FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Transfer Function Widget for Tri-Modal Volume Exploration

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Lucas da Cunha Melo
Matrikelnummer 01429462

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Univ.-Doz. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dipl.-Ing. (FH) Dr. Christoph Heinzl
Dipl.-Ing. Johannes Weissenböck, BSc

Wien, 6. November 2018

Lucas da Cunha Melo

Eduard Gröller

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Transfer Function Widget for Tri-Modal Volume Exploration

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Lucas da Cunha Melo
Registration Number 01429462

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Univ.-Doz. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Dipl.-Ing. (FH) Dr. Christoph Heinzl
Dipl.-Ing. Johannes Weissenböck, BSc

Vienna, 6th November, 2018

Lucas da Cunha Melo   Eduard Gröller

Erklärung zur Verfassung der Arbeit

Lucas da Cunha Melo
Guntherstraße 13/20
1150 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 6. November 2018

Lucas da Cunha Melo

Acknowledgements

To the Vienna University of Technology for providing me with a good academic environment. To the supervisors of this work for helping me give my best without losing motivation. Christoph Heinzl for the guidance and lessons throughout the development of the work presented here. Johannes Weissenböck for all the advice, selflessness and availability to hear and answer my questions. To all friends whose collaboration made the past years a much smoother and more exciting experience. Parnia Ardakani for her love and support, no matter how tough things got. À minha avó Maria José das Neves Welkovic e, principalmente, aos meus pais Carlos Henrique da Cunha Melo e Welika Welkovic da Cunha Melo pelo constante amor, apoio e educação ao longo da minha vida, sem os quais esta experiência sequer teria sido possível.

Kurzfassung

Volumetrische Visualisierung ist ein unverzichtbares Werkzeug in medizinischen Studien und Materialwissenschaften, da sie die Analyse und Untersuchung der inneren Struktur von Objekten ermöglicht. Wegen der Einführung neuer Geräten und Techniken werden die Datensätze zunehmend komplexer, sodass es in der Materialanalyse immer häufiger wird, mit drei Modalitäten gleichzeitig umzugehen. Beispielsweise kann ein einzelner industrieller Computertomographiescan eines Objekts Absorptionskontrast-, Differenzphasenkontrast- und Dunkelfeldkontrastbilder (Modalitäten) ausgeben. Moderne Software-Lösungen kommen jedoch mit diesen Fortschritten ins Hintertreffen, da immer noch keine Applikationen für die Verwendung dieser neuen Anwendungsfälle zur Verfügung steht. Ziel dieser Bachelorarbeit ist es, die Machbarkeit einer Benutzeroberfläche für eine Transferfunktion für tri-modale Datensätze zu testen, wobei der Fokus auf der Benutzbarkeit liegt und die Anforderungen der Materialanalyse berücksichtigt werden. Der Entwicklungsprozess wird von den ersten Konzepten bis hin zu einem Software-Prototyp beschrieben und die resultierende Softwarelösung vorgestellt. Diese wurde als Module für open_iA entwickelt, eine Anwendung für visuelle Analyse und volumetrische Verarbeitung. Das Werkzeug verwendet für jede Modalität die standardmäßige 1D-Farb-Opazitäts-Transferfunktion zusammen mit einer auf baryzentrische Koordinaten basierenden Methode, um die drei Modalitäten miteinander zu vermischen. Für die Modalitäten eines industriellen Computertomographie-Datensatzes (Absorptionskontrast, differentieller Phasenkontrast und Dunkelfeldkontrast) kann das Werkzeug die Volumenexploration erfolgreich beschleunigen und erleichtern.

Abstract

Volumetric visualization is an essential tool in medical studies and materials science, as it enables the analysis and exploration of the inner structure of objects. With the introduction of new equipment and techniques, the datasets become more complex, making it an increasingly common task in material analysis to handle three modalities at once. For instance, a single industrial computed-tomography scan of an object can output absorption contrast, differential phase contrast and dark-field contrast volumes (modalities). However, modern software is falling behind with these advancements, as they still do not provide dedicated tools for this use. This thesis aims to test the feasibility of a transfer function user interface for tri-modal datasets, with a focus on usability, taking into consideration the needs of material analysis experts. The development process (from the initial concepts to a software prototype) is described, and the resulting tool, developed as a module for open_iA (a visual analysis and volumetric processing application), is presented. It uses the standard 1D color-opacity transfer function for each modality together with a barycentric coordinates method to blend the three modalities together. For the modalities of an industrial computed-tomography dataset (absorption contrast, differential phase contrast and dark-field contrast), the tool can successfully accelerate and facilitate volume exploration.

Contents

CHAPTER 1

Introduction

Volumetric visualization is an essential tool in medical studies and materials science. Being able to visualize the inside of structures, dynamically showing what is of interest (e.g., bones in a computed tomography scan) and hiding the rest (e.g., organs and skin) makes this kind of visualization a powerful tool for data exploration. Transfer Functions (TFs) are at the core of this capability, as they allow the user to assign a color and opacity to a voxel based on its value (be it scalar or multivariate).

As new extraction methods and equipment emerge, more profound analysis of volumetric data is made possible. For instance, SkyScan 1294 [Sky] uses X-rays to scan objects and construct three separate volumes (tri-dimensional images): absorption contrast, differential phase contrast and dark-field contrast.

Each of these volumes (modalities) is sensitive to different features of the scanned specimen, providing additional information about it. However, despite the available scanning equipment and techniques, modern visualization software do not provide an efficient way to explore multimodal volumetric datasets. VGSTUDIO MAX [VGS], Amira-Avizo [Ami], MeVisLab [MeV], none contain dedicated tools for multimodal volume exploration.

A scalar (uni-variate) volume contains a single variable per voxel. All three volumes constructed with SkyScan 1294 [Sky], for instance, are uni-variate. Moreover, there exist datasets that contain many variables per voxel (multivariate), such as meteorological datasets, which can have data about, for example, pressure, temperature or wind speed.

There has been scientific effort to design a TF widget for bi-variate [KKH02], tri-variate [ZSH12] and even n-variate [ZK10] [GXY11] datasets. Ljung et al. [LKG+16] use dimensionality (among five other aspects) to classify TFs in a state-of-the-art report, showing that special attention and understanding is required for a proper multidimensional TF design.

However, the dimensionality alone is not sufficient to define a multimodal TF. Methods that work well for multivariate datasets do not necessarily deliver satisfactory results when applied to multimodal datasets. Moreover, although TF methods developed explicitly for multimodal datasets are not a novel idea [BRB$^+$13], to the knowledge of this author, a tri-modal TF design focusing on data exploration is new territory.

This thesis aims to test the feasibility of such a design, presenting an interface that combines three TFs using barycentric coordinates, avoiding the need to continually switch between individual TFs to achieve the desired result. Especial focus is given to the usability of the tool, taking into consideration the needs of material experts currently working with tri-modal datasets. The developed design is implemented as a module in open_iA [ope].

This thesis is structured as follows: in Chapter 2, an overview of the related work is given, as well as a brief discussion of their applicability to this thesis. In Chapter 3, the conceptual phase is gone through, where the widget's design process is discussed. Next, the prototyping phase of the development is described in Chapter 4, where a software prototype is presented. In Chapter 5, the implementation of the selected design in open_iA is described. Chapter 6 presents the results achieved with the developed interface, including a case study based on a common material analysis scenario, and, finally, in Chapter 7, a discussion is provided on the findings, the conclusions are drawn, and the planned future work is described.

CHAPTER 2

State of the art

An essential aspect to consider when designing a multimodal TF is dimensionality, as the addition of modalities inherently results in the addition of new dimensions. Ljung et al. [LKG$^+$16] provide a state-of-the-art report on the research surrounding TFs, covering the definition and basic concepts, as well as more specific aspects, classified in six categories (amongst them, dimensionality). The report also stresses that a number of dimensions higher than two poses challenges concerning the user interface, assuming that the display medium is limited to two dimensions (e.g., a computer monitor).

For that reason, whenever faced with multivariate datasets, researchers have attempted to present them in lower dimensions, aiming to provide increased usability of the TF editor. The standard color-opacity widget (containing a histogram of attribute data on the background for context) enables precise editing of a 1D TF; however, it already requires two degrees of freedom (and, consequently, two dimensions) to achieve that.

One attempt of dimension reduction for an user interface is presented by Kniss et al. [KKH02], where the gradient magnitude of each scalar point is used as a second property, assigning it to the vertical axis of the TF editor while maintaining the data value on the horizontal axis, shown in Figure 2.1. The gradient magnitude is indicated by f', as it is the first derivative of the scalar function f. Materials A, B, and C present in the volume (circulated) have low gradient magnitude values, as they are relatively homogeneous. Boundaries D, E and F (marked with arches) between materials have higher gradient magnitude values.

A drawback of this method compared to the classic 1D TF is the lack of precision when manipulating color and opacity. Whereas the color is set individually per control point in the 1D TF and the control point's height defines the opacity, the gradient magnitude 2D TF [KKH02] requires regions (boxes and pyramids) to be defined. The user assigns the color per region and the opacity is set automatically based on the defined shape.

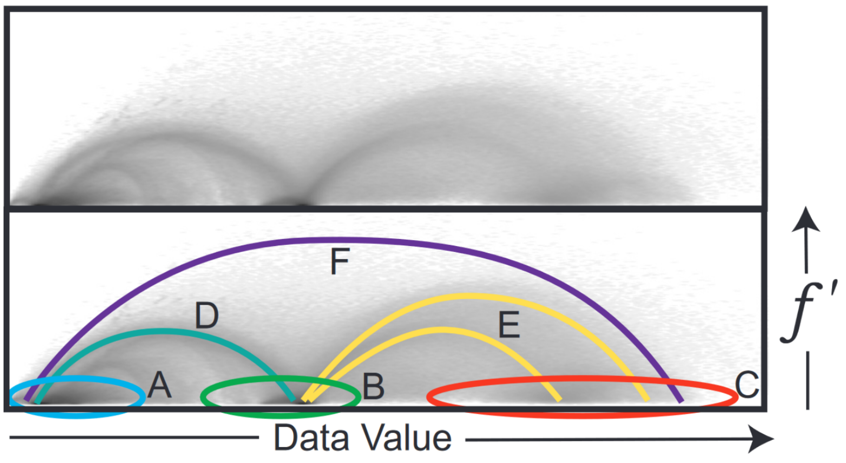


Figure 2.1: By using the gradient magnitufe f', homogeneous materials (circles A, B, C) can be separated from transitional regions (arches D, E, F). Reprinted from [KKH02].

Another concept presented by Ljung et al. [LKG+16] is that of a separable or non-separable TF. Non-separable 2D TFs, such as the one presented by Kniss et al. [KKH02], are better at classifying features compared to separable 2D TFs. They also require the storage or representation of a full 2D function, meaning that keeping track of a 2D array or texture is usually required [LKG+16], whereas separable 2D TFs are defined by a combination of two separate 1D TFs.

Stretching the gradient magnitude TF concept to cover even higher dimensions, Zhou et al. [ZSH12] combine a 2D TF (e.g., scalar $\times$ gradient magnitude) with a set of 1D TFs to enable the definition of a separable 3D TF, capable of making use of three properties. This separation of the 3D TF admits a loss of classification precision in exchange for interactivity, as the separated 1D and 2D TFs can be intuitively displayed on a screen (a 2D display medium), in contrast to the 3D TF. If more than three attributes are available, a combination is selected prioritizing a high correlation between attributes and the information each attribute contains. Figure 2.2 shows the presented tool classifying spheres in a synthetic dataset. The instance shown in Figure 2.2a uses only the 2D TF, while the one in Figure 2.2b combines it with two 1D TFs, resulting in a more precise classification.

The interaction is achieved by allowing the user to define regions within the 2D TF with shapes (ellipses, rectangles) and a lasso tool. Essentially, this results in a selection of voxels by their primary and secondary attributes (those represented in the 2D TF). For each defined region, a 1D TF is provided to enabled a further selection of voxels by their tertiary attribute.

To enable the interaction with TFs of even higher dimensions, researchers have often resorted to the parallel coordinates plot (PCP) [ZK10] [GXY11]. PCP is an attractive method to represent multivariate datasets and their relationships, as it can fit a (theoret-

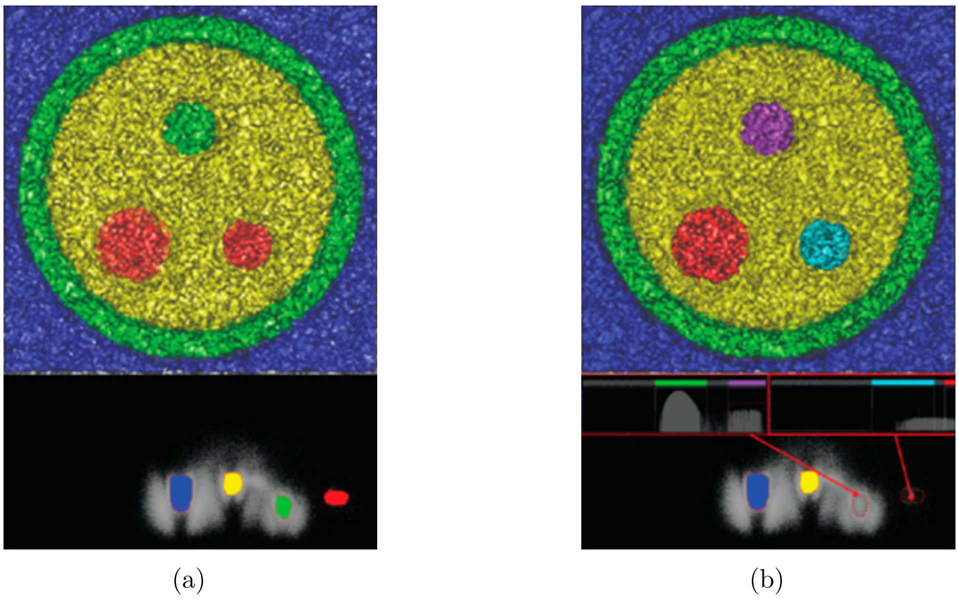


Figure 2.2: Features in the synthetic dataset are selected using the 2D and 1D TFs. Only the 2D TF is used to select features in (a), resulting in the two bottom spheres belonging to the same class, as well as the top sphere and the shell. Further classifying the defined regions with an 1D TF allows all regions to be uniquely classified, as shown in (b). Reprinted from [ZSH12].

ically) unlimited number of dimensions in a 2D representation (practically limited by the size of the display). However, a disadvantage of this method is that not all links between the axes are shown, but only those between adjacent axes. Moreover, this loss of context introduces the necessity to sort the axes in a meaningful order, which also involves the definition of "meaningfulness" itself.

Zhao et al. [ZK10] tackle these issues by developing a global optimization method to find the most meaningful order to display the parallel axes, defined based on the number of polyline aggregations in the PCP. As shown in Figure 2.3, the PCP defined by the user can highlight internal regions of interest. Additionally, a dimension reduction technique was employed as an attempt to represent the relationships lost in the PCP representation. The user is allowed control over two parameters of the dimension reduction technique (number of neighbors and target embedding space dimension) and the interaction itself is achieved by assigning a color and opacity to the classified regions, as seen in Figure 2.4, where a high opacity is assigned to the dark and light blue regions.

Aiming to further simplify the user interaction of a PCP interface, Guo et al. [GXY11] present a widget that uses parallel coordinates, Multi-Dimensional Scaling (MDS) and direct volume view interaction to manipulate the TF. The multiple MDS plots, embedded in the PCP, have adjustable distance metrics, displayed below each MDS plot. , and both of them provide data clustering and data distribution information. Lasso and magic

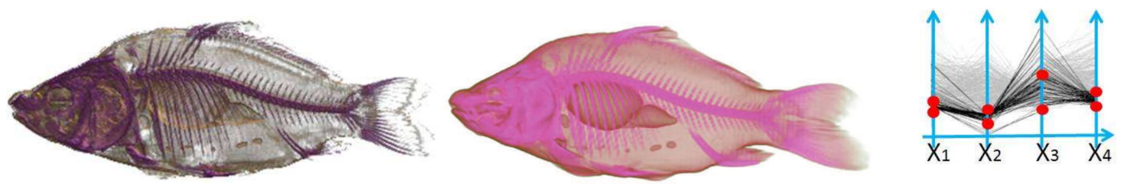


Figure 2.3: Rendering results generated by a gradient magnitude 2D TF (left) and by the presented parallel coordinates method (center). The TF design uses four parameters (right).

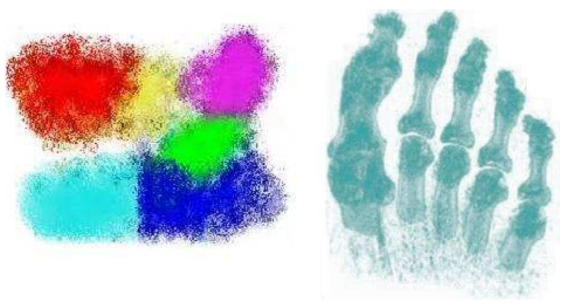
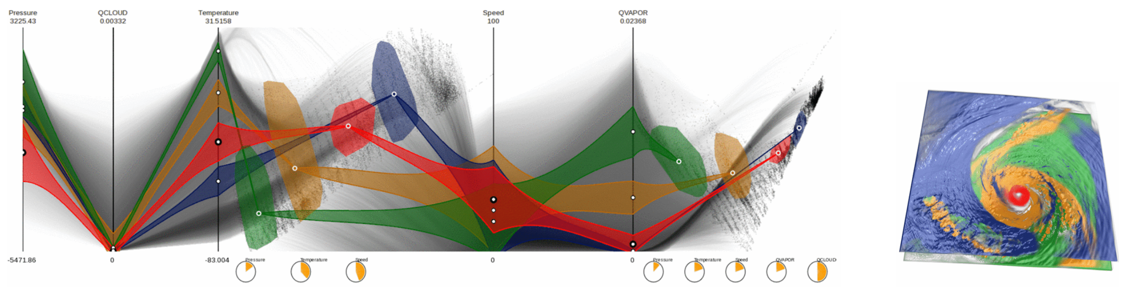


Figure 2.4: Rendering result (right) generated by assigning a high opacity to the classes shown as dark and light blue on the embedded 2D space (left).



(a) (b)

Figure 2.5: The PCP with embedded MDS plots are shown in (a). The regions on the MDS plots and colored ribbons linking the central points (in the regions and in the parallel axes) illustrate the TF. The circles below the MDS plots display the weights of each displayed attribute. The rendering result is shown in (b). Reprinted from [GXY11].

wand tools can be used to select features, providing the TF interaction for the MDS plots. Each parallel axis has a central point that can be manipulated by changing its size (height) and vertical position, represented by the colored ribbons (in an illustrative manner). The result is shown in Figure 2.5. Direct volume view interaction is also possible via sketching on the desired region.

All methods mentioned present higher dimension solutions for the display and interaction of TFs. However, they all focus on multivariate datasets, the majority of them ([KKH02], [ZSH12] and [ZK10]) extracting the variables from the scalar volume (e.g., gradient magnitude, variance, skewness). Methods that deliver satisfactory results for multivariate volumes are not necessarily useful to explore multimodal datasets, and significantly less scientific work has been done in this direction.

Bramon et al. [BRB$^+$13] present a fully automatic scheme to visualize bi-modal volumetric data that defines the colors and opacities of a TF, given a predefined 1D TF for each modality. Using information theory strategies, information maps are created and used to fuse two input datasets (modalities), selecting the most informative source dataset for each voxel. The fused color is then determined using the information maps and the initial 1D TFs, while the opacity is determined using an optimization procedure, followed by the binning of the intensities and the fusion of the gradients.

Bramon et al. [BRB$^+$13] also argue that transferring the primary decisions when defining a TF to the user "may introduce errors and also makes the reproducibility of the method difficult", considering the complexity of such a task, which involves understanding the structure of the input, their relationship, what has to be visualized, and how. The proposed framework can function entirely without the user's intervention, allowing it at only two points: changing the (initially automatically) chosen reference image for the color fusion, and proposing the target distribution for the opacity optimization to decide which features to enhance. The result can be seen in Figure 2.6.

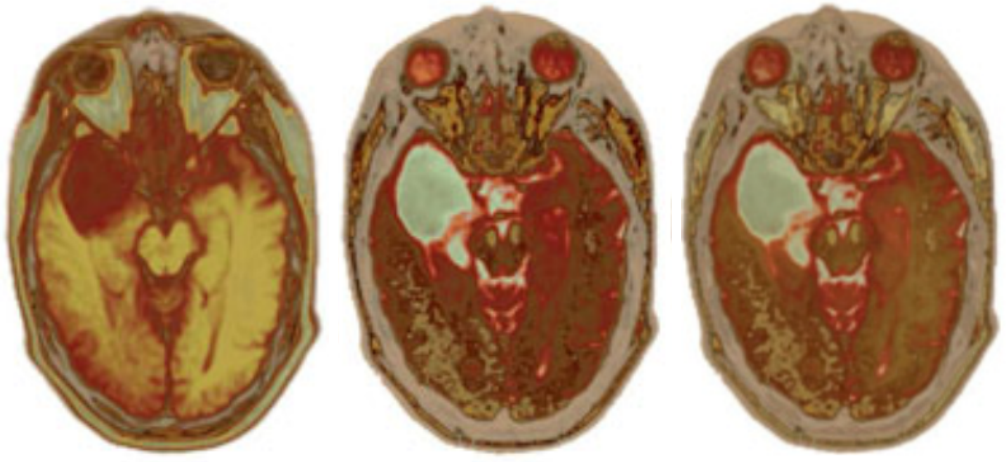


Figure 2.6: Bi-modal visualization (right) of the MR-T1 (left) and MR-T2 (center) modalities. Reprinted from [BRB$^+$13].

CHAPTER 3

Concepts

Two main concepts were considered to create a tri-modal TF and its user interface. The first concept, inspired by parallel coordinates TF approaches [ZK10] [GXY11], is based on a star chart, while the second one users barycentric coordinates together with a triangle component in the user interface. Both main concepts are presented in the following sections, as well as three more concrete ideas that sprouted from the second concept.

3.1 Star chart

A parallel coordinates plot is an attractive method for the representation of multidimensional data due to its easy scalability, accommodating multiple dimensions in a 2D representation. This advantage, however, comes with considerable drawbacks. Each axis of a parallel coordinates plot can only be visually linked to a maximum of two other axes. Additionally, the decision must be made of what axes relate to each other or whether they relate at all, and in what order to display them. That results in either limiting the meaningfulness of these correlations or transferring that responsibility to the user, increasing the domain expertise required to use such a tool.

For those reasons, arranging the axes in polar coordinates (as opposed to side by side, ultimately becoming a star chart) is a considerable alternative. With the condition that the number of axes is kept at three, all pairs of axes remain connected. However, because polar coordinates are used, lines linking low values between two axes become shorter than lines linking high values, risking the introduction of bias and compromising perception. Furthermore, this representation only enables the understanding of pairwise connections between the modalities, requiring the user to keep track of three simultaneous relationships (for the modalities A, B and C, the relationships AB, AC and BC). Nevertheless, a star chart approach was considered to develop a tri-modal TF widget.

In the conceived idea, each scalar triple a, b, c with corresponding coordinates in modalities A, B and C of the same size becomes a triangle in the star chart space by plotting the scalars in the axes and joining the plotted points. In other words, each voxel triple can be represented by a triangle. The user interacts with the star chart by defining triangles and assigning a color to them. An example sketch is shown in Figure 3.1, where each axis represents the range of the modalities A, B and C, the colored triangles are set by the user, and the black perimeter represents a data point. Such a method would allow geometric properties (such as the area of the triangles) to be used in the TF definition.

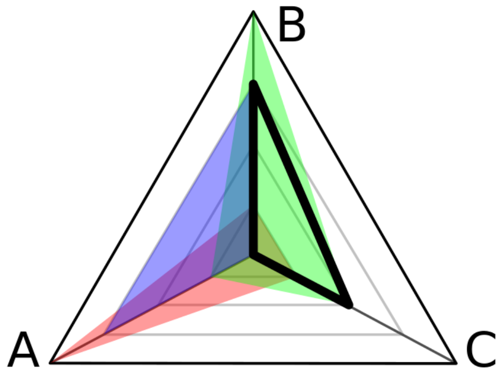


Figure 3.1: A concept of a star chart widget for tri-modal volume exploration. The axes represent the ranges of modalities A, B and C. The gray lines connect the axes in the positions 25%, 50% and 75%. The thick outlined triangle in the middle represents a scalar triple a, b, c with coordinate X in volume space and with $a = 0\% = A(X)$, $b = 75\% = B(X)$ and $c = 50\% = C(X)$. The colored triangles are defined by the user.

However, it is difficult to determine not only whether such properties create meaningful results to render a volume, but also whether this concept can produce an intuitive tool in volume exploration for domain experts. Due to these challenges, this concept was not further developed.

3.2 Barycentric coordinates

The barycentric coordinates concept is analogous to the standard color-opacity TF, where the color is defined through control points (by assigning them colors), and the control point's height determines the opacity. This concept consists of an equilateral triangle with each intensity histogram (one for each modality) placed on an edge, as shown in Figure 3.2. Inside the triangle, the user can manipulate the control points. A point's Barycentric coordinates α, β and γ (calculated from its Cartesian coordinates x and y) control each modality's influence on the rendered volume, while the point's height (Cartesian coordinate z) controls the opacity. Figure 3.2a shows a 3D representation instance of this concept. Figure 3.2b shows a top-down view of the same instance.

Four concrete ideas branched from this concept. A brief description and discussion of

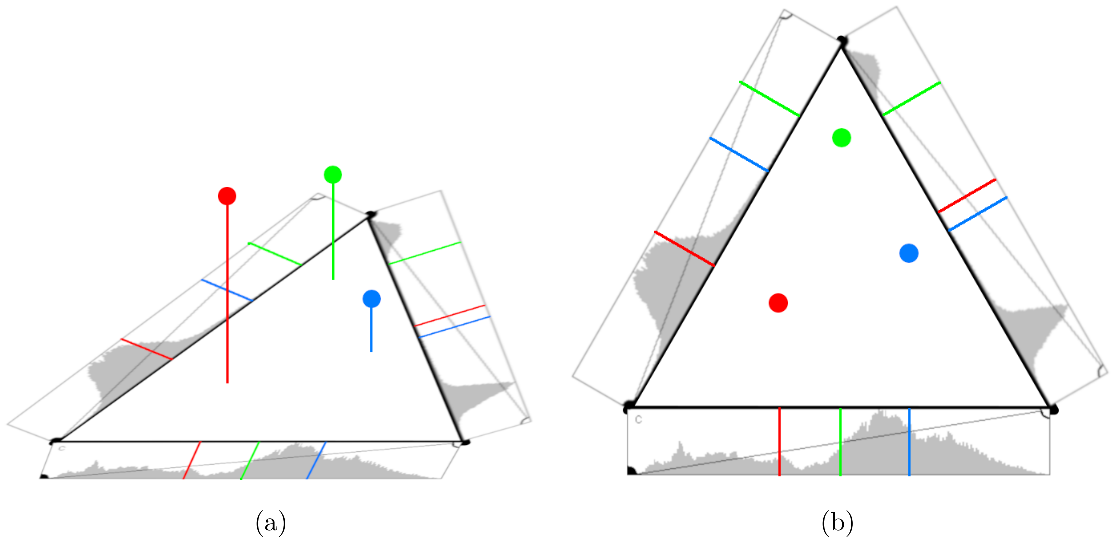


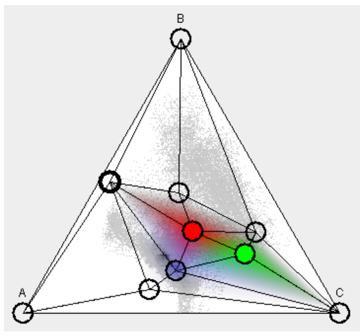
Figure 3.2: An initial concept for a tri-modal TF widget based on barycentric coordinates. (a) Control points can be added on the triangle and assigned a color, while the opacity is defined based on a point's height. (b) The same instance is shown in a top-down view.

each of them are provided in the paragraphs below, together with a comparison of their pros and cons, shown in Table 3.1.

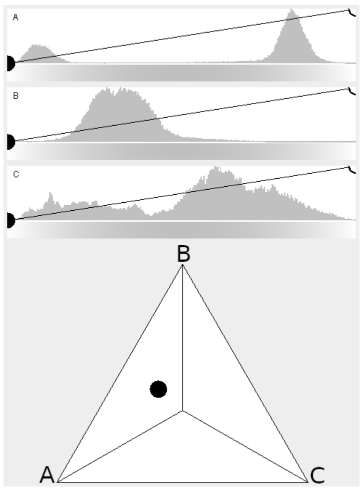
Barycentric Mapping: Analogous to the color-opacity 1D TF, which defines a 1D texture mapping scalar values to a color and opacity, the Barycentric Mapping widget uses control points to define a 2D texture mapping triples of scalar values (one from each modality) to a color and an opacity. This mapping can be achieved by globally normalizing each modality (volume) so that all values are in the range $[0, 1]$. Next, all voxel triples (α, β, γ) across modalities with same coordinates are again normalized. This double normalization results in (α, β, γ) satisfying $\alpha \geq 0$, $\beta \geq 0$, $\gamma \geq 0$ and $\alpha + \beta + \gamma = 1$, allowing each voxel triple to be mapped to a position inside a 2D triangle, where the TF texture is defined. However, since all three modalities influence the new voxel values, it is not guaranteed that this approach will lead to the results expected by the user. This mapping also allows frequency points to be plotted on the triangle, providing context information to help the user set meaningful control points. The TF texture can be edited with image editing tools, such as brushing. However, to keep the similarity with the 1D TF, secondary triangles are triangulated using the control points, and the colors of pixels in between are interpolated, as shown in Figure 3.3a.

Transfer Function Stack: Barycentric coordinates can be used to linearly interpolate between three values, as they always add up to one. Based on this property, the Transfer Function Stack widget allows the user to define the position of a single control point inside a triangle, where the point's barycentric coordinates are used to interpolate the values obtained from three separate TFs (one for each modality). This approach makes possible to use any desired uni-modal TF widget separately for each of the modalities while providing control over how they are mixed during rendering. These features make this approach more intuitive for domain experts, as familiar TFs (such as 1D color-opacity or 2D gradient magnitude [KKH02]) can be used to set the TFs of individual modalities. However, in contrast to the Barycentric Mapping method, no direct context information can be provided, as the triangle does not represent a TF texture, but merely controls a parameter in the underlying separable TF. Figure 3.3b shows this idea using the color-opacity histogram widget for each of the individual modalities.

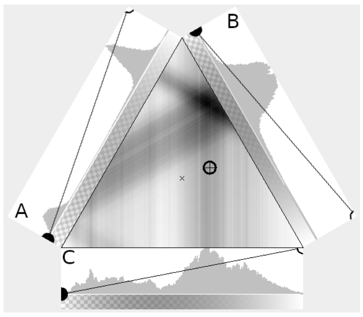
Transfer Function Triangle: Using the same approach of the Transfer Function Stack, the Transfer Function Triangle attempts to better make use of the screen space by laying out the individual TFs on the edges of the triangle. Additionally, this alternative improves the intuitiveness of the tool by letting the control point's weight be defined based on its proximity to the individual TFs. For instance, modality A's weight α increases when the control point is placed near the edge where A's TF widget is placed. This layout is shown in Figure 3.3c, where the positioning of the 1D TF widgets is used to project the histogram frequencies onto the triangle (similar to a simple backprojection) as an attempt to provide context about the data. A drawback of this approach is that, for two of the modalities (A and B in Figure 3.3c), their TF widget must be displayed at an angle, risking a decrease of user accuracy when positioning control points.



(a)



(b)



(c)

Figure 3.3: The three barycentric-coordinates-based concepts: (a) Barycentric Mapping with plotted frequency points and colored control points that control the underlying 2D texture; (b) Transfer Function Stack with a single control point on a triangle, controlling the weights of individual TFs (here, 1D color-opacity TFs); and (c) Transfer Function Triangle with an exemplary context information method (here, a simple backprojection of the histograms)

Concept	Pros	Cons
Barycentric Mapping	Context information (frequency plot) helps directly set control points	Unfamiliar to domain experts; ability to deliver expected results needs verification; imprecise opacity definition (compared to the 1D TF)
TF Stack	Individual TFs can be changed (customizable, familiar to domain experts); control point meaning clear (the closer to a vertex, the larger the respective modality's influence)	No context information that directly helps position the control point; occupies larger screen area in comparison to two other prototypes; distance between triangle edges and their respective modality's TF requires extra user effort
TF Triangle	Control point meaning clear (the closer to an edge, the larger the respective modality's influence); individual TFs' positioning expands context information possibilities	No context information that directly helps position the control point; individual TFs' rotation decreases user accuracy

Table 3.1: Pros and cons of the conceptual tri-modal TF widgets.

CHAPTER 4

Prototypes

The three barycentric concepts presented in Section 3.2 were realized in a software prototype using the Java programming language [Jav] with the bundled Swing GUI Toolkit. The prototype, shown in Figure 4.1, does not perform volume rendering, but, instead, uses the defined TF to render a 2D image from three input 2D slices.

The panel on the right (with the label "Image fusion") is dedicated to setting the inputs required to render the result, while the whole remaining space on the left displays the TF widget. The three grayscale images (labeled "Modality A," "Modality B" and "Modality C") are the input slices, imported into the prototype as PNG images and converted to a normalized grayscale image in the range $[0, 1]$. The "Transfer function" combo box at the top-right changes the TF widget of the left panel to one of the prototypical TF and interface designs. The available options are "Barycentric mapping," "Histogram stack" and "Histogram triangle." The "Color interpolation" combo box sets the color interpolation method used through the whole rendering pipeline. The "Replace all modalities" button allows the user to set new input images dynamically.

4.1 Barycentric mapping

Barycentric coordinates are used in the prototype to represent a triple relationship visually, as they allow three coordinates α, β and γ to be uniquely defined with only two degrees of freedom (since $\alpha + \beta + \gamma = 1$), enabling a 2D visualization of this relationship. The designed technique uses barycentric coordinates to define a 2D non-separable TF that uses a preprocessed image (2D, 3D or of any dimension) to map each element (e.g., pixel or voxel) to a position in a 2D texture. In this section, a prototype implementing this technique for 2D images is presented in detail.

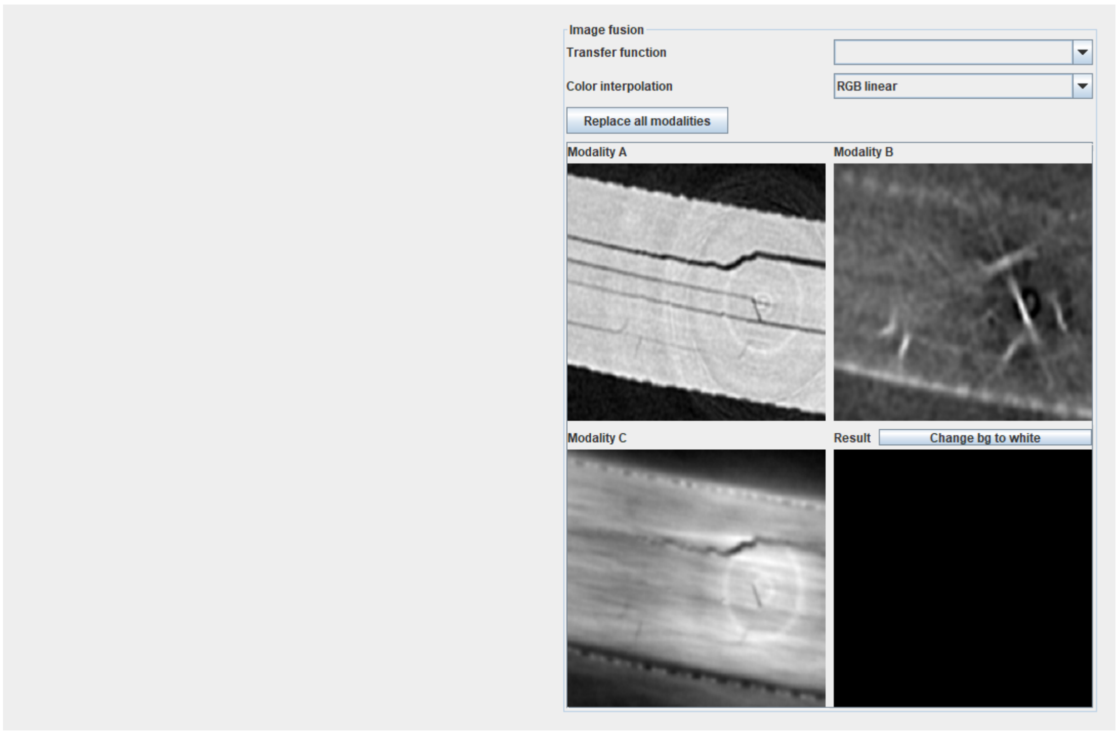


Figure 4.1: The basis components for the prototype in the "Image fusion" panel. The "Transfer function" combo box selects a TF widget, which then appears on the empty area on the left. The "Color interpolation" combo box controls the color interpolation method used during the whole pipeline. The prototype uses the three input slices (modalities A, B, and C) and the selected TF to render the result image. The input images can be changed with the "Replace all modalities" button and the result's background can be set to black or white.

4.1.1 Preprocessing

In the prototype, the three input scalar images A, B and C of equal size s are used to generate an image R of size s with tri-variate data, representing the barycentric coordinates α, β and γ. For each pixel of coordinates x, y in the input images, the resulting barycentric coordinates are defined as the locally-normalized ratio of each image value $a = A[x][y]$, $b = B[x][y]$ and $c = C[x][y]$. This means that the values a, b, c at coordinates (x_1, y_1) in A, B and C will result in the same α, β, γ values as $a \cdot k, b \cdot k, c \cdot k$ at (x_2, y_2).

Note that R can also be bi-variate, as it does not need to contain the γ coordinate since that can be calculated as $1 - \alpha - \beta$. However, not storing γ will require its calculation for every pixel (or voxel) on every render cycle.

Figure 4.2 shows the generated image R, where the α, β and γ components are coded to *red*, *green* and *blue* respectively. If the scalar values of A, B and C are not in the range $[0, 1]$, an initial global normalization is also required. Algorithm 4.1 shows this calculation for tri-dimensional images.

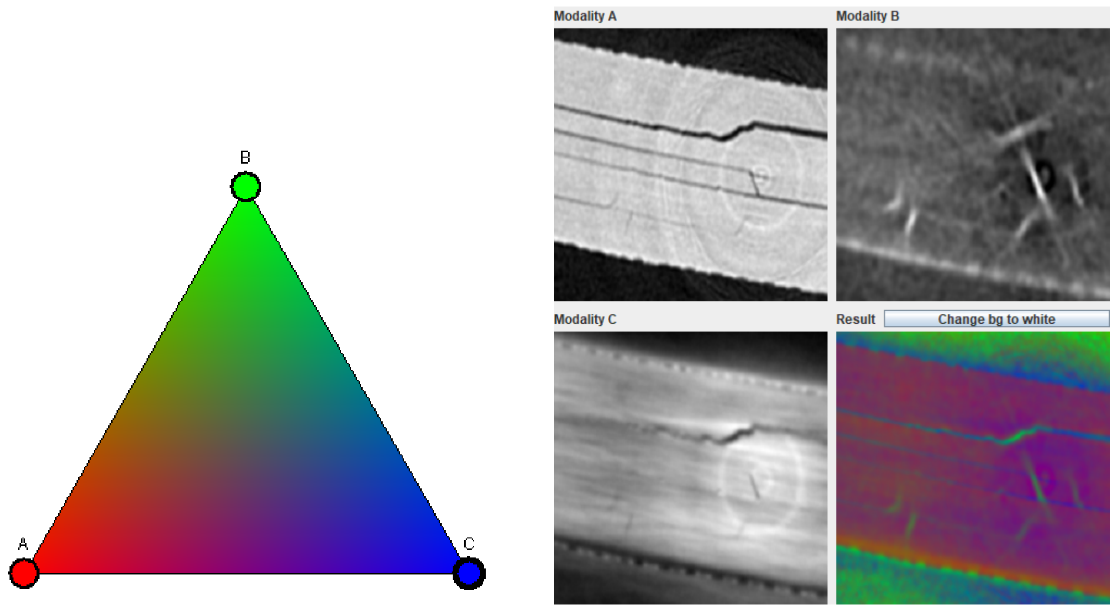


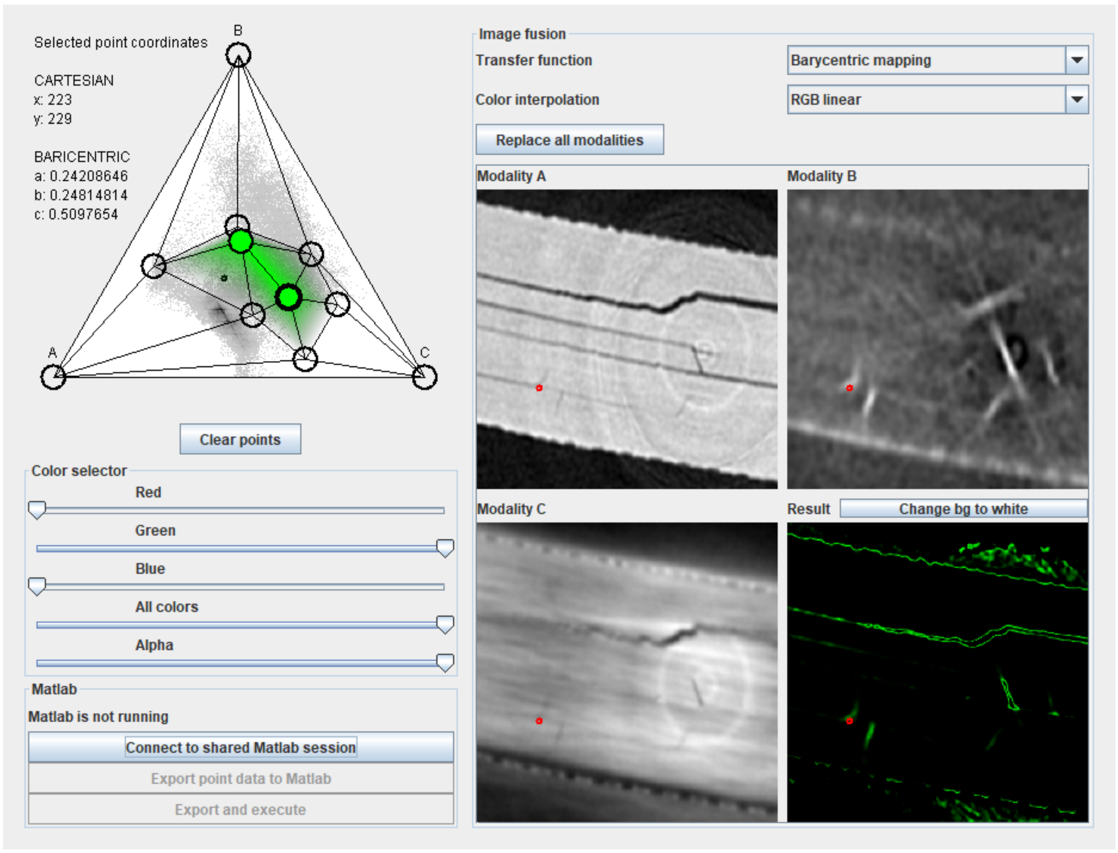
Figure 4.2: The barycentric coordinates image R is calculated from the three Modalities A, B and C. The resulting image ("Result") illustrates R by coding the coordinates α, β and γ to *red*, *green* and *blue* respectively.

4.1.2 Transfer function definition

The Barycentric Mapping TF can be defined by the user using the provided widget shown in Figure 4.3. Control points can be added, moved and removed (except for the vertex points). All control points can have their color changed by using the sliders in the "Color selector" panel (the "All colors" slider controls the "Red," "Green" and "Blue" sliders simultaneously).

The control points are joined using an incremental 2D Delaunay triangulation algorithm [Die] and the formed triangles are rendered onto the 2D TF texture by interpolating the vertices' colors. This method was chosen due to its analogy to the standard color-opacity TF, where controls points are used as well. However, this concept allows the texture to be edited using other methods, such as brushing or defining shapes (e.g., circles or rectangles). Note that control points do not need to be contained within the triangle, as they serve merely as a painting tool.

Additionally, the prototype can export the control points' data to a shared MATLAB [MATa] session. This functionality is made possible by the MATLAB API for Java [MATb] and facilitates the preview and evaluation of a 3D representation of the TF



Selected point coordinates

CARTESIAN
x: 223
y: 229

BARICENTRIC
a: 0.24208646
b: 0.24814814
c: 0.5097654

Transfer function: Barycentric mapping

Color interpolation: RGB linear

Matlab is not running

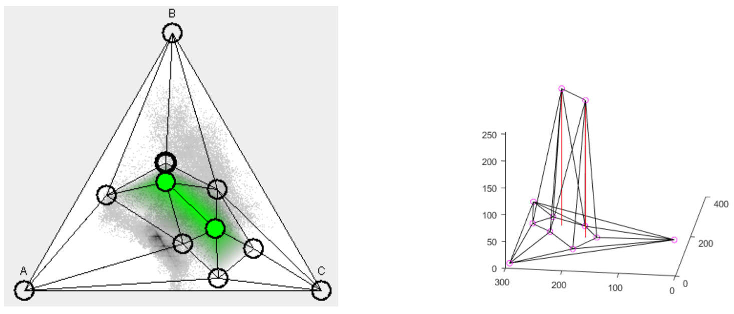
Figure 4.3: The developed prototype in "Barycentric mapping" mode. The control points on the triangle (top-left) define a 2D texture that is used to render the result image (bottom-right).

widget, where the *alpha* values of the control points are plotted in a vertical axis, shown in Figure 4.4.

Figure 4.4a shows the TF defined in the prototype, while Figure 4.4b shows the resulting MATLAB 3D plot in different angles. The magenta circles represent the control points, and the red lines are guides to help tell the point's height relative to the base (at height zero). However, even with the red guides, this representation remains unintuitive and makes it difficult to perceive a control point's position in the 3D space. Additionally, this approach does not easily allow for the texture to be edited using other methods, such as brushing. For those reasons, this representation was not further developed.

4.1.3 Context information

The fact that each element in the generated tri-variate image R can be mapped to a (x, y) position in a triangle (and therefore, in a 2D texture) enables the plotting of data frequency directly on the TF widget, done once in the widget initialization and every time



(a) (b)

Figure 4.4: An example TF defined in the prototype (a) has its control points plotted in 3D (b), where the height of a point represents its opacity.

Algorithm 4.1: Barycentric coordinates image preprocessing

```
Input: Three scalar volumes (modalities) $A$, $B$ and $C$ of size $s$
Output: One tri-variate volume of size $s$ where the variables have the values $\alpha$,
        $\beta$ and $\gamma$ with $\alpha \geq 0$, $\beta \geq 0$, $\gamma \geq 0$ and $\alpha + \beta + \gamma = 1$
1  $R \leftarrow$ tri-variate volume of size $s$;
2  $maxA \leftarrow$ largest value in $A$;
3  $maxB \leftarrow$ largest value in $B$;
4  $maxC \leftarrow$ largest value in $C$;
5  foreach voxel in $R$ with coordinates $x, y, z$ do
       // global normalization
6      $a \leftarrow A[x][y][z] \div maxA$;
7      $b \leftarrow B[x][y][z] \div maxB$;
8      $c \leftarrow C[x][y][z] \div maxC$;
9      $sum \leftarrow a + b + c$;
       // local normalization
10     if $sum = 0$ then
11         $\alpha \leftarrow 1 \div 3$;
12         $\beta \leftarrow 1 \div 3$;
13     else
14         $\alpha \leftarrow a \div sum$;
15         $\beta \leftarrow b \div sum$;
16     end
17     $\gamma \leftarrow 1 - \alpha - \beta$;
18     $R[x][y][z] \leftarrow (\alpha, \beta, \gamma)$;
19 end
20 return $R$;
```

the widget is resized. To achieve this plotting, all elements (α, β, γ) in R are transformed to Cartesian coordinates (x, y) (using the widget's triangle vertices), which are then used to increment a frequency counter for the (x, y) position. Finally, after all voxels are processed, the counters are used to determine the color of the (x, y) pixel. Algorithm 4.2 shows this calculation for a tri-variate image R.

In the prototype, the color is determined as follows: being min the smallest counter value greater than zero and max the largest, the counter value at (x, y) is linearly transformed from the range $[min, max]$ into the range $[48, 255]$, and the resulting value is used as the *red*, *green* and *blue* components of the pixel at (x, y). The target range starts at 48 to more clearly differentiate between the empty pixels (where the frequency is zero) and the non-empty pixels. The resulting plot is shown in Figure 4.3.

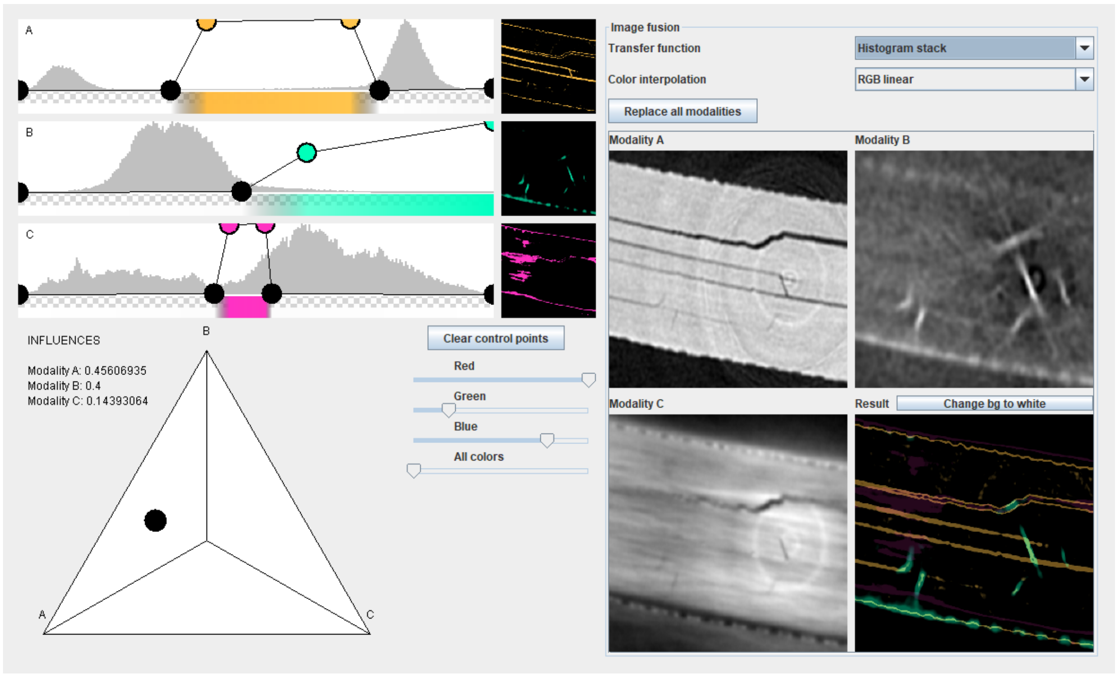
Figure 4.3 also shows the effect of hovering the mouse over the result image. This action highlights, with a red circle, the hovered data point on all images (inputs and output), and, with a black circle, the corresponding point on the TF texture. The inverse interaction is also possible, where, by hovering over a frequency point, the multiple corresponding pixels are highlighted on all images.

Algorithm 4.2: Frequency points plot

```
Input: One tri-variate volume $R$ where each element has the values $\alpha$ , $\beta$ and $\gamma$
       with $\alpha \geq 0$, $\beta \geq 0$, $\gamma \geq 0$ and $\alpha + \beta + \gamma = 1$
Input: One triangle $triangle$ in screen space coordinates
Output: One 2D image with the frequency points plotted
1  $bounds \leftarrow triangle$.getBounds();
   // triangle's top-left corner must be in origin (0,0) to
      match freq array
2  $triangle$.translate($-bounds$.getLeft(), $-bounds$.getTop());
3  $freq \leftarrow$ 2D integer array of dimensions $bounds$.getWidth() $\times$
    $bounds$.getHeight();
4  foreach $voxel$ in $R$ with coordinates $x, y, z$ do
       // transform barycentric coordinates into cartesian
5      $bary \leftarrow R[x][y][z]$;
6      $cart \leftarrow$ baryToCart($bary$, $triangle$);
7      $freq[cart$.x()][$cart$.y()] $\leftarrow freq[cart$.x()][$cart$.y()] $+ 1$;
8  end
9  $min \leftarrow$ smallest value $> 0$ in $freq$;
10 $max \leftarrow$ largest value in $freq$;
11 $I \leftarrow$ empty image of dimensions $bounds$.getWidth() $\times$ $bounds$.getHeight();
12 foreach $pixel$ in $I$ with coordinates $x, y$ do
13     $I[x][y] \leftarrow$ freqToColor($freq[x][y]$, $min$, $max$);
14 end
15 return $I$;
```

4.2 Histogram stack

The Barycentric Mapping TF widget, although providing context information on the texture (directly related to the rendering results), is an unfamiliar method to domain experts. Its efficiency in providing expected and meaningful results for the user's application needs thorough studying. For this reason, a second prototype was implemented exploring the Transfer Function Stack concept. Figure 4.5 shows the first implemented user interface variant that supports this method, called "Histogram stack" in the prototype.



INFLUENCES

Modality A: 0.45606935
Modality B: 0.4
Modality C: 0.14393064

Transfer function: Histogram stack
Color interpolation: RGB linear

Figure 4.5: The developed prototype in "Histogram stack" mode. The single control point in the triangle (bottom-left) sets the weights of the modalities' individual TFs (top-left), used to render the result image (bottom-right) from the three input images (modalities A, B, and C). The individual TFs plot a histogram of the frequency of scalar values as context information and is used in the prototype to render each modality. The colors of the control points can be set using the "Red," "Green," "Blue" and "All colors" sliders, while their opacity is based on the control point's height. Interacting with any histogram, as well as with the triangle's control point, results in an immediate re-rendering of the result image.

In this variant, the TF widgets for the individual modalities are laid out on top of each other and displayed on the top-left. The prototype uses the standard 1D TF showing a histogram (with the frequency of the scalar values in an image), as it is the most common form of TF [LKG$^+$16]. Its control points can be moved, where their height represent the opacity, and their color can be changed, controlling an underlying 1D TF. Any change in the individual TFs triggers an immediate re-rendering of their respective local image and the "Result" image (see figure 4.5). However, other TFs work just as well in theory, the only limiting factor in practice being the available screen space. To the right of each histogram, the renderings of their respective individual modality are shown. The control points' colors can be changed using the four color sliders, in the same way as in the Barycentric Mapping prototype presented in Section 4.1.

On the bottom-left, a triangle with a single control point is displayed. By clicking or dragging anywhere inside of the triangle, the control point's position is set to that location, the barycentric coordinates recalculated, and the result image is immediately re-rendered. The lines in the triangle meet on the barycenter, where $\alpha = \beta = \gamma = \frac{1}{3}$.

The final blended result is shown on the bottom-right. If the triangle's control point is set to one of the vertices (labeled A, B, and C), the final result does not produce the exact same image as the corresponding local result of an individual modality and its TF, as the blending interpolates only the colors and sets the *alpha* value to 255. Choosing to interpolate the *alpha*, while consistent with the in-between renderings, produces images that are more difficult to read, as the selected features become less opaque.

4.3 Histogram triangle

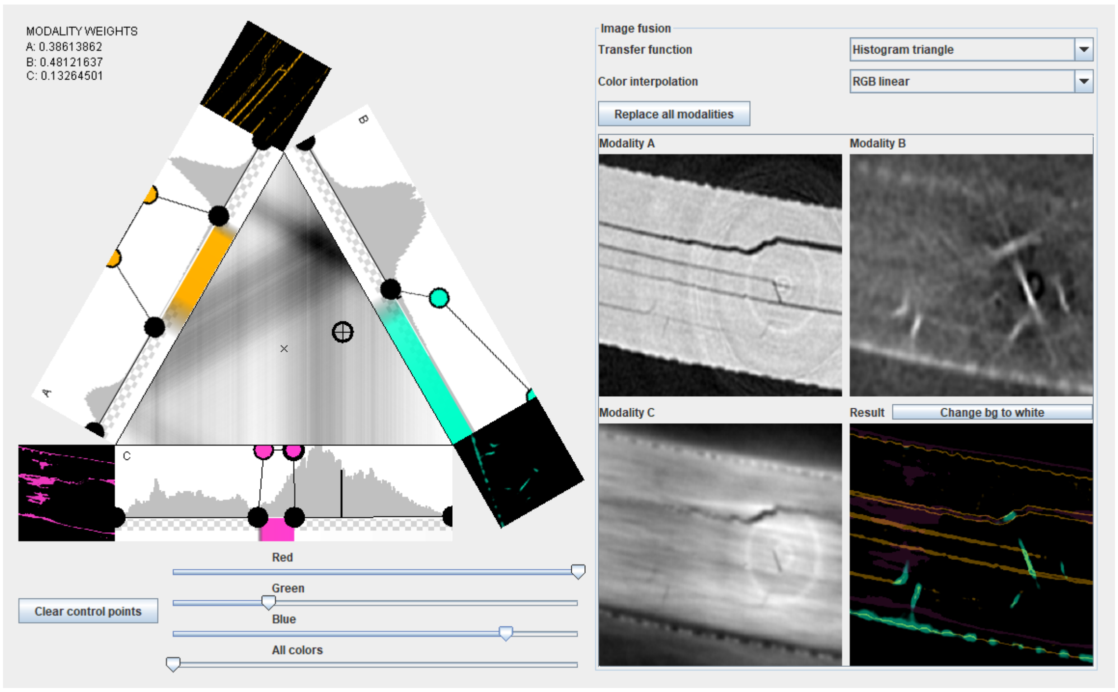
A second layout was implemented for the same rendering method used in the Transfer Function Stack as an attempt to better use the screen space and enable the testing of alternative data context displaying techniques. The Transfer Function Triangle, called "Histogram triangle" in the prototype, is shown in Figure 4.6.

All components function the same as the "Histogram stack" prototype, only differing in the layout and the triangle's orientation (rotated by 60°clockwise). A modality's influence (weight) in the final rendering is defined based on the control point's proximity to the modality local rendering (next to its histogram), unlike the concept discussed in Section 3.2, which defined the influence based on the proximity to the histogram.

Additionally, a conceptual method of showing context information was tested. The idea is to project the histogram values onto the triangle, analogous to a simple backprojection. This technique was, however, rejected, as it neither contributes with information about the data nor it helps with the placement of the control point. Figure 4.6 shows the results.

4.4 Discussion and evaluation of the prototypes

All three implemented prototypes were tested and evaluated using color combinations suggested in ColorBrewer's [CBr] "qualitative" palette. The TFs and rendered results are shown in Figures 4.3, 4.5 and 4.6. The images used are slices extracted from a carbon-fiber-reinforced polymer with impact damage dataset scanned with Skyscan 1294 [Sky], which produces three complementary (perfectly registered) modalities in one scan. To evaluate the efficacy of the prototype, the task of highlighting features of



MODALITY WEIGHTS
A: 0.38613862
B: 0.48121637
C: 0.13264501

Image fusion
Transfer function: Histogram triangle
Color interpolation: RGB linear

Figure 4.6: The developed prototype on "Histogram triangle" mode. The single control point in the triangle (left) sets the weights of the modalities' individual TFs (on the triangle's edges), used to render the result image (bottom-right). The texture in the triangle is a projection of the histogram's values, similar to a simple backprojection; an attempt to show context information.

interest to domain experts (e.g., cracks or pores) was set. These features are emphasized differently in each modality and, therefore, perceived differently as well. The provided prototypes allowed the use and investigation of all the information of the three modalities simultaneously in one visualization.

The "Barycentric mapping" technique, although offering a non-separable TF that allows for direct manipulation, was considered less intuitive than the barycentric blending technique used in the "Histogram stack" and "Histogram triangle" alternatives. Requiring the opacity to be set per control point does not decrease the precision of the tool, confirming the idea that a 3D representation is not necessary. Additionally, the method appeals more at first due to the direct linking between TF and the input/result images, but still requires trial-and-error to achieve the desired results.

The "Histogram stack" and "Histogram triangle" tools demonstrated to enable a quicker highlighting of the desired features. The domain experts also considered it to be a more intuitive tool, as it contains more familiar concepts, such as the standard color-opacity TF. However, the drawback, when comparing this tool with the "Barycentric mapping" alternative, is the lack of context information about the data.

Another rejected design decision regarding the "Histogram triangle" is the rotation of the modalities' TF local rendering (next to each histogram), as it decreases the control point positioning precision. It was also considered unintuitive that the weights of the modalities in the final rendering are set based on the control point's proximity to the triangle vertex when basing them on the proximity to a modality's histogram (triangle's edge) is the expected behavior.

Regarding scalability, none of the methods naturally support it. In order to add a fourth modality, both TFs would need to be represented in three dimensions (with a pyramid instead of a triangle), with the control point also lying in the 3D space. Each added modality further increases the dimensionality by one, requiring a k-simplex to represent a $(k+1)$-modal dataset.

Scaling them down is, however, possible. The "Barycentric mapping" can be represented in two dimensions by a histogram, using a pre-processed bi-variate volume containing the interpolation values α and β for every voxel to render the result (or one value α, since $\alpha + \beta = 1$). The "Histogram stack" and "Histogram triangle" can blend two modalities by having two interpolation values α and β, which can be represented by, for instance, a slider (instead of a control point in a triangle).

CHAPTER 5

Implementation

open_iA is an open-source tool for scientific visualization, analysis, and processing of volumetric datasets, developed and maintained by the Computed Tomography Research Group at the University of Applied Sciences Upper Austria, Campus Wels [CTR]. Intended as a framework for research prototypes and tools, open_iA is built using Qt [Qt] (a cross-platform GUI SDK) the Visualization Toolkit (VTK) [VTK] (an open-source toolkit for 3D visualization) and the Insight Segmentation and Registration Toolkit (ITK) [ITK] (an open-source toolkit for image analysis.

The core of open_iA provides, amongst other widgets, a 3D volume renderer, three 2D slicers for each of the volume's axes, a histogram (used to manipulate the TF of the renderer and the slicers) and a data modalities dialog, where the loaded datasets are listed. Extensions to the core can be developed in the form of modules.

The tri-modal TF widget was implemented in C++ as an open_iA module, making use of the available core widgets to render the modalities with the defined TF. Both "Histogram stack" and "Histogram triangle" variants from the prototype were implemented.

5.1 Functionality

The widget can be initialized regardless of how many modalities (`iAModality`) are loaded. However, if there are less than three modalities available, the inner components are not initialized until it is detected that a third modality is added. If a modality is removed so that there are less than three modalities available, the widget is disabled.

Figure 5.1 shows the enabled widget in "stack" mode on the bottom-right corner, outlined in red. It contains three histograms, three slicers and one triangle control, of classes `iADiagramFctWidget`, `iASlicer` and `iABarycentricTriangleWidget` respectively.

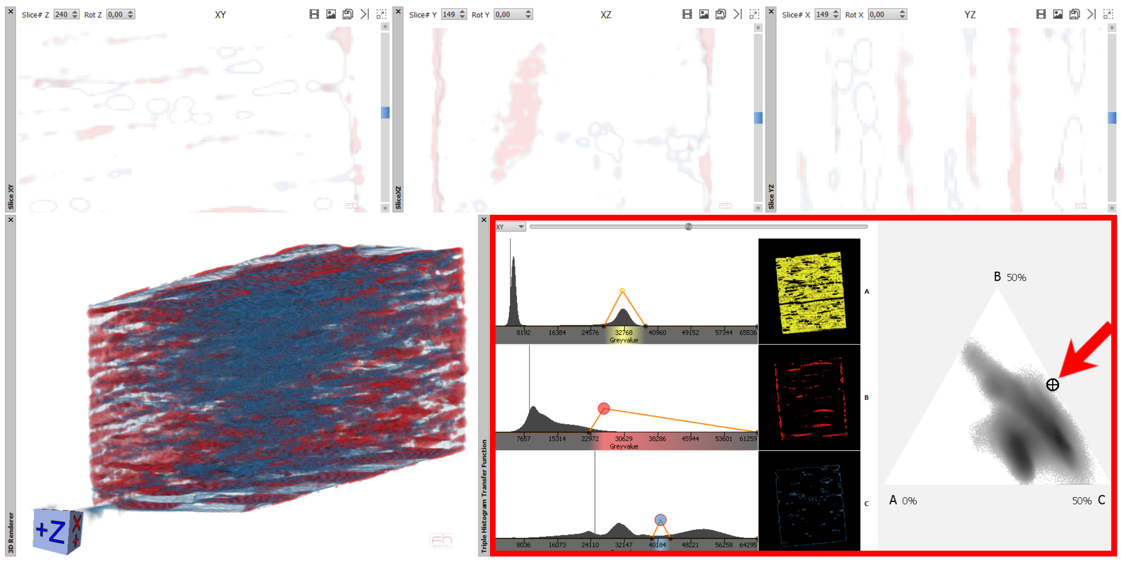


Figure 5.1: The implemented tri-modal TF widget in "stack" mode (outlined in red), where the individual TFs, their respective slicers, and the triangle are shown. The red arrow points to the control point. Above the histogram stack, a combo box allows the user to select the slicing axis of the local slicers, while a slider controls the current slice number. On the top, the three main slicers are shown, zoomed-in to clearly show the features (fibers and pores). On the left, the volume view uses the TFs defined in the developed widget to render the volume.

The first step of the widget's initialization is to set up open_iA's main renderer so that it can use three volumes simultaneously. That is achieved by first creating an append filter (with `vtkImageAppendComponents`) and adding the volumes (`vtkImageData`) of all three `iAModality` instances. A `vtkVolumeProperty` must be created as well, where the color and opacity functions (`vtkColorTransferFunction` and `vtkPiecewiseFunction` respectively) are set. Then, a `vtkSmartVolumeMapper` is created, its blend mode set to "composite", and its input data set to the previously defined append filter (with its `vtkImageAlgorithm::GetOutput()` method). A `vtkVolume` must then be initialized, setting its property to the created `vtkVolumeProperty` and its mapper to the created `vtkSmartVolumeMapper`. Finally, a `vtkRenderer` is created (adding the created `vtkVolume` to it) and added to the main renderer (`iARenderer`). This set-up enables VTK to do the color mixing of the three TFs without changing lower level components, such as the shader program generated by VTK.

The weighting of each modality in the final rendering does not require modifying the shader either. That is done by manipulating the control points of the modalities' individual TFs, limiting their opacity value based on the α, β and γ values, which are set using the triangle's control point. The functionality and drawing of the triangle are contained in the class `iABarycentricTriangleWidget`. This class also keeps track of the control point's coordinates and emits a signal [QtS] when they change.

The class `iATripleModalityWidget`, responsible for managing the widgets and their interconnections, receives this signal from `iABarycentricTriangleWidget` and uses the barycentric coordinates α, β and γ of the control point to modify the TFs of the modalities A, B and C by changing the range of the control points' opacity. For instance, for an original opacity range of $[0,1]$ and $\alpha = 0.25$, the range of A's TF opacity is changed to $[0, 0.25]$.

This range change does affect the appearance of `iADiagramFctWidget`, where the control points' heights are limited by the α, β or γ values, decreasing the precision of when adding and moving control points, or completely preventing interactions for values close to zero. To avoid that, a second `iADiagramFctWidget` is created, where the control points' opacity always remains in the range $[0,1]$. The user interacts with this proxy TF, while, on the background, the effective TF is manipulated to keep the opacity within the barycentric-coordinates-defined range.

To calculate and draw the frequency points in the triangle, Algorithm 4.2 was used in the same way as described in Section 4.1.3. The pixels on the screen are colored by transforming a value in the frequency array from the range $[min, max]$ to $[48, 255]$, similarly to the prototype. The frequency points must also be recalculated every time the triangle size changes (i.e., when the widget is resized).

5.2 Layout and interaction

While `iATripleModalityWidget` provides the functionality and the managing and wiring of all components, the layout is handled externally. The Stack and Triangle layouts are implemented in the classes `iAHistogramStack` and `iAHistogramTriangle` respectively. The former is shown in Figure 5.1, and the latter, in Figure 5.2.

In both layouts, the slicers have zooming and panning interactions, similar to the main (core-provided) slicers. A mouse click or drag on any position inside the triangle sets the control point's coordinates to that position. Additionally, hovering the mouse over the modality labels "A," "B" and "C" highlights them to indicate that a click is possible. When clicked, they set the corresponding modality's weight to 1 and, consequentially, the remaining two other weights to 0.

The slicers' axis and slice number are linked with the main slicer views. In open_iA, there are three main slicers XY, XZ, and YZ (one for each axis) that provide scrollbars, used to change the current slice number. By pressing a scrollbar or changing its position, their respective slice axis and number is applied to the TripleHistogramTF's slicers, as well as TripleHistogramTF's slicer mode combo box and slice number slider. Moreover, interacting with the slicer mode combo box, as well as the slice number slider, has a reverse effect, automatically changing the main slicer corresponding to the combo box's selected value.

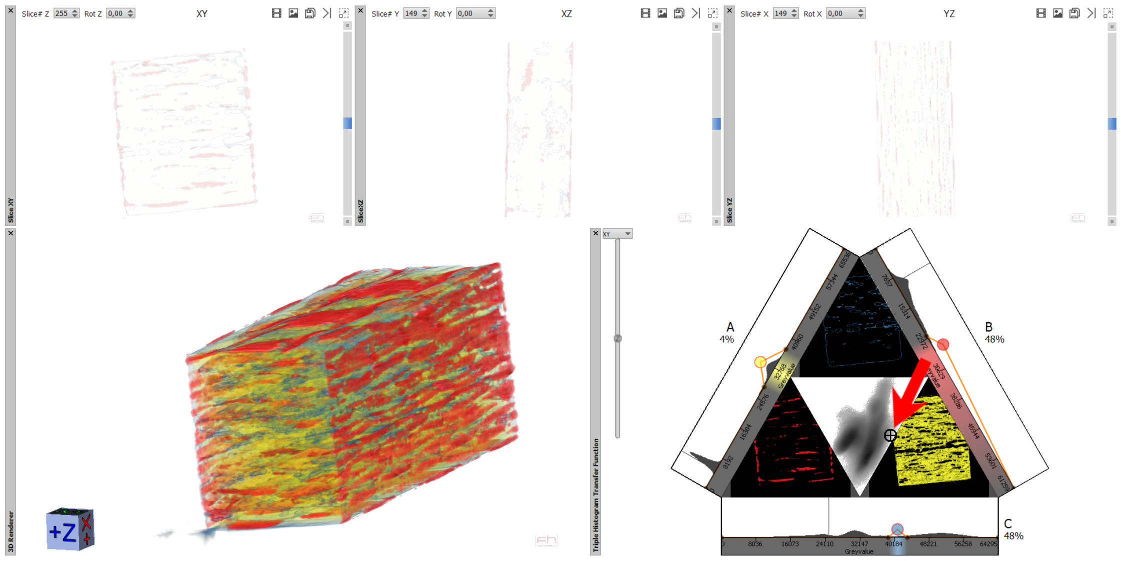


Figure 5.2: The implemented tri-modal TF widget in "triangle" mode (bottom-right). The individual TFs are laid out on the edges of the triangle, while their respective slicers are cut to fit in a triangular shape, placed opposite to their histogram. The red arrow points to the control point. The main slicers (top) are not zoomed in, which makes features difficult or impossible to spot (see Figure 5.1 for a zoomed-in instance).

The layout in `iAHistogramStack` is achieved through Qt-provided layout classes. A `QSplitter` separates the left part (histograms, slicers, combo box, and slider) from the right part (triangle control), allowing the user to manually resize them if desired.

`iAHistogramTriangle` does not follow the layout approach provided by Qt, as no layout class is used. Instead, each widgets is manually resized with `QWidget::resize(const QSize &)` and manually rendered to the screen. To determine the position and rotation of the widgets, a `QTransform` is created and applied to the `QPainter` before rendering a widget. The histograms are rendered with the method `QWidget::render(QPainter*)`, and the slicers, with the method `QGLWidget::grabFrameBuffer()`, accessible through `iASlicer`'s friend class `iASlicerWidget`. Additionally, the mouse event must be manually forwarded to the widgets. That is done by checking if the mouse event's point is contained inside a specific widget. If it is, this widget's `QTransform` is used to transform this point to the widget's space, the mouse event's point is set to the transformed point, and the event is sent with `QApplication::sendEvent(QObject*, QEvent*)`.

Furthermore, in this layout, the triangle is rotated in such a way that, when a control point is set, its proximity to a histogram directly relates to the corresponding modality's weight in the final rendering. For instance, if the control point is on the vertex touching the histogram of modality A, then $\alpha = 1$.

The empty spaces on the left, top and right corners of the original triangle utilized to display slicers, each corresponding to the histogram opposite to it in the triangle. The drawback with this solution is that the rectangular slices do not efficiently fit in the triangular area, which is compensated for by enabling zooming and panning interactions.

CHAPTER 6

Results and evaluation

Both proposed TF widgets "Histogram stack" and "Histogram triangle" were applied to tri-modal datasets to evaluate their efficacy in performing a frequent task in material analysis. The volume used is from a Carbon Fiber Reinforced Polymer (CFRP) laminate specimen scanned with SkyScan 1294 [Sky], producing an ABSorption contrast (ABS), a Differential Phase-Contrast (DPC) and a Dark-Field Contrast (DFC) modalities. A typical use case for the analysis of this kind of material is identifying individual features (e.g., fibers, pores). It is also useful for domain experts to be able to compare features (and their proportions) of two or more modalities.

Figures 6.1, 6.2 and 6.3 show the result of the individual TFs, achieved by setting the control point on the triangle vertices. These results are equivalent to having only one modality loaded, with only one TF. The colors were chosen with the help of ColorBrewer [CBr]. In Figure 6.1, the fiber bundles are highlighted in orange from the ABS modality (A). In Figure 6.2, fiber bundles parallel to the Z-axis are highlighted in cyan from the DFC modality (B). In Figure 6.3, pores are highlighted in magenta from the DPC modality (C). The hull of the CFRP is also visible in magenta, due to the nature of the DPC scanning technique, which precisely detects faces of structures.

The triangle's control point is set at the barycenter in Figure 6.4, which sets the weights of modalities A, B and C to $\alpha = \beta = \gamma = \frac{1}{3}$ respectively. With all modalities equally weighted, it is possible to see features of each of them in their specified colors.

Figure 6.5 shows fiber bundles from modality B in cyan and pores from modality C in magenta, while the rest of the volume is transparent. That was done by setting the triangle's control point as distant from modality A's histogram as possible and exactly between B and C, resulting in $\alpha = 0\%$, $\beta = 50\%$ and $\gamma = 50\%$ for modalities A, B and C respectively.

The "Histogram triangle" widget alternative showed itself to be more intuitive when interacting with the triangle's control point. With the "Histogram stack," it is necessary

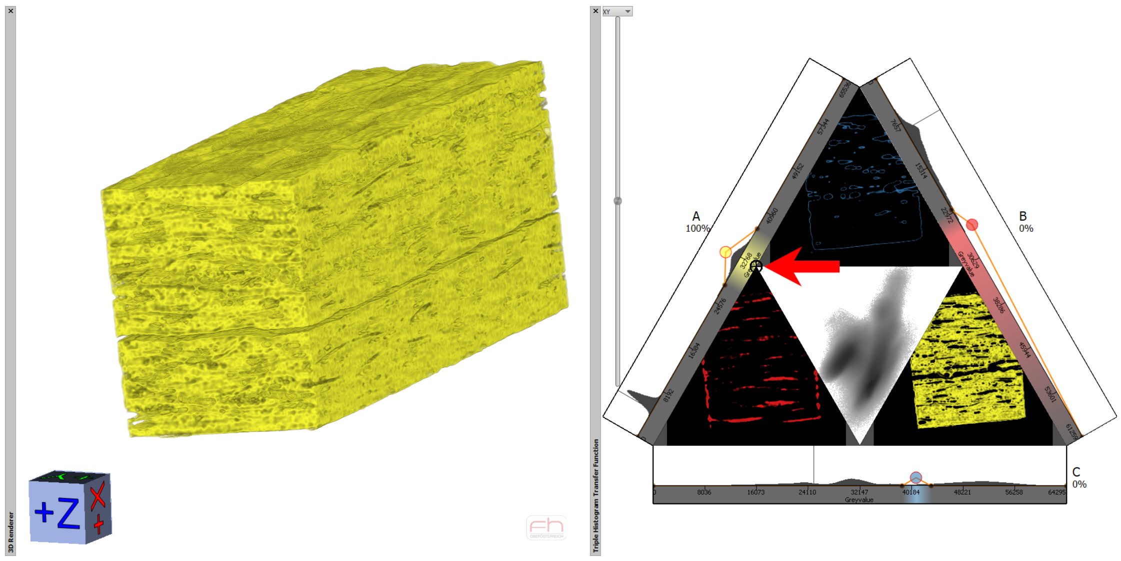


Figure 6.1: Rendering results with $\alpha = 100\%$, $\beta = 0\%$ and $\gamma = 0\%$. Only modality A is visible.

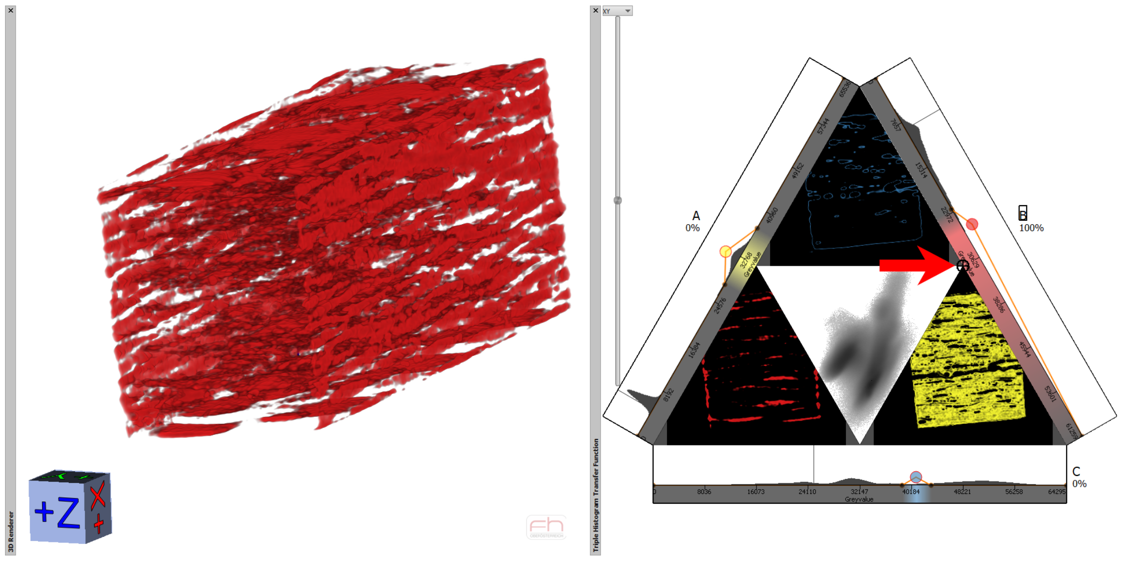


Figure 6.2: Rendering results with $\alpha = 0\%$, $\beta = 100\%$ and $\gamma = 0\%$. Only modality B is visible.

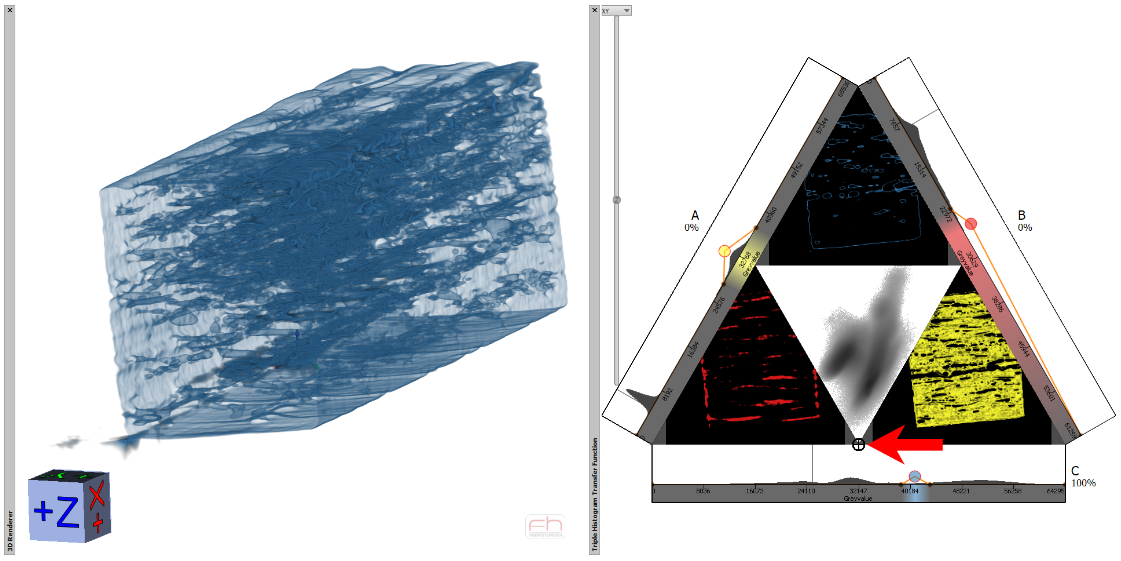


Figure 6.3: Rendering results with $\alpha = 0\%$, $\beta = 0\%$ and $\gamma = 100\%$. Only modality C is visible.

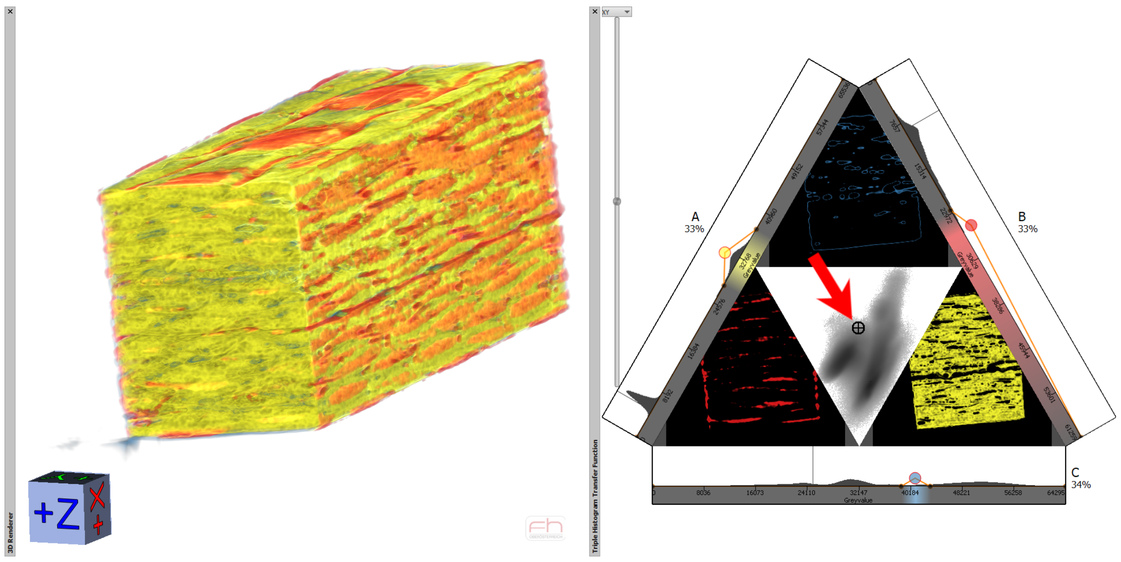


Figure 6.4: Rendering results with $\alpha = 33\%$, $\beta = 33\%$ and $\gamma = 34\%$. All modalities are visible.

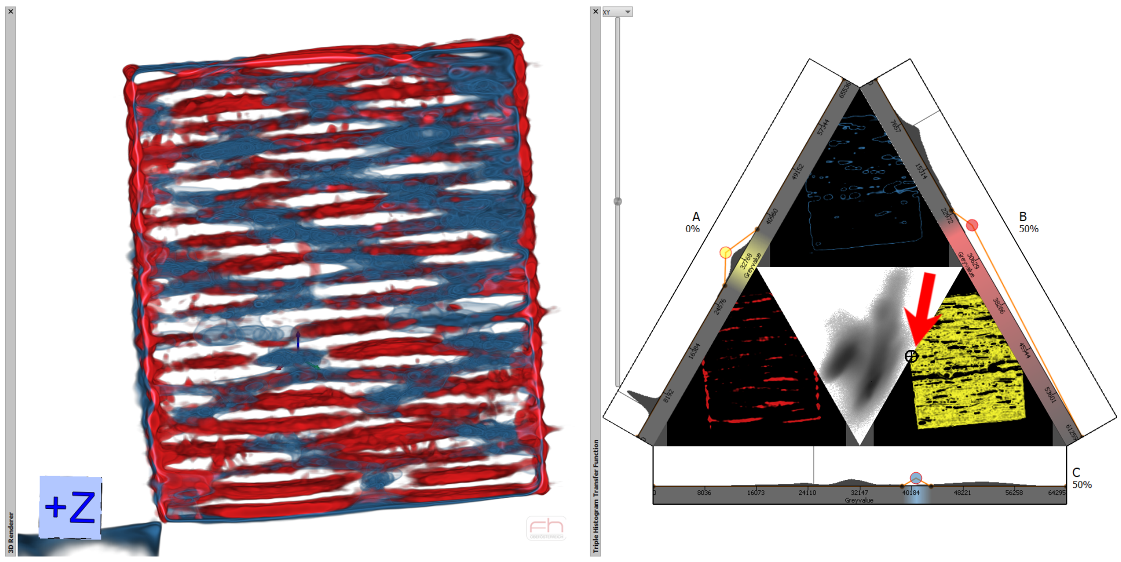


Figure 6.5: Rendering results with $\alpha = 0\%$, $\beta = 50\%$ and $\gamma = 50\%$. Only modalities B and C are visible.

to keep track of what vertex of the triangle relates to what modality, leading to a constant attention switch between the top part (histograms and slicers) and the triangle. The triangular layout also results in a more efficient use of the screen space.

However, there are trade-offs for this compactness. Visualizing the desired features on the slicers may require the user to pan and zoom constantly. Although possible to fit the slice in a triangle in such a way that it is completely visible, it becomes smaller, and the initial issue of unused space arises again.

The plotted frequency points do not directly help to set the weights of the modalities. In other words, setting the triangle's control point on darker or lighter areas does not result in more or less meaningful results. This context information only gives insight on the voxel-level relationship between the modalities. On Figures 6.1 through 6.5, for instance, the darker regions closer to modality C's histogram indicate that the majority of voxels in the barycentric-combined volume has a higher γ value. The white region close to modality A's histogram indicate that no voxels have a higher α value than β and γ.

The widget was implemented and tested on an Intel® Core™ i7-6500U CPU at 2.50GHz, 8 GB RAM, and an AMD Radeon™ R5 M330 GPU with Microsoft Visual Studio Enterprise 2015© [Vis] integrated development environment. In this set-up, manipulating the slicers (setting axis or current slice number) runs smoothly, editing control points in the individual TFs has a slight delay, and moving the triangle's control point has a considerable delay (over one second). Other interactions (such as panning, rotating and zooming the 3D view, the main slicers and the local slicers) run smoothly.

Tests were also performed on an Intel™ i7-3770 at 3.4GHz, 32 GB RAM, and an Nvidia™

Geforce® GTX 1080 GPU. In this set-up, only moving the triangle's control point has a slight delay, and all other interactions run smoothly.

CHAPTER 7

Conclusion

In this thesis, the feasibility of a tri-modal TF widget is tested. First, two conceptual ideas are presented and discussed, one of which evolves into two concrete methods, implemented in a prototype application. Next, this prototype is explained in detail, including the algorithms used and layout decisions, and then evaluated. The implementation of the refined prototyped methods and layouts in open_iA is described, followed by an evaluation based on real scenarios of domain experts in material analysis, finishing with a comparison of the two implemented layouts.

The presented tool demonstrates that it is feasible to design usable TF interfaces focusing on volume exploration. The introduction of a simple weighting component (a triangle) is sufficient to accelerate and facilitate the highlighting of features across modalities. The widget will be available as a module of open_iA under the name "TripleHistogramTF."

Switching UI components is also possible, if necessary to better meet the specific needs of an application. For example, the color-opacity TFs (as used in open_iA) can be replaced by other TFs, such as the 2D gradient magnitude [KKH02]. The slicers inside the triangle can also be replaced, as they serve only to guide the user.

Improvements in the development method and on the widget are planned to be explored in future work. For instance, user studies are necessary to verify the conclusions and more precisely compare the implemented representations (stack and triangle). Moreover, more meaningful ways to show context information could be explored, aiming to help the user more directly choose an appropriate control point position. The visibility of the main slicers, as shown in Figures 5.1 and 5.1, also needs improvement to achieve similar clarity to that seen in the prototype, shown in Figures 4.5 and 4.6. Performance improvements are also planned (e.g., GPU implementations), along with the ability to switch between the "stack" and "triangle" layouts (currently only possible programmatically).

The Barycentric Mapping idea presented in Section 4.1 is also planned to be further explored. User studies can be carried out to precisely evaluate its meaningfulness, as well as test alternative interaction possibilities, such as brushing. Furthermore, this work hopes to initiate a more widespread availability of multimodal volume exploration tools, a lacking feature in current visualization software, as discussed in Chapter 1.

List of Figures

List of Tables

List of Algorithms

Bibliography

[Ami] Amira-Avizo. https://www.fei.com/software/amira-avizo. Accessed: 25.09.2018.

[BRB$^+$13] R. Bramon, M. Ruiz, A. Bardera, I. Boada, M. Feixas, and M. Sbert. Information theory-based automatic multimodal transfer function design. *IEEE Journal of Biomedical and Health Informatics*, 17(4):870–880, jul 2013.

[CBr] ColorBrewer. http://colorbrewer2.org/#type=qualitative&scheme=Dark2&n=3. Accessed: 23.09.2018.

[CTR] CT Group at the University of Applied Sciences Upper Austria. www.3dct.at/. Accessed: 24.09.2018.

[Die] Johannes Diemke. Incremental 2D Delaunay triangulation algorithm. https://github.com/jdiemke/delaunay-triangulator. Accessed: 22.09.2018.

[GXY11] Hanqi Guo, He Xiao, and Xiaoru Yuan. Multi-dimensional transfer function design based on flexible dimension projection embedded in parallel coordinates. In *2011 IEEE Pacific Visualization Symposium*. IEEE, mar 2011.

[ITK] ITK. https://itk.org/. Accessed: 24.09.2018.

[Jav] Java. https://www.java.com. Accessed: 24.09.2018.

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

[LKG$^+$16] Patric Ljung, Jens Krüger, Eduard Gröller, Markus Hadwiger, Charles D. Hansen, and Anders Ynnerman. State of the art in transfer functions for direct volume rendering. *Computer Graphics Forum*, 35(3):669–691, jun 2016.

[MATa] MATLAB. https://mathworks.com. Accessed: 25.09.2018.

[MATb] MATLAB API for Java. https://de.mathworks.com/help/matlab/matlab-engine-api-for-java.html. Accessed: 22.09.2018.

[MeV] MeVisLab. https://www.mevislab.de. Accessed: 25.09.2018.

[ope] open_iA. https://github.com/3dct/open_iA. Accessed: 24.09.2018.

[Qt] Qt. https://www.qt.io/. Accessed: 24.09.2018.

[QtS] Qt Signals & Slots. http://doc.qt.io/archives/qt-4.8/signalsandslots.html. Accessed: 25.09.2018.

[Sky] SkyScan 1294 overview. https://www.bruker.com/products/microtomography/in-vivo-micro/skyscan-1294/overview.html. Accessed: 22.09.2018.

[VGS] VGSTUDIO MAX. https://www.volumegraphics.com/products/vgstudio-max.html. Accessed: 25.09.2018.

[Vis] Visual studio. http://microsoft.visualstudio.com/. Accessed on 29.10.2018.

[VTK] VTK. https://www.vtk.org/. Accessed: 24.09.2018.

[ZK10] Xin Zhao and Arie Kaufman. Multi-dimensional reduction and transfer function design using parallel coordinates, 2010.

[ZSH12] Liang Zhou, Mathias Schott, and Charles Hansen. Transfer function combinations. *Computers & Graphics*, 36(6):596–606, oct 2012.