

# **Automatically Generated Transfer Function Galleries for High Dimensional Multivariate Data**

Markus Muchitsch



# Automatically Generated Transfer Function Galleries for High Dimensional Multivariate Data

Master's Thesis  
at  
Graz University of Technology

submitted by

**Markus Muchitsch**

Institute for Computer Graphics and Vision (ICG),  
Graz University of Technology  
A-8010 Graz, Austria

1<sup>st</sup> September 2011

Advisor: Univ.-Prof. Dipl.-Ing. Dr. Dieter Schmalstieg  
Co-Advisor: Dipl.-Ing. Dr. Bernhard Kainz





# Automatisch Erstellte Transferfunktions-Galerien für Hochdimensionale Multivariate Daten

Diplomarbeit  
an der  
Technischen Universität Graz

vorgelegt von

**Markus Muchitsch**

Institut für Computergrafik und Vision (ICG),  
Technische Universität Graz  
A-8010 Graz

1. September 2011

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter: Univ.-Prof. Dipl.-Ing. Dr. Dieter Schmalstieg  
Betreuer: Dipl.-Ing. Dr. Bernhard Kainz





## Abstract

This thesis deals with the design of transfer functions for the visualization of volumetric data sets by means of direct volume rendering. Thereby, we focus on the visualization of medical data sets. However, this technique is also used in numerous other scientific disciplines such as astronomy, physics, meteorology, and geology. In this context transfer functions define, which structures of a data set are visible and how they appear. To this, contained voxels are assigned optical properties such as color and opacity depending on their data values (i.e. intensity). A scalar data set is frequently insufficient for an unambiguous classification of different structures, as they are often defined by overlapping value ranges. A better discriminability can be achieved using multivariate data sets in which each voxel is described by multiple parameters. For their visualization multi-dimensional transfer functions are required. The design of simple one-dimensional transfer functions by means of primitive editors already entails a time consuming, error-prone and inefficient trial and error approach. At the creation of multi-dimensional transfer functions this issue drastically intensifies. In any case users require knowledge about the technical background of transfer functions as well as the data set to be visualized. Furthermore, a lot of experience in handling the chosen editor is needed to gain suitable results.

This work presents a system with the primary goal to allow users a simple, intuitive and efficient creation of transfer functions. For this purpose special histogram calculations are applied in order to detect value ranges of structures in the transfer function space. Based on them, initial transfer functions are created. This method spares the tedious trial and error approach of the manual editor. In addition, only minimal knowledge on the data set to be visualized is required. Thumbnails created from the initial transfer functions are arranged in a gallery. For the latter a high amount of clearness as well as a simple and efficient navigation are decisive requirements. Therefore two approved user interface concepts, which are known for their good usability have been selected and adapted to enable efficient transfer function design. By interacting with the thumbnails depicted in the gallery, transfer functions associated to them can be adapted and combined within a reasonably constrained scope. Since interactions are not performed in the transfer function space, knowledge about the impact of transfer functions on the visualization of a data set is not necessary. Even for users experienced in handling primitive editors, this technique entails a significant acceleration of the design work and improvement of the created results. The possibility for further adaptation and combination of transfer functions is important to create visualizations according to the current task.

In order to allow the distinction of structures with overlapping value ranges, the presented system works with multi-dimensional separable transfer functions. Using conventional multi-dimensional transfer functions, a reasonable interaction is only possible with up to three dimensions. In contrast separable transfer functions allow the application of an arbitrary amount of dimensions. These can be adapted separately and are combined according to a certain scheme. The complexity, increasing with the dimensionality of separable transfer functions, is hidden by the user interface of the developed system. Erroneous classifications, potentially occurring at the application of separable transfer functions, are prevented by means of a *color test* performed in the used render system. Weaknesses identified at the color test based evaluation are addressed with a preliminary prototype for improved evaluation of multi-dimensional separable transfer functions.





## Kurzfassung

Diese Diplomarbeit beschäftigt sich mit dem Design von Transferfunktionen zur Visualisierung volumetrischer Datensätze mittels Direct Volume Rendering. Dabei liegt der Fokus auf der Visualisierung medizinischer Datensätze. Diese Technik findet aber auch in zahlreichen anderen wissenschaftlichen Disziplinen wie Astronomie, Physik, Meteorologie, und Geologie Anwendung. Dabei bestimmen Transferfunktionen, welche Strukturen innerhalb eines Datensatzes in welcher Weise sichtbar sind. Das erfolgt, indem sie den darin enthaltenen Abtastpunkten (Voxeln) in Abhängigkeit von ihren Datenwerten (z.B. Intensität) optische Eigenschaften wie Farbe und Opazität zuweisen. Ein skalarer Datensatz ist zur eindeutigen Klassifikation verschiedener Strukturen häufig unzureichend, da diese oftmals zumindest teilweise durch gleiche Wertebereiche beschrieben werden. Eine bessere Unterscheidbarkeit kann durch die Verwendung multivariater Datensätze, in denen jeder Abtastpunkt anhand mehrerer Parameter beschrieben ist, erzielt werden. Zu deren Visualisierung werden multidimensionale Transferfunktionen benötigt. Bereits das Design einfacher Transferfunktionen mittels primitiver Editoren ist mit einem zeitintensiven, fehleranfälligen und ineffizienten Trial and Error Vorgehen verbunden. Diese Problematik verstärkt sich beim Erstellen multidimensionaler Transferfunktionen noch drastisch. In jedem Fall benötigen Benutzer Kenntnisse über die technischen Hintergründe von Transferfunktionen sowie über den zu visualisierenden Datensatz. Zusätzlich ist Übung im Umgang mit dem jeweils verwendeten Editor erforderlich, um brauchbare Ergebnisse erzielen zu können.

Diese Arbeit präsentiert ein System, dessen primäres Ziel es ist, Benutzern ein einfaches, intuitives und effizientes Erstellen von Transferfunktionen zu erlauben. Zu diesem Zweck werden mittels spezieller Histogramm-Berechnungen Wertebereiche von Strukturen im Transferfunktionsraum detektiert und damit initiale Transferfunktionen erstellt. Dadurch wird nicht nur das mühsame Trial and Error Verfahren überflüssig, sondern es sind auch nur mehr minimale Kenntnisse über das zu visualisierende Datenset erforderlich. Die mittels initialer Transferfunktionen erstellten Thumbnails werden in einer Galerie angeordnet. Für diese sind eine hohe Übersichtlichkeit sowie eine einfache und effiziente Navigation entscheidende Anforderungen. Daher wurden zu deren Implementierung zwei bewährte, für ihre Bedienerfreundlichkeit bekannte Benutzeroberflächen-Konzepte herangezogen und für das effiziente Design von Transferfunktionen angepasst. Über Interaktionen mit den in der Galerie dargestellten Thumbnails können die mit ihnen assoziierten Transferfunktionen innerhalb eines sinnvoll eingeschränkten Rahmens adaptiert und kombiniert werden. Weil dabei keine direkte Interaktion im Transferfunktionsraum stattfindet, ist kein unmittelbares Wissen über die Auswirkungen von Transferfunktionen auf die Visualisierung des Datensatzes mehr notwendig. Auch bei geübtem Umgang mit primitiven Editoren bringt diese Technik im Vergleich eine deutliche Beschleunigung der Designarbeit und Verbesserung der erstellten Ergebnisse mit sich. Die Möglichkeit zur weiteren Anpassung und Kombination von Transferfunktionen ist wichtig, um Visualisierungen gemäß der aktuellen Aufgabenstellung zu realisieren.

Um eine Unterscheidung von Strukturen mit überlappenden Wertebereichen zu ermöglichen, arbeitet das System mit multidimensionalen separierbaren Transferfunktionen. Während mit konventionellen multidimensionalen Transferfunktionen eine sinnvolle Interaktion nur mit bis zu drei Dimensionen möglich ist, erlauben separierbare Transferfunktionen die Verwendung einer beliebigen Anzahl von Dimensionen. Diese können getrennt voneinander verändert werden und ihre Verknüpfung erfolgt entsprechend einem gewissen Schema. Die mit der zunehmenden Dimensionalität separierbarer Transferfunktionen ebenfalls verbundene steigende Komplexität wird durch die vom entwickelten System zur Verfügung gestellten Benutzeroberflächen verborgen. Potentielle Falschklassifikationen, die bei der Verwendung von separierbaren Transferfunktionen auftreten können, werden mit Hilfe eines *Farbtests* im verwendeten Render System verhindert. Schwachstellen bei der Auswertung mittels Farbtest werden mit einem provisorischen Prototypen zur verbesserten Evaluierung multidimensionaler separierbarer Transferfunktionen behoben.



## Statutory Declaration

*I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.*

\_\_\_\_\_

Place

\_\_\_\_\_

Date

\_\_\_\_\_

Signature

## Eidesstattliche Erklärung

*Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.*

\_\_\_\_\_

Ort

\_\_\_\_\_

Datum

\_\_\_\_\_

Unterschrift



# Acknowledgements

First of all, I want to acknowledge my supervisor Dieter Schmalstieg and my co-advisor Bernhard Kainz. Thank you, Prof. Schmalstieg, for your feedback, hints for improvement and review of this thesis. Thank you, Bernhard, for your constant motivation, your fast and competent support at any time, and for organizing a funding of this work.

Furthermore, I want to thank my family and my friends for accompanying and supporting me throughout this thesis, my studies and my whole life. At this point I also want to thank you, Lissi, for being the loving person you are and for always being there for me.

Moreover, I would like to thank my closest studying colleagues and friends Felix Nairz, Wolfgang Pürstner, Martin Kandlhofer, Martin Köstinger and Arun Ramkissoon for spending many productive, hard working, and enjoyable hours together.

This work was funded by the European Union in FP7 VPH initiative under contract number 223877 (Project "Impact").



# Contents

<b>Contents</b>	<b>XVI</b>
<b>List of Figures</b>	<b>XIX</b>
<b>Listings</b>	<b>XXI</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Outlook . . . . .	1
1.2 Structure . . . . .	2
1.3 Input Data Sets . . . . .	3
1.4 Direct Volume Rendering . . . . .	7
1.5 Transfer Functions . . . . .	16
<b>2 Related Work</b>	<b>23</b>
2.1 Data Centric Approaches . . . . .	23
2.2 Image Centric Approaches . . . . .	44
2.3 Transfer Function Design Specific User Interfaces . . . . .	47
2.4 Selected General User Interfaces . . . . .	50
2.5 Handling the Drawbacks of Multi-Dimensional Transfer Functions . . . . .	53
<b>3 System Overview and Functionality</b>	<b>57</b>
3.1 Basic Workflow . . . . .	57
3.2 Precalculation Overview . . . . .	58
3.3 Alpha Histogram and Partial Range Histogram . . . . .	59
3.4 Transfer Functions . . . . .	70
3.5 User Interface and Functionality . . . . .	71
3.6 Using Separable Transfer Functions . . . . .	93
<b>4 Implementation</b>	<b>103</b>
4.1 Tools . . . . .	103
4.2 Implementation Structure Overview . . . . .	104
4.3 Alpha Histogram and Partial Range Histogram . . . . .	105
4.4 Transfer Functions . . . . .	107
4.5 Data Mountain Based Exploration Area . . . . .	110
4.6 Cover Flow Based Exploration Area . . . . .	122
4.7 Renderer Integration . . . . .	125
4.8 Visualization of the Combined Transfer Function . . . . .	126
4.9 Storage Area . . . . .	127
4.10 Saving and Loading Data Using XML . . . . .	127
4.11 Using Separable Transfer Functions . . . . .	127

**5 Results 133**  
5.1 User Interface Evaluation . . . . . 133  
5.2 Presentation of Selected Visualization Examples . . . . . 139

**6 Future Work 159**

**7 Conclusion 161**

**Bibliography 169**



# List of Figures

1.1	Data Set Grid Overview . . . . .	4
1.2	Creating Data Set Slices Using Computed Tomography (CT) . . . . .	6
1.3	Trilinear Interpolation of Data Set Samples . . . . .	9
1.4	Post-Classification Direct Volume Rendering (DVR) Pipeline . . . . .	9
1.5	Comparison: Pre-Classification and Post-Classification of Data Set Samples . . . . .	11
1.6	Splatting: Convolution of a Voxel with a Kernel Mask . . . . .	13
1.7	Splatting a Voxel on the Image Plane . . . . .	13
1.8	Splatting: Computing Image Plane Pixel Values from Overlapping 2D Kernels . . . . .	13
1.9	Data Set Transformation into Sheared Object Space . . . . .	14
1.10	Optimized Shear-Warp Volume Rendering . . . . .	15
1.11	Basic Editors for Transfer Function (TF) Design . . . . .	17
1.12	Pattern Recognition Process . . . . .	20
2.1	Arch Representation of Data Set Boundaries . . . . .	24
2.2	Overlapped Arches Resolved in Second Derivative Plot . . . . .	25
2.3	Overlapped Arches and Corresponding Data Set . . . . .	25
2.4	Arch and Low High (LH) Histogram Representation of a Data Set . . . . .	26
2.5	Overlapped Arches Resolved by LH Histogram Representation . . . . .	26
2.6	Noisy Data Set Boundaries in Arch and LH Histogram Representation . . . . .	27
2.7	Biased Data Set Boundaries in Arch and LH Histogram Representation . . . . .	27
2.8	Comparison of Visualizations Based on Standard and Mirrored LH Histogram . . . . .	28
2.9	Comparing Visualizations Resulting from Size and Gradient Magnitude Based TFs . . . . .	30
2.10	Comparing Visualizations Resulting from Size and Conventional Intensity Based TFs . . . . .	31
2.11	Two Ways for Calculating a Voxels Ambient Occlusion . . . . .	31
2.12	The Occlusion Spectrum . . . . .	32
2.13	Calculating a Visibility Histogram . . . . .	33
2.14	Demonstration of the Expressiveness of Visibility Histograms . . . . .	33
2.15	Haidacher's Statistical TF Workflow . . . . .	35
2.16	Extraction of Statistical Properties for Data Set Samples . . . . .	36
2.17	Sample's Inner Sphere and Hull Required for the Extraction of Statistical Properties . . . . .	36
2.18	Dot Representation of the Statistical TF Space . . . . .	37
2.19	Opacity Distribution in the Statistical TF Space . . . . .	37
2.20	Alpha Histogram (AH) Peak Area Definition . . . . .	39
2.21	Weight Range Meaning for Partial Range Histogram (PRH) Creation . . . . .	40

2.22	Cycle of Peak Detection and Initial Histogram Adjustment for PRH Calculation . . . . .	42
2.23	TF Design Processes Based on Stochastic Search Techniques . . . . .	44
2.24	Overview of Marks' Design Gallery . . . . .	46
2.25	Overview of Wu and Qu's Framework for Editing Direct Volume Rendered Images (DVRI)	47
2.26	TF Space Peak Shapes . . . . .	48
2.27	König's User Interface Components for TF Specification . . . . .	49
2.28	Basic Concept of Dual Domain Interaction . . . . .	51
2.29	Data Mountain (DM) Depiction with Bookmark Thumbnails . . . . .	52
2.30	Picture Flow Depiction of a Photograph Album . . . . .	53
2.31	Switching Between Images in CF . . . . .	54
2.32	Comparison: General and Separable 2D TFs . . . . .	56
3.1	Basic System Workflow . . . . .	58
3.2	AH Calculation Process . . . . .	60
3.3	Data Set Splitting . . . . .	61
3.4	Two Definitions of a Histogram Apex and Crease . . . . .	62
3.5	Computation of Initial PRHs . . . . .	65
3.6	Basic Merging of PRHs . . . . .	68
3.7	Extended Merging of PRHs . . . . .	69
3.8	Handling Intersecting Regions in the TF Space . . . . .	72
3.9	DM Based Design Gallery User Interface Overview . . . . .	74
3.10	CF Based Design Gallery User Interface Overview . . . . .	75
3.11	Functionality of the DM Based Exploration Area . . . . .	78
3.12	Calculating a New Z Value of a Thumbnail Dragged on the DM . . . . .	80
3.13	Navigation and Arrangement on the DM . . . . .	81
3.14	Working Process for Using System Generated Variations on the DM . . . . .	82
3.15	Functionality of the CF Based Exploration Area 1 . . . . .	85
3.16	Functionality of the CF Based Exploration Area 2 . . . . .	86
3.17	Functionality of the CF Based Exploration Area 3 . . . . .	88
3.18	User Interface Changes for Separable TFs . . . . .	94
3.19	Setting Up Separable TFs . . . . .	97
3.20	Color Test Benefits and Restrictions at Combining Multi-Dimensional Regions of Interest	98
3.21	Concatenating Volumes in the Renderer . . . . .	99
4.1	DM Scene Graph Overview . . . . .	112
4.2	DM Ground Plane and Thumbnail Scene Graph Structure . . . . .	113
5.1	User Interface Evaluation: Reference Image of the Test Data Set . . . . .	134
5.2	User Interface Evaluation: Objective Evaluation . . . . .	137
5.3	User Interface Evaluation: Subjective Evaluation . . . . .	138
5.4	User Interface Evaluation: Distribution of Best and Worst Method Ratings . . . . .	139
5.5	Dice Example: Settings Comparison . . . . .	141
5.6	Dice Example: Single Default And Detailed Settings Structures . . . . .	142
5.7	Dice Example: Detailed Settings, no AH . . . . .	143

5.8	Dice Example: Detailed Settings Structures, no AH . . . . .	144
5.9	Case2_CT Example - Cranial CT Scan: One-Dimensional Visualizations . . . . .	145
5.10	Case2_CT Example - Cranial CT Scan: Initial Histogram and AH . . . . .	146
5.11	Incisix Example - Cranial CT Scan: One-Dimensional Visualizations . . . . .	146
5.12	Incisix Example - Cranial CT Scan: Initial Histogram and AH . . . . .	147
5.13	Case2_T1_post Example - Cranial Magnetic Resonance Imaging (MRI) Scan: Initial Histogram and AH . . . . .	148
5.14	Case2_T1_post Example - Cranial MRI T1 Scan: One-Dimensional Visualizations . . . . .	149
5.15	Case2_FLAIR Example - Cranial MRI Fluid Attenuated Inversion Recovery (FLAIR) Scan: Initial Histogram and AH . . . . .	151
5.16	Case2_FLAIR Example - Cranial MRI FLAIR Scan: One-Dimensional Visualizations . . . . .	152
5.17	Case2_fmRI_tMAP Example - Cranial Functional Magnetic Resonance Imaging (fMRI) Scan: One-Dimensional Visualizations . . . . .	153
5.18	Case2_fmRI_tMAP Example - Cranial fMRI Scan: Initial Histogram and AH . . . . .	154
5.19	Case2_T1_pre - Case2_brainmask Example: Multi-Dimensional Visualizations . . . . .	155
5.20	Pig42 Example: Multi-Dimensional Visualizations . . . . .	156
5.21	Pat14 Example: Multi-Dimensional Visualizations . . . . .	157



# Listings

3.1	Optimized AH Execution Order . . . . .	63
3.2	Creation of All n-Tuples of n Vectors . . . . .	96
3.3	Creating All Subsets of a Vectors Elements . . . . .	102
4.1	Loading Data Sets Via the Insight Toolkit (ITK) . . . . .	105
4.2	Splitting Data Sets Using ITK . . . . .	106
4.3	Block Histogram Calculation Via ITK . . . . .	107
4.4	Adding Manipulator to Scene Graph . . . . .	118
4.5	Removing Manipulator from Scene Graph . . . . .	118
4.6	Positioning DM Variations . . . . .	121
4.7	Saved Session XML file structure . . . . .	128
4.8	Saved Settings XML File Structure . . . . .	129
4.9	Saved TF XML File Structure . . . . .	129
4.10	Saved Intermediate Precalculation Results XML File Structure . . . . .	130



# Chapter 1

## Introduction

Direct volume rendering (DVR) is a popular and frequently used technique to visualize volumetric data sets in various areas of application. This includes astronomy, physics, meteorology, geology, computational fluid dynamics, automobile surface design and many others. For this work the visualization of medical data sets such as those acquired by computed tomography (CT) or magnetic resonance imaging (MRI) are of special interest. In medicine, DVR visualizations are an increasingly important complement to the traditional slice-by-slice viewing of data sets. The reason for this is, that particularly complex volumetric structures can be conceived much easier by viewing their 3D depictions than by browsing a stack of slices. Moreover, constantly increasing data set resolutions also make slice-by-slice viewing very time consuming. The traditional approach for the visualization of 3D structures of data sets is surface rendering. This however only depicts the surface of structures which is defined by voxels of a certain iso-value. In contrast DVR allows to flexibly utilize all interesting voxels, irrespective of their data value, in order to create a visualization. This is especially useful to depict nested structures, whose spatial relation can be conceived better if the interior of compounding structures is depicted. For a long time surface rendering has still been preferred because of its higher speed on equivalent hardware. However, as a result of the constantly increased hardware speed DVR can also be performed at interactive frame rates nowadays. Therefore, this approach is usually preferred to surface rendering in the meanwhile.

### 1.1 Motivation and Outlook

To create visualizations with DVR, so called transfer functions (TFs) are required to define which structures of a data set are visible and how they appear. This is achieved by mapping data values to optical properties. However, the design of proper TFs resulting in meaningful visualizations is a difficult task. In this context two fundamental interdependent challenges can be identified.

First, TF design has to be simple, quick and intuitive in order to allow users to concentrate on their actual work - grasping information from created visualizations. This is especially difficult to achieve because of the large number of degrees of freedom of TFs and the lack of constraints and guidance of conventional design editors [44]. In a basic editor a polyline is usually used to define a simple 1D TF over the whole TF domain. It can be adjusted by moving control points between its segments. Thereby, each control point has two degrees of freedom. Assuming that every region of interest is defined by approximately three control points, there exists a large number of potential TFs. Since the design process is neither constrained nor guided according to the used data set, a user may only obtain a desired visualization by finding appropriate control points positions using trial and error interaction. Manual design with such an editor is especially problematic, because there is no straight forward correlation between a TF adaptation and a resulting modified data set visualization. Small changes in the TF domain frequently result in strong and unexpected changes of the renderer output. Hence, TF design based on simple editors can be considered as an unintuitive, time consuming and tedious trial and error work.

Second, a clear distinction of data set features has to be achieved. For this purpose multivariate data sets and corresponding multi-dimensional TFs are essential. By describing voxels with multiple data values the separability of structures is enhanced. Unfortunately, the large number of degrees of freedom of multi-dimensional TFs also complicates their design. A possible approach to reduce the complexity of high-dimensional TFs are separable TFs. They represent a single multi-dimensional TF by a number of lower dimensional ones which are combined according to a certain scheme. However, without additional measures they have the potential to falsely classify features.

In order to tackle the described problems this thesis presents a TF design gallery which deploys an automatic data centric approach in combination with an intuitive user interface. Moreover, the used render system provides important functionality to make the over all concept work.

The data centric approach identifies value ranges of features of interest in the TF space using the alpha histogram (AH) and partial range histogram (PRH) method. Based on them initial TFs are created. Each of them defines the visualization of one data set structure. Automatic generation of initial TFs can be considered as a restriction of the TF space to interesting regions and saves users a lot of tedious trial and error work. However, the initial TFs alone are not sufficient. Users must still be able to adapt and combine them according to their visualization task. This can be achieved via the design gallery user interface. It neatly arranges a set of thumbnails, representing the initial TFs. By interacting with them users can compose desired visualizations without any knowledge about the concept of TFs. In order to allow a simple and efficient exploration of the structures depicted on the initial thumbnails, the system provides two user interface versions based on the approved Cover Flow (CF) [7] and Data Mountain (DM) [73] approach. Moreover, the presented framework allows a multi-dimensional classification of data set structures to ensure their clear distinction. This is implemented using separable TFs. In order to eliminate false classifications resulting from them an additional color test in the used renderer followed by a homogeneity test in the design gallery are performed. In order to address weaknesses identified at the color test based TF evaluation, we provide a preliminary prototype for improved evaluation of multi-dimensional separable transfer functions (preliminary nD prototype).

## 1.2 Structure

The remaining parts of Chapter 1 deal with the basics of topics related to TF design on the one hand and with TF design itself on the other hand. The first includes the composition and acquisition of data sets as well as the basics and most important techniques of DVR. Moreover, a comparison of DVR to other methods to view and visualize data sets is discussed. The second includes a formal description of TFs and a discussion about the challenges of their design. Thereby, the shortcomings of one-dimensional TFs and the increasing complexity of designing multi-dimensional ones are elaborated on. Moreover, we discuss how classification, object recognition and segmentation relate to TF design. Chapter 2 introduces a series of (semi-)automatic data and image centric approaches as well as user interfaces to facilitate manual TF design. The latter include TF design specific and selected general user interfaces from other areas of research. Customized versions of the latter are used for the system presented in this work. Furthermore, a number of methods to deal with the difficulties of multi-dimensional TFs are presented. Chapter 3 and 4 present the methods and implementation details of our developed TF design system. This combines a automatic data centric approach to obtain initial TFs with an efficient user interface for task specific adaption and combination of the precalculated structures. Moreover, the system allows a multi-dimensional classification of features based on separable TFs. Thereby, it includes mechanisms to avoid false classifications. In order to render images based on generated TFs a high performance rendering system is deployed. Chapter 5 evaluates the presented system. Therefor, the developed user interfaces have been tested by a number of users. Moreover, a series of visualizations based on one- and multi-dimensional TFs demonstrate the capabilities of the system. Chapter 6 discusses potential areas of improvement. Chapter 7 draws a brief conclusion of the whole work.



## 1.3 Input Data Sets

The current chapter deals with the composition and acquisition of data sets, which can be visualized by DVR techniques. Thereby, Section 1.3.1 presents the basic components of data sets and their potential organization dependent on different areas of application. Section 1.3.2 gives a brief overview of different data set sources and provides a closer description of the most important medical data acquisition methods.

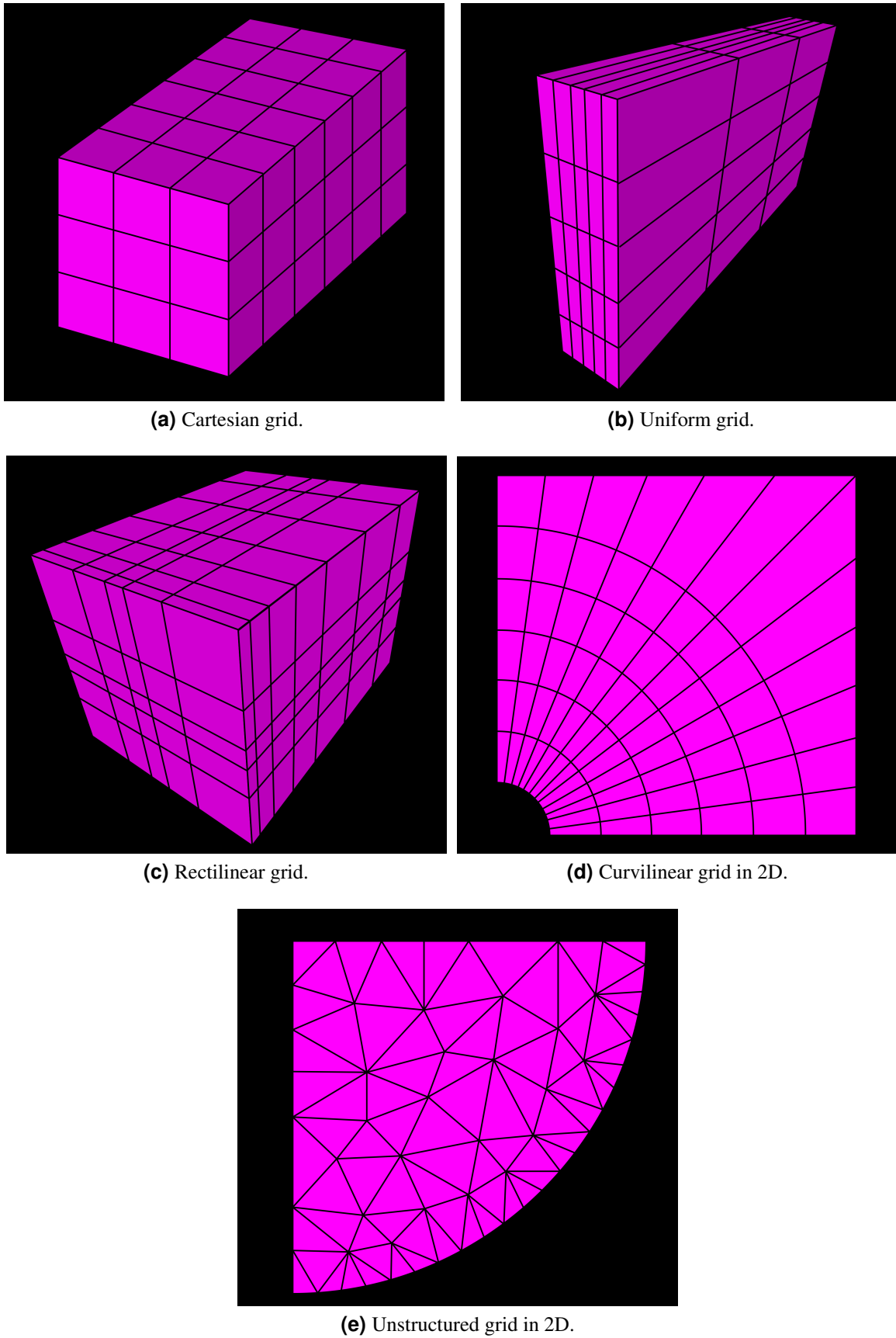
### 1.3.1 Data Set Composition

An input data set or volume is a discretized representation of a continuous 3D field, which holds data values for different positions within the field, given by the coordinates  $(x,y,z)$ . Thereby, each data value is represented by a voxel (volume element), which constitutes the 3D pendant of a pixel in a 2D image. Hence, a voxel can be considered as a quadruple  $(x,y,z,w)$ , at which the first three variables define the position in the volume and the last one stands for the data value(s). If only one data value is associated with each voxel, the volume is called scalar field. If each voxel holds multiple data values, it is called of multivariate data set or vector field. Input data sets can result from many different sources (see Section 1.3.2), and hence the contained data values may constitute a variety of properties and entities. A frequent source of data sets are medical scanning devices such as CT and MRI (see Section 1.3.2). In a CT data set for example, voxels usually contain intensity values which describe a X-rays attenuation throughout its transition of an investigated object. Examples for multivariate data would be fluid flow fields and tensors. Throughout the literature there are two common interpretations of a voxel. In the first case a voxel is considered as a cube with related data value(s), which occupies a small region within the volume. The second interpretation describes voxels as points in the 3D space with related data value(s) and an interpolation scheme to fill the empty spaces between them. In the following the second interpretation is used predominantly. This interpretation also allows a simple explanation of different data set structures, expressing the base frame of the volume as a grid. Thereby, voxels represent points on intersecting grid positions.

In the simplest and used case, voxels are grid points on a uniform cartesian grid. This means that the cells, formed by connecting the single voxels with edges (which are part of the grid) are cubes of uniform size (see Figure 1.1 (a)). Thereby, "cartesian" implies, that the distance of a grid point to all adjacent voxels is the same. For uniform or regular grids the distance between adjacent voxels can be different in every dimension. As a result, uniform cuboid cells are shaped by the grid points (see Figure 1.1 (b)). A further generalization are rectilinear grids, in which the distance between voxels may differ in the same dimension (see Figure 1.1 (c)). Another variation are curvilinear grids as shown in Figure 1.1 (d). In addition to structured grids, grids can also be *unstructured* (see Figure 1.1 (e)). Allowing an arbitrary arrangement of grid points within the volume, is of special interest to represent data sets from application areas like computational fluid dynamics and computed electromagnetic fields. Unstructured grids also deserve special attention as they frequently require different visualization methods. In the following we assume regular grids.

### 1.3.2 Data Acquisition

The visualization of volumetric data sets is relevant for a large number of applications. Data sets typically result from numerical simulations, geometric modelling or some sort of measurement apparatus. This work focuses on data acquired by a measurement apparatus, or more precisely with a scanning device used to obtain medical 3D data. There exist a large number of methods to capture medical images. Some of the most widely used approaches are ultrasound, CT, MRI, single photon emission computed tomography (SPECT), positron emission tomography (PET) as well as functional magnetic resonance imaging (fMRI). All these approaches are briefly presented in the following.



**Figure 1.1:** Overview of different grids describing possible data set structures. Images from [96] and [97].

### 1.3.2.1 Ultrasound

An essential part of each ultrasound system is a probe consisting of one or more acoustic transducers. Each of them forms a source and receiver component at once. The probe sends high-frequency sound waves into a patient's body. There they are partially reflected at different tissue boundaries such as between soft tissue and bones. Reflected sound waves, called echoes, are subsequently detected by the probe. The depth of a found tissue can be determined from an echo's return time to the probe and the speed of sound. The intensity of a returning echo is mapped to a gray-scale value which can be depicted on a screen. Thereby, an echo's intensity expresses the density difference of materials forming the reflecting boundary. [19], [14]

One of the most popular ultrasound methods is named B-mode. In order to obtain a 2D image with this approach a linear array of transducer (phased array) is usually used. The waves of the single transducers are emitted in the same direction. Hence, a phased array produces a parallel sound propagation. By evaluating the returning echoes a rectangular texture slice can be created. Note, that the scanner does not need to be moved to capture the required data. In order to obtain a volumetric data set from ultrasound scans, the phased array must be rotated to emit sound waves to different directions. As a result, a number of scanned 2D areas can be stored together with a corresponding angle. Using this information a 3D volume can be created from the single slices. Alternatively, a matrix of transducers can be used to create a volumetric data set without moving the scanning device. [14]

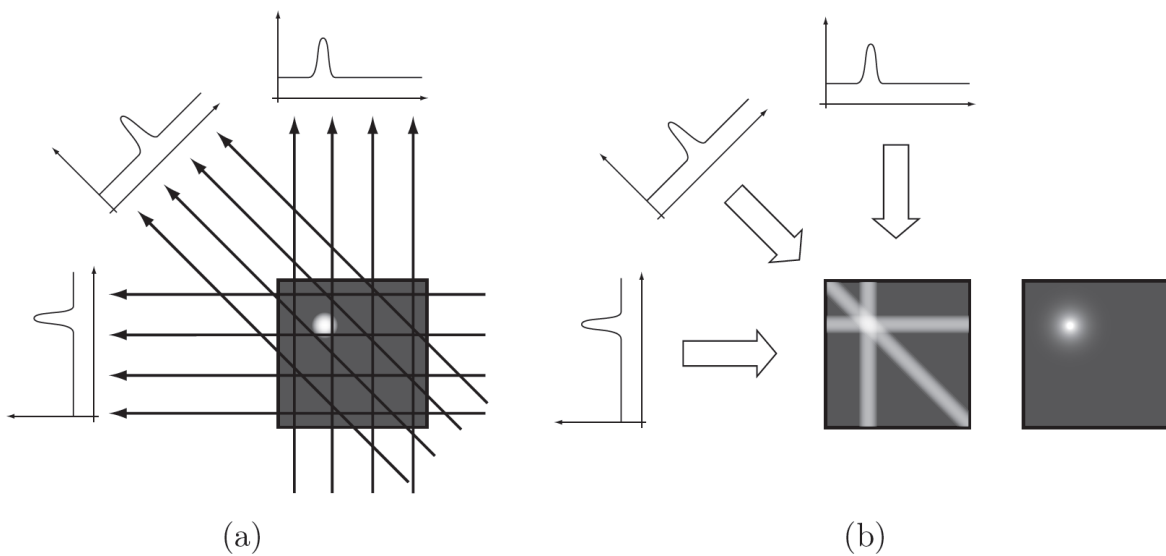
Benefits of ultrasound scanning are the low cost and small size of required equipments. Moreover, it is a fast and non-invasive procedure which does not expose patients to radiation. On the downside ultrasound data sets contain more noise than those resulting from other medical imaging modalities. [82]

### 1.3.2.2 Computed Tomography

CT is a X-ray based scanning method, reconstructing a 3D volume from multiple 2D images of the object of interest (usually a patient) taken from different angles at a number of defined positions. Thereby, the object of interest lies on a moving table which is shifted through a scanning device in order to obtain the raw data for the single slices at different positions. The scanning device consists of a X-ray emitter and a corresponding detector, which are positioned on opposite sides of the investigated object. Performing a full rotation around the object of interest they create X-ray projections from different angles. Thereby, the emitter sends X-rays through the body of the patient, which are attenuated depending on the traversed tissues. Subsequently, they are captured by the detector on the opposite side of the body. All X-ray images resulting from one rotation are projected back into a cross-sectional slice. These slices are then combined to form a volume. See Figure 1.2. [14], [15]

### 1.3.2.3 Emission Computed Tomography

PET and SPECT are nuclear medicine imaging techniques and describe variations of the emission computed tomography (ECT). In comparison to the CT the object of interest is not irradiated by external rays. Instead, depending on the area of application, a short-lived radioactive substance (radionuclide) is injected prior to the examination. For the SPECT, the used radionuclide emits gamma rays, which are detected by one or multiple gamma-cameras rotating around the object of interest. From these projections, the distribution of the radionuclide within the body can be deduced. Similar to the CT multiple projections can be combined to one slice. For the PET, the used radionuclides emit positrons. The interaction of a positron with an electron of the body results in the emission of two photons in opposite directions. These photons are captured by a number of detectors which are arranged around the object investigated. Thereby, coincidences in time and space, registered from two opposing detectors capturing two corresponding photons, are used to deduce the spatial distribution of the injected substance and to compute single slices which form the 3D data set. In comparison to MRI and CT which capture



**Figure 1.2:** (a) shows how different X-ray images are captured from different angles. (b) shows how the captured images are used to create a cross-sectional slice, which forms a part of the final volume. Image from [1].

anatomical structures, PET and SPECT are used to provide information about metabolic activities. PET and SPECT data can also be used together with CT or MRI data in order to combine anatomical and metabolic information. [14], [3]

#### 1.3.2.4 Magnetic Resonance Imaging

Various tissues within a body are differentiated by MRI depending on their nuclear magnetic resonance. Exposing the object of interest to a strong magnetic field, certain atomic (usually hydrogen) nuclei which are magnetic, are aligned according to this field. Due to the additional application of a radio frequency field the alignment of the atomic nuclei can be deflected. As a consequence they start to spin and produce a measurable rotating magnetic field. As soon as the radio frequency field is switched off, the deflection of the rotation starts to decrease until the atomic nuclei are again aligned parallel to the magnetic field. The time required for readjustment is called relaxation time and differs according to chemical connections and the molecular surrounding of the spinning hydrogen nuclei. As a result, different tissues can be identified by measuring different relaxation times. Other than CT, MRI is less suited for bones, since they only contain little water which, is essential for this approach. However, for the depiction of soft tissue, MRI is usually better suited than other imaging modalities, since it provides a greater contrast between different soft tissues. [15], [3]

Functional magnetic resonance imaging is a specialization of the conventional MRI. It allows to depict changes in cerebral perfusion, which result from metabolic activities being closely related to neuronal activities. Thereby, the measurements performed in fMRI are based on the so called Blood Oxygen Level Dependency (BOLD) effect [67]. In order to activate brain areals for identification, probands usually accomplish cognitive tasks during a fMRI scan. In order to provide a spatial context to the brain regions shown by the fMRI data set, it is frequently depicted together with a volume resulting from traditional MRI. [1], [4]

## 1.4 Direct Volume Rendering

DVR is a very popular technique to visualize medical data sets and constitutes the context of TF design which is the actual topic of this thesis. Therefore, this section initially presents the underlying basics of and the major steps performed in DVR including the stage at which TFs are applied. In addition, several well known DVR techniques are introduced. Subsequently, the benefits and weaknesses of this visualization approach are discussed in comparison to other possibilities to view and visualize (medical) data sets.

### 1.4.1 Direct Volume Rendering Basics

A substantial requirement for visualizing a volume using DVR is to know, how light interacts with the medium inside the volume it traverses. This can be described by the equation of light transfer, which assumes that light propagates along straight lines until it encounters a medium in the volume. Therefore, the data set can be imagined as a volume of semi-transparent, light-emitting particles represented by sample points. As soon as a light ray hits a particle, interactions in terms of absorption, emission and scattering occur. Emission adds light energy, absorption reduces it and scattering can do both. As a result, the radiance of a single light ray is influenced by a series of interactions, starting at the point it enters the volume until the point it leaves it. The radiance at the latter point can subsequently be used as a part(pixel value) of the 2D image of the volumetric data. [15], [60]

While absorption and emission are relatively easy to handle, scattering makes the evaluation of the light equation computationally expensive. The reason for this is, that a light ray, hitting a certain particle, may be scattered out in many directions, whereby the radiance of the ray is attenuated. Conversely, light rays which scatter from other particles may be redirected, hit a certain particle and aggravate the radiance of a ray passing it. This results in a number of in-scattering and out-scattering rays at every particle. This large amount of possible interactions and their simulation would make the evaluation of the light equation too costly for most applications. As a result, a number of simplified optical models, which omit certain parts of the original equation, have been introduced. [15]

The most prevalent type is the so called "Emission-Absorption Model", which allows particles to emit and absorb light, but omits scattering and indirect illumination effects. It offers a good trade-off between generality and computational efficiency. For this model the following equation, called the volume-rendering integral, calculates the radiance of a ray traversing the volume at the point leaving it:

$$I(D) = I_0 e^{-\int_{s_0}^D k(t) dt} + \int_{s_0}^D q(s) e^{-\int_s^D k(t) dt} ds \quad (1.1)$$

Thereby,  $s$  describes the position along the ray,  $s = s_0$  and  $s = D$  denote the positions at which the ray enters and leaves the volume, respectively.  $I_0$  is the initial radiance of the ray at  $s_0$  and  $I(D)$  the radiance of the ray at position  $D$ , where it leaves the volume after all interactions took place. Moreover,  $k(t)$  and  $q(s)$  represent the absorption and emission coefficient. The first term of the equation calculates how the initial radiance  $I_0$  of a ray at  $s_0$  is attenuated throughout the transition of the volume. The result stands for the contribution of  $I_0$  to the radiance  $I(D)$  which leaves the volume at  $D$ . The second term calculates the combined emission contributions of the single particles along the ray to its final radiance. Thereby, the emission of each particle is attenuated by the remaining absorption coefficients along the ray, from location  $s$  of this particle to the point  $D$  where the ray leaves the volume. [15]

Originally the just described volume-rendering integral does not include scattering effects, because they are computationally expensive. However, greater realism can be achieved by extending the emission coefficient of the previous equation by a local illumination component. Thereby, it is assumed that light from an external source directly hits each point in the volume, without any previous interaction

with intermediate matter. This approach is called single scattering of external light. Local illumination models like Phong or Blinn-Phong are usually applied for its approximation. Therefore, an equivalent of the normal vector for surfaces is required. In DVR the gradient of the scalar field is used for this purpose. The benefit of this extension lies in the improved visualizations that can be achieved at unchanged computational complexity. [15]

Since it is not generally possible to evaluate the volume-rendering integral analytically, it must be numerically approximated. A common approximation of the volume-rendering integral is a Riemann sum, by which the integration is split into  $n$  discrete equidistant intervals (intervals of different length are possible as well but are not considered in the following):

$$I(D) = \sum_{i=0}^n c_i \prod_{j=i+1}^n T_j \quad (1.2)$$

Thereby,  $T_j$  and  $c_i$  describe the transparency and color contribution of the  $i^{\text{th}}$  segment, which reaches from sample position  $s_{j-1}$  to  $s_j$ , respectively. They are approximated by  $T_i \approx e^{-k(s_i)\Delta x}$  and  $c_i \approx q(s_i)\Delta x$ , where  $\Delta x = (D - s_0)/n$  describes the length of the intervals. Moreover,  $c_0 = I(s_0)$  holds, which means that  $c_0$  stands for the radiance (color) of the ray at the entry point of the volume. [15]

In practice the discretized volume-rendering integral is computed by iteratively evaluating the input contributions along the ray. This can either be done by a front-to-back or back-to-front compositing scheme. In the first case a viewing ray is traversed from the camera position towards the volume. Thereby, the following equations must be evaluated at each iteration along the ray:

$$C'_i = C'_{i-1} + (1 - A'_{i-1})C_i, \text{ with } : C'_0 = 0 \quad (1.3)$$

$$A'_i = A'_{i-1} + (1 - A'_{i-1})A_i, \text{ with } : A'_0 = 0 \quad (1.4)$$

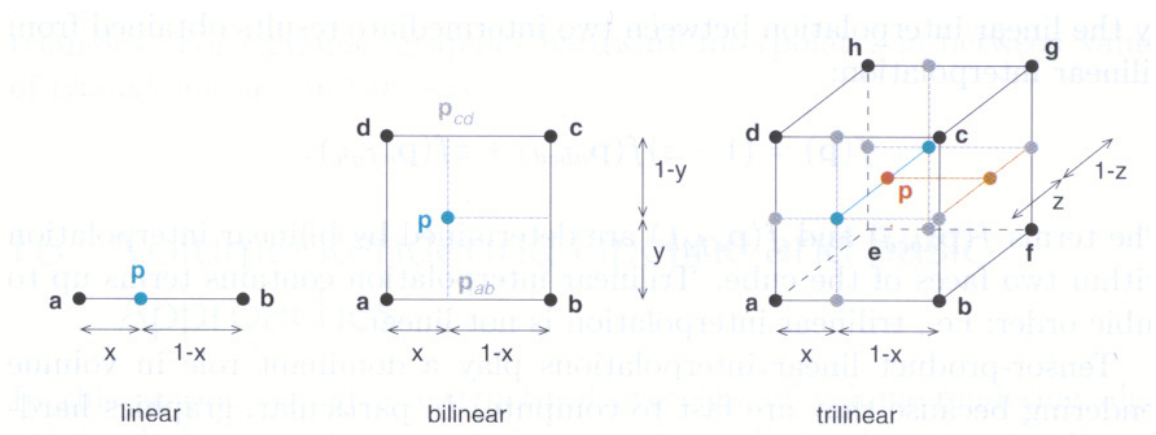
Thereby,  $C'_i$ ,  $A'_i$  and  $C'_{i-1}$ ,  $A'_{i-1}$  are the accumulated color and opacity at the current and previous position  $i$  and  $i - 1$ , respectively.  $C_i$  and  $A_i$  represent the color and opacity of the volume at the current location  $i$ . Thereby, it should be noted, that opacity weighted (=premultiplied) colors are used throughout these compositing operations. This is important, since separate interpolation of color and opacity may result in artifacts. It also means that the colors  $C'_i$  always denote a quadruple of  $RGB\alpha$  (=red, green, red, alpha) values.

For the second type of compositing scheme the viewing ray is traversed in the opposite direction. Here the final  $RGB\alpha$  value is computed by evaluating the following equation at each step along the ray:

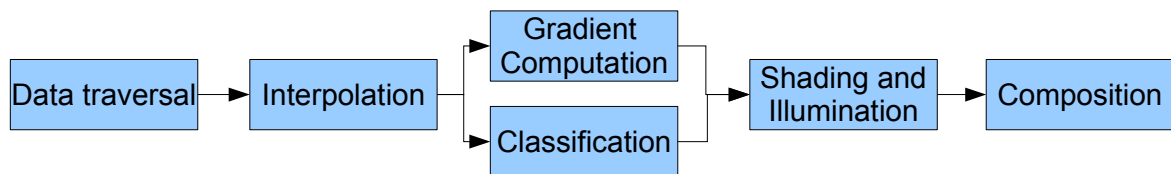
$$C'_i = C_i + (1 - A_i)C'_{i+1}, \text{ with } : C'_n = 0 \quad (1.5)$$

The only new expression in this equation is  $C'_{i+1}$ , which describes the accumulated color at the previous location along the ray. Back-to-front compositing does not require an iterative update of the accumulated opacity, since the accumulation of the opacity is constantly integrated in each update of the accumulated color. [20]

When considering the evaluation of a ray, one may notice that the samples which subdivide the single rays, are usually not at the same position as voxels on the grid. As a result, the continuous volume must be "reconstructed" in order to obtain data values of the samples along the ray. A successful reconstruction in general depends on two major factors: First, the sampling frequency used to discretize the original volume must be high enough to comply the Nyquist theorem. This allows a later reconstruction without loss of information. Secondly, the kind of filter which is used to interpolate the ray samples from the surrounding voxels plays an important role. A popular filter for uniform grids in volume rendering is the so called tensor-product reconstruction filter, which corresponds to trilinear interpolation. Using this filter, the sample value can be computed very fast as a sequence of linear interpolations (see Figure 1.3).



**Figure 1.3:** Trilinear interpolation consists of a sequence of linear interpolations. Image from [15].



**Figure 1.4:** A post-classification volume rendering pipeline.

By now a number of filters exist, which perform better reconstructions for real-time volume rendering, such as the cubic B-spline filter. [15]

**Note:** In the context of the evaluation of an optical model it is important to know that the impact of a particle (= reconstructed sample on a ray) on an approaching light ray primarily depends on the particles optical properties. In the discussed simplified model, the optical properties of a particle are given in terms of an emission and absorption coefficient, which are represented by a particles color and opacity, respectively. The assignment of color and opacity is usually performed by a TF which maps from one or many data values, describing voxels in a volume, to corresponding colors and opacities. As a result, TFs have a major impact on the images generated by a rendering system.

### 1.4.2 The Volume Rendering Pipeline

It is possible to extract several stages from the descriptions in section 1.4.1 which are part of most DVR techniques and are usually combined to form a pipeline for these approaches. A short outline over a possible sequence of these components according to [15] is presented in the following and Figure 1.4.

**Data traversal:** As one of the first steps sample points are defined within the volume. They are used to discretize the volume-rendering integral. Their position is usually not at grid points of the input data set.

**Interpolation:** This step corresponds to the reconstruction of the sample points, defined in the data traversal stage, from surrounding grid points provided by the input data set. A frequently used technique, which provides a good speed-quality tradeoff is trilinear interpolation. However, Engel et al. [15] also present several other, more accurate approaches. Interpolation is performed for each sample point.

**Gradient Computation:** The gradient depicts the amount and direction of change within a certain small region. Since the direction of change is usually perpendicular to a boundary depicted in a volume

it can be used for local illumination calculations, similar to a surface normal when illuminating a surface. The gradient is also computed for each sample.

**Classification:** This step maps data values contained in the volume such as sample intensities to optical properties. These are required as a part of the evaluation of the optical model applied. The mapping is based on TFs which result from (semi-)automatic precalculations or are user defined. The typical optical properties which form a TFs output are color and opacity. In the optical model described in Section 1.4.1 color and opacity are described by the emission and absorption coefficient, respectively. Due to this stage the volume is classified/segmented into different perceptual regions.

**Shading and Illumination:** This component describes an extension of the emission term in the volume rendering integral. By performing a local illumination, the depiction of samples can be significantly enhanced without considerable increase of computational complexity. This is frequently done with single scattering which makes use of the computed sample gradients.

**Composition:** Throughout this stage the contributions of corresponding sample points in the volume are combined to form pixel values in the output image. Therefore, the discretized volume rendering integral is iteratively evaluated by either the front-to-back or back-to-front order.

It is important to note that the order of these components can vary to a certain degree throughout different DVR approaches as well as within different implementations of them. A typical example is the sequence in which interpolation and classification occur. If classification is performed first (pre-classification) voxel data values (i.e. intensities) are mapped to colors and opacities. Subsequently, color and opacity of a sample point is interpolated directly from the colors and opacities of the surrounding voxels. Conversely, if classification is performed afterwards (post-classification), the data value of a sample is interpolated from the data values of surrounding voxels. Then the mapping in the classification stage is performed directly from the already interpolated data value at a sample position to the color and opacity at this point. Varying results that may occur from these different arrangements can be seen in Figure 1.5.

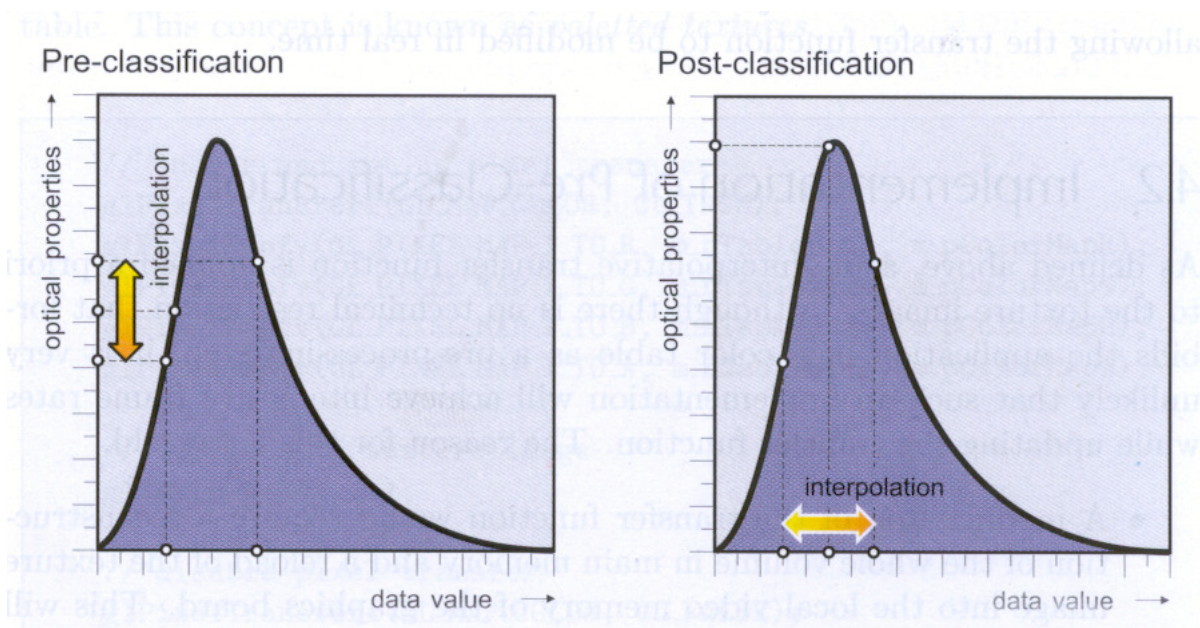
### 1.4.3 Direct Volume Rendering Techniques

According to the order of data traversal DVR techniques are categorized into two main groups: image order and object order techniques.

Image order or backward mapping algorithms traverse the volume starting from the single pixels in the 2D image plane. For each pixel a number of contributing voxels is identified. Based on them the final color and opacity of each pixel is determined. Since calculations have to be executed for every pixel, the performance of this algorithm strongly depends on the resolution of the final image. Image order techniques are generally considered to deliver images of higher quality.

On the other hand, object order or forward mapping methods traverse the voxels of the volume according to an organized scheme and project them to corresponding pixels in the image plane. Here the performance is related to the complexity of the objects. In general object order approaches are considered to be faster. Since they traverse voxels in storage order data coherences can be exploited. They could easily be made parallel on computers with limited local memory, since only a small area around a voxel is required for processing its impact on a rendering. Object order algorithms are especially efficient for sparse volumes. Since voxels which do not contribute to the rendering do not need to be processed, they have significantly reduced computational costs. For image order techniques on the other hand, the contributing voxels which must be accessed for one pixel are spread throughout the volume. Hence, many of the object order optimizations are not possible. Due to the new possibilities of GPU processing this no big problem anymore.





**Figure 1.5:** Pre-classification interpolates optical properties directly from the already classified voxels. Post-classification interpolates a samples data value from the surrounding voxels and classifies the resulting value. Image from [15].

The difference between backward and forward mapping methods can be considered in the way they perform reconstruction. While the first interpolates sample values from adjacent voxels, the latter obtains reconstructed values from the "energy" of voxels spread into space [95]. This difference can be best seen on the example of ray casting and splatting.

In the following three of the most popular DVR techniques, including ray casting, splatting and shear-warp factorization, are presented.

### 1.4.3.1 Ray Casting

Ray casting is the most widely used image order technique for DVR and is well known for the high quality images it creates as well as for the computational burden it causes. Today Levoy's paper [50] is seen as the starting point for ray casting, although Blinn [2] and Kajiya [30] already presented similar approaches before. Raycasting can be performed for perspective and orthographic projections and uses a front-to-back compositing scheme.

The basic cycle of ray casting is described in the following. Since it provides the most direct implementation of the volume-rendering integral the current explanations are very similar to descriptions of the previous sections. Through each pixel of the image on which the volume should be displayed, one ray (or multiple rays if supersampling is performed) is shot into the volume from a camera behind the plane. Subsequently, a number of sampling points are determined for each ray. Therefore, the ray is usually partitioned into several intervals of uniform length, starting from the first intersection point of the ray with the volume and finishing at the point at which the ray exits the volume. Alternatively, a ray can also be sampled at points, where the ray enters and leaves objects within the volume. The ray samples are the points which partition the intervals. As soon as the sample points are located, their colors and opacities are determined, which is usually done by trilinear interpolation. Depending on the used order of the rendering pipeline (pre- or post classification) the color and opacity of a sample is either determined directly from the interpolated intensity value at the sample position or interpolated from the colors and opacities of the adjacent voxels. Subsequently, the viewing rays are traversed, starting from the image pixels, into the volume direction. Thereby, the optical properties of the sample points are composed into

a single pixel value on the output image based on a front-to-back compositing scheme.

Since ray casting is computationally costly a series of optimization techniques have been developed. For example, *Early Ray Termination* means that the traversal of a ray is already stopped as soon as the accumulated opacity exceeds a certain limit. This approximates a value close to one, which means that materials behind the current one are almost or entirely occluded. Hence, they do not contribute to the composited pixel value of the output image and a further evaluation of the ray is not necessary. *Empty space skipping* is a technique which is especially suited to accelerate ray casting of mainly transparent volumes. Thereby, a separate data structure, often an octree, allows to easily find completely or almost transparent regions within a data set. Each node in the octree hierarchy holds the minimum and maximum data value (i.e. intensity) of all underlying nodes. Together with the TF, nodes which represent empty areas can be identified. Data within such regions does subsequently not need to be obeyed. The octree must be recomputed whenever the TF is changed. Further optimization techniques can be found in [15].

However, even with techniques for acceleration the use of ray casting for interactive applications has been problematic for a long time. Due to the emergence of GPU ray casting this issue seems to be less and less of concern. In general GPU ray casting provides two major benefits: On the one hand the overall traversal time for all viewing rays can be significantly reduced by assigning them to parallel GPU processing pipelines which run concurrently. On the other hand the volume data can be accessed faster when stored in the graphic cards memory.

A powerful API, which greatly enhances the flexibility in GPU programming is NVIDIA's Compute Unified Device Architecture (CUDA). The benefits provided by this architecture are for example exploited in a high-performance ray casting system introduced by Kainz et al. [29] which is also used for the system presented in Chapter 3 and 4.

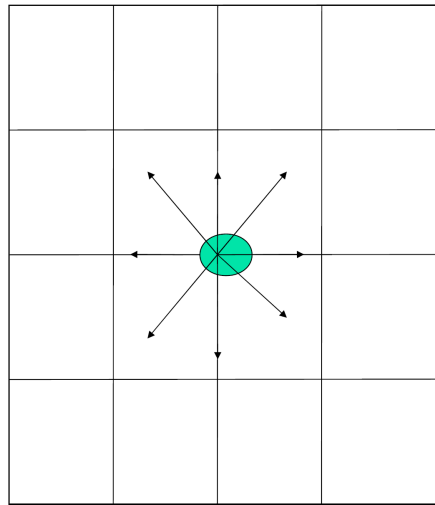
### 1.4.3.2 Splatting

Splatting is a classical object order volume rendering technique. It was initially introduced and by Westover [94], who also released an enhanced version [95] later on. The algorithm works for both, perspective and orthographic projections. The original implementation is suited for regular grids, however Engel et al. [15] state that it may be used on an unstructured data set as well.

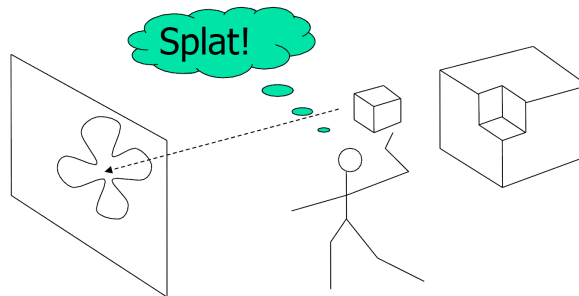
The basic approach is to convolve each voxel in the volume with a 3D reconstruction kernel. Frequently a 3D Gaussian kernel is used. In this context convolution can be understood as the spreading out of the input value to many output values. In other words the data value of the voxel to which a kernel is applied influences each value of the kernel mask (see Figure 1.6). The resulting 3D image is subsequently splatted onto the image plane. This corresponds to the integration of a 3D to a 2D kernel and its projection to the image plane. The emerging 2D projection, which spans over a certain region of image plane pixels, is called footprint. This step is frequently compared to a snowball which spreads out when it hits a wall (see Figure 1.7). Accordingly, the spreading of a projected voxel is represented by the corresponding 2D kernel on the image plane. Depending on the kind of kernel, the footprint has a certain impact on each covered pixel. After the kernel for each voxel has been projected to the image plane, each output pixel may be covered by a series of footprints (see Figure 1.8). By integrating the contributions of all footprints covering a pixel its final value is determined. The final image subsequently consists of all accumulated pixel values.

In practice the chosen kernel can be integrated from 3D to 2D in advance and subsequently convolved with every voxel, since integration and convolution are linear operations. This approach yields the benefit that the 3D kernel must only be integrated once for every view.

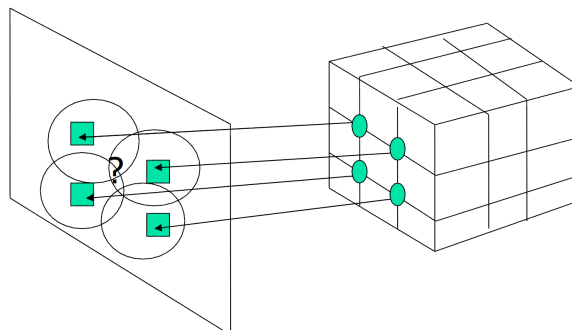
Generally, splatting is considered to be a very fast algorithm. This is especially true for sparse volumes since only non-transparent voxels which affect the output image must be considered. Transparent ones are simply not splatted. In addition, splatting makes it possible to flexibly use various reconstruction kernels without additional computational costs [25]. Another benefit is, that it can be done in parallel,



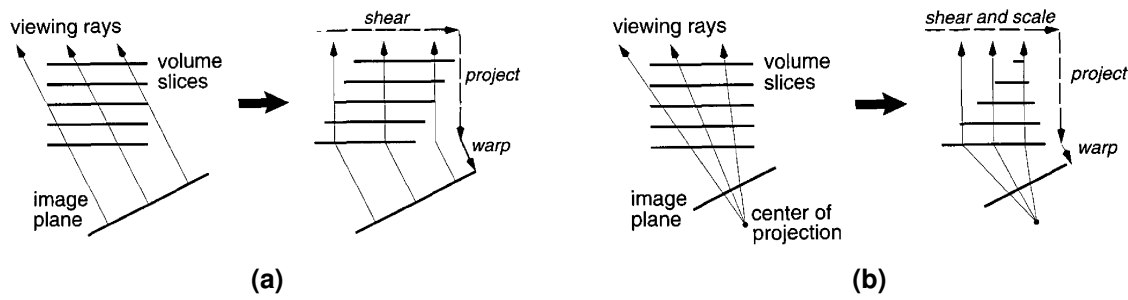
**Figure 1.6:** The convolution of a voxel with a kernel mask affects each value in the mask. Image from [80].



**Figure 1.7:** The voxel is splatted on the image plane. Image from [80].



**Figure 1.8:** The green points represent the centers of the 2D reconstruction kernels on the image plain, represented by projected voxels. Between them there exist a number of pixels which are covered by multiple kernels. Their values are computed from the contributions of all overlapping kernels. Image from [80].



**Figure 1.9:** Data set transformation into sheared object space for parallel and perspective projections. (a) For parallel projections the transformation of a volume to the sheared object space is performed by translating each slice. (b) For perspective projections the transformation of a volume to the sheared object space is performed by translating and scaling each slice. Images from [49].

by subdividing the volume and applying the algorithm to separate parts [95].

Splatting did also experience a series of improvements since its initial implementation by Westover [95]. For example, Müller and Crawfis [61] deploy image-aligned sheets in order to avoid the occurrence of artefacts when using sheets aligned to major axes. However, the higher accuracy achieved entails increased computational costs as well. Moreover, Müller et al. [62] introduced an implementation which allows pre-classification and shading. A more detailed overview of different developments in splatting can be found in [28] and [59].

### 1.4.3.3 Shear-Warp Volume Rendering

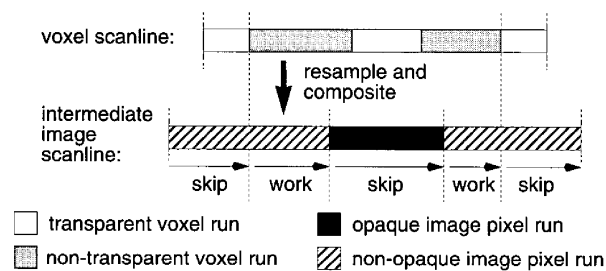
Shear-warp volume rendering is actually another object-order algorithm. However, the implementation of Lacroute and Levoy [49], which is discussed in the following is also frequently described as a hybrid approach since it aims to unify the efficiency of object-order algorithms with the image quality of image-order algorithms. Lacroute and Levoy's approach assumes a rectilinear grid and can be applied for parallel as well as perspective projections as opposed to an earlier implementation of Cameron and Unrill [5]. For a better understanding it is helpful to imagine the data set as a stack of slices which is aligned along one of the main axes.

Ray casting requires a lot of time for computations to trace all the viewing rays throughout the volume as well as to interpolate their sample values from surrounding voxels. Both tasks can be performed more efficient, if the viewing rays traverse the volume perpendicular to a stack of slices forming the data set. This is the point at which the shear-warp algorithm applies.

The basic approach for this technique is to transform the data set into the so called sheared object space, in which all viewing rays are parallel to the third coordinate axes. This space allows an efficient projection of the volume.

First, the object space coordinate system is transposed, that the z-axis forms the principal viewing axis. Subsequently, the data set slices are transformed to the sheared object space. For parallel projections this is achieved by shearing the data set slices in a way that the viewing rays traversing the volume become parallel to the third coordinate axis (see Figure 1.9 (a)). Perspective projections additionally require the slices to be scaled. This takes different viewing ray directions into account (see Figure 1.9 (b)). After these transformations each slice must be resampled. As soon as the volume is in sheared object space, it can easily be projected onto an intermediate image, which is parallel to the data set slices. This corresponds to a front-to-back composition. Finally, the actual output image is obtained by warping the intermediate image to image space and performing another resampling step.

In order to make the rendering process more efficient Lacroute and Levoy exploit geometric



**Figure 1.10:** Areas of transparent voxels and opaque pixels in the according scanlines are not processed. The remaining areas are resampled and composed. Image from [49].

properties of the shear-warp space. An important property is the parallelism between pixel scanlines in the intermediate image and voxel scanlines of the volume. Lacroute and Levoy use this characteristic to simultaneously traverse run-time encoded scanlines of voxels and corresponding pixels scanlines. Thereby, every opaque pixel also holds an offset to the next not opaque one in the scanline. The simultaneous traversal allows to omit regions within the scanlines which hold either transparent voxels or entirely opaque pixels. The remaining regions are resampled and composited (see Figure 1.10). Resampling is greatly facilitated, because all voxels within a slice are scaled by the same factor. For parallel projections voxels are even scaled uniformly throughout the whole volume. For a perspective view the process just described is very similar. However, run-length encoded volumes is that they must be created in a preprocessing step and require an opacity TF for construction. Consequently this method is not applicable when users should be able to interactively modify the TF. As a response Lacroute and Levoy introduce a version of the algorithm which disclaims to use run-length encoding. Instead a min-max octree [98] data structure in combination with a multi-dimensional summed-area table [13] is used to identify transparent regions within each voxel scanline. Potential drawbacks of this approach are restrictions for the opacity TF and performance losings when changing the major viewing axes. In addition, Lacroute and Levoy note that the bilinear voxel interpolation- used for resampling - may cause artifacts. Moreover, blurring may result from repeated resampling, which is done once for each slice and than once to warp the intermediate image to the final image space.

By now there also exist a series of enhancements for Lacroute and Levoy's approach. For example, a post-classification and post-shading implementation was introduced by Sweeny and Müller [83]. Further references on improved algorithms can be found in [28].

#### 1.4.4 Direct Volume Rendering in Comparison

This thesis is primarily concerned with the design of appropriate TFs, which should allow to create meaningful images of a medical 3D data set using a DVR system. Thereby, the question remains what makes DVR the technique of choice to provide a medical doctor with useful insights on a data set. Therefore, two other well known and widely used methods to view and visualize 3D data sets, namely slice by slice viewing and surface rendering, are opposed to DVR in the following.

The oldest and still most widely used method to view 3D data sets in medicine are slice-by-slice viewing approaches. Thereby, the data set is represented as a stack of slices. A physician views it by switching through the slices and building a mental model of the tissues which are spread over several slices. However, this is frequently a difficult task, since it is hard to perceive the spatial relationship of coherent parts of structures which extend to multiple slices, as well as between different structures within a data set. This is especially true for large or complex structures such as vessel trees. Another limiting factor of this approach is the required time to browse through the large amount of slices. A frequently used way in order to reduce the time effort, is to condense the original stack into a smaller amount of slices. However, this may on the one hand lead to the removal of possibly important data of high resolution. In addition, constantly increasing resolutions of volumetric data sets concurrently entail

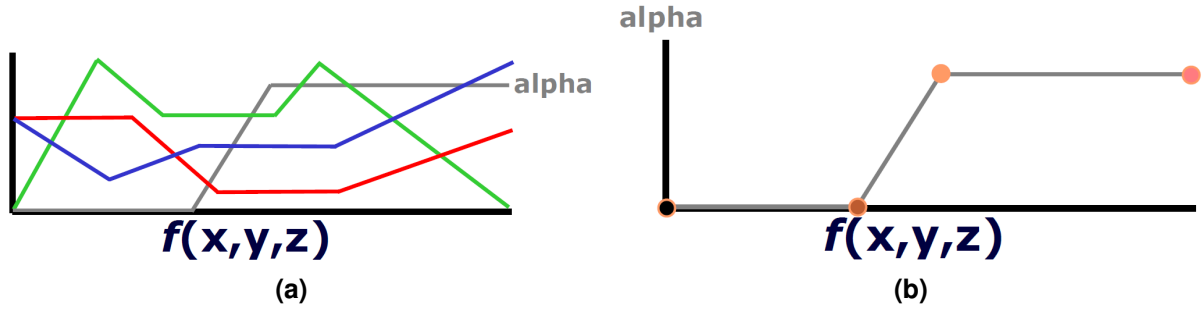
increasing numbers of slices [53]. As a result, it seems beneficial to utilize a 3D visualization of a data set in order to obtain a fast and explicit insight on features of interest. [78]

Beside DVR, surface rendering is another popular 3D rendering technique. Thereby, an intermediate step is required, which extracts geometric representations (usually triangle meshes, known as iso-surfaces) of the structures contained in the data set at first. Subsequently, the resulting surface is rendered with standard triangle rasterization. The probably most popular approach used to compute a surface representation is the marching cubes algorithm [52]. However, surface rendering seems to be inferior compared to DVR especially in medical use for a number of reasons. A major benefit of DVR is the fact that not only boundary voxels contribute to the final image. Instead, all voxels of the data set may contribute. In addition, TFs allow to flexibly assign different optical properties and especially opacities to different structures as well as to different voxels within a structure. As a result, DVR does not only allow to see surfaces of structures such as surface rendering, but also their inside as well as other structures which may be enclosed. This is of special importance in cases in which not only a structure should be seen but also its context. For example, in order to plan an operation of a tumor, a surgeon also requires knowledge on the surrounding tissues, not only the tumor itself. Another example would be the visualization of vessels running through some tissue. In addition, the flexible assignment of optical properties by TFs, makes DVR a very good tool for the exploration of data sets in contrast to surface rendering. Moreover, Kindlmann [32] states that data sets containing noise or measurements artifacts frequently result in extracted surfaces which do not match with actual object boundaries any more. These effects do also affect DVR images, however there they may result in a fuzzy image which reflects the inadequateness of the data set instead of displaying distinct but misleading shapes. Similarly, Lundström [53] argues that the poor contrast between tissues and indistinct boundaries in medical data sets, lead to significant problems in the extraction of appropriate surfaces. A major benefit, which was used to argue in favor of surface rendering for a long time has been its higher speed on equivalent hardware. Although the surface extraction may take some time depending on the approach used, it can be done as a preprocessing step. As a result, the subsequent surface rendering could be done very fast and at interactive frame rates very early. In contrast DVR could not achieve realtime speed for a long time. However, due to the advances in hardware speed and especially in the use of the GPU and its memory for rendering, this earlier advantage of surface rendering is hardly an argument in favor of this technique in the meanwhile. In relation to visualizing surfaces (with DVR or surface rendering) [78] Napel demurs, that surface rendering leads to a clear separation of surfaces since it inherently comprises a segmentation step as opposed to DVR, where overlapping value ranges could lead to unwanted visualizations. However, in the meanwhile this concern is diminished due to the advances in the research on multi-dimensional TFs.

Concluding it can be said that DVR seems to provide a lot of benefits compared to the other discussed approaches, however this does not make it a universal remedy. DVR may indeed deliver very useful visualizations when an appropriate TF has been found. However, this proves to be a difficult task, which is subject of the further parts of the thesis. Especially experienced physicians are frequently used to slice-by-slice viewing approaches [53]. As a result, they may prefer this method to the efforts of creating proper TFs. Hence, DVR may more be regarded as complement than a substitute to slice-by-slice viewing at the moment, although it gains importance.

## 1.5 Transfer Functions

TFs play an essential role in DVR, since they have a major impact on the output images generated by a rendering system. Recalling the DVR pipeline, TFs operate in the classification stage. There they classify voxels or interpolated ray samples, depending on whether pre- or post-classification is performed, in the volume by assigning them optical properties (usually color and opacity), based on their data values. Subsequently, the rendering system can evaluate the underlying optical model which represents a simplification of the lighting equation, in order to determine the output image.



**Figure 1.11:** Two examples of interfaces for basic transfer function design. Editor (a) provides a four polylines to define each of the  $RGB\alpha$  values over the data value range. Editor (b) only provides a polyline for the definition of the  $\alpha$  value. The color values can be defined at the points along this line. Images from [41].

As a result, TFs can be seen as a tool to modify the output image of a rendering system for a certain volumetric data set. They can be utilized by users in order to define how different structures in the data set should be visualized.

In general a TF can be considered as a mapping from  $n$  data values, which are properties defining each sample  $s$  in a volume such as intensity and gradient magnitude, to  $m$  optical properties:

$$c = \tau(s), \tau : \mathbb{R}^n \rightarrow \mathbb{R}^m \quad (1.6)$$

Thereby,  $\tau$  represents the TF,  $s$  describes a sample for which the optical properties should be determined and  $c$  is a vector holding of the optical output properties. In the current thesis we use  $m = 4$  and  $n \geq 1$ . Hence, such a TF describes a mapping from a set of  $n$  arbitrary data values to four optical properties stored in the vector  $c$ . Here  $c$  holds the red, green and blue channels representing the color as well as the opacity and is a vector of the format  $RGB\alpha$ . In the simplest case, when the volume is only represented by a scalar field a TF performs a mapping from an intensity value to a  $RGB\alpha$  quadruple.

## 1.5.1 Transfer Function Design

In the meanwhile the purpose of TFs has been clarified. However, the question about how they can best be made available to users still remains. Users should be provided with the required means to efficiently design appropriate TFs resulting in meaningful renderings. Before the general developments in this area are discussed, one should be aware, that the TF design process basically consists of two steps: the identification of features of interest in the TF space and the assignment of optical properties to them. The second step must be differentiated from the mapping from data value to optical properties which the TF performs later. Here we only determine the optical properties which the TF assigns to samples when applied.

### 1.5.1.1 Basic (Manual) Transfer Function Design

Initially, very basic manual approaches have been proposed to accomplish the task of TF design. One requirement for these systems is a real-time response. A user must immediately see the image, resulting from a change of the TF in order to be able to further improve it. A simple example of an early interface for TF design can be seen in Figure 1.11 (a). A separate polyline consisting of a number of points connected by linear ramps is defined for each of the  $RGB$  channels of a color and the opacity, over the full data value range. By changing the point positions and amount of points of each polyline a user can adopt the contributions of each optical property on y-axis, to the corresponding data value on the x-axis.

Based on the insight that the specification of an appropriate opacity mapping is the primary concern of TF design, simplified approaches similar to Figure 1.11 (b) were introduced. Here, only the opacity is defined by a polyline, while the color can be defined at the points along the line. The opacity is of special importance since it defines what is visible. As a result, its specification has a major impact on the definition of a region's extent. Colors on the other hand facilitate the visual distinction of different regions. Therefore, it is important to assign a single color to a coherent region and to assign easily distinguishable colors to different regions. Realistic colors, however, are not necessary. It is usually sufficient to assign *false colors* to data set features in order to understand the content of a rendered image. These are colors which do not correspond to the actual appearance of depicted structures.

Unfortunately, even the design of a simple opacity TF is a complex task. This is especially true, if users do not possess any a priori knowledge about a given data set. In this case the identification of exact value ranges of interesting structures in the TF space is especially difficult. The design stage is further complicated by the fact that small changes in the TF often lead to strong and unintuitive changes in the output image. This makes it difficult to slowly adjust a TF toward an appropriate rendering. In addition, not only a single structure itself is important. Its context should also be depicted with an appropriate opacity - high enough to see a context, but low enough to allow to see the actual structure. In order to give users at least a clue about the value ranges of interesting structures, the global histogram of the data set is frequently depicted in the background of such a coordinate system. The basic idea here is that peaks correspond to regions of interest. However, frequently interesting features are not visible in a histogram, while noise forms peaks.

Basically, the problem of these approaches is the large number of degrees of freedom, in combination with the lack of constraints and guidance. This also includes that these basic methods perform the identification of features of interest simultaneously with the assignment of optical properties, although a stepwise processing, as done by König and Gröller [40], would be much easier.

### 1.5.1.2 One-Dimensional versus Multi-Dimensional Transfer Functions

A simple 1D TF, as considered previously, maps only from a single data value to an opacity and eventually color value. However, one dimension is often insufficient to reliably differentiate between different structures. For example, data sets generated from MRI or CT scans frequently contain different tissues, which share at least partially the same intensity ranges. Since spatial information is usually lost in the TF domain, it is impossible to differentiate these overlapping tissues in the TF domain with only one dimension. When the user assigns optical properties which fit for one tissue, at least a part of the other one will automatically be assigned the same properties which may not be appropriate.

As a result, it is necessary to classify samples by multiple characteristics. This significantly increases the ability to distinguish different structures, since it is less likely that overlaps of tissues occur in multiple value ranges. One approach for obtaining multiple data values per sample is to acquire multiple data sets from the same patient using different imaging modalities. Alternatively, a series of data values which are derived from the intensity values in a certain neighborhood of a voxel can be used as additional differentiators between samples. Examples are the gradient magnitude [33], curvature [24] or occlusion [10] of a voxel. In general TFs, which assign optical properties depending on more than one data value are called multi-dimensional TFs.

However, besides their contribution to an improved feature classification, multi-dimensional TFs also add significant complexity to the TF design process. The enormous handling requirements for the increased degrees of freedom, as well as the additional memory requirements frequently lead to significant problems. Hence, 1D TFs are often preferred.



### 1.5.1.3 Advanced Transfer Function Design

The task of TF design can be considered as unintuitive, time consuming and tedious. As a consequence of the troubles resulting from basic TF design approaches, a series of (semi-)automatic methods, as well as improved user interfaces for manual adoption and special methods to deal with multi-dimensional TFs have emerged.

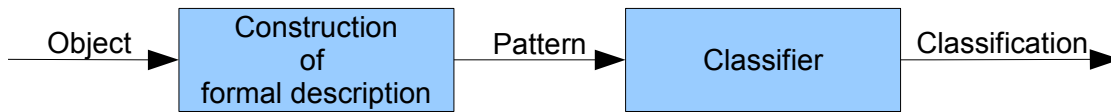
Semi-automatic and automatic approaches are mainly used in order to identify data value ranges of regions of interest and assign an initial opacity and color to them. Based on this foundation it is easier for a user to further adopt and combine the given results to desired TFs. (Semi-)automatic approaches can basically be divided into data- and image centric methods. Data centric approaches analyze the data set in order to identify value ranges of interesting features. Subsequently, color and opacity are assigned to the values in a range, based on their location in the full range of data values and a predefined scheme. For example, the color is assigned over a full value range, while the opacity is assigned depending on a function used for distribution. Possible shapes of an opacity distribution over a value range are a trapezoid, tend, linear ramp and gaussian curve. Examples of data centric approaches can be found in [33], [10], [21], [55], [54]. Image centric methods on the other hand often start with random or user given TFs and iteratively optimize them based on an evaluation of the resulting rendered images of TFs from previous iterations. For the further development of the TFs genetic algorithms or other stochastic search techniques are applied. Examples can be found in [22], [100]. Thereby, it should be mentioned that the image centric approach presented in [100] is designed for editing TFs.

Although the presented approaches to define initial TFs are very useful, and significantly facilitate the further TF design process, they are solely not sufficient. As Engel et al. [15] states, "[...] fully automatic transfer-function design is not practical because the definition of an object or feature of interest depends on the task and the kind of data being visualized". As a consequence a user should be provided with an appropriate user interface which allows to intuitively adopt precalculated TFs. According to Engel et al. [15] it is especially important that such user interfaces constraint user interactions with the TF domain in a meaningful way and guide the user towards desired TFs. An example for such a user interface is introduced by König and Gröller [40] which guides the user step by step, from value range definition, over color assignment to opacity assignment, through the TF design process. A further example, are the interaction widgets provided by Kniss et al. [42], which allow the user to adopt a TF by interactions in the spatial domain. Another interesting approach is presented by Marks et al. [57], who combines an image centric method with a design gallery. At first a large number of perceptually different images is created and then presented in the design gallery according to a specific arrangement scheme. The gallery also allows to combine single features. Note, that user interfaces could also be considered as image centric approaches, with the user being the instance evaluating the image, and adopting the TF accordingly.

In order to deal with the hassle of multi-dimensional TFs, methods for dimensional reduction as well as procedural techniques, which obviate the burden of memory requirements of multi-dimensional TFs have been introduced. See [69] and [44] respectively.

## 1.5.2 Comparison and Relations: Transfer Function Design - Classification, Object Recognition and Segmentation

The primarily purpose of this thesis is to facilitate the design of appropriate TFs which should allow a rendering system to generate images in which important structures are properly separated and emphasized. In order to achieve this goal it is important to extent the basic TF design techniques by more advanced methods. These frequently go back to methods known from digital image processing, such as segmentation, classification and object recognition. However, these terms frequently have different meanings in relation to TFs. Therefore, this section explains the differences and similarities of these terms in the context of digital image processing and the creation and application of TFs in DVR.



**Figure 1.12:** The main steps of the pattern recognition process. Image recreated from [81].

In order to facilitate these explanations, we consider the essential steps leading to the creation and application of TFs using a data centric approach. These steps are listed in the following and are referred to as TF-process in this work.

1. **Construction of TF-domain:** Extract pattern (i.e. intensity and gradient magnitude value) from each voxel in the volume and apply them in the TF domain (feature space).
2. **Identification of value ranges:** Create classes from the patterns (which represent voxels) in the TF space. These classes define the value ranges in the different dimensions.
3. **Creation of TFs:** This is done by specifying the distribution of optical properties over the value ranges.
4. **Application of TFs:** As a part of the classification stage in the DVR pipeline, a TF is applied to the single samples in the volume in order to assign optical properties to them. This leads to a partitioning of the volume and can be considered as a classification or segmentation of the voxels in the volume.

Note, that steps one to three are part of the TF-design process.

**Pattern Recognition & Classification:** Engel et al. [15] states that the "[...] process of finding an appropriate TF is often referred to as classification". Moreover, he defines classification in volume visualization as "[...] the process of identifying features of interest based on abstract data values" and concludes, that "[...] classification is essentially a pattern-recognition problem". More accurately classification can be considered as one of the main steps of the object recognition process as shown in Figure 1.12.

Considering the first two steps in the listing above this correlation can be explained in more detail. A pattern recognition process usually starts with the definition of scalar properties, called descriptors or features, which are used to describe an object. For each object the corresponding feature values are consolidated in a vector (in the simplest case) which forms a pattern. Subsequently, all objects can be arranged in a feature or pattern space, whose dimensionality depends on the number of descriptors used for the objects, according to their pattern. Thereby, the feature space is spanned by the set of all possible patterns [81]. For a 2D example, the features space can be depicted as a 2D plot with one axis for each feature. The objects are assigned as points depending on their pattern. In conjunction with TFs, objects are voxels. The features describing them are properties such as intensity, gradient magnitude or curvature and the pattern space is the TF domain. A pattern arranged in the TF space could hence be a vector of intensity and gradient magnitude.

As soon as all patterns have been applied classes are formed from them. Classification is performed by a classifier. This may work with many different methods such as neuronal networks, statistical pattern recognition or syntactic pattern recognition. One possibility for classification, which is frequently used in TF-design, are cluster analysis methods such as the k-means algorithm. For example, the systems of Haidacher et al. [21] and Correa and Ma [10] are based on such techniques. A great benefit of these approaches is that the number of classes do not need to be known in advance. Clustering divides the

set of all patterns, and therefore the objects they represent, in the feature space into subsets depending on their similarity to each another. As a result of this process a number of clusters representing different classes emerge [81], [19]. In relation to TFs each cluster represents a region of interest in the TF domain and defines its value ranges due to its extent. Each cluster consists of the voxels representing this region of interest. At this point the features of interest are identified. Therefore, the pattern recognition task of the TF-process described above as well as the involved classification are accomplished.

In addition to the previous definition, the term classification is also frequently used for the application of the TF such as by Levoy [50] or Müller [60]. This corresponds to step four in the TF-process. In this context the terminology refers to what is done in the classification stage in the DVR pipeline: Due to the mapping from data values to optical properties, samples are aggregated to coherent regions/classes, which can easily be distinguished, in the volume.

**Segmentation:** According to Gonzales and Woods [19], image segmentation subdivides an image into its constituent regions or objects". Moreover, Engel et al. [15] considers segmentation as an object recognition process. Based on these descriptions it seems logic, that segmentation may also form a part of the TF-process. Indeed several authors consider segmentation as a part of the TF-process as described above. According to Müller [60] for example a TF performs two tasks: segmentation - "the partitioning of the input image into multiple regions" - and classification - which he uses as a term for the application of the TF such as described in step four of the TF-process. Moreover, Müller [60] states, that classification was recently seen as a form of segmentation. Based on these considerations, both segmentation and classification are mainly related to the effect resulting from the application of TFs.

Above segmentation was considered in the context of the TF-process. In the following the focus lies on the delimitation of the application of segmentation methods from the TF-process. When considering simple image segmentation methods and their application on 2D or 3D images independent from the TF-process, there are still significant differences as opposed to the TF-process. Region growing for example, starts from a seed point for each region, and repeatedly extends by adjacent pixels/voxels which conform certain criteria for growth. Splitting techniques divide the input image into adjacent regions until each region fulfills a certain homogeneity criterium. Merging approaches merge adjacent regions as long as the new combined regions comply a homogeneity criterium. All these methods have in common that they operate in the spatial domain. The regions of interest (classes) used for the specification of TFs however are usually identified in the TF domain (feature space). For threshold based segmentation on the other hand the difference to the TF-process seems to be less distinct. Frequently thresholds are determined by applying parametric clustering methods, which are classification techniques, on the global histogram of an image. Subsequently, they are deployed to partition the image [81]. Thereby, the histogram of a scalar image can be considered as a one-dimensional feature space in which the classification takes place. Although there are similarities between conventional image segmentation and the task of TF specification, Kindlmann [32] identifies a major difference in their results: "Segmentation leads to a three dimensional model, based on volume data, while a TF is just one parameter of the direct volume rendering process, which creates simply an image".

Concluding it can be said that all, object recognition, segmentation and classification may play a role in the TF-process. However, there needs to be distinguished between using such methods as a part of the TF-process and between their independent application on volumes or other data. An accurate distinction of such methods from the process of TF specification and application is difficult. This observation is also confirmed by Lundström [53], who states that "there is no well-defined distinction between classification and segmentation, and advanced TFs". Moreover, one should differentiate between the domains in which classification/segmentation may take place. For example, the result of a data centric approach used for TF design would be a classification/segmentation in the TF domain. On the other hand the application of the TF leads to a classification/segmentation of the volume, which constitutes the spatial domain. Note, that the DVR process is still not completed by this segmentation. At least the composition stage would still follow.



# Chapter 2

## Related Work

The current chapter discusses a number of different approaches which facilitate the creation of TFs. This includes data and image centric methods to generate an initial set of meaningful TFs and related images as described in Section 2.1 and 2.2, respectively. Moreover, user interfaces which allow to modify the original set of TFs are presented. Thereby, the user interfaces discussed in Section 2.3 are created specifically for TF design while those from Section 2.4 originate from different areas of research. In addition, Section 2.5 discusses methods which allow to mitigate the drawbacks related to multi-dimensional TFs.

Some of the presented approaches are used in Chapter 3 and 4 to describe a powerful combination which aims to create good TFs and meaningful images in a fast and intuitive way.

### 2.1 Data Centric Approaches

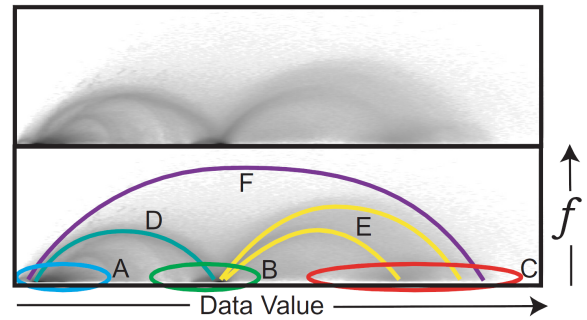
Data centric approaches identify regions of interest by analyzing a scalar or multivariate data set. Thereby, the values describing single voxels may be intensity values resulting from various imaging modalities or properties derived from them. The resulting TFs are used for rendering. This section introduces a selection of data centric approaches which use different derived properties beside the intensity values of input volumes.

#### 2.1.1 Transfer Functions Based on Derivatives

##### 2.1.1.1 Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering

Kindlmann and Durkin [33] introduce a way to semi-automatically generate multi-dimensional opacity TFs for data sets in which boundary regions are the structures of interest.

This approach uses the intensity value as well as the first and second directional derivatives at each sample position for the classification of features of interest. In order to reveal a relation between data values ( $f$ ) and their first and second directional derivatives ( $f'$ ,  $f''$ ) independent of the corresponding voxels positions in the data set, a volume histogram is created. It is a three-dimensional histogram (with the axes  $f$ ,  $f'$ ,  $f''$ ) in which each bin holds the frequency of those voxels from the data set with the same  $f$ ,  $f'$  and  $f''$  values. For its creation, the authors suggests that it is sufficient to measure  $f$ ,  $f'$  and  $f''$  once per voxel at the volumes original sample points. By projecting the histogram along the  $f'$  or  $f''$  axes, scatterplots of  $f''$  or  $f'$  versus  $f$  are obtained. A projection of  $f'$  versus  $f$  results in arches as shown in Figure 2.1. Thereby, each arch describes a boundary which is shared by two adjacent materials. Each end of the arch corresponds to a homogeneous region within these materials. The information stored in the volume histogram and corresponding scatterplot is subsequently utilized to provide users with high-level control in order to define boundary opacities without knowing the involved data values.



**Figure 2.1:** A projection of the 3D histogram along the  $f'$  axes on a scatterplot results in arches which depict boundaries between two adjacent materials. The encircled ends of each arch belong to the homogenous regions on both sides of the boundary. Image from [43].

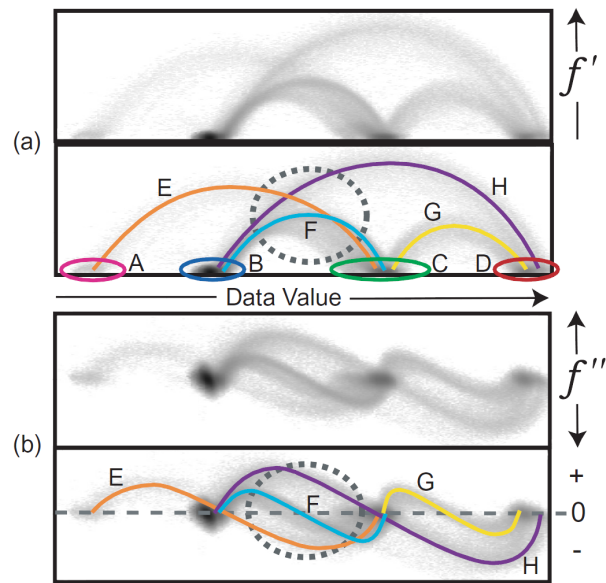
The approach of Kindlmann and Durkin [33] is an essential facilitation in TF construction if the user is interested in boundary regions. However, a projection of  $f'$  versus  $f$  may also cause problems. Certain spatial arrangements of features of interest consisting of certain materials can lead to overlapping arches. This means that boundaries partially overlap in both, intensity and gradient magnitude. An example can be seen in Figure 2.3. The boundaries described by these arches cannot be differentiated unambiguously. In order to resolve this problem the use of the second directional derivative is of interest. Figure 2.2 shows how overlapping arches in a scatterplot of  $f'$  versus  $f$  (a) are resolved when plotting  $f''$  versus  $f$  (b).

### 2.1.1.2 Visualization of Boundaries in Volumetric Data Sets Using Low High Histograms

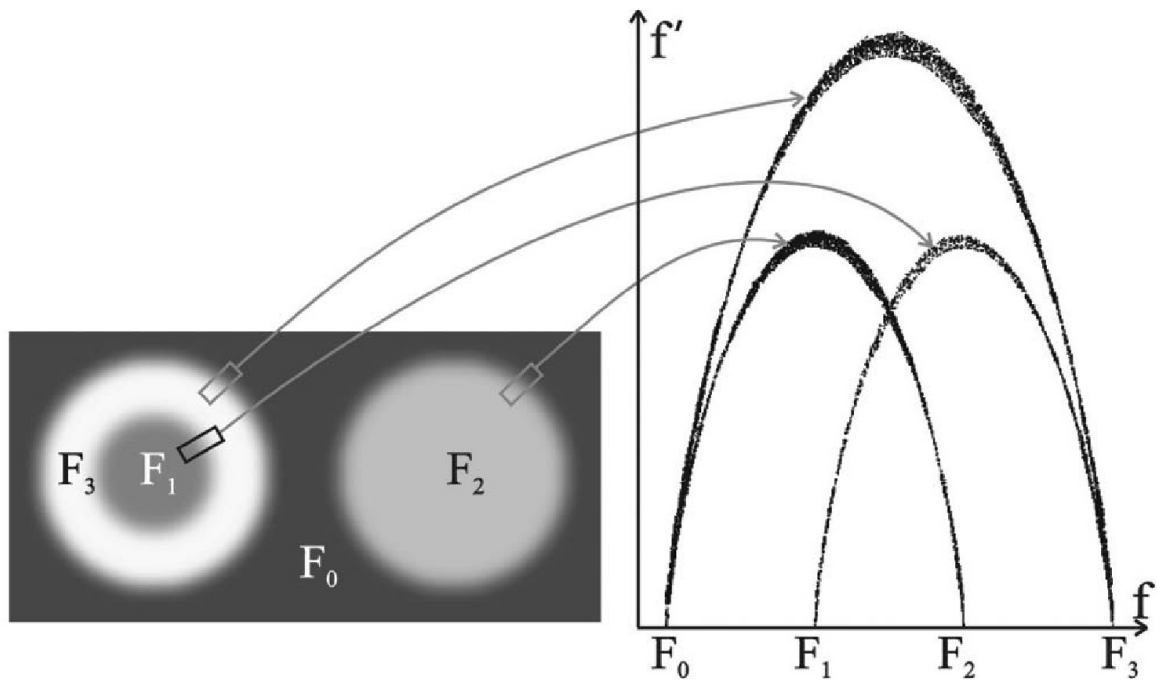
Sereda et al. [88] introduce another approach based on derivatives, which, in the ideal case, offers a more compact depiction of boundaries as points instead of arches. This especially facilitates their selection within a large number of features in the data set and also solves the problem of overlapping arches. In order to obtain the depiction of the boundary regions, the material values on both sides of a boundary are detected for each boundary voxel and then depicted in a Low High (LH) histogram. Thereby,  $F_H$  and  $F_L$  label higher and lower intensities of the two boundary materials, respectively. The LH histogram has two dimensions. Each of the values  $F_H$  and  $F_L$  defines one axis. A comparison of the LH histogram and arch representation of a simple data set can be seen in Figure 2.4. Figure 2.5 shows how intersecting arches are resolved in a LH histogram representation.

For the construction of the LH histogram, the gradient magnitude of each voxel in the data set is determined at first. If the gradient magnitude of a voxel is smaller or equal a certain  $\epsilon$ , this voxel lies within one of the materials forming the boundary. For these voxels their intensity value is assigned to  $F_H$  and  $F_L$  ( $F_H = F_L = f(x)$ ). All other voxels lie on boundaries. Their  $F_H$  and  $F_L$  values are determined by integrating the gradient field in both directions, to the right and left of the boundary, until a stopping criterion is reached. This is either a local extremum or an inflex point. In the end, each voxel has an assigned pair of values  $[F_H, F_L]$  from which the LH histogram can be generated by accumulating voxels with the same pair  $[F_H, F_L]$ . In the ideal case, in which materials on both sides of a boundary have constant values, each boundary occurs as a point in the left upper half of the LH histogram. This depiction can be seen compared to an arch representation of the same data set in Figure 2.5. The voxels of the materials surrounding a boundary, form the diagonal in the LH histogram.

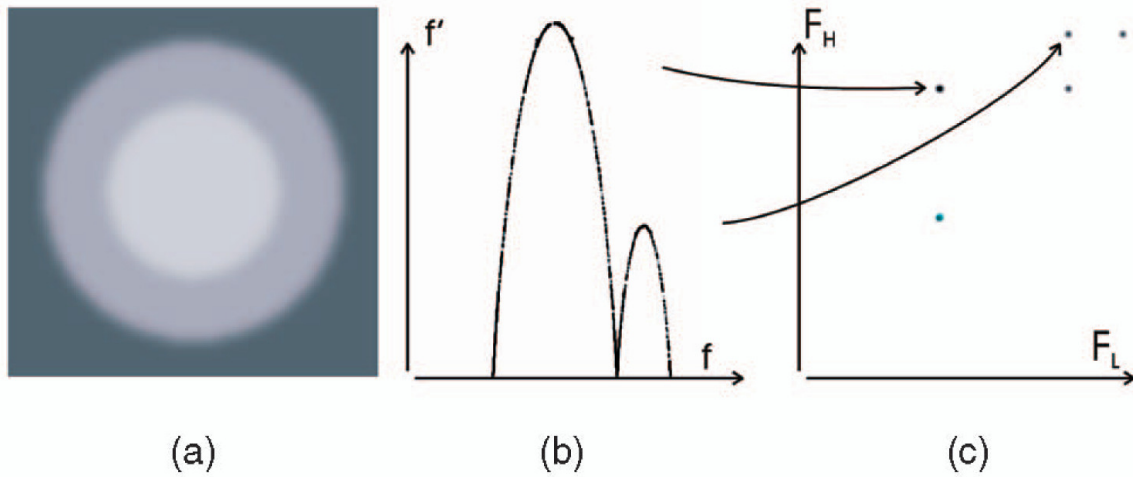
This ideal situation rarely occurs. Noise, bias and thin objects cause the depiction of boundaries to be less compact. Noise leads to blobs, bias deforms points of a boundary to lines and thin objects cause horizontally and vertically elongated blobs in the LH histogram. However, these effects also influence arch representations. Therefore, LH histograms still facilitate the identification and selection of boundaries by offering a more compact depiction. The difference for noisy data as well as data with



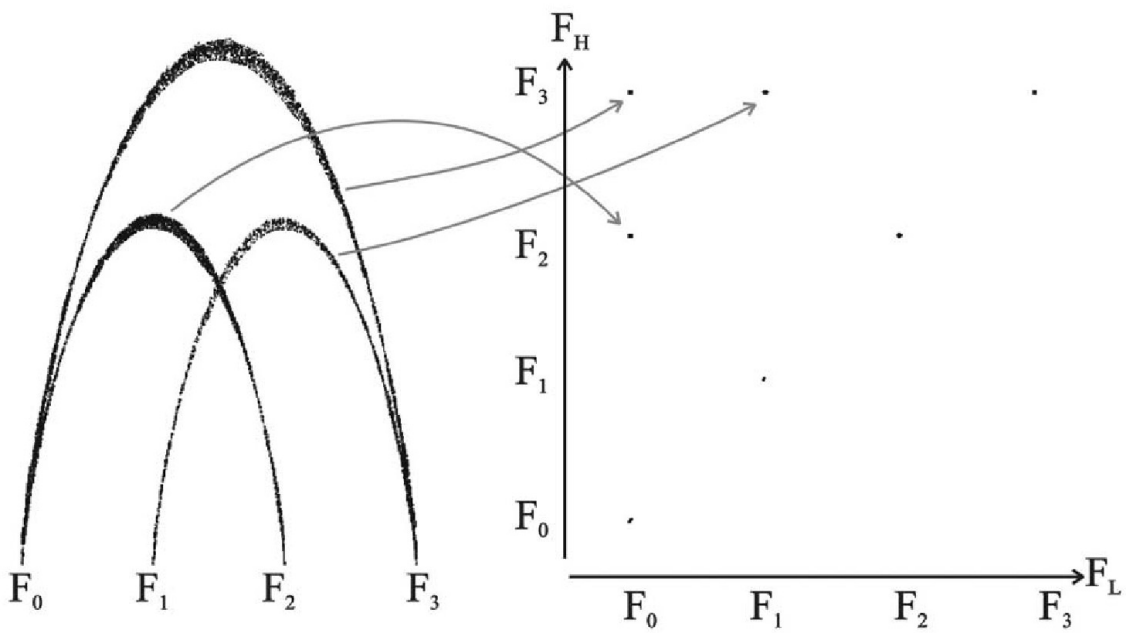
**Figure 2.2:** Overlapping arches occurring in the projection along  $f''$  can be resolved with a projection along  $f'$ : A, B, C, D describe the homogeneous regions/materials between which the boundaries E, F, G, H are located. (a) shows a scatterplot of  $f'$  versus  $f$  in which the arches E, F, H partially overlap. Hence, the corresponding boundaries cannot be differentiated unambiguously. In (b) a scatterplot of  $f''$  versus  $f$ , depicting the same boundaries, is shown. It resolves the arch overlaps shown in (a). Hence, using  $f''$  as a third dimension provides a better separability of the corresponding boundaries. Image from [43].



**Figure 2.3:** Crossing arches resulting from a certain spatial arrangement of materials of certain intensities. Image from [88].

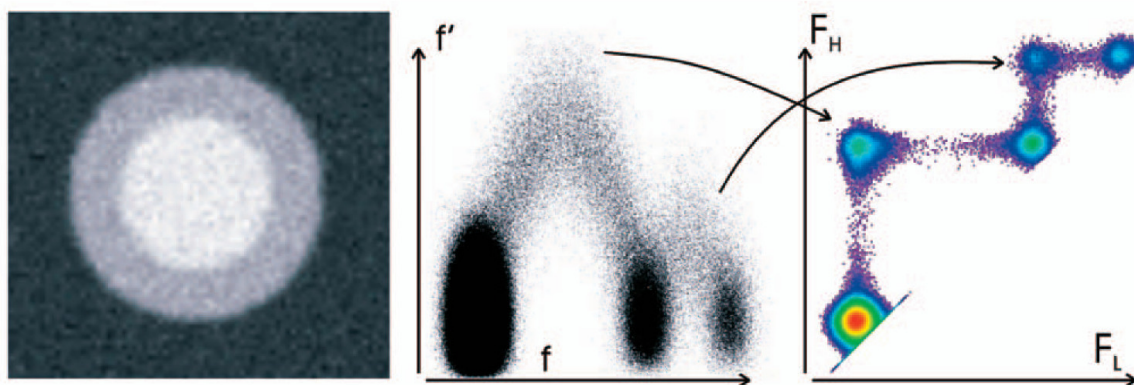


**Figure 2.4:** (b) and (c) depict the arch and LH Histogram representation of the data set shown in (a) respectively. (b) depicts boundaries as arches, while (c) depicts boundaries as points. Image from [88].

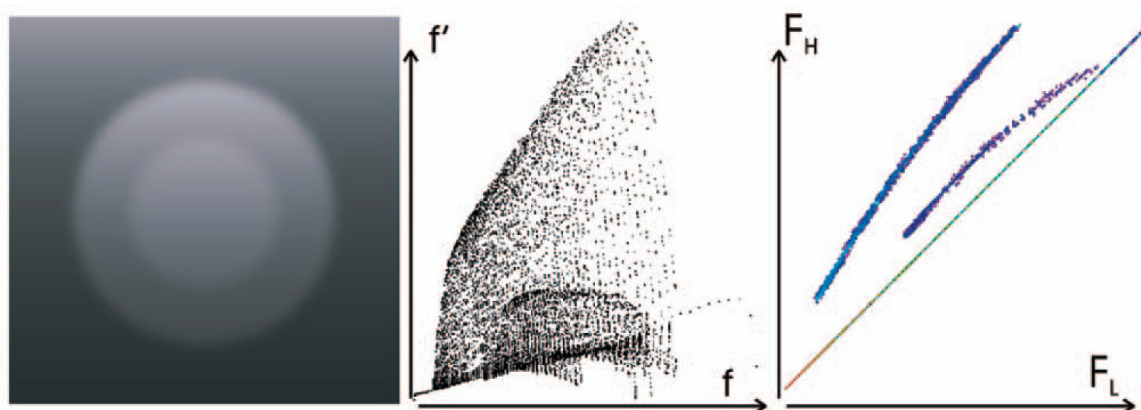


**Figure 2.5:** Overlapping arches are resolved in the LH histogram representation. Image from [88].





**Figure 2.6:** Boundaries of a data set with added noise shown in the arch and LH histogram representation. Image from [88].

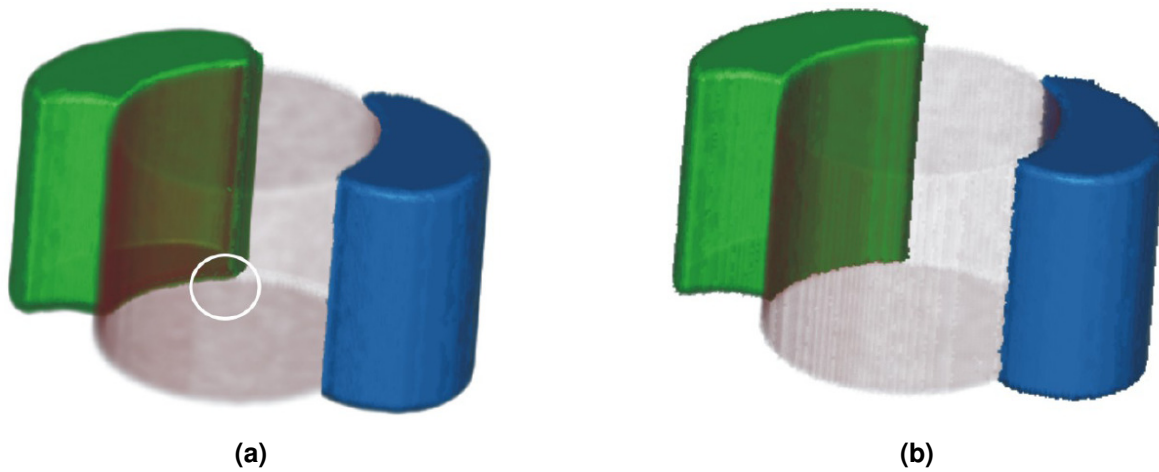


**Figure 2.7:** Boundaries of a data set with strong bias shown in the arch and LH histogram representation. Note, that a bias of this strength is usual for MRI data sets. Image from [88].

bias can be seen in Figure 2.6 and 2.7, respectively. Finally, voxels of a detected boundary are modulated according to the gradient magnitude which is used as a third dimension.

An issue related to LH histograms is that neighboring materials share the same boundary, which circumvents an unambiguous classification of boundary voxels to one structure. As a response to this problem Sereda et al. introduce mirrored LH histograms [87]. Thereby, the second derivative is utilized throughout the integration of the gradient field in order to approximate the "exact" edge of the boundary. Subsequently, voxels are assigned to the LH histogram depending on whether their intensity is higher or lower than that of the edge voxel. A voxel with higher intensity is projected above the diagonal of the plot, while a voxel with lower intensity is projected below. The second case is realized by exchanging the corresponding pair  $[F_H, F_L]$  of a voxel. Due to this approach it can be circumvented that a data set structure "loses" a boundary. The different visualizations of a data set using the conventional and mirrored LH histogram can be seen in Figure 2.8.

A drawback of all derivative based approaches is that they only help to identify boundary regions. However, this is sufficient for many applications.



**Figure 2.8:** A comparison of data set visualizations based on standard and mirrored LH Histograms. (a) shows a visualization resulting from the use of a standard LH Histogram. Note, that the boundary of the green structure to the red structure is missing. (b) shows a visualization resulting from the use of a mirrored LH histogram. In contrast to Figure (a) the boundary of the green structure to the red is depicted now. Image from [87].

### 2.1.2 Curvature Based Transfer Functions

Hladuvka et al. [24] present a new class of TFs which can be used in addition to other classes like those working with density values. They use the principal direction and curvature of a point on a surface to describe its curvature. Depending on their signs, the principal curvature values describe different shapes (plane, parabolic cylinder, paraboloid, hyperbolic paraboloid) of the neighboring surface of a voxel. Subsequently, when used as data values in the TF domain the two principal curvature values can be used to emphasize or suppress certain features and assign colors to them, depending on their shape. Moreover, the use of curvature allows to see structural changes within objects, which enables to set smooth transitions of the optical properties within objects. This method also has several drawbacks. It is hard to find an appropriate and robust algorithm for principal curvature magnitude estimation which is generally applicable. Furthermore, the curvature estimation can be very time consuming which may be problematic if hard real-time constraints for interactive rendering have to be fulfilled. However, the estimation can be performed as a preprocessing step and stored as a separate volume.

Kindlmann et al. [34] also suggest the use of different components of curvature information as a part of a multi-dimensional TF domain. For curvature measurement they rely on a range of methods from differential geometry as well as from signal processing and filter design. They utilize curvature as a part of a multi-dimensional TF in three different areas. First, the application of curvature in TFs helps to enhance non-photorealistic renderings by avoiding constantly changing contour thicknesses. Secondly, it can be used for smoothing noisy object surfaces which frequently result from medical imaging technologies. Thereby, an isotropic surface smoothing can be performed by a minimization of the integral of total curvature, which is similar to Gaussian image blurring. Finally, regions of high geometric uncertainty can be visualized using the flowline curvature, which serves as an indicator of change between different iso-surfaces.

### 2.1.3 Feature-Size Based Transfer Functions

A further dimension which can be used within multi-dimensional TFs is the size of features to which voxels belong to. In comparison to the gradient magnitude, the application of the feature-size allows to

classify not only borders but also the insight of materials.

### 2.1.3.1 Structure Size Enhanced Histogram

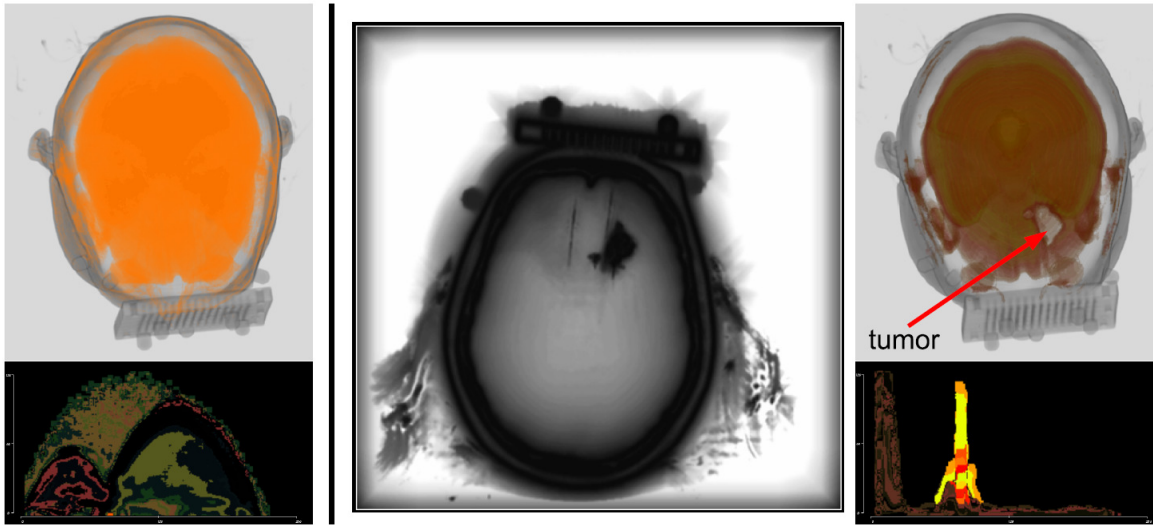
The most recent approach in this direction has been presented by Wesarg and Kirschner [93]. They assert that humans differentiate between location and size when perceiving structures in image data. Therefore, they introduce an approach based on a new type of 2D histogram which assigns voxels to bins according to their intensity and the size of the structure they belong to. In addition, they incorporate spatial information which classify the 2D bins. This histogram is called structure size enhanced (SSE) histogram and is utilized in order to specify multi-dimensional TFs.

As a first step, for each voxel the size of the structure it belongs to, must be estimated. Therefore, 26 direction vectors are defined in each of the directions of the 26 immediate neighbors of the currently regarded voxel  $P_{ref}$ . Beginning from  $P_{ref}$ , in each direction the number of steps (voxels) to the end of the enclosing structure is counted. The counting stops, if a voxel intensity exceeds or falls below a certain threshold. More precisely the intensity  $I_{curr}$  at the next voxel must be within the range  $[I_{ref} - \tau I_{max}; I_{ref} + \tau I_{max}]$ . Thereby,  $I_{ref}$  is the intensity at the starting voxel,  $I_{max}$  the maximum intensity of the data set and  $\tau$  a tolerance value between 0 and 0.5. Subsequently, a multi-scale approach is used to define the structure size instead of summing up all counted voxels. Thereby, the number of voxels in each direction are assigned to a certain scale interval. Each interval has a certain associated scaled size number. These size numbers are summed up for all directions in order to define the size of the structure. For example, for a  $512 \times 512 \times 512$  dataset the intervals would be 512-257,256-129,128-65,64-33,32-17,16-9,8-5,4-3,2,1. These are 10 intervals. To the smallest interval(1) a scaled size 1 is assigned, while a scaled size of 10 is assigned to 512-257. Hence, a direction reaching 7 voxels to the left would be assigned a scaled size of 4. After this, the SSE histogram can be generated by assigning each voxel to a corresponding bin, according to its intensity and structure size value. In this representation manipulations widgets as introduced by Kniss et al.[42] can be used to select structures of interest. Alternatively, an opacity TF can be automatically specified based on the 2D bin locations of voxels. In order to incorporate spatial information two ways are presented. The first is to calculate the euclidian mean distance of each 2D bin to a user defined reference point. The resulting values are stored in a distance map. The second method is based on the approach introduced by Roettger et al. [74]. Finally, Wesarg and Kirschner suggest to define a TF based on their method so that the opacity of a voxel increases with the intensity and decreases with the structure size. This is reasonable since large regions with low intensity (for example air) are usually of low interest in medical data sets.

Later, Wesarg and Kirschner [92] have compared the application of feature size and gradient magnitude as a second TF dimension in 2D histograms. They conclude that feature size is preferable for two reasons. First, it allows a better discrimination of structures in the 2D histogram and more importantly 2D histograms based on structure size are much easier to interpret by the user.

### 2.1.3.2 Size-Based Transfer Functions: A New Volume Exploration Technique

Another approach using feature size is introduced by Correa and Ma [9]. Here, for each voxel the size of the feature a voxel belongs to is stored in a scale field. A scale field is a 3D scalar field which holds the corresponding feature size for every voxel. Scale fields are computed based on continuous space theory ([45], [51], [99]) and a set of scale detection filters. One of the main benefits of scale fields is that they provide a continuous representation of size which also allows to capture small variations in size. Previous multi-scale approaches, such as pyramid representations, often incorporate a discrete and disperse representation of size which rarely allows to detect small size variations. Slightly varying sizes are assigned the same discrete size value instead. Another originality of this approach is that size is not used to form a 2D TF together with the intensity dimension of the data set. Instead two 1D TFs are used



**Figure 2.9:** A head CT data set containing a tumor is visualized in order to compare results from gradient magnitude and feature size. To the left a visualization based on intensity and gradient magnitude with the corresponding joint histogram is shown. The tumor could not be discriminated from the surrounding tissue. To the right a visualization based on intensity and feature size with the corresponding color coded SSE histogram is shown. The tumor appears as a whole. The middle depicts an axial slice of the structure size estimation. Image from [91].

in order to avoid a too complex feature space. Thereby, the size TF maps to color and opacity. See Figure 2.10 to see a comparison to conventional approaches.

## 2.1.4 Visibility and Occlusion Based Transfer Functions

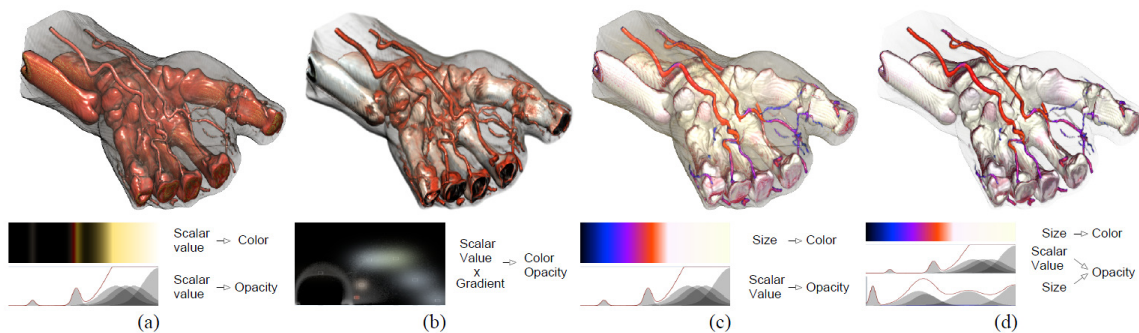
### 2.1.4.1 The Occlusion Spectrum for Volume Classification and Visualization

With "The Occlusion Spectrum" Correa and Ma [10] introduce a new approach which performs classification of data set features using occlusion as a second TF dimension. According to the authors, the most important merit of this technique is its capability to separate not only boarder, but any structures of same intensity contained in volumetric data. Further advantages are the coherence, robustness to noise and bias, as well as its good results irrespective of the the feature size.

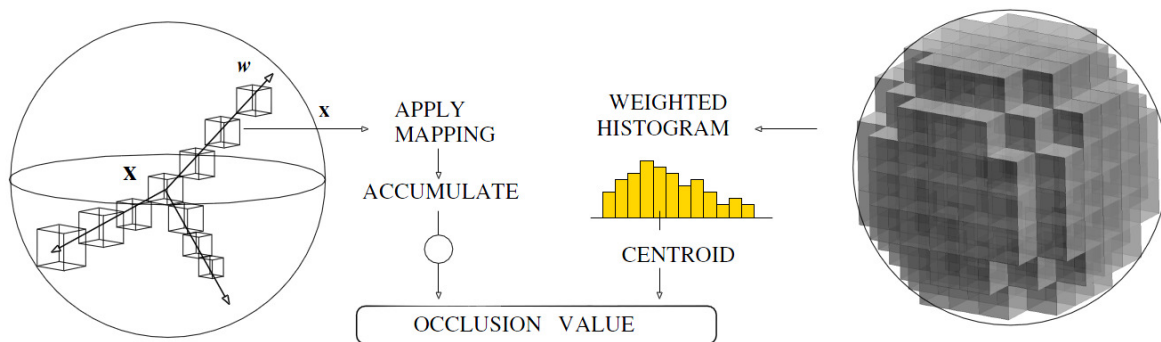
The basic step's involved in this approach are the following:

- Calculate the ambient occlusion for each voxel in the data set.
- Generate the occlusion spectrum. It is represented by 2D Histogram of intensity and occlusion.
- Identify structures of interest in the spectrum by clustering.
- Emphasize structures of interest. Therefor, the space of an interesting structure spanned by scalar value and occlusion in the spectrum is mapped to  $RGBA$  values.

Correa and Ma [10] define the ambient occlusion as the weighted average of intensities in a spherical neighborhood. In a discretized way the ambient occlusion  $O(x)$  is given by the equation  $O(x) \approx \frac{1}{N} \sum_{\phi=0}^{\pi} \sum_{\theta=0}^{2\pi} A(x, \omega(\theta, \phi))$ . Thereby,  $x$  is the location of a voxel and  $A(x, \omega)$  is the directional occlusion along direction  $\omega$  which is described by  $A(x, \omega) = \sum_{t=0}^T M(x + t\omega)$ . Thereby,  $T$  is the number of samples along direction  $\omega$  and  $M(x)$  is a visibility mapping function of a sample  $x$ . An equivalent



**Figure 2.10:** A hand data set is visualized. Thereby, the vessels should be emphasized as opposed to the bones. (a) depicts the result of a classification based on intensity only. (b) shows the result of a classification based on a 2D TF using intensity and gradient magnitude. (c) depicts a visualization with a size based TF used for assigning different colors on various features. (d) shows a visualization which uses a size based TF in order to map color and opacity. Image from [92].

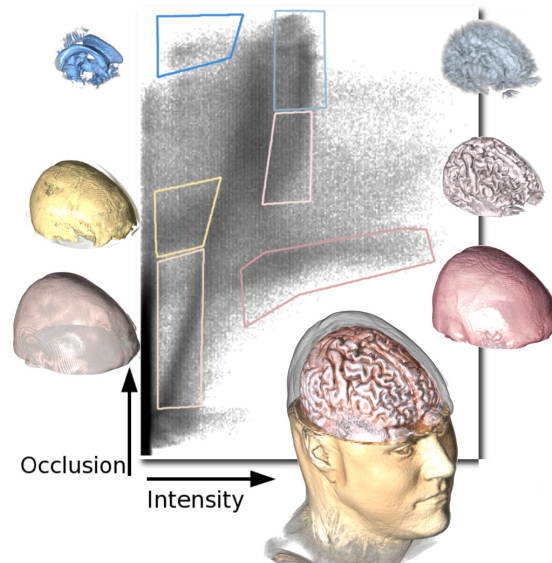


**Figure 2.11:** Ambient occlusion can be calculated via the histogram of the spherical neighborhood of a voxel or by averaging the directional occlusion in various directions around a voxel. Image from [10].

and more elegant way to receive the occlusion for a voxel is to generate the histogram of the spherical neighborhood, to weight it according to the mapping function  $M(x)$  and to calculate the centroid of this histogram. This can be done by  $O(x_0) = \frac{1}{N} \sum_{x \in N_R(x_0)} M(x) = \frac{\sum_i i f_i}{\sum_i f_i}$ . Thereby,  $f_i$  tells us how often  $M(x) = i$  is true. Hence,  $\sum_i f_i$  corresponds to the total number of voxels ( $N$ ) in the spherical neighborhood  $N_R(x)$  of voxel  $x$  and radius  $R$ .

After this step, each voxel has an occlusion value which is determined by the voxels neighborhood. This value can be - and most likely it is - different from voxels which contain the same intensity but belong to a different structure with a different neighborhood. This is what subsequently allows the separation of regions of interest with the same or overlapping intensities. However, there are two more things that must be obeyed during this step. First, it is important that the radius of the neighborhood is larger than the searched structures. Otherwise the ambient occlusion of a voxel would not encode any information of a features surrounding and it would occlude itself instead. Second, there are several possible visibility mappings  $M(x)$  which influence the separability of structures. After that, the occlusion spectrum can be calculated which is the basis for the TF generation. In Figure 2.12 an example for the occlusion spectrum of a MRI data set can be seen.

In addition, there is another problem that may occur. As already mentioned, there exist several possible visibility mappings  $M(x)$ . Each of them may maximize the separability of structures in some



**Figure 2.12:** The occlusion spectrum of a MRI data set. The location of a certain structure within the spectrum depends on its intensity and surrounding defined by the ambient occlusion of the voxels contained in the structure. Image from [10].

intensities and hinder it in others. Hence, the mapping must be chosen adaptively for each intensity interval of interest. Therefore, the occlusion spectrum is calculated several times for each intensity interval of interest using different parameter settings of the visibility mapping. For each occlusion spectrum of an interval, the clusters, which represent structures in the volume, are calculated. Then the variance of means is determined for each set of clusters belonging to one interval and a certain set of parameters. Finally, the mapping with the largest variance is kept for each interval. It provides the best separation of structures.

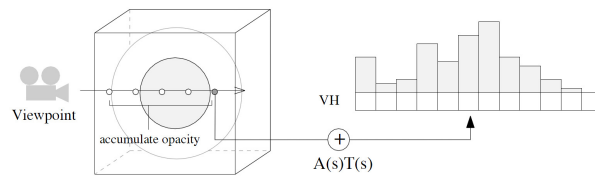
Summing up, the occlusion spectrum is a very useful approach which is able to distinguish structures of overlapping or same intensity but varying neighborhood. In comparison to gradient magnitude approaches like from Kindlmann and Durkin [33], the occlusion spectrum highlights structures instead of boundaries which is beneficial in a wide range of applications. However, the user must take care to use a sufficiently large voxel neighborhood. Moreover, the occlusion spectrum must be calculated several times using varying visibility mappings, to be able to select the optimal solution for different intensity intervals within a data set. Thereby, the user is responsible for defining the intensity intervals of interest.

#### 2.1.4.2 Visibility Histograms and Visibility-Driven Transfer Functions

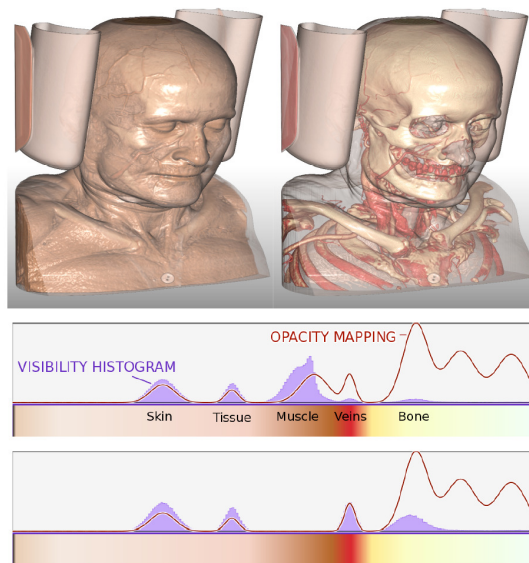
Correa and Ma [11] present a visibility based approach. It assists users in designing TFs which provide a good visibility on interesting features for a certain viewpoint. Visibility histograms (VHs) form the foundation of this approach. They depict the contribution of a data set's scalar values to the final image for a certain viewpoint. A VH in its simplest form is a histogram of visibility versus intensity. Instead of the intensity any other kind of scalar field value can also be used. The VH can either be used as a feedback mechanism assisting the user in manual TF specification or as a part of a semi-automatic method, which aims to maximize the visibility of important structures.

In order to create a VH, the visibility  $T(p)$  for each sample and the TF defined opacity  $A(X(p))$  of each classification value (i.e. intensity) is required first.

**Calculating the Visibility:** A sample's viewpoint depended visibility  $T(p)$  is expressed by the accumulated opacity of this sample  $p$  to the eye position  $E$  and hence conveys the opacity contribution of



**Figure 2.13:** The steps involved in calculating the VH: First, the accumulated opacity for a sample is calculated. It is subsequently used to weight the original opacity of a regarded sample. The resulting value is added to the corresponding histogram bin. Image from [11].



**Figure 2.14:** Demonstration of the expressiveness of visibility histograms. The VH (purple plot), the opacity function (red line) and the two corresponding renderings are depicted. The upper plot belongs to the left image. Here the VH shows that the bone and veins tissue is occluded by muscle tissue, which is not obvious when regarding the opacity function on its own. This information allows the user to adopt the TF accordingly, resulting in the lower plot and the corresponding right image. Image from [11].

this sample to the final image. It can be denoted by the formula  $T(p) = e^{\int_{-p}^E \tau(t) dt}$ . Thereby,  $\tau(t)$  is a samples attenuation coefficient which is usually given by the opacity TF.

**Calculating the Visibility Histogram:** As soon as the visibility for each sample has been determined the VH can be created by weighting the sample's opacities using their visibility and adding the resulting values to the corresponding bins of the histogram. This weighting is the main difference to a usual data histogram in which samples are weighted uniformly. Given visibilities and opacities, the contribution of each sample point  $p$  to the VH is expressed by  $VH(x) = VH(x) + T(p)A(x)$ , where the left and the right  $VH(x)$  express the new and the old value stored at bin  $x$ , respectively.  $x$  denotes a certain value within the range of values of the scalar field and is a bin index. A depiction of the VH calculation is shown in Figure 2.13.

**Manual and Semi-Automatic Transfer Function Design:** Correa and Ma [11] suggest a manual and an automatic approach for TF design based on visibility. In the first case a graphical depiction of the VH serves as a feedback mechanism. The VH is constantly updated when viewpoint or opacity function are changed. The application can best be explained on the example in Figure 2.14. Here the red

line describes the opacity TF while the purple plot describes the VH. The x- and y-coordinates describe intensity and opacity values, respectively. The upper plot corresponds to the left image and shows that the bones are occluded by muscle tissue and are barely visible as a result although the opacity TF gives more importance to the bones. However, if the opacity of the occluding muscle tissue is reduced, the bones can be seen better as in the lower plot and right image.

Manual TF specification, even with feedback, can be very difficult, since minor changes in the TF may lead to great changes in visibility. Hence, Correa and Ma [11] additionally suggest an automatization of the TF design process by transforming it to an energy minimization problem. Thereby, the quality of a TF is determined by an energy/cost function which depends on certain parameters. It must be minimized with respect to the energy components user satisfaction and visibility. User satisfaction means, that the system generated TF must be as similar as possible to an initial user defined TF. It is calculated with the square difference between original and system TF. The second criterium on the other hand demands that the visibility of important voxels must be as high as possible in order to minimize this part of the energy function. It is expressed by the negative sum of visibilities which are calculated using the system TF, and are weighted by the according opacity of the original TF. Additionally, the energy function depends on certain constraints which assure reasonable values of the parameters determining this function.

**Extensions:** In a constitutive work Correa and Ma [12] extend their previous approach by several facets. First, they extend the VH to multiple dimensions. The general notion of the definition of an arbitrary n-dimensional VH is given by  $VH[x_1, \dots, x_n] = VH[x_1, \dots, x_n] + T(p)A(x_1, \dots, x_n)$  where  $(x_1, \dots, x_n) = X(p)$  is a vector of classification values for a point  $p$ . Correa and Ma provides an example based on the classification values intensity and gradient magnitude.

As a second extension, two types of viewpoint independent VHs are introduced which provide the user with visibility information of important features, irrespective of the viewpoint. This is important since it frequently may be of interest to design TFs which allow a good visibility on certain structures independent of a viewer's position. For the first type - omni-directional visibility histograms (OVH) - VHs are calculated from a number of viewpoints which are placed on a circumscribing sphere or cylinder. These VHs are then summed up to a total VH, the OHV. It must only be recalculated when a user changes the opacity but not when the viewing direction is changed. The second type - radial visibility histogram (RVH) - can be calculated by less computational costs. This type measures the visibility along radial rays, which requires a previous transformation of volume points from cartesian to spherical or cylindrical coordinates. The drawback of radial visibility functions is that they may misinterpret the visibility of structures, which cannot be seen radially. As a possible solution a hybrid visibility, combining OVH and RVH is introduced.

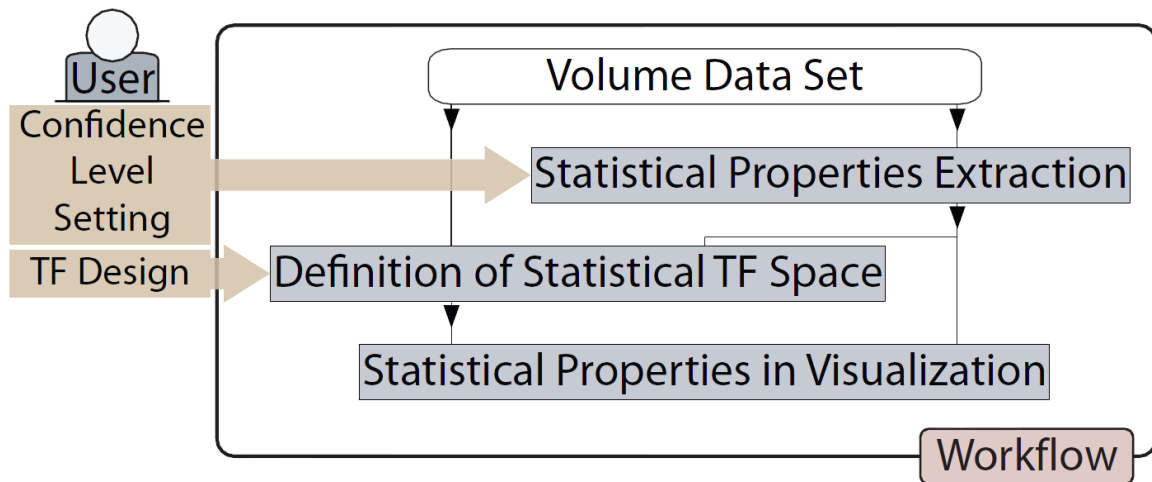
## 2.1.5 Transfer Functions Based on Statistical Analysis

### 2.1.5.1 Volume Visualization Based on Statistical Transfer Function Spaces

Haidacher et al. [21] introduces a TF space based on statistical properties which is especially suitable to separate different features of interest in noisy volume data sets. The statistical properties which define the TF space are also used for semi-automatic classification of different materials - not only borders - within the data set. Moreover, they are applied in the rendering process to improve the visual quality of the resulting images.

Basically, three steps are necessary for this workflow. They are outlined in Figure 2.15. First, the extraction of statistical properties is performed for each sample. Applied to a plot, which uses these properties as axes and defines the statistical TF space, they form clusters. A cluster represents a material or a boundary between materials in the volume data. Based on these clusters meaningful TFs can be designed. As a third step the statistical properties are utilized within the rendering process to improve the visualization results.

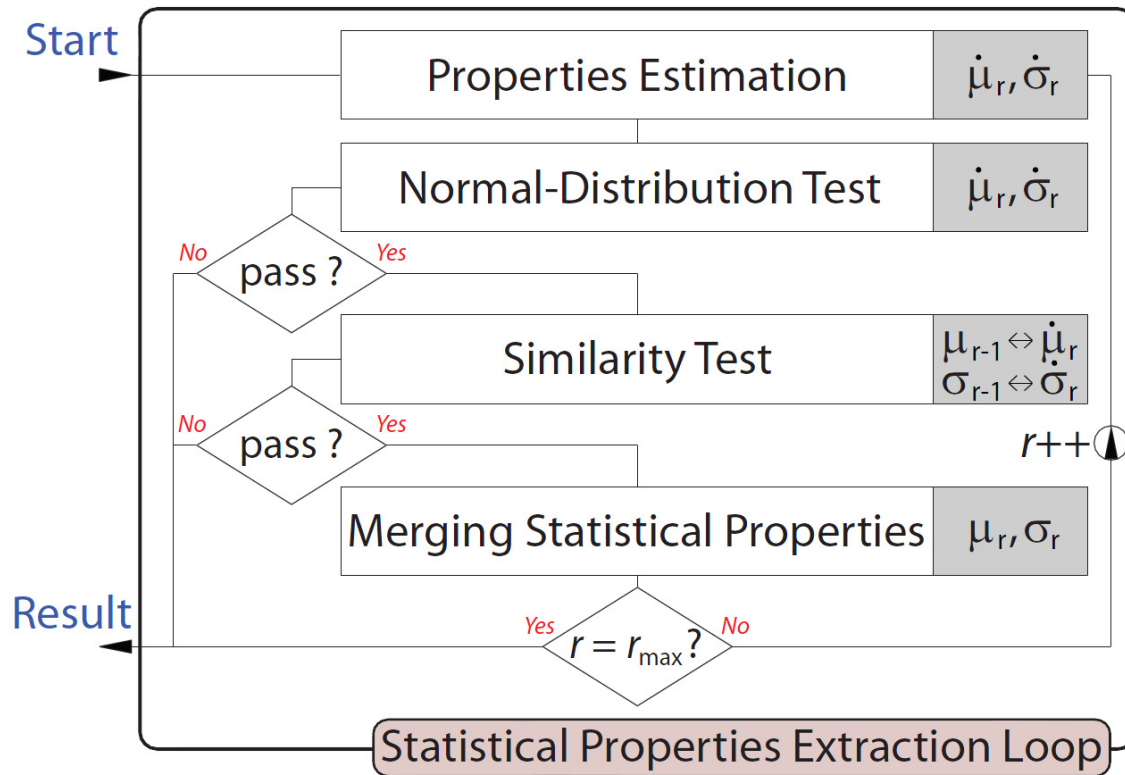




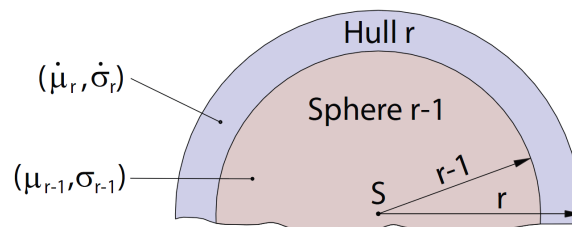
**Figure 2.15:** Statistical TF workflow. Image from [21].

The first step is done as a preprocessing step and performs a semi-automatic extraction of the statistical properties for every sample point of the volume data set. The used statistical properties are mean ( $\mu$ ) and deviation ( $\sigma$ ) of the data values in a spherical neighborhood around a sample. The main issue of this step is to find the right radius for the sphere so that it ideally encloses a homogeneous region. To achieve that, the extraction loop in Figure 2.16 may have to be traversed several times. Initially, the sphere starts with a default radius of one. Then, for each iteration the spherical neighborhood is divided into an inner sphere and a hull. The first contains all voxels within a radius of  $r-1$ . The later contains all voxels within a radius  $r$  minus those within the radius  $r-1$  (see Figure 2.17). As a first step of the extraction loop the mean  $\mu_r$  and deviation  $\sigma_r$  of the hull voxels are determined. A normal distribution test is performed which checks whether the hull voxels belong to one material. If it fails, this means the hull voxels are not normally distributed. In other words, two or more peaks are included in the histogram distribution. In this case, the mean and deviation value of the current inner sphere ( $\mu_{r-1}, \sigma_{r-1}$ ) are kept as statistical properties for this sample. If the test succeeds, a similarity test is performed to compare the hulls and inner sphere's statistical properties. It allows to find out whether the hull belongs to the same material as the inner sphere. Therefore, the so called "Welch's test" [89] is used which requires a user defined confidence level as input. The confidence level determines how similar the compared mean and deviation values of hull and inner sphere must be, to pass the test. For the choice of this value the user should also take the noise within the data set into account. The similarity test is harder to pass for noisy data. If the test fails, hull and sphere do not belong to the same material and once again, the current inner sphere's mean and deviation ( $\mu_{r-1}, \sigma_{r-1}$ ) are kept as statistical properties for this sample. If the test succeeds as well, the hull's and inner sphere's statistical properties are merged, the radius  $r$  is increased by one voxel, and the new iteration starts. In the next iteration the new inner sphere consists of the voxels of the old inner sphere, plus those of the old hull. The new hull contains the voxels within the new radius  $r$  minus those within the new inner sphere. The extraction loop exits when a maximal radius  $r$  is reached or if one of the tests fails.

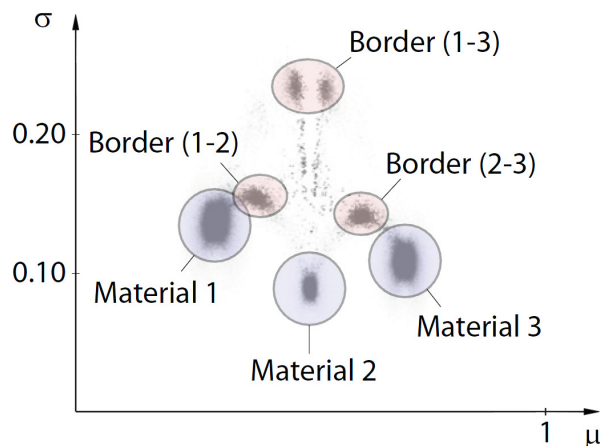
As soon as the mean and deviation values for each sample point are obtained, the statistical TF space can be defined. A 2D plot is created in which the mean (and data value) of each sample point is applied to the horizontal axes while the corresponding deviation values are applied to the vertical axis. As a result, dots representing mean and deviation pairs form a number of clusters as depicted in Figure 2.18. Each cluster stands for a TF region, which represents a material or a border between two materials, and is enclosed by an elliptical area with the center  $(\mu_R, \sigma_R)$ . The color and the opacity of a certain TF region are generally defined by the values  $C_R$  and  $\alpha_R$ , respectively. However, the opacity of a TF region



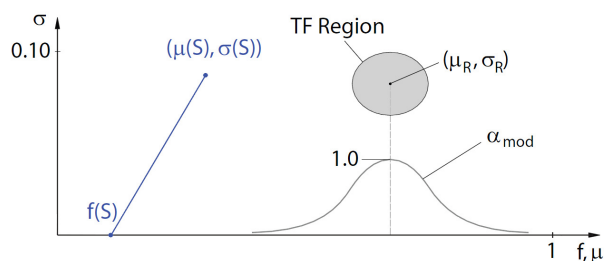
**Figure 2.16:** Loop for the extraction of statistical properties for the data set samples. Image from [21].



**Figure 2.17:** Inner sphere and hull of a sample. Image from [21].



**Figure 2.18:** Dot representation in the statistical TF space. Image from [21].



**Figure 2.19:** On the bottom right of the TF space, an opacity distribution of a TF region consisting of a cluster can be seen. Image from [21].

varies throughout a material according to the formula  $\alpha_{mod} = \alpha_R * e^{-\frac{1}{2}(\frac{f - \mu_R}{\sigma_R})^2}$ , where  $f$  is the data value of a certain sample and  $(\mu_R, \sigma_R)$  the center of the region. The distribution of the opacity according to this formula is graphically depicted in Figure 2.19. It can be seen, that outlier sample points in the TF region, which are more likely not to be part of the region, are assigned lower opacities and hence have a lower influence on the final image. To further enhance this tendency, the opacities of samples can additionally be weighted by the radius  $r_{break}$  at which the extraction loop for a certain sample was stopped. Thereby, a larger  $r_{break}$  means higher opacity. According to this, color and opacity are assigned to single TF regions.

After that, the statistical properties can also be used as a part of the rendering process. The shading process can be enhanced by substituting the original data values of a sample by its mean value to estimate the gradient direction. Moreover, Haidacher et al. [21] uses the deviation instead of the gradient magnitude for shading border regions. According to the authors, these substitutions are especially useful in noisy data sets because mean and deviation are less sensitive to noise than the original data value and gradient magnitude.

### 2.1.6 Alpha Histogram and Partial Range Histograms

The current section presents two methods from Lundström et al. ([55], [54]), which are also using statistical properties resulting from an analysis of the data set in order to generate TFs. However, they differ from most of the above mentioned methods in a significant way. Most of the previously discussed approaches introduce new data values, which are derived from the intensity values of the original 3D scalar field. These are subsequently further evaluated together with the intensity values, for example by

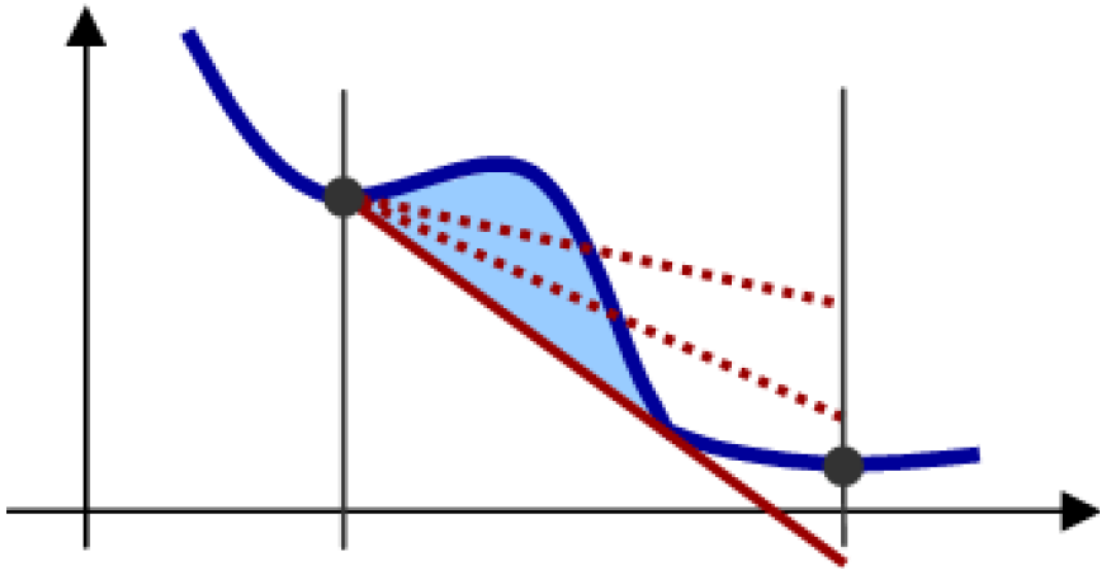
putting them into joint histogram which can be analyzed. The AH and PRH method on the other hand are specialized on analyzing 1D histograms generated from 3D scalar fields. The usually contained data values of these fields are voxel intensities. However, also other values such as occlusion [10], curvature [34], or the statistical properties described by Haidacher et al. [21] can be considered.

### 2.1.6.1 The Alpha Histogram: Using Spatial Coherence to Enhance Histograms and Transfer Function Design

According to Lundström et al. [55] the construction of TFs would be easy, if features of a data set would appear as clearly visible peaks in the corresponding histogram. This would mean, that the value ranges of the regions of interest would be easy to find. Subsequently, it is easy to assign different optical properties to the different value ranges and receive an appropriate image with clearly separated regions. However, in histograms minor features of data sets often happen to be covered by noise. Moreover, two features may consist of materials with overlapping intensity ranges. This leads to undistinguishable feature peaks in histograms. A major reason for this is the loss of all spatial information concerning the features in the data set, in usual histograms.

At this point, the AH applies by incorporating required spatial information and hence exposing value ranges of spatial coherent materials. It is generated by summing up amplified local histograms. The exact steps required for the construction are described in the following:

1. Divide input data set: The first step is to divide the input data set into local regions. These can also be overlapping. The appropriate size of local regions depends on the size of the data set and the features in it. Special shapes and sizes are reasonable if any a priori information about the shape of interesting structures is given. Otherwise a cubic standard shape can be used. This step can formally be represented by  $D = \bigcup_{i=1}^k N_i$  and  $N_i \cap N_j = \emptyset, i \neq j$ , where  $D$  is the volumetric data set,  $N_1 \dots N_k$  represent distinct regions and  $k$  stands for the number of spatial regions.
2. Generate local histograms: The next step is to generate a local histogram for each of the regions in the data set. Each local histogram extends over the full intensity range comprised by the input data set. The creation of local histograms can formally be expressed as  $H_n(N, x) = |N \cap D_x|$ , where  $N$  is the set of data values of the current neighborhood,  $D_x$  is a set of those data values which are equal to  $x$  within  $D$  and  $H_n(N, x)$  is the local histogram of the neighborhood  $N$ .
3. Amplify local histograms: In a third step, the created local histograms are amplified by an alpha value ( $\alpha$ ). Thereby, the frequency value of each bin is raised to the power of  $\alpha$ . The alpha value must be larger than one. This means, that high frequency-values are amplified stronger than low ones. A large alpha value adds to this effect. Too large values do not contribute any further improvements to the results, since a normalization is performed later on.
4. Accumulate amplified local histograms: Subsequently, the amplified local histograms are summed into the so called AH. Bins of different local histograms which are representing the same intensity value are accumulated.
5. Normalize accumulated histogram: Finally, the AH is normalized by raising the frequency value of each bin to the power of  $1/\alpha$ . This results in a histogram, which covers a smaller area than the original one. The reason for this can best be explained if we consider a bin of the original volume histogram, whose frequency is distributed on the corresponding bins of two local histograms: Assume  $c$  is the frequency of a bin in the original histogram.  $a$  and  $b$  are the frequencies of the corresponding bins in two local histograms, so that  $c = a + b$ . Furthermore,  $\alpha$  is the alpha value by which the local histograms are amplified. Then,  $\sqrt[\alpha]{a^\alpha + b^\alpha} < c$  applies. For example, if  $a = 3$ ,  $b = 4$ ,  $c = 7$  and  $\alpha = 2$ , then  $3^2 + 4^2 = 25$  and  $\sqrt{25} = 5$  while  $3 + 4 = 7$ . Therefore, a second normalization can be performed to make the AH cover the same area as the original histogram.



**Figure 2.20:** A peak area enclosed by the peak shape (blue) and a baseline (red). Image from [55].

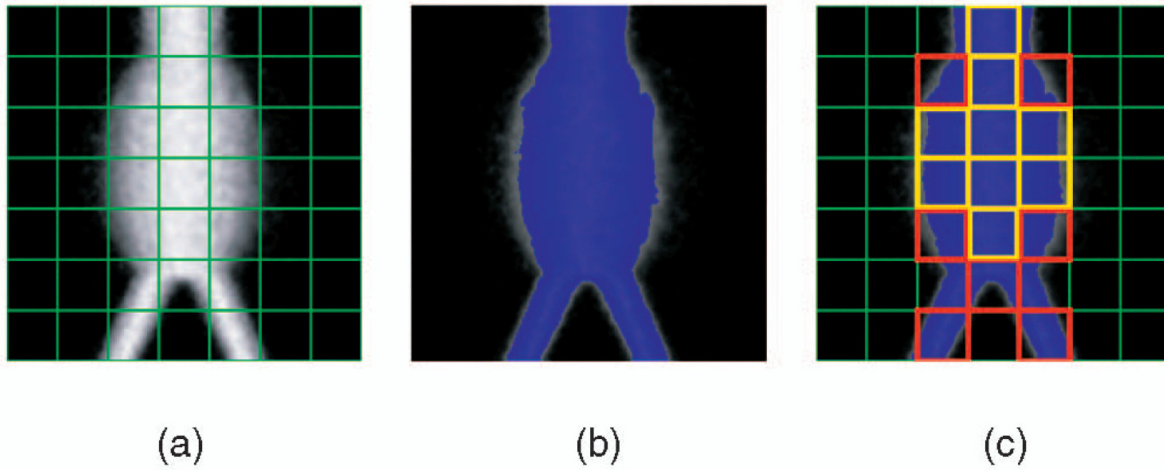
Steps three, four and five until the first normalization can be summarized by  $H_\alpha(x) = (\sum_{i=1}^k H_n(N_i, x)^\alpha)^{1/\alpha}$ , where  $\alpha$  is the exponent by which the local bin frequencies are amplified and by which the amplified and summed bin frequencies are normalized. The second normalization is described by  $\tilde{H}_\alpha(x) = \frac{|D|}{\sum_x H_\alpha(x)} H_\alpha(x)$ , which means, that the total number of voxels (frequencies) from the original data set is divided by a summation of the frequencies of all bins in the AH. The resulting factor is multiplied with each bin of the AH. This leads to a total frequency in the AH, which is equal to the total frequency in the original histogram.

Apart from varying block sizes and alpha values, the Lundström et al. also suggest some additional variations for the AH, which can be used in order to incorporate a priori knowledge. These include the selective amplification of certain intensity ranges of interest or data set specific neighborhood shapes. Another possible variation is the use of overlapping neighborhoods with a weighting window.

In addition to the calculation of the AH, the discussed paper also includes a peak detection scheme which extracts interesting data value ranges from the enhanced histogram. The peak detection is carried out in three steps. Initially, selective smoothing of minimal peaks and creases is applied until all of them are removed from the AH. Subsequently, the whole histogram is smoothed until a user defined number of peaks remains. Due to the initial smoothing step, the smoothing iterations for the whole histogram are reduced in order to maintain subtle peaks which may be of interest. For both smoothing stages a mask of the form [0.25 0.5 0.25] is used. In the last step a further reduction of the remaining peaks is performed until a second user defined number is reached. The peak covering the least area is removed, while a neighboring peak may be extended to cover the removed peak. Thereby, the area covered by a peak is defined as the area which is enclosed by the peak shape at the top and a baseline which terminates it at the bottom. The baseline proceeds as a tangent from a starting point at the highest valley of the peak to a second point of the peak curve. A valid baseline can be seen in Figure 2.20.

### 2.1.6.2 Local Histograms for Design of Transfer Functions in Direct Volume Rendering

With "Local Histograms for Design of TFs in DVR" Lundström et al. [54] introduce an approach, which allows to integrate information on spatial relations into the TF model. The most important issue of



**Figure 2.21:** Meaning of the weight range for the creation of PRHs. In (a) the image is divided into block shaped neighborhoods, while (b) depicts the pixels within a certain intensity range. In (c) the blocks whose weight range is high enough to be added to the PRH are yellow framed. In these blocks the amount of pixels within a certain intensity range is high enough. Image from [54].

this work is to show, how to detect tissues in an unknown data set. This problem is addressed by an exhaustive multiple peak search, followed by a merging step. As a result, a number of PRHs, defining value ranges of interesting structures in the TF space, are obtained. In general, this approach defines the same problem as the AH method. If peaks belonging to features of a data set would be clearly visible, the corresponding tissue value ranges would be easy to identify. However, unimportant features such as the background frequently place dominant peaks in the histogram which may cover important features with only small peaks. Therefore, Lundström et al. [54] suggest a detection scheme based on PRHs, which tries to find the value ranges of all tissues in the data set. According to Lundström et al. [54], a PRH is the histogram for a set of neighborhoods that are typical for a given data value range.

In order to find the voxels within a certain value range which should be added to the PRH, the data set must be divided into local regions (i.e. cubic blocks of  $8^3$  are suggested). Subsequently, the PRH for a value range is created by calculating the so called weight range for each local region of the data set. The weight range is defined by  $w_r(\phi, N) = \frac{|N \cap V_\phi|}{|N|}$ , where  $N$  is a voxel neighborhood of arbitrary shape,  $V_\phi$  denotes the set of voxels within the range (i.e. the intensity range)  $\phi$  and  $|V|$  is the cardinality of a set  $V$ . In other words, the number of voxels within a neighborhood which are within a certain data value range (for example the intensity range) is divided by the total number of voxels in this neighborhood. After that, for each block whose weight range is larger or equal than a defined  $\epsilon$ , all voxels are added to the PRH. The basic idea behind a PRH is, that it is a histogram that is created for a certain value range (for example of intensity values) of the data set. Not all voxels of the data set which are within this value range are added to the PRH. Only those which are locally concentrated are. Figure 2.21 illustrates the meaning of the weight range on the example of a 2D image.

The exact iterative steps in order to calculate a PRH are described in the following. First, the data set must be divided into local regions and a global main-histogram must be calculated.

1. **Find highest peak of main histogram:** This step is a simple iteration through all bins of the main histogram whereby the index of the bin with the highest frequency must be found and saved.
2. **Fit Gaussian to peak:** Subsequently, a Gaussian curve is fitted to the peak in order to estimate it's width. The result of this step is a value range in the main histogram defined by a mean ( $\mu$ ) and deviation ( $\sigma$ ) value of the current peak.

3. **Create the PRH for middle part of the peak:** Now the PRH for the determined value range is calculated as described before. The weight range for each local region of the data set is computed at first. For each block whose weight range is larger or equal than a user defined  $\epsilon$ , all voxels are added to the PRH. In case the PRH is empty,  $\epsilon$  must be decreased and this step is executed again.
4. **Remove PRH intensity frequencies from main histogram:** In this step all the voxels that have just been added to the PRH are removed from the main histogram. For each histogram index the PRH bin is subtracted from the corresponding one in the main histogram. The removal of frequencies changes the shape of the current main histogram. This may lead to the exposure of new, previously covered peaks which belong to data set features.
5. **Repeat steps 1-4** until main histogram is empty
6. **Merge similar peaks:** The final step is to merge similar peaks since they are very likely to belong to the same tissue. In order to determine the similarity between PRHs, Gaussian curves are fitted to them again. Then, if the following criterion which was empirically found by Lundström et al. [54] is met, two PRHs are merged:  $\sigma_{max}/\sigma_{min} \leq 4$  and  $\mu_{max} - \mu_{min} \leq \sigma_{min} * \max(1, 2 - \sigma_{min}/40)$ . Thereby,  $\mu_{max}$  and  $\mu_{min}$  are the larger and smaller mean value of the two PRHs and  $\sigma_{max}$  and  $\sigma_{min}$  are the larger and smaller deviation value of the two PRHs. This step is also an iterative approach since Gaussians are also fitted to merged PRHs and if possible they are merged again. If no more merges are possible this step is finished.

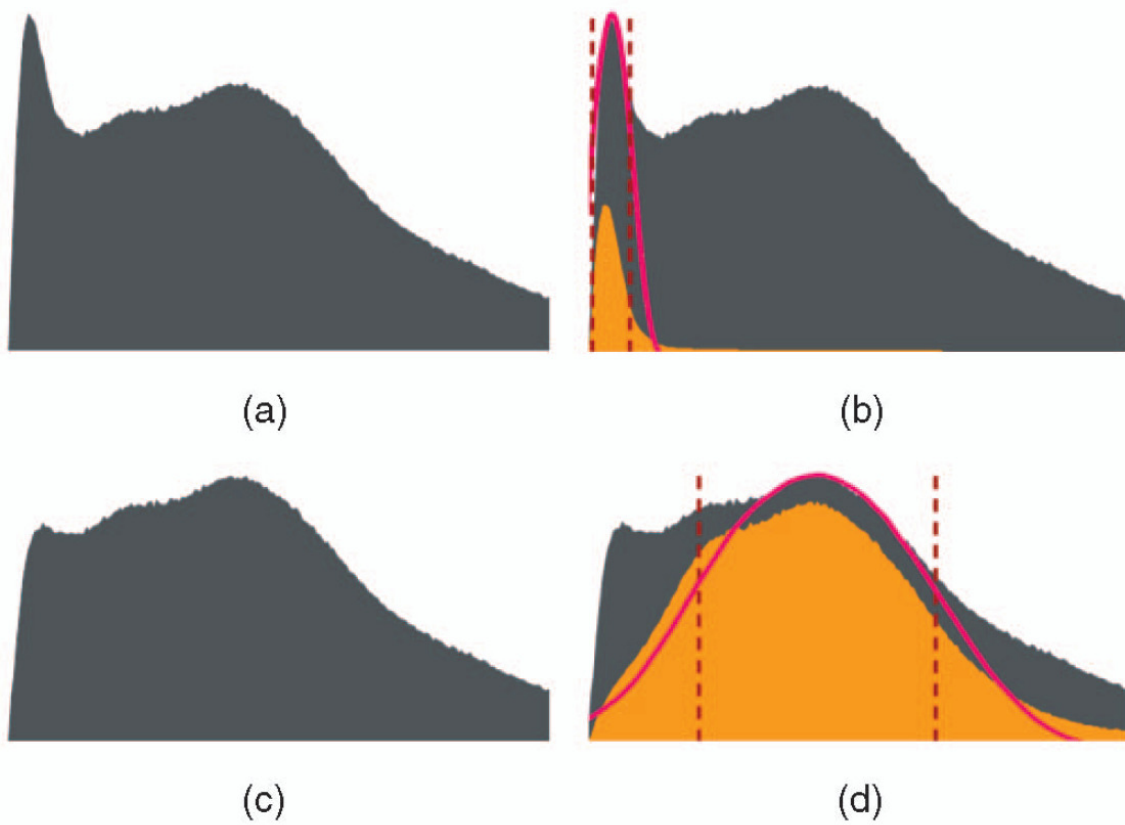
The cycle of steps for removing the highest peaks from the data sets histogram is shown in Figure 2.22.

In addition to the basic PRH identification, Lundström et al. [54] introduce two optional extensions. The first performs a local, block-specific adaption of all tissues intensity ranges, depending on their presence in the surrounding of a considered block. This is especially useful to enhance the resulting PRH, for volumes containing tissues with non-static intensity ranges. The second performs an additional local histogram analysis in order to derive criteria for a better discrimination of overlapping tissue ranges.

### 2.1.6.3 GPU Based Histogram Computation with CUDA

The basic approach is to divide a data set into multiple parts and let different CUDA threads process them. Thereby, the question raises for the best location to store the histogram data and how to store it. A simple approach would be to store one histogram in the global device memory. However, reading from and writing to global memory is costly and also inefficient if it is performed non-coalesced. Moreover, a single histogram leads to collisions between writing threads, resulting in large performance drops [48].

In order to overcome this situation shared memory may be used. Podlozhnyuk [70] suggests to create one sub-histogram for each thread of a block. Therefore, the shared memory is divided into a number of banks, which can be accessed simultaneously. Each bank holds the sub-histogram of a thread. As a result, all threads may concurrently and quickly write data values to their sub-histogram in shared memory bank-conflict-free. Subsequently, per-thread sub-histograms are merged into per-block sub-histograms, which are then written to global memory using atomic operations or a certain merging kernel. However, this approach is limited by the size of shared memory. Podlozhnyuk [70] uses a G8x hardware. It consists of 16 Stream Multiprocessors (SM) each at 16KB shared memory and 768 threads at the most (as well as 8 thread blocks) [35]. For efficient execution they recommend an average of 192 threads per block. Partitioning the shared memory to this amount of threads only allows to create one sub-histogram of 64 one-byte bins per thread. Single-byte bins limit the data size to be processed by one thread to 255 byte (255 is the highest frequency number a bin of one byte can hold). This in turn allows a maximum data set size of approximately 3MB to be computed ( $16SM \times 768threads \times 255byte \approx 3MB$ ) without reading and writing new data from and to global memory. In order to mitigate this limitations, Podlozhnyuk [70]



**Figure 2.22:** Cycle of peak detection and initial histogram adjustment. (a) shows the initial histogram. In (b) the PRH resulting from the blocks with a weight range high enough is depicted in yellow. (c) shows the new histogram which remains after the PRH has been removed from the initial histogram. (d) shows the PRH for the new highest peak. Image from [54].



introduces a second method which only creates one sub-histogram per warp. On a GeForce 8 a warp forms a union of 32 threads at most. Only one warp is executed on a SM at a time. Depending on the number of warps per thread block, significantly larger histograms can be generated. Podlozhnyuk [70] use sub-histograms of 256 bins and provides 4 bytes for each bin. However, the data sets used for AH and PRH calculations frequently comprise data value ranges of 3000 to 4000 gray-scale values. Moreover, this approach introduces intra-warp shared memory collisions, which must be handled. This is achieved by tagging a histogram bin "[...] according to the last thread that wrote to them" [70]. The distribution of the data processed by a warp determines the amount of collisions and thus the performance of this approach. In the worst case all threads of a warp sequentially write to the same bin.

In order to allow the calculation of histograms with larger numbers of bins, Shams and Kennedy [79] suggest to partition data value ranges in to sub-ranges and run the algorithm based on per-warp sub-histograms multiple times. For each iteration only data elements within the currently considered sub-range are processed. The performance of this approach drops with increasing numbers of bins. Moreover, the throughput of this method still depends on the distribution of the processed data. Alternatively, Shams and Kennedy [79] introduce a second approach, which generates a sub-histogram per thread and subsequently combines them into a single one, similar to the 64 bin histograms from [70]. However, they are stored in the global device memory. Using this method, sub-histograms can be updated collision-free and the algorithms performance is independent from the data distribution. Large numbers of histogram bins can be handled within a single interaction. In order to avoid inefficient reading from and writing to global memory, Shams and Kennedy [79] buffer bin values in the shared memory and only update corresponding bins in the global memory when those in the shared memory overflow. In order to work efficiently, this approach requires an appropriate trade-off between the number of used threads and the number of bin bits for intermediate storage. Larger numbers of threads entail smaller numbers of bits for single bins and require more global memory updates. A smaller amount of threads on the other hand may not utilize the GPU resources to capacity. For increasing numbers of bins this method also entails heavy performance losses.

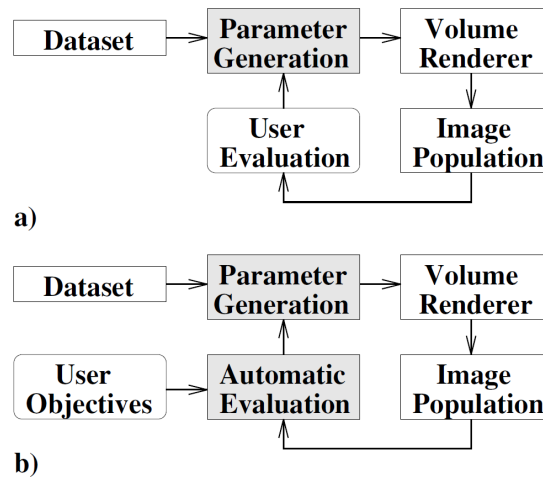
#### 2.1.6.4 Notes on Using Histograms for Transfer Function Design

When regarding various statistical methods used for TF design, it is obvious that histograms are repeatedly used as a tool for analyzing data sets. However, Carr et al. [6] claim that histograms have certain drawbacks which have been overseen for a long time. They demonstrate that histograms represent spatial function representations which are equivalent to nearest neighbor interpolation, which is known as one of the worst interpolants. In addition, they suggest to use iso-surface statistics as a way to remedy defects of histograms. Later Scheidegger et al. [76] present a corrected formulation which is based on weighting statistics with the inverse gradient magnitude. According to [76] the insights of the paper are also especially interesting for approaches such as from Lundström et al. [54].

### 2.1.7 Further Transfer Function Approaches

#### 2.1.7.1 Spatialized Transfer Functions

In the paper "Spatialized TFs", Roettger et al. [74] present an approach which performs segmentation in the TF domain by encoding spatial information into multi-dimensional TFs. Intensity and gradient magnitude are used as TF dimensions, however an application for other dimensions is possible as well as shown by Wesarg and Kirschner [91]. Hence, as a first step, a 2D histogram of intensity versus gradient magnitude is created. Subsequently, a distance norm is calculated for each entry in the histogram. It describes the average position of all voxels contained in the currently regarded entry, in the data set. After that, entries with a similar distance, which are assumed to belong to the same region of interest are detected. Finally, voxels belonging to entries with a similar average position are assigned a common color.



**Figure 2.23:** Image (a) depicts the traversed TF design process when the user evaluates the intermediate results, while (b) shows the process including an automatic evaluation based on user defined criteria. Image from [22].

This approach distinguishes itself from other methods due to the fact that it assigns colors to features, based on spatial information. Therefore, it uses the color channels as additional TF dimensions, without adding three further dimensions to the actual TF.

## 2.2 Image Centric Approaches

Image centric approaches analyze rendered images in order to find a proper TF. They frequently start with random or user given TFs which are used to create initial thumbnails. The resulting images are analyzed in order to rate their quality. Based on that, the underlying TFs are adopted. After multiple iterations the TFs evolve, so that they allow to render a set of thumbnails depicting interesting regions.

### 2.2.1 Generation of Transfer Functions with Stochastic Search Techniques

He et al. [22] introduces an approach which assists users in developing appropriate TFs for the visualization of volumetric data sets, using stochastic search techniques.

Basically, a number of randomly or user generated TFs form the starting point of this algorithm. A library of typical TFs exists, which can initially be edited by the user. Each TF is represented by an array of floating point numbers and all of these vectors are stored in a vector again. These initial TFs are used to produce initial images which can be rated by the user by assigning a fitness value (pick or don't pick in the simplest case). Based on this rating and the applied stochastic search technique, an intermediate set of TFs is chosen from the complete previous set. Every sample of every chosen TF contained in the intermediate set has a certain chance to be mutated. In order to keep the TF smooth a Gaussian filter is applied during the mutation operation. In addition to the mutation, there is also a certain chance that previous TFs are recombined using a crossover operator. The new set of mutated and recombined TFs are subsequently used to create corresponding pictures, using a renderer once more. These images are rated by the user again.

Besides letting the user rate the resulting intermediate images by watching them, this approach also allows to specify objective criteria, such as image entropy, by which the determination of the intermediate set of TFs is influenced. The basic cycles traversed for user and automatic evaluation of intermediate results are depicted in Figure 2.23.

The main advantage of this approach is, that it allows the user to determine appropriate TF for datasets without working in their parameter space. Instead the user can focus on the resulting images. By evaluating them, good TFs are further evolved. After several iterations, this leads the user to a final set of appropriate images in a straight forward way. On the downside, the initial TFs for this approach must be selected by the user. These, however, have a significant impact on the required time for finding satisfying results.

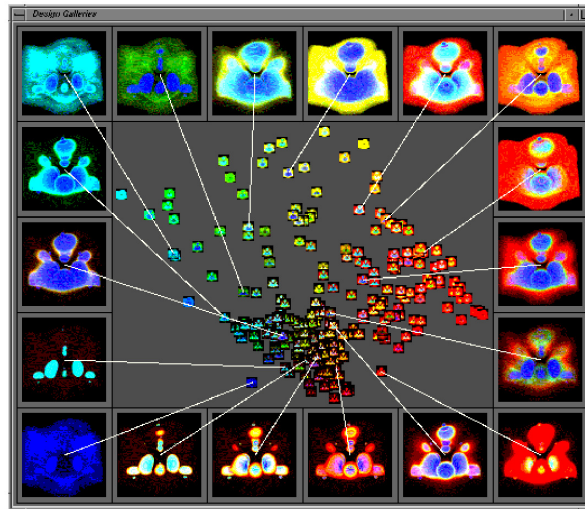
### 2.2.2 Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation

Marks et al. [57] introduces a general approach for parameter tweaking in computer graphics which can also be utilized for finding appropriate TFs. Thereby, the broadest selection of perceptually different thumbnails (and their TFs) for a data set are presented to the user within a so called design gallery. These thumbnails are automatically generated and organized.

In general, the following six key elements are required in the design gallery system: input vector, mapping process, output vector, distance metric, dispersion method and arrangement method.

The input vector contains a number of parameters, which determine the output (i.e. an image) depending on a mapping process done by a renderer. An output vector on the other hand contains a number of values, which are subjective relevant qualities to describe the output. A distance metric is used to determine the similarity between output graphics by calculating the distance between their corresponding output vectors. A dispersion method is responsible for finding a set of input vectors which lead to well distributed appropriate output graphics. Finally, an arrangement method is responsible for organizing the output graphics resulting from the dispersion method in the design gallery, so that they are easy to explore by the user.

In coherence with TF design Marks et al. [57] provide an exemplary description of the six elements as described in the following. For a data set with intensity values from 0 to 255 the opacity TF is defined by eight control points which are connected to a polyline. Each control point can be modified in terms of its position within the intensity range and in terms of the opacity. Hence, each control point has two degrees of freedom. This results in the first 16 values, which are put in the input vector. Further, input vector values result from a color TF. The mapping process is as usual performed by a renderer. The output vector contains 24 values, which result from the YUV values for eight manually selected representative pixels from a data set. A small number of pixels is sufficient for dispersion, since they - if appropriately chosen - are representative for different regions of interest in the data set. Moreover, this small set of pixels leads to much less computational cost in comparison to whole image based dispersion. In order to determine the similarity between output graphics, the euclidian distance is used as a distance metric. It is computed between the output vectors of two output graphics. For dispersion, an evolutionary strategy is applied in this example. Thereby, a random set of input vectors, their corresponding output vectors and the number of iterations (evolutionary steps) are required as input. Then the function perturbs a randomly chosen input vector and uses it as a substitute for an existing input vector which is worse. An input vector is inferior to another one, if the average nearest neighbor distance of its output vector to all other images output vectors is lower than the average distance of the second considered output vector. The worst input vector corresponds to the output vector with the lowest average distance. This means, that this input vector leads to an image which is very similar to many other output graphics. Therefore, it is not representative and can be substituted by a better one. After all iterations, a broad selection of perceptually different thumbnails and their TFs has been created. As a last step, these thumbnails must be arranged in the design gallery. The arrangement is achieved by applying a multi-dimensional scaling approach [24] on a matrix of distances computed from the output vectors (or full size images) of the output graphic. An example of the resulting design gallery can be seen in Figure 2.24. Each of the six elements could also be different to a certain degree, even when used for finding TFs. For example, one could use less control points or only allow to vary them in opacity.



**Figure 2.24:** Marks' design gallery giving an overview over a well arranged selection of thumbnails resulting from many different color and opacity TFs. Image from [57].

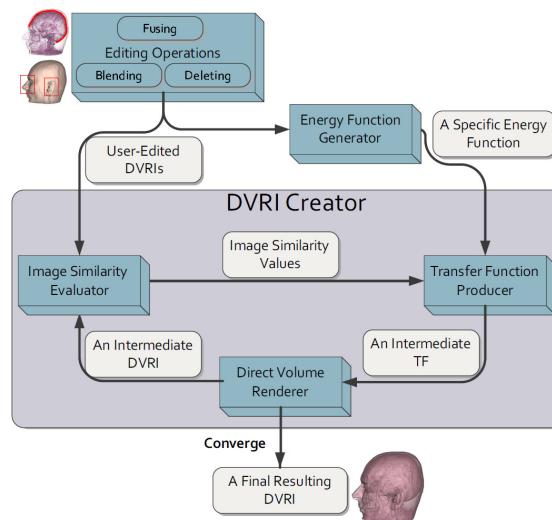
The great advantage of the described method is the provided overview over a large set of thumbnails expressing certain features of the data set in different variations depending on the underlying TFs. This allows the user to select preferred depictions of features from the design gallery. However, it may still take same exploration effort to navigate through a large amount of preview thumbnails. Moreover, the system does not provide any simple means to adapt created TFs. Another drawback of this method is that the output vector used for evaluation must be selected by the user. This requires the user to know representative regions in the data set in order to select pixels from them.

### 2.2.3 Interactive Transfer Function Design Based on Editing Direct Volume Rendered Images

Another image centric approach for TF design which uses stochastic search techniques is introduced by Wu and Qu [100]. They create an intuitive, general and robust framework for editing direct volume rendered images (DRVIs) which can also be utilized to design TFs.

Their approach is based on the idea, that TFs and their corresponding direct volume rendered images, which are created with methods such as from Kindlmann and Durkin [33] or Marks et al.[57], frequently deliver only partially satisfying results. For example, DVRI may contain too much context which the user would like to remove. Moreover, several regions of interest may be depicted in separated DVRI - however their spatial relationship can only be captured if they are combined into a single comprehensive image. As a result, the Wu and Qu's framework takes DVRI and their TFs resulting from other methods as input. Subsequently, it allows to edit them in terms of fusing multiple features from different DVRI into a comprehensive one, blending two DVRI and deleting undesired features within DVRI. The problem related to editing is considered as optimization problem.

An overview of the system architecture is shown in Figure 2.25. In order to perform editing the user must initially select a certain operation, choose the source DVRI which should be involved, and mark the regions of interest. The latter can be done with a rectangle, provided by the user interface, or by using semi-automatic feature selection tools. Then a score from 0 to 1 must be assigned to each DVRI which are the expected similarity values between source and target DVRI. Subsequently, the system starts to search for the target DVRI based on the similarity values. At first an energy function is created depending on the desired editing operations. It is handed to the TF producer and will be used to evaluate intermediate solutions. Subsequently, a certain amount of intermediate TFs are created by the TF producer using stochastic search techniques. Wu and Qu [100] apply genetic algorithms for this purpose.



**Figure 2.25:** System architecture and overview of the involved steps in the editing process of DVRI. Image from [100].

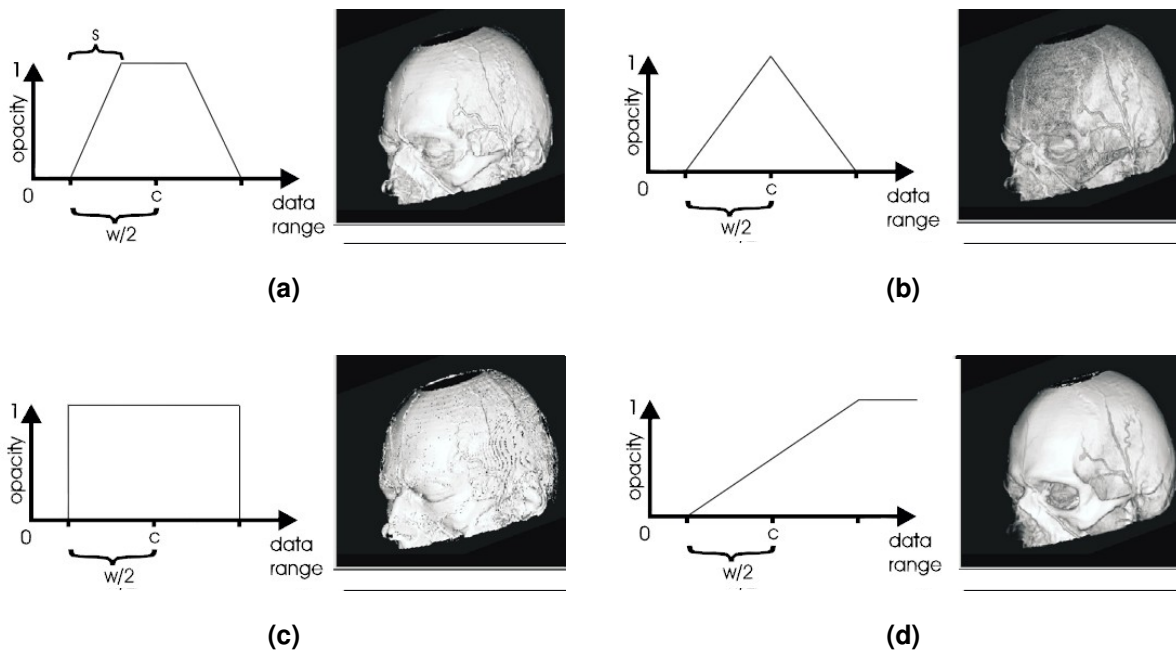
Then, the renderer creates intermediate DVRI from the corresponding TFs. The intermediate DVRI in turn are passed to the image similarity evaluator which calculates the similarity between each intermediate DVRI and the source DVRI. The resulting values are sent to the TF producer. This deploys the similarity value in the energy function in order to determine the energy value of each intermediate TF. These are subsequently used as an indicator for the appropriateness of the corresponding TFs. Based on it inappropriate TFs are removed and better ones are created for the next cycle. Thereby, smaller energy values stand for better TFs. As soon as the search algorithm converges a final DVRI is generated as an output by the system.

## 2.3 Transfer Function Design Specific User Interfaces

### 2.3.1 Mastering Transfer Function Specification by Using VolumePro Technology

König and Gröller [40] introduce a user interface in order to facilitate the specification of an appropriate TF. Thereby, the TF design process is basically divided into three sequential steps. Initially, regions of interest describing peaks are chosen. Subsequently, a color is defined for each peak. In a final step colored peaks are combined into a single TF with certain opacities for each peak. Thereby, settings from earlier steps can be adjusted without dropping settings from later ones. For example, the extend of a region of interest can still be modified after its color has been chosen.

In the first step data value ranges containing certain materials from the data set and the contribution (an initial opacity) of each data value within the range are specified. Intensity values of the voxels are used as scalar values. However, regions of interest can also be defined by every other kind of TF domain such as those using gradient magnitude or principal curvature. The contributing data value ranges are referred to as peaks. The way a peak contributes to the visualization depends on its shape (i.e. trapezoid, box, tend or ramp) which can be chosen by the user. A peak is defined by the peak center ( $c$ ), peak width ( $w$ ) and slope width ( $s$ ), which is only required for trapezoids (see Figure 2.26). The most important part of the user interface for this step is a bar, which by default depicts the whole range of data values (see Figure 2.27 (a)). On this bar, peaks are shown by plotted markers within this range. A zoom can be used to concentrate on certain parts of the value range. In the background the histogram over the value range is depicted to facilitate the search for features. In order to specify the peaks, König and Gröller

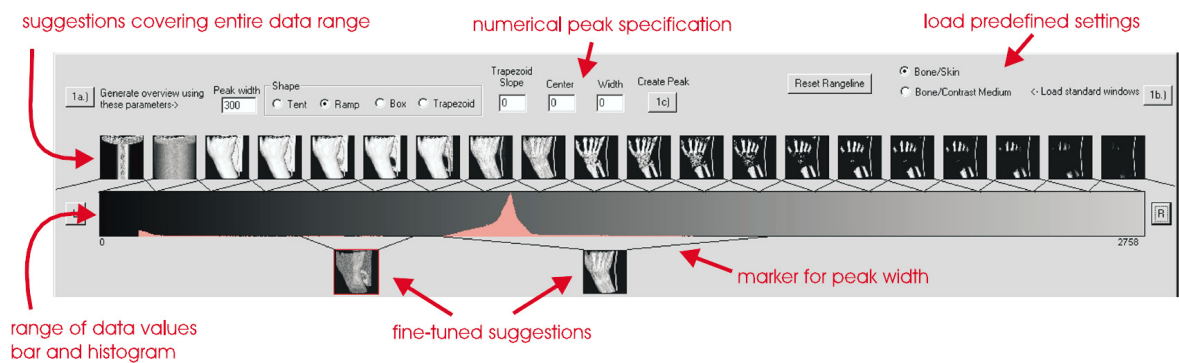


**Figure 2.26:** Different shapes which can be used to represent a peak (region of interest) within a TF lead to differently rendered images. All of them are described by a peak center ( $c$ ) and a peak width ( $w$ ). The trapezoid shape is additionally described by a slope width ( $s$ ). In this Figure (a) shows a trapezoid, (b) a tent, (c) a box and (d) a ramp shape. Images from [40].

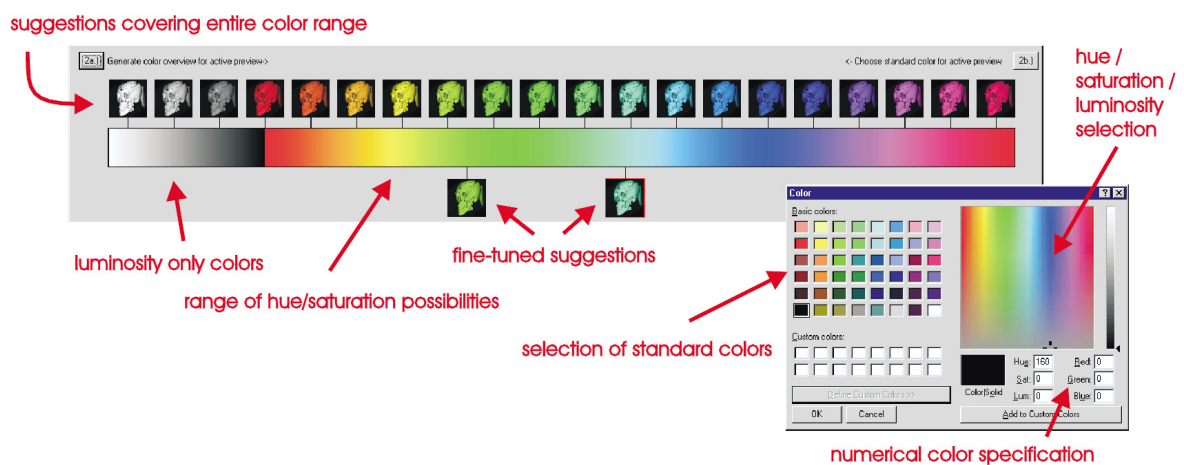
suggest several techniques. First of all, a number of preview images based on peaks covering the whole data value range can be created. Peaks which already fit, can be used as TF, or otherwise selected for fine-tuning. Then the center and width of the peak can be modified by mouse gestures. Other possibilities are to define peaks by standard values (useful to display certain combinations of materials) or to simply enter numerical values for the peak center and width. The repeated application of the techniques allow the user to create a set of peaks, visualizing the desired features.

In a second step, the color for each of the peaks can be specified. Therefore, a selection of colors is applied to the currently regarded peak, which is selected by the user. The corresponding thumbnails are subsequently depicted in the user interface. For each thumbnail a line indicates the peaks color position on a color bar. This bar is representing the hue and saturation domain of colors (see Figure 2.27 (b)). If none of the suggested colors exactly suits, the users can select the thumbnail with the best color and adopt its hue and saturation by mouse. If the user is already sure about the desired color, it is also possible to assign it by "dragging and dropping". Alternatively, a color editor such as in Figure 2.27 (b) can be used to choose colors. By using these approaches for each of the preview images, an appropriate color is obtained for each value range.

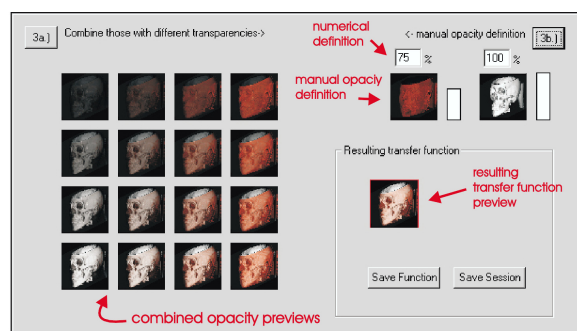
The last step helps to obtain an opacity for the peaks with already assigned colors which are combined into a common TF. This is necessary to generate an image which depicts the structures of all peaks. A number of suggested opacity combinations are created for the selected "colored peaks". For only two peaks, König and Gröller [40] use a 4x4 matrix of combined thumbnails with increasing opacity of peak one in horizontal direction and increasing opacity for peak two in vertical direction (see Figure 2.27 (c)). If none of the suggested combinations fit, the opacity of each peak can also be defined manually by modifying an opacity bar to the right of the considered peaks or numerically by entering the percentage of complete opaqueness (see Figure 2.27 (c)). The manual definition can be done from the scratch, or starting with the values of a suggested combination.



(a)



(b)



(c)

**Figure 2.27:** Overview of the user interface components of Königs' TF design system. It contains one component for each of the three TF specification stages it involves. (a) shows the user interface component for the first TF specification stage. It allows to specify data value ranges of regions of interest. (b) shows the user interface component for the second TF specification stage. It allows to specify the colors for the peaks by providing several possible color suggestions and enabling manual fine tuning. (c) shows the user interface component for the third TF specification stage. It allows to combine the colored peaks, by providing several possible opacity combinations and enabling manual fine tuning. Images from [40].

This approach provides a user interface which combines a stepwise interactive TF specification with suggestions for each phase, provided by the system. By that, the user is guided to desirable TFs and corresponding rendered images. However, the identification of data value ranges, the choice of their colors as well as their opacities is still in the hands of the user, although the framework facilitates this process due to the provided guidance.

### 2.3.2 Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets

Kniss et al. [42] identify three main issues which impede the specification of TFs. The first two are the large number of degrees of freedom of TFs as well as the lack of guidance and constraints in interfaces for setting TFs. The third problem is, that TFs are inherently non-spatial, which makes the isolation of features with similar data values difficult. Further they assert, that multi-dimensional TFs mitigate the third problem but further impair the first and second one. Kniss et al. [42] consider this as a conceptual gap between the spatial and TF domain. Moreover, they assert, that the previously mentioned approaches ([40], [57], [22]) are only suitable for 1D TFs.

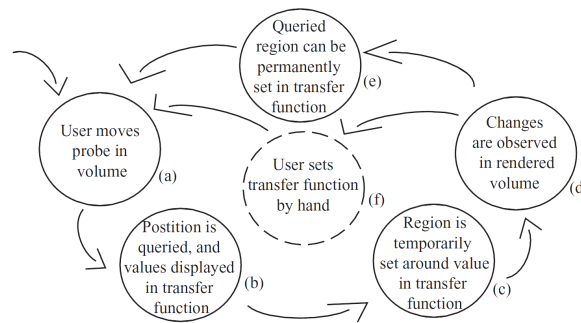
Therefore, the authors introduce a set of direct interaction widgets. They allow intuitive and convenient specification of 3D TFs based on data value, gradient magnitude and a second directional derivative following the work of Kindlmann and Durkin [33]. Conventionally, a TF is defined by setting and modifying a number of control points and watching the impact on the rendered images. The presented widgets allow to reverse this process by setting the TF directly in the spatial domain. Thereby, the interaction simultaneously affects both the TF and spatial domain. This lessens the conceptual gap between them by giving the user cues about how they are related. This concept is called *dual-domain interaction* and can best be understood by considering Figure 2.28. Initially, a data probe widget is used (a) to query a position within the data set defining the spatial domain. Immediately, the data value, gradient magnitude and second derivative associated with this position are depicted in the TF widget (b). The part of the TF which is defined by the three values at the probe position is temporary and automatically highlighted by a high opacity value (c). This temporary change of the TF automatically affects the rendered image (d) and allows the user to see whether the selected position belongs to an interesting region within the data set. If this is the case, the temporary TF can be kept permanently (e). Instead of keeping the automatically generated TF, it is also possible to simply use the feedback (identified ranges in the TF space) to adopt the TF manually (f). This process can be repeated until the total TF is suitable.

This approach allows a simple and intuitive interactive adoption of TFs. It is well suited to adopt a current TF exactly to the users needs. It does not perform any automatic detection of interesting features but allows to take precalculated images and corresponding TF from approaches such as from Kindlmann and Durkin [33]. Moreover, it does not specifically target the problem of giving an overview of the resulting images of different TFs. However, users frequently want to have an overview of different depictions of various features, such as provided by Marks et al. [57]. This is of special interest if the user is not very familiar with the contents of the data set.

## 2.4 Selected General User Interfaces

Many of the image centric approaches described above include a special graphical user interface in which rendered thumbnails are arranged (i.e. Marks et al. [57]). They also provide certain interaction techniques which allow to navigate through the rendered thumbnails and adopt them by changing their TFs. In addition, there exist a number of user interfaces which were specifically developed for the use in TF design (i.e. König and Gröller [40], Kniss et al. [44]). However, there are also approaches from other research areas which may be utilized for TF design. They can be extended to a powerful framework to arrange, modify, and combine TFs due to interaction with their corresponding rendered images.





**Figure 2.28:** Basic concept of dual domain interaction. Image from [42].

Since two such approaches are of interest for the system presented in Chapter 3 and 4 their background is described in the following.

### 2.4.1 Data Mountain: Using Spatial Memory for Document Management

Robertson et al.'s [73] DM is a document management system based on a 3D spatial layout which allows users to arrange and navigate through large amounts of documents. The DM is defined as a tool for managing web site bookmarks and is compared to the Internet Explorer (IE4) favorites mechanism.

Thereby, the basic workspace consists of rectangular planar surface which is tilted at  $65^\circ$  and covered by an arbitrary texture. It serves as a passive landmark. The surface also serves as an underground for thumbnails representing documents which can be freely arranged by users. Thereby, the bottom of each thumbnail touches the DM (see Figure 2.29 (a)). A fixed viewing position and a perspective view is used. In order to place documents on the mountain initially, their corresponding thumbnails are located in a preferred viewing position (see Figure 2.29 (b)). From there the user can drag the thumbnails to the desired location. The position of the thumbnails and therefore the way they are arranged and grouped can also be changed by dragging. The dragging movement happens continuous and is constrained to the DM surface. Additionally, a page avoidance behavior is applied, which is especially important for the dragging action. This means that a minimum distance between all thumbnails is always kept, no matter whether they are currently moved or not. If a thumbnail gets too close to others during dragging, they are temporarily displaced in order to maintain the minimum distance. If displacements cause other thumbnails to get too close to each other further shifts may take place. This kind of page avoidance behavior has mainly prevailed for two reasons. On the one hand the user can continually see which thumbnail positions would result from a current termination of dragging. On the other hand, a thumbnail can never be entirely occluded.

By simply clicking on a thumbnail, it is moved to the preferred viewing position by a slow-in/slow-out animation. A further click in this position allows to either open the document or put it back to its previous position on the DM. In addition to the animations, the described interactions are also accompanied by various different audio cues in order to reinforce visual cues.

Robertson et al. [73] consider the main advantage of the DM in its capability to leverage natural human capabilities. First, a human's *pre-attentive ability to recognize spatial relationships* allows to take advantage of the third dimension in order to display more thumbnails without entailing an extra cognitive load. In addition, to providing the user with the benefits of a 3D user interface, the DM can still be navigated by 2D interaction schemes. This is achieved by fixing the viewpoint and constraining thumbnail movements along the DM surface. Moreover, the authors assume that the DM exploits the spatial memory of humans in a 3D environment. This process is mainly supported due to the free and



**Figure 2.29:** (a) The DM consisting of a tilted and textured plane with 100 thumbnails. (b) The DM with a thumbnail in the preferred position. Images from [73].

very personal arrangement of thumbnails on the DM. It is further facilitated by the page avoidance behavior, animations and audio cues which accompany each movement. The advantages of the DM have been underlined with a user study which compares it with the IE4 favorites mechanism. The study revealed significant advantages of the DM.

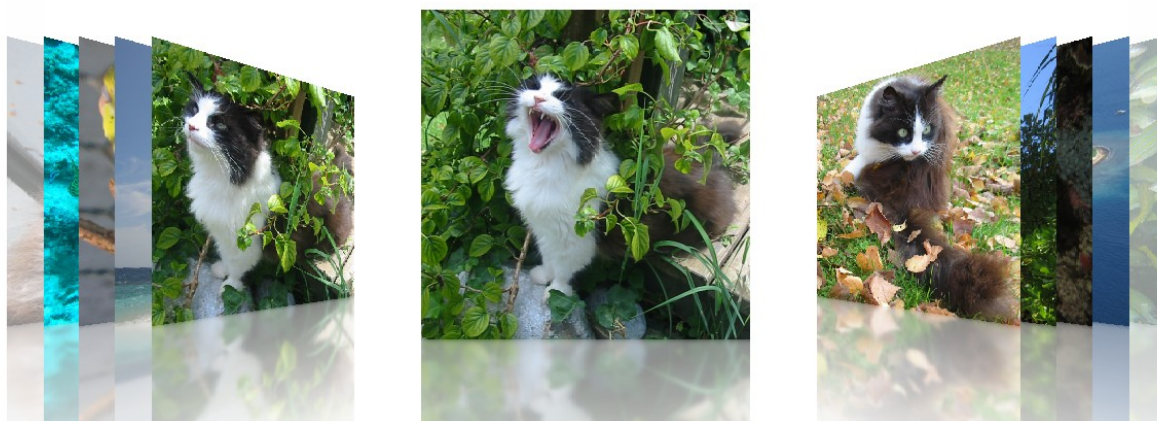
However, the publication of the DM entailed a debate about the advantages of a third dimension in user interfaces. An interesting user study from Cockburn and McKenzie [8] evaluated differences between a 2D and 3D version of the DM for storing, organizing and retrieving tasks. Interestingly, their results did not reveal significant differences for mean task times. The results stayed the same as the amount of thumbnails has been increased. Initially, it was expected that the 3D interface provides more powerful and natural schemes in order to deal with large numbers of thumbnails. A possible explanation are problems of the users with visual matching of smaller thumbnails located towards the top of the DM. In general users have accessed the right areas of the screen in retrieval tasks very fast, but had troubles as soon as they had to rely on visual matching to accurately identify the target thumbnail. However, user significantly preferred the 3D version in subjective assessments. Also note that the page/thumbnaill avoidance behavior, which contributed largely to the rating of the DM in the study from Robertson et al. [73], was not included in this study. A later study from Tavanti and Lind [85] revealed that 3D interfaces increase the performance in spatial memory tasks.

After all, the DM offers many qualities which seem to be of interest for a user interface for TF design. It allows the user to perform very personal arrangements in three dimension and additionally supports the spatial memory due to textures used as passive landmarks. Thereby, the tilted plane allows a good overview over a large amount of thumbnails. In addition, the simple 2D interaction scheme can simply be extended to allow operations required for TF design. A way to use the DM for this purpose is presented in Chapter 3 and 4.

## 2.4.2 Cover Flow: Animated Graphical User Interface for a Display Screen or Portion Thereof

A further user interface which is well suited for the interaction with large numbers of thumbnails is called CF. Its is defined as "*animated graphical user interface for a display screen or portion thereof*" [7].

It can basically be imagined as a stack of horizontally arranged images which the user can thumb through [17]. From all the thumbnails in the stack only some can be seen on the screen.



**Figure 2.30:** A series of photographs depicted in the CF clone Picture Flow [23]. It allows users to easily flip through the stack of images.

From the displayed thumbnails there is one centered, which is straight in front of the user. To the left and the right of the centered one, there are a number of horizontally tilted thumbnails which allow the user to see what comes next and what he just saw. A depiction of a series of photographs in the CF clone Picture Flow can be seen in Figure 2.30. By a certain user input, depending on the device the CF technique is used on, one can switch to the thumbnails to the left or right of the center. This process is depicted by a fluent animation.

CF was initially developed by the American artist Andrew Coulter Enright and subsequently implemented by the Macintosh developer Jonathan del Strother. In 2006, CF was bought by Apple and is since then utilized as a part of numerous applications.[17], [18] Recently Apple also has been granted a design patent on CF, with the number D613,300 [7]. Besides the short textual description of the patent which is in the first paragraph of this section, Apple used a number of pictures to depict the patent design and the animation to switch between images. Some representative images from the patent are used to explain the steps included in an animation for switching to the next image in Figures 2.31 (a-e).

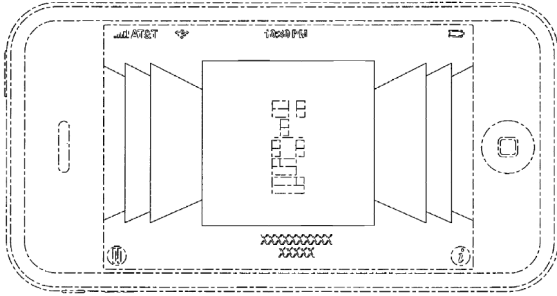
Currently CF is for example used in iTunes to navigate through music album covers or in the Macintosh Finder to scan through the items contained in a folder. These items can be music files with a related cover image as well as previews of text and PDF documents from which the first page is used to be depicted. Irrespective of the application and depicted contents, the great advantage of CF can be seen in the possibility to flip through a large amount of images very fast and to see several adjacent pictures at the same time. [63]

Accordingly, CF is also interesting for the exploration of a selection of DVR thumbnails representing different TFs. It can be used as a user interface to navigate through the DVR images and to subsequently delete, modify or combine a selection of interesting ones. In chapter 3 and 4 a CF implementation, adapted to meet the needs of a gallery for TF design, is presented.

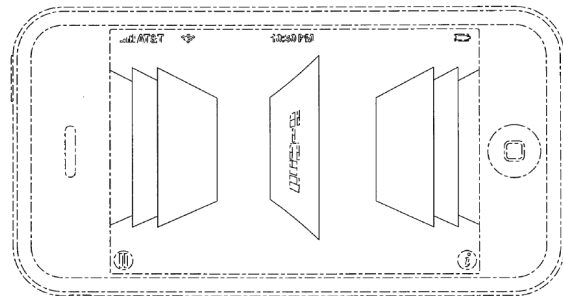
In the meanwhile there also exist a number of other implementations such as Image Flow [17] or Picture Flow [23] which are inspired by the original CF design. The latter is an implementation of CF using Nokias cross-platform application framework Qt and serves as the bases for the extended implementation described in chapter 3 and 4.

## 2.5 Handling the Drawbacks of Multi-Dimensional Transfer Functions

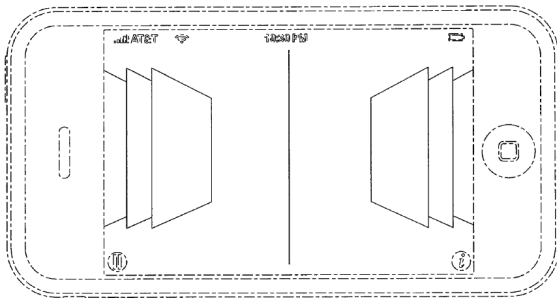
Many of the above described approaches for TF design are based on more than one data value. However, multi-dimensional TFs also incorporate a number of drawbacks which must be considered.



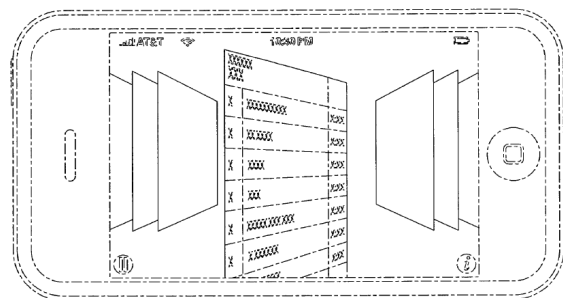
(a) A currently centered picture is still in the starting position, surrounded by several tilted images to the left and right.



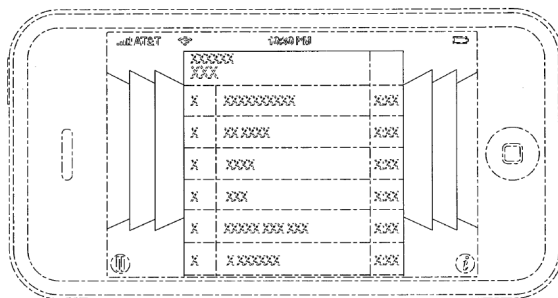
(b) The picture centered before is tilted in the animation in order to be placed to the right, so that the next picture from the left side can be placed in the center.



(c) The animation is still in progress.



(d) The upcoming center image is on the way to be moved to its new position, while the previous center image is already placed on the right side of the stack.



(e) The new picture finally reached its new position and the animation is finished.

**Figure 2.31:** Explanation of the CF animation for switching to a new image. Figures (a-e) describe the sequence of steps involved in this process. Images from Apple’s CF patent D613, 300S [7].

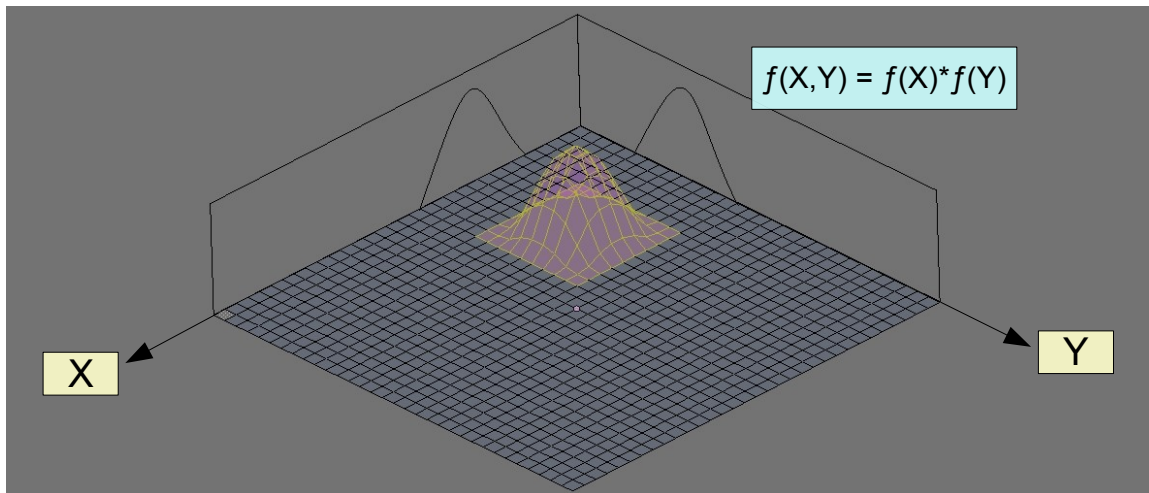
Especially the increased complexity makes the TF design process, which is already unintuitive for 1D TFs, even more complicated [69]. In addition, memory requirements can become an issue when multi-dimensional TFs are stored as textures. Engel et al. [15] suggest that a 3D TF represented by a texture of the same dimensionality may easily require more memory than the volumetric data set itself.

A series of approaches for multi-dimensional TFs are based on dimensional reduction. Pinto and Freitas [69] introduce an approach to facilitate the design of multi-dimensional TFs by transferring the TF design process from a  $nD$  space to a 2D map space. Thereby, a self organizing map is trained using the data sets voxels signatures. As a result of the training, a number of voxel signatures which are representative for data set structures are stored as weight vectors of the self organizing map and are accessible via the coordinates of the 2D maps. The benefit of this approach is that TFs can be designed in a 2D space irrespective from the dimension of the input volume's voxels. The dimensional reduction also entails a loss of information.

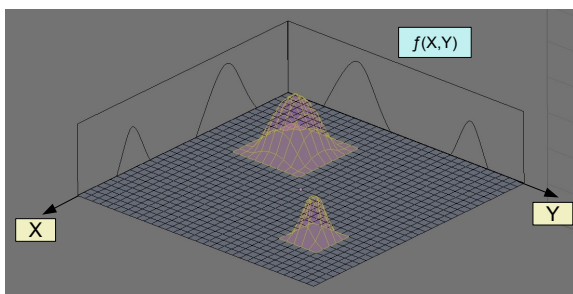
Alternatively, principal component analysis and independent component analysis can be used to obtain a reduced TF domain. Both perform linear transformations to create a reduced TF space that best represents the original one. A drawback of this approach is the unintuitivity of the new mixed space as well as the loss of information which may lead to incorrect mapping results compared to the original TF domain [15]. Examples for approaches based on independent and principal component analysis are demonstrated by Takanashi et al. [84] and Salama et al. [75], respectively.

Another approach are separable TFs. Thereby, a multi-dimensional TF is substituted by a number of lower dimensional TFs. For example, a 4D TF could be substituted by two 2D or four 1D TFs. In order to combine the separate TFs their optical properties can simply be multiplied. This facilitates the handling of high dimensional TFs and reduces their memory requirements significantly. Figure 2.32 (a) shows how two 1D TF in which regions of interest are highlighted by gaussian curves, can be combined to one separable 2D TF. As a result, a gaussian blob defines the region of interest that is emphasized by the separable TF. If each dimension of a separable TF highlights only a single structure, it is easy to obtain the desired combined TF from them. In this case separable and general TF identify the same region. However, frequently each dimension emphasizes multiple regions of interest. In this case there is no simple way to determine which regions of the different dimension belong together. As a result, this approach frequently leads to a false feature classification [44]. Figure 2.32 (b) and (c) illustrate this effect. This Figure compares a general 2D TF that highlights two structures with a corresponding separable TF. Figure 2.32 (b) shows the general 2D TF which clearly emphasizes two regions of interest. There exist unique combinations of  $x$  and  $y$  data values, which can be mapped optical properties. Figure 2.32 (c) shows the corresponding separable TF. As a result of the multiplication of the two 1D TFs, two additional regions occur that are not supposed to be emphasized. It is difficult to avoid this effect, since we do not know which data value regions of each dimensions belong together.

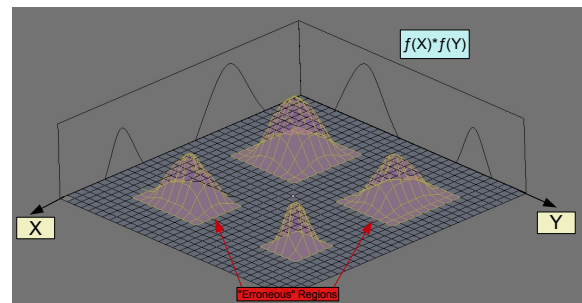
The system presented in Chapter 3 and 4 applies an additional color and homogeneity test in order to circumvent this behavior. Moreover, a preliminary  $nD$  prototype addresses identified weaknesses of the color test.



(a)



(b)



(c)

**Figure 2.32:** Comparison of general and separable 2D TFs. In the depicted figures the horizontal axis defines the the data value ranges, while the vertical axis defines an according optical property (i.e. the opacity). (a) shows that a general and an according separable 2D TF provide the same results, if there is only one feature to classify. (b) shows the result of a general 2D TF classifying two features. (c) shows corresponding result of a separable 2D TF. Thereby, the multiplication leads to two erroneously classified features. Images based on [15].

## Chapter 3

# System Overview and Functionality

The previous chapters discuss the basics as well as the challenges of TF design. As it is the major goal of this work to create an efficient system for TF design, the current chapter introduces a framework combining a number of efficient methods for easy, intuitive and efficient specification of meaningful TFs.

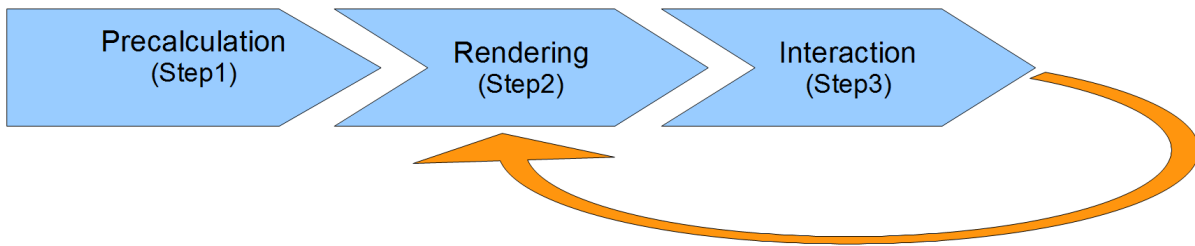
A data centric method, combining the AH [55] and PRH [54] calculation, is employed to prevent tedious trial and error work. In addition, the system includes image centric capabilities: A design gallery allows to depict a selection of interesting data set features, which the user may combine according to his requirements. However, instead of providing an overly large amount of initial depictions as suggested by Marks et al. [57], the user interface allows an individual adaption within a reasonable and well defined scope. In order to make the system as user friendly as possible an intuitive implementation requiring minimal precognitions was aspired. Therefor, two user interface versions based on the approved CF [7] and DM [73] approach are provided by the system. Therein, the user may easily interact with thumbnails representing TFs. In addition, the system is designed to allow a high dimensional classification of features using separable TFs. For their evaluation, a basic approach using a color test to avoid erroneous classifications is deployed. Additionally, a preliminary nD prototype addressing weaknesses of the basic approach is presented.

In the following, Section 3.1 provides an overview of the workflow and interaction of the single system components. Section 3.2 gives an overview of the necessary precalculations of the presented system. Subsequently, Section 3.3 discusses the combined AH and PRH approach, which is part of the precalculation process, in more detail. Section 3.4 explains how TFs are represented, how they are initially setup as well as how they can be adapted and combined. Section 3.5 introduces the single components of the design galleries user interface as well as their interactions and functionality. Finally, Section 3.6 explains the use of separable TFs in the system.

### 3.1 Basic Workflow

The presented system is a design gallery which obtains initial TFs using an automatic data centric approach and provides an efficient user interface for further TF specification. In addition, it uses a high performance rendering system [29] to generate images with different TFs. The whole system is designed in a way that allows to deal with multiple separated TFs concurrently.

The deployed data centric approach is performed as a part of a preprocessing step and delivers value ranges of regions of interest within the data set, which are subsequently used to generate initial TFs. These are passed to the rendering system to generate associated thumbnails. Subsequently, the created thumbnails are arranged in the design gallery and the user may, amongst others, adapt and combine TFs by interacting with the user interface. Whenever a user conducts an operation which changes a TF or



**Figure 3.1:** Basic system workflow: At first the precalculation (step 1) is performed once in order to generate initial TFs. The renderer (step 2) is subsequently consulted to create associated thumbnails which are arranged in the design gallery. By interacting (step 3) with the user interface existing TFs can be adapted or combined to new ones. This constantly requires the renderer to create new images. Hence, there exists a constant interaction loop between the last two steps.

creates a new one, the rendering system is consulted and generates according images. This is necessary to provide immediate visual feedback. In order to create images from initial, adapted or combined TFs the high performance rendering system from Kainz et al. [29] is used.

Considering these explanations, the basic workflow of the presented framework can be described by three steps, whose relation is depicted in Figure 3.1.

- **Precalculation:** This step automatically generates a set of initial TFs using a data centric approach. The user can only influence it by changing settings for the precalculations in advance.
- **Rendering:** The rendering system from Kainz et al. [29], was integrated in the framework. It is used to create images of the data set for certain TFs.
- **Interaction:** Throughout this step, the user may modify and combine TFs as necessary in an easy and intuitive way. This step constantly interacts with the previous one since users require immediate visual feedback on new or adapted TFs.

## 3.2 Precalculation Overview

The first step of the overall system workflow shown in Figure 3.1 consists of a number of precalculations. They are only performed once and form three substeps:

1. The AH calculation (see Section 3.3.1),
2. the PRHs calculation (see Section 3.3.2), and
3. the TF setup (see Section 3.4.1).

The AH and PRH method are combined to form an efficient tool for determining value ranges of data set structures in the TF space. An AH enhances the information content of a volume histogram by incorporating local information. In the presented system it is used as an input for the PRH calculation in order to improve the quality of detected value ranges. Thereby, intermediate AH computation results also required for the PRH calculations are reused in order to avoid redundant computations. However, the PRHs may also be computed without AH support. In addition, there is another essential adaption of the PRH calculation. The merging step as described by Lundström et al. [54] is modified to detect a larger range of different regions of interest by automatically modifying merging criterions. Moreover, an additional check is performed to find a better epsilon value for the allocation of data set blocks to a PRH.



The results of the AH and PRH calculation are subsequently passed to a TF setup scheme in order to create initial TFs. Each of them represents a single data set structure and is generated by determining a certain color and opacity for an according value range. Throughout the whole precalculation process a series of filters are deployed to eliminate redundant regions of interest or TFs.

The identification of value ranges of interesting features constraints the TF space to certain regions and guides users to appropriate TFs and structure visualizations. Based on this, a user may easily adapt TFs in terms in color and opacity and combine them to depict structures in a context. The choice of a data centric approach combining the AH and PRH method has a number of reasons. They are briefly listed in the following and are discussed in more detail in the subsequent section.

- First of all, Lundström et al. ([54], [55]) obtained good clinical results for both, the AH based peak detection scheme as well as the PRH based method.
- The AH and PRH methods are very well suited to be combined since the exhaustive peak search, which is part of the latter, benefits from the distinct peaks provided by an AH. Moreover, both approaches include a number of equal processing steps which have to be performed only once.
- Both techniques may easily be applied for all kinds of data, since they focus on the analysis of histograms, independent of the represented data. Volumes resulting from various medical scanning devices, as well as volumes consisting of derived data values can be used as an input. Due to that this combination is well suited for the concept of separable TFs. An arbitrary number of different volumes (for the same content) can be computed independently and their results combined accordingly to a certain scheme.
- Both methods do not require any user input during execution, although it should be noted that there are a number of parameters which influence their results. However, the default settings frequently deliver satisfying results. Additionally, the system provides functionality to save and load settings. This allows to provide users with a number of predefined settings and to help them to create settings according to their requirements.

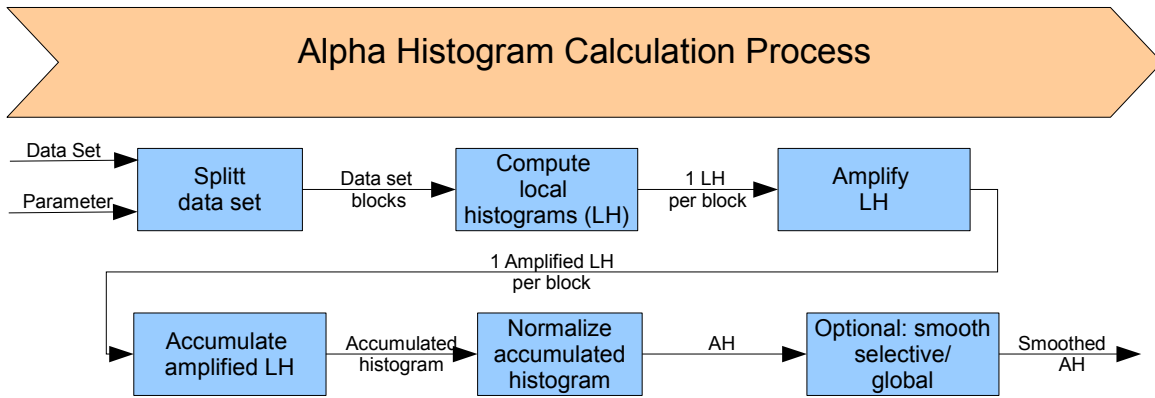
### 3.3 Alpha Histogram and Partial Range Histogram

This section presents an improved implementation based on the AH and on PRHs introduced by Lundström et al. ([55], [54]). AH and PRH calculation are the first two preprocessing steps of the discussed system and provide initial value ranges of regions of interest in the TF space for a given input volume. Each value range is subsequently incorporated into a single TF, according to its position in the value range of the TF space (see Section 3.4.1).

#### 3.3.1 Alpha Histogram

The AH calculation forms the first part of the precalculations. Thereby, an enhanced histogram (the AH) emphasizing local concentrations of the same material is calculated from a 3D data set. It is used in order to improve the results of the PRH calculations forming the next precalculation step. The general steps required for the computation of the AH have already been presented in Section 2.1.6.1. Based on them the following discusses the most important aspects, involved in the single steps of the presented implementation, including the impact of certain parameters. The basic implementation sequence including the in- and outputs of each step are also depicted in Figure 3.2.

**Loading the Volume:** Prior to the actual computations to create an AH, a selected data set must be loaded (the volume data must be read from a selected file). The required functionality is provided by the Insight Toolkit (ITK). The implementation of this step is described in Section 4.3.1.



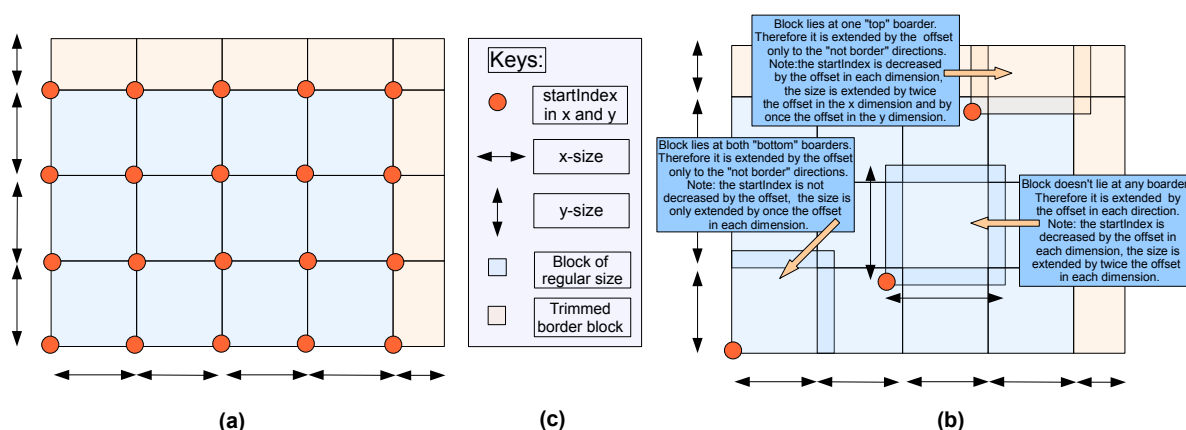
**Figure 3.2:** Overview of the AH calculation process. Thereby, the blue rectangles describe the performed operation of a working step. The arrows between them describe the in- and output of the working steps.

**Splitting the Data Set:** After the volume has been loaded it must be divided into blocks of certain size in order to be able to compute the AH. For the extraction of a certain block its location in the original data set must be defined in terms of a start index and size for each dimension. A detailed description of how a block can be extracted with these parameters, using ITK, is presented in Section 4.3.1.

In order to obtain all blocks, which represent the entire input data set together, the extraction is performed multiple times. In the current implementation a threefold nested for-loop is applied, in which the start index of the desired region is adapted for each iteration (see Figure 3.3 (a)). The size of the blocks on the other hand mainly remains constant. Only the last blocks in each dimension may be assigned smaller sizes to fit in the volume. Another issue is the handling of overlapping blocks which may optionally be used (see Figure 3.3 (b)). The overlap in each dimension is defined by a certain offset number. A special function takes the offsets into account and adjusts a regions start indexes and size depending on its location in the input volume. Basically, there is a distinction between blocks which are at the border of the volume in one or more dimensions and those which are not. For the latter the block is extended by the offset in each dimension. Therefore, the size of the region is increased by twice the offset number in each dimension and the start index is reduced by once the offset in each dimension. Blocks at the volume border are only extended in the direction of their inner sides. Therefore, the size is increased by the single offset for border dimensions. A blocks start index is adjusted in a certain dimension if the block borders the "upper" end of the the data set in this dimension (by once the offset). In order to store the extracted regions, a pointer to each extracted image is saved in a vector.

In conjunction with the current step, the choice of the block size (region size) deserves special attention. It has a serious impact on computation times, memory requirements as well as the resulting AH. For instance, a block size of  $8^3$  applied to a volume with the dimensions  $256^3$  results in  $32^3$  blocks. In contrast a region size of  $6^3$  leads to  $43^3$  blocks, and results in significantly higher memory requirements and calculation times. In this context it should be noted, that the computation time may be reduced to a certain degree by using GPU implementations. The memory requirements, which are frequently more of a problem, however remain. From considering the effort-benefit relation of a GPU implementation we opted in favor of a CPU implementation for both the AH and PRH calculation. A closer discussion to justify this choice can be found in Sections 2.1.6.3 and 4.3.2.

With regard to the quality of an AH Lundström et al. [55] suggests a block size should not be smaller than  $6^3$  in order to properly exploit spatial coherences. In addition, they note that features with scales equal to the block size are especially emphasized in the AH. In the current implementation block sizes of  $6^3$  and  $8^3$  proved to be useful for most data sets. Concerning the block shape Lundström et al. [55] found that anisotropic shapes may be beneficial for data sets containing features which significantly elongate



**Figure 3.3:** Splitting the data set into blocks with and without offset on the example of a 2D image. (a) shows the single blocks into which a data set is split by traversing for loops which adopt the startIndex accordingly. Note, that blocks at the upper boarder of a dimension may be trimmed to fit in the volume. (b) shows how offsets impact a blocks start index and size, depending on its position in the data set. (c) shows the keys for (a) and (b).

in certain directions. However, an according adjustment requires a priori knowledge on the features in the data set. In addition, for a data set containing features extending in different dimensions, the results are impaired. Hence, the current implementation only provides cubic block size, although an adaption to anisotropic shapes could easily be incorporated.

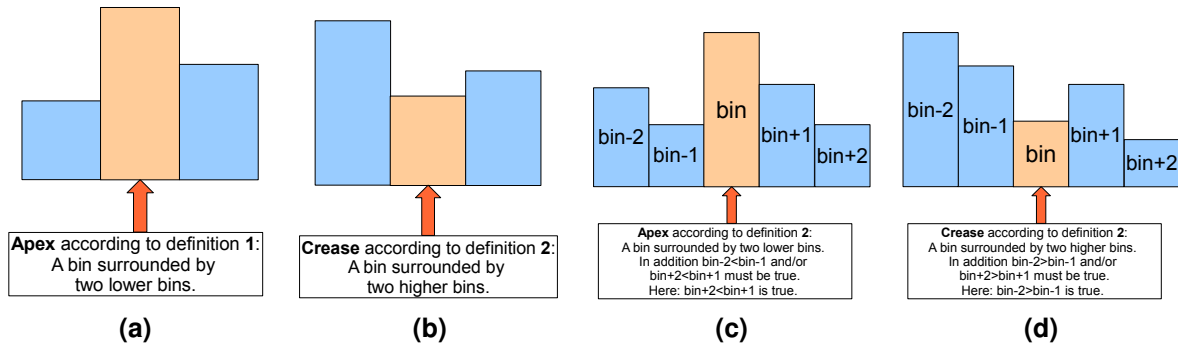
**Histogram Calculation:** After the extraction of the single image blocks, their local histograms are determined. The histogram calculation for each block is performed using ITK functionality and is described in Section 4.3.1.

**Amplification, Accumulation and Normalization:** In order to obtain the AH of a data set three more steps are executed in the following order: amplification of the local histograms, accumulation of the local histograms, normalization of the accumulated histogram. In the amplification step each bin of each local histogram is raised to the power of alpha. In the following step the frequencies of corresponding bins of all local histograms are accumulated and stored in a new histogram. The normalization stage finally raises each bin of the accumulated histogram to the power of  $1/\alpha$ . The second normalization (see Section 2.1.6.1) is omitted, since it does not provide any extra benefits. The result of these steps is an unsmoothed AH.

The only parameter that influences the results of these three steps is the alpha value. Thereby, a high alpha value increases both, the amplification of peaks representing features of interest as well as noise [55]. In the current implementation the default alpha value is three. It may however be adjusted by the user. Lundström et al. [55] also present a series of tests using significantly higher alpha values.

**Smoothing:** In addition to the actual AH calculation, Lundström et al. [55] also present a peak detection scheme. It is based on selective and global smoothing as well as an additional step incorporating small peaks into larger adjacent ones until a predefined number of peaks remains (see Section 2.1.6.1). In the current implementation the last step is omitted, since the AH is used as a means for the computation of PRHs. AH smoothing on the other hand is of interest to obtain distinct and consistent peaks (not jagged), and was therefore implemented. It is especially useful for high alpha values and noisy data sets.

The used filter for both, selective and global smoothing operations is a  $1/4 * [1 \ 2 \ 1]$  mask. As described in Section 2.1.6.1 selective smoothing is only applied to minimum peaks (apex) and creases.



**Figure 3.4:** Presentation of two definitions of apex and crease on the example of a histogram cutout. (a) and (b) show an apex and crease according to definition one (see Figure), respectively. (c) and (d) show an apex and crease according to definition two (see Figure), respectively.

Thereby, it is important to carefully consider the definition of these structures. In this work two definitions are used. The first is equivalent to that used by Lundström et al. [55]: A minimum peak is a bin surrounded by two lower bins, and a crease is a peak surrounded by two higher bins (see Figure 3.4 (a) and (b)). The second definition was tested for performance reasons and is an extension of the first. Thereby, an apex is still a bin surrounded by two lower bins at position  $\text{bin-1}$  and  $\text{bin+1}$ . In addition, there must either be a higher bin to the left of the bin at  $\text{bin-1}$  and/or to the right of  $\text{bin+1}$ . Conversely, a crease is surrounded by two higher bins at  $\text{bin-1}$  and  $\text{bin+1}$  and requires a lower one to the left and/or right of  $\text{bin-1}$ ,  $\text{bin+1}$  respectively (see Figure 3.4 (c) and (d)). For both definitions smoothing is stopped either if no more apex/creases are detected through one traversal of the histogram or if the number of detected apex/creases does not change for a predefined number of iterations.

The second performs better in terms of the required calculation time, since it leads to a significantly reduced amount of matches and iterations until the stopping criterion is reached. For the first definition a reasonable runtime can be obtained by stopping smoothing after approximately 10000 unchanged iterations. At this stage, however, there frequently still exist many apices/creases.

Following the selective smoothing step, global smoothing may be applied. However, this additional step frequently leads to significant distortions of the AH which impair the results of the following PRH calculations. Hence, it will not be considered any further.

**Optimization:** In order to be able to perform an efficient computation of the AH for data sets which are split into a large number of blocks, we optimized the execution order. Instead of performing the steps one (split data set) to four (accumulate amplified local histograms) in Figure 3.2 as a whole - meaning each step must be finished for all input data structures before the next starts - they are sequentially performed for each defined data set region - meaning the steps 2-3 immediately executed after the extraction of a block. The according pseudo code can be found in Listing 3.1. This implementation is very useful if the main memory is not large enough to hold all data set regions and the corresponding histograms. In this case the processing order of the AH steps in Figure 3.2 can be very slow because swapping between main memory and hard disk is required. If the histogram computation, amplification and accumulation is performed immediately after the extraction of a region the swapping requirements are significantly lower. As an example the AH of the case2\_CT data set - which is also presented in Section 5.2.1 - has been computed twice. Once with the optimized and once with the standard implementation. The data set has dimensions of  $166 \times 214 \times 230$ . For a block size of four, which leads to the extraction of 131544 ( $=42 \times 54 \times 58$ ) blocks, the standard implementation required more than 3 minutes, while the optimized solution took approximately one minute.

```

1 // Used data structures
2 startIndex, size;
3 currentBlock;
4 currentHistogram;
5 alphaHistogram;
6
7 //Optimized AH execution order
8 for each block defined by a startIndex and size
9     currentBlock = getDataSetRegion(startIndex, size);
10    currentHistogram = computeHistogram(currentBlock);
11    for each bin of currentHistogram
12        alphaHistogram[bin] += pow(currentBlock[bin], alpha);

```

**Listing 3.1:** Optimized AH execution order in pseudo code. `startIndex` and `size` are two vectors, each consisting of 3 values, defining the extend of the single blocks the data set is split into. `currentBlock` is the extracted data set block whose extension is defined by the current `startIndex` and block size. `currentHistogram` is the histogram computed of the `currentBlock`. `alphaHistogram` is the AH data structure. The AH is continually formed by sequentially adding the amplified bins of the single block histograms.

### 3.3.2 Partial Range Histogram

The computation of PRHs is the second precalculation step and results in a number of value ranges (region of interest) in the TF space, defined by their mean and deviation values. They are subsequently used to set up initial TFs.

Basically, this chapter presents an implementation for the calculation of PRHs as described in Section 2.1.6.2. However, it is customized to allow use of a previously computed AH. Thereby, care is taken to avoid redundant calculations. Moreover, the presented PRHs calculations are adapted to provide a broader range of different regions of interest. Since this extension frequently leads to the repeated detection of equal and/or similar data set features in the TF space we deploy filter to eliminate them. Furthermore, we added an additional check to this precalculation step in order to refine the selection of blocks forming PRHs. Building on the basis from Section 2.1.6.2 the most important implementation issues in the context of this work are presented.

**Preparation:** The purpose of the preparation stage is to provide all required data for the actual PRH calculations. This includes the regular volume histogram, the AH as well as the local histograms of the blocks the data set is divided into. The latter are used to compute weight ranges of data set regions for certain value ranges and to form PRHs. They are required to determine whether a blocks voxels are added to a PRH (see Section 2.1.6.2, PRH calculation step 3). The AH replaces the original histogram for the detection of the highest peak and "fitting a curve" on it (see Section 2.1.6.2, PRH calculation step 1, 2). As a result, it must also be updated for each created PRH (see Section 2.1.6.2, PRH calculation step 4). However, it is not possible to simply subtract PRH frequencies from the AH, since it comprises a smaller and differently distributed area than the regular histogram. Therefore, the original histogram is still required in order to define an appropriate relation between AH and the actual amount and distribution of voxels in the data set. Moreover, the empty volume histogram still serves as a stopping criterion for the exhaustive peak search. In order to define a relation between AH and original histogram, a new vector with one entry for each AH-original histogram bin pair is generated. Each entry contains a value defining the relation between the frequencies of two corresponding bins of both histograms, as shown in the following:

$$relationVector[i] = \frac{alphaHistogram[i]}{originalHistogram[i]} \quad (3.1)$$

Thus, this vector allows to appropriately update the AH, by multiplying PRH bins with the according vector entry, before they are subtracted from AH bins.

The initial data for the further steps can be obtained on three different ways depending on whether the AH is actually calculated and whether certain parameters for the AH and PRH computations correspond:

- In the first case the AH is computed using the same block size as for the PRH calculation and no offset. As a result, the local histograms computed for the AH can simply be passed to the PRH class, as a vector of pointers. This spares the effort of loading the data set, splitting it into blocks and computing the corresponding local histograms. In addition, the AH and original histogram may also be copied from the AH class.
- In the second case the AH is computed using different block sizes as for the PRH calculation and/or an offset. This requires the input data set to be loaded and splitted into blocks as well as to calculate the corresponding local histograms. AH and original histogram may still be copied from the AH class.
- The last case is concerned with the computation of PRHs without a previous AH calculation. It requires all calculations to be performed in the PRH class.

Note: Loading and splitting of the data set as well as the calculation of the local histograms works in the same way as for the AH. However, the volume blocks for the PRH calculations may not and cannot be overlapping [54]. It is important to individually decide for each block whether it is added to a PRH or not.

**Basic Creation of Initial Partial Range Histograms:** The creation of initial PRHs comprises the steps 1-5 in Section 2.1.6.2. In the described implementation they are repeatedly traversed in a while-loop until the original histogram of the data set is empty. Figure 3.5 provides an overview of the steps explained in the following.

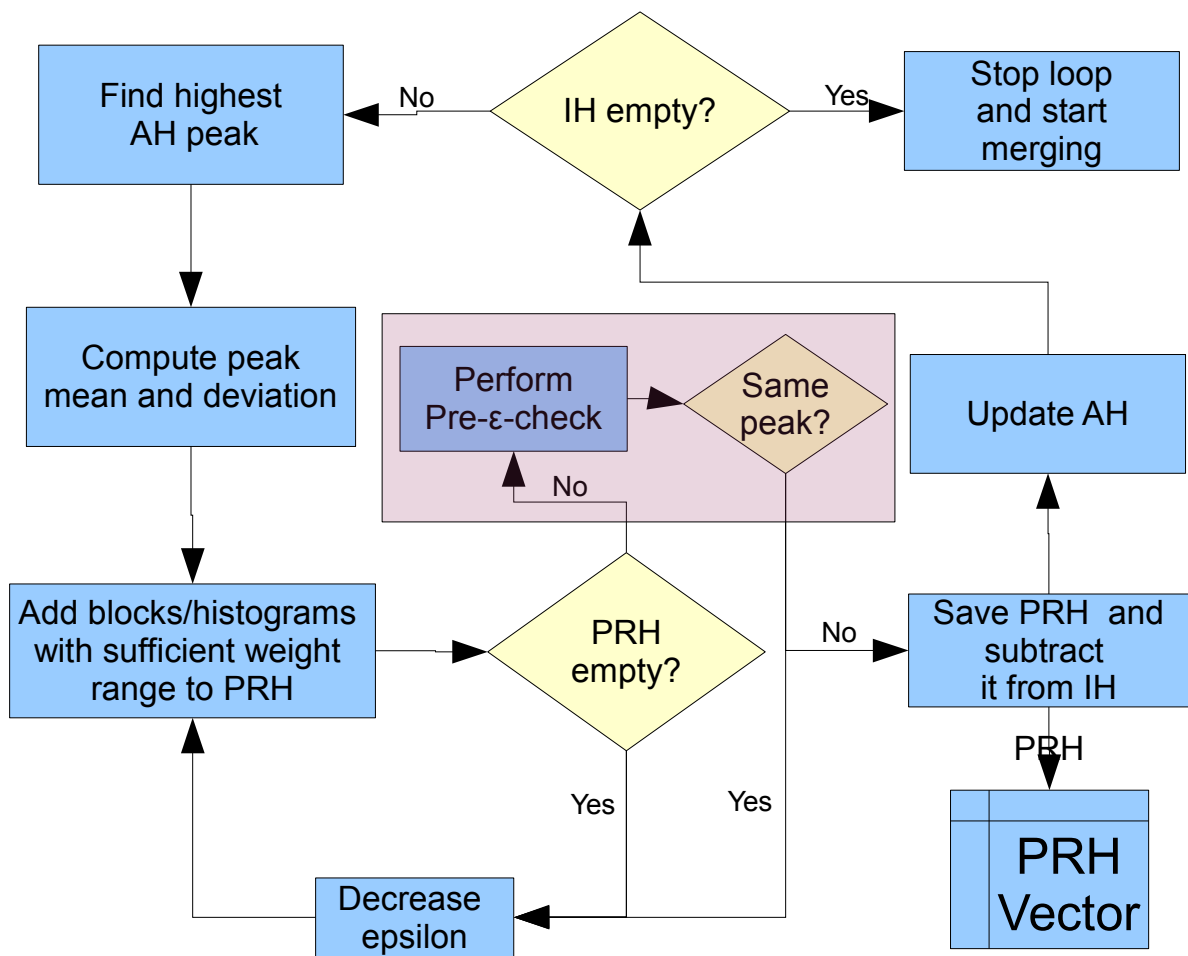
Each iteration of the while-loop starts with the traversal of the current AH, in order to determine and store the index of the bin with the highest frequency. Starting from this position, bins to the left and the right are attended until the lowest point of a valley is reached or the histogram range would be exceeded. From the bins within that range the arithmetic mean ( $\mu$ ) and deviation ( $\sigma$ ) are computed according to equations 3.2 and 3.3 [31]:

$$\mu = \frac{1}{n} \sum_{i=s}^m f_i x_i \quad (3.2)$$

$$\sigma = \frac{1}{n-1} \left[ \sum_{i=s}^m f_i x_i^2 - \frac{(\sum_{i=s}^m f_i x_i)^2}{n} \right] \quad (3.3)$$

Thereby,  $s$  and  $m$  are the index of the start and end bin, defining the considered value range, respectively.  $x_i$  is the data value at bin  $i$  (which is  $i$  in our case),  $f_i$  the number of frequencies at bin  $i$ ,  $n$  the total number of frequencies within the considered value range and  $i$  is the index counting from  $s$  to  $m$ . The resulting value range reaches from  $\mu - \sigma$  to  $\mu + \sigma$  and is used for the computation of the weight range.

In order to determine whether a block should be added to a PRH its so called weight range ( $w_r$ ) is computed and compared to a certain epsilon ( $\epsilon$ ) value. It is calculated by dividing a blocks amount of voxels within the value range from  $\mu - \sigma$  to  $\mu + \sigma$  by its total amount of voxels. A naive approach to determine the total and partial voxel numbers would be to traverse each voxel of a block and count them. This can however be done more efficiently, by accumulating bin frequencies of a blocks histogram within



**Figure 3.5:** Overview of the process for the calculation of initial (unmerged) PRHs.

the desired range. For the partial voxel number this would be the range from  $\mu - \sigma$  to  $\mu + \sigma$ . In order to compute the accumulated bin frequency of an arbitrary value range in constant time we created the accumulated histogram of each data set block. Using them the voxels within a certain value range can be determined by subtracting the frequency at bin  $\mu - \sigma - 1$  from that at bin  $\mu + \sigma$ . The application of the accumulated histograms is especially useful if a large number of while iterations and a frequent adaption of the  $\epsilon$  value occurs at the computation of the initial PRHs. It should be noted, that the application of accumulated histograms entails some initial computation time and additional memory requirements. This, however, is usually compensated by the subsequent time savings. The main benefit of the utilization of histograms for counting voxels is that single blocks do not repeatedly need to be traversed. Moreover, they are very useful if the same blocks are used for AH and PRH calculation. In that case the work of loading and splitting the data set as well as computing corresponding local histograms can be omitted. Instead the PRHs can be determined using the copied data from the AH class.

By applying the described calculations for each block, those with a weight range larger or equal a certain  $\epsilon$  are determined. For them **all** frequencies of the corresponding local histogram are added to the PRH of the current while-iteration. For the remaining blocks non of the frequencies are added. After each block has been treated, the PRH is checked for its total frequency. For an entirely empty PRH the epsilon value is decreased by a certain amount and the weight range check is repeated. This is done until the PRH is not empty any more.

As soon as a PRH is not empty it is saved to a vector holding all resulting PRHs. In addition, the original histogram and AH must be updated. For the first, this is usually done by simply subtracting the PRH bins from the according bins of the original histogram. However, towards the end of the while-loop-iterations it frequently happens that a PRH bin contains a higher frequency than a corresponding original histogram bin. In this case the according original histogram bin is simply set to zero. In addition, the PRH bin is corrected, to the last frequency of the corresponding original histogram bin. This ensures that the sum of frequencies contained by all PRH only hold the amount of voxels that are contained by the data set. In order to update the AH, the values of the according bins of the PRH are multiplied by the corresponding entry of the vector holding the relations between AH and original histogram bins.

After the AH and original histogram are updated, their remaining frequencies are checked. If they are empty the program stops extracting PRH and proceeds to the merging step. Otherwise the next iteration of the while-loop is started. The "emptiness" check is performed on the original histogram since the AH usually contains a slightly positive or negative frequency value in the end, resulting from the "rounding errors" when updating its bins.

In the course of this step the  $\epsilon$  parameter plays an important role, since it has a great impact on which blocks are assigned to a certain PRH. While a high  $\epsilon$  contributes to the detection of homogeneous structures, a low one detects inhomogeneous ones [54]. In the current work a default value of 0.9 with an adaptor of 0.1 for empty PRHs turned out to deliver good results. However, the user may adjust both via the settings.

**Extended Creation of Initial Partial Range Histograms:** In order to further improve the performance and results of the initial PRH calculation it is extended by the following facets.

We add a further epsilon check (called *pre-epsilon-check* in the following) in addition to the weight range test to obtain more flexible epsilon values for the creation of PRHs at first. It helps to decide whether a current PRH resulting from a certain epsilon value should be kept or if a better one can be obtained from a lower epsilon. As mentioned using an appropriate epsilon and corresponding adaptor is very important. These values have a great impact on the amount and composition of initial PRHs and thus on the results and performance of the PRH calculation.

Very low epsilon values may cause undesirable blocks which do not or do hardly contribute to a certain structure to be added to a PRH. Very high epsilon values, on the other hand, may be too restrictive at the selection of blocks. The conventional weight range test adapts the epsilon value to circumvent

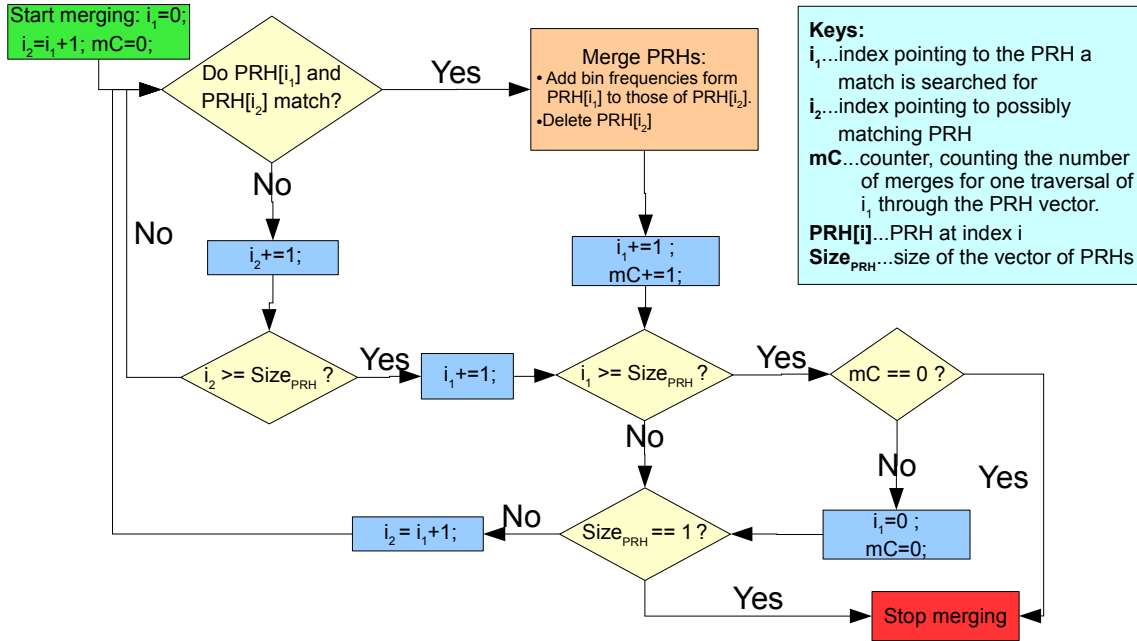


the detection of empty PRHs. However, a refined and more flexible adaption of this value is frequently reasonable. High epsilon values, especially in combination with a low adaptor, often lead to very small initial PRHs. As a result, their subtraction from the original histogram and AH may not reveal a new highest peak and therewith value range for the next iteration of the PRH calculation. Hence, redundant PRHs consisting of the same blocks may be obtained and saved numerous times. This is undesirable because as a consequence the same voxels are subtracted from the original histogram and AH multiple times falsifying their distribution. Moreover, the repeated detection of the same highest peak also entails a large amount of redundant weight range checks for the same value range and epsilon. This significantly increases the required calculation time for both the initial PRH computation and the subsequent merging step.

In order to overcome these shortcomings we add a *pre-epsilon-check*. Accordingly to this check, a suitable epsilon is as high as possible, but low enough to deliver a PRH, whose subtraction from the original histogram and AH reveals a new highest peak. This test is performed immediately after a negative regular weight range test (meaning the PRH is not empty). This can be seen in Figure 3.5, in which added computations are highlighted by a purple rectangle. It tests whether the PRH constructed for the current value range and epsilon would result in the same value range for PRH extraction in the next while-iteration. This is done by subtracting the current PRH from a temporary original histogram and AH, performing the peak search on the updated AH and subsequently computing mean and deviation for the identified peak. If the resulting value range is the same as for the current PRH, this and the corresponding temporary precalculations are discarded and the current epsilon value is reduced by the epsilon adaptor. The test is repeated until an epsilon which reveals a new peak for the next iteration is found. For an efficient application of this extension the adaptor value should not be too low. Otherwise the performance of the initial PRH calculation could be impaired because the described check must frequently be traversed until an appropriate epsilon is found. Moreover, especially when using a small adaptor, a changing identification of equal PRHs may occur. This means the same peaks are not identified in succession but are revealed over and over throughout the single iterations since they are never completely removed. This happens because the removed peak parts are large enough to make another peak the highest one for the next iteration but they are still not entirely removed.

As a further improvement an *initial PRH filter* was added. It is executed at the end of the initial PRH calculation and simply removes all but one initial PRH with equal mean and deviation. This is important for the runtime of the merging step for the case that redundant initial PRH remain despite of the *pre-epsilon-check*.

**Basic Merging:** Basic merging corresponds to step 6 of Section 2.1.6.2. It is implemented to traverse the PRH vector just created with two indices and combine similar histograms (see Figure 3.6). The first index ( $i_1$ ) starts at zero and points at the PRH for which a matching partner is searched. The second index ( $i_2$ ) points at possible matches. For each  $i_1$ ,  $i_2$  starts at  $i_1 + 1$  and traverses the following vector entries until a match is found or the end of the vector is reached. In the first case, the frequencies of the PRH  $i_2$  points to are added to the corresponding bins of the first one. Then the "second" PRH is deleted. In addition,  $i_1$  is increased by one and  $i_2$  set to  $i_1 + 1$ . Subsequently, a match for the PRH to which the new  $i_1$  points is searched. In the second case no merge is performed but  $i_1$  and  $i_2$  are still updated in the same way. As soon as  $i_1$  reaches the end of the vector, a check is performed on whether any merges occurred during the last traversal of  $i_1$ . If, so another traversal is performed on the reduced PRH vector, starting with  $i_1 = 0$ . Otherwise the merging step is finished. Usually there are a number of traversals until a final, small number of PRH remains which constitute regions of interest in the TF space. Optionally an additional filter step can be applied on the resulting PRH series to delete very small PRHs which are a remainder of the merging procedure. By empirical testing, it turned out to be reasonable to delete all PRHs with a total frequency smaller than 0.0001 times the total number of data set voxels in many cases. As a consequence a set of useful PRH describing regions of interest are kept. Note, that the similarity check to determine whether two PRH should be merged is performed according to the formula explained



**Figure 3.6:** Basic merging of PRHs. This approach is used to combine similar regions in the TF space, which are likely to belong together.

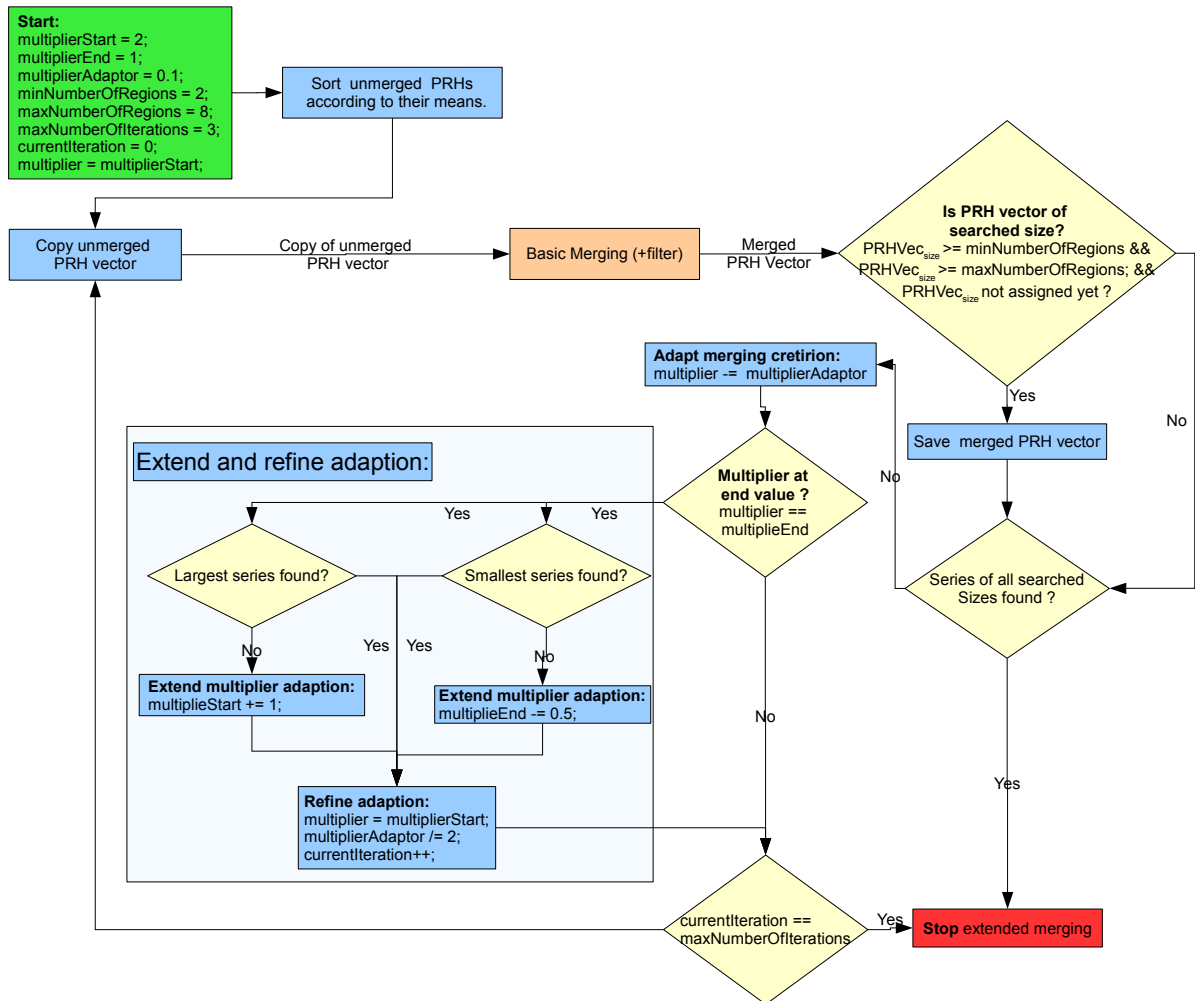
in Section 2.1.6.2, step 6.

**Extended Merging:** In the current implementation the basic merging step just explained is extended by a number of facets in order to obtain a wider range of different region of interest (PRH) (see Figure 3.7). Therefore, the unmerged PRH vector is copied and merged to a final set of PRHs multiple times using slightly adapted merging criteria for each iteration. As a result, a number of different PRH series are obtained. The adapted merging check can be found in equation 3.4:

$$(\sigma_{max}/\sigma_{min} \leq 4 * multiplier) \wedge (\mu_{max} - \mu_{min} \leq multiplier * \sigma_{min} * \max(1, 2 - \sigma_{min}/40)) \quad (3.4)$$

As can be seen a multiplier has been added to both parts of the merging criterion. Thereby, a high multiplier makes merging easier and hence leads to a lower number of PRH in a series. Conversely, a low multiplier leads to a high number of PRH in a series since they need to be very similar in order to be merged.

In order to constraint the results and multiplier range, six additional parameters are introduced: the multiplier start, multiplier end, multiplier adapter, smallest series size, largest series size and a maximum number of iterations value. The multipliers used in the equation result from the multiplier start, multiplier end and multiplier adaption value. By default these values are set to 2, 1 and 0.1 respectively. This means the basic merging step is performed for the values 2, 1.9, 1.8, ... 1. The "smallest series size" and "largest series size" value define series of which size are searched for. If a vector of merged PRHs, resulting from merging with a certain multiplier, is within the searched size range and no previous series of that size has been saved yet, it is stored. The restriction to only save one series of a certain size is reasonable because series of the same size usually result from similar multipliers and tend to consist of very similar regions. Saving all of them would result in overly large numbers of redundant regions. As soon as all series of searched size have been found merging using different multipliers is stopped. However, this may require multiple iterations from a multiplier start to end. Thereby, the multiplier adaption scheme is extend and refined for each iteration after the first one. At the end of each iteration after merging has been performed



**Figure 3.7:** Extended merging of PRHs. This approach extends the basic merging step just introduced to deliver a larger number of different regions of interest. Therefore, a scheme which performs basic merging multiple times, using a series of adapted merging criteria is deployed.

with the current multiplier end value, two checks are performed. They determine whether PRH series of the largest and smallest searched size have already been detected and saved. If a series of the smallest size has not been found yet, the multiplier start value is increased. This makes merging easier and hence leads to smaller series. Conversely, if a series of the largest searched size has not been detected yet the multiplier end is decreased. In addition, the search is refined by halving the adapter value. Then the search for series is repeated using the multipliers resulting from the new start, end and adapter value. This extension and refinement may be repeated for a user defined number of iterations. If the number of regions of interest in a data set is not known at all, a user can simply search for a large range of series sizes. As a result, all created PRH series are saved, and the number of iterations defines the duration of the search.

After extended merging there exists a vector of PRH series. The mean and deviation of the contained PRHs define the value ranges of initial regions of interest in the TF space.

In addition to the present explanations on extended merging, we performed a number of important optimizations which significantly accelerate the computation of PRH series. First, the mean and deviation values for each PRH in the unmerged PRH vector are computed in advance and stored in separate vectors. A copy of those vectors is passed to the merging function whenever the basic merging step is executed for a certain multiplier. As a result, the mean and deviation values of two PRHs which are checked for similarity do not repeatedly need to be computed, but can simply be looked up. In case a merge occurs the affected mean and deviation vector entries are updated. Secondly, the already used multiplier values to adapt the merging criterion must be saved. This circumvents the repeated computation of equal PRH series resulting from the same multiplier, throughout later merging-search iterations using refined and extended multiplier ranges. Finally, prior to copying and merging, the initial PRHs are sorted according to increasing mean values. Therefore, the bubble sort algorithm shown in [77] has been adapted accordingly. The proximity of similar PRHs accelerates all of the subsequent merging steps for different multipliers.

## 3.4 Transfer Functions

A TF is defined by a number of points with certain  $RGB\alpha$  values, which are distributed over the data value range. In order to define the whole TF space, the color and opacity values between these points are interpolated. In this work regions of interest are defined by three points. Their positions ( $\mu$ ,  $\mu - \sigma$  and  $\mu + \sigma$ ) are defined by the mean and deviation values resulting from the AH and PRH calculation. The detailed implementation of regions of interest and TFs in the design gallery is described in Section 4.4.1. This also includes a description of the data structure which unifies the actual TF with the image representing it in the design gallery.

### 3.4.1 Setup

In order to setup initial TFs each region of interest, resulting from the AH and PRH calculation, is assigned a  $RGB\alpha$  value. Color and opacity are both determined according to the mean value position in the TF space. Note, that the retrieved opacity is only assigned to the mean point of the region of interest, while the corresponding deviation positions are assigned an opacity of zero. The retrieved color is assigned to all three points.

The TF setup process also comprises an additional filter step to remove equal or similar regions of interest which frequently occur as a result of the extended merging step (see Section 3.3.2, Paragraph "Extended Merging"). For the use of 1D TFs only equal regions of interest are removed. Thereby, two regions of interest are considered equal if they have exactly the same mean and deviation value. At the application of separable TFs the set of initial TFs per dimension is additionally reduced by very similar ones. This is reasonable since the combination of the single dimensions usually results in a

very large number of initial separable TFs (see Section 3.6.1, Paragraph "Separable Transfer Function Precalculation Process"). In order to identify similar regions of interest the extended PRH merging criterion from Equation 3.4 is deployed with a very small multiplier. The detailed implementation of the setup process is described in Section 4.4.2.

### 3.4.2 Adaption and Combination

After initial TFs have been set up, users are provided with the functionality to individually adapt and to combine them.

Their adaption is very simple. In order to change the color of a region of interest, each of the points defining it are assigned the new *RGB* value. The opacity is changed by assigning a new  $\alpha$  value to the mean point.

The creation of combined TFs is more complex and requires a number of considerations. Basically, a number of regions of interest are unified by setting all the points defining them in a new combined TF. However, it should be obeyed that single regions frequently overlap leading to undesired colors and opacities in and around the overlapping areas. This happens because colors and opacities are interpolated between points of different regions of interest. Therefore, overlaps between regions of interest should be eliminated during the creation of the combined TF to ensure appropriate visualizations. Basically, overlaps are resolved by removing the deviation points reaching into the other region and replacing them by a new intermediate point. This adds a smooth transition between two regions of interest.

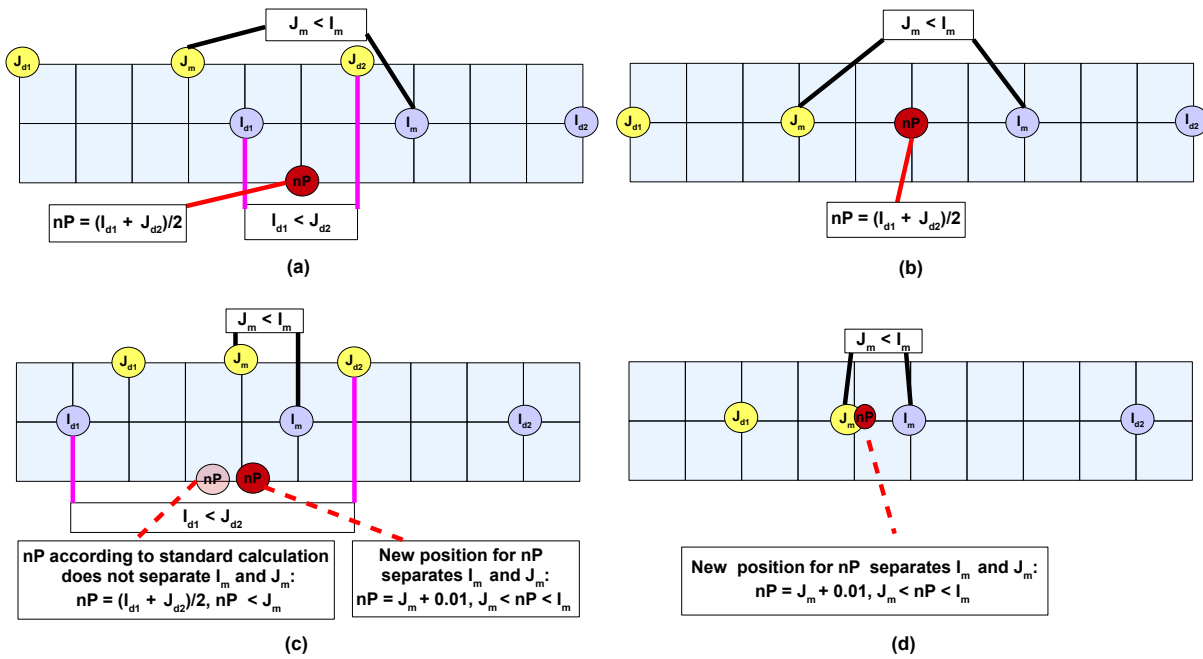
The following description explains how two overlapping regions  $i$  and  $j$  are handled in detail. Therefore, the mean and deviation points of each region are defined.  $i_m$  stands for the mean point position of region  $i$ .  $i_{d1}$  and  $i_{d2}$  describe the deviation points to the left and right of the mean, respectively.  $j_m$ ,  $j_{d1}$  and  $j_{d2}$  define region  $j$  in the same way.

Assume  $j_m < i_m$  and  $i_{d1} < j_{d2}$  (see Figure 3.8 (a)). In this case both points,  $i_{d1}$  and  $j_{d2}$  are discarded and replaced by a single new point ( $nP$ ) separating the two regions. In order to position the new border according to the extends of the original regions, the points position is calculated depending on  $i_{d1}$  and  $j_{d2}$  by the following formula:  $nP = (i_{d1} + j_{d2})/2$ . In the simplest case, if  $j_m < nP < i_m$  is true, this assignment is enough (see Figure 3.8 (b)). However, there are also cases in which  $j_m > nP$  or  $nP > i_m$  is true. Then  $nP$  is set to  $j_m + 0.01$  or  $i_m - 0.01$ , respectively. If this position does not lie between the two means  $j_m, i_m$ , which may at very close regions of interest,  $nP$  is set to  $(j_m + i_m)/2$ . As a result, the new point is always between the regions mean values and separates them. An example, showing regions overlapping this way and the resulting new point can be found in Figure 3.8 (c) and (d). The color of the new point does not matter because its opacity is zero.

In order to resolve the intersections of each subsequent pair  $j$  and  $i$  according to the above description all regions of interest are initially sorted according to increasing mean values. Subsequently, we iterate through each pair by using their indices  $j_{idx}$  and  $i_{idx}$ , where  $j_{idx} = 0 \dots n_{roi} - 2$  and  $i_{idx} = j_{idx} + 1$ . Thereby,  $n_{roi}$  stands for the number of currently combined regions of interest.

## 3.5 User Interface and Functionality

The user interface represents an image centric method with the user as the TF evaluating instance. Its purpose is to provide users with the required means for efficient TF specification starting from initial ones generated in the precalculation step. Therefore, it is important to provide easy and intuitive ways to work with TFs. An important concept in this context, which enables the work with TFs on an abstract level, is to represent them by images. This saves user's cognitive load of thinking about the impact of changes in the TF space. The user interface constraints interactions in a meaningful way and offers the user guidance in adapting TFs as required by providing examples. Moreover, it uses simple and approved



**Figure 3.8:** Shows how to handle intersecting regions in the TF space. (a) and (b) show a standard case in which the overlapping deviation points  $i_{d1}$  and  $j_{d2}$  are replaced by a new point  $nP$  lying exactly half way between them. (c) and (d) show a more special case. If placed halfway between the overlapping deviation value, the new point  $nP$  would not separate the mean points  $j_m$  and  $i_m$  of both regions. Instead it lies to the left of the smaller mean  $j_m$ . Therefore, it must be replaced to a point slightly to the right of  $j_m$ .

approaches to browse thumbnails representing TFs. The user interface of the discussed system basically consists of three main sections:

- *The exploration area* is the part of the user interface in which most interaction takes place. It is used to arrange the thumbnails representing TFs as well as to browse, adapt, delete and recreate them. Moreover, the selection of thumbnails allows to combine their TFs into a single one. As a result, an image depicting multiple structures can be created. There exist two major implementations for this area. One makes use of the CF design, presented in Section 2.4.2, while the other is based on the DM concept from Section 2.4.1. The latter is an improvement from an exploration area user interface based on a 3D billboard. Here, thumbnails are arranged in 3D without being mounted to a reference structure.
- *The combination area* is used to depict the rendering resulting from all selected thumbnails in the exploration area from a desired direction and distance. The combined image is shown directly in the output widget of the rendering system. This area also provides a visualization of the combined TF in its domain. Moreover, it allows to access a manual TF editor for fine tuning and to save desirable TFs to the storage area.
- *The storage (or saved combinations) area* allows users to save interesting TF combinations as thumbnails and provides an overview of them. From this area, TFs can be deleted or saved to disk.

Figures 3.9 and 3.10 provide an overview of the user interface with both, the DM and CF versions of the exploration area. Thereby, the exploration, combination and storage area including their most important features, actions and relations are presented. Additionally, the menu and toolbar can be seen in both figures.

In the following sections all user interface components mentioned above are described in detail. Moreover, a user interface component which shows visualizations of the calculated AH and PRHs and allows to interact with them is presented.

### 3.5.1 The Exploration Area

In the presented user interface the exploration area is usually the part in which most interaction takes place. It provides the following core functionality:

- An arrangement scheme for thumbnails representing TFs,
- means for browsing thumbnails,
- a scheme to adapt TFs and their associated thumbnails, and
- a scheme to select thumbnails, and to join their depicted structures in the combination area

These functions are realized in a way that allows fast, intuitive and effective design of meaningful TFs. For this purpose a number of different implementations have been tested throughout the development of the exploration area.

The original idea has been to arrange thumbnails in a 3D billboard according to a certain parameter for each dimension. However, it often proved to be difficult to keep track of the current position (camera position) and to navigate efficiently in the 3D billboard space. Moreover, it is hard to find appropriate parameters for a meaningful arrangement.

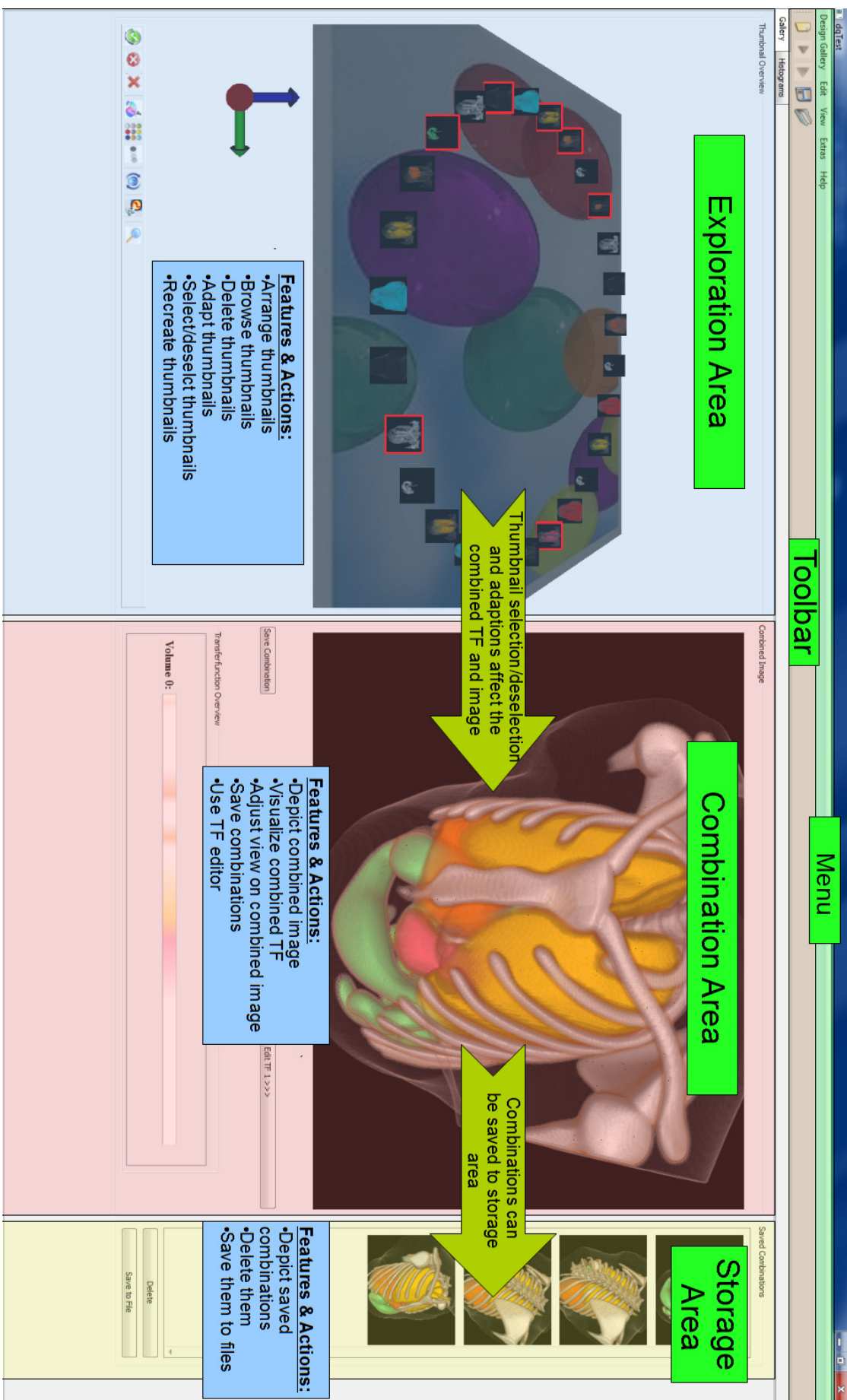
As a result, the billboard has been enhanced to an implementation based on Robertson et al.'s [73] DM presented in Section 2.4.1. Due to the new approach, users are able to individually and easily adapt an initial default arrangement of thumbnails as required. Moreover, positioning thumbnails on the skew plane greatly improves the overview of and navigation in the scene. Unfortunately, there is a significant weakness of the billboard which the DM implementation cannot resolve entirely: The restricted visibility of small thumbnails.

Another implementation which makes use of the CF design [7] presented in Section 2.4.2 has been consulted. The main difference of this approach is that thumbnails are arranged and browsed in one dimension with a constantly good view on them. This remedies the visibility problem of the DM and allows to browse thumbnails in a very intuitive and simple way. However, there are also a number of merits of the DM based implementation. This especially concerns the individual arrangement of thumbnails in 3D as well as the adjustable view. A detailed comparison of the characteristics of the CF and DM based exploration area can be found in Table 3.3 and Section 3.5.1.4.

The following subsections discuss all approaches and their advantages and disadvantages which have already been broached above. Section 3.5.1.1 starts with a description of the functionality of the billboard implementation. Section 3.5.1.2 extends and adapts these statements in order to explain the features of the DM based exploration area. Subsequently, the implementation using the CF design is presented in Section 3.5.1.3. Finally, Section 3.5.1.4 directly compares, the DM and CF based approach as well as their advantages and disadvantages.

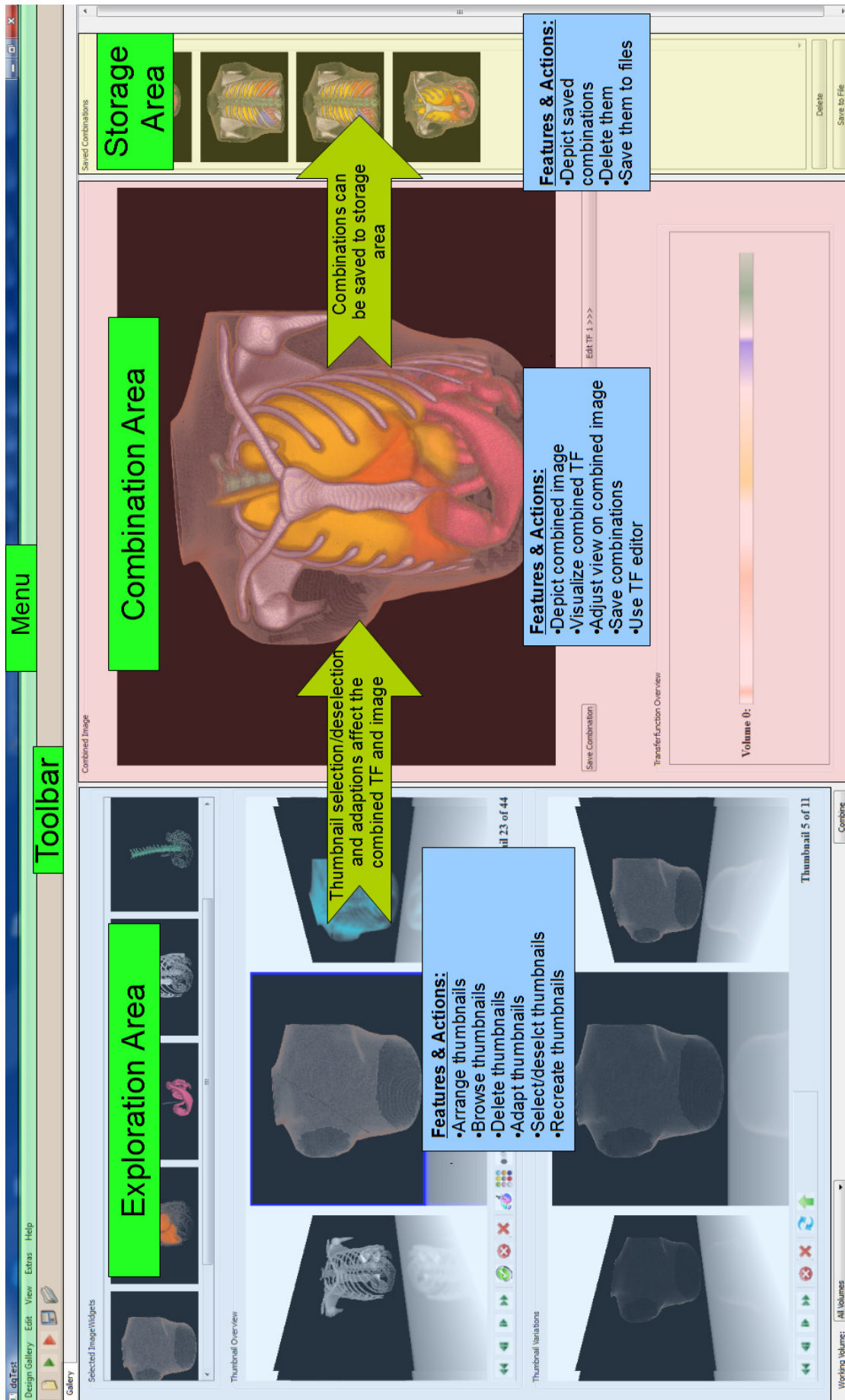
#### 3.5.1.1 3D Billboard Exploration Area

This section describes the billboard based exploration area. The billboard implementation is a predecessor of that based on the DM. Several DM figures are used to explain the predecessor *billboard* because of similar functionalities in both approaches.



**Figure 3.9:** Overview of the design gallery user interface with the exploration area based on the DM design. Thereby, green rectangles contain labels of the single areas. The notes in the blue rectangles describe the most important features and actions of each area. The green arrows show the most important effects from actions in one area on another.





**Figure 3.10:** Overview of the design gallery user interface with the exploration area based on the CF design. Thereby, green rectangles label the single areas. The notes in the blue rectangles describe the most important features and actions of each area. The green arrows show the most important effects from actions in one area on another.

The billboard is basically a 3D space of a certain extend in which thumbnails (represented by flat and textured 3D cuboids) are arranged accordingly to one parameter per dimension. This allows to automatically locate similar thumbnails close to each other. However, only a limited number of parameters is available for positioning them. These are the normalized mean value, deviation value, grayscale value (derived from the RGB color) and opacity of a region of interest in the TF space. Their combined application is not optimal, since a region of interest's color and opacity are derived from its mean value in the TF setup scheme.

In order to *browse* the thumbnails, the billboard basically offers three default actions by which the camera position and view direction can be changed. These are panning, rotating and zooming (= move camera closer or further away along a straight line of the current viewing direction) (see Figure 3.13 (c)). In order to allow users to constantly see thumbnails from the front, they are automatically rotated according to the cameras viewing direction. This provides an optimal view. A drawback of browsing the thumbnails by navigating through the billboards 3D space is, that one can easily get lost. This even gets worse due to the automatically adjusted thumbnail rotation. In order to mitigate this problem a so called gnomon [86] was added to the bottom left corner of the billboard widget. This is a little coordinate system which allows to keep track of the current orientation of the camera. However, it does not provide any information about the user's position in the billboard. Another way to alleviate this problem is the *reset view* function. It allows to restore the initial overview of the whole scene by resetting the camera to its starting position and viewing direction. A further problem of billboard browsing is the required time to navigate from one thumbnail to the next. A facilitation of this issue is achieved by allowing thumbnails to be translated to a preferred view close to the camera and back by a single click (see Figure 3.13 (d)).

A thumbnail may be *selected* by simply performing a left mouse button (LMB) click on it. Subsequently, the cuboid on which the thumbnail is displayed is framed by red lines along its edges. Upon *deselection*, performed by a further LMB click, the frame disappears. The selection and deselection of thumbnails is of special importance because the TFs of all selected thumbnails are combined into one which determines the image depicted in the combination area. Thereby, the combined TF and its corresponding image are immediately and automatically updated whenever a thumbnail is selected or deselected. This constant feedback on changes is an important part of interactive TF design. See Figure 3.11.

There are basically two ways to *adapt* a TF associated with a thumbnail in terms of its color and opacity. As a first option one may perform a middle mouse button (MMB) click on a thumbnail in order to choose a new color and/or opacity from a color picker dialog (See Figure 3.11). The updated TF immediately effects the thumbnail rendering, the combined TF as well as the combined image. This option is desirable if one is sure about the color and/or opacity of choice. Alternatively a user may press "CTRL + wheel up or down" on a thumbnail in order to create a series of system generated color or opacity variations, respectively. Thereby, the pressed thumbnail is translated closer to the camera and its variations are positioned in a surrounding circle. All modifications as well as the center thumbnail can then be selected and deselected as usual. This kind of arrangement provides a good overview of choices and allows to test the impact of variations on the combined image (TF). Moreover, it clearly stresses this constellation from the remaining thumbnails. Having found a desired alteration, one may again click on it with "CTRL + wheel up or down". As a result, it is exchanged with the center thumbnail which makes it the new center. This operation can be performed multiple times. A further "CTRL + wheel up or down" click on the new center deletes all surrounding thumbnails and places it back to the original position. Variation circles may be open for an arbitrary number of thumbnails at once, but can not be nested (variations of variations in the circle are not possible). This functionality is very useful for testing the impact of variations of multiple structures in a combination. See Figure 3.11. The process of working with system generated variations is depicted in Figure 3.14 (a-d). A detailed description of the applied transformations to position thumbnails in a variation circle and the according code can be found in Section 4.5.5, Paragraph "Variations - Creation, Interaction, Resolution" and Listing 4.6, respectively.

The system also allows to *delete* a single or all selected thumbnail(s) from the billboard. This can be

done by pressing the corresponding button below the 3D scene. Thereby, the deletion of a thumbnail in the center of a variation circle deserves special attention since it also causes all alterations to be removed. This operation is usually required to delete unnecessary thumbnails resulting from the precalculation step, such as those depicting air. See Figure 3.11.

By pressing the *recreate* thumbnails button new depictions are rendered for each thumbnail using unchanged TFs. This step only makes sense if the camera position or orientation of the renderer in the combination area is changed. Frequently it may be the case, that the initial viewing direction and distance of the renderer camera does not provide satisfying images. In addition, one may want to adapt the viewing direction and distance several times during the design of TFs depending on the currently observed structures. The recreation button allows to update all thumbnails according to the adjusted settings and requirements. See Figure 3.11.

### 3.5.1.2 Data Mountain Based Exploration Area

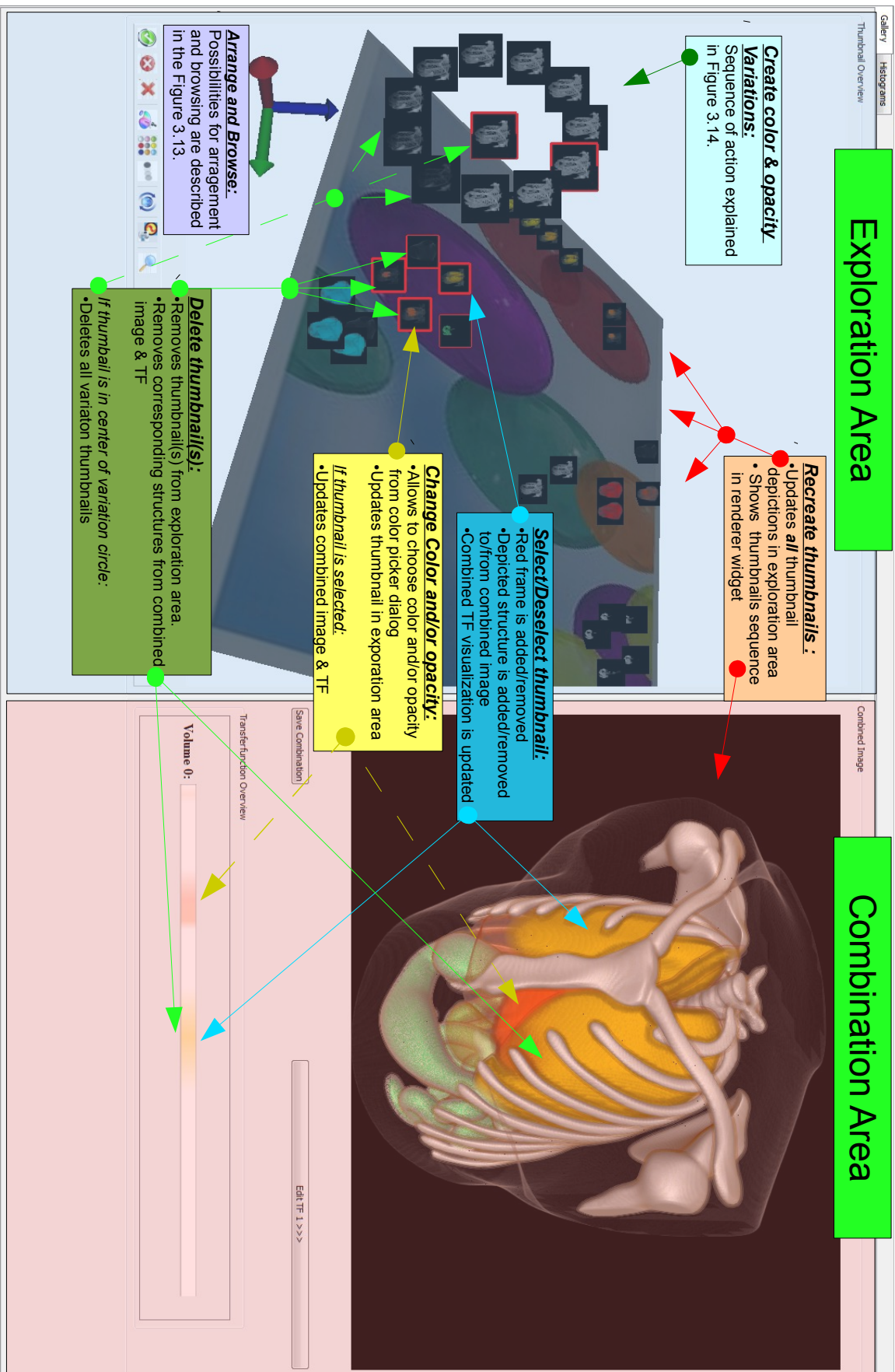
As a response to the drawbacks of the 3D billboard, it has been enhanced to a user interface based on the DM described in Section 2.4.1.

In the following this section describes how the DM based exploration area differs from that based on the billboard. Subsequently, an explanation of how thumbnails movements are restricted to the DM plane is provided. The last paragraph discusses the benefits and limitations of the DM based exploration area.

**Modifications Compared to the Billboard Based Exploration Area:** The main modification in the new exploration area is that thumbnails are now positioned on a skew textured plane which forms the DM. Thereby, the flat cubes are not arranged accordingly to certain parameters but in a default circle as can be seen in Figure 3.13 (a). From there, the user may adapt thumbnail positions by dragging them to any point on the plane (see Figure 3.13 (b)). This can be done by keeping the LMB pressed on a selected thumbnail. Then the thumbnail moves with the mouse cursor until the LMB is released. Thereby, mouse movements to the left and right change the X coordinate of the thumbnail, while up and down movements change the Y and Z coordinate at once. An arbitrary DM texture serves as a passive landmark, which helps to individually arrange the contents.

All other functions correspond to that of the billboard. This includes the selection/deselection, adaption (based on color picker and variation circle), deletion and recreation of thumbnails. An overview of these functions is also provided in Figure 3.11. In addition, Figure 3.14 (a-d) shows how to work with the variations circle. Moreover, translating single thumbnails to the preferred view (see Figure 3.13 (d)) as well as resetting the view works equally. The same goes for adjusting the camera position and viewing direction by panning, rotating and zooming (see Figure 3.13 (c)). In addition, the thumbnails are still rotated according to the cameras orientation. Note, that thumbnails in the variation circle and center as well as those in preferred view cannot be dragged anywhere.

**Dragging Thumbnails on the Data Mountain Plane:** In order to restrict thumbnail movements to the DM plane the bounding boxes of the plane and thumbnails are required. As a result of the bounding box computation, we know the ground plane and thumbnail cube minimum and maximum values in each dimension. Those of the ground plane are given by  $gcXMin$ ,  $gcYMin$ ,  $gcZMin$ ,  $gcXMax$ ,  $gcYMax$ ,  $gcZMax$ . And those of the thumbnail cube given by  $tcXmin$ ,  $tcYmin$ ,  $tcZmin$ ,  $tcXmax$ ,  $tcYmax$ ,  $tcZmax$ . Using them as well as the thumbnails width ( $thumbCubeWidth$ ) and height ( $thumbCubeHeight$ ), the following code restricts the new X and Y ( $newX$ ,  $newY$ ) coordinate values to which a thumbnail may be translated. Note, that thumbnail coordinates always refer to its center position, which is half its height above the DM plane.



**Figure 3.11:** Presentation of possible actions in the DM based exploration area and their effects. The discussed actions are: arrangement and browsing, selection/deselection, change of color and/or opacity, variation creation for color and opacity, deletion and recreation. Each action and its effects are summarized in a colored rectangle labelled with a bold headline. The corresponding arrows with solid lines point to examples of potentially affected manifestations. Arrows with dotted lines point to manifestations being only affected under certain circumstances. For example, a thumbnail must be selected that a change of its color affects the combined TF and image.

```

float newX, newY, newZ;
...
if(tcXmin < gcXMin_){
    newX = gcXMin_ + thumbCubeWidth/2;
}else if(tcXmax > gcXMax_){
    newX = gcXMax_ - thumbCubeWidth/2;
}else{ }

if(tcYmin < gcYMin_){
    newY = gcYMin_ + thumbCubeHeight/2;
}else if(tcYmin > gcYMax_){
    newY = gcYMax_ + thumbCubeHeight/2;
}else{ }
...

```

These checks basically compare corresponding minimum and maximum bounding box values of DM plane and thumbnail cube. If those of the thumbnail cube exceed/deceed those of the DM plane in a dimension, the new thumbnail center coordinate is adjusted to align compared boundaries. In more detail these checks can be described as following:

- Left X dimension check: If the thumbnails left border exceeds that of the DM plane, its center is positioned half its width to the right of the corresponding DM border in the X dimension.
- Right X dimension check: If the thumbnails right border exceeds that of the DM plane, its center is positioned half its width to the left of the corresponding DM border in the X dimension.
- Bottom Y dimension check: If the thumbnails bottom border exceeds that of the DM plane, its center is positioned half its height above of the corresponding DM border in the Y dimension.
- Top Y dimension check: If the thumbnails bottom border exceeds the DM planes top border, its center is positioned half its height above of the corresponding DM border in the Y dimension. The difference of this check to the others results from the fact that only the thumbnail cubes bottom boundary must be within the DM plane bounding box in order to restrict thumbnails movements to the DM plane.

The thumbnails new Z value is derived from the new Y value and the DM planes gradient as shown in the following code:

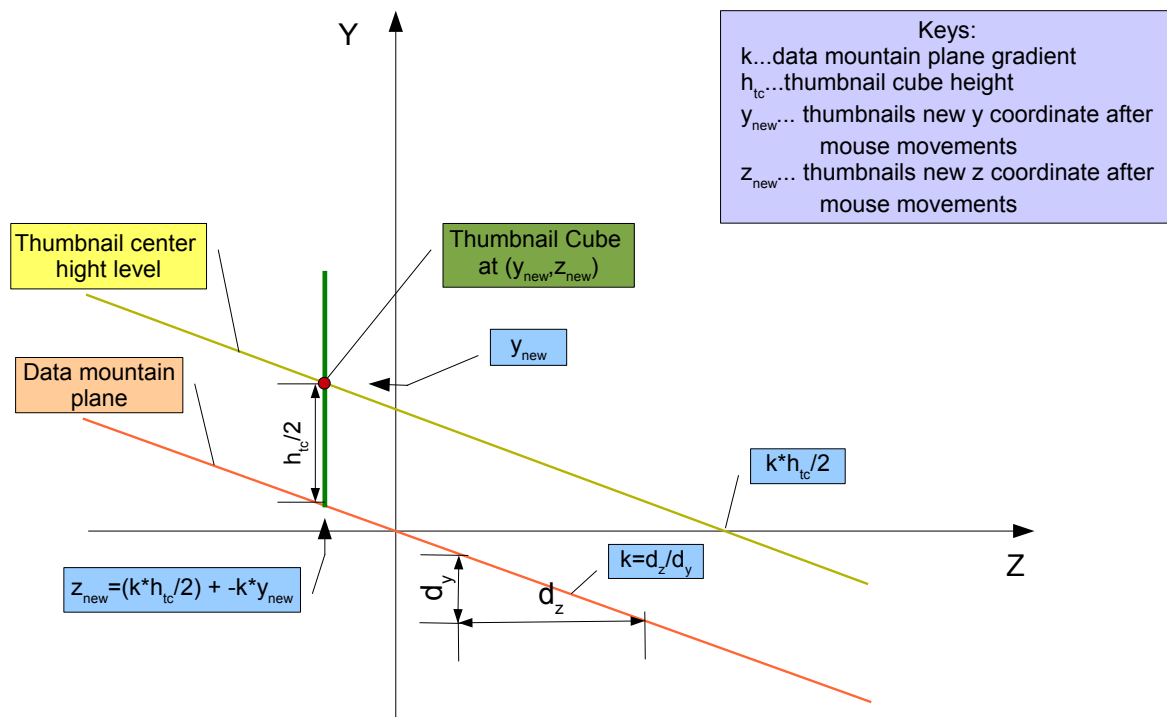
```

float k = (gcZMax_ - gcZMin_)/(gcYMax_ - gcYMin_);
newZ = (k*thumbCubeHeight/2) + -k*newY;

```

This equation is the result of two circumstances. First, the ground planes center lies in the scenes coordinate system center and reaches from positive Z and negative Y to negative Z and positive Y coordinates. Moreover, the thumbnail translation actually happens half a thumbnails cube height above the DM plane, where the thumbnails centers are. (see Figure 3.12)

**Benefits and Limitations:** Considering the impact of these changes on the process of TF design a number of benefits can be identified. First of all, the user is provided with a much better overview, since all thumbnails are bound to a skew reference plane instead of being spread over a visually unconstrained 3D space. In addition, the individual arrangement on the plane allows to group desired thumbnails within a close distance. This further contributes to a user's overview and allows to find, reach and variate different structures of a desired combination very fast. A further essential benefit is the possibility to perform a configuration in 3D space by a simple 2D interaction scheme. In other words, the benefits of browsing on the DM can be expressed by the individual and constrained arrangement of thumbnails.



**Figure 3.12:** Calculation of a dragged thumbnail's new z value.

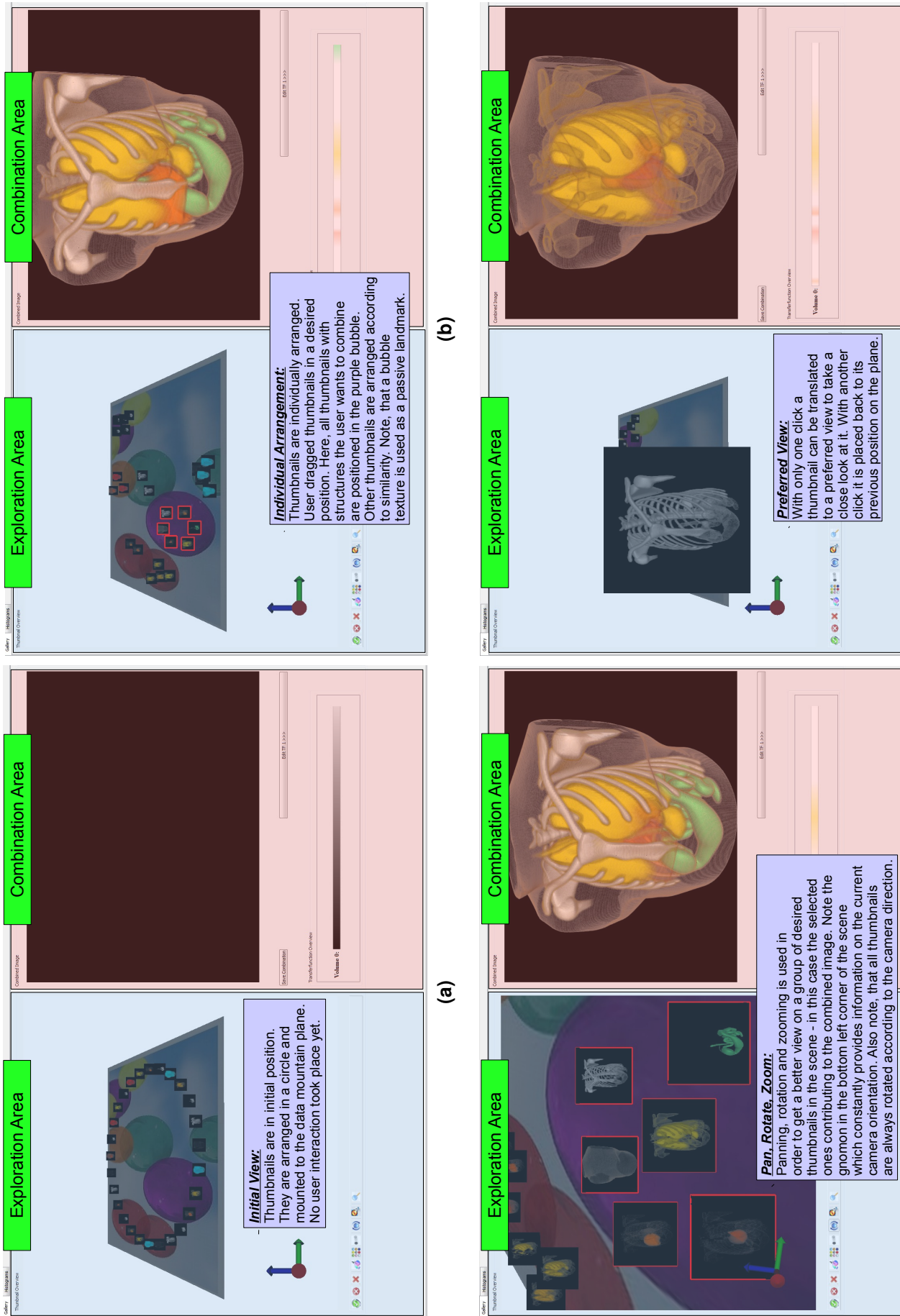
This allows to better memorize the position of thumbnails or groups of them, and to optimize their accessibility through their potential spatial proximity. For the same reason, the requirements for panning, rotating and zooming are significantly reduced.

However, there are also billboard related problems which cannot be solved by the DM. The most apparent one is the bad visibility of structures depicted on thumbnails viewed from a far distance. This problem was already mentioned in Section 2.4.1 describing Robertson et al.'s [73] DM, which is used to bookmark web sites. In the current context this issue is even more present. For bookmarking it may be sufficient to roughly see the content of a thumbnail to be reminded of the associated web site. In TF design however, both an overview of thumbnails as well as the possibility to clearly recognize depicted structures is required. Only the first requirement is sufficiently provided by the DM concept. For the latter it may be necessary to frequently switch between close view and overview at different thumbnails to clearly recognize their structures. Although the preferred view function facilitates this task it is not an ideal solution if frequent switching is necessary. An alternative approach for faster browsing including a distinct view on thumbnails is presented in the following section.

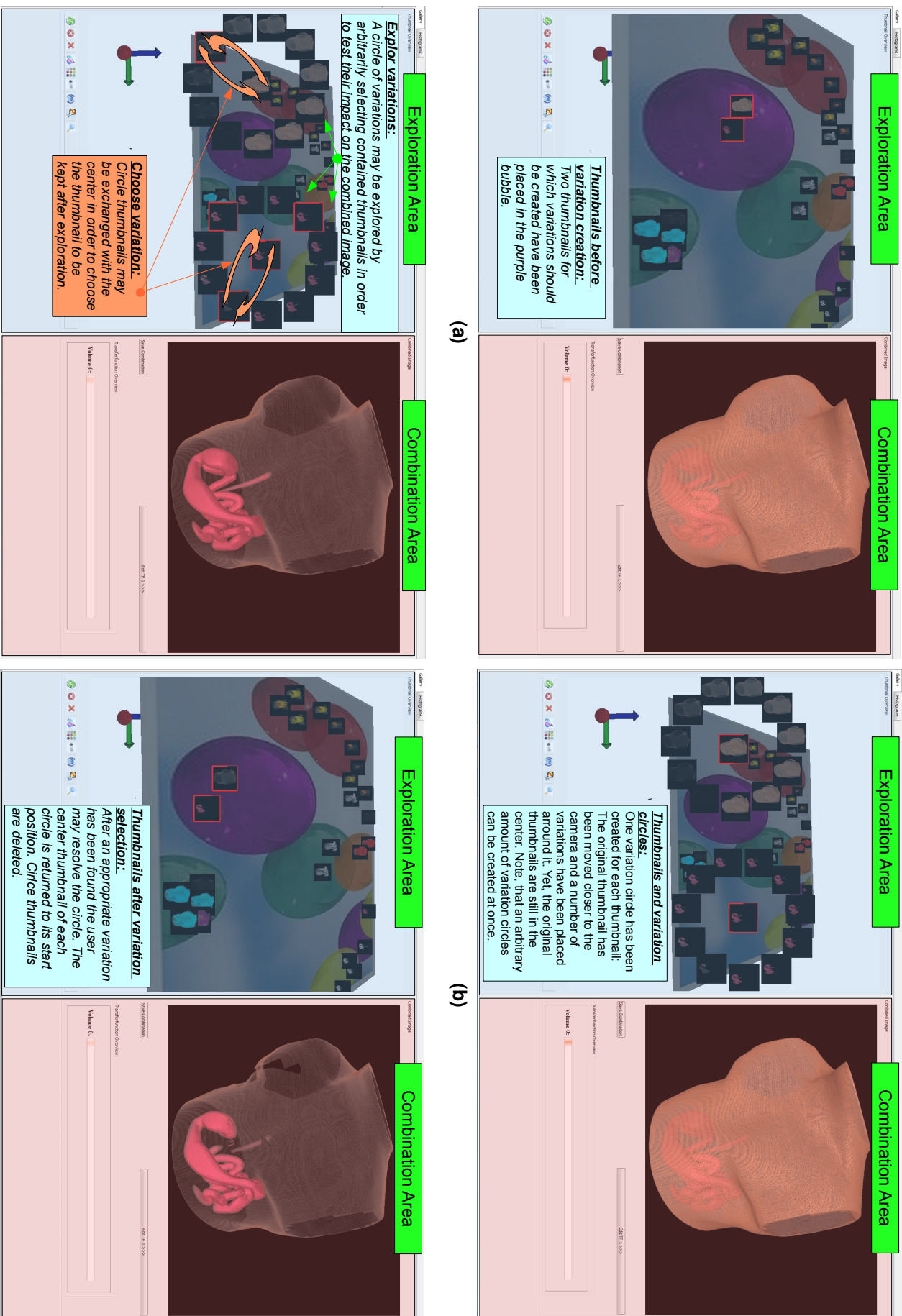
### 3.5.1.3 Cover Flow based Exploration Area

The current section explains the composition and functionality of the CF design based exploration area consisting of three subdivisions:

- The **main browser** area arranges thumbnails resulting from initial TFs accordingly to the CF design and allows to browse, select/deselect, delete and adapt them. All structures of selected thumbnails are visualized together in the combination area.
- The **variation browser** area allows to depict color and opacity variations of the currently centered thumbnail in the main browser. It is also implemented based on the CF design.



**Figure 3.13:** Presentation of different thumbnail arrangements and means for browsing. (a) shows the initial thumbnail arrangement, before any user interaction took place. (b) shows an individual arrangement, generated by dragging thumbnails along the DM plane. (c) shows an improved view on a group of thumbnails, obtained by panning, rotating and zooming. (d) shows a thumbnail in preferred view.



**Figure 3.14:** Presentation of the working process for using system generated variations. (a) shows two thumbnails to be varied in their initial position. (b) shows these thumbnails in their emphasized position with the variation circle around them. (c) demonstrates how to work in the variation circle. (d) shows the varied thumbnails which are translated back to the original thumbnails position.



- The **selected thumbnails** area shows a very small depiction of each currently selected main browser thumbnail and provides quick access to them.

**Main Browser Area:** The main browser is an implementation based on the CF design introduced in Section 2.4.2, extended to the needs of TF design. It forms the heart of the exploration area and contains the thumbnails representing initial and adapted TFs. It allows to browse, select/deselect, delete and adapt them. Moreover, thumbnails can be recreated for different view points. Figures 3.15 and 3.16 provide an overview of the functionality of the discussed area and its interactions with other parts of the user interface. In order to carry out the main browser operations described in the following the user interface offers a whole range of possibilities. These include the buttons arranged below the main browser as well as a context menu and a series of mouse and keyboard actions. A listing of all options to perform various main browser operations is depicted in Table 3.1. For the sake of clarity the descriptions below only reference the provided buttons which are labelled with numbers. In Figures 3.15 and 3.16 as well as in Table 3.1 these numbers are associated to the according buttons and their function.

For the explanation of *browsing* it is useful to recall the description of the CF design from Section 2.4.2. Thumbnails are considered as a horizontal stack of slices of which only a straight centered one as well as a number of tilted thumbnails to its left and right can be seen. By interacting with the browser, users may change the centered thumbnail and its surrounding. The motions from one center to another are depicted by smooth animations. In order to scroll through the thumbnails in the main browser the buttons one to four below it can be used (see Figure 3.15). The buttons with a single arrow to the left and right (button 2, 3) can be used to move one slide/thumbnail to the corresponding sides, respectively. The buttons with a double arrow to the left and right (button 1, 4) allow faster browsing by moving ten slides to the corresponding sides at once. The combination of slow and fast browsing allows users to navigate through a given set of TFs as required. The user may either take a close look at single thumbnails or quickly browse over large numbers of thumbnails to get an overview of them or find a certain one. Fast browsing is especially useful if large numbers of thumbnails are placed in the main browser. In order to allow users to keep track of their current position in the stack of slides, the index of the current and total number of thumbnails is depicted in the bottom right corner of the main browser area. Due to the emphasized position of the centered thumbnail, shown structures are always clearly visible in the CF based browser. Although the main browser does not provide an overview of all contained thumbnails, it shows the user previous and following thumbnails. In the current configuration only one thumbnail to the left and right of the center is visible. Therefore, the structures depicted on them are also clearly recognizable.

*Selecting and deselecting* of a currently centered thumbnail can be performed using button five of the main browser area (see Figure 3.15). A selected thumbnail can be recognized by the blue boarder framing it. These actions are basically required for the same purpose and effects as in the billboard and DM implementation. They are of special importance since the TFs of all selected thumbnails are combined to a single one. The resulting TF is subsequently used to render an image depicting all the structures emphasized by the single TFs at once. Thereby, a selection and deselection operation immediately affects the combined image, which is shown in the combination area. This allows users to easily see the impact of a single structure on a combined rendering and is hence an important part for successful TF design. In addition, a selected thumbnail is also added to the selected thumbnails area which provides an overview of the single TF thumbnails which currently contribute to the combined image. This also includes those which are currently not visible in the main browser. Deselected thumbnails are immediately removed from this area.

In order to *adapt* a TF and its representative thumbnail in terms of color and opacity, there exist a number of possibilities (see Figure 3.15).

Using button eight a dialog is opened, which allows to directly choose a new color and/or opacity for the centered thumbnail. This option is desirable if one is sure about the color and/or opacity of choice.

For example, white would be a common choice to depict a bone. Alternatively, the buttons nine and ten initiate the generation of a selection of color and opacity variations of the currently centered thumbnail, respectively. These variations are depicted in the according browser, from which the user may select a desired option. Due to that the system guides a user by offering a number of possibilities to choose from. Therefore, this approach is best suited if a user is not sure about the appropriate color or opacity. Changes performed with any of the described methods do immediately affect the depiction in the combination area as well as the according image in the selected thumbnail area. Once again the immediate effect of TF changes and the fact that color and opacity modifications are constrained to single thumbnails (TFs), allows the user to easily and intuitively explore the impact of a number of different depictions of a feature on the combined image (TF). As can be seen the basic idea for adaption is the same as used in the billboard and DM exploration area. The implementation however differs significantly. Instead of creating variation circles, the alternated thumbnails are also depicted in a CF based browser. This also implies that only the variations of one initial thumbnail may be shown at a time, as well as a constantly good view on shown variations.

The buttons six and seven allow to *delete* the currently centered and all selected thumbnails, respectively (see Figure 3.16). This makes sense, since the precalculations frequently deliver a number of TFs depicting uninteresting or very similar structures. Moreover, one may want to delete adapted thumbnails added from the variation browser.

Finally, button 11 allows to rerender all thumbnails without a previous change of their associated TFs (see Figure 3.16). This action follows the same purpose as the according operation described for the billboard implementation in Section 3.5.1.1.

**Variation Browser Area:** The variation browser is used to depict color and opacity variants of the thumbnails in the main browser. Figure 3.17 provides an overview of the functionality of this area and how it interacts with other parts of the user interface. Similar to the main browser, the variation browser offers a wide range of possibilities to perform interactions which are summarized in table 3.2. Also here, the descriptions below only reference the provided buttons which are labelled with numbers.

Since the variation browser is also implemented according to the CF design, browsing works similar as for the main browser. There are also four buttons for slow and fast browsing to the left and right (buttons 1 to 4). The only difference is that fast browsing only moves five instead of ten thumbnails to the according direction, because the variation browser usually contains less thumbnails.

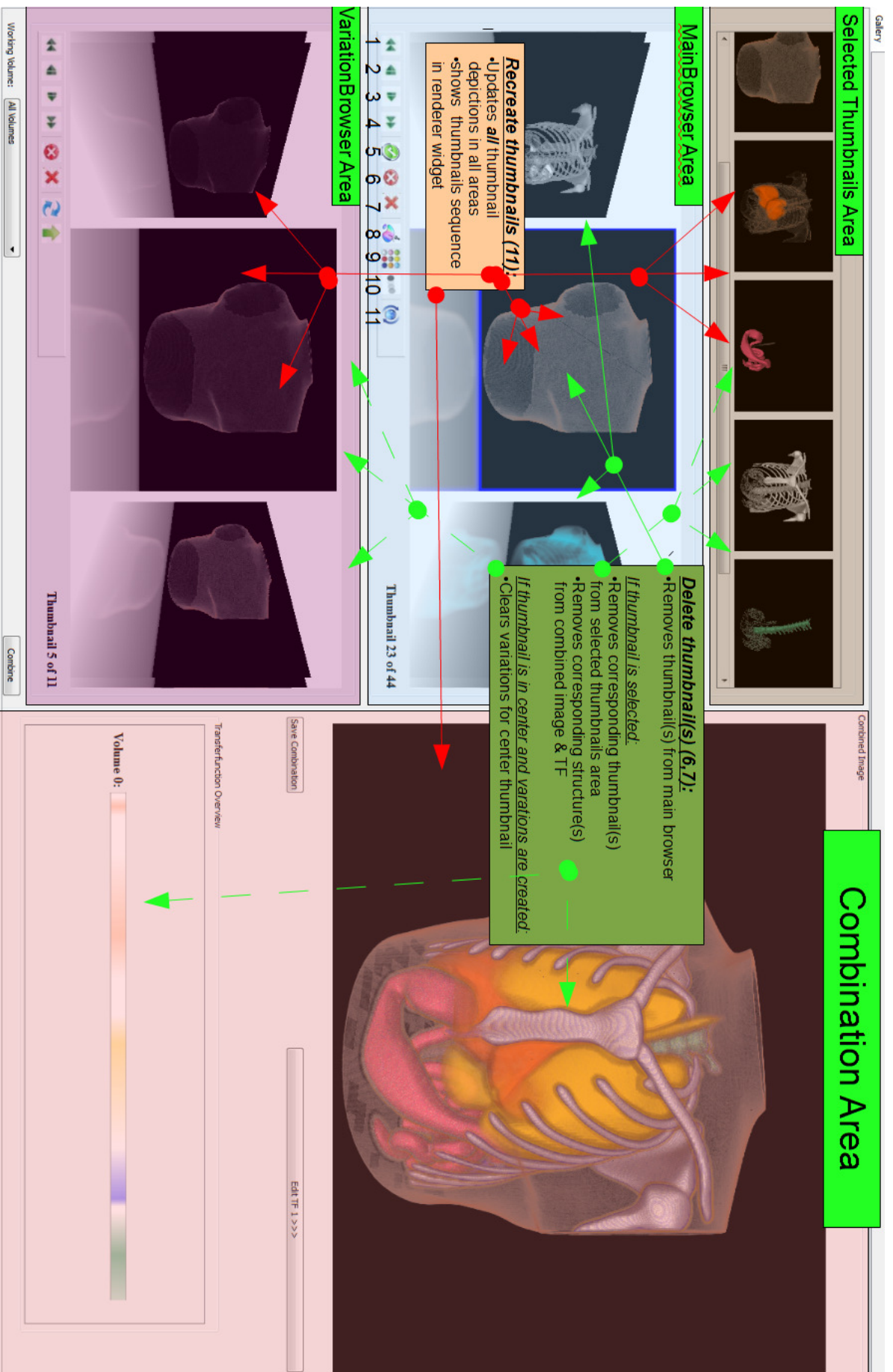
The buttons five and six allow to delete the currently centered or all variations, respectively. All variations in the browser are also deleted if the centered thumbnail of the main browser is changed. This is reasonable, since a mix of variations from different source thumbnails would be confusing.

Moreover, the variation browser offers two ways to transfer depicted variations to the main browser. Button seven exchanges the currently centered variation with the according main browser thumbnail. This means that the latter is not simply discarded, but moved to the variation browser. This allows to easily reverse the exchanges. Alternatively, button eight simply transfers the centered variation from variation to main browser in a "cut and paste" manner. This allows to hold different depictions of the same feature in the main browser. This is useful if different visualizations of one feature are required for various combinations of structures.












**Selected Thumbnails Area:** The purpose of the selected thumbnails area is to provide an overview of the selected thumbnails and an easy way to access them (see Figure 3.17). Whenever a main browser thumbnail is selected, an according image is also added to the selected thumbnails area. Due to that the user may constantly keep track of the single structures which are part of the current combined TF and rendering. Thereby, images shown in the selected thumbnails area are always up to date, showing current depictions of the associated thumbnails. This means, whenever color, opacity or view on a structure is changed, not only the main browser thumbnail, but also the according image in the selected thumbnails



**Figure 3.15:** Presentation of possible actions in the main browser of the CF based exploration area and their effects. The discussed actions are: browsing, selection/deselection, change of color and/or opacity and variation creation for color and opacity. Each action and its effects are summarized in a colored rectangle labeled with a bold headline. Moreover, each action correlates with an accordingly numerated button. The numbers can be found in the mentioned headlines and below the buttons. The corresponding arrows with solid line point to examples of potentially affected manifestations. Arrows with dotted lines point to manifestations being only affected under certain circumstances. For example, a thumbnail must be selected that a change of its color affects the combined TF and image.











**Figure 3.16:** Presentation of possible actions in the main browser of the CF based exploration area and their effects. The discussed actions are: deletion and recreation. Each action and its effects are summarized in a colored rectangle labeled with a bold headline. Moreover, each action correlates with an accordingly numbered button. The numbers can be found in the mentioned headlines and below the buttons. The corresponding arrows with solid line point to examples of potentially affected manifestations. Arrows with dotted lines point to manifestations being only affected under certain circumstances. For example, a thumbnail must be selected its deletion affects the combined TF and image.

Action	Button icon (no.)	Context menu	Mouse click	Mouse Wheel	Keyboard
Move one left	 (2)	-	LMB on left third	Wheel down	Arrow left
Move one right	 (3)	-	LMB on right third	Wheel up	Arrow right
Fast left	 (1)	-	CTRL + LMB on left third	-	CTRL + Arrow left
Fast right	 (4)	-	CTRL + LMB on right third	-	CTRL + Arrow right
Select/Deselect	 (5)	Entry 1	LMB on center third	-	Enter or Return
Delete center	 (6)	Entry 3	-	-	Delete
Delete selected	 (7)	Entry 2	-	-	CTRL + Delete
Open color picker	 (8)	Entry 4	MMB	-	CTRL + C
Create color variations	 (9)	Entry 5	SHIFT + RMB	CTRL + Wheel up	CTRL + Arrow up
Create opacity variations	 (10)	Entry 6	CTRL + RMB	CTRL + Wheel down	CTRL + Arrow down
Recreate thumbnails	 (11)	Entry 7	-	-	CTRL + R

**Table 3.1:** Listing of all possible ways to perform different main browser operations via buttons, context menu as well as mouse and keyboard. Buttons for a certain action are represented by small according images and have an associated number which can be seen in brackets. In order to perform an action via the context menu a user must do a RMB click on the main browser widget and subsequently select the according entry with the LMB in the upcoming list. The number of this entry is registered in the context menu column. For the performance of actions using mouse clicks and wheel, the according columns specify: the required keyboard modifier, the mouse button/wheel direction and as necessary the required mouse cursor position on the main browser widget. If no position is mentioned the click may be performed anywhere on the main browser widget. For keyboard actions the according column simply lists the modifiers and buttons to be pressed.



**Figure 3.17:** Presentation of possible actions in the variation browser and selected thumbnails area of the CF based exploration area and their effects. Discussed variation browser actions are: browsing, deletion as well as use and transfer of thumbnail variations. Discussed selected thumbnails area actions are: fast access and deselection of main browser thumbnails. Each action and its effects are summarized in a colored rectangle labeled with a bold headline. Moreover, each action correlates with an accordingly numerated button. The numbers can be found in the mentioned headlines and below the headline. The corresponding arrows with solid line point to examples of potentially affected manifestations. Arrows with dotted lines point to manifestations being only affected under certain circumstances.

Action	Button icons (no.)	Context menu	Mouse click	Mouse Wheel	Keyboard
Move one left	 (2)	-	LMB on left third	Wheel down	Arrow left
Move one right	 (3)	-	LMB on right third	Wheel up	Arrow right
Fast left	 (1)	-	CTRL + LMB on left third	-	CTRL + Arrow left
Fast right	 (4)	-	CTRL + LMB on right third	-	CTRL + Arrow right
Exchange variation	 (7)	Entry 1	LMB	-	Enter or Return
Transfer variation	 (8)	Entry 2	CTRL + LMB	-	CTRL + Enter or Return
Delete center	 (5)	Entry 3	-	-	Delete
Delete all	 (6)	Entry 4	-	-	CTRL + Delete

**Table 3.2:** Listing of all possible ways to perform different variation browser operations via buttons, context menu as well as mouse and keyboard. Buttons for a certain action are represented by small according images and have an associated number which can be seen in brackets. In order to perform an action via the context menu a user must do a RMB click on the variation browser widget and subsequently select the according entry with the LMB in the upcoming list. The number of this entry is registered in the context menu column. For the performance of actions using mouse clicks and wheel, the according columns specify: the required keyboard modifier, the mouse button/wheel direction and as necessary the required mouse cursor position on the variation browser widget. If no position is mentioned the click may be performed anywhere on the variation browser widget. For keyboard actions the according column simply lists the modifiers and buttons to be pressed.

area is adapted. For a large number of selected thumbnails it may happen that the selected thumbnails area does not provide enough space to depict all of them. In this case a horizontal scrollbar occurs as long as required and allows to determine the thumbnails to be depicted. As soon as a thumbnail is deselected the according image also disappears from the selected thumbnails area.

Besides the usual way of deselecting thumbnails in the main browser, it can also be done by a single RMB click on the according image in the selected thumbnails area. This is a very convenient possibility to remove currently unrequired structures from the combined depiction without having to browse the main browser. Likewise, a LMB click on an image in the selected thumbnails area offers a practical way to center the associated thumbnail in the main browser. This allows to easily switch between combined features and to focus on the work with them without being distracted by browsing efforts. The change from one center to a new one is depicted as a series of "slide change" animations as usual.

#### **3.5.1.4 Analysis and Comparison of Cover Flow and Data Mountain Based Exploration Area**

In the *DM based exploration area* all thumbnails are placed on a skew reference plane in 3D space. One of its big advantages is the possibility to provide a good overview of all or a desired group of thumbnails. It can be obtained by individually arranging them via a simple 2D interaction scheme and adjusting the camera view accordingly. These features allow to quickly find and access thumbnails on the reference plane. However, the DM based exploration area incorporates a fundamental weakness: the bad recognizability of structures on thumbnails too far away from the camera. Therefore, it is necessary, to frequently switch between a close and far view. Especially when browsing a large amount of thumbnails "in a good view" this weakness becomes obvious as constant switching requires time and takes the focus from the actual TF design work. The preferred view function mitigates this issue, although there is still considerable potential for improvements. A further important feature of the exploration area is the creation of variation sets. The DM based approach allows to generate them as multiple variation circles, clearly contrasted from the other thumbnails on the reference plane. The overview of all options and the possibility to easily test various combinations of different depictions of structures are the great benefits of this approach. Unfortunately, the circle arrangement also suffers from the general visibility problem of the DM. On the one hand, a certain distance from the camera is required in order to be able to show all variations at once. On the other hand, the recognizability of single structures becomes worse with an increasing distance. Moreover, one should not create too many circles at the same time in order to avoid confusion.

Briefly: The DM based exploration area provides a good overview and great freedom for testing TF combinations. However, the navigation which is required to switch between thumbnails in a good view is elaborately.

The *CF based exploration area* uses three spaces in which thumbnails are arranged along one dimension. The main browser is one of them and basically contains all structures to work with. It displays only three thumbnails at a time from a fixed view and does not provide any means for an individual arrangement. The great benefit of this approach is the clear visibility of structures shown on the displayed thumbnails. Moreover, it provides a simple and intuitive browsing scheme allowing to easily and efficiently switch between thumbnails in a good view. Since there is no need to perform any view adjustments, full concentration on the actual work of TF design is possible. The downside of this concept is its inability to form groups of thumbnails and to adjust the view on a desired selection. For large numbers of thumbnails it may therefore be difficult to quickly access those of interest. For example, two structures to be combined may be depicted on thumbnails far apart in the CF slide stack. This requires a user to search for these thumbnails in order to access them and additionally spend the time to flip the slides between them. To remedy these shortcomings the selected thumbnails area has been introduced. It allows to group and therewith gives an overview of thumbnails of interest for the current combination and provides fast and simple access to them. It should be noted, that this area only allows to form one



	<b>DM</b>	<b>CF</b>
View	Arbitrary thumbnails and a desired amount of them can be depicted at once.	Only a limited number of certain thumbnails can be depicted at once.
	View on thumbnails can be flexibly adjusted.	There exists a predefined view on thumbnails.
	A good visibility of structures requires individual camera adjustments.	The predefined view causes a permanently good visibility of structures.
Browsing	Browsing thumbnails in "good view" is elaborate and therefore distracting.	Innately simple, intuitive and efficient browsing at good view.
	Fast access on arbitrary thumbnails in overview.	Fast access only on selected thumbnails (especially relevant for large numbers).
Arrangement	Individual arrangement of thumbnails possible.	No individual arrangement possible.
	Groups can be arbitrarily formed (by arrangement).	Restricted formation of groups (only one consisting of the selected thumbnails).
Variation Sets	Variations sets may exist for an arbitrary amount of original thumbnails at the same time. For a good visibility view adjustments are required.	A variation set may only exist for one thumbnail at a time. A good view is provided innately.

**Table 3.3:** Comparison of the DM and CF based exploration area.

group at a time and does not allow to incorporate deselected thumbnails. Another issue which the main browser cannot solve satisfactory, is the creation of variations. Therefore, the variation browser has been introduced. It allows to view the original thumbnail in direct comparison with its alterations. Similar to the main browser the variation browser does only allow to see three thumbnails simultaneously. These are always clearly visible and easy to browse. Moreover, there usually exists only a small number of variations which does not require additional overview. Note, that only the variation set for one thumbnail can be contained in the variation browser at a time.

Briefly: The CF based exploration area provides a good view on thumbnails as well as efficient, simple and intuitive means for browsing. In return it is not possible to obtain a freely adjustable overview of arbitrary and individually arranged groups of thumbnails.

Table 3.3 compares the characteristics of the systems analysed above.

### 3.5.2 Combination Area

The main purpose of the combination area is to provide immediate visual feedback on the impact of actions performed in the exploration area. It basically consists of a renderer widget, a visualization of the current combined TF and a number of buttons (See, amongst others, Figure 3.9 or 3.10).

The most important part of this area is the widget depicting the current output generated by the rendering system from Kainz et al. [29] for a given data set and a certain TF. Most of the time the current combined TF resulting from the selected thumbnails is set in the renderer. Thus, the widget shows a composition of the according data set structures. For every adaption, selection and deselection operation in the exploration area a new combined TF is immediately created and passed to the rendering system. As a result, the data set visualization is updated accordingly. This procedure ensures constant feedback on performed actions. The widget also allows to adjust the view on depicted structures in real time in terms

of panning, rotating and zooming. This enables users to interactively explore the data set visualization for a certain combined TF from all directions and distances. As mentioned the depiction in the renderer widget is constantly adapted according to the currently set TF. As a result, it shows an according image whenever an action causes a new TF to be set. The generation of initial or varied thumbnails, as well as the recreation of thumbnails for a new camera view leads to the depiction of a fast sequence of different images. This gives the user a short impression on the upcoming thumbnails and rendering progress, which also reduces the subjective waiting time. In addition, the widget shows the images resulting from color or opacity changes for a short moment. After each of these operations the recent combined TF is set again automatically.

Another part of the combination area is a visualization of the current combined TF. It depicts the location as well as color and opacity distribution of regions of interest in the TF domain. This provides valuable additional information for users with the required background knowledge.

Finally, the discussed area provides two buttons. The "Save Combination" button copies the current TF to the storage area, where it is depicted by a thumbnail. The "Edit TF" button on the other hand opens a TF editor, which provides a more direct manual access to the current combined TF. It depicts a TF as a set of points with certain colors and opacities. These points can be adapted in terms of their color, opacity and position within the value range of the TF space. Hence, the provided functionality is more similar to that of traditional TF editors and well suited to manually fine tune a combined TF. The application of the TF editor for manual adjustments on the foundation of predefined TFs, is however by far more simple than using it to design TFs from the scratch.

### 3.5.3 Storage Area

The storage area allows to store useful combined TFs providing meaningful renderings. They are represented by thumbnails, which are vertically aligned (See Figure 3.9 or 3.10). If the stored thumbnails exceed the space provided by this area a vertical scrollbar appears for as long as required and allows to determine the combinations to be shown. Each of the depicted thumbnails can be selected and deselected by a simple LMB click. Thereby, selected thumbnails are framed by a blue boundary.

In addition, the area provides two buttons. The "*Delete*" button removes all selected thumbnails and associated TFs. The "*Save to File*" button saves all TFs corresponding to selected thumbnails to files on the hard disk. TFs stored in such files can directly be loaded by the renderer.

### 3.5.4 Menu and Toolbar

In addition to the functionality of the already discussed areas, there are a number of essential actions which are provided by the menu and toolbar. These are choosing a data set, initiating, saving and loading a session as well as to resetting the system. Moreover, a widget for the adjustment of precalculation parameters may be accessed.

The *start session* operation initiates the precalculations for a selected data set. As a result of this process, the thumbnails, representing the initial TFs are arranged in the design gallery, which is ready to receive user interactions. In addition to the regular start session operation, there exists another one. Note, that the results obtained from the AH and PRH calculations are automatically stored in an extensible markup language (XML) file for each run. The *second start session* operation uses these results for TF setup if possible. This allows to spare the time for multiple AH and PRH calculations for a data set using the same parameters. This functionality is especially useful when working with multiple TFs since once calculated results from different corresponding volumes may be arbitrarily combined.

By *saving and loading a session* the exact state of the exploration, combination and storage area can be saved and restored. This allows a user to stop and resume the TF design work at any point. Saving a

session works very fast and requires little memory, since all required data is simply stored in an XML file. Loading may take some time, because it requires the thumbnails representing TFs to be rendered again.

The *reset* operation, resets the system. After that, a new data set can be selected and processed.

The menu also provides access to a *settings widget*, which allows to adjust a number of parameters for the calculation of the AH and PRHs. In general the current implementation tries to minimize the number of adjustable parameters in order to reduce the required user knowledge on data sets. Nevertheless, there are some, which have significant impact on the calculation results. Depending on the configuration of a data set different parameter values are beneficial. In order to reduce the requirements for a lot of manual parameter adjustments the system allows to *save* and *load settings*. This enables users to define and reuse settings they found to be best for their data sets as well as to use settings specifically predefined for certain situations.

### 3.5.5 Histogram Visualizations

In addition to widgets representing the design gallery for TF design, there is also one that provides a visualization of the histograms computed throughout the precalculation steps. These are a data sets original histogram, AH and PRHs.

Currently the histogram widget is implemented as a second tab of the mainwindow. Thereby, the original histogram is positioned in the top left and the AH in the top right quarter of the widget. The bottom half is used in order to depict a PRH. Thereby, a user may switch through the single PRH using the *"Next"* and *"Previous"* buttons. In order to take a closer look at certain structures a zooming functionality has been added. By holding the LMB a rectangle can be dragged around a certain part of the histogram. Upon release this part fills the whole area reserved for current histogram. This procedure can be repeated multiple times for a histogram. Each time a user presses the RMB he moves one zoom hierarchy back until the original histogram is depicted again.

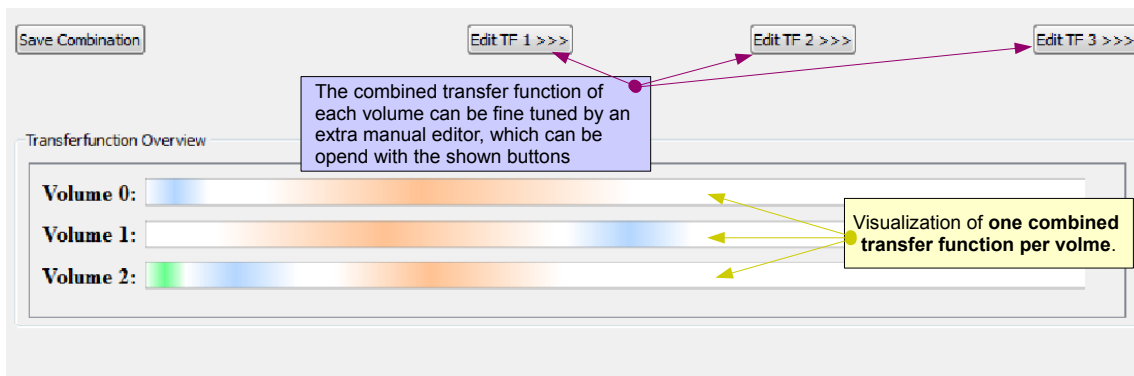
## 3.6 Using Separable Transfer Functions

The presented design gallery allows also to classify features of interest based on multiple arbitrary data value dimensions. Thereby, we do not use one multi-dimensional TF, but multiple one-dimensional ones. Consequently a set of 1D TFs is responsible for a visualization of a multivariate data set. For the purpose of the design gallery a multivariate data set is defined by an arbitrary amount of registered scalar volumes. The evaluation of separable TFs in order to obtain desired visualizations is performed by the render system. Their initial creation is achieved by a cooperation of design gallery and render system. Nothing changes for the user interaction.

In the following we will introduce a basic approach for the application of separable TFs and a preliminary nD prototype extending it. The basic approach provides a full integration between design gallery and renderer. It performs a combined evaluation of all TFs of the single scalar volumes. As a result, all visualized structures result from regions of interest of the same dimensionality. In practice this approach reveals some weaknesses which we address in the preliminary nD prototype.

### 3.6.1 Basic Approach

**Basic Workflow and User Interface Adaption:** In order to work with separable TFs the user must choose two or more registered scalar volumes. The data value range of each of them defines one dimension of the separable TF. Subsequently, the system generates initial separable TFs according to the precalculation process described in the next paragraph. Thereby, the TF evaluation process in the renderer is combined with a homogeneity test in the design gallery. This allows to detect and discard all separable TFs which do not define data set features. The rendered images showing interesting structures,



**Figure 3.18:** Presentation of the user interface changes for separable TFs, shown on the example of a 3D separable TF.

corresponding to the remaining separable TFs, are arranged in the CF or DM based exploration area as described in Section 3.5.1.2 and Section 3.5.1.3. Subsequently, users may adapt and combine the depicted structures as simple as with the one-dimensional application of the design gallery. The user interface is only adapted by two facets. The combination area comprises  $n$  visualizations of combined TFs (see Figure 3.18, yellow box). One for each dimension of the combined separable TF. In addition, the design gallery's user interface allows to open a single editor for manual adjustment of each dimension of the combined separable TF (see Figure 3.18, purple box). Saving separable TFs to disk in the storage area, results in a single TF file for each dimension.

**Separable Transfer Function Precalculation Process:** The precalculation process to create initial separable TFs extends that of one-dimensional ones by several steps. Figure 3.19 illustrates the process on the example of two-dimensional separable TFs. At first AH and PRH are computed in a separate thread for each loaded volume. The resulting mean and deviation values are used to create a vector of regions of interest for each volume. During this process redundant vectors are subsequently removed. So far the usual precalculations have simply been performed separately for each volume. In order to classify data set features by multiple data values we combine regions of interest of different dimensions to form separable TFs. Since we have only  $n$  vectors of unrelated regions of interest we can not determine which regions of interest of different dimensions describe corresponding features first. However, a color test in the renderer in combination with a homogeneity test in the design gallery allows to check whether a set of TFs of different dimensions apply to the same data set's structure. In order to perform these tests we need to create all possible variations of regions of interest at first. Thereby, each variation is a  $n$ -tuple, consisting of one region of interest from each dimension. Each tuple can be considered as a  $n$ -dimensional separable TF. Note, that all instances of one tuple are assigned the same color and opacity to facilitate the color test. The pseudo code for the creation of all  $n$ -tuples out of  $n$  vectors of regions of interest is shown in Listing 3.2. It is an adaption of the code from [16]. We initialize one iterator to the beginning of each vector. Then we increase the iterator of the last vector at each iteration until its last element is reached. In this case the iterator is set back to the vectors first element and its predecessor is incremented by one. In case its predecessor also reaches its last element it is also set back to its vectors beginning and the predecessor of the predecessor is increased. This process continues until the iterator of the first vector is increased and reaches its last element. The procedure of this algorithm can also be imagined as a constantly increasing odometer. As soon as all possible tuples have been created a color test is performed for each of them. Therefor, one separable TF is passed to the renderer at a time. This assigns color and opacity values to each volume's voxels according to the corresponding dimension of the separable TF. Subsequently, all involved volumes are concatenated using the *AND* operator and the additional color test. This means only superimposed regions of different

volumes which have very similar colors and opacities larger than zero are visible in the final rendering (see Figure 3.21).

A homogeneity test is applied in order to determine whether the resulting image depicts a structure or is empty. Thereby, a border pixel, which usually does not belong to a structure is used as a reference. It is compared to all other pixels to find out how much it differs from the reference structure. If this number is above a certain threshold, a structure is depicted on the image. This separable TF is kept and depicted in the design gallery by its corresponding image. If the number of different pixels is below the threshold, the tuple and image are discarded.

Note, that the creation of all possible combinations of regions of interest quickly leads to very large amounts of tuples. Therefore, the removal of redundancies from the single vectors previous to the combination step is especially important for high dimensional cases (see Section 3.4.1).

**Evaluation of Separable Transfer Functions:** Rendering an image based on a separable TF works as explained in the following and shown in Figure 3.21. Note, that the renderer manages a volume and corresponding TF by a special node data structure as explained in Section 4.7. Such nodes are depicted as "Node A" and "Node B" in Figure 3.21. At the beginning of the evaluation process each one-dimensional TF is passed to the renderer. Because all involved volumes must be registered the renderer also knows which of their areas (voxels) are superimposed. Using this information the renderer may concatenate all involved data sets as desired. The default concatenation mode supported by the renderer works with an *OR* operator. It simply visualizes all emphasized regions from both data sets independently. For the evaluation of separable TFs we additionally implemented a concatenation mode based on an *AND* operator and an additional color test in order to determine the visible parts for the rendering. It only depicts superimposed regions which are emphasized by an equal color and have an opacity larger than zero in all loaded scalar data sets. The color test avoids erroneous feature classifications, resulting from unintended interactions between non corresponding regions of interest of different dimensions. A simple opacity test is not sufficient because they frequently occur for multiple regions of interest per TF dimension. The opacity test only looks for superimposed regions that are defined by an opacity larger than zero. Figure 3.20 (a) and (b) illustrates how erroneous feature classifications resulting from a simple opacity test are prevented by the color test. We want to combine two structures A and B, each defined by a two-dimensional region of interest. As long as these structures are considered independently their extend is clearly defined by the overlaps of A1-B1 and A2-B2. However, as soon as their TFs are combined an additional unintended structure resulting from the overlap A1-B2 occurs if we only evaluate the TFs according to opacity overlaps as shown in Figure 3.20(a). This behavior is avoided using an additional color test as shown in Figure 3.20(b) A more detailed explanation of the concatenation implementation can be found in Section 4.11.

### 3.6.2 Discussion of a Preliminary nD Prototype

The basic multi-dimensional approach performs an *AND* concatenation between all loaded volumes to identify overlaps of regions of interest of different dimensions. An additional color test ensures that only overlaps of corresponding regions of interest of different dimensions are rendered as a structure. As a result, we obtain a series of n-dimensional regions of interest which can be combined without leading to erroneously classified features. In principal, the basic approach works. However, it also comprises several weaknesses which are discussed in the following and which we address with a preliminary nD prototype.

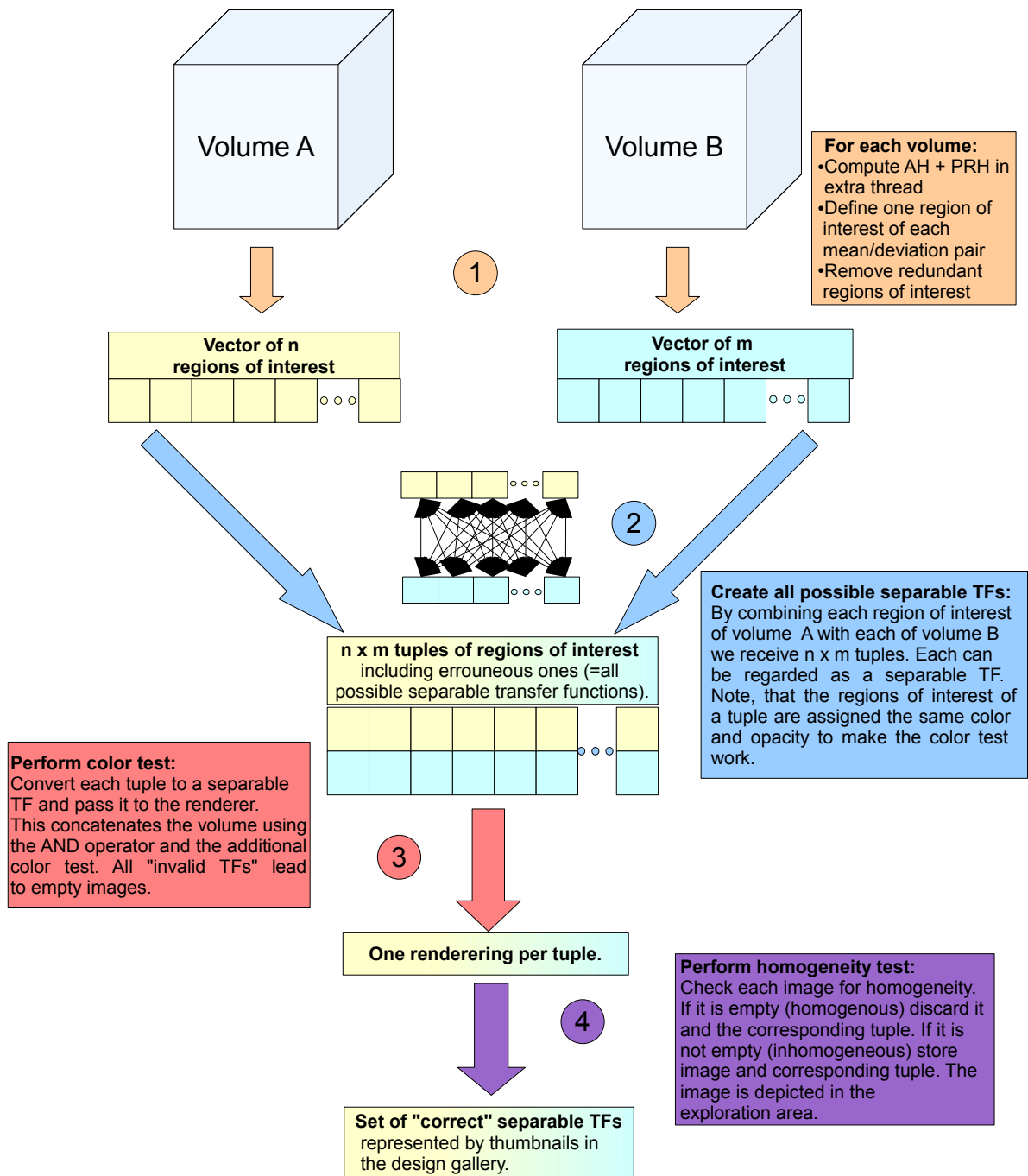
**Analysis of the Basic Approach's Weaknesses:** The first issue of the basic approach concerns the color test. This prevents unintended interactions of non corresponding regions of interest of different dimensions by only highlighting overlaps of equal color. However, a certain one-dimensional region of

```

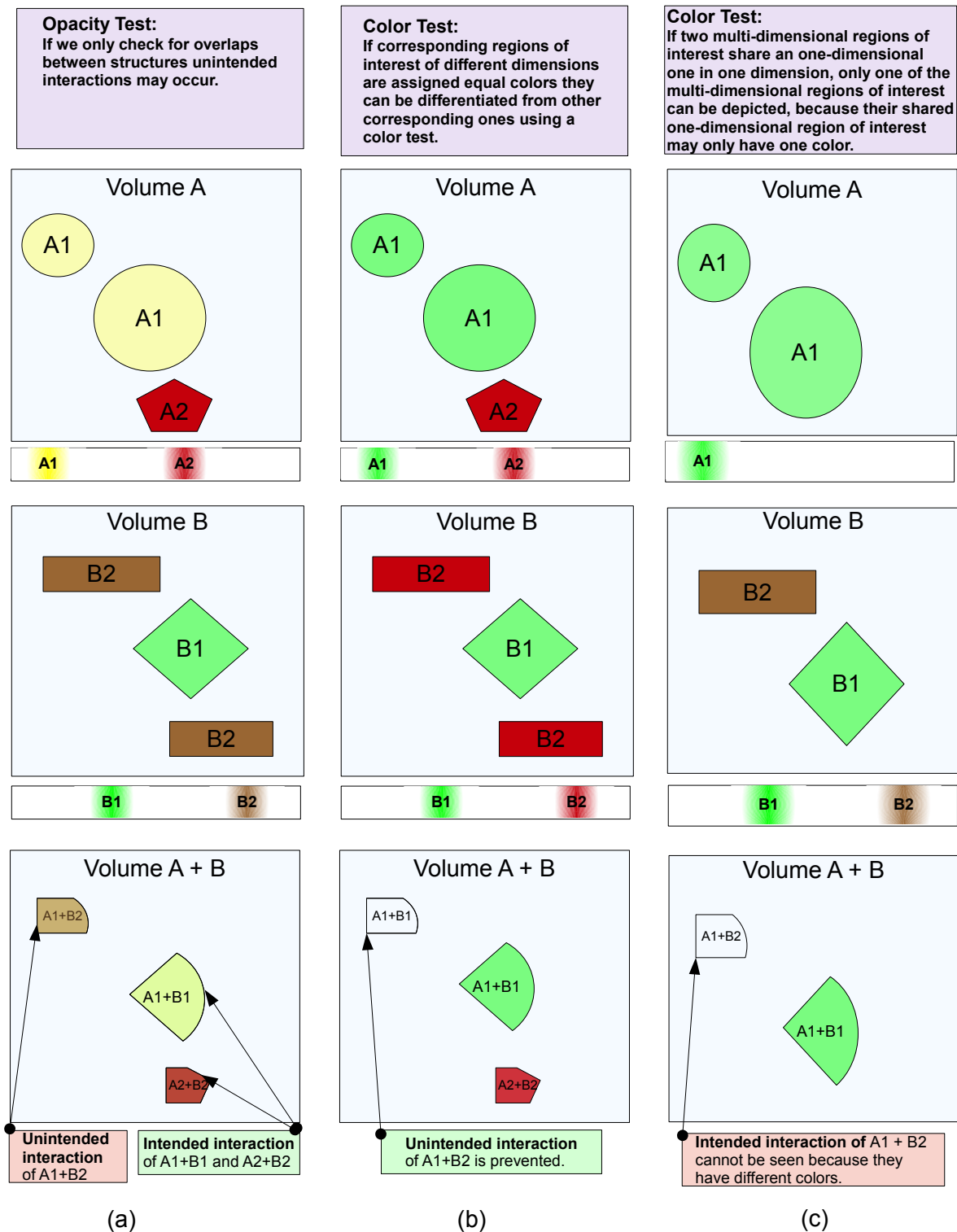
1  /* Vector holding one vector of regions of interest for each volume
2  */
3  vector<vector<RegionData*>> regionDataVectorStorage;
4
5  /* Set all regions of interest in regionDataVectorStorage. Each
6     subvector holds all identified regions of interest of one
7     dimension.
8  */
9  setRegionData(&regionDataVectorStorage);
10
11 /* Initialize a vector holding one iterator to each RegionData
12    vector. At the beginning each iterator points to the first
13    dimension of one dimension.
14 */
15 vector<rdi> itVec(regionDataVectorStorage.size());
16 for(int i = 0; i < itVec.size(); i++){
17     itVec[i] = regionDataVectorStorage[i].begin();
18 }
19
20 int numVecs = itVec.size();
21 /* Create n-Tuples until the last region of interest of the first
22    dimension is reached
23 */
24 while (itVec[0] != regionDataVectorStorage[0].end()) {
25     /* Process the elements currently pointed to by the iterators.
26     */
27     createNTupleFromCurrentIteratorElements(itVec);
28
29     /* Increment iterator of last dimension by 1
30     */
31     ++itVec[numVecs-1];
32
33     /* Starting from the last iterator, check for each iterator if it
34        reached its last element. If so set it back to its first
35        element and increase its "predecessor iterator" by one.
36     */
37     for (int i = numVecs-1; (i > 0) && (itVec[i] ==
38         regionDataVectorStorage[i].end()); --i) {
39         itVec[i] = regionDataVectorStorage[i].begin();
40         ++itVec[i-1];
41     }
42 }

```

**Listing 3.2:** Algorithm for the creation of all n-tuples of n vectors. Each n-tuple contains one element of each vector. The algorithm can be imagined as a odometer spinning through its digit wheels. The algorithm has been adapted to create all possible n-dimensional regions of interest in the design galleries precalculation process.

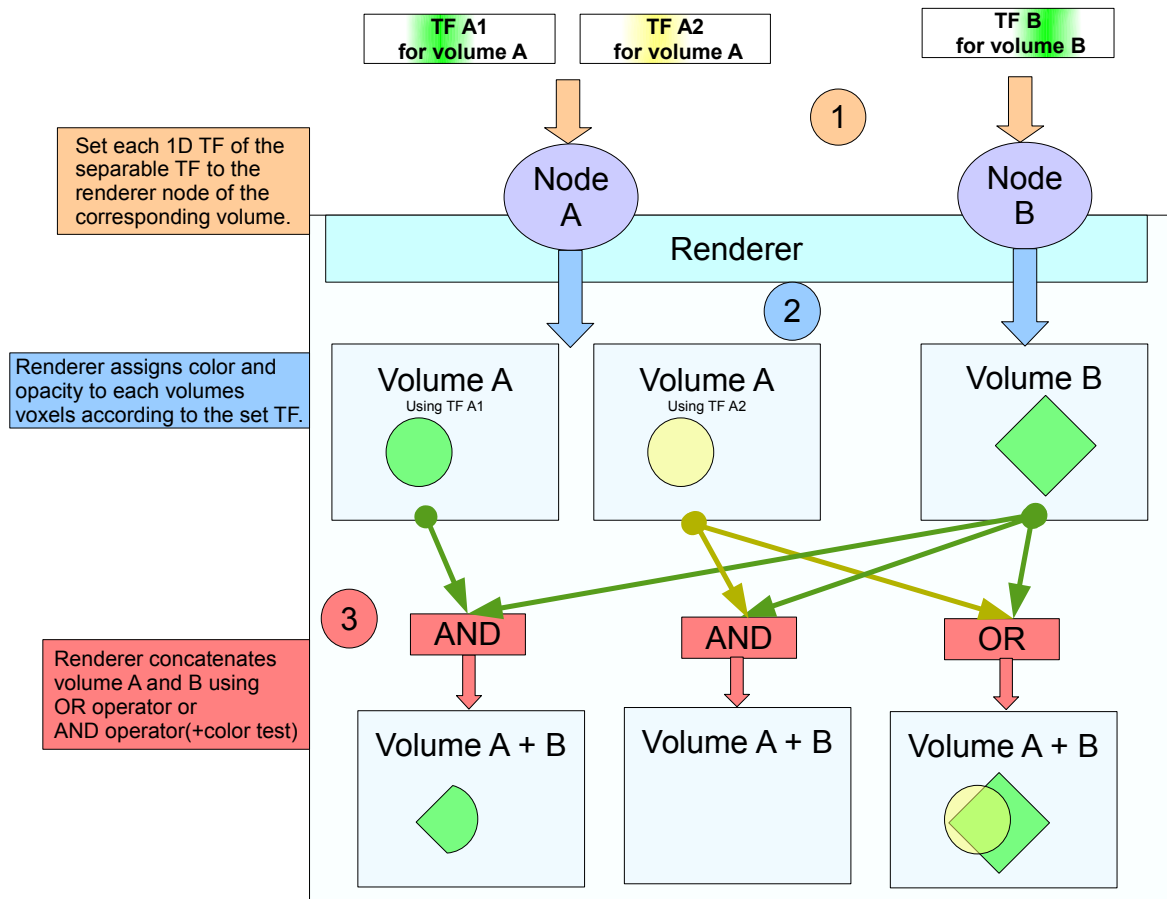


**Figure 3.19:** Presentation of the process to setup initial separable TFs. Note, that step 3 and 4 are performed sequentially for each tuple of regions of interest.



**Figure 3.20:** Color test benefits and restrictions for combining multi-dimensional regions of interest on the example of 2D TFs. (a) Shows how unintended interactions between not corresponding regions of interest of different dimensions occur if we combine TFs solely based on an opacity test. (b) shows how these unintended interactions can be avoided using an additional color test. (c) shows the restrictions of the color test. Note, that each column initially shows two separate volumes with corresponding TFs below and the combination of them at the bottom.





**Figure 3.21:** Abstract presentation of how the renderer creates a data set visualization for a 2D separable TF in dependence of the concatenation mode. Thereby, "Node A" and Node B" represent the data structure managing a loaded volume and the corresponding TF. The *OR* concatenation depicts the emphasized structures from both volumes (see bottom right *Volume A+B*). The color of superimposed parts is mixed. The *AND* concatenation depicts superimposed parts of the two volumes if they have a very similar color and an opacity larger zero (see bottom left *Volume A+B*). Otherwise the *AND* with color test concatenation results in an empty visualization (see bottom center *Volume A+B*).

interest may be part of multiple multi-dimensional regions of interest. Since we only have one TF per dimension the shared region of interest may only have one color. This in turn means that just one multi-dimensional region of interest passes the color test. Hence, only the corresponding structure is depicted. Figure 3.20 (c) illustrates the problem based on two two-dimensional regions of interest with one shared one-dimensional region. The described problem can only be avoided if the two multi-dimensional regions of interest are assigned the same color. This, however, is undesirable since it prevents a clear visual distinction of structures and again allows the unintended overlaps from (a). Similar effects may also occur if two regions of interest of one dimension overlap. By resolving the overlap, the results of the multi-dimensional regions of interest of both one-dimensional ones can be affected.

Another issue concerns the static concatenation of volumes. The basic multi-dimensional approach simply creates all possible n-tuples of regions of interest of different dimensions. If there is an overlap between all of them, the structure it is depicted. This means that only structures resulting from n-dimensional regions of interest may be depicted, where n correspond to the number scalar volumes defining the multivariate data set. Structures which would result from regions of interest of one to n-1 dimensions are lost. In many cases the highest dimensional regions of interest define the "best" structures. However, it may also occur that lower dimensional structures are important or preferable. For example, one scalar volume may only contain a snippet of the other volumes.

**Preliminary nD Prototype Solution:** In the preliminary nD prototype both issues of the basic approach are addressed by two essential adjustments. First, we allow to flexibly adjust how volumes are concatenated at runtime. Second, we use a separate set of TFs with only one region of interest per dimension for each 1D and nD region of interest. This set is separately evaluated.

In order to allow a flexible concatenation of volumes we extend the data structure managing a volume and the corresponding TF by a set of parameters which specify a concatenation mode and the volumes to concatenate with. These parameters can be changed at runtime and induce the ray caster to immediately update the shown rendering. The basic concatenation modes are *OR* and *AND*. Each of them can additionally be combined with an *NOT(AND)*, *NOT(OR)* concatenation. For the *OR* concatenation nothing changes. The sample points of the current volume are simply assigned the  $RGB\alpha$  values of the corresponding TF. For the *AND* operation we additionally have a list of indices of the volumes and corresponding TFs to be concatenated with. This list may also be adjusted at runtime. Only regions for which an overlap exists between the current volume and all of the volumes given by the index list are depicted. The additional *NOT(AND)* and *NOT(OR)* concatenations means that the result of a conventional *OR* or *AND* is only depicted if it does not overlap with a structure of another *OR* or *AND*, whose results are defined by an extra list of volume indices and corresponding TFs. These two concatenation modes are practical if we identify a structure A and a structure B which defines a part of A. Then we can render A with an additional *NOT(AND)* or *NOT(OR)* in order to avoid B from being partly covered by A. A low level description of these concatenation modes can be found in Section 4.11.

Based on the flexible concatenation of volumes each initial structure corresponding to a k-tuple of regions of interest may be rendered and set in the design gallery. Thereby,  $k$  describes the dimensionality of a region of interest and may be a number from one to  $n$ , with  $n$  being the amount of loaded volumes. One-dimensional regions of interest are rendered by employing the *OR* operator. Multi-dimensional ones are rendered via the *AND* operator. In order to create all  $k$ -tuples of regions of interest we initially create all subsets of volumes of the total amount of volumes (which defines the highest possible dimensionality of TFs). This can be done with the algorithm shown in Listing 3.3. For each  $k$  from one to  $n$  we initialize a vector of  $k$  elements holding volume indices from zero to  $k - 1$  (Listing 3.3, lines 8 to 10). Subsequently, the algorithm increases the indices as following in order to obtain all combinations of  $k$  indices (Listing 3.3, line 24 to 37). The index stored at the last position ( $k-1$ ) of the index vector (*setIdxVec*) is increased until it reaches the value  $n-1$ . Then the index stored at the previous position ( $k-2$ ) is increased by one and that at  $k - 1$  is set to  $setIdxVec[k - 2] + 1$ . Similar, if the index at  $k - 2$  reaches  $n - 2$ , the index at  $k - 3$  is increased by one and that at  $k - 2$  is set to  $setIdxVec[k - 3] + 1$ .

This procedure is performed until the index at  $k = 0$  reaches its final position (at  $(n - 1) - (k - 1)$ ) (Listing 3.3, Line 21 to 22). The application of the current index combination happens in line 14 and 15. In our case we use the indices to access the regions of interest of the corresponding volumes and create all  $k$ -tuples of them, based on the algorithm in Listing 3.2 described in Section 3.6.1.

The flexible concatenation of volumes allows to create the initial rendering for all regions of interest of each dimensionality. In order to solve the combination of multiple regions of interest of different dimensionality we set up a set of  $k$  TFs for each  $k$ -tuple of regions of interest of different dimensions and evaluate it separately. This way the concatenation of each multi-dimensional region of interest is independent of other concatenations. Moreover, the color test is not required in this case. Since the renderer allows the shared use of volumes between data structures managing them this approach does not entail additional memory requirements.

```

1  /* Create subsets for all sizes of k from 1 to numberOfVolumes
2  */
3  int n = numberOfVolumes;
4  for(int k = 1; k <= n; k++){
5
6     /* Initialize a vector of k indices.
7     */
8     vector<int> setIdxVec(k);
9     for(int x = 0; x < k; x++)
10        setIdxVec[x] = x;
11
12    /* Generate all subsets of dimensionality k.
13    */
14    bool generateCombinations = true;
15    while(generateCombinations)
16    {
17        /* Use indices to adress the according volumes to create k-
18        tuples from their regions of interest.
19
20        /* Check if the first index has reached its final position. If
21        so stop index traversal. Else increase predecessor index of
22        each index which reached its final position and "set back"
23        indices which reached their final position.
24
25        */
26        if(setIdxVec[0] == ((n-1)-(k-1))){
27            generateCombinations = false;
28        }else{
29            bool increased = false;
30            for(int i = 1; (i<k) && k != n ; i++){
31                if(setIdxVec[i] == ((n-1)-(k-1-i)) && !increased){
32                    setIdxVec[i-1] += 1;
33                    for(int j = i; (j < k); j++){
34                        increased = true;
35                        setIdxVec[j] = setIdxVec[j-1] + 1;
36                    }
37                }
38            }
39            if(!increased)
40                setIdxVec[k-1]+=1;
41        }
42    }
43 }

```

**Listing 3.3:** Algorithm for the creation of all subsets of a vectors elements. For our purpose the vector elements are indices of loaded volumes.

# Chapter 4

## Implementation

This chapter describes the implementation details of the presented system. The chapter starts with a brief introduction of the most important libraries and tools used for the implementation in Section 4.1. Following, a short overview of the system structure is provided in Section 4.2. Subsequently, Section 4.3 introduces implementation details of the data centric approach based on AH and PRHs. Section 4.4 introduces the data structures to represent TFs in the presented program. Moreover, it explains how they are set up using the AH and PRH calculation results and how they can be combined. After that, Section 4.5 and 4.6 present the implementation of the DM and CF based exploration area, respectively. Section 4.7 explains how the rendering system from Kainz et al. [29] is deployed in the design gallery. Subsequently, Section 4.8 shows how the combined TF shown in the combination area is visualized. Following, Section 4.9 explains the composition and functioning of the storage area, holding the saved TFs. Section 4.10 introduces how a current working session, TFs, precalculation settings and intermediate results can be saved using XML. Finally, Section 4.11 explains how the design gallery implementation must be extended to work with separable TFs, as well as the role of the used renderer for the evaluation of separable TFs.

### 4.1 Tools

The presented design gallery was developed on Windows 7, using the programming language C++ and the development environment Visual Studio 2008. Moreover, a series of further tools have been used. They are listed and briefly described in the following. Closer remarks are mentioned throughout the following sections as required. This selection of tools also allows a platform independent compilation and usage of the design gallery.

**Cmake:** CMake is a "[...] cross-platform, open-source build system. CMake is a family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files. CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice." [36]. For this work it is used to create a Visual Studio 2008 project file including the source code and library files required for the design gallery system.

**Insight Toolkit:** "ITK is an open-source, cross-platform system that provides developers with an extensive suite of software tools for image analysis. [37] ITK is implemented in C++. ITK is cross-platform, using the CMake build environment to manage the configuration process." [38]. For this work ITK is applied in the AH and PRH precalculation steps.

It is used to load data sets of the of Mayos Analyze format [58], split them as well as to compute and process histograms of them.

**Qt:** "Qt is a cross-platform application and UI framework. It includes a cross-platform class library, integrated development tools and a cross-platform IDE. Using Qt, you can write [...] applications once and deploy them across many desktop and embedded operating systems without rewriting the source code." [64]. Basically, the whole design gallery consists of Qt derived classes. Qts primarily purpose for this work is to provide a graphical user interface for the design gallery. Moreover, it is also used for internal functionality such as the communication between system components via Qt signals and slots.

**Qwt:** Qwt provides "Qt Widgets for Technical Applications. [...] The Qwt library contains GUI Components and utility classes which are primarily useful for programs with a technical background." [71]. In the context of this work it is used to create histogram visualizations of the original histogram, AH and PRHs resulting from a loaded and processed data set.

**Coin3D:** "Coin3D is an OpenGL based, retained mode 3D graphics rendering library. It is implemented in C++ and publicly released with the source code open for your perusal. The application programmer's interface (API) is fully compatible with SGI's Open Inventor, the de facto standard 3D graphics API for complex visualization applications." [47]. In this work Coin3D is used to implement the DM based exploration area described in Section 3.5.1.2.

**Quarter:** "Quarter is a light-weight glue library that provides seamless integration between Systems in Motions's Coin3D high-level 3D visualization library and Trolltech's Qt 2D user interface library." [46]. Quarter includes a widget to render the 3D scene represented by a Coin3D scene graph. Moreover, it provides the functionality to translate Qt events into Coin3D events and forward them to the according Coin3D library component. In this work Quarter is used to integrate the rendering system from Kainz et al. and the DM based exploration area into the design gallery user interface.

**Compute Unified Device Architecture:** "CUDA is NVIDIAS parallel computing architecture that enables dramatic increases in computing performance by harnessing the power of the GPU (graphics processing unit)." [66]. The main advantage of transferring operations to the GPU lies in their parallel execution. Therefore, parallel computations are significantly accelerated while sequential ones hardly benefit from this approach. Ray casting is a technique which greatly benefits from parallelization (i.e. composition of single rays). This fact is exploited by and contributes to the performance of the rendering system [29] used in the presented framework. The design gallery itself only uses CUDAs API to initialize the renderer and copy resulting images from its memory.

## 4.2 Implementation Structure Overview

The following sections concentrate on the design gallery functions and their implementation. This section provides a short overview of the program structure to facilitate the further discussion.

The origin of the user interface is formed by the usual Qt MainWindow [65] class. It holds the menu, toolbar and a central widget. Moreover, the AH and PRH calculations are performed in extra threads initiated in the MainWindow. The central widget is a QTabWidget [65]. This holds one or more HistogramWidgets and one DGWidget. A HistogramWidget is responsible for visualizing the original histogram, AH and PRHs for one data set. However, this visualization does not form the focus of this work. Hence, it is only pointed out that the implementation is based on Qwts provided histogram example. The most important Qwt classes used are the QwtPlot, QwtPlotZoomer and QwtPlotPanner [72].

```

1  typedef signed short PixelType;
2  static const unsigned int Dimension = 3;
3  typedef itk::Image<PixelType, Dimension> InputImageType;
4  typedef itk::ImageFileReader<InputImageType> ReaderType;
5  typedef itk::AnalyzeImageIO ImageIOType;
6
7  ReaderType reader_ = ReaderType::New();
8  reader_>SetFileName(inputimg);
9  ImageIOType::Pointer analyzeImageIO_ = ImageIOType::New();
10 reader_>SetImageIO(analyzeImageIO_);
11 try{
12     reader_>Update();
13 } catch (itk::ExceptionObject & e)
14 { ... }

```

**Listing 4.1:** Loading a data set using ITK.

The DGWidget is the actual main part of the design gallery. It contains all further widgets, forming the exploration, combination and storage area.

## 4.3 Alpha Histogram and Partial Range Histogram

This section deals with implementation issues of the AH and PRH method. They are implemented in single classes which make it easy to utilize them in various applications. Each AH - PRH pair is computed in an extra instance of a class derived from QThread. This prevents the user interface from freezing while the precalculations are performed. Moreover, this implementation is of importance if multiple data sets are analyzed concurrently, which is necessary to work with separable TFs (see Section 3.6 and 4.11). For both approaches the ITK provides important functionality to accomplish tasks such as loading data sets, splitting them, creating histograms and modifying them (see Section 4.3.1). Furthermore, this section discusses why we decided to perform AH and PRHs calculations on the CPU and not on the GPU (see Section 4.3.2).

### 4.3.1 Insight Toolkit Based Calculations

**Loading a Data Set:** Using ITK the task of loading a data set for AH and/or PRH calculation can be accomplished via the class `itk::ImageFileReader` [39] and an appropriate subclass of the abstract superclass `itk::ImageIOBase` [39]. The first is used to manage the reading process. Depending on the data set file format it uses a certain subclass of the `itk::ImageIOBase` in order to perform the actual low level reading task in the background [26]. The code sequence for reading a data set can be found in Listing 4.1.

As a first step the type of input image must be defined (lines 1-3). Thereby, the `InputImageType` defines the properties of the data set used in the program after reading. Subsequently, a reader object is instantiated and the name of the file holding the volume is set (lines 7,8). Moreover, an appropriate subclass of the `itk::ImageIOBase` must be instantiated and handed to the reader (line 9,10). This step could also be omitted, since ITK also provides a factory which automatically generates the correct `itk::ImageIOBase` according to file suffixes. In the current code example the `itk::AnalyzeImageIO` [39] subclass is explicitly instantiated and passed to the reader. It is suitable for the used *Analyze IMAGE FILE FORMAT* [58] of the used data set files. Finally, reading is triggered by calling the *Update* method (line 12).

```

1  typedef itk::RegionOfInterestImageFilter <InputImageType,
   OutputImageType> RegionFilterType;
2  ...
3  OutputImageType::RegionType desiredRegion;
4  desiredRegion.SetSize( size );
5  desiredRegion.SetIndex( startindex );
6  RegionFilterType::Pointer regionFilter = RegionFilterType::New();
7  regionFilter->SetRegionOfInterest( desiredRegion );
8  regionFilter->SetInput( reader->GetOutput() );
9  try{
10     regionFilter->Update();
11 } catch (itk::ExceptionObject & e)
12 { ... }

```

**Listing 4.2:** Splitting the input data set into blocks using ITK.

**Extracting Data Set Blocks:** In order to split the data set for the AH and/or PRH calculation a series of blocks have to be extracted from the original data set as shown in Listing 4.1. The basis for this is provided by the `itk::RegionOfInterestImageFilter` [39], which allows to extract a desired region from a given image. Therefore, a user defines the input image region to be contained in the output image in terms of a start index and size for each dimension (lines 3-5). The defined region must be from a desired `OutputImageType`, which is the same as the `InputImageType` of the data set in this case. As soon as the `itk::RegionOfInterestImageFilter` has been instantiated with the according `InputImageType` and `OutputImageType` the defined region to be extracted as well as the input image are set (lines 6-8). The input image, which is the data set, is obtained as an output from the `itk::ImageFileReader`. Finally, the *Update* method is called in order to obtain valid data from the filter (line 10).

**Creating Image Histograms:** The histogram of an ITK image can be generated using an `itk::Statistics::ScalarImageToHistogramGenerator` [39]. The corresponding code sequence can be found in 4.3. In addition to the input image, the generator requires the desired minimum intensity, maximum intensity and number of bins for the histogram to be generated. The minimum and maximum intensity is computed from the input data set using an `itk::MinimumMaximumImageCalculator` [39] (lines 5-10). From these values the number of histogram bins is derived (lines 13). The usage of the input volume value range is necessary since the single local histograms need to be accumulated after amplification. After all parameters have been determined and set (lines 14-17), the generator is executed by calling the *Compute* method (lines 18). The resulting local histogram is subsequently assigned to a histogram pointer.

Note, that the generator still holds the actual histogram and must not be deleted. In addition, the histogram cannot be modified since it is represented by a const pointer to the histogram contained by the generator. Therefore, it needs to be copied into a non const data structure prior to further computations at first.

**Modifying Histograms:** The most important methods to work with ITK histograms, which are constantly used throughout the AH and PRH calculation are listed in the following:

```

histogram->GetFrequency(bin, 0);
histogram->SetFrequency(bin, value);
histogram->IncreaseFrequency(bin, value);
histogram->GetTotalFrequency();

```

Thereby, line one retrieves the frequency of a certain histogram *bin*. Line two sets the frequency of



```

1  typedef itk::MinimumMaximumImageCalculator<InputImageType>
   MinMaxImgCalculatorType;
2  typedef itk::Statistics::ScalarImageToHistogramGenerator<
   InputImageType> HistogramGeneratorType;
3  typedef HistogramGeneratorType::HistogramType HistogramType;
4  ...
5  intensityRangeCalculator_ = MinMaxImgCalculatorType::New();
6  intensityRangeCalculator_>SetImage( reader_>GetOutput() );
7  intensityRangeCalculator_>ComputeMinimum();
8  intensityRangeCalculator_>ComputeMaximum();
9  int intensityMax_ = intensityRangeCalculator_>GetMaximum();
10 int intensityMin_ = intensityRangeCalculator_>GetMinimum();
11 unsigned int numberOfHistogramBins_ = intensityMax_-intensityMin_;
12 ...
13 HistogramGeneratorType histogramGenerator = HistogramGeneratorType
   ::New();
14 histogramGenerator->SetInput( imageregion );
15 histogramGenerator->SetNumberOfBins( numberOfHistogramBins_ );
16 histogramGenerator->SetHistogramMin( intensityMin_ );
17 histogramGenerator->SetHistogramMax( intensityMax_ );
18 histogramGenerator->Compute();
19
20 HistogramType::ConstPointer histogram = histogramGenerator->
   GetOutput();

```

**Listing 4.3:** Calculating a histogram from a data set block as well as the required parameters using ITK.

a certain *bin* to *value*. Line three is used to increase a current *bin* by *value*. And line four retrieves the total frequency resulting of all histogram bins.

### 4.3.2 CPU Versus GPU Implementation

Histogram calculations on the CPU are very simple. Implementing them efficiently on the GPU, on the other hand, is a very complex task and entails a series of limitations depending on the used hardware. For this reason we decided to perform the AH and PRH calculations on the CPU in this work.

Performing efficient histogram calculations on the GPU using CUDA, requires the usage of a sufficient amount of sub-histograms to avoid performance losses from collisions and a memory with fast access. Unfortunately, the shared memory, which provides the required access speed is very small. As a result, the calculation of large histograms (in terms of bin number and size) requires complex approaches. However, their performance also drops significantly with increasing histogram sizes because of additional algorithm iterations ([79], method 1) or costly read/write operations from/to global memory ([79], method 2) as described in Section 2.1.6.3.

## 4.4 Transfer Functions

### 4.4.1 Representation

An important requirement for TF specification via the user interface is to represent them as thumbnails. By interacting with them a user should be able to easily combine and adapt them. For this purpose the current implementation includes two essential data structures. The first is an easily adaptable and combinable representation of regions of interest in the TF space. The second unifies a TF with a corresponding image and contains all important data related to this representation.

- The `RegionData` class contains all data describing a region of interest in the TF space. This includes a normalized mean and two deviation positions defining the regions extend and a  $RGB\alpha$  value, represented by a `QColor` [65], at each of these points. This representation allows to easily adapt single features. Putting several `RegionData` objects in a vector provides a practical way to represent a transfer function emphasizing multiple structures. In order to create a TF that is defined over the whole TF space and can be passed to the renderer system, the `RegionData` information is used to generate a `QGradient` [65] (done by the `ImageWidget` class explained in the following). This is done by defining a `QGradientStop` [65] for each `RegionData` position value and its corresponding color. The gradient stops are subsequently set in the gradient. For this work a linear gradient is used. It linearly interpolates the color values between the gradient stops. Note, that a vector of `RegionData` objects may contain intersecting regions. Sections 3.4.2 and 4.4.3 describe how these can be resolved before the according `QGradient` is generated. Section 4.7 explains how a `QGradient` is utilized by the rendering system.
- The `ImageWidget` class represents a TF and the corresponding data set visualization. It is derived from the Qt class `QWidget` [65] which can be displayed as a part of the user interface. An object of this class stores and maintains all data related to the TF it represents. The most important data contained by an instance of this class is a vector of `RegionData` objects as well as a corresponding `QImage`. The image is obtained from the rendering system which visualizes the data set according to the `QGradient` passed to it as a TF.

In order to define the appearance of an `ImageWidget` its `paintEvent` function is reimplemented. Therein, a `QPainter` [65] is used to draw the `ImageWidget` depending on its selection state.

For a deselected `ImageWidget` the rendering created from its TF is simply drawn on it. This is done by the following code:

```
QPainter widgetPainter(this);
widgetPainter.drawImage(0, 0, widgetImage_);
```

The `QPainter` is initialized with a pointer to the widget to draw on (the current `ImageWidget`). Subsequently, the painters `drawImage` method is invoked. It is used to draw the output image the renderer generated for the according TF on the `ImageWidget`.

For a selected thumbnail the image resulting from the rendering system is additionally framed by a blue rectangle. This highlights selected thumbnails and is added using the following code:

```
QRect rect(0, 0, width() - 1, height() - 1);
QPen widgetPen(Qt::SolidLine);
widgetPen.setColor(QColor(0, 0, 255));
widgetPen.setCapStyle(Qt::SquareCap);
widgetPen.setJoinStyle( Qt::MiterJoin);
widgetPen.setWidth(5);

QBrush widgetBrush(widgetImage_);

QPainter widgetPainter(this);
widgetPainter.setPen(widgetPen);
widgetPainter.setBrush(widgetBrush);
widgetPainter.setRenderHint(QPainter::Antialiasing, true);

widgetPainter.drawRect(rect);
```

Thereby, line one defines a rectangle of the widgets size to be drawn. Lines two to six define how the boarder of that rectangle, framing the widget, should be drawn. The next line initiates a `QBrush` [65] with the rendered image. It defines the fill pattern of the rectangle. In this case this

is the rendered image. The following lines define a painter to draw on this ImageWidget, set the pen and brush defining how the rectangles border and interior are drawn and initiate the drawing process.

In addition to the paintEvent, the ImageWidget also reimplements a mousePressEvent. Whenever a LMB click is performed on an ImageWidget, its selection state is changed and the paintEvent is called in order to update the visualization accordingly. Note, that the storage area is the only user interface component which directly integrates ImageWidgets as a part of the user interface. In the CF based exploration area it serves as data structure for TF related data and provides a visualization to be shown on a slide. In the DM based exploration area it is only used as a data structure to manage all data related to a TF.

#### 4.4.2 Setup

Section 3.4.1 provides a basic description of how initial TFs are set up based on the mean and deviation values obtained from the AH and PRH calculation. This section shows how the approach is applied to the specific data structures used for the presented implementation.

In order to create the initial ImageWidgets representing TFs the following steps are performed:

- A RegionData object is created for each mean/deviation pair. The mean and deviation positions are normalized and stored in the data structure. In addition, the color for all points defining the region is determined depending on the mean position within the value range. A mean points opacity is also assigned according to its position, while the deviation points opacity is set to zero.
- Subsequently, the system searches for equal or very similar RegionData objects depending on whether one-dimensional or multi-dimensional separable TFs are set up and deletes redundant ones. Two RegionData objects are equal if they have exactly the same mean and deviation value. In order to find similar ones the criterion already used for merging PRH (see Equation 3.4) is applied with a very small multiplier. The removal of similar RegionData objects is important for the use of separable TFs, to avoid overly large amounts of RegionData combinations (see Sections 3.6.1 and 4.11).
- Following, one ImageWidget object is created for each RegionData object. According images are generated by the renderer and stored in the ImageWidget objects. These visualizations are used to represent TFs in the design gallery and allow the user to interact with them. It should be noted, that each ImageWidget only holds one RegionData object at this point. How a user may combine multiple initial regions of interest, resulting in a new ImageWidget containing several RegionData objects is described in Sections 3.4.2 and 4.4.3.

#### 4.4.3 Adaption and Combination

Section 3.4.2 describes the approach to adapt and combine TFs in the design gallery. This section shows how the approach is applied to the specific data structures used for the presented implementation.

In order to adapt a TF in the exploration area a new color or opacity value must be set in the RegionData object contained in the ImageWidget representing the TF. Subsequently, an according QGradient is created for the updated RegionData object. It is handed to the renderer which creates an according image to replace the previous one stored in the ImageWidget.

A combined TF is also represented by an ImageWidget. However, in contrast to the ImageWidget objects representing initial TFs this one usually contains not only one, but a whole series of RegionData objects. Hence, the combined ImageWidget can basically be created as described in the following:

- Copy the RegionData object of each selected thumbnail in the exploration area and store it in the corresponding vector of the combined ImageWidget.
- Generate a QGradient based on all RegionData objects and pass it to the renderer.
- Store the resulting image in the combined ImageWidget.

In order to resolve overlapping regions that may occur, the approach explained in Section 3.4.2 is applied. Thereby, the deviation points reaching into the other region are deleted and replaced by a new intermediate point. The new point is stored in both of the deleted deviation value points. However, only one point is marked to be displayed. Points marked not to be displayed are not adopted in the QGradient which is used to pass a TF to the rendering system.

## 4.5 Data Mountain Based Exploration Area

### 4.5.1 Implementation and Integration Basics

For the implementation of the DM based exploration area and its integration in the design gallery the Coin3D and Quarter library play an important role. Coin3D is used to create a DM based interactive 3D scene as described in Section 3.5.1.2. In the background this scene is represented by a scene graph. The integration of the Coin3D scene into Qts user interface is realized via Quarters main class, the QuarterWidget [46]. Since it is derived from the QGLWidget [65] it can simply be integrated into the Qt based user interface like any other QWidget. Moreover, it provides an area to render a 2D image of a Coin3D scene on. The rendering process is controlled by a SoRenderManager [47] which is a member of the widget. The QuarterWidget also allows convert Qt events into Coin3D events. These are subsequently handled by the SoEventManager [47] which is also a member of the widget. In order to render a Coin3D scene on a QuarterWidget, its graphs root node must be set in the widget. This can be done using the following code:

```
dataMountainViewer_ = new QuarterWidget();
dataMountainViewer_->setSceneGraph(overallRoot_);
```

### 4.5.2 Coin3D Scene Graph Basics

Before the further implementation of the DM based exploration area is discussed, some important information about Coin3D scene graphs is recalled:

Basically, a Coin3D scene graph consists of an ordered collection of nodes describing a 3D scene. Thereby, each node contains certain information. Shape nodes for example describe an objects shape. Property nodes may define surface material, texture and geometric transformations affecting subsequent shape nodes. Group nodes consolidate a number of sub-nodes. As soon as a scene graph has been created various actions (subclasses of SoAction [47]) can be applied to it. Starting from a certain node, the whole scene graph below it is traversed from top to bottom and left to right. Thereby, a nodes reaction to various actions depend on its type. Some node types do not react to certain actions at all. Most reactions affect a traversal state which exists for every action. It consists of a set of elements or parameters at a certain time. Depending on the attended nodes this state is modified throughout the scene graph traversal. Examples for actions are the SoGLRenderAction [47] which renders the scene described by the graph, the SoGetBoundingBoxAction [47] which determines the bounding box of a certain scene graph part or the SoHandleEventAction [47] which delivers events to scene graph nodes. [90]

In the context of actions the order of traversal and therefore the arrangement of nodes is very important, since following nodes inherit the traversal state of previous ones. According to the traversal order,

following nodes are those to the right and below a certain node. The importance of the node order can be illustrated on the example of the render action. Property nodes for example change the actions traversal state and therewith the rendering appearance of all following shape nodes. A color node for example reacts on a render action by changing the stored color value in the traversal state. Shape nodes render themselves using values from the traversal state. As a result, all shape nodes following a color node appear in the corresponding color in the scene. However, in some cases a traversal state should only be changed for a certain sub-part of the scene graph. Therefore, the SoSeparator [47] node which is a special group node plays an important role. It stores the actions traversal state before it traverses its sub-nodes and restores it after that. As a result, traversal state changes in the sub-graph do only affect other nodes in this part of the graph. [90]

Another substantial aspect of Coin3D, which plays an important roll for the presented implementation are callback functions. They "[...] provide an easy mechanism for introducing specialized behavior into a scene graph or prototyping new nodes without subclassing. A callback function is a user-written function that is called under certain conditions." [90].

### 4.5.3 Scene Graph Composition

The DM based 3D scene is represented by the scene graph depicted in Figure 4.1. A SoSeparator forms the overall root node, which is set in the QuarterWidget. The connected SoSelection [47] node below, provides a convenient way to select and deselect shape nodes below it. It is used to select and deselect the cube nodes representing thumbnails. The SoSeparator below the over all root forms the starting point of the sub-scene-graph depicting the gnomon in the scene. The SoEventCallback [47] node under the SoSelection is used to catch and handle certain events resulting from user interactions with the renderer canvas. The SoSeparator below the SoSelection comprises the nodes representing the DM plane and the thumbnails on it. The SoBaseColor [47] node is positioned to the very left of this sub-graph and therefore affects the appearance of all shape nodes occurring in it. The SoSeparator to the right contains all nodes which determine the DMs ground plane. Each of the subsequent SoSeparators comprises a set of nodes defining a single TF thumbnail. In this part of the scene graph the SoSeparators are especially important to prevent an impact of "earlier" thumbnails nodes on "later" ones. Note, that the term thumbnail in the context of the scene graph is used to describe the SoSeparator and all its sub-nodes forming one thumbnail in the scene from now on.

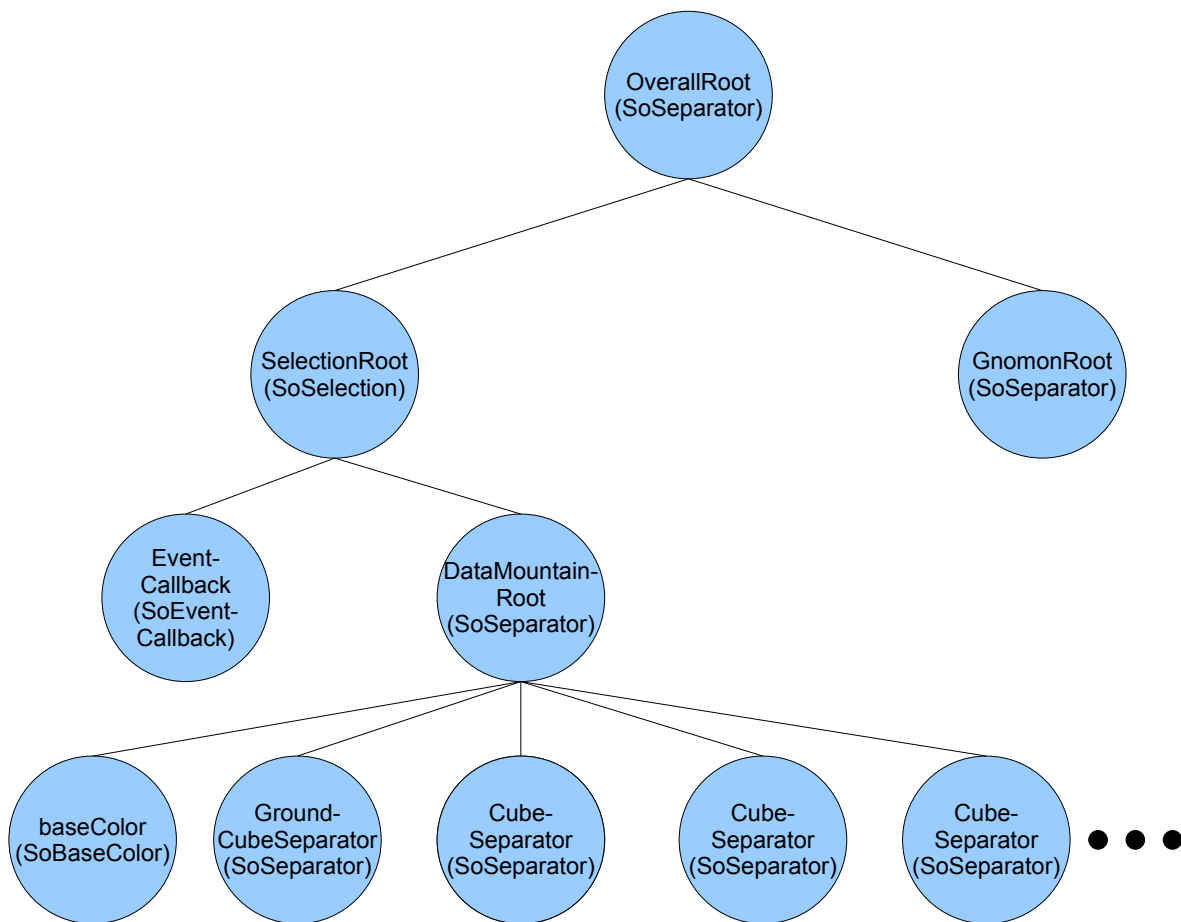
**Ground Plane Composition:** The skew DM plane forming the underground to place and move thumbnails on is represented by four nodes comprised by a SoSeparator. These are a SoPickStyle, SoTexture2, SoTransform and SoCube node [47] (see Figure 4.2 (a)).

The SoPickStyle node is used to make the cube forming the ground plane shape unpickable. This can be done by setting the according pick style:

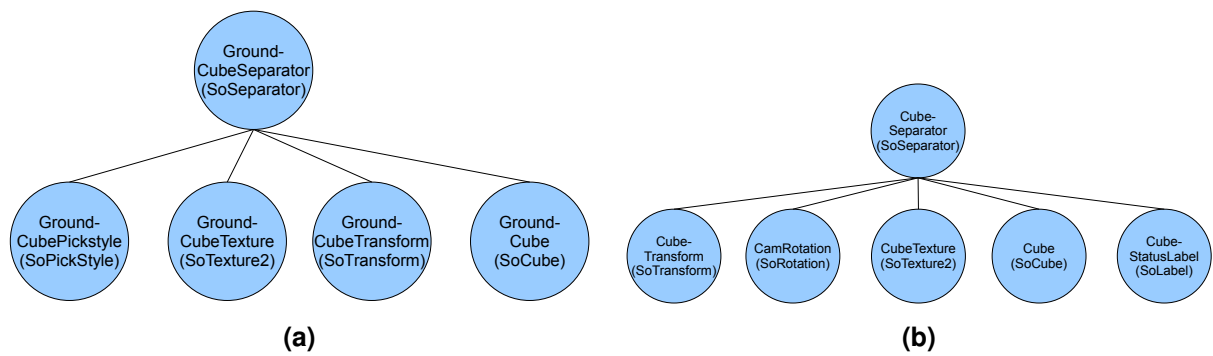
```
SoPickStyle* gCPickStyle = new SoPickStyle();
gCPickStyle->style.setValue(SoPickStyle::UNPICKABLE);
```

This is important since the ground plane should not react to any user interactions. Using the SoTexture2 node a 2D texture represented by a SoSfImage [47] is mapped on the ground plane. The texture provides passive landmarks for the DM. In this case, an arbitrary image file loaded from the hard disk is used as texture. It can be loaded by the following code:

```
SoTexture2* gCTexture = new SoTexture2;
gCTexture->filename.setValue(SbString("bubbles.jpg"));
```



**Figure 4.1:** Overview of the Coin3D scene graph depicting the DM based 3D scene.



**Figure 4.2:** Overview of scene graph nodes representing the DM ground plane (a) and a DM thumbnail (b).

The SoTransform node is used to rotate the plane about the x-axis to make it skew:

```
SoTransform *groundCubeTransform = new SoTransform;
groundCubeTransform->translation.setValue(0, 0, 0);
groundCubeTransform->rotation.setValue(SbRotation(SbVec3f(1, 0, 0)
, 0.6f));
```

Finally, the SoCube represents the plane shape itself. In order to make it flat and large enough its height is reduced to a minimum, while its depth and width are extended:

```
SoCube* groundCube = new SoCube;
groundCube->height = 0.1;
groundCube->depth = 100;
groundCube->width = 100;
```

**Thumbnail Composition:** Each DM thumbnail, depicting an interesting data set structure, is represented by a set of nodes consolidated under a SoSeparator. These are a SoTransform, SoRotation, SoTexture2, SoCube and SoLabel [47] (see Figure 4.2 (b)). The SoTransform node is used to set the corresponding cubes position and rotation in the scene. It is set at a number of occasions. This includes the initial thumbnail arrangement in a circle on the DM, the creation of a variation circle or the completion of a dragging operation. The adjacent SoRotation node is used to rotate the cube according to the camera orientation. Therefore, the nodes rotation is connected to the camera orientation using the following code:

```
SoRotation* cubeCamRotation = new SoRotation;
cubeCamRotation->rotation.connectFrom(&camera_->orientation);
```

This means the nodes rotation value is immediately updated when the cameras viewing direction changes. As a result, the corresponding thumbnail cube is oriented according to the cameras viewing direction at all times. This actually means its front face looks away from the camera. However, this is not a problem since the texture is applied to all sides of a cube. This means the camera always has an optimal view on the texture on the cubes back side. The texture node is used to map a rendering resulting from the data set visualization with a certain TF on the corresponding cube. The SoCube provides the actual thumbnail shape, which is textured, positioned and oriented according to the previous nodes. In order to make it look like a sheet in front of the camera its depth is reduced to a minimum, while its height and width are extended to an appropriate size. Finally, the SoLabel node is used to store the current status of the thumbnail:

```
SoLabel* statusLabel = new SoLabel();
statusLabel->label.setValue("StartPosition");
```

The default status of a thumbnail is "StartPosition" which remains for as long as the thumbnail cube is positioned on the reference plane. It can also be set to "InCenter", "InCircle" and "InPreferredView" for thumbnails which are in the center of a variation circle, are a variation in the circle and those in preferred view, respectively. These states are important since they determine the reactions to certain input events applied to thumbnails (as explained in Section 3.5.1.2).

In order to associate the thumbnails in the 3D scene with the data structure holding the actual TF data, a QMap [65] of the form  $Qmap < SoSeparator*, ImageWidget* >$  is used. This means each separator node comprising the explained sub-nodes representing a thumbnail in the scene is related to an ImageWidget. This map is referred to as *thumbnailMap* in the following.

**Gnomon Composition:** The gnomon composition and function will not be described in this work, since it does not belong to the galleries core functionality. However, it should be said that it has been implemented based on the tutorial presented on the open inventor home page [86].

#### 4.5.4 Event Handling

User interactions form the starting point for any operation performed in the DM scene. The following explains how events are delivered to a Coin3D scene intergraded in Qt's graphical user interface using Quarter.

In Coin3D an instance of the class SoEventManager forms the starting point for event handling. The manager obtains Coin3D events from the used graphical user interface binding, which translates input events accordingly. In this work Quarter is the Qt binding for Coin3D. The QuarterWidget provided by it, holds a pointer to the SoEventManager instance. Hence, Coin3D events attain the event manager as following: A user interacts with the render canvas of the QuarterWidget. As a result, an instance of a QEvent [65] subclass emerges. Depending on its type it is translated into an according instance of a SoEvent [47] subclass via Quarter. The Coin3D event is subsequently passed to the event manager instance, which basically provides two ways for handling the incoming events. The event can either be passed to Coin3Ds State Chart XML (ScXML) based state machine [47] or directly to the scene graph. The presented system makes use of both approaches. Therefore, an according navigation state must be set in the event manager. This can be done via the QuarterWidget using the following code:

```
dataMountainViewer_->getSoEventManager()->setNavigationState(
    SoEventManager::MIXED_NAVIGATION);
```

The first method provides a convenient way for "non-model" user interaction in the 3D scene. This primarily includes the navigation with the camera by panning, rotating and zooming. The reaction (change of state) to user interactions can simply be defined in an XML file. Coin3D already includes a file describing a default behavior. For this work it has been slightly adapted suit the interaction needs in the DM scene: A LMB drag in the empty space does now pan the scene instead of rotating it while the MMB drag rotates it.

In order to directly pass a Coin3D event to the scene graph it is wrapped into a SoHandleEventAction which is applied to the scene graph. The action traverses the graph in the usual order from top to bottom and left to right until a node is found which handles the delivered event. Many nodes such as shape, property, transformation, light and camera nodes ignore the event handle action. Group nodes, including separators, only forward it to its children from left to right until one accepts it. Nodes potentially handling events used in this work are the SoSelection and SoEventCallback as well as manipulator nodes.



The following initially discusses these nodes event handling behavior individually. Based on that we can define how events reach their "handling node" in the scene graph at hand.

In order to explain a manipulators event handling behavior its role in the scene must briefly be explained. In this work, a manipulator node replaces a thumbnails SoTransform node while it is selected. During this time it spans a physical object (geometry) around the thumbnail cube. This serves as a user interface to change a thumbnails position on the DM. When an event handle action, resulting from pressing the LMB, reaches a manipulator node it initially checks whether its geometry was picked (= LMB click on the geometry). This information is retrieved from the event handle action. If it was picked its sets the event handled and grabs all following mouse events. This means they are directly delivered to the manipulator without previous traversal of the scene graph. Otherwise the event handle action will further traverse the scene graph. When the user releases the LMB the according event is also forwarded to the manipulator. At this point the manipulator checks whether its geometries position changed from mouse press to the corresponding release event. If so, it stops grabbing and sets the event handled. If the position did not change it also stops grabbing but applies the event to the scene graph. [90]

The SoSelection allows to select and deselect shape nodes below it with a LMB click, or more accurate release. When an event handle action of a LMB release reaches this node, it simply forwards it to its child nodes like a usual group node. However, if none of the child nodes handles the action the selection node does, if a shape node below it was picked.

The SoEventCallback node allows to implement an event handling behavior that is not delivered by the selection or manipulator nodes. The node catches event handling actions and forwards the contained event to callback functions written by the programmer. For the presented system the node only includes a callback function which handles mouse events. It is added to the node by the following code:

```
generalEventCallback_ = new SoEventCallback;
generalEventCallback_>addEventCallback(SoMouseButtonEvent::
    getClassTypeId(), mouseEventWrapper, (void*) this);
```

In the code above the first argument makes the added callback function react to mouse events only. The second argument is the callback function itself. Since this is only a static wrapper function a pointer to the object holding the SoEventCallback is the third argument to be able to access its members.

Note, that the event handle action does not stop the scene graph traversal automatically after a callback function of a SoEventCallback has been executed. If this is desirable the programmer has to set the event handled manually. For this work this is done for events initiating the variation creation, color change or preferred view. LMB press and release events are forwarded unhandled since they are intended for the selection and manipulator nodes.

Based on the descriptions above the traversal of event handle actions through the DM scene graph can be defined by the following order. At first the action reaches the SoSelection node. This forwards it to its child nodes. There it initially arrives at the SoEventCallback node. It handles certain mouse events and stops the scene graph traversal of the event handling action in this case. Events not handled by this nodes callback further traverse the graph and may attain one or more manipulator nodes. In this case only LMB press/release are of further interest. If the action reaches a picked manipulator for which the mouse position changed between press and release event, both these events as well as their intermediate motion events are handled. Otherwise it will "return" to the selection node at some point. This checks whether the incoming event results from a LMB release and if one of the shape nodes below it is picked. If this is the case, the SoSelection handles the event. Otherwise the event is not handled at all.

In a nutshell: A LMB press-drag-release on a cube is handled by the manipulator node if the cube is selected. A LMB release on a cube is handled by the selection node. All other TF design specific mouse actions are handled by the SoEventCallback node. The camera navigation is performed via the ScXML based state machine.

### 4.5.5 Function Overview

In the following the single functions which are initiated via Coin3Ds event handling system or buttons are presented.

**Select/Deselect:** Thumbnail selection and deselection in the design gallery is based Coin3Ds SoS-election node [47]. Section 4.5.4 explains how events reach this node and under which circumstances they are handled. This paragraph explains the nodes default behavior for events to be handled and how it is used and extended for the needs of this work.

By default the selection node manages a list of paths to selected shape objects (selection list). The standard behavior for adding and removing picked objects to and from the list is defined by the selection policy. For the purpose of this work the "SoSelection::TOGGLE" policy is used. It allows to keep multiple objects selected at a time. A LMB click on the unselected thumbnail selects it, while clicking on a selected one deselects it.

In order to add visual feedback to the selection and deselection of thumbnails an instance of the class SoBoxHighlightRenderAction [47] is used. It replaces the SoGLRenderAction it is derived from and which is usually applied to the scene graph to render the scene. In contrast to the conventional render action it additionally adds highlighted bounding boxes around selected objects in the rendered scene. In order to replace the default render action by the highlighted one, it is passed to the render manager:

```
dataMountainViewer_ -> getSoRenderManager() -> setGLRenderAction (new
    SoBoxHighlightRenderAction);
```

In order to extend the selection nodes standard mechanisms three callback functions, invoked at different occasions and times, are added to it. These are the selection, deselection and pick filter callback function and can be added using the following code:

```
selectionRoot_ -> addSelectionCallback (selectThumbnailWrapper, (void
    *) this);
selectionRoot_ -> addDeselectionCallback (deselectThumbnailWrapper, (
    void*) this);
selectionRoot_ -> setPickFilterCallback (pickFilterCB);
```

The selection callback is called upon object selection (LMB on unselected object). It is used to update the selection status of the ImageWidget associated with the selected thumbnail and to update the combined TF and visualization accordingly. Since the selection callback only receives the path to the selected cube as a parameter, the according thumbnail separator must be extracted from it. Using it the associated ImageWidget can be retrieved from the *thumbnailMap*. In addition, the selection callback replaces the thumbnails transformation node with a manipulator node, if it is in "StartPosition". This allows to drag the thumbnail on the DM plane. The deselection callback is called upon object selection and basically reverses this processes. It sets the associated ImageWidget to deselected, updates the combined TF and visualization and removes the manipulator if set.

For the explanation of the pick filter callback it is important to recall that the selection and deselection of shape objects is based on adding/removing a path to/from the selection list. This path describes the way from the selection node to the shape object (thumbnail cube) on which the LMB has been clicked in the scene graph. It is also passed to the selection and deselection callback, where it is the basis for further actions. Usually this does not cause any problems. However, in this work a selected thumbnail gets spanned by the geometry of the added manipulator. As a result, a LMB click to deselect a cube actually leads to a selection of the manipulators geometry covering the thumbnail cube. In order to avoid this behavior the pick filter callback can be used. It "is invoked when an object is picked and about to be selected or deselected" [90]. It obtains the path to be used by the selection/deselection callback and can

modify it before it is forwarded. Hence, we use the pick filter callback to check whether the end of a path is a manipulator node, which is the case at each deselection. If so, the path gets truncated by the last part leading to the manipulator and is extended to the corresponding cube node. This can be done very easy since manipulator and cube are contained by the same thumbnail separator. The corresponding code can be seen in the following:

```
int manipIndex = pickedPath->getIndex(pickedPath->getLength()-1);
filteredPath = pickedPath->copy(0, pickedPath->getLength()-1);
filteredPath->append(manipIndex + 3);
```

The first line retrieves the manipulators index in the set of the thumbnail separators child nodes. The second line copies that part of the original path that reaches from selection node to thumbnail separator node. The third line finally extends the path to point to the according thumbnails cube node, which is 3 positions to the left of the manipulator.

**Dragging Thumbnails:** For the individual arrangement of thumbnails on the DM plane Coin3D manipulators and corresponding draggers are used. They are essentially described as following:

Manipulators are subclasses of other nodes (such as SoTransform [...]) that employ draggers [...] to respond to user events and edit themselves. [...] A manipulator replaces a node in the scene graph, substituting an editable version of that node for the original. When interaction finishes, the original (non-editable) node can be restored. Each manipulator contains a dragger that allows the user to edit its fields. [...] A dragger is a node in the scene graph with specialized behavior that enables it to respond to user events. All Inventor draggers have a built-in user interface, and they insert geometry into the scene graph that is used for picking and user feedback. [90]

In the context of this work manipulators of the type SoTransformBoxManipulator [47], derived from the class SoTransform are used to replace thumbnail transformation nodes. Once added to the scene graph, the manipulator has the same impact on the graph as the node it substitutes, since it is derived from it. The used manipulator type deploys SoTransformBoxDraggers [47] to provide users with a user interface geometry to modify its own transformation values in terms of scaling, rotation and translation. At its removal from the graph a manipulator is replaced by a transformation node adopting the manipulators current transformation value. As a result, user transformations performed on thumbnail cubes are maintained. The manipulator for a thumbnail is already initialized at thumbnail creation. It is stored in the corresponding ImageWidget which also contains the methods for adding and removing it to and from the scene graph. These methods are called in the selection nodes selection and deselection callback, respectively. The code for adding and removing a manipulator is shown and described in Listing 4.4 and 4.5.

The SoTransformBoxManip and its dragger were chosen for the task of moving thumbnails on the DM plane, since they allow to translate an object in all directions. Their scaling and rotation functionality, however, is not required for thumbnail arrangement. Therefore, the according geometry parts of the dragger are removed immediately after the manipulators initialization. This can simply be done by the following code:

```
SoDragger* tbManipDragger = currentTranfBoxManip->getDragger();
tbManipDragger->setPart("rotator1", NULL);
tbManipDragger->setPart("rotator2", NULL);
tbManipDragger->setPart("rotator3", NULL);
tbManipDragger->setPart("scaler", NULL);
```

```

1  SoSeparator* thumbnailSeparator = (SoSeparator*) selectionpath->
   getNode(2);
2  ...
3  SoTransform* replaceXformNode = (SoTransform*) thumbnailSeparator
   ->getChild(0);
4  SoPath *pathToXform = selectionpath->copy();
5  ...
6  pathToXform->pop();
7  int xfIndex = thumbnailSeparator->findChild(replaceXformNode);
8  pathToXform->append(xfIndex);
9
10 thumbnailXformPath_ = pathToXform;
11 thumbnailManip_->replaceNode(pathToXform);
12 ...

```

**Listing 4.4:** Code for adding a manipulator to the scene graph. Lines 1 to 8 retrieve the path to the transformation node to be replaced by the manipulator. Lines 1 and 3 fetch the separator of the currently processed thumbnail and the corresponding transformation node, respectively. In line 4 we obtain the path to the currently selected geometry, which is the thumbnails cube node. Lines 6 to 8 truncate this path to point to the thumbnail separator and extend it to lead to the corresponding thumbnail transformation node. Line 11 uses the path to replace the transformation node by an according manipulator. Line 10 saves it for the manipulators removal.

```

1  ...
2  thumbnailManip_->replaceManip(thumbnailXformPath_, NULL);
3  ..

```

**Listing 4.5:** Code for removing a manipulator from the scene graph. The replaceManip function replaces the manipulator at the passed thumbnailXformPath\_ position by a passed transformation node which adopts the manipulators transformation value. The second "NULL" parameter means that a new transformation node is created. The path to the manipulator is the same as that used for adding the manipulator in Listing 4.4. It has been stored in the ImageWidget.

In addition, another important adaption of the draggers standard behavior is required in order to restrict the thumbnail translations to the DM plane. Therefore, a motion callback is added to the manipulators dragger. It is called whenever the dragger is moved and performs a number of checks, to ensure the thumbnails bottom always stays in touch with the DM plane. A prerequisite for these checks is to know the current bounding box of the moved thumbnail as well as that of the DM plane. They are obtained by applying a `SoGetBoundingBoxAction` [47] the separator nodes of this structures. Based on the bounding box information, and the thumbnail dimensions the required checks to restrict the thumbnail movement are performed as explained in Section 3.5.1.2, paragraph "Dragging Thumbnails on the Data Mountain Plane".

**Variations - Creation, Interaction, Resolution:** Figure 3.14 in Section 3.5.1.2 introduces the working process for using system generated variations. This paragraph explains the implementational details of each stage included. These are the creation of the variation circle, the exchange of variations with the center thumbnail as well as the translation back to the DM plane.

For the creation of a variation circle the corresponding thumbnail must be in "StartPosition", on the DM plane. In case the manipulator is active it is deactivated now. Then the thumbnail to be varied is translated closer to the camera by inserting a new, temporary `SoTranslation` node before its transformation node. The new translation value is computed by subtracting the translation value of the center thumbnails transformation node from the camera position in each dimension, and halving the resulting values.

```
SbVec3f camPos = camera_>position.getValue();
SbVec3f centerPoint = centerCubeTransform->translation.getValue();
SbVec3f newCenterPoint = (camPos - centerPoint)/2;
```

Moreover, the status of the thumbnail is changed to "InCenter".

Subsequently, a new thumbnail, consisting of the usual nodes (Figure 4.2 (b)), is created for each variation and arranged around the centered one. In addition, an `ImageWidget` containing a varied TF is created for each variation thumbnail and is as usual associated to it via the `thumbnailMap`. In order to relate a center thumbnail with its variations, a further `QMap` of the form `QMap < SoSeparator*, SbList < SoSeparator* > * >` (`circleMap`) is utilized. Thereby, the key is a center thumbnails separator pointer, while the value is a pointer to a list of according variation thumbnail separator pointers.

Another important part of the variation creation is to set the thumbnail nodes parameters. The rotation node is connected to the camera orientation and the cube node is created using the usual dimensions. The texture image is obtained from a rendering based on its `ImageWidgets` TFs. This is defined by the same region of interest in the TF space as the center thumbnails, but assigns a different color or opacity to this region. The `SoLabel` is set to the status "InCircle". The transformation nodes value, which defines a variation thumbnails position and orientation, is computed by multiplying a series of translation and rotation matrices. The way these matrices are determined and contribute to the total transformation is described in the following. A prerequisite for their determination are the following computations:

- Computation of a rotation axes to rotate variations about the center thumbnail. It is defined by a directed vector, calculated by subtracting the coordinates of the center thumbnail from those of the camera. See Listing 4.6, line 2.
- Computation of a rotation interval to be able to distribute thumbnails in uniform angles around the center thumbnail. It is computed by dividing the whole circle (360°) by the amount of variation thumbnails. See Listing 4.6, line 3.
- Definition of an initial accumulated rotation value and a corresponding counter rotation value. The first starts at zero and is iteratively updated by the rotation interval for each thumbnail to determine

their position in the variation circle. The counter rotation is the negated accumulated rotation. It is used to compensate the rotation used to position a thumbnail in the circle and therewith maintains the right up direction. See Listing 4.6, line 4 - 5.

Subsequently, starting from a thumbnail cube located in the center of the scenes coordinate system, the following transformations are required in order to bring it in an appropriate position in the circle around the center thumbnail:

- Counter rotate cube about the rotation axes to maintain up direction. The counter rotation value is the negated accumulated rotation interval. It is the negated rotation interval for the first thumbnail and is decreased by the negated rotation interval for each following thumbnail. See Listing 4.6, lines 10 - 12.
- Translate thumbnail in the Y dimension according to the desired circle radius. This position forms the starting point for locating thumbnails in the circle using the accumulated rotation. Thereby, the circle radius is chosen according to the amount of variations. See Listing 4.6, lines 15 - 17.
- Rotate thumbnail cube about the rotation axes to the desired position in the variation circle. The rotation value is the positive accumulated rotation interval. It is the rotation interval for the first thumbnail and is increased by the positive rotation interval for each following thumbnail. See Listing 4.6, lines 20 - 22.
- Translate thumbnail cube (including previous transformations) to the position of the center thumbnail. This translation actually consists of two. These are the translation values of the center thumbnails *SoTransform* and its new *SoTranslation* node (the latter was added to move it closer to the camera). See Listing 4.6, lines 30 - 31 and 25 - 27, respectively.

After having determined the required transformations for a thumbnail cube, their matrix representations are unified into a single one by multiplying them. This single matrix is now set in the *SoTransform* matrix of a certain variation thumbnail. See Listing 4.6, lines 34 - 39.

After the variation circle has been constructed, circle thumbnails may be exchanged with the one in the center. Therefor, their transformation and label nodes are swapped. Moreover, the temporary translation node used to place the center thumbnail closer to the camera is moved to the new center thumbnail. Therewith, their positions in the scene are changed. Of course the according entry in the *circleMap* is also replaced with a new adapted one, where the new center separator is the key and the old center separator is in the list. This approach allows to exchange variations with the center without any interaction with the *ImageWidgets* in the background.

Finally, a variation circle can be resolved. Therefor, circle thumbnails and associated *ImageWidgets* are retrieved via the *circleMap* and *thumbnailMap*, respectively. Both are deleted. The new center thumbnail is conveniently moved back to the previous original thumbnails position on the DM plane, by removing the temporary translation node. In addition, its status is changed to "StartPosition".

**Preferred View:** Moving thumbnails in preferred view and back works analog to the translation of the center thumbnail of a variation circle. However, the added translation node moves thumbnail cube much closer to the camera in preferred view. Moreover, the thumbnails status is set to "InPreferredView".

**Change Color:** Equally as in the CF based exploration area the new color is obtained using a *QColorDialog* [65]. Using it, the TF contained by the *ImageWidget* associated to the thumbnail the user applied the "change color" action to is updated. The new TF is subsequently passed to the renderer to obtain and according image. This is used as a new thumbnail texture.

```

1 // Precalculations
2 SbVec3f rotationAxes=camPos - centerPoint;
3 float amountOfRotation=(1.5707963f * 4)/(float)numberOfVariations;
4 SbRotation accumulatedRotation=SbRotation(rotationAxes, 0);
5 SbRotation accumulatedCounterRotation= SbRotation(rotationAxes, 0);
6
7 ...
8
9 // Counter rotation
10 currentCounterRotation=SbRotation(rotationAxes,-amountOfRotation) *
    currentCounterRotation;
11 SbMatrix counterRotMatrix;
12 currentCounterRotation.getValue(counterRotMatrix);
13
14 // Translation in y dimension by circle radius value
15 float distance = max(10, 6*(numberOfVariations/7));
16 SbMatrix surroundingTranslMatrix;
17 surroundingTranslMatrix.setTranslate(SbVec3f(0, distance, 0));
18
19 // Rotate to circle position
20 currentRotation = SbRotation(rotationAxes, amountOfRotation ) *
    currentRotation;
21 SbMatrix surroundingRotMatrix;
22 currentRotation.getValue(surroundingRotMatrix);
23
24 // Translate by center thumbnails temporary translation nodes
    translation value
25 SoTranslation* centerTranslation = (SoTranslation*)
    centerCubeSeparator->getChild(0);
26 SbMatrix tempTranslMatrix;
27 tempTranslMatrix.setTranslate(centerTranslation->translation.
    getValue().getValue());
28
29 // Translate by translation value of center thumbnails
    transformation node
30 SbMatrix asCenterCubeTranslMatrix;
31 asCenterCubeTranslMatrix.setTranslate(centerCubeTransform->
    translation.getValue().getValue());
32
33 // Compute unified matrix
34 SbMatrix newTransformMatrix = asCenterCubeTranslMatrix *
    tempTranslMatrix * surroundingRotMatrix *
    surroundingTranslMatrix * counterRotMatrix;
35
36 // Create new transformation and insert into scenegraph
37 SoTransform* newTransform = new SoTransform;
38 newTransform->setMatrix(newTransformMatrix);
39 currentSeparator->insertChild(newTransform, 0);

```

**Listing 4.6:** Code for positioning DMs variation thumbnail in a circle around the center thumbnail. Lines 2 - 5 show the required precomputations. Lines 9 - 36 show the computation of the single rotation and translation matrices contributing to the total transformation. Lines 38 - 41 show the computation of the unified matrix, how it is set in a transformation node and how this is inserted at the corresponding position under the thumbnail separator.

**Reset View:** In order to be able to reset the initial camera view its position and orientation is simply stored after the creation of the initial thumbnails before any user interaction took place.

**Recreate Thumbnails:** In order to recreate all thumbnail images according to the new renderer view the following steps must be performed. First of all, the thumbnail separators are obtained by applying a search action on the root separator directly above them. For each of them the according ImageWidget is retrieved. The TF stored in it is set in the renderer. The resulting image is subsequently used to update the texture of the Coin3D thumbnail. The texture of the Coin3D thumbnail is obtained by applying a search action on the according thumbnail separator. Hard coded access to the separators texture child is not sufficient since the texture nodes position may vary. The center thumbnail of a variation circle for example has an additional translation node at the beginning.

**Delete:** By pressing the delete button, all selected thumbnails are deleted. They are obtained using the selection list managed by the selection node. The list originally holds the paths to the thumbnail cubes, but they can simply be truncated to point the according separators. The associated ImageWidgets are obtained using the *thumbnailMap*. As soon as a thumbnail and its ImageWidget are retrieved they and their map entry are deleted. A thumbnail is deleted from the scene by removing its separator from the scene graph. Thumbnails which are in the a variation circle or in its center require additional considerations. For a circle thumbnail, the corresponding entry in the list hold by the *circleMap* must also be deleted. For a center thumbnail its associated variation thumbnails and their related ImageWidget as well as the according *circleMap* entry must be deleted.

## 4.6 Cover Flow Based Exploration Area

The CF based exploration area consists of the three vertically arranged components already introduced in Section 3.5.1.3: the main browser, variation browser and selected thumbnails area.

The heart of the first two areas is formed by a CF design based QWidget implementation. These widgets are called the main and variation browser. In addition, each of these areas contains a number of buttons, realized via a QToolBar [65] holding a number of QActions [65]. These provide one way to interact with the browsers (see table 3.1 and 3.2). The third area is implemented by depicting a number of widgets, each associated with a single selected main browser ImageWidget, in a QScrollArea [65].

### 4.6.1 Main- and VariationBrowser

The Main- and VariationBrowser class are both derived from the "PictureFlow" [23] class and extend it to meet the needs of TF design in the presented system. The PictureFlow class is an implementation of the CF design in Qt. It is derived from a QWidget and holds a QVector of QImage pointers representing the slides to be viewed. The class basically reimplements the QObject paintEvent to draw its current center slide and one or more to its left and right on itself according to the CF design. Moreover, it provides a set of functions to add and remove slides as well as to browse them. Thereby, it implements movements to different slides as smooth animations. In order to invoke browsing functions the class also provides reimplementations of the QObject keyPressedEvent and mousePressEvent.

The PictureFlow class functionality provides well suited means to browse thumbnails. However, in order to provide the functionality of the main and variation browser in the context of the presented design gallery system, as described in Section 3.5, a whole range of extensions is required. These include synchronizing "PictureFlow" slides with associated ImageWidgets, adding communication channels to the rest of the design gallery, extending the possibilities for user interaction and providing methods to react on them. All these extensions are implemented in the Main- and VariationBrowser class and will



be presented in the following paragraphs. Additionally, the PictureFlow class itself has also been slightly extended by two functions to add and remove slides at arbitrary positions.

#### 4.6.1.1 Synchronizing Slides and ImageWidgets

The actual purpose of the CF slides is to depict structures resulting from data set visualizations with certain TFs. Therefore, a vector of ImageWidget pointers (containing TFs and corresponding renderings) has been added to the Main- and VariationBrowser class. For each ImageWidget one slide is added to the corresponding QImage vector from the PictureFlow class. Thereby, an ImageWidget and its corresponding slide are accessible via the same index in their vectors. All performed user operations do always affect both corresponding vector entries. In order to obtain an image to be depicted as a slide an ImageWidget visualization is rendered into a QImage via a QPainter. This is realized using the following lines of code:

```
QPainter *painter = new QPainter();
QImage *image = new QImage(QSize(currentImageWidget->width(),
                                currentImageWidget->height()),
                            QImage::Format_ARGB32);

painter->begin(image);
currentImageWidget->render(painter);
delete painter;
```

Thereby, a QImage of the ImageWidgets size and appropriate format is initialized at first. Handing the image to the painters begin function allows it to start painting on it. Note, that a painter can draw on any kind of QPaintDevice including QImages. Calling the ImageWidgets render function renders it into the painters paint device. The resulting QImage is set as a slide. Note, that the ImageWidgets QImage is not directly used as a slide since it does not necessarily correspond to the ImageWidgets final appearance. This is emphasized by a blue frame when selected.

#### 4.6.1.2 Extended Event Handling

In order to allow users to perform TF design related actions according to personal preferences the PictureFlow classes event handling must be extended. The goal is to provide the possibilities for interaction listed in Table 3.1 and 3.2.

This can be achieved by reimplementing a series of event handling functions in the Main- and VariationBrowser class. This includes the mousePressEvent, keyPressEvent which have already been overwritten in the PictureFlow class as well as the wheelEvent and contextMenuEvent function. The corresponding QEvent subclasses are the QMouseEvent, QKeyEvent, QWheelEvent and QContextMenuEvent [65]. Starting from the reimplemented functions the actions listed in table 3.1 and 3.2 can be executed. Depending on the user input, event handling functions call methods in the Main-/VariationBrowser and communicate with other system components to initiate required actions. To differentiate varying user inputs within one event handling function the parameters provided by the single QEvent subclasses are used. Mouse press events contain information about the mouse cursor position as well as the pressed button and modifiers. Wheel events additionally hold a value specifying the amount of wheel rotation in a positive or negative direction (up or down). The context menu event reacts to the right mouse button and its position. The key press events react to keyboard keys and additional modifiers.

#### 4.6.1.3 Communication with the Design Gallery

In order to allow the main and variation browser to interact with other system components Qts signal and slot system is used [65]. It provides a simple way for communication between arbitrary objects and replaces callback functionality of other toolkits. The basic idea is that signals are emitted at certain

events and slots are called as a response to them. Thereby, a slot reacting to certain signal can be in the same or in another object. In order to make a slot react to a signal they must be connected. Thereby, one signal can be connected to multiple slots and vice versa. Note, that slots also represent normal member functions. A closer description can be found in Qts documentation [65]. How signals and slots are used for the integration of the Main- and VariationBrowser into the remaining system is shown throughout the function overview in the next paragraph. Note, that signals and slots are also used to connect user interface buttons to functions of an application. Thereby, the Qt buttons emit a series of default signals.

#### 4.6.1.4 Function Overview

In the following the single main and variation browser actions, initiated by the described event handling functions are briefly introduced. Thereby, the simultaneous update of ImageWidgets and slides as well as the communication via the signal and slot system is constantly used.

**Browsing (Main Browser):** Browsing in the main browser always follows the same scheme. A PictureFlow function is called to move to the corresponding slide. Moreover, two signals are emitted. One to clear the current variation browser slides and ImageWidgets and one to update the "progress display" in the DGWidget.

**Select/Deselect (Main Browser):** This function sets the according ImageWidget to selected/deselected, causes it to repaint itself (with or without blue frame) and updates the according slide. In addition, it emits a signal to add/delete an according PreviewWidget in the selected thumbnails area (see Section 4.6.2). Moreover, a signal to update the combined TF and the according visualization is emitted.

**Deletion (Main Browser):** This function deletes the according ImageWidget and associated slide from their vectors. In addition, two signals are emitted to the DGWidget. One to induce the deletion of the corresponding PreviewWidget (only if ImageWidget was selected) and one to update the "progress display". This is done for one or many thumbnails, depending on the kind of delete. The DGWidget also updates combined TF and visualization if required.

**Create Variations (Main Browser):** This function emits a signal with the index of ImageWidget to be varied and the variation mode (color or opacity) to the DGWidget. This creates ImageWidgets with color or opacity varied TFs and according renderings and sets them in the variation browser (which also sets the according slides). After that, the combined TF is set in the renderer again.

**Change Color (Main Browser):** This function also emits a signal with the corresponding ImageWidgets index. The DGWidget opens a color picker, adopts the ImageWidgets RegionData by the picked color and creates an according rendering by setting the new TF in the renderer. Based on it, the ImageWidgets QImage and main browser slide are updated. The DGWidget also updates combined TF and visualization if required.

**Recreate Thumbnails (Main Browser):** This function emits a signal to the DGWidget. This deletes all slides from main and variation browser (not the ImageWidgets). Then it creates a new rendering for each ImageWidget used to update their QImage and subsequently the according slides. After that, the combined TF is set in the renderer again.

**Browsing (Variation Browser):** This works analog to browsing the main browser. Of course the signal to delete variation browser ImageWidgets and slides is not emitted in this case.

**Exchange Variation (Variation Browser):** To perform this function the variation browser emits a signal to the DGWidget. This exchanges the pointers of the centered main and variation browser ImageWidget, but keeps their vector positions unchanged. In addition, each of the corresponding slides is updated according to the new ImageWidget. If the old main browser ImageWidget was selected, some additional actions must be performed. First, the selection status of both ImageWidgets is swapped. Second, the PreviewWidget is updated according to the new ImageWidget associated to it and therewith its appearance (see Section 4.6.2 for a closer explanation).

**Transfer Variation (Variation Browser):** In order to perform this function the variation browser also emits a signal to the DGWidget. This removes the corresponding ImageWidget and slide vector entries in the variation browser and inserts new entries in corresponding main browser vectors. In addition, the progress display of both browsers is updated.

**Deletion (Variation Browser):** The delete center function deletes the corresponding ImageWidget and associated slide from their vectors. Additionally, a signal is emitted to the DGWidget in order to update the "progress display". The delete all function deletes all ImageWidgets and slides.

#### 4.6.2 Selected Thumbnails Area

The selected thumbnails area basically consists of a QScrollArea in which objects of the class PreviewWidget are arranged horizontally. The PreviewWidget class is also derived from a QWidget. It is used to represent selected ImageWidgets and provide fast access to them. Whenever an ImageWidget is selected a corresponding PreviewWidget is created and extends the content depicted in the QScrollArea. In order to associate ImageWidget - PreviewWidget pairs, an according QMap is created ( $Qmap < ImageWidget*, PreviewWidget* >$ ). Using it allows to easily find PreviewWidgets and update or delete them, on ImageWidget adaption, deselection or deletion. The most important components of a PreviewWidget object are a pointer to the associated ImageWidget as well as a reimplementations of the paintEvent and mousePressEvent function. The paintEvent function is called to update a PreviewWidget's appearance whenever the corresponding ImageWidget has been adapted. It simply acquires the QImage from the associated ImageWidget, scales it to an appropriate size and draws it on itself using a QPainter. The mousePressEvent reimplementations reacts to two actions: a LMB and RMB click. The first simply emits a signal which induces the MainBrowser object to move the ImageWidget slide, corresponding to the current PreviewWidget, to the center position. The second emits a signal inducing the DGWidget to delete the PreviewWidget and deselect the according ImageWidget. Note, that the QScrollArea may only hold one widget. Therefore, an intermediate widget holding all PreviewWidgets that can be seen in the scroll area is required. It must be resized whenever a PreviewWidget is created or destroyed. The QScrollArea parameters are adjusted to automatically add a horizontal scrollbar when the intermediate image exceeds its size.

## 4.7 Renderer Integration

The presented design gallery uses the high performance ray caster from Kainz et al. [29]. It allows to render complex scenes with multiple intersecting volumes (each with its own TF) at interactive frame rates. In the context of the discussed system these characteristics are especially important to be able to work fluently with high dimensional separable TFs (see Section 3.6, 4.11). A critical factor for the high performance of the renderer is its implementation using NVIDIA's CUDA. This allows to perform rendering operations on the GPU in a very effective and massively parallel way.

The core of the rendering system API is formed by a number of special Coin3D node implementations, which can be consolidated in a scene graph. Therefore, the basic integration of the renderer into

a Qt application can be performed via a `QuarterWidget`, by passing the scene graph root to it. In the design gallery this `QuarterWidget` is the renderer widget positioned in the combination area, introduced in Section 3.5.2. It shows the data set visualization for the currently set TF and allows to interactively explore it in terms of panning, rotation and zooming.

The scene graph set in the `QuarterWidget` basically consists of a `SoSeparator` root node, holding one `SoPerspectiveCamera`, at least one `CudaRenderer` node and one `CudaComposer` node. Thereby, the last two node types are an essential part of the rendering system.

Each `CudaRenderer` node is associated with one volume and a corresponding TF to visualize it. As long as we work with one-dimensional TFs only one node is used. However, the use of separable TFs requires multiple volumes and corresponding TFs to be loaded. In order to induce a renderer node to load a volume the according name string must simply be passed to its according `SoSFString` field [47]. For the assignment of the TF, however an additional API component of the render system is required to convert `QGradients`, which represent TF in the design gallery, into a format appropriate for the rendering system. The `QTTF` class. There exists one instance of this class for each renderer node. Whenever a new TF, represented by a `QGradient`, is set in a `QTTF` instance it converts it into a format appropriate for the rendering system and passes it to the render node associated to it. In the presented implementation TFs are not directly set in the `QTTF` instance. Instead they are passed to a TF editor instance (from `QtGradientDialog` class) which forwards it to a `QTTF` instance. The TF editor is a user interface to modify existing or design new `QGradients`. Its purpose is to allow to further fine tune a TF combined from the selected thumbnails in the design gallery. For the current implementation an editor is connected to a `QTTF` instance via a signal and slot relation. Whenever the `QGradient` of an editor is adapted, its new version is immediately set in the corresponding `QTTF`. This in turn forwards it to the renderer node, which initiates the creation of a new data set visualization which is also shown in the `QuarterWidget`.

The final rendering of the ray caster is stored in the `CudaComposer` data structure. In order to copy the image data to the design gallery for further usage, this provides a special function. It retrieves a pointer to the image data stored in the composer and uses this data to create a `QImage`. The corresponding code can be seen in the following:

```
Cuda::HostMemoryHeap<uchar4, 2>* renderedDataDestination =
    AsyncCudaComposer::getGlobalColorBufferDedicated();

QImage* currentImage =
    new QImage( (uchar*) renderedDataDestination->getBuffer(),
               renderedDataDestination->size[0],
               renderedDataDestination->size[1],
               renderedDataDestination->getPitch(),
               QImage::Format_ARGB32);
```

The first line retrieves the pointer to the rendering stored in the composer node. The following lines set up the `QImage` using the retrieved data. Therefor, parameters passed to the `QImage` constructor are the actual image data, its dimensions, the bytes and an according image format, respectively.

## 4.8 Visualization of the Combined Transfer Function

In order to create a visualization of the current combined TF a new `QWidget` derived class has been implemented. Its `paintEvent` has been reimplemented to use a `QGradient`, representing the TF, to determine the rendered widgets appearance. The essential parts of the according code can be found in the following:

```

QPainter painter(this);
QRect rect(0, 0,width() - 1, height() - 1);
...
painter.setBrush(QBrush(tfGradient_));
...
painter.drawRect(rect);

```

Thereby, line two defines a rectangle of the widgets size to be drawn. The next sets a gradient as the painters brush. This is used to paint the interior of the rectangle to be drawn. The last line draws the rectangle filled by the gradient defined brush.

The instance of this class always obtains the most recent QGradient whenever the combined ImageWidget is updated and immediately uses it to repaints itself. Therefore, it constantly provides the current visualization of the combined TF in the TF space.

## 4.9 Storage Area

Saving a TF to the storage area is very simple. The combined ImageWidget representing it is simply copied and stored in an extra vector. Thereby, the copied ImageWidget and the contained image are scaled to an appropriate size.

In order to display these ImageWidgets in the user interface they are added to a scroll area in the same way as the PreviewWidgets in the selected thumbnail area. Removing them also works in the same way. The deletion however is initiated by pressing a corresponding button and is performed for all selected ImageWidgets in this area. In order to allow to select and deselect the ImageWidget the mousePressEvent function has been reimplemented. This causes the already explained blue frame to be added and removed.

## 4.10 Saving and Loading Data Using XML

As mentioned in Section 3.5, the design gallery provides functionality to save and load a current working session, AH and PRH calculation settings, TFs as well as intermediate precalculation results (after PRH calculation). All these tasks have in common that involved information is saved in and restored from files of XML format, using Qts QDomDocument and various QDomNode subclasses [65]. The first basically represents an XML document, by forming its root node and providing access to the document data. The latter is the base class of all Dom tree nodes. In order to actually write and read such documents to and from hard disk the QFile and QFileDialog classes [65] are used.

Listing 4.7 demonstrates the XML file structure of a session, saved using the CF based user interface. A set of modular methods, depending on each other have been implemented to store and load certain system components. This includes methods to save and load ImageWidgets, their RegionData vectos, single RegionData objects and more.

The XML file structures for the remaining tasks is pretty simply. They are briefly shown and described in the Listings 4.8, 4.9 and 4.10.

## 4.11 Using Separable Transfer Functions

**General Adaption of the Basic Approach:** In order to work with separable TFs the design gallery implementation is adapted by several facets.

All design gallery methods related to TF adaption and combination are performed on every dimension of the separable TF. This includes methods for color changes, variation creations as well as for

```

1 <!DOCTYPE SavedSession>
2 <session id="1337">
3   <dataSets>
4     <dataSet name="C:\Diplomarbeit\...\dice.hdr"/>
5     ... More dataSets for separable transfer functions
6   </dataSets>
7   <explorationArea>
8     <mainBrowser centerIndex="7">
9       <thumbnails>
10        <imageWidget isSelected="0" id="0">
11          <regionDataStorage>
12            <regionDataVector>
13              <region>
14                <mean g="29" position="0.123..." r="255" a="62" b="0"/>
15                <deviation position1="0" g="29" position2="0.260..." r="255" a="0" b="0"/>
16              </region>
17            </regionDataVector>
18            ... More regionDataVectors for separable transfer functions
19          </regionDataStorage>
20        </imageWidget>
21        ...
22      </thumbnails>
23    </mainBrowser>
24    <variationBrowser centerIndex="5">
25      <thumbnails>
26        <imageWidget isSelected="0" id="10000">
27          ...
28        </imageWidget>
29        ...
30      </thumbnails>
31    </variationBrowser>
32  </explorationArea>
33  <savedCombinations numSavedCombinations="2" isEmpty="0">
34    <imageWidget isSelected="0" id="7331">
35      <regionDataStorage>
36        <regionDataVector>
37          <region>
38            <mean g="243" position="0.524..." r="255" a="255" b="0"/>
39            <deviation position1="0.380..." g="243" position2="1" r="255" a="0" b="0"/>
40          </region>
41          <region>
42            <mean g="117" position="0.030..." r="0" a="51" b="255"/>
43            <deviation position1="0" g="117" position2="0.093..." r="0" a="0" b="255"/>
44          </region>
45          <region>
46            <mean g="113" position="0.279..." r="255" a="109" b="0"/>
47            <deviation position1="0.0645..." g="113" position2="0.495..." r="255" a="0"
48              b="0"/>
49          </region>
50          ...
51        </regionDataVector>
52        ... More regionDataVectors for separable transfer functions
53      </regionDataStorage>
54    </imageWidget>
55    ...
56  </savedCombinations>
</session>

```

**Listing 4.7:** XML file structure for a saved session using the CF based user interface. The used data set names are stored to load the according volumes in the renderer when restoring a session. Both, the main and variation browser contain a set of ImageWidgets and hold the index of the one centered in the CF widget. The ImageWidgets includes an "is selected" attribute to memorize those currently affecting the combined visualization. Moreover, it holds one or more RegionData vectors, depending on the amount of used TFs. In the main and variation browser an ImageWidgets RegionData vector only holds one RegionData object. This is defined by a mean and two deviation positions (in the TF space) and their  $RGB\alpha$  color. The combined ImageWidget and TF visualization do not need to be saved. They can be recreated from the selected ImageWidgets. The saved combination area also consists of a set of ImageWidgets. However, here they usually contain RegionData vectors holding more than one RegionData instance.

```

1 <!DOCTYPE properties>
2 <properties>
3   <ahProperties>
4     <property value="3" name="alphaValue"/>
5     <property value="8" name="ahBlocksize"/>
6     <property value="0" name="ahOffset"/>
7     <property value="1" name="ahCalculateAH"/>
8     <property value="0" name="ahSmoothSelective"/>
9     <property value="0" name="ahSmoothGlobal"/>
10    <property value="20" name="ahRemainingPeaks"/>
11  </ahProperties>
12  <prhProperties>
13    <property value="0.9" name="epsilonValue"/>
14    <property value="0.1" name="epsilonSubtractor"/>
15    <property value="8" name="prhBlocksize"/>
16    <property value="2" name="multiplierStart"/>
17    <property value="1" name="multiplierEnd"/>
18    <property value="0.1" name="multiplierAdaptor"/>
19    <property value="2" name="smallestSeries"/>
20    <property value="8" name="largestSeries"/>
21    <property value="3" name="numIterations"/>
22  </prhProperties>
23 </properties>

```

**Listing 4.8:** XML file structure for saved settings. A settings XML file basically consists of a set of property tags. Each of them holds a name and corresponding value attribute. Thereby, AH and PRH property tags are each summarized by different tags above them.

```

1 <!DOCTYPE Gradient>
2 <gradientData spread="PadSpread" startX="0" startY="10"
3   coordinateMode="LogicalMode" type="LinearGradient" endX="100"
4   endY="10">
5   <stopData position="0">
6     <colorData g="117" r="0" a="0" b="255"/>
7   </stopData>
8   ...
9 </gradientData>

```

**Listing 4.9:** XML file structure for a saved TF. A TF is summarized by a XML structure storing a QGradient in terms of a number of QGradientStop tags. These are represented by their position in the gradient as well as an  $RGB\alpha$  color value.

```

1 <!DOCTYPE PRHData>
2 <prhData numSeries="9" numHistBins="2291">
3   <series size="2" number="0">
4     <PRH mean="232.5135876754815" deviation="244.0361091822847"/>
5     <PRH mean="1110.238011702584" deviation="309.9155048130218"/>
6   </series>
7   <series size="3" number="1">
8     <PRH mean="239.6353741899056" deviation="251.5646716667923"/>
9     <PRH mean="1120.201769135872" deviation="301.385045138491"/>
10    <PRH mean="1217.988607346297" deviation="86.36433691866078"/>
11  </series>
12
13  ...
14
15 </prhData>

```

**Listing 4.10:** XML file structure for saved intermediate precalculation results. The intermediate precalculation results, stored after the PRH computation basically consists of a number of PRH series tags contained by a prhData tag. This also contains the total TF space extend and the number of series as an attribute. Each series tag holds its size as an attribute and a number of PRH tags. These are defined by a mean and deviation value in the TF space.

the creation of a combined TF. A separable TF and the corresponding image are represented by an ImageWidget further on. However, this holds one RegionData vector per dimension of the separable TF now ( $vector < vector < RegionData* >>$ ). The rendering system requires n CudaRenderer nodes, QTTransferFunction and QtGradientDialog objects (see Section 4.7). One for each dimension of the separable TFs. Moreover, the ray caster must load one scalar volume for each TF dimension. Each volume is associated with one CudaRenderer node. An image is obtained by passing all dimensions of the corresponding TF to the renderer. Therefore, one QGradient is created from each RegionData vector and handed to the corresponding CudaRenderer node via a TF editor (see Figure 3.21). Subsequently, the ray caster evaluates the separable TF as described in Section 3.6 and shown in Figure 3.21.

**Evaluation of Separable Transfer Functions for the Basic Approach and the Preliminary nD Prototype:** In order to explain the implementation of the concatenation process we will consider the *getVolumeSample* method in the following. It is called once for every sample (of the rays traversing the volume) of each volume and retrieves the  $RGB\alpha$  value of this sample for the further render process. Two essential parameters of this method are the position of the current sample point and the ID of the volume it belongs to.

For the default *OR* concatenation, this information is utilized to look up the gray value of the volume at the sample position. Using the gray value the corresponding  $RGB\alpha$  of the associated TF is retrieved. This is subsequently shaded and the resulting  $RGB\alpha$  value is returned for the further rendering process. This means the *OR* concatenation treats all samples of each volume independently.

In order to perform the *AND* concatenation of the basic approach the *getVolumeSample* method is adjusted by some simple facets. Whenever the *getVolumeSample* method of the first volume (with ID 0) is called we retrieve the  $RGB\alpha$  value of all volumes samples at this position. The  $RGB\alpha$  values of the first volume is compared to that of all others. If the color test is passed, meaning the colors of all samples correspond to that of the first and the opacity of all samples is larger zero, the  $RGB\alpha$  value of the current volume is shaded and returned for the further rendering process. Whenever the *getVolumeSample* method is called for an other volume but the first (with an ID larger 0) a  $RGB\alpha$  value of (0, 0, 0, 0) is returned.

In order to allow a flexible concatenation, a set of further parameters are passed to the



*getVolumeSample* method. Therefore, the *CudaRenderer* node has been extended by a number of fields which can be set at runtime and immediately influence the depicted rendering. Since the used parameters are integers or lists of them we use the *SoSFInt32* [47] and *SoMFInt3211* [47] fields to hand them to the *CudaRenderer* node. The parameters of a certain *CudaRenderer* node are only passed to calls of *getVolumeSample* of the according volume. The most important parameter is the concatenation mode. This may induce the following concatenations: *NONE*, *OR*, *AND*, *AND<sub>with</sub>*. All but the first may additionally be combined with a *NOT(OR)* or *NOT(AND)*. *NONE* simply returns an "empty" sample of (0,0,0,0). *OR* and the standard *AND* work as described above. The *AND<sub>with</sub>* additionally requires a list of indices of the volumes to concatenate with. This may also be set via a field of the *CudaRenderer* node. For this mode the current volumes samples  $RGB\alpha$  is only compared to those of the other volumes in the list. The *NOT(OR)* uses an additional volume index. If the sample of this volume is not "empty" the currently evaluated *AND* or *OR* is not depicted irrespective of their evaluation result. The *NOT(AND)* requires an additional list of volume indices. If the *AND* concatenation of the samples of these volumes is not "empty" the currently evaluated *AND* or *OR* is not depicted irrespective of their evaluation result. Note, that updated field values of the *CudaRenderer* node are recognized by *SoSensors* [47], which immediately invoke callback functions to apply the new values.



# Chapter 5

## Results

This section evaluates the presented design gallery. Therefore, the DM and CF based user interfaces are compared to a manual TF editor in Section 5.1. Moreover, a series of visualizations generated based on one- and multi-dimensional TFs are presented in Section 5.2.

For both, the user interface evaluation and the generation of data set visualizations a system consisting of an AMD Phenom(tm) II X4 955 Processor 3.20 GHz, 4 GB main memory and a Nvidia GeForce GTX 275 with 896 MB graphics memory have been used. The utilized operating system is Windows 7 Professional, 64 bit.

### 5.1 User Interface Evaluation

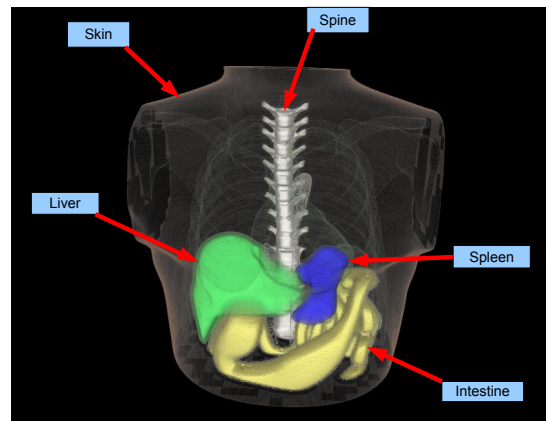
This section investigates the practical applicability of the described design gallery user interfaces (DM and CF based) and compares them to a conventional manual editor. Thereby, Section 5.1.1 presents the concept, material and methods for the evaluation, while Section 5.1.2 presents and interprets the obtained results.

#### 5.1.1 Evaluation Concept, Material and Method

**Test Arrangement Overview:** After a brief introduction on all three approaches using a test data set, nine probands had to recreate a given visualization of a synthetic data set with clearly separated selected structures of certain color and opacity. This task had to be performed once as accurate and once as fast as possible with each method. Subsequently, a semi-structured interview was performed with all probands in order to elicit their subjective impressions on the used methods. Based on the quality of the recreated visualizations, the required time and the collected interview answers, the three mentioned approaches have been evaluated and compared.

**Data Acquisition:** The following describes the detailed procedure which had been performed with each proband in order to gain the required data:

- Explain task: I kindly ask you to recreate a visualization of a medical data set with three different approaches which I will explain to you in the following.
- Explain user interfaces based on a test data set:
  - *DM & CF* interaction is explained according to the descriptions from Section 3.5.1.2 and 3.5.1.3. Thereby, it is especially important that users understand the specific concept of browsing, combining and adapting color and opacity of initial structures with the different user interfaces.



**Figure 5.1:** Reference image for the user interface evaluation. All structures to be visualized are clearly indicated: skin, spine, spleen, intestine and liver.

- The *Manual editor* allows to define TFs with an arbitrary amount of *points* which can be set at various locations in the TF space. At these positions a color and opacity value can be defined. The color and opacity between these points is interpolated. In order to allow users to work with this approach, the basic concept of how transfer functions work must be explained to them. Moreover, it is necessary to explain a strategy to find structures with this approach: We suggested to search one structure after the other by setting three points with the same color. The outer points are assigned an opacity of zero, while the inner one needs a higher value. By moving these points and changing the center point opacity a structure can be visualized appropriately.
- Show the user the visualization to recreate (see Figure 5.1). The corresponding image is constantly displayed on a second screen throughout the test. For our test, we use the Phantom data set from [56].
- Let the user recreate the visualization with all three tools: Once as fast as possible and once as accurate as possible. Store the TF of all six results and stop the required design time for all six cases.
- Interview user for the subjective evaluation:
  - How do you rate your design speed with the CF/DM/manual approach (1 best to 8 worst)?
  - How do you rate the intuitivity of the CF/DM/manual approach (1 best to 8 worst)?
  - How do you rate the clearness of the CF/DM/manual approach (1 best to 8 worst)?
  - Is there something especially negative about working with the CF/DM/manual approach?
  - Is there something especially positive about working with the CF/DM/manual approach?
  - Which approach did you like best and why?
  - Which approach did you like the least and why?

**Evaluation and Statistical Methods:** From the data acquired throughout the procedure described above, a number of measures for the objective and subjective evaluation of the three used design approaches and their comparison are obtained. Objective measures are the quality of a created visualization and the required time for this task. In order to obtain quality values, we create a visualization with all saved TFs of each user and compare them to the reference image [56]. This is done by computing the accumulated *RGB* difference of all corresponding pixels of both images and dividing it by the amount

of pixels. Subsequently, the result is transformed into a percentage. Thereby, all visualizations must be considered from the same view. The variation between reference and user visualization describes the quality of the latter. The subjective criteria are derived from the quantifiable interview questions. They include the subjective design speed, intuitivity and clearness.

All quantifiable measures are evaluated per approach and task (as accurate and as fast as possible) using the same statistical methods and descriptions. These are:

- Arithmetic mean ( $\mu$ )
- Standard deviation ( $\sigma$ )
- Median ( $M$ )
- Quartiles ( $Q_1, Q_3$ )
- Interquartile range ( $IQR = Q_3 - Q_1$ )
- Minimum and maximum (MIN, MAX): These are the lowest and highest measured value for a certain criteria such as the design time.
- Upper and lower outliers ( $O_U, O_L$ ): These are very untypical values for a certain criteria which are significantly outside the  $IQR$ . Measured values are initially classified as outliers if they satisfy one of the following equations:
  - $O_U > Q_3 + IQR * 1.5$
  - $O_L < Q_1 - IQR * 1.5$

Note, that there is a certain tolerance range above and below the first and third quartile, within which measured values are not considered to be outliers.

Mean and deviation are very sensitive to outliers. Therefore, median, quartiles, etc. have been deployed in order to increase the statistical expressiveness.

The answers on the not quantifiable interview questions are utilized in the following section in order to explain the evaluation outcome.

### 5.1.2 Results and Interpretation

Figures 5.2 and 5.3 provide an overview of the evaluated objective and subjective quantitative measures of all three design approaches. In the following these results are discussed in combination with the answers to the qualitative interview questions.

**Objective Measures:** Figure 5.2 summarizes the evaluation of all objective measures for all tools and tasks. One aspect that can be seen in the plots is, that the manual editor clearly entails the largest error quote and time requirements for both the most accurate and fastest possible design work. Another expected observation is, that fastest possible task execution leads to higher error quote at decreasing design times for most tools. The decreasing time requirement is additionally favored by the fact, that the same visualization had to be recreated for all tasks. At first the design had to be performed as accurate and then as quick as possible. Concerning the latter, this led to a certain customization facilitating a lower demand of time. Similarly, it can be explained, that the CF approach shows a higher error rate for its accurate than for its fast execution. Since the CF approach was used at first, users have not been familiar with the data set at this point.

Another interesting aspect we have observed, relates to the distribution of time effort during the task execution. Using the CF and DM approach, a significant amount of time has been required to identify

correct structure colors and opacities. For the manual editor on the other hand a lot of time has already been used to actually identify the structures themselves.

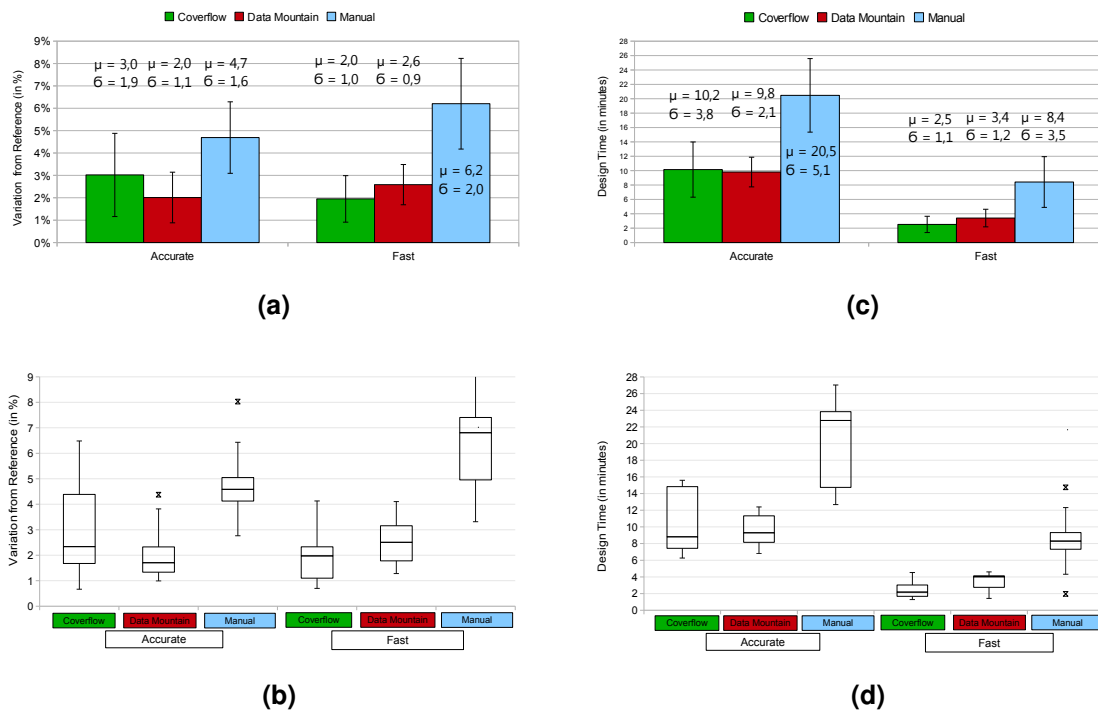
Moreover, we have observed, that the DM error rate for the fast execution has been higher than that of the CF approach. The reason for this could be the smaller thumbnail size and the related inferior recognizability of structures.

**Subjective Measures and Assessments:** Figure 5.3 summarizes the evaluation of all quantitative, subjective measures for all tools and tasks. Similar as for the objective evaluation, the manual editor also scored worst at all subjective ratings. The mentioned user reasons for these results turned out to be well known problems of conventional manual editors. These include the unintuitive work, the associated trial and error approach as well as the required technical explanations. In contrast to the CF and DM method, no predefined selection of initial structures exists (regions of interest). In order to discover a structure without any knowledge about their value ranges in the data set, a random approach must be taken. Control points defining a TF may only be set at arbitrary positions and must be shifted randomly. In order to finally obtain a complete and correct depiction, a series of further fine adjustments is required. This approach is unintuitive because there exists no straight forward correlation between adaptations in the TF space and the depicted data set visualization. Accordingly, a systematic design procedure is only feasible to a limited extend resulting in a high demand of time and a high error rate. Furthermore, technical explanations are inevitable for a target-orientated application of the editor. Users are primarily interested in their practical work and do not want to deal with technical backgrounds.

As expected, the ratings of the CF and DM user interface clearly exceed that of the manual editor. For the CF approach, the high intuitivity and efficiency of the navigation was especially praised. As a result of the easy comprehensibility and large amount of possibilities for navigation, users found their ways of interaction most convenient. Thereby, the fluent and quick browsing of the given structures was perceived very comfortable. The fast access via the selected thumbnails area was judged to be especially useful. It allows to quickly jump between selected structures as well as to deselect them. Hence, users especially appreciated, that they could simply choose the desired structures at first and quickly access them for detailed adaption. The good visibility of structures on the large thumbnails was also perceived very positively. This is, amongst others, very important for the selection of appropriate variations. In general, this system was qualified to be a very intuitive. Due to the usage of the popular CF design [7] from Apple, much of the provided functionality has already been known.

Critical supporters of the DM system missed a complete overview of all given structures. This may extend the required time to find desired structures at a large number of thumbnails. Apart from this, only some minor and easily recoverable concerns have been mentioned. For example, a dialog to prevent an undesired deletion of thumbnails was requested. Moreover, the sequence of thumbnails according to increasing opacity should be preserved after the usage of a variation via *exchange variation* (see section 3.5.1.3).

The main advantage of the DM approach is its great freedom to display comprised contents. In this context users especially emphasized the possibility to see all thumbnails at a glance and to individually arrange them. DM supporters stated that this approach accommodates their mode of operation: at first they are provided with an overview from which they can go into detail. Interestingly, the CF rating on *clearness* still slightly exceeds that of the DM. This assessment must be seen in the context of an important potential for improvement of the DM navigation. Relating to this, users repeatedly requested simplifications and more common and familiar interactions schemes. Examples are the creation and selection of thumbnail variations with "CTRL+Wheel up/down" as well as a facilitated zoom mode. As a result of the unaccustomed navigation, users could not fully concentrate on the utilization of the system benefits such as its clearness. Thus, not only the *intuitivity* rating but also that of the *clearness* has been affected to a certain degree. In this context it should be noted, that mainly experienced computer users criticized the navigation. In contrast, less experienced probands quickly got used to these

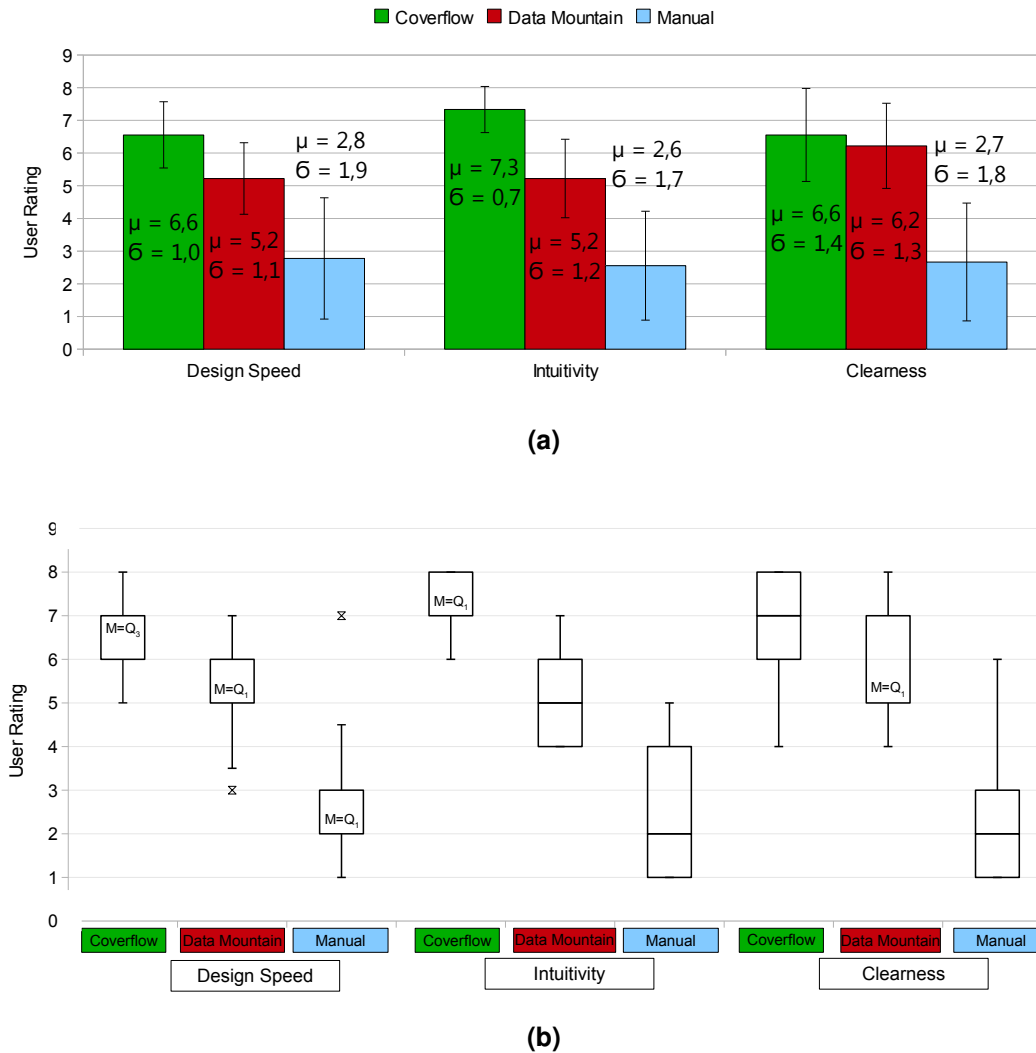


**Figure 5.2:** Graphical presentation of the objective user interface evaluation results. (a) Depiction of the error rate (variation from the reference image in %) per tool at both, maximum accuracy and quickness on the basis of mean and deviation. (b) Depiction of the error rate (variation from the reference image in %) per tool at both, maximum accuracy and quickness on the basis of median, first and third quartile, IQR and outliers. (c) Depiction of the design time per tool at both, maximum accuracy and quickness on the basis of mean and deviation. (d) Depiction of design time per tool at both, maximum accuracy and quickness on the basis of median, first and third quartile, IQR and outliers. Note: In (a) and (c) the bar depicts the arithmetic mean, while the whiskers represent the standard deviation. In (b) and (d) the box represents the *IQR* from  $Q_1$  to  $Q_3$ . The intermediate line in the box indicates  $M$ . The whiskers represent the tolerance range within which measured values outside the *IQR* are not considered to be outliers. The upper whisker reaches from  $Q_3$  to  $\min(Q_3 + IQR * 1.5, MAX)$ . The lower whisker reaches from  $Q_1$  to  $\max(Q_1 - IQR * 1.5, MIN)$ . The  $x$  signs below and above them indicate outliers.

interaction schemes.

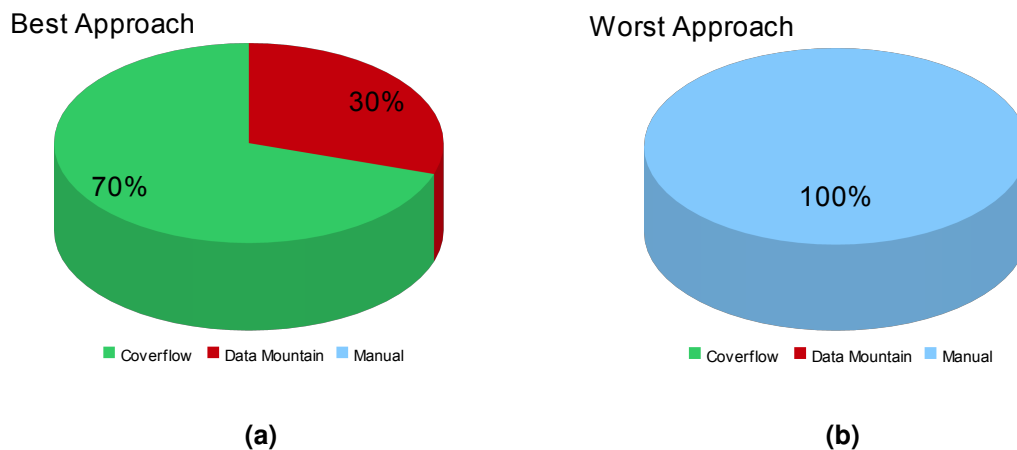
A further point of criticism is the small size of thumbnails requiring a high zooming effort. Furthermore, users demanded that the last selected thumbnail should be especially highlighted in order to allow the unambiguous utilization of buttons.

**Summary:** Both the DM and CF approach are clearly preferred to the manual editor. This can distinctly be seen in Figure 5.4 (b). The choice for one of the remaining user interfaces depends on the importance of certain features to individual users (see Figure 5.4 (a)). Thereby, the CF based approach is preferred on average, since it provides a more intuitive overall package with a navigation scheme which is already familiar to many users. The DM navigation on the other hand requires several improvements.



**Figure 5.3:** Graphical presentation of the quantifiable, subjective user interface evaluation results. (a) Depiction of the design speed, intuitivity and clearness of each tool on the basis of mean and deviation. (b) Depiction of the design speed, intuitivity and clearness of each tool on the basis of median, first and third quartile, IQR and outliers. Note: In (a) the bar depicts the arithmetic mean, while the whiskers represent the standard deviation. In (b) the box represents the *IQR* from  $Q_1$  to  $Q_3$ . The intermediate line in the box indicates  $M$ . Sometimes the median is equal to one of the Quartiles. This is explicitly noted in the plot. The whiskers represent the tolerance range within which measured values outside the *IQR* are not considered to be outliers. The upper whisker reaches from  $Q_3$  to  $\min(Q_3 + IQR * 1.5, MAX)$ . The lower whisker reaches from  $Q_1$  to  $\max(Q_1 - IQR * 1.5, MIN)$ . The  $x$  signs below and above them indicate outliers.





**Figure 5.4:** Graphical presentation of the distribution showing how often the single approaches where rated to be the best and worst method. (a) Depiction of the best approach ratings in percent. (b) Depiction of the worst approach ratings in percent.

## 5.2 Presentation of Selected Visualization Examples

This sections presents a series of visualizations generated with the developed design gallery in order to highlight its capabilities and limitations.

The one dimensional examples in Section 5.2.1 are used to demonstrate visualizations of structures that are detected with the precalculation process described in Section 3.2. We show the impact of various parameters and the type of data set on both the detected regions of interest and the time required for their precalculation.

Multi-dimensional examples show the visualizations which can be obtained with the developed scheme for the evaluation of separable TFs.

### 5.2.1 Visualizations Based on One-Dimensional Transfer Functions

In the following a series of data set visualizations based on one-dimensional TFs are presented. The names of used data sets are:

- dice
- case2\_CT
- incisix
- case2\_T1\_post
- case2\_FLAIR
- case2\_fMRI.tMAP

In order to provide a broad range of representative results the data set selection consists of a syntectic data set (dice), two CT data sets (case2\_CT, incisix) and three MRI data sets captured with different parameters (case2\_T1\_post, case2\_FLAIR, case2\_fMRI.tMAP).

All *case2* data sets are courtesy of Prof. B. Terwey, Klinikum Mitte, Bremen, Germany and have been downloaded from [27]. They are scans of the same human head, performed with different imaging modalities. The *incisix* data set has been downloaded from [68]. Note, that the data sets were scaled to appropriate dimensions (and intensity ranges) since the used renderer requires the full data set to be stored in the graphic card memory, which is limited in size.

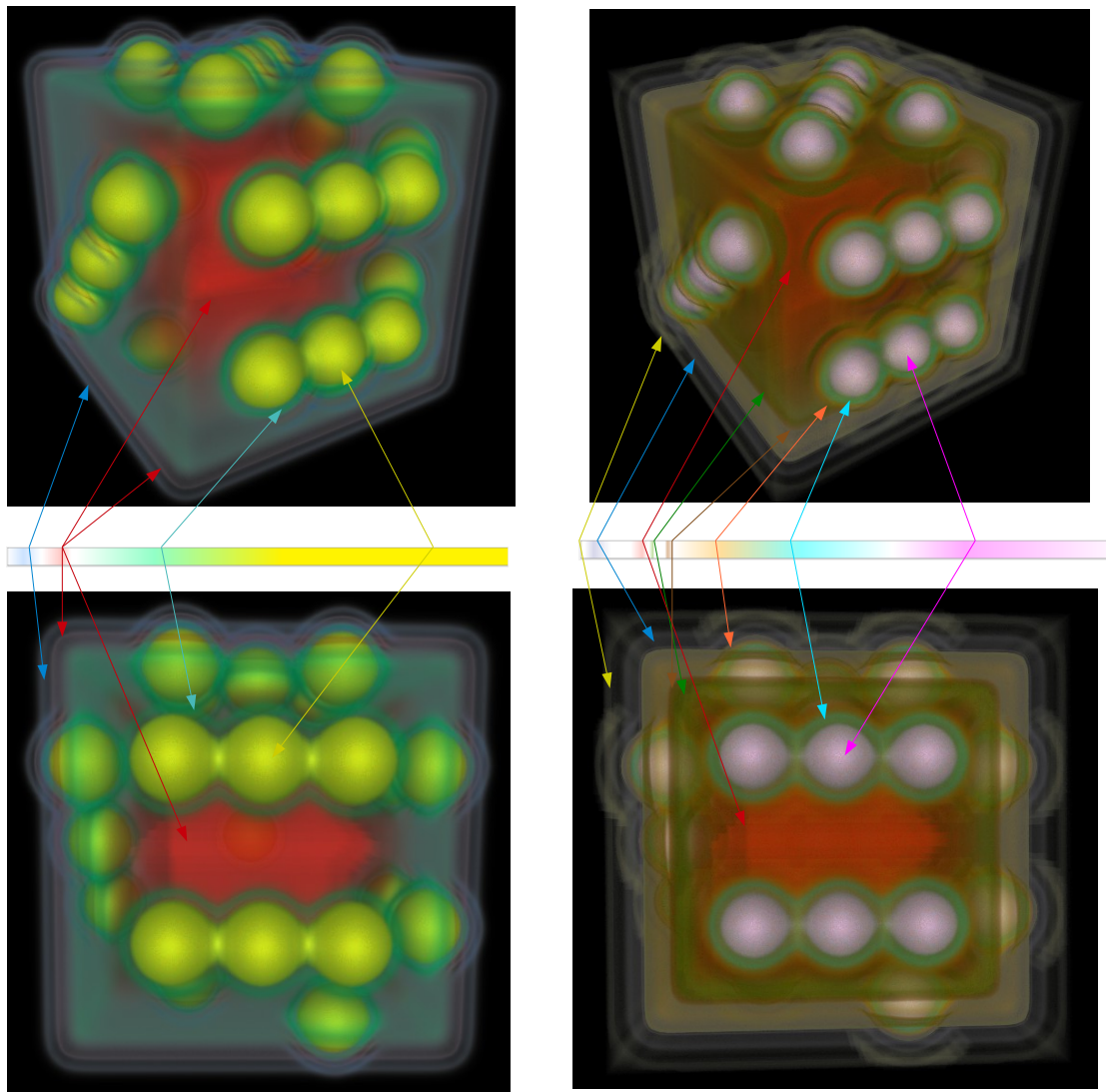
**Syntectic Example - Dice:** The first considered data set is the syntectic dice consisting of multiple layers around its pips. It has dimensions of  $64^3$  and an intensity range from -1023 to 800. The purpose of this example is to demonstrate the impact of different settings at optimal conditions on both the visualization result and the required precalculation time. Therefore, the dice data set has been precalculated twice, once with default settings and once with very "detailed" settings (see Table 5.1).

Using default settings (see Table 5.1, row 1) the system divides the data set into four elementary structures shown in Figure 5.6 (a) - (d) and the left half of Figure 5.5. These are its pips (yellow), a layer surrounding it (green) as well as the dice's outer boarder (blue) and its interior (red). Using more "detailed" settings (see Table 5.1, row 2) a finer separation of the data set into eight structures of well defined extend can be achieved (see Figure 5.6 (e)-(l) and right half of Figure 5.5). While the default settings result in coalesced pips (see Figure 5.6(d)) with only one massive structure around them (see Figure 5.6 (c)), the "detailed" settings identify more clearly separated pips (see Figure 5.6(l)) which are surrounded by four finer layers (see Figure 5.6 (h)-(k)). Moreover, the interior of the dice identified using "detailed" settings only exhibits minor overlaps with exterior layers of the dice in comparison to that found with default settings (compare Figure 5.6 (g) and (b)). Furthermore, an additional outer layer has been identified with "detailed" settings (see Figure 5.6(e)). The main reasons for the finer separation are the increased epsilon, the corresponding smaller adaptor as well as the smaller block size and the higher alpha value. Since this data set is synthetic and does not contain noise the impact of the high epsilon value and corresponding adaptor can be seen very well. The boarder of the dice consists of multiple highly homogenous regions which are identified by these settings.

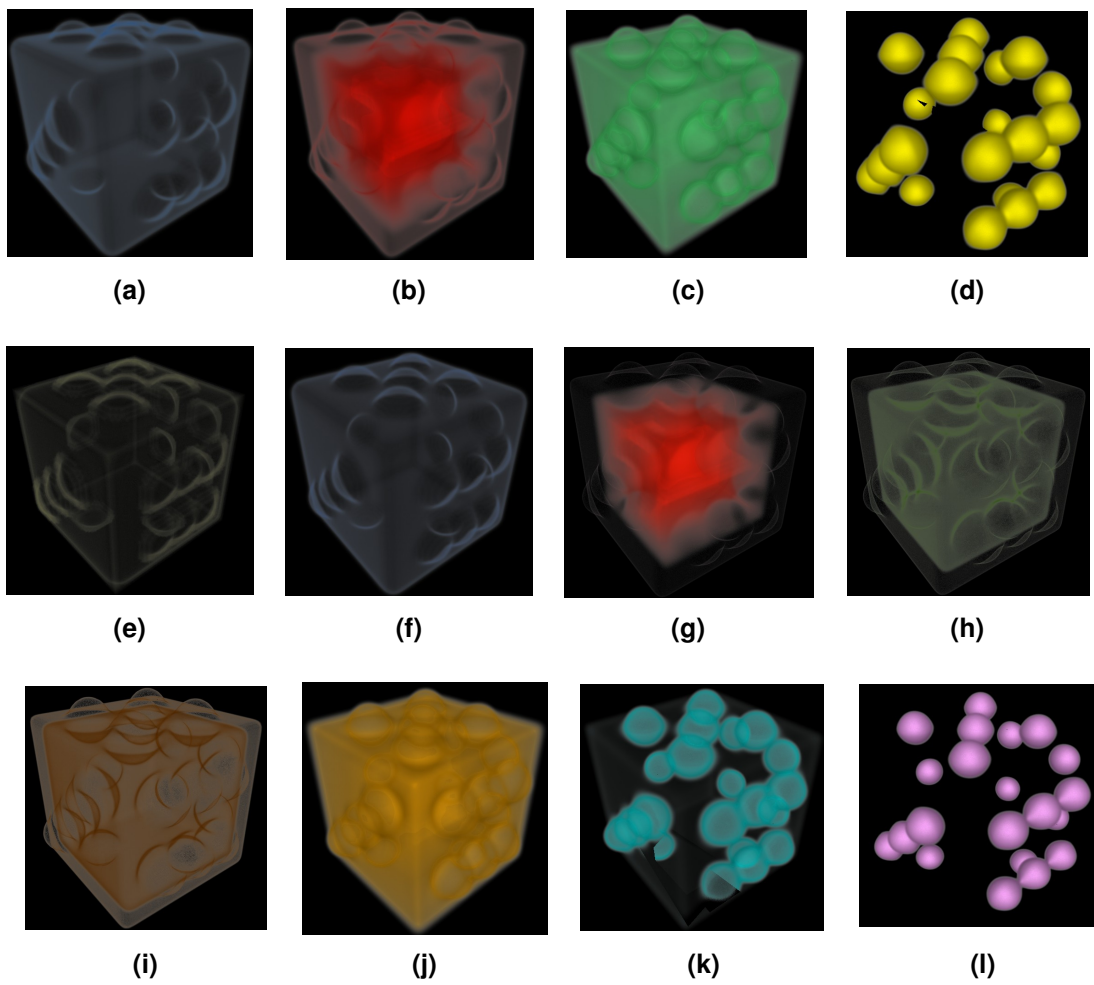
The description above points out that the "detailed" settings lead to a finer distinction of data set structures. However, they also entail increased precalculation times. While the precalculation with default settings only took 5 seconds, the adjusted settings needed 16 seconds. Reasons for the increasing precalculation time are the smaller block size as well as the extremely high epsilon value and the according low epsilon adaptor. The epsilon settings especially lead to a frequent repeating of the weight range test and a large number of initial PRHs. The precalculation time is further extended by the larger amount of identified structures resulting in more thumbnails to be rendered. While only 9 images had to be rendered for the default settings, 80 images were required for the "detailed" settings.

Figures 5.8 and 5.7 show the single dice structures obtained with the "detailed" settings, without using the AH calculation, as well as a combination of them. Thereby, it can be observed that fewer and less distinct structures are identified. Also, the pips which spread over a larger part of the TF space could not be identified separately.

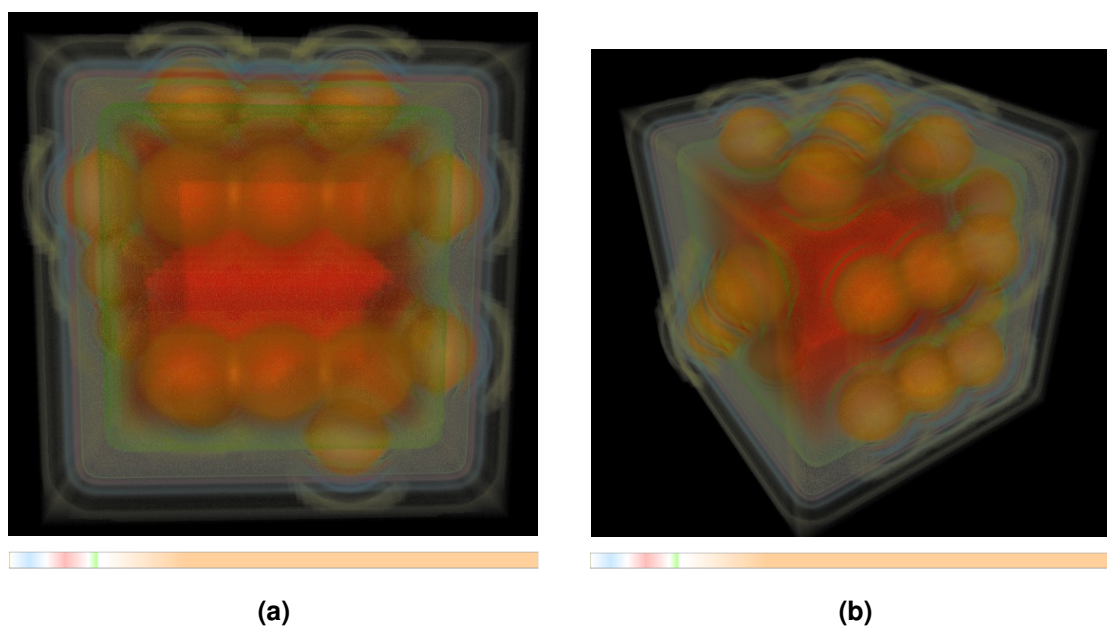
**Computed Tomography Examples - Case2\_CT and Incisix:** This section demonstrates visualizations of two data sets resulting from CT scans of a human head. At first we will consider the *case2\_CT* data set. It has been scaled to dimensions of  $166 \times 214 \times 230$  and an intensity range from -662 to 1106. The data set comprises three main regions: the cranial bone, soft tissue and the teeth's enamel (see Figure 5.9 (a) - (c)). Since CT scans are not good at differentiating soft tissues the skin as its outliet layer can be seen best. Interestingly only the enamel of some of the molars could be identified as separate structure while the remaining teeth are entirely depicted as a part of the cranial bone. This is because the latter have a similar intensity range as the cranial bone. Assumably, because they are dental crowns. In addition, the system detected a region showing the contours of the bronchus and the paranasal sinus. This structure overlaps with the boarder between air and skin (see Figure 5.9 (e) and (f)). The interior of



**Figure 5.5:** Dice example: Comparison of visualization results obtained using different precalculation settings. The images and the TF shown in the left half of the figure are obtained using settings from Table 5.1, row 1 (default settings). The images and the TF shown in the right half of the figure are obtained using settings from Table 5.1, row 2 ("detailed" settings).



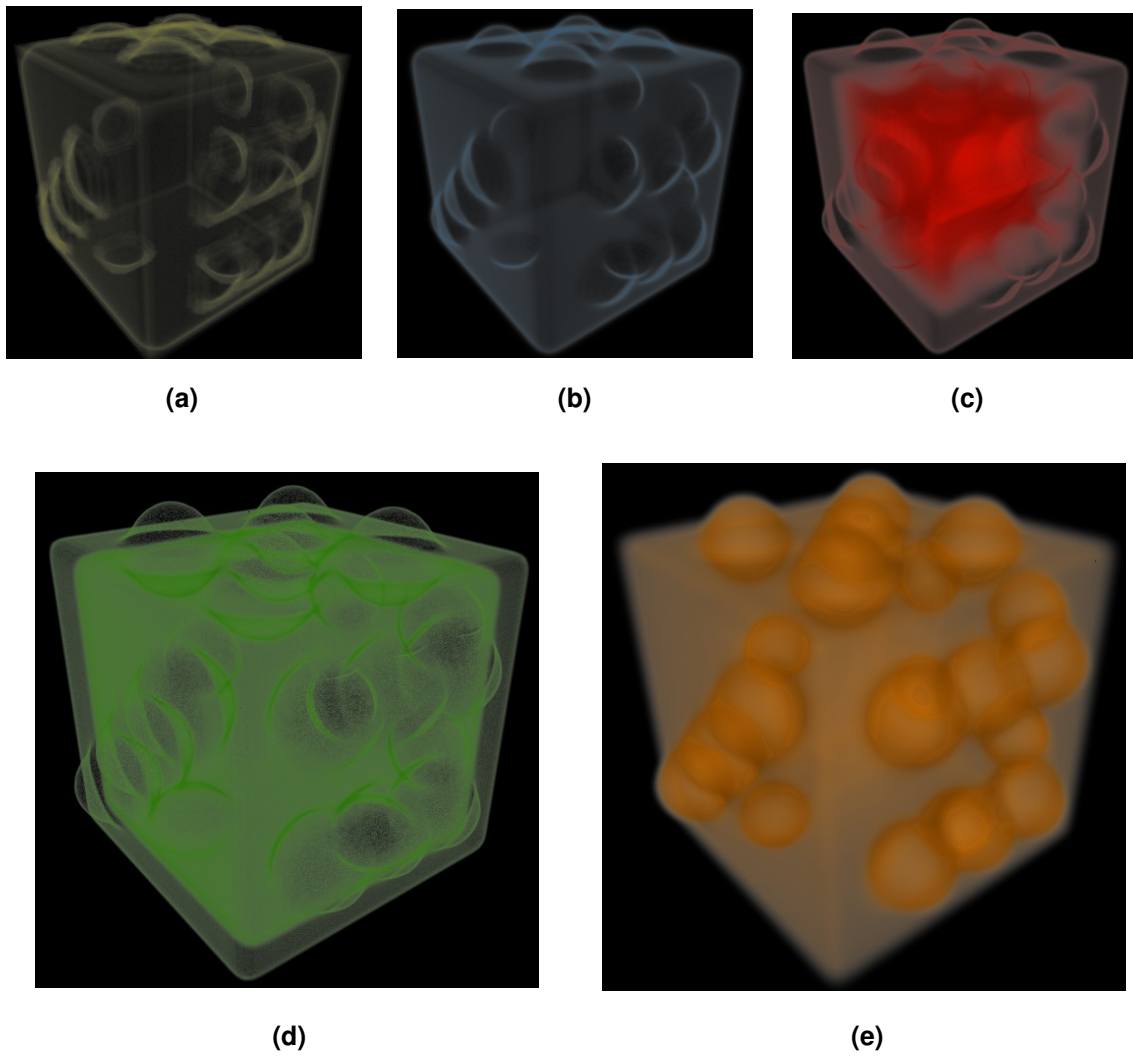
**Figure 5.6:** Dice example: Figures (a) - (d) show structures identified using default settings (see Table 5.1, row 1). Figures (e) - (l) show structures identified using "detailed" settings (see Table 5.1, row 2).



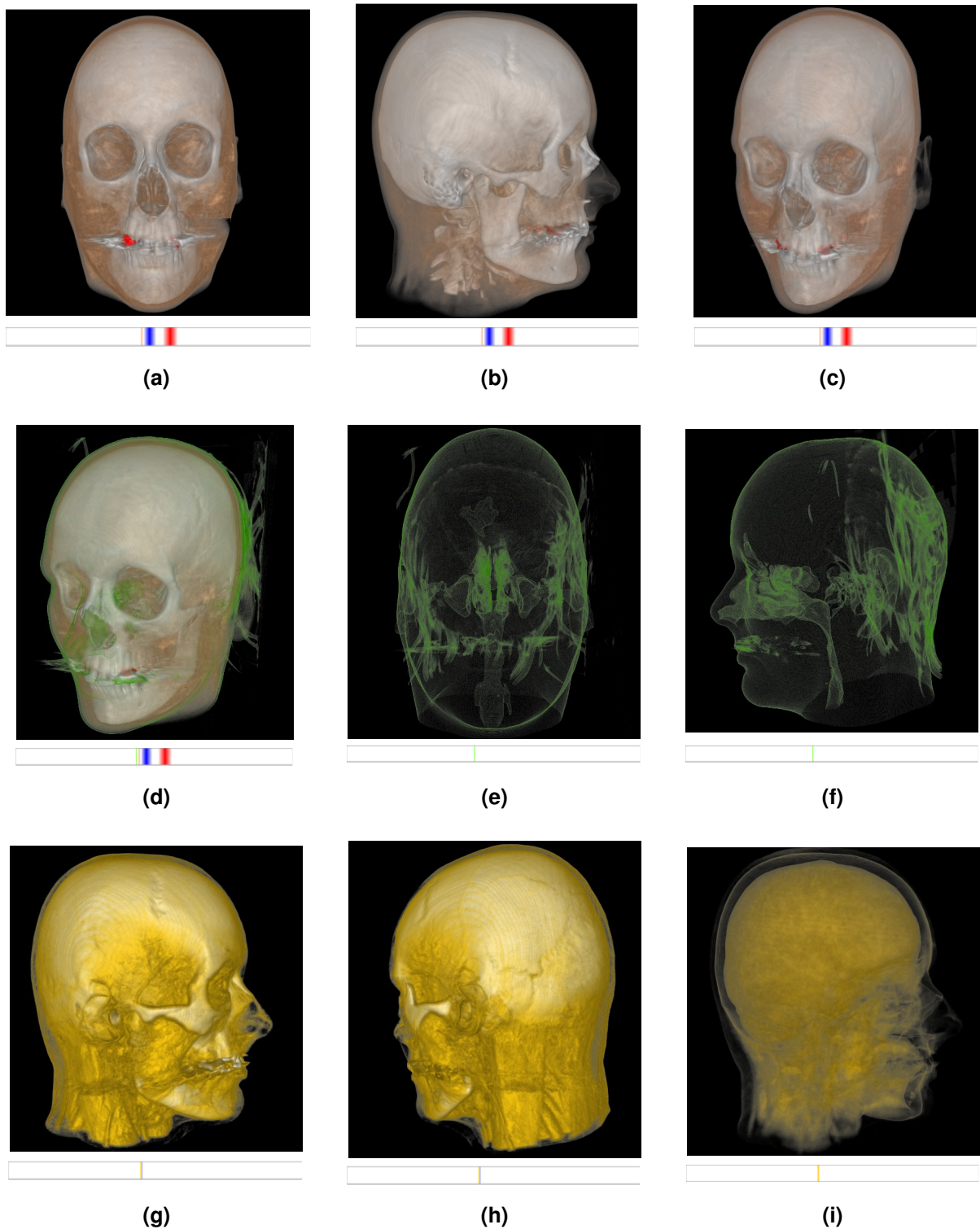
**Figure 5.7:** Dice example: Visualization resulting from a precalculation with "detailed" settings (see Table 5.1, row 2) but no AH calculation.

this structure overlaps with the air around the remaining data set and can therefore not be seen. Figure 5.9 (d) shows a combination all structures mentioned yet (except the air). They could be clearly separated although they spread over a very small part of the total value range and are located close to each other. Figure 5.9 (g) to (i) show another detected structure. In (g) and (h) we can see muscle tissue and vessels lying underneath the skin with the cranial bone in the background. In (i) we can see the same structure with lower opacity, revealing a fuzzy view on the brain. The presented results have been obtained using the settings from Table 5.1, row 3. Thereby, the epsilon value of 0.9 is necessary in order to detect the enamel structure. All remaining structures could also be found with an epsilon of 0.7. The used higher epsilon value entails an increased runtime resulting from the repeated traversal of the pre-epsilon-check. Moreover, the high epsilon also leads to a considerable amount of redundant initial PRHs, despite of the pre-epsilon-check (see Section 3.3.2 for explanation). However, the initial PRH filter removed them prior to the merging step. Another interesting aspect is that the precalculation results of this data set are very insensitive to different alpha values. Assumably, the reason for this is that the original histogram already comprises distinct peaks (see Figure 5.10). Note, using a finer block size, further substructures and variations of the current one could be identified. These are not shown.

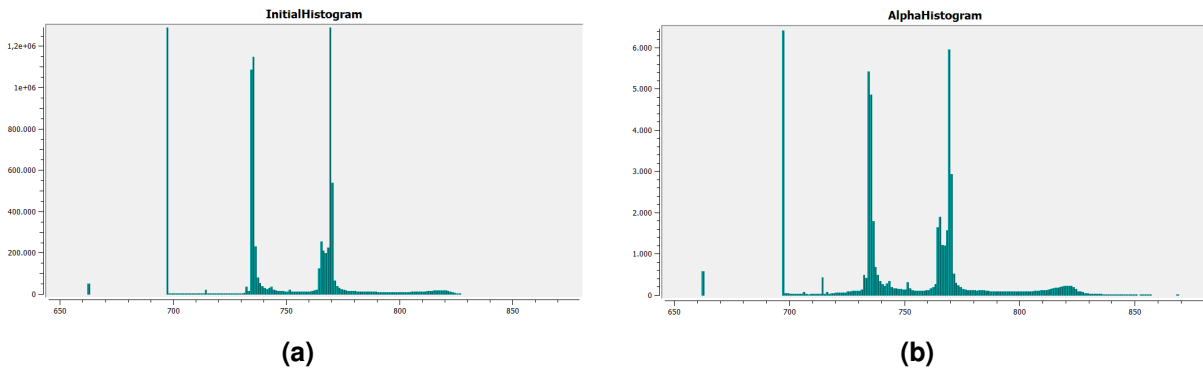
In order to compare the analysis results of the data set, we consider another one containing a clipping of a CT head scan - the incisix data set. It has been scaled to the dimensions 256x256x88 and has an intensity range reaching from -1024 to 3071. Using the same settings (see Table 5.1, row 4) similar structures as for the case2\_CT data set have been identified: The cranial bone, the skin overlapping with other soft tissue, the enamel as well as the interior of the bronchus and paranasal sinus which overlaps with the air around the head (see Figure 5.11). Only the contours of the latter structure as well as the muscle and vessel tissue, which have been detected in the case2 CT could not be identified. This indicates, that the use of similar precalculation settings for similar data sets delivers comparable results. Note, that the enamel of all teeth could be identified in the incisix data set. Assumably, because the teeth of the scanned person were still healthy.



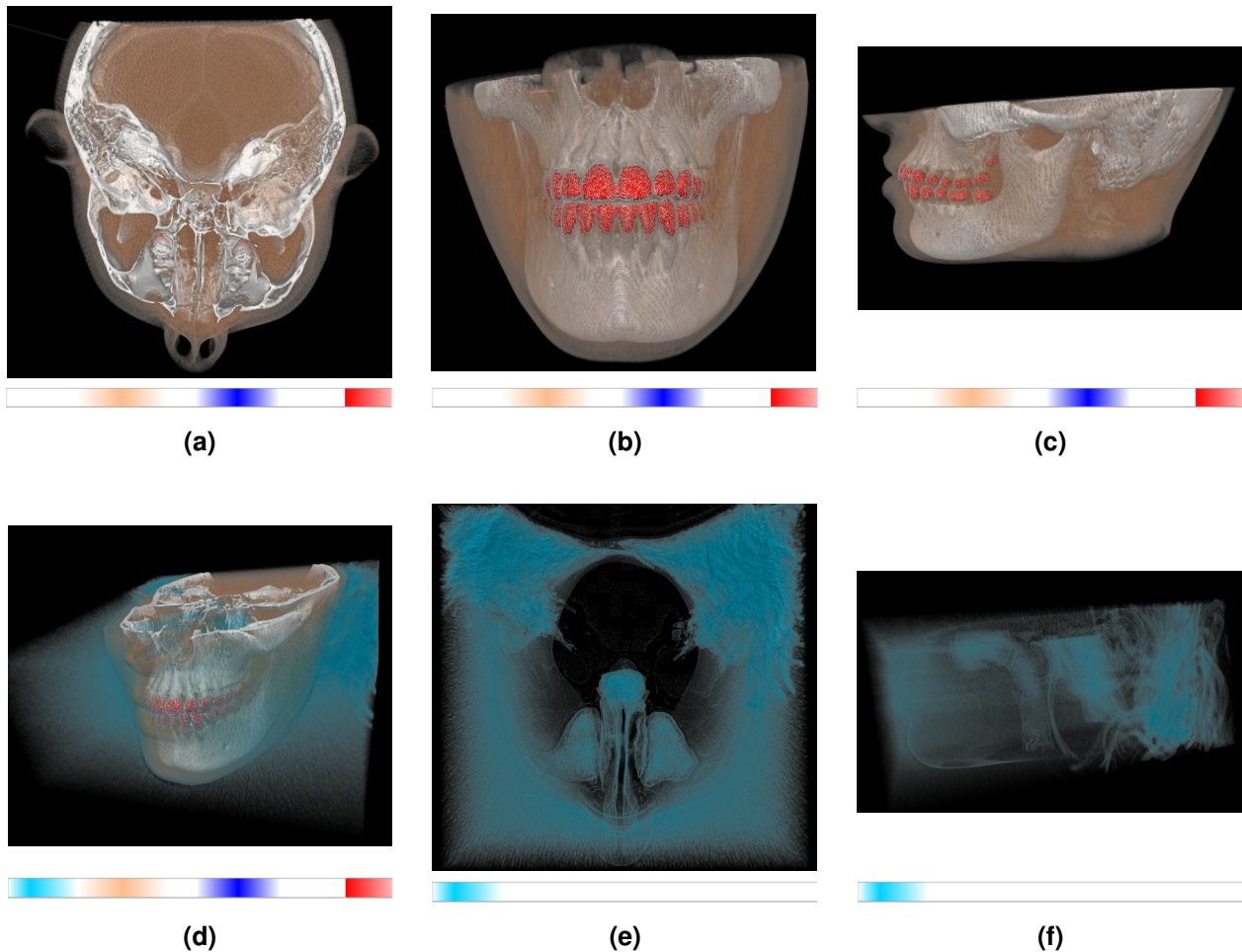
**Figure 5.8:** Dice example: Structures identified using "detailed" settings (see Table 5.1, row 2) but no AH calculation.



**Figure 5.9:** Case2\_CT example - cranial CT scan: (a), (b) and (c) show visualizations of a combination of the skin, cranial bone and enamel identified in the data set. (d) additionally shows a structure depicting the bronchus and paranasal sinus. (e) and (f) show visualizations of the contours of the bronchus and the paranasal sinus overlapping with the boarder between air and skin. (g) and (h) show detected muscle tissue and vessels in combination with the cranial bone. (i) shows the brain. In the TF space (i) is defined by the same value range as the muscle/vessels tissue shown in (g) and (f), but a lower opacity. Note, that white structures are depicted as blue regions in the TF space. The used settings can be found in Table 5.1, row 3).

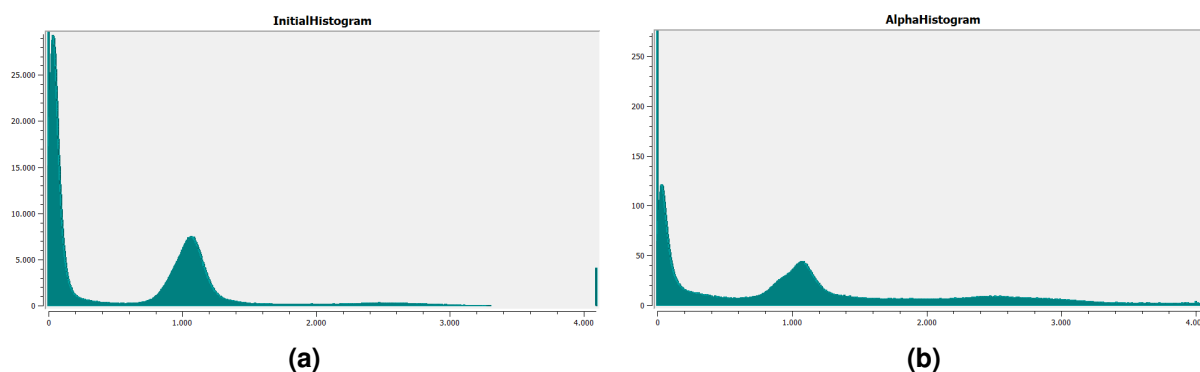


**Figure 5.10:** case2\_CT example - cranial CT scan: Initial histogram (a) and AH (b) of the case2\_CT data set. Both histograms show distinct peaks. The used alpha value can be found in Table 5.1, row 3).



**Figure 5.11:** Incisix example - cranial CT scan: (a), (b) and (c) show visualizations of a combination of the skin, cranial bone and enamel identified in the data set. (c) Shows a visualization of a combination of the skin, cranial bone, enamel as well as the interior of the bronchus and the paranasal sinus. (d) and (e) show visualizations of the interior of the bronchus and the paranasal sinus overlapping with the air around the skin. The used settings can be found in Table 5.1, row 4.



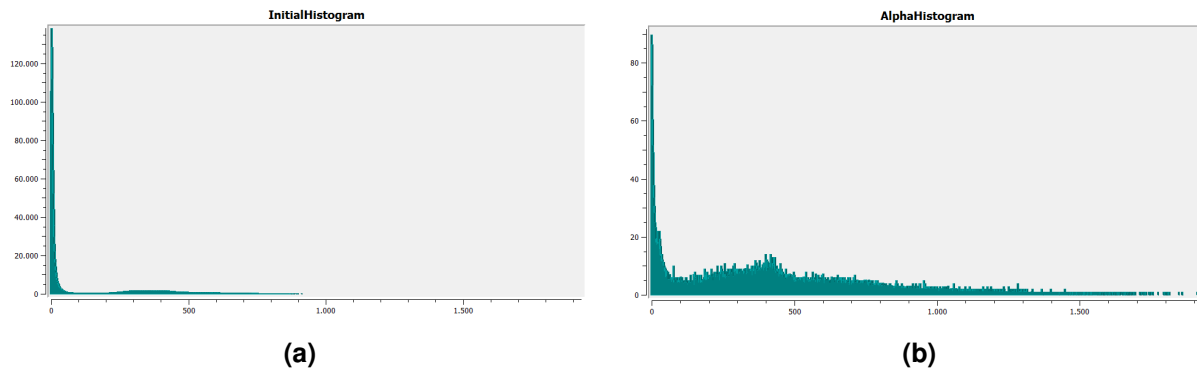


**Figure 5.12:** Incisix example - cranial CT scan: Initial histogram (a) and AH (b) of the Incisix CT data set.

**(Functional) Magnetic Resonance Imaging Examples - Case2\_T1\_post, Case2\_FLAIR and Case2\_fmri\_tMAP:** This paragraph presents and discusses several MRI data set visualizations. In general the extraction of distinct structures from MRI data sets is very difficult. The reason for this is the high bias and the associated overlapping intensity ranges of different tissues. This is especially true for data sets constructed from the traditional T1 and T2 parameters. For this reason Lundström et al. [54] focus on visualizing CT data sets with their PRHs. In the following we show rendering results of several MRI scans with our system.

At first we consider the case2\_T1\_post data set which comprises contrast agent enhanced vessels. It was captured using the T1 parameter and is a scan of the same head as from the case2\_CT. The data set was scaled to dimensions of  $128^3$  and has an intensity range from 0 to 1931. As a result of the high bias in the data set there are no distinct histogram peaks (see Figure 5.13 (a)) indicating structures. The AH improves this situation, but still delivers a flat intensity distribution as can be seen in Figure 5.13 (b). Therefore, it is not possible to extract distinct, non-overlapping tissues. However, the visualizations can be optimized to a certain degree by using according settings. We found the settings from Table 5.1, row 5 to provide suitable results. Using them, relatively good visualizations of the brain, the vessels and the skin/cranial bone could be identified (see Figure 5.14 (a) and (b)). The addressed overlaps, however, cannot be entirely circumvented. The brain overlaps with the chin skin and the boarder between air and skin/cranial bone. Despite of the contrast agent the vessels partly overlap with the skin/cranial bone at the back of the head although this is hardly visible with the current vessel value range and opacity settings. Figure 5.14 (c) and (d) show a combination of two separately extracted brain regions in combination with the already shown vessels. Using the settings from Table 5.1, row 6 a better separation of the brain could be identified (see Figure 5.14 (e) and (f)). However, no vessels could be found with these settings. This data set also contains a brain tumor. This tissue constitutes a very inhomogeneous region which overlaps with a lot of other tissues and could therefore not be extracted separately.

This data set is a good example to demonstrate the impact of the epsilon value on the identification of regions of interest. Therefore, we also run precalculations once with a very high and once with a very low epsilon without the pre-epsilon-check. Thereby, the application of a low epsilon of 0.6 with an adaptor of 0.1 identified very inhomogeneous structures. The identified skin is an especially interesting example to demonstrate the nature of MRI data sets and the impact of the epsilon value (see Figure 5.14 (g)). Its value range is distributed over almost the whole TF space indicating the high data set bias. Using a high epsilon value, this region could not be identified. Running the precalculations with an epsilon of 0.98 and an adaptor of 0.02 delivered a finer separation of the data sets into more homogenous structures forming parts of tissues. These "homogenous" initial PRHs – identified with a high epsilon – are of course voided to a certain extend as a result of the merging step. Still, the final structures were less homogenous than those identified with a low epsilon. While the results of these settings are also visually



**Figure 5.13:** Case2\_T1\_post example - cranial MRI scan: Initial histogram (a) and AH (b) of the case2-T1-post MRI data set. The used alpha value can be found in Table 5.1, row 6).

appealing, the associated long precalculation time is problematic. The fine epsilon settings lead to the extraction of more than 16000 initial PRHs of which approximately 15500 were redundant and deleted by the initial PRH filter. This entails a very long runtime of approximately eight minutes with initial PRH filtering.

The pre-epsilon-check turned out to be especially useful for the analysis of this data set. However, for very fine epsilon settings these settings can also entail a long runtime as explained in Section 3.3.2.

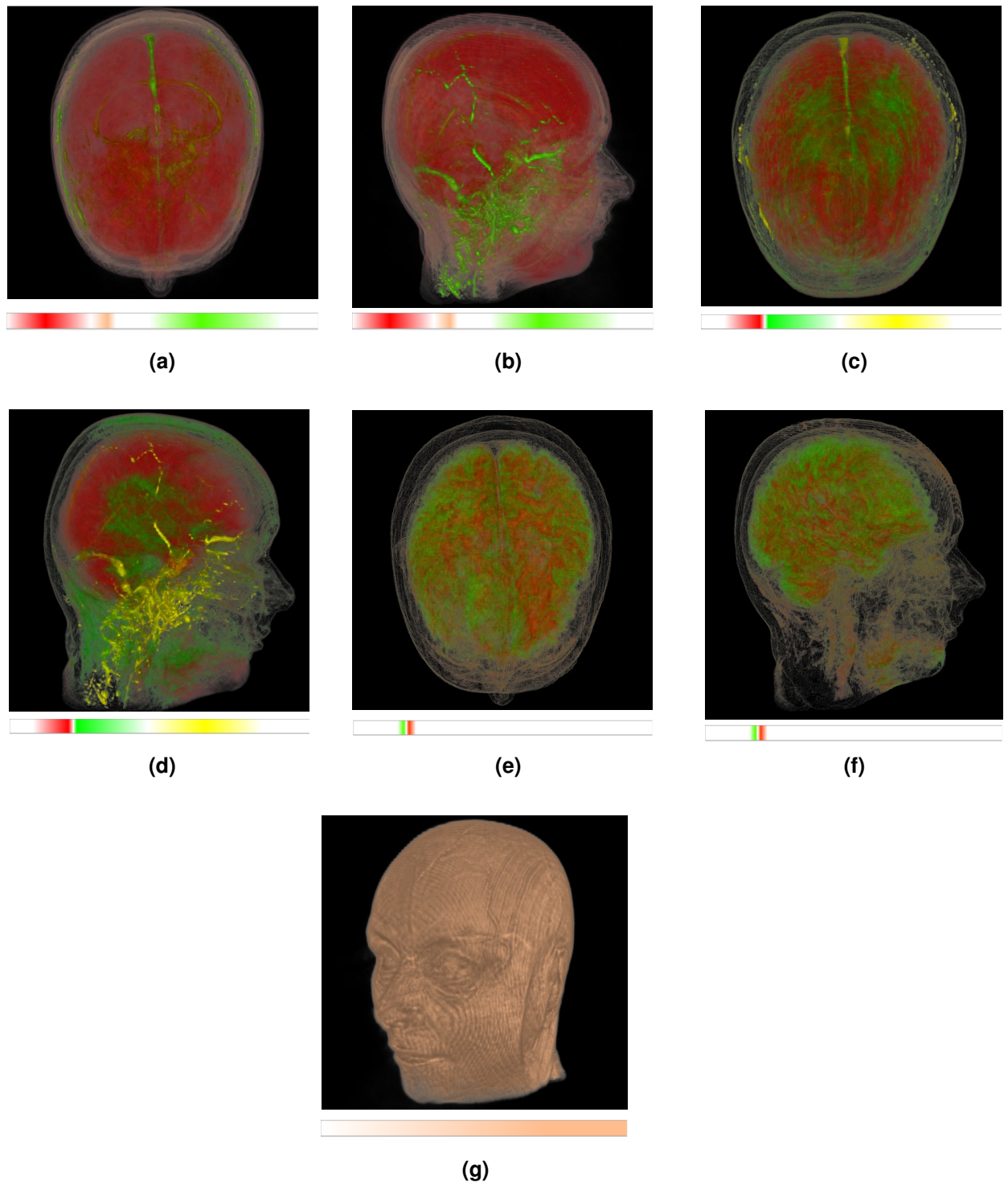
Another data set is a fluid attenuated inversion recovery (FLAIR) MRI scan of the case2 head. It was scaled to dimensions of  $128^3$  and has an intensity range from 0 to 1311. Since the FLAIR method allows to null out fluids it helps to improve the recognizability and separability of captured structures. Considering the histograms in Figure 5.15, we can still see a flat distribution which, however, comprises distinguishable peaks. As a result of the better separability of structures and the reduced overlaps the visualized structures are shown more clearly. Analyzing this data set we could also identify the lesion around the tumor which could not be found in the case2\_T1\_post data set. Figure 5.16 (a) and (b) show the most important structures identified including the brain, the tumor (in 2 regions), and the skin. Figure 5.16 (c) and (d) additionally show the tissue filling the subdural space, another tumor region and the borders between air and skin as well as skin and brain. Figure 5.16 (e) and (f) additionally show the air around the head.

The next presented data set is a fMRI scan of the case2 head. It was scaled to dimensions of  $166 \times 214 \times 231$  and an intensity range from -1024 to 1200. Using the settings from Table 5.1, row 8, a series of brain regions of different cerebral perfusion could be identified (see Figure 5.17). The corresponding original histogram and AH can be found in Figure 5.18 (a) and (b), respectively.

## 5.2.2 Visualizations Based on Multi-Dimensional Transfer Functions

**Case2\_T1\_pre and Case2\_T1\_brainmask:** The first example illustrating the multi-dimensional capabilities of the design gallery is based on the case2\_T1\_pre and case2\_brainmask data sets. The first has been scaled to a size of  $128^3$  and has an intensity range reaching from 0 to 3162. It is basically the same scan as the case2\_T1\_post data set, but without contrast agent enhanced vessels. Accordingly, the settings from Table 5.1, row 5 have been used for its precalculation. The second data set is a segmentation of the brain region of case2\_T1\_pre.

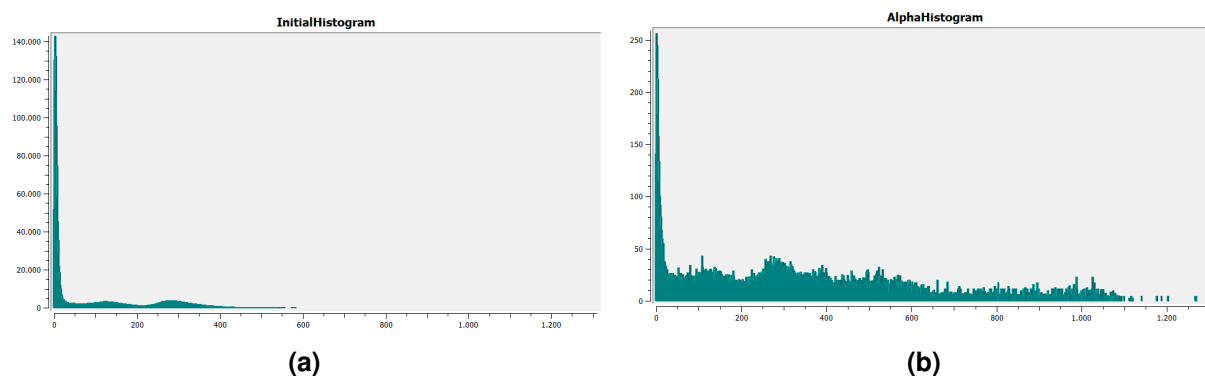
The results of the evaluation of the case2\_T1\_pre data set alone and in combination with the case2\_brainmask are shown in Figure 5.19. Thereby, the one-dimensional and multi-dimensional results are shown in Figures (a) - (h), (o), (p) and (i) - (n), respectively. When comparing the images we can see that all overlaps occurring in one-dimensional results (see Figure (e) - (f)) are eliminated in the



**Figure 5.14:** Case2\_T1\_post example - cranial MRI T1 scan: (a) and (b) show combined visualizations of the brain, the vessels and the skin/cranial bone. (c) and (d) show combined visualizations of two separately extracted brain regions in combination with the already shown vessels. The settings used for (a) - (d) can be found in Table 5.1, row 5. (e) and (f) show combined visualizations of two brain regions identified using the settings from Table 5.1, row 6. (g) shows a visualization of the skin detected using an epsilon value of 0.6 and an adaptor of 0.1 for the PRH calculation. From the TF below we can see, that this is a very inhomogeneous structure spreading over a large value range in the TF space.

Data set	Settings	Alpha	Block Size	Epsilon	Adaptor	Series Sizes	Multi Start	Multi End	Iterations	Options	Time
dice	default	3	8	0.9	0.1	2-16	2	0.2	3	RTF, PEC	00:05
dice	detailed	8	6	0.98	0.02	4-16	2	0.2	3	RTF, PEC	00:16
case2_CT	-	3	8	0.9	0.1	2-100	2	0.2	3	IPF, RTF, PEC	00:55
incisix	-	3	8	0.9	0.1	2-100	2	0.2	3	IPF, RTF, PEC	00:17
case2_T1_post	A	16	6	0.9	0.1	2-100	2	0.2	3	IPF, RTF, PEC	00:18
case2_T1_post	B	8	6	0.98	0.02	2-100	2	0.2	3	IPF, RTF, PEC	08:13
case2_FLAIR	-	8	8	0.9	0.1	2-100	2	0.2	3	IPF, RTF, PEC	00:17
case2_fMRI_tMAP	-	3	8	0.9	0.1	2-100	2	0.2	3	IPF, RTF, PEC	00:30

**Table 5.1:** Table, listing the used settings for the presented one-dimensional examples. Legend: *IPF* = *initial* – *PRH* – *filter*, *RTF* = *redundant* – *TF* – *filter* *PEC* = *pre* – *epsilon* – *check*



**Figure 5.15:** Case2\_FLAIR example - cranial MRI FLAIR scan: Initial histogram (a) and AH (b) of the case2\_flair MRI data set. The used alpha value can be found in Table 5.1, row 7).

corresponding multi-dimensional ones (see Figure (i) - (l)). The brain identified in 1D (see Figure (e) and (f)) for example overlaps with a lot other soft tissue which makes it fuzzy and harder to recognize. The brain identified in combination with a second dimension (see Figure (i) and (j)) on the other hand is distinctly separated from other tissues and can be seen very clearly. Similarly, the one-dimensional results shown in Figure (g) and (h) show the boarder of the air and skin as well as that between the subdural space and the brain. If we take a closer look we can additionally recognize the contours of the eyes and the brain tumor lesion contained in the data set. A combination of this structure with the segmented brain region contained in case2\_brainmask on the other hand only shows the boarder between the subdural space and brain and the tumor contours.

Figure (a) - (b) and (m) - (n) show a combination of the most important one-dimensional and multi-dimensional structures. Thereby, we recognize that the multi-dimensional results are much more distinct. However, we can also see that they contain less structures. As a result of the *AND* concatenation used for volume evaluation, only overlapping regions are depicted. Since case2\_brainmask only contains one structure comprising the brain region there are no overlaps for many case2\_T1\_pre structures. Hence, they are not identified in the multi-dimensional classification.

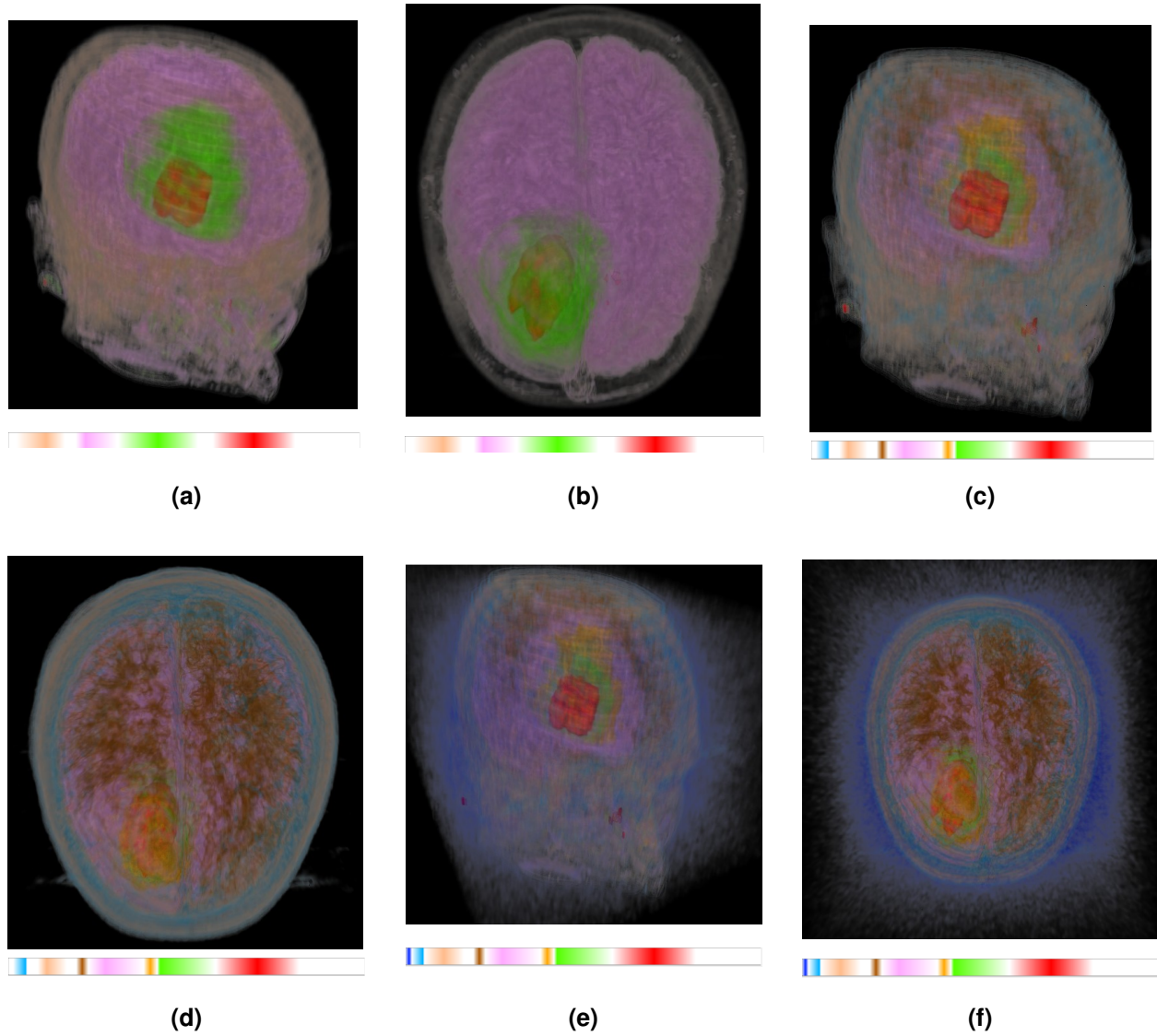
Note, the color test has been deactivated for this example. Other wise the concatenation of multi-dimensional structures shown in (m) and (n) would not be possible.

**Pig42 Example:** This example uses a series of volumes each containing one tissue segmented from the pig42 data set, which is a CT scan of a pigs torso, or a structure derived from them. These are a liver, vessels, two lesions and two dilatations of the lesions marking a region of several cm around them. The dilatations have been created using a mask of five voxels radius. The vessel volume has been scaled to dimensions of  $128^3$ , all others to  $64^3$ .

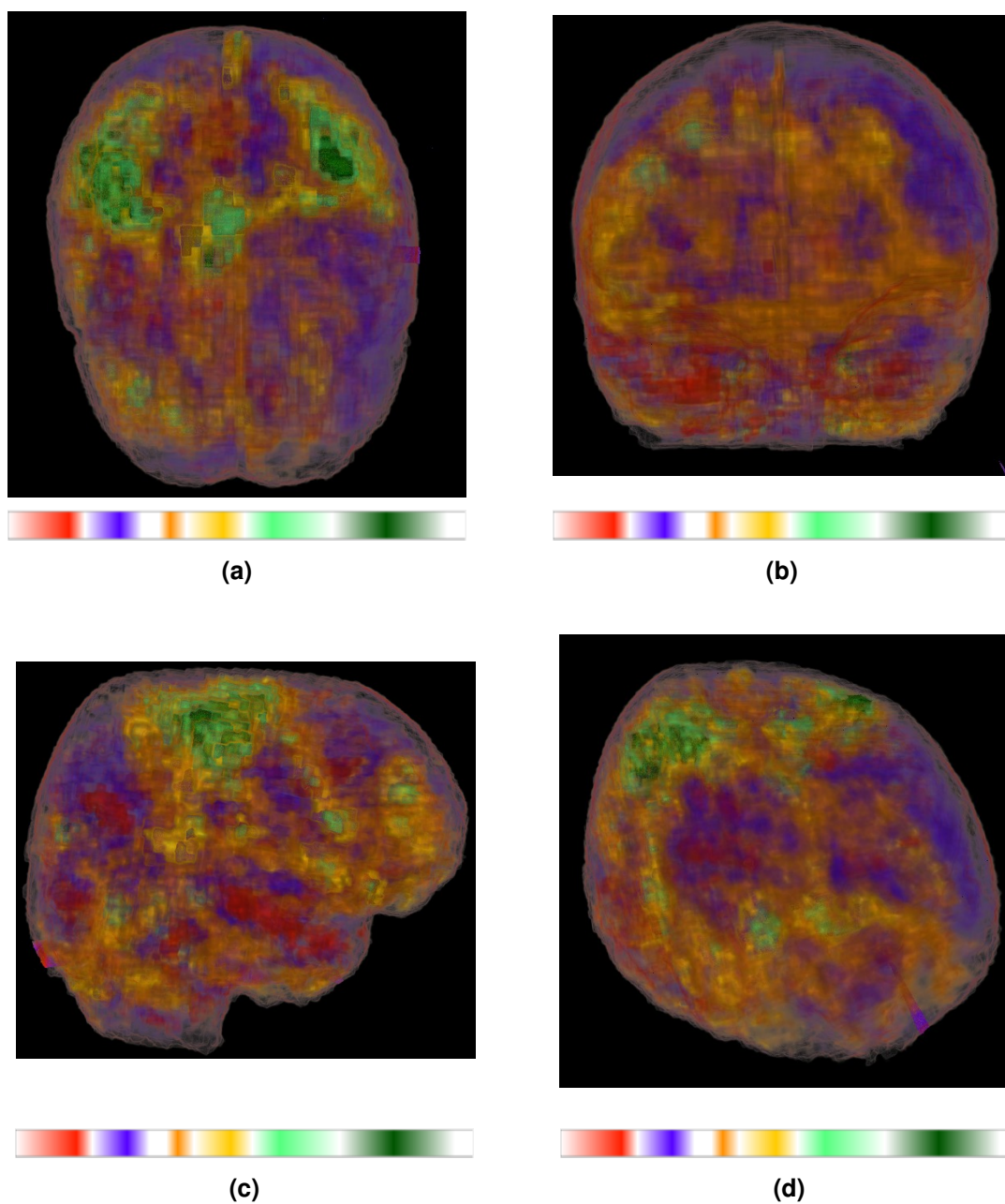
The visualizations presented in this example are created with the preliminary nD prototype presented in Section 3.6.2. Using segmented, definite structures allow to clearly demonstrate how the prototype addresses the restrictions of the basic multi-dimensional approach.

In order to allow compositions of structures as shown in Figure 5.20 single regions of interest of one dimension must be part of multiple regions of interest of arbitrary dimensionality and volumes, whose  $RGB\alpha$  values can be independently adapted. Each combined region of interest must evaluated separately.

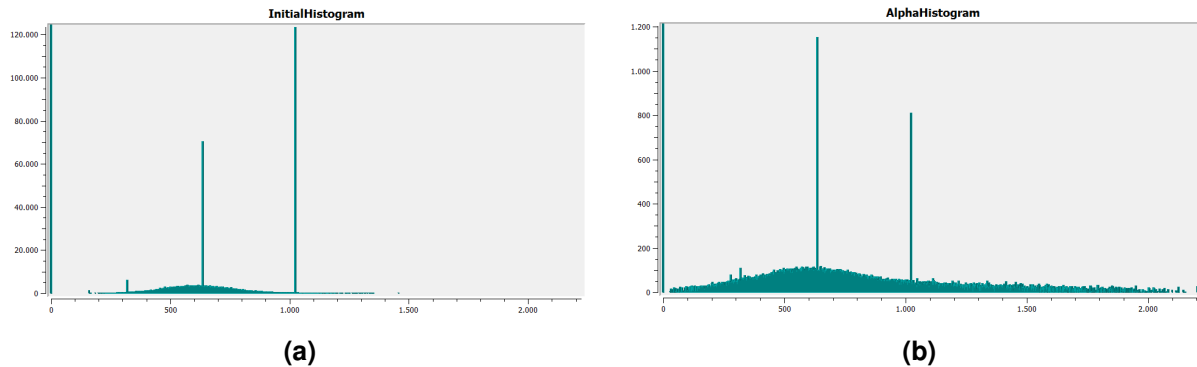
In Figure 5.20 (a) each volume is rendered with an *OR* concatenation in order to show the spatial relation of their structures. (b) and (c) depicts visualizations which would be reasonable for real tissues. (b) contains structures resulting of *OR* concatenations of the liver, vessels and lesion. In addition, a



**Figure 5.16:** Case2\_FLAIR example - cranial MRI FLAIR scan: (a) and (b) are showing a visualization of a combination of the identified skin(cranial bone), tumor and brain. (c) and (d) additionally show tissue filling the subdural space (fat tissue), another tumor region and the borders between air and skin as well as skin and brain. (e) and (f) additionally show the air around the head. The used settings can be found in Table 5.1, row 3).



**Figure 5.17:** Case2\_fMRI\_tMAP example - cranial scan: Shows multiple brain regions of different cerebral perfusion from (a) top, (b) front, (c) side and (d) skew view. The used settings can be found in Table 5.1, row 8).



**Figure 5.18:** Case2\_fMRI\_tMAP Example - Cranial fMRI Scan: Initial histogram (a) and AH (b) of the case2\_fMRI\_tMAP data set. The used alpha value can be found in Table 5.1, row 8).

structure resulting from an *AND* concatenation of the dilated lesion and the vessels is included. From this visualization a surgeon who wants to remove a malignant structure could clearly see the vessels parts close to it which can easily be injured. In (c) this visualization is further extended. It additionally shows a second lesion (*OR*), a structure resulting of an *AND* concatenation of the dilatation of this lesion with the vessels and a structure resulting from the *AND* concatenation of both dilatations and the vessels. This means our combined visualization in (c) consists of regions of interest of 1, 2 and 3 dimensions. This composition would not be possible in the basic multi-dimensional approach.

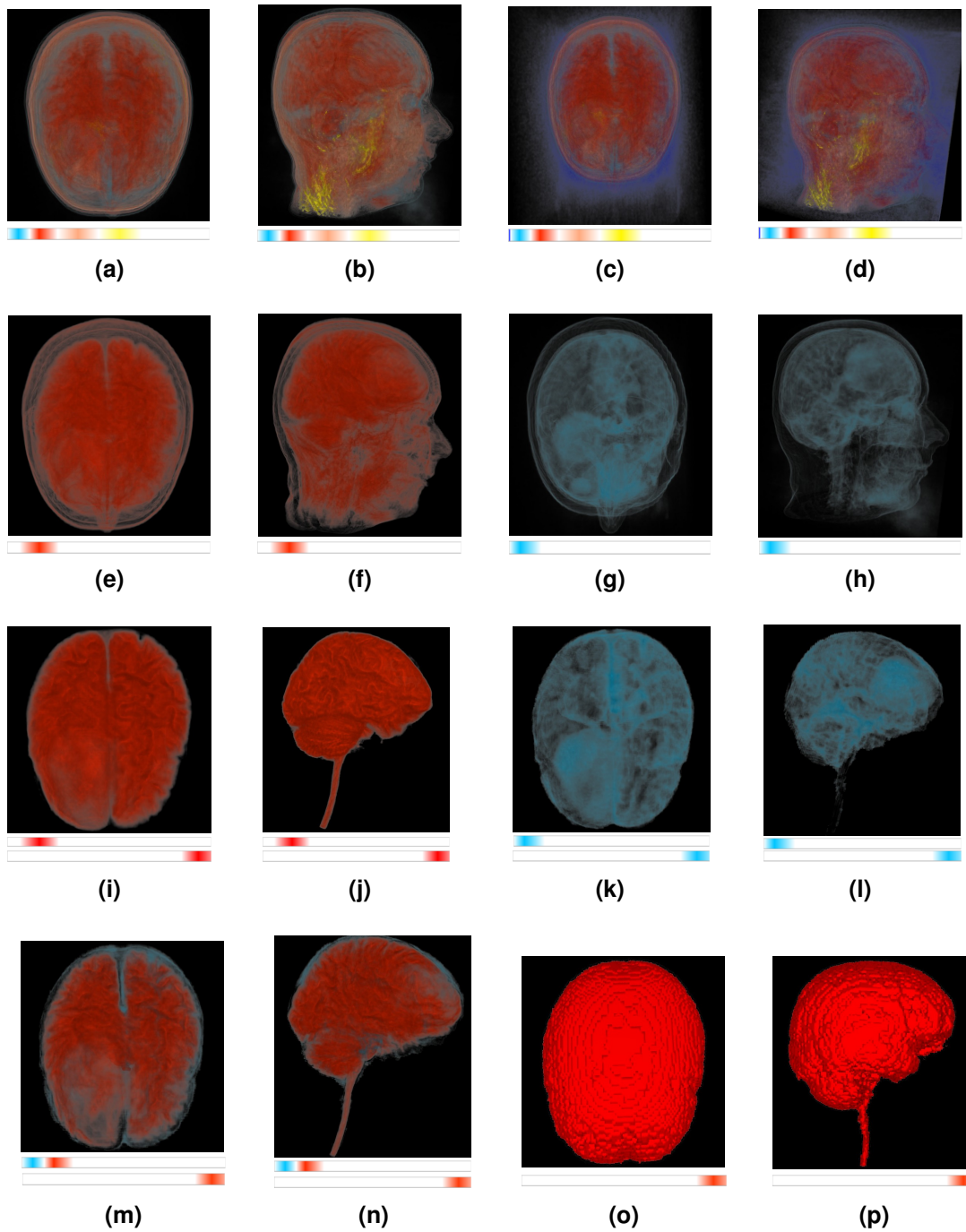
**Pat14 Example:** The purpose of the pat14 example is to demonstrate visualizations resulting from multi-dimensional classification of real medical data sets. It is based on two CT scans (pat14\_002, pat14\_007) of a human torso. In the second one heart and aorta are contrast agent enhanced. The intensity ranges of the first and second volume reach from -1024 to 1475 and -1024 to 1142, respectively. Both have been scaled to a size of  $128 \times 128 \times 80$ .

Figure 5.21 (a) and (b) show two structures of the non contrast agent enhanced volume pat14\_002. Both are concatenated with the structure shown in (c), which is extracted from pat14\_007. The results of the *AND* concatenation between the structures shown in (a) and (c) as well as (b) and (c) are depicted in (d) and (e), respectively.

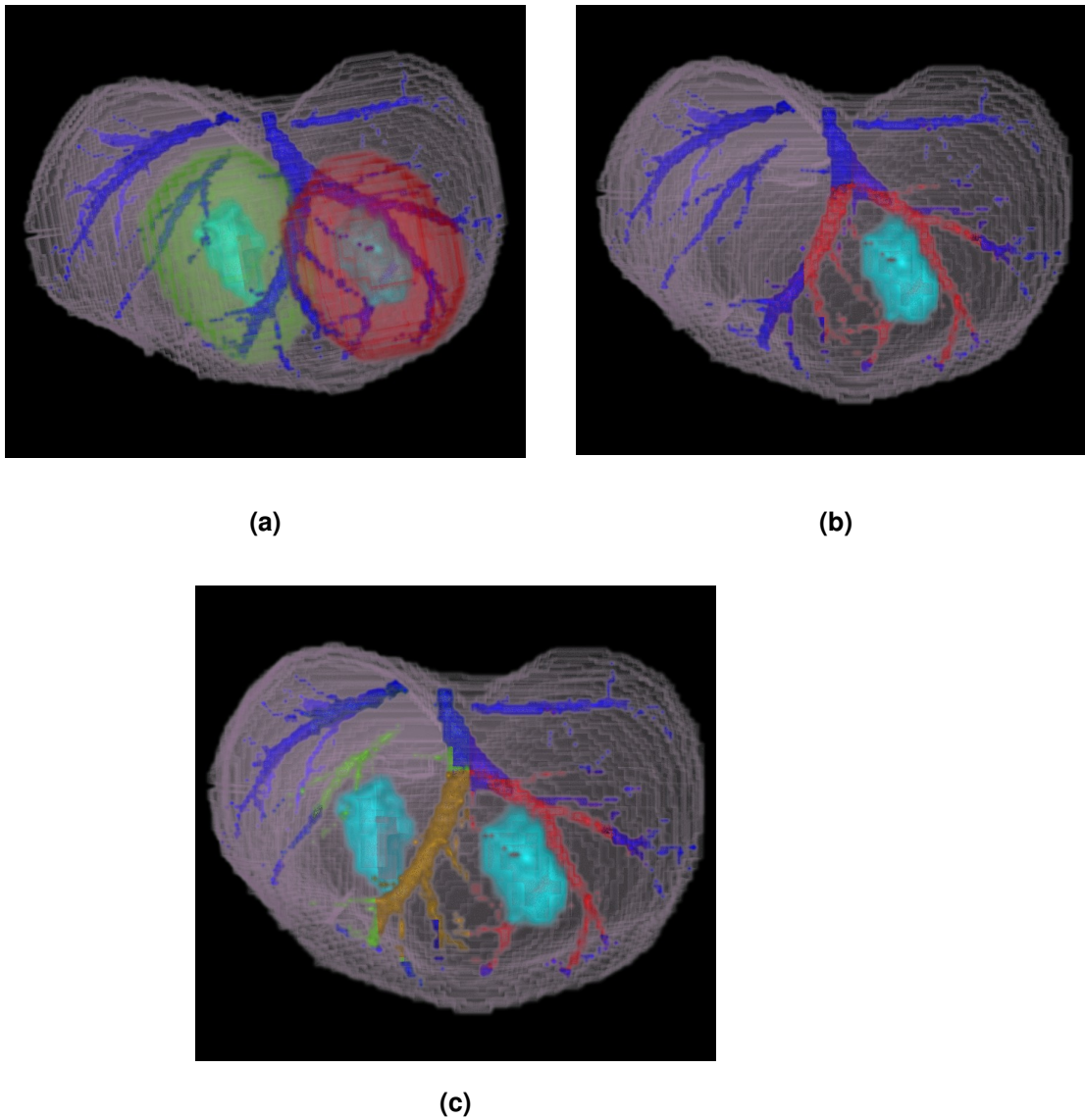
(c) shows, that the value range of spine and ribs interferes with that of the contrast agent enhanced aorta and heart in pat14\_007. Hence, using one-dimensional TFs, distinguishing these structures as well as visualizing them with different optical properties, is impossible. In pat14\_002, on the other hand, most soft tissues are within the same value range as shown in (b). Moreover, the spine and ribs shown in (a) interfere with a background structure not belonging to the body.

By concatenating (b) and (c) heart and aorta can be isolated. The ribs and spine from (c) as well as the remaining soft tissue from (b) disappear. From concatenating (a) and (c) spine and ribs can be extracted. The aorta and heart from (c) as well as the background structure from (a) are not depicted. By combining both multi-dimensional structures, a visualization with spine and ribs being clearly differentiated from heart and aorta can be obtained (see Figure 5.21 (f)).

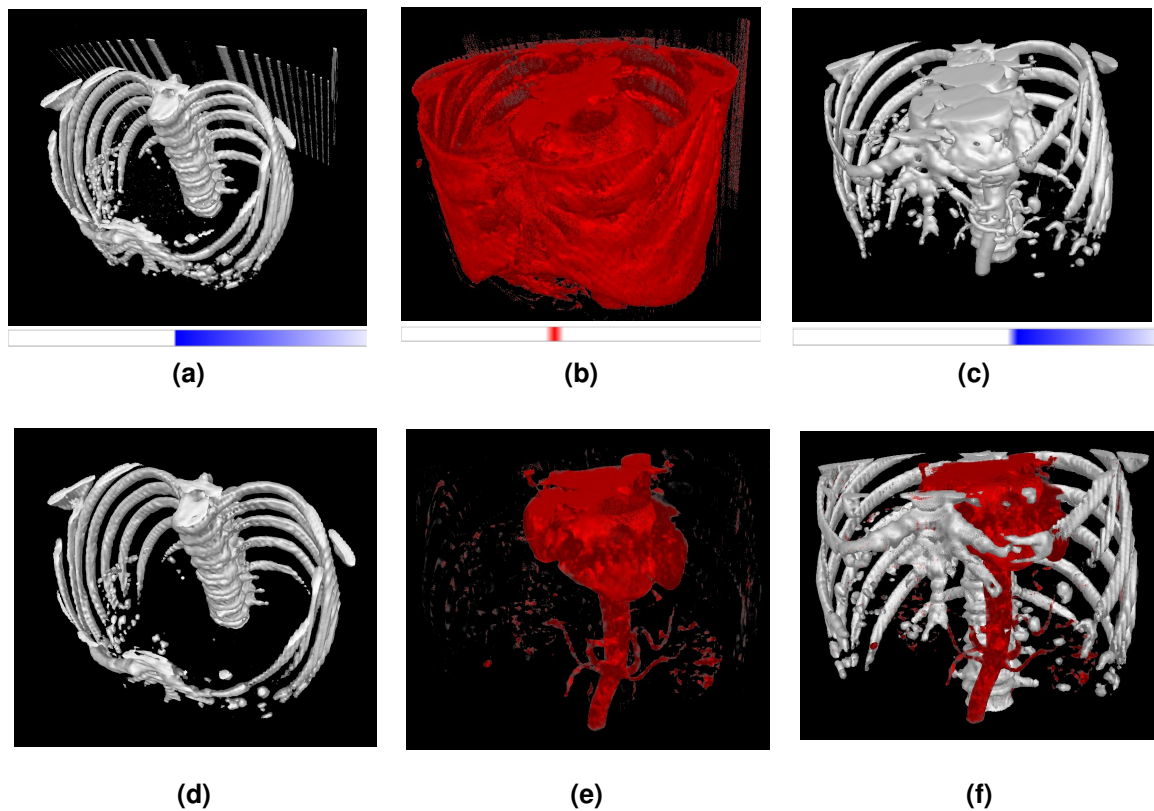




**Figure 5.19:** Case2\_T1\_pre - case2\_brainmask Example: (a) - (d) show compositions of one-dimensional structures of the case2\_T1\_pre data set. (o) and (p) show the segmented brain mask contained in the case2\_brainmask data set. (e)-(h) show two structures of the case2\_T1\_pre alone. (i)-(l) shows an AND concatenation of these structures with the brainmask of the second volume. (m) and (n) shows a composition of both multi-dimensional structures.



**Figure 5.20:** Pig42 example: (a) shows the spatial relation of all structures used for this example by applying *OR* concatenations to their volumes. (b) shows four structures: liver, lesion, vessels and those parts of the vessels close to the lesion. The last structure results from an *AND* concatenation of the dilatation of the lesion and the vessels, all others from *OR* concatenations. (c) shows seven structures: liver, two lesions, vessel as well as vessel parts close to the first, second and both lesions. The last structure results from an *AND* concatenation of the dilatation of both lesions and the vessels, the two before from an *AND* concatenation of the dilatation of one lesion and the vessels, all others from *OR* concatenations.



**Figure 5.21:** Pat14 Example: (a) shows spine and ribs of pat14\_002. They interfere with a background structure not belonging to the body. (b) shows soft tissue of equal intensity of pat14\_002. (c) shows spine and ribs as well as contrast agent enhanced heart and aorta of pat14\_007. As a result of the contrast agent these structures share the same value range in the TF space. (c) shows spine and ribs resulting from an *AND* concatenation of the structures in (a) and (c). (d) shows heart and aorta resulting from an *AND* concatenation of the structures in (b) and (c). (f) results from the combination of the two-dimensional structures shown in (d) and (e). Note, that white structures are depicted as blue regions in the TF space.



## Chapter 6

# Future Work

An interesting extension of the precalculation would be (semi-)automatic methods helping to find initial settings for certain data sets. This could be approaches that estimate the size of data set structures or the volume homogeneity to suggest a block size or epsilon value. Moreover, the search for different initial value ranges by introducing new automatically adjusted parameters can be extended. For example, the merging criterion in the PRH calculation could be adapted to use separate multipliers to check mean and deviation differences. Other possibilities would be to create more initial PRHs by performing their computation for different block sizes and epsilon values. As the computational effort extends it may also be useful to establish GPU solutions for the precalculation. This will be possible as soon as fast memory structures on the GPU get large enough to facilitate our extended histogram calculations.

It would also be interesting to test multiple curve fitting approaches to define the value ranges for the partial range calculation. Finally, the precalculation results should be compared with reference results according to quantitative criteria to be able to objectively rate their quality. This would also allow to reliably test how new features contribute to the precalculation process.

Improving the user interface is mainly about facilitating TF design by enhancing its intuitivity and providing new practical functions. A beneficial enhancement for the CF based user interface would be to allow switching between multiple widgets in the selected thumbnails area of which each displays references to a set of thumbnails. As a result it would be possible to store multiple groups of structures. At the same time the selected thumbnails area could be adapted to allow to deselect thumbnails without removing them from the area. This would make it easier to test the impact of certain structures on the combined rendering result. An additional button could allow to set all thumbnails referenced in a preview widget selected in order to allow a fast switching between structure combinations. These adaptations aim to facilitate keeping track of structure combinations, improving the overview and switching between combinations. Primary improvements for the DM based user interface concern its intuitivity. More common keys should be used. Testing different structure variations could be facilitated by allowing only one selected thumbnail in an according circle. The selection could be made easier by defining an extra key. Another point of criticism on the DM based user interface was the bad recognizability of structures. To that end the remaining space in the exploration area could be used to provide a large view on several thumbnails - for example the three last selected ones. Another potential improvement is to provide an easy way to swap the selection of whole groups of thumbnails. This could for example be achieved by dragging a frame around a multiple thumbnails. Allowing a further processing of saved combinations would be practical. It would be also of interest to allow storing adaptations that are performed in the TFs editor. This however is difficult because there is no simple way to keep track of the correlation between QGradientStops and the positions defined in the RegionData data structure. Finally, the DM and CF based user interface could be combined to enable users to switch between desired views as required at runtime. Therefore, changes performed in one user interface must automatically affect the depiction of the other one.

The major prospective task concerning the multi-dimensional capabilities of the design gallery is to fully integrate the preliminary nD prototype described in Section 4.11. A significant challenge associated with this approach is further increased amount of initial detected structures. In order to handle this situation, different actions can be taken. On the one hand further filters could be deployed to avoid redundant structures which may also result from different concatenations. On the other hand the user interface could be adapted to arrange thumbnails according to similarity. This could be done on the DM while it would be very challenging in the CF user interface. Alternatively, only one of many similar structures could be depicted. Similar structures could be shown only if invoked by certain user actions.

Another possible extension of the design gallery would be to allow to define regions of interest by multiple different shapes (see Figure 2.26).

## Chapter 7

# Conclusion

The primary goal of this thesis is to present a system for the efficient design of qualitative TFs. Therefore, we consider three major aspects. The automatic precalculation of initial TFs, the creation of an effective user interface for an individual task specific adaptation and the high dimensional classification of data set features. Throughout this thesis we show a series of approaches dealing with each of these aspects. The presented design gallery combines all three in a comprehensive framework for efficient TF design.

The precalculation identifies all essential structures contained in a data set. Even very small and very close peaks can be identified. This is achieved by a well integrated combination of the AH and PRH method. The quality of the results is limited by the character of the analyzed data sets. Overlapping value ranges between different structures, which especially occur in MRI data sets of high bias, prevent a distinct separation of tissues. In order to facilitate the identification of appropriate structures we create more distinct histogram peaks during the AH calculation. We extend our search to a broad range of regions of interest due to the extended merging step (see Section 3.3.2) and added an additional pre-epsilon-check (especially useful for MRI) to find an appropriate homogeneity level for structures of certain value ranges. Improvements of the precalculation results are mainly identified for data sets with flat histogram distributions.

The additional smoothing step as a part of the AH precalculation is helpful for data sets with very jaggy histograms. These frequently occurs for histograms enhanced with a high alpha value. Apart from the block size the epsilon value and according adaptor are the most important parameters to influence the precalculation result. Due to the application of the pre-epsilon-check we manage to find more suitable homogeneity levels for certain value ranges. In addition, this also reduces the precalculation time by avoiding redundant peak identifications. All in all the precalculation process depends on a series of parameters, which must be optimized to find appropriate and desired structures. However, similar settings can be used for similar data sets. Hence, once found settings can be reused via an according save and load functionality of the design gallery. In order to perform the precalculation process within a reasonable time a series of optimizations have been deployed. This includes caching of repeatedly required values as well as the improved execution order of the AH calculation. The regular application of filters removing redundancies helps to keep the data amount for subsequent steps low. Further accelerations are achieved by sorting initial PRHs according to increasing mean values and using accumulated histograms to count the amount of voxels within value ranges.

From our user interface evaluation we conclude that the DM and CF based user interface clearly facilitate and accelerate the design of TFs in comparison to manual editors. Especially the CF based approach was highly appreciated for its simple and intuitive use. Some users missed an overall overview of all thumbnails. Apart from that only minor suggestions for improvements were expressed. The DM based approach was praised for its good overview of all thumbnails and for the high degree of freedom for their arrangement. However, especially experienced users demanded a simplification of the DM navigation - especially in terms of using more common keys.

One-dimensional TFs are frequently insufficient to clearly separate data set structures. Multi-dimensional TFs are usually very memory intensive and tedious to design. It is very difficult to provide user interfaces for the adjustment of TFs of more than three dimensions. Therefore, the presented design gallery was implemented to work with multi-dimensional separable TFs. This allows to work with an arbitrary amount of data value dimensions without intensive memory requirements. Thereby, the provided user interface hides the complexity increasing with the dimensionality of separable TFs. As a result the TF design process always stays equally simple for users irrespective of the currently deployed dimensionality. The basic multi-dimensional approach however still revealed some weaknesses. On the one hand, the color test prohibits the combination of multi-dimensional regions of interest with a shared one-dimensional one. On the other hand, it does not allow to generate and combine structures of different dimensionality. These issues have been addressed with an additional preliminary nD prototype. This prototype provides flexible concatenations between volumes and an independent evaluation of the single combined regions of interest. As a result it allows to create and combine regions of interest of arbitrary dimensionality and makes the color test dispensable.



# Bibliography

- [1] Beyer, J. (2009). *GPU-based Multi-Volume Rendering of Complex Data in Neuroscience and Neurosurgery*. Phd thesis, Vienna University of Technology. (Cited on page 6.)
- [2] Blinn, J. F. (1982). Light reflection functions for simulation of clouds and dusty surfaces. *SIGGRAPH Comput. Graph.*, 16(3):21–29. (Cited on page 11.)
- [3] Bui, A. A. T. and Taira, R. K. (2010). *Medical Imaging Informatics*. Springer Publishing Company, Incorporated, 1st edition. (Cited on page 6.)
- [4] Buxton, R. B. (2002). *Introduction to Functional Magnetic Resonance Imaging*. Cambridge University Press. (Cited on page 6.)
- [5] Cameron, G. G. and Undrill, P. (1992). Rendering volumetric medical image data on a simd-architecture computer. In *Proceedings of Third Eurographics Workshop on Rendering*, pages 135–145, Bristol, UK. (Cited on page 14.)
- [6] Carr, H., Brian, D., and Brian, D. (2006). On histograms and isosurface statistics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1259–1266. (Cited on page 43.)
- [7] Chaudhri, I. (2010). Animated graphical user interface for a display screen or portion thereof. (Cited on pages 2, 52, 53, 54, 57, 73 and 136.)
- [8] Cockburn, A. and McKenzie, B. (2001). 3d or not 3d?: evaluating the effect of the third dimension in a document management system. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA. ACM. (Cited on page 52.)
- [9] Correa, C. and Ma, K.-L. (2008). Size-based transfer functions: A new volume exploration technique. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1380–1387. (Cited on page 29.)
- [10] Correa, C. and Ma, K.-L. (2009a). The occlusion spectrum for volume classification and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1465–1472. (Cited on pages 18, 19, 20, 30, 31, 32 and 38.)
- [11] Correa, C. D. and Ma, K.-L. (2009b). Visibility-driven transfer functions. In *PACIFICVIS '09: Proceedings of the 2009 IEEE Pacific Visualization Symposium*, pages 177–184, Washington, DC, USA. IEEE Computer Society. (Cited on pages 32, 33 and 34.)
- [12] Correa, C. D. and Ma, K.-L. (2011). Visibility histograms and visibility-driven transfer functions. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):192–204. (Cited on page 34.)
- [13] Crow, F. C. (1984). Summed-area tables for texture mapping. *SIGGRAPH Comput. Graph.*, 18(3):207–212. (Cited on page 15.)

- [14] Dössel, O. (2000). *Bildgebende Verfahren in der Medizin: von der Technik zur medizinischen Anwendung*. Springer-Verlag. (Cited on pages 5 and 6.)
- [15] Engel, K., Hadwiger, M., Kniss, J. M., Rezk-Salama, C., and Weiskopf, D. (2006). *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA. (Cited on pages 5, 6, 7, 8, 9, 11, 12, 19, 20, 21, 55 and 56.)
- [16] Fernando, S. (2011). Stackoverflow: Odometer implementation. <http://stackoverflow.com/questions/1700079/howto-create-combinations-of-several-vectors-without-hardcoding-loops-in-c>. (Cited on page 94.)
- [17] Finn, R. (2010). Imageflow. <http://finnrudolph.de/ImageFlow>. Last visited: 27.08.2010. (Cited on pages 52 and 53.)
- [18] Foundation, W. (2010). Cover flow. Last visited: 27.08.2010. (Cited on page 53.)
- [19] Gonzalez, R. C. and Woods, R. E. (2008). *Digital Image Processing*. Pearson Prentice Hall, 3rd edition. (Cited on pages 5 and 21.)
- [20] Hadwiger, M., Ljung, P., Salama, C. R., and Ropinski, T. (2008). Advanced illumination techniques for gpu-based volume raycasting. Course Notes. (Cited on page 8.)
- [21] Haidacher, M., Patel, D., Bruckner, S., Kanitsar, A., and Gröller, M. E. (2010). Volume visualization based on statistical transfer-function spaces. In *PacificVis'10: Proceedings of the 2010 IEEE Pacific Visualization Symposium*, pages 17–24. IEEE Computer Society. (Cited on pages 19, 20, 34, 35, 36, 37 and 38.)
- [22] He, T., Hong, L., Kaufman, A., and Pfister, H. (1996). Generation of transfer functions with stochastic search techniques. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 227–ff, Los Alamitos, CA, USA. IEEE Computer Society Press. (Cited on pages 19, 44 and 50.)
- [23] Hidayat, A. (2010). Pictureflow. <http://code.google.com/p/pictureflow/>. Last visited: 27.10.2010. (Cited on pages 53 and 122.)
- [24] Hladuvka, J., König, A., and Eduard, G. (2000). Curvature-based transfer functions for direct volume rendering. In *Proceedings of Spring Conference on Computer Graphics and its Applications 2000 (SCCG 2000)*, pages 58–65, Budmerice, Slovakia. (Cited on pages 18 and 28.)
- [25] Huang, J., Shareef, N., Crawfis, R., Sadayappan, P., and Mueller, K. (2000). A parallel splatting algorithm with occlusion culling. (Cited on page 12.)
- [26] Ibáñez, L., Schroeder, W., Ng, L., Cates, J., and the Insight Software Consortium (2005). *The ITK Software Guide Second Edition Updated for ITK version 2.4*. (Cited on page 105.)
- [27] IEEE Computer Society (2010). Ieee visualization contest 2010. <http://viscontest.sdsc.edu/2010/>. (Cited on page 140.)
- [28] Jonsson, M. (2005). Volume rendering. Master thesis, Umea University. (Cited on pages 14 and 15.)
- [29] Kainz, B., Grabner, M., Bornik, A., Hauswiesner, S., Muehl, J., and Schmalstieg, D. (2009). Ray casting of multiple volumetric datasets with polyhedral boundaries on manycore gpus. *ACM Trans. Graph.*, 28(5):152. (Cited on pages 12, 57, 58, 91, 103, 104 and 125.)
- [30] Kajiya, J. T. and Von Herzen, B. P. (1984). Ray tracing volume densities. *SIGGRAPH Comput. Graph.*, 18(3):165–174. (Cited on page 11.)

- [31] Köhler, W., Schachtel, G., and Voleske, P. (1996). *Biostatistik*, volume 2. Springer-Verlag. (Cited on page 64.)
- [32] Kindlmann, G. (1999). Semi-automatic generation of transfer functions for direct volume rendering. Masters thesis, Faculty of the Graduate School of Cornell University. (Cited on pages 16 and 21.)
- [33] Kindlmann, G. and Durkin, J. W. (1998). Semi-automatic generation of transfer functions for direct volume rendering. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 79–86, New York, NY, USA. ACM. (Cited on pages 18, 19, 23, 24, 32, 46 and 50.)
- [34] Kindlmann, G., Whitaker, R., Tasdizen, T., and Moller, T. (2003). Curvature-based transfer functions for direct volume rendering: Methods and applications. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 67–ff, Washington, DC, USA. IEEE Computer Society. (Cited on pages 28 and 38.)
- [35] Kirk, D. and mei Hwu, W. (2008). Cuda threads. <http://courses.engr.illinois.edu/ece498/a1/textbook/Chapter3-CudaThreadingModel.pdf>. (Cited on page 41.)
- [36] Kitware Incorporation (2010a). Cmake. <http://www.cmake.org/cmake/project/about.html>. (Cited on page 103.)
- [37] Kitware Incorporation (2010b). Insight toolkit. <http://www.itk.org/>. (Cited on page 103.)
- [38] Kitware Incorporation (2010c). Insight toolkit: About. <http://www.itk.org/ITK/project/about.html>. (Cited on page 103.)
- [39] Kitware Incorporation (2011). Insight toolkit 4.0 documentation. <http://www.itk.org/Doxygen/html/classes.html>. (Cited on pages 105 and 106.)
- [40] König, A. H. and Gröller, E. M. (1999). Mastering transfer function specification by using volume-pro technology. (Cited on pages 18, 19, 47, 48, 49 and 50.)
- [41] Kniss, J. (2003). Transfer functions: Classification. Course Slides. (Cited on page 17.)
- [42] Kniss, J., Kindlmann, G., and Hansen, C. (2001). Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 255–262, Washington, DC, USA. IEEE Computer Society. (Cited on pages 19, 29, 50 and 51.)
- [43] Kniss, J., Kindlmann, G., and Hansen, C. (2002). Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285. (Cited on pages 24 and 25.)
- [44] Kniss, J., Premoze, S., Ikits, M., Lefohn, A., Hansen, C., and Praun, E. (2003). Gaussian transfer functions for multi-field volume visualization. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 65–ff, Washington, DC, USA. IEEE Computer Society. (Cited on pages 1, 19, 50 and 55.)
- [45] Koenderink, J. J. (1984). The structure of images. *Biological Cybernetics*, 50(5):363–370. (Cited on page 29.)
- [46] Kongsberg SIM AS (2010). Quarter 1.0.1a documentation. <http://doc.coin3d.org/Quarter/>. (Cited on pages 104 and 110.)
- [47] Kongsberg SIM AS (2011). Coin3d 4.0.0a documentation. <http://doc.coin3d.org/Coin/>. (Cited on pages 104, 110, 111, 113, 114, 116, 117, 119, 126 and 131.)

- [48] Koppaka, S., Mudigere, D., Narasimhan, S., and Narayanan, B. (2010). Fast histograms using adaptive cuda streams. *CoRR*. (Cited on page 41.)
- [49] Lacroute, P. and Levoy, M. (1994). Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 451–458, New York, NY, USA. ACM. (Cited on pages 14 and 15.)
- [50] Levoy, M. (1988). Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37. (Cited on pages 11 and 21.)
- [51] Lindeberg, T. (1990). Scale-space for discrete signals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3):234–254. (Cited on page 29.)
- [52] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169. (Cited on page 16.)
- [53] Lundström, C. (2007). *Efficient Medical Volume Visualization : An Approach Based on Domain Knowledge*. Phd thesis, Linköping University. (Cited on pages 16 and 21.)
- [54] Lundström, C., Ljung, P., and Ynnerman, A. (2006a). Local histograms for design of transfer functions in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1570–1579. (Cited on pages 19, 37, 39, 40, 41, 42, 43, 57, 58, 59, 64, 66 and 147.)
- [55] Lundström, C., Ynnerman, A., Ljung, P., Persson, A., and Knutsson, H. (2006b). The alpha-histogram: Using spatial coherence to enhance histograms and transfer function design. In *Euro-Vis'06: Proceedings of Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 227–234, Lisbon, Portugal. Eurographics Association. (Cited on pages 19, 37, 38, 39, 57, 59, 60, 61 and 62.)
- [56] Marin, T., Wernick, M. N., Yang, Y., and Brankov, J. G. (2010). Motion-compensated reconstruction of gated cardiac spect images using a deformable mesh model. In *ISBI'10: Proceedings of the 2010 IEEE international conference on Biomedical imaging: from nano to Macro*, pages 520–523, Piscataway, NJ, USA. IEEE Press. (Cited on page 134.)
- [57] Marks, J., Andalman, B., Beardsley, P. A., Freeman, W., Gibson, S., Hodgins, J., Kang, T., Mirtich, B., Pfister, H., Ruml, W., Ryall, K., Seims, J., and Shieber, S. (1997). Design galleries: A general approach to setting parameters for computer graphics and animation. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 389–400, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. (Cited on pages 19, 45, 46, 50 and 57.)
- [58] Mayo Foundation (2010). Mayo analyze file format. <http://eeg.sourceforge.net/ANALYZE75.pdf>. (Cited on pages 104 and 105.)
- [59] Meissner, M. (2000). *Occlusion Culling and Hardware Accelerated Volume Rendering*. Phd thesis, Eberhard-Karls-Universität. (Cited on page 14.)
- [60] Mueller, D. C. (2008). *Direct Volume Illustration for Cardiac Applications*. Phd thesis, Queensland University of Technology. (Cited on pages 7 and 21.)
- [61] Mueller, K. and Crawfis, R. (1998). Eliminating popping artifacts in sheet buffer-based splatting. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 239–245, Los Alamitos, CA, USA. IEEE Computer Society Press. (Cited on page 14.)
- [62] Mueller, K., Möller, T., and Crawfis, R. (1999). Splatting without the blur. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 363–370, Los Alamitos, CA, USA. IEEE Computer Society Press. (Cited on page 14.)

- [63] Nelson, T. (2010). Finder views: Using finder views. [http://macs.about.com/od/switchersnewusers/ss/finderviews\\_5.htm](http://macs.about.com/od/switchersnewusers/ss/finderviews_5.htm). Last visited: 27.08.2010. (Cited on page 53.)
- [64] Nokia Corporation (2010). Qt: What is qt? <http://qt.nokia.com/>. (Cited on page 104.)
- [65] Nokia Corporation (2011). Qt 4.6 documentation. <http://doc.qt.nokia.com/4.6/>. (Cited on pages 104, 108, 110, 114, 120, 122, 123, 124 and 127.)
- [66] NVIDIA Corporation (2010). Cuda: What is cuda? [http://www.nvidia.com/object/what\\_is\\_cuda\\_new.html](http://www.nvidia.com/object/what_is_cuda_new.html). (Cited on page 104.)
- [67] Ogawa, S., Lee, T.-M., Nayak, A. S., and Glynn, P. (1990). Oxygenation-sensitive contrast in magnetic resonance image of rodent brain at high magnetic fields. *Magnetic Resonance in Medicine*, 14(1):68–78. (Cited on page 6.)
- [68] OsiriX Foundation (2011). Pubimage data sets. <http://pubimage.hcuge.ch:8080/>. (Cited on page 140.)
- [69] Pinto, F. d. M. and Freitas, C. M. D. S. (2007). Design of multi-dimensional transfer functions using dimensional reduction. In *EuroVis'07: Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 131–138, Norrköping, Sweden. Eurographics Association. (Cited on pages 19 and 55.)
- [70] Podlozhnyuk, V. (2007). Histogram calculation in cuda. Technical report, NVIDIA. (Cited on pages 41 and 43.)
- [71] Rathmann, U. and Wilgen, J. (2010a). Qwt - qt widgets for technical applications. <http://qwt.sourceforge.net/index.html>. (Cited on page 104.)
- [72] Rathmann, U. and Wilgen, J. (2010b). Qwt 6.0.0 documentation. <http://qwt.sourceforge.net/index.html>. (Cited on page 104.)
- [73] Robertson, G., Czerwinski, M., Larson, K., Robbins, D. C., Thiel, D., and Van Dantzich, M. (1998). Data mountain: Using spatial memory for document management. (Cited on pages 2, 51, 52, 57, 73 and 80.)
- [74] Roettger, S., Bauer, M., and Stamminger, M. (2005). Spatialized transfer functions. In *EuroVis'05: Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 271–278, Leeds, United Kingdom. Eurographics Association. (Cited on pages 29 and 43.)
- [75] Salama, C. R., Keller, M., and Kohlmann, P. (2006). High-level user interfaces for transfer function design with semantics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1021–1028. (Cited on page 55.)
- [76] Scheidegger, C. E., Schreiner, J. M., Duffy, B., Carr, H., and Silva, C. T. (2008). Revisiting histograms and isosurface statistics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1659–1666. (Cited on page 43.)
- [77] Sedgewick, R. (1992). *Algorithmen in C*. Addison-Wesley, 6th edition. (Cited on page 70.)
- [78] Shahidi, R. (1996). Surface rendering versus volume rendering in medical imaging: techniques and applications (panel). In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 439–440, Los Alamitos, CA, USA. IEEE Computer Society Press. (Cited on page 16.)
- [79] Shams, R. and Kennedy, R. A. (2007). Efficient histogram algorithms for NVIDIA CUDA compatible devices. In *ICSPCS'07: Proceedings of the International Conference on Signal Processing and Communications Systems*, pages 418–422, Gold Coast, Australia. (Cited on pages 43 and 107.)

- [80] Shen, H.-W. (2010). [www.cse.ohio-state.edu/hwshen/788/sp01/splat.ppt](http://www.cse.ohio-state.edu/hwshen/788/sp01/splat.ppt). Last visited: 04.10.2010. (Cited on page 13.)
- [81] Sonka, M., Hlavac, V., and Boyle, R. (1998). *Image Processing, Analysis, and Machine Vision*. Brooks/Cole Publishing Company, 2nd edition. (Cited on pages 20 and 21.)
- [82] Sunden, E. (2010). Real-time dvr illumination methods for ultrasound data. Master thesis, Linköping University. (Cited on page 5.)
- [83] Sweeney, J. and Mueller, K. (2002). Shear-warp deluxe: the shear-warp algorithm revisited. In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 95–ff, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association. (Cited on page 15.)
- [84] Takanashi, I., Lum, E. B., Ma, K.-L., and Muraki, S. (2002). Ispace: Interactive volume data classification techniques using independent component analysis. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, pages 366–ff, Washington, DC, USA. IEEE Computer Society. (Cited on page 55.)
- [85] Tavanti, M. and Lind, M. (2001). 2d vs 3d, implications on spatial memory. In *INFOVIS '01: Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, pages 139–ff, Washington, DC, USA. IEEE Computer Society. (Cited on page 52.)
- [86] Visualization Sciences Group (2010). Open inventor: Gnomon. <http://www.mc3dviz.com/openinventor-forum/showthread.php?t=25>. (Cited on pages 76 and 114.)
- [87] Šereda, P., Vilanova, A., and Gerritsen, F. A. (2006a). Mirrored lh histograms for the visualization of material boundaries. *Vision Modeling and Visualization (VMV)*, pages 237–244. (Cited on pages 27 and 28.)
- [88] Šereda, P., Vilanova Bartroli, A., Serlie, I. W. O., and Gerritsen, F. A. (2006b). Visualization of boundaries in volumetric data sets using lh histograms. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):208–218. (Cited on pages 24, 25, 26 and 27.)
- [89] Welch, B. L. (1947). The generalization of students problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35. (Cited on page 35.)
- [90] Wernecke, J. (1993). *The Inventor Mentor: Programming Object-Oriented 3d Graphics with Open Inventor, Release 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. (Cited on pages 110, 111, 115, 116 and 117.)
- [91] Wesarg, S. and Kirschner, M. (2009a). 3d visualization of medical image data employing 2d histograms. In *VIZ '09: Proceedings of the 2009 Second International Conference in Visualisation*, pages 153–158. (Cited on pages 30 and 43.)
- [92] Wesarg, S. and Kirschner, M. (2009b). Gradient magnitude vs. feature size: Comparing 2d histograms for transfer function specification. In *CGI '09: Proceedings of the 2009 Computer Graphics International Conference*, pages 115–119, New York, NY, USA. ACM. (Cited on pages 29 and 31.)
- [93] Wesarg, S. and Kirschner, M. (2009c). Structure size enhanced histogram. In *Proceedings of Bildverarbeitung für die Medizin*, Informatik Aktuell, pages 16–20. Springer. (Cited on page 29.)
- [94] Westover, L. (1989). Interactive volume rendering. In *VVS '89: Proceedings of the 1989 Chapel Hill workshop on Volume visualization*, pages 9–16, New York, NY, USA. ACM. (Cited on page 12.)
- [95] Westover, L. (1990). Footprint evaluation for volume rendering. *SIGGRAPH Comput. Graph.*, 24(4):367–376. (Cited on pages 11, 12 and 14.)

- [96] Wikimedia Foundation (2010a). Data set grid types. [http://en.wikipedia.org/wiki/Regular\\_grid](http://en.wikipedia.org/wiki/Regular_grid). (Cited on page 4.)
- [97] Wikimedia Foundation (2010b). Data set grid types. [http://en.wikipedia.org/wiki/Unstructured\\_grid](http://en.wikipedia.org/wiki/Unstructured_grid). (Cited on page 4.)
- [98] Wilhelms, J. and Van Gelder, A. (1990). Octrees for faster isosurface generation. *SIGGRAPH Comput. Graph.*, 24(5):57–62. (Cited on page 15.)
- [99] Witkin, A. P. (1983). Scale-space filtering. In *IJCAI'83: Proceedings of the 8th International Joint Conference on Artificial Intelligence*, volume 2, pages 1019–1022, Karlsruhe. (Cited on page 29.)
- [100] Wu, Y. and Qu, H. (2007). Interactive transfer function design based on editing direct volume rendered images. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):1027–1040. (Cited on pages 19, 46 and 47.)