasets and for letting me experience clinical routines within the last five years.

I would also like to thank the students Matthias Bolte, Robert Gmyr, Simon Kokerbeck, Sebastian Sindelar, Tobias Isenberg, Nico Bredenbals, Sebastian Lehrig, Peter Stilow, Alexander Weinhold, Philip Wette, and Sebastian Pahnke that worked with me during their Bachelor and Master theses to enhance the software "Volume Studio" to its current state. The same is true for our project group "Volume Rendering". All of them did a great job and, if so, are deservedly authors or co-authors in the listed publications. Especially, I would like to thank Matthias Bolte, who was also a student assistant at the very beginning of Volume Studio. I have to thank him for both, half a year of great design discussions before programming started that essentially shaped this framework, and for all his help during the first three and a half years.

But I would also like to thank all the other students that broadened my mind with off-topic theses. Especially the collaboration with the department of cognitive psychology in several Bachelor and Master theses, seminars, and some common publications taught me to conduct user studies with care. Namely, I want to thank Prof. Dr. Ingrid Scharlau, Jan Tünnermann, Frederic Hilkenmeier and Birte Heinemann.

Finally, I would like to thank my friends, all my family, especially my parents Rose and Michael, and Carmen, my girlfriend. Throughout the five years of this PhD studies, you always supported me, gave me strength and love, although I had little time for you.

Contents

Chapter 1

Introduction

Hand-drawn medical illustrations help to understand and impart the morphology and functioning of organs since centuries. They have been improved over and over again, and are the favored resource to describe facts among physicians. Nowadays, modern imaging techniques allow for completely new methods of data acquisition, and latest computer science provides unknown possibilities for visualization. Hence, computer generated visualization has joined the traditional, hand-drawn illustration in many areas. While traditional art is only capable of displaying information a posteriori and for selected individuals only, todays technology allows for an a priori visualization of the data of any person. Nevertheless, the quality of the traditional art has not been achieved yet. This PhD thesis will address the technical prerequisites, that are necessary to be able to generate high-quality illustrations as in anatomy atlases but based on individual patient data.

Visualizations are well suited to illustrate data and facts. Even in imaging techniques, visualizations can be more expressive than the acquired images themselves. Especially in radiology, that is defined by images that do not correspond to the natural vision, untrained viewers are often not able to interpret the original images. This is all the more the case when considering three-dimensional volumetric radiologic imaging, namely computed tomography. This technique acquires several hundreds of image slices, that are difficult to observe for laymen. Even trained physicians sometimes reach their limits, when confronted with these images. Recently a study [DVW13] showed that 20 out of 24 radiologists searching for nodules in a CT dataset of the lung missed an artificially inserted dancing gorilla of the size of a matchbox. Although it was not the task to find the gorilla, it shows that watching these images is very demanding. In combination with the time pressure, prevalent in todays clinics, it is obvious that the visualization, including preparation, selection and maybe reduction of data is an important factor to medical imaging.

Subsequently, many great and elaborate visualization techniques have been proposed that address these problems and reveal visual insight into the data. This includes cuts, deformations and illumination algorithms, which are the most important techniques for visualizations as in anatomy atlases. Most of the proposed visualization techniques are based on applications programmed especially for this purpose. However, many good visualizations and especially illustrations in anatomy atlases are characterized by a combined use of several techniques. *Hence, a major problem is the development of a system that supports a wide variety of visualization techniques while making distinct techniques rapidly reusable in combination with others.* Reusability has been a topic of major interest in the field of model-driven software architecture for many years. One substantial technique that evolved to cope with this problem, is to structure the software into smaller and smaller building blocks. In combination with the size of the building blocks their functionality decreases. At some size the functionality might be a basic procedure that can be reused in many situations. Thus, building blocks are a substantial technique to support reusability. Another substantial technique is the visual representation of the software's structure and the functioning. This allows for a much easier understanding of the software description, resulting in less errors and a more rapid development. Both techniques, reusable building blocks as well as a visual representation of the software, are advantageous for combining visualization techniques, too. A major difficulty, though, is the efficient usage of CPU and GPU algorithms in combination, without bothering the creator of the visualization with this problem.

Assuming that such a system that makes all techniques combinable in an easy way would be accessible. The aim is now to use this system to create a visualization that compares to the hand-drawn illustrations in anatomy atlases but rests upon individual patient data. Necessary information has to be retrieved from the dataset and needs to be mapped to visual attributes. Many data characteristics besides the ordinary data value have been used to distinguish tissue e.g. by size or gradient. Nevertheless, often the dataset simply does not exhibit the information we would need for an appropriate distinction of tissue. However, much information arises from the knowledge of the trained radiologist. He is able to interpret two features with the same data characteristics differently, if he takes into account the context of the organ. Integrating additional knowledge into the program in a form that is easily processible for computers would allow for a visual representation of these differentiations and in turn would make the visualization more readable for someone who is not a trained expert. That is an important requirement towards visualizations that are as easy to understand as hand-drawn illustrations in anatomy atlases.

Both, the flexible visual prototyping of GPU-based visualizations consisting of reusable building blocks as well as the integration and processability of additional information which substitutes some of the expert's knowledge is the technical aim of this thesis. The application of this will be a visualization catalogue that produces interactive visualizations of the heart that are based on individual patient data. To test the approach's flexibility uninfluenced it is important to create a visualization catalogue, that is conceived without knowledge of this approach or systems that are similar. Otherwise the visualizations, and thus the task to test the approach, will be influenced by the capabilities of the approach. The hand-drawn illustrations of the heart in the "Atlas of Human Anatomy" by Frank H. Netter [Net10], published in 1989, serve well for this purpose. They are still widely used among students and physicians to understand the complex structure of the heart's cavities, and they have been drawn in a time, when interactive volume rendering did not exist. In case of the human heart, the structure of the anatomy varies considerably between patients, and it would be desirable to have such illustrations for individual patients to understand the distinct patients' anatomy. As a result the digital visualization catalogue will not only be a feasibility test for the technical aims of this thesis, but it will provide additional spatial insight into individual hearts that can hardly be learned from other visualizations. For the realization of this catalogue, datasets of CT-angiographies of the heart are used, which are at the moment the best morphological imaging technique for this organ in terms of resolution.

Chapter 1 gives a rough overview to traditional and computer-based medical visualizations. In *Chapter 2* a flexible dataflow-based multi-volume rendering framework is proposed, that allows for arbitrary logical combinations of several volumes. In *Chapter 3*, this first version of the framework is enhanced by a new processing strategy that additionally allows for cuts and deformations, while assuring correct illumination. *Chapter 4* briefly explains how to use these techniques to create visualizations as in anatomy atlases and how to automatically adapt these visualizations to individual patient data using a model of the heart. In *Chapter 5* a visualization catalogue imitating Netter's illustrations is presented, which will be used in combination with a user study to evaluate the framework's capabilities. *Chapter 6* concludes this thesis.

1.1 Contribution

The main contribution of this work is a modular shader framework for GPU-based volume rendering which allows the user to *deform*, *illuminate*, and *combine* datasets in multi-modal scenes. In medicine, *deformed* visualizations are important to depict complex organs clearly laid out and uncovered while *illumination* is important for spatial comprehension and perception of structural details. A *combination* of modalities is required to extract all contained information from the data, which can be necessary for a comprehensive diagnosis. The main advantage of the framework is, that such GPU-based visualizations can be created rapidly and easily by connecting reusable building bricks. The union of all four – deformation, illumination, logical combinations and reusable building bricks – is novel and original.

The visualization of CT-angiographic datasets of the heart has been chosen as field of application. It will be shown, how the dataflow-graph of the shader framework can be utilized to visualize this abstract data of absorption values in a way that looks like real hearts. That incorporates the extraction of missing information about color and material to depict the different types of tissue realistically. To reach this aim, different types of transfer functions are *combined* and additional model-based information is added. The ability of the framework to correctly process combinations of building bricks, that *deform* and *illuminate* datasets, will be used to create visualizations of dissected hearts that reveal the complex inner structure of the cavities, just like in anatomy atlases. The literature provides no solution to generate such visualizations efficiently from individual patient data. The proposed methodology is a step towards this aim.

1.2 Related Work

This section presents publications that are related to the topic of this thesis. It will put this thesis into a wider context and will present alternative approaches. For this reason, primarily publications at a higher level are listed here. Publications that are similar on a technical detail on the the other hand, can be found in the related work sections of the respective chapters, close to where the technique is described. The next Section 1.2.1 focuses on shader frameworks, while the following Section 1.2.2 addresses anatomical illustrations.

1.2.1 Shaderframework

Many modern volume visualizations use GPU-based volume ray casting [KW03], which is superior to texture slicing [CCF94, CN94] in terms of flexibility and image quality and compared to CPU-based approaches, it is superior in terms of interactivity. However, graphics programming, and especially GPU-based graphics programming is complex, error-prone and time-consuming. Reasons are the programming at a low (close to hardware) level and the extraordinarily difficult debugging. Even so, programming new visualizations often implies recurrent subtasks, which immediately suggests to reuse parts of programs. To account for the complexity and error-proneness of GPU-based graphics programming, as well as the demand for a simple reuse, this thesis proposes a visual programming concept that structures the visualization task into smaller and easier sub-tasks. Building bricks, that solve these sub-tasks, can be arranged and combined visually in a dataflow graph. Although visual programming concepts are thoroughly researched, only a few approaches have been presented that consider GPU-based ray casting. In most applications the flexibility of programming is substituted by the possibility to chose between a fixed number of scenarios, e.g. one volume, two volumes, with or without illumination. This is not objectionable, great visualization programs have been presented that serve a specialized but fixed task, e.g. the diagnosis of coronary heart disease [KGK*12]. However, this is not the aim and not the topic of this thesis. This thesis is about efficiently creating *new* visualizations. Bitter et al. [BvUW*07] extensively compare and test four frameworks (SciRun [SCI14], MITK [Ger14], VolView [Kit14], MevisLab [MF14]) on their ability to create new visualization programs, which is great. The focus of this comparison, though, is the preprocessing and barely the rendering. Hence, the related work, that will be presented in the following paragraphs, is selected with a focus on volume rendering and the constraint that it does not predetermine a number of fixed scenarios.

In the framework Voreen, Meyer-Spradow et al. [MSRMH09, MSRMH10] structure the ray casting in three OpenGL sub-tasks and four shader sub-tasks. These sub-tasks are filled with placeholders and can be replaced by concrete techniques. The three

OpenGL sub-tasks are meant to define the rays of the ray casting and to start their processing in the shader. The four shader sub-tasks are gradient calculation, transfer function, shading and compositing. A change of this structure is not intended, hence, visual representation is unnecessary and missing at the shader stage. On the stage of complete image-buffers a dataflow graph can be created to allow for image level intermixing based on the depth-buffer. In volume rendering, this can create practicable results if the scene does not exhibit many semi-transparent parts. If possible, dynamic branching on the GPU is avoided by distinguishing cases during compilation of the shader. This is remarkably easy because shaders are compiled at run-time anyway. In fact, all approaches that can be found in the literature utilize this performance benefit.

In the framework MevisLab, Rieder et al. [RPLH11] aim at enabling and disabling rendering components as easy as possible. The components are arranged in a linear list, thus, the order of execution can be manipulated. The results of a component are written to a global data structure that is handed over to the next component. It is not intended that components at a later position of the list influence the run-time behavior of components at previous positions. As a result, adding new deformations, cuts, or combinations of transfer functions might necessitate a modification of shading and shadowing algorithms. The visual representation of the dataflow is a list.

Like all others Rößler et al. [RBE08b, RBE08a] compose their shader program during runtime from individual components. Compared to our graph-based approach, the specification of the processing is done in a tree structure that does not allow for a logical combination of data values. A fixed set of information for illumination, transfer functions and other nodes in the tree, is extracted at the beginning of the processing. This includes the gradient, curvature, and other information. As a result, this information can not incorporate modifications at a later stage done by transfer functions or elastic deformations, which is necessary for a correct illumination of these techniques.

The Octreemizer described by Plate et al. [PHF07] is very similar to the first development stage of our software as is described in Section 2.6 "Creating Volume Visualizations flexibly"; that is, any logical combination of transfer functions is possible. Differences are: the Octreemizer uses texture slicing, our approach uses ray casting, and the preprocessing is visually separated from the shader framework, while our approach nests both, preprocessing and rendering, in one single dataflow graph, which leads to a better overview. Compared to the second development stage, as described in Section 3.2 "Demand-Driven Dataflow", it is not intended that components at a later position of the dataflow graph influence the run-time behavior of components at previous positions. As a result, deformations and complex logical combinations can not be shaded correctly without major changes to the program each time a new visualization is created using these features.

Other important development frameworks that can be used to create new volume visualizations include SCIRun [SCI14] and ImageVis3D [CIB14] originating from the SCI institute in Utah, DeVIDE [BP08] from the graphics group at the Delft University of Technology and XIP [PTK*08]. However, they do not concentrate on a modular shader framework.

1.2.2 Atlases of Anatomy

The illustration of anatomy has a very long history and, in the meanwhile, has become a many-faceted type of communication. In the Western World, Andreas Vesal is said to be the founder of the modern era of anatomy. In 1543 he published the first anatomy atlas that was based on sections of human cadavers "De Humani corporis fabrica" (see electronic version [Ves43]). Earlier works are mainly based on animals' cadavers, due to religious restrictions. Since then, more and more detailed atlases have been published. In the last two centuries, the most influencing have been "Gray's Anatomy" [GC58], Netter's "Atlas of Human Anatomy" [Net10] and the "Prometheus" [SSS09], which is also known as the "Thieme's Atlas" in Anglophone countries. The "Prometheus" is the newest of the aforementioned atlases and uses illustrations by Karl Wesker and Markus Voll. The illustrations were created by using graphic tablets to electronically draw the figures. An exceptional position among all atlases is Rohen's atlas [RYLD06], which features photographs of dissections instead of drawings. Opinions diverge on the optimal representation. In terms of understanding the anatomy, hand-drawn illustrations are more clear and easier to perceive. What all atlases have in common is that their creation is exceptionally expensive, because it takes interdisciplinary experts and a lot of time to create them. Unfortunately, this is also true for using these images in a PhD Thesis for the sake of comparison. Only some older illustrations lost their copyright and can be used for free. Figure 1.1 shows a cut through the heart, as published in Gray's Anatomy in 1918. *Newer* illustrations with watermarks can be found at the following links, however, up-to-date illustrations are not publicly available.

- http://www.netterimages.com/image/704.htm
- http://www.netterimages.com/image/738.htm
- http://www.netterimages.com/image/730.htm

They all show detailed structures of anatomy, highlighting different aspects. The highlighting techniques are either artistic or surgical, and include coloring, illumination, labeling, magnification, cuts, and deformations.

Atlases using Computer Graphics

Wan et al. [WLC*12] mention that in the meanwhile computer graphics begins to transform anatomical atlases. Modern computer graphics allows for interactive and ani-

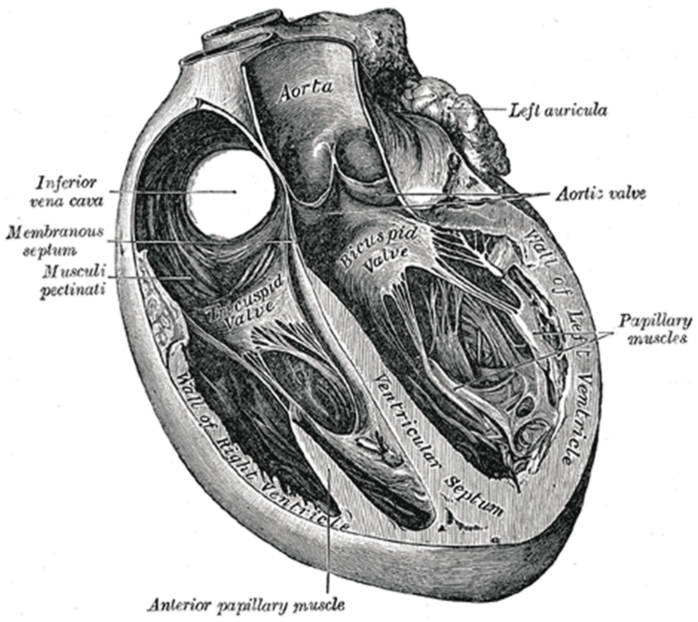


Figure 1.1: *Figure 498 from Gray's anatomy [Gra18]: "Section of the heart showing the ventricular septum".*

mated visualizations. Several digital atlases are available for human anatomy, including the Visible Body [Arg14], Cyber-Anatomy [Cyb14], and Zygote's Anatomical-Model Library [Zyg14]. Although the artistic procedure has been replaced by 3D modeling and texturing, these atlases compare fairly well to their hand-drawn counterparts; both are created by specialized artists in a very manual and time-consuming way. Hence, any current atlas available is created once preferably for an average human. However, the variability of anatomy among humans is considerable. That is, individual differences, the amount of fat tissue and diseases significantly influence the anatomy. For creating individual atlases, the creation process needed to be dramatically easier.

In computer science, there have only been a few approaches that have been proposed that try to easy the creation of anatomical illustrations. The results are either polygon-based, or volume-based. Ju [Ju05], DeLaurier [DBB*08] and Wan [WLC*12] create polygon-based atlases, which are similar to the digital atlases and can use traditional pipeline rendering. The difference is, that these atlases are created from scanned volume data, which – at least in theory – enables individual non-invasive atlases. Höhne et al. [HBR*92] create an anatomical atlas which renders volume data using the marching cubes algorithm [LC87]. They provide great tools for exploration, like coloring, cuts, and labeling. However, in 1992 rendering took about a minute and and interaction responses up to 10 seconds. All four approaches still require several tools and a tedious workflow for creating the atlas illustrations. An automated workflow could help to decrease the amount of necessary work.

Although used for applications other than anatomical illustrations, a lot of techniques that can be utilized for automating the creation of such illustrations, have been done in the field of statistical atlases and computational models. Since 2010, the annual international workshop STACOM [CPR*10] concentrates on “Statistical Atlases and Computational Models of the Heart”. Several techniques have been presented that are capable of locating the individual parts of the human heart in volume datasets. Besides the big cavities, even smaller parts like the valves can be automatically located with a considerable precision (compare [IVM*10]). This thesis will utilize the advances in the research area of heart models to enhance volume visualizations in such a way, that anatomical atlas illustrations can be created automatically for individual patients.

Often, when imitating illustrations in volume rendering, the term “illustrative volume rendering” is used. This embraces a wide field of techniques, and most of the techniques are different to the concept presented in this thesis. One aspect aimed in illustrative volume rendering is to imitate the brushwork of the hand-drawn illustrations, in form of hatching [DCLK03], stippling [LMT*03] and silhouettes [KWTM03]. These techniques can ease the perception of the surfaces' orientation but were originally invented to overcome missing shades in printing. Modern colored hand-drawn medical illustrations avoid a visible brushwork, hatching and stippling. Thus, some of these techniques seem to fulfill mostly artistic aims [RBGV08].

However, another aspect in illustrative rendering is context preserving emphasis. This is important to the topic of this thesis because it includes cuts and deformations. Chen et al. [CCI*07] categorize different approaches to volume deformations quite well. They enable views into opaque objects that would not be possible without. Considering the heart, which is an organ with a lot of inner cavities, cuts and deformations are very important. Hence, in this thesis a whole chapter is dedicated to cuts and deformations (Chapter 3). It lays emphasis on correct illumination and rapid prototyping of deformed volume rendering.

Chapter 2

Flexible Multi-Volume Rendering

There are several competing imaging techniques with adherent advantages and disadvantages. Sometimes, it is useful to use several scanners to gain new information by combining their acquired images. Hence, modern software must be able to visualize many different situations and data types. When new situations – in terms of data – occur, visualizations have to be developed. Creating a "good" visualization is a difficult task which can be done by either trial and error and/or by using heuristics. Accordingly, it is okay that the developer needs to be a domain expert. Though to try out visualizations flexibly, it is important that the development of a visualization does not take a long time. As a deduction, the software system needs to be able to process several datasets at the same time and also needs to provide an interface to prototype new visualizations using this data. It is difficult to design such a software system that does neither introduce restrictions to the mapping from arbitrary input data to arbitrary visual attributes, nor necessitates a major reprogramming for each visualization. In the best case programming is not necessary at all.

Another problem is the technical restrictions given by hardware-accelerated visualizations which do not allow every operation a user can think of. For example, the graphics hardware allows for a very fast processing of the data, when processing it in parallel, e.g. during rendering. In contrast the CPU is more efficient when sequential algorithms run on the data. A rapid prototyping system should only allow for a processing of the data that is technically viable. At the same time it should hide the technical background from the user, or at least not demand the user to understand it for a correct usage of the system.

This chapter can be divided into two parts. In the first part some basic information about volume rendering, volume data, transfer functions, and illumination is provided in a compactness and a selection that is appropriate to understand the matter of this thesis. Since volume rendering is not a new topic, detailed descriptions can be found in the textbooks by Engel et al. [EHK*06] and Preim and Bartz [PB07]. In the second part, multi-volume rendering is explored. Multi-volume rendering is a rather new topic and comes with new possibilities, as well as, new visualization problems. An integrated, but flexible approach will be presented in Section 2.6 that breaks down the visualization problem, including the processing on the GPU, into smaller sub-tasks that are much easier to solve. This way a rapid prototyping of volume visualizations is possible.

In the context of this thesis, *multi*-volume rendering is essential. Much of the additional information that is necessary for atlas visualizations needs to be saved in extra volumes. This way the data can be evaluated very quickly to allow for interactive rendering. A flexible rapid prototyping is essential, too. In the case of visualizations like in anatomy atlases, the visualization task is so demanding, that the solution process needs to be tested by trial and error. To write new software for each trial is not practical.

2.1 Direct Volume Rendering

At the very beginning of computer based volume rendering Kajiya [KH84] and Levoy [Lev88] contributed most of the theories that are still valid today. From an algorithmic point of view many aspects have already been researched at this time. Especially Levoy proposed several algorithmic optimizations regarding speed and quality [Lev90]. Later these concepts where transferred to the GPU [CCF94, CN94, KW03, RGW*03], which enabled interactivity and, thus, the possibility to use volume rendering in practical applications. Nowadays many leading topics of conferences deal with the necessity to close the gap between research and application. The most fundamental excerpt of principles of direct volume rendering will now be explained. As the name suggests there is also the possibility for *indirect* volume rendering. Indirect volume rendering does not visualize the volume data itself, but shows any proxy-geometry, whose spatial characteristics or visual attributes reflect some attributes of the volume data. Indirect volume rendering will not be discussed in this section.

The most widely used model for direct volume rendering is the emission-absorption model. At each spatial position in the volume colored light is emitted, which will then be attenuated by occluding parts of the volume on its way to the camera. Hence, the light value I at a pixel of the screen is defined by the integral of the light that is emitted on the ray towards the camera, scaled by the absorption between the the camera and the emitting position [DH92]:

$$I(\vec{x}, \vec{w}) = \int_0^T I(t) \cdot e^{-\int_0^t \sigma(t')\, dt'}\, dt \tag{2.1}$$

with $\vec{x}$ being the center of projection, $\vec{w}$ the normalized ray direction, $I(t)$ the light emitted at position t in direction $-\vec{w}$ and $e^{-\int_0^t \sigma(t')\, dt'}$ the absorption coefficient summed up to position t regarding the absorption function σ. T is the maximum distance from which light is emitted, e.g. ∞. This model neglects reflection, refraction and scattering, hence, it is called ray casting, and not ray tracing. Figure 2.1 explains this principle visually.

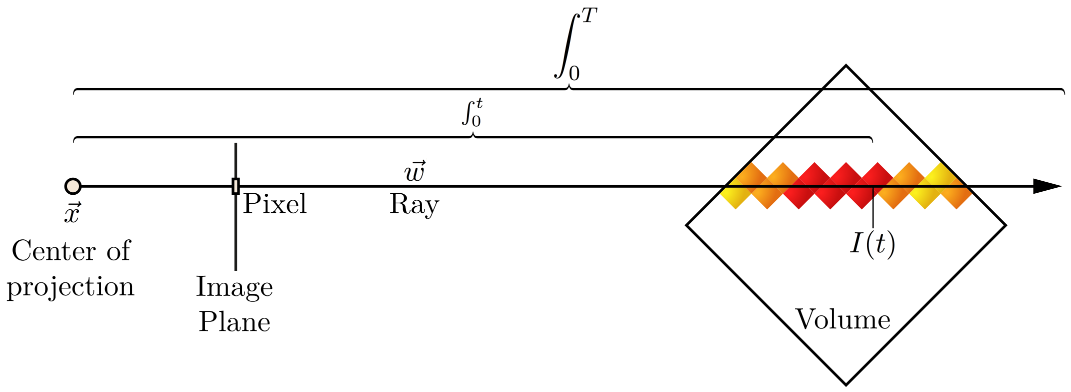


Figure 2.1: *Principle of direct volume rendering. The light intensity I at each position t is computed using the value of the volume at this position. To define a pixel's color the light intensities are integrated over all t along a ray and attenuated by the integrated absorption coefficients up to t.*

The emitted light I at each spatial position is not necessarily a constant scalar data value, it can rather be a term that depends on transfer functions, the camera position, and local gradients, which are typically used as surface normal in volume rendering. This way coloring, shading, and even more complex illumination situations can be modeled from a dataset, which originally stores only one-dimensional scalar values. $I(t)$ actually necessitates the possibility to calculate the emitting light at every continuous spatial position. Common volume datasets on the other hand, are a set of data values specified at discrete positions in a grid. To overcome this, data has to be interpolated in between the discrete positions. The most obvious solution would be to use the data value of the discrete position, which is nearest to the continuous position. However, a better approach is to interpolate the data values trilinearly. Figure 2.2 shows a comparison between nearest neighbor interpolation and trilinear interpolation when using an emission which emulates shading. Even more precise interpolation schemes are possible but not supported by hardware acceleration.

Now, that the discrete data is continuously interpolated, the integral itself will be discretized for an efficient algorithmic implementation. Taking n samples in a distance of 1, equation 2.1 can be written as

$$I = \sum_{0 \le i < n} \left(I(i) \cdot \prod_{0 \le j \le i} e^{-\int_j^{j+1} \sigma(t')\, dt'} \right) \tag{2.2}$$

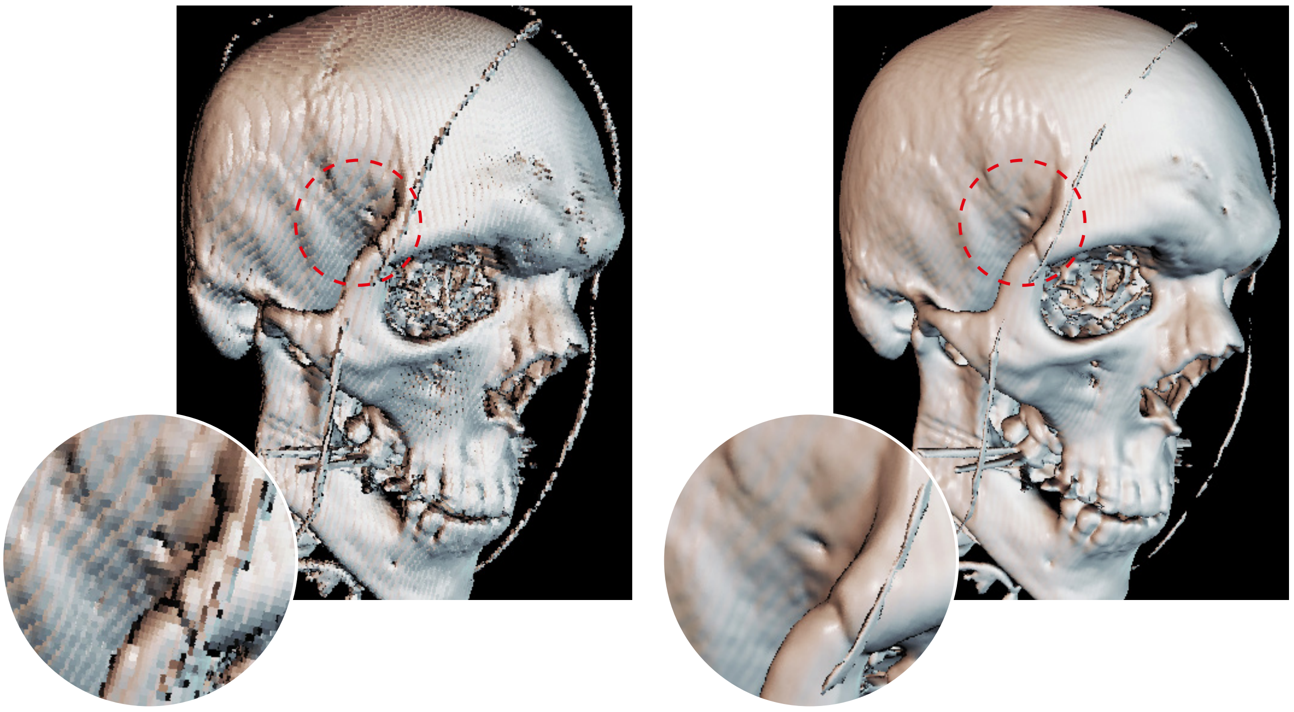


Figure 2.2: *Comparison of nearest neighbor interpolation and trilinear interpolation using an emission function which emulates shading. The marked regions are magnified for a better visibility of the effect. CT data sources obtained from the Visible Human Project [Vis14].*

with $e^{-\int_j^{j+1} \sigma(t')\,dt'}$ being the translucency of step j. The opposite of the translucency equals the commonly used *alpha*:

$$\alpha(j) = 1 - e^{-\int_j^{j+1} \sigma(t')\,dt'} \tag{2.3}$$

Now, let β be the portion of light, that is absorbed by the volume in steps 0 to i:

$$\beta(j) = 1 - \prod_{0 \leq j \leq i} 1 - \alpha(j) \tag{2.4}$$

then, the overall sum of light at a pixel can be written as

$$I = \sum_{0 \leq i < n} I(i) \cdot (1 - \beta(i)). \tag{2.5}$$

Using the over operator [PD84] this formula can be efficiently implemented in $\mathcal{O}(n)$ using a loop that iterates over i and reuses the result of iteration $i - 1$.

2.2 Modalities

The property of tissue that is acquired compares to the term “modality”. The modality in turn is not to be put on a level with the imaging technique, because some imaging techniques are able to acquire different properties of a tissue. Magnetic resonance imaging, for example, can be parameterized to acquire several different properties of a tissue. Additionally, specialized tracers can be used to emphasize distinct properties of tissue, resulting in distinct modalities. The decision which modality will be acquired depends not only on the requirements, but also on the type of organ, the symptoms, the patient, the accessibility, the available time, and the costs. Depending on the the patient’s condition, allergies, and age some of the imaging techniques can be harmful and should be regarded. For the one that performs the physical examination, the informative value and the properties of a modality are of utmost importance. The same is true for the one

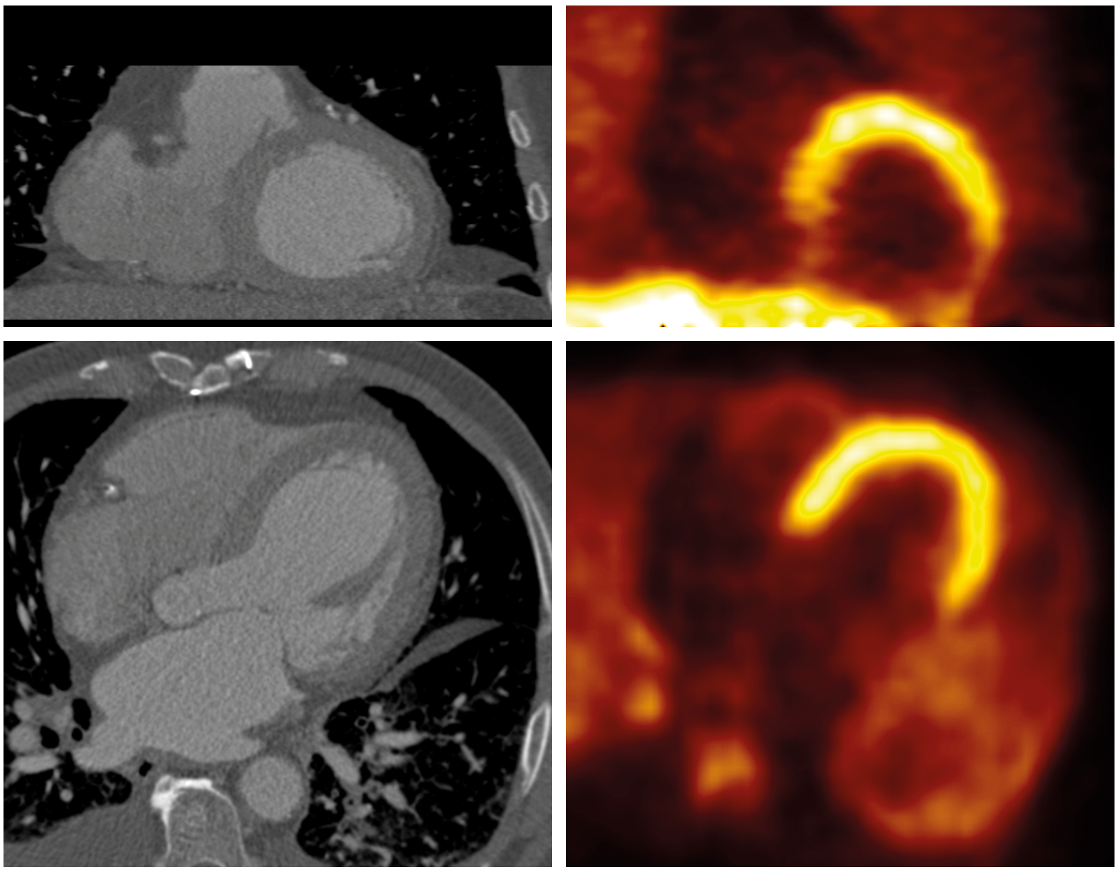


Figure 2.3: *Comparison between CTA data (left) and PET data (right) of one patient. Congruent regions are shown.*

who creates the visualization, but he will need some technical information too. The next three subsections will categorize modalities according to different points of view. Only information that is related to the topic of this thesis will be handled. A detailed but uncomplicated explanation of CT can be found in the book by H. Alkadhi [ALSS11], and a detailed explanation of MRI and PET are located in Lipton [LK08] or Phelps [Phe06], respectively.

Informative Value First of all, every modality is defined by its imaging aim, which can be one of morphological or functional assessments. Figure 2.3 left and right show the same patient at the same slices; left in a morphologic CT angiographic dataset, and right in a functional PET dataset. The morphology describes the structure and shape of tissue. Morphology, as well as the pathologic state of an organ that can be derived by the outer shape, can be made visible by different measurements, like reflection of ultrasonic, or absorption of X-ray radiation. A special technique which aims at depicting vessels is the so-called angiography. In an angiography, it is not the vessel, but the content that is made visible to infer the inner form of the vessel. Blood has a very similar absorption value to X-ray radiation as muscles and some other tissues. That is why in most types of angiography, like computed tomography angiography, a contrast agent is injected to increase the absorption value of blood. The second imaging aim, functional assessment, does not depict the shape but the function or functional efficiency of an organ. In most cases, this functional efficiency is made measurable by handing radioactively marked tracers that are metabolized. A wide variety of tracers are available that are used in different metabolism processes and, thus, depict different bodily functions. Only a few techniques are able to depict some functions without any tracers, like the Functional Magnetic Resonance Imaging (fMRI) that can measure the oxygen percentage. Table 2.1 lists imaging techniques that are used in modern medicine and associates them to functional or morphological. Due to the amount of techniques

Type of assessment	*Related imaging techniques*
3D Morphologic	X-ray Computed Tomography (CT)
	Magnetic Resonance Imaging (MRI)
	3D ultrasound
3D Functional	Positron Emission Tomography (PET)
	Single Photon Emission Computed Tomography (SPECT)
	Computed Tomography (CT) perfusion imaging
	Functional Magnetic Resonance Imaging (fMRI)

Table 2.1: *3D imaging techniques, that are used in human medicine, associated to type of assessment.*

the table only lists techniques that acquire 3D images and are used in human medicine. Some hybrid devices are able to acquire two modalities at the same time, like PET-CT, SPECT-CT or PET-MRI. The visualizations of this thesis try to mimic the morphological illustrations from anatomy atlases, which is why in this thesis, morphological imaging is most relevant. Nevertheless, there are hand-drawn illustrations in anatomy atlases that depict the functioning of organs.

Technique Beneath the informative value, the technical basis can be used to categorize volumetric imaging techniques. Not all imaging techniques are suitable to reconstruct volumes, because the physical measurement has to be able to pervade tissue. In the context of medical imaging four different physical techniques are used: *X-rays*, whose absorption by tissue is measured, *radionuclides*, which are injected into the body and whose radioactive decay is measured, *magnetic resonance*, that is the resonance from hydrogen atoms to strong magnetic fields, and finally, *ultrasonic*, whose echo reflected from tissue is measured. All medical imaging techniques can be associated to one of these techniques. In biology, confocal microscopy is used in addition to these techniques, which can be used to create volumes from small objects using *visible light*. Finally, simulation and algorithmic processing can create new volumes. The first two mentioned measuring techniques expose the patient to a radiation dose, which is why they are only allowed if the disease and the chances of therapy are reasonable compared to the harm of the assessment. In addition the dose has to be as small as possible, otherwise it would be easy to create sharp images without noise. Measurements based on magnetic resonance and ultrasonic are practically hazard-free.

Data Characteristics The characteristics of the data are crucial to the developer of the visualization software, because different characteristics typically need to be processed differently. Volume datasets in general associate one or more data values to many defined spatial positions. The data values can be one or more scalar values, vectors or tensors of a higher order. Physical imaging techniques associate only quantitative values. Nevertheless, it is possible to create volumes of nominal or ordinal values by simulation or algorithmic processing. A typical example of a volume with nominal values is a common segmentation. Spatial positions can be either ordered in a regular grid or distributed irregularly. Most techniques produce regular grids, which can be processed much more efficiently. The data of other techniques (like 3D ultra-sonic images) are typically rerasterized to a regular grid. To display volumes continuously the data values between the defined spatial positions are typically interpolated, i.e. the interpolated dataset associates a data value to *every* position in space. This is all the more important in multi-volume rendering, because the defined positions in space do not have to be congruent in two datasets.

2.2.1 CT-Angiographie

In this thesis almost only CT angiographic datasets are used (see Figure 2.4). Compared to other imaging techniques CT scans are extremely fast, which qualifies it for images of the beating heart at high resolution. Additionally, the morphologic nature of CT scans is just perfect for anatomical illustrations. Angiography highlights blood vessels

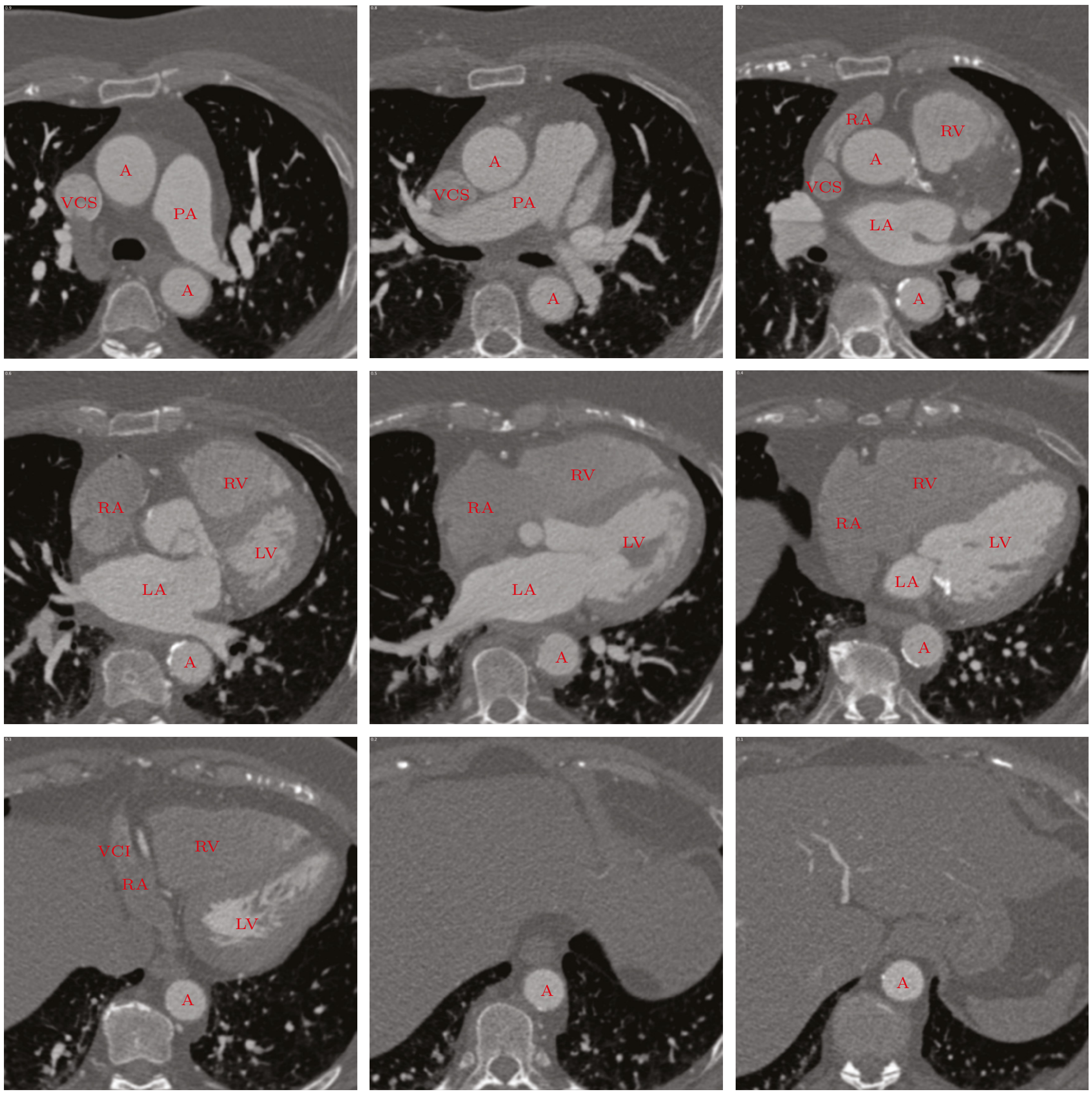


Figure 2.4: *Nine slices from a CTA dataset. Cavities and great vessels are labeled. A: aorta, LA: left atrium, LV: left ventricle, PA: pulmonary artery, RA: right atrium, RV: right ventricle, VCS: vena cava superior, VCI: vena cava inferior.*

using a iodine-containing contrast agent, which is necessary to distinguish blood and muscle tissue, which shows nearly the same data value otherwise. From a medical point of view, this contrast is used to examine the coronary arteries, the most common reason for coronary heart disease, and in turn is the most common cause of death in the world [FAF13]. The exposure to radiation and the cancer risk associated to this radiation on the other hand, is commonly assumed to be small [SBLM*09, BdGMK*09].

The patient is imaged, just when the contrast agent has filled up the arteries. The rest of the heart might not be contrasted that well at this point in time. Often, this makes it difficult to recognize the right ventricle and the right atrium correctly.

The volume data is reconstructed from the acquired data in a volume that consists of roundabout 512^3 voxels, each voxel containing 12 Bit that map to one of 4096 different Hounsfield units (HU). Hounsfield units are standardized to air (-1000 HU) and water (0 HU). Hounsfield units of particular tissues vary, depending on the consulted literature. Summing up the values of several resources [Fee09, ALSS11, Wik14] the Hounsfield values of muscle tissue can range from +10 to +50 HU, fat tissue from -100 to -50 HU and uncontrasted blood from +30 to +65 HU. On top of this, HU depend on the voltage of the X-ray tube, i.e. the Hounsfield value of a tissue can be different when using 80 kV or 120 kV [ALSS11].

Reducing the radiation dose to the minimum possible in human medicine introduces a major noise ratio. In the best datasets provided to us, the (normally distributed) noise still deviated the true HU by ± 48 units. Figure 2.5 shows a profile through a CTA dataset and a histogram of that dataset. The histogram was overlaid with the histogram of four segmented tissues (fat, muscle, blood in right ventricle, blood in left ventricle). The distribution of the tissues' data values shows a typical overlap of the data ranges. To ensure that these segmentations only cover the respective tissue, a very conservative erosion filter was applied to the segmentation. Hence, the distribution should only incorporate values of the segmented structure, however, the absolute number of voxels is inaccurate.

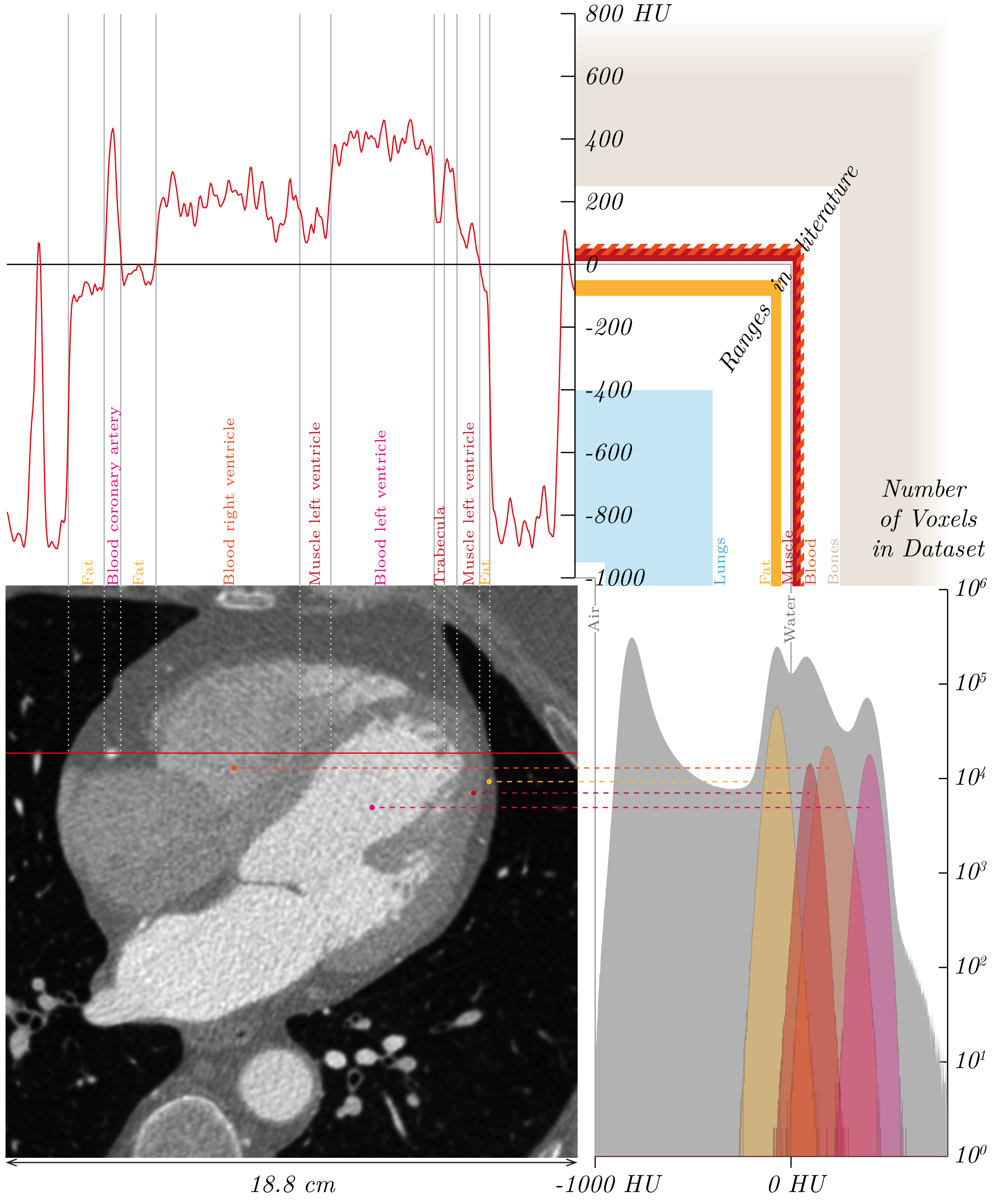


Figure 2.5: *Slice view, profile and histogram of a CTA dataset. The profile (top left) is calculated along the red line in the slice view, annotations describe the curve. In the histogram (bottom right) four regions have been overlaid to illustrate the distribution of fat, muscle, blood in the right ventricle, and blood in the left ventricle. The upper right corner shows data ranges in the literature for selected tissues (parallel to x and y axes of the histogram and profile, respectively).*

2.3 Transfer Functions

When rendering volume data, it is essential to cope with the massive mutual occlusion of data. In direct volume rendering, which is the used technique in this thesis, this problem is tackled by using transparency. Transfer functions are used to map data values to visual attributes like color and opacity. This way features can be highlighted or faded out. This section will provide an overview of different transfer functions stated in the literature. The content of this section is mainly based on the publication “Survey of Transfer Functions Suitable for Volume Rendering” by Arens and Domik [AD10]. Some of the introduced transfer functions are essential for creating the visualizations of this thesis. In Section 2.6 at the end of this chapter a superordinate concept for coherent usage of these transfer functions is provided.

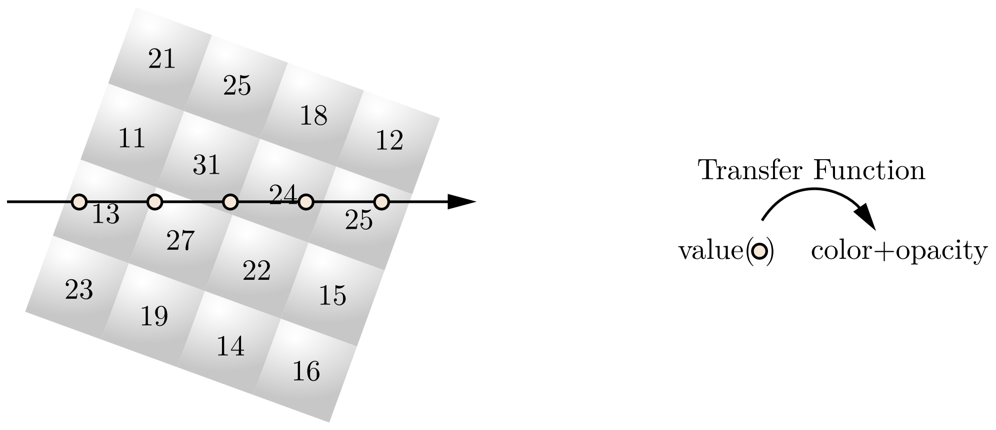


Figure 2.6: *A transfer function typically specifies a color and an opacity for each sample position along each ray depending on the value at this sampling position.*

As explained, transfer functions map data values to color and opacity with the result that features can be emphasized or hidden. This is done on a per sample basis, as shown in Figure 2.6. Since different organs often correspond in the signal they produce in medical 3D scanners, a simple distinction by measured value is not sufficient in most cases, thus, additional metrics are computed that help to differentiate types of tissue, leading to two- and multi-dimensional transfer functions. Often, decisions made by algorithms are not acceptable in medical usage, hence, automatic segmentation of a dataset is not applicable in every case, in spite of the fact that many segmentation algorithms are very complex to implement and can have a major runtime. Transfer functions are much more harmless and flexible since they typically do not provide a binary decision. Rather they demand a visual exploration of datasets (which is good [PLB*01]) and provide a sort of original view to the data in different contrast ratios. Nevertheless, higher dimensional transfer functions are not often used in practice, due to the fear of high amount of user interaction, missing precision or lack of intuition in their use.

For a further clarification which transfer function (TF) fits best for which field of application and task, their risks and properties must be compared. In this section the differences of the following six TF types *1D data-based TF*, *gradient-based 2D TF* [Lev88, KD98, KKH02], *curvature based TF* [KWTM03], *size-based TF* [CM08, WK09b], *texture-based TF* [CR08], and *distance-based TF* [TPD06] are compared in a table. It would hardly be possible to compare all published TFs, therefore a representative selection has been chosen, covering the most relevant types. Besides some fundamental properties, like used *metrics*, *real-time capabilities*, *memory consumption* and degree of *automaticity* we also give an assessment about the needed *user interaction*, *constraints* and the quality of *feature emphasis*. Additionally, widgets and methods are listed that work together with the TF types to support the user.

In literature the use of the term"transfer function" is not consistent. It can be a single specification of a mapping from its domain (that is the used metrics) to its range (in our case always a color and opacity), a technique how to specify such a mapping, or a type of such a mapping, defined by its domain and its codomain. We use the third, defining the type of a function by its domain and codomain, as suggested in [Kin02].

2.3.1 Previous work

The development of TFs for volume rendering started with volume rendering itself. Since tissues in medical data often share one signal and a 1D data-only driven TF is not able to distinct between samples with the same signal, a lot of effort was spent to develop two-dimensional or higher-dimensional TFs. The use of the gradient allowed to distinct transitions between tissues [Lev88], enhanced with the second derivative more distinctions in transitions were possible [KD98]. Driven by these capabilities of separation even more second and higher order metrics were calculated to distinguish between tissues, e.g. [HKG00, KWTM03, CM08, CR08, PRMH10]. When specifying these TFs in the domain of data values (Figure 2.7 left), instant visualization in the spatial domain (Figure 2.7 right) is quite important. Only direct feedback allows a trial and error approach for specifying a TF.

However, the higher the dimension of the TF, the higher the complexity of specifying it. Even two-dimensional TFs require a fair amount of user interaction to find a meaningful specification. To address this issue, intuitiveness of metrics and widgets, as well as automated techniques have been a research topic in the past. Concerning the first matter, size seems to be a much more intuitive metric than the gradient [WK09a]. In "The Transfer Function Bake-Off" [PLB*01] four possible ways for specifying a gradient based 2D TF are compared regarding their different user interface approaches. Two of them use histograms in the domain of data values for a first insight into the underlying

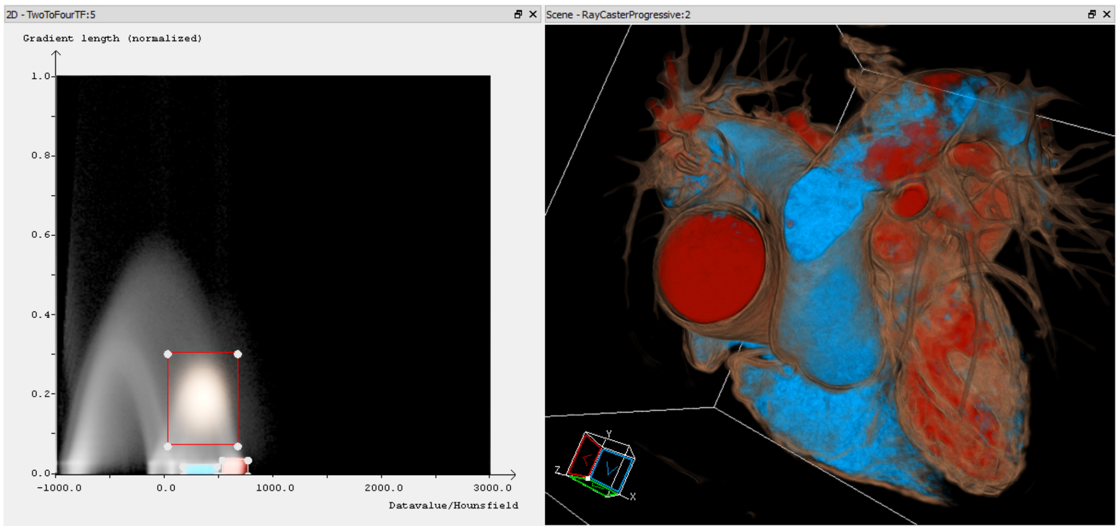


Figure 2.7: *The spatial domain (right) provides visual feedback of a TF that is specified in the domain of data values (left).*

data, leading to a better identification of features [KD98]. They are not very intuitive, though. Several widgets are presented in [KKH01] to support the complex handling in both domains, leading to a more intuitive so called *dual-domain interaction*. Other intuitive image centric interface approaches are presented in [MAB*97, Ma99, KG00], where the user chooses from a couple of renderings in the spatial domain, created by possible TF specifications.

Automated techniques are especially needed in higher dimensional TFs [TLM03, CR08]. Nevertheless, 2D-TFs are also supported by neural networks and spatial grouping, leading to semi-automatic interfaces [ZEAD08] and pre-colorized histograms in the domain of data values [RBS05]. But often the important process of data exploration gets lost when automated techniques are used.

2.3.2 Comparison of Transfer Functions

The analysis consists of both, a short per TF description with illustrations demonstrating their properties, as well as a comparison of the TFs in a table (see columns of Table 2.2). Several properties (rows of Table 2.2) are regarded:

- Real-time capability. Since it is difficult to compare this property objectively, we divide it into *real-time editability*, *pre-processing* and the possibility to compute the TF in shader-programs. Regarding the latter, the number of *texture look-ups* shall give a quantitative statement, as texture look-ups mostly constitute the critical part of rendering time. Due to caching, the expressiveness of this value can be inconsistent.

Times quoted are the original measurements in the referenced papers. The *ranking* is based on the amount of texture look-ups and time of preprocessing.

- Memory consumption during run-time, i.e. in particular the number of *additional volumes* needed during rendering (this is often a bottleneck, since they need to reside on the graphics hardware). If worth mentioning, *other data structures* used are listed, too. The *ranking* is based on the amount of additional volumes in shading and preprocessing stages.
- The ability to emphasize features. Since the TFs emphasize different kinds of *regions*, it is not possible to compare their quality in a fair way. Therefore, we only provide a *ranking* that considers how many features can be distinguished based on the number of used metrics. This ranking is vague and needs to be considered together with the *risk of a critical misinterpretation* caused by inadequate display of the data. This risk is influenced by the provided automaticity and the number of used metrics. Automaticity substitutes the trial and error specification that reveals properties of the dataset and is therefore critical. The number of used metrics is influencing, because every additional metric enhances the domain of data values, hence the space where a feature can be overlooked. Once again, the ranking does not mean that the texture-based TF is the best one, but is capable of distinguishing more features.
- *Constraints to the datasets*, especially how *noise* effects the TFs.
- Type and amount of user interaction as well as useful prior knowledge. The *type of interaction* describes the task that needs the main effort by the user. Specification on a single 2D diagram is preferred over specification on two 1D diagrams. The *duration* is indicated by the time needed to specify a TF with specialist's knowledge. Since this value varies heavily depending on the use case and task, it can rather be seen as a ranking among the TFs. If specialist's knowledge is absent, specifying is done by trial and error only and time needed can be much higher. This *helpful knowledge* is subdivided into technical (i.e. understanding of the metric computation) and application (i.e. understanding of the dataset and its respective values) knowledge, hence it is possible to decide if a certain TF is suitable for an explicit group of users.
- *Automated techniques* and *widgets* known to support the user to cope with the complexity of the input metrics. If entries are not listed, it does not mean that they do not work with this TF, only that there is no such experience listed in publications. The image-based approach in [MAB*97] provides a number of possible renderings and is therefore usable for all TFs. Hence, it is not mentioned explicitly.
- How much does a perfect segmentation benefit from the TFs, i.e. is it reasonable to use this TF additionally if you already have a precise segmentation, e.g. of the coronaries.

	Name	**1D data-based TF**	**gradient-based TF**	**curvature-based TF**	**size-based TF**	**texture-based TF** (automatic/semi-automatic)	**distance-based TF**
	Metrics	*Data value*	*Data value, gradient*	*Data value, curvature*	*Data value, scale*	*Data value, 20 texture metrics*	*Data value, distance to previously segmented structure*
	Literature (e.g.)	-	[Lev88,KKH02]	[KWTM03,PF05]	[CM08]	[CR08]	[TPD06]
Real-time capability	Real-time editing*	*Yes*	*Yes*	*Yes*	*Yes*	*No* **	*Yes*
	# 3D texture look-ups per sample	*1(data)*	*2/7* ** *(1 data + 1/6 gradient: needed for illumination anyway)*	*2/27* **	*2 (data + scale)*	*2 (data + label) / 21 (data + metrics)*	*2 (data + distance)*
	#1D + 2D look-ups	*0/1* **	*1*	*1*	*1*	-	*1*
	Time of preprocessing	-	*<1s for a dataset of* 256^3 */ -*	*<1s for a dataset of* 256^3 */ -*	*~10s for a dataset of* 256^3 *and 128 scales*	*261s / 233s for a dataset of* $256^2 \times 128$	*20s for a dataset of* 256^{3**}
	Real-time ranking***	○○○	●○○ **	●●○	●●○	●●●	●●○
Memory	Additional volumes during rendering	-	*1 / -*	*1 / -*	*1*	*1 / up to 20 (not necessarily full size)*	*1 (not necessarily full size)*
	Other data structures	*- / 1D-look-up-table*	*2D-look-up table*	*2D-look-up table*	*2D-look-up table, 3 volumes in preprocessing*	*20 metric-, 1 GLCM-, 1 run-length-volumes in preprocessing*	*Segmentation, 2D-look-up table*
	Memory ranking***	○○○	●○○	●○○	●●○	●●●	●○○
Emphasis	Emphasized regions	*Monotone areas*	*Surfaces* **	*Surfaces, contours*	*Monotone areas*	*Textured areas*	*Neighborhood*
	Emphasis ranking***	●○○	●●○	●●○	●●○	●●●	●●○
	Risk of a critical misinterpetation	*Low*	*Moderate*	*Moderate*	*Moderate*	*High (no exploration process)*	*Low*
Constraints	Constraints to dataset	-	*No additional information in blurry datasets compared to 1dTF*	-	-	*The bigger the dataset, the smaller the precision (memory consumption)*	*Only segmented data, not practicable for time-dependent data*
	Error-prone to noise	*Medium*	*Very*	*Medium / very* **	*Little*	*Very*	*No, only the segmentation*
User interaction	Type of interaction	*Specification on 1D diagram*	*Specification on 2D diagram*	*Specification on 2D diagram (checkbox or similar for contours only)*	*Specification on 2D diagram*	*Specify a number of different structures / setting of weights, mark one/two ROIs*	*Specification on 2D diagram*
	Duration	*<1 Min*	*1 – 2 Min*	*1 – 2 Min*	*1 – 2 Min*	*0 / >2 Min*	*1 – 2 Min*
	Helpful prior knowledge	*Application: values*	*Application: values, technical: gradient*	*Application: values*	*Application: values*	*- / technical: statistics (optional)*	*Application: values*
Automaticity	Type of automaticity	-	*Spatial analysis and neural networks (possible)*	-	-	*Clustering of structures with similar texture properties*	-
	Assisting widgets (possibility)	*1D histogram, data probe*	*2D histogram, data probe*	-	*2D histogram*	*Dictionary with predefined weights for metrics*	*2D histogram*
	Enhances a precise segmentation	*No*	*No*	*Yes, additional information about curvature*	*Yes, e.g. allows to colorize constriction in vessels*	*No*	*Yes, e.g. displays surrounding features*

Table 2.2: *Comparison of the six inspected transfer functions. Read the itemization in section 2.3.2 for a more detailed description of the rows.* **Considering the real-time specification of the mapping: Input metrics → RGBA (If 'Yes', the mapping implicates its computation in shader). There might be some other options, that require an update of the pre-processing.* ***See the description of the individual transfer function for more detail.* ****Ranking:* ○ ○ ○ *low quantity of computational costs / needed memory / distinguishable features,* ● ● ● *high quantity of computational costs / needed memory / distinguishable features.*

Since it is not possible to compare every TF, we compare a representative selection of types as explained earlier. Our argument for choosing the following 6 TFs is as follows: The 1D data-based TF serves as a reference. The 2D gradient-based TF leads to a quite common enhancement to distinguish surfaces. The 2D distance-based TF stands for TFs that are not based on the measured signal, but on additional information (in this case a segmentation). The 2D size-based TF provides a metric more intuitive than gradient, namely size. The texture-based TF is representative for higher-dimensional TFs that take into account a combination of several metrics, meeting their complexity with some automaticity. The 2D curvature-based TF was chosen as representative of the group of TFs mostly used for non-photorealistic rendering. Wherever possible, measurements are adopted from the original papers, but are enhanced with own evaluations and experiences (e.g. error-proneness to noise, helpful prior knowledge, or duration of specifying TF).

To compare the TFs visually, every TF description is supplemented by an explaining 2D Figure (Fig. 2.8–2.13). The Figures show four circular features, sharing one grey-level histogram, but are different in size, texture, and background. Those areas of the four features, that can be distinguished by the individual TFs, are colorized in red and blue. The content of the figures is rather contrived and is not supposed to compare the behavior of the TF types with real data. They serve for a simple visual explanation.

1D data-based transfer function: The one-dimensional data-based TF maps data values directly to color and opacity. Hence, it is not capable of differentiating between samples that share the same data value. Although they appear visually quite different, in Figure 2.8 the data-based TF can not distinguish the four features.

On the other hand, it is the TF that needs the lowest amount of computation power and memory. The mapping can either be implemented using a look-up texture or a procedural function (compare [EHK*06], Section 10.5.2). Furthermore, it provides a very high safety regarding errors that lead to a misinterpretation and needs little user interaction.

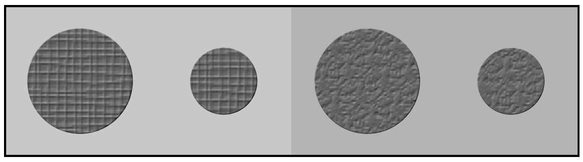


Figure 2.8: *The 1d data-based TF can not distinguish between the four circular features, since they have the same grey level. If trying to colorize one feature, all four would be colored.*

2D gradient-based transfer function: In volumes, the gradient-based TF is especially helpful to display surfaces of features [Lev88, KKH02]. It maps the data value and the gradient magnitude to color and opacity. Hence it is capable of differentiating between samples that share the same data, but not the same neighborhood. Thus, in Figure 2.9 it is possible to distinguish between the borders/surfaces of the left and the right features, that are placed on a different background. The gradient metric is highly error-prone to noise. On the other hand, gradient-based TFs only marginally provide additional information (compared to the data-based TF) in very blurry volumes, like PET-scans. Theoretically, it is possible, to not only highlight surfaces with a high gradient length, but also homogenous regions, which have a gradient length near to zero. These gradients can not be used for illumination, though, since their direction does not point to a meaningful direction.

The standard way of gradient computation (central differences) takes six extra texture look-ups. Since the gradient is needed for illumination anyway, this normally does not lead to a higher computation time. The gradient can be precomputed, reducing the texture lookups to 2. This results in an additional volume needed on the graphics card.

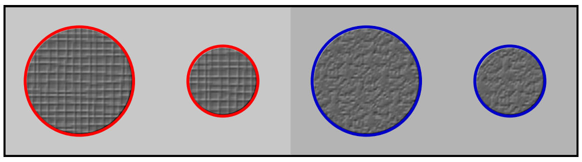


Figure 2.9: *The left and the right features are placed in a different environment. Since this leads to a different gradient at the feature borders (surfaces in 3D), the gradient-based TF can distinguish the borders of the left and right features.*

Curvature-based transfer function: In combination with the camera position, curvature-based TFs are mainly used to display a contour around features. More generally, it is possible to map data and curvature to any color and opacity, leading to ridge and valley visualization or emphasis of smallest bumpiness. In Figure 2.10 the borders/surfaces of the small and the big features can be distinguished, since they exhibit a different curvature. The effort to specify a curvature-based TF and the error-proneness depend on the usage. If it is used for contouring features, it is easy to specify, and noise has only medium impact. To pointedly highlight bumpiness, its effort is comparable to the gradient-based TF. Since many feature share the same curvature, it is rather difficult to emphasize a targeted feature only. Instead, some additional unintended emphasis will occur. Noise amplifies this behavior dramatically.

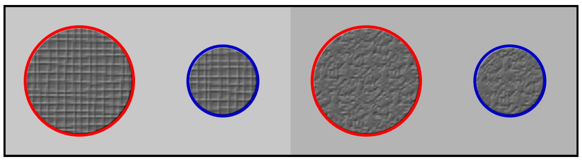


Figure 2.10: *The curvature-based TF is able to distinguish between the borders of the small and the big features, since they have a different curvature. With additional information about the camera position it can be used to contour the features in 3D space.*

The computation of the curvature needs at least a 26-point neighborhood. A real-time GPU variant of the approach presented in [KWTM03] can be found in the GPU Gems 2 [PF05], or in [HSS*05] for iso-surfaces; but curvatures can be computed in the preprocessing too, reducing the amount of necessary texture look-ups to two.

Size-based transfer function: The size-based TF maps data values and size to color and opacity. In Figure 2.11 the big and the small features can be distinguished.

The size of a feature is quite complex to define and has been determined in very different ways. This analysis relies on the approach of [CM08], which uses the scale space for size detection. This approach benefits from a low error-proneness to noise. Furthermore, the quite complex preprocessing can be performed using the GPU, and is therefore, fast.

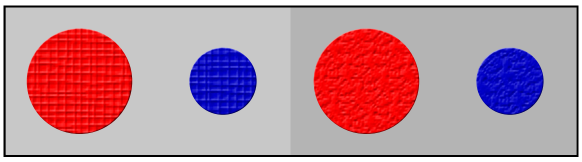


Figure 2.11: *The size-based TF is able to distinguish between the big and the small features.*

Texture-based transfer function: The texture-based TF massively differs from the other TFs in the number of metrics. It uses 20 texture metrics [CR08], six 1st order, seven 2nd order statistics and seven run-length matrices. Since it is too elaborate to map every of the 20 used texture metrics manually, the texture-based TF provides one full and three semi-automatic ways of specifying it. The fully automatic way uses clustering to recognize structures and only needs the number of structures as input. Since this automated technique is barely controllable, the ability to emphasize a determined feature is poor and the risk of a wrong result, leading to critical misinterpretation, high. That is why three further semi-automatic possibilities of parameterizing this TF are given. One, rather manual, where single statistics can be weighted, a second one to specify two structures to be distinguished, and a third option, that tries to separate a single marked structure. The first one needs a very high understanding of the used metrics and is therefore difficult and time-consuming to use. Table 2.2 holds entries for the “automatic” and the “semi-automatic” options. Since noise makes texture analysis difficult, datasets have to be very clean for this method. However, in the artificial Figure 2.12 the texture-based TF is able to distinguish the left and the right feature due to their different texture.

Computing the metrics and clustering is very expensive leading to high preprocessing times combined with very high memory consumption. Up to 20 volumes are needed; one for each statistical metric. Excepting the full automatic method, the memory amount and number of texture look-ups is the same during rendering. Two additional volumes are needed in pre-processing: one holding the Gray Level Co-occurrence Matrices (GLCM), and one for the run-length matrices. The texture analysis not necessarily requires full volume resolution, if less precision is acceptable. That is why this TF is not limited to small datasets, but involves a trade-off between time and precision.

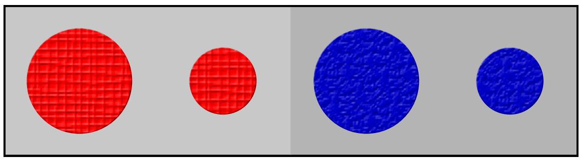


Figure 2.12: *The texture-based TF distinguishes the left and the right features by means of their different textures.*

Distance-based transfer function: The distance-based TF also differs from the other TFs, since it uses a previously segmented structure instead of the measured signals for an additional metric computation. This additional metric is the distance to the surface of the segmented structure. In Figure 2.13 the distance-based TF can colorize features in the neighborhood of the reference structure differently from features further away.

In [TPD06] the preprocessing time needed is indicated with 20 seconds for a 256^3 volume, but it is easy to imagine heuristics or the computation on the GPU to cut processing time. For simple geometric reference forms like points and spheres, there is no need for pre-computation at all. The needed distance volume during rendering needs not to be the same resolution as the dataset, since distances are trilinearly interpolated. It leads only to small inaccuracy in regions near a surface. The need for a segmentation makes this TFs rather unfeasible for time-dependent data.

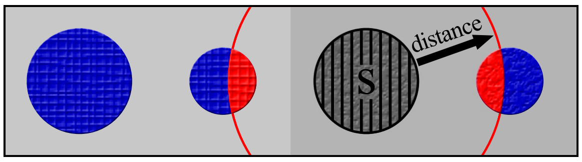


Figure 2.13: *The distance-based TF uses the distance to a given segmented object S, to provide additional information in the near environment of the feature.*

2.3.3 Discussion

The described TF types differ in behavior and abilities: some need technical expertise, and some application-oriented expertise. Some emphasize areas, some surfaces. So it is not a single TF that is perfect for everything. Rather it depends on the boundary condition, such as the dataset, the user, and the available time for a TF to be a perfect candidate for a particular field of application and task. Sometimes a trial and error approach is right, sometimes a more automated technique is needed. All in all, expertise is required to determine the most suitable TF type.

This turns out to be a problem during programming of a volume visualization software, because it is not always possible to predict the boundary conditions of all visualizations in the future. Hence, it is difficult to predict the needed TFs and especially their combinations. An optimal adaptability to new conditions can only be assured if, the assortment of TF types is comprehensive, or if the exact use and combination of TFs is not determined when programming the software, but at the moment the visualization is created. In Section 2.6 a framework will be proposed, that accounts for this problem by leaving the options open until a visualization is created.

One aspect has been touched upon only marginally. Using complex transfer functions, it may be that the illumination is quite complex, too. Features can be made visible whose surface does not correspond to the data's gradient direction, which is typically used to calculate the illumination. This can also be true for the common gradient based TF when regions with a very small gradient length (i.e. homogeneous regions) are highlighted. The next section will describe this problem in more detail. A general solution to this problem regarding any transfer function will be presented in Section 3.2.

2.4 Illumination

Shadows and shading do not only support a much more realistic appearance, but they are an essential cue for depth and structure. This section is necessary to understand the argumentation of the following sections. Hence, the content is reduced to the necessary information. A detailed insight into the state of the art concerning interactive volumetric illumination in volume rendering can be found in the survey of Jönsson et al. [JSYR12].

In reality, lighting is "only" an emission, propagation, exchange and absorption of electromagnetic radiation with a specific wavelength, which, in a very small portion, hits the eye or the camera respectively. Due to algorithmic reasons, computer graphics distinguishes between shadows and shading. For the computation of *shadows*, it is always necessary to take into account objects in the global environment that can occlude sources of light. As a result the casting of shadows is computationally expensive to compute. Prominent representatives of algorithms that can compute shadows are Ray Tracing [Whi80, KG79] and Radiosity [GTGB84]. But there are many more algorithms that

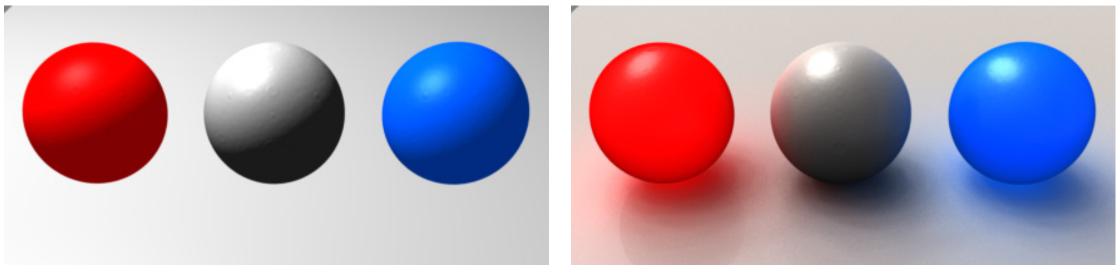


Figure 2.14: *Phong shading (left) compared to ray traced shadows and reflections (right) in volume rendering. The adjacency of the bottom and the spheres is not obvious in the Phong shaded scene. The red and the blue spheres are emissive.*

mainly diverge in the kind of approximation and, thus, in their execution time and quality. In contrast, for the computation of *shading*, only the relative position of the lit surface, the light source, and the camera is taken into account. That is, only local properties that are easy to compute are used. Prominent representatives of shading algorithms are the models by Lambert [Lam60], Phong [Pho75] and Torrance [CT82]. Besides shading and shadows you can distinguish reflection, refraction and emission. Even more complex properties like color dispersion are of little relevance in volume rendering. Figure 2.14 shows a comparison between Phong shading and ray traced shadows and reflection in volume rendering.

2.4.1 Gradient

The angle between lit surface and incoming light is the most essential parameter for many calculations of lighting – e.g. shading, reflection, refraction. As stated in Section 2.1 the gradient is used as approximation of the surface normal in volume rendering. Hence, the precision of the calculated gradient is the crucial parameter for the precision of most lighting calculations. As a result, most volume rendering engines compute an additional gradient volume in preprocessing. During sampling, this gradient volume is interpolated at positions between the grid. By this means, simple sampling patterns for computation of the gradient can be replaced by more complex sampling patterns, without influencing the rendering performance. Figure 2.15 shows the both most used sampling patterns.

Transfer Functions vs. Gradient The gradient is typically calculated on the original volume data. If this gradient shall be used to shade a volumetric scene, it requires that the volumetric scene has exactly the same surfaces as the original volume dataset. However, transfer functions are used to define the opacity within a volume dataset. This might lead to different surfaces in the visualized scene compared to the data. This appar-

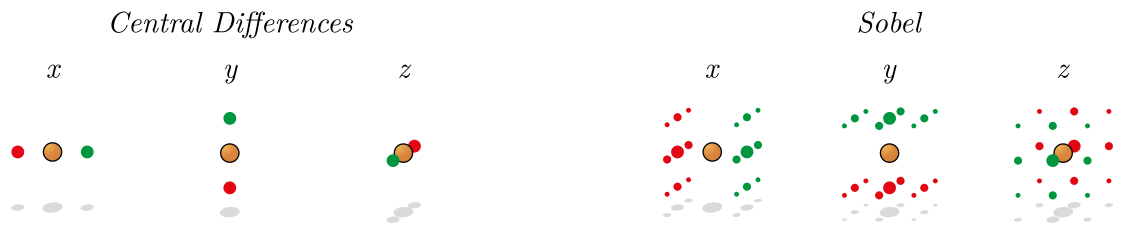


Figure 2.15: *Illustration of common sampling patterns for computation of x, y and z component of the gradient. Orange dots indicate the sampling position the gradient shall be computed for. Red and green dots indicate sample positions, at which the volume is sampled to compute the gradient. The size of the dots indicates their weight. The weighted sum of the values at red positions is subtracted from the weighted sum of the values at green positions to determine the x, y, or z component, respectively. Drop shadows help to understand the pattern.*

ent problem does not constitute in most practical examples. For example, let us imagine a 1D data-based transfer function defining the transparency. Any arbitrary parameterization can only create surfaces at such positions where two neighboring positions do not share the same data value. Hence, at these positions, there is always a gradient that points from one side to the other and is perpendicular to the created surface. Solely, the direction of the computed gradient can be opposite to the real surface's normal. As a result, volume rendering algorithms often use a trick and compute two-sided shading. The same property regarding the surface normal and the gradient is true for the 2D gradient-based transfer function. When using other transfer functions this property is disputable, and correct shading might be more complex.

A general solution to this problem is the computation of the true negative opacity gradient. That means, the gradient is not computed before, but after the mapping to transparency. Because this mapping is part of the user-interaction, this technique rules out precomputed gradients. However, since negative opacity gradients are computed after transfer functions map the data values to color and opacity, the frequency of the opacity signal is in most cases much higher than the frequency of the volume data. As a result, the size of the gradient sampling patterns might need to be adapted to be smaller than the voxel grid.

One special treatment is necessary in homogeneous regions in a dataset. They do not exhibit meaningful gradients and thus no surfaces, which could be illuminated. If they are visualized semi-transparent, i.e. as a gaseous material, shading has to be ignored or computed in a different way.

2.4.2 Perception of Depth and Structure through Illumination

As stated at the beginning of this section, shadows and shading considerably contribute to the perception of depth and structure. When comparing lit and unlit renderings the difference is obvious, but depth perception depends on the chosen shadow algorithm, too. It is extremely elaborate to show these differences in perception, because this is only possible performing large-scale user studies. According to that, only a very few studies that show significant results can be found in literature (despite the fact that this is the only way to prove a visualization to be good). Particularly, a comprehensive study between shadowing algorithms can be found the publication by Lindemann and Ropinski [LR11]. Measurements between shading algorithms can be found in the publications by Domik et al. [DSAS11, DASH11, DATS12]. Figure 2.16 shows three of the tasks that where performed in the user studies. In the first task, participants had to decide which picture in a pair represents rougher patterns. In the second task, participants had to decide which picture in a pair contains the artery that is bent stronger than the other. In the third task, participants had to decide if the marker on a planar coronary image is representing the same location as the marker on either the Phong or style rendered data. The examined style transfer functions [BG07] are a special case regarding shading algorithms. Actually, they are transfer functions that map a gradient vector to color. Nevertheless the mapping function is typically used to store a complex lighting situation. This way complex light-material interaction can be simulated. On top of that, style transfer functions can be easily combined with contouring algorithms, when integrating curvature as a metric. As a result from the above mentioned studies, differences in the perception of depth and structure between the different algorithms can be shown. The use of this for a concrete application purpose, on the other hand, is even more difficult to show.

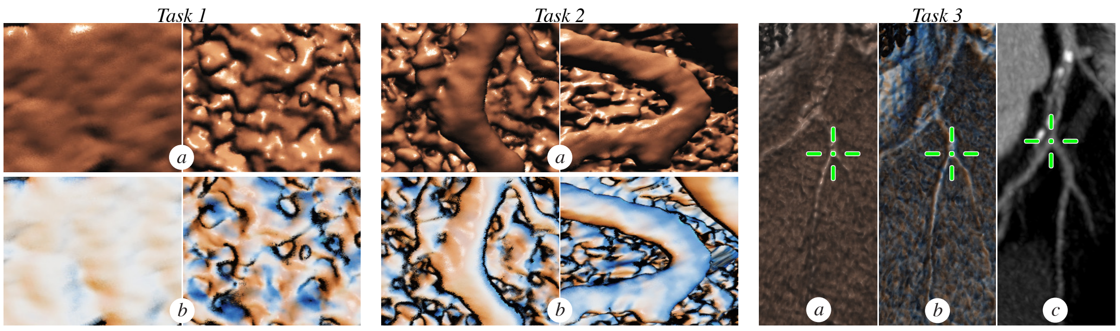


Figure 2.16: *(Task 1, a) Two Phong shaded volumes showing different roughness in image. (Task 1, b) Same pair, but Style shaded. (Task 2, a) Two Phong shaded volumes showing coronaries with a different curvature. (Task 2, b) Same pair, but Style shaded. (Task 3, a-b) Phong and Style rendered volumes, resp. (Task 3, c) Planar reference image. All three images show markers for a specific position.*

2.5 Combinations of Several Volumes

In volume visualizations that use several volumes (i.e. *multi*-volume rendering) a decision has to be made how these volumes are displayed together. So called intermixing strategies propose approaches to that displaying. Cai and Sakas [CS99] list the most used strategies. An intuitive strategy is to determine a visual appearance for each volume and to display both volumes using a blending of samples sorted by their distance to the camera. Two more strategies are listed in this publication. However, in Arens et al. [ABD11, AD11] it is pointed out that there are other possibilities to blend volumes. Especially if the volumes are overlapping modalities which measure different properties of the same object, it can be meaningful to use all the properties in combination to define visual attributes. This will be called logical combination in this thesis.

2.5.1 Logical Combinations: An Example

A transfer function can be used to hide unimportant structures by transparency and visually highlight important structures. A prerequisite for this is, that the different structures can be differentiated by the data values. At a first glance, this is not the case comparing the Hounsfield values of coronary arteries and heart's cavities. Even when using any other transfer function, presented in Section 2.3, it is not possible to display the coronary arteries individually. This example will be examined in more detail in this subsection because it clearly demonstrates that logical combinations of transfer functions can be necessary to visualize the contained information of a dataset. The example and the content of this subsection originate to a great extent from the publication of Arens et al. [ADW*10].

The visualization of coronary arteries is a challenging task for algorithms, radiologists and CT scanners. Many factors make the detection difficult. They are small, permanently in motion due to heart beats and respiration, they are only visible using contrast agents which can distribute unevenly, and they are in the direct neighborhood to other vessels filled with blood, like the heart's cavities and the lung vessels. To differentiate them anyway, a three-dimensional feature space is used. It is defined by the metrics data value, size and variance. Using only one or two of the metrics, it is not possible to display the coronary arteries individually. The coronary arteries share the same Hounsfield values with the arteries of the lung and the cavities. The size is the same as the size of the lung vessels. The variance in turn is in the same range as the variance of the neighboring cavities. However, when combining the three metrics, the lungs can be ruled out by variance, the cavities can be ruled out by size and the fat and muscles can be ruled out by the data value. The Venn diagram in Figure 2.17 depicts these facts clearly.

The values' ranges of the named structures are not clearly distinguished in the single

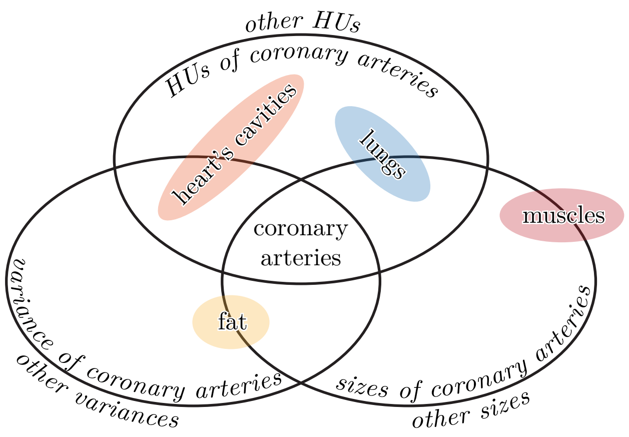


Figure 2.17: *Venn-diagram showing the overlapping data ranges of tissues.*

domains of the metrics. They are rather fuzzy and overlapping. Hence, it is important to set the ranges of what the user sees manually in an interactive way. This way the user can explore the dataset and decide what he wants to see. Figure 2.19 shows different user defined ranges in the domain of data values. When only parameterizing the transfer function based on the data value, the user can “select” the contrasted blood volume and the tissues overlapping with this Hounsfield range. Figure 2.18 (a) shows a result, based on this data-based transfer function. When adding the variance as additional metric, the lungs' arteries can be faded out when mapping a high variance to high transparency as can be seen in Figure 2.18 (b). When trying to highlight small features by using a size-based metric, every small feature will be highlighted. This is especially problematic regarding the lungs, which can be observed in Figure 2.18 (c). Big regions, like the cavities, on the other hand, are not highlighted. When combining all three transfer functions the visualization in Figure 2.18 (d) is possible. Small aberrations are not avoidable. In Figure 2.19 (f) the transfer function results (a) – (e) can be compared to a perfect segmentation.

This example demonstrates that logical combinations can be necessary to express all of the contained information of volume data. In this example one measured modality has been combined with two metrics that were computed on this modality, but of course, logical combinations can also be used to visualize the information contained in several measured modalities. To write new programs each time a new data situation occurs is not practical. The next section will propose a framework solution, that does not necessitate any programming to solve this problem.

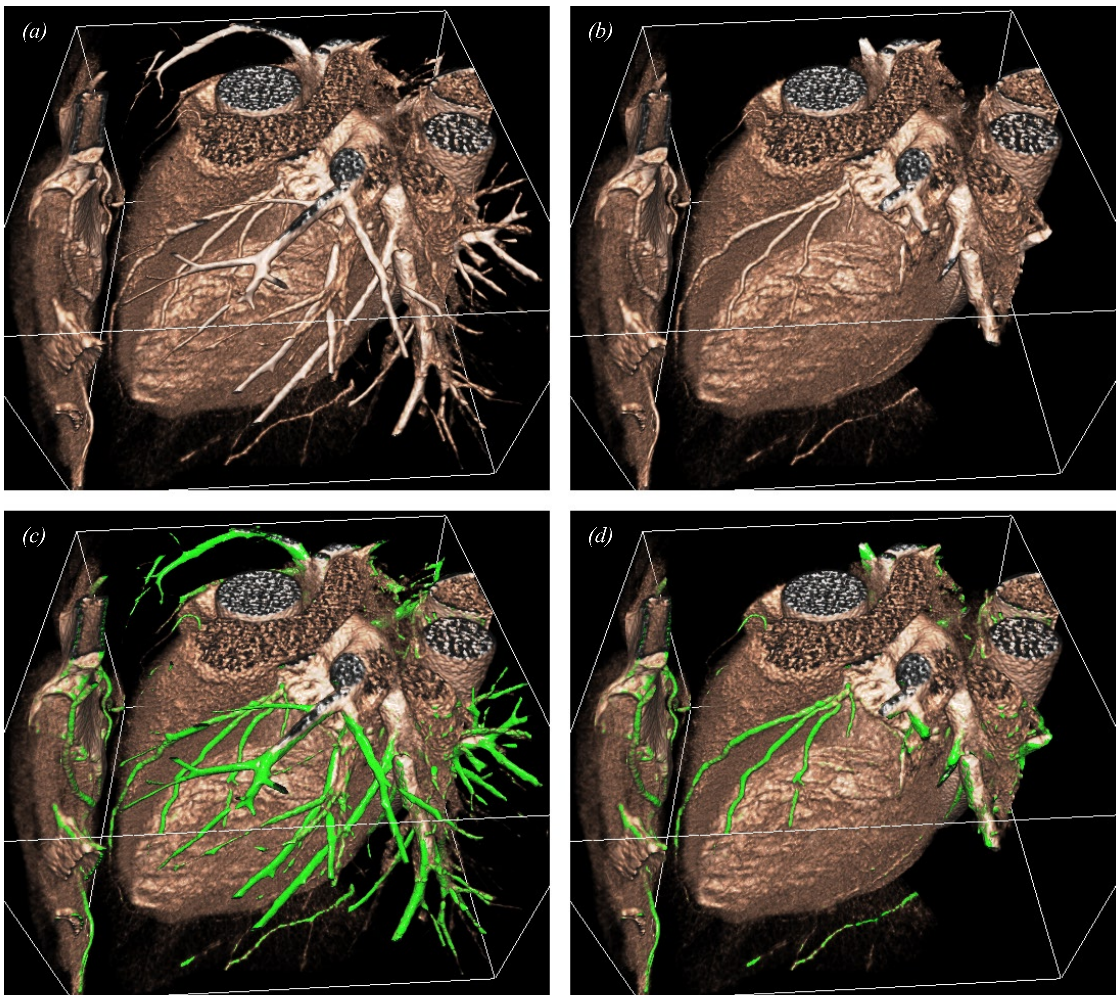


Figure 2.18: *(a) Visualization using only a 1D data-based transfer function. The blood volume and minor parts of the muscles and bones are visible. (b) Same as (a) but enhanced by a variance-based transfer function to hide the pulmonary vessels. (c) Same as (a) but enhanced by a size-based transfer function specialized to tubular structures to emphasize small vessels. (d) Combination from (b) and (c).*

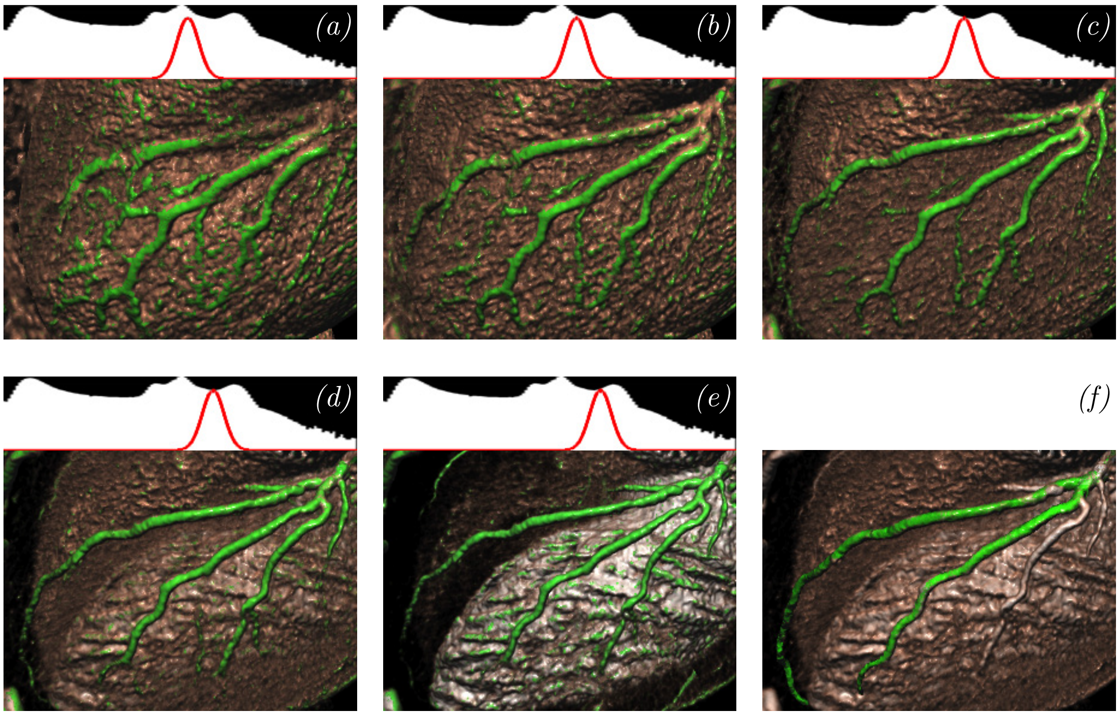


Figure 2.19: *(a) – (e) Left coronary artery emphasized by the same logical combination of transfer functions as in Figure 2.18 but using different specifications of the data-based transfer function (red lines overlaid to the histogram) to select the visible parts of the coronaries. (f) Left coronary artery emphasized by a perfect segmentation.*

2.6 Creating Volume Visualizations flexibly

Section 2.5 has shown that the different types of tissue in the human heart demand different types of logical combinations to be highlighted when using CT-angiographic datasets. Due to the multitude of possibilities and tissues, it is not practical to write new programs each time the suitability of a new combination is tested. This section presents a system, that makes it possible to try out logical combinations of transfer functions during runtime by visual rapid prototyping. This way the expenditure of time to create a suitable visualization is decreased considerably. The first version of the system has been published in “Seamless Integration of Multimodal Shader Compositing into a Flexible Ray Casting Pipeline” by Arens et al. [ABD11]. The following section is mainly based on this paper, some details are used from [AD11] and [ABD13b].

2.6.1 Introduction to Direct Multi-Volume Rendering

Multimodal data acquisition appears to be the most promising method for diagnosis by means of volumetric datasets in future. However, visualization of multimodal data is still quite complicated. Especially when modalities exhibit diverging resolutions, extents, or orientations (e.g. they were taken from different scanners), their combined visualization is complex. While alternative visualization methods like glyphs (e.g. [OHG*08]), or “linked feature display”, are already well investigated, up until a couple of years ago, it was not technically possible to visualize multimodal data via direct volume rendering due to the lack of computation power. Instead, compositing could only be applied in preprocessing, utilizing resampling of the volumes to a common multimodal grid, which introduces a lot of sampling errors and is time-costly. Therefore flexible experimentation with multimodal compositing was not imaginable. In the last few years there has been a great deal of work on GPU-based rendering methods for multiple overlapping volumes. However, it is not only important to display volumes, but also to combine their provided information in order to attain a more expressive visualization. Most of the proposed systems are capable of displaying several volumes simultaneously, but not to combine their information. In this section, a flexible, but integrated data flow is presented for exploring multimodal compositing schemes in direct volume rendering. Using this, the data flow can be easily constructed and retraced from preprocessing to the final composed color. This approach is aimed at engineers that experiment with new modalities and visualization methods. An intuitive and simple interface is provided, as volumes, computed metrics (e.g. curvature or size), and segmentations are treated uniformly within the dataflow graph. On the other hand, this approach supports a deep understanding of the created visualizations, which is necessary to improve them. Hence, new logical combinations of modalities, metrics, transfer functions, and segmentations can be created in an efficient way but without programming skills.

2.6.2 Problem

In the example in Figure 2.20, three congruent modalities A, B and C are given, each showing different information of one region. Let us assume a medical context in which those regions of tissue that are harmful are highlighted only in A and B but not in C. That means surrounding tissue that is not harmful exhibits the same data values if considering only one modality. Hence, a combination of these modalities is necessary to determine the harmful tissue. To avoid inaccuracies this combination has to be done on a per sample basis, which, in GPU-based ray casting, is only accessible in shader programs on the GPU. It is possible, of course, to write a shader program that combines the modalities in volume rendering. However, to demand from the user the ability to program shaders is overdone, compared to the rather simple task of combining modalities. Hence, it is desirable to specify these logical combinations without programming skills and without being confused by technical GPU-issues. On top of it all, the immanent degree of freedom should not be constrained.

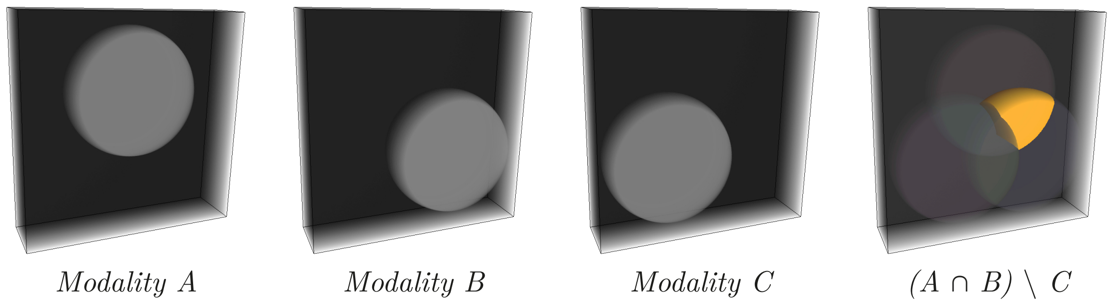


Figure 2.20: *Artificial example of a logical combination. The yellow region necessitates all three modalities to be defined.*

2.6.3 Previous and Related Work

Most of the recent papers dealing with multimodal volume rendering focus on rendering multiple arbitrary aligned volumes correctly in real-time, including all overlapping parts. There are essentially two strategies for solving this problem for GPU-based approaches: depth peeling [RBE08b, RBE08a, PHF07, BBPtHR08] and binary space partitioning [LLHY09, LF09]. Based on these techniques, the information of these different volumes shall be combined logically. This can be done by programming new shaders.

To shorten the development time, there have been proposed several techniques to specify the behavior of shader programs like SuperShaders [McG05], micro shaders [Har04], or a combination of that including stages like transformation and texturing [TD07]. All of them combine multiple shading techniques into one shader but lack flexibility and precise control compared to programming. Several volume rendering frameworks adopt these approaches [RPLH11, RBE08b, RBE08a, MSRMH09] to modify the behavior of the shader program.

In [CS99], three intermixing strategies have been formulated (Illumination Level Intermixing, Accumulation Level Intermixing, and Image Level Intermixing) that were implemented in most of the stated publications. But these strategies are formulated in such a way, that a single volume is always given a visual appearance *unaffected* by the other volumes. The intermixing strategy only enables the joint display of the volumes with their individual appearance. This is because only *chains* of processing stages are defined, whose color values are joined by the selected intermixing scheme. *Graphs* are required in order to make use of the full information given at a sample's location in multimodal data. This enables the computation of a sample's visual appearance *affected* by all volumes.

To sum it up, the functionality of graph-based data processing and the flexibility of programming shaders would be desirable. Finally, in case of a modifiable and, thus, unsteady program behavior, a visual representation of the behavior would help to understand the visualization. One possible approach to fulfill these requirements is visual programming using dataflow graphs. The major goals of visual shader programming are to hide the low-level complexity of a modern shader programming language from the user while preserving flexibility and to reuse programmed features. This is done by replacing the textual representation of a shader programming language with a dataflow graph, and a visual editor that allows the user to combine nodes representing shading building blocks to complete shader programs. The complexity of a single node can vary from a single arithmetic operation to complete shading techniques, such as Phong shading. Several publications deal with visual shader programming. The first approach with a graphical user interface were the Building Block Shaders by Abram and Whitted [AW90] which implemented the Shade Trees by Cook [Coo84]. Efficient versions have been proposed that are transformed into GPU-shaders [GD06, MSPK06], but their abstraction level is too low for use in complex visualization systems.

In [PHF07], shader composing by visual programming using dataflow graphs is proposed for multimodal volume rendering. Compared to the dataflow graph presented in this section, the system in [PHF07] lacks the direct visual connection between incoming data of the preprocessing and the shader graph, leading to a data flow that is difficult to retrace. Additionally, their system is using texture slicing whereas the system presented here uses GPU ray casting (see Table 2.3).

2.6.4 Visual Programming using a Dataflow Graph

Many volume rendering systems try to simplify the user interface as much as possible, regardless of technical details and properties. Such a concept is only adequate if the

	[RBE08b]	[PHF07]	[BBPtHR08]	[LLHY09]	[RPLH11]	[MSRMH09]	[ABD11]
Arbitrary oriented volumes	+	+	+	+	+	+	+
GPU-based Rendering	+	+	+	+	+	+	+
Rapid Prototyping of Shaders	+	+	−	−	+	+	+
Dataflow Graph for Shaders	−	+	−	−	−	−	+
Ray Casting / Texture Slicing	RC/TS	TS	RC	RC/TS	RC/TS	RC	RC

Table 2.3: *Differentiation between Volume Studio [ABD11] and other publications of GPU-based multi-volume rendering systems.*

behavior of the software is always the same. In case of a modifiable and thus unsteady pipeline, it is necessary to understand the behavior of the pipeline e.g. by a visual representation of the processing. Visual programming using dataflow graphs is very common in image processing and visualization. It provides a flexibility that is similar to programming and it is less demanding than coding from scratch. Building blocks can easily be reused and the visual representation eases the understanding. But the degree of freedom when constructing a GPU-program is bounded, due to the fact that it needs to be technically realizable (i.e. global functions operating on volumes in the preprocessing and local functions operating on a per sample basis that need to be performed on the graphics hardware, dictate a rough framework to the structure of the dataflow graph).

To not constrain the immanent degree of freedom combining modalities, a graph-based visual programming of the data flow is proposed in this section. In contrast to [PHF07], we decided to construct it graphically in one single dataflow graph, despite the named technical issues. This eases the control of the data flow from loading to final coloring. It consists of nodes with input and output ports, nodes containing nodes and connections that define the data flow between compatible ports. To make clear and enforce the technically given order of preprocessing functions and local functions per sample, these groups of nodes are visually *nested*. Figure 2.21 shows a simple dataflow graph. Outside the rendering node is the data flow of the preprocessing stages. Inside the rendering node is a subgraph that produces a color and an opacity for each sampling position. This transition is thereby visible and enforced but unobtrusive concerning the interaction with the user interface.

The shading nodes of the subgraph provide GLSL snippets that are composed to a shader program which is used for the GPU-based rendering. The connection between the outer and inner data flow is specified clearly by connection ports. Small icons at the ports and more detailed tooltips reveal the signature of a port, and hence, which nodes can be connected. See Figure 2.22 for the icons of the most used port signatures. All nodes and connections are inserted via drag&drop into the dataflow graph, whereby connections between incompatible ports and circles within the graph are automatically suppressed.

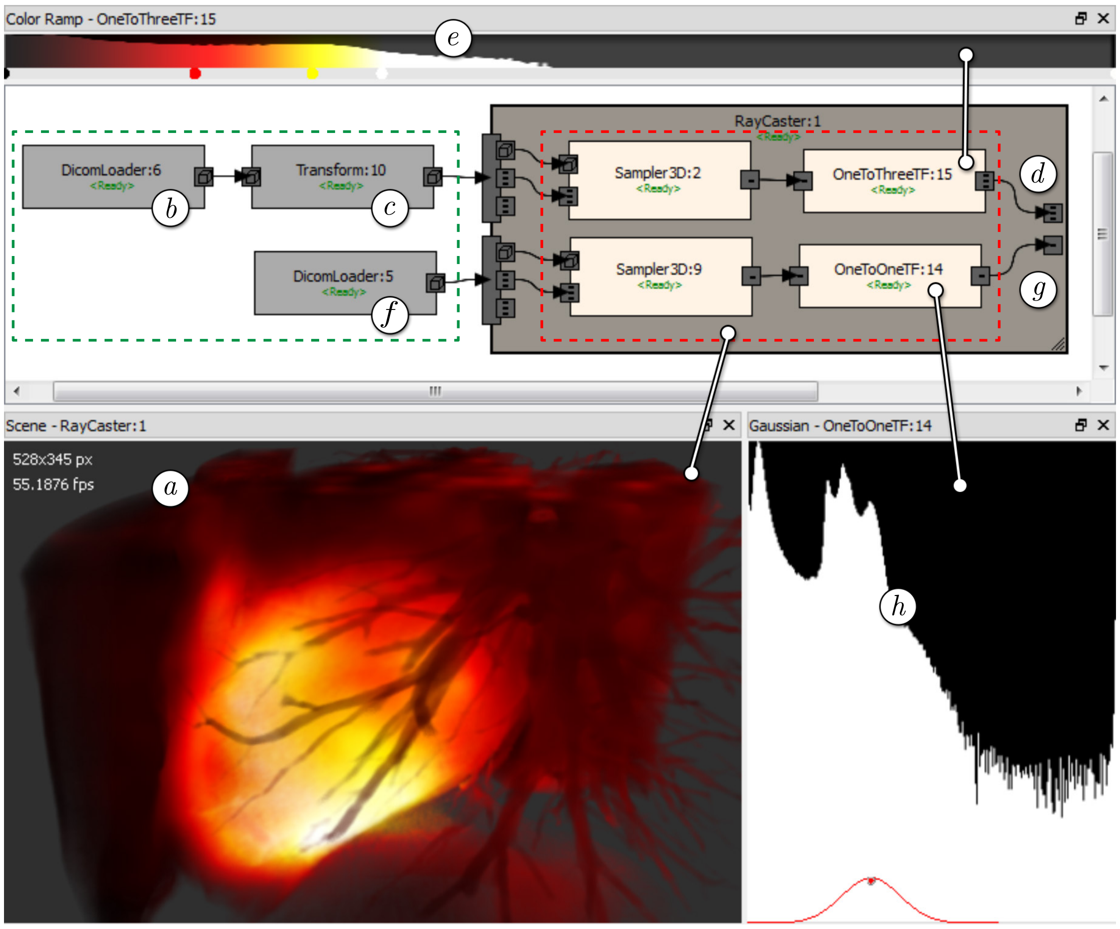


Figure 2.21: *Simple example of a multimodal visualization in our graph-based system showing a heart using the ray casting renderer (a). Data flows from left to right. After loading the PET volume (b) it is co-registered to the CT volume using a transformation node (c). The color output for each sample (d) is determined using this PET volume and a color ramp transfer function (e). The CT volume (f) on the other hand serves as the basis for the α value (g) of each sample, defined by a 1D TF (h). The three superimposed connection lines indicate which node provides which widget. Green dashed line indicates nodes in preprocessing, red dashed line indicates shading nodes.*

The dataflow concept fits to all renderers that provide sampling positions. In most GPU-based rendering algorithms sampling positions are either produced by rendering geometry that defines the entry and exit positions of rays followed by a ray-traversal, or by rendering slices through the volume. We decided to not pass rendering technique details like entry/exit texture as parameters through the dataflow graph since this part of the rendering pipeline is almost always the same. Hence, renderers must provide their

Float
3-Component-Vector
4-Component-Vector
Volume

Figure 2.22: *Icons of the most common port signatures.*

own sampling position computation. But that clearly simplifies the specification of the dataflow graph, since loops and more complex hardware-dependent operations are not necessary. It brakes down the visualization task to specify a color and an opacity for each sampling position.

Nodes Although being technically severely different, the user handling of the preprocessing nodes and the shading nodes are absolutely identical. This way the user is not confronted with the difference of shaders, CPU programs, or GPGPU programs. As a second advantage, the whole data flow can be visualized in a single graph, leading to a much easier creation and specification of the dataflow. Nodes that contain inner dataflow graphs can be displayed "expanded" and "collapsed" to improve the overview.

The result of many nodes does not only depend on the input values, but on additional *properties*. These properties are displayed on a property pane that comes with every node. See Figures 2.24 – 2.26 for an exemplary shader node that uses properties. To develop new nodes more rapidly, we programmed a generator that produces dummy nodes (preprocessing nodes as well as shader nodes) with ports and properties that are automatically arranged on the property pane and saved, when the visualization is being saved. This way the programming of new nodes is reduced to a minimum (e.g. for most shading nodes only the body of one GLSL function has to be programmed).

Preprocessing Graph In the preprocessing part of the dataflow graph data is loaded and processed by algorithms that need to run only once, or at least not every frame, and do not affect the shader program. The processed data are basically volumes, meshes, and points that are passed through by the connections. Volumes are passed through voxel by voxel instead of moving whole buffers. This way memory consumption can be reduced to a minimum because many functions can be implemented inline without caching (e.g. a node that crops the incoming volume is implemented by just modifying the requested index). Ports provide metadata to hold resolution, voxel spacing, transformation etc.. In contrast to software systems that use image level intermixing, in this stage there is no data flow containing images or other rendering data.

Some of the preprocessing nodes make use of OpenCL to reduce execution time. Additionally, a generic OpenCL node processes OpenCL programs that contain simple convolutions. Other nodes compute more complex metrics such as median, standard deviation, size [CM08], vesselness according to [PBB05] and [FNVV98], curvature [KWTM03], and distance to a segmentation to use them as additional modality.

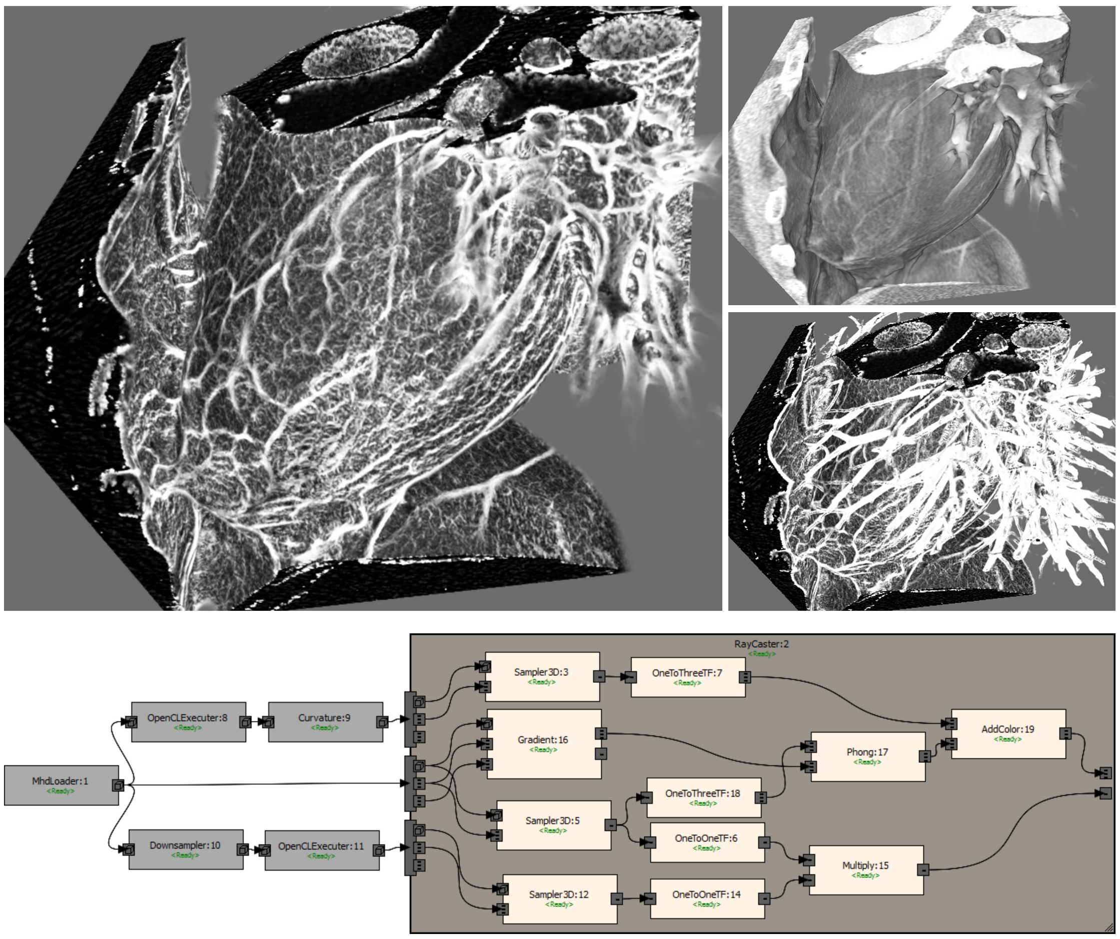
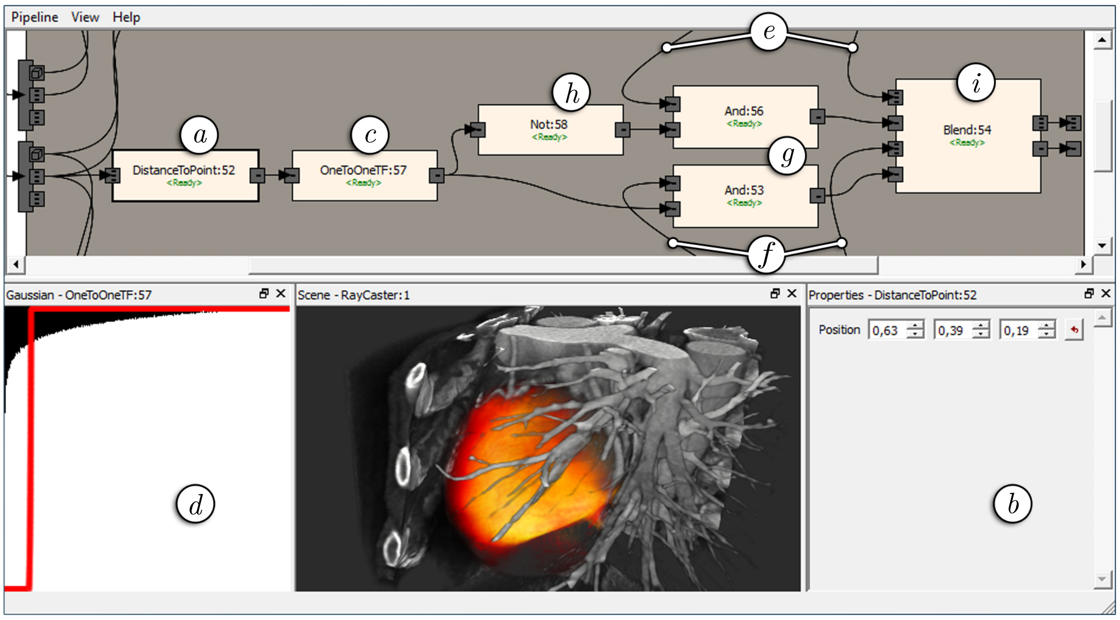


Figure 2.23: *Example of how different metrics computed in preprocessing can serve special visualizations. Shown are details on the pericardium in a CT-angiographic dataset enhanced by curvature and Gaussian metrics. The pictures on the right show results without the use of curvature (upper) and without suppressed lungs (lower). Bottom: Dataflow graph of the upper left visualization with five preprocessing nodes on the left and a rendering node containing a shading graph on the right. In the upper sub-graph the CT data is preprocessed by a Gaussian filter followed by a computation of the curvature κ_1 to emphasize small structures. The lower sub-graph is used to suppress the lung vessels by eroding them with a widespread gaussian filter. The ray casting node contains 3 samplers, 4 transfer functions, a gradient node, a phong node, a multiply- and an add-node to combine the metrics usefully. The whole data flow from preprocessing to final shading can be retraced in one graph.*

Figure 2.23 shows how metrics computed in the preprocessing can emphasize small structures while simultaneously suppressing the lung vessels (that share the same value and curvature). With data flow *chains* only, these visualizations would not be possible. The two Gaussian filters make use of the stated generic OpenCL node.

Shading Graph The shading graph is a sub-graph of the rendering node. It contains shading nodes that provide GLSL shader functions. Due to the varying nodes, the varying number of volumes and meshes used, the shader is generated during runtime by the *shader composer*. This way costly conditional branches in the shader are avoided. Input and output ports are directly made available as GLSL variables and properties are automatically bound to uniforms. To avoid naming collisions between multiple instances of the same shading node the variables are given a unique name internally.



Position 0,63 0,39 0,19

Figure 2.24: *The upper part of the figure shows a portion of an inner shading graph. The node "DistanceToPoint" (a) computes the sample's distance to a point specified in the properties (b). Based on this metric the transfer function (c) defines a function (d) that distinguishes near and far samples. In the not shown part of the dataflow graph two appearances (i.e. color and opacity per sample) are defined, one for the near region (e) and one for the far region (f). The "And" nodes (g) in combination with the "Not" node (h) use the mapping of the "distance transfer function" to set the alpha value of one appearance to zero before blending (i). See Figure 2.25 and 2.26 for C++ and GLSL shader code used for the "DistanceToPoint" node.*

```
/** This shading element computes the sampling position's distance to the specified point. */
DistanceToPoint::DistanceToPoint() :
    ShaderElement("DistanceToPoint"),
    _samplingPosInputPort(new Port(this, "vec3", "samplingPos", Port::Direction_Input)), // create input port
    _distanceOutputPort(new Port(this, "float", "distance", Port::Direction_Output)),    // create output port
    _pointProperty(new Vector3Property("point", "Point",                                 // create property, id, label
                                       "Point to compute distance to", this,             // description
                                       vec3(0.5f), vec3(0.0f), vec3(1.0f), vec3(0.01f))) // default, min, max, step
{
    addPort(_samplingPosInputPort);                                                      // add ports
    addPort(_distanceOutputPort);
    addProperty(_pointProperty);                                                         // add and connect property
    connect(_pointProperty, SIGNAL(valueChanged()), this, SIGNAL(shaderOutputChanged()));
    addShaderUniform("point", "vec3", _pointProperty);                                   // add uniform "point" and
}                                                                                        // bind to _pointProperty
```

Figure 2.25: *C++ code for constructing the "DistanceToPoint" node (Fig. 2.24 (a)) and its properties (Fig. 2.24 (b)). Only a few lines have to be modified when creating new nodes with the node generator. Input and output ports are directly made available as GLSL variables. Properties are bound using addShaderUniform. See Figure 2.26 for the corresponding shader code.*

```
distance = length(point - samplingPos);
```

Figure 2.26: *The complete GLSL shader code that has to be programmed for the "DistanceToPoint" node. The variables samplingPos, distance, and point are bound to their corresponding ports and properties by the shader composer. See Figure 2.25 for the corresponding C++ code.*

The position of the samples depends on the renderer. Therefore the renderer passes the volumes including the current sampling position as well as the texel spacing to the inner shading graph. This *integrated embedding* is also represented by a triple-port that connects data flow between the inner and outer graph. The expected result of each sampling procedure is an RGB-color and an opacity value, which is why these two ports are placed on the right side of each renderer. Within the graph all compatible ports can be connected, this way, e.g. the gradient can serve as color.

For *logical combination* of the modalities, nodes like AND, OR, XOR, NOT, ADD, ADD-Color, Multiply, Multiply-Color, and Blend-RGBA are available. Each of them combines the values at the input ports using the corresponding logical operator. Additional operators can be generated within seconds using the node generator. Additionally two-dimensional transfer functions can be used to map any two floats within the dataflow graph to one value. All in all, for the mapping of values a one-dimensional, a two-dimensional, and a style-based *transfer function* are provided. Each of them can output a three-dimensional vector (typically used as color), a floating point value (typically used as alpha), or both. Using the logical operators an arbitrary-dimensional transfer function can be constructed.

Figures 2.25 and 2.26 show the main C++ and GLSL code that is mandatory for a shader node that computes the sample's distance to a specified point in the properties. Figure 2.24 shows how a transfer function and logical operators can be used to merge two appearances dependent on the result of this node. Since the distance is computed in the shader, the sphere with colored appearance can be moved through the volume in realtime by changing the position of the specified point in the properties.

The texture lookup is done within *sampling* nodes in the dataflow graph. This way sampling options, such as nearest neighbor, trilinear interpolation, and gradient, can be chosen independently from the input volume. Additionally, segmentations can be processed by the same nodes as used for other modalities by choosing nearest neighbor sampling.

Inspection using Histograms As in programming, it is often useful to inspect intermediate results when setting up a dataflow graph. Single connections, though, transport millions of data values each frame. Thus, it is not useful to inspect connections on a per sample basis. One simple way is to connect an outgoing port directly to the color-output of the renderer. This way, the characteristic data of a port can be seen at the correct spatial position in the volume rendering as color or gray value. However, histograms are another very useful data inspection tool. In a software, where the input of transfer functions can be specified arbitrarily, the computation of histograms needs to be flexible, too. At each position of the shading graph, 1D and 2D histograms can be computed that accumulate the values up to this stage in the graph. Figure 2.27 shows a simple example of histograms taken at different stages of the graph. This is a particularly useful feature when combining several metrics and modalities by logical operators. Every transfer function makes use of this feature, too. The computation of the histograms is done using the method of [SH07] on the GPU and was extended to 2D histograms. The computation takes less than 100 ms, leading to an almost real-time update when changing a transfer function. To achieve this speed we reduced the number of samples to 5 million, which reduces the quality of the histogram to be hardly noticeable.

2.6.5 Discussion

Visual programming using dataflow graphs is successfully used for data processing in many visualization systems. In this section, this concept has been adopted to the area of GPU-based volume rendering. A shader framework has been presented that allows a construction of arbitrary multi-volume dataflow graphs on the GPU. Volumes in form of original data or modalities, metrics, and segmentations can be arbitrarily combined using logical operators rather than simply displaying them together. This way the advantages of evolving imaging technologies can be combined without programming. The visual

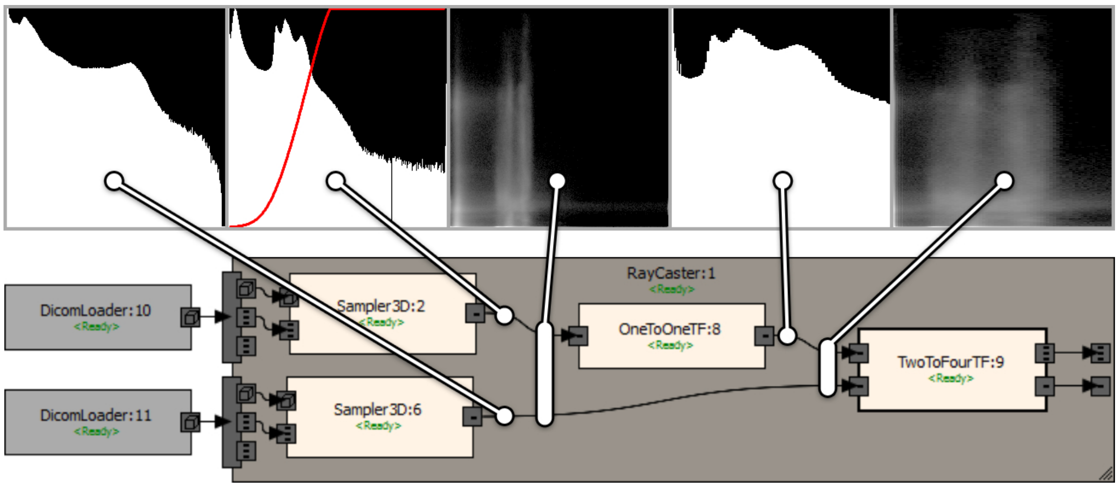


Figure 2.27: *Five histograms taken at several stages of the graph. The first three histograms show the original data in 1D and 2D histograms. The fourth and the fifth show the histograms after manipulation by a 1D transfer function as shown by the red line.*

programming concept integrates the dataflow on the GPU seamlessly into the dataflow graph of the preprocessing stages on the CPU.

Programming using textual languages is a complicated task. A lot of syntax rules, variable scopes, variable types, and loop-structures need to be comprehended by the programmer. Programming GPU-programs is even more complicated due to a number of implicitly defined execution and processing rules, missing output possibilities for debugging, and several hardware-accelerated data-structures that should be favored over others. Compared to that, creating volume visualizations using the dataflow graph is rather simple and it effectively uses the power of the GPU. Only three elementary tasks are demanded from the user: Adding nodes (via menu or drag&drop), connecting nodes (by dragging edges between two ports), and parameterizing nodes (by changing the properties in the property tab or using optional widgets, e.g. transfer function widget). Furthermore, only one structure that is not directly visualized needs to be comprehended by the user, that is the execution per sample in the shader framework. However, it demands from the user the ability to understand how color and opacity can be used and processed. That is, the user should know about transfer functions and what happens if two colors and opacity values are multiplied or added, respectively. The approach is, thus, aimed at engineers that experiment with new modalities and visualization methods.

In the context of this thesis the proposed dataflow-based concept necessitates a redefinition of the term "transfer function". In Section 2.3 a transfer function described a function that was defined by the domain and the codomain of its mapping. Now, the domain and codomain can be freely connected within the dataflow graph. Hence, it is more suitable to define the type of a transfer function by the *data characteristics* of the domain and the codomain. A simple transfer function would be a function that maps one floating point value to another floating point value. Accordingly, in Volume Studio this type of transfer functions has been named “OneToOneTF”, implicitly assuming floats. The 2D gradient-based transfer function from Section 2.3 in turn consists of a “TwoToFourTF” whose input ports have been connected to a gradient length (one float) and the data value (one float), and whose output ports have been connected to color (three floats) and transparency (one float).

From a theoretical point of view the proposed dataflow graph can be seen as a breaking down of the visualization task into simpler subtasks. The original task to map about 512^3 voxel values arranged in space to e.g. 1024×768 pixels on the screen, i.e. RGB-values, has been divided into smaller problems. The projection to the image plane is done by the chosen rendering node (e.g. slice view, curved planar reformation, or ray caster). The user is confronted with the much easier task to map one or more data values at every position in space to color and opacity. The origin and the processing of the data can be easily retraced in the dataflow graph.

In Chapter 3 this concept will be enhanced once again, which grants the user even more control regarding the visualization process.

Chapter 3

Illumination in Deformed Multi-Volume Rendering

In hand-drawn illustrations, cuts, deformations, and illumination are an important technique to uncover hidden parts, preserve context, and to make the structure and shape perceivable. Many algorithmic approaches have been presented for illumination and several for deformations. Nevertheless, combining both is still a challenge. This chapter describes, in Section 3.2, how the previously presented shader framework can be enhanced to make both techniques accessible without difficulty. The main focus was the flexible usage, combinability, and reusability. However, in many situations the proposed technique comes along with a considerable advantage in performance compared to a naive approach. Nevertheless, the presented technique makes it easy to combine several very demanding techniques, which in turn can result in long rendering durations. As a result it is necessary to think about a new rendering paradigm which complies with both, the high demands in terms of quality in medical renderings, as well as, the demands in terms of interactivity while using such a framework. In Section 3.3 a progressive rendering is presented which is completely orthogonal to the techniques used in the framework, and thus, fulfills the demands while not restricting the flexibility, which is the problem of most other techniques. At the end of the chapter, a special visualization technique will be presented, the curved planar reformation. By a reinterpretation of the original algorithm as a deformation, it has been possible to realize this technique within the proposed shader framework. As a result all techniques that are available in the framework, e.g. illumination, are usable in the curved planar reformation, too. All in all, this chapter is mainly based on the publications by Arens et al. [ABD13b, AIB*11] and the technical report by Gmyr, Arens and Domik [GAD14]. It will rather present the technical realizations and possibilities, than giving concrete examples. Those can be found in the next chapters.

3.1 Volume Deformation

As mentioned, cuts and deformations are an important visualization technique in hand-drawn illustrations. The artists are typically anatomists that depict the organs as they look like during a section of the organ. Accordingly, many cuts can be found in such illustrations. In volume rendering these cuts are most often realized by segmentations or volumetric deformations. Whereas real organs are subject to physical laws, which are also visible in the drawings, the virtual world is not restricted by these laws. The most prominent advantage of this is that the tissue will not be destroyed after it has been virtually cut. Hence, many different sections can be done with the same organ, one after another and – afterwards – be undone. From a technical point of view, cuts, rigid, and elastic deformations can be clearly differentiated. The following three subsections will explain these categories in more detail.

3.1.1 Cuts

In most cases, the user of a volume rendering engine is not interested in seeing the whole volume. Instead, he wants to see only sections of the volume. To reveal these parts, cuts are used to define what is visible and what should be faded away. The complexity of cuts varies considerably. The most simple cut is the planar cut. It is easy to determine for a sample's position, if it is inside or outside of the area that shall be faded away. That is why this decision can be made rapidly within the shader program. Preprocessing is not necessary. As a result, the parameters of the plane equation – and thus, the position and orientation of the cut – can be modified during runtime. Its big disadvantage is that it does not adapt to the contours of tissue. More complex cuts, like spheres or surfaces of a higher polynomial degree, can be used. At some degree of complexity, it is not efficient anymore to decide the visibility within the shader. Instead, the visibility can be discretized in preprocessing and saved in a rasterized volume. Then, the computation in the shader will be replaced by a texture-lookup to this volume. The disadvantage of this technique is of course that a change to the cutting function necessitates a renewed and time-consuming preprocessing. The discretized cutting function can be saved either within nominal data values – essentially, that is a segmentation – or in quantitative values. The nominal variant allows for saving several cuts in one volume very efficiently. However, linear interpolation of nominal data yields not the desired result and, thus, cuts suffer from staircase artifacts. Quantitative values, on the other hand, need much more memory but result in a smoother cut due to linear interpolation. Additionally, when using quantitative values, the cut can be saved with a specific thickness which can be used to interactively relocate the cut in a small degree of freedom. This process compares to the mentioned distance-based transfer functions. Weiskopf et al. analyze the quality of the different cutting types [WEE02, WEE03].

3.1.2 Rigid Deformation

When using cuts, it is sometimes more useful to put cut away data aside, instead of fading it out. This way, context is preserved, which is why there are many examples in anatomy atlases. From a mathematic point of view, this deformation is characterized by different rigid transformations that are applied to the different pieces. Exploded views are a typical application of this technique [BG06]. Rigid transformations, in this case, consist of rotations and translations, which can be easily realized by a single 4×4 matrix multiplication. From a technical point of view, these transformations can be realized by using multi-volume rendering [RBE08b, RBE08a, BG06] (i.e. a dataset is rendered several times using different transformations and mutual exclusive cuts). Because the transformations are rigid, they can be applied to the view rays instead of single sampling positions, which is much more efficient.

3.1.3 Elastic Deformation

Much more interesting is the elastic deformation, i.e. bending and distorting the dataset. Real organs are no rigid bodies, and hence, it is in most cases sufficient to bend or fold over the tissue, instead of cutting it away. This way the context is preserved and the organic elasticity becomes visible. In hand-drawn illustrations the preparations are dissected and deformed before drawing. Thus, real physical behavior comes for free in hand-drawn illustrations. In volume rendering, correct physical behavior is difficult to achieve. Strain measures can only be emulated and simulated in a very simplified manner. The result of such a simulation is a function that maps a transformation to different spatial positions. For the application of such a deformation function, two variants are possible:

1. *Deformation in preprocessing:* The first possibility is to deform and resample the volume in preprocessing. The resampled result is a regular grid and contains the applied deformation. Hence, it can be rendered without additional technical effort. Although easy to implement, the deformation in preprocessing has considerable disadvantages. First of all, it suffers from aliasing artifacts, if the resolution is not massively increased. On top of that, a change to the deformation parameter necessitates another resampling and is, thus, not possible in real-time. Finally, there are severe difficulties regarding transfer functions that have to be solved. At the cutting edges and transitions to background, data values can not be blended to zero. When linearly interpolating during rendering, the transfer function would be fed with totally wrong data values at these transitions, which results in an incorrect visualization.

2. *Deformation during rendering:* The alternative solution to this is a deformation during rendering. The data volume stays original and unaltered, which is advantageous.

The disadvantage is, it is much more complicated to realize. For the transformation is not equal for every sampling position as in rigid deformations, the elastic deformation can not be realized by a transformation to the entry and exit positions of a viewing ray as in multi-volume rendering. Instead of that, for every sampling position an inverse deformation has to be determined, that allows for a lookup at the correct position in the original volume. Similar to the situation regarding cuts, the interactivity depends on the complexity of the function. If there are quite simple formulas for the deformation function which can be evaluated rapidly for every sampling position in the shader, parameters of the deformation – e.g. degree of bending – can be modified during rendering. If the deformation function is too complex, it should be evaluated in preprocessing and discretized in another volume. The inverse deformation, valid at a specific sampling position, can then be determined using a texture lookup during rendering. A change to any parameter necessitates a renewed preprocessing.

Many methods exist to realize deformations in volume visualizations. Chen et al. [CCI*07] categorize them quite well. Methods that do not resample the data, but use inverse deformations include ray deflectors [KY97], free-form lattices [CHM03], spatial transfer functions [CSW*03], and 3D displacement maps [CSC06]. In the more recent approach by Correa et al. [CSC10], regions can constrain the deformation to make it appear more physically. All GPU-based approaches use elaborate and complex methods to get the right shading at cuts and deformed tissue.

3.2 Demand-Driven Dataflow

A main challenge for illustrations, as in anatomy atlases, is the flexible real-time combination of various (geometric) deformations, and shading and shadows (influenced by the deformations) to depict shapes and structures.

The shader framework presented in Chapter 2.6 is designed to combine any data volumes and transfer functions, to determine a color and opacity for each sampling position. When using this framework, it became clear that its strategy to process the dataflow is not flexible enough to handle deformations and complex illumination strategies. This section describes a processing strategy, which makes these features accessible in a dataflow-based shader framework for the first time.

The situation in Figure 3.1 clarifies the inherent complexity of the illumination when the appearance in a multi-modal scene is determined by logical combinations and deformations. The figure shows the resulting structure of a logical combination $C = A \wedge \neg B$ of two volume datasets A and B, A containing a cuboid and B containing a sphere. As explained in Section 2.4, it is indispensable for correct shading in volume rendering to know the gradient vector at each sampling position, which is determined by a set

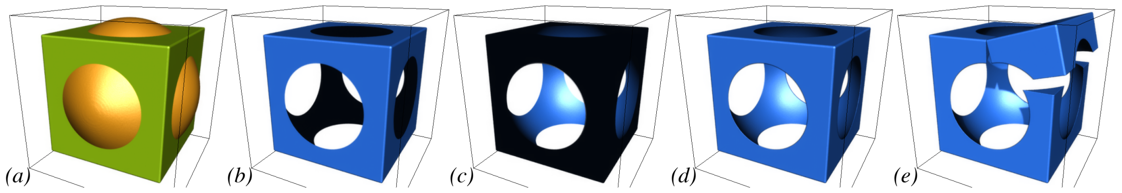


Figure 3.1: *A visualization exemplifying the problem with shading and shadowing in multimodal and deformed volume data. (a) Two volumes, one containing a cube and the other containing a sphere, shown in a multi-volume rendering. (b) - (e) Logical combination of the two volumes, illuminated using the cube's gradient (b), the sphere's negative gradient (c) and the true negative opacity gradient of the logical combination (d). (e) Additional deformation and ray-casted shadows.*

of neighboring data values. However, in the shown scene the gradients at different positions are determined by different datasets. The spherical recess inside the resulting structure should be shaded using the negative gradient of the sphere (dataset B). On the contrary, the flat faces at the outside of the structure should be shaded using the cuboids gradients (dataset A). In spite of that, the edges at the transition between both shapes need a mixture of both gradients. Figure 3.1 (b) and (c) show the wrong shaded result in each case, if only one dataset is used for computing the gradients; inside and outside of the shapes the values are homogeneous, hence the gradient vector has a length of zero. In Figure 3.1 (e), a cut and a rigid deformation is added. For a correct rendering, some gradients have to be added at the cuts, whereas some other gradients of the cube and the sphere have to be rotated. All in all, a lot of special cases have to be differentiated. Additionally, a hard shadow is added to the scene in Figure 3.1 (e). It has to take into account the resulting transparency from the logical combination and the deformation, too. The shadow is computed by shooting a second ray towards the light source, on which transparency is sampled in specific distances. Interestingly, the computation of the gradient for shading, the computation of the shadow, as well as, the computation of the deformation, can be solved by the same processing strategy in the shader. The following section describes this strategy and how to implement it into the proposed graph-based shader framework.

The approach has been published by Arens et al. in [ABD13b]. Some additional technical details are described in the technical report [ABD13a]. This section is mainly based on these two publications.

3.2.1 Introduction

As described in section 2.6, a mapping from the data values at a particular position to visual attributes, like color and opacity, has to be specified. This typically includes transfer functions and logical combinations which have to be realized within a shader in GPU-based rendering (pre-classification is no option in many cases). On top of that, various visualization techniques, like shading or cut and peel, can be applied, e.g. to enhance depth perception or insight into the data.

A fundamental problem of common volume visualization systems is the time that it takes the user to create appropriate visualizations. To allow for experimentation with various visualization techniques, it is necessary that the aforementioned mapping to visual attributes is as flexible as possible. The most flexible but most time consuming way is to write new (shader-)program code for each visualization. Hence, most volume rendering engines are restricted to a small set of selected and predefined mappings from multimodal data to visual attributes. Only few systems provide the user with a freely constructible shader during runtime that can be specified by a dataflow graph and reusable building blocks as described in the previous chapter. This is most likely due to the fact that in dataflow graphs many visualization techniques can be realized, but compared with free programming, there still exist some restrictions. Especially shading techniques and deformations make all dataflow graphs that can be found in literature reach their limitations. This, in turn, results in much special case handling, which finally leads to a lot of programming overhead or bad usability. The reasonable dataflow concept eventually turns out to be counterproductive. In the following subsection, it will be pointed out that the reason for the inflexibility is the use of data-driven evaluation in such systems. Hence, a demand-driven evaluation is proposed that is customized to shader-based volume rendering. It makes it easy to combine deformations, shading and shadowing for visualization of multimodal volume data on the GPU while providing all advantages of a dataflow graph. The approach does not introduce any measurable overhead.

Initially the problem will be discussed in more detail and it will be shown how related works handle that problem. This is followed by the main concepts of dataflow evaluation models. Based on these concepts ,the solution customized to shaders is proposed. To consider reproducibility, an easy way to implement this concept is presented and details of the shader composer are shown, which builds the complete shader program from the shader function bodies of the nodes and their inbound connections. Afterwards, some design decisions are explained and a small selection of resulting possibilities, plus a performance, and a benefit evaluation is given.

3.2.2 Problem

In multi-field and multimodal volume visualizations it is not only important to display several fields and modalities, but also to combine the contained information in order to attain a more expressive visualization [ABD11]. The difficulty is that the depicted features are not determined by a single data value of one volume, but rather by the logical combination of several volumes, e.g. “visible *IF*: high data value in volume A, *AND* low data value in volume B”. In such situations, as well as in deformed scenes, precomputed information can be outdated when any parameter changes. Thus, it is often not feasible to compute intermediate results (e.g. the gradient or occlusion information) in pre-processing.

The problem tackled in this section is not self-explanatory. Hence, the following three paragraphs explain why not only a new processing strategy is necessary, but also why the user of the software is confronted with this problem. Without loss of generality, a ray casting based rendering is assumed in this section and explaining illustrations are drawn two-dimensionally while being true for three dimensions, too.

Deformation on a per Sample Basis: Several ways exist to imitate deformations in volume visualizations. Due to better results, many approaches in the literature are inverse deformations which will be used here, too. I.e. the data stays untouched, but the sampling of the data is deformed. This circumvents a renewed rasterization of the data which would introduce a considerable loss of quality and aliasing problems, especially in multi-volume rendering when volumes do not exhibit the same resolution and orientations. During rendering, the scene is sampled at many positions to determine the appearance at these positions. Figure 3.2 exemplifies how sampling along a viewing ray works with inverse deformations. The sampling is at positions P_1 to P_5. The cuboid of data is visually cut along its topside and folded over. This is realized by changing the sampling positions as in Figure 3.2 (right). For sampling position P_1, data has to be looked up at P_1', for P_2 at P_2', for P_3 at P_3', for P_4 there is no data defined, and for P_5 the data is looked up at the unchanged position. That means sampling positions might have

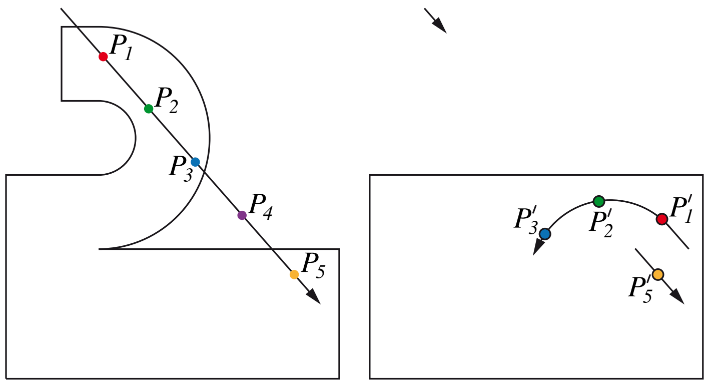


Figure 3.2: *Principle of common inverse deformations. When sampling in a deformed scene at positions P_1 to P_5 along a viewing ray (left), the data is looked up at positions P_1' to P_5' (right). The volume data stays unchanged. P_4' is not defined.*

to be modified or not evaluated at all. It is also possible, that data has to be looked up at more than one position to determine the overall appearance. Especially in dataflow-based shader frameworks, in which data is passed through the processing graph, inverse deformations are difficult to add and typically require special-case handling.

Illumination and Deformation: All hand-drawn anatomic illustrations of the heart use shading and shadows to depict shapes and structures. Typically, a diffuse frontal light, slightly moved towards the upper left, is used. It casts a soft shadow. There are several ways to compute shadows. As in reality, the algorithmic casting of shadows has to be influenced by the cuts and deformations.

To compute shadows during the evaluation of the sampling positions via ray casting is close to reality. This section will concentrate on this precise approach to compute shadows. Thus, secondary rays are cast that either hit light or matter. See Figure 3.3 for different patterns of secondary sampling rays. The technical problem is that these rays have to take into account the aforementioned inverse deformations and cuts. That is, they have to be transformed as in Figure 3.2. The same is true for shading where computation always requires the correct (i.e. deformed and cut) surface normal. This surface normal is no longer defined by the data, but by the used visualization techniques. It is possible (and may be more performant) to write new algorithms, that compute the correct gradient whenever visualization techniques are combined. But it is not very flexible for the creator of visualizations.

That is, in Figure 3.1 (e), it is not trivial to compute the shading and the shadow, for the rendered data still has, despite all deformations, the shapes of a cube and a

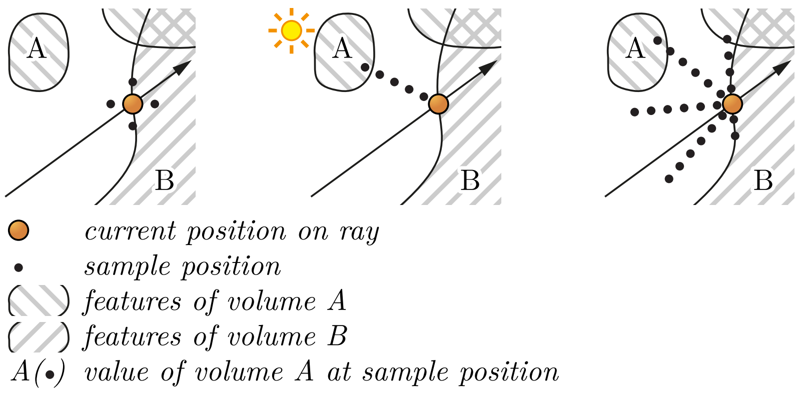


Figure 3.3: *Three sampling patterns for computing illumination. Left: Gradient for Phong shading. Center: ray-casted hard shadows with one point light. Right: ray-casted shadows with environmental illumination and 5 rays. The legend underneath will be used for all figures in this subsection.*

sphere. We found no concept in the literature that solves this problem universally for dataflow-based shading graphs. That is, shading and shadowing nodes do not have to be adapted to correctly illuminate any visualization, taking into account all deforming nodes and cutting nodes (and of course transfer functions etc.). All solutions we found in the literature introduce a special case handling and are therefore not that flexible. Technically it will be shown that this problem can be reduced to the ability of nodes being able to *modify*, *skip*, or *add* sampling positions *for whole subgraphs*.

Gradients in Multi-Volume Rendering: Logical combinations (as presented in Section 2.6) necessitate a massively larger dataflow when combined with sampling patterns (such as the gradient). This turns out to be a major usability problem when using dataflow graphs and is explained in more detail below. To be clear, Figures 3.4, 3.5 and 3.6 show the *necessary* dataflow for computation, i.e. it is the same, regardless of whether it is specified by a dataflow graph or by a fixed and predefined shader program.

Figure 3.4 shows the necessary dataflow for two OR-combined volumes when color and opacity has to be determined for the current position on the ray. Both, the value of volume A and the value of volume B, have to be evaluated at the sample position. The sample position is equivalent to the current position on the ray.

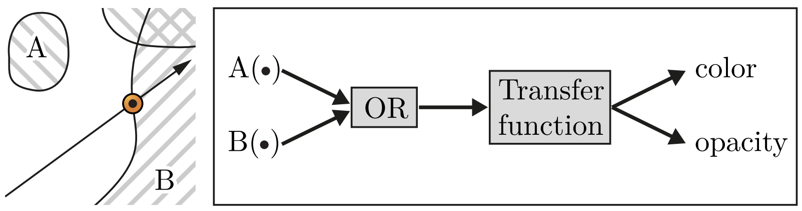


Figure 3.4: *Mapping color and opacity to two OR-combined volumes with a transfer function necessitates the shown dataflow, which needs to be evaluated for each position on every ray within direct volume rendering (legend in Figure 3.3).*

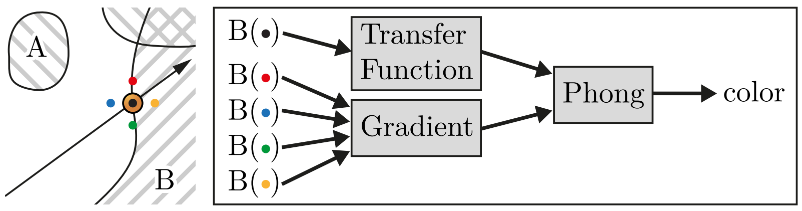


Figure 3.5: *Necessary dataflow of single volume rendering (only volume B) with Phong shading. Different sampling positions are colorized to be distinguishable (legend in Figure 3.3).*

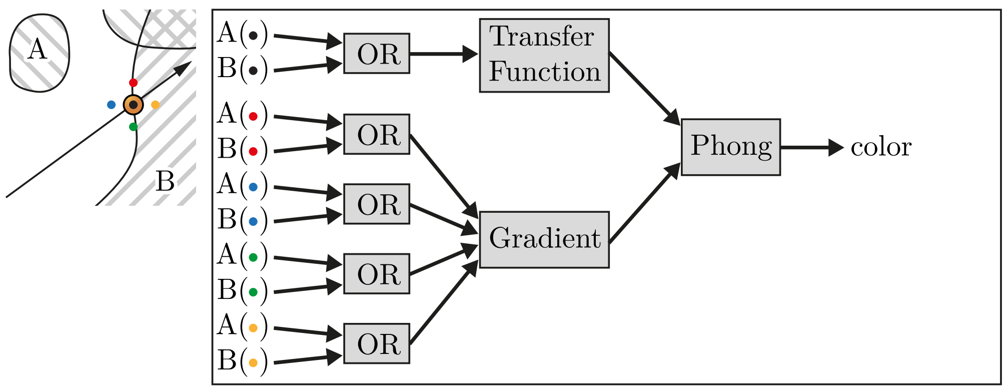


Figure 3.6: *Same situation as in Fig. 3.4 but with Phong shading applied as in Fig. 3.5. The necessary dataflow is considerably bigger when the sampling pattern of the Gradient node is used in combination with two volumes (legend in Figure 3.3).*

Figure 3.5 shows the necessary dataflow with a Gradient and a Phong node if only one volume is sampled. Figure 3.6 shows the increase of the necessary dataflow if both volumes are logically combined in the same situation. The whole dataflow in front of the node that uses sampling patterns, i.e. the Gradient node, is traversed for each sampling position in the pattern. As a result, in traditional dataflow-based systems the user would have to create a graph with many duplications (consider that the gradient actually has six sampling position in 3D).

To not replicate the whole necessary dataflow visually in diagrams, sampling patterns are handled as a special case in current state of the art systems that use dataflow diagrams in volume rendering. This special case handling limits the *flexibility* and *extendability* of these systems. For example, the *flexibility* is affected when special cases either limit the combinability of a node with other nodes, or do not allow for loops in sampling patterns, or do not allow for adjusting and dynamic sampling patterns. The *extendability* is affected when adding new nodes with new sampling patterns requires the developer to reprogram a central component, such as the shader composer. The goal of this section is to replace this special case handling of different input data sampling patterns with a flexible and general approach that supports arbitrary input data sampling patterns without affecting the flexibility or extendability of the system as a whole.

3.2.3 Related Work

Shading and shadows in deformed volume rendering: Jönsson et al. [JSYR12] compare and categorize state-of-the-art algorithms for volumetric shadows in interactive volume rendering. They report that some algorithms can be adapted to support clipping planes. Arbitrary oriented cuts or elastic deformations are only supported by half-angle slicing [KPH*03]. However, shadowing by half-angle slicing is not compatible with ray casting and does not solve the problem with gradients for shading. For our use case a flexible combination of any deformations has to be illuminated correctly, and due to the experimentation status, it is not feasible to reprogram the shadowing algorithm for every change of the deformations. Thus, we decided for a simple but flexible ray-tracing based approach, similar to Li and Mueller [LM05]. Despite inferior performance compared to half-angle slicing, it is still interactive when used in our progressive ray casting system.

Others invest a rather big effort to compute correct shading. Correa et al. [CSC06] distinguish three kinds of normals that are processed differently. For unmodified regions, they use a preprocessed normal. For deformed regions, they calculate a rotation matrix from the deformation function and safe it in an additional volume. For normals at cuts, they blend the cut normal and the normals in the vicinity to achieve a correct shading. This might lead to a better performance compared with on-the-fly computing during rendering. However, when parameters of the deformation function change, this approach necessitates a repeated preprocessing. Furthermore, in a dataflow-based shading graph, as used in this thesis, there are many possible sources for changes to the normal, e.g. cuts, complex transfer functions, deformations, or logical combinations. All of them would necessitate a special treatment which might not be predictable. Hence, a computation of the gradient on-the-fly, based on the true negative opacity is the only feasible solution.

Sampling Patterns in Dataflow-based Shader Programming: Dataflow-based data processing is very common in visualization. In spite of that, besides the system presented in this thesis, there are only four other volume rendering systems in the literature whose *shader framework* can be considered dataflow-based in the broader sense. All four systems are data-driven. The technique presented in [RBE08b, RBE08a] supports a fixed set of prefetched and pre-calculated sampling patterns for the current sampling position, such as the data value, the gradient vector, and a curvature metric. These values are provided to each node. Voreen [MSRMH09] uses dataflow graphs for image-level intermixing, but not on a per sample basis. However, the images are blended in the shader supporting depth-maps, which can result in correct visualizations in many situations. MeVisLab has been extended by a Shader Framework [RPLH11], that is based on the SuperShader concept [McG05], to enable effortless rapid prototyping. It uses a linearly ordered pipeline instead of a graph or tree-based concept. In [PHF07], the shader composer contains a global special case to realize the special input data sampling pattern of the Gradient node. It enables the Gradient node to have only one input port and to be at any position in the pipeline. If a Gradient node is detected in the pipeline, the shader composer duplicates the branch in front of the Gradient node six times, once for each sampling position. Other sampling patterns are difficult to add, especially deforming patterns or patterns using conditional loops.

In the framework presented earlier in this thesis (see Section 2.6), the Gradient node can sample the volume itself six times and calculate the gradient vector. This concept does not allow to place other nodes in front of a Gradient node in the dataflow graph as the special case in [PHF07] does. This is because the Gradient node requires direct access to the volumetric dataset in order to realize its sampling pattern. Errors as in Figure 3.1 (b) and (c) occur and the flexibility is restricted.

3.2.4 Concept of a Dataflow-based Shading Graph for Deformations and Shadows

The nodes in dataflow-graphs represent data processing operations and can consume, produce, and modify data. There is the possibility of data-driven and demand-driven evaluation. In general, whenever the input data of a node changes, its output data might be outdated and has to be updated by evaluating the node. The details about how a node handles its input data and how this affects other nodes is subject to the selected evaluation model.

In the *Data-Driven Evaluation model*, a node is evaluated as soon as it has new data available on its input ports. During its evaluation, a node writes data to its output ports. This can trigger re-evaluations of the nodes that are connected to these output ports because they might have new data available on their input ports. As stated, all shader frameworks we found in the literature make use of this data-driven evaluation model.

In the *Demand-Driven Evaluation model*, the demand for output data defines the evaluation order of connected nodes. A node is evaluated when another node requests its output data. The demand-driven evaluation model is also called lazy evaluation [Wad71] as nodes are evaluated only when there is a request for their output data. *The technical requirement to render deformed and illuminated multi-volume scenes correctly is a general solution to allow nodes to modify, skip, and add sampling positions.*

Systems with data-driven evaluation come with two limitations that hinder this:

1. Nodes have no control over the evaluation of other nodes that provide them with input data.

2. The current sampling position is controlled outside of the dataflow graph and is constant inside the graph. The multi-volume rendering system feeds the dataflow graph with data prefetched at the current sampling position. The individual nodes have no control over the sampling position.

The proposed concept of *Demand-driven dataflow* avoids these limitations. The demand-driven evaluation model is used to avoid the first limitation. It gives nodes control over the evaluation of other nodes that provide them with input data. For ex-

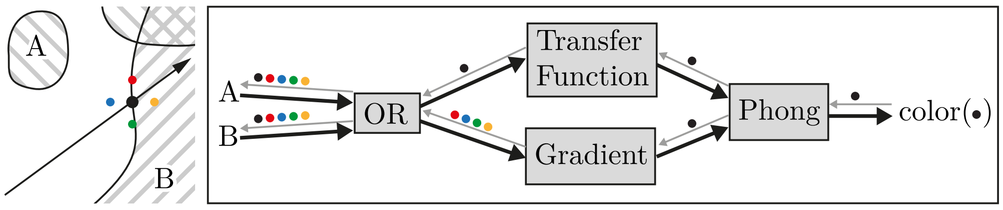


Figure 3.7: *Demand-driven dataflow: Data flows in direction of the black arrows. It is requested for specific sampling positions (indicated by the gray arrows and dots). The Gradient node realizes its sampling pattern on its own by changing the position and number of requests.*

ample, the Phong node can decide if and when (and how often) to evaluate the Gradient node to get the surface normal vector for the current sampling position. To avoid the second limitation, the nodes are now able to *request data for a specific sampling position.* The shader composer can realize this request as a sampling position parameter to the nested shader function calls (see Section 3.2.5 for application details). Figures 3.8 and 3.7 explain this concept visually. In Figure 3.7 the Gradient node realizes its sampling pattern on its own by changing the position and number of requests. In Figure 3.8 the functioning of a deformation is depicted.

Now the sampling position can be different between two requests inside the dataflow graph. This makes it possible to insert inverse deformation nodes, gradient nodes, and illumination nodes. There is no global sampling position, instead every node is requested for port-data at a specific sampling position. Only this specific sampling position can

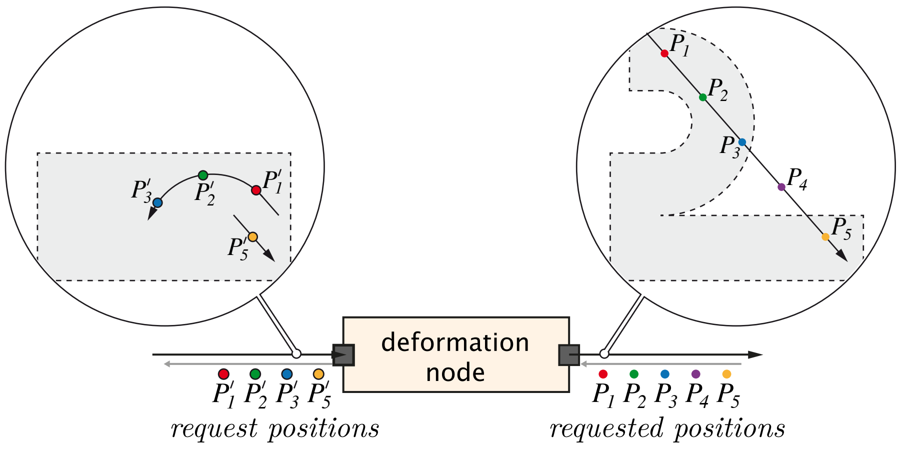

P'_3 P'_2 P'_1 P'_5
P_1 P_2 P_3 P_4 P_5
deformation node
P'_1 P'_2 P'_3 P'_5
request positions
P_1 P_2 P_3 P_4 P_5
requested positions

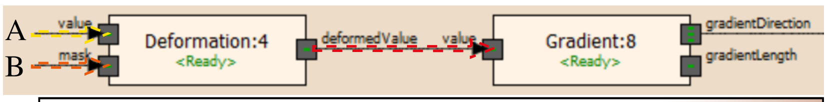
Figure 3.8: *Data is requested for specific sampling positions. A node can now change that sampling positions in its own request or add or skip requests. Dataflow direction is indicated by black arrows, request direction by gray arrows (compare with Figure 3.2).*

be used by nodes (unchanged or modified) to request data from preceding nodes. This way deformations influence all preceding nodes and several deformations can be combined correctly without the need to adapt any node. E.g. nodes computing shadows will evaluate sample positions in a ray towards the light. Every evaluation of a sample position is a request to the preceding dataflow graph and will run through all deformations, cuts and transfer functions in this subgraph. Hence, nodes computing shadows will be influenced correctly by all preceding operations. Nodes, such as the Gradient node, can realize their special input data sampling pattern on their own. This circumvents the necessity for a duplication of the graph's branch in front of the node. Other nodes without sampling patterns, such as the OR node, are passing on the sampling position they are evaluated for in their own requests. Finally, the volume is sampled considering the arbitrarily modified sampling position.

Usual demand-driven evaluation models do not have parameters in their requests to modify the requested data but only produce, consume, and modify data. This is because all parameters that could change the result would always have to be passed through the whole dataflow, leading to a very big request's signature. The reason why this unusual addition of a parameter to the requests is possible in volume rendering is the heavy, and sole dependence of the rendering process on the sampling position. It would be possible to add other parameters like the light position, but it is difficult to think of practical examples where the light position changes between sampling points. Although this parameter in the signature is not part of the demand-driven evaluation model, the approach in this section will be referred to as demand-driven evaluation model for simplicity.

3.2.5 Shader Composition Process

Whereas the former subsection explains the concept more theoretically this subsection explains how to implement the concept in an easy way using the shader compiler's preprocessing stage and call-by-value-return evaluation strategy. Some more application details are given to account for reproducibility. Explanations will be made using a small connected graph. Figure 3.9 (top) shows an excerpt of a dataflow graph, which is specified by the user. "Deformation:4" is a node that cuts and translates a masked area by a user-defined displacement (see circle). "4" is the ID of the node. In lines 22 − 37 the node's rather simple code block is depicted, which needs to be programmed by the programmer. The variable `samplePosition` stores the position that data is requested for. When reading from ports or writing to ports, their name (in this example `value`, `mask` and `deformedValue`) is used as variable. When the inbound data needs to be evaluated at a different sampling position, the port's name prefixed by a `get_` is used as a function call (lines 27 and 31) using call-by-value-return. This way each node can evaluate the preceding subgraph due to its demands.



```
 1  uniform vec3 Deformation_4_displacement;
 2
 3  void PreviousNodeA_ID_portName(in vec3 samplePosition, out float portName){
 4      ...
 5  }
 6
 7  void PreviousNodeB_ID_portName(in vec3 samplePosition, out float portName){
 8      ...
 9  }
10
11  void Deformation_4_deformedValue(in vec3 samplePosition, out float deformedValue)
12  {
13      float value;
14      PreviousNodeA_ID_portName(samplePosition, value);
15      float mask;
16      PreviousNodeB_ID_portName(samplePosition, mask);
17      vec3 displacement = Deformation_4_displacement;
18
19  #define get_value PreviousNodeA_ID_portName
20  #define get_mask PreviousNodeB_ID_portName
21
22      vec3 requestedPosition = samplePosition;
23      // inverse of deformation:
24      vec3 inverseDeformedPosition = requestedPosition - displacement;
25
26      float maskDeformedPosition;
27      get_mask(inverseDeformedPosition, maskDeformedPosition);
28
29      if(maskDeformedPosition == 1.0) { // deformed position inside mask?
30          // sample at deformed position:
31          get_value(inverseDeformedPosition, deformedValue);
32      } else if(mask == 1.0) { // inside the not defined area?
33          deformedValue = 0.0; // do not sample
34      } else { // inside the untouched area
35          deformedValue = value; // sample at requested position
36          // same as get_value(requestedPosition, cutValue);
37      }
38
39  #undef get_mask
40  #undef get_value
41  }
42
43  void Gradient_8_gradientDirection(in vec3 samplePosition,
44                                    out vec3 gradientDirection)
45  {
46      float gradientLength;
47  #define get_value Deformation_4_deformedValue
48
49      vec3 negative, positive, difference;
50      float maxGradientLength = 1.732050808; // = sqrt(3.0)
51
52      vec3 offset = vec3(global_basicSampleDistance, 0.0, 0.0);
53      get_value(samplePosition - offset.xyz, negative.x);
54      get_value(samplePosition + offset.xyz, positive.x);
55      get_value(samplePosition - offset.yxz, negative.y);
56      get_value(samplePosition + offset.yxz, positive.y);
57      get_value(samplePosition - offset.yzx, negative.z);
58      get_value(samplePosition + offset.yzx, positive.z);
59
60      difference = negative - positive; // central difference
61
62      gradientDirection = normalize(difference);
63      gradientLength = length(difference) / maxGradientLength;
64
65  #undef get_value
66  }
```

Deformation node code block

Gradient node code block

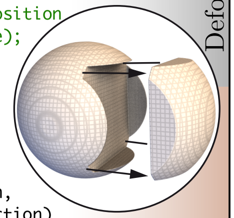


Figure 3.9: *Top: The user connects a simple deformation node, that cuts and translates a masked area by a user-defined displacement, and a Gradient node, that can be used for illumination. Code: The programmer programs universal code blocks (marked gray), while connections and explicit naming are realized in the skeleton code generated automatically by the shader composer (marked reddish). Each connected output port has its own function. The shader composer realizes connections between ports by either define statements (l. 47, ll. 19-20) – if sample position is changed – and/or by initialization (ll.13-16) – if sample position stays the same. Circle: Visual result, when the deformation node is applied to a sphere using a cubic mask and a global illumination node.*

When connected to other nodes in a dataflow graph, the code of these nodes has to be composed to a single shader. The shader composition process consists of four steps:

1. Discover the relevant part of the dataflow graph, i.e. every node that is in the subgraph connected to the output.
2. Find a topological order of the relevant shading nodes.
3. Collect the additional parameters from each shading node, i.e. uniform variables storing user input (see user defined `displacement` in Figure 3.9).
4. Generate skeleton code for the complete shader program (see Figure 3.9, marked reddish).

The shader composer generates a separate function for each output port of each node. This function takes the sampling position as in-parameter and provides the output data as out-parameter (GLSL supports call-by-value-return). If more than one node of one type is inserted into the dataflow graph, a naming collisions would occur. Even if two nodes share the same port name, this is a problem. To avoid this the shader composer adds a unique prefix to each parameter and port of a node. Additionally, it initializes the parameters of the corresponding node inside the function body without the unique prefix (l. 1 and l. 17). This allows the programmer to not know the unique prefix, but to program as if this node was the only one. Connections are realized either by initialization (if the sample position stays), or by define statements (if the sample position is modified), or both (if both is used).

The proposed shader composer allows for programming any deformations. Additionally, the true negative opacity gradient will be automatically computed if the Gradient node is placed at the corresponding position. The same is true for more complex sampling patterns as in ray casting based shadow computation. In the code example, the Gradient node is connected to the output port of the Deformation node. The shader composer injects a define statement (l. 47) into the function body that maps the `get_value` symbol to the function of the connected `deformedValue` output port of the Deformation node. In the Gradient code block there are six calls to the `get_value` function for six different sampling positions. This way each request to the `gradientDirection` port causes six (offset) requests for neighboring positions in the Gradient node, which are then forwarded to the Deformation node that may modify each position again before data is requested from node A or B respectively.

3.2.6 Design Decisions and Results

Ray Evaluation Nodes A ray evaluation technique defines the sampling positions at which the shading dataflow graph is evaluated along a ray (called *current position on ray*

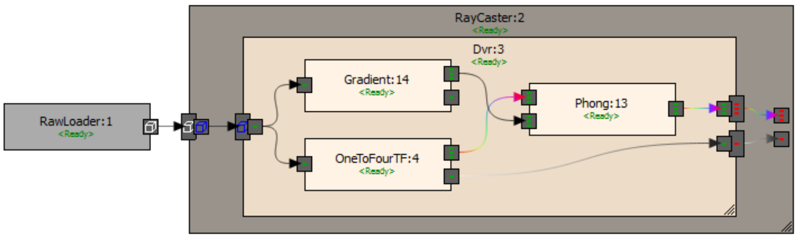


Figure 3.10: *In demand-driven dataflow mode the composition scheme of DVR can be realized as a ray evaluation node and can easily be replaced by ISO, MIP, and MIDA.*

before). It also defines how to determine the overall color for a ray from all the samples along it. The RayCaster node used to have Direct Volume Rendering (DVR) hardcoded as its ray evaluation technique. There are several other ray evaluation techniques, such as isosurface rendering (ISO), Maximum Intensity Projection (MIP) and Maximum Intensity Difference Accumulation (MIDA) [BG09]. With demand-driven dataflow the ray evaluation technique can be moved out of the rendering node into a separate shading node. This is possible because using demand-driven evaluation such a shading node can evaluate its data providing nodes *multiple times* for *different sampling positions* along the ray. Now, the rendering node does not specify the sampling position anymore, but specifies a ray instead. A ray evaluation node, such as the new Dvr node in Figure 3.10, determines the sampling positions for a ray and evaluates its internal dataflow graph at these sampling positions to accumulate the final color and alpha for a given ray. It can easily be replaced and the composition is not hardcoded in the RayCaster node. As a result, there are four different request's signatures that are valid in different areas of the dataflow graph and are visualized by different colors. Only ports with the same request's signatures can be connected. Figure 3.11 shows the different signature colors. This design decision does not only allow for an independent implementation of the ray evaluation nodes, it also enables the use of several different ray evaluation nodes that can be blended on a per ray level (i.e. image level intermixing). Figure 3.12 shows image level intermixing of MIP and DVR.

⬜ *Preprocessing* 🟦 *Volume Signature* 🟩 *Sample Signature* 🟥 *Ray Signature*

Figure 3.11: *The request's signature of a port is indicated by the color of the port's icon. White indicates preprocessing nodes, blue indicates requests on a per volume level, green indicates requests on a per sample level, and red indicates ray-based requests. Only ports that exhibit the same request's signature can be connected.*

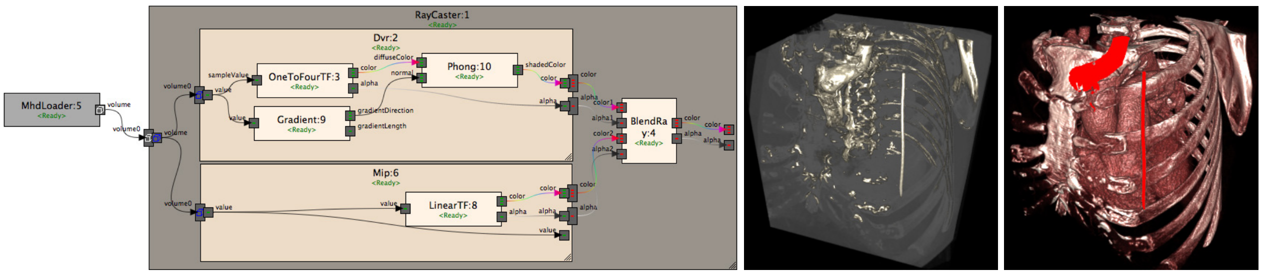
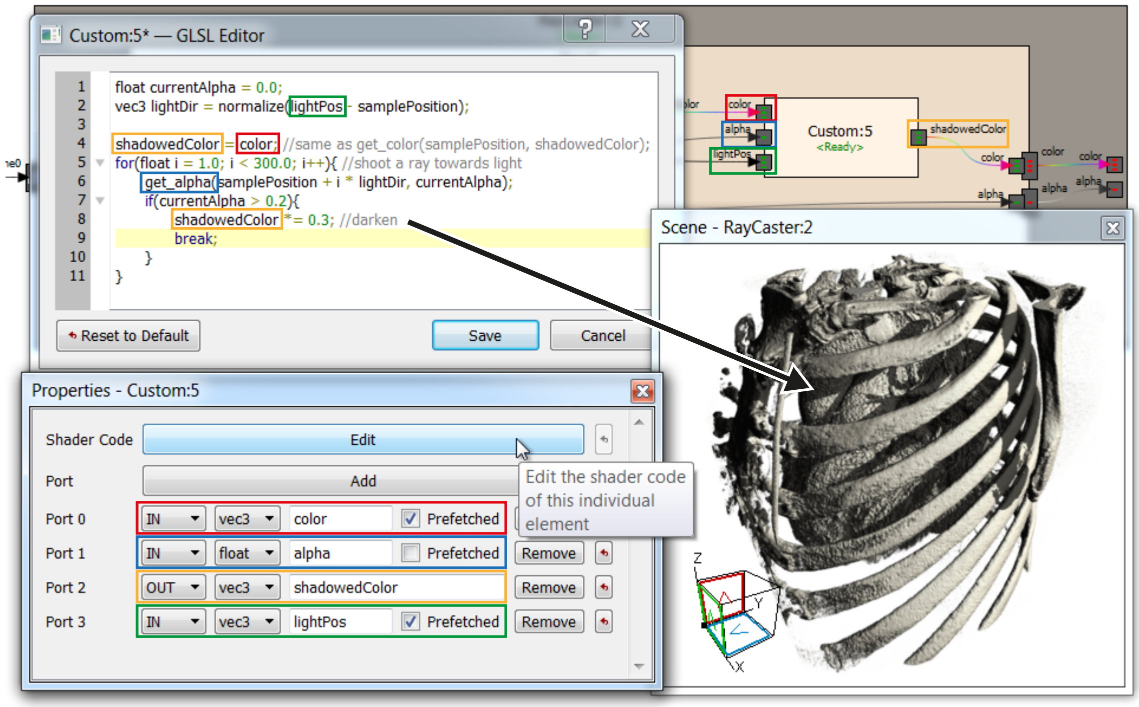


Figure 3.12: *Image level intermixing of MIP and DVR. Left: Dataflow graph using a Dvr node (upper) and a Mip node (lower) for ray evaluation. Both results are overlaid using a BlendRay node. Center: Result of the dataflow on the left side, i.e. DVR is laid over MIP. Right: Opposite example with MIP over DVR, i.e. the connections of the BlendRay node have been interchanged (and different transfer functions have been chosen).*

Zero Alpha Check It is common that the alpha part of the dataflow graph is cheap to evaluate and the color part is much more expensive as it is modified by shading, whereas opacity is not. The dataflow graph in Figure 3.10 exposes this characteristic:

The internal color input port of the Dvr node is connected to a branch that contains all three internal nodes. In total it requires seven texture fetches for every position on the ray. The internal alpha input port of the Dvr node is connected to the OneToFourTF node only. It only requires one texture fetch for every position on the ray. The Dvr node can utilize this characteristic to avoid unnecessary evaluations of its internal dataflow graph systematically. If a sampling position has a zero alpha value, then there is no need to evaluate the dataflow graph for the color value at the same sampling position as this color does not contribute to the overall color of the ray. Demand-driven dataflow allows the Dvr node to evaluate the alpha part first and then skip the color part when the alpha value is zero. This is possible due to the demand-driven evaluation model allowing for short-circuited evaluation. Every other node can use this tactic, too. In data-driven dataflows this is not possible because nodes cannot control when or how often to evaluate their previous nodes.

Rapid Prototyping Rapid prototyping is a key feature in terms of flexibly creating visualizations. Concerning this matter, the integration of the proposed demand-driven dataflow into our graph based rapid prototyping environment turned out to be easy and seamless. The shader code of every individual node can be edited arbitrarily during run-time in a simple editor-window, i.e. the node's behavior can be changed without editing and recompiling C++ code. The input and output ports are available as variables of the same name. The unique prefixes are automatically injected using the same define statements as in Figure 3.9 and are not visible to the programmer. If the data is needed for a different sample position, the input ports can also be used as function calls. See Figure 3.13 for a shadow node (11 lines of code) that requires demand-driven evaluation and is created completely during run-time.



```
float currentAlpha = 0.0;
vec3 lightDir = normalize(lightPos - samplePosition);

shadowedColor = color; //same as get_color(samplePosition, shadowedColor);
for(float i = 1.0; i < 300.0; i++){ //shoot a ray towards light
    get_alpha(samplePosition + i * lightDir, currentAlpha);
    if(currentAlpha > 0.2){
        shadowedColor *= 0.3; //darken
        break;
    }
}
```

Shader Code	Edit				
Port	Add				
Port 0	IN	vec3	color	✓ Prefetched	
Port 1	IN	float	alpha	Prefetched	Remove
Port 2	OUT	vec3	shadowedColor		Remove
Port 3	IN	vec3	lightPos	✓ Prefetched	Remove

Figure 3.13: *A Custom node (top-right) that realizes ray casted shadows, completely created during run-time within one minute. The ports of the Custom node are added and specified in the node's properties (bottom-left). In the GLSL code of the node (top-left) input ports can be either accessed as prefetched variable (`color`, `lightPos`) or via function call according to the demand-driven evaluation mode when sample position needs to be modified (`get_alpha`). Code, ports, and properties that belong together are marked with the same color.*

Concatenating Visualization Techniques If several sampling patterns are concatenated and, therefore, depend on each other, demand-driven evaluation with the sampling position in the request's signature is the only practicable way to produce correct results. Figure 3.14 uses several concatenated techniques to visualize a heart dataset:

1. A dataflow of logical nodes to treat Hounsfield values of blood and tissue differently.

2. A segmentation-based cut away of the surroundings.

3. An elastic deformation – by changing the sampling position [BV09] – applied to the tissue.
4. Phong shading using negative opacity gradients.
5. Ray Casting based global illumination for soft shadows.

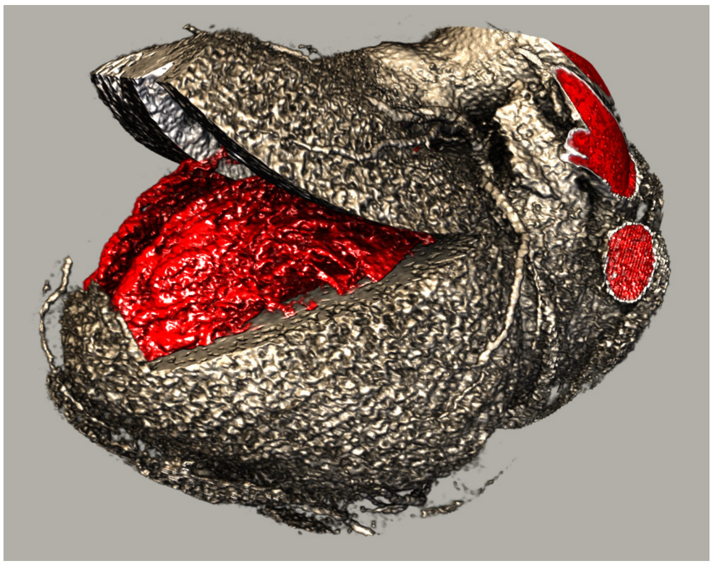


Figure 3.14: *Visualization of a heart dataset. In this example, many techniques are concatenated and sampling patterns are dynamically adjusting but rendering is still correct.*

Each technique takes into account the prior techniques correctly without introducing any special case or programming overhead, e.g. the global illumination automatically casts shadows with respect of the deformation. In this example, the sample positions are additionally not known a priori, but are determined during run-time in the shader. This is because of the adjustable deformation and the global illumination which shoots a bunch of rays dependent on the gradient of the surface, which, in turn, is dependent on the segmentation and the transfer function. It is difficult to imagine a data-driven dataflow to realize this concatenation.

3.2.7 Evaluation

Using the proposed concept, it is very simple to program deformations and complex global illumination. Using the common data-driven evaluation, it is not. Every node in a common data-driven evaluation that changes the sampling position needs direct access to the samplers, i.e. it has to be the first node in the dataflow graph. Suffering from this restriction, it is not possible to use more than one deforming node, nor to combine deformations with illumination nodes performing ray-casting. Other (e.g. screen-space) algorithms would have to be used for illumination that are by no means comparable.

That is why we decided to compare both concepts using rather simple dataflow-graphs with Phong shading and without deformations. Then the only node that needs to change the sampling position is the Gradient node, which can be the first node in the dataflow graph if gradients do not need to be the true negative opacity gradients.

For simplicity we will refer to the data-driven evaluation as “the old” mode and to the demand-driven evaluation with modifiable sampling position as “the new” mode. The old mode is representative for all dataflow-based systems that use a data-driven evaluation strategy.

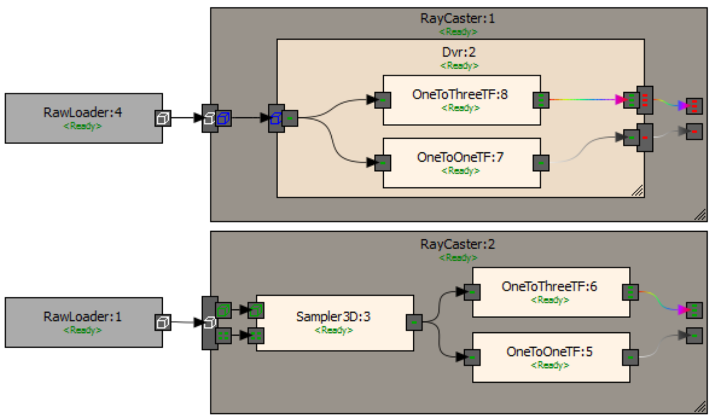


Figure 3.15: *In this example (details not important) the assembly codes* compiled *from the new demand-driven mode (top) and the old data-driven mode (bottom) dataflow graphs are identical although severely different in uncompiled GLSL-code.*

Assembly Code Comparison When comparing time measurements of the new and the old mode, it turned out that they were the same. Hence, we changed the evaluation for a deeper insight. AMD’s GPU ShaderAnalyzer and NVIDIA’s NVemulate allow to compare the old and the new mode on the assembly code level and to evaluate whether the differences in the composed GLSL code result in actual differences on the assembly code level. This comparison is done using high-end GPUs from AMD and NVIDIA. Figure 3.15 shows two equivalent dataflow graphs in the old and the new mode with one volumetric dataset as input, and separate transfer function nodes for color and alpha value. The composed *GLSL* code for the old and the new mode is severely different in its structure.

In contrast, the *compiled* assembly code is identical in both cases on AMD and NVIDIA GPUs (Zero Alpha Check turned off). Identical assembly code in the old and the new mode proves that the difference in the structure of the composed GLSL code between data-driven and demand-driven evaluation model has no effect on the resulting assembly code in this example. Hence, the performance is exactly the same. We found some more examples where the old and the new mode resulted in identical assembly code. However, in most cases we noticed very small differences in operation and instruction count, in equal parts benefiting demand-driven and data-driven mode. The AMD and NVIDIA shader compilers seem not to be affected by the more complex function call structure of the composed GLSL code in the new demand-driven mode.

Double Evaluation Avoidance Nodes can have multiple output ports. In the new demand-driven mode each node is evaluated for each connection to its output ports. This results in multiple evaluations of a single node with multiple connected output ports for each single dataflow graph evaluation. This especially poses a problem for the ray evaluation nodes as they have two output ports for color and alpha and typically have a high evaluation time. The shader composer can detect this situation and generate code that avoids such double evaluation under the condition that all involved requests are for the same sampling position and originate from the same node.

On the other hand, it turned out that the compiler is also able to detect and avoid many double evaluations. The AMD and NVIDIA shader compilers prefer to inline all function calls in a shader program. In the new demand-driven mode each connected output port results in a function in the final GLSL code and each connection to an output port results in a function call. Function call inlining combined with double evaluations in the dataflow graph can result in assembly code duplication. The demand-driven mode example in Figure 3.15 (top) contains a double evaluation that is *not* avoided by our shader composer. The internal output port of the Dvr node (the port on the left of the two transfer functions) is connected to two nodes that evaluate it for the same sampling position. Nevertheless, the resulting assembly code for this dataflow graph is identical in the old and the new mode. This implies that the AMD and NVIDIA shader compilers are able to detect and avoid the first assembly code duplications due to double evaluations and function call inlining.

Performance Comparison As mentioned in Section 3.2.7, the assembly code is barely affected by the new demand-driven mode and timings are almost always exactly the same. Hence, in this section mainly Zero Alpha Check (ZAC) is evaluated. As explained in Section 3.2.6, the new mode is able to evaluate the opacity value first and only to compute the color value if opacity is different to zero. This can enable ray casting based shadow algorithms to run in realtime, because they are run extremely seldom (e.g. speedup factor of 3.48 in Figure 3.13). But also in normal Phong shaded scenes ZAC improves performance reasonably. In a simple test-scene containing a sphere (like the sphere in Figure 3.1 (a)) the new demand-driven mode, with ZAC enabled, reduces the frame duration by half (see Figure 3.16 for exact numbers). This is mainly due to the saved texture fetches in the transparent areas as can be seen by the load of the texture unit (TU) that decreases by 62%. The load of the arithmetic logic unit (ALU) decreases by 17%. The additional conditional branches seem to not significantly influence the result.

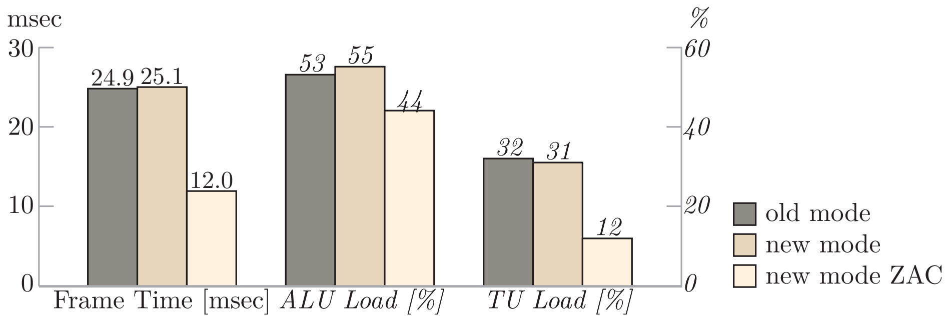


Figure 3.16: *Comparison of the performance of data-driven and demand-driven mode with and without Zero Alpha Check (ZAC) enabled in a simple Phong shaded test-scene containing a sphere (dataflow graph as in Fig. 3.10 using a step transfer function for alpha).*

Usability First of all, the new mode does not introduce any major changes to the user. A demand-driven shader framework can be used exactly the same way as a data-driven shader framework. Compared to our implementation of the old data-driven model, the only difference is that the sampling nodes are eliminated. However, in the implementation presented in [PHF07], there were no sampling nodes necessary anyway. When using this data-driven dataflow, some visualizations are not possible without introducing special case handling (e.g. Figure 3.14). Those which are possible, often demand a larger dataflow graph, especially when multimodal data is used. Figure 3.17 shows a comparison of equivalent demand-driven and data-driven dataflow graphs in a simple multimodal scene where the correct negative opacity gradient is used for Phong illumination. That is, the gradient is computed *after* determining the opacity by a combination of two modalities to ensure correct Phong shading. As explained earlier, in the old data-driven model Gradient nodes need to be the first nodes in the graph. In the old model, this means that the nodes "Not", "And", and "OneToFourTF" have to be duplicated for each of the six sampling positions of the Gradient node. The old data-driven dataflow graph exactly represents the whole necessary dataflow on execution level, whereas the new demand-driven dataflow hides any duplicated branches from the user. This way it is much easier for the user to create visualizations that use sampling patterns. When using more complex sampling patterns than the central difference gradient, it would not be reasonable to create multimodal visualizations using the old data-driven dataflow graphs. Using the new model, it is much easier for the user to create visualizations that use nodes that modify, add, or skip sampling positions, which is crucial when creating dissection visualizations as in anatomy atlases. Regardless of the amount of nodes, it is more intuitive that a Gradient node or a deformation node can be placed anywhere in the dataflow graph and is not restricted to positions directly at the volume input ports.

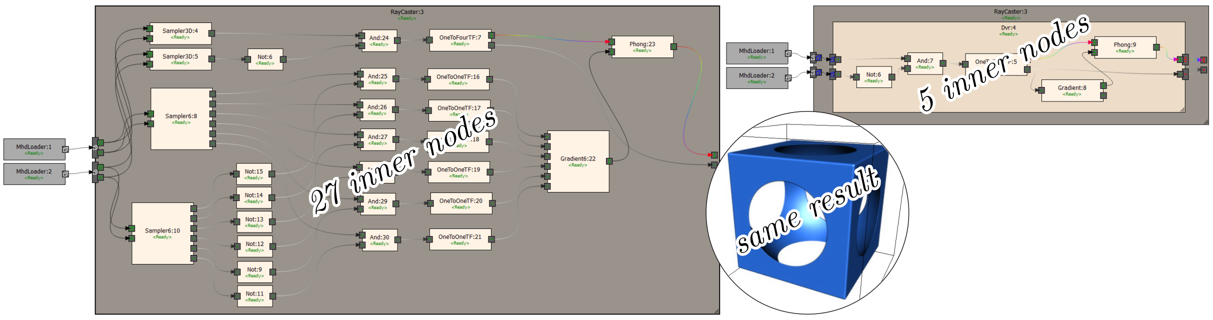


Figure 3.17: *Comparison of the dataflow graphs' sizes for the Phong shaded logical combination $C = A \wedge \neg B$ in the superimposed circle (same as Figure 3.1 (d)) using negative opacity gradients (A being the cube dataset and B the sphere). Both do not use special case handling but the dataflow for the old data-driven mode (left) needs 27 inner nodes, whereas in the new demand-driven mode (right) only five inner nodes are needed.*

3.2.8 Recapitulation

In this section a demand-driven dataflow is proposed that solves problems caused by deformations, illumination, and other sampling patterns when used in a dataflow based shading graph. We used the demand-driven evaluation model and extended it by adding the sampling position as a parameter to the requests. It avoids any special case handling regarding sampling patterns and, hence, is enabling the user to easily create very complex dataflows with many cascading visualization techniques. Dataflow-based systems can now use deformations and other dynamic sampling patterns seamlessly. Especially in multimodal visualizations this concept provides the user with flexibility. The concept is well suited for GPU-based volume rendering and gains much performance in many situations compared to a data-driven dataflow. We show its applicability by replacing the evaluation model of a graph-based rapid prototyping framework for GPU-based multi-volume rendering with the proposed demand-driven dataflow. We explain how to implement this concept in an easy way and propose beneficial design decisions. To show its usefulness, we compare some exemplary visualizations in terms of usability and performance to the typically used data-driven dataflow. Furthermore, we investigate the influence of the new technique on the assembly code level.

3.3 Interactive High Quality Rendering

Using the presented shader framework, it only takes a few mouse-clicks to create a visualization that exceeds by far the capabilities of current hardware regarding a fluent and interactive rendering. For a flexible usage of a shader-framework, which enables such demanding visualizations, a rendering system which stays interactive under every condition is necessary. On the other hand, this system needs to be able to output high-quality images as necessary in medical volume-visualizations. To meet both requirements, a progressive rendering was integrated into the framework which has been described in the technical report by Gmyr, Arens and Domik [GAD14]. This section explains the capabilities of this algorithm and is mainly based on the mentioned technical report.

3.3.1 Stochastic Sampling

In some areas of volume visualization such as medical imaging precision and accuracy of rendered images, is of utmost importance. In volume visualization, a high precision and accuracy necessitate a high sampling rate. However, a high sampling rate incorporating supersampling is no guarantor for a high image quality. What all regular sampling patterns have in common is that they produce aliasing artifacts when the image function contains frequency components beyond the Nyquist frequency [DW85]. To guarantee a high image quality without aliasing artifacts, one has to either solve the rendering equation analytically, which is not possible in most cases, or use stochastic sampling in combination with a high sampling rate. An example of stochastic sampling can be found in the human visual system, as noted by Cook [Coo86] as well as Mitchell [Mit87]. The example originates from Yellott's [Yel83] research on the distribution of cones in the human eye. He states that a particular region of the eye, called the extrafoveal region, does not suffer from aliasing artifacts for images that contain frequency components above the Nyquist frequency implied by the local cone density. Yellott finds the explanation for this phenomenon in the distribution of the cones which is called Poisson disk distribution [Coo86]. While point sampling of an image function which contains frequency components above the Nyquist frequency always introduces some error [DW85], sampling with a Poisson disk distribution scatters high frequencies into broadband noise instead of generating much more objectionable aliasing artifacts like Moiré patterns and jagged edges.

To generate high-quality images, the aforementioned rendering system was enhanced by a stochastic sampling of the image plane. A Poisson disk distribution was used to distribute the starting positions of the rays over the image plane. The sampling density was increased to support supersampling with 16 rays per pixel on average.

Other volume rendering systems utilize similar approaches. Kratz et al. [KRHH11] presented a variation of Levoy's approach [Lev90] for GPU-based rendering that is geared towards volume ray casting. They replace Levoy's simple heuristic based on the variance by a more sophisticated technique based on finite elements method to achieve explicit error control. While error estimation, ray casting, and reconstruction are executed on the GPU, the management of the underlying hierarchical data structures remains on the CPU. They use a regular sampling pattern and do not consider supersampling.

Kainz et al. [KSH*11] describe a GPU-based approach for volume ray casting that uses features in object space rather than features in image space to adaptively control the progressive refinement process. Their technique of organizing rays into buckets resembles the use of batches in our approach. They use a regular sampling pattern and a simple interpolation scheme for reconstruction.

The quality of antialiasing is often demonstrated via checkerboard patterns. Figure 3.18 shows a checkerboard pattern at an oblique angle rendered with conventional ray casting on the left, conventional supersampling in the middle, and with the proposed approach on the right. The left image contains jagged edges on the bottom and strong aliasing artifacts towards the top. The middle image eliminates the jagged edges at the bottom, but still suffers from aliasing towards the top. In contrast, the image generated with the used approach has soft edges and contains an increasing amount of noise towards the top which is especially concentrated in the corners of the image. While the smoother edges are a result of supersampling and filtering, the noise demonstrates the desired property of stochastic sampling to scatter high frequency components of the image function into noise instead of creating false patterns.

Figures 3.19 and 3.20 show an illuminated human thorax using conventional ray cast-

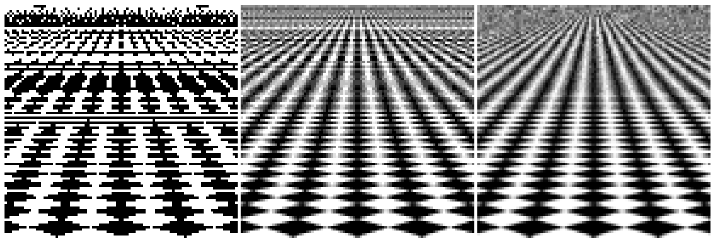


Figure 3.18: *Aliasing on an oblique checkerboard pattern. All three* 96 × 96 *pixel images show the same scene containing a checkerboard pattern at an oblique angle. The images were generated using regular sampling on a grid with one sample per pixel (left), regular sampling on a grid with 16 samples per pixel (middle), and our approach, i.e. stochastic sampling with 16 samples per pixel on average (right).*

ing and the approach proposed in this section, respectively. The most striking difference in quality between the two images is the amount of noise on the surfaces in the left image. This noise is actually a result of an insufficient sampling rate along the ray as well as a low number of samples used for shadow computation. In this example, the color produced by a ray almost exclusively depends on the first sample on the ray that enters the surface. Since interleaved sampling [KH01] along the ray is used for these visualizations, the depth of this first sample is randomized. The shadow samples are randomized, too. In the right image, the combination of interleaved sampling with the stochastic sampling and supersampling in image space effectively result in a comparatively high randomized volume sample distribution along all three dimensions, which leads to a considerably smoother appearance. Another noteworthy difference are the edges. While the edges are jagged in the left image the transition into the background in the right image is much softer. This is of course a direct consequence of the screen space supersampling. Finally, the small structures in the lung vessels can hardly be spatially associated in the left image due to aliasing artifacts and noise. This is much more easy in the right image.

3.3.2 Progressive Rendering

Besides image quality, interactivity of a volume visualization plays a very important role to enable the user to explore the data. Unfortunately, a high sampling rate comes along with a high computation time. Furthermore, some rendering techniques are too computationally expensive to achieve interactive frame rates. Especially when experimenting with visualization techniques in a rapid prototyping environment, the runtime is not always predictable. Given that both demands are contradictory one has to decide for a trade-off between quality and speed. If a reduced quality (e.g. through approximations) is not acceptable some visualizations may take several seconds to render. Nevertheless, it is advantageous to depict preliminary images in the meantime to allow for interactive navigation and parametrization. This technique is known as progressive refinement, which can provide both, interactivity, as well as, high quality images, when time is available.

In our approach we decided to avoid popping artifacts during refinement, but we provide a smooth transition between initially unsharp images and the finally sharp result. The final result is not approximative while intermediate results are not discarded, i.e. they are not computed superfluously, but contribute to the final image. This was realized by a *hierarchical* Poisson disk distribution of the casted rays. Every prefix of this hierarchical distribution is also a Poisson disk distribution, i.e. the rays are still well distributed over the image plane. This way the distribution can be divided into smaller batches of rays, which can be evaluated successively. Intermediate frames can be reconstructed from the already evaluated batches, while adding batches refines the

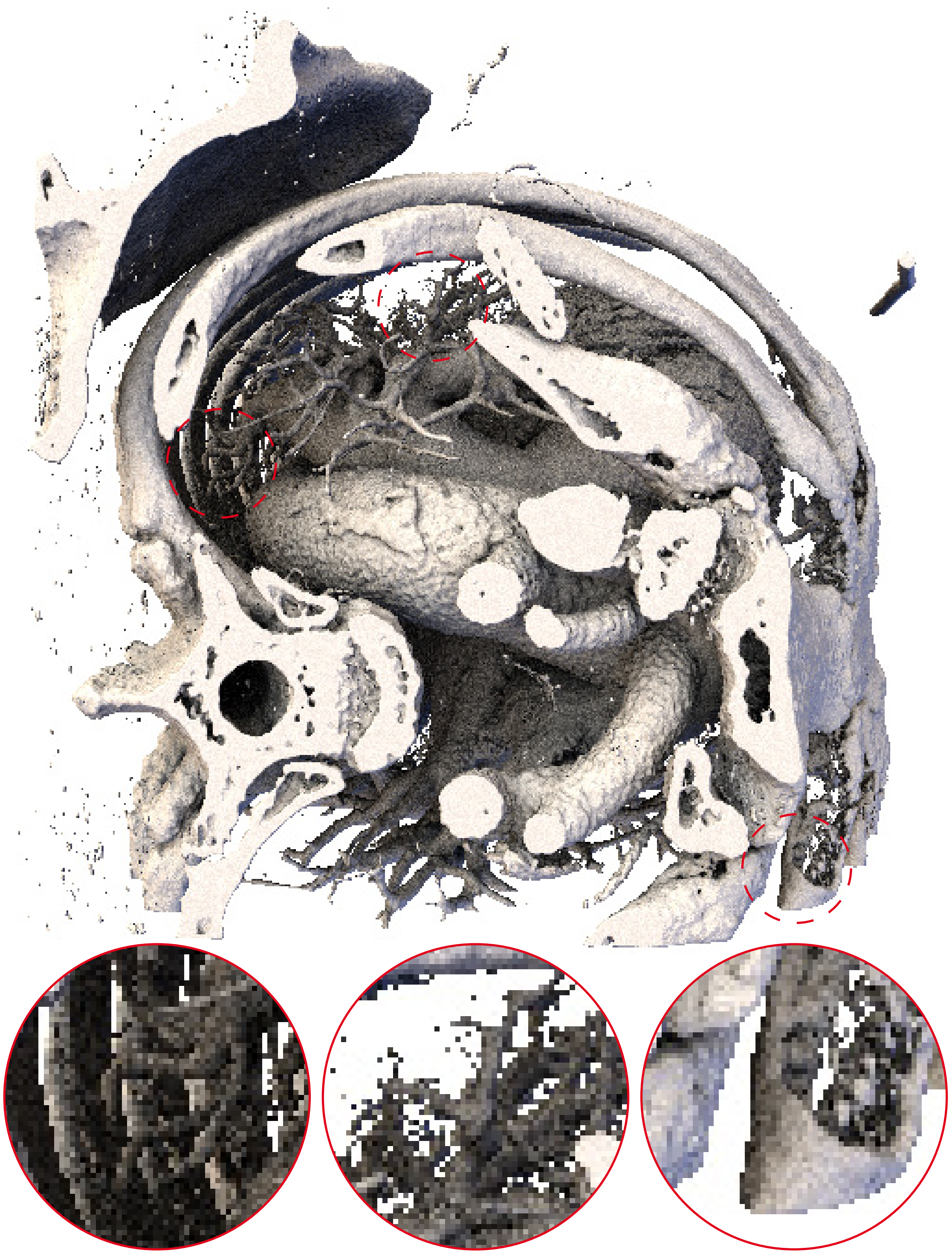


Figure 3.19: *CT Data Set of a thorax rendered using conventional ray casting with 1 sample per pixel.*

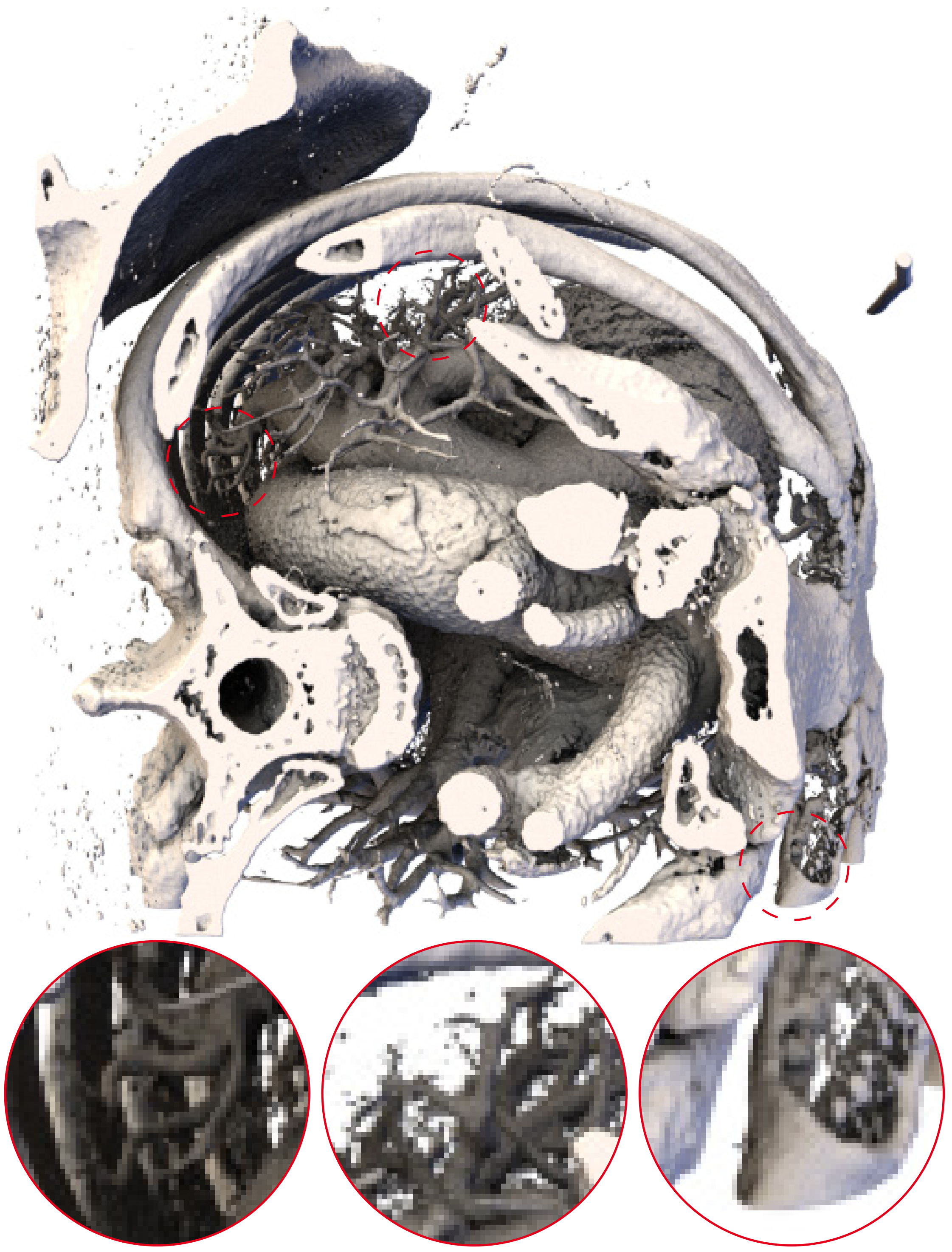


Figure 3.20: *CT data set of a thorax rendered using our approach with an average of 16 samples per pixel.*

rendered image. For reconstruction the evaluated rays are filtered in image space using a raised cosine filter. The size of this filter is adapted to the current density of the evaluated rays, resulting in possibly unsharp intermediate images. If the density of rays is denser than one ray per pixel, the filter size is not decreased further, resulting in a supersampling of the image plane. During refinement, the number of rays in the current batch is adapted to a size which is predicted to run at 30 frames per second, based on the time the last batch took to render. If rendering is very expensive our approach initially presents the user an unsharp approximation of the rendering. During the refinement, this approximation sharpens within a smooth transition over time. This can be explained best by a video which can be found in the supplementary material that comes with this document. The video shows the Stanford asian dragon [XYZ] and the stag beetle by Gröller et al. [GGK] during camera movement and altering complex illumination. Figure 3.21 shows the same scene in five different stages of progressive refinement. To be able to explain the smooth transition in a printable format we use the scene in Figure 3.22 which contains a volume dataset of many interweaved horizontally aligned helices in an orthographic projection. The x-axis of the image is used to show different intermediate results in progression. The top row shows 1024 forced intermediate results that are equally distributed between 0x and 2x supersampling (each intermediate result is shown in 1 pixel width). In the bottom row, a section of the subsequent intermediate result of 4x supersampling and the final image of 16x supersampling can be found as well as a conventional rendering (that is sampling on a regular grid without supersampling). Several properties of our approach can be observed. First of all, the transition is smooth. Second, the scene is constructed in a way that severe aliasing artifacts in many frequencies occur when sampling in a regular grid. In all intermediate frames our approach scatters frequencies above the corresponding Nyquist frequency into broadband noise. That is due to the hierarchical Poisson disk distribution, where all prefixes are

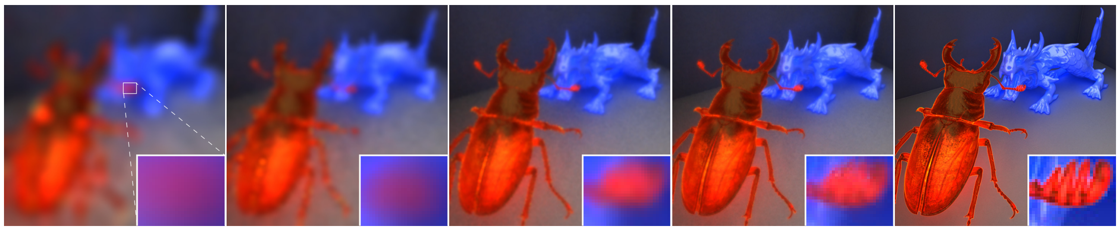


Figure 3.21: *Stanford asian dragon [XYZ] and CT scan of a stag beetle by Gröller et al. [GGK] using emissive transfer functions. The rightmost image uses 16 samples per pixel on average and took several seconds to render, whereas the leftmost image can be displayed at an interactive frame rate. Our progressive ray casting approach renders a smooth transition between both ends.*

also Poisson disk distributions. Finally, it has to be mentioned that the intermediate frame in this Figure are forced. In common scenes, the transition does not start at 0x supersampling and necessitates less intermediate frames.

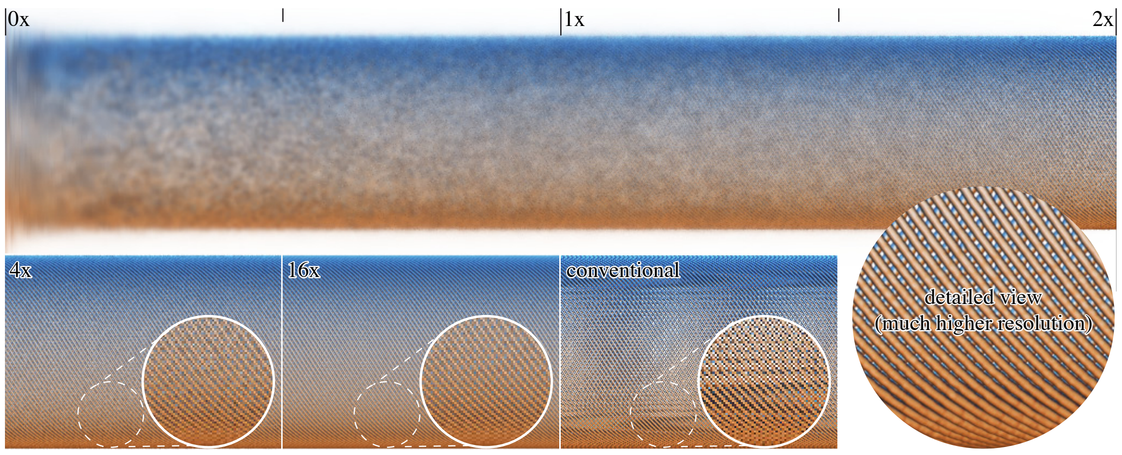


Figure 3.22: *Rendering of an artificial volume data set containing interweaved helices to show the progression of our approach. Top: Progression in our approach starting with the initial frame up to 2 samples per pixel on average. Each column of pixels originates from a new intermediate frame. Bottom row, left to right: Our approach at 4 samples per pixel on average, our approach at 16 samples per pixel on average, conventional rendering, structural details.*

3.3.3 Flexibility

To not limit the flexibility of the proposed shader framework, the progressive rendering extension was designed to be orthogonal to all other used techniques. Many approximative algorithms are closely interwoven with the underlying data structures and techniques. In a rapid prototyping environment, where techniques should be exchangeable, such an interwoven design can be very restricting. The hierarchical progressive sampling used here only determines a chronological order in which rays are shoot through the image plane. Hence, the only requirement to the underlying visualization is an interface which allows for evaluating the image function at a specific position. From the progressive renderer's point of view, the visualization is treated as a black-box for ray-casting, and thus, the flexibility is not restricted. Figure 3.21 shows three volumes (asian dragon, stag beetle and box) at the same time using global illumination and emissive transfer functions.

3.3.4 Performance

In our approach the frame rate remains comparably constant around 30 frames per second. Figure 3.23 visualizes the batch size adaption used in our approach to maintain interactivity. Again, the underlying experiment uses the scene from Figure 3.21 but now the camera path was chosen to induce alternating rendering complexity. Our approach successfully adapts to the rendering complexity; the batch size varies between $\sim$ 1000 and $\sim$ 19000 sampling positions. Beginning at frame 270, the graph shows the behavior of our approach during a progression phase without user interaction.

Comparing the performance of our approach to conventional ray casting is difficult because our approach attempts to decouple the frame rate from the rendering complexity. As a rough estimate of the overhead induced in our approach, the time required to reach an image with 1 sample per pixel, on average, can be compared between the two approaches. For the scene depicted in Figure 3.21, our approach requires 4.28 times more time than conventional ray casting. However, when rendering at a lower resolutions, the same experiment results in a factor as low as 1.15. This leads us to the conclusion that most of the overhead is due to reconstruction using large filter sizes. In general, this factor varies strongly depending on the data set and the visualization technique.

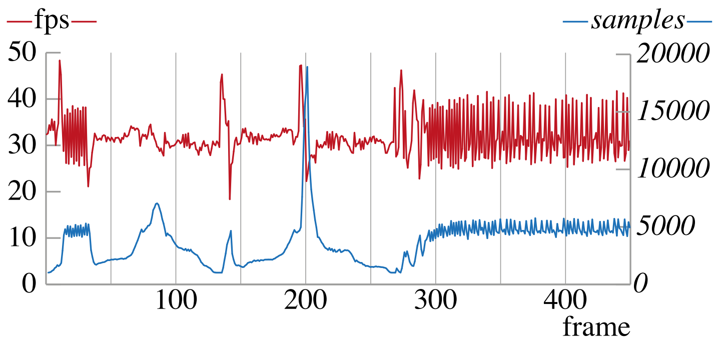


Figure 3.23: *Frame rate and batch size on a camera path with varying complexity.*

3.3.5 Discussion

Progressive Rendering is not new. But to the best of our knowledge, this approach is the first GPU implementation of progressive rendering using stochastic sampling to avoid false patterns in aliasing artifacts.

The approach is suitable to generate high quality images. This is especially useful when creating visualizations as in anatomy atlases, which introduce very high spatial frequencies at cuts. But also normal transfer functions often introduce high frequencies depending on their slope. This leads to disturbing artifacts using normal sampling. As can be seen in Figures 3.20 and 3.19, a combination of stochastic supersampling and interleaved sampling [KH01] turned out to be highly capable of eliminating image artifacts that relate to sampling. Besides image quality the proposed approach assures interactive rendering at 30 frames per second in almost every situation. This way it is much more convenient to explore different visualization techniques in combination, which can otherwise result in major performance issues. This is an important property when using the aforementioned shader framework.

3.4 Illumination in Curved Planar Reformations

Only a few deformations are used in practice in today's medicine. Above all, this is true for the Curved Planar Reformation [KFW*02] (CPR). The CPR is a sophisticated deformed visualization of vessels. It is capable of displaying information at once that is spread over hundreds of image slices, which is why it is successfully applied in diagnosis. It not only shortens the duration of diagnosis, but also improves its quality. In Arens et al. [AIB*11], the CPR visualization has been enhanced by techniques that enable depth perception. The content of the following section is mainly based on this publication and on the article [DA12].

3.4.1 Visualization of Coronary Arteries

The coronary artery disease is the most common cause of death in the world [FAF13], especially in industrial nations. Above all, two pathologies occur: the building of hard-plaque which narrows the diameter of the vessels, and thus, can lead to a severe chronic oxygen deficiency of the hearts muscles, and so-called soft-plaques, an early stage of hard-plaque, which can disrupt, and thus, cause a blockage. Both types of plaque can not be cured but can be surgically treated. To decide on an optimal treatment it is essential to find, identify, locate, measure the plaques, and to measure the oxygen deficiency. Medical imaging and the appropriate visualization of the data is the key for this decision.

A modern computed tomography scanner acquires several hundred images at a resolution of approx. 3 pixels per mm and a size of 512^2 pixels when imaging the human heart. See Figure 3.24 (left) for a single image slice. The analysis of the coronary arteries in this stack of images can be quite challenging and is prone to errors. Even trained physicians sometimes reach their limits when confronted with these images.

Differing types of visualization algorithms facilitate and accelerate the handling of this data. E.g. direct volume rendering, Figure 3.24 (right), supports spatial orientation and comprehension in this data. The course of the arteries can be inspected concerning the overall anatomy. However, in three dimensions there is occlusion, which is not a problem in two-dimensional visualizations. The so-called Curved Planar Reformation [KFW*02] (CPR) is a technique dedicated to the visualization of vessels and solves the problem of occlusion, see Figure 3.24 (middle). On top of this, the whole course

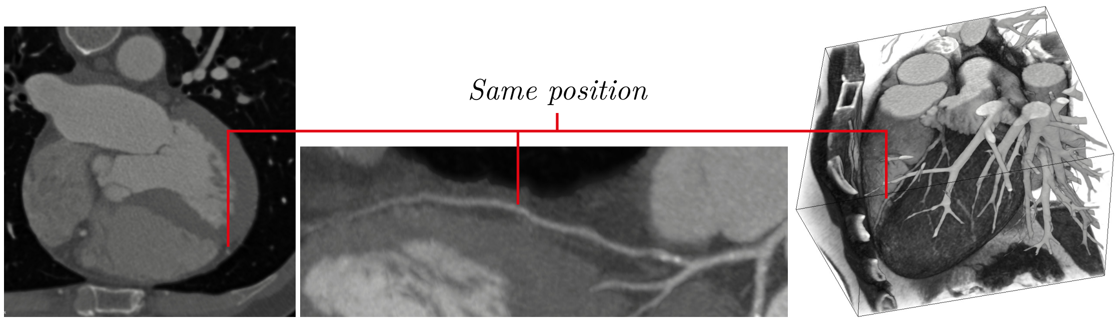


Figure 3.24: *Three visualizations of a dataset. The red lines mark the same position of a coronary artery in each visualization. Left: Slice View. Center: Curved Planar Reformation (CPR). Right: Direct Volume Rendering.*

of an artery can be inspected in one single image instead of being spread over several hundred images. That is, the amount of unnecessary information – everything that is not connected to coronary arteries – is reduced to a minimum.

As a result the CPR is a very effective and often used method to inspect the coronary arteries. Nevertheless, it is not perfectly expressive since spatial information is not visualized. This is due to the techniques used in combination with the CPR to determine the color of a pixel: Slice View and Maximum Intensity Projection (MIP). Both only use one single data value to determine the final color of the pixel depicted on the screen. To display spatial information using occlusion and illumination, multiple data values along the depth axis are necessary. Direct Volume Rendering (DVR) is one of the techniques that uses multiple data values along the depth. In the following three paragraphs and in Figure 3.25 the differences between Slice View, MIP and DVR are explained.

Slice View Considering the slice view, only data at a single slice through the centerline is being used to calculate the CPR image. Only few data values have to be sampled, hence, this algorithm uses less computational power compared to the others.

Maximum Intensity Projection (MIP) MIP is a very common technique in medical volume visualization. Several data values are sampled along a ray, but only the biggest is used to determine the appearance of the corresponding pixel on the CPR image.

Direct Volume Rendering (DVR) DVR uses all sampled data values along a ray to determine the appearance of the pixel. The over-operator is used to combine these values. It supports depth perception by implementing occlusion. Transparency and shadows can be used to modify the perception of depth.

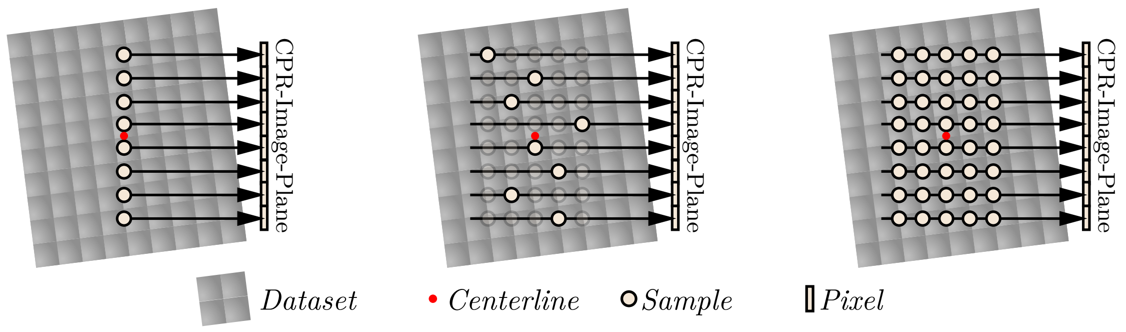


Figure 3.25: *Illustration of the sampling in a CPR using three different compositing techniques. The perspective is a side-view, i.e. the red dot marks the centerline of the artery which is perpendicular to the paper. The viewer of the CPR looks at this visualizations from the right side, where the image plane is. For each pixel of the image plane the bright dots on the viewing ray indicate at which positions the dataset is sampled regarding the used compositing technique. Left: In a slice view, only a slice of data is sampled. Middle: In a MIP several positions are sampled (translucent dots) but only one sample position contributes to the pixels color (the sample with the highest value). Right: In a DVR all sampling positions along a ray contribute to the color of a pixel regarding their transparency.*

3.4.2 Method

The combination of DVR and CPR enables depth-perception in vessel visualization. The key to the combination of both techniques is to treat the CPR as a deformation. The original algorithm by [KFW*02] is an iterative approach. To treat it as a deformation, the sampling of the volume data has to be bent in such a way that the picked artery is completely sliced by the image plane. To create such a bending, it is indispensable to know the positions of the centerline of the artery. These can either be set manually, which is very demanding, or calculated automatically. The automatic detection and segmentation of coronary arteries is a complex task. Noise, poor signal-to-noise ratio, motion artifacts and abnormalities complicate the detection. Schaap et al. [SMvW*09] provide a comprehensive overview and evaluation of current centerline detection algorithms. In this thesis the centerline is assumed to be given. Nevertheless, the quality of the centerline is one of the key factors for an accurate deformation of the arteries.

In CPR visualizations the arteries are never overlapping. This is not only a visual advantage, but it can be used to accelerate the algorithm. The deformation can be applied to the whole projection ray, instead of deforming every sampling position individually. That is, only the starting point and endpoint have to be deformed, the other sampling positions can be linearly interpolated. This would not be possible if succeeding portions

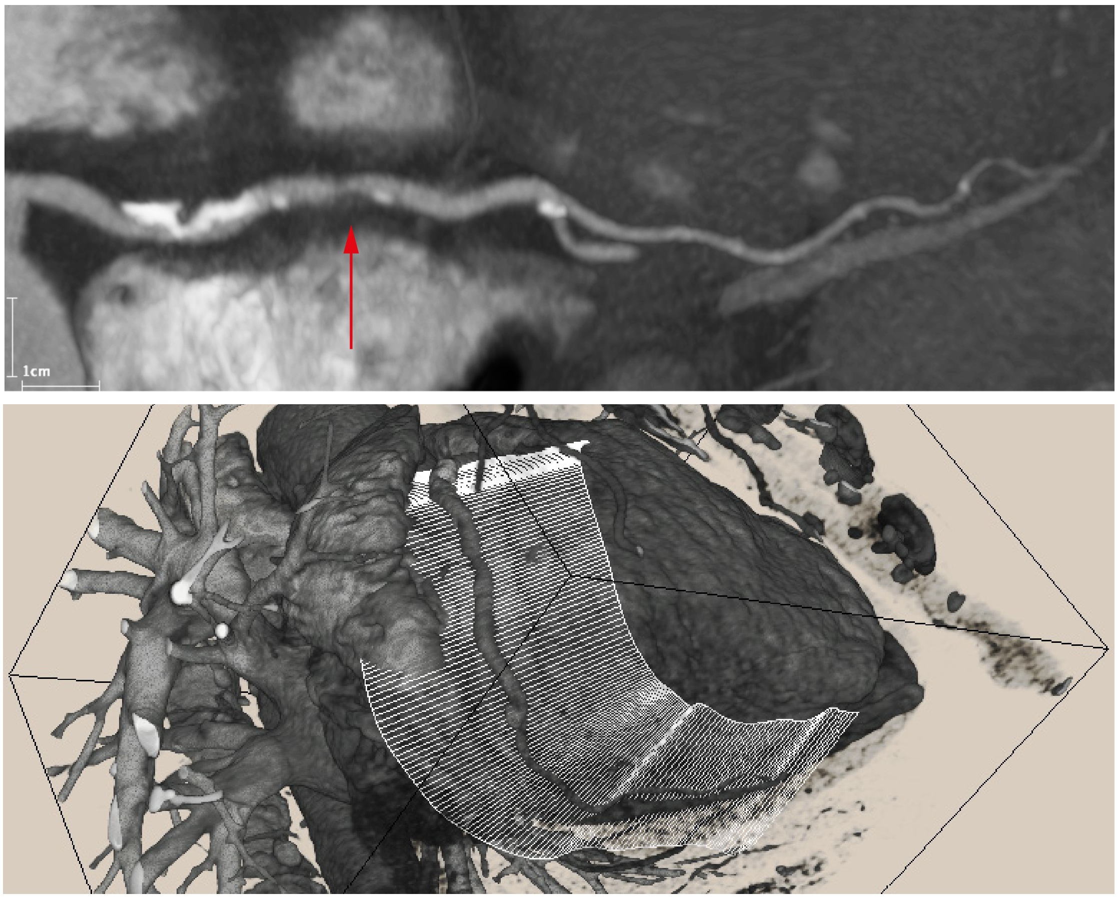


Figure 3.26: *The upper part of the image shows a CPR, the lower part shows the corresponding curved cutting plane (white). Bright spots in the vessel are hard plaques, the red arrow points to a soft-plaque.*

of the volume are deformed differently. To calculate the deformed starting points and endpoints of the projection rays in our CPR-node, a B-Spline is matched to the centerline and an up-vector is defined. Given that, it is possible to determine a "curved plane" which slices the artery at the centerline and orients itself depending on the up-vector. Figure 3.26 visualizes such a curved plane. The local normals to the curved plane are then used to determine the starting points and endpoints of the projection rays. The curved plane can now, interactively, be rotated around the B-Spline. On top of that, the number and position of the control points of the B-Spline can be used to interactively change the bending of the visualized artery (see Figure 3.27).

Treating the CPR as a deformation to the projection rays enables simple but flexible usage in the proposed shader framework. Nodes can be reused, and the flexible composing of techniques is also available. I.e. transfer functions, logical operators, and

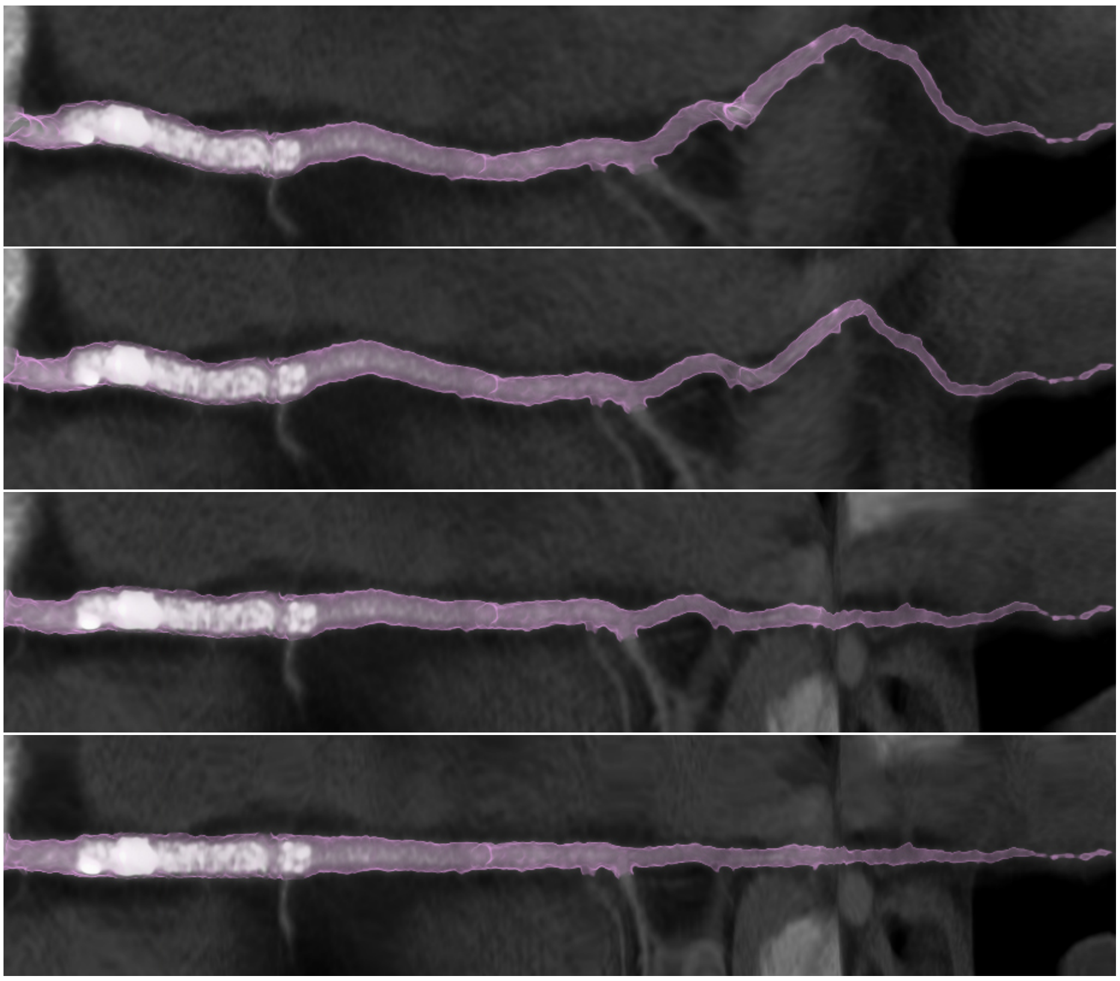


Figure 3.27: *Treating the CPR as a deformation in the shader framework, the degree of bending can be changed interactively.*

most illumination nodes can be utilized. On top of this, the nodes implementing the compositing techniques DVR, MIP, and MIDA (see Section 3.2.6) can be used, enabling the visualization to calculate occlusion. If the length of the rays is set to 0, the result is a slice view.

3.4.3 Spatial Comprehension in Curved Planar Reformations

There are several degrees of freedom and new possibilities when enabling the use of the flexible shader framework in curved planar reformations. Apart from logical combinations coloring, shading and transparencies can be added and modified (see Figure 3.28).

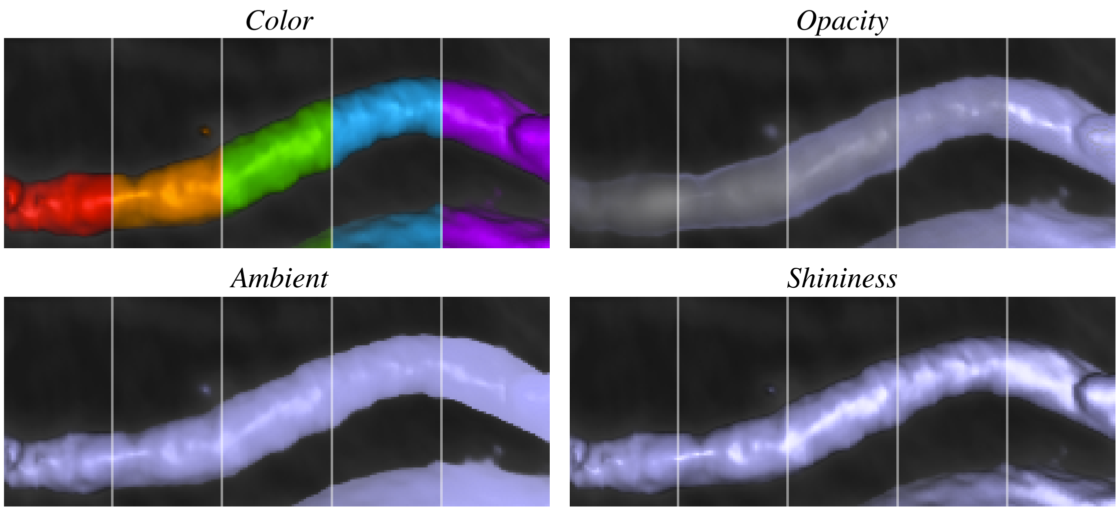


Figure 3.28: *Treating the CPR as a deformation, Phong shading and other nodes from the shader framework can be used in the CPR, too. Shown are different parameters of Phong shading in a CPR.*

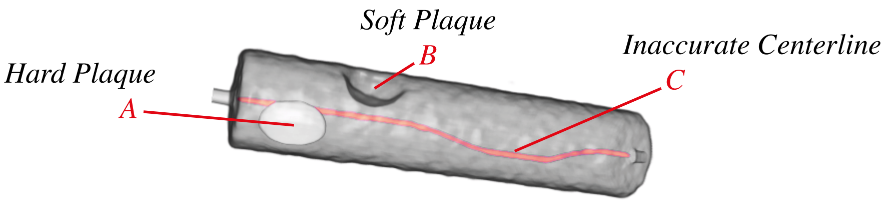


Figure 3.29: *This overview demonstrates the structure of the artificial dataset used for Figure 3.30.*

To assess the spatial structure of an artery visually, the visualization of a thin and very transparent shaded surface was tested. All other regions values are displayed unshaded – and thus, unmodified – and applied with a transparency proportional to the Hounsfield value. As a result, original data values can be used to find plaques and the visualization is quite similar to what physicians are used to. All advantages of the described visualization become apparent in Figure 3.30 using an artificial dataset (see Figure 3.29). At first, hard-plaques are visible at any rotation angle, which is an advantage compared to the slice view (see position A). Additionally, soft-plaques are visible, which is an advantage compared to the MIP (see position B). Furthermore, a spatial comprehension of the artery is possible, which is not the case in the slice view and the MIP. Compared to the slice view, the technique is less prone to inaccurate centerlines, which can result in a false narrowing of the artery (see position C). Finally, it is the only

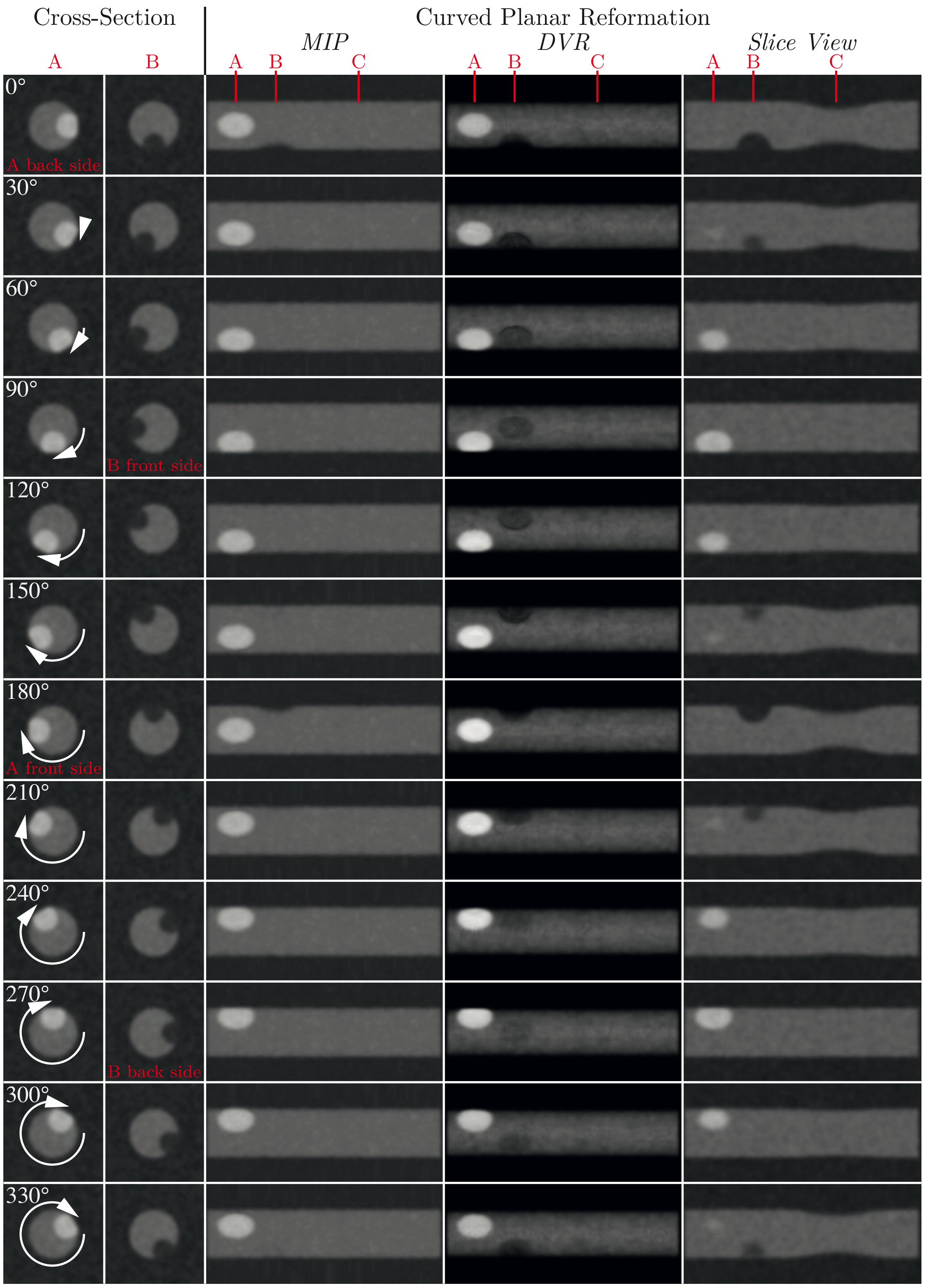


Figure 3.30: *Advantages and disadvantages of DVR, MIP and Slice View in a CPR demonstrated using an artificial dataset (see Figure 3.29). The cross-sections in the left columns show which plaque is in front in each row.*

visualization that uses all data values of the artery instead of presenting a selection. A weakness of the proposed method is the higher sensitivity to noise, compared to the MIP. Table 3.1 sums up these properties. These advantages are valid for artificial datasets, but Figure 3.31 suggests that the same is true for real clinic datasets.

	MIP	DVR	Slice View
Visibility of hard-plaques	+	+	∘
Visibility of soft-plaques	−	+	∘
Spatial comprehension	−	+	−
Use of all data values	−	+	∘
Resistant to inaccurate centerlines	+	+	−
Resistant to noise	+	∘	∘
Expense	∘	−	+

Table 3.1: *Advantages and disadvantages of MIP, DVR and Slice View in a CPR regarding the artery examinations (+ good performance, ∘ medium performance, − low performance). The medium performance for the slice view is based on the necessity to rotate the view.*

3.4.4 Conclusion

Treating the CPR as a deformation enables the simple usage with direct volume rendering and, as a result, illumination. This combines all advantages of both techniques, as the complete artery can be inspected within one visualization and spatial comprehension is established. The structure of the artery wall can be recognized much better, and vaulting and bulging become visible. Using MIP or Slice Views, some of these structure are either not visible at all, or the view has to be rotated. The early diagnosis of plaques, i.e. in the soft-plaque state, might be supported by this technique, which would diminish the risk of a heart attack.

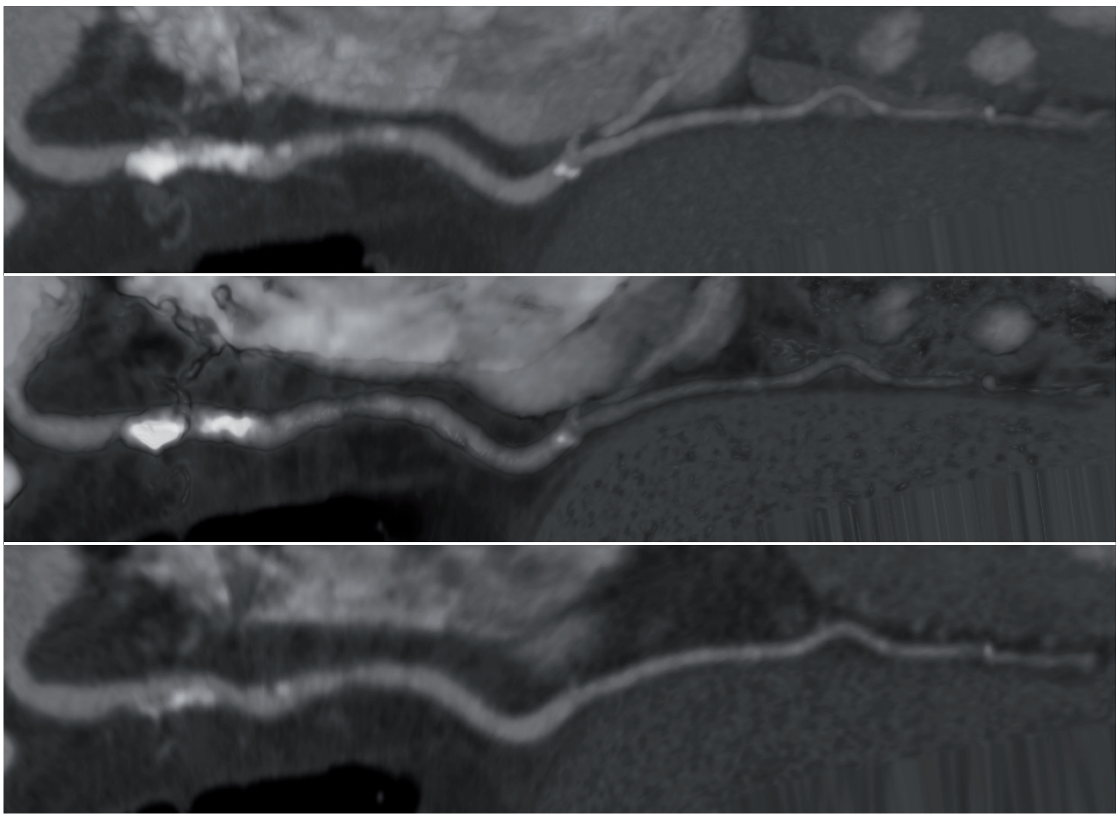


Figure 3.31: *Exemplary comparison of MIP (top), DVR (center) and Slice View (bottom) using a real dataset. DVR allows for an extended depth and structure perception.*

Chapter Discussion

In Chapter 2, a multi-volume rendering framework has been presented which provides an intuitive but flexible interface. It enables the user to combine the information of several volumes using transfer functions and logical operators. However, for creating illustrations as in anatomy atlases, the possibility of deformations, global illumination, and high-quality rendering was missing. In this Chapter 3, an alternative evaluation strategy was proposed to enable these capabilities within a shader-based dataflow graph for the first time.

Besides the necessary features, the changes to the system allow for a different view to the visualization task. In the previous concept the visualization task was to determine a color and a transparency for each position based on the data values at this position. I.e., in a scene with two volumes $A : \mathbb{R}^3 \to \mathbb{R}$ and $B : \mathbb{R}^3 \to \mathbb{R}$ the task was to define a function:

$$f(A(s), B(s)) \to \mathit{color} \times \mathit{opacity}$$

Note that A and B were sampled at the sampling position s. The proposed change to the evaluation strategy now allows nodes to use values at different positions than the sampling position in the dataflow graph. The task is now to define a function:

$$f(s, A, B) \to \mathit{color} \times \mathit{opacity},$$

where A and B can be sampled due to the demands. That is, additionally to local operators, *global operators* can be used to determine the appearance. This enables global illumination algorithms and deformations in the dataflow graph but also negative opacity gradients for correct Phong shading. The technical background, i.e. the execution of the dataflow graph on the graphics hardware is completely hidden from the user. Furthermore, a third level of editing possibilities has been inserted between the preprocessing and the sample processing, the ray evaluation. Eventually, rendering does not determine a color for a spatial position, but for each pixel of the image plane. But the compositing of a pixel's color from many successive sampling positions was fixed in the original concept. This was replaced by a completely free mapping

$$f(r_{start}, r_{end}, A, B) \to \mathit{color} \times \mathit{opacity},$$

where r_{start} and r_{end} denote the starting and the end position of the ray through the according pixel. The user is now supported solving this complex task by compositing nodes. These nodes solve this task by iterating along the ray and give the user the aforementioned easier task per sample position. Typically, the user wants to use one of the compositing techniques DVR, MIP, or MIDA and specify only the innermost dataflow graph per sample position. However, it is possible to combine several compositing techniques using logical operators, just like explained at a per sample basis. This enables image-level intermixing as in Voreen. On top of this r_{start} and r_{end} can be manipulated by nodes in the dataflow graph in exactly the same demand-driven way as the sample position can be changed. The presented CPR visualization node uses this abstraction level, too. As a result, most of the nodes known from normal volume rendering can be used in CPR visualizations, including illumination, direct volume rendering, and logical operators. Also, a magnifying glass effect as proposed in [WZMK05] can be realized in a node. The different abstraction levels become visible in the user interface, too. Subgraphs are nested in parent nodes that introduce a hierarchy of abstraction levels. As a result nodes from different abstraction levels in the dataflow graph can not be interconnected incorrectly. It prevents the user from using data in a wrong scope in a simpler way compared to textual programming. On top of this, the nesting nodes encapsulate all necessary loops and parallel executions. As a result, loops and parallel executions are not demanded from the user of the visual programming language, which simplifies its usage tremendously.

Global operators are computationally intensive by nature. Hence, a progressive rendering algorithm was proposed that supports interactivity even in extremely complex visualizations. At the same time, it provides stochastic supersampling to gain a much higher image quality. This way, a flexible try-out is not restricted to visualizations that naturally run in real time, and expensive visualizations do not freeze the application, but can be investigated interactively.

CHAPTER 4

Visualizing Dissections of the Heart

The intense combination of the techniques presented in the previous chapters can be used to fulfill the technical demands of many illustrations that can be found in anatomy atlases. To test the suitability of the proposed dataflow concept, one goal in the next two chapters, is to imitate illustrations from Netter's "Atlas of Human Anatomy" [Net10] using CTA volume datasets. This chapter details on the realization. As mentioned in the introduction, the acquired data is not sufficient to imitate such illustrations. To create real-time volume visualizations, additional information has to be integrated into the shader framework in a processible and efficient way. This integration is the topic of Section 4.1. The practical usage of this data is the topic of the next three sections, i.e. Section 4.2 describes the practical implementation of different tissues, and Sections 4.3, and 4.4 explain the practical use of cuts and deformations or illumination, respectively. Results can be found in the next chapter.

If possible, the reader may now have a look into the Netter's "Atlas of Human Anatomy" [Net10] or Thieme's "Atlas of Anatomy" [SSS09] to get an idea of the desired results. If this is not possible the following listing contains links to the three most important illustrations in a watermarked version. Unfortunately, the publishing of these images in this thesis is not possible.

- http://www.netterimages.com/image/730.htm
- http://www.netterimages.com/image/704.htm
- http://www.netterimages.com/image/738.htm

4.1 Additional Information using a Model of the Heart

Computer-based visualization of medical volume data poses a big challenge, because the acquired data is limited and contains specialized information. Different parts of an organ often exhibit the same data value, hence, it is impossible to imitate illustrations in anatomy atlases only based on this data. For instance, it is not possible to extract the reddish and bluish hue of the arteries or veins, respectively. To improve this ill-posed situation the data is often enhanced with additional information. One simple approach is to assign a tag to each voxel that indicates the matter it belongs to. In volume rendering this approach is called classification. Many different algorithms have been presented that try to help classifying volume data. Anyhow, the problem with tagged voxel data is, a voxel is either tagged or not. It is a binary decision that does not cope with uncertainties. To overcome this issue, statistical algorithms have been presented that compute a fuzzy classification. If two different matters shall be visualized differently a fuzzy classification is a good data basis. However, if cuts shall be placed according to anatomy atlases, information about the right positioning of the cut is still missing.

An approach to solve both problems is the creation of a model that represents the organic structure of the heart. Several whole heart models have been presented [ZBG*07, IVM*10, WPM*10]. The advantage of a model is, any information that holds for all hearts can be added to that model *once*. If the model is then deformed congruent to the dataset, this information can be retrieved at the right position. Finding a congruent deformation is called registration and is an active field of research in computer vision. Results of a registration can yield a valuable precision (less than 0.67 mm point-to-mesh error in average [WPM*10]). In this thesis, registration is not further discussed and assumed to be given. Finally, using this congruent deformation function the additional information needs to be deformed as well. In this thesis, an elastic thin-plate spline deformation function is used to deform information as exactly as possible. Basic information about this procedure will be given in Subsection 4.1.1. Subsection 4.1.2 presents an overview of possible information types and their usage from a computer scientist's point of view.

At this position, the reader might come to the question, why is the original data necessary for the visualization. If the model of the heart and its co-registration is very precise, the model itself could be displayed. This might be true for morphologic datasets like a CTA scan, however, at least in a PET scan the underlying data itself needs to be visualized. And on top of this, at the time this thesis was written, it was not possible to create a precise model of the heart including all individual anatomic details, like papillary muscles, fat tissue distribution, and vessel courses. To take advantage of a model despite the imprecision, a similar approach to the statistically quantitative volume visualization by Kniss et al. [KVUS*05] is chosen. Based on the imprecision the model exhibits, the distance to a classified structure can be used to define a probability for a sample position to belong to that classified structure. Finally, this distance based classification is used for a probability weighted assignment of visual properties as in [KVUS*05]. Although no precise classification is done, most of the ambiguities can be ruled out this way.

4.1.1 Elastic Co-Registration

As described in the introduction to this section, the deformation function, given by the co-registration of the model to the CTA data, is assumed to be given. Nevertheless, a few key data shall be given in this subsection to provide the reader with a better understanding of the overall procedure. The model, that is used in this thesis, is a polygonal mesh that consists of 1021 vertices. To eliminate individual peculiarities, the model was averaged using 11 CTA datasets. It was created within the project group "Volume Rendering" and the Master's thesis by Sindelar [Sin12]. Figure 4.1 shows this mesh and its individual parts. The corresponding deformation function is provided in the form of a table that links the position of these 1021 control points to the position in the target dataset. The precision of the deformation function can be expected to be within a point-to-mesh distance of 0.77 mm. However, 1021 control points are not sufficient to describe the complex and fine structured details of the valves or the right atrial appendage. Fat tissue, arteries, pericardium, or other surrounding tissue is not reproduced.

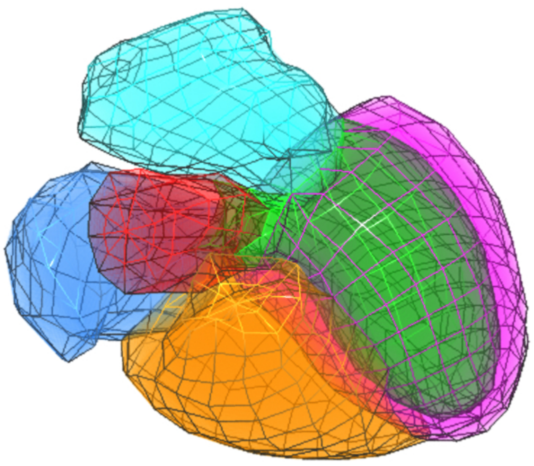


Figure 4.1: *Model used in this thesis for co-registration.*

To make the deformation function usable for additional information such as a fuzzy classification in volume rendering, it is important to know the mapping for *every* spatial position to the target dataset. Hence, positions in-between are interpolated. For this a GPU-based radial basis function interpolation is used that is based on thin-plate splines. Using this continuous deformation function any spatial data, including volumes, can be deformed and rerasterized using a preprocessing node in the dataflow graph. Details can be found in Kokerbeck [Kok12]. The nodes marked by the green dashed line in Figure 4.4 in Section 4.2 will show the application of this elastic deformation.

4.1.2 Data-Structures and Usage of Additional Information

Additional information in volume rendering can be of different kinds. This not only includes additional volumes, additional axes, spatial positions, or vectors, but also simple scalar values. Table 4.1 shows common data structures that are often used in volume rendering.

Data structure	*Characteristic*	*Common usage examples*
Volume	scalar	Are used as masks, fuzzy classification or shadow volumes. Interpolation possible.
	nominal	Are used as masks. No interpolation, but memory efficient, when using several categories.
	vector	Are used for colors and material properties but also for complex deformations. Interpolation possible.
1D/2D image	scalar	Are used as lookup-table e.g. for specifying environmental light or style transfer functions.
Point data	scalar	Are used as parameter, e.g. for specifying transfer functions.
	vector	Are used as parameter, e.g. for positioning of cutting planes or lights.
Functions	-	Simple to evaluate functions are used for cuts and deformations.

Table 4.1: *Common data structures in volume rendering.*

Primarily, data structures that use direct addressing are used because this is the fastest method to access data in computer science. This is necessary because several millions of samples have to be evaluated per frame in direct volume rendering. Sometimes other data structures can be found in direct volume rendering, but most of them are ruled out due to performance issues. For example, a polygonal mesh is a very unsuited type of data. If sample positions must be evaluated to be inside or outside of a polyhedron, a lot of planar equations have to be evaluated. If enough memory is available, the decision of a position being inside or outside a polyhedron is better discretized on a regular three dimensional grid (i.e. volume) in preprocessing. This way the information can be addressed directly.

Model-based Transfer Functions As described in Chapter 2, a transfer function is defined by the domain, the type of the mapping, and an explicit specification. The stated additional information can be utilized in the domain and the specification.

As described in Section 2.3, *distance* is a well suited metric that can be used in combination with the data value in a two-dimensional transfer function. It allows for specifying different colors and transparencies dependent on the data value and distance to a structure. Additional information, in the form of a point or a plane, can easily be used to calculate the distance of a sample position to this structure during runtime. A transfer function can then be used to define transparencies along an axis. Figure 4.2 shows such an example, which is also part of the Netter [Net10] and "Prometheus" [SSS09]. Regarding the heart, there are also two defined axis – the so called "long-axis" and the "short-axis" – which can be used to define cuts. Corresponding figures can be found in Chapter 5.

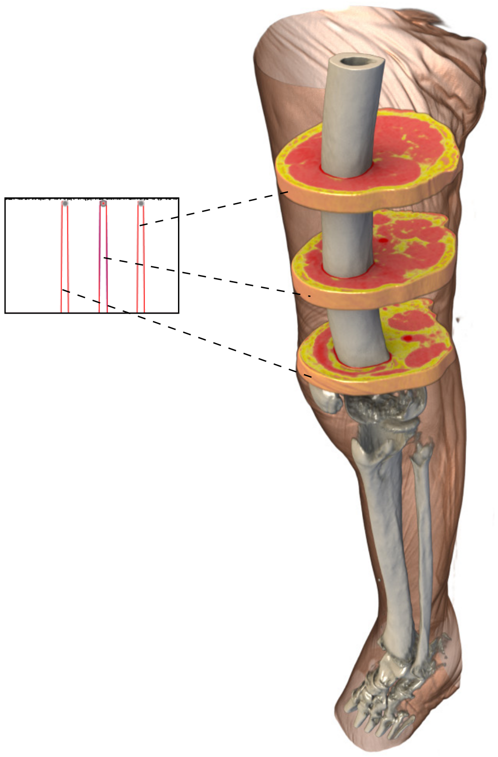


Figure 4.2: *Distance to the xy-plane used in transfer function to define opacity. Image level intermixing (see Subsection 3.2.6) is used to depict the skin behind the inner structures. Dataset "KESKONRIX" taken from the Osirix sample image sets [Osi14].*

When using distances computed to classifications done by a model, such transfer functions will be named model-based transfer functions [FLAK11] and can be used in a completely different manner, e.g. the distance to the left atrium and left ventricle can be used to color deoxygenated blood bluish and oxygen-rich blood red. Additionally, fuzzy regions can be defined. This comes with several advantages. First of all, surrounding tissue that occludes the heart can be faded out depending on the distance. This in turn allows for a more precise specification of the data-based dimension of the transfer function, instead of leaving a safety margin to the data values of the otherwise occluding tissues. Finally, regarding the dataflow-based shader concept presented in this thesis, different shading subgraphs can be utilized for different regions. This allows for a different processing of the individual parts of the organ. In the next subsection, this concept will be massively used.

4.2 Colorization and Material

Colors in anatomical illustrations relate strongly to the real appearance of the tissue. Hence, most colorizations are consistent between the atlases. Muscles are red, conjunctive tissue is white, fat tissue is yellowish, arteries are red and veins are bluish, which corresponds to the natural appearance. However, for a better discriminability the saturation is considerably increased in most atlases. The pulmonary artery is purple in Netter's atlas [Net10]. This is assumedly motivated by the fact, that this artery, which should be red, transports blood poor in oxygen, which is typically connected to the color blue. Other atlases, as the “Prometheus” [SSS09] paint this artery red. The right atrium is also painted in a purplish red in Netter's atlas, which can not be recognized in “Prometheus”.

Material properties are also strongly related to the real appearance, e.g. fat tissue is quite shiny, while muscle tissue is less shiny, but has an anisotropic gloss parallel to the muscles fibers in anatomical illustrations.

Color and material properties suffer the most from the ill-posed situation that tissues can not be differentiated in a volume dataset. Similar, or even equal, Hounsfield values can make the distinction very difficult or impossible. However, even if the values differ, noise heavily influences the detection of the tissue's boundaries. In reality, the surface of the ventricle's muscle is quite smooth. In a volume dataset, surfaces are only implicitly defined. That is, positions on the one side of a surface exhibit a different value than positions on the other side. For example, a transfer function can make this transition visible by assigning an alpha > 0 to the data value averaged from the values of both sides. Given an infinitely high sampling rate this would result in a iso-surface. If the data values are now distorted by noise, this iso-surface would translate depending on the local minima and maxima of the noise. That is, they depart from the original position of the surface. Since noise is, by definition, not systematical, the original surface cannot be perfectly reconstructed. The lower the signal-to-noise ratio is, the more the detected surface will translate from its original position. As a result, the surface between two tissues that share a similar Hounsfield value (e.g. fat and muscle) is more error-prone to noise than a surface between two tissues that have more differentiating values (e.g.

muscle and bones). Hence, the roughness of a surface does not only depend on the noise but is essentially dependent on the signal. Figure 4.3 illustrates this relation of roughness at an artificial dataset containing two arches that exhibit a different signal and thus a different signal-to-noise ratio.

In this thesis, Gaussian filters have been used to reduce noise and thus to flatten surfaces. Of course this introduces a reduction of the frequency that can be detected.

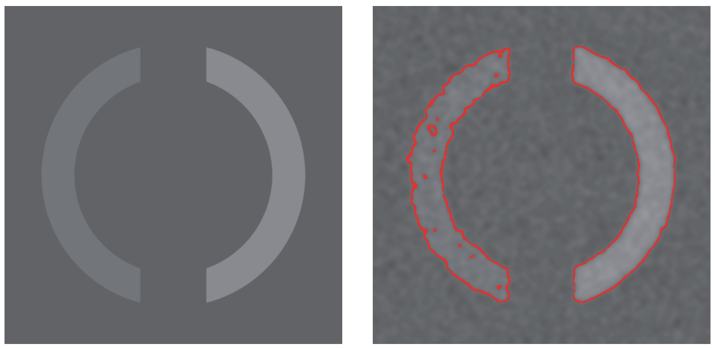


Figure 4.3: *Relation of iso-lines and signal-to-noise ratio. Left: Undistorted data in two contrast graduations. Right: Distorted values and iso-lines. The same noise and blur results in a different roughness of the iso-lines.*

Another difficulty is the differentiation of tissue in terms of color if they share a range of Hounsfield values. Some details can be distinguished by adding another metric to the transfer function like lung vessels and coronary arteries (see Section 2.5). Figure 4.4 shows seven different transfer function specifications (red dashed line) that were used to visualize the tissues differently at the heart dissections in the next chapter.

However, some details like the reddish and bluish coloring of arteries and veins necessitate a classification. In this thesis, distance metrics generated using the heart model are used within transfer functions to manage this differentiation. Although distance-based transfer functions are the technical mean to specify this differentiation, from a theoretical point of view it is a fuzzy classification. Six overlapping fuzzy classifications have been necessary to distinguish the different types of tissue (Figure 4.4, green dashed line). Seven distance-based transfer functions use this information to narrow the impact of the aforementioned transfer functions (Figure 4.4, orange dashed line) as proposed in the probability weighted assignment in [KVUS*05].

Some objects in hand-drawn illustration are not visible in the data at all. For example, the coronary veins are only seldom visible in CTA data. The same is true for the inferior vena cava which enters the right atrium at the lower right, back side of the heart. The reason for both is the intentionally unequally distributed contrast agent, which is injected to highlight the more important arteries (see Subsection 2.2.1). Hence,

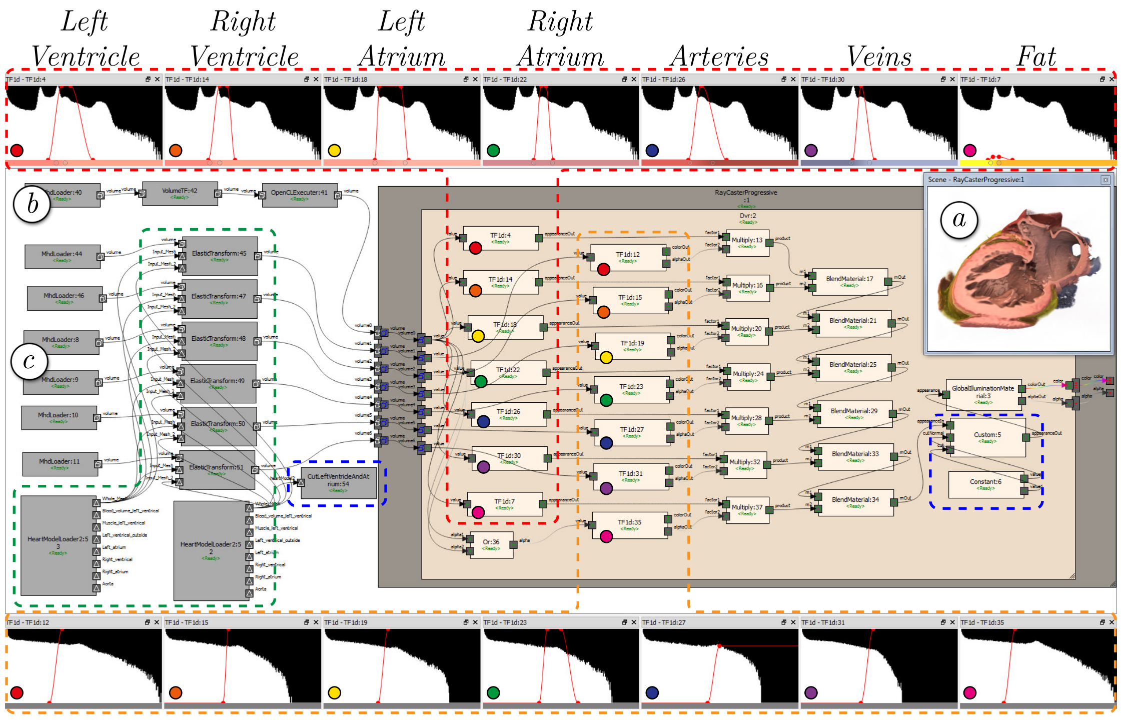


Figure 4.4: *Complete dataflow graph for creating a crosscut through the left ventricle and the left atrium (a), built up from basic nodes. (b) The CTA-dataset is loaded and preprocessed by a subtle gaussian filter to reduce noise. The red dashed line marks the transfer functions that determine the appearance of the distinct types of tissue. The green dashed line marks the application of the deformation function to the additional information (c). The result is a fuzzy classification that is used in distance-based transfer functions (orange dashed line) to narrow the impact of the aforementioned transfer functions. The blue dashed line marks the nodes that use the deformation function to automatically set the cutting plane. In later visualizations, several nodes will be grouped.*

those veins can typically not be visualized. Similar to that, very small or thin details are not visible in CTA datasets, too (e.g. third degree coronary arteries, tendinous chords). Even the actual wall of the great veins and arteries can not be seen often. However, a major ramp is present at these positions, which serves well for visualization.

Material Properties in the Shader Framework Material properties are specified by many parameters. The traditional Phong reflection model uses four parameters for ambient reflection, four parameters for diffuse reflection, four parameters for specular reflection and a thirteenth parameter for specifying the shininess. To determine the values of all these parameters using the proposed port types of the shader framework requires many connections between the nodes. To simplify this, a material-type was added to the available ports, which encapsulates all these parameters in a struct and shows a circle ∘ as symbol. Besides the traditional parameters a superordinate alpha channel was added. This channel can be used to directly weight materials depending on the distance-based fuzzy classification. Figure 4.4 uses material structs and weighting massively.

4.3 Deformations for Dissections of the Heart

In hand-drawn illustrations of the human heart, cuts are used to hide surroundings, to illustrate cross-sections, or to uncover hidden parts. Simple planar cuts as well as complex cuts can be necessary. Deformations can enhance visualizations with cuts by preserving the context when uncovering hidden parts. While deformations of the real heart follow complex rules of physics, we noticed that most of the illustrations can be imitated by a combination of three basic elastic deformations and rigid transformations.

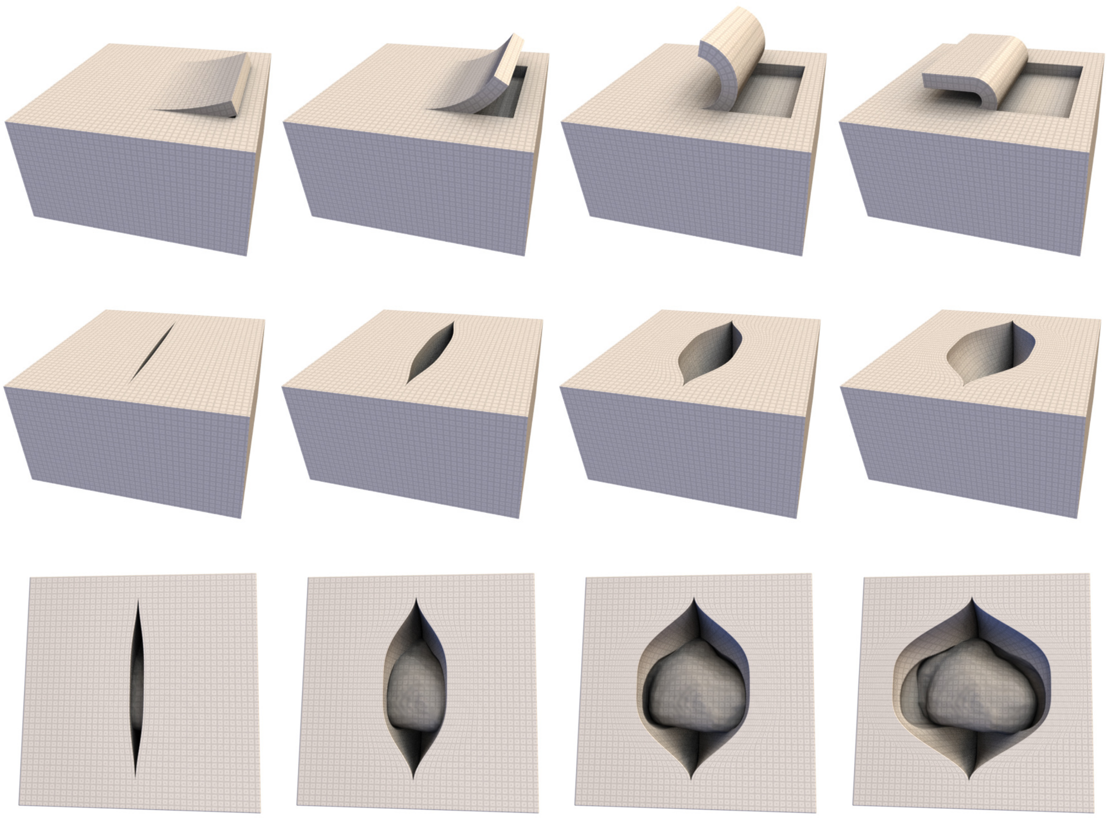


Figure 4.5: *First row: Different stages of a masked peel deformation. Second row: Different stages of an opening deformation. Third row: Opening deformation with heart-shaped mask applied. Note the inverse-heart-shaped recess in the volume.*

The top row of Figure 4.5 shows the most used elastic deformation. When applied to non-planar material it produces a lot of stretching and shrinking. This is why it often has to be combined with a preceding bending deformation making the matter planar and a following inverse bending (Figure 4.6). Finally, there is an opening deformation, see Figure 4.5 (second row). A mask is used to define cuts and the region of influence 4.5 (bottom and top row). The masks are not nominal but scalar data, as a result, blending can be used to eliminate artifacts at the transitions of the masks [WEE02, WEE03].

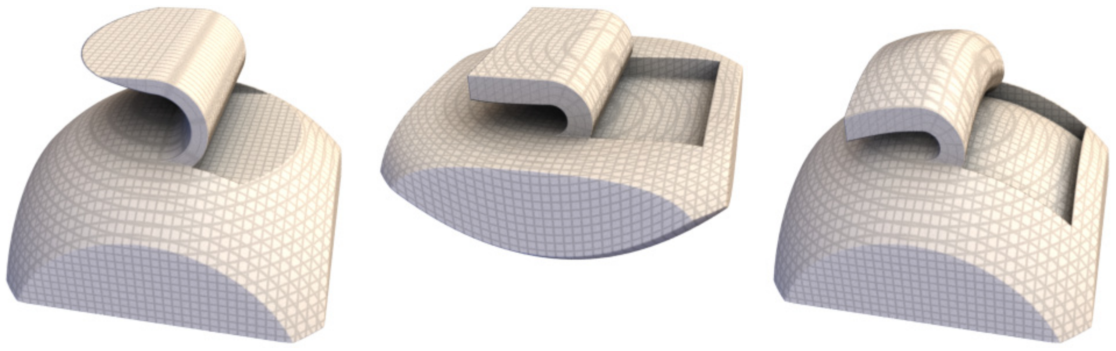


Figure 4.6: *Left: Masked deformation applied to a volume dataset containing a hemisphere. Middle: Same dataset with a second previously applied deformation Δ to shape the data in a more planar form. Right: A third deformation, inverse to Δ is applied afterwards.*

In the literature, there are many sophisticated ways to realize deformations. However, the deformations listed here can all be implemented inversely in rather simple formulas (see Birkeland and Viola [BV09]). This means the degree of deformation can be bound to a simple variable and the deformation can be animated smoothly in real-time without preprocessing. See Figure 4.5 for four different degrees of deformations. Nevertheless, other inverse deformation techniques are perfectly suited to be used in the proposed shader framework.

The specification of deformations is elaborate, because many parameters have to be set. First, the scope of the deformation, including cuts, has to be set. Second, the main axes of the deformation need to be specified. Finally, the degree of deformation must be set. Using several concatenated deformations, this procedure can be quite demanding. However, when creating atlas visualizations, the user typically just wants to set the last parameter, the degree of deformation. The other parameters are complex and should be prepared by a specialist. Since these parameters and coordinates vary from dataset to dataset, the deformation function mentioned earlier must be used to deform these parameters to individual datasets. Thus, the precision of the cuts and deformations depends on the precision of the deformation function.

4.4 Illumination and Shadows

Light and shadow is very important in hand-drawn illustrations. They support the perceptibility of smallest structures and are a crucial cue to depth in two-dimensional images. Especially the complex structures in the hearts' cavities benefit from this. For example the structure of papillary muscles and the neighboring trabeculae is very hard to understand without shadows.

Figure 4.7 shows a crosscut through the ventricles to depict the difference between no illumination, Phong shading, and global illumination. To establish a beneficial per-

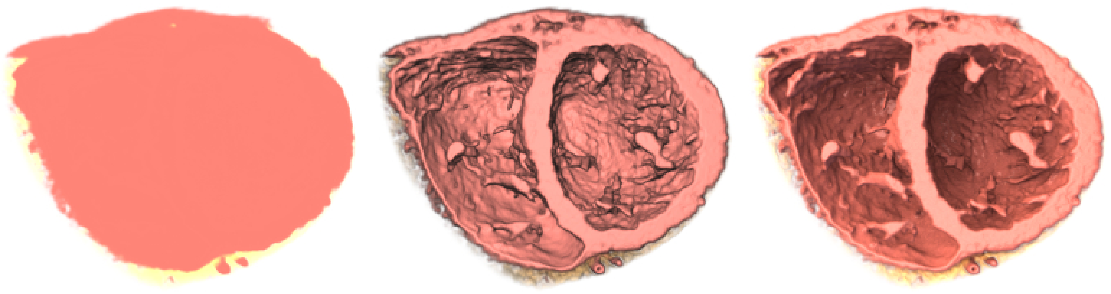


Figure 4.7: *Left: Only diffuse color without shading. Middle: Phong shading. Right: Global illumination. Compare with "Prometheus" [SSS09].*

ception of all sides and structures of the a heart, hand-drawn illustrations depict a rather complex lighting situation with several diffuse area lights, were only frontal light casts shadows. First, this prevents structures to be in very dark areas, which is beneficial for visibility. Second, there are no shadows from far away occluders which can accidentally be mistaken for a structure of the depicted organ. Such a misinterpretation can happen if an unwise or wrong lighting situation is established. In a study which is part of the publication of Domik et al. [DATS12], the depth perception using local Phong shading was compared to the depth perception using global illumination. Participants of the study were asked to choose from two marked points, which one is closer to the viewer. The same situation was shown using both illumination setups. Surprisingly, there were a few pairs of images where the estimation of the global illumination performed worse than the estimation using Phong shading. Figure 4.8 shows such an image pair. Assumedly, the shadow casted by the rib onto the ventricle was misinterpreted as a wrong bending of the ventricle's surface.

Hence, the light sources casting shadows should be rather in front of the object, to ensure shadows being near to the occluder. Regarding rather frontal lighting, hard shadows are not, in every case, a good estimator for depth. Soft shadows on the contrary vanish and blur depending on the distance, even when using frontal lighting.

Whereas other modular systems we found in the literature are data-driven and invest

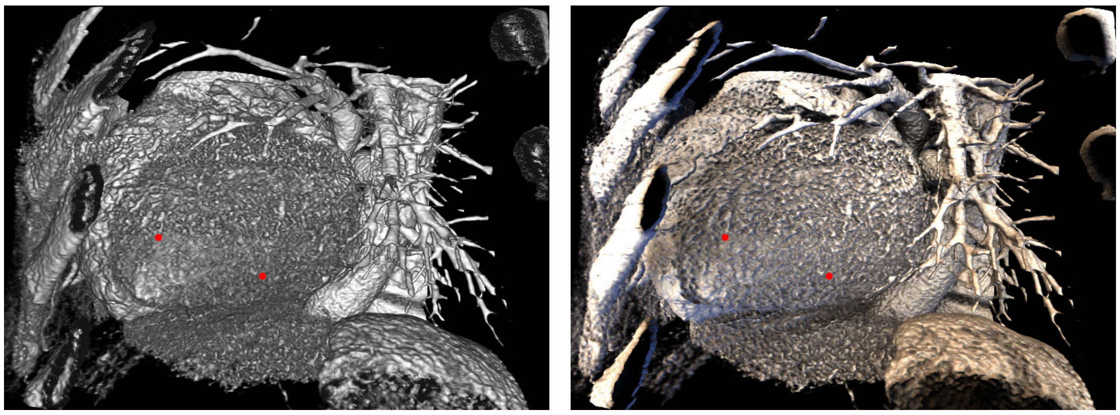


Figure 4.8: *Visual user study from [DATS12]. The task is to determine the red dot, that is closer to the viewer. Left: Phong shading. Right: global illumination. In this image pair, global illumination was outperformed by Phong shading.*

a rather big effort to compute shading and shadows correctly. This ability is inherent in the proposed demand-driven dataflow-based concept. When sampling along a shadow ray, all preceding nodes are evaluated, i.e. it is always the true opacity that is used to compute shadows and gradients for shading. Figure 4.9 shows three different deformed volume visualizations that only use transparency and illumination to depict shape and structures, i.e. all matter is purely white and shades come exclusively from illumination. Compared to precomputed gradients and more sophisticated shadowing algorithms the proposed approach introduces an increased computation time. However, the demand-driven evaluation allows to *not* evaluate shading and shadows if the sampling position is transparent. This makes the rendering time dependent on the number of visible samples, which is the bottle-neck of every volume rendering. However, in combination with the proposed progressive rendering (see Section 3.3), rendering is fast enough to enable the user to interact and parameterize all displayed scenes of this thesis. In other applications, where speed is of bigger importance than quality, half-angle slicing or even screen space shadows might be a better solution.

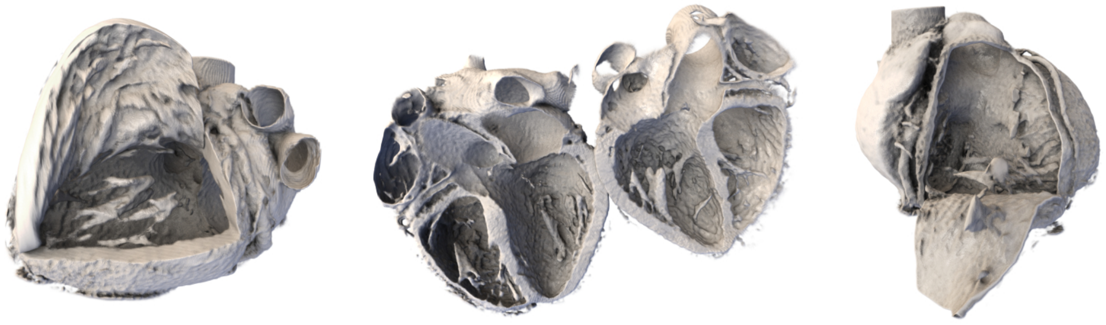


Figure 4.9: *Three volume visualizations of a CT-angiographic dataset dissected and illuminated according to "Atlas of Human Anatomy" by Frank H. Netter [Net10]. Inner details, like the papillary muscles, can be easily perceived.*

Chapter 5

Results and Evaluation

McCall [McC77], Boehm [Boe78] and the ISO/IEC 9126 [ISO01] give a lot of software characteristics that indicate software quality. On top of that, usability concerns can be checked using the heuristics in ISO 9241-110 [ISO06]. The mentioned heuristics and characteristics were not especially devised for visual end-user programming, and that introduces some extra demands to the user and the system. For example, the user is kind of a programmer, and internal software quality metrics, like reusability, become important for the user too. Nevertheless, many criteria could be investigated or adapted to evaluate the system. However, this evaluation will not try to guarantee that the proposed framework is a "good" software. Actually, only a few criteria answer the question if the proposed dataflow concept and the presented implementation fulfill the aimed goal to support "a wide variety of visualization techniques while making distinct techniques rapidly reusable in combination with others" (see introduction, Chapter 1). These criteria are *reusability, extendability, understandability, functionality* (including suitability), and *efficiency.* While these five criteria build the subsections of Section 5.2, it has to be mentioned that the visualization results of this thesis are a strong indicator for functionality and suitability. Despite that, the visualization catalogue is a very important part of this thesis. Hence, it will be presented in its own Section 5.1. It imitates the heart illustrations in Netter's atlas. The visualizations presented are not only meant to ease the communication process between radiologists and therapeutic physicians, but also the communication to patients. However, not all illustrations, that can be found in the atlas, can be recreated using the proposed techniques. Hence, limitations are pointed out and difficulties are discussed.

A comparison test of different software systems in this field of application might have given some extra information on the capabilities of the proposed framework compared to others. Anyhow, in terms of the number of supported features, other systems outperform Volume Studio due to different amounts of man-power in the development. On the other hand, the visual rapid prototyping of deformations, illumination and logical combination of data is a novel and original technique in Volume Studio. Thus, other systems do not support that. Hence, a fair comparison test is difficult to conduct.

5.1 Visualization Catalogue

Dissections in anatomic heart illustrations are an effective visualization technique and support a variety of applications in exploration and communication. In this section, a set of dissections found in anatomy atlases of the human heart are imitated based on individual patient data. This allows radiologists, therapeutic physicians, and surgeons to compare, explore, and discuss volume data in view of atlas illustrations and to ease the communication with patients. Especially realistic intra-cardiac views are new to clinic routines and allow for a better understanding of the individual complex structures of the heart's cavities. The main challenge for these illustrations is the flexible real-time combination of various cuts, (geometric) deformations, and shadows (influenced by the deformations and cuts) to depict shapes and structures, as well as the distinct treatment of data dependent on additional information. A possible technical solution to this has been presented in Chapters 3 and 4. The effectiveness and applicability of the approach will be shown by imitating illustrations of heart dissections taken from an anatomy atlas using the patient's individual volume data. This section is mainly based on the results in [ABD13b] and [ADW*14].

All Figures in this section imitate hand-drawn illustrations of anatomy atlases. As mentioned earlier, hand-drawn illustrations are quite expensive. Unfortunately, this is also true for publishing these illustrations in scientific publication for the purpose of comparison. Wherever possible a direct link to a watermarked version of the image is given instead.

Figure 5.1 shows the an exterior view to the heart. It is revealed and colorized using the heart model. All following visualizations use this revealed colorized view.

5.1.1 Cuts

Figures 5.2 and 5.4 show simple planar cuts along the long axis and along the short axis of the heart. Nevertheless, the cuts reveal insight into anatomy's details that couldn't be noticed without. Like all other images too, these visualizations are based on individual CT angiographic volume data. Figure 5.4 shows slices that are oriented exactly the same the cuts, so the correctness and the correspondence of the visualizations can be judged by comparison to the underlying data. The individual visualizations in Figure 5.4 can be automatically generated for individual datasets using the model-based deformation function. The preprocessing takes 75 seconds to deform the fuzzy classifications and to place the cut. Initially, the cut will always be at the same position in the individual hearts, i.e. it will go through the same positions of the left atrial appendage, the left

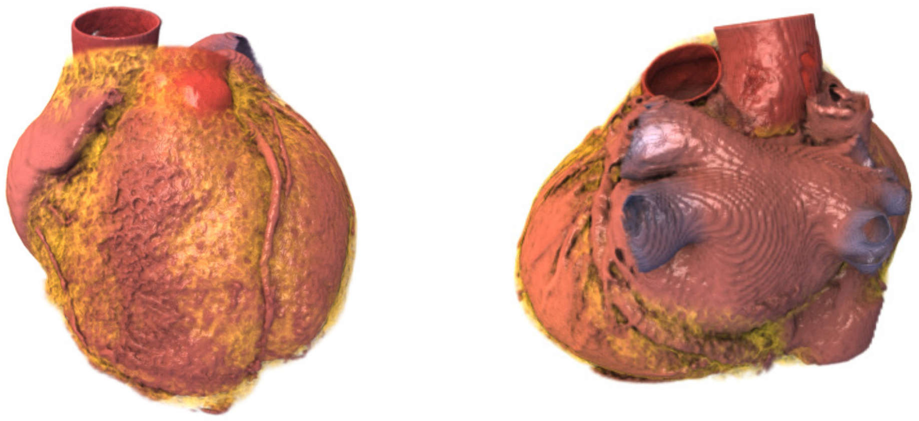


Figure 5.1: *Colorization applied to revealed heart. Compare with hand-drawn illustration* `http://www.netterimages.com/image/752.htm`

atrium and the left ventricle in Visualizations 1 of Figure 5.4. However, the user of the interactive visualization has the freedom to change the cut's position and orientation.

The correct shadowing of these images is of major importance to depict the individual structures and shapes. Especially the papillary muscles and trabeculae are clearly articulated. Chen et al. [CMA04] concentrate on the visualization of these structures. However, they use Phong shading to emphasize the shapes and structures, which makes the structures much harder to perceive than in our images.

Non-planar cuts are possible too, by saving cut masks with the heart model. In the next subsection Figure 5.3 uses a non-planar cut.

5.1.2 Rigid Deformations

Figure 5.3 shows a rigid deformation. The cut is not planar, but curved, so it crosscuts the four cavities and vessels of every individual heart at the same positions as in the hand-drawn illustration by Netter. The rigidly deformed half of the heart is laid besides the other half. As a result, context is preserved and the whole heart can be inspected from the inside.

5.1.3 Elastic Deformations

Figure 5.5 shows three elastic deformations. The first and the second image make use of several concatenated deformations to minimize stretching and shrinking. The third image uses an opening operation as in Figure 4.5 (second row). The degree of the deformations can be interactively modified from a closed heart to the state depicted in the images. The cuts are non-planar in the first and the second image and are stored with the heart model.

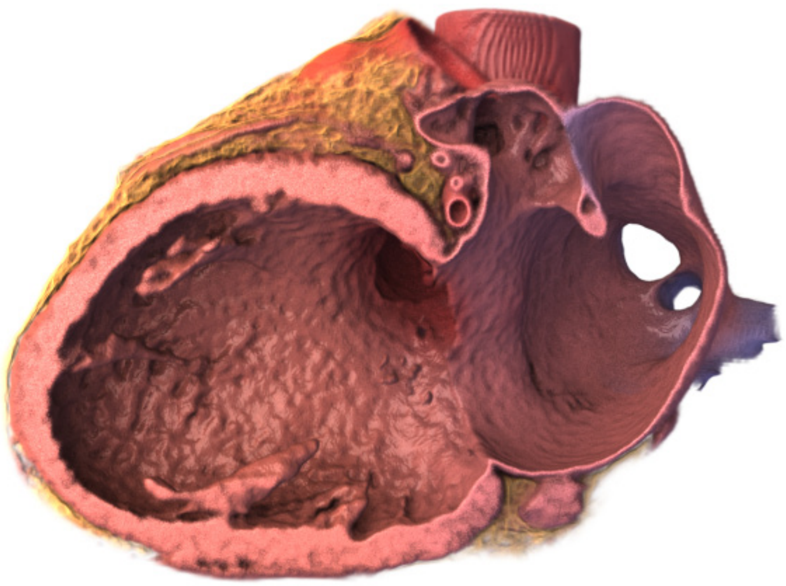


Figure 5.2: *Simple planar cut through left atrium and left ventricle. Compare with hand-drawn illustration* `http://www.netterimages.com/image/730.htm`

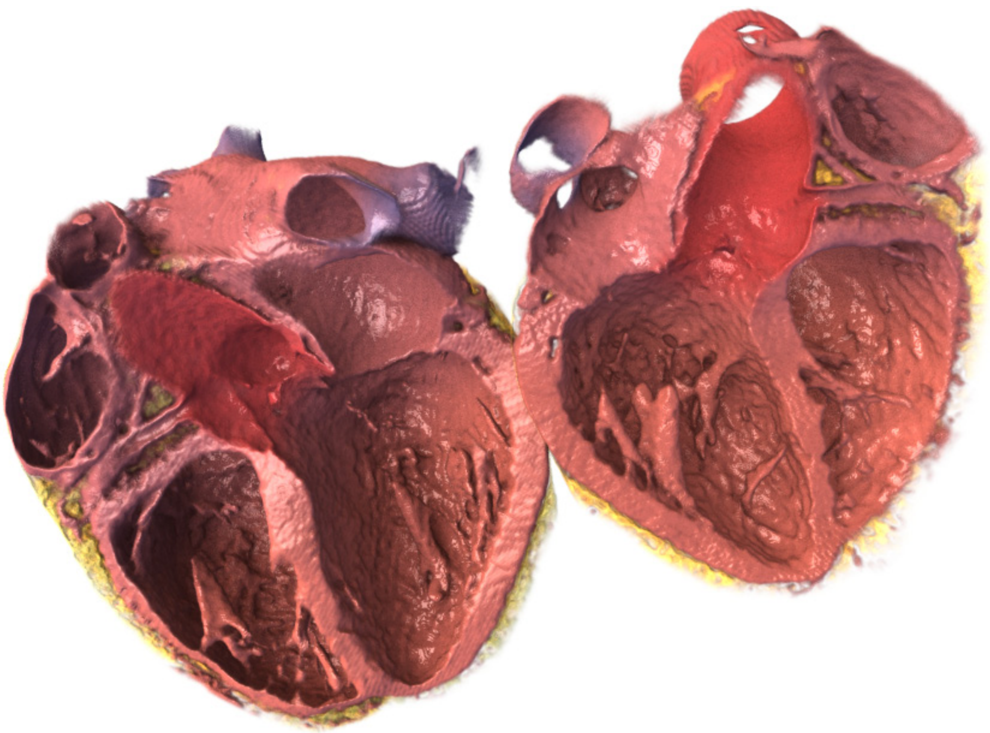


Figure 5.3: *Rigid deformation after a non-planar cut through all cavities. Compare with* `http://www.netterimages.com/image/704.htm`

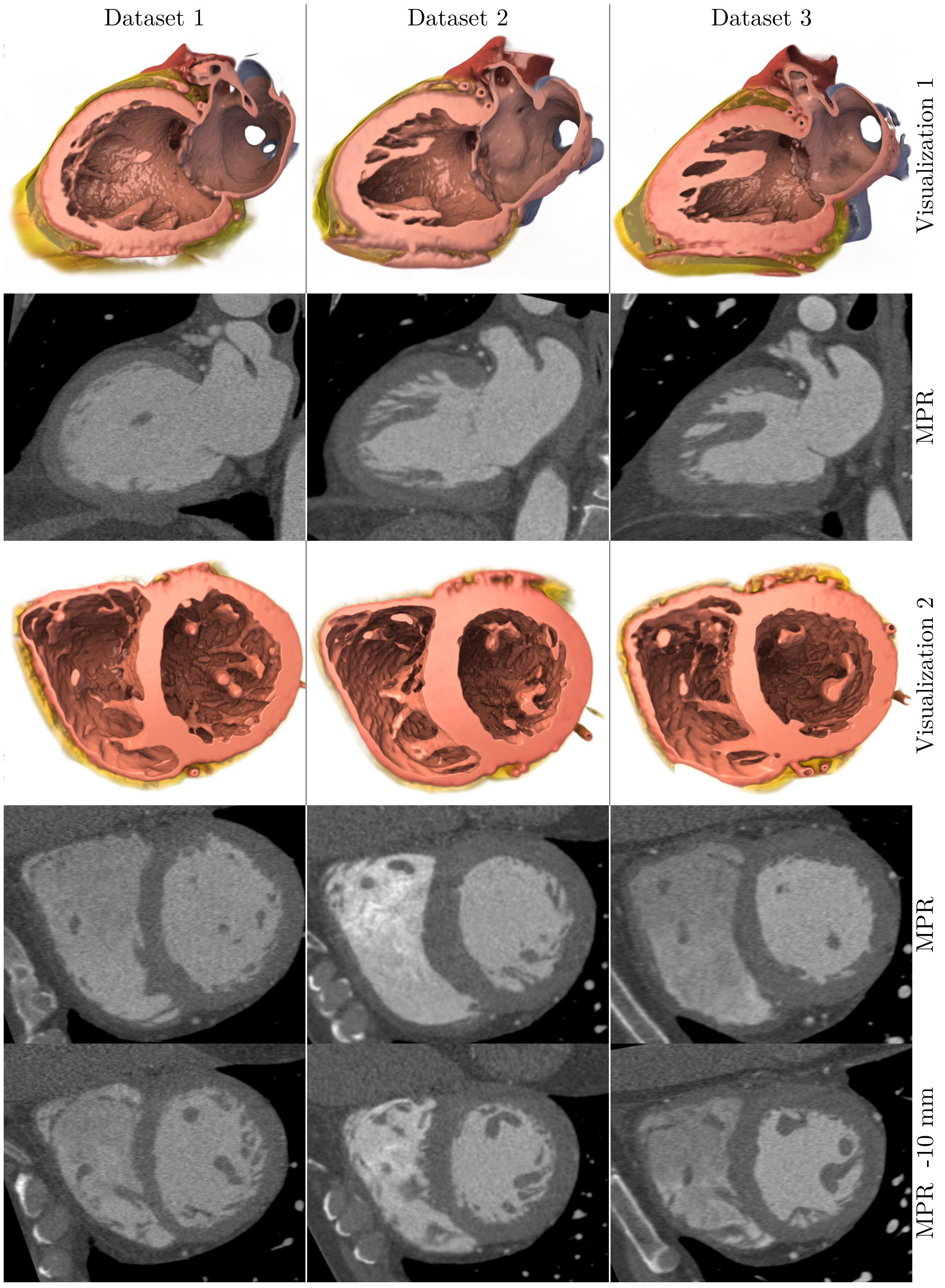


Figure 5.4: *Visualizations of cuts compared to the data [-385 HU, 979 HU]. MPRs underneath the visualizations are exactly at the cuts' position. Last row is 10 mm off.*

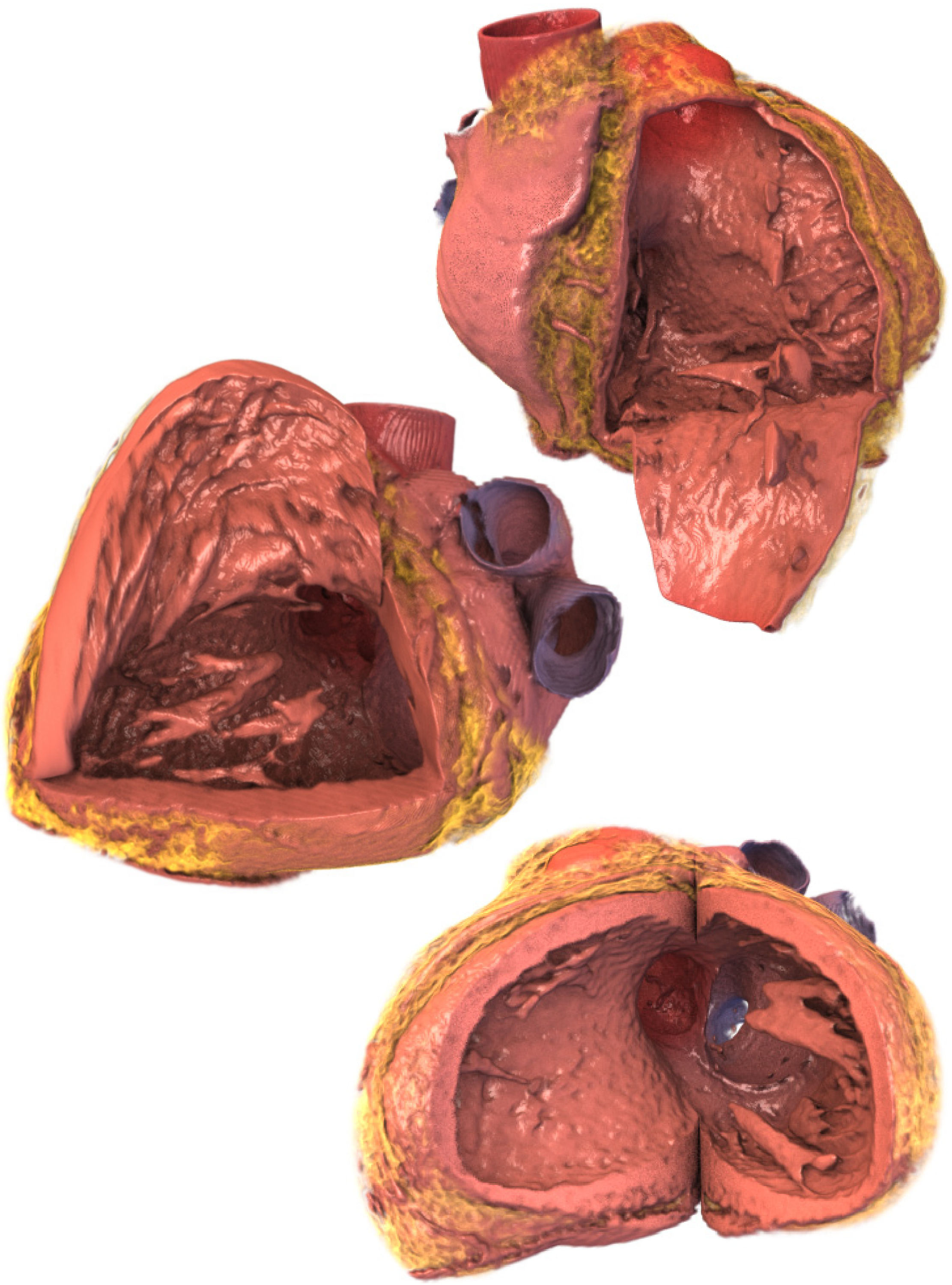


Figure 5.5: *Elastic deformations. Compare with hand-drawn illustrations: (top) right ventricle* `http://www.netterimages.com/image/738.htm`*, (middle) left ventricle* `http://www.netterimages.com/image/730.htm` *and (bottom) right ventricle cut through the apex* `http://www.netterimages.com/image/739.htm`*.*

5.1.4 Creating Dissections with the Shading Graph

When fuzzy classifications and cutting masks are available in the heart model, creating dissections with the shading graph is rather fast. Any combination of transfer functions, deformations, and illumination can be connected to a dataflow graph. Due to the underlying evaluation model, the proper collaboration of the nodes is guaranteed. Figure 5.6 demonstrates the usage of the system in an exemplary dataflow graph. The circular images show the result, that would appear, if the following nodes would be omitted. (g) – (i) are illuminated for better contrast, but in fact, they actually look like (j). Four volumes are used in combination to determine the cut-away of the surroundings, material properties, and region of influence for deformations. Compared to Figure 4.4, less fuzzy classifications are used and nodes (c) – (g) group several preprocessing nodes and transfer functions. Once created, the visualizations can be automatically adapted to individual patients by using the model-based deformation function.

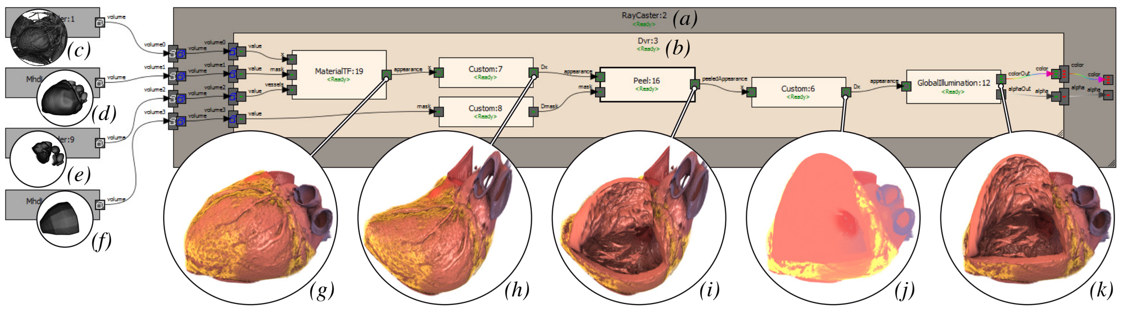


Figure 5.6: *Dataflow graph of Figure 5.5 (middle) in our shader framework. (g) - (i) are illuminated for better contrast, but in fact, they actually look like (j). (a) Ray casting node with four volume ports as input, defines rays and awaits color and opacity per ray. (b) Direct volume rendering node, defines sampling positions on a ray and awaits color and opacity per sampling position. (c) CT angiographic dataset of the human heart. (d) and (e) Fuzzy classifications to reveal the heart and to assign material properties. (f) Mask of the cut. (g) Transfer function applied. (h) Deformation applied to shape the data and the cut's mask in a more planar form. (i) Elastic peeling deformation applied, using the cut's mask. (j) Inverse deformation applied to undo the deformation in (h). (k) Global illumination applied taking into account all previous cuts and deformations.*

5.2 Evaluation

To evaluate the reusability, the extendability, the understandability, and the suitability, a user study was conducted. We asked 22 computer science students to create volume visualizations using transfer functions and cutting nodes. They were only given a *Ray-*

Casting and an empty *Dvr* node, i.e. for the mapping from data to visual attributes, they had to program new nodes from scratch. All students were new to the system and attended a single lecture (90 minutes) on general volume visualization. 81% claimed to be familiar with shader programming. After a ten-minute introduction to the system all students were given 3 tasks:

Task 1: The first task was to program a piecewise linear transfer function and to create a real-time volume visualization with it, that maps data to a specified color and opacity.

Task 2: The second task was to program a node that shows only a slice of 10 mm thickness. The user should be able to interactively move the slice through the dataset.

Task 3: The third task was to create a second reddish transfer function with different parameters and to create a real-time volume visualization that uses one transfer function above the slice and one transfer function underneath.

Figure 5.7 shows the solution of one participant. Timings were taken and an additional questionnaire was given. The user study was conducted in a time slot that was only 45 minutes (minus 10 minutes introduction). Unfortunately, not all students finished within this time slot, so missing tasks were given as homework. This causes some participants' timings and answers to the questionnaire are not available. The results of this user study will be presented and discussed in the following sections.

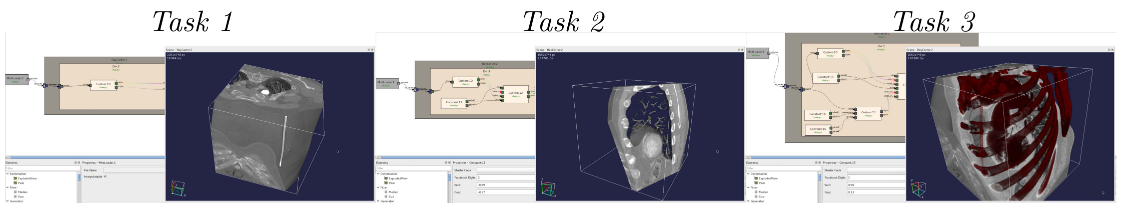


Figure 5.7: *Solution of one participant to the three tasks of the user study.*

5.2.1 Reusability

"The degree to which a software module or other work product can be used in more than one computer program or software system." [IEE91]

In the proposed visual programming language, nodes are equivalent to "software modules" and a visualization compares to a "computer program". One main advantage of an environment such as Volume Studio, is the saved time compared to programming when creating new visualizations. Existing nodes can be reused for several visualizations and they can provide parameters that allow for an adaption to individual demands. Hence, nodes can fit in many different situations allowing for an endless number of visualizations. However, nodes have to be programmed at least once and the advantage of saved time is only existent if nodes are reused.

Having no active usage of the framework outside of the research group, it is difficult to measure an objective amount of reuse. In the user study (which is not objective too, because the tasks are designed to suggest reusing the nodes), 56.2% of the nodes used by the participants in Task 3 are nodes that are reused from the previous tasks. In the heart visualizations presented in Section 5.1, all visualizations share the same *GlobalIllumination* node and the same node containing the transfer functions. The top and middle visualizations in Figure 5.5 share the same deformation node.

A very important factor that supports the reusability of nodes is the strict modularity of nodes within the dataflow graph and the programming of universal code blocks that are automatically translated to explicitly named code blocks (see Section 3.2.5). McCall [McC77] states that modularity is strongly related to reusability. Another important circumstance within the shader graph, that supports the reuse of nodes, is the normalization of all data values, colors, and alpha values to the interval $[0, 1]$ done by the graphics hardware. Transfer functions, logical operators, and other math nodes in turn produce results that are normalized, too. This allows nodes to be valid for many use cases. Additionally, *all* coordinates in the shader framework are given in world space coordinates using Millimeters as units. Compared to typical volume rendering, which tries to solve as many operations in texture space coordinates, this adds a matrix multiplication per sample position. However, it enables nodes using cuts, deformations, or sampling patterns to be simple and universal. Both, the strict normalization of values, as well as the common coordinate system, relate to the concept of *generality* in McCall's "Factors in Software Quality" [McC77], which is strongly related to reusability. Self-descriptiveness is another mentioned criterion related to reusability. Although all nodes have a clear description and interfaces that are easily recognizable due to the input and output ports, self-descriptiveness is a factor that could be further improved, e.g. by icons that are simpler to understand.

5.2.2 Extendability

"The ease with which a system or component can be modified to increase its storage or functional capacity." Syn: expandability; extensibility [IEE91]

At the moment the Framework includes 122 nodes, that support – as claimed in the introduction – "a wide variety of visualization techniques". Though, the number of supported techniques is mainly influenced by the number of employed developers. Hence, extensibility of the proposed concept is the more interesting factor.

Due to the inherent modular design of the dataflow-based shader framework, extending the system is easy. Operators and visualization techniques can be added to the system in encapsulated nodes without changing the system. A plugin management even allows the separate compilation and dynamic loading of nodes during runtime. Existing ports and properties can be reused to shorten development time. Any data that can be stored in shader uniforms (including textures) can be used to evaluate and process data at local or global sampling positions. Despite this encapsulation and flexibility, it is possible to think of techniques that are not easy to integrate as a node. Half-angle slicing [KPH*03], for example, is an illumination technique that is closely interwoven with the rendering algorithm texture slicing [CCF94, CN94]. It is no problem to implement this technique in the proposed framework, but it is not easy to think of it as a node. It would rather be a checkbox in the texture slicing node. The same is true for the progressive rendering technique presented in Section 3.3. In general, algorithms that do not work independently from each other are difficult to separate into individual nodes.

To test the difficulty of programming new nodes in the proposed framework, the participants of the user study where asked to program two transfer functions and one slicing operator from scratch and to create different visualizations with that (see Task 1 - 3 in Subsection 5.2). All participants succeeded with Task 1 and Task 2. Only one student failed to solve Task 3. Different solution processes were possible. On average the participants created 4 new nodes. 16 students solved Task 1 in the given time slot, and 7 students solved Task 1 and Task 2 in the given time slot. There is only one timing available for Task 3. From those who solved the tasks within the given time slot, Task 1 took 20.3 minutes on average, Task 2 took 17.1 minutes on average, and Task 3 took 9 minutes (see Figure 5.8 for standard deviations). Those who did not solve the tasks within the given time slot, went on at home, hence, no timings are available. As a result, time measurements are biased, because more "good" participants are contained in the measurement.

Overall, it can be claimed that it is possible for computer science students to extend the system with simple nodes if they have a basic knowledge about volume visualization and minimal knowledge about the system. It was very beneficial for the students that they were able to modify ports and shader code of their nodes during runtime. By this means they were able to directly see their programming results and potential failures.

5.2.3 Understandability

"The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use." [ISO01]

To make programming more understandable to some particular audience is one of the most common goals of visual programming languages [Bur99]. The proposed framework provides several cues that ease the understanding of the usage. By depicting the whole dataflow directly from preprocessing to the final color by drawing directed edges between nodes, the processing becomes explicit, which is a common strategy according

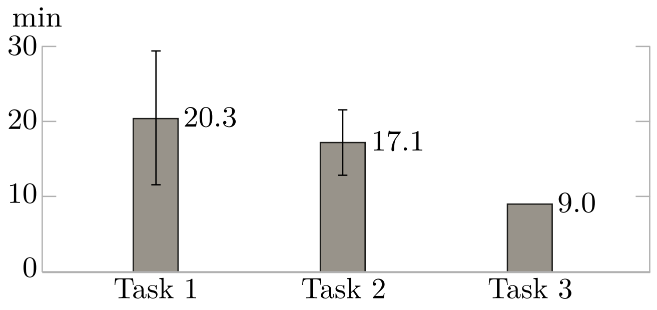


Figure 5.8: *Average time needed for the three tasks. Error bars indicate ±1 standard deviation. It was not possible to compute a meaningful standard deviation for the last task, because timing is available only for one participant.*

to Burnett. Furthermore, structuredness directly influences understandability [Boe78], which is clearly articulated by using visual nodes representing processing steps. There are several more cues that support the understanding:

- The connectivity between ports is indicated by the port's icon and color (must be the same). In addition to the color of the ports, areas of the dataflow graph that can not be connected are divided by nesting nodes, e.g. preprocessing and shader nodes.
- The type of a port is indicated by small icons that relate to the type (e.g. three stripes for the type *vec3*, one stripe for a *float*, and four stripes for type *vec4*).
- Connections that relate to color are visualized using a color ramp. Connections that are related to opacity are visualized using a black-white ramp.
- Edges are directed, which is visualized by arrows. Hence, the flow direction is clear to the user.
- 1D and 2D histograms are supported on any stage of the shader pipeline. This enables the user to inspect data at any position in the graph and to specify transfer functions purposefully.
- The positioning of the Gradient node and other global operators (deformations, illumination nodes) is not restricted to the first position in the shader graph (compare data-driven versus demand-driven dataflow in Section 3.2).
- Using the framework is only three elementary tasks to the user: Adding nodes (via menu or drag&drop), connecting nodes (by dragging edges between two ports), and parameterizing nodes (by changing the properties in the property tab or using optional widgets, e.g. transfer function widget).
- The status of the node is displayed on the nodes using the colored keywords Error, Incomplete, NeedsUpdate, Updating, Ready.

- Inspector for detailed inner and outer state of a node. Console for error, warning, debug, and info messages.

The user study shows, that the participants were particularly successful, given three different visualization tasks. When asked directly, the majority agreed or strongly agreed that the dataflow concept was easy to understand. See Figure 5.9 for detailed results and error bars.

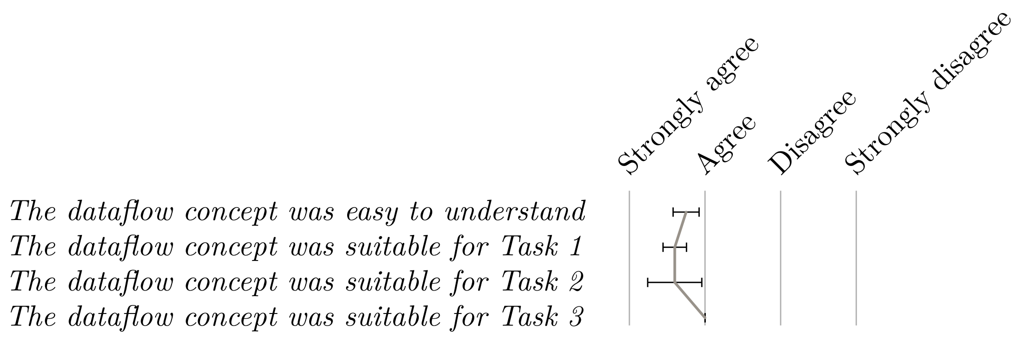


Figure 5.9: *Questions to understanding and suitability. Error bars: ±1 standard error (For the last question only one answer was given. Hence, a meaningful error bar could not be calculated).*

5.2.4 Functionality and Suitability

Functionality: *"The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions."* [ISO01]
Suitability: *"The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives."* [ISO01]

The successful imitation of hand-drawn illustrations taken from anatomy atlases in Section 5.1 demonstrates the functionality of the proposed framework and the use of real clinic volume data demonstrates the suitability. When given three specific visualization tasks (see beginning of Section 5.2), the majority of computer science students agree or strongly agree that the dataflow concept is suitable for some specific visualization tasks (compare Figure 5.9). However, this inspection does not consider the limitations of the proposed framework regarding other datasets, other visualizations, and other users. This consideration will be done in the next three paragraphs.

Datasets In all datasets that we have access to, the contrast agent is sufficiently homogeneous in the left ventricle and left atrium. Though, in the right ventricle and even more in the right atrium, the concentration of contrast agent is often extremely varying. Even worse, in about 13 % of the datasets, the right ventricle is not contrasted at all and thus, not distinguishable from muscles' tissue. Figure 5.10 (left) shows what happens if the contrast agent is not present in the right ventricle.

Depending on the determined radiation dose, the signal to noise ratio in the datasets varies tremendously. This is especially problematic when visualizing small (or thin) features like valves or surfaces in low-contrast areas. Figure 5.10 (right) tries to mimic a complex cut, where all valves are visible (compare `http://www.netterimages.com/image/815.htm`). But artifacts related to noise heavily disturb the appearance of the thin valves.

These problems, though, do not relate to the proposed Framework; every other program in the literature would have problems with these datasets.

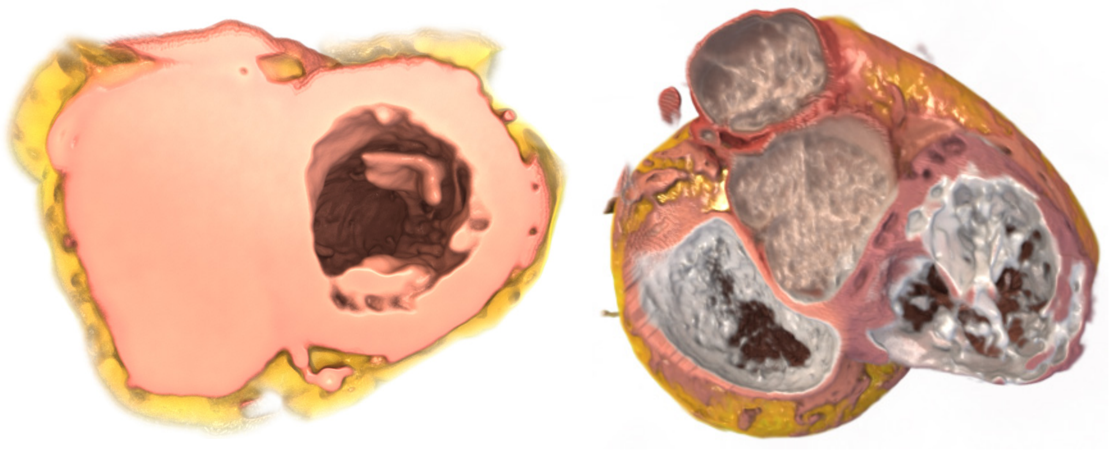


Figure 5.10: *Functional limits. Left: Blood without contrast agent in the right ventricle can not be distinguished from muscles' tissue. Right: Non-planar cut through all four valves (compare* `http://www.netterimages.com/image/815.htm`*). Valves are thin and suffer more notably from noise.*

Visualizations Using the proposed techniques, the success for creating visualizations as in anatomy atlases is mainly dependent on the image details. If all necessary details can be perceived in the data, the visualization should be possible. However, a lot of diseases and details shown in anatomy atlases are not possible to identify in CT data. For example, neural pathways are rather not visible in CT scans. Individual fibers of muscles are too small and exhibit a too low contrast to be visible in clinical CT data. On top of this, some deformations are too complex to be built from a combination of the three basic deformations shown in this thesis. Some of them seem to necessitate a physical simulation and a much more complicated deformation algorithm, like the deformable models by Nealen et al. [NMK*06] must be implemented. However, semantic colorization, as the assignment of different colors to different areas of the heart, are

		Possible using individual datasets			*Reason*	
Image No.	*Fig.*	No	Limited	Yes	Missing data	Deformation
649			■		■	■
671 / 697			■		■	
702		■				■
704	5.3			■		
730	5.2			■		
738	5.5			■		
739	5.5		■		■	
752	5.1			■		
766				■		
770		■			■	■
815	5.10		■		■	
823		■			■	
853			■		■	■
858			■		■	
2282	5.4*			■		
8938		■			■	
50572				■		

Table 5.1: *Possible or impossible visualizations of Netter's atlas. The image number indicates the stock number of the illustration at* `http://www.netterimages.com` *(directly linked in electronic PDF file). *Not exactly the same, scar tissue with restrictions.*

possible as long as this information can be integrated as additional information to a model. Table 5.1 lists all heart illustrations in Netter's atlas [Net10] and indicates which are possible, or impossible, or possible with limitations in the proposed framework using individual CT datasets due to the authors opinion. If not possible, the reason is given, which can either be a missing representation of the visualized feature in the data or a too complex deformation. Illustrations that do not only show the heart, or do show possible treatments, are not listed.

Regarding technical restrictions to the concept during the last five years, no volume rendering technique was spotted, which could not be integrated. However, some techniques massively benefit from the dataflow-based concept, and some techniques do not benefit. E.g. a visualization techniques like the Curved Planar Reformation (Section 3.4), can use all advantages of the dataflow-based concept. Other algorithms, e.g. half-angle slicing [KPH*03], do not profit from the modular concept. Then the polygon-based visualization techniques do not benefit at all in the current concept.

All in all, it is not the proposed framework that makes the impossible visualizations impossible. Every other framework would have the same difficulties. On the contrary, the flexibility and the simple combinability of techniques suppose that Volume Studio is a rather suited platform to make the best of difficult cases.

Users Regarding the users of the platform, framework-specific problems are obvious. Not all types of users are able to use the visual programming system and even less users are able to extend the system. If the aim is to be usable for a maximum sized group of users without major learning overhead, simpler interfaces (with more restrictions) must be provided. MevisLab [MF14] and Voreen [Vor14] provide convenient ways to create more simple interfaces for visualizations that contain only limited sets of parameters. Volume Studio provides simple support to create interfaces on top of any visualization, too. Any functionality of the framework can be addressed and controlled by JavaScript and new GUI elements can be added during runtime. Hence, automated execution sequences can be made available through a few buttons. This includes the script-based creation of whole new dataflow graphs. However, the creation of a simple interface to a *specific* task is a completely different topic than the creation of flexible interface to an *unspecific* task. Hence, the description of the scripting layer was completely ignored in this thesis.

All in all, I believe that the clearly and explicitly laid out dataflow graph provides completely new possibilities to users that are not good in programming or not able to program. For programming users, I assume a major advantage in time saving granted by simple reusability, and a major advantage in maintainability granted by strict modularity.

5.2.5 Efficiency

"The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions." [ISO01]

Most of the performance evaluations have been done in the other chapters of this thesis. Section 3.2 shows that the demand-driven dataflow of the framework does not introduce any overhead compared to a data-driven dataflow which is used by all other frameworks. Section 3.3 shows that even very expensive techniques combined in a dataflow graph can be navigated in real-time when using a progressive renderer. Hence, the performance could be claimed to be appropriate, even under many not-stated conditions.

However, the visualizations shown in this chapter are computationally expensive. The dissections in Section 5.1 are rendered with high sampling rates and global illumination to account for printable image quality, as in anatomy atlases. Each image took several seconds to render. The only reason why the visualizations were enabled to be specified interactively is the progressive renderer that starts with a rough approximation and progressively refines the image when time is available (See video in the supplementary material that comes with this document). When using no progressive rendering and no anti-aliasing, the rendering of Figure 5.5 (middle) takes 4.18 seconds on a Nvidia GTX480. Figure 5.11 compares different illumination modes applied to that figure. When the three deformations (see Figure 5.6 (h) – (j)) are not applied, the rendering costs are approximately cut by half. This gives only an impression of possible rendering costs, but the duration for rendering a frame depends heavily on the parameters. For example, when the transfer function is changed to a steep step function, rendering costs improve by more than an order of magnitude, while changing the transfer function to a less steep function with more semi-transparent samples makes the situation much more expensive.

Besides rendering costs, the time a user needs to create a visualization is important to create visualizations for individual patients. Creating a visualization, as in this chapter, for the first time is an elaborate task. Especially the creation of cutting masks is time consuming. However, once a visualization is created, it can be adapted to other datasets in an automated way. If all deformations and cuts are defined relative to control points in the heart model, the creation is fully automatic. The computational costs depend on the used dataflow graph and dataset size and take 1 – 2 minutes for the visualizations in this chapter. E.g. the visualizations of Figure 5.4 take 75 seconds to generate for a new patient using the dataflow graph of Figure 4.4. When creating the visualizations in Figure 5.5 the shader graph in Figure 5.6 was used. It only uses four fuzzy classifications that have been grouped in two volumes, which reduces the processing time.

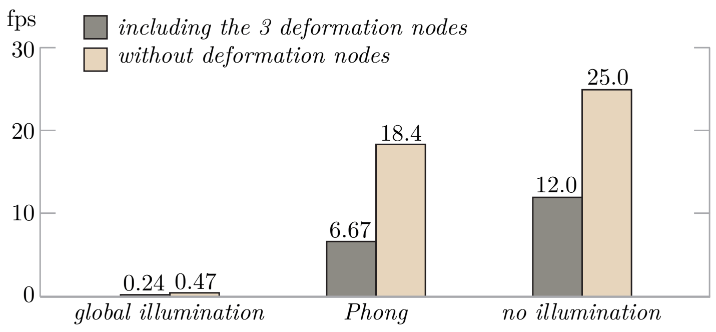


Figure 5.11: *Frames per second without progressive rendering of Figure 5.5 (middle) using global illumination, Phong, or no illumination. Bright bars show the results when the deformations in Figure 5.6 (h) – (j) are not applied. Resolution:* 552 × 471 *pixels, no anti-aliasing, shadows: 13 rays á 18 samples.*

5.3 Detmold Child

The techniques presented in this thesis are, of course, suitable to be used in other visualizations than those taken from anatomy atlases. Especially the cuts can reveal insight. This is particularly true for volume datasets which do very often contain information which is not visible at the surface of objects.

Figure 5.12 shows four different views to a CT scan of a child mummy – the famous "Detmold Child". The child's age was 8 to 10 months when it died 6500 years ago, which was found out regarding the growth of the teeth. The child was imaged to find out the cause of death. Earlier scans revealed a possible heart disease. However, the earlier diagnosed heart disease is very common and does not typically cause death.

In contrast to living humans, the mummy is completely dried out and contains no water or blood. As a result, the Hounsfield values are completely different. On top of that, the heart's anatomy is very unusual, although quite well preserved. Hence, even cardiologists are not trained in viewing this data and spatial orientation on slice views is very difficult. As a result, the course of some vessels and especially their connections among each other are very difficult to retrace. This might be the reason why earlier CT scans did not reveal the true heart disease. Using curved cuts and illumination, the true heart disease was visualized very comprehensibly. This is especially useful for publishing the results to others that do not have access to the CT data.

The lower visualizations exhibit a checkerboard as background. Each square is 1 cm times 1 cm, giving a heart that is very small. In GPU-based rendering, trilinear interpolation is used to create a continuous signal from discrete data points (see Section 2.1). Trilinear interpolation, though, is not perfect and leads to artifacts when zooming in very close, which is necessary in this case due to the small extents. To overcome this issue, the dataset was interpolated in Volume Studio during preprocessing using a lanczos filter [Duc79] with a kernel diameter of 11 voxels, which resulted in a (near-)perfect interpolated version of the dataset, which had a three times higher resolution in each direction. The tiny structures would be very blocky without this sophisticated interpolation. Despite this preprocessing, only data-based transfer functions, cuts, and global illumination were used to create the visualizations. The cuts were set interactively by trial and error using the carve map approach presented in Schwier's thesis [Sch07].

Before the visualizations were created using Volume Studio, radiologists from the Heart and Diabetes Center North Rhine-Westphalia created visualizations using their software tools by Siemens. However, the anatomy could not be depicted adequately. Here, the flexible construction of Volume Studio turned out to be very practicable to deal with this uncommon data and uncommon anatomy.

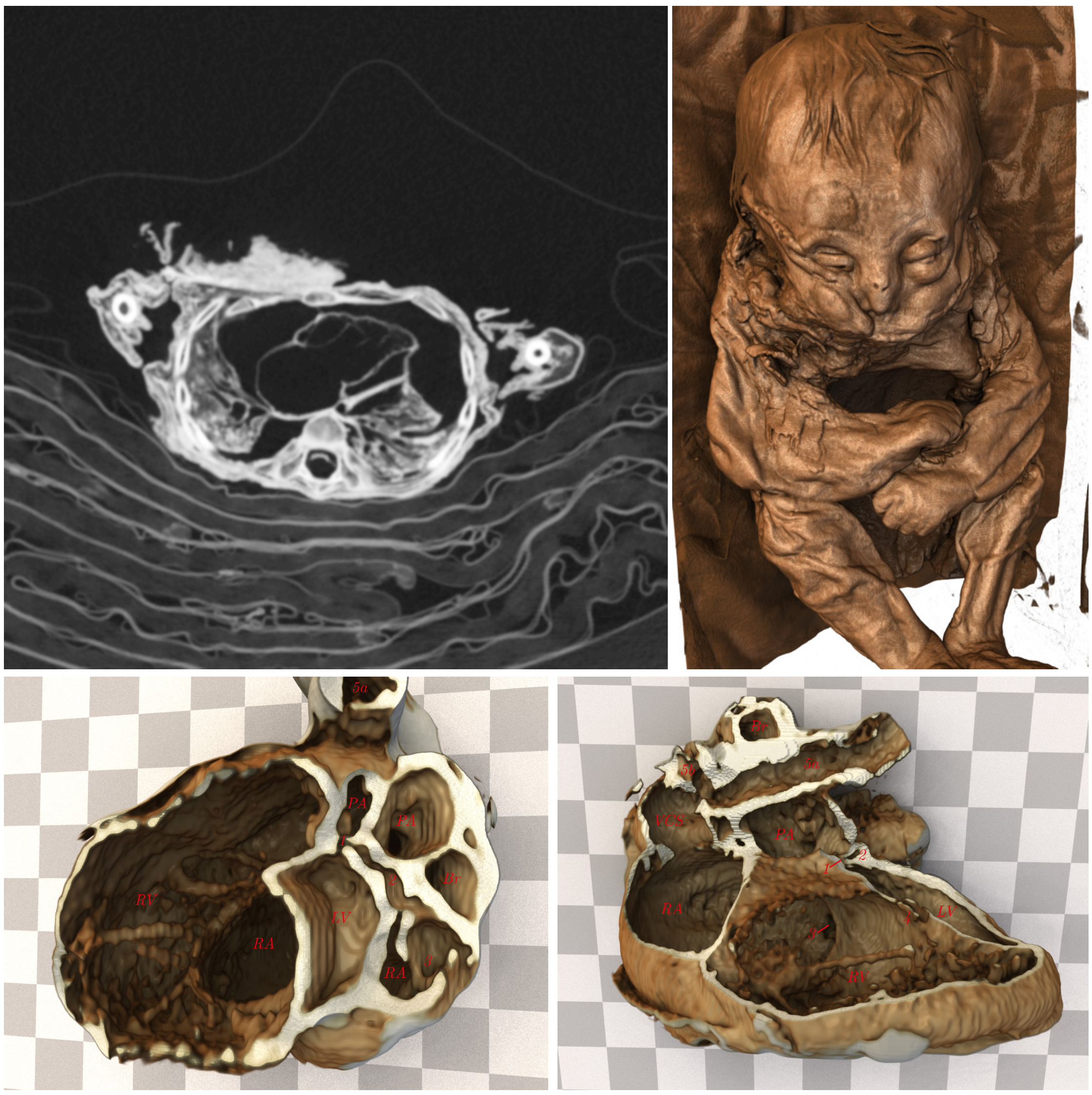


Figure 5.12: *Four visualizations of a CT scan of a child mummy. Top left: Single slice through the thorax. The heart is still well preserved after 6500 years. Top right: Outer view, using global illumination. Bottom left and right: Isolated heart, cut to reveal inner structures. RV: Right ventricle; LV: Left ventricle; RA: Right atrium (Left atrium is missing); PA: Pulmonary artery; VCS: Vena Cava Superior; Br: Bronchus; 1, 2, 3: Unusual connection between PA, RA and LV; 4: unusual connection between LV and RV; 5a, 5b: Brachiocephalic veins.*

Chapter 6

Conclusion

The aim of this thesis was to design and develop a shader framework for volume rendering that is flexible enough to cope with a wide variety of techniques, which should be easily combinable with each other. To test the proposed framework, hand-drawn heart illustrations from Netter's anatomy atlas were taken to be imitated using individual patient data.

The decision was made in favor of a dataflow-based visual programming concept, which is more simple to use than programming, but more flexible than a shader with predefined functionality and scenarios which can be enabled and disabled (Chapter 2). The drawbacks of this and other previously presented dataflow-based shader frameworks were the inability to flexibly cut and deform logical combinations of multiple volumes, while assuring correct illumination. These capabilities are necessary for creating virtual dissections as in anatomy atlases. In Chapter 3, a modification of the processing strategy was proposed that copes with global operators, like deformations and illumination, easily, while introducing no processing overhead. As a result, much more complex and expensive visualizations could be easily created by simply connecting building bricks in a dataflow graph. To still assure interactive navigation and manipulation, an orthogonal progressive rendering approach was developed that does not diminish the flexibility of the proposed shader framework, but ensures interactivity, as well as, outstanding image quality in almost every situation. Chapter 4 explains briefly how to use and combine visualization techniques to create visualizations of the heart, as in anatomy atlases. On top of this, it tackles the automated creation of such visualizations from individual patient data by using a model of the heart. Additional information, necessary to enrich the information of CT-datasets, is automatically added and is used to colorize, cut, and deform dissections of the heart correctly. Finally, in Chapter 5 a catalogue of possible visualizations is given and the shader framework is evaluated in terms of reusability, extendability, understandability, functionality, suitability, and efficiency by conducting a user study.

Hand-drawn illustrations in anatomy atlases serve to understand the human morphology or functioning. They are easier to understand and more natural than slices from CT scans while being less demanding to the viewer. The per-patient-visualizations created in this thesis allow radiologists, therapeutic physicians, and surgeons to compare, explore, and discuss volume data in view of atlas illustrations and to ease the communication with patients. Especially a realistic intra-cardiac view is new to clinic routines and allows for a better understanding of the individual complex structures of the heart's cavities. Some other hand-drawn illustrations can not be imitated yet. Especially the missing information in CT-data creates an ill-posed relation between the data and visualization aim. Some deformations are, on the other hand, too complex, to be simulated correctly. However, in future the amount of data radiologists, therapeutic physicians, and surgeons are exposed to, will increase further. This leads to a very demanding inspection of the data, and computational preparation of the data to an easier to understand shape can help to make less failures. Illustrations, as in anatomy atlases, are one future mean to ease the burden of the physicians. Other visualization techniques, like the curved planar reformation, are already effectively used in clinical practice. To close the gap here between clinical practice and academic research, more simple ways have to be found to implement and develop new visualizations. The proposed shader framework is one possibility to create new GPU-based visualizations quicker. However, the creation of such visualizations is still very demanding. Here, the advances in the research area of heart models are very promising to automatically adapt visualizations to other datasets. This way, not only the effort can be reduced to a minimum, it also enables reproducible and comparable visualization results that do not suffer from inter-individual variations among radiologists. In this thesis, a small and delimited set of datasets was used, where correct deformation functions from one patient to the other were known. As a result, visualizations were successfully adapted to individual patients automatically.

While the evaluation chapter indicates that the dataflow is well suited for visualization purposes, it is not suited to specify interactions with visualizations flexibly. At the moment, the use of the a scripting language is necessary to connect user input to arbitrary functionality. Another dataflow layer might save time in this field of application, too. Furthermore, the interaction techniques integrated to create cuts and deformations interactively, are very unnatural (i.e. sliders). Here a more natural interaction with the shown image itself, as in [BKW08], is desirable.

Using the framework within the last years, it became clear to me, that the almost unconditional reusability and modularity accelerates the development massively. As a result, in a very limited time-span our small research group implemented and combined many visualization techniques that typically take a lot of time. The same is true for the rapid prototyping of shaders, which allows programming while the program is executed. It made shader programming fast and easy, which is typically not a property of shader programming, due to the inherent inconvenient debugging. The framework turned out to be well suited to create volume visualizations flexibly, and many more visualizations are possible apart from those shown in this thesis. Figure 6.1 shows a combination of an atlas visualization and a PET scan, which is used for light emission. It shows the functioning of the metabolism as a glowing heat, while revealing scar tissue at the bottom of the left ventricle.

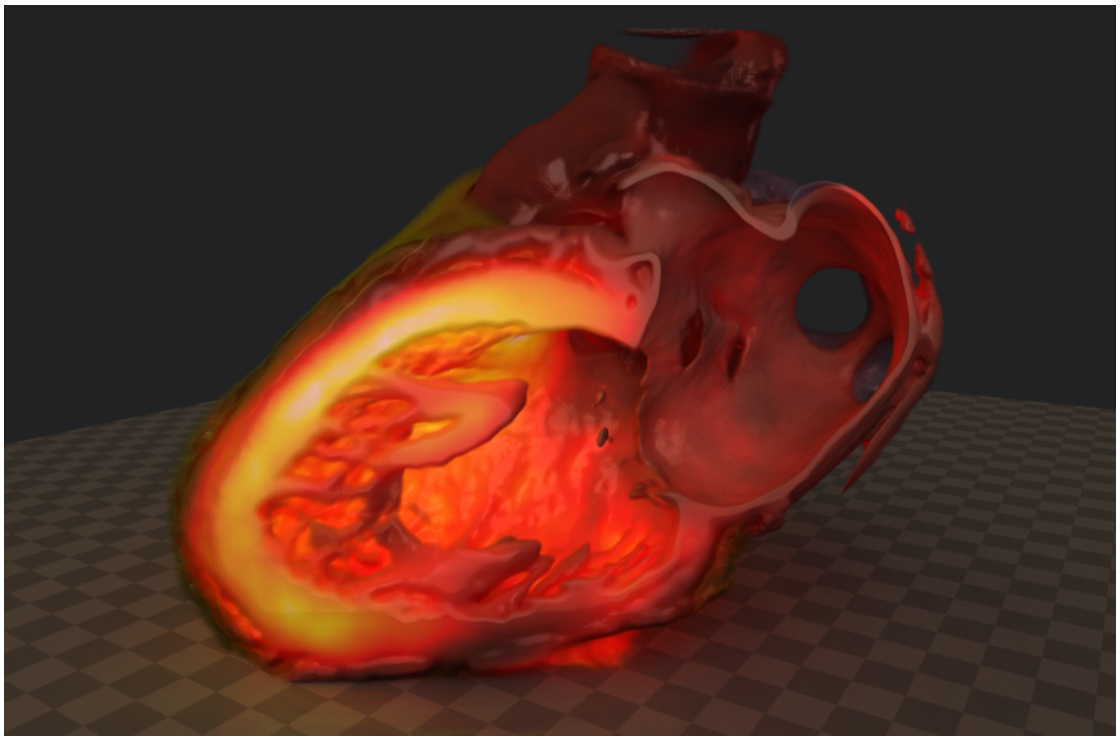


Figure 6.1: *Atlas visualization, as in Figure 5.4, enhanced by a PET dataset to show the functioning of the metabolism as a glowing heat. At the bottom of the left ventricle, the dark area clearly reveals scar tissue.*

Publications (Chronological Order)

[WAED10] WETTE P., ARENS S., ELSNER A., DOMIK G.: Extending the corkscrew algorithm to find bifurcations of vessels. In *Proceeding of Computer Graphics and Imaging 2010* (2010), Sappa A., (Ed.), vol. 679.

[AD10] ARENS S., DOMIK G.: A survey of transfer functions suitable for volume rendering. In *Volume Graphics* (2010), Westermann R., Kindlmann G. L., (Eds.), Eurographics Association, pp. 77–83.

[ADW*10] ARENS S., DOMIK G., WEISE R., FRICKE H., BURCHERT W.: Weniger Sehen - mehr Verstehen: Größen-basierte und Textur-basierte Transferfunktionen zur Volumenvisualisierung von CT Daten der Koronargefäße. In *DGMP* (2010).

[ABD11] ARENS S., BOLTE M., DOMIK G.: Seamless integration of multimodal shader compositing into a flexible ray casting pipeline. In *VMV* (2011), Eisert P., Hornegger J., Polthier K., (Eds.), Eurographics Association, pp. 347–352.

[AD11] ARENS S., DOMIK G.: Volume Studio: Flexible Multi-Volume Ray Casting Pipeline mit nahtlos integriertem Rapid Prototyping von Shadern. In *Informatik 2011: Workshop Emerging Technologies for Medical Diagnosis and Therapy* (2011), vol. 192 of *LNI*.

[DASH11] DOMIK G., ARENS S., SCHARLAU I., HILKENMEIER F.: How useful is computer graphics for medical diagnoses. In *Informatik 2011: Workshop Emerging Technologies for Medical Diagnosis and Therapy* (2011), vol. 192 of *LNI*.

[DSAS11] DOMIK G., STEFFEN F., ARENS S., SCHARLAU I.: Usefulness of style transfer functions in medical diagnosis. In *SIGGRAPH Posters* (2011), ACM, p. 89.

[AIB*11] ARENS S., ISENBERG T., BOLTE M., DOMIK G., WEISE R., HOLZINGER J., FRICKE H., BURCHERT W.: Räumliche Wahrnehmung ct-angiographischer Koronararterien in einer Curved Planar Reformation mittels Direct Volume Rendering. In *NukMed* (2011).

[DA12] Domik G., Arens S.: Früherkennung von Plaque in den Koronargefäßen - Räumliche Wahrnehmung bei der Untersuchung von Koronararterien in CT-Datensätzen. In *ForschungsForum Paderborn* (2012), Risch N., (Ed.).

[DATS12] Domik G., Arens S., Tünnermann J., Scharlau I.: Evaluierung medizinischer Volumenrendering-Algorithmen durch empirische Studien. In *FIFF-Kommunikation* (2012).

[ABD13a] Arens S., Bolte M., Domik G.: *Pull-based Dataflow for Multi-Volume Visualization.* Tech. Rep. tr-ri-13-332, University of Paderborn, Department of Computer Science, June 2013.

[ABD13b] Arens S., Bolte M., Domik G.: Visualizing dissections of the heart in a dataflow-based shader framework for volume rendering. In *VMV* (2013), Bronstein M. M., Favre J., Hormann K., (Eds.), Eurographics Association, pp. 231–232.

[DASF13] Domik G., Arens S., Stilow P., Friedrich H.: Helping high schoolers move the (virtual) world. *Computer Graphics and Applications, IEEE 33*, 1 (Jan 2013), 70–74.

[ADW*14] Arens S., Domik G., Weise R., Holzinger J., Burchert W.: Optimierte 3D-Morphologiedarstellung tomographischer Datensätze des Herzens. In *NukMed* (2014).

[GAD14] Gmyr R., Arens S., Domik G.: *Interactive GPU Ray Casting using Progressive Blue Noise Sampling.* Tech. Rep. tr-ri-14-339, University of Paderborn, Department of Computer Science, June 2014.

Bibliography

[ABD11] Arens S., Bolte M., Domik G.: Seamless integration of multimodal shader compositing into a flexible ray casting pipeline. In *VMV* (2011), Eisert P., Hornegger J., Polthier K., (Eds.), Eurographics Association, pp. 347–352.

[ABD13a] Arens S., Bolte M., Domik G.: *Pull-based Dataflow for Multi-Volume Visualization.* Tech. Rep. tr-ri-13-332, University of Paderborn, Department of Computer Science, June 2013.

[ABD13b] Arens S., Bolte M., Domik G.: Visualizing dissections of the heart in a dataflow-based shader framework for volume rendering. In *VMV* (2013), Bronstein M. M., Favre J., Hormann K., (Eds.), Eurographics Association, pp. 231–232.

[AD10] Arens S., Domik G.: A survey of transfer functions suitable for volume rendering. In *Volume Graphics* (2010), Westermann R., Kindlmann G. L., (Eds.), Eurographics Association, pp. 77–83.

[AD11] Arens S., Domik G.: Volume Studio: Flexible Multi-Volume Ray Casting Pipeline mit nahtlos integriertem Rapid Prototyping von Shadern. In *Informatik 2011: Workshop Emerging Technologies for Medical Diagnosis and Therapy* (2011), vol. 192 of *LNI*.

[ADW*10] Arens S., Domik G., Weise R., Fricke H., Burchert W.: Weniger Sehen - mehr Verstehen: Größen-basierte und Textur-basierte Transferfunktionen zur Volumenvisualisierung von CT Daten der Koronargefäße. In *DGMP* (2010).

[ADW*14] Arens S., Domik G., Weise R., Holzinger J., Burchert W.: Optimierte 3D-Morphologiedarstellung tomographischer Datensätze des Herzens. In *NukMed* (2014).

[AIB*11] Arens S., Isenberg T., Bolte M., Domik G., Weise R., Holzinger J., Fricke H., Burchert W.: Räumliche Wahrnehmung ct-angiographischer Koronararterien in einer Curved Planar Reformation mittels Direct Volume Rendering. In *NukMed* (2011).

[ALSS11] Alkadhi H., Leschka S., Stolzmann P., Scheffel H.: *Wie funktioniert CT?* Springer, 2011.

[Arg14] Argosy Publishing, Inc.: Visible body. `http://www.visiblebody.com`, 2014. Accessed: 2014-03-27.

[AW90] Abram G. D., Whitted T.: Building block shaders. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), SIGGRAPH '90, ACM, pp. 283–288.

[BBPtHR08] Brecheisen R., Bartroli A. V., Platel B., ter Haar Romeny B. M.: Flexible gpu-based multi-volume ray-casting. In *VMV* (2008), Deussen O., Keim D. A., Saupe D., (Eds.), Aka GmbH, pp. 303–312.

[BdGMK*09] Berrington de González A., Mahesh M., Kim K.-P. P., Bhargavan M., Lewis R., Mettler F., Land C.: Projected cancer risks from computed tomographic scans performed in the united states in 2007. *Archives of internal medicine 169*, 22 (2009), 2071–2077.

[BG06] Bruckner S., Gröller M. E.: Exploded views for volume data. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1077–1084.

[BG07] Bruckner S., Gröller M. E.: Style transfer functions for illustrative volume rendering. *Comput. Graph. Forum 26*, 3 (2007), 715–724.

[BG09] Bruckner S., Gröller M. E.: Instant volume visualization using maximum intensity difference accumulation. *Computer Graphics Forum 28*, 3 (June 2009), 775–782.

[BKW08] Bürger K., Krüger J., Westermann R.: Direct volume editing. *IEEE Trans. Vis. Comput. Graph. 14*, 6 (2008), 1388–1395.

[Boe78] Boehm B. W.: *Characteristics of Software Quality*, 1st ed., vol. 1. North-Holland Publishing Company, June 1978.

[BP08] Botha C. P., Post F. H.: Hybrid scheduling in the devide dataflow visualisation environment. In *SimVis* (2008), Hauser H., Straßburger S., Theisel H., (Eds.), SCS Publishing House e.V., pp. 309–322.

[Bur99] Burnett M. M.: *Visual Programming*. John Wiley & Sons, Inc., 1999.

[BV09] Birkeland A., Viola I.: View-dependent peel-away visualization for volumetric data. In *Proceedings of the 2009 Spring Conference on Computer Graphics* (New York, NY, USA, 2009), SCCG '09, ACM, pp. 121–128.

[BvUW*07] Bitter I., van Uitert R., Wolf I., Ibanez L., Kuhnigk J.-M.: Comparison of four freely available frameworks for image processing and visualization that use itk. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (2007), 483–493.

[CCF94] Cabral B., Cam N., Foran J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 Symposium on Volume Visualization* (New York, NY, USA, 1994), VVS '94, ACM, pp. 91–98.

[CCI*07] Chen M., Correa C., Islam S., Jones M. W., Shen P.-Y., Silver D., Walton S. J., Willis P. J.: Manipulating, deforming and animating sampled object representations. *Computer Graphics Forum 26*, 4 (2007), 824–852.

[CHM03] Chen H., Hesser J., Männer R.: Ray casting free-form deformed-volume objects. *The Journal of Visualization and Computer Animation 14*, 2 (2003), 61–72.

[CIB14] CIBC:, 2014. ImageVis3D: An interactive visualization software system for large-scale volume data. Scientific Computing and Imaging Institute (SCI), Download from: `http://www.imagevis3d.org`.

[CM08] Correa C. D., Ma K.-L.: Size-based transfer functions: A new volume exploration technique. *IEEE Trans. Vis. Comput. Graph. 14*, 6 (2008), 1380–1387.

[CMA04] Chen T., Metaxas D., Axel L.: 3d cardiac anatomy reconstruction using high resolution ct data. In *MICCAI 2004*, Barillot C., Haynor D., Hellier P., (Eds.), vol. 3216 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, pp. 411–418.

[CN94] Cullip T. J., Neumann U.: *Accelerating Volume Reconstruction With 3D Texture Hardware.* Tech. rep., University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1994.

[Coo84] Cook R. L.: Shade trees. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1984), SIGGRAPH '84, ACM, pp. 223–231.

[Coo86] Cook R. L.: Stochastic sampling in computer graphics. *ACM Trans. Graph. 5*, 1 (1986), 51–72.

[CPR*10] Camara O., Pop M., Rhode K. S., Sermesant M., Smith N., Young A. A. (Eds.):. *Statistical Atlases and Computational Models of the Heart, First International Workshop, STACOM 2010, and Cardiac Electrophysiological Simulation Challenge, CESC 2010, Held in Conjunction with MICCAI 2010, Beijing, China, September 20, 2010. Proceedings* (2010), vol. 6364 of *Lecture Notes in Computer Science*, Springer.

[CR08] Caban J. J., Rheingans P.: Texture-based transfer functions for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1364–1371.

[CS99] Cai W., Sakas G.: Data intermixing and multi-volume rendering. *Computer Graphics Forum 18* (September 1999), 359–368(12).

[CSC06] Correa C. D., Silver D., Chen M.: Discontinuous displacement mapping for volume graphics. In *Eurographics/IEEE VGTC Workshop on Volume Graphics* (Boston, Massachusetts, USA, 2006), Eurographics Association, pp. 9–16.

[CSC10] Correa C. D., Silver D., Chen M.: Constrained illustrative volume deformation. *Computers and Graphics 34*, 4 (2010), 370 – 377.

[CSW*03] Chen M., Silver D., Winter A. S., Singh V., Cornea N.: Spatial transfer functions: a unified approach to specifying deformation in volume modeling and animation. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics* (New York, NY, USA, 2003), VG '03, ACM, pp. 35–44.

[CT82] Cook R. L., Torrance K. E.: A reflectance model for computer graphics. *ACM Trans. Graph. 1*, 1 (Jan. 1982), 7–24.

[Cyb14] Cyber-Anatomy:. `http://www.cyber-anatomy.com`, 2014. Accessed: 2014-03-27.

[DA12] Domik G., Arens S.: Früherkennung von Plaque in den Koronargefäßen - Räumliche Wahrnehmung bei der Untersuchung von Koronararterien in CT-Datensätzen. In *ForschungsForum Paderborn* (2012), Risch N., (Ed.).

[DASH11] Domik G., Arens S., Scharlau I., Hilkenmeier F.: How useful is computer graphics for medical diagnoses. In *Informatik 2011: Workshop Emerging Technologies for Medical Diagnosis and Therapy* (2011), vol. 192 of *LNI*.

[DATS12] Domik G., Arens S., Tünnermann J., Scharlau I.: Evaluierung medizinischer Volumenrendering-Algorithmen durch empirische Studien. In *FIFF-Kommunikation* (2012).

[DBB*08] DeLaurier A., Burton N., Bennett M., Baldock R., Davidson D., Mohun T., Logan M.: The mouse limb anatomy atlas: An interactive 3d tool for studying embryonic limb patterning. *BMC Developmental Biology 8*, 1 (2008), 1–7.

[DCLK03] Dong F., Clapworthy G. J., Lin H., Krokos M. A.: Nonphotorealistic rendering of medical volume data. *IEEE Comput. Graph. Appl. 23*, 4 (July 2003), 44–52.

[DH92] Danskin J., Hanrahan P.: Fast algorithms for volume ray tracing. In *VVS '92: Proceedings of the 1992 workshop on Volume visualization* (New York, NY, USA, 1992), ACM Press, pp. 91–98.

[DSAS11] Domik G., Steffen F., Arens S., Scharlau I.: Usefulness of style transfer functions in medical diagnosis. In *SIGGRAPH Posters* (2011), ACM, p. 89.

[Duc79] Duchon C. E.: Lanczos filtering in one and two dimensions. *Journal of Applied Meteorology 18*, 8 (2014/06/07 1979), 1016–1022.

[DVW13] Drew T., Võ M. L.-H., Wolfe J. M.: The invisible gorilla strikes again: Sustained inattentional blindness in expert observers. *Psychological Science 24*, 9 (2013), 1848–1853.

[DW85] Dippé M. A. Z., Wold E. H.: Antialiasing through stochastic sampling. In *Proceedings of SIGGRAPH '85* (1985), pp. 69–78.

[EHK*06] Engel K., Hadwiger M., Kniss J., Rezk-Salama C., Weiskopf D.: *Real-Time Volume Graphics*. AK Peters, 2006.

[FAF13] Finegold J. A., Asaria P., Francis D. P.: Mortality from ischaemic heart disease by country, region, and age: Statistics from world health organisation and united nations. *International journal of cardiology 168*, 2 (09 2013), 934–945.

[Fee09] Feeman T.: *The Mathematics of Medical Imaging: A Beginner's Guide*. Springer Undergraduate Texts in Mathematics and Technology. Springer, 2009.

[FLAK11] Forsberg D., Lundstroem C., Andersson M., Knutsson H.: Model-based transfer functions for efficient visualization of medical image volumes. In *Image Analysis*, Heyden A., Kahl F., (Eds.), vol. 6688 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 592–603.

[FNVV98] Frangi A., Niessen W., Vincken K., Viergever M.: Multiscale vessel enhancement filtering. In *Medical Image Computing and Computer-Assisted Interventation - MICCAI'98*, Wells W., Colchester A., Delp S., (Eds.), vol. 1496 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1998, pp. 130–137.

[GAD14] Gmyr R., Arens S., Domik G.: *Interactive GPU Ray Casting using Progressive Blue Noise Sampling*. Tech. Rep. tr-ri-14-339, University of Paderborn, Department of Computer Science, June 2014.

[GC58] Gray H., Carter H. V.: *Anatomy : descriptive and surgical*. J.W. Parker, London, 1858.

[GD06] Goetz F., Domik G.: Visual shaditor: a seamless way to compose high-level shader programs. In *ACM SIGGRAPH 2006 Research posters* (New York, NY, USA, 2006), SIGGRAPH '06, ACM.

[Ger14] German Cancer Research Center: Mitk: The medical interaction toolkit. http://www.mitk.org, 2014. Accessed: 2014-05-22.

[GGK] Gröller M., Glaeser G., Kastner J.: Stag beetle. http://www.cg.tuwien.ac.at/research/publications/2005/dataset-stagbeetle/.

[Gra18] Gray H.: Anatomy of the human body. http://www.bartleby.com/107, 1918. Accessed: 2014-03-27.

[GTGB84] Goral C. M., Torrance K. E., Greenberg D. P., Battaile B.: Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Comput. Graph. 18*, 3 (Jan. 1984), 213–222.

[Har04] Hargreaves S.: *ShaderX3: Advanced Rendering with DirectX and OpenGL*. Charles River Media, 2004, ch. Generating Shaders from HLSL Fragments, pp. 555–568.

[HBR*92] Hohne K.-H., Bomans M., Riemer M., Schubert R., Tiede U., Lierse W.: A volume-based anatomical atlas. *Computer Graphics and Applications, IEEE 12*, 4 (July 1992), 72–78.

[HKG00] Hladuvka J., König A., Gröller E.: Curvature-based transfer functions for direct volume rendering. In *In Bianca Falcidieno, editor, Spring Conference on Computer Graphics 2000* (2000), pp. 58–65.

[HSS*05] Hadwiger M., Sigg C., Scharsach H., Bühler K., Gross M. H.: Real-time ray-casting and advanced shading of discrete isosurfaces. *Comput. Graph. Forum (Proc. Eurographics) 24*, 3 (2005), 303–312.

[IEE91] IEEE: Ieee standard computer dictionary. a compilation of ieee standard computer glossaries. *IEEE Std 610* (1991).

[ISO01] ISO/IEC: *ISO/IEC 9126. Software engineering – Product quality.* ISO/IEC, 2001.

[ISO06] ISO: *ISO 9241-110: Ergonomics of human-system interaction - Part 110: Dialogue principles.* ISO, 2006.

[IVM*10] Ionasec R., Voigt I., Mihalef V., Grbic S., Vitanovski D., Wang Y., Zheng Y., Hornegger J., Navab N., Georgescu B., Comaniciu D.: Patient-specific modeling of the heart: Applications to cardiovascular disease management. In *Statistical Atlases and Computational Models of the Heart*, Camara O., Pop M., Rhode K., Sermesant M., Smith N., Young A., (Eds.), vol. 6364 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 14–24.

[JSYR12] Jönsson D., Sundén E., Ynnerman A., Ropinski T.: Interactive Volume Rendering with Volumetric Illumination. Cani M.-P., Ganovelli F., (Eds.), Eurographics Association, pp. 53–74.

[Ju05] Ju T.: *Building a 3D Atlas of the Mouse Brain.* PhD thesis, Dept. of Computer Science, Rice Univ., 2005.

[KD98] Kindlmann G., Durkin J. W.: Semi-automatic generation of transfer functions for direct volume rendering. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization* (New York, NY, USA, 1998), ACM, pp. 79–86.

[KFW*02] Kanitsar A., Fleischmann D., Wegenkittl R., Felkel P., Gröller E.: Cpr - curved planar reformation. In *IEEE Visualization* (2002).

[KG79] Kay D. S., Greenberg D.: Transparency for computer synthesized images. *SIGGRAPH Comput. Graph. 13*, 2 (Aug. 1979), 158–164.

[KG00] König A., Gröller M. E.: *Mastering Transfer Function Specification by using VolumePro Technology.* Tech. Rep. TR-186-2-00-07, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, Mar. 2000.

[KGK*12] Kirisli H., Gupta V., Kirschbaum S., Rossi A., Metz C., Schaap M., Geuns R., Mollet N., Lelieveldt B., Reiber J., Walsum T., Niessen W.: Comprehensive visualization of multimodal cardiac imaging data for assessment of coronary artery disease: first clinical results of the smartvis tool. *International Journal of Computer Assisted Radiology and Surgery 7*, 4 (2012), 557–571.

[KH84] Kajiya J. T., Herzen B. P. V.: Ray tracing volume densities. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1984), ACM Press, pp. 165–174.

[KH01] Keller A., Heidrich W.: Interleaved sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), pp. 269–276.

[Kin02] Kindlmann G.: Transfer functions in direct volume rendering: Design, interface, interaction. In *SIGGRAPH 2002 Course Notes* (2002).

[Kit14] Kitware Inc.: Volview: A volume visualization system. `http://www.volview.org`, 2014. Accessed: 2014-05-22.

[KKH01] Kniss J., Kindlmann G., Hansen C.: Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *VIS '01: Proceedings of the conference on Visualization '01* (Washington, DC, USA, 2001), IEEE Computer Society, pp. 255–262.

[KKH02] Kniss J., Kindlmann G., Hansen C.: Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 270–285.

[Kok12] Kokerbeck S.: *Elastische, modellbasierte Deformation von Volumendaten.* Master's thesis, University of Paderborn, June 2012.

[KPH*03] Kniss J., Premoze S., Hansen C., Shirley P., McPherson A.: A model for volume lighting and modeling. *Visualization and Computer Graphics, IEEE Transactions on 9*, 2 (April 2003), 150–162.

[KRHH11] Kratz A., Reininghaus J., Hadwiger M., Hotz I.: *Adaptive Screen-Space Sampling for Volume Ray-Casting.* Tech. rep., ZIB, 2011.

[KSH*11] Kainz B., Steinberger M., Hauswiesner S., Khlebnikov R., Schmalstieg D.: Stylization-based ray prioritization for guaranteed frame rates. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2011), NPAR '11, ACM, pp. 43–54.

[KVUS*05] Kniss J., Van Uitert R., Stephens A., Li G.-S., Tasdizen T., Hansen C.: Statistically quantitative volume visualization. In *Visualization, 2005. VIS 05. IEEE* (Oct 2005), pp. 287–294.

[KW03] Krüger J., Westermann R.: Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), VIS '03, IEEE Computer Society, pp. 38–.

[KWTM03] Kindlmann G., Whitaker R., Tasdizen T., Moller T.: Curvature-based transfer functions for direct volume rendering: Methods and applications. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 67.

[KY97] Kurzion Y., Yagel R.: Interactive space deformation with hardware-assisted rendering. *IEEE Comput. Graph. Appl. 17*, 5 (Sept. 1997), 66–77.

[Lam60] Lambert J.: *Photometria sive De mensura et gradibus luminis, colorum et umbrae.* Sumptibus viduae Eberhardi Klett, typis Christophori Petri Detleffsen, 1760.

[LC87] Lorensen W. E., Cline H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1987), SIGGRAPH '87, ACM, pp. 163–169.

[Lev88] Levoy M.: Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3 (1988), 29–37.

[Lev90] Levoy M.: Volume rendering by adaptive refinement. *The Visual Computer 6*, 1 (1990), 2–7.

[LF09] Lux C., Fröhlich B.: Gpu-based ray casting of multiple multi-resolution volume datasets. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II* (Berlin, Heidelberg, 2009), ISVC '09, Springer-Verlag, pp. 104–116.

[LK08] Lipton M. L., Kanal E.: *Totally Accessible MRI: A User's Guide to Principles, Technology, and Applications*. Springer, New York, NY, 2008.

[LLHY09] Lindholm S., Ljung P., Hadwiger M., Ynnerman A.: Fused multi-volume dvr using binary space partitioning. *Comput. Graph. Forum 28*, 3 (2009), 847–854.

[LM05] Li S., Mueller K.: Accelerated, high-quality refraction computations for volume graphics. In *Proceedings of the Fourth Eurographics / IEEE VGTC conference on Volume Graphics* (Aire-la-Ville, Switzerland, 2005), VG'05, Eurographics Association, pp. 73–81.

[LMT*03] Lu A., Morris C. J., Taylor J., Ebert D. S., Hansen C., Rheingans P., Hartner M.: Illustrative interactive stipple rendering. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (Apr. 2003), 127–138.

[LR11] Lindemann F., Ropinski T.: About the influence of illumination models on image comprehension in direct volume rendering. *Visualization and Computer Graphics, IEEE Transactions on 17*, 12 (2011), 1922–1931.

[Ma99] Ma K.-L.: Image graphs-a novel approach to visual data exploration. In *Visualization '99. Proceedings* (Oct 1999), pp. 81–88.

[MAB*97] Marks J., Andalman B., Beardsley P. A., Freeman W., Gibson S., Hodgins J., Kang T., Mirtich B., Pfister H., Ruml W., Ryall K., Seims J., Shieber S.: Design galleries: a general approach to setting parameters for computer graphics and animation. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 389–400.

[McC77] McCall J.: *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager*. No. ADA049055 in 1-3. General Electric, November 1977.

[McG05] McGuire M.: *ShaderX4: Advanced rendering techniques*. Charles River Media, 2005, ch. The SuperShader, pp. 485–498.

[MF14] MeVis Medical Solutions AG, Fraunhofer MEVIS: Mevislab: A development environment for medical image processing and visualization. `http://www.mevislab.de`, 2014. Accessed: 2014-05-22.

[Mit87] Mitchell D. P.: Generating antialiased images at low sampling densities. *SIGGRAPH Comput. Graph. 21*, 4 (1987), 65–72.

[MSPK06] McGuire M., Stathis G., Pfister H., Krishnamurthi S.: Abstract shade trees (preprint). In *Symposium on Interactive 3D Graphics and Games* (March 2006).

[MSRMH09] Meyer-Spradow J., Ropinski T., Mensmann J., Hinrichs K.: Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations. *Computer Graphics and Applications, IEEE 29*, 6 (nov.-dec. 2009), 6 –13.

[MSRMH10] Meyer-Spradow J., Ropinski T., Mensmann J., Hinrichs K.: Interactive Design and Debugging of GPU-based Volume Visualizations. In *Computer Graphics Theory and Applications* (2010), pp. 239–245. short paper.

[Net10] Netter F.: *Atlas of Human Anatomy.* Netter Basic Science. Elsevier Health Sciences, 2010.

[NMK*06] Nealen A., Müller M., Keiser R., Boxerman E., Carlson M.: Physically based deformable models in computer graphics. *Computer Graphics Forum 25*, 4 (2006), 809–836.

[OHG*08] Oeltze S., Hennemuth A., Glasser S., Kühnel C., Preim B.: Glyph-Based Visualization of Myocardial Perfusion Data and Enhancement with Contractility and Viability Information. In *VCBM* (2008), pp. 11–20.

[Osi14] Osirix: Dicom sample image sets. `http://www.osirix-viewer.com/datasets/`, 2014. Accessed: 2014-05-30.

[PB07] Preim B., Bartz D.: *Visualization in Medicine - Theory, Algorithms, and Applications.* Morgan Kaufman Publishers, Burlington, USA, 0 2007.

[PBB05] Pock T., Beichel R., Bischof H.: A novel robust tube detection filter for 3d centerline extraction. In *18th Symposium on Operating System Principles (SOSP* (2005), Springer-Verlag, pp. 481–490.

[PD84] Porter T., Duff T.: Compositing digital images. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1984), ACM Press, pp. 253–259.

[PF05] Pharr M., Fernando R.: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005.

[Phe06] Phelps M.: *PET: Physics, Instrumentation, and Scanners*. Springer, 2006.

[PHF07] Plate J., Holtkaemper T., Froehlich B.: A flexible multi-volume shader framework for arbitrarily intersecting multi-resolution datasets. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (Nov. 2007), 1584–1591.

[Pho75] Phong B. T.: Illumination for computer generated pictures. *Commun. ACM 18*, 6 (June 1975), 311–317.

[PLB*01] Pfister H., Lorensen B., Bajaj C., Kindlmann G., Schroeder W., Avila L., Raghu K., Machiraju R., Lee J.: The transfer function bake-off. *Computer Graphics and Applications, IEEE 21*, 3 (May/Jun 2001), 16–22.

[PRMH10] Prassni J.-S., Ropinski T., Mensmann J., Hinrichs K. H.: Shape-based transfer functions for volume visualization. In *IEEE Pacific Visualization Symposium (PacificVis 2010)* (mar 2010), pp. 9–16.

[PTK*08] Prior F., Tarbox L., Krych J., Paladini G., Moeller T., Pearson J., Smith K.: Xip: the extensible imaging platform. In *AMIA Symposium* (2008).

[RBE08a] Rössler F., Botchen R. P., Ertl T.: Dynamic Shader Generation for Flexible Multi-Volume Visualization. In *Proceedings of IEEE Pacific Visualization Symposium 2008 (PacificVis '08)* (2008), pp. 17–24.

[RBE08b] Rössler F., Botchen R. P., Ertl T.: Dynamic Shader Generation for GPU-based Multi-Volume Raycasting. *Computer Graphics and Applications 28*, 5 (2008), 66–77.

[RBGV08] Rautek P., Bruckner S., Gröller E., Viola I.: Illustrative visualization: New technology or useless tautology? *SIGGRAPH Comput. Graph. 42*, 3 (Aug. 2008), 4:1–4:8.

[RBS05] Röttger S., Bauer M., Stamminger M.: Spatialized transfer functions. In *EuroVis* (2005), pp. 271–278.

[RGW*03] Roettger S., Guthe S., Weiskopf D., Ertl T., Strasser W.: Smart hardware-accelerated volume rendering. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 231–238.

[RPLH11] Rieder C., Palmer S., Link F., Hahn H. K.: A Shader Framework for Rapid Prototyping of GPU-Based Volume Rendering. *Computer Graphics Forum (Special Issue on Eurographics Symposium on Visualization) 30*, 3 (2011), 1031–1040.

[RYLD06] Rohen J., Yokochi C., Lütjen-Drecoll E.: *Color Atlas of Anatomy: A Photographic Study of the Human Body.* Lippincott Williams & Wilkins, 2006.

[SBLM*09] Smith-Bindman R., Lipson J., Marcus R., Kim K.-P., Mahesh M., Gould R., Berrington de González A., Miglioretti D. L.: Radiation dose associated with common computed tomography examinations and the associated lifetime attributable risk of cancer. *Arch Intern Med 169*, 22 (2009), 2078–86.

[Sch07] Schwier M.: *Interaktive Transferfunktionen und Rendering Techniken für medizinische Volmendaten.* Master's thesis, University of Paderborn, 2007.

[SCI14] SCI Institute:, 2014. SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI), Download from: `http://www.scirun.org`.

[SH07] Scheuermann T., Hensley J.: Efficient histogram generation using scattering on gpus. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 33–37.

[Sin12] Sindelar S.: *Anatomy-aware Whole Heart Modeling using Splines and Mesh Fitting.* Master's thesis, University of Paderborn, January 2012.

[SMvW*09] Schaap M., Metz C., van Walsum T., van der Giessen A., Weustink A., Mollet N., Bauer C., Bogunović H., Castro C., Deng X., Dikici E., O'Donnell T., Frenay M., Friman O., Hoyos M. H., Kitslaar P., Krissian K., Kühnel C., Luengo-Oroz M. A., Orkisz M., Smedby Ö., Styner M., Szymczak A., Tek H., Wang C., Warfield S. K., Zambal S., Zhang Y., Krestin G. P., Niessen W.: Standardized evaluation methodology and reference database for evaluating coronary artery centerline extraction algorithms. *Medical Image Analysis 13/5* (2009), 701–714.

[SSS09] Schulte E., Schünke M., Schumacher U.: *Prometheus - Lernatlas der Anatomie: Innere Organe ; 118 Tabellen.* Prometheus. Georg Thieme, 2009.

[TD07] Trapp M., Döllner J.: Automated combination of real-time shader programs. In *Eurographics 2007 Shortpaper* (2007), Cignoni P., Sochor J., (Eds.), The Eurographics Association, pp. 53–56.

[TLM03] Tzeng F.-Y., Lum E. B., Ma K.-L.: A novel interface for higher-dimensional classification of volume data. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 66.

[TPD06] Tappenbeck A., Preim B., Dicken V.: Distance-based transfer function design: Specification methods and applications. In *SimVis* (2006), pp. 259–274.

[Ves43] Vesalius A.: De humani corporis fabrica. `http://vesalius.northwestern.edu`, 1543. Accessed: 2014-03-26.

[Vis14] Visible Human Project:. `http://www.nlm.nih.gov/research/visible/`, 2014. Accessed: 2014-03-27.

[Vor14] Voreen: Volume rendering engine. `http://www.voreen.org/`, 2014. Accessed: 2014-05-22.

[Wad71] Wadsworth C. P.: *Semantics and pragmatics of the lambda calculus.* Ph.D. thesis, Programming Research Group, Oxford University, Sept. 1971.

[WEE02] Weiskopf D., Engel K., Ertl T.: Volume clipping via per-fragment operations in texture-based volume visualization. In *Proceedings of the Conference on Visualization '02* (Washington, DC, USA, 2002), VIS '02, IEEE Computer Society, pp. 93–100.

[WEE03] Weiskopf D., Engel K., Ertl T.: Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Transactions on Visualization and Computer Graphics 9*, 3 (July 2003), 298–312.

[Whi80] Whitted T.: An improved illumination model for shaded display. *Commun. ACM 23*, 6 (June 1980), 343–349.

[Wik14] WIKIPEDIA: Hounsfield scale. `http://en.wikipedia.org/wiki/Hounsfield_scale`, 2014.

[WK09a] WESARG S., KIRSCHNER M.: 3d visualization of medical image data employing 2d histograms. *International Conference in Visualisation 0* (2009), 153–158.

[WK09b] WESARG S., KIRSCHNER M.: Structure size enhanced histogram. In *Bildverarbeitung für die Medizin* (2009), pp. 16–20.

[WLC*12] WAN Y., LEWIS A. K., COLASANTO M., VAN LANGEVELD M., KARDON G., HANSEN C.: A practical workflow for making anatomical atlases for biological research. *IEEE Computer Graphics and Applications 32*, 5 (2012), 70–80.

[WPM*10] WEESE J., PETERS J., MEYER C., WÄCHTER I., KNESER R., LEHMANN H., ECABERT O., BARSCHDORF H., HANNA R., WEBER F. M., DÖSSEL O., LORENZ C.: The generation of patient-specific heart models for diagnosis and interventions. In *Statistical Atlases and Computational Models of the Heart*, Camara O., Pop M., Rhode K., Sermesant M., Smith N., Young A., (Eds.), vol. 6364 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 25–35.

[WZMK05] WANG L., ZHAO Y., MUELLER K., KAUFMAN A.: The magic volume lens: an interactive focus+context technique for volume rendering. In *Visualization, 2005. VIS 05. IEEE* (Oct 2005), pp. 367–374.

[XYZ] XYZ RGB INC.: Stanford asian dragon. `http://graphics.stanford.edu/data/3Dscanrep/`.

[Yel83] YELLOTT J. I. J.: Spectral consequences of photoreceptor sampling in the rhesus retina. *Science 221* (1983), 382–385.

[ZBG*07] ZHENG Y., BARBU A., GEORGESCU B., SCHEUERING M., COMANICIU D.: Fast automatic heart chamber segmentation from 3d ct data using marginal space learning and steerable features. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on* (Oct 2007), pp. 1–8.

[ZEAD08] ZUKIC D., ELSNER A., AVDAGIC Z., DOMIK G.: Neural networks in 3d medical scan visualization. In *3IA 2008: International Conference on Computer Graphics and Artificial Intelligence* (New York, NY, USA, 2008), ACM Press/Addison-Wesley Publishing Co., pp. 183–190.

[Zyg14] Zygote Media Group:. http://www.zygote.com, 2014. Accessed: 2014-03-27.

List of Figures

List of Tables