High-Level User Interfaces for Transfer Function Design based on Semantic Models

Diploma Thesis in Computer Science

submitted

by

Maik Keller

Written at

Computer Graphics and Multimedia Systems Group
Faculty 12
University of Siegen, Germany

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Siegen, den 24. August 2006

Übersicht

Betrachtet man das Gebiet der Volumenvisualisierung vom technologischen Standpunkt aus, so sind dort in den letzten Jahren eine Reihe neuer und nützlicher Verfahren entwickelt worden, wie beispielsweise multidimensionale Transferfunktionen und ausgeklügelte Shadingmodelle. In der Praxis ist die moderne Volumenvisualisierung allerdings noch nicht weit verbreitet. Die Benutzer berichten von Problemen, die hauptsächlich auf die Bewältigung der Komplexität visueller Parametereinstellungen zurückzuführen sind. Genau genommen liegen die Schwierigkeiten in der Spezifizierung der optischen Eigenschaften, die mit Hilfe von Transferfunktionen vorgenommen werden. Die bereits existierenden Automatisierungsverfahren bieten nicht die Flexibilität, die erforderlich ist, um gute Ergebnisse für unterschiedliche Datensätze liefern zu können. Ihr Einsatz ist darüber hinaus oft auf allgemeine Anwendungsfälle beschränkt, da sie für die Darstellung bestimmter Strukturen nicht spezifisch genug sind. Alternativ kann die Transferfunktion auch manuell eingestellt werden. Dies ist jedoch ein sehr langwieriger und zeitaufwändiger Prozess und selbst für Visualisierungsexperten ist der Einfluss einer Parameteränderung auf das Ergebnisbild oft schwer vorhersehbar.

Bisweilen fehlt eine klare Semantik, um schnell, effektiv und zielorientiert arbeiten zu können. Die vorliegende Ausarbeitung schlägt daher eine Methode zur Erstellung semantischer Modelle zur Volumenvisualisierung vor. Dabei wird ein aus dem Bereich der Computer-Animation bekanntes Verfahren in Verbindung mit *Principal Component Analysis* angewendet. Dies erlaubt es Visualisierungsexperten, Modelle zu entwickeln, die anschließend von Benutzern auf intuitive Weise angewendet werden können.

Diese Ausarbeitung führt den Leser in die theoretischen Grundlagen der Entwicklung semantischer Modelle ein. Im weiteren Verlauf wird ein Framework vorgestellt, mit dessen Hilfe diese Modelle realisiert werden können. Anschließend werden zwei Modelle beispielhaft für die Visualisierung medizinischer Datensätze erläutert (Computertomographie und Magnetresonanztomographie).

Abstract

From the technological point of view, a variety of new and useful features has been added to volume rendering algorithms in recent years, including multi-dimensional transfer functions and sophisticated shading models. However, advanced volume rendering techniques are not widely used in practice. This is mainly due not to technological reasons, but to difficulties in managing the complexity of visual parameter assignment which is done by means of transfer functions. Users report on problems concerning the process of specifying optical properties for datasets. On the one hand, automatic approaches of designing a transfer function are often not adaptable and flexible enough in order to realize a desired visualization. On the other hand, manual transfer function assignment is a challenging task even for expert-users, since appropriate user interfaces, especially for multi-dimensional transfer functions, are difficult to operate.

This thesis proposes an approach of transfer function design which deals with semantics. For this purpose, an additional abstraction layer for parametric models of transfer functions is introduced which facilitates the specification of optical properties. The method of calculation is based on principal component analysis and adapts concepts from the field of computer animation.

Additionally, a framework is presented which allows visualization experts to design high-level transfer function models which can be used intuitively by non-expert users. As a result, user interfaces are obtained which provide semantic information for specific application areas. Within this thesis, two semantic transfer function models for medical visualization are developed, namely Magnetic Resonance Imaging and Computed Tomography Angiography.

Besides the practical work, the following aspects are discussed in this thesis: an overview of the basic techniques which this work implies, the development of a theoretical semantic transfer function model, the practical realization and implementation of the transfer function model into a framework, and the visualization of the results based on semantic models.

Acknowledgements

First of all, I would like to thank my supervisors Prof. Dr. Andreas Kolb and Dr. Christof Rezk-Salama of the Computer Graphics Group of the University of Siegen, Germany, for their support, their supervision of my work and the opportunity to visit Siemens Corporate Research (SCR) in Princeton, New Jersey, USA, within the scope of this project. Dr. Rezk-Salama had always time for explaining complex topics and sophisticated techniques in an understandable way. It was a privilege to develop the software of this thesis together with such an experienced visualization expert.

I am very much obliged to the staff of the group of Dr. Gianluca Paladini at the Department of Imaging and Visualization at SCR. Therefore, I am especially grateful to my supervisors in Princeton: Dr. Paladini, Dr. Klaus Engel and Dr. Thomas Möller for helpful discussions, their valuable advice and new ideas. I was privileged to work with such an experienced and capable group of people. I would also like to thank Peter Kohlmann and Julien Gein for some software development and providing support concerning the developer's framework.

The CT angiography data was generously provided by Dr. Bernd Tomandl of the Klinik of Neuroradiologie, Bremen, Germany. Furthermore, the MRI data is by courtesy of Dr. Christopher Nimsky of the Department of Neurosurgery, University of Erlangen-Nuremberg, Germany. I would also like to thank Dr. Matthias Richter of the Ev. Krankenhaus, Plettenberg, Germany, and Dr. Mischa Braun of the Charite, Berlin, Germany, for providing their medical experience.

Finally, and above all, I would like to express my deep gratitude to my family, my girlfriend Kathrin Ohrndorf (who also proof-read this thesis), and her family, for their love and their absolutely extraordinary emotional and financial support.

Maik Keller

Preface

This work describes the results of the diploma thesis project that I carried out as a student of the Computer Graphics Group of Prof. Dr. Andreas Kolb of the University of Siegen. The preparations of this work started during my internship at Siemens Corporate Research (SCR Princeton, New Jersey, USA). This project was established by my supervisor Dr. Christof Rezk-Salama, Computer Graphics Group of the University of Siegen, and by the group of Dr. Gianluca Paladini and my supervisors Dr. Klaus Engel and Dr. Thomas Möller of the Department of Imaging and Visualization at SCR.

Within the scope of this thesis, a high-level user interface based on semantic models is proposed in order to facilitate transfer function adjustment in volumetric datasets. The existing object-oriented C++ visualization framework "*OpenQVis*" is extended with several transfer function models and editors. The results of the semantic models are visualized on general purpose graphics hardware by volume rendering.

This thesis is divided into 8 chapters. After the introduction in chapter 1, an overview of the related work is given in chapter 2, followed by a review of the theoretical basis in chapter 3, including concepts from computer animation. Subsequently, theoretical aspects of a new semantic model are introduced in this context in chapter 4. Chapter 5 proposes efficient ways of the design of semantic models in practical cases. In chapter 6, a realization of the approach extending the OpenQVis framework is described. The results of the new transfer function model which are based on two medical application scenarios are presented in chapter 7. Chapter 8 concludes the presented thesis and comments on future work. Finally, the appendix contains information about the implemented semantic models and reference datasets.

Abbreviations

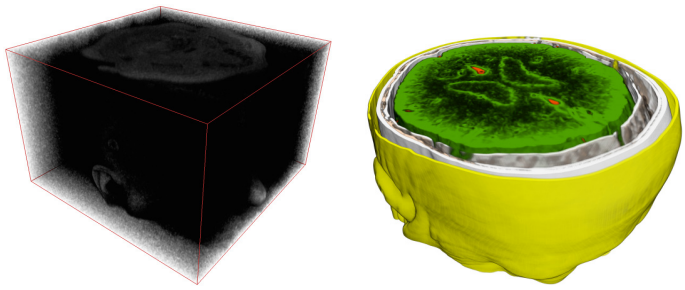
1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
CT	Computed Tomography
CTA	Computed Tomography Angiography
DTD	Document Type Definition
GPU	Graphics Processing Unit
GUI	Graphical User Interface
MR	Magnetic Resonance
MRI	Magnetic Resonance Imaging
PCA	Principal Component Analysis
RGB	Red, Green, Blue
RGBA	Red, Green, Blue, Alpha
SCR	Siemens Corporate Research
SVD	Singular Value Decomposition
UI	User Interface
UML	Unified Modeling Language
XML	Extensible Markup Language

Contents

Chapter 1

Introduction

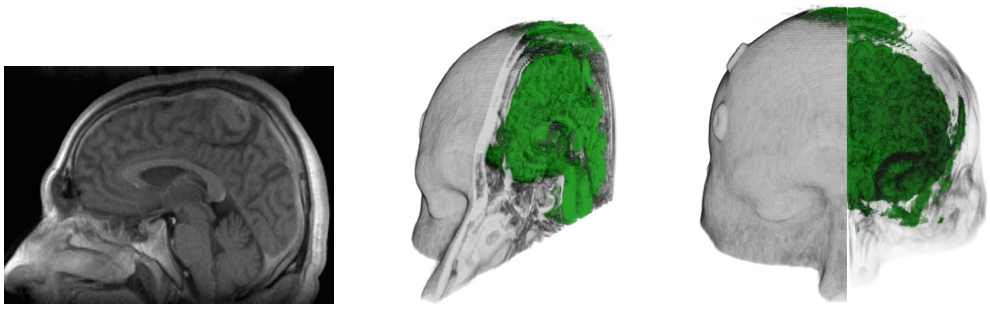
In recent years, interactive high-quality volume rendering has advanced from expensive high-end graphics workstations to inexpensive desktop computers. That is one of the reasons why volume rendering has gained great importance in many application areas such as medical applications, natural and computational science, industrial design, and engineering. In the past, problems with memory management have been successfully solved, making it possible to navigate interactively through a volume dataset in real time. In addition to this, a variety of useful features has been added to volume rendering algorithms, including multi-dimensional transfer functions and sophisticated shading models.



(a) (b)

Figure 1.1: Volumetric datasets and transfer functions. Figure 1.1(a) shows a volume dataset without any appropriate transfer function. There are no clear structures visible. In Figure 1.1(b) a transfer function is applied to a CTA dataset which separates the structures contained in the data.

In practice, and especially in medical imaging, however, advanced volume rendering techniques are not used as frequently as you would expect them to be used. This is mainly due not to technological reasons, but to difficulties in managing the complexity of visual parameter assignment. The process of specifying optical properties with regard to datasets, which is usually done by means of transfer functions, is a tedious and time consuming process for most users. The visual effect of parameter modification often is hardly predictable even for visualization experts. In general, there are no intuitive



(a) (b) (c)

Figure 1.2: MRI head in slice and volumetric point of view. Figure 1.2(a) illustrates a traditional slice image. Figures 1.2(b) and 1.2(c) are examples of volume rendering images of the same MRI volume dataset.

user interfaces which could offer possibilities to access such features without profound knowledge of the algorithms. An alternative might be to take automatic and semi-automatic approaches. But, on the one hand, they are difficult to adapt to a wide range of datasets and, on the other hand, they often are not specific enough to realize a desired visualization. If automatic approaches fail, non-expert users are left alone and end up in struggling with parameters (see Figure 1.1).

At least there are two different ways which determine a transfer function. The first way is to think of a transfer function as a large color table. For each data value, optical properties can be obtained by a table lookup. In the field of multi-dimensional transfer functions, however, this representation is not appropriate due to the memory requirements for storing the lookup table. The second way is to introduce an additional layer of abstraction. The modern user interfaces for transfer function specification use simple shapes such as boxes, ramps, and trapezoids as primitive objects to manipulate the transfer function by moving the primitives directly on the screen. The main benefit of primitive objects is the significant reduction of the degrees of freedom that the user must manage in order to establish a suitable transfer function.

Yet the manipulation of primitives is still a difficult task, especially for non-expert users in a multi-dimensional transfer function environment. The transfer function's specification often is a trial and error process, because of the number of parameters, which is still too high and confusing. Due to this fact, the approval of volume rendering in scientific visualization is not distributed as widely as it could be in everyday use.

With regard to the visualization of medical data applied by physicians, a lack of clear semantics has been noticed. Physicians often suggest to make the vessels more sharp, to fade out the soft tissue, and to improve the contrast between skin and bone, for instance. Even visualization experts who are familiar with the underlying transfer function model and the respective editor have problems to figure out which modifications of the primitives will yield the desired result.

Difficulties arising from complex applications being used by non-expert users are not a problem

which is unique to visualization only. Other fields like computer animation have already found solutions to a large degree. This thesis adapts concepts from computer animation to the field of direct volume rendering applications in order to improve the usability of transfer function design. The goal of this work is to provide a high-level user interface based on semantics, mainly for clinical use, which is easy to apply and adaptable to many application scenarios. Therefore, the user interface contains a list of structures which the dataset consists of. In contrast to a *Magnetic Resonance Imaging* (MRI) dataset, where the separation of "*skin*" and "*brain*" structures might be important (see Figure 1.2), structures like "*soft tissue*", "*bone*", and "*vessels*" might be interesting in a *Computed Tomography Angiography* (CTA), for example. Additionally, a list of attributes, such as "*visibility*", "*contrast*", "*sharpness*", and "*color*" completes each entry of the list of structures and, based on clear semantics, it enables the user to explore the dataset interactively as well as and goal-directed . Finally, the underlying technical model translates the manipulations into a modification of the transfer function.

This approach allows visualization experts to design transfer function models for specific examination purposes which afterwards can be used intuitively by non-expert users such as physicians, for example.

Chapter 2

Related Work

Direct volume rendering research tends to focus either on making the rendering algorithms faster or on extending existing rendering methods to work with a wider variety of datasets. From the technological point of view, new and useful features have been added to volume rendering algorithms. A lot of sophisticated techniques try to solve the volume rendering integral in real-time, including the shear-warp algorithm [LL94], 3D texture slicing [CCF95, WVW94], 2D texture mapping [RSEB+00], pre-integration [EKE01], GPU-based rendering methods [KW03, RGWE03, SSKE05, EWRS+06, EWRS+05], and special purpose hardware. The ideas which are described in this diploma thesis are independent from the specific implementation of the rendering algorithm.

Most approaches with respect to direct volume rendering for scientific purposes are currently based on a simplified physical model of light transport [Lev88a], in which light is assumed to travel along straight lines. Thus, the integration of radiative energy along rays of sight is done by considering physical quantities describing the light: emission and absorption are assigned to every voxel. Kniss et. al. [KPHE02] develop a more complex and elaborate model of light transport involving shadows and translucency. Their model requires the specification of additional physical quantities.

If the dataset's scalar values alone are not sufficient to achieve the optical properties required for rendering, so called multi-dimensional transfer functions[1] may be used. Multi-dimensional transfer functions are basically highly superior to traditional one dimensional (1D) transfer functions. Additionally to the scalar values, the magnitude of the first and second order derivatives of the scalar field are frequently used to expand the domain of the transfer function. Vega et al. demonstrate the benefit of two dimensional (2D) transfer functions for visualizing blood vessels in CTA data [HST+04]. Kniss et al. use multi-dimensional transfer functions to classify co-registered multivariate magnetic resonance imaging (MRI) data [KSW+04] and Kindlmann et al. propose an algorithm to detect the material boundaries based on the first and second derivatives of the scalar data [KD98]. Kindlmann's approach is probably the best method currently available to visualize shapes and structures in an unknown volume dataset. However, the complexity of parameter specification significantly increases with each additional dimension.

[1] For further information about multi-dimensional transfer functions see section 3.2.

In recent years, a lot of research on automatic and semi-automatic transfer function design has been made. These approaches can be categorized into image-driven as well as data-driven techniques. The major purpose of a transfer function is to create meaningful images. In consequence, image-driven techniques for automatic transfer function design analyze the information contained in the images which are generated with different parameter settings. Methods for setting visual parameters reported in literature either explore the parameter space interactively (*interactive evolution* [Koc90, Sim91, TL92]) or search for optimal settings based on an objective quality measure (*inverse design* [Sim94, vdP93, KPC93, WK88]).

The first image-driven method which I would like to outline has been presented by He et al. [HHKP96]. Their method uses genetic algorithms to create a good transfer function. The system randomly generates a set of transfer functions and renders small images for each one. After having been introduced to this set of renderings, the user then picks the few renderings that seem to best display the interesting structures of the volume data, and a new population of transfer functions is stochastically generated based on those that the user picked (like in [KG99]). The process iterates until the user feels that the best transfer function has been found. Alternatively, an image-processing metric like entropy, variance, and energy is used as an objective fitness function to evaluate the rendered images without human guidance. The process eventually results in a transfer function which maximizes the fitness function. The method succeeds in generating good renderings and frees the user from having to edit the transfer function manually. The second method proposed by [MAB+97] addresses the problem of parameter tweaking in general, with applications of light placement for rendering, motion control for articulated figure animation as well as transfer functions in direct volume rendering. The basic idea of this approach is to generate a very large set of transfer functions automatically and randomly and to organize small thumbnail renderings, which result from these transfer functions, into a *design gallery*. Design galleries arrange the results and organize them efficiently so that intuitive browsing can be made by the user, who peruses these thumbnail images, selecting the most appealing rendering.

Both methods seem to create reasonable transfer functions, but at the cost of completely preventing the user from a manual experimentation with the transfer function. As the process which generates the transfer function is a random process, the user has to choose between results which were presented by the system. The quality of the final transfer function is always uncertain. It is significantly that the generated transfer functions are not constrained at all by any measured properties of the data. Instead of interacting with the parameter space of transfer functions directly, the user explores it indirectly. This means that he only sees the results of the parameter setting for the rendering. Furthermore, the introduced methods may only generate good transfer functions for renderings from one particular point of view. If the point of view changes, the algorithms will iterate in a different way and may generate completely different results. It can be said that these methods are not appropriate for finding good transfer functions but rather for finding good rendered images.

In contrast to image-driven methods, data-driven techniques analyze the volume data instead of the generated images. The process of transfer function design is thus decoupled from the influence

of image related parameters such as viewing position and pixel resolution. Fang et al. [FBT98] face the problem of transfer function design from an image processing point of view. They use a transfer function to transform a three dimensional (3D) scalar volume to a 3D red, green, blue, and alpha (RGBA) volume and apply 3D image processing operations directly to the volume data. A disadvantage of this method is the high cost in memory that is required to store intermediate results for the image processing operations. A related technique was presented by Sato et al. [SWB+00], who applied 3D image filters in order to accentuate local intensity structures. The transfer function domain is a multi-dimensional feature space in their concept. In order to overcome the limitations inherent in conventional 1D opacity functions, they propose a classification method for tissues that employs a multi-dimensional opacity function, which is a function of the 3D derivative features calculated from a scalar volume, as well as the volume intensity. Tissues of interest are characterized by explicitly defined classification rules based on 3D filters which highlight local structures. The 3D local structure filters are formulated using the gradient vector and Hessian matrix of the volume intensity function combined with isotropic Gaussian blurring. Three years earlier, Bajaj et al. [BPS97] described a tool for assisting the user in selecting isovalues for effective isosurface volume visualizations of unstructured triangular meshes for isosurface rendering. By exploiting the mathematical properties of the mesh, important measures of an isosurface such as surface area and mean gradient magnitude can be computed with great efficiency. The results of these measurements are integrated into the same interface, which is used to set the isovalue.

The principles of multi-dimensional transfer functions have been investigated before by various researchers. Levoy [Lev88b] introduced two styles of transfer functions. Both are multi-dimensional and both are using gradient magnitude for the second dimension. One transfer function is intended for the display of interfaces between materials, the other for the display of isovalue contours between more smoothly varying data.

Summing it up, it can be pointed out that image-driven techniques are based on a trial-and-error generation of images to navigate the space of transfer functions. With this approach, the user can examine the results to find the best rendering. Data-driven techniques are based on the analysis of the volume data and provide methods to visualize shapes and structures in an unknown volume dataset. In most practical cases the user knows exactly what structures are contained in the dataset. He wants to visualize these structures of interest as fast as possible, without detailed knowledge of the rendering algorithm or the transfer function. For this purpose, in this diploma thesis an additional level of abstraction will be introduced which completely hides the transfer function from the user by providing a limited set of semantic parameters.

Chapter 3

Background

The following chapter provides an overview of the components that are included in the approach of high-level user interfaces based on semantic models. Volume data are described and it is said how they can be used for image rendering. In addition to this, this chapter gives an introduction to the field of multi-dimensional transfer functions, which includes 2D histograms based on data values, gradient magnitude and the second directional derivative. Section 3.3 outlines the technique of *Principal Component Analysis* briefly, which is used to approximate the range of input data to a lower-dimensional subspace. This chapter is concluded by the idea of this thesis, which is about facilitating visual parameter assignment based on concepts borrowed from computer animation.

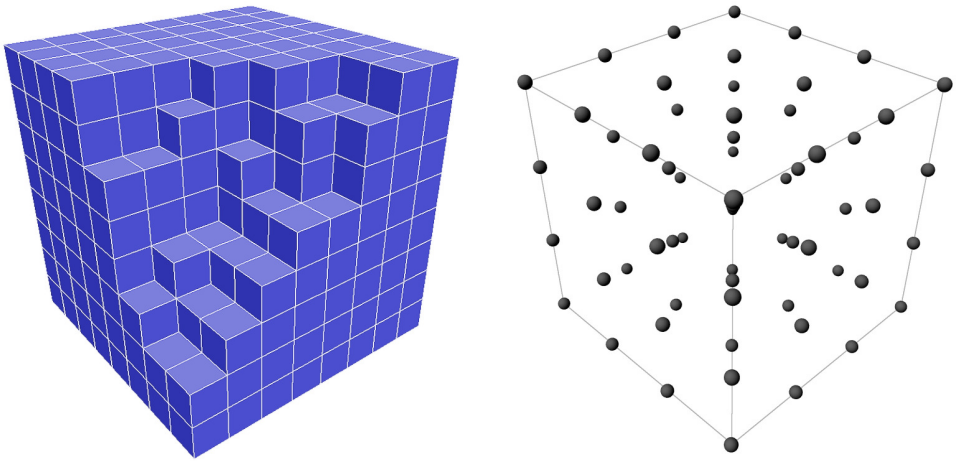
3.1 Volume Data

A discrete volume dataset simply is a three-dimensional array of cubic elements called voxels. A voxel (which is a coined word based on the words volumetric and pixel) is a volume element which represents a value in 3D space. It is analogous to a pixel, which represents 2D image data. Figure 3.1 shows two interpretations of a voxel value which depend on the usage: a unit of space in the volume as well as a sample point in a grid [Kau94].

Volume datasets can be divided into measured and simulated data and they differ in the structure of the underlying grid. Measured datasets, as they are created by the following techniques, which are mainly used in medical imaging, are usually acquired on a uniform *rectilinear* grid:

- **Computed tomography (CT).** CT, originally known as computed axial tomography (CAT or CAT scan) or body section roentgenography, is a medical imaging method employing tomography where digital geometry processing is used to generate a three-dimensional image of the internals of an object from a large series of two-dimensional X-ray images taken around a single axis of rotation.

- **Magnet resonance imaging (MRI).** An MRI uses powerful magnets to excite hydrogen nuclei in water molecules in human tissue, producing a detectable signal. Traditionally, an MRI creates



(a) (b)

Figure 3.1: Figure 3.1(a) shows cubes, with each cube representing a unit of space with particular properties in a larger volume. In Figure 3.1(b) voxels are interpreted as point samples in a 3D grid.

a large series of 2D images of a thin slice of the scanned object, just like a CT scan. The difference between a CT image and an MRI image is a question of details. In CT, X-rays must be blocked by some form of dense tissue in order to create an image, thus the image quality will be poor when you look at soft tissues. An MRI can only manage hydrogen based objects. This means that bone, which is calcium based, will be a void in the image and will not affect soft tissue views.

- **Ultrasound (US).** Medical ultrasonography uses high frequency sound waves of between 2.0 to 10.0 megahertz that are reflected by tissue in varying degrees producing 2D images. Doppler capabilities of modern scanners allow the blood flow in arteries and veins to be assessed.

In contrast to this, grid design algorithms for finite element simulation usually produce unstructured grids based on tetrahedrons and prisms. The value of a voxel may represent various properties. In CT scans, the values are Hounsfield units ([Hou73]), giving the opacity of material to X-rays. Different types of values are acquired from MRI or ultrasound.

The rendering of data represented as 3D scalar fields is known as volume rendering. In many ways, it has an even greater need for hardware acceleration than polygon rendering does, since volume datasets are generally much larger than polygon datasets which are used for surface visualization [FvDFH96]. Furthermore, voxel calculations are simpler than polygon calculations. Compared to surface data, which determine only the outer shell of an object, volume data describe the internal structures of solid objects. In addition to medical and scientific data representation, volume data allow the modeling of fluid and gaseous objects as well as of natural phenomena such as clouds, fog, fire or water [RS02]. The first architecture, which was designed in order to accelerate the display of

volume datasets, classifies voxels as either *occupied* or *empty* [Mea85]. This approach minimizes the amount of processing but also obscures data in the interior of the object. Nowadays, implemented rendering algorithms in most cases fully use the programmable graphics pipeline for displaying and editing the volume data in real-time[1].

Physically, a scalar volume can be interpreted as a continuous three-dimensional signal:

$$f(\vec{x}) \in \mathbb{R} \quad \text{with} \quad \vec{x} \in \mathbb{R}^3. \tag{3.1}$$

In signal processing, the Nyquist Rate f_N is the minimum sampling rate (in samples per second) which is required to avoid aliasing when sampling a continuous signal. If the input signal is real and band-limited, the Nyquist rate is twice the highest frequency contained within the signal, which reads

$$f_N = 2f_H, \tag{3.2}$$

with f_H being the highest frequency component contained in the signal. In order to avoid aliasing effects, the sampling rate f_S must exceed the Nyquist rate: $f_S > f_N$. According to the sampling theory, an ideal reconstruction of the signal requires the convolution of the sample points with a $sinc$ function (Figure 3.2)

$$sinc(x) = \frac{\sin(\pi x)}{\pi x}. \tag{3.3}$$

Due to the fact that this function has infinite extend and thus all sampling points of the original signal (and not only those in the close vicinity) should be considered, it follows that in practice it is intractable to apply $sinc$ for an exact reconstruction. Furthermore, it must be taken into account that a signal's shape is determined by its frequency spectrum. The sharper and more angular a waveform is, the richer it is in high-frequency components. Signals with discontinuities have an infinite frequency spectrum. Any sharp boundary between different materials ends in an infinite extend in the signal's frequency spectrum, which will produce aliasing artifacts if the signal is reconstructed from discrete samples. In practice, the ideal 3D $sinc$ filter is replaced by either a box filter or a tent filter in order to reconstruct a continuous signal from the 3D array of voxels.

Sampling theory again becomes important in the topic of transfer functions. The order in which signal reconstruction and transfer function are executed in the volume rendering pipeline can make a significant difference in the quality and speed of renderings. If a transfer function is applied *before* the reconstruction of the signal, the optical properties themselves are interpolated from the grid samples. The image is rather disturbing, not only because the frequency spectrum is changed, which might invalidate an initial assumption of band limitation and will thus strongly violate the sampling theorem, but also because of the blocky appearance. This effect is caused by important features, which may be missed if the data value changes rapidly from one grid sample point to the next. If the continuous signal is reconstructed first and then the transfer function is applied, it is ensured that the sampling

[1] Recent volume rendering visualization techniques are mentioned at the beginning of chapter 2.

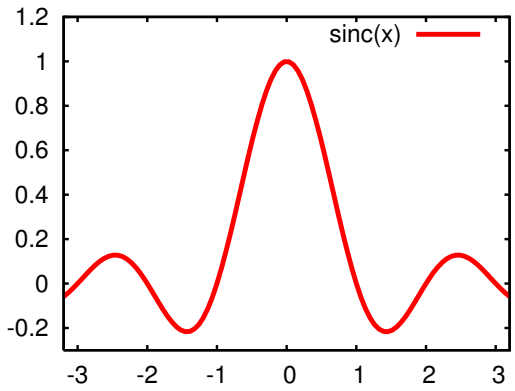


Figure 3.2: The ideal reconstruction filter for one dimensional signals is the $sinc$ filter.

theorem is obeyed, although the transfer function will introduce another high-frequency component to the signal. As long as the transfer function itself is band-limited, the signal is protected from any aliasing. The volumetric shapes result in a much better representation by the high-frequency of the transfer function applied as a so-called post-classification[2].

3.2 Multi-Dimensional Transfer Functions

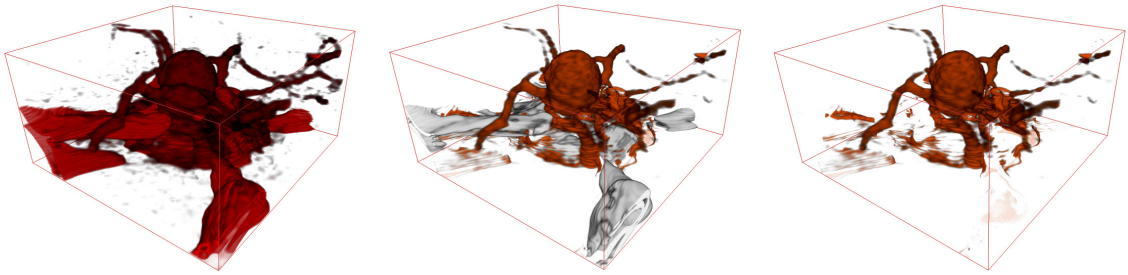
The term "*classification*" refers to the assignments of optical properties to data values. The process of classification is one of the most important steps in the volume rendering pipeline, as it creates the optical properties that will emphasize an important feature or obscure unimportant ones. Optical properties such as opacity and color are assigned to data values with the help of a transfer function. The quality of the resulting visualization will be largely dependent on how well these optical properties capture features of interest. In general, the design of a transfer function is a manual and time-consuming procedure, which requires detailed knowledge of spatial structures that are represented by the dataset. There are three difficulties regarding the process of specifying good transfer functions [Kni02]:

- It is difficult to identify features of interest in the transfer function domain. A feature of interest may be easily identifiable in the spatial domain, while, on the other hand, the range of the concerning data values may be difficult to isolate in the transfer function domain, although other, uninteresting regions may contain the same range of data values.
- The transfer function may have an enormous number of degrees of freedom, which may be pretty difficult to handle by the user.

[2]For detailed information about different reconstruction filters for volume data and pre- and post-classification of transfer functions see [Nov93, RS02].

- Typical user interfaces do not guide the user with regard to setting all the control points of the transfer function based on dataset specific information. Consequently, the user must rely on a trial and error process in finding good images.

Traditionally, the easiest way to define a transfer function is to determine color and opacity values for a voxel as a function of its scalar value. In practice, a transfer function is realized as a lookup table of fixed size. Scalar values are assigned to the RGB color entries of the lookup table in order to consider the emission of colored light. The opacity value for each voxel is determined as a scalar value between 0 and 1 which represents the absorption coefficients.



(a) (b) (c)

Figure 3.3: Structure rendered with 1D and 2D transfer functions. The adding of the gradient magnitude as a second axis to the transfer function disambiguates the boundaries (Figures 3.3(b) and 3.3(c)), whereas Figure 3.3(a) is generated with a 1D transfer function which offers no possibility to separate the blood vessels from the bone.

Rather than classifying a sample based on a single scalar value, multi-dimensional transfer functions allow a sample to be classified based on a combination of values. These values are the axes of a multi-dimensional transfer function. Thus, the probability with regard to isolating and differentiating between structures in the dataset increases. Figure 3.3 illustrates the differences of a 1D and a multi-dimensional (2D) transfer function.

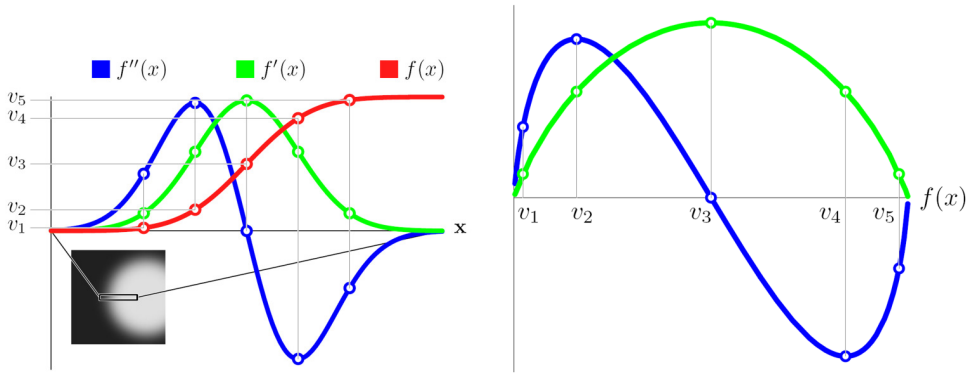
The first and second order derivatives can be taken into account in order to define a multi-dimensional parameter domain in addition to the scalar value. The gradient vector is the first order derivative for a scalar function $f(x, y, z)$ representing the 3D data, and is defined by using the partial derivatives of f in x-, y- and z-direction as

$$\nabla f = (f_x, f_y, f_z) = \left(\frac{\delta}{\delta x} f, \frac{\delta}{\delta y} f, \frac{\delta}{\delta z} f \right). \tag{3.4}$$

As a vector, it points into the direction of the greatest change. The gradient magnitude is a scalar quantity which describes the local rate of change in the scalar field. It is computed as the absolute value of the vector

$$f' = \|\nabla f\| = \sqrt{{f_x}^2 + {f_y}^2 + {f_z}^2}. \tag{3.5}$$

For notational convenience, f' is used to indicate the magnitude of the gradient of f. This value is useful as an axis of the transfer function, because it distinguishes between homogeneous regions, which have a low gradient magnitude, and regions of change, which have a high gradient magnitude. Please consider the relationships between the different axis in Figure 3.4(a) in order to get a better idea of multi-dimensional transfer functions. The image analyzes one segment of a slice from a synthetic cylinder dataset. Please note that at the mid-point of the boundary between the two materials (*background and cylinder*) the first derivative is at a maximum and the second derivative has a zero crossing. Because of blurring, the boundary is spread over a range of positions.



(a) (b)

Figure 3.4: As a function of position, Figure 3.4(a) shows the relationship between data values (f), gradient magnitude (f') and the second directional derivative (f''). In Figure 3.4(b), the derivatives are displayed as a function of data value. Figure: courtesy of Gordon Kindlmann [KD98].

The patterns of which the datasets consist can be visualized with histograms. In general, a histogram is a structure for representing a discrete approximation of a probability distribution function. Histograms are useful due to the fact that they can provide a compact summary of a large amount of data. Statistical quantities can be measured easier from a histogram than directly from the data. It is a good approach to examine the relationship between f, f' and f''. A common way to visualize a histogram is to produce two dimensional scatterplots of data value versus first derivative, or data value versus second derivative, as seen in Figure 3.5. In these scatterplot images, the data value and their derivatives are aligned to the axes as in Figure 3.4(b) in order to facilitate comparison. The darkness in the scatterplots encodes the number of hits for a given gradient/value pair. The results of the scatterplots closely conform to the curves of the continuous linear sampling example in Figure 3.4(b).

Since materials are relatively homogeneous, their gradient magnitudes are low. Boundaries between the materials are shown as arches. Figure 3.6(a) shows a histogram of data value versus gradient magnitude of a CTA dataset. The materials can be seen as circular regions added to the histogram. The labeled regions of blood vessels, bone and skin were isolated by using a 2D transfer function and can be found in the rendered image of Figure 3.6(b).

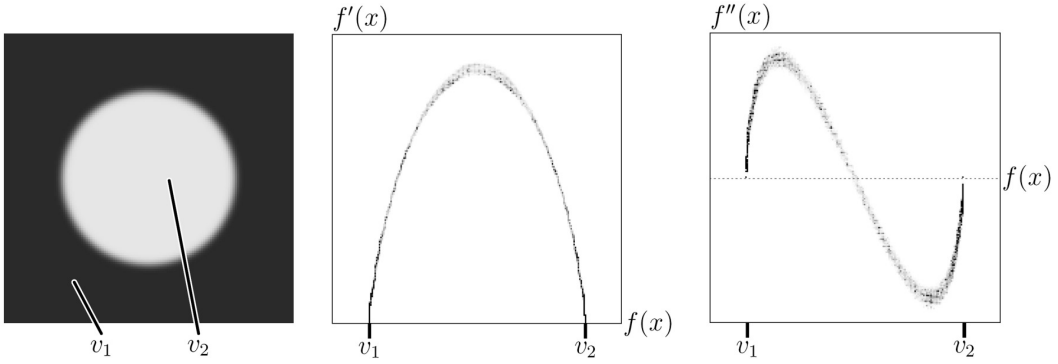


Figure 3.5: Cross-section and histogram scatterplots for a synthetic cylinder dataset. In Figure 3.5(a) a cross-section of the dataset is shown with only one visible material boundary. Figures 3.5(b) and 3.5(c) display a projection of the histogram volume, showing the relationship between f and f' and between f and f' with the indicated values v_1 and v_2 from Figure 3.5(a) on the $f(x)$ axes. Figure: courtesy of Gordon Kindlmann [KD98].

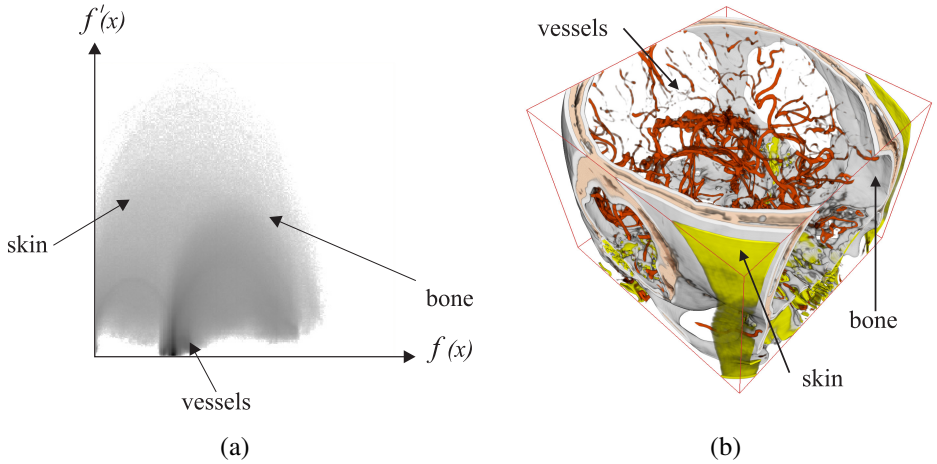


Figure 3.6: A volume rendering based on a 2D transfer function of data value and gradient magnitude showing bone, skin and blood vessels. In Figure 3.6(a), some of the structures which are part of the dataset are labeled in the histogram. Figure 3.6(b) displays the rendered image.

The arches that define material boundaries in a 2D transfer function often overlap. This prevents sometimes a homogeneous region from being properly isolated by the 2D transfer function of data value and gradient magnitude (f'). This effect can be avoided by using transfer functions based on data value and the second derivative (f''). More details on these measurements can be found in the semi-automatic transfer function design method by Kindlmann and Durkin [KD98, Kin99]. Their approach is capable of determining material boundaries within a given dataset by evaluating statistical information about the first and the second order directional derivatives with the help of histograms.

3.3 Principal Component Analysis

One of the statistical techniques frequently used in various fields of applications is called *Principal Component Analysis* (PCA). It is a powerful method for data analysis and pattern recognition and it can also be adapted to reduce a complex dataset to a lower dimension in order to reveal the sometimes hidden, simplified structure. In the following, the PCA theory [PFTV92] is briefly described to get familiar with the technique, which is used in chapter 5 of this thesis to approximate the range of input data in a lower dimensional subspace.

The input for the PCA is a set of m parameter vectors $\mathbf{p}_i \in \mathbb{R}^n$, with each vector consisting of n parameter values. These vectors can be arranged into a matrix $\mathbf{P} = [\mathbf{p}_0,\ \mathbf{p}_1,\ \ldots,\ \mathbf{p}_{m-1}] \in \mathbb{R}^{m \times n}$. The vector $\bar{\mathbf{p}}$ is the vector of mean values of all input variables defined by:

$$\bar{\mathbf{p}} = \frac{1}{m} \sum_{i=0}^{m-1} \mathbf{p_i}. \tag{3.6}$$

In order to calculate the mean-deviated representation $\mathbf{D}$ of the matrix $\mathbf{P}$, the mean vector $\bar{\mathbf{p}}$ has to be subtracted from all input vectors

$$\mathbf{d}_i = \mathbf{p}_i - \bar{\mathbf{p}}. \tag{3.7}$$

This process is called *centering* and results in a new matrix

$$\mathbf{D} = [\mathbf{d}_0,\ \mathbf{d}_1,\ \mathbf{d}_2,\ \ldots,\ \mathbf{d}_{m-1}] \in \mathbb{R}^{m \times n}\ . \tag{3.8}$$

PCA can be described as a linear transformation $\mathbf{A} \in \mathbb{R}^{n \times n}$ of $\mathbf{D}$ into a matrix $\mathbf{Y}$ according to:

$$\mathbf{AD} = \mathbf{Y}. \tag{3.9}$$

After the transformation, the row vectors of $\mathbf{Y}$ contain the components of the maximum variance in descending sequence. Figure 3.7(b) shows the 2D scatterplot of the mean-deviated input vectors. The mean vector $\bar{\mathbf{p}}$ is the new center of the scatterplot. With the help of the mean-deviated matrix $\mathbf{D}$, the symmetric and positive semidefinite covariance matrix $\mathbf{C_P} \in \mathbb{R}^{n \times n}$ can be calculated by:

$$\begin{aligned} \mathbf{C_P} &= \frac{1}{m-1} \sum_{i=0}^{m-1} (\mathbf{p_i} - \bar{\mathbf{p}})(\mathbf{p_i} - \bar{\mathbf{p}})^{\mathbf{T}} \\ &= \frac{1}{m-1} \mathbf{D}\mathbf{D}^{\mathbf{T}}. \end{aligned} \tag{3.10}$$

The elements on its main diagonal are the variances of the vector components. All other entries describe the covariance between two different components. In order to achieve the transformation presented in equation 3.9, matrix $\mathbf{A}$ has to be determined by calculating the eigenvalues λ and the eigenvectors $\mathbf{e}$ of the covariance matrix $\mathbf{C_P}$. The row vectors of matrix $\mathbf{A}$ are the eigenvectors ordered by their eigenvalues in descending sequence.

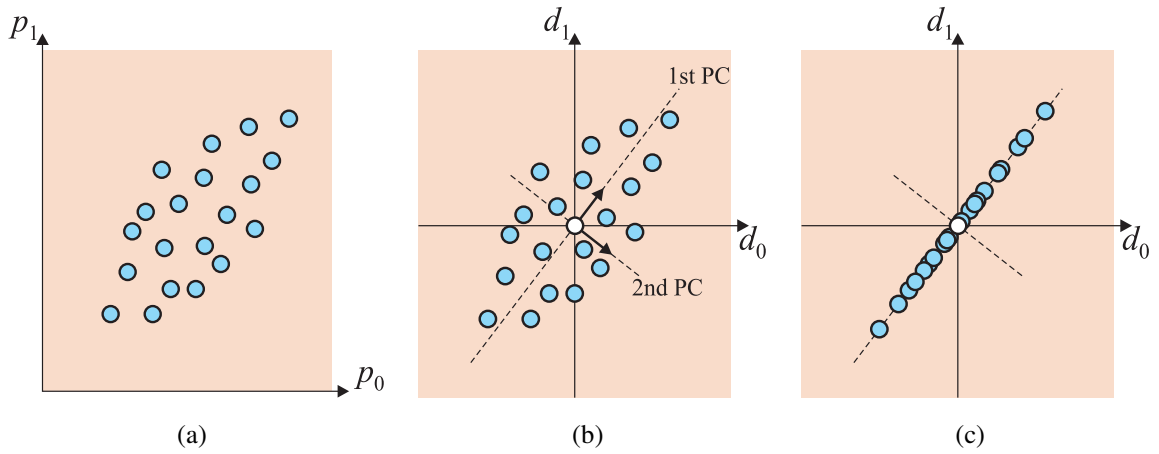


Figure 3.7: Figure 3.7(a) shows a scatterplot of the input vectors as point samples in a simple 2D example. In Figure 3.7(b) the input vectors are mean-deviated and PCA is used to determine the axis of the maximum variance. Figure 3.7(c): Reducing the input vectors to the first principal component approximates the data in a lower-dimensional subspace.

The eigenvector corresponding to the largest eigenvalue is called the first principal component and determines the axis of maximum variance of the joint probability distribution. Smaller variances in a lower-dimensional subspace are marked by the following principal components which are orthogonal to all other principal components. The values of $\mathbf{Y}$ are calculated with the help of the principal components. The mean-deviated original data can be restored with the equations

$$\mathbf{D} = \mathbf{A}^{-1}\mathbf{Y}, \tag{3.11}$$

$$\mathbf{D} = \mathbf{A}^{\mathbf{T}}\mathbf{Y}, \tag{3.12}$$

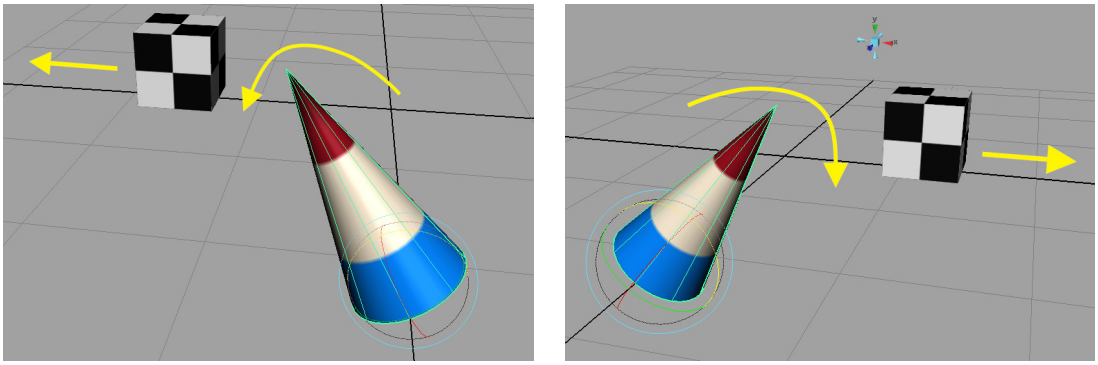
$$\mathbf{P} = (\mathbf{A}^{\mathbf{T}}\mathbf{Y}) + \bar{\mathbf{p}}. \tag{3.13}$$

The inverse of $\mathbf{A}$ in equation 3.11 is equal to its transpose in 3.12 because its elements are the unit-length eigenvectors of the dataset. As $\mathbf{D}$ represents the mean-deviated data, the mean vector $\bar{\mathbf{p}}$ has to be added to obtain the original dataset $\mathbf{P}$ once again (see equation 3.13). A more general solution called "*Singular Value Decomposition*" (SVD) could also be used in order to reduce the dimensionality of the data [Shl05]. The methods PCA and SVD are closely related.

3.4 Computer Animation

The inspiration of the user interface design presented in this thesis is based on concepts from computer animation. Direct connections of attributes between the two objects can be established in order to manipulate an object with the help of another object. For instance, connecting a cone's rotation

attribute to a cube's position attribute results in a moving cube while the user is rotating the cone. Direct connections are the strictest way to connect attributes of objects. The example shows that there is no way for a direct connection to be *weighted*. This means that the cube would move out of the image if the cone's rotation would continue. In the modeling and animation package Alias Maya™, for example, this is possible by creating so-called "*driven keys*". Other animation tools may use different names for a similar concept. Driven keys are neither direct and rather rigid connections of attributes nor conventional key frames, where an object is told to be in position 1 at one time and in position 2 at another time. Driven keys are particularly key frames that are specified with respect to an abstract parameter axis and not with respect to the time axis. They are an excellent tool for semi-automation. In the example mentioned above, the cone should serve as a lever which has a predefined motion sequence. Turning the lever counterclockwise moves the cube to the left until it reaches its final position. The same applies to turning it clockwise, which moves the cube to the right end of the scene. With the help of driven keys, the desired behavior is achieved without a single conventional key frame (see Figure 3.8). In an animation, standard key frames can be used for the lever's rotation. The position of the cube will update automatically.

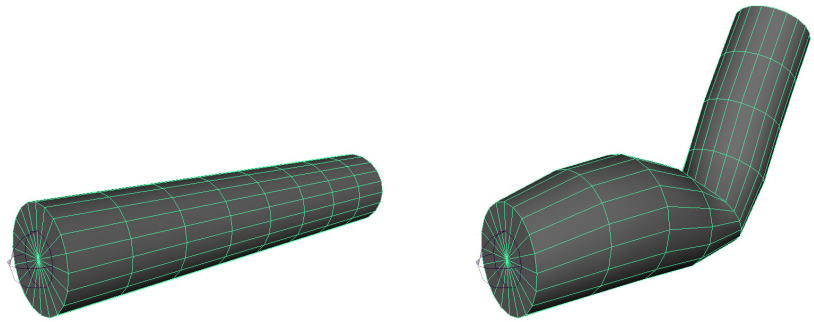


(a) (b)

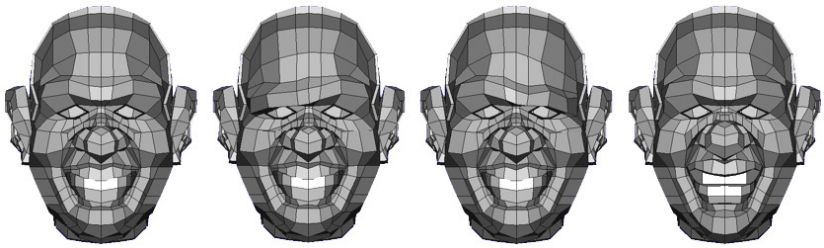
Figure 3.8: The position of the cube's movement is predefined and based on a weighted connection via driven keys. Turning the cone to the left in Figure 3.8(a) will move the cube to the left until it reaches its final position. Figure 3.8(b) shows the movement of the cube in the opposite direction by turning the cone to the right.

A more complex example dealing with character animation points out the advantages of driven keys with respect to the user interface design of this thesis. In a typical production, a technical director is responsible for creating the articulated model of a single character, which can afterwards be controlled intuitively by an animator. According to the underlying story board, each individual character has a set of expressions that he must be able to perform. For example, the character's face could be able to smile and frown. The animator uses high-level parameters in order to control the facial expressions directly. From the technical director's point of view, however, each facial expression consists of a combination of multiple low-level parameters such as the activation of specific muscles,

which are nothing but objects with an abundance of attributes. Generating a smiling face, for example, will involve the movement of the lips, the cheeks, the eyelids and the eyebrows, which are controlled by different predefined blend-shapes [RCB05]. Additionally, the jaw, which is controlled by the kinematic skeleton, will open slightly and wrinkles will become visible on the forehead, which may be displayed by textures and bump maps. Examples of blend shapes and corresponding expressions are shown in Figure 3.9. In order to provide intuitive control of the innumerable attributes for each facial expression, the technical director compiles combinations of the low-level parameters into high-level parameters used by the animator with the help of driven keys. This way, the technical director creates semantic parameters such as "*smile*" and "*frown*" and hides the complex setup of low-level parameters from the animator.



(a)



(b)

Figure 3.9: The simple example in Figure 3.9(a) demonstrates how blend shapes and driven keys work in a basic muscle flex. Rotating the elbow in z-direction results in a mighty and strong arm. In character animation (Figure 3.9(b)), facial expressions are implemented by using predefined blend shapes in combination with kinematics and other deformation techniques. The technical director hides much of the complexity of low-level parameter assignment from the animator by introducing an additional level of abstraction.

Chapter 4

Semantic Transfer Function Models

This chapter summarizes the drawbacks of existing transfer function design models and introduces a new model based on semantics. The benefit of this technique is exposed later on and a formal model is developed.

4.1 Transfer Function Parameters

The automatic and the semi-automatic design of transfer functions in the field of data-driven and image-driven techniques (see chapter 2) are still topics of active research. However, the most frequently used way of transfer function adjustment is the manual visual editing of the transfer function [RS02]. As already described in the previous chapter, transfer functions are usually stored and applied as color tables. Current implementations of color table editors vary in the representation and in the degrees of freedom for the transfer function. A simple user interface of transfer function adjustment is shown in Figure 4.1. Each of the RGBA channels can be adjusted separately referring to a histogram which is displayed in the background of the editor. Finally, the arranged intensities are mapped to an 1D array of RGBA quadruples, which the lookup table is consisting of.

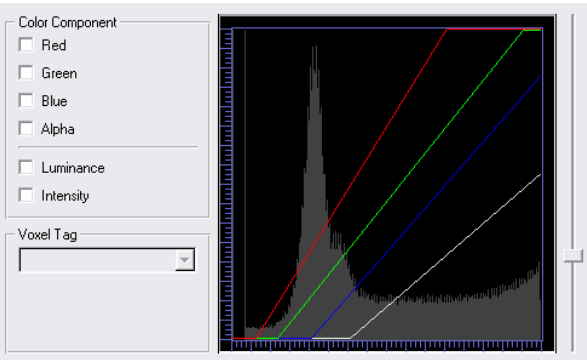
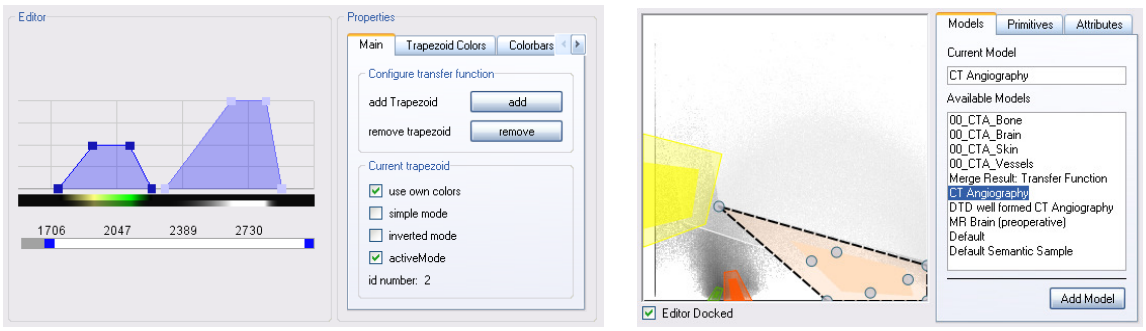


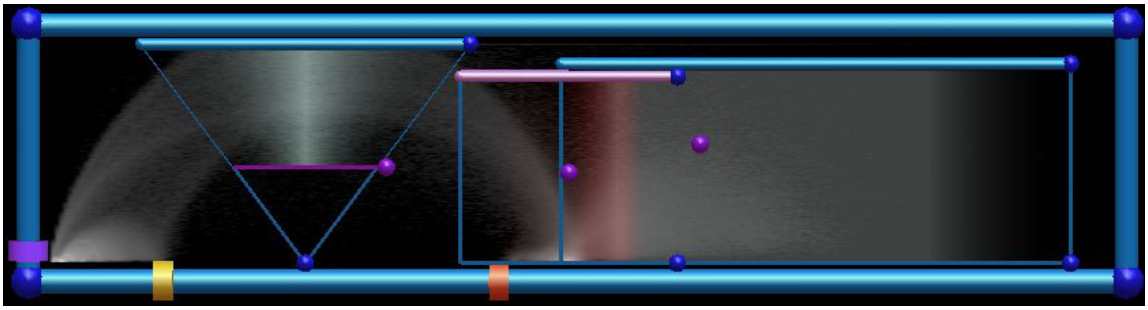
Figure 4.1: Editor for the adjustment of a 1D transfer function. A separate editing of each of the RGBA channels provides high degrees of freedom.

Finding appropriate transfer functions that lead to a desired image is a tedious and time consuming process of the manual tweaking of parameters. The higher the degrees of freedom of the transfer function adjustment, the more it is difficult to accomplish good renderings. While adding dimensions to the transfer function enhances the ability to isolate features of interest in a dataset (see section 3.2), it tends to make the space of the transfer function, which already is not intuitive, even more difficult to navigate. It would be helpful for researchers to use simplified editors which could be handled intuitively and fast.

The modern user interfaces for transfer function specification often provide an additional layer of abstraction by introducing simple shapes as primitive objects such as boxes, ramps or trapezoids. Examples of primitives in case of 2D transfer functions are paraboloid shapes introduced by Vega et al. [HST+04] and trapezoids suggested by Kniss et al. [KKH01]. Different types of user interfaces for 1D and 2D transfer functions are shown in Figure 4.2.



(a) (b)



(c)

Figure 4.2: User interfaces for transfer function assignment. Figure 4.2(a) shows a 1D transfer function editor which supports ramps and trapezoids. Different types of primitive shapes are supported by the 2D editor of Figure 4.2(b). It shows a template used for CT data. The direct manipulation widgets presented by Kniss et al. are visible in Figure 4.2(c).

Regardless of its individual representation and its dimensionality, a transfer function can be considered simply as a collection of parameters. At the lowest level of abstraction, a transfer function may be implemented as a simple lookup table with each entry in this table referring to a separate parameter.

With respect to an additional abstraction layer which uses primitive shapes to adjust the transfer function, each primitive has a set of parameters such as position and control points, and color and opacity values. Modifying the primitive's shape results in a parameter change which directly influences the transfer function. Table 4.1 lists some examples of parameters of a quadrilateral primitive.

Number	**Name**	**Description**
1	POINT1_X	position of point 1 in x-direction
2	POINT1_Y	position of point 1 in y-direction
3	POINT2_X	position of point 2 in x-direction
4	POINT2_Y	position of point 2 in y-direction
5	POINT3_X	position of point 3 in x-direction
6	POINT3_Y	position of point 3 in y-direction
7	POINT4_X	position of point 4 in x-direction
8	POINT4_Y	position of point 5 in y-direction
9	COLOR_RED	value of color channel red
10	COLOR_GREEN	value of color channel green
11	COLOR_BLUE	value of color channel blue
12	OPACITY	the primitive's opacity
...	...	...

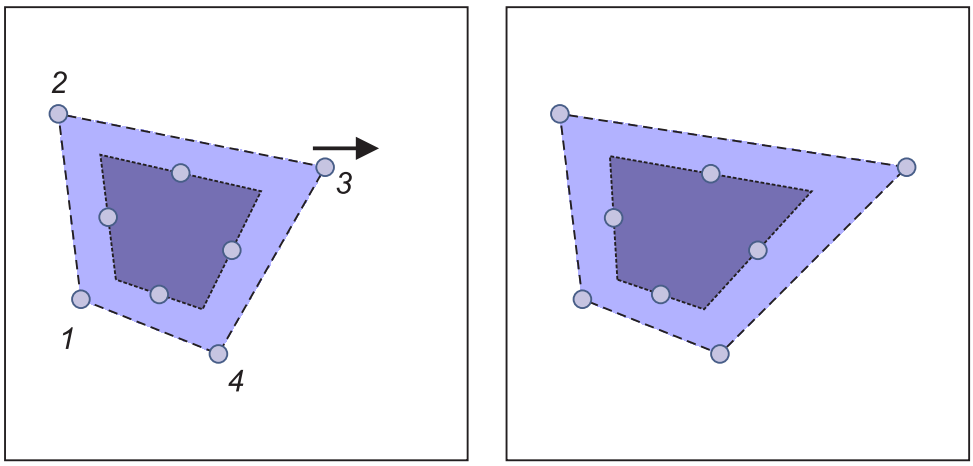
Table 4.1: A few parameters of a quadrilateral primitive, such as position, color and opacity parameters. Changing the parameters (i.e. modifying the shape) directly influences the transfer function.

Figure 4.3 illustrates a change of a quadrilateral parameter. Shifting point 3 will result in a change of the parameter numbers 5 and 6 according to table 4.1. In this example, the point is moved to the right. This movement responds to the x-direction and refers to parameter number 5. Some of the primitive's inner points are also effected by this point-shift, which is explained in detail in chapter 6.

A single transfer function can be defined by multiple and various primitives. Most implementations convert primitives into a color table representation before rendering. As an alternative, (if programmable graphics hardware is used) transfer functions can be specified procedurally and evaluated at run-time, like the multi-dimensional Gaussian primitives proposed by Kniss et al. [KPI$^+$03, EWRS$^+$06]. Independent of the kind of abstraction layer or number of primitives, the parameters which represent an individual transfer function can be specified as an array of n floating point values $\mathbf{p}$:

$$\mathbf{p} = (p_0, p_1, p_2, \ldots, p_{n-1}) \in \mathbb{R}^n. \tag{4.1}$$

For instance, if a transfer function would consist of a quadrilateral primitive, the array of equation 4.1 would be filled with values of parameters listed in table 4.1.



(a) (b)

Figure 4.3: A modification of the shape of a quadrilateral. Figure 4.3(a) shows the primitive before moving point 3 to the right. Figure 4.3(b) illustrates the new shape after the change of parameters.

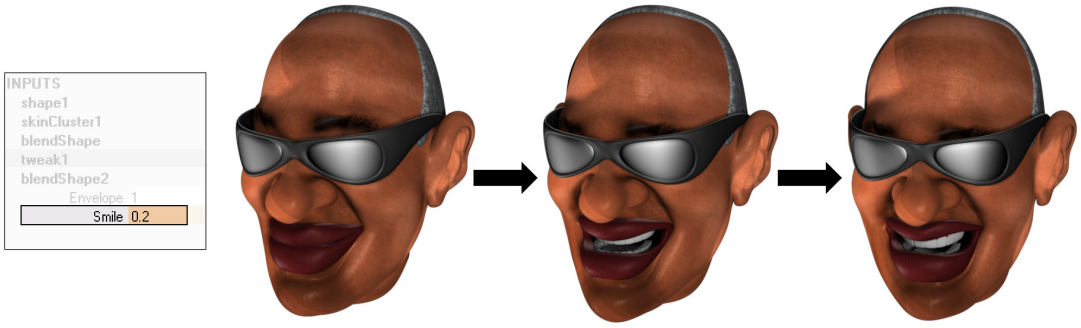
4.2 Semantic Parameters

Kniss et al. describe in [KKH05] a typical session of creating a good transfer function, which is a natural process of exploration, specification and refinement. Exploration is the procedure by which a user familiarizes himself or herself with the dataset. During the specification stage, the user creates a rough draft of the desired transfer function. In the system which Kniss et al. use, features of interest can be added to the transfer function independent of further investigation of the volume. In the last step, the user can refine the transfer function by manipulating control points of the primitives. Finally, this is an iterative process. The users continue the exploration, specification and refinement steps until they are satisfied with the fact that all features of interest are visualized. For a non-expert user this is a extremely difficult, time consuming task to obtain good transfer functions.

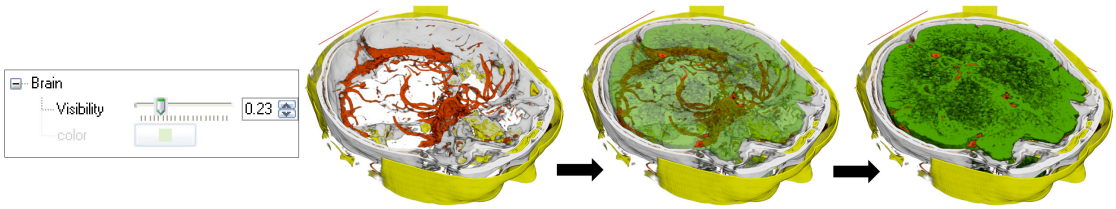
In section 3.4, a concept (driven keys) is described which allows the user to control an enormous number of attributes easily by introducing an additional layer of abstraction. In computer animation, for instance, the technical director creates semantic parameters such as *smile* or *frown*, and hides the complex setup of low-level parameters from the animator. Could the draft of computer animation described above be adapted to the task of defining a new transfer function model? The basic idea is that the visualization expert, who is familiar with all the parameters involved in image design, will play the role of the technical director transferred from the field of computer animation. If you introduce semantic parameters as an additional abstraction layer in order to control the transfer function, this may simplify the process of finding features of interest in the dataset for a non-expert client.

For example, a semantic parameter like “Brain Visibility” may be useful for a transfer function model to visualize the brain’s structures in a CTA dataset. Figure 4.4 illustrates the behavior of such a semantic parameter. A simple slider or spin-box can be useful to adjust the values. In correspondence

to low-level parameters such as the activation of specific muscles in facial animation, the low-level parameters in transfer function design are the values of the array $\mathbf{p}$, which represents an individual transfer function.



(a)



(b)

Figure 4.4: Semantic parameters as an additional abstraction layer. In Figure 4.4(a) the smile of the character is controlled by a semantic parameter. The higher the value, the more distinctive the smile. The pictures from the left to the right show an increasing value. Figure 4.4(b): The technique of an additional abstraction layer is adapted to the transfer function design. A slider controls the value of the brain's visibility in a CTA dataset. An increasing value leads to a more opaque brain structure (pictures from the left to the right).

The approach of this thesis deals with an effective technique that is developed to hide the complexity of parameter assignment from the non-expert client. As a basis for implementing semantics in the new transfer function model, a set of semantic parameters $\mathbf{s}$

$$\mathbf{s} = (s_0, s_1, s_2, \ldots, s_{t-1}) \in \mathbb{R}^t, \tag{4.2}$$

is introduced. The number of semantic parameters t will usually be significantly smaller than the number of low-level parameters n of which a transfer function is composed, although this is not a necessary condition.

4.3 Influence and Key Concept

Each of the semantic parameters $s \in \mathbb{R}$ introduced in the previous section has an influence on the vector of low-level parameters $\mathbf{p}$. The influence $\mathbf{q}(s)$ is defined as the function $\mathbf{q}(s) : \mathbb{R} \mapsto \mathbb{R}^n$ and the final n-dimensional low-level parameter vector $\mathbf{p}$ is computed by summing up the influences of all semantic parameters

$$\mathbf{p} \;=\; f(\mathbf{s}) \;=\; \sum_{i=0}^{t-1} \mathbf{q}(s_i). \tag{4.3}$$

A simple example of a quadrilateral primitive is given in Figure 4.5, which forms a transfer function that consists of three semantic parameters. In general, the function $f : \mathbb{R}^t \mapsto \mathbb{R}^n$ that maps semantic parameters to a transfer function can be an arbitrary function. The sum is chosen to keep the model intuitively understandable.

$\mathbf{q}(s_0)$	=	(	0.24	0.21	0.11	0.71	0.8	0.65	0.42	0.16	0	0	0	0	)
$\mathbf{q}(s_1)$	=	(	0	0	0	0	0	0	0	0	0.2	0.2	0	0	)
$\mathbf{q}(s_2)$	=	(	0	0	0	0	0	0	0	0	0	0	0	0.75	)
$\mathbf{p}$	=	(	0.24	0.21	0.11	0.71	0.8	0.65	0.42	0.16	0.2	0.2	0	0.75	)
			↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	
			POINT1_X	POINT1_Y	POINT2_X	POINT2_Y	POINT3_X	POINT3_Y	POINT4_X	POINT4_Y	COLOR_RED	COLOR_GREEN	COLOR_BLUE	OPACITY	

Figure 4.5: Parameters of a transfer function. The figure represents a transfer function which consists of a quadrilateral primitive. The order of the parameters is transferred from table 4.1. The parameter array $\mathbf{p}$ consists of the sum of all influences $\mathbf{q}(s)$. The influence vector $\mathbf{q}(s_0)$ is responsible for the primitive's shape, whereas $\mathbf{q}(s_1)$ manipulates the color and $\mathbf{q}(s_2)$ controls the primitive's opacity value.

Similar to the driven keys described in section 3.4, which are used to weight a connection between various attributes, the influence of a semantic parameter s_i is specified by a variable set of keys $(\sigma_j\ \hat{\mathbf{q}}_j)$ and by step-by-step linear interpolation

$$\mathbf{q}(s_i) = \begin{cases} (\frac{\sigma_1 - s_i}{\sigma_1 - \sigma_0})\ \hat{\mathbf{q}}_0 + (\frac{s_i - \sigma_0}{\sigma_1 - \sigma_0})\ \hat{\mathbf{q}}_1, & \text{for } \sigma_0 \le s_i < \sigma_1 \\ (\frac{\sigma_2 - s_i}{\sigma_2 - \sigma_1})\ \hat{\mathbf{q}}_1 + (\frac{s_i - \sigma_1}{\sigma_2 - \sigma_1})\ \hat{\mathbf{q}}_2, & \text{for } \sigma_1 \le s_i < \sigma_2 \\ \vdots & \vdots \\ (\frac{\sigma_k - s_i}{\sigma_k - \sigma_{k-1}})\ \hat{\mathbf{q}}_{k-1} + (\frac{s_i - \sigma_{k-1}}{\sigma_k - \sigma_{k-1}})\ \hat{\mathbf{q}}_k, & \text{for } \sigma_{k-1} \le s_i \le \sigma_k \end{cases}$$

with $\mathbf{q}(\sigma_j) = \hat{\mathbf{q}}_j$. Because of the interpolation between influences specified by keys, at least two keys are necessary to compute a valid influence $\mathbf{q}(s_i)$.

With some effort it might be possible for an experienced user to specify the keys for a given dataset manually, like the direct influence of the semantic parameter "*visibility*" on the opacity of a single transfer function primitive. In Figure 4.6, this simple relationship is demonstrated: the adjustment of the slider of Figure 4.4(b) to a value of 0.75 will result in the influence $\mathbf{q}(s_2)$ presented in Figure 4.5, for example. In more complex relationships, more sophisticated techniques must be used to find appropriate keys that yield the desired visual results.

$$
\begin{array}{lll}
s_2 = 0.75 & & \\
\sigma_0 = 0.0 & \hat{\mathbf{q}}_0 & = (\; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \;) \\
\sigma_1 = 1.0 & \hat{\mathbf{q}}_1 & = (\; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 1 \;) \\
\hline
 & \mathbf{q}(s_2) & = (\; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0.75 \;) \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \downarrow \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{OPACITY}
\end{array}
$$

Figure 4.6: An example of keys and influences. The semantic parameter *visibility* s_2 has the value 0.75. With the help of keys ($\sigma_0 = 0$ and $\sigma_1 = 1$) and their influence vectors ($\hat{\mathbf{q}}_0$ and $\hat{\mathbf{q}}_1$) the influence $\mathbf{q}(s_2)$ is computed. The result is based on linear interpolation.

The semantic models which this thesis is about, however, should not only function with regard to one specific dataset. Instead, a semantic model will be developed which is applicable to different datasets. It is required that the datasets have been recorded with a similar tomographic sequence and with the same examination purpose in mind. This applies to most examination procedures in medical routine. Some examples will be shown in chapter 7.

Chapter 5

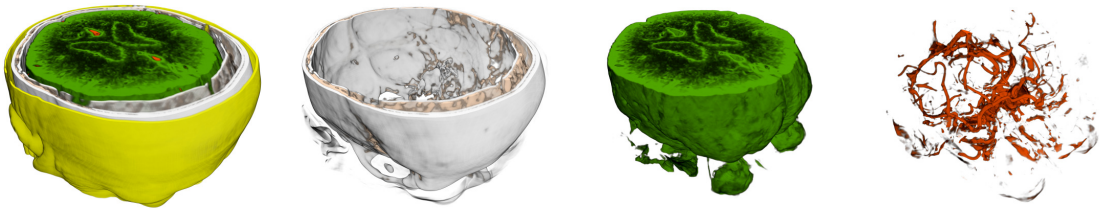
Design of Semantic Models

The previous chapter has shown that the concept of computer animation is capable of developing semantic transfer function models. A technique has still to be elaborated which is able to generate semantic models for more complex relations regarding a transfer function and different datasets. Now section 5.3 describes how this problem can be solved efficiently with PCA. The chapter closes with a section about creating additional semantics.

During this chapter, primitives objects are used to adapt structures of the datasets, but finally, chapter 6 contains a detailed description of the primitives' properties and features.

5.1 Principles

Before starting the implementation of the semantic model, the expert-user who will design the model should talk in detail about the relevant datasets' structures of interest the client-user wants to have visualized. Furthermore, the required semantic parameters have to be defined. It is a challenge to find an appropriate set of weights which yield the desired result when the semantic parameters are finally used to instantiate the transfer function according to equation 4.3.



(a) (b) (c) (d)

Figure 5.1: *Entities* of a CTA dataset. In Figure 5.1(a) a combination of all relevant structures of the dataset is visualized. Each of the other images contains a single structure, such as bone (Figure 5.1(b)), brain tissue (Figure 5.1(c)) and blood vessels (Figure 5.1(d)).

Semantic Parameters
Brain Sharpness
Brain Visibility
Bone Color
Skin Contrast
Skin Color
Skin Sharpness
Vessels Contrast

Entities
Brain
Bone
Skin
Vessels

Groups
Brain
Sharpness
Visibility
Bone
Color
Skin
Contrast
Color
Sharpness
Vessels
Contrast

Figure 5.2: An example of semantic parameters for a CTA dataset. The table on the left contains a list of potential semantic parameters. All structures in which the user is interested are listed in the table in the middle. The table on the right groups all semantic parameters under the entities they belong to.

It is assumed that you do not need a semantic model for each individual dataset, but instead of this only one model is needed for similar datasets with the same examination purpose in mind, such as CTA datasets, for instance. The basis of the approach of designing semantic models is a set of reference datasets for each examination purpose. Ideally, this set should statistically represent the range of possible datasets. In practice, however, this condition can hardly be verified and it is suggested to use as many datasets of a specific type as available.

The next step is to list all relevant structures contained in the data. Each of the structures is called an "*entity*". Please note that there is a difference between semantic parameters, such as *visibility* or *color*, and structures, such as *bone* and *skin*. To be more precise, semantic parameters are grouped under *entities*. Figure 5.1 displays a couple of entities of a CTA dataset which comprise skin, bone, brain tissue and blood vessels. In Figure 5.2, the relation to semantic parameters is illustrated in an example.

Each entity is represented by one or more primitives the transfer function consists of. Based on these primitives, a transfer function model is created which is used as a template for manual adaption and refinement. In the following step, the transfer function template has to be adjusted to all reference datasets. Each of the reference datasets is loaded one after another into the volume renderer. The data values in the datasets are slightly different and the user interface for primitives is used to adapt the transfer function model to each individual dataset. An *instance vector* for each reference dataset is obtained by this adaption. The set of instance vectors is analyzed in order to create semantic parameters by using PCA for finding similarities to approximate the vectors in a lower-dimensional subspace. The following sections describe the process of template creation and adaption in detail, as well as computing semantic influence vectors and creating additional semantic parameters.

5.2 Template Creation and Adaption

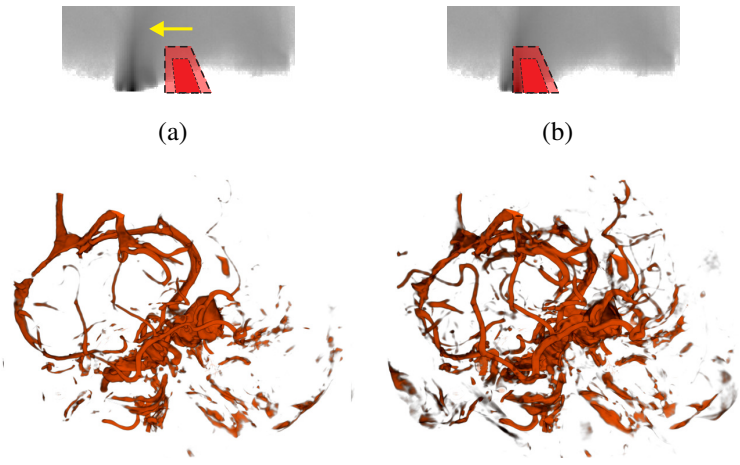
As already explained before, each entity is represented by one or more primitives with a set of primitive parameters

$$\mathbf{prim} = (prim_0,\ prim_1,\ prim_2,\ \ldots,\ prim_{g-1}). \tag{5.1}$$

One of the reference datasets is loaded into the system in order to create a transfer function template. As many primitives objects as needed are added to the transfer function and their parameters are concatenated. A quadrilateral primitive, for example, has $g = 12$ parameters (see section 4.5). A transfer function $\mathbf{p}$ which contains two of the primitives (for every entity a separate primitive) is then specified as an array of $n = g + g = 24$ elements and is defined as

$$\begin{aligned} \mathbf{p} &= (p_0,\ p_1,\ p_2,\ \ldots,\ p_{n-1}) \\ &= (\mathbf{prim}_{\text{entity1}},\ \mathbf{prim}_{\text{entity2}}) \\ &= (prim_{\text{entity1 }0},\ \ldots,\ prim_{\text{entity1 }11},\ prim_{\text{entity2 }0},\ \ldots,\ prim_{\text{entity2 }11}). \end{aligned} \tag{5.2}$$

As it has already been pointed out, modifying the shape of the primitives results in a parameter change and directly influences the transfer function.

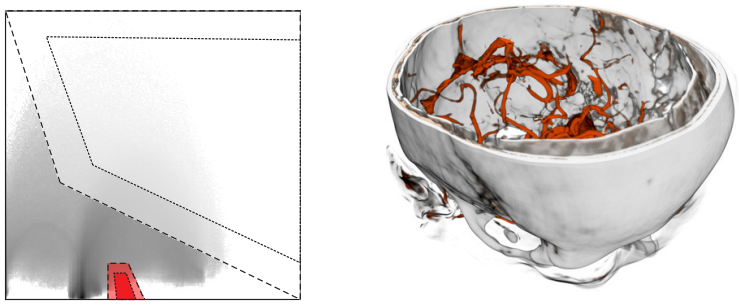


(a) (b) (c) (d)

Figure 5.3: The process of aligning a primitive to the structures of the dataset. The movement of the position of the quadrilateral primitive as shown in Figures 5.3(a) and 5.3(b) results in a better rendering of the blood vessels. In Figure 5.3(c), some parts of the vessels are missing. The position of the primitive in Figure 5.3(b) approximates the blood vessels in a better way, as shown in Figure 5.3(d). A part of the histogram scatterplot is visible in the background of the primitive editor.

All primitives are aligned to the structures of the dataset as good as possible in order to achieve the desired visual result. Figure 5.3 illustrates the procedure of finding a good transfer function which

approximates the blood vessels. The process of moving the appropriate primitive just a small distance into another direction visualizes the vessels in a better way. In Figure 5.4, two primitives ($\mathbf{prim}_{\text{bone}}$ and $\mathbf{prim}_{\text{vessels}}$) are arranged as good as possible to achieve a good transfer function. It is a tedious and time consuming process of manual tweaking of parameters to align the primitives to the structures of the dataset.



(a) (b)

Figure 5.4: A transfer function showing bone and vessels. The red quadrilateral primitive in Figure 5.4(a) represents the vessels entity. The large white primitive covers the data of the bone. Figure 5.4(b) displays the rendered image based on the transfer function.

In order to create a transfer function template which can be adapted to other datasets, all primitives have to be aligned to the datasets in the best possible way in order to visualize the structures described in the entities. The template is now available for the reference datasets.

In the next step, the template has to be adjusted to the reference datasets. The justification to an individual dataset requires a modification of the underlying primitives, as the structures in the datasets are slightly different for each dataset. The scatterplot of a histogram is helpful in order to identify the differences in the structures. Often only a small number of parameters is affected by the justification. Finally, after having made manual adaption and refinement, an instance of the parameter vector $\mathbf{p}_i$ of the transfer function for each reference dataset i is obtained. The scheme in Figure 5.5 resumes the process of template creation and adaption. These modifications, however, will most likely not affect all of the low-level parameters the transfer function consists of. In many cases, only a limited subset of parameters is involved in the modification.

5.3 Data Approximation

As already mentioned in the previous section, only a small number of parameters has to be adjusted to achieve good transfer functions. In addition to this, the modified parameters are usually highly correlated. PCA can be used to approximate the data of transfer functions in a lower-dimensional subspace in order to take advantage of the correlation between the different data.

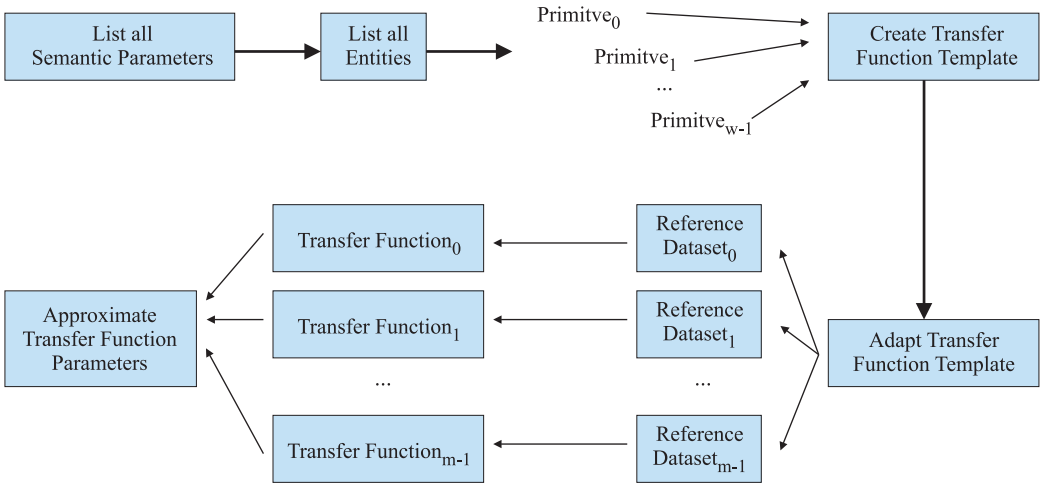


Figure 5.5: The process of template creation and adaption. The parameters of the instance vectors can be approximated with PCA, for example.

In section 3.3, the PCA theory is briefly described. Each instance of the parameter vectors $\mathbf{p}_i$ can be interpreted as a point sample in the n-dimensional low-level parameter domain of the transfer function. Please remember that a transfer function $\mathbf{p}$ is specified as an array of n values. The components of the instance vector $\mathbf{p}_i$ are interpreted as random variables with Gaussian probability distributions in order to apply PCA. If you assume that m reference datasets are available, then m instance vectors $\mathbf{p} = (p_0, p_1, p_2, \ldots, p_{n-1})$ are extracted from the template's adaption to the reference datasets, one instance vector $\mathbf{p}_i$ for each reference dataset i. According to section 3.3, the matrix of all input vectors can be written as

$$\begin{aligned} \mathbf{P} &= \begin{bmatrix} \mathbf{p}_0[0] & \cdots & \mathbf{p}_{m-1}[0] \\ \vdots & \ddots & \vdots \\ \mathbf{p}_0[n-1] & \cdots & \mathbf{p}_{m-1}[n-1] \end{bmatrix} \\ &= [\mathbf{p}_0,\ \mathbf{p}_1,\ \mathbf{p}_2,\ \ldots,\ \mathbf{p}_{m-1}]. \end{aligned} \tag{5.3}$$

The vector of mean values $\bar{\mathbf{p}}$ is approximated by averaging the instance vectors $\mathbf{p}_i$ analogously to equation 3.6. The symmetric and positive semidefinite covariance matrix can be computed like in equation 3.10

$$\mathbf{C_P} = \frac{1}{m-1}\mathbf{D}\mathbf{D}^{\mathbf{T}}, \tag{5.4}$$

with $\mathbf{D}$ being the mean-deviated representation of the matrix $\mathbf{P}$ of input vectors. As a result, the possible range of data is approximated in a lower-dimensional subspace, if the instance vectors are represented by only a few principal components. The quality of such an approximation is determined by the importance of the respective eigenvalues λ_j. The importance of a component axis can be

determined by dividing the eigenvalue λ_j by the sum of eigenvalues

$$I(\lambda_i) = \frac{\lambda_i}{\sum_{j=0}^{n-1} \lambda_j}. \tag{5.5}$$

Principal components with an importance of a value which is close to zero can be safely omitted without a significant loss in accuracy. The goal is to use PCA to determine a single parameter axis in the low-level parameter space based on the reference datasets. This axis will approximate the modifications of the transfer function template to the parameter vector $\mathbf{p}$. The importance of the first principal component is usually close to 1.0 throughout the experiments, which means that the approximation error can be neglected. Reducing the data to a single axis facilitates the transfer function to be controlled by only one semantic parameter. In the following, this semantic parameter is called "*adapt template*" since it adjusts the transfer function's primitives to the dataset. In practice, splitting the instance vectors into groups of primitives which, after being adapted to all reference datasets, belong to the same entity may result in transfer functions easier to cope with. PCA will then be performed separately for each group[1] (see section 7).

Each semantic parameter can be represented by a spin-box or a single slider in the user interface. A non-expert user is able to interact with the semantic parameters, and this process adapts the transfer function to its individual dataset without any knowledge about the underlying transfer function model. All instance vectors $\mathbf{p}_i$ are projected on the axis spanned by the first principal component $\mathbf{e}_0$ in order to determine the minimum and maximum values of the semantic parameter. This is done by computing the n-dimensional dot product

$$\mu_i = \mathbf{e}_0 \cdot (\mathbf{p}_i - \bar{\mathbf{p}}). \tag{5.6}$$

The influence vectors, which have been introduced in section 4.3 and which are specified by a variable set of keys $(\sigma_j\ \hat{\mathbf{q}}_j)$, then can be computed by

$$\begin{aligned} \sigma_0 &= 0.0\,, & \hat{\mathbf{q}}_0 &= \bar{\mathbf{p}} + \min(\mu_i) \cdot \mathbf{e}_0\ , \\ \sigma_1 &= 1.0\,, & \hat{\mathbf{q}}_1 &= \bar{\mathbf{p}} + \max(\mu_i) \cdot \mathbf{e}_0\ . \end{aligned}$$

The values for the keys σ_0 and σ_1 can be chosen arbitrarily, since they determine the range of the semantic parameter. Please remember that with the help of keys the influence $\mathbf{q}(s)$ of the semantic parameter s is computed on the basis of linear interpolation.

[1] Since the adapt template parameter only affects the parameters of the primitives which change the shape, all other parameter values such as opacity can be set to 0 while applying PCA to the instance vectors. Additional semantic parameters can control these parameters separately (see section 5.4).

5.4 Additional Semantics

The semantic parameter *adapt template* offers the possibility to visualize structures of an unknown dataset in a very fast and effective way. Therefore, the tweaking of just one parameter allows the user to find an appropriate transfer function for a specific task. A modification of the semantic parameter's value results in a change of the low-level parameters of the underlying transfer function as well as of the primitives' shape. Usually the primitives which are contained in the transfer function have much more parameters than those which control the shape (see table 4.1, numbers 1-8). Semantic parameters can be applied to all parameters. The following sections give an insight into the creation of additional semantics.

5.4.1 Visibility

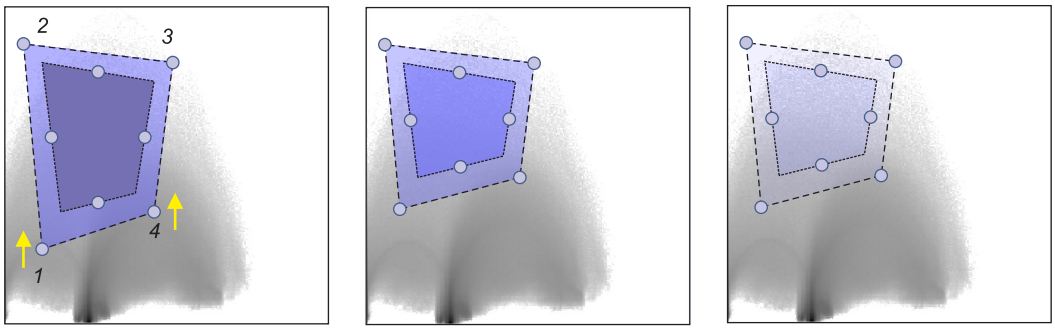
A simple example in Figure 4.6 computes an influence vector for the semantic parameter "*visibility*" exclusively based on the interpolation of the opacity parameter of the primitive. Turning down the opacity will cause a rather homogeneous structure to slowly vanish. In many cases, however, it is desirable to turn down opacity for low gradients first. This will change a previously opaque structure into a transparent shell before it finally completely disappears. Such a behavior can be implemented by including more than just the opacity parameter for interpolation and by using multiple keys for a semantic parameter.

5.4.1.1 Manual Approach

The first approach, which describes the manual implementation of the list of keys, is a fast and easy way to achieve the desired effect. The starting point is the set of parameters $\mathbf{p}_{\text{opaque}}$ which only influences the visibility of the entity's primitive in such a way that the structure of interest is opaque. In order to find a transfer function $\mathbf{p}_{\text{shell}}$ which fulfills the desired request of fading out the structures of interest in a homogeneous region first, the primitive has to change its shape in a way in which it does no longer cover any lower gradients.

This is achieved in a 2D primitive editor by shifting the parameters which control the lower part of the primitive in the direction of the higher gradients. The transparency of the structure, which is controlled by a parameter vector $\mathbf{p}_{\text{transparent}}$, increases simultaneously. Due to the set of parameters $\mathbf{p}_{\text{invisible}}$ the entity finally disappears entirely. Finally, the keys for a semantic visibility parameter can be defined by:

$$
\begin{aligned}
\sigma_0 &= 0.0\,, & \hat{\mathbf{q}}_0 &= \mathbf{p}_{\text{shell}} + \mathbf{p}_{\text{invisible}}\,, \\
\sigma_1 &= 0.5\,, & \hat{\mathbf{q}}_1 &= \mathbf{p}_{\text{shell}} + \mathbf{p}_{\text{transparent}}\,, \\
\sigma_2 &= 1.0\,, & \hat{\mathbf{q}}_2 &= \mathbf{p}_{\text{opaque}}\,.
\end{aligned}
$$



(a) (b) (c)

Figure 5.6: The impact of the semantic parameter *visibility* on the primitive. Figures 5.6(a) and 5.6(b) demonstrate the behavior of the primitive for turning down the semantic parameter. The points 1 and 4 move up in the direction of higher gradient values and the primitive vanishes slowly. Decreasing the value affects the primitive's opacity parameter and causes it to disappear (Figure 5.6(c)).

In Figure 5.6, the outcome of turning down the parameter *visibility* is shown. The primitive becomes more transparent and its shape contracts while the parameter changes. In the next step, the primitive starts to disappear completely. An example of a key configuration of the visibility parameter can be seen in Figure 5.7. The values are based on an adapt template parameter, which adjusts the primitive's shape to the basic structures of the dataset. Because of summing up all semantic parameters, the adapt template parameter provides the primitive's basic shape and the visibility parameter can influence the primitive in the desired way.

$s = 1.0$

$\sigma_0 = 0.0$	$\hat{\mathbf{q}}_0$	=	(	0	0.1	0	0	0	0	0	0.1	0	0	0	0	)
$\sigma_1 = 0.5$	$\hat{\mathbf{q}}_1$	=	(	0	0.1	0	0	0	0	0	0.1	0	0	0	0.65	)
$\sigma_2 = 1.0$	$\hat{\mathbf{q}}_2$	=	(	0	0	0	0	0	0	0	0	0	0	0	1	)
	$\mathbf{q}(s)$	=	(	0	0	0	0	0	0	0	0	0	0	0	1	)
					↓						↓				↓	
					POINT1-Y						POINT4-Y				OPACITY	

Figure 5.7: A more complex example of keys with regard to the semantic visibility parameter s. The y-coordinates of the points 1 and 4 increase between the keys σ_1 and σ_2 when it is assumed that the default coordinate system for primitives has its origin (0, 0) in the bottom-left corner, and the value of the opacity parameter decreases slowly. If s is between σ_0 and σ_1, the position parameters do no longer vary, only the opacity parameter does. The influence vector $\mathbf{q}(s)$ is calculated for $s = 1.0$, and this results in a completely opaque structure.

Please pay attention to the physical accuracy of this technique. The way how the primitive's lower

points in the direction of the higher gradients are moved is based on experience. It is suggested to perform a step-by-step adjustment of the primitive's shape with respect to the desired optical effects in order to design a visibility parameter that refers to the reference datasets more accurately. This method is described in the following.

5.4.1.2 Mean Value Approach

This approach considers the specific structures contained in each reference dataset. First, the semantic parameter *adapt template* is used to yield the best visual results for the structures of interest. This results in a vector $\mathbf{p}_{\text{base } i}$ for each dataset i. These vectors can be modified to adapt the desired effects of a visibility parameter, which results in a vector $\mathbf{p}_{\text{AdjustedVisibility } i,\, j}$ for each dataset i and each step j. It is important to proceed step-by-step, as the primitive's shape should adapt the structures in the dataset as close as possible. With each step the primitive covers less of the lower gradients and is more transparent. The difference vectors

$$\delta_{\text{visibility } i,\, j} \;=\; \mathbf{p}_{\text{AdjustedVisibility } i,\, j} - \mathbf{p}_{\text{base } i} \tag{5.7}$$

are calculated next. The mean values are calculated for all difference vectors of a mutual step j:

$$\overline{\delta}_{\text{visibility } j} = \frac{1}{m} \sum_{i=0}^{m-1} \delta_{\text{visibility } i,\, j} \;. \tag{5.8}$$

A vector $\overline{\delta}_{\text{visibility } j}$ is obtained for each step, and this leads to a semantic parameter *visibility* with the following keys, which are based on k steps:

$$\begin{aligned} \sigma_0 \;&=\; \;\;0.0\,, & \hat{\mathbf{q}}_0 \;&=\; \overline{\delta}_{\text{visibility } 0}\;, \\ \sigma_1 \;&=\; -1.0\,, & \hat{\mathbf{q}}_1 \;&=\; \overline{\delta}_{\text{visibility } 1}\;, \\ &\vdots & & \\ \sigma_{k-1} \;&=\; -(k-1.0)\,, & \hat{\mathbf{q}}_{k-1} \;&=\; \overline{\delta}_{\text{visibility } k-1}\;. \end{aligned} \tag{5.9}$$

5.4.2 Color

The semantic parameter "*color*" enables the user to change the color of an entity. This is achieved by affecting the color parameters of all primitives which belong to the same entity. The semantic parameter is used to control a range of color values. There is no need to apply PCA to any instance vectors, since the keys $(\sigma_j, \hat{\mathbf{q}}_j)$ are defined manually. The step-by-step linear interpolation of colors is given by the low-level parameter vectors $\mathbf{p}_{\text{color } j}$ for each color j. Thus, the semantic parameter *color*

is defined by the following keys:

$$\begin{aligned} \sigma_0 &= 0.0\,, & \hat{\mathbf{q}}_1 &= \mathbf{p}_{\text{color 0}}\,, \\ \sigma_1 &= 1.0\,, & \hat{\mathbf{q}}_1 &= \mathbf{p}_{\text{color 1}}\,, \\ & \vdots & & \\ \sigma_{k-1} &= k - 1.0\,, & \hat{\mathbf{q}}_{k-1} &= \mathbf{p}_{\text{color k-1}}\,. \end{aligned}$$

The parameters COLOR_RED, COLOR_GREEN and COLOR_BLUE control the color of a quadrilateral primitive (see table 4.1), for instance. Figure 5.8 illustrates a simple example of defining the semantic parameter *color*.

$s = 6.5$

$\sigma_0 = 0.0$	$\hat{\mathbf{q}}_0$	=	(	0	0	0	0	0	0	0	0	0	0	0	0	)
$\sigma_1 = 1.0$	$\hat{\mathbf{q}}_1$	=	(	0	0	0	0	0	0	0	0	0	0	1	0	)
$\sigma_2 = 2.0$	$\hat{\mathbf{q}}_2$	=	(	0	0	0	0	0	0	0	0	0	1	0	0	)
$\sigma_3 = 3.0$	$\hat{\mathbf{q}}_3$	=	(	0	0	0	0	0	0	0	0	0	1	1	0	)
$\sigma_4 = 4.0$	$\hat{\mathbf{q}}_4$	=	(	0	0	0	0	0	0	0	0	1	0	0	0	)
$\sigma_5 = 5.0$	$\hat{\mathbf{q}}_5$	=	(	0	0	0	0	0	0	0	0	1	0	1	0	)
$\sigma_6 = 6.0$	$\hat{\mathbf{q}}_6$	=	(	0	0	0	0	0	0	0	0	1	1	0	0	)
$\sigma_7 = 7.0$	$\hat{\mathbf{q}}_7$	=	(	0	0	0	0	0	0	0	0	1	1	1	0	)
	$\mathbf{q}(s)$	=	(	0	0	0	0	0	0	0	0	1	1	0.5	0	)
												↓	↓	↓		
												COLOR_RED	COLOR_GREEN	COLOR_BLUE		

Figure 5.8: Keys for the semantic parameter *color*. A range of colors of the RGB color space is covered by the predefined influences. $\mathbf{q}(s)$ is the result of the linear interpolation between keys $\sigma_6 = 6$ and $\sigma_7 = 7$ for $s = 6.5$. This leads to the color *light yellow* $(1,\ 1,\ 0.5)$.

The colors used in the keys of the example in Figure 5.8 are visualized in Figure 5.9. If the value of the semantic parameter is $s = 6.5$, then the interpolation takes place between the keys $\sigma_6 = 6$ and $\sigma_7 = 7$ and this results in the RGB color *light yellow*.

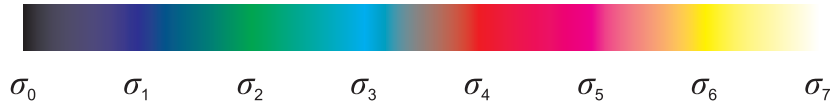


Figure 5.9: Interpolation of colors. The palette of colors is labeled with the keys defined in Figure 5.8.

In chapter 6, a more user-friendly implementation of color definition for entities is introduced, which differs from the separate linear interpolation of keys.

5.4.3 Contrast

In order to create a semantic parameter which controls the contrast between different entities, a procedure similar to the one described in section 5.2 has to be developed. This additional semantic parameter has to be specified with respect to the adapt template parameter because of the sum in equation 4.3. Two methods of defining the semantic parameter "*contrast*" are described in the following sections.

5.4.3.1 Mean Value Approach

Therefore, a transfer function model has to be adapted to all reference datasets. Instead of creating a template from scratch and adapting each parameter to all datasets manually, the semantic parameter *adapt template* may be used. Its value has to be adjusted to the best visual effect in order to identify the specific structures of the datasets. This results in a vector $\tilde{\mathbf{p}}_i$ of low-level parameters for each reference dataset i. The calculation of keys is performed in two steps. At first the modifications are applied to the primitives that are necessary to increase the contrast. This results in a vector $\tilde{\mathbf{p}}_{\mathrm{contrast}+\ i}$ for each dataset. A couple of actions exist that are capable of increasing the contrast between entities. For example, the brightness of the colors can be modified. Another possibility is to edit the opacity slopes at the borders of the primitive. Table 5.1 lists four more parameters which belong to a quadrilateral primitive.

Number	Name	Description
...	...	...
13	SLOPE_BORDER12	opacity slope at border (points 1 and 2)
14	SLOPE_BORDER23	opacity slope at border (points 2 and 3)
15	SLOPE_BORDER34	opacity slope at border (points 3 and 4)
16	SLOPE_BORDER41	opacity slope at border (points 4 and 1)

Table 5.1: Additional parameters of a quadrilateral primitive. Changing the parameters directly influences the opacity slope at the primitive's borders. Possible values are 0 = *no blending* to 1= *maximum blending distance* ranging from the border to the core of the primitive.

After that, this step is repeated, but this time the vectors $\tilde{\mathbf{p}}_{\mathrm{contrast}-\ i}$ contain the decreased contrast parameters for each dataset. Finally, the difference vectors

$$\delta_{\mathrm{contrast}+\ i} = \tilde{\mathbf{p}}_{\mathrm{contrast}+\ i} - \tilde{\mathbf{p}}_i \tag{5.10}$$

$$\delta_{\mathrm{contrast}-\ i} = \tilde{\mathbf{p}}_{\mathrm{contrast}-\ i} - \tilde{\mathbf{p}}_i \tag{5.11}$$

and their respective mean values

$$\overline{\delta}_{\text{contrast}+} = \frac{1}{m} \sum_{i=0}^{m-1} \delta_{\text{contrast}+\, i} \tag{5.12}$$

are calculated (and analogously also for $\overline{\delta}_{\text{contrast}-}$). In the following, the semantic parameter *contrast* is defined with three keys:

$$\begin{aligned} \sigma_0 &= -1.0\,, & \hat{\mathbf{q}}_0 &= \overline{\delta}_{\text{contrast}-}\,, \\ \sigma_1 &= 0.0\,, & \hat{\mathbf{q}}_1 &= \vec{0}\,, \\ \sigma_2 &= 1.0\,, & \hat{\mathbf{q}}_2 &= \overline{\delta}_{\text{contrast}+}\,. \end{aligned}$$

5.4.3.2 PCA Approach

Please note that the values in section 5.4.3.1 are computed solely on the mean values of instance vectors. The method of applying PCA to all instances of difference vectors is a more accurate method. Similar to section 5.3 about data approximation, PCA can be used to extract a single parameter axis in the low-level parameter space of all $\delta_{\text{contrast}+\, i}$ as well as $\delta_{\text{contrast}-\, i}$ vectors. The computation of the keys differs from the proceeding of the adapt template parameter. All vectors $\delta_{\text{contrast}+\, i}$ are projected on the axis spanned by the first component $\mathbf{e}_0$ in order to determine the minimum and maximum values.

$$\mu_{\text{contrast}+\, i} = \mathbf{e}_0 \cdot (\delta_{\text{contrast}+\, i} - \overline{\delta}_{\text{contrast}+}) \tag{5.13}$$

($\mu_{\text{contrast}-\, i}$ is calculated analogously). The keys and predefined influence vectors are then computed by

$$\begin{aligned} \sigma_0 &= -1.0\,, & \hat{\mathbf{q}}_0 &= \overline{\delta}_{\text{contrast}-} + \min(\mu_{\text{contrast}-\, i}) \cdot \mathbf{e}_0\,, \\ \sigma_1 &= 0.0\,, & \hat{\mathbf{q}}_1 &= \vec{0}\,, \\ \sigma_2 &= 1.0\,, & \hat{\mathbf{q}}_2 &= \overline{\delta}_{\text{contrast}+} + \max(\mu_{\text{contrast}+\, i}) \cdot \mathbf{e}_0\,. \end{aligned}$$

5.4.4 Sharpness

Sometimes it is desired to manipulate the sharpness of the structures contained in the dataset. Therefore, the visualized entities are enhanced or dampened. The semantic parameter "*sharpness*", which is able to handle such a justification, can be implemented in different ways.

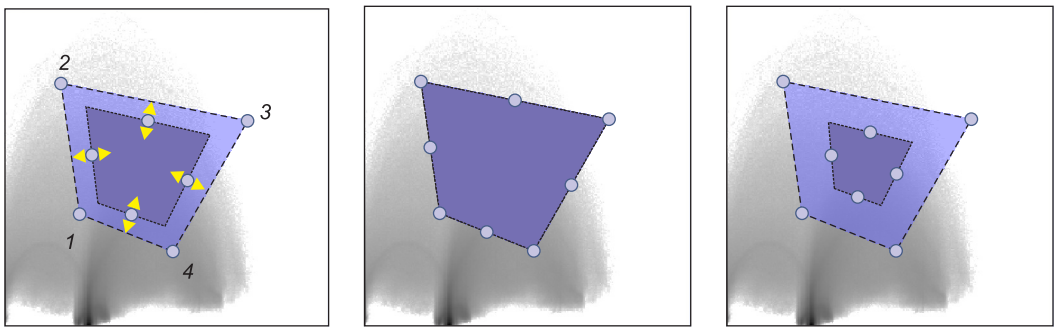
5.4.4.1 Manual Approach

This approach describes the manual implementation of keys. A vector $\mathbf{p}_{\text{sharpness}+}$, which enhances the structures of interest, is defined, as well as a vector $\mathbf{p}_{\text{sharpness}-}$, which decreases the sharpness. A

possible way to change the visual appearance in the requested manner is to modify the opacity slopes at the border of a primitive (see table 5.1). Hence, the keys for the semantic parameter *sharpness* are defined by

$$\begin{aligned} \sigma_0 &= -1.0\,, & \hat{\mathbf{q}}_0 &= \mathbf{p}_{\text{sharpness}-}\,, \\ \sigma_1 &= 0.0\,, & \hat{\mathbf{q}}_1 &= \vec{0}\,, \\ \sigma_2 &= 1.0\,, & \hat{\mathbf{q}}_2 &= \mathbf{p}_{\text{sharpness}+}\,. \end{aligned}$$

The impact of the modification of opacity slope parameters on the primitive is shown in Figure 5.10. The more the sharpness value is increased, the closer the opacity slope control points move to the outer border. The opposite happens while turning down the semantic value: the control points move to the interior of the shape.



(a) (b) (c)

Figure 5.10: Quadrilateral primitive and opacity slope parameters. In Figure 5.10(a), all opacity slope parameters are in a middle position. This is in contrast to Figure 5.10(b), which contains all opacity slope parameters of the value 0 (no blending) and to Figure 5.10(c), which contains all opacity slope parameters of the value 1.0 (maximum blending distance).

A sample configuration of keys of the parameter *sharpness* is shown in Figure 5.11. The meaning of the values is explained in table 5.1. The resulting low-level vector of the sharpness parameter has to be considered with respect to the influence vector based on the output of an adapt template parameter which controls the primitive's basic shape, since the mapping function of equation 4.3 sums up the influences of all semantic parameters.

Again, this solution is not based on the individual structures of each reference dataset. The quality of the result depends strongly on the outcome of the adapt template parameter. For example, if the structures are already enhanced to a great extend by the use of opacity slope parameters, then it is hardly possible to use the sharpness parameter to sharpen the entity's structure even more, although it may be a value which still should have the possibility to enhance or to dampen the structure.

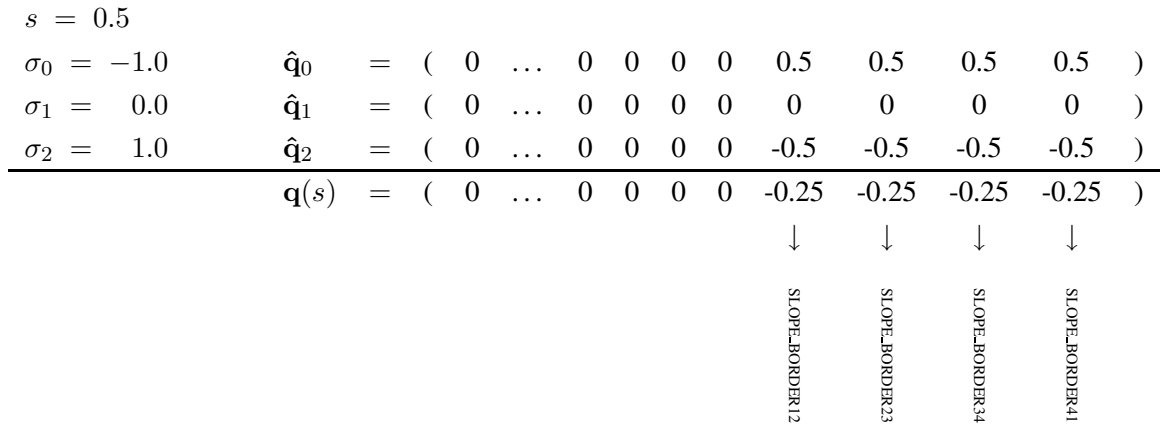


Figure 5.11: Keys for the semantic parameter *sharpness*. The vector $\mathbf{q}(s)$ of the sharpness parameter has to be considered with respect to the influence vector based on the output of an adapt template parameter. In this example, the opacity slope parameters are reduced, which leads to a rather sharpened structure.

5.4.4.2 Mean Value Approach

The second approach is similar to the one described in section 5.4.3.1, which concerns the semantic parameter *contrast*. It takes all reference datasets into account.

The adapt template parameter adapts the transfer function as good as possible, and this results in a vector $\tilde{\mathbf{p}}_i$ of low-level parameters for each reference dataset i. In the next step, modifications are applied to the primitive in order to enhance the structures of interest. For example, these modifications are realized by adapting the opacity slopes, and this leads to a vector $\tilde{\mathbf{p}}_{\text{sharpness+}\,i}$ for each dataset. This step is also accomplished for vectors $\tilde{\mathbf{p}}_{\text{sharpness-}\,i}$, which contain the variants of the structures which are less sharp. The next step is the calculation of the difference vectors

$$\delta_{\text{sharpness+}\,i} = \tilde{\mathbf{p}}_{\text{sharpness+}\,i} - \tilde{\mathbf{p}}_i \tag{5.14}$$

$$\delta_{\text{sharpness-}\,i} = \tilde{\mathbf{p}}_{\text{sharpness-}\,i} - \tilde{\mathbf{p}}_i \tag{5.15}$$

and finally, their respective mean values are computed by

$$\overline{\delta}_{\text{sharpness+}} = \frac{1}{m} \sum_{i=0}^{m-1} \delta_{\text{sharpness+}\,i} \ . \tag{5.16}$$

The computation has to be repeated analogously for $\overline{\delta}_{\text{sharpness-}}$. According to this approach, the configuration of keys for the semantic parameter *sharpness*, which is calculated with mean values,

reads as follows:

$$\begin{aligned} \sigma_0 &= -1.0\,, & \hat{\mathbf{q}}_0 &= \bar{\delta}_{\text{sharpness}-}\,, \\ \sigma_1 &= 0.0\,, & \hat{\mathbf{q}}_1 &= \vec{0}\,, \\ \sigma_2 &= 1.0\,, & \hat{\mathbf{q}}_2 &= \bar{\delta}_{\text{sharpness}+}\,. \end{aligned}$$

In addition to the techniques of defining keys for a sharpness parameter, which have already been introduced, an approach based on PCA would be also possible[2].

5.4.5 Discussion

In this section, the design of additional semantics such as *visibility*, *color*, *contrast* and *sharpness* is proposed. Most of the semantics can be implemented in many different ways. The process of differentiating manual key constructions and of applying PCA to instance vectors, as well as keys based on mean values referring to all reference datasets, all result in different accuracies concerning the adaption of the underlying transfer function to the peculiarities of each dataset.

In addition to this, there may be different modifications to the primitives that yield similar results. In order to decrease the visibility of a structure, either the opacity value can be turned down or the size of the primitives can be reduced, for instance. For this reason it is necessary to use similar actions to achieve a specific task for all reference datasets, otherwise the calculation of the final low-level parameter vector will fail when summing up each influence of a semantic parameter.

[2]The approach of using PCA to calculate the keys of the semantic parameter *sharpness* would be similar to the solution demonstrated in section 5.4.3.2.

Chapter 6

Implementation

For the development and testing of semantic transfer function models, a framework is necessary that can be used for input, output, user interaction, and the visualization of the data. As semantic models are closely connected to traditional transfer function models, user interfaces for 1D and 2D transfer functions are needed as well as an interface for the remote control of the transfer functions. In addition to this, the rendering system should be able to handle volume visualization with interactive frame rates on general purpose hardware. An implementation of a semantic transfer function model software has been realized within the scope of this diploma thesis.

In this chapter, "*OpenQVis*" - a framework which provides methods for interactive high-quality volume visualization on general purpose hardware - is presented. The aim of this rendering system is to achieve high image quality comparable to traditional ray-casting solutions at interactive frame rates on inexpensive hardware platforms. The framework has been extended with different transfer function editors, including the high-level user interface based on semantic transfer function models.

Section 6.1 presents an overview of the core architecture and describes the main features of the system. Later on, the traditional transfer function editors and the new semantic editor are introduced.

6.1 Architecture

The purpose of the OpenQVis system[1] is to provide a volume rendering environment which achieves high image quality and interactive frame rates. The program is based on a volume rendering system which uses the graphics processing unit (GPU) for rendering algorithms. The system is implemented in C++ and OpenGL. Rendering techniques are supported that use both 3D texture slicing and 2D multi-texture based rendering. OpenQVis is programmed by using the following three components: Qt = a cross-platform C++ development library, Coin3d = a high-level 3D graphics toolkit for developing cross-platform real-time 3D visualization and visual simulation software fully compatible with SGI's

[1]The OpenQVis project was started in the year 2000 [RS02]. It is free software and can be redistributed or modified under the terms of the GNU General Public License as published by the Free Software Foundation. For more information please see `http://openqvis.sourceforge.net`.

Open Inventor, and SoQT = the interface between Qt and Coin3d.

The transfer functions have been implemented with a programmable graphics hardware and post-interpolative 1D and 2D dependent texture lookups.

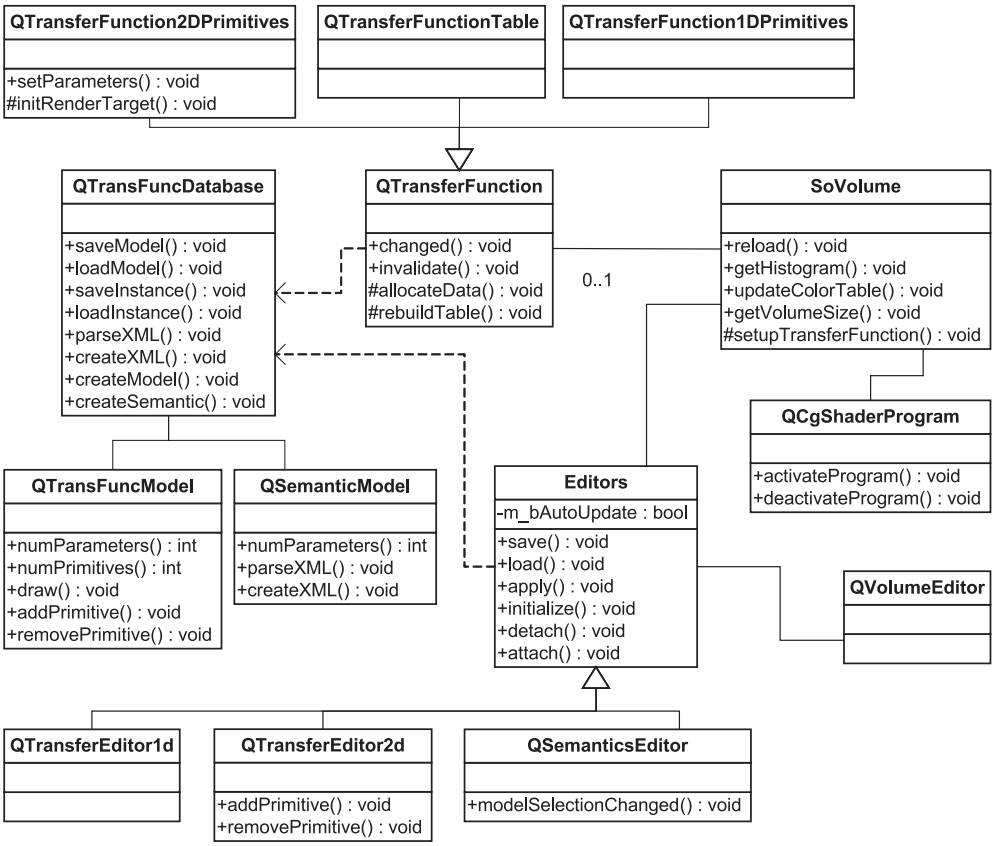


Figure 6.1: An UML component diagram of the OpenQVis kernel components.

The core of the software is an open inventor node (SoVolume) which realizes the volume rendering algorithms by using cg (c for graphics), a high-level shader language (QCgShaderProgram). The system is extended with several kinds of transfer functions (e.g., QTransferFunctions2DPrimitives). These generate textures which are used by the rendering component. Corresponding editors (QTransferEditor1d, QTransferEditor2d, QSemanticsEditor) provide appropriate user interfaces to control the transfer functions. Each transfer function model (QTransFuncModel) adjusted by the user is managed by a database (QTransFuncDatabase) which keeps the models organized and which can connect transfer functions with a semantic model (QSemanticModel). A unified modeling language (UML) component diagram of the basic parts is shown in Figure 6.1. The architecture of the program is simplified and outlined in order to keep the model understandable.

6.2 Traditional Transfer Functions and Editors

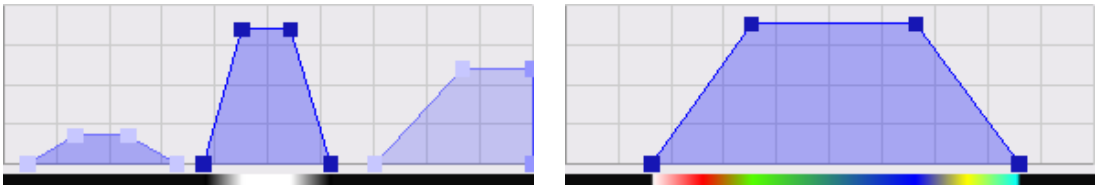
The rendering system supports different types of transfer functions. Since all transfer functions should be relatively easy to handle for the user, the editors provide primitives that are more convenient to adjust than splines, curves, or lines separated by RGBA channels. However, the system can be extended easily by implementing additional primitives in order to adapt the structures of the dataset as good as possible. Various transfer functions and editors are presented in the following sections. The 1D transfer function is briefly described, whereas the 2D transfer function is discussed more detailed.

6.2.1 1D Transfer Functions

As already described in section 3.2, the easiest way to define a 1D transfer function is to determine RGBA entries in a fixed size 1D lookup table of the number of scalar values. Instead of specifying the entries manually, the transfer function can be modeled as a superposition of separate primitive objects. For that reason, the main benefit of primitive objects lies in the significant reduction of the degrees of freedom that the user must manage in order to establish a suitable transfer function.

The 1D transfer function is configured in the 1D transfer function editor, which supports two basic types of primitives, namely:

- Trapezoids with 5 parameters: 2 values for the x-coordinates of the lower vertices, 2 values for the x-coordinates of the upper vertices, and one parameter for the height of the trapezoid. In addition to this, 6 different RGB colors can be defined for color interpolation.
- Ramps with 3 parameters: one value for the x-coordinate of the lower vertex, one value for the x-coordinate of the upper vertex, and one parameter for the height of the ramp. The colors can be adjusted by defining 4 different RGB colors.



(a) (b)

Figure 6.2: 1D Primitive Editor. In Figure 6.2(a), the primitives on the left and in the middle are trapezoids with different sizes, and the right primitive is a ramp. The interpolation of colors assigned to the primitive is illustrated by the *colorbar* visible in the bottom edge in Figure 6.2(b).

Some primitive objects of 1D transfer functions are shown in Figure 6.2. All parameters, except those for colors, can be modified by moving the handles marked in Figure 6.3. The primitive's position and shape both specify the data that are assigned with color and opacity values. According to this, the

range of values which is not covered by the primitive's dimension is mapped to zero opacity, therefore these values are not colored in the resulting image. The color and opacity values are interpolated linearly between the position parameters. It is possible to adjust different shapes, which are useful to meet a wide range of application scenarios.

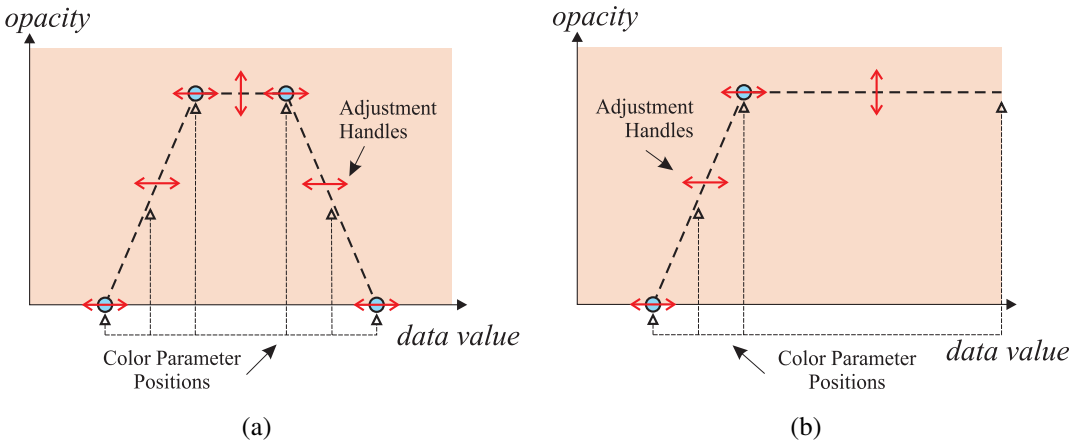


Figure 6.3: Primitives for 1D transfer functions. In Figure 6.3(a), a trapezoid is displayed and in Figure 6.3(b) a ramp. The positions, where the colors are defined and interpolated, are marked. The red arrows indicate the directions of the control points, which are movable by the user. They can be translated in the directions of the arrowheads.

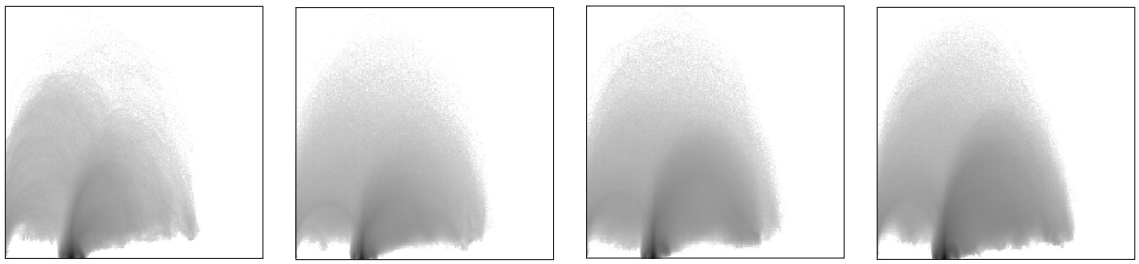
An arbitrary number of primitives can be added in order to define the transfer function. Originally, the trapezoid is placed in the center of the visible area of the transfer function editor and can be converted easily to a simple ramp. The primitives are adapted to the data by moving the handles as illustrated in Figure 6.3. Some restrictions are implemented, because the creation of deformed primitives should be avoided. Its components are movable only within a valid range. For example, the height which defines the opacity cannot become smaller than zero. Finally, the transfer function editor offers a couple of ways to assign colors to scalar values. On the one hand, colors can be applied to the primitives; on the other hand, color tables can be used, which are already defined and which specify a RGB color for each scalar value. If colors and data values are linked to each other and the primitives' are used to determine opacity values, the color tables are helpful to visualize the various structures which the dataset contains.

6.2.2 2D Transfer Functions

In contrast to 1D transfer functions which classify a sample based on a single scalar value, 2D transfer functions allow a sample to be classified based on a combination of values. These values are the axes of a 2D transfer function. The gradient is a first derivative for volume datasets based on scalar values. As a vector, it points to the direction of the fastest change. The gradient magnitude is another fundamental local property of a scalar field, as it characterizes how fast values are changing (see

section 3.2). By using gradient magnitude as a second dimension it is assumed that regions of change tend to be regions of interest. This allows the structure to be differentiated with varying opacity or varying color, according to the magnitude of change. The gradient values used in the system are calculated with the help of a multi-level technique. Each level is created by down-sampling the previous level. The gradient magnitude is calculated for each level separately. After that, each level is sampled up to the resolution of the previous level and is averaged. This technique results in smooth gradient data without the necessity of additional filtering.

2D histograms are useful to adjust the transfer function to the structures of interest. In the scatterplot images, the data value and the gradient magnitude are aligned to the axes. As already pointed out in section 3.2, the darkness in the scatterplots encodes the number of hits for a given pair of gradient magnitude and data value. Figure 6.4 illustrates the slightly different structures of the datasets which are visualized in histograms.

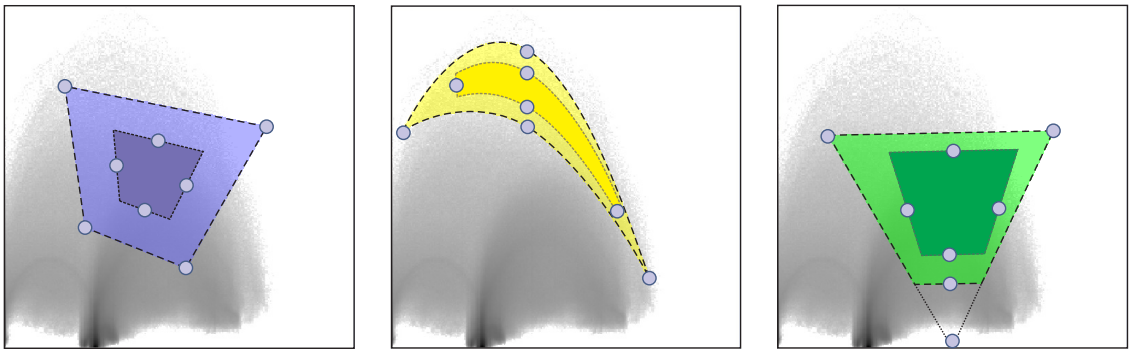


(a) (b) (c) (d)

Figure 6.4: Histograms of CTA datasets. Various datasets result in slightly different histograms which are part of the primitive editor. A histogram of dataset CTA_12 is shown in Figure 6.4(a), CTA_22 in Figure 6.4(b), CTA_29 in Figure 6.4(c), and CTA_38 in Figure 6.4(d).

With regard to the reduction of the degrees of freedom that the user must manage, the 1D approach of using primitives also works in multi-dimensions, although it is hard to find adequate primitives. The different types of 2D primitives supported by OpenQVis are shown in Figure 6.5 and comprise:

- Quadrilaterals with 16 parameters: 8 values for the (x, y)-coordinates of the four vertices, 4 parameters for the color and the opacity (RGBA) of the primitive and 4 parameters for the opacity slopes at the borders.
- Trapezoid primitives with 14 parameters: one value for the x-position of the base vertex, 4 values for the (x, y)-coordinates of the two upper vertices, and one parameter for the shifting the lower base line, as well as the parameters for RGBA and for the opacity slopes which are the same as in the quadrilateral.
- Parabolic primitives with 15 parameters: 6 parameters for the (x, y)-coordinates of the control points for the upper arc and one parameter for the lower indent, as well as the parameters for RGBA and for the opacity slopes which are the same as in the trapezoid primitive.



(a) (b) (c)

Figure 6.5: 2D Primitive Editor. The primitives are colored according to the color parameters. Figure 6.5(a) shows a quadrilateral primitive, Figure 6.5(b) a paraboloid primitive, and Figure 6.5(c) a trapezoid primitive.

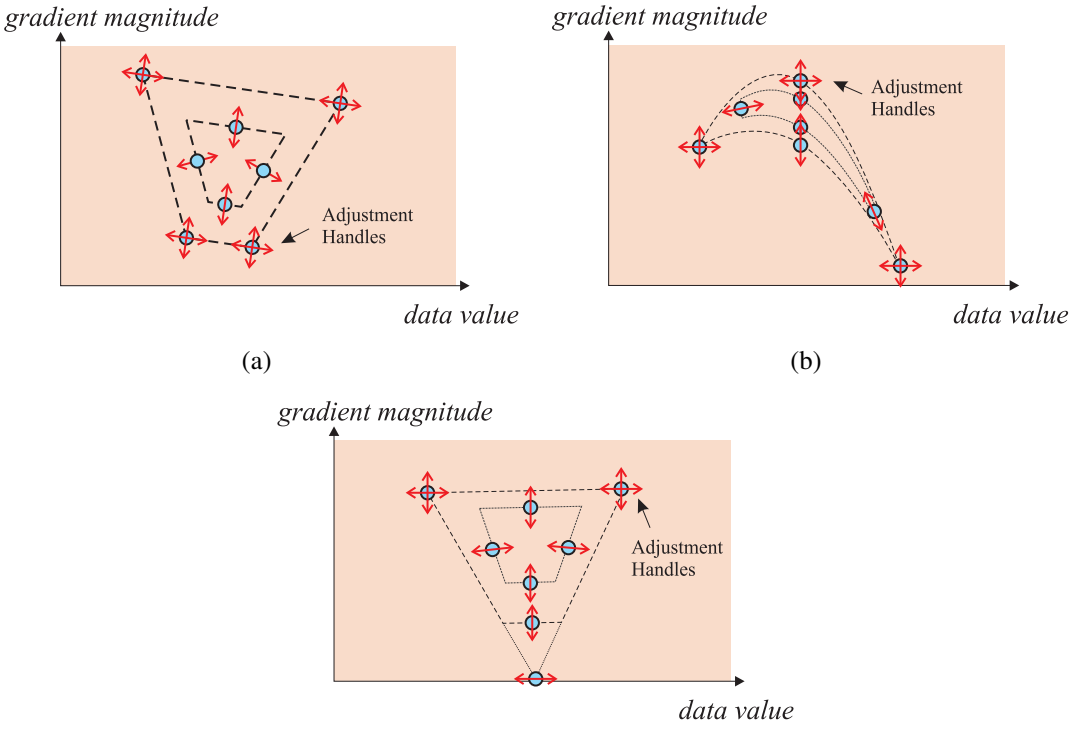
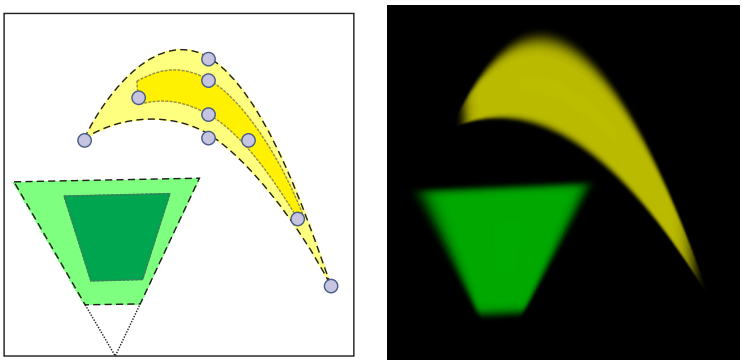


Figure 6.6: Primitives for 2D transfer functions. The three primitives are: a quadrilateral in Figure 6.6(a), a paraboloid in Figure 6.6(b), and a trapezoid in Figure 6.6(c). The red arrows indicate the directions of the movable control points.

All parameters except those for RGBA can be modified by moving the control points as shown in Figure 6.6. The color is assigned to the entire primitive and the opacity is interpolated linearly starting from the primitive's border up to the opacity slope control points, whereas the maximum opacity value is modified separately.

With the implementation of additional primitives, the system can be extended easily by deriving from the base class *QPrimitive* which provides several interfaces which are responsible for a conversion to an XML file, the assignment of colors, and the drawing of elements. It is possible to add as many primitive objects as required to the scene in order to obtain the desired effect. Then, the 2D transfer function table is generated by rendering the shape of the primitives into an off-screen render target. Finally, this is bound to a 2D dependent texture for volume rendering (see Figure 6.7). The correct composition of overlapping the primitives is important. In practice, the RGB values are blended based on their opacities. Later on, a maximum operation is used to calculate the final opacity value.



(a) (b)

Figure 6.7: An off screen render target which is bound to a 2D dependent texture for volume rendering. Figure 6.7(a) shows a paraboloid and a trapezoid primitive adjusted in the primitive editor. Figure 6.7(b) displays the primitives directly rendered into an off-screen buffer, which will be bound to a 2D dependent texture later on.

6.3 Transfer Function Designer

All transfer functions can be saved in an XML file format for further editing. Every change of the primitives in the editors influences its parameters and is directly mapped to the low-level parameters which describe the transfer function. The file contains the root element *Model* which enables a registration by the database. Thus, the transfer function can completely be restored. The file continues by defining the child element *TransferFunctionModel* which includes an element *Primitives* which has an arbitrary number of sub elements, with each of these containing a single primitive. The sub elements which are available are *Quad2D*, *Paraboloid2D*, and *Paraboloid2D*. These elements have an attribute *name* and a child element *Parameters* containing all low level parameters of the primitive.

Listing 6.1 presents the code of a transfer function consisting of a trapezoid and a paraboloid primitive. Please note that the element *Parameters* with its values in line 6 and 11 is explained in section 6.2.2.

```
1  <Model name="CT_test_model">
2   <TransferFunctionModel name="Transfer Function">
3    <Primitives>
4     <Trapezoid2D name="Structure1">
5      <Parameters>
6       0.3 0.1 0.5 0.6 0.4 0.2 0 0.6 0 1 0.6 0.5 0.4 0.5
7      </Parameters>
8     </Trapezoid2D>
9     <Paraboloid2D name="Structure2">
10     <Parameters>
11      0.2 0.3 0.9 0.8 0.5 0.1 0.3 1 1 0 0.7 0.5 0.7 0.3 0.5
12     </Parameters>
13    </Paraboloid2D>
14   </Primitives>
15  </TransferFunctionModel>
16 </Model>
```

Listing 6.1: An XML file of a transfer function consisting of a trapezoid primitive and a paraboloid primitive.

Each instance of the transfer functions which is adapted to the reference datasets is saved in the file format described above. These files form the basis for the analysis of dimensionality reduction and data approximation as already mentioned in section 5.3. The automatic performance of PCA for a design of the semantic parameter *adapt template* is made with the tool "*Transfer Function Designer*", which completes OpenQVis.

In order to apply PCA to the low-level parameters of the transfer functions, the files are split into groups of primitives which belong to the same entity. This results in a file for each entity and for each dataset containing the entity's primitives. The entities of all files of a common dataset are merged in order to ensure the proper order of parameters. To be precise, every entity file is filled up with zero values for the parameters of the remaining entities. As the parameters of the primitives are concatenated when applying PCA, it is important that primitives are sorted in the same way with regard to all resulting files for all datasets. This guarantees a correct correlation of the dimensions of the input vectors for PCA. When loading files of each entity and of each reference dataset into the Transfer Function Designer, PCA is performed separately for each entity.

The result is an XML data file including information for semantic models which are arranged by semantic parameters and which are grouped with respect to their entities. Figure 6.8 illustrates the process of creating the XML data file, which also provides information for the user interface design of a semantic model, for example, minimum and maximum values for user interface components. Please see section 6.4.1 for further information about the output file. Figure 6.9 shows files which are split and merged on the basis of the code sample from Listing 6.1.

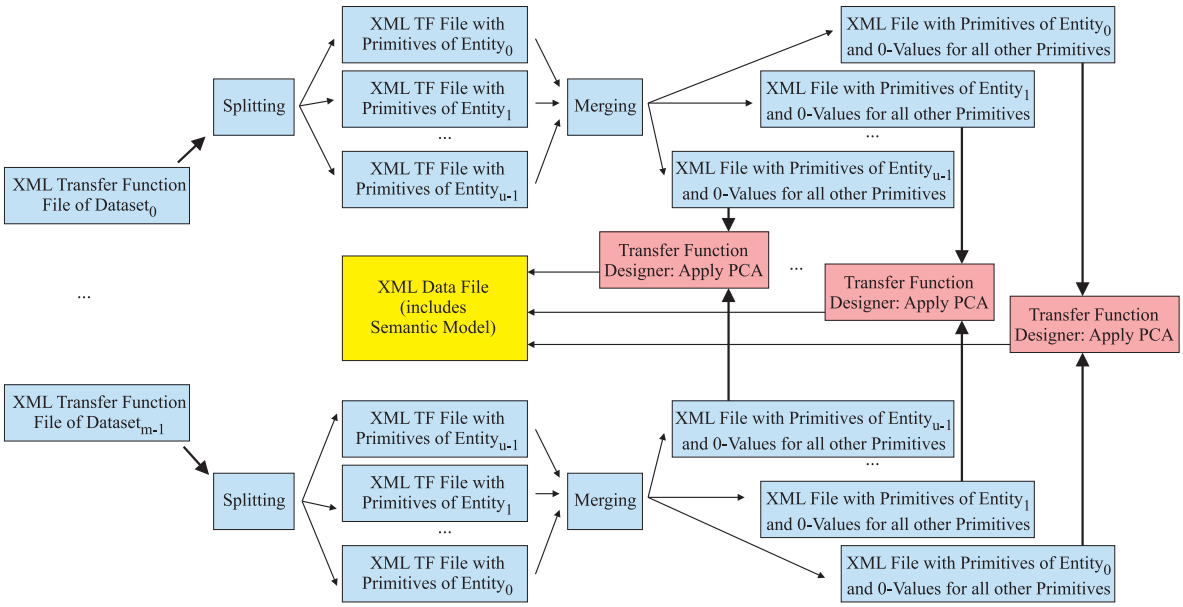


Figure 6.8: The process of XML data file creation. The instances of different transfer functions of the reference datasets are prepared for further analysis.

```
<Model name="CT_test_model">
 <TransferFunctionModel name="Transfer Function">
  <Primitives>
   <Trapezoid2D name="Structure1">
    <Parameters>
     0.3 0.1 0.5 0.6 0.4 0.2 0 0.6 0 1 0.6 0.5 0.4 0.5
    </Parameters>
   </Trapezoid2D>
  </Primitives>
 </TransferFunctionModel>
</Model>
```

(a) `Structure1_split.xml`

⇓

```
<Model name="CT_test_model">
 <TransferFunctionModel name="Transfer Function">
  <Primitives>
   <Trapezoid2D name="Structure1">
    <Parameters>
     0.3 0.1 0.5 0.6 0.4 0.2 0 0.6 0 1 0.6 0.5 0.4 0.5
    </Parameters>
   </Trapezoid2D>
   <Paraboloid2D name="Structure2">
    <Parameters>
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    </Parameters>
   </Paraboloid2D>
  </Primitives>
 </TransferFunctionModel>
</Model>
```

(c) `Structure1_merged.xml`

```
<Model name="CT_test_model">
 <TransferFunctionModel name="Transfer Function">
  <Primitives>
   <Paraboloid2D name="Structure2">
    <Parameters>
     0.2 0.3 0.9 0.8 0.5 0.1 0.3 1 1 0 0.7 0.5 0.7 0.3 0.5
    </Parameters>
   </Paraboloid2D>
  </Primitives>
 </TransferFunctionModel>
</Model>
```

(b) `Structure2_split.xml`

⇓

```
<Model name="CT_test_model">
 <TransferFunctionModel name="Transfer Function">
  <Primitives>
   <Trapezoid2D name="Structure1">
    <Parameters>
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    </Parameters>
   </Trapezoid2D>
   <Paraboloid2D name="Structure2">
    <Parameters>
     0.2 0.3 0.9 0.8 0.5 0.1 0.3 1 1 0 0.7 0.5 0.7 0.3 0.5
    </Parameters>
   </Paraboloid2D>
  </Primitives>
 </TransferFunctionModel>
</Model>
```

(d) `Structure2_merged.xml`

Figure 6.9: The splitting and merging of files for the Transfer Function Designer. In Figures 6.9(a) and 6.9(b), the code of Listing 6.1 is split into separated primitives. In Figures 6.9(c) and 6.9(d), the files are merged with all primitives and filled up with parameters of the value zero. It is assumed that the primitive *Structure1* belongs to a different entity than *Structure2* does.

6.4 Semantic Transfer Functions

In addition to traditional transfer functions, the framework offers a transfer function design based on semantics. In Figure 6.10(a), the architecture of the semantic parts is briefly outlined in an UML diagram. The semantic model class (QSemanticModel), which is connected to the database (QTransFuncDatabase) (see program architecture in Figure 6.1), keeps the semantic features organized and sums up the influences of the semantic parameters. Qt's model/view framework is implemented in order to visualize the current semantic values. The structure of the semantic parameters is created internally with a connection of entities (QSemanticEntity) and parameters (QSemanticParameter) in a highly variable order, which offers a more flexible structure as demonstrated in Figure 6.10(b).

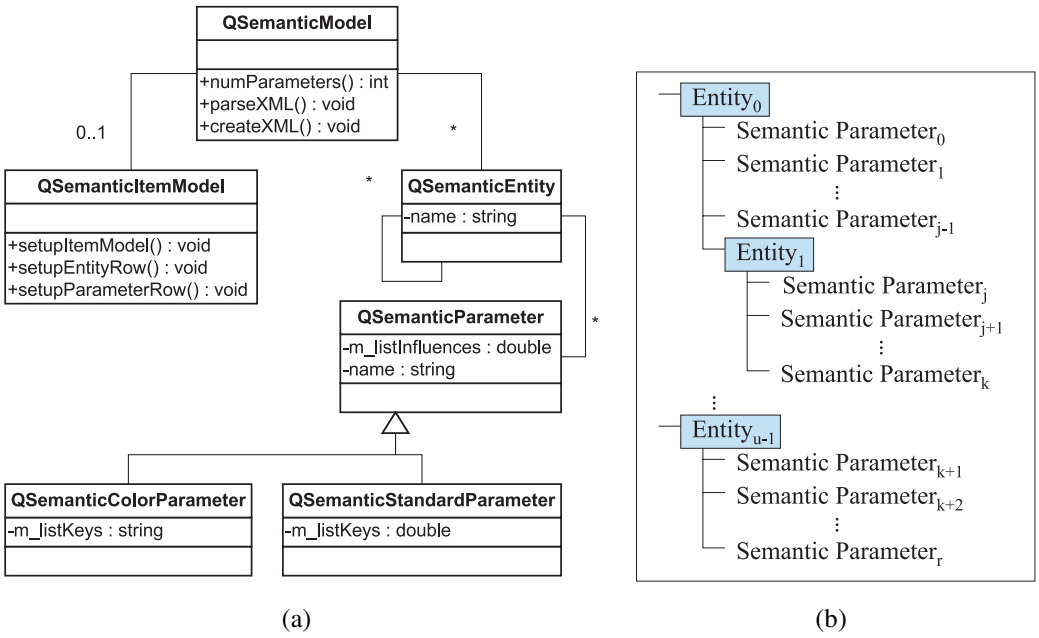


Figure 6.10: An UML diagram of semantic model components and an entity structure including semantic parameters. In Figure 6.10(a), an UML component diagram of the OpenQVis components is shown, which deals with semantic models. Figure 6.10(b) illustrates a nested entity structure with semantic parameters.

All semantic parameters contain lists of key/value pairs for interpolation. Then, the influence vector of each semantic parameter is calculated dependent on its current value. Please note that the length of the arrays for interpolation as well as the length of influence vectors is always the sum of the number of parameters of all primitives of which the transfer function consists. Only the values which are affected by a semantic parameter are different from zero.

6.4.1 XML Data File

The XML data file is computed by the Transfer Function Designer. On the one hand, it contains information about the transfer function and its initial values; on the other hand, it provides basics for a semantic user interface, which facilitates the adjustment of transfer functions based on a semantic model. The syntax of the file is XML and can be edited and extended easily by the user. A Document Type Definition (DTD) is available in the appendix (Listing A.1).

The first part of the data file describes the transfer function model. Similar to Listing 6.1, all primitives the transfer function is composed of are listed, including their names and parameters. The transfer function can be interpreted by the database without any additional information and primitives will be restored. Then, the primitives can be transformed in the primitive editor. In addition to this, the second part of the file contains a semantic model for the design of a novel user interface, which enables the user to control the adaption of the transfer function to new datasets without any knowledge of the underlying transfer function model. The semantic model is grouped by entity elements which are usually named according to the structures of the dataset which they represent. Possible sub elements of an entity are semantic parameters such as the parameters *visibility* and *contrast*, as well as *sub entities* (see Figure 6.10(b)).

```
1  <Entity name="Vessels">
2    <StandardParameter name="Adapt Template" type="double"
3                                             min="0" max ="1"
4                                             default="0.5">
5      <Influence key="0">
6        ... 0.1 0.7 0.1 0.8 0.2 1   0.3 0.7 0 0 0 0 1 0 1 1 ...
7      </Influence>
8      <Influence key="0.5">
9        ... 0.2 0.8 0.2 0.9 0.3 0.9 0.3 0.8 0 0 0 0 1 0 1 1 ...
10     </Influence>
11     <Influence key="1">
12       ... 0.3 0.8 0.3 0.9 0.4 0.9 0.4 0.9 0 0 0 0 1 0 1 1 ...
13     </Influence>
14   </StandardParameter>
15  ...
16 </Entity>
```

Listing 6.2: An excerpt of an XML data file showing key definitions of entity *vessels*.

A reason for using nested entities might be the reproduction of anatomical structures controlled by the model. For example, the entity *bone* is divided into an *outer* and *inner* region. In order to initialize the semantic parameters, necessary information is provided by attributes such as default-, minimum- and maximum values, as well as the name and data type. The influence of each of the semantic parameters on the low-level parameters is specified by a set of keys implemented as sub elements. In Listing 6.2, a code excerpt of the semantic parameter *adapt template* is shown which is provided by the Transfer Function Designer according to the transfer function instances of the template. Then this

template is adapted to the reference datasets[2].

In addition to the element *StandardParameter*, an element *ColorParameter* is introduced. The set of keys $(\sigma_j\, \hat{\mathbf{q}}_j)$ for color parameters is defined by splitting the color into its components. In variation to other semantic parameters, which are calculated by a step-by-step linear interpolation, the influence of a color parameter $s_{color} \in \mathbb{R}^k$ is specified by the multiplication of each individual component

$$\mathbf{q}(s_{color}) = \begin{cases} s_{color}[0] \cdot \hat{\mathbf{q}}_0, & \text{for } \sigma_0 \\ s_{color}[1] \cdot \hat{\mathbf{q}}_1, & \text{for } \sigma_1 \\ \vdots & \vdots \\ s_{color}[k-1] \cdot \hat{\mathbf{q}}_{k-1}, & \text{for } \sigma_{k-1} \end{cases}$$

with $\mathbf{q}(\sigma_j) = \hat{\mathbf{q}}_j$ and σ_j is an arbitrary name with regard to each component of s_{color} . Examples for the attribute *mode* are the color models RGB and CMYK. An advantage of this technique is the simplified color management which enables the user to specify an entity's color directly. Listing 6.3 illustrates the syntax and key definition of the element *ColorParameter*.

```
<ColorParameter name="Color" type="color"
                             mode="rgb"
                             default="0.3 0.6 0">
  <Influence key="red">
    ... 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 ...
  </Influence>
  <Influence key="green">
    ... 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ...
  </Influence>
  <Influence key="blue">
    ... 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 ...
  </Influence>
</ColorParameter>
```

Listing 6.3: The element *ColorParameter* for simplified color management.

6.4.2 User Interface Generation

The semantic model defined in the XML data file is edited and adjusted by the user in the semantic editor, which is implemented by using Qt's UI components. The user interface is generated by parsing the data file and translating it into a clearly structured editor. A model/view architecture [GHJV94] forms the basis for managing the relationship between the data and the way it is presented to the user. In general, the model/view classes which are provided with Qt can be separated into the three groups *models*, *views*, and *delegates*, which communicate with each other. Each of these components, which can be extended easily, is defined by classes which provide mutual interfaces.

[2] In the sample code, the low-level parameters of all remaining primitives are indicated by inverted commas, because otherwise the concatenation of primitives would be too elaborate (see section 5.2).

- **Model**. The model is addressed by views and delegates to access data. It also informs the view about changes to the data held by the data source.
- **View**. The view provides information about the user's interaction with the items being displayed.
- **Delegate**. A delegate renders the items of data. During the process of editing, the delegate tells the model and the view about the state of the editor.

The separation of functionality introduced by this architecture offers greater flexibility to customize the presentation of items. Figure 6.11(b) illustrates the architecture of the editor. The delegate part manages the central user interface components, including sliders and spin boxes which are shown in Figure 6.11(a). Both elements are highly user friendly considering the adjustment of a single value. Spin boxes offer the possibility to enter values manually for fine tuning, for example.

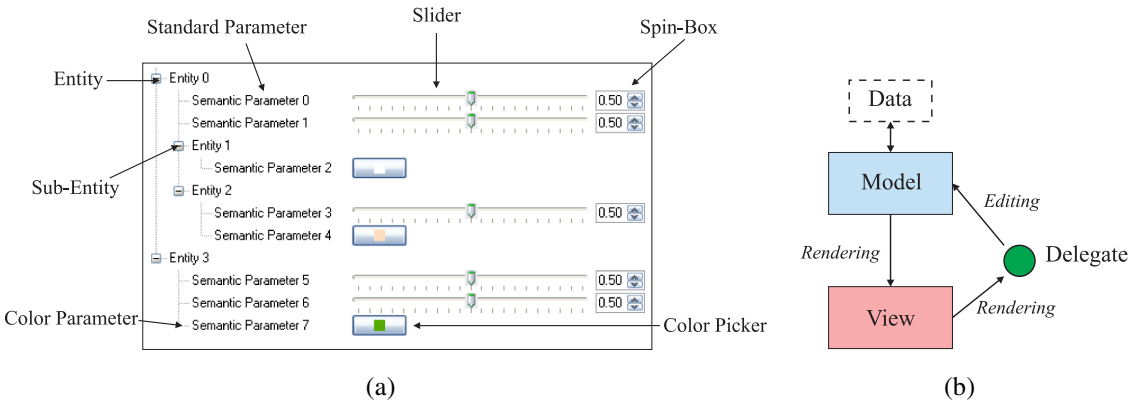


Figure 6.11: Figure 6.11(a) describes the user interface components of the semantic editor. In Figure 6.11(b) the concept of the model/view architecture is illustrated.

The user determines the current values of semantic parameters by moving and editing the user interface components. As already explained in section 4.3, the influence of a semantic parameter s_i is specified by the set of keys. The involved keys σ_j and σ_{j+1}, which are interpolated linearly, are chosen with regard to the current value of the semantic parameter, $\sigma_j \leq s_i < \sigma_{j+1}$. The sliders are movable within the range which is set in the data file. Alternatively, color semantic parameters, mainly color parameters based on ColorParameter elements, are specified with a color picker. As already mentioned in section 6.4.1, each of the keys is multiplied by the corresponding color component.

6.4.3 Parameter Back Mapping

Modifying semantic parameters by moving sliders, for example, results in a parameter change of the underlying transfer function. This means that the change of a semantic parameter causes the recalculation of the final low-level parameter vector $\mathbf{p}$ by summing up the influences of all semantic parameters

(see 4.3). Please remember that the transfer function consists of low-level parameters of concatenated primitives. Thus, the modification of the semantic parameters is immediately mapped back to the low-level parameters of the primitives. The relationship of parameters and primitives is based on the order of the primitives which are defined in the data file's section about the transfer function model. Finally, the vector of low-level parameters is decomposed and assigned to the matched primitives. A signal informs the primitive editor to redraw its primitives because of the change of the parameters, which leads to an updated shape of the primitives. With regard to 1D transfer functions, a new 1D texture is calculated which functions as a lookup table. With respect to 2D transfer functions, the shape of the primitives is rendered into an off-screen render target bound to a 2D texture as already described in section 6.2.2.

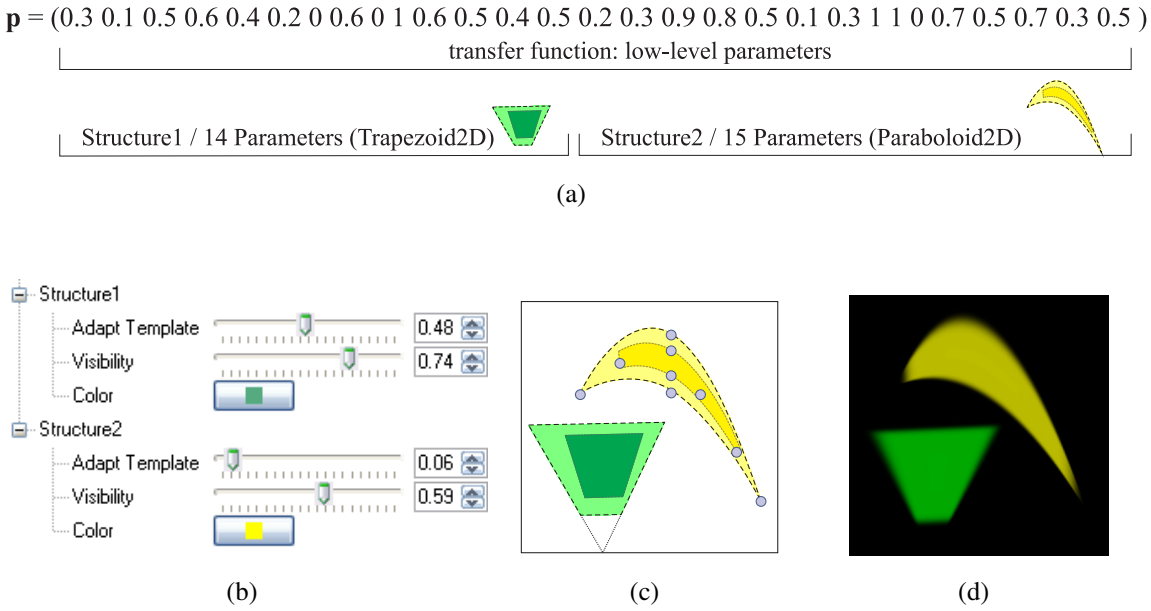


Figure 6.12: The semantic editor and the process of parameter mapping. The semantic editor which is shown in Figure 6.12(b) is useful to visualize the structures of the dataset without any knowledge about the underlying transfer function. The low-level parameters, which are changed by the semantic editor, are mapped back to the primitives (see Figures 6.12(a) and 6.12(c)). The primitives are rendered directly into an off-screen buffer which is demonstrated in Figure 6.12(d).

Chapter 7

Results

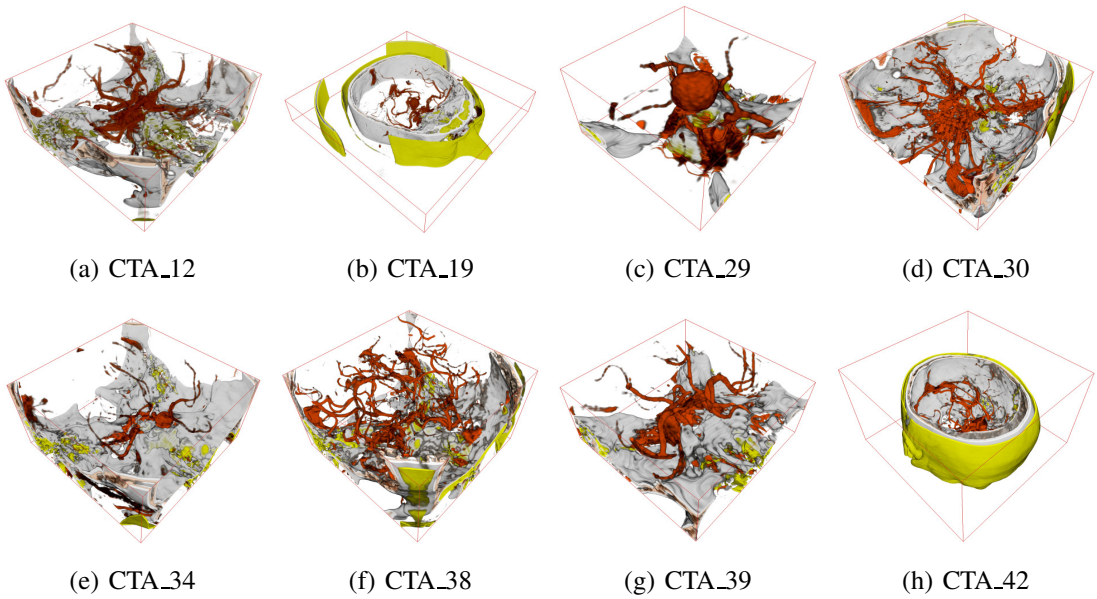
This chapter presents the results of the transfer function design based on semantic models. After outlining general proceedings, CTA and MRI datasets are analyzed and visualized. Furthermore, the results are evaluated including the reference to the stability of the eigenvectors, operating aspects, and the clinical study.

7.1 Principles

As already pointed out in section 5.2, at first a template is necessary which is adapted to all reference datasets. One of the reference datasets is loaded into the system in order to create a template, and as many primitives as needed are added to the primitive editor. The type of primitives chosen for the visualization is dominated by the structures in the dataset. Finally, all primitives are aligned as good as possible to the structures described in the entities in order to achieve the best visual result. After having created the template, it is saved and adapted to all remaining reference datasets. The following aspects have to be taken into consideration in order to create semantic models successfully:

- It is assumed that the template is adapted to the new structures, whereas the number of primitives has been retained unchanged.
- The dimension of input data, which is the sum of the number of parameters, must be the same for each data vector and for each reference dataset.
- As different modifications to the primitives may lead to similar results, it is required to use similar actions to achieve a specific task for all reference datasets.

If the rules above are neglected, the computation model based on PCA will fail. After the template has been adapted to all reference data sets, the transfer functions are split into groups of primitives which belong to the same entity. This technique keeps the primitives organized and allows to perform PCA to each entity separately (see section 6.3). Due to the different scales of the primitives it is



(a) CTA_12 (b) CTA_19 (c) CTA_29 (d) CTA_30

(e) CTA_34 (f) CTA_38 (g) CTA_39 (h) CTA_42

Figure 7.1: The transfer function template for CTA datasets shown in Figure 7.2(a) is adapted to 8 out of 10 reference datasets from the clinical study on intracranial vessels.

necessary to study each entity individually. A rather large modification of a large primitive may have a less significant visual effect on the final rendition than a subtle modification of a small primitive may have. If PCA is performed for the complete parameter vector containing all primitives of all entities, the subtle but important adjustments to smaller primitives can be lost easily because the covariance matrix is dominated by the large variance of other low-level parameters.

Alternatively, PCA can be applied to all input vectors at once. Besides the drawbacks described above, the adjustment of a semantic parameter which adapts the whole transfer function results in hardly predictable effects on the semantic model. The introduction of additional scaling factors for individual components of the parameter vector may be a solution for primitives which differ appreciably for different entities.

The implementation of the techniques described in this diploma thesis has been evaluated by using collections of real patient datasets of two different studies in clinical practice, namely CTA and MRI data. Details of all datasets are available in appendix B. All images in this section are generated by using 2D transfer functions and 12 bit volume data.

7.2 CTA Datasets

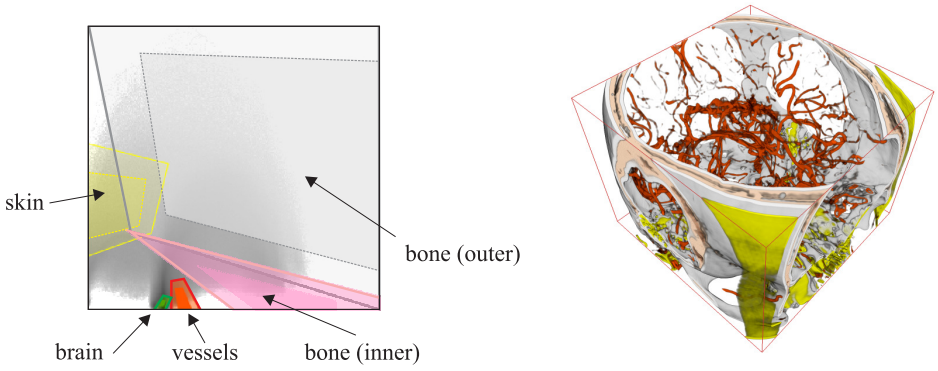
The first application example is about the visualization of CTA data collected within clinical studies at the Department of Neuroradiology at the University of Erlangen-Nuremberg. This collection of data was acquired for the purpose of operation planning concerning the treatment of intracranial

aneurysms. An aneurysm is a dilation or ballooning of a blood vessel by more than 50 percent of the diameter of the vessel. It most commonly occurs in the arteries at the base of the brain. The larger an aneurysm gets, the more likely it is to burst.

Ten different datasets are used as reference data in order to construct the semantic model. Six additional datasets are used for evaluation. The resolution of the slice images is fixed at 512×512 with 12 bits per voxel. The number of slice images for each individual dataset varies between 90 and 260. The experiments have shown that semantic models do not change significantly if different datasets are chosen as a reference set from the collection. The structures found in the 2D histogram considerably vary among the datasets, mainly because of the different fields of view during data acquisition.

7.2.1 Semantic Model

A template for the transfer function is created for the entities *bone structures*, *brain/soft tissue*, *skin/-cavities*, and *blood vessels*. In Figure 7.2(a), the template for CTA datasets is shown, which contains one primitive for the vessels (red), the brain (green), and the skin (yellow), as well as two primitives for the bone (white and pink for the inner and outer bone). The bone structures are represented by two separate primitives in order to improve their visual appearance. All primitives are quadrilaterals due to the fact that for the user the adjustment of the control points is highly flexible and intuitive.

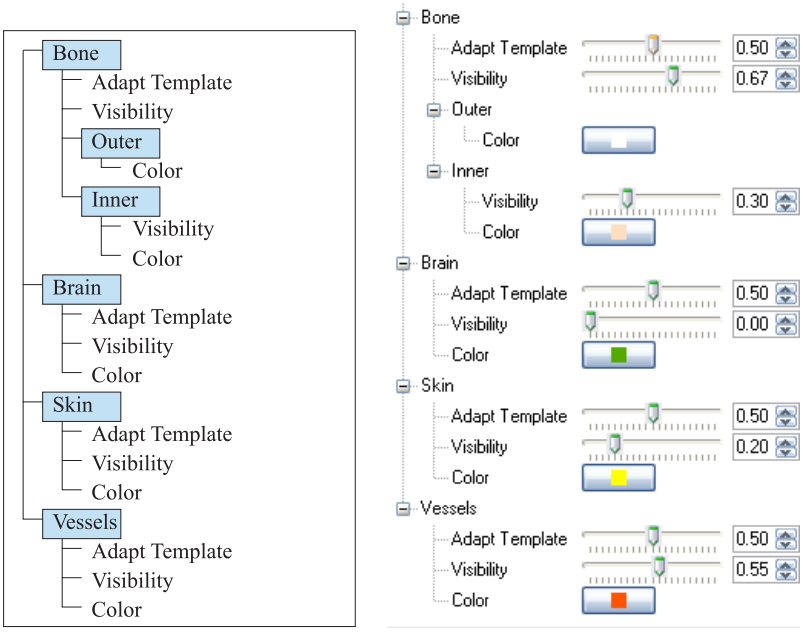


(a) (b) CTA_41

Figure 7.2: Template for CTA datasets. Figure 7.2(a) shows the template for CTA datasets with 5 primitive objects inside the editor. The template is applied to the reference dataset CTA_41, which is shown in Figure 7.2(b).

The result of the template adaption with regard to the reference datasets is shown in Figure 7.1. The visualization of the brain's primitive is neglected in order to get a better view of the structures. The process of performing PCA for each entity creates a semantic model with separate adapt template parameters for each structure of interest. Furthermore, a semantic model consists of additional semantic parameters. Concerning the semantic model for CTA datasets, each entity is completed with

visibility and color parameters. The visibility parameters for vessels, brain and skin simply specify the opacity of the respective primitive. For the bone structures, the visibility parameter can be used to simultaneously fade out both structures (inner and outer bone). The opacity of the inner bone structures can separately be controlled by an additional semantic parameter to enhance the visual appearance. The scheme of the semantic model is illustrated in Figure 7.3(a). Finally, the user interface, which is shown in Figure 7.3(b), is generated automatically with respect to the underlying semantic model.



(a) (b)

Figure 7.3: Scheme and user interface for CTA datasets. Figure 7.3(a) shows a scheme of the entities and associated semantic parameters. The generated user interface is displayed in Figure 7.3(b).

7.2.2 Visualization

Some example images created for datasets which are not part of the reference set are shown in the following (see Figures 7.4 and 7.5). The semantic model for the CTA data, which has already been introduced in the previous section, is applied by adjusting the semantic parameters in the user interface in order to achieve the best results. First, the adapt template parameter is used to stress the structures in oder to visualize the desired entities. Additional semantic parameters are adapted subsequently to improve the visual appearance.

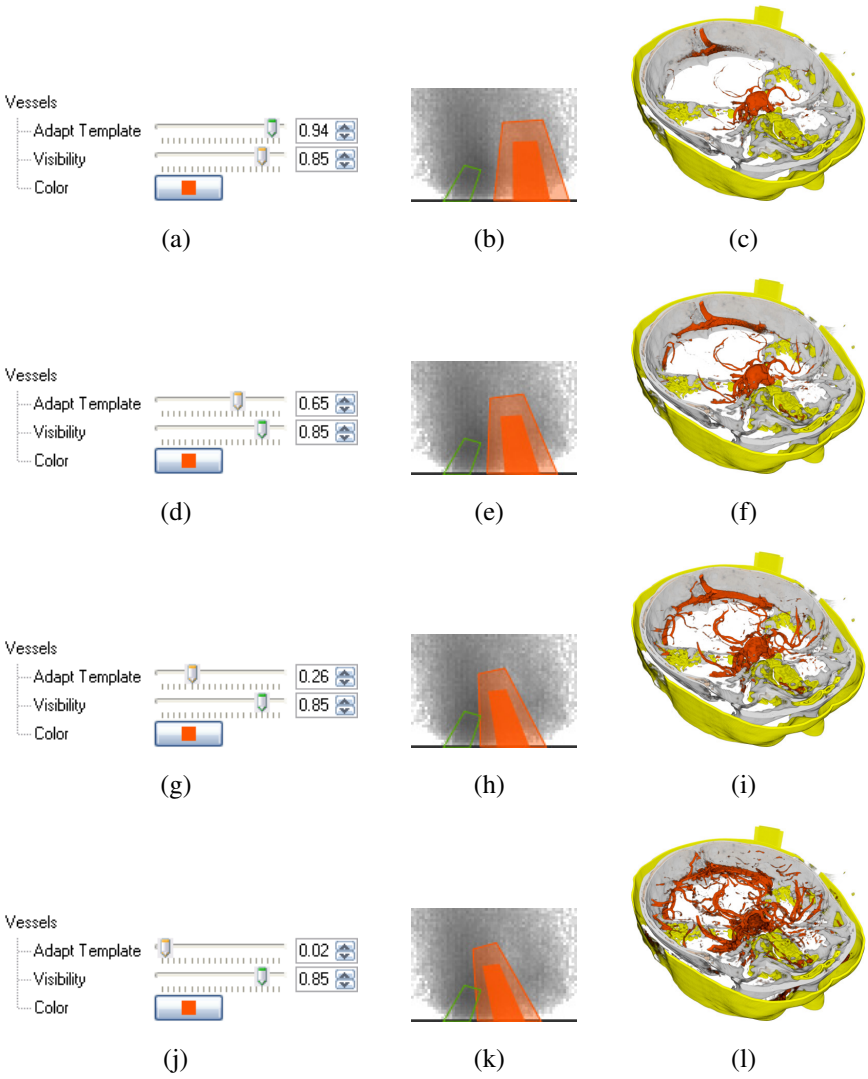
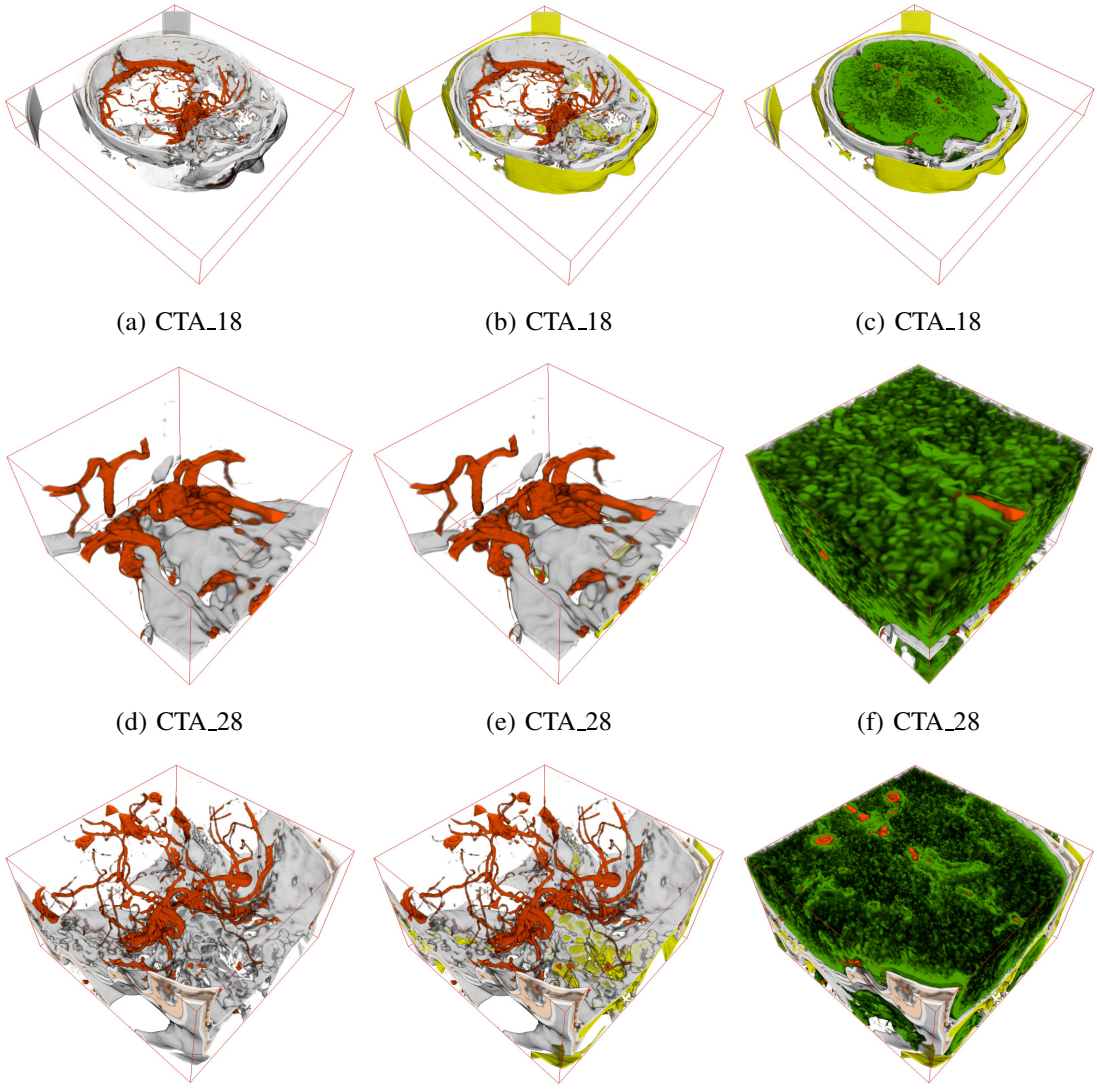


Figure 7.4: An example of a template adaption to dataset CTA_18 which is not part of the reference set. The images have been generated by decrementally adjusting the adapt template parameter, i.e. by moving the slider for blood vessels as shown in the left column. This results in a change of the underlying transfer function's low-level parameters and manipulates the primitive's shape (middle column: red primitive for blood vessels). The rendered images based on the presented transfer functions are displayed in the right column.



(a) CTA_18 (b) CTA_18 (c) CTA_18

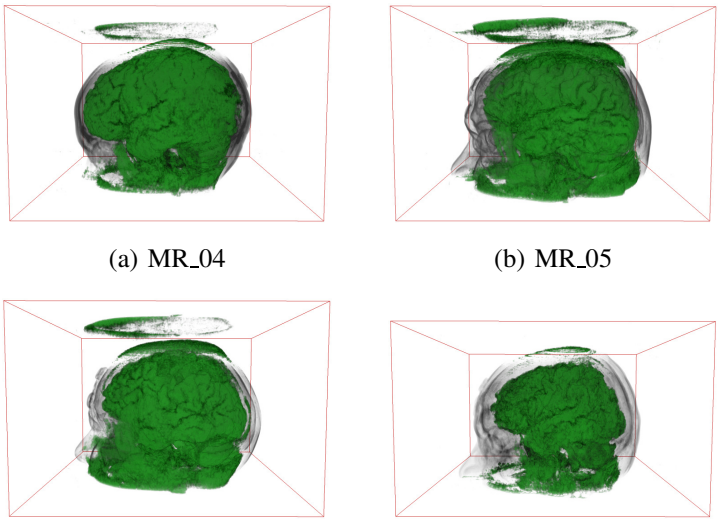
(d) CTA_28 (e) CTA_28 (f) CTA_28

(g) CTA_40 (h) CTA_40 (i) CTA_40

Figure 7.5: The semantic model for CTA applied to three of the datasets which are not contained in the reference set. The left column shows the bone and blood vessels, the middle column adds the skin by increasing the visibility parameter, and the right column visualizes the brain in addition to the other entities by adjusting the visibility parameter.

7.3 MRI Datasets

The second application example deals with the visualization of preoperative MRI data acquired for the planning of tumor resection in the brain. This time, the data were provided by the Department of Neurosurgery of the University of Erlangen-Nuremberg. The term "tumor" is primarily used to denote abnormal growth of tissue. This growth can be either malignant or benign. Malignant tumors are cancerous and have a potential to invade and destroy neighboring tissues and create metastases. Benign tumors do not invade neighboring tissues and do not seed metastases, but they may grow to great size locally. They usually do not return after surgical removal.



(a) MR_04 (b) MR_05

(c) MR_06 (d) MR_07

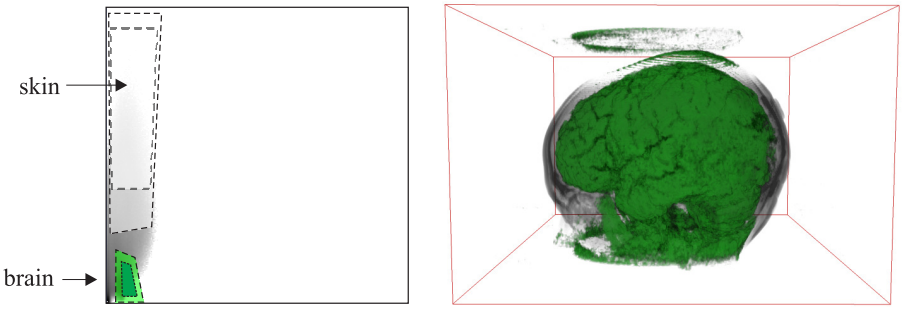
Figure 7.6: The transfer function template for MRI datasets shown in Figure 7.7(a) is adapted to 4 out of 7 reference datasets acquired for the planning of tumor resection in the brain.

Seven out of ten datasets are chosen as reference datasets and the remaining three datasets are used for evaluation. A list containing the reference set is available in appendix B.4. The resolution of the slice images is fixed at 256×256 and the number of slices varies between 100 and 150 with 12 bits per voxel. The image quality is limited due to the short period of time which was allowed for the data acquisition in clinical practice . In consequence, a significant noise is contained in the data.

7.3.1 Semantic Model

The structures visualized by MRI differ from those in CTA, of course, since the former uses strong magnetic fields and non-ionizing radiation in the radio frequency range. Therefore, the semantic model consists of just two entities, namely *brain tissue* and *skin*. The template for MRI datasets is shown in Figure 7.7(a). It contains one primitive for each entity, in particular for the skin (white), and the brain (green). Both of the primitives are quadrilaterals. Please also note that the histograms which

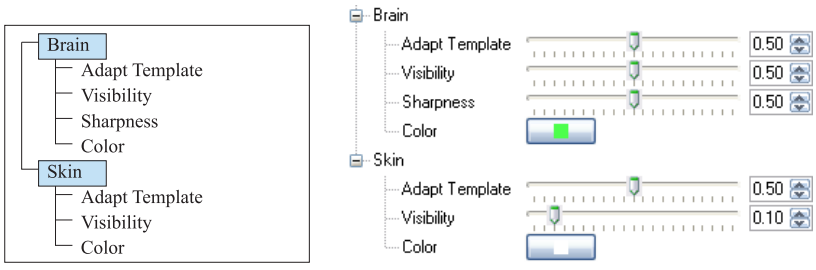
are displayed in the background of the primitive editor have a completely different structure than the ones of the CTA datasets.



(a) (b) MR_04

Figure 7.7: Template for MRI datasets. Figure 7.7(a) shows the template for MRI datasets with 2 primitive objects inside the editor. The template is applied to the reference dataset MR_04, which is shown in Figure 7.7(b).

The result of the process of adapting the template to reference datasets is shown in Figure 7.6. In the next step, PCA is applied to each entity separately in order to build a semantic model. Besides the semantic parameters *adapt template*, *color*, and *visibility*, an additional parameter *sharpness* is introduced. Now the sharpness and fuzziness of the brain's surface can be controlled, which strictly speaking means that the opacity slopes at the primitive's border are modified as described in section 5.4.4. The visibility parameter for the skin is implemented in such a way that it first turns down opacity for low gradients in order to produce a transparent shell of the structure (see section 5.4.1). Figure 7.8(a) shows the scheme of a semantic model for MRI data which leads to the user interface displayed in Figure 7.8(b).



(a) (b)

Figure 7.8: Scheme and user interface for MRI datasets. Figure 7.8(a) shows a scheme of the entities and the associated semantic parameters. The generated user interface is displayed in Figure 7.8(b).

7.3.2 Visualization

Some example images of those datasets not included in the reference set are shown in the following (see Figures 7.9 and 7.10). The semantic model which has been developed in the previous section is applied to the datasets by adjusting the adapt template parameter and by adapting the remaining parameters for the best visual results.

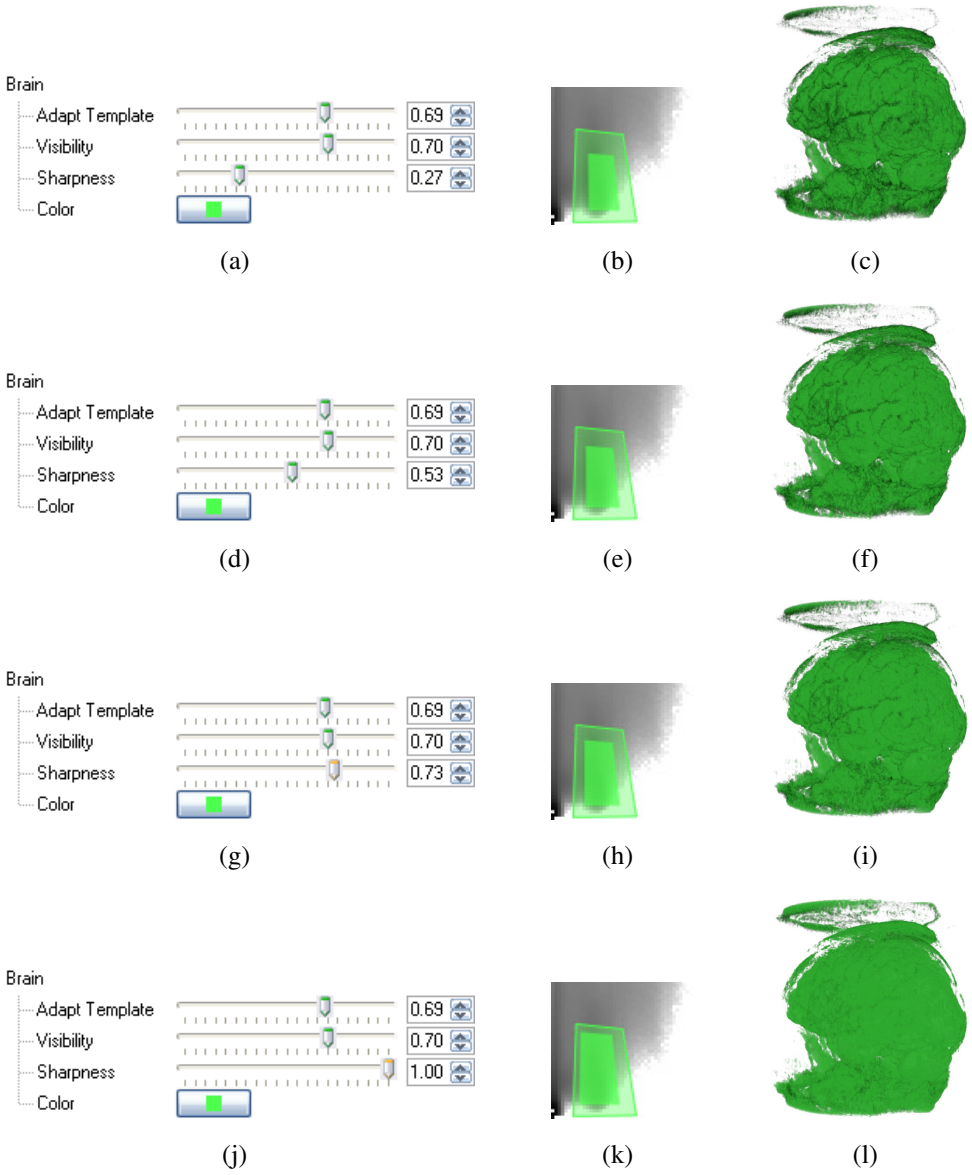
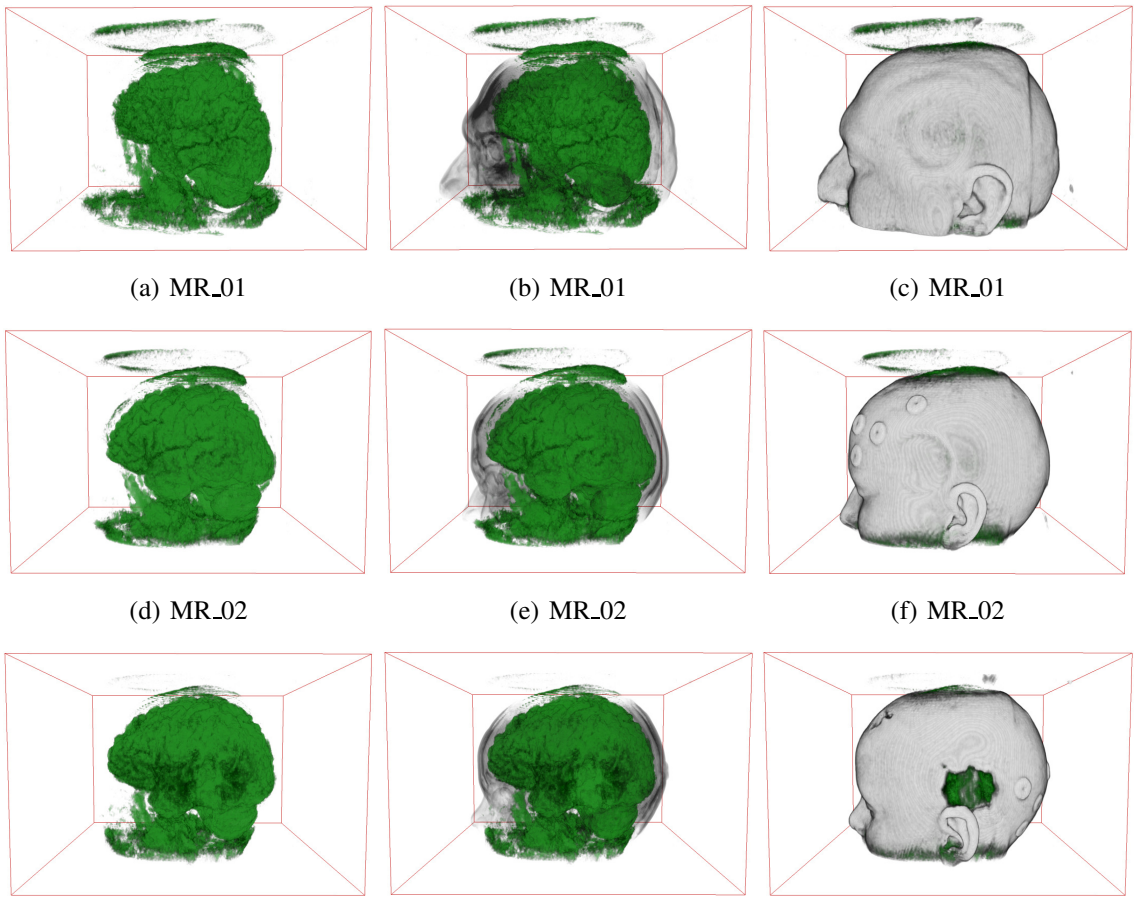


Figure 7.9: An example of template adaption and sharpness parameter to dataset MR_02 which is not part of the reference set. The images have been generated by adjusting the sharpness parameter incrementally, i.e. by moving the slider for the brain as shown in the left column. This results in a change of the underlying transfer function's low-level parameters and manipulates the primitive's opacity slope parameters (middle column: green primitive for the brain). The rendered images which are based on the presented transfer functions are displayed in the right column.



(a) MR_01 (b) MR_01 (c) MR_01

(d) MR_02 (e) MR_02 (f) MR_02

(g) MR_03 (h) MR_03 (i) MR_03

Figure 7.10: The semantic model for MRI applied to three of the datasets which are not contained in the reference set. The left column shows the brain, the middle column adds the skin by increasing its visibility parameter, and the right column visualizes the skin in addition to the other entity by adjusting the visibility parameter to the level of full opacity.

7.4 Evaluation

Within this thesis it is not possible to consider all aspects which would make a convincing evaluation necessary. This is due to the fact that a clinical environment and a large number of real patient data would be necessary to obtain, which would cause an enormous effort. This chapter concludes with a brief evaluation with respect to a technical analysis on the one hand, and to operating aspects on the other hand.

7.4.1 Stability of Eigenvectors

The importance and the stability of each semantic parameter's first eigenvector is investigated to evaluate the quality of PCA, which creates the basis for the adapt template parameter. In this case, a principal component is defined as stable, if the dot product of the normalized axes of different evaluations does not fall below 0.9. The axis spanned by the first principal component is relatively stable for more than twelve CTA reference datasets. Furthermore, the stability significantly varies for each entity and also depends on which datasets are chosen as reference sets. In table 7.1, the differences concerning the importance of each entity are extracted.

Importance of the first Principal Component									
Entity	No. of Datasets:	12	11	10	9	8	7	6	5
Bone		0.95	0.94	0.92	0.93	0.91	0.89	0.87	0.88
Brain		0.99	0.98	0.96	0.96	0.94	0.91	0.89	0.87
Skin		0.90	0.87	0.85	0.84	0.82	0.75	0.74	0.67
Vessels		0.85	0.81	0.76	0.74	0.72	0.70	0.63	0.57

Table 7.1: Importance of the first principal component on a varying number of datasets for different entities.

In the worst case, the importance of the first principal component is 0.85 for 12 reference datasets and 0.57 for 5 datasets. Both values are measured for blood vessels. The pathways of the entities *vessels* and *skin* in a variable number of reference datasets is outlined in the diagram of Figure 7.11.

Similar results are turned out for MRI datasets. The importance of the first principal component is 0.6 in the worst case and 0.8 in the best case due to the limited number of reference data sets available. A higher number of training datasets would have been necessary for a more detailed analysis.

Various tests with 4 different expert-users have shown that the significance of the first principal component is slightly higher if the manual template adaptation for all datasets is performed by the same person instead of by several persons. This shows that the stability of the described approach depends on the designer's template adaptation strategy, which might be considered as a drawback of the proposed technique.

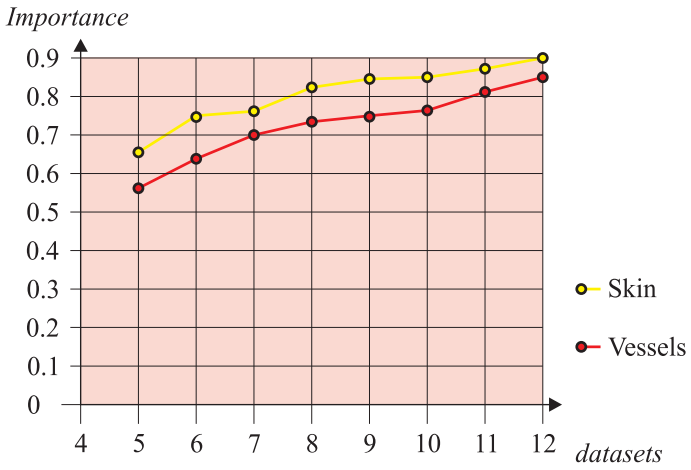


Figure 7.11: The pathways of the entities *vessels* and *skin* according to the importance of the first principal component displayed in table 7.1.

7.4.2 Operating Aspects

A survey with regard to the usability of the semantic editor has been made in the following way: first, ten unexperienced users explore the primitive editor which handles 2D transfer functions. Then the same persons study the semantic editor with regard to the visualization of entities and the tweaking of semantic parameters. In both cases, the users are requested to try to display the bone and the vessels of the dataset CTA_18.

All users agree that the primitive editor is useful in exploring a dataset. The primitives are manipulable very easily and the visual feedback of the rendered image gives an idea of the dataset's structures. When it comes to a specific task, seven users visualize a part of the bone's structures, mostly including parts of the skin. Only one out of ten users is able to isolate the vessels in the primitive's editor. It seems to be impossible for unexperienced users to adjust characteristics of a structure without any additional explanation.

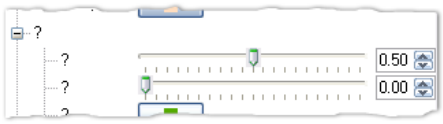


Figure 7.12: An user interface for CTA datasets with unlabeled parameters.

In the next step, the semantic editor is presented to the users. In order to evaluate the accurate meaning of the semantic parameters, the users are first asked to explore the user interface of the CTA model created in section 7.2.1 with unlabeled parameters (see Figure 7.12). In most cases, the users describe the intended visual effects such as visibility and color properly. The adapt template parameter, however, is often described inadequately by the users. Sometimes it is mixed up with the

visibility parameter. After showing and explaining the labels to the users, some of them propose that "*fine tuning*" and "*trimming*" would be more adequate terms instead of the term "adapt template". All users are able to adjust the semantic parameters on their own in order to obtain the desired visual results. The meaning of nested entities is also clear to most persons. In addition to this, a semantic model is presented to the users which is based on only 4 reference datasets. The average importance value of the principal components is less than 0.7. The users' personal visual impression is that this semantic model still reacts the same. This impression might be explained by the considerable amount of redundancy contained in the data.

7.4.3 Clinical Study

A detailed study regarding the usability in clinical environment is still pending. Up to now, the ideas and the approach presented in this thesis have been evaluated by a surgeon and a radiologist so far. Their feedback concerning high-level user interfaces based on the semantic models is very positive. With the help of a developer, the surgeon designed a simple semantic model for CT data. Then he explored the parameter space in the high-level user interface based on semantics. He was extremely enthusiastic about the quality and the tremendous speed with which he achieved the desired visual results. However, he pointed out that the method used to design semantic models for new scenarios is sophisticated and complicated for unexperienced users. A solution might be a wizard which leads through the process of model design.

The radiologist was asked to perform the data visualization for a typical clinical examination based on the semantic model which has been developed in section 7.2.1. He pointed out that he would prefer the simplified user interface instead of the primitive-based editors he is accustomed with. He appreciated the adapt template parameter for manual tweaking, although he emphasized that an automatized initial setup would also be desirable.

Chapter 8

Conclusion

This chapter concludes this thesis with a short summary of the proposed approach concerning transfer functions based on semantic models. The summary is followed by an outlining of limitations and the sketching out of possible solutions and of further aspects for future work.

8.1 Summary

Apart from medical applications, volume rendering is of great importance in natural and computational science, industrial design, engineering, and in many other application areas. And although such technical problems as the evaluation of the underlying physical model, the interactive exploration of volume data as well as the memory management with regard to large datasets have been overcome successfully, the existing solutions are still not used in practice as frequently as you would expect. Many users report difficulties when specifying optical properties for datasets. The manual assignment of transfer functions, which maps properties to values of the datasets, is time consuming and even for experienced users the results are often hardly predictable. Automatic approaches are neither available for all kinds of visualization tasks nor satisfying in many applications. In the field of medical visualization, physicians complain about a lack of clear semantics in this process. For example, they suggest to sharpen the vessels, to fade out the soft tissue, and to improve the contrast between two structures. But even for a visualization expert who is familiar with the underlying transfer function model it is sometimes challenging to obtain the desired results. This is due to an enormous number of degrees of freedom the transfer function model might have. That is why there is still a need for new transfer function designs and corresponding user interfaces.

An overview of related work in the field of volume rendering and transfer function design is presented in chapter 2 of this thesis. Up to now, image-driven techniques and data-driven techniques in transfer function design have not satisfied the claims of modern physicians. Therefore, the transfer functions are adapted to datasets manually with the help of a primitive-based editor. Primitive shapes, such as trapezoids [KKH01] and paraboloids [HST$^+$04], which are introduced in section 4.1, consist of a set of parameters describing the transfer function. But the adjustment of primitives in order

to obtain the desired visual results is still a time consuming process because of the large number of degrees of freedom the primitives may still have. The introduction of semantic parameters as an additional abstraction layer is a solution for this. Each of the semantic parameters s has an influence on the vector of the low-level parameters $\mathbf{p}$ and thus the transfer function is affected. The low-level parameters are mapped back to the primitives which then update their parameters for texture creation. The concept of semantics is borrowed from the field of computer animation where the technical director creates semantic parameters such as *smile* and *frown* and hides the complex setup of low-level parameters from the animator. Similar to the driven keys, which are also part of the computer animation concept, the influence of a semantic parameter is specified by a variable set of keys and a step-by-step linear interpolation (see section 4.3).

Chapter 5 deals with the design of semantic models. The basis of the design is a set of reference datasets for a specific examination purpose. A template is created which consists of one or more primitives for each structure of interest. Then the primitive editor is used to adapt the transfer function template to each individual dataset. In order to create semantic parameters, the set of reference transfer functions is analyzed by using PCA which is capable of finding similarities and which approximates the vectors in a lower-dimensional subspace. The chapter concludes with an insight into the creation of additional semantic parameters such as *visibility*, *color*, *contrast*, and *sharpness*.

Besides the theoretical model of semantics, the visualization framework OpenQVis is presented. This allows to create semantic models for the visualization, processing, and evaluation of volume data. In chapter 6, the main aspects of the framework are described. Traditional 1D and 2D transfer function editors are implemented as well as the transfer function designer and the new semantic model, including an automated user interface design. The presented results of the new transfer function model are based on CTA and MRI datasets. In order to achieve the desired results, the adaption of transfer functions no longer is an inconvenient tweaking of parameters which is only reserved for visualization experts.

8.2 Further Considerations

In general, the design of transfer functions in volume rendering applications is a manual, tedious and time consuming procedure which requires detailed knowledge of spatial structures contained in the dataset. In this diploma thesis, a framework is presented for the implementation of semantic models for the transfer function adjustment, which can be used effectively to hide the complexity of visual parameter assignment from the non-expert user, as visualization experts create the transfer function template for a given clinical scenario.

As a result it can be said that it does not seem necessary to provide a control system for all kinds of modifications to the low-level parameters in order to present a successful user interface. Although the design of a semantic model with a large number of parameters is created easily, it is more helpful for non-expert users if the user interface is restricted to a limited number of essential parameters. The

proposed semantic models and parameters for CTA and MRI datasets turned out to be completely sufficient for creating all desired visual representations. This is possible without a significant loss in flexibility, due to a considerable amount of redundancy in a low-level transfer function model.

Finally, the concept of high-level user interfaces based on semantic models could increase the acceptance of volume rendering in scientific application scenarios. This is due to the simplification of adjusting transfer functions, which is no longer a manual, tedious and time consuming process, even for non-expert users. All in all, the concepts described in this diploma thesis are not restricted to transfer function design. They can be used to provide intuitive user interfaces for different kinds of visualization tasks.

8.3 Limitations and Future Work

Although the semantic model and the generated high-level user interface offer a new and innovative technique of transfer function adjustment, it has to be considered that the proposed approach is limited in some cases.

The creation of a semantic model is bound to an application scenario with a specific examination purpose. That is the reason why two different models are needed for CTA and MRI datasets. A visualization expert has to design the semantic models in a 1D oder multi-dimensional primitive editor, which is still a time consuming process. As it has already been said in section 7.4.1, the significance of the first principal component is slightly higher if the manual template adaptation for all reference datasets is performed by the same person instead of by several persons. Thus, the quality of the semantic model depends to a certain degree on the designer's strategy. In general, a risk of the presented approach is to over-parameterize or under-parameterize the parameter space of the transfer function, as it is always kept under the control of the designer. More importantly, the reference datasets should statistically represent the range of all possible datasets in order to achieve an appropriate approximation of the input data. This can hardly be verified in practice and it is suggested to use as many datasets of a specific type as available.

The handling of the user interface can still be improved by presenting an automatized initial transfer function adjustment. For example, the techniques described in [RSHSG] can be investigated in order to achieve this task in a future version. A sophisticated and elaborated wizard may be useful to lead through the process of creating templates and semantic models. In addition to this, you could imagine to extend the transfer function designer with database features which handle all recorded application scenarios and proper semantic parameters for a faster and simplified semantic model creation. A future challenge might also be the implementation of additional parameters such as edge properties, which are frequently used in non-photo realistic rendering, and material properties for local illumination and translucent rendering.

A further challenge for future work concerns the Gaussian probability distribution of the low-level parameters (see section 5.3). The assumption of the distribution is made with the intention to

motivate to use the PCA, as the distribution is defined by the mean value and the variance. It could be worthwhile to find out if an analysis using higher order statistics such as *Independent Component Analysis* can be used to derive non-linear semantic parameters. However, this technique would require an enormously large number of reference datasets in order to compute reliable results.

Appendix A

Semantic Models

In the following, an XML Document Type Definition (DTD) for transfer function models based on semantics is displayed (see Listings A.1 and A.2). The purpose of a DTD is to define the legal building blocks of the XML document. Thus, the DTD structures the document with a list of elements.

```
<!DOCTYPE Model [
        <!ELEMENT Model (TransferFunctionModel, Semantics)>
        <!ATTLIST Model
                name            CDATA #REQUIRED
                numParameters   CDATA #REQUIRED
        >
        <!ELEMENT TransferFunctionModel (Primitives*)>
        <!ATTLIST TransferFunctionModel
                name      CDATA #REQUIRED
        >
        <!ELEMENT Primitives (Paraboloid2D* | Trapezoid2D* | Quad2D*)>
        <!ELEMENT Paraboloid2D (Parameters)>
        <!ATTLIST Paraboloid2D
                name      CDATA #REQUIRED
        >
        <!ELEMENT Trapezoid2D (Parameters)>
        <!ATTLIST Trapezoid2D
                name      CDATA #REQUIRED
        >
        <!ELEMENT Quad2D (Parameters)>
        <!ATTLIST Quad2D
                name      CDATA #REQUIRED
        >
        <!ELEMENT Parameters (#PCDATA)>
```

Listing A.1: Part 1 of the DTD which defines the structure of the XML file for semantic transfer function models.

```
25
26          <!ELEMENT Semantics (Entity*)>
27          <!ATTLIST Semantics
28                  name        CDATA #REQUIRED
29          >
30          <!ELEMENT Entity (Entity* | ColorParameter* | StandardParameter*)>
31          <!ATTLIST Entity
32                  name        CDATA #REQUIRED
33          >
34          <!ELEMENT ColorParameter (Influence+)>
35          <!ATTLIST ColorParameter
36                  name        CDATA #REQUIRED
37                  type        CDATA #REQUIRED
38                  mode        CDATA #REQUIRED
39                  default     CDATA #IMPLIED
40          >
41          <!ELEMENT StandardParameter (Influence+)>
42          <!ATTLIST StandardParameter
43                  name        CDATA #REQUIRED
44                  type        CDATA #REQUIRED
45                  default     CDATA #REQUIRED
46                  min         CDATA #IMPLIED
47                  max         CDATA #IMPLIED
48          >
49          <!ELEMENT Influence (#PCDATA)>
50          <!ATTLIST Influence
51                  key         CDATA #REQUIRED
52          >
53  ]>
```

Listing A.2: Part 2 of the DTD which defines the structure of the XML file for semantic transfer function models.

Appendix B

Datasets

B.1 CTA

The image data was recorded with a Siemens Somatom Plus 4 spiral-CT scanner. During data acquisition, the non-ionic contrast agent (100ml) was applied in all cases. The delay time was chosen with respect to the circulation time for each individual patient.

Dataset	Modality	Size
CTA_12	CT	512x512x91
CTA_16	CT	512x512x72
CTA_18	CT	512x512x62
CTA_19	CT	512x512x77
CTA_22	CT	512x512x85
CTA_25	CT	512x512x31
CTA_27	CT	512x512x173
CTA_28	CT	512x512x64
CTA_29	CT	512x512x101
CTA_30	CT	512x512x167
CTA_34	CT	512x512x121
CTA_38	CT	512x512x246
CTA_39	CT	512x512x121
CTA_40	CT	512x512x138
CTA_41	CT	512x512x189
CTA_42	CT	512x512x242

Table B.1: An overview of the CTA datasets which are used for the design of a semantic model.

Table B.1 presents an overview of the properties of the datasets that have been used. The design of the semantic model for CTA datasets in section 7.2.1 is based on the reference datasets listed in Table B.2.

	Reference Set
1	CTA_12
2	CTA_19
3	CTA_22
4	CTA_29
5	CTA_30
6	CTA_34
7	CTA_38
8	CTA_39
9	CTA_41
10	CTA_42

Table B.2: An overview of all CTA reference datasets.

B.2 MRI

The preoperative MRI datasets were acquired for the planning of tumor resection in the brain.

Dataset	**Modality**	**Size**
MR_01	MRI	256x256x112
MR_02	MRI	256x256x112
MR_03	MRI	256x256x112
MR_04	MRI	256x256x112
MR_05	MRI	256x256x112
MR_06	MRI	256x256x112
MR_07	MRI	256x256x102
MR_08	MRI	256x256x102
MR_09	MRI	256x256x112
MR_10	MRI	256x256x112

Table B.3: An overview of the MRI datasets which are used for the design of a semantic model.

	Reference Set
1	MR_04
2	MR_05
3	MR_06
4	MR_07
5	MR_08
6	MR_09
7	MR_10

Table B.4: An overview of all MRI reference datasets.

List of Figures

List of Tables

Listings

Bibliography

[BPS97] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In *IEEE Visualization '97 (VIS '97)*, pages 167–174, Washington - Brussels - Tokyo, October 1997. IEEE.

[CCF95] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS*, pages 91–98, 1995.

[EKE01] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading, May 22 2001.

[EWRS+05] K. Engel, D. Weiskopf, C. Rezk-Salama, J. Kniss, and M.Hadwiger. Real-time volume graphics. In *ACM SIGGRAPH Course Notes*. 2005.

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

[FBT98] Shiaofen Fang, Tom Biddlecome, and Mihran Tuceryan. Image-based transfer function design for data exploration in volume visualization. In *IEEE Visualization*, pages 319–326, 1998.

[FvDFH96] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer graphics: Principles and practice in C*. Addison-Wesley, 2nd edition, 1996.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[HHKP96] Taosong He, Lichan Hong, Arie E. Kaufman, and Hanspeter Pfister. Generation of transfer functions with stochastic search techniques. In *IEEE Visualization*, pages 227–234, 1996.

[Hou73] Godfrey N. Hounsfield. Computerized transverse axial scanning (tomography): Part 1. description of system. *British Journal of Radiology*, 46:1016–1022, 1973.

[HST+04] F. Vega Higuera, N. Sauber, B. Tomandl, C. Nimsky, G.Greiner, and P. Hastreiter. Automatic adjustment of bidimensional transfer functions for direct volume visualization of intracranial aneurysms. In *SPIE Medical Imaging*, 2004.

[Kau94] Arie E. Kaufman. Voxels as a computational representation of geometry, August 26 1994.

[KD98] Gordon L. Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *VVS*, pages 79–86, 1998.

[KG99] Andreas H. König and Eduard M. Gröller. Mastering transfer function specification by using volumepro technology. Technical report, March 29 1999.

[Kin99] Gordon Kindlmann. Semi-automatic generation of transfer functions for direct volume rendering. Master's thesis, Cornell University, 1999.

[KKH01] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In Thomas Ertl, Ken Joy, and Amitabh Varshney, editors, *Proceedings of the Conference on Visualization 2001 (VIS-01)*, pages 255–262, Piscataway, NJ, October 21–26 2001. IEEE Computer Society.

[KKH05] Joe Kniss, Gordon Kindlmann, and Charles Hansen. *Multidimensional Transfer Functions for Volume Rendering*. Elsevier, 2005.

[Kni02] Joe Kniss. Interactive volume rendering techniques. Master's thesis, 2002.

[Koc90] Sandeep Kochhar. A prototype system for design automation via the browsing paradigm. In *Graphics Interface '90*, pages 156–166, May 1990.

[KPC93] John K. Kawai, James S. Painter, and Michael F. Cohen. Radioptimization: goal based rendering. In *SIGGRAPH*, pages 147–154. ACM, 1993.

[KPHE02] Joe Kniss, Simon Premoze, Charles D. Hansen, and David S. Ebert. Interactive translucent volume rendering and procedural modeling. In *IEEE Visualization*, 2002.

[KPI+03] Joe Kniss, Simon Premoze, Milan Ikits, Aaron E. Lefohn, Charles Hansen, and Emil Praun. Gaussian transfer functions for multi-field volume visualization. In Greg Turk, Jarke J. van Wijk, and Robert Moorhead II, editors, *IEEE Visualization*, pages 497–504. IEEE Computer Society, 2003.

[KSW+04] Joe Kniss, Jürgen P. Schulze, Uwe Wössner, Peter Winkler, Ulrich Lang, and Charles D. Hansen. Medical applications of multi-field volume rendering and VR techniques. In Oliver Deussen, Charles D. Hansen, Daniel A. Keim, and Dietmar Saupe, editors, *VisSym*, pages 249–254, 350. Eurographics Association, 2004.

[KW03] Jens Krüger and Rüdiger Westermann. Acceleration techniques for GPU-based volume rendering. In Greg Turk, Jarke J. van Wijk, and Robert Moorhead II, editors, *IEEE Visualization*, pages 287–292. IEEE Computer Society, 2003.

[Lev88a] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[Lev88b] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[LL94] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH*, pages 451–458. ACM, 1994.

[MAB+97] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 389–400. ACM SIGGRAPH, Addison Wesley, August 1997.

[Mea85] D. Meagher. Applying solids processing to medical planning. In *Proceedings of NCGS'85*, pages 372–378, 1985.

[Nov93] Kevin Novins. *Towards accurate and efficient volume rendering*. PhD thesis, Cornell University, 1993.

[PFTV92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[RCB05] Kiaran Ritchie, Jake Callery, and Karim Biri. *The Art of Rigging, Volume 1*. Alias conductors program. Cg Toolkit, 2005.

[RGWE03] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, and Thomas Ertl. Smart hardware-accelerated volume rendering. In *In Procceedings of EG/IEEE TCVG Symposium on Visualization VisSym 03*, pages 231–238, 2003.

[RS02] Christof Rezk-Salama. *Volume Rendering Techniques for General Purpose Graphics Hardware (Volumenvisualisierung auf handelsüblicher Grafik-Hardware)*. PhD thesis, University of Erlangen-Nürnberg, 2002.

[RSEB+00] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. In Stephan N. Spencer, editor, *Proceedings of the 2000 SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware (EGGH-00)*, pages 109–118, N. Y., August 21–22 2000. ACM Press.

[RSHSG] C. Rezk-Salama, P. Hastreiter, J. Scherer, and G. Greiner. Automatic adjustment of transfer functions for 3D volume visualization. In B. Girod, G. Greiner, H. Nieman, and H.-P. Seidel, editors, *Proceedings of the 2000 Conference on Vision, Modeling and Visualization (VMV-00)*, pages 357–364.

[Shl05] Jonathon Shlens. A tutorial on principal component analysis. `http://www.snl.salk.edu/~shlens/pub/notes/pca.pdf`, 2005.

[Sim91] Karl Sims. Artificial evolution for computer graphics. *Computer Graphics*, 25(4):319–328, 1991. ACM SIGGRAPH 91 Conference Proceedings, Las Vegas, Nevada, July 1991.

[Sim94] Karl Sims. Evolving virtual creatures. In *SIGGRAPH*, pages 15–22. ACM, 1994.

[SSKE05] Simon Stegmaier, Magnus Strengert, Thomas Klein, and Thomas Ertl. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In Eduard Gröller and Issei Fujishiro, editors, *Eurographics/IEEE VGTC Workshop on Volume Graphics*, pages 187–195, Stony Brook, NY, 2005. Eurographics Association.

[SWB+00] Y. Sato, C.-F. Westin, A. Bhalerao, S. Nakajima, N. Shiraga, S. Tamura, and R. Kikinis. Tissue classification based on 3d local intensity structures for volume rendering. In Hans Hagen, editor, *IEEE Transactions on Visualization and Computer Graphics*, volume 6 (2), pages 160–180. IEEE Computer Society, 2000.

[TL92] S. Todd and W. Lathan. *Evolutionary Art and Computer Graphics*. Academic Press, 1992.

[vdP93] Michiel van de Panne. Sensor-actuator networks. In *SIGGRAPH*, pages 335–342. ACM, 1993.

[WK88] Andrew Witkin and Michael Kass. Spacetime constraints. In *Proceedings of SIGGRAPH 88*, pages 159–168, 1988.

[WVW94] Orion Wilson, Allen Van Gelder, and Jane Wilhelms. Direct volume rendeing via 3D textures. Technical Report UCSC-CRL-94-19, University of Califonia, Santa Cruz, 1994.