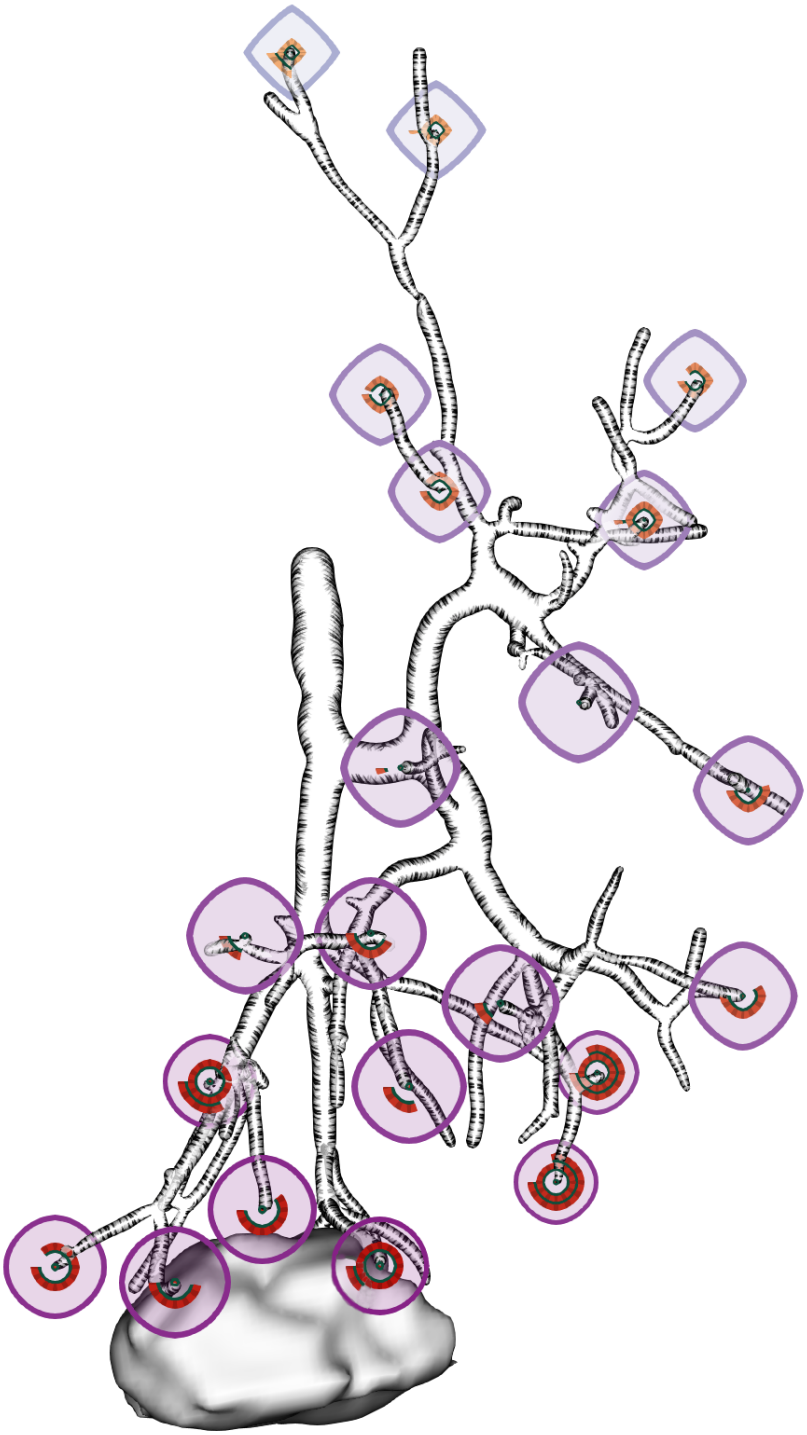
ABSTRACTION OF BIO-MEDICAL SURFACE DATA FOR ENHANCED COMPREHENSION AND ANALYSIS

NILS LICHTENBERG



ABSTRACTION OF BIO-MEDICAL SURFACE DATA FOR ENHANCED COMPREHENSION AND ANALYSIS

by

NILS LICHTENBERG

Approved Dissertation thesis for the partial fulfillment of the requirements for a
Doctor of Natural Sciences (Dr. rer. nat.)
Fachbereich 4: Informatik
Universität Koblenz-Landau

Chair of PhD Board:	Prof. Dr. Maria Wimmer
Chair of PhD Commission:	J.-Prof. Dr. Tobias Krämer
Examiner and Supervisor:	J.-Prof. Dr. Kai Lawonn
Further Examiners:	Prof. Dr. Lars Linsen
	Prof. Dr. Timo Ropinski
Date of doctoral viva:	January 31, 2020

Opa, ich hätte dir gerne noch davon berichtet ...

ABSTRACT

Bio-medical data comes in various shapes and with different representations. Domain experts use such data for analysis or diagnosis, during research or clinical applications. As the opportunities to obtain or to simulate bio-medical data become more complex and productive, the experts face the problem of data overflow. Providing a reduced, uncluttered representation of data, that maintains the data's features of interest falls into the area of *Data Abstraction*. Via abstraction, undesired features are filtered out to give space - concerning the cognitive and visual load of the viewer - to more interesting features, which are therefore accentuated. To address this challenge, the dissertation at hand will investigate methods that deal with *Data Abstraction* in the fields of liver vasculature, molecular and cardiac visualization. Advanced visualization techniques will be applied for this purpose. This usually requires some pre-processing of the data, which will also be covered by this work. *Data Abstraction* itself can be implemented in various ways. The morphology of a surface may be maintained, while abstracting its visual cues. Alternatively, the morphology may be changed to a more comprehensive and tangible representation. Further, spatial or temporal dimensions of a complex data set may be projected to a lower space in order to facilitate processing of the data. This thesis will tackle these challenges and therefore provide an overview of *Data Abstraction* in the bio-medical field, and associated challenges, opportunities and solutions.

ZUSAMMENFASSUNG

Biomedizinische Daten existieren in verschiedenen Formen und mit unterschiedlichen Repräsentationen. Experten nutzen diese Daten für die Analyse oder Diagnose, in der Forschung oder im klinischen Alltag. Da die Möglichkeiten, biomedizinische Daten aufzunehmen oder zu simulieren, komplexer und produktiver werden, stehen die Experten vor dem Problem des Datenüberflusses. Die Bereitstellung einer reduzierten, übersichtlichen Darstellung von Daten, die die interessanten Merkmale der Daten aufrechterhält, fällt in den Bereich der *Data Abstraction*. Über die Abstraktion werden unerwünschte Merkmale herausgefiltert, um - was die kognitive und visuelle Belastung des Betrachters betrifft - Raum für interessantere Merkmale zu schaffen, die dadurch hervorgehoben werden. Um diese Herausforderung zu meistern, werden in der vorliegenden Dissertation Methoden untersucht, die sich mit der *Data Abstraction* in den Bereichen Lebergefäß-, Molekül- und Kardio-Visualisierung befassen. Zu diesem Zweck werden fortgeschrittene Visualisierungstechniken eingesetzt. Dies erfordert in der Regel eine gewisse Vorverarbeitung der Daten, die auch durch diese Arbeit abgedeckt wird. *Data Abstraction* selbst kann in verschiedenen Formen umgesetzt werden. Die Morphologie einer Oberfläche kann beibehalten werden, während ihre visuellen Darstellung abstrahiert wird. Alternativ kann die Morphologie in eine verständlichere, greifbarere Darstellung geändert werden. Darüber hinaus können räumliche oder zeitliche Dimensionen eines komplexen Datensatzes auf einen niedrigeren Dimensionsraum projiziert werden, um die Verarbeitung der Daten zu erleichtern. Diese Arbeit wird sich diesen Herausforderungen stellen und daher einen Überblick über *Data Abstraction* im biomedizinischen Bereich und die damit verbundenen Herausforderungen, Chancen und Lösungen geben.

DANKSAGUNGEN

Wie die meisten wissen, bin ich kein Mensch vieler Worte, ...

... aber man kann ja mal eine Ausnahme machen.
Nur, wo fängt man an? Bei den Lehrern, die mir eine dunkle Zukunft vorhergesagt haben, wohl eher nicht. Dann doch erst mal ganz klassisch bei meinem Doktorvater Kai Lawonn, der mich bereitwillig auf sein DFG-Projekt gesetzt, geduldig mit mir diskutiert, mich korrigiert, inspiriert und mir jegliche Freiheiten eingeräumt hat. Also Kai, danke für dein Vertrauen und die gute Zeit, absolute Empfehlung für einen Lachgesicht-Stempel!
Auch möchte ich mich bei Stefan Müller und Dietrich Paulus bedanken, die mir überhaupt erst den Weg zur Promotion ermöglicht und mich bei meiner initialen Themenfindung sowie im weiteren Verlauf unterstützt haben. An dieser Stelle auch vielen Dank an alle ehemaligen Kollegen, für eure stetige Hilfsbereitschaft.
Noeska, I thank your for our collaboration and your great advice on how to properly structure a scientific paper. Our first work was exactly what I needed to really get started and to get the confidence for successive publications.
Sandy, deine ausgezeichnete Betreuung meiner Masterarbeit am DKFZ hat sicherlich auch zu meinem weiteren Weg beigetragen und es hat mich sehr gefreut, dass wir anschließend nochmal gemeinsam an die Thematik anknüpfen konnten. Danke dafür.
Den Kollegen Ajay A. George, Pascal Heimer und Diana Imhof verdanke ich sehr interessante Einblicke in den Bereich der Pharmazie. Danke für die produktiven Diskussionen und die gemeinsame Arbeit.
Es gibt auch einige Studenten, denen ich danken möchte. Ihr möget euch angesprochen fühlen: Ohne eure ausgezeichneten Abschlussarbeiten wäre es für mich nicht möglich gewesen die letztendliche thematische Breite meiner Dissertation zu erreichen. Danke euch!

Meiner Familie danke ich für die gedrückten Daumen, die angezündeten Kerzen und dafür dass ihr an mich geglaubt habt. Vor allem meinen Eltern. Eure Unterstützung hat mir immer erlaubt, das in Angriff zu nehmen, was mir gerade vernünftig vorkam oder Spaß gemacht hat und war damit natürlich maßgeblich meinen Weg.
Sugi, du hast mir ebenso den Rücken freigehalten. Gerade dann, wenn ich zu Deadline-Zeiten mal wieder einen Tunnelblick entwickelt habe, hast du für uns gemeinsam nach rechts und links geblickt um das Ziel sicher zu erreichen.

CONTENTS

Part I

PREAMBLE

The thesis at hand makes several contributions to the field of data abstraction and visualization in a bio-medical context. The content is based on the manuscripts published by the author of this thesis which are presented as stand-alone chapters throughout this written work. In the following, an introductory text will clarify the relationship of the individual publications and therefore emphasize the combined contribution of this thesis to the visualization community. After that, the structure of this theses will be summarized.

1 INTRODUCTION

This thesis is placed in the area of bio-medical visualization. In this field, a multitude of disciplines come together, that are required to reach from raw bio-medical data to a final, comprehensive visualization. The raw data has to be pre-processed, translated into a displayable and task-oriented representation, and then to be rendered to a display. Following the formulation of the *Visualization Pipeline* by Haber and McNabb [59], the three consecutive main steps after raw data acquisition are the *Data Enrichment*, the *Visualization Mapping* and the final *Rendering*. During *Data Enrichment*, given data may be optimized for subsequent steps, or additional attributes are derived from the data. The *Visualization Mapping* transforms the data into a so called *Abstract Visualization Object* (AVO, see [59]), which contains information about, e.g., geometries or colors to be applied, in order to allow for a comprehensive visualization of the data. The AVO, as the name suggests, is an abstraction of an object. For example, the world map is a 2D abstraction of the earth, see Figure 1.1. This process of abstraction can be employed in various ways and on very different levels of abstraction. Finally, the *Rendering* step creates an image to be displayed. The displaying step can be dictated by the available hardware or the task and utility of the visualization. Generating a single still image may be allowed to take an arbitrary amount of time, while an interactive application requires several images to be rendered per second.

These three steps are each very broad and highly connected within this visualization pipeline. The thesis at hand makes several contributions to the *Visualization Mapping* stage, but also to the two other stages. It will show that, when utilizing view-dependent visualizations, the three steps form a loop, where the *Rendering* result is used as an input for subsequent *Data Enrichment* and *Visualization Mapping*. An example, which further highlights the link between the individ-

Figure 1.1: The 2D map of the earth is an example for an AVO of the earth itself.

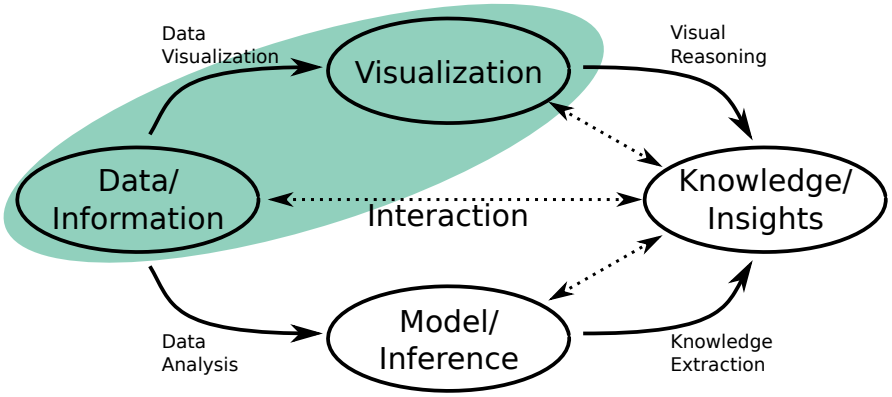


Figure 1.2: Concept of *Visually Enabled Reasoning* by Meyer et al. [154]. The main coverage of this thesis is highlighted green.

ual steps of the *Visualization Pipeline*, and is also picked up during this thesis, is *Illustrative* or *Non-Photorealistic Rendering* (NPR). NPR techniques are applied during the *Rendering* step, but rely on an appropriate pre-processing of the data. By their concept, illustrative rendering techniques can also be seen as an additional abstraction of the rendered AVO and thus, they can be accounted for both, the *Visualization Mapping* and *Rendering*.

The *Visualization Pipeline* can be set in a larger context using the formulation of *Visually Enabled Reasoning* by Meyer et al. [154], depicted in Figure 1.2. They argue that visualization, combined with user interaction, enables the interactive process of gaining insight into data. The contribution of this thesis covers a part of their concept, highlighted green in Figure 1.2. Therefore, the methods presented throughout the work at hand aim to provide responsive, i.e., real-time or interactive, tools and foundations for visualization. These create a basis for further research or the implementation of responsive visualization concepts.

In the following, the motivation for and the structure of this thesis will be described. The state-of-the-art and related work of each topic and publication are discussed in the respective chapter.

1.1 MOTIVATION AND OVERVIEW

To narrow the application range, this thesis mainly focuses on the visualization of liver vasculature, but also draws the bio-molecular domain and ensembles of anatomical data as an example.

Focus-and-Context visualization.

Part II begins by addressing the visualization of ensembles. If multiple objects of interest are visualized in a combined view, the observer may have trouble to distinguish individual structures or features. Then, focus-and-context applications can be employed to guide the visual perception of the observer. Such applications utilize different information channels from visual perception [28] to make selected

objects appear as either focus- or context [70]. For example, depth-of-field, transparency, different colors or differently shaped patterns can be exploited for this. Part II contributes to this field by utilizing illustrative line-drawing techniques to achieve continuous levels of abstraction for individual structures in an ensemble.

Surface parameterization

The rendering of patterns or textures requires the displayed AVO to be equipped with texture coordinates. The surface parameterization domain deals with the generation of such. It has a wide range of applications and has been intensely researched for a long time [47]. Global parameterization, for instance, plays an important role in remeshing algorithms [15, 92, 176] that aim to distribute vertices optimally on a surface. The vertices are then usually aligned to a pre-defined direction field. While the above mentioned parameterization algorithms focus on remeshing tasks, it is conceivable that the proposed ideas have the potential to be translated to the visualization domain. Hence, Chapter 3 of this thesis will make an effort to build a bridge from the parameterization- or meshing-domain to the visualization domain.

Visualization of liver vasculature.

For liver surgeries, image-guided planning and navigation has become an important pre- and intra-operative tool for surgeons and interventional radiologists. It enables new minimal-invasive procedures, and has the potential to improve the accuracy and success of existing surgical and interventional approaches [32]. Incorrect spatial interpretation is a common perceptional problem in 3D applications and visualizations [116]. Here, the term spatial perception distinguishes between techniques that encode the distance from an object to the observer (*depth*), the distance between objects (*inter-object distance*), and the *shape* of an object. Several studies have shown that users often have serious problems estimating depth and inter-object distances in virtual worlds [82, 96, 208]. The need to assess spatial information of 3D models accurately during an intervention has led to the development of several techniques to improve spatial perception, which is still part of active research. In the field of liver surgery, it is important for the clinician to fully understand the vascular structures within the liver. Structural complexity of the multiple branches and cluttered blood vessels does not make this an easy task. And so there is a demand for specialized visualization techniques that simplify the comprehension of such medical data. Therefore, part III of this thesis aims to contribute to this field in terms of improved depth perception to enhance spatial comprehension of 3D vascular data.

Abstraction by dimension reduction.

An alternative approach to visualizing complex structures and to make them more comprehensive and conceivable, is to reduce their complexity to the required or necessary features. While NPR techniques do not alter a surface itself, but rather only highlight a subset of features, the morphology or dimensionality of a data object can as well be changed for purposes of abstraction. Such as the world map is a 2D AVO of the earth, structures from the medical domain

can as well be mapped to lower dimensions. The report by Kreiser et al. [109] underlines that there is a high interest in techniques that transfer spatio-temporal data to the 2D domain. The main advantage of a 2D depiction of a data set is that it can be perceived at a glance on a 2D display. No interaction, like rotating a 3D object, is required to fully grasp the visualization. The present work contributes to this area in Part IV by two examples. First, a task-oriented AVO of large molecular dynamics (MD) data is created for MD simulation visualization. Data sets produced by MD simulations grow ever bigger with more powerful computers and embrace longer time ranges or are temporally more densely sampled. At the same time, it becomes increasingly difficult for the domain experts to analyze the 3D and time resolved simulation results. Therefore, divide-and-conquer strategies are required to allow chemists to work themselves from high level features in a data set to low level features that provide insight into the processes that determine the final outcome of a chemical reaction. As the extraction of such high level features requires some sort of data abstraction, it can be well placed into the *Visualization Mapping* step. The second example are segmentations of the mitral valve (MV, one of the four human heart valves) that are transformed to the 2D domain in order to support clinical assessment. The motivation is, again, a more comprehensive view on complex data. The MV may develop deficiencies that may cause minor symptoms but also life threating conditions for a patient. Physicians and surgeons are then interested in the behavior and morphology of the valve over a cardiac cycle, in order to plan and conduct restorative interventions. However, the 3D structure of the valve for a single time step of the cardiac cycle may already be highly convoluted and therefore difficult to comprehend and analyze. Hence, a mapping to 2D space is feasible to support surgeons in their analysis task.

In brief, this dissertation contributes to the improvement of perception and comprehension of medical vascular data sets, as well as to the support of bio-medical analysis tasks through dimension reduction. This is done by applying different, task-oriented approaches of data abstraction to the input data. Further, aspects of the parameterization domain are taken into account, modified and applied to the respective visualization tasks.

1.2 THESIS STRUCTURE

The main content of this thesis is presented in three parts. Each part consists of chapters that are based on individual publications or manuscripts that are currently under peer-review. These publications or manuscripts are the result of collaborative work with students, colleagues and domain experts. Nils Lichtenberg, the author of this dissertation, was significantly involved in all of these works in

terms of theoretical and practical contributions. Details are provided in a report of individual share to cooperative contributions, submitted to the doctoral committee.

Part II begins with an investigation on how illustrative rendering techniques can be employed to achieve different levels of visual abstraction and covers the *Data Enrichment* and *Rendering* step of the *Visualization Pipeline* (Chapter 2). This is exemplary implemented as an interactive focus-and-context visualization for ensembles of medical surface objects. Then, concepts of the parameterization- or mesh-processing domain are used to generate periodic texture coordinates for surface meshes (Chapter 3). It is shown how these periodic coordinates can be employed to encode scalar data or for tasks of visual abstraction. Even dynamic scalar fields can be illustrated in a frame-coherent and smooth manner.

Part III focuses on the visualization of liver vasculature. Here, the whole *Visualization Pipeline* is covered and also the loop, i.e., connecting the *Rendering* and the *Data Enrichment* steps, is introduced. The part begins with a contribution to enhanced depth perception, by view-dependently augmenting the vasculature with glyphs (Chapter 4). The glyphs are positioned at previously extracted candidate positions.

In the subsequent chapter, the candidate extraction is further improved and solved more robustly (Chapter 5). In contrast to the general parameterization technique of the previous part, a new hybrid approach is presented. It is tailored to tree-like structures and hybrid in the sense that part of the solution is found in world space and part is found in screen space. The screen space parameterization provides options for view-dependent illustrations of the vessel surface, that would otherwise be harder to achieve.

The next chapter picks up the idea of screen space parameterization and generalizes this concept (Chapter 6). Examples are, however, predominantly given for vascular input data. Furthermore, the vascular data is rendered implicitly, based on a *Signed Distance Field* (SDF). Once given, an SDF provides promising properties that can be used for interactive visualization, as proposed in the respective chapter. Lastly, the vascular input data is additionally represented as a 2D graph abstraction, in order to support the comprehension of complex branch topology and associated information.

Finally, this part summarizes recent methods that attempt to improve depth perception of vascular trees, using auxiliary, i.e., supportive tools (Chapter 7).

Part IV further pursues the topic of 2D abstraction. Contributions are provided in the domain of drug design (Chapter 8) and cardiac surgery (Chapter 9). In either case, complex surface representations are transferred to 2D depictions of the data. For the molecular data of

the drug design domain, complex 3D and time resolved simulation data is transferred to a heat-map, uncovering structural and temporal correlations of residue movement. The field of cardiac surgery is supported by an approach that unfolds complex heart valve structures to an uncluttered 2D representation. Expert feedback underlines the feasibility of both approaches to support analysis tasks in the biomedical field.

At last, a final conclusion is given in Part V. Here, the contributions of this thesis to the visualization community are wrapped up and discussed, as well as goals for future research are being extracted.

Part II

ILLUSTRATIVE ABSTRACTION

This part uses existing illustrative line-drawing techniques and shows how they can be combined with a novel hatching method to achieve different levels of visual abstraction of a rendered surface (Chapter 2). The hatching method is based on a previous sampling of the surface. From this, the necessity for controllable and uniform surface sampling is derived. This topic will be covered in the subsequent contribution (Chapter 3), presenting a field aligned parameterization and sampling method, that yields a basis for further visualization approaches.

This part consists of the following papers:

Lichtenberg, N., Smit, N., Hansen, C., Lawonn, K., "Sline: Seamless Line Illustration for Interactive Biomedical Visualization." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2016, pp. 133–142. DOI: 10.2312/vcbm.20161281

Lichtenberg, N., Smit, N., Hansen, C., Lawonn, K., "Real-time field aligned stripe patterns." In: *Computers & Graphics* 74 (2018), pp. 137–149. DOI: 10.1016/j.cag.2018.04.008

2

SLINE: SEAMLESS LINE ILLUSTRATION FOR INTERACTIVE BIO-MEDICAL VISUALIZATION

ABSTRACT In medical visualization of surface information, problems often arise when visualizing several overlapping structures simultaneously. There is a trade-off between visualizing multiple structures in a detailed way and limiting visual clutter, in order to allow users to focus on the main structures. Illustrative visualization techniques can help alleviate these problems by defining a level of abstraction per structure. However, clinical uptake of these advanced visualization techniques so far has been limited due to the complex parameter settings required.

To bring advanced medical visualization closer to clinical application, we propose a novel illustrative technique that offers a seamless transition between various levels of abstraction and detail. Using a single comprehensive parameter, users are able to quickly define a visual representation per structure that fits the visualization requirements for focus and context structures. This technique can be applied to any bio-medical context in which multiple surfaces are routinely visualized, such as neurosurgery, radiotherapy planning or drug design. Additionally, we introduce a novel hatching technique, that runs in real-time and does not require texture coordinates. An informal evaluation with experts from different bio-medical domains reveals that our technique allows users to design focus-and-context visualizations in a fast and intuitive manner.

2.1 INTRODUCTION

Human anatomy consists of many closely arranged structures. In 3D visualization of anatomy, the proximity and number of structures can cause perception problems, hampering the focus on structures of interest. Especially for regions featuring complex anatomy, such as the pelvis, many organs are arranged in a confined region and visualization of these spatial arrangements is difficult. Furthermore, the number of different structures visualized can lead to visual clutter or distraction from the areas of interest. For many medical visualization tasks, a pathology or target structure needs to be visualized in anatomical context. Simply showing every structure is then not ideal, due to a potential visual overload.

Advanced visualization techniques, such as focus-and-context techniques, are able to emphasize structures of interest, and de-emphasize context structures. Context representations are then abstracted and

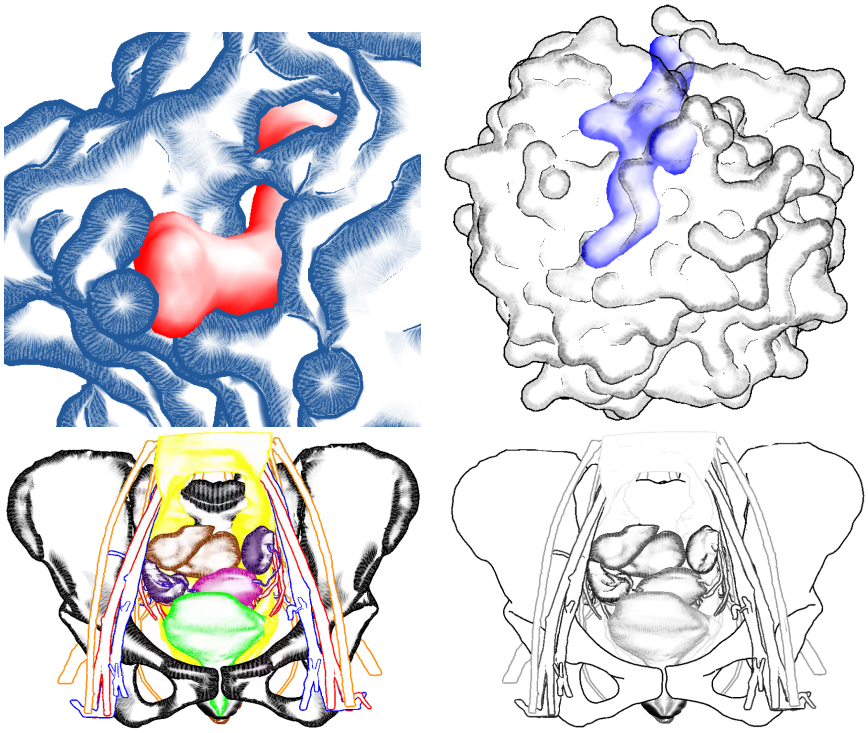


Figure 2.1: Bio-medical focus-and-context scenes generated with Sline.

serve to provide an indication of the spatial extent, without presenting much shape detail. In this way, distraction from the focus structure(s) is prevented. The focus structures are then presented in a detailed way, so that shape and spatial perception is supported. Illustrative, or non-photorealistic rendering (NPR) techniques can be employed to provide the necessary level of abstraction.

While illustrative techniques can be successfully applied to medical visualization problems, e.g., to visualize multimodal data or for surgical treatment planning, clinical uptake of these novel techniques so far is limited. In clinical contexts where segmentation is routinely performed, such as for neurosurgery or radiotherapy planning, structures are often visualized using Phong shading combined with a simple opacity setting. Rendering structures in a similar shading style could lead to visual distraction from focus structures, or in case the structures are rendered in a transparent way, reduced shape perception. While applications such as these would benefit from more advanced visualization techniques, a limiting factor is the complex parameter specification required to generate effective visual representations. Furthermore, many illustrative techniques, such as silhouettes, suggestive contours, and hatching exist, but they are not presently available in a comprehensive implementation.

In this paper, we present the integration of several NPR techniques into a single application, entitled Sline (Seamless Line Illustration). With this, our main contributions are the following:

1. Our technique provides a seamless transition between surface rendering styles, from high levels of abstraction to less abstraction and finally to a smooth, non-illustrative appearance.

2. Using a single parameter setting per structure, users have the ability to quickly and intuitively visualize a complete 3D scene with several focus and context structures.

3. We introduce a novel real-time hatching technique, which does not require texture coordinates.

We define an ordered set of rendering styles to represent different stages of abstraction. Sline does not require texture coordinates for its rendering styles, in order to make our technique suitable for a wide range of bio-medical research data. Each rendering style depends on several parameters of which a subset will be used for the transition among the stages. By mapping these parameter subsets to a single comprehensive parameter, users can intuitively and quickly set the desired visual representation for structures or groups of structures. The resulting visualization can convey focus-and-context information through choice of rendering styles, even without the use of color (see Figure 2.1).

The rest of this paper is structured as follows. First, in Section 2.2, we describe work related to illustrative visualization techniques and their bio-medical applications. In Section 2.3, we describe the methods that constitute Sline. Afterwards, we present the results of Sline in three case studies combined with an informal evaluation with domain experts in Section 2.4, followed by a discussion in Section 2.5 as well as a conclusion and outlook on future research directions in Section 2.6.

2.2 RELATED WORK

The basis of our work focuses on a smooth transition from distinct illustrative visualization techniques at varying levels of abstraction to a smooth, non-illustrative rendering style. With this work, we mainly focus on line drawing techniques. Thus, in this section we will give an overview of the most commonly used line drawing techniques: *feature lines* and *hatching* approaches. Furthermore, we discuss applications of these techniques in the bio-medical domain.

FEATURE LINES Feature lines aim to represent the strongest shape cues of a surface mesh with lines. When an artist is asked to draw a surface with only a few lines, the lines would be placed at the most significant regions. Although this is a very subjective task, some lines will be placed more often than others, depending on the geometric details of the underlying surface. Especially if the normals of the surface vary strongly, e.g., at a strong edge, a line would commonly be drawn

there. Feature line techniques aim to automate this process, such that these lines are placed based on shading or surface information. Shading approaches take the normals of the surface mesh and the light vector into account. Mostly, a headlight is used where the light vector coincides with the view direction of the camera. The most intuitive feature line in this category is the contour. The contour is defined as the loci of points where the normal and the view direction are mutually perpendicular. As the normal is only defined at the vertices of the mesh, the dot product of the view vector and the normal is determined at these points. Afterwards, the signs of the vertices per triangle are checked and if the sign changes, a line is constructed on the triangle that interpolates the negative and positive value such that a zero-crossing is obtained. This contour yields a reasonable impression of the surface, but is not sufficient for a detailed spatial impression. Another approach, which is based on shading, was presented by DeCarlo et al. [40]. Xie et al. [231] presented *photic extremum lines*. Their approach identifies regions of high variance in the shading. For this the maximum of the magnitude of the light gradient in direction of the light gradient is determined. The user has the possibility to add more light sources to influence the result. This feature can be useful when regions with a high amount of noise produce too many feature lines. Adding more light sources that point to this region reduces the number of lines. This approach was later improved by Zhang et al. [236] to improve the runtime performance. Zhang et al. [237] also presented *Laplacian lines* as an extension of the Laplacian-of-Gaussian edge detector used in image processing to surface meshes. The Laplacian of the normal is determined component-wise, again yielding a vector. Afterwards, the dot product with the resulting vector and the view direction is determined, and the zero-crossings are used as potential candidates for the feature lines. Although this approach is fast in comparison to other approaches, the pre-processing step of determining the Laplacian is very time-consuming, as the authors suggested to use the Belkin weights [7].

Regarding the feature line approaches that are based on geometric properties of the surface, Interrante et al. [84] proposed *ridges and valleys*. Their approach was adapted to triangulated surface meshes by Ohtake et al. [163]. The ridge and valley lines require principal curvature information. First, the zero-crossing of the directional derivative of the greatest curvature along the corresponding principal curvature directions is determined. Depending on the sign of the second directional derivative and on the sign of the curvature, the feature line is either a ridge or a valley. A more advanced approach was presented by Judd et al. [91]. Their *apparent ridges* are determined similar to the ridges and valleys, but in contrast they employ their own definition of curvature. They presented a view-dependent curvature such that contours are also determined.

All the presented feature line methods have advantages and disadvantages. An overview of feature line techniques, which summarizes these positive and negative aspects, can be found in the survey by Lawonn and Preim [126].

HATCHING In contrast to feature lines, where the most salient regions are depicted by single lines, hatching tries to convey a spatial impression of the surface, by covering the surface with a large number of line strokes. The stroke style depends on the underlying shading, such that the number of lines increases for regions with dark shading. The first approach of hatching lines on a surface was introduced by Hertzmann and Zorin [75]. They determined the principal curvature directions and smoothed them to obtain less noisy results. Afterwards, they determined the integral lines that represent the lines over the surface. A texture-based approach was introduced by Praun et al. [172]. They used lapped textures that vary in hatching size and different sets of textures encoding the brightness of the shading. For dark regions a cross-hatched texture was used, and for bright regions only a few lines were drawn. Finally, these textures were projected on the surface. In contrast, Zander et al. [233] proposed to use geometrical lines as streamlines. The lines are then individually propagated on the surface along the principal curvature direction. Cylinders around the lines ensure a specific distance from one line to another. A dynamic approach was presented by Kim et al. [99]. They determined the principal curvature direction on the GPU and aligned hatching textures along this curvature direction. With their method, a hatching approach was introduced that can be applied on animated surfaces in a frame-coherent manner. A line drawing technique that combines the advantages of feature lines and hatching was presented by Lawonn et al. [121]. They determined feature regions and the contour margin, which are the starting point for streamline propagation. Later, they improved their approach such that animated surfaces can be illustrated with hatching lines as well [127]. In contrast to our method, they used a noise texture to generate the hatching, which may result in blurry lines.

BIO-MEDICAL APPLICATIONS Illustrative techniques, that also include line drawings, have been successfully applied to bio-medical data in various contexts. In the biology domain, Tarini et al. [214] presented edge cueing techniques for molecular visualization. Using depth-aware halos, depth-revealing contour lines, and intersection-revealing contour lines, they enhanced real-time visualization of molecules. Weber et al. [229] developed the ProteinShader application, which renders macromolecules using illustrative techniques. They employ real-time half-toning, bend textures and edge-line generation algorithms to visualize proteins, but in contrast to our approach, rely

on texture mapping to attain their results. Parulek et al. [166] utilize level-of-detail to enable the interactive rendering of large molecules. They apply a seamless transition from the display of the solvent-excluded surface to sphere rendering. More closely related to the work presented here, Zwan et al. [238] presented illustrative molecular visualization with continuous abstraction. The molecules were visualized using various levels from a space filling representation to abstract ribbons, specifically designed for molecular data.

In the medical domain, Interrante et al. [84] enhanced transparent skin surfaces with the aforementioned *ridge and valley* lines for radiation therapy treatment planning. Dong et al. [42] presented NPR techniques for segmented volumetric medical data. They generate silhouette points and strokes in order to provide volumetric hatching, but only employ this to generate static images. Tietjen et al. [217] combined silhouettes, surface, and volume rendering in a scene-graph-based application aiming at surgical education and planning. Their method successfully integrated strokes with surface and volume rendering, but did not include hatching and required complex interaction for adjustments. Ritter et al. [184] presented work on real-time illustration of vascular structures. They spatially accentuated vessels using hatching, distance-encoded surface and shadow illustrations. Based on this work, Lawonn et al. [122] proposed a combination of supporting lines, view-aligned quads, hatching and illustrative shadows to improve depth assessment of complex 3D vascular models.

Jainek et al. [85] combined volume and surface rendering to visualize anatomical and functional brain data using illustrative techniques. Gasteiger et al. [51] proposed a texture-based method to hatch anatomical structures derived from clinical volume datasets. They apply curvature-based hatching by incorporating model-based preferential directions of the underlying anatomy. Chu et al. [30] presented perception-aware depth cueing for illustrative vascular visualization, using depth-aware silhouettes, color-coded shading strokes and line-culling highlights. Svakhine et al. [210] depth enhanced medical volume visualization with artistic styles for outlining features and conveying depth information. Hansen et al. [64] visualized 3D planning models for augmented reality in liver surgery using illustrative techniques. In their work, they combine distance-encoding silhouettes and surfaces with procedural textures for intra-operative visualization. Svetachov et al. [211] illustrated brain fiber tracts from DTI data in an interactive application using interactive slice-based hatching. Born et al. [20] proposed illustrative visualization of cardiac and aortic blood flow from 4D MRI acquisitions.

In contrast to previous work, our method is the only method that supports a seamless interactive transitions between different levels of abstraction by integrating several illustrative rendering techniques,

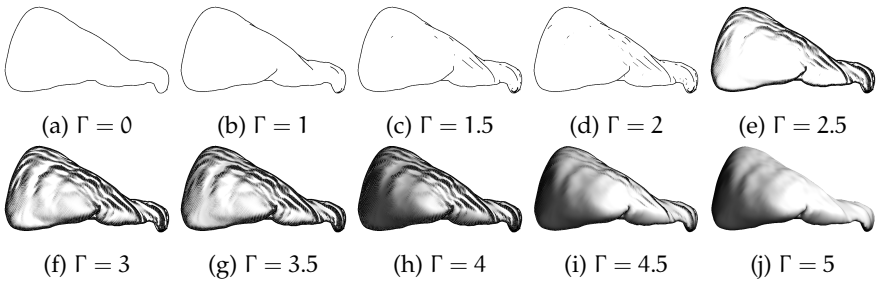
without relying on textures or assumptions about the underlying models.

2.3 METHOD

Our method, Sline, provides a smooth transition from silhouettes, to more detailed illustrative styles, to Phong shading for surface meshes. In this way, we achieve a transition from a very abstract representation along several levels of abstraction to a non-illustrative visualization. For every mesh $\mathcal{M}$ in the scene an individual rendering style can be chosen. To attain a step-wise decrease in illustrative levels of abstraction, we employ the following techniques in order:

1. (a) Silhouettes and (b) Contours (Section 2.3.2)
2. Suggestive Contours (Section 2.3.3)
3. Hatching (Section 2.3.4)
4. Phong Shading

An overview illustration of these stages is given in Figure 2.2.



(a) $\Gamma = 0$ (b) $\Gamma = 1$ (c) $\Gamma = 1.5$ (d) $\Gamma = 2$ (e) $\Gamma = 2.5$

(f) $\Gamma = 3$ (g) $\Gamma = 3.5$ (h) $\Gamma = 4$ (i) $\Gamma = 4.5$ (j) $\Gamma = 5$

Figure 2.2: The Sline method applied to a surface model of a clinical CT scan of the liver. Sline provides a seamless transition between various levels of abstraction, from silhouettes to contours, to suggestive contours (top row), to hatching, and finally to Phong Shading (bottom row) with intermediate states. The user parameter Γ (see Section 2.3.5) is given for each state.

The next sections describe the preprocessing, implementation of the algorithms, and the parameterization of the transition between consecutive rendering steps respectively.

2.3.1 *Preprocessing*

Silhouettes and contours do not require any preprocessing. The suggestive contours as in [40] and our hatching approach require the computation of the radial surface curvature $\kappa_{\mathbf{w}}(\mathbf{p})$ and its directional derivative $D_{\mathbf{w}}\kappa_{\mathbf{w}}$. Here, $\mathbf{p} \in \mathcal{M}$ is a point on the mesh $\mathcal{M}$ and $\mathbf{w}$ is the projection of the view vector $\mathbf{v}$ on the tangent plane at $\mathbf{p}$. Note that $\mathbf{v}$

points from the camera towards $\mathcal{M}$. The scalar field $\kappa_{\mathbf{w}}$ is computed based on the light gradient vector field $\mathbf{k}$. We compute $\mathbf{k}$ and $\kappa_{\mathbf{w}}$ as in the technique proposed by Lawonn et al.[127] and use the method described by Judd et al.[91] to compute $D_w\kappa_{\mathbf{w}}$. Finally, we define a feature size $\mathcal{F}$, that will be used to adjust parameters that depend on the world size of a mesh. For example, the curvature on the surface of a large sphere is smaller than that on a small sphere. Thus, we set $\mathcal{F}$ to the bounding sphere radius of $\mathcal{M}$.

2.3.2 *Silhouettes and Contours*

Silhouettes describe the outline of an object, while contours are found where the surface normal of an object is orthogonal to the view vector. Our silhouettes and contours use the same algorithm, which is similar to the object space algorithm described by Hertzmann [74]: We find zero crossings on the triangle edges where $\langle \hat{\mathbf{n}}, \mathbf{v} \rangle = 0$ holds, with $\langle ., . \rangle$ defining the dot product. Here, $\hat{\mathbf{n}}$ denotes the interpolated normal along the edge between two vertices. If we find zero crossings on two edges of a triangle, we create a contour patch with four vertices - two for each of the two edges, given by:

$$\hat{\mathbf{p}}_0; \;\; \hat{\mathbf{p}}_1 = \hat{\mathbf{p}}_0 + a \cdot \hat{\mathbf{n}} \cdot \mathcal{F}, \tag{2.1}$$

where $\hat{\mathbf{p}}_0$ is the location of the zero crossing on the edge with corresponding $\hat{\mathbf{n}}$ and a controls the width of the contour. If only the silhouette is required, the inner contour patches are discarded using a stencil test. We render $\mathcal{M}$ into the stencil buffer and then render the contours with the stencil test enabled, allowing the contours to be drawn only where $\mathcal{M}$ has not been drawn into the stencil buffer. To avoid terminology confusion between contours and suggestive contours, we will refer to the contours described in this section as *actual contours*.

2.3.3 *Suggestive Contours*

Suggestive contours, proposed by De Carlo et al. [40], can be understood as an elongation of actual contours to support shape perception. Suggestive contours extend actual contours, as these represent contours in nearby camera locations. If we look at $\mathcal{M}$ from location A and see a suggestive contour, this contour might be an actual contour seen from location B, where B is close to A. In order to determine the suggestive contours, $\kappa_{\mathbf{w}}$ and $D_{\mathbf{w}}\kappa_{\mathbf{w}}$ (recall Section 2.3.1) are required. The algorithm checks for the following constraints, based on the work by De Carlo et al., to find suggestive contour patches:

1. Search for a zero crossing of $\kappa_{\mathbf{w}}$ at point $\hat{\mathbf{p}}$ on each triangle edge.
2. Check for $0 < t_d < \frac{D_{\mathbf{w}}\kappa_{\mathbf{w}}(\hat{\mathbf{p}})}{\|w\|}$.

3. Check for $0 < \theta_c < \operatorname{acos}\left(\frac{-\langle \mathbf{n}(\hat{\mathbf{p}}), \mathbf{v}(\hat{\mathbf{p}})\rangle}{\|\mathbf{v}(\hat{\mathbf{p}})\|}\right)$.

Condition 2 uses t_d to ensure that $D_{\mathbf{w}}\kappa_{\mathbf{w}}$ is positive and to suppress regions with frequent changes of the sign of $D_{\mathbf{w}}\kappa_{\mathbf{w}}$ that might result from noise. The purpose of condition 3 is to discard suggestive contour lines that would appear in regions where the angle between $\mathbf{v}$ and $\mathbf{n}$ is too small. With increasing θ_c, suggestive contours are only visible the closer they are to actual contours.

If two or more edges of a triangle meet the conditions, suggestive contour patches are generated similarly to the method in Section 2.3.2. The suggestive contour patches are oriented orthogonally to their corresponding triangle surface. This has the effect that suggestive contour lines close to actual contours appear thicker, since they are oriented (almost) perpendicular to the view ray. Recall that actual contours appear where the surface normal and the view ray are orthogonal. In contrast, suggestive contours further away from actual contours will be drawn in a less thick style, because their orientation tilts away from the camera.

2.3.4 *Line Search based Hatching*

In NPR, hatching is a technique to cover a surface with a large number of line strokes that support the spatial impression of the surface. Line integral convolution (LIC) has been used to achieve hatching-like results [127], based on a LIC algorithm by Huang et al. [79]. The method by Huang et al. proposes to integrate the streamlines over a depth dependent noise texture, resulting in a very dense set of thin lines with a low black-white contrast. In our work, we are interested in more distinctive strokes with more contrast. A noise texture is not suitable for our needs, because a coarse noise texture is either blocky or blurred. Furthermore, we want to disregard texture coordinates with respect to medical data sets, because surface data generated from medical volume data does not usually feature texture coordinates. To address the goal of distinctive strokes, we suggest to map a coarse distribution of seed regions - called *smart seeds* - onto $\mathcal{M}$, from which the strokes can pick up their color. Since we are not integrating over a noise texture, we simply perform a line search, which we describe in Section 2.3.4.3. The following tasks arise from our requirements:

1. Map seeds onto $\mathcal{M}$ without texture coordinates (Section 2.3.4.1).
2. Define a function to generate seed regions from the seed locations (Section 2.3.4.2).
3. Compute a line search such that single strokes do not merge (Section 2.3.4.3).

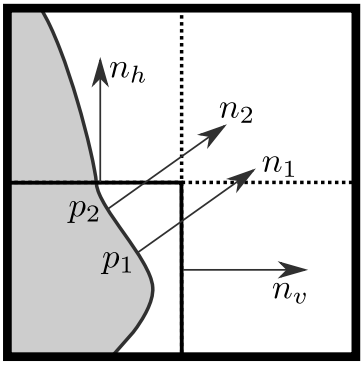


Figure 2.3: Projection example: points are projected along their normals.

Merging strokes occur when two strokes touch and result in a single, broadened stroke, which is undesirable. One stroke ending in the start of another stroke is still allowed, because that results in an extended stroke of equal width.

2.3.4.1 *Seed Mapping*

A solution for task 1 can be found in the polycube maps approach by Tarini et al. [215]. They subdivide the space around a mesh into an isotropic grid and practically inflate the mesh until it fills a specific set of grid cells. Figure 2.3 depicts how the mapping of surface points to the grid faces (or edges in 2D) is done. Each point is projected along its surface normal $\mathbf{n}$. For $\mathbf{p}_1$ we can see that $\langle \mathbf{n}_v, \mathbf{n}_1 \rangle > \langle \mathbf{n}_h, \mathbf{n}_1 \rangle$, so that $\mathbf{p}_1$ is projected onto the vertical edge. In the case of $\mathbf{p}_2$ we see that $\langle \mathbf{n}_v, \mathbf{n}_2 \rangle > \langle \mathbf{n}_h, \mathbf{n}_2 \rangle$ as well, so that p_2 is projected onto the vertical edge, even though the projection crosses the horizontal edge. This concept is used analogously for the faces of a 3D grid cell. In general, a point with surface normal $\mathbf{p}_n$ is projected on face f with normal f_n, if $\langle \mathbf{p}_n, f_n \rangle$ is the maximum of the dot products of all face normals and $\mathbf{p}_n$. The fact that surface points can be projected outside their original grid cell, results in less distortion of the mapped texture. We define the grid in local coordinates of $\mathcal{M}$ and use the cell faces to generate smart seeds. Therefore, we do not need any further processing for the mapping. For each cell face, we generate a texture as shown in the left of Figure 2.4.

2.3.4.2 *Smart Seeds*

Our goal is to generate thick lines that do not merge with other lines. To achieve this, our seeds are made up from three different regions, namely a *core region*, a *halo region* and a *contact region* (see Figure 2.4). The core region is used to produce the actual stroke and is always black, while the halo region can be adjusted in brightness to control the contrast among different strokes. The contact region will be used to prevent multiple strokes from merging. Strokes that run through

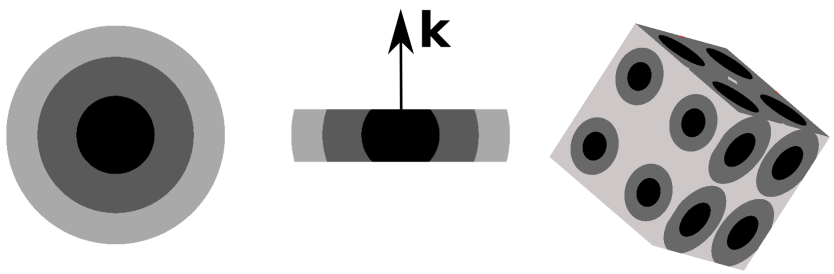


Figure 2.4: Core region (black), halo region (dark gray), contact region (light gray) (left), gradient of the seed center (**k**) (center), and larger radii on faces turning away from the camera (right).

the contact region of other seeds will be drawn in a lighter way, with the effect that different core strokes cannot appear in immediate contact. Due to our implementation of the line search (see Section 2.3.4.3), we trim the seed in the direction of the local light gradient **k** to make all regions accessible from outside (see Figure 2.4, center). Additionally, the radius of a seed is increased for faces that turn away from the view, to compensate for perspective distortion (see Figure 2.4, right and Eq. 2.3). We do this because small, perspectively clinched seeds contain less fragments and are thus more likely to be discarded by the rasterizer.

The computation of the smart seeds proceeds dependent on their local **k**. A screen space representation of Mesh $\mathcal{M}$ is rendered into a preparation texture TEX_p. In the fragment shader, each fragment's 3D location **p** is projected based on their normal as described in Section 2.3.4.1 and we get the projected point $\mathbf{p}'$. $\mathbf{k}(\mathbf{p})$ is projected onto the corresponding grid cell face and yields the normalized vector $\mathbf{k}'(\mathbf{p})$. Let the center of the grid cell face be **c**, the size of the cell be c_s and the seed radius be $s_r = \frac{1}{4}c_s$. c_s should be chosen dependent on $\mathcal{F}$ and the distance of $\mathcal{M}$ to the camera.

Dependent on its location, $\mathbf{p}'$ is classified as a seed region $C \in \{\texttt{Core, Halo, Contact}\}$. The boundaries of the regions are given by the distance $\|\mathbf{p}' - \mathbf{c}\|$ at $\frac{1}{2}s_r$, s_r and βs_r, where $\beta \geqslant 1$ is a factor to determine the contact region size, which is set to $\beta = 1.8$ in this work. The trimming of the seed is done using:

$$\begin{aligned} \mathbf{q}_1 &= \mathbf{c} + \frac{s_r \cdot \mathbf{k}'(\mathbf{p})}{4} \\ \mathbf{q}_2 &= \mathbf{c} - \frac{s_r \cdot \mathbf{k}'(\mathbf{p})}{4} \\ d_1 &= \langle \mathbf{k}'(\mathbf{p}), (\mathbf{q}_1 - \mathbf{p}') \rangle \\ d_2 &= \langle \mathbf{k}'(\mathbf{p}), (\mathbf{q}_2 - \mathbf{p}') \rangle \end{aligned} \tag{2.2}$$

If d_1 and d_2 have the same sign, the fragment at **p** is invalid. Using $\mathbf{k}'(\mathbf{p})$ instead of $\mathbf{k}'(\mathbf{c})$ allows the trimmed seed to adapt to the

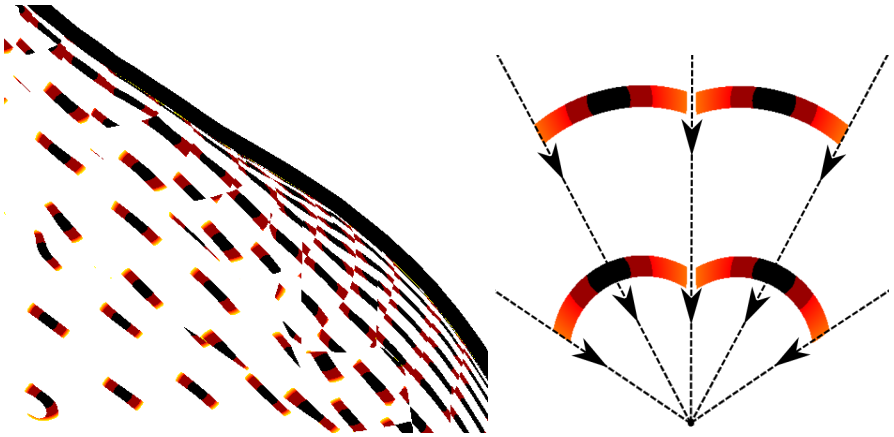


Figure 2.5: The seed trimming is calculated per fragment, based on the fragment's projected light gradient $\mathbf{k}'$. This aligns the valid seed region with the vector field (right). Valid regions are colored based on their properties (black to red).

surrounding vector field $\mathbf{k}$ (see Figure 2.5, right). Seeds on faces that turn away from the viewer have their radius increased by

$$s_{r_m} = s_r(2 - |\langle \mathbf{v}, \mathbf{n} \rangle|), \tag{2.3}$$

where s_{r_m} is the modified seed radius. Figure 2.5 (left) shows a section of the resulting texture TEX_p, which includes the contour of the mesh and will be used as input for our line search computation. The last step is to encode seed region properties in the color channel of each fragments output. The red (R) color channel stores the stroke color, which is set to 0 for the core region and some value $\in [0, 1[$ for the halo region (we use 0.6 for our figures). For the contact region we set R to 1, since 1 is used as the background color, e.g., no stroke should be drawn from here. To prevent different strokes from merging we assign a permeability factor ρ to each fragment and store it in the green (G) color channel. Core and halo regions are set to zero permeability, to stop any merging strokes. For the contact region we set

$$\rho = \frac{\|\mathbf{p} - \mathbf{c}'\| - s_r}{s_r \cdot \beta - s_r} \tag{2.4}$$

to produce a smooth gradient from outside the contact region to the edge of the halo region. The next section will describe the application of TEX_p.

2.3.4.3 *Line Search*

For any starting point on $\mathcal{M}$ our line search algorithm follows the underlying vector field $\mathbf{k}$ in positive and negative direction, until it hits a smart seed. The color for $\mathcal{M}$ at that starting point is determined by the distance traveled from the starting point to the smart seed

and by the seed region that has been hit. The computation is done in the fragment shader. Each fragment has access to a texture TEX_v, containing the view projected vector field $\mathbf{k}$ and TEX_p. In contrast to common LIC algorithms we follow the vector field in both, its forward and backward direction. Another difference is that we do not integrate over the color along a line. Instead, we search for the first occurrence of a seed region to stop the iteration. Thus we use a line search method, instead of LIC. Let $\mathbf{x}$ be the position of a fragment in screen space, then $TEX_p(\mathbf{x})_{\{R,G\}}$ gives access to the texture's color channels and $TEX_v(\mathbf{x})$ gives access to the vector field at position $\mathbf{x}$. Let $d \in [-1, 1]$ define the positive or negative direction of the iteration, i be the number of iterations, τ be a dampening factor that accumulates the loss of permeability when iterating through a seed's contact region with τ_w controlling the impact of τ and λ be the step size of the iteration. $\mathcal{S}$ is the aggregate of all smart seed `Core` and `Halo` pixel locations except for pixels of the smart seed covering $\mathbf{x}$, when $\mathbf{x}$ is used as the starting position of our line search. Our algorithm to find a fragment's color in one direction of the vector field is then given by Algorithm 1.

Algorithm 1 Line search algorithm

1: $i_d = 0$;
2: $\mathbf{x}' = \mathbf{x}$
3: $\tau = 1$
4: **while** $\mathbf{x}' \notin \mathcal{S} \wedge \mathbf{x}' \neq$ contour **do**
5: $\quad i_d++$
6: $\quad \mathbf{x}' = \mathbf{x}' + \mathbf{d}\ \lambda\ TEX_v(\mathbf{x}')$
7: $\quad \tau = \tau\ (1 - \tau_w\ (1 - TEX_p(\mathbf{x}')_G)$
$\mathbf{R_d} = \tau\ TEX_p(\mathbf{x}')_R + (1 - \tau)$

Algorithm 1 runs for $d = -1$ and $d = 1$ and we get the result R_d with the number of required iterations i_d. The final color for the fragment is computed as an interpolation of both results:

$$R = \frac{R_{-1}\ i_1 + R_1\ i_{-1}}{i_1 + i_{-1}} \tag{2.5}$$

The interpolation yields a smooth transition between core and halo regions of extendingextending strokes. Figure 2.6 (right) shows the result of this process. It can be observed that our method is capable of generating thick strokes with good contrast.

We do not want to draw hatching lines over the full mesh, but rather only in regions where they support the (suggestive) contour lines. A solution for this problem is presented in the next section.

2.3.4.4 *Blending*

Our intention is to show the hatching lines only in regions where they support the contour and suggestive contour lines of the model. The

contours basically rely on the light gradient direction $\mathbf{k}$ and on $\langle \mathbf{n}, \mathbf{v} \rangle$, while actual contours are restricted to both vectors being orthogonal and the suggestive contours allow the threshold $„_c$. Thus, we want to find a function that takes $\mathbf{n}$, $\mathbf{v}$, $\mathbf{k}$ and θ_c and outputs a value $\mathcal{O}_h \in [0, 1]$ that would be used as opacity for the hatching strokes. We define $\mathcal{O}_h$ as follows:

$$\begin{aligned} w_{\mathbf{k}} &= \langle \mathbf{k}, \mathbf{v} \rangle \\ w_a &= \mathrm{clamp}_{[0,1]} \left(\frac{\mathrm{acos}(-\langle \mathbf{n}, \mathbf{v} \rangle) - \theta_c}{\frac{1}{2}\pi - \theta_c} \right) \\ \mathcal{O}_h &= \max(w_a \cdot w_{\mathbf{k}} \cdot \mathcal{F}, 0) \end{aligned} \tag{2.6}$$

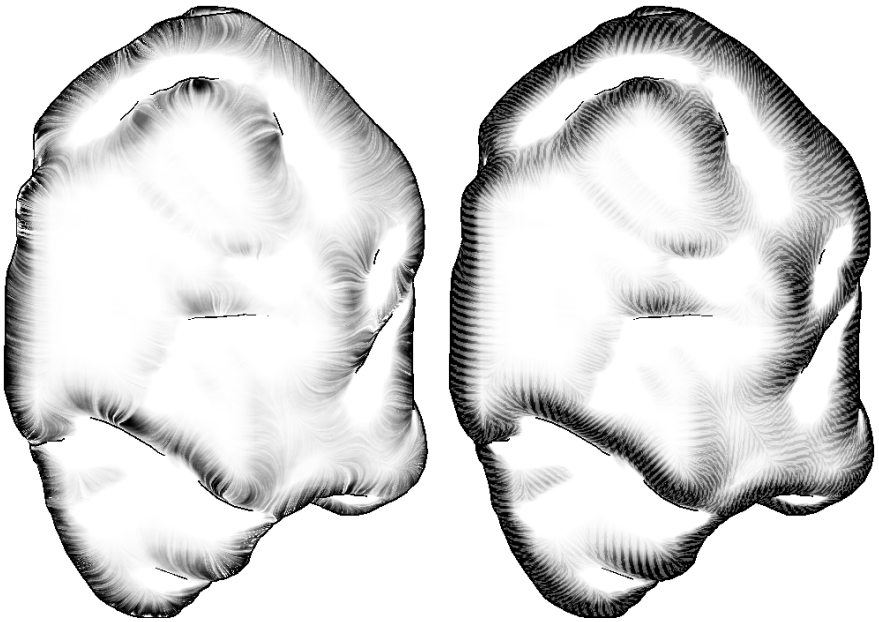
As $\mathrm{clamp}_{[l,u]}(...)$, we define a mapping to $[l, u]$, where values $< l$ or $> u$ are set to l or u respectively. $w_{\mathbf{k}}$ weighs the light gradient in dependence of the view. w_a is a mapping of valid angles $\in [\theta_c, \frac{1}{2}\pi]$ to $[0, 1]$.

We multiply all of these factors. Thus, either the viewing angle or the projected $\mathbf{k}$ can be responsible for eliminating hatching lines. As we disregard negative results, only regions where $\mathbf{k}$ points away from the viewer can have hatching lines, e.g., where curvature increases with distance to the camera. Note that $\mathbf{k}$ is not normalized to take the curvature magnitude into account. See Figures 2.2f and 2.6 (right) for the final result of our hatching method.

We can interpret the blending as a hint to where suggestive contours might appear in nearby camera positions and that it relates to suggestive contours in the same way as suggestive contours relate to actual contours. If we rotate the camera such that a suggestive contour line appears where hatching was visible in the previous camera position, the line becomes more distinct and will finally transition into an actual contour. Rotating into the opposite direction leads the suggestive contour line to fade away (see Figure 2.7).

2.3.4.5 *Robustness*

The visual output of the presented hatching method mainly depends on the grid cell size (or seed radius) and the brightness of the halo region. The grid is defined in model space, so if we zoom out the seeds shrink in screen space. This will ultimately result in heavy flickering when moving the camera, as soon as the seeds are smaller than a pixel. In that case, seeds may or may not appear in TEX_p and so will the hatching lines that depend on them. A simple approach to reduce this kind of artifact is to increase the grid cell size with increasing distance of the camera to the object. Consequently, the hatching strokes of distant objects are bigger and fewer, in relation to the object's size. As we zoom in, the hatching becomes more detailed. Another factor that can reduce flickering is the color of the halo region. Hatching strokes of the lighter halo region color are less salient and if these



(a) LIC (b) Line Search

Figure 2.6: Comparison of the noise texture based LIC (a) and our method (b) applied to a tumor model. While the LIC method produces blurry lines, our method yields characteristic strokes.

pop in and out, the flickering is less severe.
The curvature vector field is view dependent, because we compute the light gradient with the light source positioned in the camera center. Thus, it changes smoothly as the camera moves. This is true, except for regions where the surface normal is (almost) perpendicular to the view vector. Here, the direction of $\mathbf{k}$ can change rapidly. This also affects the hatching lines and causes visual artifacts. Using Eq. 2.6, these regions are discarded, since $\mathcal{O}_h = 0$ at these locations if θ_c is large enough.

2.3.5 *Transition Parameterization*

The rendering techniques described in Section 2.3 depend on one or more parameters. For simplicity and intuitive interaction, without requiring user knowledge about the underlying parameters, we map a single parameter to all feasible input of one shading technique. Feasibility of a parameter to be mapped is given, if that parameter can support the seamless transition among our shader stages. We will now define the mapping of our transition parameters for the different stages defined at the beginning of this section.
The user sets a parameter $\Gamma \in [0, 5]$ to define the transition level. The

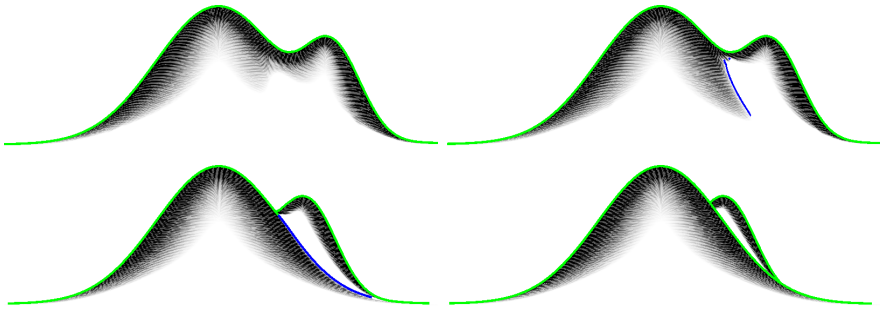


Figure 2.7: From top left to bottom right, the camera is rotated to the left. The suggestive contour (blue) appears nearby a hatched area and then moves along the hatched area and becomes an actual contour (green).

parameters $\mathcal{T}_{\{S,C,SC,H,P\}} \in [0,1]$ describe the transition from one rendering style to the next and depend on Γ (recall Figure 2.2).

$$\begin{aligned} \text{Silhouettes: } \mathcal{T}_S &= 1 \\ \text{Contours: } \mathcal{T}_C &= \mathrm{clamp}_{[0,1]}(\Gamma) \\ \text{Suggestive contours: } \mathcal{T}_{SC} &= \mathrm{clamp}_{[0,1]}(\Gamma - 1) \\ \text{Hatching: } \mathcal{T}_H &= \mathrm{clamp}_{[0,1]}(\Gamma - 2) \\ \text{Phong shading: } \mathcal{T}_P &= \mathrm{clamp}_{[0,1]}(\Gamma - 3) \end{aligned} \tag{2.7}$$

That way, the stage in transition is modified by Γ, while all previous stages are in a fixed, visible state.

SILHOUETTES AND CONTOURS: According to Eq. 2.1, we set the width a:

$$a = \mathit{width} \cdot \mathcal{T}_{\{S,C\}}. \tag{2.8}$$

This will fade in contours from 0 to width, where width is a predefined parameter controlling the maximal width of the contour and is set to 0.006 for our examples.
Note that a is scaled by $\mathcal{F}$ in Eq. 2.1, so that width defines the contour width as a fraction of the bounding sphere of $\mathcal{M}$.

SUGGESTIVE CONTOURS: According to Condition 2 and 3 in Section 2.3.3, we set t_d and θ_c:

$$\begin{aligned} t_d &= (1 - \mathcal{T}_{SC}) t_{d\,min} + t_{d\,min} \\ \theta_c &= \frac{\pi}{2} - (\frac{\pi}{2} - \theta_{c\,min}) \mathcal{T}_{SC} \end{aligned} \tag{2.9}$$

where $t_{d\,min}$ and $\theta_{c\,min}$ are predefined minimal thresholds. This will cause suggestive contours to grow from actual contours, by decreasing θ_c and t_d from high values to their respective minimal values.

HATCHING: According to Eq. 2.6, we set θ_c:

$$\theta_c = \frac{1}{2}\pi(1 - \mathcal{T}_H) \tag{2.10}$$

This will expand the regions around contours and suggestive contours where hatching will be enabled.

PHONG SHADING: We want to fade in Phong shading from locations where hatching is not visible and define the Phong opacity $\mathcal{O}_p$ as:

$$\begin{aligned} \theta_c(\mathcal{T}_P) &= \frac{1}{2}\pi(1 - \mathcal{T}_P) \\ w_a(\mathcal{T}_P) &= \mathrm{clamp}_{[0,1]}\left(\frac{\mathrm{acos}(-\langle \mathbf{n}, \mathbf{v}\rangle) - \theta_c(\mathcal{T}_P)}{\frac{1}{2}\pi - \theta_c(\mathcal{T}_P)}\right) \\ w_{\mathbf{k}}(\mathcal{T}_P) &= -\langle \mathbf{k}, \mathbf{v}\rangle(1 - \mathcal{T}_P) - \mathcal{T}_P \\ \mathcal{O}_p &= \mathrm{clamp}_{[0,1]}(w_a(\mathcal{T}_P)\, w_{\mathbf{k}}(\mathcal{T}_P)) \end{aligned} \tag{2.11}$$

$[\theta_c(\mathcal{T}_P), \frac{\pi}{2}]$ defines the range of valid angles between $\mathbf{v}$ and $\mathbf{n}$. $w_a(\mathcal{T}_P)$ will always be positive for valid angles and zero for invalid angles. $w_{\mathbf{k}}(\mathcal{T}_P)$ is only positive if $\mathbf{k}$ points towards the camera, thus $\mathcal{O}_p$ can only be positive in that case. $\theta_c(\mathcal{T}_P)$ and $w_{\mathbf{k}}(\mathcal{T}_P)$ practically result in the opposite of what we defined for valid hatching regions (recall Eq. 2.6) and Phong shading will first be visible in regions without hatching. With $\Gamma = 4$ we get a combined result of all shading techniques (see Figure 2.2h). A final transition (with $\Gamma > 4$) applies $\mathcal{T}_{\{S,C,SC,H,P\}} = 1 - \mathrm{clamp}_{[0,1]}(\Gamma - 4)$ equally to all stages and we modify $\mathcal{O}_p$ for the Phong shading as

$$\mathcal{O}_{p_m} = \mathcal{T}_P\ \mathcal{O}_p + (1 - \mathcal{T}_P), \tag{2.12}$$

where $\mathcal{O}_{p_m}$ is the modified Phong opacity. At the highest parameter setting, all illustrative shading will fade away an leave simple Phong shading as the final stage.

2.4 RESULTS

To evaluate the utility of our application, we had a semi-structured interview with five domain experts, based on three case studies shown in demo videos. We chose to use videos to reach as many domain experts as possible, including ones that are not in the same physical location as us, and to avoid having these experts needing to install our software. In this way, all users have the same experience and impression of the presented technique. The five experts consisted of two medical doctors, two pharmacists, and one biologist. For the case studies, we used anatomical data from the abdomen and pelvis (see Figure 2.8 and 2.9), as well as a protein dataset (see Figure 2.1 (left)).

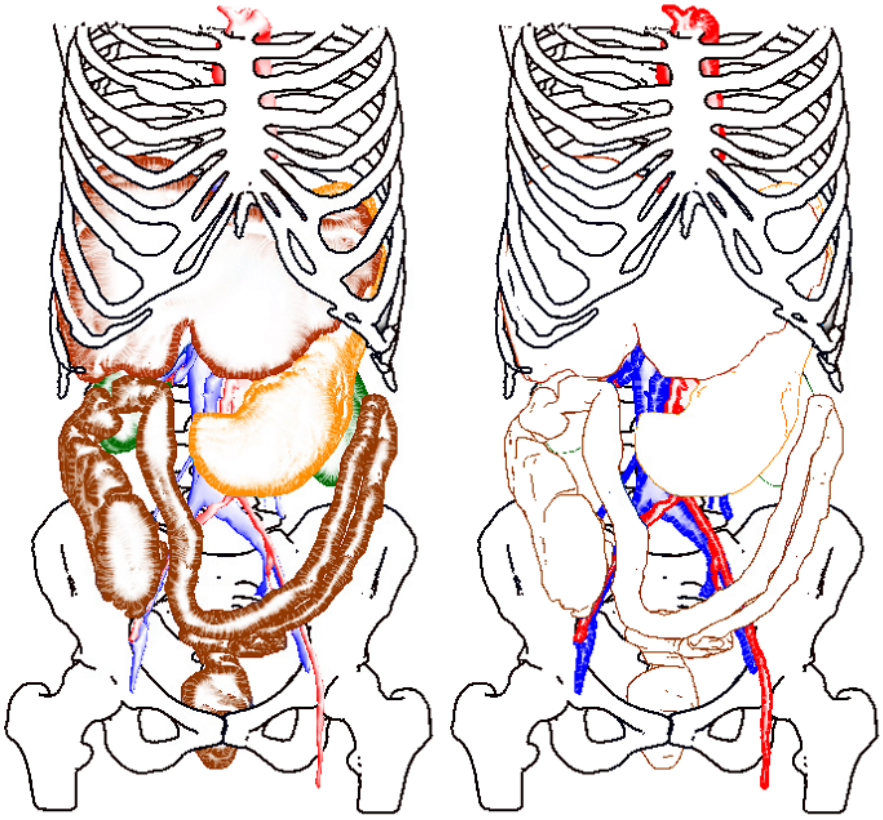


Figure 2.8: Abdomen dataset: by setting the rendering styles of the structures, users can shift the focus from the internal organs (left) to the vascular structures (right).

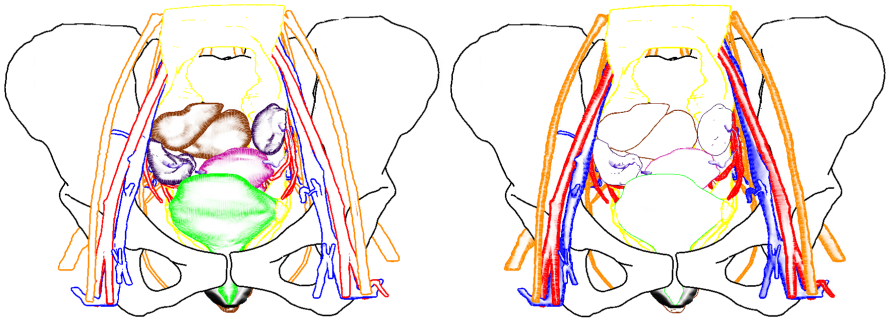


Figure 2.9: Pelvis dataset: hatching the organs as focus with other anatomy as context (left), and branching structures as focus (right).

The abdomen dataset features surface reconstructions based on segmentations from the Visible Human cryosection and CT scan. The pelvis dataset is the Virtual Surgical Pelvis atlas [106], which is based on cryosectional data from the Visible Korean Female and insights gained from histology studies. The protein dataset features PDB-ID 5H9N from the protein data bank (PDB), and contains the crystal structure of LTBP1 Y114A mutant in complex with leukotriene C4.

MEDICAL EXPERTS: After seeing the pelvis case study, the first expert envisions using Sline for treatment planning of oncologic surgical procedures, e.g., cervix or prostate carcinoma, since these procedures are usually accompanied by interdisciplinary communication of different domain experts. The same applies to patient-doctor communication in such cases, as well as for teaching purposes to educate students in the spatial relations of the anatomy. According to her, patient-doctor communication is a very important field and especially operations are very difficult to explain, because there are no information sheets. She stated that patients do appreciate when physicians explain complicated operations to them, and Sline can definitely help there. A function to export user defined 3D scenes and annotations, with certain structures highlighted, would improve the usability of Sline for communication with students.

The second expert said that the tool looks very intuitive, quick and easy to use. She estimates she could set up a visualization for the whole Virtual Surgical Pelvis dataset, featuring 28 anatomical structures, within ten minutes. She appreciates the schematic and simplified representations that can be made using Sline, and sees potential for use in treatment planning, education and doctor-patient communication. Furthermore, she proposes to use it in interdisciplinary communication, in presentations to peers for instance. She states that the combination of rendering styles and colors can really help focus attention of the viewer, but care must be taken to pick the right colors for the right structures.

PHARMACY/BIOLOGY EXPERTS: Experts from the pharmaceutical domain see a potential to quickly highlight different protein and ligand configurations for comparison. This requires the overlay of multiple 3D objects, to make structural differences visible. With our technique, different structures can be quickly set to focus or context, e.g., to flexibly support a presentation in interdisciplinary communication or for teaching purposes. The size and shape of docking cavities in the protein, where ligands can bind to, are also important information and would preferably be highlighted separately, which is not possible at this time, since we set Γ per complete mesh.
The biology expert stated that Sline helped him to remedy visual overload by reducing the amount of detail for uninteresting parts of

the model, while still keeping enough details to see the shape of the context. By using a medium level of abstraction in combination with color, interesting objects can be shown in a clear way. The color also provides a highlight, which makes it easier to focus on certain objects. He states the main advantage of Sline is the simplicity and ease of use, and having just one parameter to obtain the desired level of abstraction. He commented that the hatching-like effect seemed to be a bit too strong, since it creates darker images than the local lighting at the end. An additional, hierarchical list of groups of available structures contained in the scene would help to select objects of interest even quicker. Besides the current application domain, he believes Sline could also be useful for preparation of illustrative renderings for black-and-white print.

CASE STUDY CONCLUSION: All experts agreed that Sline can help focus the viewer's attention on important structures, and that the ordering of decreasing abstraction levels is intuitive. They also affirm that creating a meaningful visualization, based on the available options, is intuitive, easy and fast, which is credited to the single parameter concept.

2.5 DISCUSSION

Our new hatching algorithm is inspired by LIC, but is capable of producing more characteristic strokes, as illustrated in Figure 2.6. To be consistent with the suggestive contours, it is based on the light gradient $\mathbf{k}$ and the radial curvature $\kappa_{\mathbf{w}}$. Since the light gradient changes if the camera position changes, the hatching technique has to be computed in real time. We achieved interactive frame rates for scenes of up to 800k faces on an NVIDIA GTX 580 graphics card using smart seeds and a simple line search. To address clinical data, the hatching does not require texture coordinates. Another positive property of our hatching is, that it relates to suggestive contours such as suggestive contours relate to actual contours (see Section 2.3.3). E.g. the hatching represents locations of suggestive contours in nearby camera locations. As Figure 2.10 shows, we can easily reduce the hatching to a stippling effect.

Our hatching concept has several limitations. In Section 2.3.4.5, we mentioned the flickering that we were not able to fully suppress. Also, Figure 2.5 reveals, that the simple seed mapping approach is not optimal. It can be observed, that some seeds are not drawn completely and are cut off at triangle edges. We assume that this is due to the ratio between grid cell size and the seed radius, which is not optimal. Another visual artifact appears when zooming in and out as we adjust the grid cell size. Doing so leads the seeds to move over the object, because they are bound to their grid cell. This effect could be

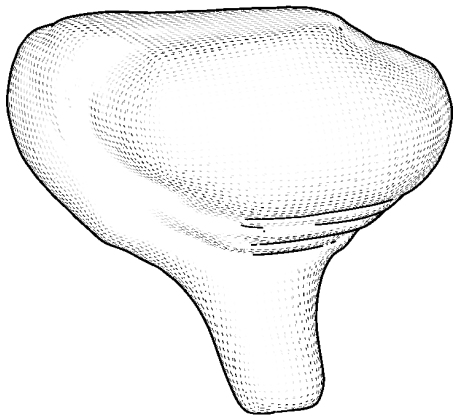


Figure 2.10: Stippling applied to a clinical MRI-based bladder model, using only one iteration of Algorithm 1.

eliminated, if we would allow multiple seeds per grid cell face. From a far distance, only a few large seeds should be visible. As the camera moves closer, the large seeds would shrink and new smaller seeds would fade in. We expect this to result in a seamless transition from a coarse to fine hatching.

One more drawback is illustrated in Figure 2.11. Hatching strokes do neither merge, nor do they extend each other. The result is a region of many short strokes that give the impression of a stippling technique. This happens due to the uniform grid and the radius that we use for the mapping of our seeds. In the presented case, a stroke can not penetrate the contact region of neighboring strokes and is prevented from merging with strokes further away. Contrarily, the reduction of the impact of the contact region might allow other strokes to merge. Imagine the seeds in Figure 2.11 were larger, then the strokes would probably merge in vertical direction and produce a thick, vertical hatching line. This would give a false impression of the underlying vector field. The issue might be addressed by a more sophisticated line search algorithm, which does not only consider single pixels on a line, but also pixel neighborhoods. We hope that such an approach will also reduce the flickering effect while rotating an object. Our feature size parameter $\mathcal{F}$ is based on the approximation of $\mathcal{M}$ as a sphere, which is feasible for compact structures. More filigree structures, such as vessels, would profit of a more precise approach, that incorporates the average surface curvature of $\mathcal{M}$, for example.

2.6 CONCLUSION AND FUTURE WORK

We have presented Sline, a technique for seamless interactive transitioning along various levels of abstraction, from silhouettes to Phong shading, for bio-medical surface data. The transition among the involved rendering states is reduced to a single parameter, to allow for

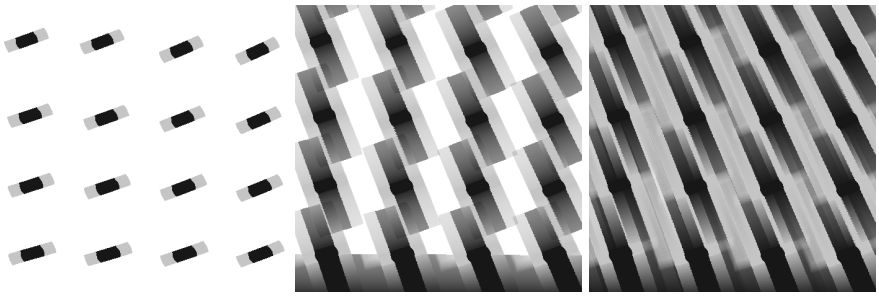


Figure 2.11: From left to right: Result of increasing number of iterations in Algorithm 1. Single strokes are prevented from merging with others, but also fail to extend other strokes.

simple interaction without any additional training for the user. For this, we defined a set of parameters that can be mapped to that single input parameter and thus modify the individual rendering techniques. By adjusting this single parameter, users can quickly and intuitively generate focus-and-context visualizations for bio-medical surface data.

As future work, we would like to explore smart visibility techniques to deal with occlusion, while still providing shape cues. In its current state, our framework only supports simple blending to suggest transparency of multiple objects. A proper use of order independent transparency, like the approximating algorithm by Vasilakis [225], could open up more areas of application and flexibility for our framework to deal with occlusion issues. For instance, the biology experts suggested to highlight cavities, which is done by a transparency mapping in the work by Borland [19]. It could be of advantage to apply this transparency mapping, or any other mapping of features, on our levels of abstraction, such that regions of interest are highlighted based on their functionality.
From the usability point of view, the simplicity of parameterization presented here, can be utilized to conduct a large scale evaluation on which rendering techniques users prefer to create their own focus and context scenes. Results from such an evaluation might be used to automate the creation of focus-and-context scenes. We plan to release Sline as an open source tool, so that it becomes freely available for people to use in their own bio-medical applications.

ACKNOWLEDGEMENTS This project was partly funded by the DFG: LA 3855/1-1 and HA 7819/1-1. We would like to thank the domain experts D. Komm, M. Krone, D. Imhof, P. Heimer and A. Kraima for their support.

3

REAL-TIME FIELD ALIGNED STRIPE PATTERNS

ABSTRACT In this paper, we present a parameterization technique that can be applied to surface meshes in real-time without time consuming preprocessing steps. The parameterization is suitable for the display of (un-)oriented patterns and texture patches, and to sample a surface in a periodic fashion. The method is inspired by existing work that solves a global optimization problem to generate a continuous stripe pattern on the surface, from which texture coordinates can be derived. We propose a local optimization approach that is suitable for parallel execution on the GPU, which drastically reduces computation time. With this, we achieve on-the-fly texturing of 3D, medium-sized (up to 70k vertices) surface meshes. The algorithm takes a tangent vector field as input and aligns the texture coordinates to it. Our technique achieves real-time parameterization of the surface meshes by employing a parallelizable local search algorithm that converges to a local minimum in a few iterations. The calculation in real-time allows for live parameter updates and determination of varying texture coordinates. Furthermore, the method can handle non-manifold meshes. The technique is useful in various applications, e.g., bio-medical visualization and flow visualization. We highlight our method's potential by providing usage scenarios for several applications.

3.1 INTRODUCTION

In surface visualization, there is often a need to visualize additional features of the data directly on the surface. If there is only one value that needs to be shown, color mapping is often employed to provide a qualitative impression of the value distribution over the surface. However, for multivariate data, the need can arise to visualize multiple values simultaneously, and simple color mapping will no longer suffice. Multiple views can be presented in such cases, but this requires mental integration for the viewer. Glyph-based or layering techniques are also able to convey multiple quantities, but may lead to clutter and occlusion [95]. To provide the user with an integrated view of multiple features, advanced visualization techniques such as illustrative visualization can be used to encode additional information. For such techniques, however, preprocessing is often required. This has the unfortunate side-effect that those techniques can no longer be employed to display dynamic changes, and there may be cases where preprocessing is undesirable or even impossible. Furthermore, when

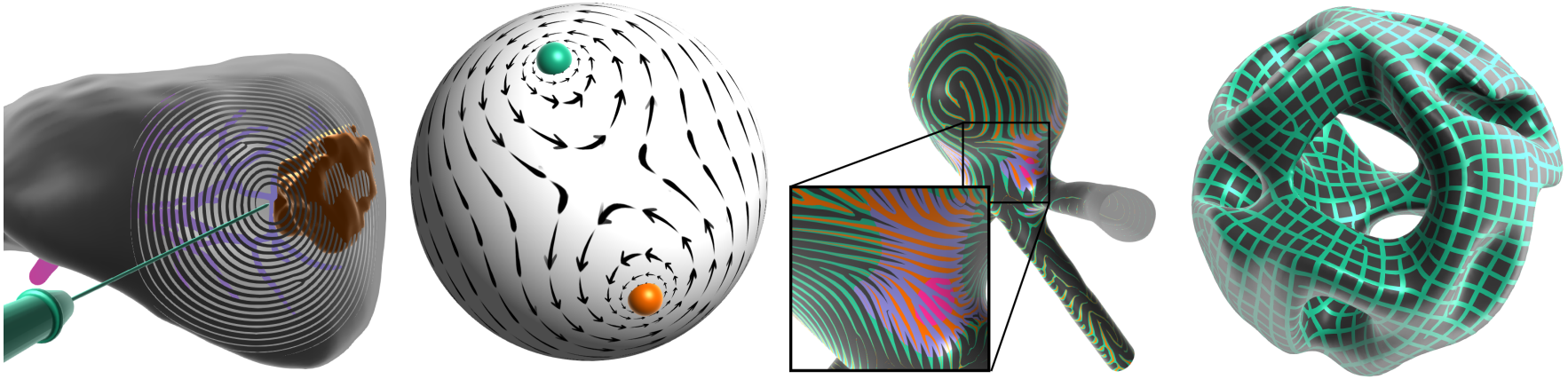


Figure 3.1: Our surface parameterization method is able to deal with dynamic changes on-the-fly and can be used in a range of real-time applications. From left to right: (1) selectively discarding fragments achieves a transparency effect, without affecting colors of underlying structures, (2) dynamic flow field visualization, with two spheres representing vortex cores, and arrow textures around them indicating flow direction, (3) visualization of a stress-tensor on aneurysm surface data [11, 87] (when zooming in, the frequency of the pattern increases to allow for more detail), (4) parameterization of the *Hyperball* with a 4-way rotational symmetry field as input.

relying on precalculation, it is not possible to update any parameters involved at run-time. Therefore, there is a need for a method that is able to provide parameterization of a surface mesh without preprocessing and that can be adjusted on-the-fly.

We propose a technique to parameterize a triangulated surface and generate a global stripe pattern on the surface, based on an underlying tangent vector field. If no vector input is available, principal curvature directions could be computed as a backup strategy. This is also possible in real-time as stated by Griffin et al. [57]. The resulting parameterization can then be used for different visualizations tasks. Existing methods [86, 103, 176] already address this kind of problem. However, these do at most focus on interactivity, while we aim for a real-time visualization, allowing dynamic input properties. Further, our problem formulation is suitable for an optimized reconstruction of the parameterization in the fragment shader. This is beneficial, e.g., if our method is used to generate local texture coordinates. To make our method suitable for real-time applications, we adapt existing approaches and aim for a local solution through local iterative optimization steps. The locality of our approach allows handling of non-manifold surfaces. Also, we can update visualizations and their parameters on-the-fly, for instance driven by dynamic vector fields, or reactive to scene changes resulting from interaction. With this, our main contributions are the following:

- We propose a technique to derive local texture coordinates from tangent vector-fields on a surface mesh, through local iterative optimizations.
- Our technique can be executed in real-time for medium-sized meshes, and thus can be used in visualization of both dynamic meshes, as well as dynamic parameter input.
- We demonstrate the potential of our technique in several usage scenarios from various domains, and compare the performance of our technique both quantitatively and qualitatively to reference methods.

We obtain periodic 1D texture coordinates based on a 1D parameterization aligned to an unoriented vector field. This can be employed for field visualization using a stripe pattern. The parameterization based on two orthogonal vector fields can be used to obtain periodic 2D texture coordinates. These can be used to visualize vector fields or arbitrary scalar properties using different textures or patterns, as we demonstrate in several examples.

3.2 RELATED WORK

In this section, we examine related work from a technical perspective, as well as from a visualization application perspective.

SURFACE PARAMETERIZATION TECHNIQUES Surface parameterization has been intensely researched for a long time [47]. Global parameterization plays an important role in global quad remeshing algorithms in order to find an optimal remeshing across the whole mesh. A survey on this topic is provided by Bommes et al. [16]. Such methods are usually complex to implement, run at most at interactive timings [43] and thus, are not applicable in real-time applications. Jakob et al. [86] proposed a method that relinquishes global optimization, yet is still able to create meshes that align with features on a global scale. This local approach makes their method parallelizable, which makes finding a solution faster by several orders of magnitude. Such techniques define, or get as input, a direction field on the surface, along which the parameterization is aligned. Proper generation of such direction fields is crucial to guarantee mesh quality for these methods. Design of these direction fields has emerged from the above requirements as an additional research area. Details can be found in the state of the art report by Vaxman et al. [227]. Our work uses as input an unoriented vector-field and does not address its further optimization. The methods mentioned so far generate vector fields, or at least require an optimized vector-field as input, and use them in successive steps. The design and visualization of direction fields is often closely coupled to allow for a visual feedback of applied changes [118]. The visualization is often done using line integral convolution (LIC) [27, 165]. However, LIC does only convey the ambiguous orientation of a vector direction $\mathbf{d} \sim -\mathbf{d}$ and cannot be used to display textures. Other methods, like the generation of texture coordinates, utilize vector-valued input to control texture orientation. Then, attention has to be paid to whether the vector field is oriented or non-oriented. Methods that take orientability into account can be used for a controlled display of orientable textures, but have to take care of visual seams [131, 171, 205], while methods that work on unoriented fields have to rely on symmetric textures [157, 221].

The most important prior art to the work presented here are the position field optimization of the Instant Field Aligned Meshes (IFAM) algorithm by Jakob et al. [86] and the technique for stripe pattern synthesis on surfaces (SPS) by Knöppel et al. [103]. The IFAM algorithm has introduced a local and parallel solution to global parameterization and the patterns that result from applying SPS are globally smooth and applicable for design and texture synthesis tasks. The interpolation scheme by Knöppel allows for a globally continuous pattern away from isolated singular points. Global continuity refers to the property that no jumps in the pattern can be found across the surface (i.e., no seams are visible). More precisely, if a piecewise continuous pattern is given and the pattern is based on a periodic function, the periodicity results in repetitive piecewise continuity across the surface, hence achieving global continuity. In contrast to SPS, our

technique finds a locally optimized solution through local iterative optimization steps, which makes it suitable for real-time applications without requiring any precalculation.

RELATED VISUALIZATION APPLICATIONS One of the potential application areas for our technique is to employ the generated stripe-patterns as an additional visual encoding channel for multivariate data visualization. Multivariate data is defined in the comprehensive survey by Fuchs and Hauser as information which has an attribute vector for each data item [49]. In the field of multivariate data visualization, Rocha et al. [185] recently proposed a real-time technique to map decals onto surfaces as a new way of representing multivariate data. The sets of images or patterns mapped to the surface are able to represent attributes of the data at the location they are mapped to, and can be used in combination with additional layered visualization elements. In contrast to their approach, we are able to handle dynamic flow patterns in addition to real-time texture coordinate synthesis, since we generate a globally continuous pattern. The work by Schroeder and Keefe [199] specifically caters to time-varying multivariate data visualization by providing an artist with an interface to sketch such visualizations. In their work, they allow artists to sketch illustrative elements that can be used as animated glyphs in a layered 2D visualization. However, their technique is focused on visualization design on a flat 2D surface. In earlier work by Kirby et al. [102], the potential of using illustrative techniques borrowing concepts from painting to visualize multivalued 2D flows was highlighted. Our technique is also able to generate illustrative strokes for flow, but extends to more complex 3D surfaces. Furthermore, we are able to animate these strokes to represent time-varying vector fields. Recent work by Roy et al. [192] use LIC to visualize the sheets of branched covering spaces. However, LIC is not suitable for expressing the unambiguous directionality of vector fields, and thus they require animations to express this aspect.

To the best of our knowledge, ours is the first work to use a globally smooth parameterization for visualization purposes, based on dynamic input data that can be updated in real-time. This concept, w.r.t. visualization purposes, is inspired by the work by Knöppel et al. [103], who generate a continuous *stripe pattern* on a surface, based on an input vector field. They also present details on the proper visualization of their parameterization results and, e.g., how to obtain texture coordinates from that. Their approach in turn is based on the method to generate a periodic global parameterization (PGP) as described by Ray et al. [176], who focus on re-meshing purposes. The stripe pattern algorithm introduces several changes in order to drastically improve the performance. Jakob et al. [86] were the first to translate the problem addressed by the above mentioned methods

to a formulation that allows a local and thus parallel execution of the optimization. However, their CPU implementation is suitable for interactive, but not for real-time performance. Furthermore, the frequency of their periodic pattern is limited by the mesh resolution.

We incorporate ideas and concepts of the above mentioned work and extend these with the goal to come up with an algorithm that allows for parameterization in real-time and is suitable for visualization purposes. We contrast the prior work in the way that we obtain coordinates for orientable textures, how these coordinates can be aligned with the underlying field on a pixel basis and we employ a convergence term for the optimization process. Furthermore, we show a range of application scenarios that can be seen as an inspirational basis for future visualizations tasks, based on dynamic field visualization.

3.3 METHOD

To obtain a surface parameterization, we aim to determine a globally smooth stripe pattern on the triangle mesh. The basic idea of the algorithm is to interpret each vertex on the mesh as a sample of an individual wave (i.e, a periodic function). This wave is described by an individual direction, which passes the vertex at a specific phase with a certain frequency. Hence, the input to our algorithm is a vector field that defines the wave directions and a scalar field that defines the wave frequencies. For a globally optimal solution all vertices can be interpreted as samples of the same wave as in Figure 3.2. The elongation of each sample on the wave can then be used to generate a periodic stripe pattern.

NOTATION For the remainder of this paper, we use the following notation. For a triangulated mesh $\mathcal{M}$, we denote V, E, F as its vertices, edges, and triangles respectively. If $(i, j) \in E$ with $i, j \in V$ then vertex i and j are connected. Similarly, $(i, j, k) \in F$ means that the vertices i, j, k form a triangle. Additionally, we use $\mathcal{N}(i)$ to describe the set of neighbors of vertex i. With $\mathbf{v}_i \in \mathbb{R}^3$, we denote the position of the vertex i in $\mathbb{R}^3$. Furthermore, we define $\mathcal{W}$ as the wave set, which consists of a set of wave directions $\mathcal{D}$, phases $\mathcal{P}$, and frequencies $\mathcal{F}$. Thus, every vertex i is assigned an individual wave $\mathbf{w}_i$ with normalized vector $\mathbf{d}_i \in \mathcal{D}$, which defines the direction of the wave, and a phase $\Phi_i \in \mathcal{P}$ with frequency $f_i \in \mathcal{F}$. Note that we consider the (normalized) direction vector $\mathbf{d}_i \in \mathcal{D}$ at vertex i as an equivalence class with $\mathbf{d}_i \sim -\mathbf{d}_i$. This means that, although this vector has a direction, the solution of a global stripe pattern is independent of this direction and is, therefore, non-oriented.

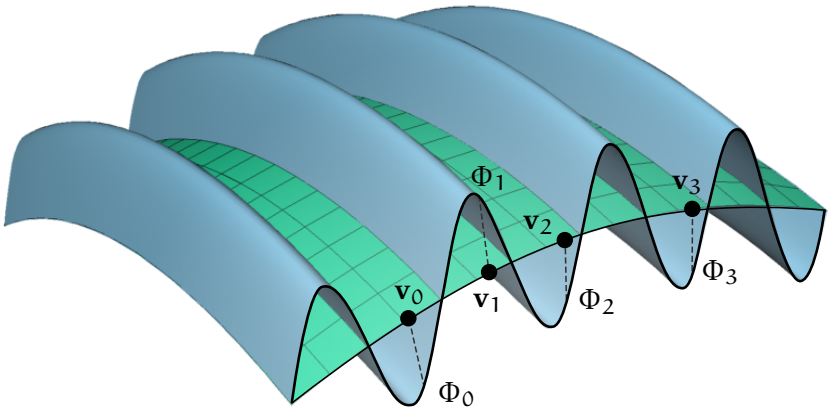


Figure 3.2: For an optimal stripe pattern, the vertex phases Φ_i along a surface align with one continuous wave, e.g., the vertex phases sample a single wave along the surface.

3.3.1 *Basics*

The phase shift Φ_{ji}, i.e., the signed difference of the phases from vertex j to i, is computed as follows:

$$\Phi_{ji} = 2\pi \cdot d_{ji} \cdot f_{ji}, \tag{3.1}$$

where d_{ji} is the aligned distance between $\mathbf{v}_i$ and $\mathbf{v}_j$ and the wave frequency is taken into account by the geometric mean, such that $f_{ij} = f_{ji} = \sqrt{f_i f_j}$. The aligned distance is the distance relative to a common reference point and is used to account for differences in the wave directions $\mathbf{d}_i$ and $\mathbf{d}_j$ (i.e., the divergence of the vector field).

Basically, we cannot directly compute a phase shift (or distance) between i and j, because both represent individual 1D waves along the surface as in Figure 3.3 (top, left). If we find an intersection C of the waves (Figure 3.3 top, right), we can regard C as the common source of the waves i and j. Thus, the phase shift is computed relative to the intersection point C. In the smooth setting, we would use the geodesic distance to C. If multiple intersection points along the surface exist, we take the closest one. Since we want to process a discrete mesh, a different attempt is made: As shown in Figure 3.3 (bottom), the direction $\mathbf{d}$ lies in the respective tangent plane. Consequently, if the tangent planes are not equal, the waves will likely not intersect in 3D space. This is why we create a plane that contains the points $\mathbf{v}_i$, $\mathbf{v}_j$ and $\mathbf{p} = 0.5(\mathbf{v}_i + \mathbf{d}_i + \mathbf{v}_j + \mathbf{d}_j)$ (Figure 3.4, left). Then, $\mathbf{d}_i^p$ and $\mathbf{d}_j^p$, which represent the projections of the respective directions into that

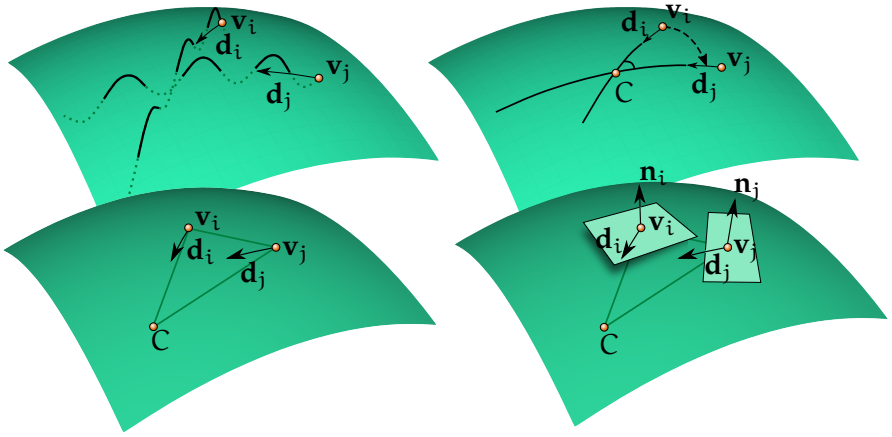


Figure 3.3: In the smooth setting, we can find an intersection C of two waves (top left), by the trace of $\mathbf{d}$ along the surface (top right). If the tangent planes of $\mathbf{v}_i$ and $\mathbf{v}_j$ are not the same, we cannot find an intersection of the wave traces in 3D, because $\mathbf{d}$ lies in the respective tangent plane (bottom).

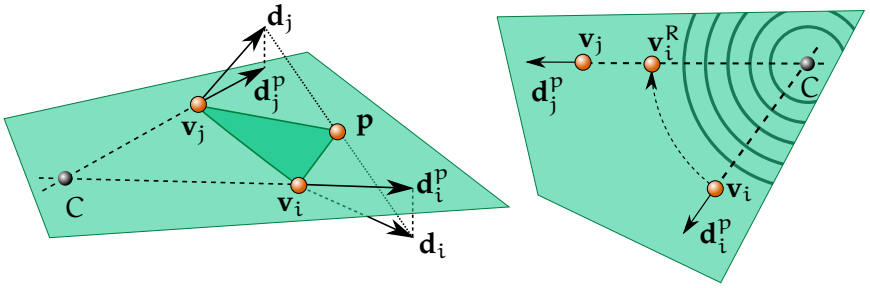


Figure 3.4: Computation of the wave intersection C (left). Rotation of $\mathbf{v}_i$ about C for alignment with $\mathbf{v}_j$, where C can be thought of the common source of both waves, indicated by the concentric circle sections (right).

plane, are used to compute C. As Φ_{ji} represents a signed difference, we have to take the direction of the waves into account:

$$d_{ji} = (|C - \mathbf{v}_j| - |C - \mathbf{v}_i|) \cdot \langle \frac{C - \mathbf{v}_j}{|C - \mathbf{v}_j|}, \mathbf{d}_j \rangle, \tag{3.2}$$

where $\langle ., . \rangle$ denotes the Euclidean dot product. Then, d_{ji} is virtually the distance between v_j and v_i^R (see Figure 3.4, right). Eq. 3.2 cannot handle the non-orientable property of the direction $\mathbf{d} \in \mathcal{D}$ that is depicted in Figure 3.5. If $\mathbf{d}_i^p$ and $\mathbf{d}_j^p$ are counter-oriented with respect to their common source C (i.e., one pointing towards and one pointing away from C), we have to take care of the ambiguity of d_{ij}. A simple adjustment to the calculation to solve this problem is described in Section 3.3.2, Eq.3.7. If the phases are consistent on the mesh, then the equation $\Phi_i = \Phi_{ji} + \Phi_j$ would hold for every edge $(i, j) \in E$. In

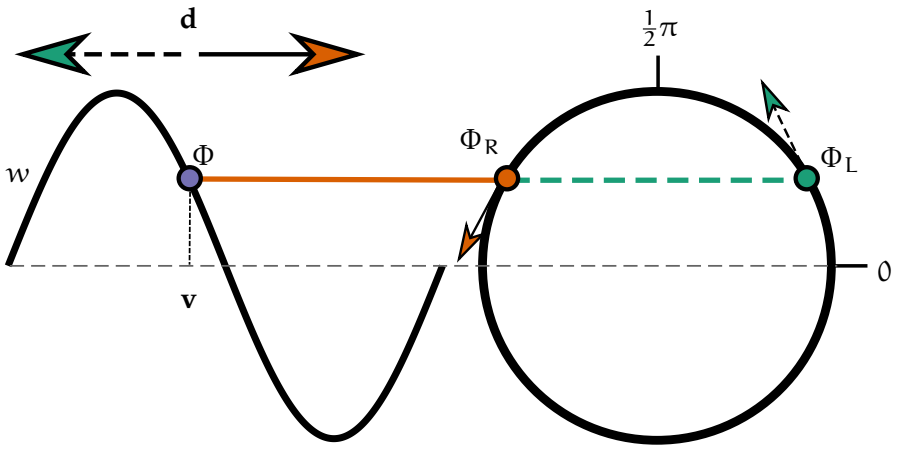


Figure 3.5: Interpreting w (left) as a sine-wave has two solutions on the unit circle (right), depending on the direction of $\mathbf{d}$. The elongation at Φ is either increasing or decreasing. In general, if Φ_R represents the phase of the wave traveling to the right, and Φ_L the one traveling to the left, $\pi - \Phi_R = \Phi_L$ holds.

general, this cannot be guaranteed, thus we aim to find the phase set $\mathcal{P}$ that minimizes the following energy:

$$\mathcal{E}(\mathcal{P}) = \sum_{(i,j)\in E} w_{ji} |\Phi_{ji} + \Phi_j - \Phi_i|^2, \tag{3.3}$$

where w_{ji} is a weight which can be chosen arbitrarily for each edge. We used $w_{ji} = |\langle \mathbf{d}_i, \mathbf{d}_j \rangle|$. Knöppel et al. [103] formulated the same minimization problem (regarding the energy formulation) and described how they can achieve a globally optimal solution. In this paper, we present a local algorithm, tailored to the GPU. Furthermore, we do not compute the energy based on the phase-shift directly. As previously done by Marcias et al. [146], it is based on the difference of the phases after transforming the phases into 2D Cartesian coordinates on the unit circle:

$$\mathcal{E}_i = \sum_{j\in\mathcal{N}(i)} w_{ji} \| \frac{1}{2}(\varphi_{ji} - \varphi_i) \|^2, \tag{3.4}$$

where $\varphi_i = (\cos(\Phi_i), \sin(\Phi_i))$ and $\varphi_{ji} = (\cos(\Phi_j + \Phi_{ji}), \sin(\Phi_j + \Phi_{ji}))$. This energy is equivalent to the one in [103], utilizing a single variable for each vertex and each edge. Our global energy is defined as:

$$\mathcal{E}_V = \sum_{i\in V} \mathcal{E}_i. \tag{3.5}$$

The rationale for using Cartesian coordinates is the handling of mismatching phases, similar to an outlier treatment, and will be described in more detail in Section 3.3.2. The relation between radial

and Euclidean distances is depicted in Figure 3.6 (left). With our approach, we achieve similar results to the ones presented by Knöppel et al. [103] by finding a local solution Φ_i such that $\mathcal{E}_i$ is minimal. We will later show (in Section 3.3.6) that only small changes to the implementation are necessary, such that we can process unit cross-fields.

3.3.2 *Local Optimization*

Our optimization strategy iteratively adjusts Φ_i to minimize the global energy $\mathcal{E}_V$. First, we initialize Φ_i with a pseudo-random value, assigning a value obtained by using the vertex ID as input for the One-at-a-Time hash, that can be found in the online version of [88]. The minimization is done in parallel for each vertex i and iteration k in two main steps:

1. For every (oriented) edge $(j, i) \in E$, a target phase $\Psi^k_{(j,i)}$ is computed. (Section 3.3.2.1)
2. Φ^{k+1}_i is determined such that $\mathcal{E}^{k+1}_V \leqslant \mathcal{E}^k_V$. (Section 3.3.2.2)

In step 2 we explicitly do not ask for every vertex that $\mathcal{E}^{k+1}_i \leqslant \mathcal{E}^k_i$ holds, but we ensure that $\mathcal{E}^{k+1}_V \leqslant \mathcal{E}^k_V$ holds. This is inspired by *Simulated Annealing* (SA), which randomly allows locally negative updates during optimization [29]. More details about the convergence are given in Section 3.3.3.

3.3.2.1 *Target Phase (Step 1)*

The goal in this step is to determine Φ^{k+1}_i for vertex i. As a preliminary value, we set

$$\hat{\Psi}^k_{(j,i)} = \Phi^k_{ji} + \Phi^k_j, \tag{3.6}$$

as we would expect $\Phi_i = \Phi_{ji} + \Phi_j$ for a perfect stripe pattern. At this point, we have to take the wave directions $\mathbf{d}$ into consideration. If the waves of the vertex i and j are counter-oriented with respect to their common source (Figure 3.4), errors occur due to the directionality of the wave function. In particular, $\hat{\Psi}^k_{(j,i)}$ does not correctly reflect the target phase that is necessary to match two such waves. In this case we have to adjust $\hat{\Psi}^k_{(j,i)}$ (see Figure 3.4 right and Figure 3.5):

$$\Psi^k_{(j,i)} = \begin{cases} \pi - \hat{\Psi}^k_{(j,i)}, & \langle \mathbf{d}_i, \mathbf{d}_j \rangle \cdot \langle C - \mathbf{v}_i, C - \mathbf{v}_j \rangle < 0 \\ \hat{\Psi}^k_{(j,i)}, & \text{otherwise} \end{cases} \tag{3.7}$$

With this adjustment $\Psi^k_{(j,i)}$ matches the waves of vertex i and j across the edge (j, i) by their elongation rather than by their phase, and thus takes counter-oriented directions into account.

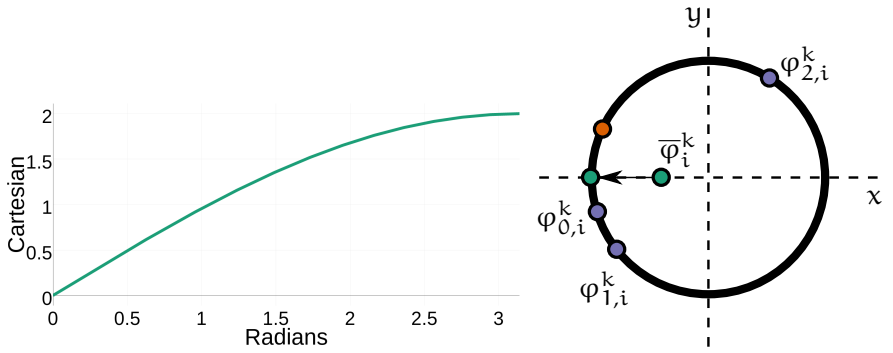


Figure 3.6: Comparison of Euclidean and radial distances of two points on the unit circle (left). $\overline{\varphi}^k_i$ is the average target phase based on the Cartesian coordinates on the unit circle ($\varphi^k_{j,i} = (\cos(\Psi^k_{(j,i)}), \sin(\Psi^k_{(j,i)}))$). We can observe that the Cartesian mean phase (green dots) differs from the radial mean phase (mean phase along the arc, orange dot) (right).

3.3.2.2 *Phase Alignment (Step 2)*

In step two, we compute the mean target phase $\overline{\varphi}^k_i$ of vertex i in Cartesian coordinates, which is based on the target phases $\Psi^k_{(j,i)}$ along the (oriented) edges $(j,i) \in E$:

$$\overline{\varphi}^k_i = \sum_{j \in \mathcal{N}(i)} w_{ji} \varphi^k_{j,i}, \quad \text{with } \varphi^k_{j,i} = (\cos(\Psi^k_{(j,i)}), \sin(\Psi^k_{(j,i)})) \tag{3.8}$$

with $w_{ji} = |\langle \mathbf{d}_i, \mathbf{d}_j \rangle|$. Since in our local approach each vertex should fit as best as it can to its neighborhood with respect to the wave direction, we favor parallel wave directions and prune orthogonal or diverging directions. We average the Cartesian coordinates of the target phases on the unit circle, because this reduces the influence of phases that are far away from the average. This has two reasons:

1. Normalization of the 2D coordinate $\overline{\varphi}^k_i$ within the unit circle does not affect the result of $\operatorname{atan2}(\overline{\varphi}^k_i)$ (Figure 3.6, right). Thus, if a phase $\varphi^k_{j,i}$ has drawn $\overline{\varphi}^k_i$ towards the center of the unit circle, that effect is partly compensated by the normalization of $\overline{\varphi}^k_i$. Consequently, such disagreeing phases have less impact on the result, which is an important property of this representation, as it helps connected vertices to share a common phase more quickly.

2. When considering the distance of two points on the unit circle, the relation between the Euclidean and the radial distance is not linear (see Figure 3.6, left). In the Euclidean representation distances greater than $\sqrt{2}$ are distinctly compressed, compared the radial counterpart. This reduces the contribution of phases

far from $\overline{\varphi}_i^k$, (i.e., outliers get pruned). We compared the presented averaging method with averaging the phases directly in radial space and found that the Cartesian averaging resulted a more uniform pattern, with less bifurcations and lower residue energy $\mathcal{E}_v$.

To estimate Φ_i^{k+1}, we compute $\overline{\varphi}_i^k$ in Cartesian space. We obtain the following offset to the current phase:

$$\Delta\overline{\varphi}_i^k = \overline{\varphi}_i^k - \varphi_i^k \tag{3.9}$$

with $\varphi_i^k = (\cos(\Phi_i^k), \sin(\Phi_i^k))$. Finally, we obtain Φ_i^{k+1} by:

$$\Phi_i^{k+1} = \operatorname{atan2}(\varphi_i^k + m_i^k + \Delta\overline{\varphi}_i^k), \tag{3.10}$$

where m_i^k is a momentum. m_i^{k+1} is updated for each iteration as

$$m_i^{k+1} = s_m \Delta\overline{\varphi}_i^k \tag{3.11}$$

with $s_m = 0.2$ being the momentum strength. For the first iteration, we set $m_i^0 = (0, 0)$. The momentum can help to keep vertex phases flexible, to avoid a premature local convergence.

3.3.3 *Convergence*

To achieve convergence, we ensure that for every iteration $\mathcal{E}_V^{k+1} \leqslant \mathcal{E}_V^k$ holds. We compute the vertex energy $\mathcal{E}_V^k$ (Eq. 3.5), after each iteration k, to keep track of the optimization progress. Then, the relative energy improvement

$$\Delta\mathcal{E}_V^{k+1} = \frac{\mathcal{E}_V^k - \mathcal{E}_V^{k+1}}{\mathcal{E}_V^k} \tag{3.12}$$

is computed for every step. Note that, if a vertex reduces its local energy, the global energy is also decreased, because $|\Phi_{ji}| = |\Phi_{ij}|$ (see Eq. 3.1). The above assumption holds, as we ensure that updates of neighboring vertices do not interfere, by applying a graph-coloring to the mesh. Details on the graph-coloring are given in Section 3.4. Convergence is detected when $\Delta\mathcal{E}_V^k$ drops below a threshold ϵ_G. The convergence term allows the algorithm to stop as soon as a certain optimization quality is reached, but also requires to keep track of the convergence progress. This is also useful during the hierarchical optimization that is described in the next section.

3.3.4 *Hierarchical Optimization*

Our algorithm can optionally employ the hierarchical structure presented in [86]. In this section we recap the purpose of this hierarchy for completeness. In the local approach, the phase of a vertex is only

optimized with respect to its direct neighbors. In order to find a solution that is globally satisfactory, the information of a vertex has to propagate across the mesh, which is problematic for high resolution meshes. This propagation of information can be speeded up by running the optimization in multiple resolutions of the mesh. A coarse resolution will quickly converge, while a fine resolution will take small features of the mesh into account. Thus, we optionally utilize the multi resolution hierarchy depicted in [86], to obtain a set of meshes that approximately halves the number of vertices with each coarser hierarchy level. We can start our optimization in a coarse level and propagate the results to the next finer level until the original mesh is reached. We switch levels as soon as the current level has converged as described in the previous section. However, it has to be mentioned that the input wave directions and frequencies in the finest level are consecutively smoothed (area weighted average of merged vertices) in coarser levels. If the smoothed properties differ greatly from the original input, the results of coarse hierarchy levels will not properly represent the final result. Hence, the hierarchy is less feasible if the input properties are not smooth.

3.3.5 *Texture Coordinates*

If we want to render a periodic texture or pattern with the values given in $\mathcal{W}$, we have to compute a per-pixel phase Φ_P in the fragment shader. Each fragment's phase is determined by the vertex phases Φ_i, Φ_j, Φ_k, where $(i, j, k) \in F$ are the vertices of the triangle generating the fragment. We cannot simply interpolate the vertex phases linearly, due to the periodicity of the wave function [103]. This is important if the actual phase shift between two vertices along a wave is larger than 2π. In these cases we have to incorporate the frequency f. Next, we describe our interpolation method, which takes f into account.

PER-FRAGMENT PHASE To compute the fragment's phase Φ_P, we need information about the wave direction $\mathbf{d}_P$ at the 3D coordinate of fragment P.

$$\mathbf{d}_P = \sum_{m \in \{i,j,k\}} s_m \cdot b_m \cdot \mathbf{d}_m, \tag{3.13}$$

where b_m is the barycentric weight of each vertex to the fragment and s_m takes care of the direction of $\mathbf{d}_m$. If the wave of a vertex m in a triangle is counter-oriented relative to the other two, we set $s_m = -1$, otherwise we set $s_m = 1$.

This is required to avoid null vectors and to consider the equivalence $\mathbf{d}_i \sim -\mathbf{d}_i$. The adjustment through s_m is crucial for the proper

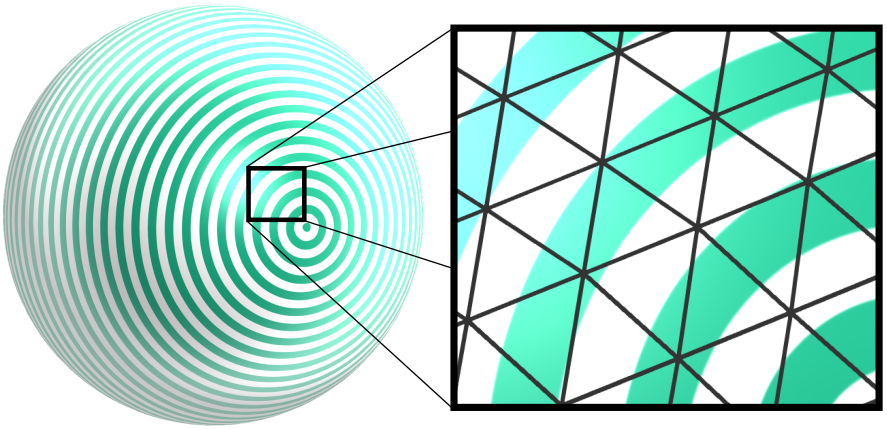


Figure 3.7: Wave pattern on a sphere. The close-up shows how our interpolation method is capable of aligning each individual fragment with the wave pattern. The direction of the pattern visibly changes even inside individual triangles, reflecting the vector field divergence.

display of textures (see Figure 3.5). Note that this can only be applied to triangles that can be oriented in a general direction. If this is not the case, we only make sure that the pattern is continuous across the triangle edges. This is achieved by using a barycentric subdivision of the triangle as described in [103].

With the 3D-coordinate of the fragment and $\mathbf{d}_P$ we can obtain a fragment phase $\bar{\varphi}^P$, similar to Eq. 3.8. In this case w_{ji} is replaced by the barycentric weight, the vertex i is replaced by the fragment, and the neighboring vertices j are replaced by the three vertices, that generate the fragment. Finally, we compute the fragment phase:

$$\Phi_P = \operatorname{atan2}(\bar{\varphi}^P). \tag{3.14}$$

The computation of individual fragment phases yields a pixel perfect alignment of the wave pattern, as shown in Figure 3.7 and the bending arrow texture in Figure 3.1 (center, left). This is an advantage over a linear interpolation as Figure 3.17 (left) shows. The IFAM source code uses linear interpolation to reconstruct the fragment parameters. This leads to visible linear segments in curved regions of the parameterization and limits the pattern frequency by the mesh resolution. The interpolation method used in [103] can handle the pattern frequency independent of the mesh resolution, but still generates linear segments inside triangles. Note that our fragment interpolation is only feasible because it is equivalent to the interpolation of vertices during the optimization process. Differing interpolation approaches would lead to distortion of the pattern.

2D COORDINATES Computing two wave patterns U, V, based on orthogonal vector fields allows to display 2D textures. For this, we

map $\Phi_p^u, \Phi_p^v \in [-\pi, \pi]$ to a range of $[0, 1]$ and use this value for texture access. Note that we need to take $\mathbf{d}$ into account again. Even if the peaks and troughs of counter-oriented waves match after our optimization, $\mathbf{d}$ still affects the texture access. Through the application of s_m in Eq. 3.13, two neighboring triangles can be counter-oriented with respect to their texture access. This happens if vertices on an edge (i, j) are counter-oriented, and for one triangle i is flipped, while for the other triangle j is flipped. Then, the texture coordinates along the edge are flipped as well. To obtain matching texture coordinates at sites of counter-oriented waves, we simply add $\pi/2$ to each fragment phase and then compute the texture coordinates. This matching only applies to textures that are symmetric along both axes (i.e., with a 180° symmetry). As in Figure 3.5, the direction in which we sample a texture depends on the wave direction. This does not account for textures that are symmetric along both axes.

3.3.6 *Cross-Fields*

As described in the previous paragraph, we can use two orthogonal vector fields to generate 2D coordinates for, e.g., texture access. Here, we shortly describe how the implementation can be optionally changed to handle two orthonormal vector fields as a cross-field with rotational symmetry. In our problem formulation the wave direction $\mathbf{d}$ was treated as an equivalence class $\mathbf{d} \sim -\mathbf{d}$, hence two orthonormal vector fields resemble a unit cross-field in the notation by Vaxman et al. [227]. We can extend the implementation to manage unit cross-fields by optimizing 2 linked stripe patterns (see Figure 3.1, right). In this case, each vertex $\mathbf{v}_i$ references 2 directions $\mathbf{d}_{i,r}$, with corresponding phases $\Phi_{i,r}, r \in \{0, 1\}$. The directions $\mathbf{d}_{i,r}$ are given by rotations of an input direction $\mathbf{d}_i$ about the normal of $\mathbf{v}_i$ by $r/2\pi$. During optimization, the individual waves are not isolated. Instead, the mean target phase $\overline{\varphi}_{i,r}$ for a given $\Phi_{i,r}$ (Eq. 3.8) is computed after removing the ambiguity (i.e., the rotational symmetry) of the cross-field: For each neighbor $j \in \mathcal{N}(i)$ we obtain $\Psi_{(j,i)}$ based on $\Phi_{j,m}$ and $\mathbf{d}_{j,m}$. Here, $m \in \{0, 1\}$ is chosen such that $|\langle \mathbf{d}_{i,r}, \mathbf{d}_{j,m} \rangle|$ is maximized. This has the effect, that $\overline{\varphi}_{i,r}$ is computed by taking the neighboring waves into account, that minimize the orientational difference to the direction $\mathbf{d}_{i,r}$. Note that this modification is optional. For the display of textures, two separate orthogonal vector fields are better suited, because they allow textures of 180° symmetry. A cross-field for example (that represents a 90° symmetry) is suitable for textures of 90° symmetry.

3.4 IMPLEMENTATION

We have defined our optimization method as a highly parallelizable problem. In this section, we describe our implementation of the algorithm on the GPU.

Generally, we would like to compute as many vertices as possible in parallel. Care has to be taken in terms of memory consistency, because the update for each vertex per iteration depends on the state of the neighboring vertices. By applying a graph coloring $\mathcal{C}$ to the mesh, we can find disjoint subsets $c \in \mathcal{C}$ of the mesh, such that there is no edge between any nodes in c, i.e., neighbored vertices do not share the same color. We can then iterate over the subsets $c \in \mathcal{C}$ and update all nodes belonging to c in parallel. In this case we do not need to take care of race conditions or memory consistency. The pipeline consists of four main steps:

1. Initialization
2. Graph Computation
3. Phase Evaluation
4. Pattern Extraction

STEP 1 We initialize the optimization by setting vertex phases to an initial pseudo-random value and calculate per vertex properties, if necessary. Per vertex properties might for instance consist of individual frequencies, or modifications to the underlying vector field. For example, the frequency might change based on the distance of a vertex to a dynamic reference point or the vector field might change with respect to time resolved vector field data. This step is performed in a vertex-shader. We do not apply further modifications to the data in order to achieve a most appropriate visualization of the input.

STEP 2 According to Jakob et al. [86] and with the neighborhood information obtained from the mesh topology, we compute a graph-coloring such that two neighboring vertices never share the same color. For this we implemented the Jones-Plassman-Luby algorithm described in the work by Naumov et al. [158]. To enable a vertex to access information to its neighbors, we store the vertex IDs of all neighbors per vertex. All data is stored in Shader Storage Buffer Objects (SSBO), so that arbitrary reads are possible.

STEP 3 We invoke a render pass, consisting of a vertex-shader, for each color of the graph and only process vertices of that active color. Like this, an active vertex can read data from its neighbors and write its updated data, without interfering memory access with its neighbors. Such a division into disjunctive sets has also been used in the

work by Choong et al. [29] within the context of a parallel SA implementation. At first, the mean target phase $\overline{\varphi}_i^k$ is computed as in Eq. 3.8. Along with $\overline{\varphi}_i^k$ we can compute the vertex energy $\mathcal{E}_i^k$, based on the state of the previous iteration. $\mathcal{E}_i^k$ is summed up over all vertices i using an atomic float counter. Thus, the convergence is checked against the energy of the last but one iteration and we can immediately obtain and apply Φ_i^{k+1} as in Eq. 3.10.

STEP 4 After the optimization has stopped, a final render-call determines the per fragment phase Φ_P as in Eq. 3.14 and extracts a pattern or texture coordinates to either display a stripe pattern or a texture. The GPU storage size which our SSBOs require, depends on how many vector fields we want to process simultaneously and on the maximum number of neighbors per vertex.

Our current implementation reserves memory for $4 + 7V + N$ components per vertex for V vector fields, if the maximum number of vertex neighbors is N. This contains: the world coordinate (3), the number of neighbors (1), the neighbor IDs (N), the direction $\mathbf{d}$ (4V), the phase Φ (1V), the frequency f (1V) and the energy $\mathcal{E}$ (1V). In this very straight-forward implementation, the color-graph allows us to neglect any memory barriers, other than separate render calls. A positive side-effect is that colors that are processed later, can already use the updated information of neighboring vertices. This effectively speeds up the propagation of information across the mesh. Further, while a graph subset $c \in \mathcal{C}$ is processed, we can be sure that the neighbors of the nodes in c do not change. Thus, a positive update of the vertices in c cannot be undone by parallel execution of their neighbors. However, the graph-coloring reduces the grade of parallelism, but if the GPU is still working at capacity, even if only vertices of single colors are computed, we expect the overhead to be at a minimum. Another optimization we can utilize depends on the quality of the graph-coloring. We utilize a naive parallel graph coloring approach and found that the last third of assigned colors covers only about 1.5% of the graph. With that in mind, we process the vertices of the last third of colors only every second iteration, to save computation time. We found that the influence on the result's quality is negligible in that case.

BRANCHES During the optimization, our algorithm automatically inserts branches to the stripe pattern. Such locations are also known as singularities of a positional symmetry field as in [86]. For completeness, we provide a short recap of this topic. A branch adds or removes a stripe segment to ensure an isometric spacing of the stripes, adapting to the surface morphology or to vector field divergence. These branches occur at locations where our interpolation (Eq. 3.8) results in an undefined phase (i.e., in the center of the unit circle

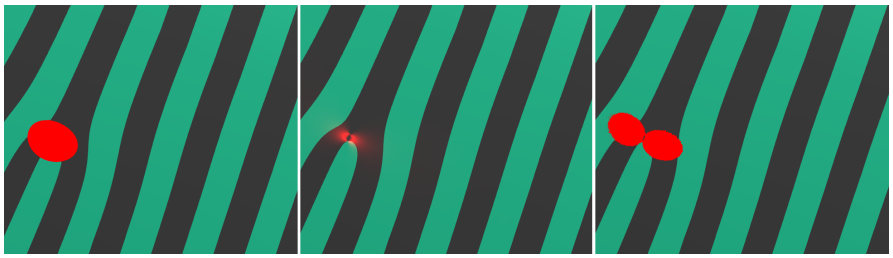


Figure 3.8: Branch properties: Area around undefined phase (left), angular deviation of the parameter gradient and the vector field, where a full red location represents a deviation of 90°(center), and angular deviation larger than 5°(right).

in Figure 3.6 (right)) for a given triangle of the input mesh. Such a location is marked in red in Figure 3.8 (left). We can find triangles that contain such a singularity by using the properties $\mathbf{w}_i$ of each of the triangle's vertices, and estimating the presence of such an undefined phase. Generally speaking, the less branches a parameterization contains, the higher its quality. The minimum number of branches depends on the mesh morphology and pattern frequency. It can be observed that the input vector field cannot be correctly depicted by the stripe pattern in branch regions. In the optimal case, the gradient of the parameterization (i.e., $\nabla\Phi$) is parallel to the vector field. The angular deviation of the gradient and the vector field are shown in Figure 3.8 (center and right). Regions where the parameterization does not properly agree with the underlying vector field are limited to branch regions. Away from branch regions, the stripe pattern is well suited to represent the vector field in a precise manner.

DYNAMIC INPUT The implementation can easily handle dynamic input (e.g. changing frequency, vector field direction or vertex position). If one of these properties changes from frame n to frame $n+1$, the output $\mathcal{P}$ at n can be used as input for $n+1$. Very few iterations of our optimization are necessary, to adjust Φ with respect to the dynamic input change. The number of iterations required to achieve a visually smooth update depends on the ratio of the frame rate and the rate at which the dynamic property changes. Rapid property changes require a higher number of update iterations. We found that updating $\mathcal{P}$ with 1-3 iterations per frame yields good visual results, while a higher number adjusted $\mathcal{P}$ too fast, resulting in jittery movements of the visualized pattern.

3.5 RESULTS

In this section, we present several usage scenarios in which we applied our technique to surface meshes from both real and artificially generated datasets. Additionally, we provide a qualitative and quan-

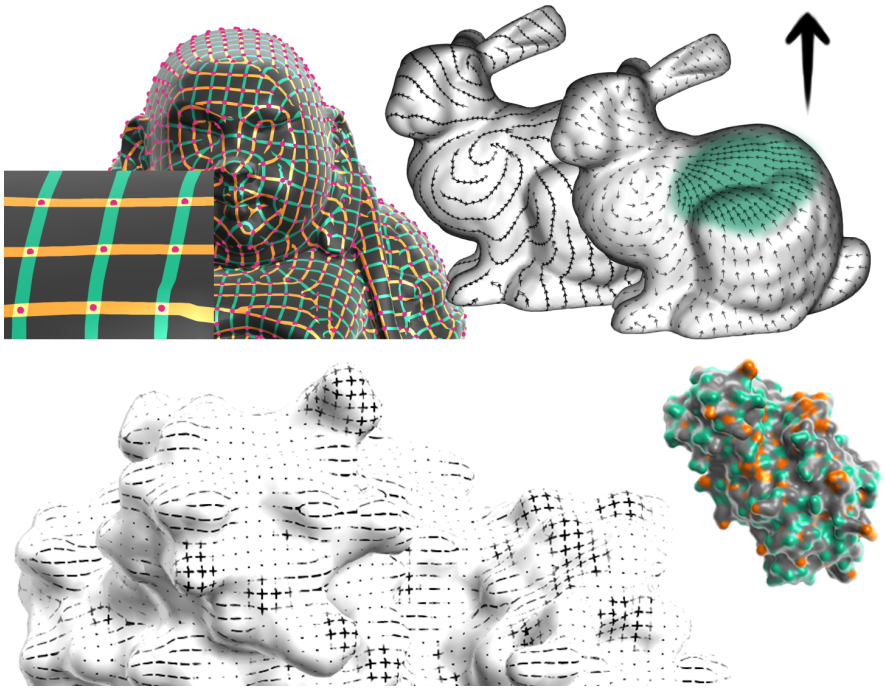


Figure 3.9: Magenta dots mark detected sample points (top, left). Anisotropic (top, center) and importance (top, right) sampling combined with arrow decals (using decal maps [185]) placed by our sampling technique visualize a vector field. The sample density is increased in the highlighted area. Electrostatics of a molecule (bottom): Comparison of a color map and user defined texture patches that represent positive (orange, +), negative (green, -) and neutral (gray, ·) charges.

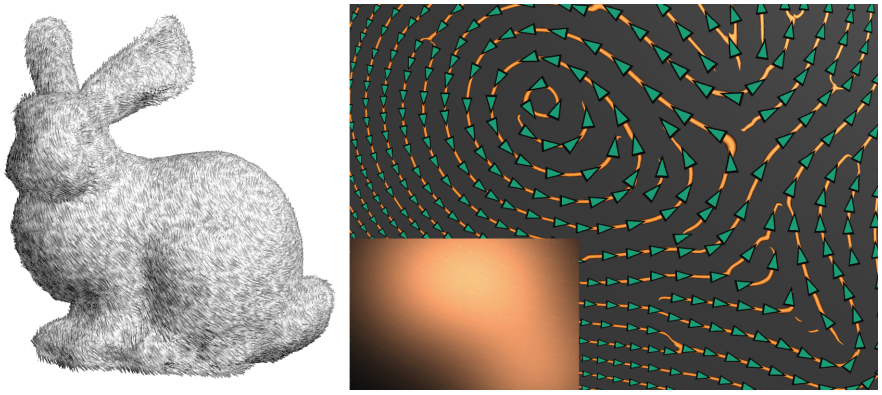


Figure 3.10: Artistic hatching achieved by rendering surface aligned quads with a stroke texture (left). Visualization of a vector and scalar field (right, the scalar field is depicted in the inset).

titative performance comparison to the work of Knöppel et al. [103] and Jakob et al. [86].

3.5.1 *Usage scenarios*

Our technique can be employed in a range of scenarios in which information needs to be visualized on surface meshes on-the-fly, that is based on scalar or tangent vector input. It can also be utilized in the course of multi-variate data visualization to some extent, because the patterns generated by our method can be employed as an information channel in addition to color or glyphs, for instance. Here, we present several concrete usage scenarios for our technique based on bio-medical and vector field visualization.

SURFACE SAMPLING Our method can be employed for structured surface sampling. After the optimization process, we can find locations on the surface with $\Phi = n \cdot 2\pi$, $n \in \{0, 1\}$, which can be thought of as the peak of a cosine wave. If two orthogonal stripe patterns are given, the intersection of their wave representations' peak locations sample the surface in a regular pattern. The search of these samples can be implemented in the geometry shader, since for each triangle, the peak locations can be estimated based on their vertices' phases Φ, directions $\mathbf{d}$ and frequencies f. In Figure 3.9 (top,left) the green and orange stripes show locations around wave peaks of orthogonal vector fields. The intersections of these peaks sample the surface in a regular pattern, with respect to the wave directions and frequencies. An anisotropic sampling has been achieved by simply using different frequencies for two orthogonal stripe patterns (Figure 3.9, top, center). Similar to the Poisson-sampling proposed by Corsini et al. [34], we can introduce an importance sampling, by increasing the frequency in more important regions (Figure 3.9, top, right). More sampling-based visualizations are shown in Figure 3.10. For the bunny on the left, we generate tangent vector field-aligned quads, based on the sampling. These quads are then rendered with an artistic hatching texture, allowing us to draw across the original mesh boundary. In Figure 3.10 (right) a vector field is visualized using arrow glyphs. An additional scalar field is used to modulate the pattern frequency, which can be employed to draw the glyphs in varying size. In this context, we think of applications in tensor field visualization, where the glyph size and spacing is controlled by the eigenvalues of the tensor. The adaptive sampling is then able to resemble a sort of glyph packing [100].

ILLUSTRATIVE BIO-MEDICAL VISUALIZATION Our technique can be employed for illustrative bio-medical visualization applications. As shown in the work by Ritter et al. [184], illustrative vascular visualization methods can be used to enhance spatial perception for

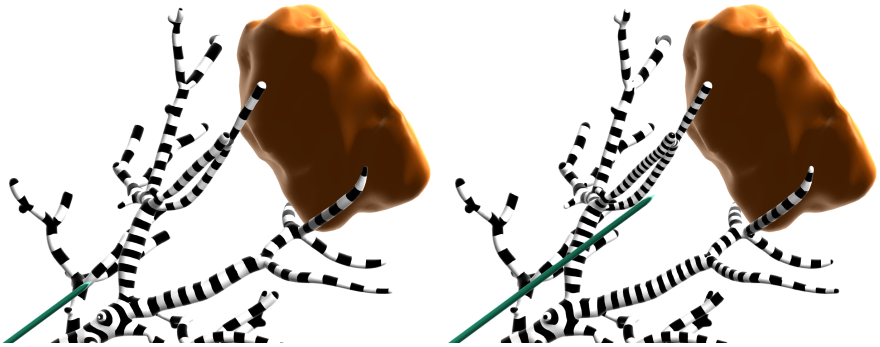


Figure 3.11: Liver vasculature is visualized using our technique, with a tumor displayed in brown and a surgical instrument indicated in green. The hatching frequency dynamically updates to encode the distance of the vessel to the needle, when the needle tip is further away (left), or closer to a vessel (right).

complex vascular structures. By using texture to encode shape and topology, the color channel is left free to encode additional information. Lawonn et al. [122] followed up on the work by Ritter et al. by developing a visualization technique for 3D vascular models in the liver, in which they used hatching styles to encode distances [184]. While their approach required preprocessing in the form of streamline calculation based on curvature, our current technique can be employed for similar purposes without any offline calculation. Curvature information, for instance, can be computed in real-time as well [57].

Since our technique is able to dynamically adjust the hatching frequency and stroke width, we can adjust our visualization on-the-fly to take novel information into account, such as changes to the scene resulting from interaction. In a surgical guidance context, hatching stroke frequency for instance could be based on the current distance to the camera [184] or one of the surgical instruments employed during an operation, as can be seen in Figure 3.11. The surfaces in this figure were reconstructed from a clinical CT dataset. By varying the hatching frequency based on instrument proximity, as the instrument gets closer to the vessel, the stripe size adjusts proportionally, such that the line width is approximately 1/10 of the distance to the needle. The hatching style and color are then still free to encode other information, for instance to use pseudo-chroma-depth to enhance depth perception. Furthermore, an interactive focus-and-context visualization can be generated using dynamic cutaways via a *binary transparency*. This effect is similar to the *screendoor focus* in [38], though our pattern follows the mesh surface, preserving geometrical features. This can be used to provide a view on nested structures, for instance the vasculature and tumor, which reside inside the liver, without altering the color perception of the structures within (see Figure 3.1, left).

Several methods address the generation of hatching strokes, based on dynamic properties, such as focus-and-context driven line generation [136] or apparent ridges [91]. Our technique is able to generate strokes of adjustable width and spacing and allows dynamic stroke directions without pre-computation. An example, along with a comparison to existing techniques by Lawonn [121] and high quality hatching by Zander [233] is given in Figure 3.12. The example shows that the parameterization can be used to draw locally varying hatching strokes, or to render a dashed silhouette, to obscure less important parts of a mesh.

When introducing color to the stripe pattern, we can display vector and scalar fields in a combined view, as done in Figure 3.1 (center, right). Here, we show the first eigenvector of a stress-tensor on the vessel surface of an aneurysm. The first eigenvector is represented by the stripe pattern. The first two eigenvalues are shown in green and orange. Purple and magenta highlight areas, where the respective eigenvalue exceeds a user-defined threshold. The space between the strokes depends on another user-defined threshold and provides information about the global relation of the tensor magnitudes. I.e., thin and thick strokes represent low and high eigenvalues, respectively. The pattern frequency can be dynamically adjusted, e.g., based on the target object's distance to the camera. As the user zooms in, the frequency increases. Our algorithm's ability to dynamically update the pattern yields a smooth transition while zooming. A similar visualization of surface stress, that requires pre-computation, can be found in Meuschke et al. [149]. However, single stream-lines that are employed to depict the tensor data in their work, may overlap and therefore impair the perception. The idea of visualizing tensor fields with orientable patterns is also implemented in the work by Auer et al. [4].

Visualizations of biological information can also benefit from illustrative visualization techniques [127, 223]. For example, molecular visualization often aims to abstract context information, or has to deal with occlusion [105], or time-varying simulation data. In Figure 3.9 (bottom) we visualize electrostatic properties of a molecule. While the color map is a common way to do so, we can use our periodic texture coordinates for a space-filling mapping of texture patches to the surface. This way, the color channel is left free for representation of other properties. In general, there are many cases in which bio-medical multivariate or dynamic data needs to be visualized on meshes where preprocessing is not possible and/or not desirable.

VECTOR FIELD VISUALIZATION Besides handling cases in which we visualize dynamically changing scene information at run-time, such as updates based on instrument location, we are also able to handle dynamically changing vector fields on the surface itself. This

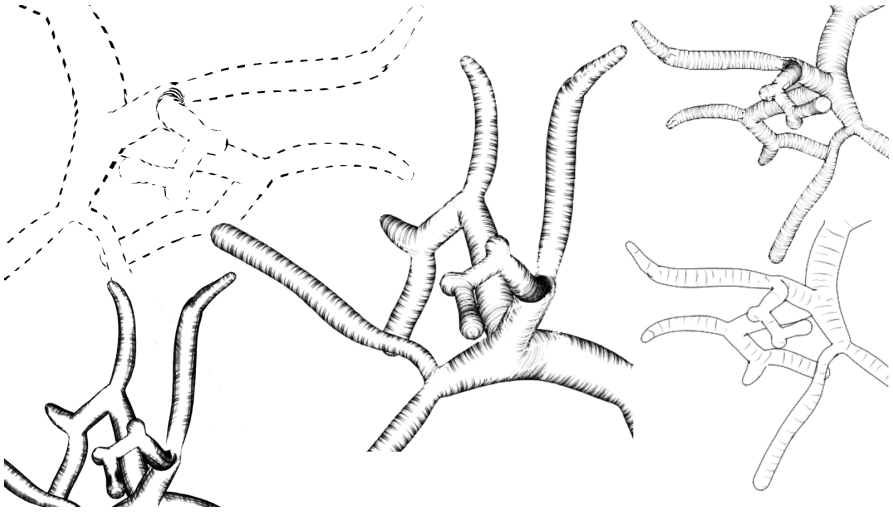


Figure 3.12: Illustrative visualization of vascular structures of the liver: Locally varying hatching strokes (center), cross hatching (left, bottom) and dashed silhouette (left, top). Comparative visualizations using the ConFis method by Lawonn [121] (right, top) and high quality hatching by Zander [233] (right, bottom).

is useful for instance in flow visualization, specifically when visualizing unsteady flow, i.e., flow which is changing over time. In the flow visualization survey by McLoughlin et al. [148], it was stated that unsteady flow is more challenging to visualize, and animation is a natural way of representing this time-dependent flow. In Figure 3.1 (center, left), we provide an example of visualizing an artificially generated time-dependent vector field on a surface. The vortex cores, represented by the orange and blue spheres in the figure, move over time and influence the vector field on the surface. Close to the vortex cores, we visualize the vector field with an animated arrow texture, while further away we animate the field with a less salient texture to emphasize the flow around the vortices. The flow can be animated, by shifting the texture access based on a periodic time parameter. The speed of the flow is then represented by the pattern frequency. For example, texture patches in regions of high frequency move slower, because they cover a smaller region in 3D space during a constant time period. Due to the globally continuous pattern that is generated every timestep, we are able to update changes to the flow on-the-fly. We further visualize a synthetic vector field on the bunny model in Figure 3.9 (top), utilizing the Decal-Maps method by Rocha et al. [185]. The decal positions are defined by our sampling method. The more regular distribution of our samples might be an advantage when sample positions are used for flow visualization. With our method, the sampling itself is able to resemble the flow direction, because the samples are found along wave peaks. It has to be noted that our technique - since it is an optimization - can only represent an approximation of the underlying vector field.

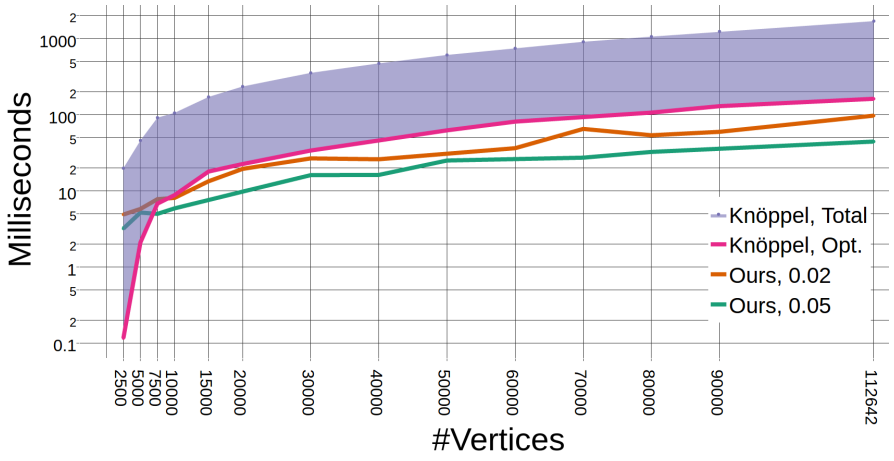


Figure 3.13: Timings for two orthogonal parameterizations of our approach compared to SPS, applied to the *Horse* mesh at different resolutions. The lower bound of the filled line displays the timings for one optimization iteration in [103] Alg. 6. The upper bound represents the total time their algorithm takes for both, building up their matrix representation ([103] Alg. 4) and computing one optimization iteration. Numbers are given for our optimization for $\epsilon_G = 0.05$ and $\epsilon_G = 0.02$ in log scale. The visual results are shown in Figure 3.15.

3.5.2 *Performance*

To assess the performance of our algorithm, we generated 2D-stripe patterns for several meshes of various sizes, ranging from 2.5k to 110k vertices per mesh. The performance tests were executed on a desktop computer environment with a 4.00 GHz i7-6400 processor, a GTX-1070 GPU and 16GB RAM. Since the problem that we solve is very similar to that targeted by SPS [103] and IFAM [86], we provide a quantitative comparison of our technique with their methods. To make the comparison with Knöppel et al. [103], we compiled the original code provided by the authors, together with the SuiteSparse and CHOLMOD packages from the Ubuntu package-manager. Both algorithms get a vector field as input, that we obtain by projecting the same global vector into the tangent plane of each vertex. Depending on the optimization parameter ϵ_G (see Section 3.3), we can optimize for speed, with lower quality pattern generation results, or optimize for quality, at the cost of computation speed. The stripe pattern algorithm can be tuned in a similar way. Their computation relies on an energy-matrix build-up, and an iterative optimization of this matrix ([103], Alg.4 and Alg.6). The number of optimization iterations in their publicly available implementation is 20. We found that the results after 4 iterations was visually difficult to distinguish from the results after 20 iterations. Nevertheless, we chose to conduct our comparison with their results after only one optimization iteration, to ob-

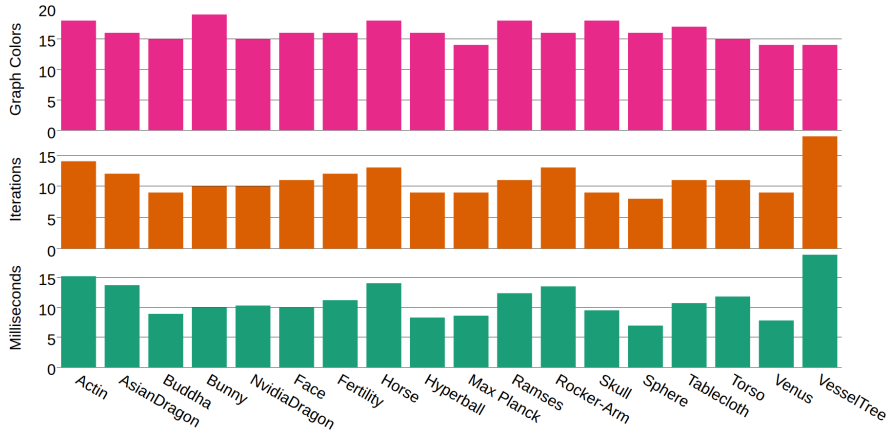


Figure 3.14: Timings of our approach for different models with 30k vertices and 60k faces each. The number of iterations correlates strongly with the runtime, indicating that GPU work-load is distributed similarly across different meshes. The number of colors in our color-graph does not correlate with the timings.

tain the lowest possible timings and to account for probable real-time ability. For the comparison with the IFAM algorithm, we use the code that was published by Jakob et al. [86]. Here, both algorithms find a solution based on a cross-field input using 10 hierarchy levels.

QUANTITATIVE PERSPECTIVE The plot in Figure 3.13 shows that our method computes stripe patterns faster than the globally optimal stripe pattern, for sufficiently large meshes. Very small meshes do not benefit enough from the parallelism of our approach. The presented timings were achieved without using the hierarchy levels mentioned in Section 3.3.4. We did this, because the hierarchy is not generally feasible. In cases where we update the parameterization frame-by-frame and use the result of the previous frame as input, we cannot take advantage of the hierarchy. Re-using results in coarser hierarchy levels for consecutive frames would not yield a frame-coherent visualization at the finest level, since the propagation to the coarser levels would smooth the input. We compare the timings our algorithm takes to converge, with the total time of the stripe pattern method, which includes building up an energy-matrix representation and optimizing that energy over one iteration. We do so, because their energy-matrix is computed based on the mesh morphology and the target frequency or orientation per vertex. Thus, for morphology changing meshes, or dynamic input, the timings displayed in Figure 3.13 represent the minimum timings per frame. Even if the energy-matrix build-up would not be completely recomputed for each frame, our algorithm still completes faster, than one optimization iteration of the reference algorithm. Convergence of our algorithm depends on the

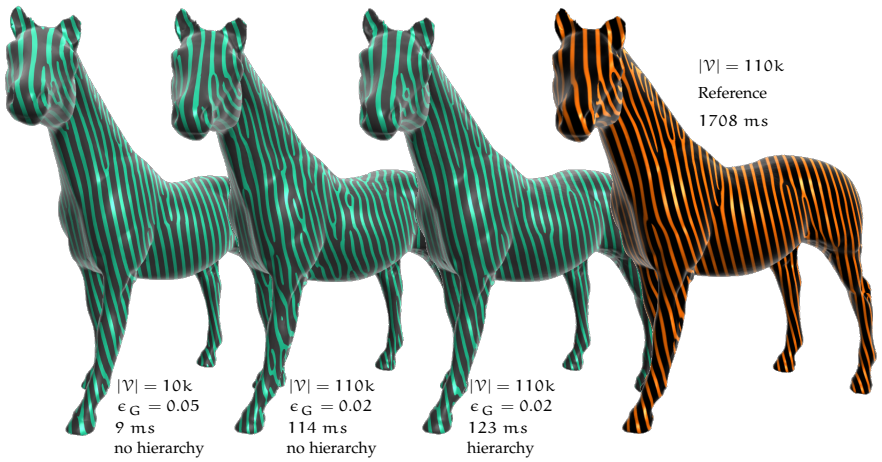


Figure 3.15: Comparison of our method and SPS (Reference, after one iteration). Note that the timings were measured while computing two orthogonal patterns, but we show only one pattern to avoid visual clutter.

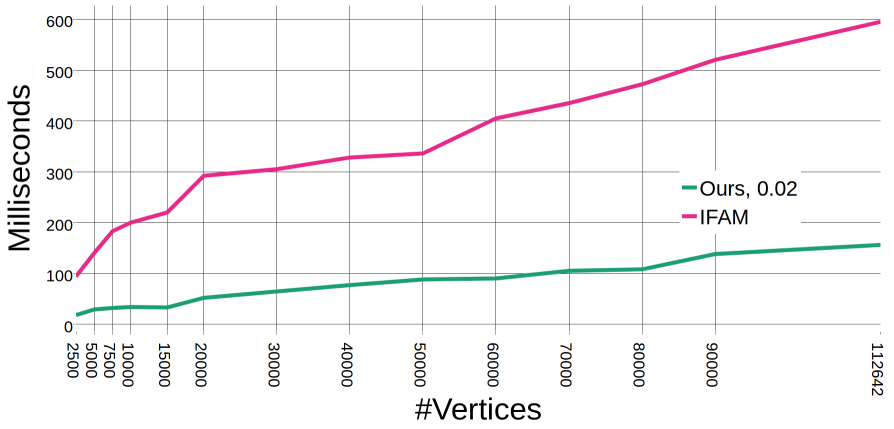


Figure 3.16: Timings of our algorithm (with $\epsilon_G = 0.02$) compared to the position field optimization by Jakob et al. [86] using a cross-field on the *horse* mesh at different mesh resolutions.

threshold ϵ_G. To account for this, we show timings for two configurations.

For dynamic input, our approach is superior to the stripe pattern algorithm. When updating an existing parameterization with dynamic input, 1-3 iterations of our algorithm are sufficient and thus only a fraction of the timings given in Figures 3.13 and 3.14 are required. The plot in Figure 3.16 compares our method with the parallel CPU optimization by Jakob et al. [86]. It shows that our GPU approach scales better with the number of vertices and is able to complete the parameterization significantly faster. From Figure 3.14 we can observe that our algorithm's performance is drastically mesh-dependent.

QUALITATIVE PERSPECTIVE If computation speed is less crucial, ϵ_G can be decreased. As processing time increases, the quality of

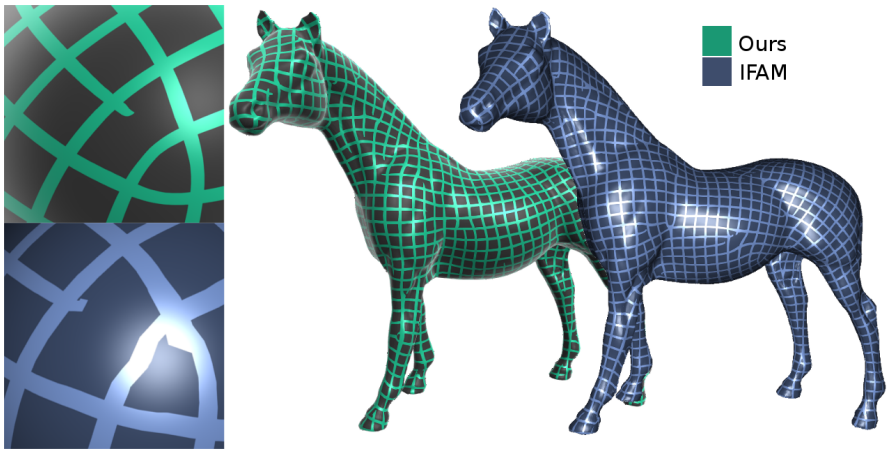


Figure 3.17: Visualization of similar parameterization results of the same sphere model (left). The *horse* mesh (225k faces) parameterized with a cross-field by our method and the method by Jakob et al. [86] (right). Note that the cross-fields are not the same, since these are based on a random seeding.

the output pattern increases as well. However, we can observe in Figure 3.15 (center, left) that the quality is nonetheless limited for high resolution meshes. In comparison we show the same model (Figure 3.15, center, right), after the optimization using 10 hierarchy levels. The processing time has slightly increased, but the visual outcome is close to the optimal reference (Figure 3.15, right). Also, in this example, the number of branches for the hierarchical optimization has been reduced by about 33% compared to the non-hierarchical one. Similar quality is achieved for a lower resolution of the mesh, without using the hierarchy (Figure 3.15, left). A visual comparison of to the method by Jakob et al. [86] can be found in Figure 3.17.

ROBUSTNESS Our method is robust against noise and incomplete meshes (see Figure 3.18) and can as well be applied to non-manifold meshes. As long as $\mathcal{D}$, $\mathcal{F}$ and the neighborhood $\mathcal{N}$ are defined, the al-

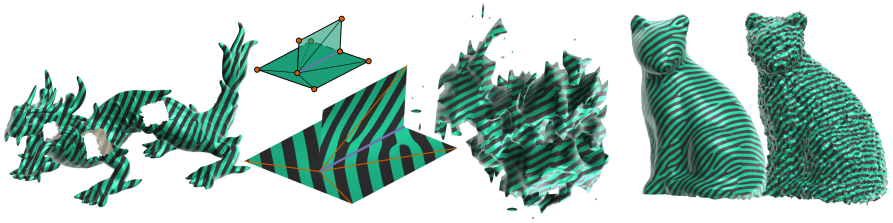


Figure 3.18: Our method can process incomplete (left) and non-manifold (center, left) meshes. The low quality tumor segmentation with unconnected parts (center, right) can also be handled, as well a noisy meshes (right).

gorithm is independent of any topological restrictions. It would even run on point clouds, but we leave this for future work.

LIMITATIONS The most significant limitation of the proposed algorithm is that it does not scale well with mesh resolution. We can address this problem with the hierarchical optimization, which allows us to parameterize large meshes at the cost of building up the hierarchy. However, the hierarchy is only applicable if the input parameters on the original mesh are already smooth. Furthermore, if we apply a frame-by-frame update of the parameterization to adapt to dynamic input, we also have to renounce the hierarchy. In such a case it might still be feasible to use the hierarchy for an initial parameterization, which can then be modified in consecutive frames. Processing dynamic input only on the level of the original mesh is crucial to maintain a frame-coherent appearance of the result. Regarding the results of our quantitative evaluation, we can state that the proposed algorithm is capable of processing meshes of up 70k vertices with approximately 30 fps. Beyond that, interactive rates are still possible but the parameterization quality is significantly impaired if not using the hierarchy (see Figure 3.15). However, the stated mesh size is appropriate for the proposed visualization tasks and especially for applications in the medical domain. Further, it might be desirable that the stripe pattern aligns with sharp features of the mesh. This is currently not supported but could be addressed by incorporating the extrinsic energy formulation in [86].

We assume that our implementation, which mainly resides in the vertex-shader, is straightforward on the GPU and sufficiently proves the real-time capability of our method. With the presented structure, there is no need for further synchronization of memory access. Contrarily, the CPU is likely to be a bottleneck here, since the GPU and CPU have to communicate for each render-call. During the optimization iterations, the number of calls amounts to the number of iterations times the number of colors in the color-graph. An implementation using CUDA, OpenCL, or compute shaders, which offer more flexibility and manually defined memory barriers, might improve the computation times for our algorithm.

3.6 CONCLUSION AND FUTURE WORK

We present a technique for the parameterization of surfaces that can be applied to surface meshes in real-time without time-consuming preprocessing steps. Our method generates a stripe pattern on arbitrary morphology on-the-fly. The work incorporates several ideas of existing work [86, 103, 176]. We adopt the representation of the periodicity in our parameterization through waves (i.e., the sine and cosine of the parameter space as in PGP), since their formulation is most intuitive in our opinion. The PGP algorithm parameterizes triangles and reconstructs per-vertex parameters from them. Hence, each vertex is initially processed n times, with n being the valence of

the vertex. In our method, vertices are parameterized directly, with respect to their neighbors, making it more suitable for parallel execution. This adopts the energy from SPS [103], defining one parameter for each vertex and each edge in a mesh (instead of parameters per vertex per triangle as in PGP). Knöppel et al. have introduced a sub-triangle interpolation scheme, to allow a pattern frequency higher than the mesh resolution. We use an extended interpolation method that is able to resemble the changes of the parameterization (especially with respect to vector field divergence, see Figure 3.7) within a triangle on a fragment basis. The interpolation method takes the vector field directions into account to compute the distance between two points on the surface. This approach is incorporated into our optimization process, to make the parameterization compatible with the interpolation method. A by-product of this action is, that we can obtain periodic patterns on the surface, that have an arbitrarily higher resolution than the mesh. This stands in contrast to IFAM, where the pattern resolution has to be lower than the mesh resolution. We further use a convergence term for our optimization process, whereas IFAM uses a fixed number of optimization iterations.

The performance evaluation and comparison to the reference method by Knöppel et al. [103] and Jakob et al. [86], reveal that aesthetically pleasing and accurate results can be generated under real-time conditions. We bring the topic of periodic parameterization to the context of data visualization, as shown by multiple examples that address different tasks. Besides animated textures on static surfaces, our technique is also capable of handling morphological changes in the surface mesh, and can be used for animated meshes. The computation for the *Ramses* model shown in Figure 3.19, containing 826k vertices, took 400 ms using 10 hierarchy levels. Hierarchies which support dynamic changes of the input have been proposed by Schertler [198]. Thus, future work should address the utility of the hierarchy for dynamic input in the context presented here, as to overcome the current limitations of this work.

Further, we would like to analyze the behavior of our approach on tessellation changing meshes, which could be applied to tasks that use several levels-of-detail.

However, with respect to the current results, we consider our method a powerful approach that provides convincing visual results. It has the potential to provide an important basis for future visualization applications.

ACKNOWLEDGEMENTS Funding: This work was supported by the DFG: LA 3855/1-1 and HA 7819/1-1, and the Bergen Research Foundation [grant number 811255].

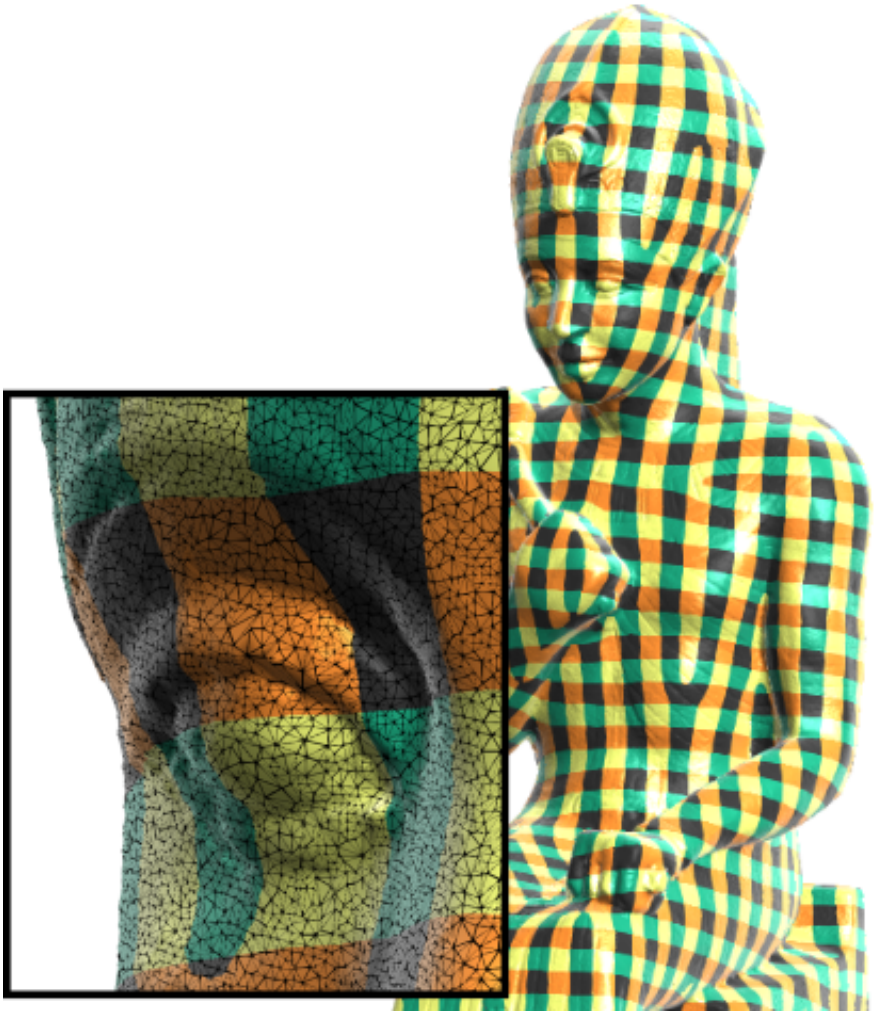


Figure 3.19: The *Ramses* model with 826k vertices, parameterized in 400 ms.

Part III

VESSEL VISUALIZATION

All works in this part deal with the visualization of liver vasculature. The predominant aspect is the enhancement of spatial and depth perception (Chapter 4), which is why this part also lays weight on view-dependent methods (Chapter 5, which is an extension of [133]). With these, the *Visualization Pipeline* by Haber and McNabb [59] forms a loop, since rendering results affect the subsequent data mapping. The part proceeds with a novel vascular tree abstraction and screen-space parameterization method (Chapter 6). A screen-based parameterization can, for example, be used to augment the void space around an object with texture patterns. Thus, even filigree structures can be highlighted by surrounding patterns. Finally, a comparative summary of a novel and existing depth-enhancement methods is given (Chapter 7), defining *Auxiliary Tools* as a visualization concept.

This part consists of the following papers:

Lichtenberg, N., Hansen, C., Lawonn, K., "Concentric Circle Glyphs for Enhanced Depth-Judgment in Vascular Models." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2017, pp. 178–188. DOI: 10.2312/vcbm.20171252

Lichtenberg, N., Lawonn, K., "Parameterization and feature extraction for the visualization of tree-like structures." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. Eurographics Association. 2018, pp. 145–155. DOI: 10.2312/vcbm.20181240

Lichtenberg, N., Lawonn, K., "Parameterization, Feature Extraction and Binary Encoding for the Visualization of Tree-Like Structures." In: *Computer Graphics Forum* (2019). DOI: 10.1111/cgf.13888

Lichtenberg, N., Lawonn, K., "Auxiliary Tools for Enhanced Depth Perception in Vascular Structures." In: *Biomedical Visualisation*. Ed. by Paul M. Rea. Springer International Publishing, 2019, pp. 103–113. DOI: 10.1007/978-3-030-14227-8

Lichtenberg, N., Krayer, B., Hansen, C., Müller, S., Lawonn, K., "Distance Field Visualization and 2D Abstraction of Vessel Tree Structures with on-the-fly Parameterization." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2019. DOI: 10.2312/vcbm.20191251

4

CONCENTRIC CIRCLE GLYPHS FOR ENHANCED DEPTH-JUDGMENT IN VASCULAR MODELS

ABSTRACT Using 3D models of medical data for surgery or treatment planning requires a comprehensive visualization of the data. This is crucial to support the physician in creating a cognitive image of the presented model. Vascular models are complex structures and, thus, the correct spatial interpretation is difficult. We propose view-dependent circle glyphs that enhance depth perception in vascular models. The glyphs are automatically placed on vessel end-points in a balanced manner. For this, we introduce a vessel end-point detection algorithm as a pre-processing step and an extensible, feature-driven glyph filtering strategy.

Our glyphs are simple to implement and allow an enhanced and quick judgment of the depth value that they represent. We conduct a qualitative evaluation to compare our approach with two existing approaches, that enhance depth perception with illustrative visualization techniques. The evaluation shows that our glyphs perform better in the general case and decisively outperform the reference techniques when it comes to just noticeable differences.

4.1 INTRODUCTION

Medical 3D models can be used for surgery planning. The vascular structures within these models, that are located in certain risk areas, can be of special interest for the surgeon. In a liver tumor resection scenario, for example, the vessels around a tumor may have a significant impact on the resection strategy that is defined by a surgeon [62, 66]. Based on their spatial location, vessels might be included in a resection volume, or represent risk structures that need to be preserved. High-quality 3D models can be obtained via CT or MRI and help a surgeon to obtain a full overview of the vasculature. It is known that depth cues are crucial for the perception of 3D scenes [178]. Thus, augmenting a scene with supporting depth cues can help a surgeon with the interpretation of medical data.

The perception of a 3D object can be naturally enhanced by stereopsis or by rotating the object in an animation, taking advantage of motion parallax. When no interactive visualization is desired or possible, other perceptual cues are necessary. Such a situation may occur in an operation room, where an interactive application may be too distractive for surgeons. Furthermore, static images might be useful for

physician-patient communication, medical documentation, and augmented reality visualization [184].

When displaying a 3D object on a 2D screen, the most significant property that is missing is the distance of a point on the surface to the viewer - or simply the depth of a point. This gap has to be filled in order to allow the observer to create a cognitive image of the 3D object. While perspective projection can be used in still images, it may however be insufficient to reveal small differences in depth. When no prior knowledge of the viewed structure is available, then perspective projection may even have no advantage at all, because single parts of the structure can be misinterpreted. For example, the projection of a circle with a larger radius, that is further away, may be as large as the projection of a smaller circle that is closer to the viewer.

In this work, we introduce a circle glyph, that is suitable to represent depth information. With this, we aim to augment static, as well as dynamic, 3D scenes with further depth information to help the viewer to get a thorough and precise understanding of a presented 3D object. We do this by the example of several vascular models, obtained from CT scans of the lung and liver. To cover the vasculature with glyphs, we come up with a graph-based selection method, that can take several properties into account. The selection method is view dependent and places more glyphs at depth-complex features, while maintaining an even distribution among the screen-space.

We examine the advantages and disadvantages of the new circle glyphs in a quantitative evaluation, that conducts a direct comparison with two existing methods. Our method is flexible and can easily be integrated into existing visualizations. The accuracy of our glyphs will be underlined by our evaluation. In summary, we make the following contributions:

1. A vessel end-point detection.
2. A feature-driven, filtered glyph placement strategy.
3. A circle glyph design, that allows for precise depth measurement in orthographic and perspective 3D visualizations.

4.2 RELATED WORK

For this work, we have to cover two main topics. The first is the field of research that deals with depth perception in computer graphics, especially in the medical domain. Secondly, there has been intensive research in the area of glyph-supported visualization. The literature also addresses general guidelines for the proper design and placement of glyphs, which strongly depend on the data and task at hand.

4.2.1 *Spatial Perception*

Spatial perception is naturally given through stereopsis. Hardware like 3D monitors or head-mounted displays (HMDs) employ this to give the viewer an impression of depth. However, HMDs can only be used by one person at a time. Hence, they require specially designed applications, particularly if collaboration of several users is desired. Stereoscopic images are further not suitable for print. Thus, it is still desirable to offer monoscopic visualizations for specific tasks. In the context of vascular structures, Kersten-Oertel [96] has covered a range of depth cues in an evaluation. In their setup aerial perspective and pseudo-chromadepth performed as better depth cues than stereopsis. Aerial perspective is an atmospheric effect, due to scattered light. Objects that are further away are perceived with less contrast [53]. An application of aerial perspective was tested for digitally reconstructed radiographs by Kersten et al. [97]. Chromadepth utilizes a similar approach. It has first been used by Steenblik [207] and uses the visible color spectrum, rather than the contrast, to encode depth. An extension, pseudo-chromadepth, has been proposed by Ropinski et al.[189]. Instead of the visible color spectrum, a range of colors from red to blue is utilized. The color-range has been chosen with respect to psychological foundations. Other approaches have used illustrative visualization techniques to enhance depth perception. For example, Ritter et al. have used illustrative shadows to emphasize the distance between vessels. Shadows between vessels of different distances can be distinguished by different numbers of hatching strokes. Generally, hatching can be employed to convey shape as done by Interrante et al. [83]. Less salient visualizations use feature lines and focus on geometric properties to make a visualization more comprehensive, as done by Zhang et al. [237]. Rendering geometry with such line-drawings can also help to address occlusion issues. A combination of line-drawings as mentioned above has been implemented in the work by Lichtenberg et al. [136]. Instead of modifying the appearance of a 3D model itself, ideas have been proposed that add supporting geometry to a scene. The first to mention is the work by Bichlmeier [13] who added a virtual mirror to a visualization. The virtual mirror allowed to add a second view perspective which can help to resolve problems with occluding geometry. Lawonn et al. used such supporting geometry as a shadow plane [122] or a cut-away object with depth information [124] to support depth perception. Further techniques that address medical visualization with respect to different tasks can be found in the report by Preim et at. [175].

In scenes with many objects, or complex objects such as vessel trees, occlusion is often a problem. Then, transparency can be used to dynamically highlight structures of interest [44]. The flexible manage-

ment of order-independent-transparency rendering is feasibly, thanks to modern GPU architecture [226].

4.2.2 *Glyphs*

Glyphs are a well accepted concept to display multivariate data. Ward et al. [228] have first introduced a taxonomy on glyph placement. Their taxonomy distinguishes between *data-driven* and *structure-driven* placement. In *data-driven* placement, the glyph location depends on the data that it represents, e.g., in a 2D plot. An example for *structure-driven* placement would be the visualization of a graph, using glyphs to represent the graph nodes.
As Ward's taxonomy can not be applied to 3D images or surface data, Ropinski et al. [188] introduced an additional placement strategy. Their *feature-driven* and *data set-driven* placement addresses glyph placement in the medical domain. Glyphs based on the iso-surface of a 3D image are *feature-driven*, while the placement on the nodes of a voxel-grid are *data set-driven*. Examples for Ropinski's feature-driven placement are given in [124], [190] and [153], where glyphs are placed on an iso-surface, orientated based on surface normal. A data set-driven approach is given in [162], where glyphs are placed on nodes of a voxel grid or segments of the AHA-heart-model.

A third possible placement strategy mentioned in their work is based on filtering. Filtering aims for the reduction of possible glyph positions to emphasize certain properties of the underlying data and to guide the viewers attention. Our placement method can be classified as a *feature-driven*, filtered placement. Anyway, we think that this classification is very general and allows for a more fine grained subdivision that distinguishes between multiple features that can be combined. We return to this topic in Section 4.3.2.

Despite the regular appearance of glyph placement strategies in literature, information is rare on how to filter glyphs on the surface of a 3D mesh to achieve a well balanced and task-oriented glyph distribution. Generally, *data set-driven* (including meshes) approaches place glyphs randomly and relax the distribution [153] or use a grid as an initial placement and avoid the underlying grid structure by jitter [119]. Mesh based approaches [149, 224] apply a vertex selection method to evenly cover the surface at different zoom levels. To our knowledge, the filtered glyph placement strategy proposed in [124] is the first to take features into account and to place glyphs at most representative locations. This approach can also be affiliated with Ropinski's *feature-driven* placement, but it is a much more restrictive example, where multiple features are combined. In the work by Rieder et al. [181] only a single circular glyph is placed at a point of interest. This glyph represents the depth of the point of interest along the view ray and is

comparable to the approach that we are going to present in this work. A thorough survey on the history and application of glyphs has been published by Borgo et al. [17]. They also composed a wide range of guidelines for the design of glyphs.

4.2.3 *Direct Foundation for this Work*

The work proposed in this paper is directly related to the publications by Lawonn et al.: *Supporting Lines* [122] (see Figure 4.8, center) and *Supporting Anchors* [124] (see Figure 4.8, left). Both approaches aim to enhance depth perception of vascular models by augmenting a 3D scene with supporting geometry. Selected locations on the vasculature are projected to simpler geometries that allow for an easier depth perception. A plane, situated beneath the mesh, is used for the supporting lines and a cylinder, surrounding a region of interest, is used for the supporting anchors. The vasculature casts a shadow on the plane and locations on the mesh are connected to their projections within the shadow by lines. Since the shadow is a natural depth cue, depth perception is supported in an intuitive way. For the supporting anchors, selected locations on the mesh are projected onto the closest point of the surrounding cylinder. Anchor shaped drawings are then used to build up a reference between the original location in the mesh and its projection. We also want to improve the depth perception by highlighting the depth of selected points on a mesh, but want to omit such additional geometry. Instead of building up a relation between surface points and their projections on simplified geometry, we place glyphs directly at the positions that these glyphs represent.

4.3 METHOD

The vascular data is represented by the mesh $\mathcal{M}$, containing vertices $\mathcal{V}$ and edges $\mathcal{E}$. An edge $e_{ij} \in \mathcal{E}$ exists if the vertices $\mathbf{v}_i, \mathbf{v}_j \in \mathcal{V}$ are connected. We aim to place glyphs at locations of vessel end-points, because these are the most representative locations on the structure when it comes to depth perception [124]. Furthermore, we want that k glyphs should evenly cover the mesh, where k is a user-defined value. Our method is split into two main parts:

1. Detection of vessel end-points (Section 4.3.1)
2. Graph based glyph placement and filtering (Section 4.3.2)

The set $\mathcal{P} \subset \mathcal{V}$ represents possible vessel end-points, that we get as input. Then the detection of actual vessel end-points $\mathcal{C} \subset \mathcal{P}$ follows. This is done by examining the neighborhood of each $\mathbf{p} \in \mathcal{P}$, which is denoted by $\mathcal{N}_\mathbf{p}$, while considering the boundary vertices of a neighborhood, denoted by $\mathcal{B}_\mathbf{p} \subset \mathcal{N}_\mathbf{p}$.

From the vertices in $\mathcal{C}$ we then build a complete graph $\mathcal{G}$ for further filtering of the candidate glyph positions. Throughout this section, we use several distance measures. For readability, any label in a variation of the following letters refers to scalars or functions related to a certain distance measure: d (geodesic distance), h (screen-space distance) and z (depth).

4.3.1 *Vessel End-Points*

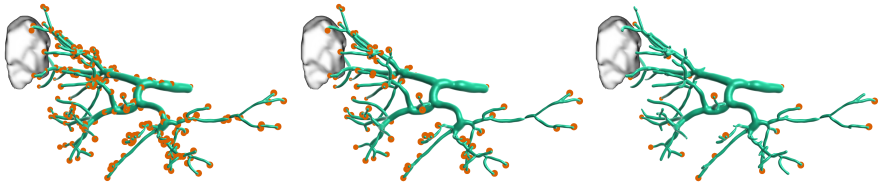


Figure 4.1: All locations in $\mathcal{P}$, found by the method by Lawonn [124] (left). Vessel end-points $\mathcal{C}$, detected by our method (center). Filtered glyph locations $\mathcal{S}$, for $k = 20$ (right).

We want to detect vessel end-points of the given mesh for the placement of our glyphs. In this work, we refer to a vessel end-point as the location of a half-spherical ending of a tubular section of the vessel model. As a first step, we determine a set $\mathcal{P} \subset \mathcal{V}$ of probable end-points that we obtain with the algorithm proposed by Lawonn et al. [124]. Their method incorporates the shape index and a connected component analysis for the end-point detection. However, the algorithm does not distinguish between vessel end-points and convex regions in general. Thus, we further classify these convex locations $\mathcal{P}$ to distinguish between vessel end-points and non-end-points.

We obtain sets of vertices $\mathcal{N}_{\mathbf{p}}$ that form a neighborhood around each end-point candidate $\mathbf{p} \in \mathcal{P}$.

$$\mathcal{N}_{\mathbf{p}} \coloneqq \{\mathbf{v} \in \mathcal{V} \,|\, d(\mathbf{v}, \mathbf{p}) \leqslant d_{\mathbf{p}}\} \tag{4.1}$$

where

$$d_{\mathbf{p}} = \underset{[d_{min}, d_{max}]}{\text{clamp}} \frac{\pi}{2} \frac{m}{\bar{\kappa}_{\mathbf{p}}} \tag{4.2}$$

and $d(\mathbf{v}, \mathbf{p})$ results in the geodesic distance between the points $\mathbf{p}$ and $\mathbf{v}$, for which we employ the heat method by Crane et al. [35]. The parameter m is a factor to control the magnitude of $d_{\mathbf{p}}$ and $\bar{\kappa}_{\mathbf{p}}$ is the mean curvature at $\mathbf{p}$. Thicker vessels will have a lower curvature and consequently $d_{\mathbf{p}}$ will be larger because it increases with $1/\bar{\kappa}_{\mathbf{p}}$, yielding larger neighborhoods. As a default set of parameters, we use $m = 3$(dimensionless factor), $d_{min} = 3$ mm, $d_{max} = 20$ mm. Note that $d_{max} = 20$ mm assumes, that no vessel has a circumference larger than 40 mm, as will be clarified next.

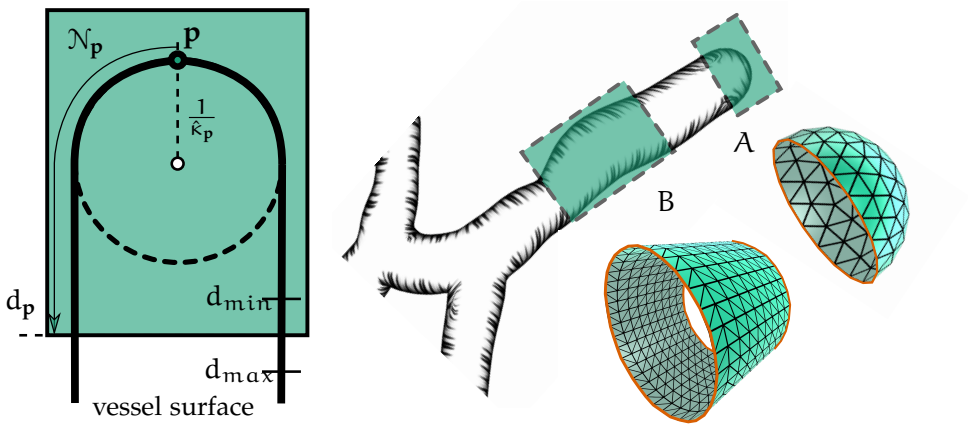


Figure 4.2: Schematic depicting $\mathcal{N}_\mathbf{p}$ (left). Valid end-point A and invalid candidate example B (right, top). Both example sets are shown as a mesh section of the vascular model (right, bottom). The boundaries of the sections are highlighted in orange.

We examine the graph topology of each neighborhood $\mathcal{N}_\mathbf{p}$ and store the boundary vertices in a set $\mathcal{B}_\mathbf{p} \subset \mathcal{N}_\mathbf{p}$. If the edges among the vertices in $\mathcal{B}_\mathbf{p}$ form exactly one cycle, we classify $\mathbf{p}$ as a vessel end-point and add it to the set of vessel end-points $\mathcal{C}$. The rationale behind this idea can be explained by the mesh illustration in Figure 4.2 (right). If a set $\mathcal{N}_\mathbf{p}$ is located at a vessel end-point, it will basically have the shape of a cup (i.e., the topology is equal to a disc). Hence, the edges between the vertices in $\mathcal{B}_\mathbf{p}$ will represent one cycle. In contrast to this, a set $\mathcal{N}_\mathbf{p}$, that is located somewhere on a branch, will not have this property. As shown in Figure 4.2 (right), example B covers a mid-section of a vessel. The corresponding $\mathcal{B}_\mathbf{p}$ contains two cycles, because of the two open ends. This method works only, if a set $\mathcal{N}_\mathbf{p}$, whose $\mathbf{p}$ is not located at a vessel end-point, covers the full circumference of the vessel. We ensure this with $d_\mathbf{p}$, which depends on the curvature (i.e., the approximated vessel radius) at $\mathbf{p}$. Further, the factor m introduces a large safe-margin by increasing the set size, while the parameters d_{min}, d_{max} control the minimum and maximum extent. A comparison of the locations in $\mathcal{P}$ and the end-points $\mathcal{C}$ detected by our method can be found in Figure 4.1.

4.3.2 *Multi-Feature Glyph Placement*

After the vessel end-point detection, we have obtained a set $\mathcal{C}$ of possible glyph locations on the input mesh. The cardinality of $\mathcal{C}$ depends on the mesh (i.e., the organ) and on the parameters used for the computation. Vasculatures in lung data sets exhibit more branchings, and thus more vessel endings compared to, for example, liver data sets. Displaying glyphs at all locations quickly overloads the visualization. Hence, we want the user to be able to control the number k of glyphs

that are visible. The selection of the k glyphs however, should be done automatically by an appropriate filter method.

Results of the quantitative evaluation in [122] have shown, that subjects found it most difficult to find the correct depth relation between two points, when the points were far away from each other (in screen space) and had a similar depth. Consequently, we would like to prefer such combinations of point pairs to show glyphs at. Another aspect we have to consider is the effect of size constancy, which is a well-known theorem in perception psychology. It says that the perceived size of an object remains constant, even though its projective image on the retina changes (i.e., an object moving away from the viewer). Due to the size constancy theorem, a thicker vessel might appear to be closer to the viewer than a thinner vessel, independent of the actual depth. For each vessel end-point $\mathbf{p} \in \mathcal{C}$ we know its approximated thickness in world space, given by

$$r_{\mathbf{p}} := \frac{1}{\bar{\kappa}_{\mathbf{p}}}. \tag{4.3}$$

This is even more important to consider when using orthographic projection which may lead to misinterpretation of the depth. To attribute for this, we want to prefer locations that represent either a thick vessel with a high depth value, or a thin vessel with a low depth value, to avoid false interpretations. Our objectives for the filter method are then:

R1 Balanced screen space distribution, preferring locations far away from each other (camera dependent).

R2 Depth aware distribution, preferring locations with similar depth (camera dependent).

R3 Favor locations that are critical with respect to size constancy (geometry dependent).

Extracting k glyph locations from $\mathcal{C}$ is done by using a graph based algorithm. The graph build-up and the selection of k glyph locations based on that graph are described in the following sections.

4.3.2.1 *Graph Buildup*

At first, we build a weighted complete graph $\mathcal{G}$ that connects all possible glyph locations $\mathbf{p} \in \mathcal{C}$. Edge costs $\mathcal{G}_{ij}$ between two locations $\mathbf{p}_i$ and $\mathbf{p}_j$ are computed as a combination of three terms (see Eqs. 4.6,4.7), covering the aforementioned objectives. Main parameters for the computation are the depth and screen space position of a location, as well as the vessel thickness $r_{\mathbf{p}}$. Given the minimal and maximal depth in $\mathcal{C}$ for a given camera position, we map the actual depth z_i of a location $\mathbf{p}_i$ to the normalized range $\bar{z}_i \in [0, 1]$.The normalized Euclidean

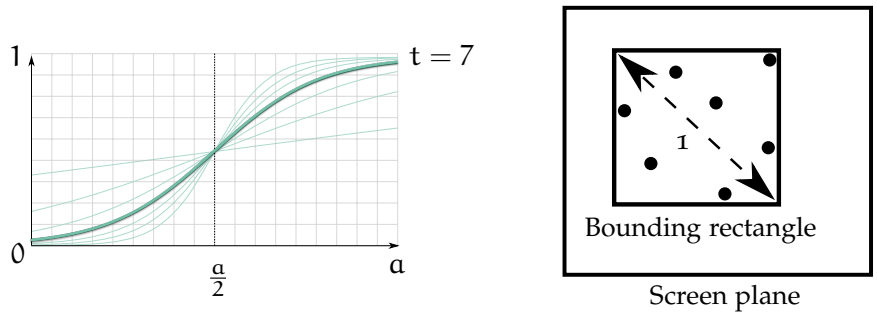


Figure 4.3: The approximated Heaviside function for different values of t (left). Bounding rectangle of the glyph locations in $\mathcal{C}$ (right).

depth distance of two possible glyph locations $\mathbf{p}_i$ and $\mathbf{p}_j$ is then obtained by $\hat{z}_{ij} = |\bar{z}_i - \bar{z}_j|$. A similar mapping is applied to the screen space coordinates of $\mathcal{C}$. After a minimal screen axis aligned bounding rectangle has been computed around all locations of $\mathcal{C}$, the screen space coordinates are normalized, such that the bounding rectangle's diagonal has length 1 (see Figure 4.3, right). The maximum euclidean distance between two coordinates within that normalized rectangle is 1 and the distance between two possible glyph locations is given by $\hat{h}_{ij}$. To stick with the notation used by Lawonn et al. [122], we denote the screen-space and depth relation with the distance labels: FF, FN, NF, NN. Here, F means far and N means near, where the first capital refers to the screen-space distance $\hat{h}_{ij}$ and the second capital to the depth distance $\hat{z}_{ij}$. For $\hat{h}_{ij} \geqslant \frac{1}{2}$ we use the label F and N otherwise. The same is done for $\hat{z}_{ij}$. For example, a pair of locations where $\hat{h}_{ij} \geqslant \frac{1}{2}$ and $\hat{z}_{ij} < \frac{1}{2}$ would be notated with FN.

To compute the edge cost g_{ij}, we further apply an approximated Heaviside function, which is a scaled sigmoid function, to the values obtained by $\hat{z}_{ij}$ and $\hat{h}_{ij}$. The Heaviside function resembles the partitioning by the F and N labels, because it can be used to map half of the parameter space to values close to zero (N) and the other half to values close to one (F). A general formulation for the approximated Heaviside function is:

$$f_{a,t}(x) = \frac{1}{1 + \exp(-(x - \frac{a}{2})t)} \tag{4.4}$$

where the parameter t controls the sharpness of the approximated Heaviside function and $[0, a]$ is the range of the input parameter x. As we want to compute f for the values $\bar{z}_{ij}$ and $\bar{h}_{ij}$, we can fix $a = 1$:

$$z_t(i,j) \coloneqq f_{1,t}(\bar{z}_{ij}), \quad h_t(i,j) \coloneqq 1 - f_{1,t}(\bar{h}_{ij}) \tag{4.5}$$

With that we obtain small values for FN labelled pairs (i,j), and high values for NF labelled pairs. We set $t = 7$ as a default value in our implementation (see Figure 4.3, left). The rather smooth transition does not completely binarize the input value a, thus allowing a differentiation inside the labels F and N. In $h_t(i,j)$ we invert the results, so that

we obtain low values for high screen space distances. The edge cost $\mathcal{G}_{ij}$ is then defined as:

$$\mathcal{G}_{ij} = \frac{z_t(i,j) + h_t(i,j)}{2} \tag{4.6}$$

covering the requirements R1 and R2. The size constancy (R3) is taken care of with a bonus term

$$\mathcal{G}_{ij} \leftarrow \begin{cases} \mathcal{G}_{ij} - b, & \text{if } r_{\mathbf{p}_i} > r_{\mathbf{p}_j} \text{ and } z_i > z_j \\ \mathcal{G}_{ij} - b, & \text{if } r_{\mathbf{p}_i} < r_{\mathbf{p}_j} \text{ and } z_i < z_j \\ \mathcal{G}_{ij}, & \text{otherwise} \end{cases} \tag{4.7}$$

where b is the bonus. Thus, edges between nodes that are critical due to the size constancy get their cost reduced by b. In this work $b = \frac{a}{2} = 0.5$ was used as a default value, because this is the location of the transition in the Heaviside function.

4.3.2.2 *Graph Filtering*

The selection of k nodes from the graph is done as follows: As an initial point pair, we select the nodes n_i, n_j with the lowest edge cost in $\mathcal{G}$ and add these to the set of selected, visible nodes $\mathcal{S}$. These are typically far away from each other in screen space and have a similar depth, thus, their distance is labelled FN. In order to find the remaining $k-2$ locations, we do an iterative search, consecutively adding nodes that best fit to the nodes in $\mathcal{S}$. We add a parameter to the graph based selection condition, that controls the minimal screen space distance h_{min} between two nodes in $\mathcal{S}$.This parameter can be used to avoid overlapping glyphs and we set a default value of $h_{min} = 0.075$ (7.5% of the maximum distance in the normalized ROI). Our selection method can be summarized in 3 steps:

1. For every node $n_i \in \mathcal{G}$, compute the average edge cost e_i and the minimal screen space distance h_i to all nodes $n_j \in \mathcal{S}$
2. Pick node $n_i \in \{\mathcal{G} | \arg\min_i e_i\}$, (i.e., $n_i \in \mathcal{G}$ with the lowest average edge cost):
   a) If $h_i \geqslant h_{min}$, add n_i to the visible set $\mathcal{S} \leftarrow \mathcal{S} \cup n_i$.
   b) Remove n_i from the graph $\mathcal{G} \leftarrow \mathcal{G} \setminus n_i$.
3. Repeat steps 1 and 2 until $|\mathcal{S}| = k$ or $\mathcal{G} = \varnothing$.

Exemplar results of the filtering are shown in Figure 4.1.

4.4 GLYPH DESIGN

For the design of our glyph, we have to take several considerations into account. From [188] we know, that visual stimuli are processed

in a pre-attentive and an attentive phase. Thus, we want to base our glyph on two main components, such that each component satisfies the conditions for either of these phases. Also they state, that glyph shapes should be unambiguously perceivable independent of viewing direction. Pop-out effects are described in literature, where color is the most significant channel, followed by size, shape and orientation. From [124] we already know that pseudo-chroma depth does not support small differences very well. So, to take advantage of the pop-out effect without losing precision, we choose to address the pre-attentive perception of our glyph by its size. A less salient property is the angle or orientation of a glyph, which is also listed as part of the geometric channel in [17]. We want to use this as the precise information channel during the attentive phase. Summed up, we want our glyph to meet the following criteria:

1. Pre-attentive stimuli through glyph size.
2. Attentive stimuli through angle/orientation.
3. Perceivable independent of view direction

Keeping in mind that we only want to display one scalar value - the depth, we can come up with a very simple glyph to match the above conditions.

PRE-ATTENTIVE PHASE: We use a circle as a very basic shape (*base circle*). Mapping the depth to the radius of the base circle perfectly matches the size constancy theorem. We can take advantage of the size constancy effect to exaggerate depth differences in a perspective view, or to imitate the effect in orthographic view. This way, the user can quickly identify clusters of glyphs that are closer or further away and hence get an overview of the vessel structure.

ATTENTIVE PHASE: In the attentive phase, we want to use an orientation attribute to allow for precise depth-judgment. We can combine this with an aspect that Ropinski et al. [190] attribute to the pre-attentive phase: The continuous or discrete mapping of an attribute to the glyph can communicate information. The discrete mapping enables better discrimination of fewer values. The continuous mapping allows to distinguish between small, just noticeable differences. Here, we combine the continuous with the discrete mapping, by filling the base circle with three concentric circles. If z_{max} is the maximum and 0 the minimal depth in a scene, and z the depth of a glyph, then we display one full inner circle if $z = \frac{1}{3}z_{max}$, two full inner circles if $z = \frac{2}{3}z_{max}$ and three full inner circles if $z = z_{max}$. This yields a discrete glyph attribute, that supports the pre-attentive phase, but can also be useful in the attentive phase. As illustrated in Figure 4.4, the depth range is subdivided into three bins reflected by the three circles. Furthermore, we cover the range between the discrete steps by

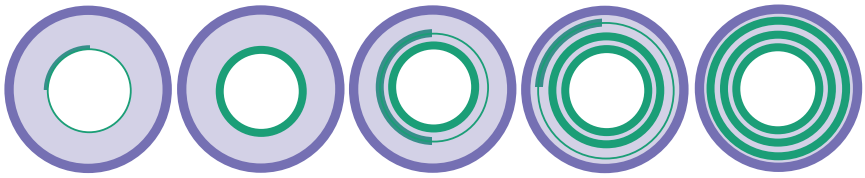


Figure 4.4: Combined discrete and continuous mapping. Depth represented by the glyph components are (left to right): $\frac{z_{max}}{12}$, $\frac{4z_{max}}{12}$, $\frac{6z_{max}}{12}$, $\frac{9z_{max}}{12}$, $\frac{12z_{max}}{12}$. The base circle is displayed in purple.

gradually filling the inner circles, which is the continuous part of our attribute mapping. During the attentive phase, the circle fill-status allows the user to precisely compare depth relations. Since every circle covers only one third of the depth range, the fill-status is more sensitive to differences and thus easier to read. The glyph could also be designed with two, or more than three circles, but we observed that three gave a good balance between sensitivity and readability.

FURTHER INFORMATION CHANNELS: If it is desired to represent a second scalar value through the glyphs, where precise perception is not crucial, other visual channels can be applied. An example is given in Figure 4.5, where the distance of a glyph location to the tumor is encoded in color and geometrical variations of the glyph. The geometric

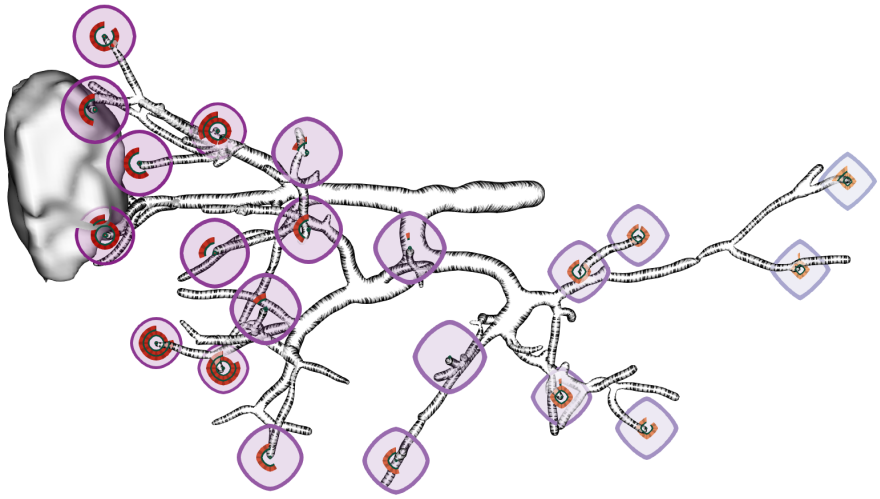


Figure 4.5: Tumor distance encoded in additional geometric channels: Close: Red, circular shape; Far: Yellow, rectangular shape.

modification only requires an adjustment of the fragment coordinates (x, y) during rendering (compare Figure 4.6, left and Eq. 4.8) The example shows, that three scalar values can be reflected by our glyph, with each scalar using a distinct visual channel.

SUMMARY: The proposed glyph is designed with respect to aspects of the pre-attentive and attentive perception phases. The pre-attentive phase is attributed to by modifying the base circle size in dependence of depth and by the discrete mapping to the three inner circles. The attentive phase is covered by the circle fill-status. The filling contains an orientational attribute that can be used for a relative comparison of the mapped value. For rendering, the glyph can be implemented as a camera-aligned billboard and is thus independent of camera direction or lighting. Details on the implementation and further considerations to make the glyph feasible in the actual visualization are given in Section 4.5.2.

4.5 IMPLEMENTATION

In this section we describe implementation details for the glyph location selection as well as for the glyph visualization and the hatching technique that we apply to the vascular models.

4.5.1 *Glyph Selection*

The selection of glyph locations is split into two main parts. The offline pre-processing to find possible locations and the online filtering of glyphs that selects glyph locations with respect to the dynamic camera features. The pre-processing time scales linearly with the number of input candidates of the set $\mathcal{P}$ and further depends on the mesh resolution. Thus, we provide timings for two exemplar meshes to give a rough estimate: For a mesh with 30k vertices, we measured an averaged time of 0.33s per candidate, while the computation of the geodesic distances, using the heat method [35], took 63% of that time. Another mesh, with 8k vertices, took 0.1s per candidate, with 71% consumed by the heat method. Note that the pre-processing can be potentially run in parallel for each candidate. In the current implementation of the online filtering, we recalculate the glyph positions for each frame. Thus, the k glyph locations may vary with each slight move of the camera. A thinkable modification would be to update the glyphs only after the camera rotation/movement exceeds a certain threshold, ensuring a stable visualization. The complete graph $\mathcal{G}$ is built up and the edge weights are computed (see Section 4.3.2.1, Eq. 4.6,4.7). Then we apply our filtering method to find k valid glyph locations (see Section 4.3.2.2) and send these to the OpenGL pipeline. The timings for the filtering depend on the number of possible locations $|\mathcal{G}|$ and the parameter k. For example, the graph buildup and the selection of $k = 15$ glyphs required 1.2 ms for a number of $n = 136$ possible glyph locations and 0.05 ms for $n = 25$ possible glyph locations, exhibiting real-time capabilities. The performance was tested

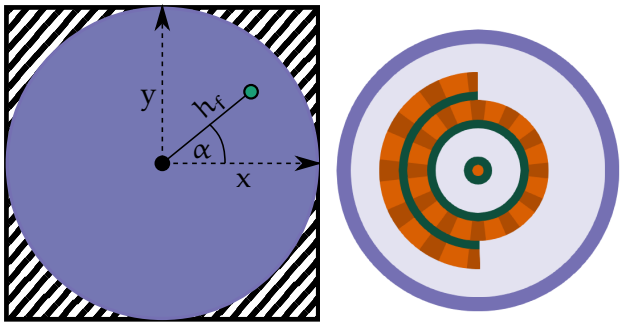


Figure 4.6: Fragment (green dot) properties obtained from *gl_PointCoord* (left). Fragments in the striped area are discarded. Final glyph rendering with one and a half concentric circles (right). The dot at the glyph center can help to identify the actual glyph position on the mesh.

on a desktop computer environment with a 4.00 GHz i7-6400 processor and 16GB RAM.

4.5.2 *Glyph Visualization*

Our glyphs can be added to an existing rendering pipeline in a simple way. As the world locations of the glyphs are given by $\mathcal{C}$, we can use point primitives to draw view-aligned quads. The size of the quads can be set in dependence of the depth of each location to account for perspective projection, or to imitate perspective distortion when using an orthographic camera. In the fragment shader, each fragment knows its location within the point primitive, given by *gl_PointCoord*. From the fragment coordinate within the quad, we can derive an angle α and a distance h_f to the quad's center (see Figure4.6, left). The distance can be modified with:

$$h_f^2 = \left(\operatorname{sgn}(x) \cdot |x|^q\right)^2 + \left(\operatorname{sgn}(y) \cdot |y|^q\right)^2 \tag{4.8}$$

where $q \in [0.5, 1]$ is the mapped distance of the glyph to a reference position (the tumor in Figure 4.5). This modification implements the transition from circular glyphs ($q = 1$) to quadratic glyphs ($q = 0.5$). Extending the range of q allows for more shape variations, but then the inner circles become hard to read. Along with the depth z, these are the properties that are necessary to render the glyph (see Figure4.6, right). Based on α, h_f and z, each fragment can be appropriately rendered. The base circle is filled with a purple transparent area. The concentric inner circles have two components. A thick, orange component with dashes and a green border between consecutive concentric circles. The dashes help to distinguish between glyphs of very similar depth. In our implementation, we integrated the glyph rendering into an existing order independent transparency pipeline. This allows us to react to situations, where glyphs are occluded by

surrounding parts of the vessel structure as in Figure 4.7, left. Here, vessels occluding a glyph are rendered with increased transparency. The transparency is unaffected at the glyph edge and increases towards the center. If the base circle (purple) is occluded, one can tell that it is actually behind the vessel. Hence, overlapping depth cues remain visible.

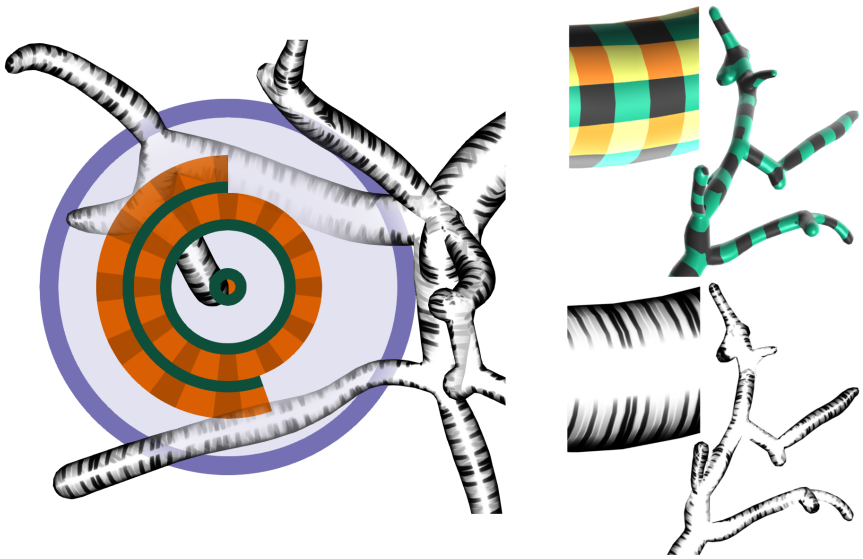


Figure 4.7: Surrounding vessels that occlude the glyph have their transparency increased (left). Visualization of the continuous parameterization based on Knöppel [103] (right, top). Hatching applied (right, bottom).

4.5.3 *Hatching*

In our visualization, the glyphs are desired to be the most prominent aspect. Therefore, we employ a less salient hatching technique that is based on a continuous parameterization, to draw the mesh. The parameterization is based on the work by Knöppel et al. [103]. Their method yields a globally continuous parameterization that is aligned with a given vector field. Using the principal curvature direction allows us to draw stripes with respect to the vessel geometry. Figure 4.7 (right) shows the parameterization and the hatching results. A 4-colored patch (yellow, green, black, orange in Figure 4.7, right top) represents texture coordinates in the range $[0, 1]$. Using these texture coordinates, we can render multiple strokes with varying length and intensity (see Figure 4.7, right bottom). The stroke variation is repeated for every 4-colored patch. We apply the hatching within a margin of the vessel contour that is nearly independent of the vessel thickness, by employing the method by Kindlmann et al. [101].

4.6 EVALUATION

To assess the performance of our visualization, we conducted a qualitative, comparative, online study. Generally, a comparison to a standard non-depth-enhanced (e.g. phong shading) and a depth-enhanced (e.g. pseudo chroma-depth [96]) visualization would be feasible to reveal the benefit of a new technique. Such evaluations have already been performed in the works by Lawonn et al. [122, 124]. The results of their questionnaires have proven that their illustrative visualizations can outperform standard techniques. Consequently, our analysis focuses on a direct comparison with the methods by Lawonn et al. Our evaluation is formulated in the same way as the previous ones by Lawonn in order to be comparable with the aforementioned standard techniques.

We implemented the depth-enhancement strategies found in [122] (*supporting lines*) and [124] (*supporting anchors*) along with our new technique. For five different data-sets, each including a vessel structure and tumor data, we randomly selected two locations to attach the supporting visualization. Then, images were captured for three camera positions for each of the three techniques. This totaled to 45 images that we used for the survey. Example images are shown in Figure 4.8. The approaches by Lawonn et al. both create a relation between the vessel structure and a supporting structure. Therefore, these techniques are expected to have less complication with overlapping vessel structures. Contrarily, our circle glyphs are placed directly at the vessel location that they represent. Hence, we assume that overlapping vessel structures will have a negative impact on their performance. The three camera positions were chosen such that the following cases were matched for the two selected vessel end-points and thus yield three difficulty levels:

1. High distance difference (L1, label FF/FN)
2. Circle glyphs partly obstructed (L2)
3. Very low depth difference (L3, label NN/FN)

Case 1 is the simplest task, and inquiries how fast decisions can be made with the respective visualization. With case 2 we aim to learn if overlapping is really a problem for the circle glyphs. The last case probes for the ability of the glyphs to distinguish just noticeable depth differences. The survey was implemented as an online questionnaire, because this enabled us to measure exact timings during the process. However, this means that we did an unsupervised study and did not observe the subjects. Subjects were first introduced to their task in a learning phase. For each visualization technique an example image was shown. The example image contained a description of the technique and two selected vessel end-points (as in Figure 4.8). Also, a

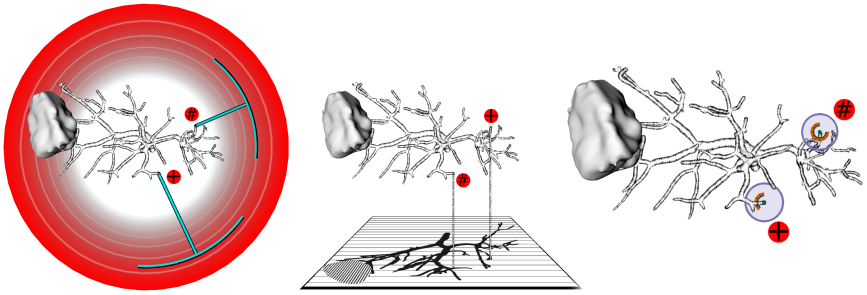


Figure 4.8: Supporting anchors (left), supporting lines (center), circle glyphs (right). Vessel end-points in question are tagged with (+) and (#).

solution to the question which vessel was further away, was included with an explanation. Then, subjects were shown two sample images per method and had to select the vessel that appeared to be further away. Answers during this training phase have not been included in the evaluation results.

After the training questions, participants were shown the 45 evaluation images in a fixed, but previously shuffled, order. Timings were measured from the point where an image was displayed until the subject selected an answer ((+) or (#) or *undecided*). Further, participants had to estimate their decision confidence with a 5-point Likert scale (1 = most inconfident, 5 = most confident). In the evaluation *undecided* answers were treated as a wrong answer, i.e., we were interested in the percentage of correct answers.

4.7 RESULTS

Our study was carried out with 24 participants (16 male, 8 female, average age 29.75 with a range of 22-57). Most of them (83%) came with a background in computer science and visualization. 29.1% of them had experience with visualizations of blood vessels. Nevertheless, we think that the results are also representative for expert users, since the depth perception is a general task and not only restricted to the medical context. Table 4.1 depicts the overall results of the study, averaged for each enhancement technique (15 questions each). Most correct answers were given with the proposed circle glyphs (93.6%), directly followed by the supporting lines (92.2%). The supporting anchors lead to 87.2% of right answers. Participants were most confident in their answers with the circle glyphs (4.35), succeeded by supporting anchors (4.21) and supporting lines (4.08). Hence, all averages are between the *very confident* (5) and *confident* (4) estimations. Subjects decided most quickly with our new method (5.92 s), closely followed by the supporting lines (6.09 s) and the supporting anchors (6.42 s) just behind. The timings for individual questions initially ranged from 1.8 to 126.2 seconds. We assume that some subjects were interrupted during the process, which lead to the high values. Hence, we computed

Table 4.1: Averaged results over all questions for each technique (Circle glyphs, Supporting Lines, Supporting Anchors).

	Lines	Anchors	Circle
Correct answer (%)	92.2	87.2	93.6
Confidence (1-5)	4.08	4.21	4.35
Time (s)	6.09	6.42	5.92

the 75% quartile (7.735 s) and removed outliers that lay above three times the 75% quartile (23.205 s). This high tolerance is in order to cover the more difficult questions. The portion of outliers was 1.82%. As mentioned in Section 4.6, the questions of the survey were subdivided into three levels of difficulty. Charts in Figures 4.9, 4.10, 4.11 show the evaluation results for the individual task levels L1, L2 and L3. The percentage of correct answers (Figure 4.9) is highest for the

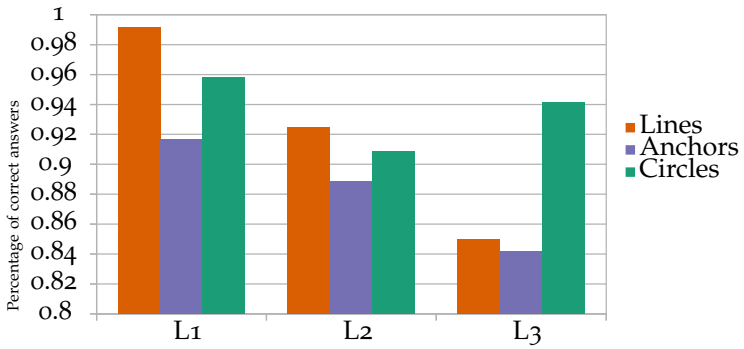


Figure 4.9: Estimation precision.

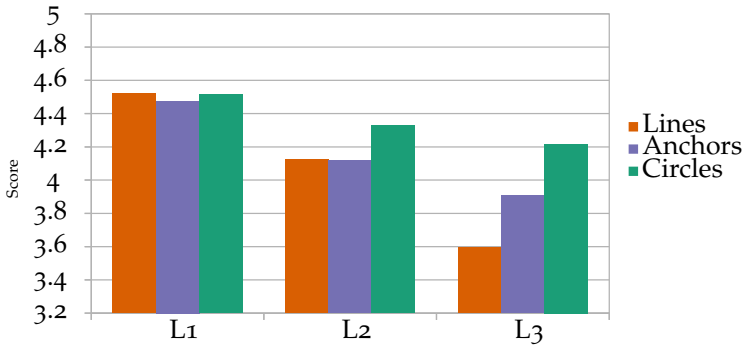


Figure 4.10: Confidence.

supporting lines in L1. This is likely due to the fact, that the shadow plane in the supporting lines visualization resembles depth cues in the most natural way. The confidence (Fig 4.10) in L1 is very similar among the techniques and higher than for the other levels. As L1 represented the easiest task, the timings (Figure 4.11) are relatively low compared to L2 and L3. But we can observe, that the supporting anchors lack behind the other two methods. For task level L2 we

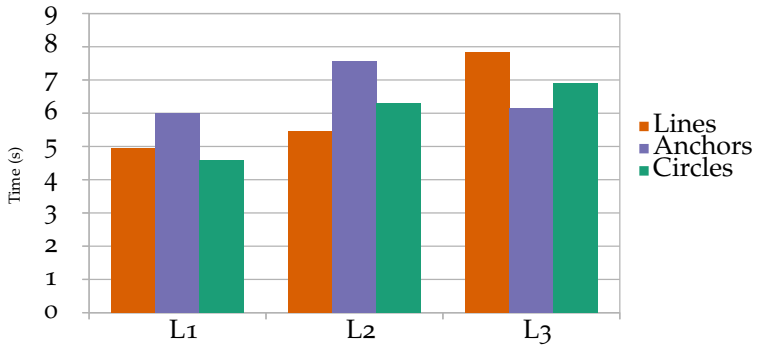


Figure 4.11: Task duration.

expected our new method to perform worse than for L1. Recall, that L2 contained situations, where the circle glyphs were partially obstructed. Since the depth difference was also smaller in this case, we can observe a negative impact on all three methods in the estimation precision, the confidence and the task duration. Contrarily to our assumption, the circle glyphs did not lose most precision (-5.56%) compared to supporting lines (-6.72%) and supporting anchors (-3.03%). Nevertheless, the fact that subjects were most confident with the circle glyphs but made the most correct choices with supporting lines in L2, underlines that overlapped glyphs were likely to be misinterpreted. The L3 task clearly shows that our approach outperforms the others in terms of accuracy. Here, 94.8% of answers were correct, while for the reference techniques 85.0% (supporting lines) and 84.1% (supporting anchors) were correct. This is accentuated by the participants' confidence. Surprisingly, the supporting anchors have now the lowest timings. A reason for this might be, that anchors were located very close to each other in L3. Thus, comparing them took less time than comparing two oppositely situated anchors.

In summary, and with the overall results in Table 4.1 at hand, we can state that all three techniques performed similarly well, with a slight advantage for the new circle glyphs. Only when it came to very small depth differences, our approach was able to outperform the others. We did not evaluate the performance of the glyph filtering technique with subjects but plan this for future work. The new approach does not require the solution of a bipartite graph matching as done for the supporting anchors [124]. Their method tries to find a best matching of two positional properties: The distribution of glyphs around the depth-cylinder, and the depth distribution (refer to Hall's marriage theorem). The graph matching is computationally more intensive and less flexible than our strategy. For example, as described in Section 4.3.2.2, we use the variable h_{min} to avoid overlapping of glyphs. Such an extension to the filtering approach can not be simply added to the bipartite matching. However, as we place glyphs on

Table 4.2: Glyph placement quality in terms of screen-space distribution.

	μ	σ^2
Bipartite	0.090	0.00210
Random	0.078	0.00241
Ours	0.104	0.00085

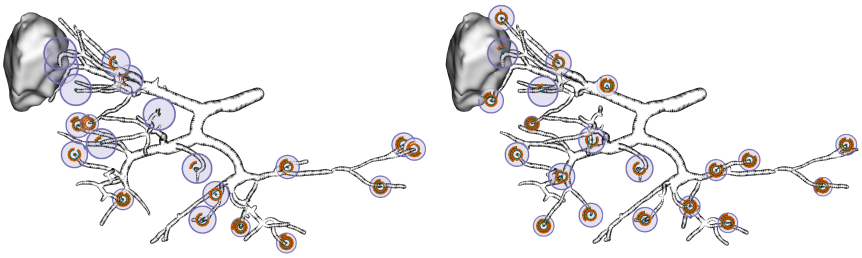


Figure 4.12: Comparison of glyph filtering methods. Bipartite matching as described in [124] (left). Our approach (right).

top of the vascular model, it is crucial that glyphs do not overlap, i.e., that glyphs have enough space around them. Hence, we are interested in maximizing the minimal distance among the visible glyphs. For an objective measurement, around 1000 random camera positions were generated for four different vascular models. For each scene, we used our technique, the bipartite matching and a random selection, to pick 16 glyph positions. We then computed the minimal distance of each glyph position to the remaining positions, to obtain variance and mean values. The results shown in Table 4.2 are averaged over the values obtained for all camera positions. The maximum possible distance between two glyphs is 1 (compare Figure 4.3, right). Also, for a uniform distribution of glyph positions, the minimal distance is maximized and equal for each sample. Thus, larger minimal distances (i.e., larger mean values) and a smaller variance can be interpreted as a hint for a more uniform distribution. From Table 4.2 it it can be observed, that the averaged minimal distance of our method is 16% larger than for the bipartite matching and 33% larger than for the random picking. More significant is the difference in variance σ^2. Here, σ^2 for the bipartite matching exceeds our approach by 145%, while the random picking yields a 181% larger value. Thus, we conclude that our technique results in a more balanced distribution across the screen-space, that is suitable for our glyphs. A comparison of glyphs placed with our technique and the bipartite matching is shown in Figure 4.12.

4.8 DISCUSSION AND FUTURE WORK

We have proposed a vessel end-point detection algorithm as an offline pre-processing step. These locations are further filtered by an online feature-driven and graph-based approach to select a subset of vessel end-points that cover the vasculature in a representative way. To support spatial perception of the presented 3D model, we place circle glyphs at the filtered locations. The glyphs are designed with respect to aspects of the pre-attentive and attentive phase in human perception. Further, our parameter mapping employs a combination of a discrete and continuous mapping to allow for a precise reading of the mapped parameter. An advantage over the reference techniques *supporting lines* and *supporting anchors* lies in the simplicity of the glyph design. We do not need to add any supporting geometry (like the shadow plane or cylinder) to the scene. Our glyphs work independently and 'in place'. As we omit such additional geometry, we can use the glyphs at any zoom level. The shadow plane for the supporting lines in contrast, is only reasonable when the whole vasculature is visible. However, the circle glyphs come with a deficiency. The angular attribute of the inner concentric circles has no relation to natural depth perception. Thus, it requires additional cognitive effort to correctly interpret filled and empty glyphs. An issue with positioning glyphs right at the position they refer to is overlap or obstruction by other structures. We have approached this problem by rendering occluding geometry with increased transparency. While the glyph itself is then better perceivable, the impression of the overall visualization might suffer.

The graph-based glyph filtering method proposed in this work takes different features into account to select a specific subset of possible glyphs. This formulation is flexible, as further features could be added to modify the resulting edge costs in $\mathcal{G}$. We conducted a quantitative evaluation and were able to show that the proposed circle glyphs are suitable for a precise and quick estimation of relative depth. We expect that the glyphs perform similar for representation of scalar fields in general. A deficiency of the evaluation is that we only tested for scenes with two glyphs visible. In a situation where multiple glyphs (10-15) are shown to allow for a general overview of the data in question, the results might be different. In this case the supporting lines and anchors probably overlap in many locations and it becomes very hard to obtain a satisfactory estimation of depth.

The example in Figure 4.5 shows, that our glyph can be modified with low effort to represent three scalar magnitudes. However, the performance of the these extended glyphs has to be assessed in another evaluation. Furthermore, a subsequent study should investigate the discretization of the (so far) continuous mappings to geometrical features. E.g., the depth dependent glyph size could be mapped to

ordered bins, so that an ordering of multiple glyphs is perceivable, even if the actual (un-binned) depth differences are very low. Apart from this, our glyphs embody a simple to implement, effective and flexible representation for scalar values on arbitrary meshes.

ACKNOWLEDGEMENTS This project was partly funded by the DFG: LA 3855/1-1 and HA 7819/1-1. We would like to thank the subjects of our evaluation for their contribution.

5

PARAMETERIZATION, FEATURE EXTRACTION AND BINARY ENCODING FOR THE VISUALIZATION OF TREE-LIKE STRUCTURES

ABSTRACT The study of vascular structures, using medical 3D models, is an active field of research. Illustrative visualizations have been applied to this domain in multiple ways. Researchers made the geometric properties of vasculature more comprehensive and augmented the surface with representations of multivariate clinical data. Techniques that head beyond the application of color-maps or simple shading approaches require a surface parameterization, i.e., texture coordinates, in order to overcome locality. When extracting 3D models, the computation of texture coordinates on the mesh is not always part of the data processing pipeline. We combine existing techniques to a simple parameterization approach that is suitable for tree-like structures. The parameterization is done w.r.t. a pre-defined source vertex. For this, we present an automatic algorithm, that detects the tree root.

The parameterization is partly done in screen-space and recomputed per frame. However, the screen-space computation comes with positive features that are not present in object-space approaches. We show how the resulting texture coordinates can be used for varying hatching, contour parameterization, display of decals, as additional depth cues and feature extraction. A further post processing step based on parameterization allows for a segmentation of the structure and visualization of its tree topology.

5.1 INTRODUCTION

The visualization community actively works on techniques for the display of vascular 3D models. The motivation for this is based on the clinical relevance for surgical planning and guidance of interventions. Different imaging modalities [123] and also physical simulations [160] contribute to the generation of data that is desired to be visualized along with medical volume or surface data. However, visual information channels are limited, and therefore new techniques emerge that aim to ease the perception and comprehension of task oriented data [175]. This also plays an important role in medical education [174], where illustrative techniques are found to highlight certain data features or to guide the attention of the viewer. Understanding the spatial structure of a given 3D object is an important aspect, especially if the user looks at the 2D projection on a common computer

monitor. In this case depth cues are missing which then have to be encoded in another way. Another broad topic is the visualization of blood flow [152]. In this area users are interested in spatial data inside a blood vessel, but also want to obtain information about surface related aspects. To encode the multitude of available data, advanced (illustrative) visualization techniques can be applied [128]. We can generally state that advanced techniques require parametric guidance on the rendered surface data. This may be texture coordinates (to display textures or patterns) or tangent vector fields (to guide pattern orientation or streamline generation). Unfortunately, such parameterizations are not always available from the medical data acquisition pipeline. Furthermore, texture coordinates can be generated in various ways, exposing different advantages and disadvantages.

In this work, we introduce a technique to generate texture coordinates that is suitable for the processing of tubular, tree-like structures, e.g., blood vessel surface data. Because of the restricted target morphology that our algorithm is designed for, we can take advantage of that morphology and come up with a very simple approach, utilizing existing and more generally designed algorithms. The initialization of the algorithm can be automated by a method that detects the end-points and root of a tree structure. Additionally, the resulting parameterization can be used for the extraction of branch locations of the mesh. We also show how we can use the output of our algorithm as an additional depth cue for a scene and the parameterization of contours. In this way, our approach can be used for the improvement of structure and depth perception, and the application of illustrative rendering techniques in order to encode multivariate data. The parameterization can be used as input for a segmentation algorithm that subdivides the vasculature at branching points. The resulting segments are arranged in a binary tree hierarchy that enables further visualization applications Thus, our main contributions are:

- A simple approach to generate texture coordinates on tree-like structures.
- An automatic, parameter free branch- and end-point detection algorithm.
- As an extension to [133], we apply a segmentation that results in a binary tree and can be used for the visualization of the vascular tree topology.

The results are brought to use in several examples.

5.2 RELATED WORK

The parameterization of surface meshes has been researched for a long time [47]. In the past this field was highly motivated by the

topic of remeshing algorithms [16]. For remeshing purposes it is usually sufficient to come up with parameterizations that are locally bijective. Globally, 1-to-n mappings from the parameter space to the domain may exist (as in a periodic function). For the display of textures or decals on a surface this can be a disadvantage, because different locations on the surface cannot be sufficiently distinguished in the parameter space. However, such periodic texture coordinates can be useful in the visualization domain, as shown by Knöppel et al. [103] and Lichtenberg et al. [138]. Other techniques aim to find a 1-to-1 mapping between the parameter and domain space. An important one to mention in this context is the one by Kälberer et al. [93]. They cut open a given mesh to obtain disc-topology and then apply an iterative integration of texture coordinates. Then, discontinuities at the cut seams are removed by a repair operation. While their approach yields reasonable results in the sense of the pure mapping, the algorithm itself relies on three complex steps and the iterative nature of the algorithm restricts it from scaling well with the mesh size. In our approach, the units in parameter space and domain space are approximately equal, which can be of advantage when mapping textures or patterns w.r.t. the object size. Another possibility is to calculate texture coordinates locally in screen-space and has been proposed by Rocha et al. [185]. They sample the surface and render spheres for each sample to activate the fragment shader. Texture coordinates can then be approximated for the activated pixels in order to draw small decals on the surface. These decals are used to represent multivariate data values.

As mentioned in the introduction, the spatial perception on 2D monitors is restricted due to missing depth cues. Consequently a range of publications can be found in the literature that address this problem, using different kinds of artificial or nature-inspired ways to encode depth. Kersten-Oertel et al. [96] provide a good reference for the performance of different depth cues in the vascular domain. Using the color channel, pseudo chroma-depth proposed by Ropinski et al. [189] employs aspects of the natural perception to intuitively map depth to a red-to-blue color scale. Applying pseudo chroma-depth to a whole mesh, however, impairs the ability to use other shading techniques to convey structure or to display additional information on the surface itself. To cope with this issue, Behrendt et al. [6] proposed to apply pseudo chroma-depth only to the contour region of a mesh to make space for supplementary data. Apart from coloring or texturing a given surface to convey data, additional geometry can be added as glyphs to an existing scene [188]. Then, one has to decide how many or where glyphs should be placed in order to achieve a clean and informative result. Later, we will present a simple method to extract branch locations and vessel end-points from a vascular mesh. Being structurally significant locations, these could be used for glyph place-

ment, as done by Lichtenberg et al. [132]. With their work, they followed up on previous methods by Lawonn et al. [122, 124], who used additional geometry in a scene to improve the spatial perception. The problem of spatial perception can be omitted when transferring the visualized data to the 2D domain, as surveyed by Kreiser et al. [109].

When color is used to encode data on a surface a trade-off has to be made. Either the colormap is disturbed by an additional shading, or geometric features are not perceivable due to missing shading. A workaround is then to use hatching strokes, which do not interfere with the colormap but are still able to depict the geometry. Hatching along a 3D surface has first been done by Hertzmann and Zorin [75] who computed integral lines along principal curvature directions. Praun et al. [172] followed with a texture based approach. Further insights into this illustrative rendering area can be obtained from the survey by Lawonn et al. [128] while an example of illustrative rendering in the context of vascular models is given by Ritter et al. [184] .

Scalar functions that are defined on a surface have been used for shape analysis. The *Reeb Graph* (RG) emerged from Morse theory [177] and has developed into an application for 3D shape topology analysis [167, 204]. The RG captures topological information of a surface based on the level-sets of a continuous scalar function defined on the surface. The *Contour Tree* (CT) [22] is a special case of the RG that does not allow cycles, i.e., a binary tree is constructed. We use an implementation by Carr et al. [26] to capture the topology of our tree structures.

5.3 METHOD

Our goal for this work is to obtain texture coordinates $\mathcal{T} \in \mathbb{R}^2$ for the visualization of a tubular and tree-like mesh $\mathcal{M}$. The two dimensions of this set are denoted by $(U, V) \in \mathcal{T}$. Important for our work here are the *Geodesics in Heat* (GiH) method by Crane et al. [35] and the *Jump Flood Algorithm* (JFA) by Rong and Tan [187]. These two algorithms have a very general domain of application and provide the basis for our parameterization of U and V. Further, we use the resulting U parameter for a segmentation based on a CT, for which we use the algorithm by Carr et al. [26]. First, we obtain an *augmented* CT $\hat{\mathcal{Y}}$ that contains a node for every vertex in $\mathcal{M}$. A segmentation of $\hat{\mathcal{Y}}$ yields the CT $\mathcal{Y}$ that contains an edge for every segment obtained from $\hat{\mathcal{Y}}$. More details on the CTs follow in Section 5.3.2.

5.3.1 *Parameterization*

This section covers the computation of the (U, V) coordinates for $\mathcal{M}$. Our approach is split into an object-space and a screen-space part.

The rationale for this will be clarified in the respective paragraphs that follow.

OBJECT-SPACE: U We intend to compute U as a continuously increasing scalar field across the mesh. The source of U (where $U = 0$) is a pre-defined vertex $\mathbf{v}_s$, which should be placed at the root (i.e., the source of blood flow) of a given vessel, to achieve the most intuitive results. We provide an automatic root-point detection algorithm that will be described later in this section. Our approach is now to define a vector field $\mathcal{Z}$, so that $\nabla U = \mathcal{Z}$ (i.e., the gradient of U equals $\mathcal{Z}$). Hence, the quality of U is determined by the quality and orientation of $\mathcal{Z}$. To define $\mathcal{Z}$ we first compute the directions of minimal curvature $\mathcal{C}$ for each triangle of $\mathcal{M}$ by applying the method presented by Rusinkiewicz [193]. The orientation of individual elements $\mathbf{c}_i \in \mathcal{C}$ is ambiguous (i.e., $\mathbf{c}_i \sim -\mathbf{c}_i$). To resolve the ambiguity, we compute the geodesic distance $\mathcal{G}$ of each vertex to $\mathbf{v}_s$, using the GiH method [35]. The GiH utilizes a relationship between heat transport from the physics domain and distance, achieving quick approximations or even exactly computed geodesic distances on arbitrary topologies through differential operators. Then, $\nabla\mathcal{G}_\Delta$ is computed as the gradient of $\mathcal{G}$ for each triangle. We then obtain $\mathcal{Z}$ element-wise as:

$$\mathbf{z}_i = \begin{cases} \mathbf{c}_i, & \text{if } \langle \mathbf{c}_i, \mathbf{g}_i \rangle \geqslant 0 \\ -\mathbf{c}_i, & \text{otherwise} \end{cases} \tag{5.1}$$

where $\mathbf{g}_i \in \nabla\mathcal{G}_\Delta$. By incorporating $\mathcal{C}$, we make sure that $\mathcal{Z}$ well describes the geometry of $\mathcal{M}$. Using $\nabla\mathcal{G}_\Delta$, we force $\mathcal{Z}$ to have a single source at $\mathbf{v}_s$. Finally, we smooth $\mathcal{Z}$ to remove noise and high frequency features of the vector-field. We do an element-wise Lapacian smoothing in the dual graph of $\mathcal{Z}$ (i.e., per vertex). The result is then lifted back to each triangle, while keeping $\mathcal{Z}$ in the tangent space of $\mathcal{M}$. Now, we can put $\mathcal{Z}$ into an equation system to compute U, such that $\nabla U = \mathcal{Z}$. Applying the divergence on both sides yields the Poisson equation

$$\Delta U = \nabla \cdot \mathcal{Z}. \tag{5.2}$$

Solving Eq. 5.2 in a least squares sense results in the coordinate U. As a last step, we rescale U to the range of $[0, \max(\mathcal{G})]$. This brings U into a relation of spatial distances along the surface of $\mathcal{M}$. Figure 5.1 shows an example result for U. It can be observed that U increases strictly monotonously away from $\mathbf{v}_s$. Generally, we can compute the distances w.r.t. an arbitrary point, but the results are most intuitive if a vessel end-point is chosen as the source. We have to note that setting $U = \mathcal{G}$ works fine for very straight structures. In these cases the geodesic distances sufficiently capture the mesh structure. This is not true for more convoluted vasculature, which brings up the necessity to incorporate $\mathcal{C}$. Figure 5.2 shows the final U compared to an

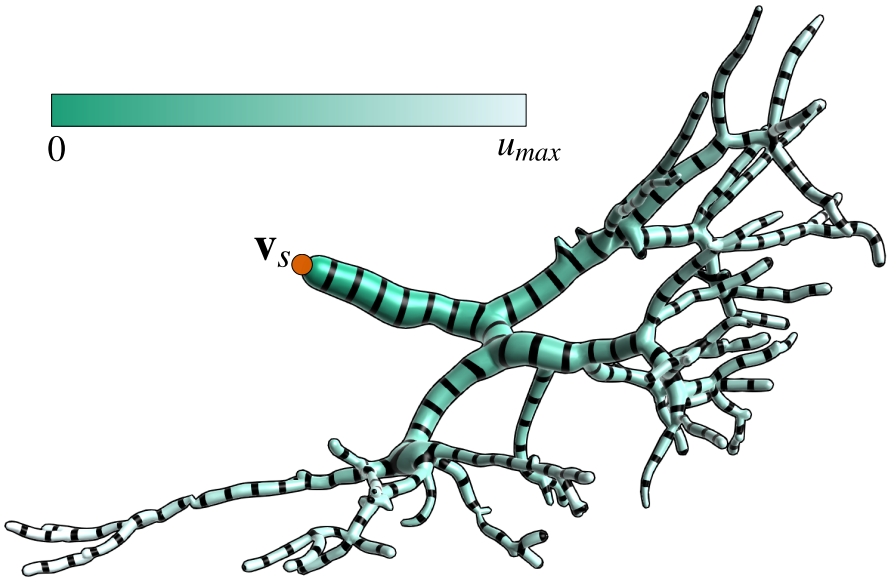


Figure 5.1: Resulting U for a given source vertex (marked orange). Isolines underline the smoothness of the result.

anticipated geodesic between two points on the surface. The geodesic in this convoluted section would not properly cover the geometry. The singularities found in U can be used for extraction of structural features as described in Section 5.4.1. Note that the assignment of U could be described as a semi-global parameterization. It is global in the sense that a set of isolines with unique U-values can be found on the shortest path from $\mathbf{v}_s$ to any given vertex. However, isolines of a certain U-value are not unique on paths to multiple vertices that are found on different branches of the tree-structure. This means that we can identify unique, individual isolines per branch segment, but not for the whole mesh.

OBJECT-SPACE: ROOT-POINT Instead of defining $\mathbf{v}_s$ manually, we provide an automatic approach to find the root of an input mesh. The algorithm consists of two steps:

1. Identify all vessel end-points.
2. From the given end-points find the root.

While previous approaches [124, 132] presented methods to find end-points, we propose a faster approach to identify these points. Lawonn et al. [124] determined the shape index of all points, then the Otsu method was applied to extract regions around the end-points and finally a shrinking was applied to get one end-point per region. Lichtenberg et al. [132] improved the technique by incorporating topological information via geodesic distances and graph analysis. Nevertheless, their approach requires some pre-defined parameters that may depend on the input mesh, making it less robust. For the first step

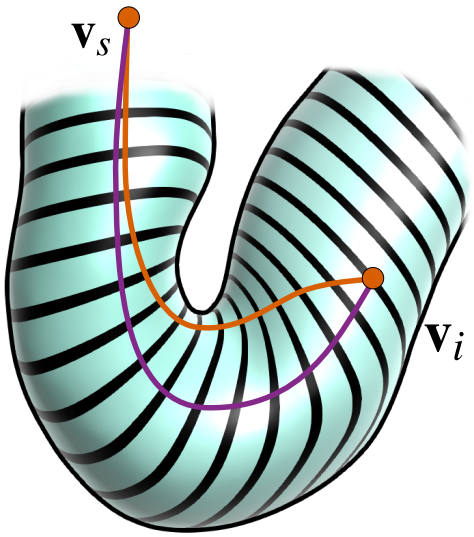


Figure 5.2: Delineated geodesic from a vertex $\mathbf{v}_i$ to a source $\mathbf{v}_s$ (orange) compared with the path along the gradient of U (purple).

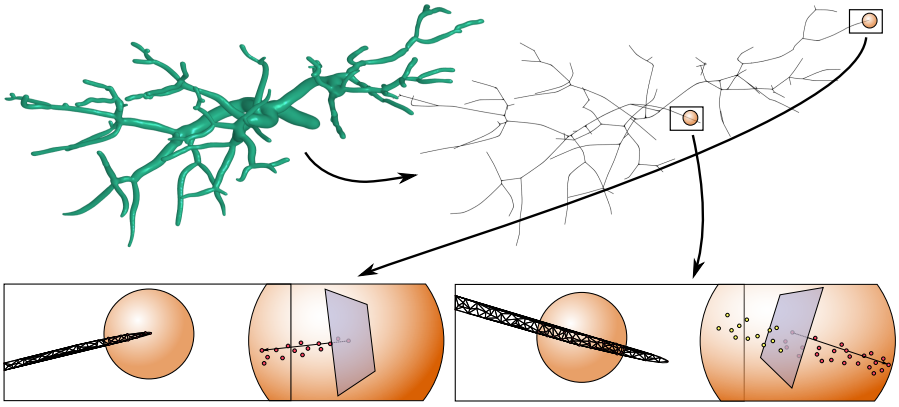


Figure 5.3: First, we apply a thinning to the mesh (top, right). Afterwards, a sphere is created for every point to test if the created plane separates the points inside the sphere (bottom). If this is not the case, an end-point is found.

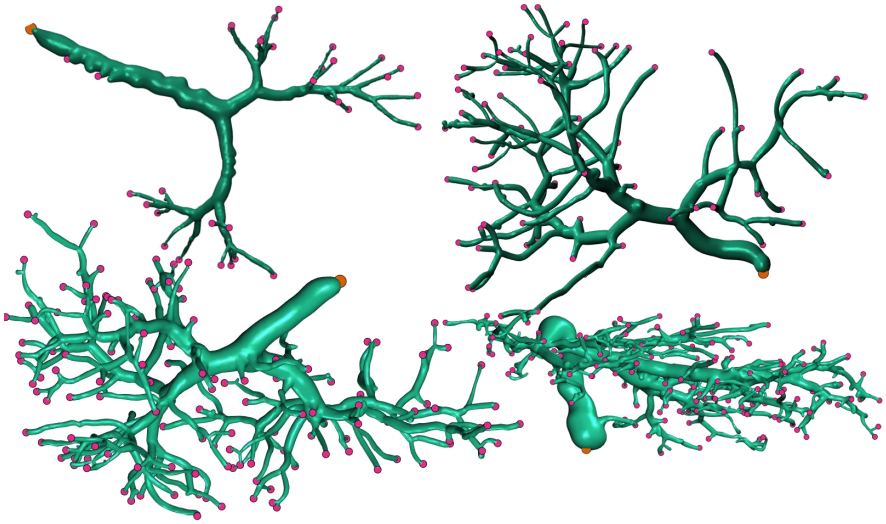


Figure 5.4: Result of our root-point (orange) detection for various meshes. Other end-points are marked magenta.

of our method we apply a thinning to the entire mesh, following the approach by Au et al. [3]. Their method is based on an iterative shrinking determined by

$$\begin{pmatrix} \mathbf{W_H L} \\ \mathbf{W}_H \end{pmatrix} \mathbf{p}' = \begin{pmatrix} 0 \\ \mathbf{W_H p}, \end{pmatrix} \tag{5.3}$$

where $\mathbf{W_L}$ and $\mathbf{W_H}$ are diagonal weighting matrices that change per iteration. For every iteration, Eq. 5.3 is solved in a least squares sense, resulting in Figure 5.3 (top, right). Afterwards, we iterate over every point $\mathbf{p}_i$ and create a sphere B_i with radius $r = k\bar{\mathbf{e}}$, where $\bar{\mathbf{e}}$ is the average edge length of the thinned mesh. The factor $k = 3$ was chosen empirically to set r well above the average edge length in the thinned mesh. This ensures to capture enough sample points in the following step. We extract all points $\mathbf{p}_j^i$ on the contracted mesh that lie inside the radius of each sphere B_i (see Figure 5.3, bottom). Finally, we take the vector $\mathbf{n}_i = \mathbf{p}_j^i - \mathbf{p}_i$, with $j = \text{argmax}_{\mathbf{p}_j^i \in B_i} \|\mathbf{p}_j^i - \mathbf{p}_i\|$. If all points $\mathbf{p}_j^i$ lie in the same half space created by the plane with the origin $\mathbf{p}_i$ and normal vector $\mathbf{n}_i$, then $\mathbf{p}_i$ is an end-point. Otherwise it is not an end-point (see Figure 5.3, bottom). With this, a parameter free detection of end-points is possible. A root-point can be found as the end-point where the absolute mean curvature is minimal, i.e., where the approximated vessel diameter is maximal. Several example results are shown in Figure 5.4.

SCREEN-SPACE: V The task of obtaining the V coordinate in object-space is conceptually more involved. In an optimal scenario, the gradient of V would be orthogonal to the gradient of U, and therefore be aligned to the direction of maximal curvature in our scenario. This

means that V evolves along the circumference of the vessel structure, which will result in discontinuous seams, where minimum and maximum values of V meet. Previous work by Kälberer et al. [93] has tackled this problem by repairing the parameterization in order to match seams with integer values. Another approach has been proposed by Ray et al. [176] and followed up upon by Lichtenberg et al. [138] who used periodic functions to avoid seams. Their methods, however, are more complex to implement and yield parameterizations with isolines that are only unique within a single periodic interval.

Instead of treating seams and singularities of V in object-space, we utilize the JFA to compute V in screen-space. This choice removes the necessity to treat seams in the parameterization, because we are only looking at the flat projection of the mesh. Furthermore, the algorithm is relatively easy to implement and executes in a few milliseconds on modern consumer graphics hardware. The drawback is that V has to be recomputed each time the projection of the mesh changes.

In our approach the JFA is used to compute the minimum distance of each pixel to the contour of a mesh $\mathcal{M}$. Thus, V will always be zero at locations next to contours, regardless of the rotation of the camera or $\mathcal{M}$, and increase away from contours (see Figure 5.5). The JFA is a method that works on grid structures and is capable of distributing content of a node in $\log_2 n$ iterations to all other nodes. If we render to a view port of resolution (x, y), then $n = \max(x, y)$. As described in their paper, the JFA [187] can be used to assign each pixel to a Voronoi cell of a set of seed pixels. For each pixel, the final result contains the minimal distance of a pixel to its referred seed pixel. In our case, the contour of a mesh is rendered and the resulting pixels are used as the set of seed pixels. Hence, after running the JFA, each pixel contains its minimal distance to the contour. A first result is shown in Figure 5.6 (left), where thin black strokes represent isovalues of V. Here, two overlapping vessel segments are shown and the result is not optimal. Some pixels that belong to the segment in the background (horizontal) are parameterized based on the distance to the contour of the foreground (vertical) segment. We address this with a

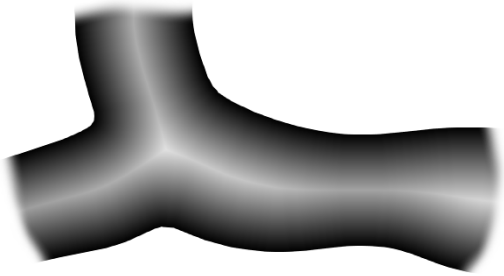


Figure 5.5: Minimal distance of each pixel on the mesh projection to the projected mesh contour, mapped to a gray scale.

single modification: As an additional parameter, we store the depth of the pixels in camera space. Now, pixels that are not part of the contour are only allowed to reference contour pixels, that have a *higher depth value*. Assuming that the 3D model has no self-intersections, we use this modification to make sure that each pixel cannot reference parts of the contour that belongs to a closer, overlapping vessel segment. The result of this change can be seen in Figure 5.6 (right). Note

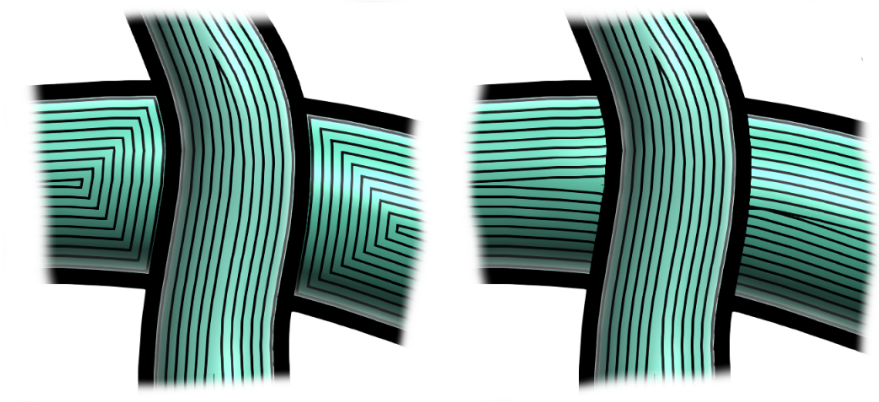


Figure 5.6: Unmodified JFA (left) and modified JFA, incorporating pixel depth (right).

that more complex overlaps result in less meaningful approximations of V, which we will address in Section 5.6. Finally, we rescale V so that (U, V) is isotropic:

$$V \leftarrow V \cdot d_p \tag{5.4}$$

where d_p is a local (per pixel) approximation of the distance of two points in world space (i.e., on the actual surface of $\mathcal{M}$), represented by two neighboring pixels. Therefore, units of V are mapped to world space as well. A combined example showing the results of the object- and screen-space parameterization is shown in Figure 5.7.

PARAMETERIZATION OF REMAINING PIXELS The results shown so far only include V for pixels representing the surface of $\mathcal{M}$. However, since the JFA is executed on the GPU for *all* pixels, we also obtain a parameterization for pixels that are otherwise discarded (i.e., the background pixels). Section 5.4.2 shows how this can be utilized.

5.3.2 *Graph generation and segmentation*

The isolines of a certain U-value are not bound to form a connected component. This is the case when isolines of a certain value are found on different branches of the vascular tree. This ambiguity can be resolved by constructing a CT $\mathcal{Y}$ based on U. We use the algorithm proposed by Carr et al [26] to do so on a surface mesh. For this, the

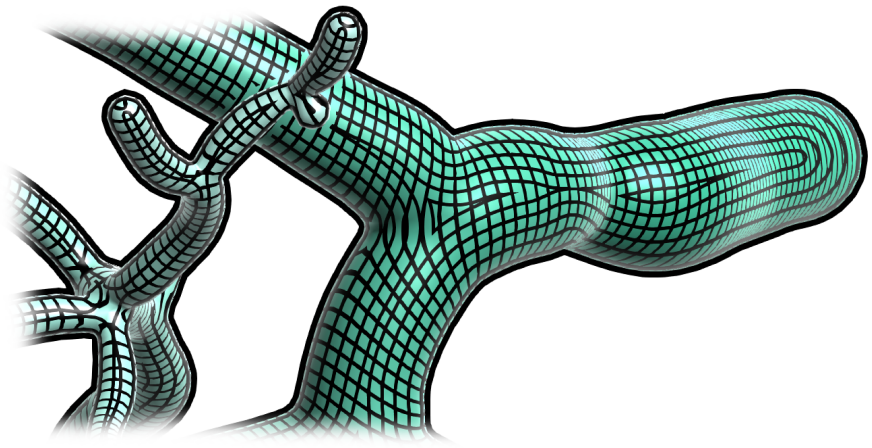


Figure 5.7: Example result showing isolines for (U, V).

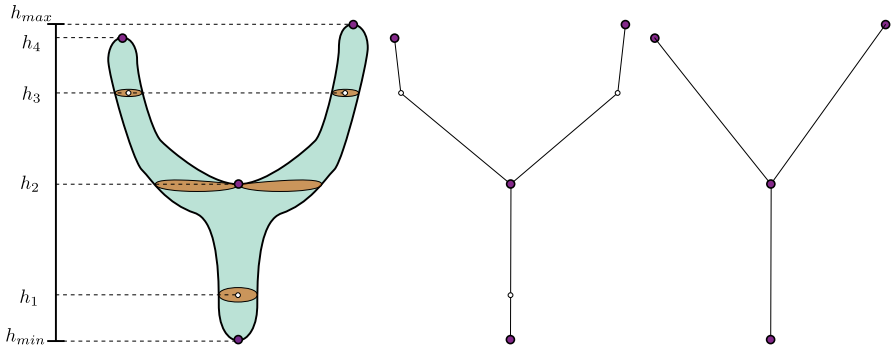


Figure 5.8: Level sets depicted for a height field h (left). Critical points (purple) as well as regular points (white) are present in the augmented CT $\hat{y}$ (center). The CT y only contains the the critical points.

U-values associated with individual vertices have to be unique across the mesh $\mathcal{M}$. If this is not the case, small permutations of duplicate values can be applied to solve the issue. The CT represents the topological changes of the level sets of U (i.e., the connected isolines). The nodes of the CT are associated with vertices of $\mathcal{M}$ that represent local minima, local maxima or saddle points of U. These points are also known as critical points from Morse theory [155]. As shown in Figure 5.8, the topology of the level sets for a certain scalar value changes at critical points (purple). Following the height field h (see Figure 5.8) from its minimal to maximal value, a level set is created at the minimum h_{min}, splits into two sets at a saddle point h_2 and vanishes at the two local maxima h_4 and h_{max}. These critical points appear as nodes in the CT y (Figure 5.8, right). Figure 5.8 (center) depicts the idea of the *augmented* CT $\hat{y}$, which is the primary output of the algorithm [26]. The augmented CT $\hat{y}$ contains critical nodes (purple) with degree equal to one or three, as well as regular nodes (white) with degree equal to two. In practice, all vertices of an input mesh, that

are not critical points, are represented by regular nodes. For clarity, Figure 5.8 (right) contains only one example regular node per edge. Each regular node resides on the path between two critical nodes. The scalar value of a regular node will always be between the scalars of the two associated critical nodes.

We use $\hat{\mathcal{Y}}$ to trace the paths emanating from each critical point, beginning at the lowest U value, i.e., the vessel root, and process the remaining points in order of increasing U (or increasing h as in Figure 5.8). In this way, we find all nodes (i.e., vertices) that lie on a path i between two critical nodes and aggregate these in a segment with index i. The terminating critical node of each path is also included in the segment. With this, a segment represents a connected space of associated vertices with $U_{max} \geqslant U > U_{min}$, where U_{max} and U_{min} are the U values of the two critical points. The node representing the root vertex (i.e., the vertex with the global minimum of U) is a special case because it has no edge incident from a node with a lower U value. It is inserted into the segment emanating from itself, i.e., the root segment. Segments and their associated nodes and edges are depicted in Figure 5.9 (left) with an orange underlay. If we refer to a segment i of $\hat{\mathcal{Y}}$, we refer to all vertices of $\mathcal{M}$ that were aggregated into that segment. By *reducing* all regular nodes from $\hat{\mathcal{Y}}$, we obtain $\mathcal{Y}$. The *reduce* operation maintains the connectivity of the tree: removing a node with neighbors A and B creates a new edge between A and B. Thus, a segment i of $\mathcal{Y}$ includes a critical node i and its incident edge. Finally, a segment i of $\mathcal{Y}$ is a representative for all vertices in $\mathcal{M}$ that are associated with the segment i in $\hat{\mathcal{Y}}$. Instead of referring to individual nodes or edges of $\mathcal{Y}$ and $\hat{\mathcal{Y}}$, we will refer to these segments throughout the paper. Since $\mathcal{Y}$ is a binary tree, we obtain $n-1$ segments, if n is the number of critical points.

The binary structure of the tree further allows us to augment each segment with a binary code B_i, that represents the path to reach that segment from the tree root. The root is represented by a single bit set to 1. The transition to the first/second child segment is denoted with a bit set to 1/0, as depicted in Figure 5.9 (left). Consecutive bits are appended to the right. The length of the binary code is equal to the layer $l(B_i)$ that the segment occupies within the tree, with $l(B_i) = a_i + 1$, where a_i is the number of ancestor segments of segment i. Hence, $B_i \in \{0,1\}^{l(B_i)}$. Based on that binary code, a simple bit-shift operation can be implemented to find out whether a segment is an ancestor or descendant of another segment. The example in Figure 5.9 (right) shows two binary codes B_1 and B_2, with $l(B_1) = 2$ and $l(B_2) = 5$. We then shift the binary code B_2 associated with the higher layer ($l(B_2) > l(B_1)$) by k right-shift operations, where $k = |l(B_1) - l(B_2)|$ is the layer difference, i.e., three in the example. This aligns the highest set bits. If the codes of B_1 and B_2 are now equal, the segment of the lower layer (B_1) is an ancestor of the segment in the higher level (B_2).

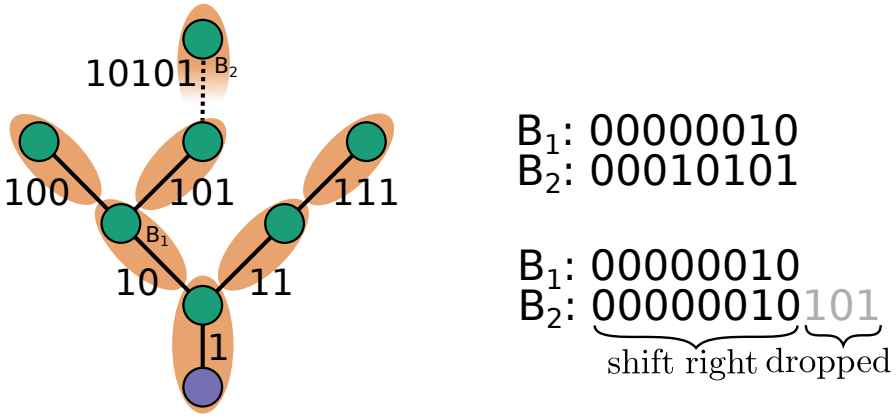


Figure 5.9: Binary encoding for CT segments (left, segments underlain orange). The root segment contains two nodes, including the root node (purple). Determination of the segment relationship by bit-shift operations (right).

Otherwise, the two segments are not found in the same sub-tree. We will address the utility of the binary code and the bit-shift operation in Section 5.4.

5.4 APPLICATIONS

This section describes various applications of the (U, V) parameters that we obtained in Section 5.3. We provide a detection of branch locations and vessel end-points, as well as a hatching technique and the rendering of parameterized contours and depth cues. These techniques may be useful for, or inspire, a range of visualization approaches in the context of vascular visualization.

5.4.1 *Branch and end-points*

We can use U as obtained in Section 5.3.1 to extract branch locations as well as vessel end-points. For this, we look at the one-ring of each vertex. It can be observed that vessel end-points are found at local maxima of U and branch points are found in saddle-shaped configurations of U. This is due to the use of $\nabla \mathcal{G}_\Delta$ in Section 5.3.1, which creates a sink in the vector field $\mathcal{Z}$ at local maxima of $\mathcal{G}$. As depicted in Figure 5.10, we can visit the ordered vertices $\mathbf{v}_0$ to $\mathbf{v}_n$ of the one-ring of vertex $\mathbf{v}_i$ and check for the difference of the U value. If $U(\mathbf{v}_i)$ is local maximum, a vessel end-point has been found (see Figure 5.10, left). If we find four regions of differing sign of d_{ji} along the one-ring, then a saddle shaped area, and hence a branch location, has been detected (see Figure 5.10, right). Here, $d_{ji} = U(\mathbf{v}_j) - U(\mathbf{v}_i)$, where j are vertices of the one-ring of $\mathbf{v}_i$. The points found here may be used for the placement of glyph objects, as in the work by Lichtenberg et

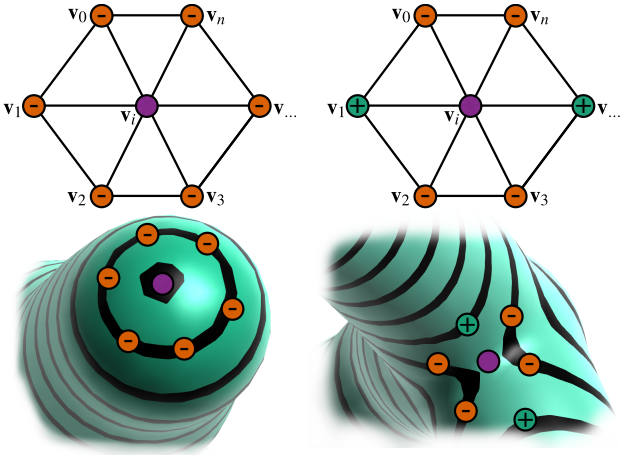


Figure 5.10: Configuration of U to detect vessel end-points (maximum, left) and branch locations (saddle, right). The reference vertex is colored purple and vertices of the one-ring with higher or lower values U are marked with a + or - sign, respectively.

al. [132], since the vessel end-points and branch locations are structurally significant features.

5.4.2 *Using V to enhance depth perception*

In Section 5.3.1 we mentioned that the JFA also assigns V values to pixels that are not part of the surface representation (i.e., background pixels). Thus, background pixels also refer to their closest contour pixel. In order to incorporate the depth information of the contour pixels, we can modify the distance function used for the background pixels' JFA iterations. Instead of searching for the contour pixel with the minimal euclidean distance d_c, we incorporate the depth, e.g. $d_c \leftarrow d_c \cdot d^a$ and determine the minimum of that term to obtain V. Here, d is the depth of a contour pixel and a controls the impact of the depth difference. With $a = 3$ we achieve a depth aware parameterization of the background pixels as shown in Figure 5.11. The result appears similar to the *Void Space Surfaces* by Kreiser et al. [108]. The difference is, however, that we do not extend the depth information into the background pixels with a smart interpolation approach. Our combination of depth and distance information results in distinguishable regions that refer to individual branch regions. The boundary of these regions are visible through the abruptly changing isolines. This can, for example, be used to draw outlines or glow effects around a given structure. Figure 5.12 shows the effect, where a larger glow radius (in screen-space) refers to less distance to the viewer. In this way, one can easily obtain information about the global orientation of a visualized structure.

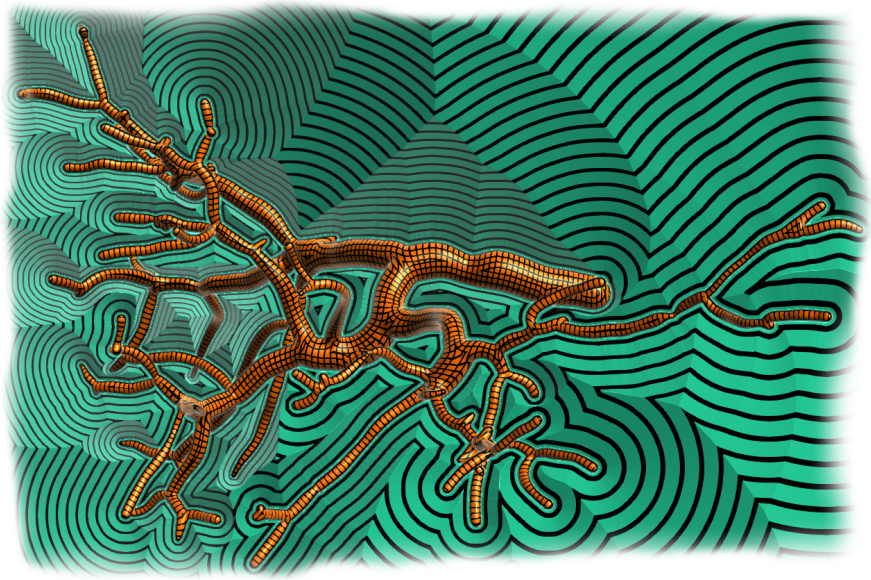


Figure 5.11: The parameterization of the background pixels in dependence of the contour depth values. Spacing of the isolines is larger in regions that are closer to the viewer.

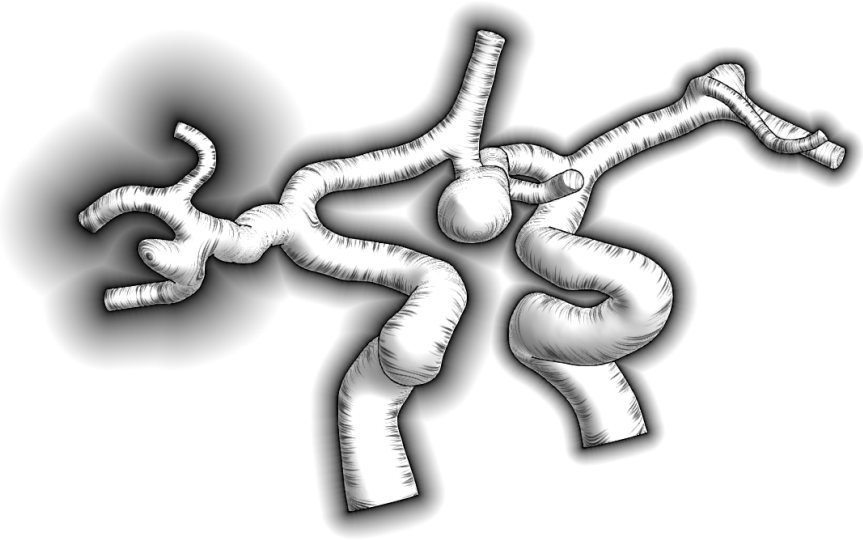


Figure 5.12: The parameterization of the background pixels is used to draw a depth dependent glow. The surface itself is rendered with varying hatching strokes.

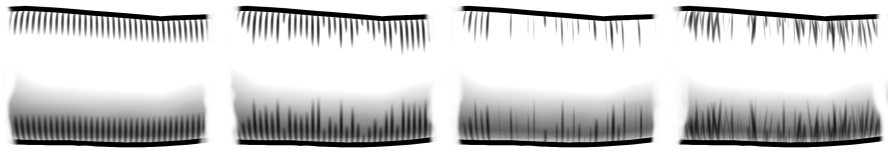


Figure 5.13: Different generic variations to the hatching strokes: no variation (left), varying length (center, left), varying width (center, right), final result after overlay of multiple strokes (right).

5.4.3 *Hatching*

Here we describe an approach to achieve varying and overlapping hatching strokes using the (U, V) coordinates. At first, we subdivide the range of U into b disjunct bins.

$$b_i = \text{floor}((U + o) \cdot f) \tag{5.5}$$

$$\hat{u}_i = (U + o) \cdot f - b_i; \tag{5.6}$$

where o is an offset to U and f controls the number (i.e., frequency) of bins. Usually, f is set to a rather large number, as it represents the number of hatching strokes within each integer interval of U. The offset o can be arbitrarily chosen, as a constant or as a function of V, for example.

Thus, each point i on the surface is assigned an integer value b_i and a bin-wise mapping $\hat{u}_i$ of U to the range $[0, 1]$. We can then obtain an intensity value via

$$h_i = \cos(\hat{u}_i \cdot 2\pi)0.5 + 0.5 + w_i) \tag{5.7}$$

to draw smooth strokes at the center of each bin along the isovalues of U, where w_i controls the width of the stroke. A maximum stroke length l_i can be applied by using V:

$$t = \text{clamp}_{0,1}(\frac{v_i}{l_i})^s \tag{5.8}$$

$$h_i \leftarrow (1 - t)h_i + t \tag{5.9}$$

where s controls the falloff of the stroke intensity. An initial rendering of equally shaped strokes is shown in Figure 5.13 (left). However, to achieve a visually appealing hatching, it is important to avoid repetitive patterns of strokes. We address this by feeding b_i into a pseudo-random generator that yields a value r_i in the range $[0, 1]$. By modifying a random seed based on b_i, multiple different random distributions r_{ij} can be generated for each b_i. This can be used to modify the stroke length $l_i \leftarrow l_i \cdot a \cdot (2r_{i1} - 1)$ (see Figure 5.13, center, left) and the stroke width $w_i \leftarrow r_{i2}$ (see Figure 5.13, center, right). Finally, multiple sets of hatching strokes can be drawn with an overlap by modifying the offset o. This offset can also be calculated in

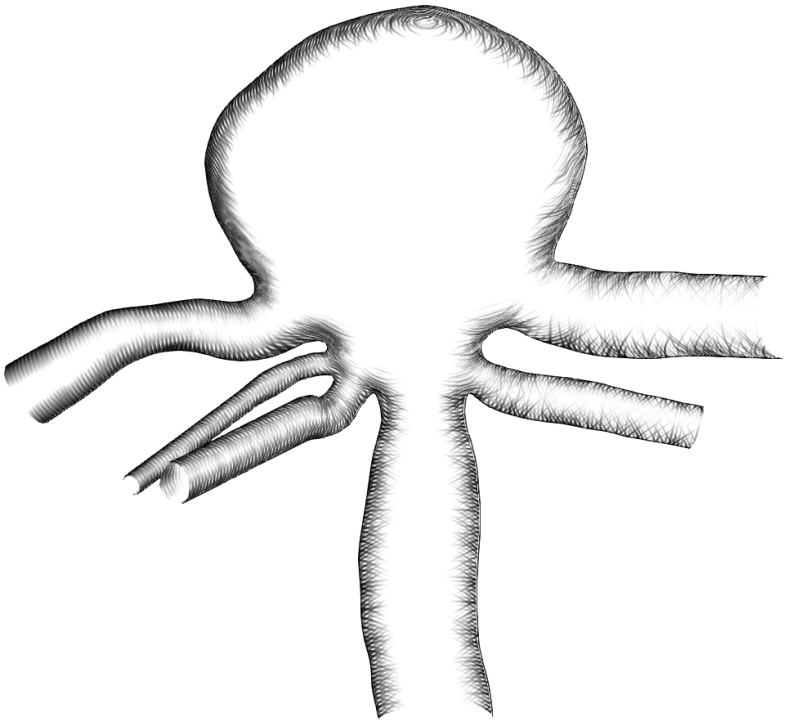


Figure 5.14: The variation of hatching strokes is shown in dependence of a scalar field (increasing from left to right).

dependence of V to achieve tilted strokes (see Figure 5.13, right). As the offset to U affects the calculation of r_{ij}, locally varying parameters for w_i and l_i are obtained. A whole vessel data set rendered with this technique is shown in Figure 5.12. The variation of strokes contrasts a reference method by Lawonn et al. [125] shown in Figure 5.24. Figure 5.14 shows how the hatching variability could be used to encode scalar information. In the illustration, parts of the surface to the left are rendered with uniform strokes, while moving towards the right, the strokes are more and more varied. In the context of clinical applications, this representation could be used to encode the distance to a reference object (e.g. a tumor or surgical instrument).

5.4.4 *Contour parameterization*

If a visualization should be used to encode multiple scalar fields the number of available information channels is often quickly exhausted. In this case one can use additional geometry to encode information. Due to the rather thin, tubular structure of vessel, the contour of the structure is a viable option to do this. The smaller the diameter of a vessel segment, the better can information from the contour represent the data of the affiliated vessel segment. For example, this has previously been done by Behrendt et al. [6], who encoded the depth

of vasculature by coloring its contour using pseudo chroma-depth. Instead of defining the contour on the surface itself (e.g. by using a Fresnel approximation), we aim to create additional geometry at locations of the rendered mesh, where the surface normal is orthogonal to the view direction. The vertices of the generated faces adopt the U values of the generating surface. Additionally, V values can be created based on the width of the contour in world space (see Figure 5.15), and therefore their scale matches the scale of V as computed in Section 5.3.1. The V-value of a point on the contour is then its distance to the generating surface. We can then display small texture patches on the contour as shown in Figure 5.16 (top). In this example, the general direction of blood flow is depicted by an arrow decal at the vessel boundary. In Figure 5.17 we use U to draw a dashed contour, which could be applied to focus-and-context applications. Alternatively, the contour can simply be colored w.r.t. the pseudo chroma-depth spectrum as in Figure 5.18, leaving the surface free for other encodings. This addresses an issue which has been tackled by Behrendt et al. [6]. They mixed the color coding of a scalar field and pseudo chroma-depth using a Fresnel term. A drawback of their method is that for small vessel segments neither the depth, nor the scalar encoding is accurately perceivable. With our method, we are able to draw contour margins of invariant size on the mesh (see Figure 5.19). Hence, for thin vessels only the contour color remains, which is still not optimal but should be preferred over a hard-to-read mixture of color scales. Additionally, we omit the shading of the object in this example, in order properly retain the color scale. Geometric features are instead depicted by the hatching strokes.

5.4.5 *Binary tree coloring*

The binary code that is associated with each segment of $\mathcal{Y}$ can be used for an automatic coloring of the vasculature, that highlights individ-

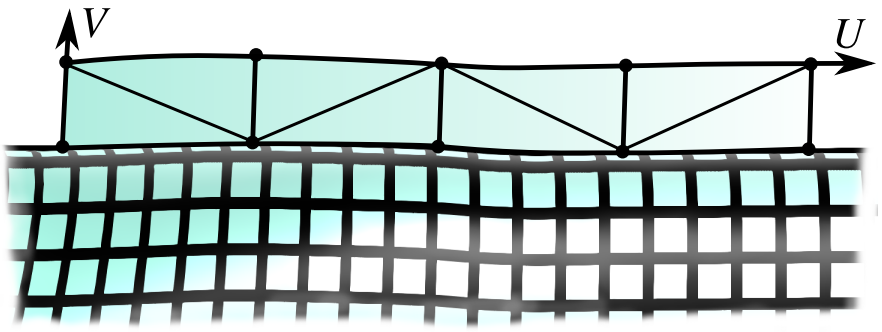


Figure 5.15: Additional contour geometry (triangles delineated at the top) is created from the data of the input mesh.

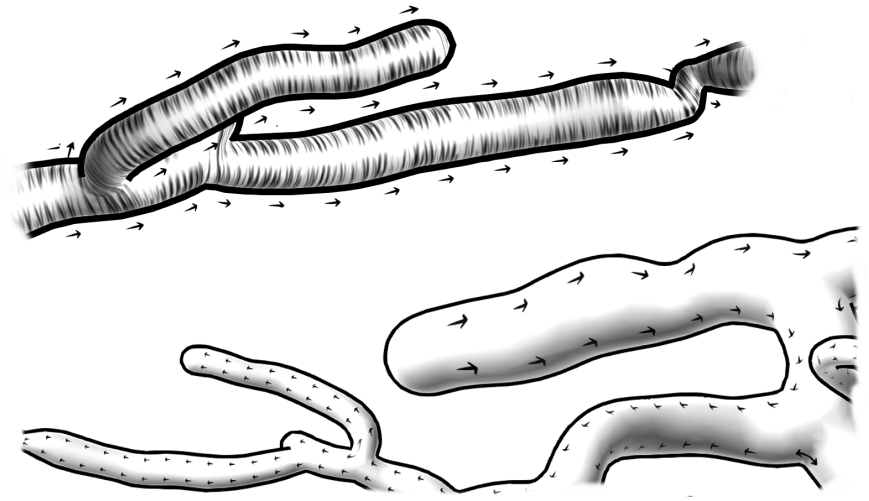


Figure 5.16: The contour of the mesh is labeled with decals that emphasize the direction of blood flow (top). The same example using the (U, V) coordinates of the initial parameterization to draw decals directly on the surface (bottom).

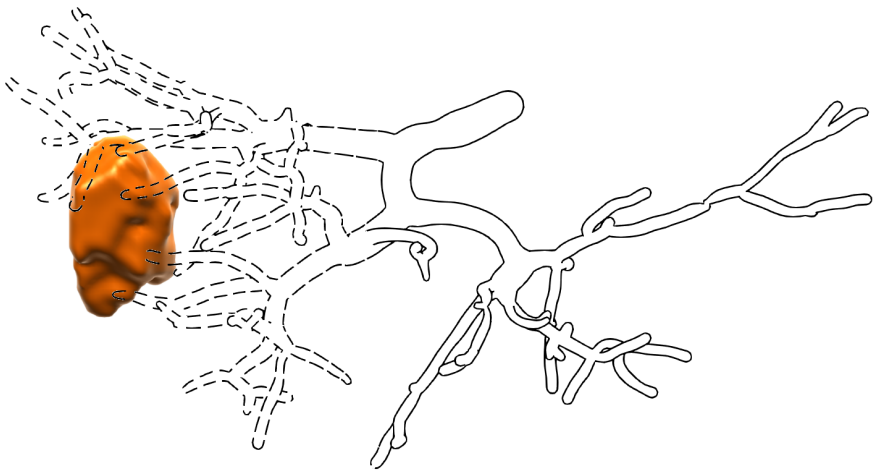


Figure 5.17: U is used to draw a dashed contour. The spacing of the dashes decreases with the distance from the tumor.

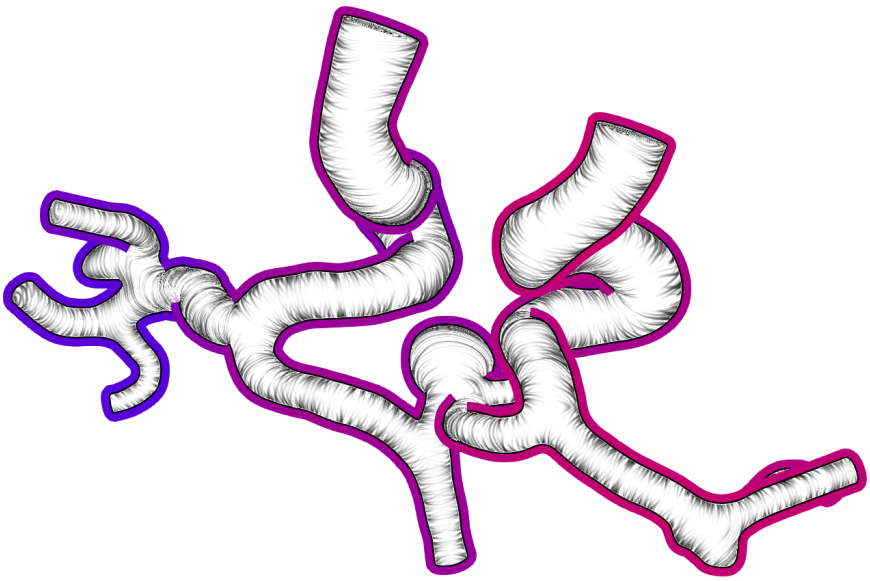


Figure 5.18: Contour used to display pseudo chroma-depth (right).

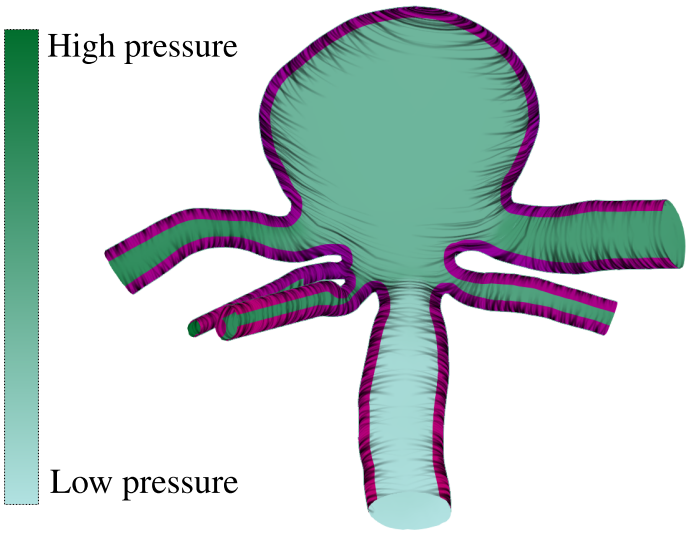


Figure 5.19: Pseudo chroma-depth applied to a contour margin. The remaining surface color codes the blood pressure on the surface.

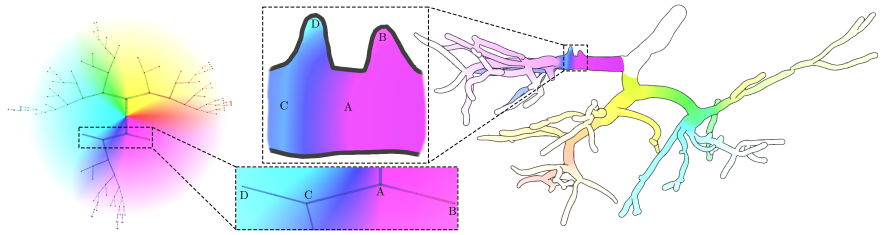


Figure 5.20: Overview plot with HSV colors (left) mapped to the 3D mesh (right). The closeup (center) underlines how the HSV colors covered by a graph segment (center, bottom) are mapped to the 3D mesh (center, top).

ual sub-trees. We compute a vector H whose elements i refer to each tree segment i:

$$H_i = \sum_{j=1}^{l(B_i)} \frac{\delta(B_i, l(B_i) - j)}{2^j}, \tag{5.10}$$

where $\delta(B_i, j)$ returns 1 if the j'th bit in B_i is set to 1 and -1 otherwise. Note that indexing into $\delta(B_i, j)$ is 0-based. The resulting H can be used as input to a color map. Further, each H_i will always be in the range $(0, 1)$ and the highest impact on the results have the higher bits, i.e., the bits associated with segments close to the tree root. Therefore, branches that occur in higher levels of $\mathcal{Y}$ can be better distinguished than branches that occur at a deeper level. For instance, the binary code 100 would result in a sum $0.5 - 0.25 - 0.125 = 0.125$. In our example (see Figure 5.20, right), we employ H as the hue value in HSV color space. We additionally calculate the saturation S based on the layer n_i of each segment, except for the root segment:

$$S_i = \left(1 - \left(\frac{n_i - 2}{d - 2}\right)^p\right)(s_{max} - s_{min}) + s_{min}, \text{with } i > 1 \tag{5.11}$$

where d is the maximum layer of $\mathcal{Y}$, i.e., the maximum length of all segment binary codes, and $p = 0.8$ controls how quick S falls off between the maximum s_{max} and minimum s_{min} saturation values. With Eq. 5.11, the leaf segments of $\mathcal{Y}$ will have a saturation of s_{min} while the two child segments of the root segment are assigned a saturation of s_{max}. A graph plot can then be generated by deriving 2D coordinates for each node i from H_i and S_s of the associated segment i. We plot this graph on an HSV color disc with a constant value set to 1, as in Figure 5.20 (left). The saturation is inverted, i.e., the saturation decreases from the center to the edge. This graph representation can be used for a color-guided integration with the 3D visualization (Figure 5.20, right). As we did not define H and S for the root segment, we place it in the center of the HSV disc and draw it in plain white in the 3D representation. The center of Figure 5.20 shows how color values are interpolated across each segment in 3D, according to the

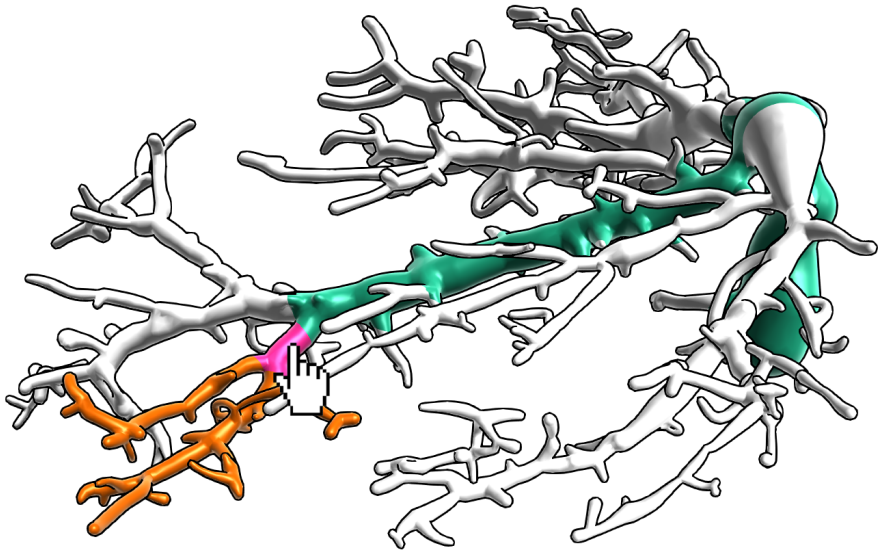


Figure 5.21: Selected segment (magenta), ancestor segments (green) and descendant segments (orange).

HSV graph representation. The inverted saturation in the overview plot allows to draw the graph in a more intuitive manner, with the more meaningful branches located at the center with a more perceptive, saturated color. The 2D plot is also an indicator for the balance of the branch topology. The magenta area in Figure 5.20 (left) is occupied only by a single end-point, revealing that an early branch with no further ramifications is present in the data set.

Another application for the binary code is to determine the lineage (i.e., ancestors and descendants) of a selected segment, as depicted in Figure 5.21. This kind of visualization may come in handy in the context of liver resections, where the segments affected by a cut are highlighted.

5.5 IMPLEMENTATION

In this section we describe details of our implementation.

OFFLINE CALCULATION OF $\mathcal{U}$ Since $\mathcal{U}$ is defined in object-space, it only has to be calculated once. Therefore we implemented the calculation of $\mathcal{U}$ in Matlab (the code is provided with the original publication [133]). In order to solve Eq. 5.2, we calculate the minimal curvature direction $\mathcal{C}$ and the gradient of the geodesic distances $\nabla \mathcal{G}_\Delta$. We then use a Matlab solver to obtain the results.

ONLINE CALCULATION OF $\mathcal{V}$ The evaluation of $\mathcal{V}$ is done anew for every frame. At first the input mesh $\mathcal{M}$ is rendered in a preparation pass. Here we generate a mask that distinguishes between back-

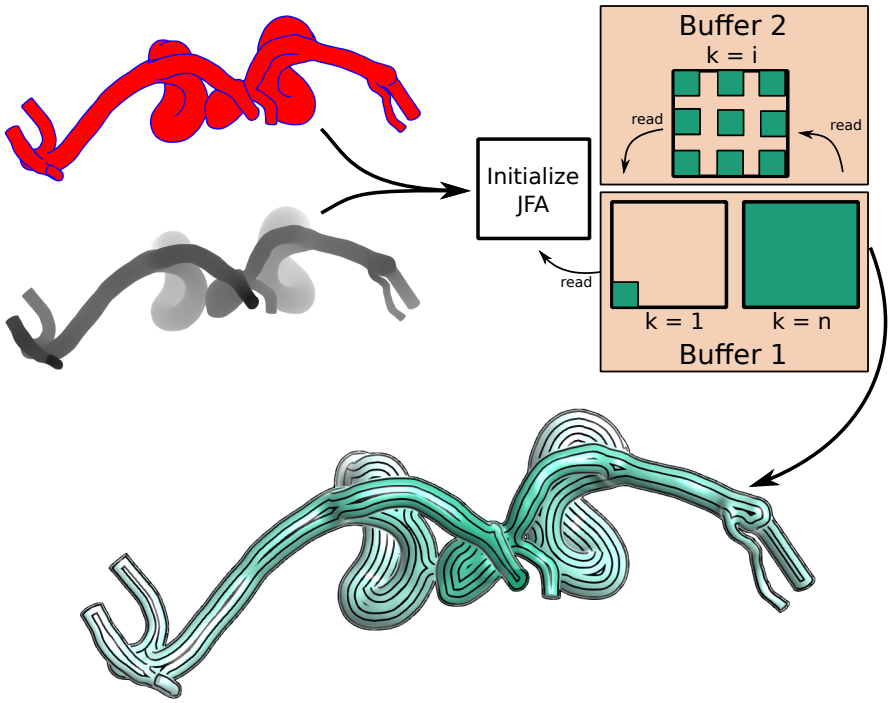


Figure 5.22: Process of the JFA execution. A mask image (left, top) determines whether a pixel belongs to background (white), contour (blue) or surface (red). A depth image (left, center) is used for our depth aware modification of the JFA. The result of each JFA iteration k is read in the next iteration by swapping two buffers. After the final iteration, the last write-buffer is used to assemble a final image based on V.

ground, contour and surface pixels (see Figure 5.22, left, top) and a depth texture (see Figure 5.22, left, center) that is used to implement the depth awareness of the the JFA as depicted earlier in Figure 5.6. From then on we use two buffers that are read from an written to in each JFA iteration k. After each iteration the read and write buffers are swapped. The JFA finds for each surface pixel the distance to the closest contour pixel. We only edited the original algorithm in a way, such that a surface pixel skips contour pixels that have a lower depth value than the surface pixel (see the shader provided with the original publication [133]). For more details on the JFA we refer to the original paper by Rong et al. [187]. After the JFA has terminated, we can read from the recent write buffer and use V as a texture coordinate.

ROOT-POINT DETECTION As an optional pre-processing step, we provide a method to find the root end-point of a vessel. The detection is done in Matlab, using our own implementation of the thinning algorithm proposed by Au et al. [3] and an iterative search over each vertex as described in Section 5.3.1 (the code is provided in the supplementary material).

Table 5.1: Execution times for the JFA in milliseconds and frames per second (FPS) of our prototype application.

Resolution	JFA (ms)	FPS
1024×768	3.5	110 - 141
1920×1080	9	52 - 65

CONTOUR TREE AND BINARY CODE COMPUTATION The CT computation and binary code generation are implemented in Matlab (see supplementary materials). To store the binary code as, for example, an *Int8* data type the binary code is padded with zeros to the left in order to fill the data type. For example, the code 10011 is stored as 00010011, i.e., a value of 19. Ambiguities cannot be introduced by the leading zeros, because the bit representing the root segment is always set to 1. To load the binary code into our prototype application, we stored each code B_i in three Int32 variables along with the code length n_i. Therefore our prototype can handle binary codes of length up to 96 bits. The bit-shift then has to be done across the three 32-bit values. For the implementation of the bit-shift we refer to the shader sample provided in the supplementary materials.

PERFORMANCE Our performance tests were executed on a desktop computer environment with a 4.00 GHz i7-6400 processor, a GTX-1070 GPU and 16GB RAM. As the offline computation of U is less crucial for the overall performance of our approach, we only provide some key data: The largest mesh that we tested, consisting of 200k triangles, took 3.5 seconds to execute. The smallest mesh with 17.5k triangles took 0.25 seconds. Considering that parts of that algorithm could be executed on the GPU, we argue that this is sufficiently fast for an interactive adjustment of U. The computation of V was found to be independent of the mesh size. As shown in Table 5.1 our prototype implementation is able to obtain V in full HD resolution in real-time. The FPS are given in ranges. Here, the lower bound of the range is the performance when we zoom very close to the mesh. This means that more pixels have to be treated in the assembly of the final images and the frame rate decreases. The upper bound is what we measured for a lower zoom factor (as found in Figures 5.12,5.14). The JFA execution time has not changed in these cases, since always all pixels are processed. The mesh shown in Figure 5.3 contains 30k vertices and the computation of the end-points including the root-point detection took $\sim$ 10 seconds. This is twice as fast as the method by Lichtenberg et al. [132]. We observed that our new approach detects end-points more faithfully while the results further depend on the choice of the detection sphere radius determined by k (recall Section 5.3, paragraph about root-point detection). A larger radius yields less end-points, detecting only more protruding branches

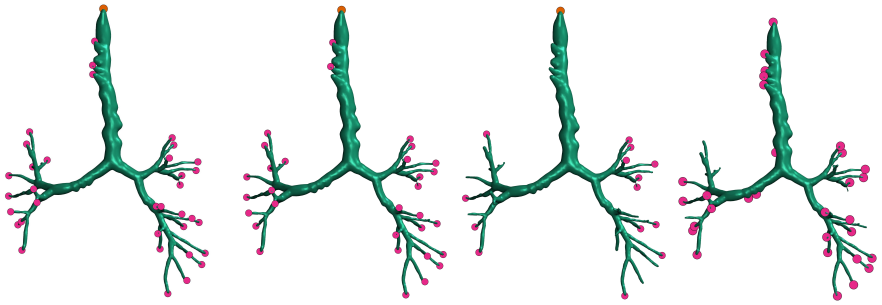


Figure 5.23: Detected vessel end-points from left to right: Changing factor $k = 3, 9, 25$ to the search sphere radius for our method and the reference method by Lichtenberg et al. [132] on the far right.

as compared in Figure 5.23. The proportion of the individual steps are 40% for the thinning, 45% for the sphere tests and 10% for the curvature test. For a mesh containing 100k vertices our algorithm took ~ 110 seconds, while the reference approach took 154 seconds. Especially the iterative execution of vertices for the sphere test (see Figure 5.3, bottom) could be improved by a parallel implementation.

CONTOUR GENERATION The contours that we used for the contour parameterization in Section 5.4.4 (see Figure 5.15) and for the contour mask (see Figure 5.22, left, top) are generated in the geometry shader. We compute $s_i = \langle \mathbf{n}_i, \mathbf{v}_{cam} \rangle$, where $\mathbf{n}_i$ is the normal of vertex i in a triangle and $\mathbf{v}_{cam}$ is the current view direction. Then we search for zero crossings of s_{ij} between two vertices i and j. If we find zero crossings on two edges of a triangle, we generate a quad with two vertices at the crossing points and two vertices that are extruded by a distance d w.r.t. the interpolated normal directions at the crossings. In the same way as the normal direction, we can interpolate U from the generating triangle. The V coordinate is then determined by setting $V = 0$ for the vertices at the crossing points and $V = d$ for the extruded vertices. This yields U coordinates that fit to the generating mesh and the V coordinates are also scaled according to the result of the JFA.

5.6 DISCUSSION

We presented an algorithm to compute texture coordinates for tubular, tree-like structures. The input is a user-defined source vertex or a vessel root-point detected by our proposed method. While techniques like that by Kälberer [93] exist, our parameterization is simpler to implement and computationally less costly. We were also able to show that the U coordinate can be used to find branch-points of the vessel. Another advantage over existing parameterization methods is that the background pixels are also addressed, allowing depth cues as in

Figure 5.24: The ConFis method by Lawonn et al. [125]. Compare with Figures 5.12,5.18,5.16,5.14.

Figure 5.12. This is due to the screen-space computation of V, but at the cost of recomputing V for each frame. Nonetheless, our algorithm performs in real-time on modern consumer hardware. The properties of V can further be used for a camera oriented display of decals on a mesh. On cylindrical parts of a structure (i.e., away from branch- and end-points) we find that the gradients of U and V are mostly orthogonal and therefore suitable for the display of patterns and textures (see Figure 5.28). The CT algorithm yields a binary tree representation of the input structure. This restriction can be seen as a limitation, because meshes with a genus other than zero can not be represented faithfully. Nonetheless, meshes with handles are robustly processed, with the handles simply being ignored by the employed algorithm [26].

LIMITATIONS The parameterization approach presented in this paper is simpler than existing approaches and yields good results for tubular structures. It is tailored to vessel trees in the sense that pixels that represent points on a vessel segment are relatively close to the contour pixels associated with the same segment. Therefore, the V approximation works well. This also accounts for the modification that we did to the JFA algorithm, where pixels may only refer to contour pixels that have a higher depth value. The shading that is usually applied to a mesh is also a factor. The locations where V is locally maximal (i.e., the centerline of the projected contour as in Figure 5.5) are often locations of specular lights. Hence, these peak locations of V, where the gradient of V is undefined, can be hidden by the shading. Regarding these susceptibilities the parameterization quality decreases as the mesh morphology is less tubular. We show a stippling pattern applied to an aneurysm in Figure 5.25. It can be observed that part of the parameterization is highly distorted. This is due to an improper computation of V, as the associated pixels refer

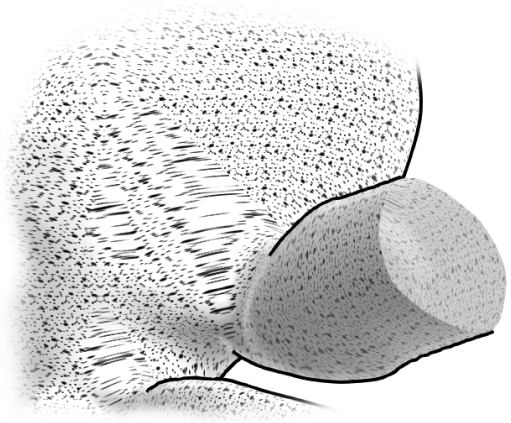


Figure 5.25: Close-up of an aneurysm data set. Our approach has issues with the spherical shape, introducing high distortion in V.

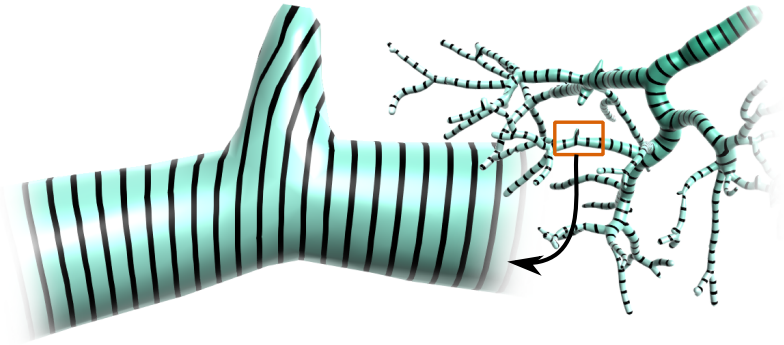


Figure 5.26: A tiny branch on the vessel structure is omitted by U.

to wrong parts of the contour. This behavior may change with a small rotation of the object, when other parts of the contour become visible or hidden. Therefore frame coherence is also an issue.

We described in Section 5.3.1 that we smooth the vector-field Z in order to remove noise and high frequency features. While this is a crucial step to obtain a proper and smooth U, it can also lead to smoothing out geometric features of the mesh. Figure 5.26 shows a close-up of a small branch segment, which is predominantly ignored in U.

Our root-point detection algorithm relies on two main assumptions: First, we assume that vessel end-points are cap-shaped, and second, we assume that the diameter of the root is larger than any other diameter. This means that we are likely to detect the head of an aneurysm, if it is larger than the healthy parts of the vessel.

The 2D graph representation of the vasculature as in Figure 5.20 (left) is currently generated by a straight-forward computation of hue and saturation values based on the segments' binary codes. While this underlines the balance of Y, only a subset of the color map is used (see Figure 5.27, left). Therefore, a further optimization of the

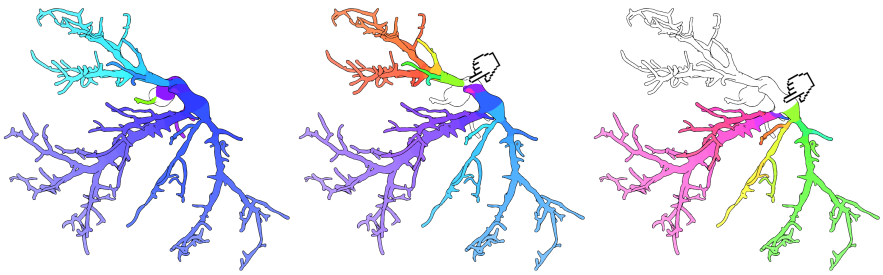


Figure 5.27: Binary tree coloring applied to the whole tree (left) and applied to user-defined sub-trees (center, right).

color assignment may be feasible. Another option is to only color user-selected sub-trees as in Figure 5.27 (center, right).

OUTLOOK As discussed, the proposed method is tailored to be applied to tubular meshes. However, we hope that the examples shown in this paper are able to support or inspire future work in this domain. In our opinion, one can build up on this work in two different ways. The first one would be to further improve the methodical details and implementation. Here, the frame coherence should be addressed. We are also optimistic that the $\mathcal{U}$ computation can be done real-time as well, which might open further interactive application scenarios. As a screen-space approach is independent of the mesh size, this may have the potential to overcome the size limit of the method by Lichtenberg et al. [138]. The second direction would be to use the current method and integrate it into more sophisticated medical tasks. Using different visual channels and additional geometry (i.e., the contour), multiple scalar fields can be independently visualized with our approach. However, the effectiveness of different combinations in a real-world task needs to be evaluated. In conclusion we are confident that particular aspects of the proposed technique are feasible in various scenarios and that it therefore creates a basis for further research.

ACKNOWLEDGEMENTS Funding: This work was supported by the DFG: LA 3855/1-1.

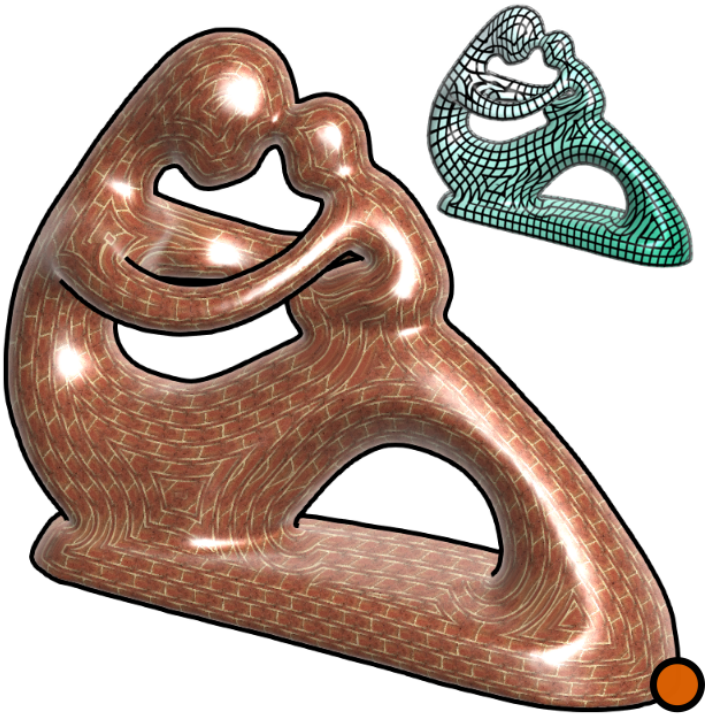


Figure 5.28: Our method applied to the fertility model, using a brick texture. A root point (orange) was also found by our method in a reasonable location.

6

DISTANCE FIELD VISUALIZATION AND 2D ABSTRACTION OF ANATOMIC TREE STRUCTURES WITH ON-THE-FLY PARAMETERIZATION

ABSTRACT In this paper, we make several contributions to the visualization of vascular structures. We show an efficient algorithm to create a distance field volume, based on a skeletal tree. Sphere-tracing this volume allows to visualize the vasculature in a flexible way, without the need to recompute the volume. Illustrative techniques, that have been frequently applied to vascular visualizations often require texture coordinates. Therefore, we propose an image-based, hierarchical optimization process that allows to derive periodic texture coordinates in a frame-coherent way and suits the implicit representation of the vascular structures. In addition to a 3D surface visualization, we propose a simple algorithm that applies a 2D parameterization to the skeletal tree nodes. This parameterization can be used to color-code the vasculature or to plot a 2D overview-graph. Depending on the weights applied to the nodes during parameterization, the overview-graph can highlight different aspects, e.g., the branching topology of the skeleton. We transfer measurements, done in 3D space, to the 2D plot in order to minimize visual clutter and self occlusions in the 3D representation. A visual link between the 3D and 2D views is established via color codes and texture patterns. The utility of our methods is shown in several prototypical application scenarios.

6.1 INTRODUCTION

Visualization methods for the investigation of vasculature have been actively researched in the past and are still a current topic. Challenges in this field range from surface extraction and generation over enhanced visualization techniques to interactive concepts for the application during a surgery. While the extraction of vascular structures dates back to the work by Gerig et al. [52], more modern and more controllable methods have been developed in the meanwhile [161]. A common goal of advanced visualization techniques is to communicate complex information about the vasculature in an understandable and comprehensible way. For this, basic concepts of human perception are considered [175]. Illustrative techniques [128] are also often applied in this context. Those are likely to require a parameterization of the surface, in order to control placement of hatching strokes, or texture patterns. For example, field-guided periodic parameterizations [176] can be employed to place stippling dots along feature lines.

Such a structural alignment is an improvement over unconstrained placement of dots, because it helps to accentuate geometric properties. Similar techniques have been used in stylistic applications [206]. The reduction of spatial dimension can also contribute to an improved tangibility of the presented data. Overlaps and visual clutter can be better controlled and avoided in a 2D representation, than in a 3D representation [109].

We position this paper in the field of surface generation and advanced vascular visualization techniques. Modern GPU architectures allow to handle larger data in a more complex way to generate images. Whereas iso-surfaces can be extracted from volume data, a specialized, volume-based representation for an implicit surface is a signed distance field (SDF). While the SDF helps to rapidly trace the implicit surface, the generation of the SDF itself can also be done at highly interactive rates [107]. The properties of the SDF allow simplified implementations of otherwise more complex visual effects. Therefore, our first contribution is to generate a SDF from skeletal vascular data and to render an exact surface from it. Second, we propose a simple algorithm to transform the 3D skeleton to an abstract 2D graph representation. The graph representation will highlight the branch-topology of the vasculature and allows to view related properties of the whole model in one gaze. To apply illustrative effects to the generated 3D surface, we require a surface parameterization. As the implicit generation of the displayed surface does not allow an off-line pre-computation of the parameterization, we introduce an image-based, hierarchical optimization process that allows to derive periodic texture coordinates. This screen space parameterization (SSP) method works on the rendered surface and executes just-in-time. A byproduct of the screen-space approach is the parameterization of the unoccupied space next to the surface, which may also be utilized for visualization purposes. Finally, we describe the applicability and usefulness of our main contributions through selected applications that benefit from the combined 2D and 3D view.

6.2 RELATED WORK

The immediate reconstruction of vascular structures from MRI or CT data, using direct volume rendering or maximum intensity projection, is not free of artifacts. Reconstruction methods that can assure, e.g. continuity of vessels, are a more sophisticated way to go. These methods can be model-free or model-based [110], with the model-based methods dominating the literature. A reason for this may be that most hemodynamics simulation applications require an explicit surface representation [160]. For example, the method by Oeltze *et al.* [161] uses a directed graph as input. Each node is assigned with a radius, determining the vessel thickness and each edge represents

a segment of the vasculature. Using convolution surfaces [14], they reconstruct a smooth implicit surface, which can then be triangulated for visualization. This allows for a faithful representation of the input radii, but does not allow arbitrary cross-sections. Schumann *et al.* [201] used a point-based implicit representation to achieve arbitrary cross-sections. The input is taken directly from binary masked volume data. The work by Kretschmer *et al.* [110] transforms signed distance fields to a potential function that describes the implicit surface. A further overview of vascular surface reconstruction literature can be found in the work by Saalfeld *et al.* [194], who use meta-balls to define and render an implicit surface on-the-fly. While most techniques that aim for an explicit reconstruction use an implicit representation as an in-between-step, it is not obvious whether the implicit representations are suitable for direct rendering. We therefore contrast the current state-of-the-art by employing an SDF to directly render the implicit surface, using an exact sphere-tracing approach. With the SDFs, we gain more flexibility in the visualization of the vasculature, as will be shown in the application examples.

SDFs are a common geometry representation in many areas ranging from collision detection using the distance information [50], surface reconstruction [164] to general rendering applications. 2D fields can be used to encode low resolution text or glyphs. These can be reconstructed without magnification artifacts encountered with bitmap data as shown by Green [56]. Information about surrounding geometry can be obtained from an SDF and be used for various effects, that are typically hard to do with traditional rendering techniques, such as soft shadows or volumetric ambient occlusion [230]. An overview over three-dimensional distance fields can be found in Jones et al. [89].

To render structured patterns on the implicit surface, a parameterization is required. The range of parameterization applications is very wide and well covered in the summary by Sheffer *et al.* [202]. As mentioned earlier, parameterizations can be aligned to an input guiding field. Vaxman *et al.* [227] provide a thorough survey of the synthesis of such guiding fields. The alignment with a guiding field can allow artistic freedom or be used to highlight structural features. An additional common goal is to reduce the distortion that is introduced by the mapping from a 2D to 3D domain. For this task, energy functions are defined and attempted to be minimized. This is a difficult task, depending on the target topology and morphology. Surfaces that cannot be trivially mapped to a plane are especially problematic. The approach by Praun *et al.* [171] attempts to circumvent this problem by locally mapping parts of a surface onto a 2D plane. The discontinuities across the mappings are resolved by blending during the texture application. The property of locality can also be introduced by using periodic mappings. In the remeshing domain this was done

by Ray *et al.* [176]. The feasibility of employing a periodic parameterization, that is aligned to a guiding vector field, for artistic or visualization purposes has been exemplified in the works by Knöppel *et al.*[103] and Son *et al.* [206]. The former works on meshes and the latter on images as input. While the execution times of both are interactive, they did not aim for real-time capabilities. Real-time execution has recently been tackled by Lichtenberg *et al.* [138] (triangles), where they moved an approach, similar to the work by Jakob *et al.* [86] (triangles and point-clouds), to the GPU. The idea is to solve a global optimization problem by local approximations. By lifting the data into a multi-resolution hierarchy structure, locality can be overcome. Such an approach is highly suitable for parallel execution of the local approximations in one hierarchy level. As our implicitly rendered surface is given in image space, we adapt this idea and apply it to the image space. The hierarchy structure will then be an image pyramid. Multi-resolution hierarchies expose the ability to propagate global features (coarse hierarchy level) into local regions (fine hierarchy level). This is why the Pull-Push Algorithm (PPA) by Gortler *et al.* [55] is an important tool for our work. We use this algorithm to traverse the hierarchy levels in order to propagate information across the screen space. The PPA is commonly used for hole-filling purposes (see [147] for an example). Thus, we can use the PPA to substitute the empty space around an object. This is also of interest in the visualization community as the work by Kreiser *et al.* [108] shows. Alternative methods to apply structured patterns were proposed: Kim *et al.* [99] deform an input hatching texture to align with a given guiding field, which is computed on the fly. This allows them to display animated meshes in an illustrative manner. At the downside, the screen space projection introduces shower-door effects. Breslav *et al.* [23] were able to reduce these artifacts to a certain grade by 2D similarity transforms. They incorporate information of the rendered 3D model to adjust their results approach accordingly. Our method contrasts the above by providing texture coordinates that are closely related to the input 3D structure (i.e., no shower-door effect occurs) and are globally periodic (i.e., no seams occur). However, we cannot utilize the whole morphology of the input structure, as we only work on the visible projection. This again, can also be of advantage because the complexity of the visible surface may be lower than that of the whole surface.

A common goal in the advanced visualization of vasculature is to encode multivariate data on the surface or to enhance spatial perception of the geometry. Recent attempts to improve spatial perception through auxiliary tools [175] have been made by Lawonn *et al.* [122, 124], Lichtenberg *et al.* [132] or Kreiser *et al.* [108]. Further, flattening techniques [109] are a prominent way to reduce the complexity of a visualization and to overcome self-occlusion. We contribute to this

area by proposing a combined 2D and 3D view of the vasculature, that are visually linked by color-codes and texture patterns.

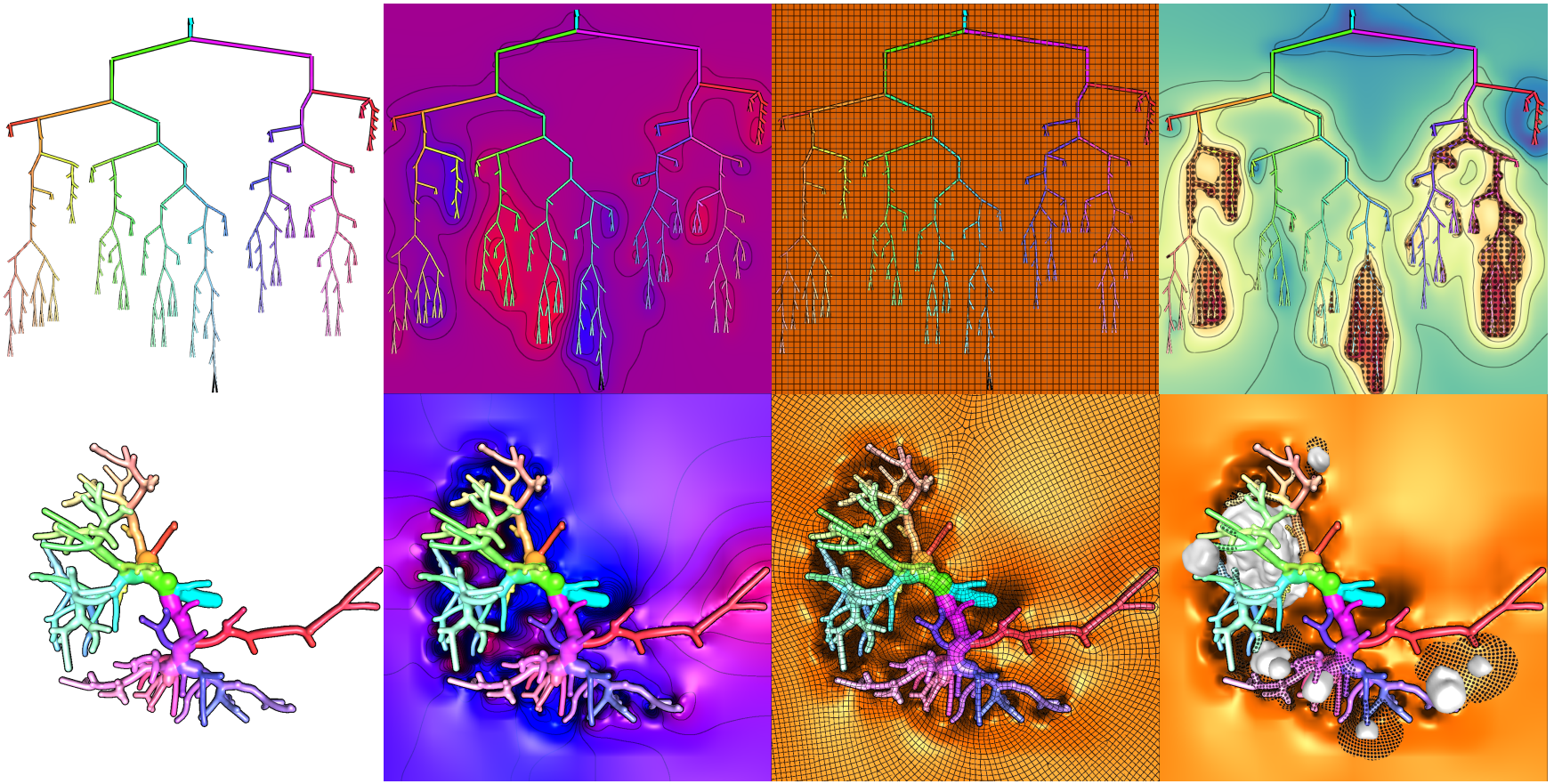
6.3 METHOD

The vascular trees to be visualized are given as a directed graph $\mathcal{T}$ with nodes $\mathcal{N}$ and edges $\mathcal{E}$. A directed edge $e(i,j) \in \mathcal{E}$ connects two nodes $(i,j) \in \mathcal{N}$ and a node i is associated with a point $\mathbf{p}_i \in \mathbb{R}^3$ and a radius $r_i \in \mathbb{R}$. Further, we assume that the directed edges point away from the root and each node has at most two children and one parent. We assume this binary branching restriction, because branches of higher degree can be broken down to binary branches by adding nodes with edges of zero length. The root is restricted to one child and no parent, so that there exists one *root edge* that then further branches into the successive sub-trees. We call nodes that have one child and one parent *regular node*, nodes that have no child and one parent *leaf node* and nodes that have two children and one parent *branch node*. By removing all regular nodes, only the root, leaf- and branch nodes remain in a reduced tree $\mathcal{T}_r$ which represents the branching structure of the vascular tree.

This section can be wrapped up as follows: First, the input graphs $\mathcal{T}_r$ and $\mathcal{T}$ are used to generate a 2D (Section 6.3.1) and 3D (Section 6.3.2) depiction of the vasculature. Then, a recap of the PPA (see Section 6.3.3) is given, which is then used for a background reconstruction (see Section 6.3.4). With the background reconstruction, we assign information to all pixels in the 2D and 3D views, that are not occupied by the vessel structure. Kreiser *et al.* [108] already did this to obtain *Void Space Surfaces* (VSS). In our pipeline, the background reconstruction ensures that the successive SSP does not need to handle void pixels. Before the SSP is applied to the 3D view (see Section 6.3.6), we generate a guiding field (see Section 6.3.5). To ensure a frame coherent parameterization, successive frames are optimized based on the results of the previous frame (see Section 6.3.7). Figure 6.1 depicts the main sections of the method.

6.3.1 *2D Graph Layout*

This section describes the transfer of the 3D vessel tree to a 2D graph representation. The goal is to layout the graph in a way that highlights the branching topology of the 3D skeletal tree. Thus, we use the reduced tree $\mathcal{T}_r$ and arrange the nodes in a way that allows to easily identify different sub-trees. Previous work [135] has already made an attempt in this direction, but fails to capture very unbalanced trees. We propose a new, simple algorithm that takes node weights into account and assigns to each node a parameter pair $(h,s) \in [0,1]^2$, to achieve a more intuitive or even task oriented graph layout. Here, h is



(a) 2D/3D Structure generation (b) Background reconstruction (c) Parameterization (d) Data mapping

Figure 6.1: Overview of our method pipeline. (a) The input graph $\mathcal{T}$ is transferred to a 2D and 3D representation (Sections 6.3.1 and 6.3.2). The 2D layout parameters are used as hue and saturation to obtain segment colors. (b) The background is reconstructed (Sections 6.3.3 and 6.3.4). Here, a pseudo-chroma depth mapping is applied as an example, imitating VSS [108]. (c) The SSP is applied to the 3D view (Section 6.3.6), the 2D view is parameterized with a uniform grid. (d) The results are utilized to apply a color- and pattern-mapping. In this example, the minimal distance of vessel segments to the tumor surfaces is encoded by color, while the pattern overlay highlights regions where the distance is smaller than 15 mm.

the horizontal and s the vertical layout parameter. By default, we set a node's weight w_i to the number of nodes in the sub-tree represented by this node. We initialize the (h, s) parameter for the root and its child node with $(0.5, 1)$. By putting them into the same location, the first edge has zero length and will be hidden in consecutive visualizations. If n_1 is the root's child node, then we start to recursively parameterize the successive nodes by calling $\mathtt{ChildLayout}(n_1, 0, 1)$, which is depicted in Algorithm 2. The call with bounds 0 and 1 means that child nodes may occupy a range between 0 and 1 for the h-parameter. If n_i is not a leaf node, i.e., it has two child nodes, we obtain the child nodes and their weights (lines 3-4). The weights are used to determine the vertical s-parameter in dependence of a user-parameter $p_s = 0.4$ (lines 5-6). Then, the available h-space is divided at a value r_t (line 7). The basic idea here is to assign a larger fraction of the available h-space to the child node with the higher weight. For very unbalanced weights, it can be beneficial to allow an overlap of the children's h-space. We account for this by computing a weight w_r, which reflects the ratio of the node weights (line 8). The impact of the weight difference is controlled by a parameter $p_h = \frac{4}{N_r}$, where N_r is the number of nodes in $\mathcal{T}_r$. Offsets m to the children's ranges are computed (lines 9-10) and applied in the call to the next recursive iteration of our procedure (lines 13-14). The h-parameter is obtained as the center of the assigned range (lines 11-12). Note, that in $\mathcal{T}_r$ a node is either a leaf node or a node with children, except for the root node. In a final step, the (h, s) parameter is scaled to the range $[0, 1]$.

Algorithm 2 2D Node Layout

1: **procedure** CHILDLAYOUT(n_i, r_1, r_2)
2: **if** NumberOfChildren(n_i) = 2 **then**
3: $c_1, c_2 = \mathtt{getChildNodes}(n_i)$
4: $w_1 = \mathtt{weight}(c_1), w_2 = \mathtt{weight}(c_2)$
5: $s(c_1) = s(n_i) - (w_1/\mathtt{max}(w_1, w_2))^{p_s}$
6: $s(c_2) = s(n_i) - (w_2/\mathtt{max}(w_1, w_2))^{p_s}$
7: $r_t = (w_2 r_1 + w_1 r_2)/(w_1 + w_2)$
8: $w_r = \mathtt{min}(\frac{w_1}{w_2}, \frac{w_2}{w_1})^{(p_h \cdot \mathtt{max}(\frac{w_1}{w_2}, \frac{w_2}{w_1}))}$
9: $m_1 = (w_r - 1)(\frac{1}{2} r_1 + \frac{1}{2} r_t - h(n_i))$
10: $m_2 = (w_r - 1)(\frac{1}{2} r_t + \frac{1}{2} r_2 - h(n_i))$
11: $h(c_1) = \mathtt{mean}(r_1 + m_1, r_t + m_1)$
12: $h(c_2) = \mathtt{mean}(r_t + m_2, r_2 + m_2)$
13: $\mathtt{ChildLayout}(c_1, r_1 + m_1, r_t + m_1)$
14: $\mathtt{ChildLayout}(c_2, r_t + m_2, r_2 + m_2)$

The layout parameters can be used to draw the vessel graph. Using (h, s) as is, a linear layout (see Figure 6.1 (top) is achieved. By mapping $h \in [0, 1]$ to $[0, 2\pi]$ and using $1 - s$ as a radius, we can as well plot the graph in a radial layout, with the root at the center as in Fig-

ure 6.11 (bottom). The (h, s) parameter can also be directly used to as hue and saturation values in HSV color space, as done throughout this paper. As the h-space is subdivided by individual sub-trees of $\mathcal{T}_r$ the coloring is able to highlight the respective sub-trees.

6.3.2 *Signed Distance Field Generation*

We use a specialized algorithm to prepare a fast sampled signed distance field for skeletal data. This algorithm requires special care, as the geometry that the skeleton represents is not just the lines themselves, but a surface of revolution around them, which is not given explicitly. The final geometry of the full skeleton is given as the union of the geometries for each segment, specified by the edges $e(i, j) \in \mathcal{E}$ with their respective radii and endpoints.

THE SIGNED DISTANCE FIELD FOR SKELETAL SEGMENTS is derived by geometric considerations, similar to Barbier *et al.* [5]. In contrast to them we use a full SDF instead of using zero for all points inside of the object. The base-geometry is a sphere-cone, which comprises of two spheres at the endpoints of a line segment connected by a tube aligned with the outer tangents. The shape can be described as a surface of revolution around the line segment so we can reduce the problem to two dimensions (see Fig. 6.2). We fix the origin at $\mathbf{p}_i$ and orient the x axis to coincide with the direction $\mathbf{p}_j - \mathbf{p}_i$. We define the distance between both points $s = |\mathbf{p}_j - \mathbf{p}_i|$ and the relative radius $r_d = r_i - r_j$. We fix a second coordinate system $(\mathbf{i}, \mathbf{j})$ located around the point where the outer tangents of both circles coincide and intersect the first circle. Its second axis $\mathbf{j}$ is parallel to the vector $\mathbf{c} - \mathbf{p}_i$, which by definition is perpendicular to the outer tangent line, thus determining the first axis $\mathbf{i}$. With some geometric considerations, the center point $\mathbf{c}$ is found by computing the offsets from $\mathbf{p}_i$ in the x and y directions along with the length between $\mathbf{c}$ and the tangent intersection with the second circle $l = \sqrt{s^2 - r_d^2}$.

$$\mathbf{c} = \begin{pmatrix} d_i \\ h_i \end{pmatrix} = \frac{1}{s} \begin{pmatrix} r_d r_i \\ r_i l \end{pmatrix} \tag{6.1}$$

This formulation is valid regardless of whether r_i or r_j is larger. If $|r_d| > s$, one circle encloses the other. In that case the above formula does not work and the distance is the distance to the larger circle. The orthonormal coordinate system around $\mathbf{c}$ is then computed as

$$\mathbf{i} = \frac{1}{s} \begin{pmatrix} l \\ -d_i \end{pmatrix} \tag{6.2}$$

$$\mathbf{j} = \frac{1}{s} \begin{pmatrix} d_i \\ l \end{pmatrix} \tag{6.3}$$

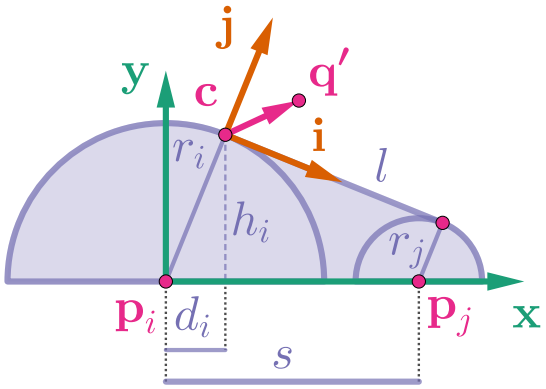


Figure 6.2: The geometry for each line segment, consisting of two points and spheres around them, which are connected by their outer tangents. The three-dimensional shape is a surface of revolution and can thus be reduced to a two-dimensional problem.

A given query point $\mathbf{q}$ in the first coordinate system can then be transformed as

$$\mathbf{q}' = \begin{pmatrix} \mathbf{i}^T \\ \mathbf{j}^T \end{pmatrix} (\mathbf{q} - \mathbf{c}) \tag{6.4}$$

The signed distance is then determined as the signed distance to one of the circles or the tubular segment:

$$s(\mathbf{q}) = \begin{cases} |\mathbf{q}| - r_i & \text{, if } \mathbf{q}'_x < 0 \\ |\mathbf{q} - \mathbf{p}_j| - r_j & \text{, if } \mathbf{q}'_x > l \\ \mathbf{q}'_y & \text{, else} \end{cases} \tag{6.5}$$

The distance field of the surface of revolution in three dimensions is obtained by projecting a query point $\mathbf{p}$ into the first two-dimensional space and then evaluating the field there. The two-dimensional coordinates are given as the projection of a query point onto the line segment and its distance to the line through $\mathbf{p}_i$ and $\mathbf{p}_j$.

$$\mathbf{d} = \mathbf{p}_j - \mathbf{p}_i \tag{6.6}$$

$$\mathbf{q} = \begin{pmatrix} \frac{|\mathbf{p}_i|^2 + \mathbf{d}\cdot\mathbf{p} - \mathbf{p}_i\cdot\mathbf{p}_j}{|\mathbf{d}|} \\ \frac{|(\mathbf{p}-\mathbf{p}_i)\times(\mathbf{p}-\mathbf{p}_j)|}{|\mathbf{d}|} \end{pmatrix} \tag{6.7}$$

FAST SIGNED DISTANCE FIELD GENERATION We use a fast GPU based algorithm to compute a sampled distance field. It is a variant of the unsigned distance field generation used in [107], which is modified to support the special non-triangle geometry for the line segments. The first step is the creation of a dynamic uniform grid. For this we adapted a common approach for triangle rasterization. For more details on the general technique, we refer to Hasselgren *et*

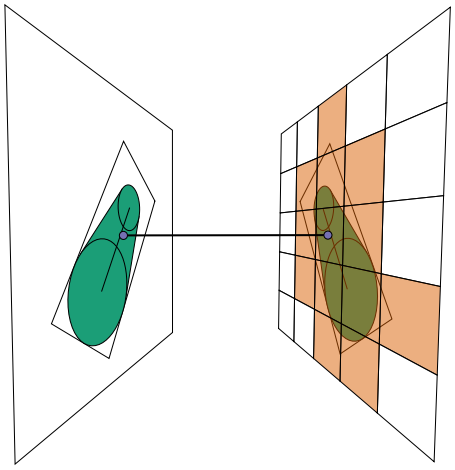


Figure 6.3: The oriented bounding box around the line is projected onto the rasterization surface to generate all fragments, that would be touched by the actual three-dimensional object under an orthographic projection.

al. [69]. The basic idea is to render all primitives with an orthographic projection in a bounding box and use the rasterized fragments to fill a uniform grid in parallel. Each primitive's coordinates are swapped such that the largest amount of fragments is generated. For triangles the largest absolute component of the normal is used as the projection direction. For a line segment we determine the three-dimensional bounding box and choose the dimension with the least extent. This ensures, that the rasterized bounding box area is maximized resulting in a better sampling of the line. This can be achieved with a geometry shader. As the geometry extends further than the line segments themselves, we generate triangles around the line. To avoid problems with extending too far into the projection direction and thus triggering clipping of needed fragments, the line is projected onto the viewing plane in a first step. Around this projected line a bounding box, oriented along the line direction, is constructed. Its size is determined by the segment length and the bigger of the two endpoint radii. During rasterization, fragments are then back projected to find the center depth value. Each fragment can then check a range around that center given by the radius. The setup for the projection is shown in Fig. 6.3. Each cell in the depth range will be tested and if the distance to the actual surface is one cell away, the cell is recorded as a cell to be used for sampling. This process is done twice, in the first step the number of primitives per cells is counted with an atomic add operation. These counts are combined with a prefix sum algorithm to determine indices to be used for storing primitive data for each cell. The second pass then stores all line segments per cell.

Afterwards, a distance transform is performed, which records the nearest cell. This is used in a last step, to refine distance by computing the exact values of a cell center to the geometry stored in the nearest

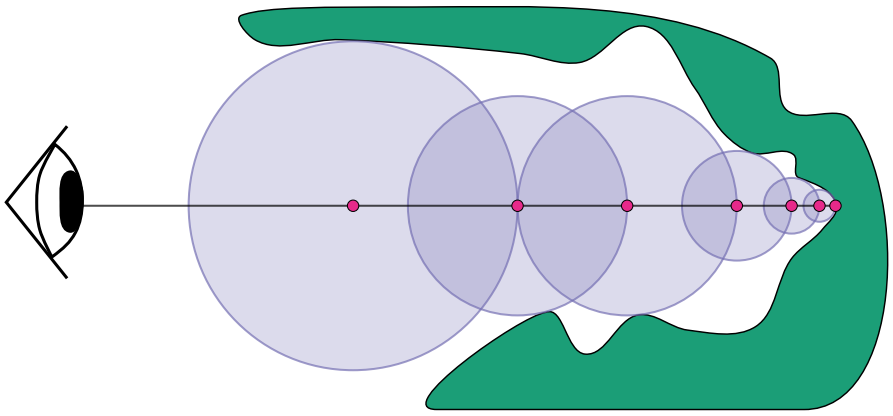


Figure 6.4: Illustration of the sphere tracing algorithm. A ray can be traversed quickly by moving by the amount specified by the SDF.

neighbor cell. Due to the analytic expressions for the distance field, this allows for determining the correct sign without any need for additional computations, that are needed when dealing with triangle meshes.

SDF TRACING ALGORITHM SDFs carry with them geometric information that can be used for various purposes. One of these is to treat it similar to a spatial acceleration data structure. For more general implicit surfaces a ray marching algorithm is required, since the surface is only given as the zero set. Depending on the type of ray marching and implicit some zeros can be missed. In contrast, SDFs can be traversed quicker while not missing any surfaces. This was observed by Hart [68] who developed the sphere tracing algorithm. It is based on the observation, that the SDF provides a minimum distance in which no geometry can be as otherwise that closer geometry would provide a smaller distance. This empty sphere is called an unbounding sphere. It provides a safe distance to move along a ray guaranteeing that no geometry is missed. This is shown in Fig. 6.4. Some problems may arise when only using a sampled distance field. Most notably, small structures can be missed, as the interpolation may not have a negative value to correctly represent a boundary. This is especially problematic when dealing with fine tree structures. To overcome this problem without introducing high resolution fields which are costly both in computation time and storage, we instead use the data computed by the SDF generation algorithm. The tracing uses the sampled distance field as long as it reports a distance larger than the cell size. For smaller distances, the nearest grid cell is looked up and the exact SDF is computed for the geometry contained in it. This also yields the index of the closest line segment. Fig. 6.5 shows the difference of using only the sampled SDF and our exact version. The per-pixel i output of the tracing is the surface position $\mathbf{p}_i$, the closest

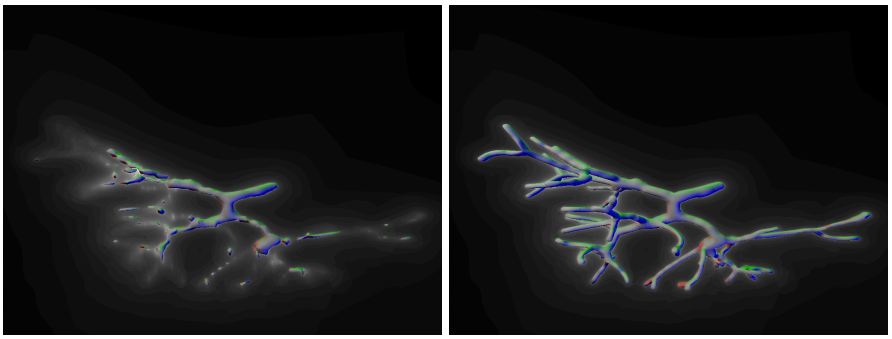


Figure 6.5: Tracing only with a sampled SDF may result in missing small structures due to values being only interpolated (left). Our version, where we move close to the surface with the cheap sampled SDF before switching to the exact computation utilizing the nearest grid cell computed before (right).

line segment's index $\in \mathcal{E}$ and the normal, given by the SDF's gradient.

6.3.3 *Pull-push algorithm*

The PPA was introduced by Gortler *et al.* [55] and makes use of a multi-resolution image pyramid to interpolate scattered data or to reconstruct missing data of an input image. As stated by Rosenfeld [191] such pyramids are an application of the divide-and-conquer principle. Hence, for a given problem, solutions are first found for coarser representations of data (pyramid top) and then successively propagated to finer representations (pyramid bottom). This successive propagation allows to compute individual parts of the image space in parallel, while maintaining a global relationship among all the parts. Therefore, the PPA is perfectly suited for execution on the GPU, which we will exploit for our real-time screen-space parameterization approach. Next, we give a short recap of the PPA as described by Gortler *et al.* [55] and provide the required formulae for later reference.

We denote each unique pixel in the pyramid with index i and the neighbors of i with three sets: the set $\mathcal{A}_i^\bullet$ is the set of neighbors of i in the same pyramid level (the four horizontal and vertical neighbors of a pixel). The sets $\mathcal{A}_i^-$ and $\mathcal{A}_i^+$ describe the neighbors of i in the next lower (finer) and next higher (coarser) level of the pyramid. This topology follows the method described in [147].

THE PULL-PHASE constructs the image pyramid from the input image by iteratively computing the coarser levels:

$$w_i = \sum_{j \in \mathcal{A}_i^-} h_{j,i} \min(w_j, 1) \tag{6.8}$$

$$g_i = \frac{1}{w_i} \sum_{j \in \mathcal{A}_i^-} h_{j,i} \min(w_j, 1) g_j \tag{6.9}$$

where $h_{j,i} \in [0, 1]$ (default: $\frac{1}{|\mathcal{A}_i^-|}$).

THE PUSH-PHASE iteratively reconstructs data from coarse to fine levels in the pyramid, where $h_{j,i} \in [0, 1]$ (default: $\frac{1}{|\mathcal{A}_i^+|}$):

$$\hat{w}_i = \sum_{j \in \mathcal{A}_i^+} h_{j,i} \min(w_j, 1) \tag{6.10}$$

$$\hat{g}_i = \frac{1}{\hat{w}_i} \sum_{j \in \mathcal{A}_i^+} h_{j,i} \min(w_j, 1) g_j \tag{6.11}$$

The above $\hat{w}_i$ and $\hat{x}_i$ contain the reconstructed information from the coarser level. If the current pixel i has already obtained information during the pull phase (i.e., $w_i > 0$), then both can be blended:

$$\alpha = 1 - (1 - w_i)^k \tag{6.12}$$

$$g_i \leftarrow \hat{g}_i(1 - \alpha) + w_i g_i, \quad w_i \leftarrow \hat{w}_i(1 - \alpha) + w_i \tag{6.13}$$

By default, we use $k = 1$, however, increasing this value decreases the influence of the information reconstructed by the push equations. Generally, this means that less attention is given to the global average of data (residing at the pyramid peak) and instead, reconstructed values depend more on their local neighborhood.

6.3.4 *Background reconstruction*

This section describes the reconstruction of the background for the 3D view. The SSP (see Section 6.3.6) is going to parameterize all pixels in screen space of the 3D view. In order to avoid that empty or undefined pixels need to be handled, we construct a valid background from the rendered vascular surface. Kreiser *et al.* [108] used *inverse distance weighting* (IDW) to build a VSS. However, we can as well use the PPA to fill the background. This does not yield the same results as IDW, but since there is no ground truth for a correct VSS, we use the faster PPA approach. The reconstruction cannot be directly applied to the 3D positions $\mathbf{p}_i$, because previously undefined pixels, where $w_i = 0$, would tend to assume the local average of the data being reconstructed. Hence, restored 3D positions would be located towards the center of the rendered surface. Instead, we only reconstruct the

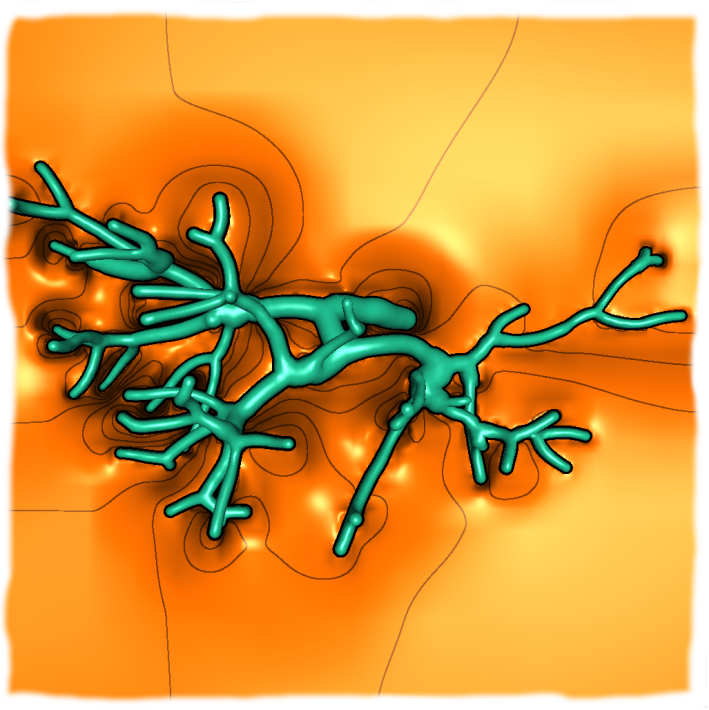


Figure 6.6: Background (orange) reconstructed from a vascular tree. Isolines highlight the depth profile, resembling VSS [108].

depth of a pixel. This yields a smooth height profile that connects with the foreground. Note that during the push phase, we set $h_{j,i}$ to bi-quadratic b-spline weights as used by [147] and the parameter k in Eq. 6.12 is set to 25. The larger k results in a more local approximation of the depth values, which yields a more expressive height profile. We can then use the inverse projection of the graphics pipeline to obtain a valid 3D position $\mathbf{p}_i$ for the given depth value and location of i in screen space [117]. A normal $\mathbf{n}_i$ can be obtained by the second derivative of the newly obtained positions. An example result is given in Figure 6.6. Note that the input surface information is not bound to originate from an SDF sphere tracing. Figures found in this paper that show surfaces other than vasculature are based on mesh inputs, since per-pixel 3D positions and normals can, of course, as well be obtained by rasterizing triangle meshes.

The above procedure is applied to the projection of the 3D surface. For the 2D view, we can as well obtain a background reconstruction. When rendering the line segments of the 2D graph layout, we write the 3D positions of the associated nodes of the input structure into the occupied pixels. The PPA is then applied with default weights to fill the background with 3D positional information. It is not purposeful to reconstruct the background based on the depth values, as done for the 3D view, because the 2D node layout does not live in a projective space. The reconstructed background of the 2D view will help to visualize 3D-based magnitudes, as will be shown in Section 6.5.

6.3.5 *Screen space guiding field*

Here, we describe how a guiding field for the SSP can be generated from the rendered surface contour. The (U, V) coordinates that we are going to generate will be aligned to a guiding vector-field $\mathcal{Z}$. We assume that no field is defined on the input structure and construct it from the so far available data. For this, we look at the depth image produced by the input surface and apply the Sobel filter to obtain a gradient τ_i at each pixel i that is part of the foreground. The resulting gradients are normalized through division by the size of the surface's bounding box size. If $|\tau_i|$ is larger than a threshold t (default: 0.2), then the pixel is considered as a contour generator and we set $\mathbf{z}_i \in \mathcal{Z}$ to the 2D projection of the normal at pixel i. We can then run the PPA with $w_i = w_f$ for the contour pixels and $w_i = 0$ for all other pixels to propagate the contour directions into the undefined pixels. The variable w_f (default: 0.2) can be used to introduce a smoothing into the existing data. Note that $\mathcal{Z}$ is considered in screen space and we process it as a 2D field. Now, in the pull and push Eqs. 6.8 through 6.11 the element g (i.e., the element being reconstructed) is a matrix representation of the 2D vector $\mathbf{z} \in \mathcal{Z}$. If $\mathbf{z} = (a, b)$, then

$$g = \begin{pmatrix} aa & ab \\ ba & bb \end{pmatrix} \tag{6.14}$$

We obtain the resulting 2D vector as the first eigenvector of the averaged matrices. This matrix representation handles the ambiguity of $\mathcal{Z}$. $\mathcal{Z}$ only defines an orientation and not a direction, i.e., $\mathcal{Z} \sim -\mathcal{Z}$. If the eigenvalues of the resulting matrix are equal, i.e., no specific eigenvectors are defined, we simply use one of the considered vectors as a substitute. Further, we set $h_{j,i} = 1$ during the pull phase. During the push phase, $h_{j,i}$ is additionally scaled by $\frac{1}{|\mathbf{p}_i - \mathbf{p}_j|^2}$, so that fragments whose 3D positions are further away have less influence in the weighted average. After a full run of the PPA, the background pixels are populated with directions that smoothly connect to the foreground. Figure 6.7 shows two example images of parameterizations using the SPP, which will be described in the next section.

6.3.6 *Screen space parameterization*

This section describes how we employ the method proposed by Lichtenberg *et al.* [138] to obtain 2D periodic texture coordinates (U, V) in screen-space that are aligned to $\mathcal{Z}$, utilizing the PPA. Note that at this point, each pixel in the image pyramid is populated with a 3D position, normal, and guiding field direction. At the bottom-line, we simply take the formulation from Lichtenberg *et al.* [138] and apply it to the topology of the image pyramid, instead of to the topology of a triangulated mesh. Recall that the topology within one hierarchy level

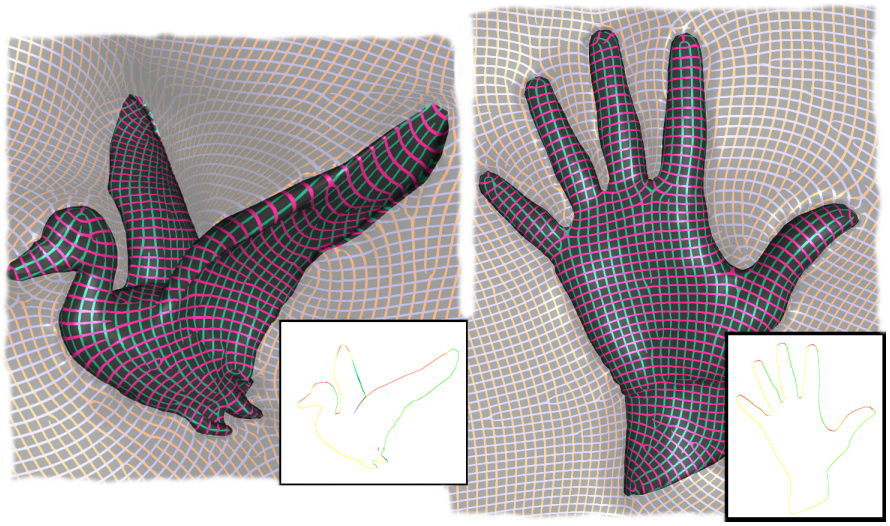


Figure 6.7: Texture coordinates (U, V) aligned to a guiding field created based on the surface contour (insets).

of the PPA pyramid is given by $\mathcal{A}^{\bullet}$ and the topology across hierarchy levels is given by $\mathcal{A}^{\pm}$. We now initialize the top hierarchy level (consisting of a single pixel i) with $w_i = 1$ and $(u_i, v_i) = (0, 0)$. Then, the push operation and r (default: 4) optimization operations take turns until the lowest pyramid level is reached. Details on both operations are described next. Since the U and V coordinates are computed separately, we only refer to U in the following. The computation of V is done analogously with a guiding field orthogonal to $\mathcal{Z}$. Since the field $\mathcal{Z}$ is defined in 2D screen space, but the parameterization algorithm takes a 3D field as input, we project each $\mathbf{z} \in \mathcal{Z}$ into the tangent plane of the respective pixel.

THE PARAMETER PUSH operation reconstructs $u_i \in U$ based on the values at $\mathcal{A}_i^{+}$. The element g in Eq. 6.11 (i.e., the element being propagated through the hierarchy levels) now refers to the Cartesian target coordinate $\varphi_{j,i}$ as in Eq. 8 in [138]. It represents the optimal parameter that a pixel i should assume w.r.t. the pixel j, in order to minimize the energy term used in [138]. The energy of two neighboring pixels is zero, if $|\langle \mathbf{z}_i, \mathbf{p}_i - \mathbf{p}_j \rangle| = |u_i - u_j| f$, where f is a scale factor. This means that the energy is at a minimum if the scaled distance in parameter space is equal to the distance of the two points $\mathbf{p}_i$ and $\mathbf{p}_j$, projected to the guiding direction $\mathbf{z}_i$. The weight $h_{j,i}$ to obtain $\varphi_{j,i}$ from a higher hierarchy level is assembled by several measures:

$$h_{j,i} = \frac{1}{(\mathbf{p}_i - \mathbf{p}_j)^2} \cdot b_j \cdot |\langle \mathbf{z}_i, \mathbf{z}_j \rangle| \tag{6.15}$$

where **p** is the 3D position represented by a pixel, **z** the direction of the guiding field at that pixel and b is a penalty applied to background pixels:

$$b_j = 1 - (1 - m)p_b \tag{6.16}$$

where p_b (default: 0.99) controls the pruning and m is the fraction of (level zero) foreground pixels represented by the pixel of the current level. For example, the pixel at the highest pyramid level represents the whole image and m would be equal to the fraction of foreground pixels in the whole image, while at the pyramid bottom m is either 0 or 1 for background and foreground pixels, respectively. We can observe that employing b_j stabilizes the parameterization at the boundary of the projection of the surface, because the background yields to the foreground. With this modification, we can apply Eq. 6.11 to compute an average Cartesian target coordinate and obtain $u_i = \text{atan2}(\varphi_{j,i})$.

THE PARAMETER OPTIMIZATION operation recomputes U within a single hierarchy level and therefore optimizes the result. We do the same procedure as in the parameter push operation (i.e., optimize the target coordinate as in Eq. 8 of [138]) but use the $\mathcal{A}^\bullet$ neighborhood instead of the $\mathcal{A}^+$ neighborhood. By optimizing the top hierarchy levels and propagating the results down to the lower levels, which are again optimized, a globally periodic parameterization is obtained. Example results can be found in Figure 6.7.

6.3.7 *Frame coherence*

The pipeline described up to this point can be applied to the initial input given by the projection of an input surface, i.e., for a static frame. If a dynamic scene is considered, e.g. a rotating object, then processing the pipeline from scratch for each frame will cause heavy coherency artifacts. In this case we attempt to recycle the results of a frame to initialize the computation of the next frame.

While the 3D reconstruction of the background pixels are smooth as long as the object movements are smooth, very abrupt changes may introduce a hectic visual appearance. To circumvent this, we smoothen the changes to the background morphology over the frames. This is easily achieved by a small adjustment to the procedure in Section 6.3.4. We render the surface as before, but instead of initializing the background with zeros, we copy pixel values from the previous frame. The weight w_i for the background pixels is set to a parameter $S_B \in [0, 1]$ (default: 0.01). Thus, a part of the previous frame's information is pulled into the pyramid (see Eqs. 6.8 and 6.9). As a result, the reconstruction of the background becomes inert and abrupt changes are softened.

The next issue that we will consider is the re-initialization of the U and V parameters. For the background, we copy the parameters from the previous frame. This does not suffice for the pixels that represent the foreground. To establish a link between a pixel in the new frame and its represented location in the old frame, we perform a reprojection. The 3D location of a pixel is transformed back into world space, using the graphics pipeline's inverted view matrix and then projected to screen space with the view matrix of the previous frame. This yields the screen space coordinate to read the previous frame's parameter from. Instead of starting at the pyramid peak as in Section 6.3.6, we pull the information of the last frame into the image pyramid. This is done analogously to the push operation in Section 6.3.6 with the $\mathcal{A}^-$ neighborhood and Eq. 6.9. As utilizing all pyramid levels in this procedure may introduce large changes from frame to frame, we limit the PPA execution to a level l_p (default: 3). Thus, only a local update is performed, achieving a temporally smooth update of U and V.

6.4 IMPLEMENTATION

This section briefs the implementation and the computation time of the individual pipeline steps. The 2D graph layout (Section 6.3.1) is computed in a C++ application, based on the input graph $\mathcal{T}_r$, the node weights and the user parameters p_s and p_h. The 2D node positions are then loaded onto the GPU and edges of $\mathcal{T}_r$ are rendered as triangle strips into the 2D view window. For the 3D sphere tracing, a screen-filling quad is generated to invoke the fragment shader and a SDF-accelerated tracing is executed. The data structure for the PPA is a set of frame buffer objects (FBO), one FBO for each level of the image pyramid. The base resolution m is set to a power of 2. Each FBO has texture attachments for each data field in our pipeline with an appropriate mipmap-level. The communication among the different steps is done solely via these textures, using the OpenGL pipeline and the main steps of our algorithm are the following:

- S1 Generate SDF volume (Section 6.3.2)
- S2 Compute 2D graph layout (Section 6.3.1)
- S3 Render input images (Section 6.3.2)
- S4 Reconstruct background (Section 6.3.4)
- S5 Compute SSP (Section 6.3.6)

Step S1 is only done once, because the input graph $\mathcal{T}$ is static. The 2D graph layout is recomputed if the user-parameters oder the weights that are assigned to $\mathcal{T}_r$ change. Then, in S3 we obtain textures for the 3D view by the SDF sphere tracing (as done for some of the figures in this paper, simply rendering a triangle mesh), holding per pixel 3D positions and normals, as well as a flag defining whether a pixel

Table 6.1: Execution time in milliseconds for steps S3-S5 of our pipeline. The values in the last column refer to timings obtained for $l_p = 3$.

Pixels \ Step	S3	S4	S5
512^2	6-12	0.34	4.36
1024^2	20-30	1.15	15.09

belongs to the foreground or background and a pointer to the closest edge of $\mathcal{T}_r$. For the 2D view, the line segments of $\mathcal{T}_r$ are drawn. In S4, we use the positional information from the input images to reconstruct the background positions for the 2D and 3D view, as described in Section 6.3.4. Lastly, S5 is executed to obtain texture coordinates (U, V) for the 3D surface and background. Here, we use two textures per pyramid level to hold the current parameter information. These texture pairs are used to synchronize the read and write access during the iterative optimization process in a ping-pong fashion. If information from a previous frame is available, we compute the reprojection and initialize the image pyramid according to Section 6.3.7.

After processing the above steps, information about the rendered surface, the reconstructed background and a periodic parameterization is available for the current frame. Section 6.5 will show several example visualization strategies that can be achieved based on the prepared data.

PERFORMANCE To asses the performance of our implementation, we use OpenGL time queries. This means the time of a render pass to execute on GPU is measured. Anything in between render passes (i.e., overhead on the CPU) is not taken into account. We executed our algorithm on a machine with a 4.00 GHz i7-6400 processor, a GTX-1080 GPU and 16GB RAM. The SDF generation is only done once, but could also be incorporated in a dynamic pipeline as our timings show. Generating an SDF volume of size 64^3 took on average 2.1 ms for input trees consisting of about 1000 edges. The generation of a 128^3 volume took 13.1 ms on average. Executing the 2D graph layout algorithm is in sub-millisecond range. Table 6.1 shows the execution times for steps S3-S5 and different screen resolutions. For step S5 we did the measures based on the recycling of U and V with a given hierarchy recycling depth $l_p = 3$.

6.5 APPLICATIONS

This section depicts potential application areas of our method.

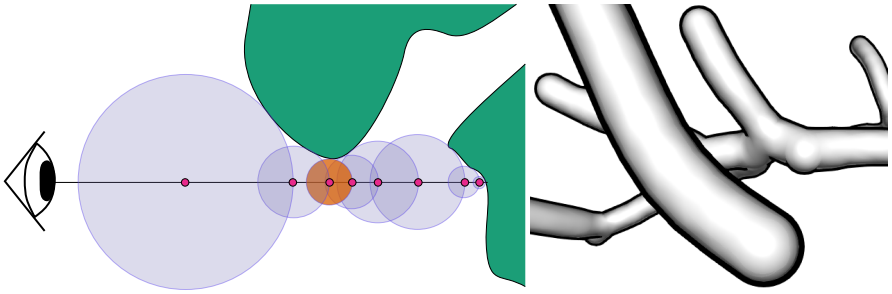


Figure 6.8: The sphere tracing iteration marked in orange depicts the minimal steps size, before the step size increases (left). It represents the minimal distance to a surface passed by the ray. Example of overlapping vessel segments with contours of constant world size (right).

ILLUSTRATIVE RENDERING STYLES FOR IMPLICIT SURFACES

We can use the periodic texture coordinates directly to run a stippling and hatching shader. The distribution of dot and line primitives will be aligned to the guiding field, which allows us to produce results similar to Son *et al.* [206], whose description we follow to generate the primitives (see Figure 6.10, left). The artistic style can also be applied to the reconstructed background in order to produce a shade effect around the object. In Figure 6.1 (right) the minimal distance of vessel segments to the tumor surfaces is encoded. Regions that fall below a threshold (15 mm) are highlighted by a stippling pattern, which is applied to both, the 2D and 3D view. By extending the pattern to the reconstructed background, even small affected areas are more easily recognizable.

Silhouettes and contours of user-defined width can be generated with information from the SDF sphere tracing. For each ray that is traced to generate the surface, we keep track of the minimal SDF value that this ray has passed (i.e., the radius of the smallest sphere as in Fig 6.4). We can then draw the silhouette for at the pixel of each ray that does not hit the implicit surface, but passes it at a minimal distance $d_{min} < d_s$, where d_s is the thickness of the silhouette. To draw contours, a modification is required. We only check and update the minimal distance d_{min} if the tracing step size would increase in the next iteration of the sphere tracing. In this way, d_{min} represents the minimal distance to a point on the surface, that the ray passed by, before hitting another surface (see Figure 6.8). An advantage of the above described contour generation is the possibility to obtain the 3D point on the ray, that is associated with d_{min} as well. This allows to render depth-dependent shadows as proposed by Ritter *et al.* [184]. If d_{shadow} is the distance of the traced surface to the point associated with d_{min}, we draw a depth shadow if $d_{min} < \min(m \cdot d_{shadow}, a)$, where m is the ratio of shadow-depth and shadow-thickness and a is the maximum thickness for the shadow. The texture coordinates

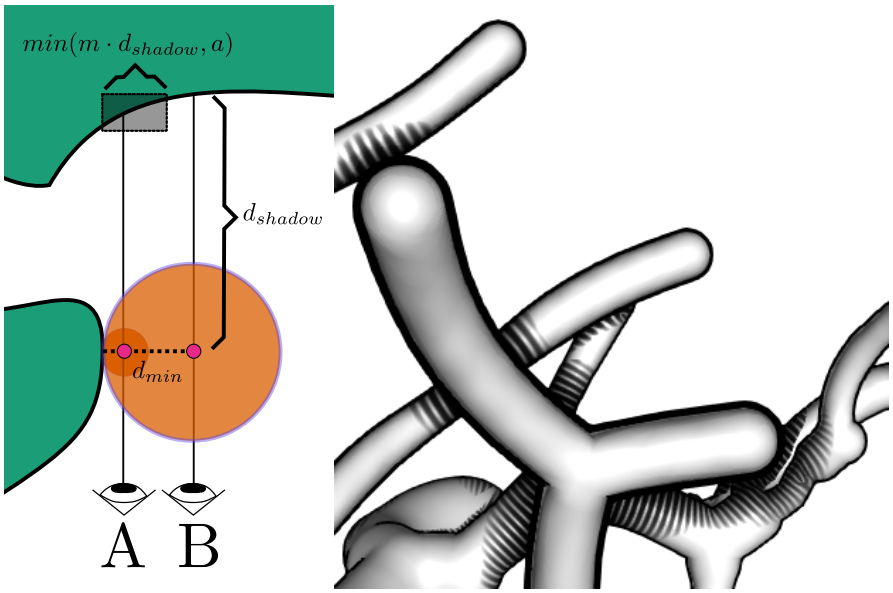


Figure 6.9: Ray A has minimum distance $d_{min} < min(m \cdot d_{shadow}, a)$ and casts a shadow. This is not the case for ray B (left). The depth dependent shadows support depth perception as described in [184] (right).

obtained by the SSP can then be used to apply a hatching scheme. See Figure 6.9 for an illustration and example.

SAMPLE EXTRACTION The (U, V) parameters can be used to extract sample positions from the rendered surface. We propose to apply a flattening to the rendered surface prior sample extraction, so that the distortion of U and V along the surface vanishes. Thus, the samples follow the contour of the surface, but not its curvature. For example, if the 3D positions of each pixel are projected into the camera plane and the normals are set to the normal of that plane, we obtain a flat representation of the input surface. Then, extracted 3D sample positions from a surface can be evenly spaced in screen space.

For each pixel i we compute its distance to the local period interval as $|(u_i, v_i)|$ and store the results in a texture. The texture is then sub-sampled to half resolution via linear interpolation. From this sub-sampled distance texture we extract pixels that represent local minima by looking at the eight immediate neighbors of each pixel. We then extract the 3D position of these pixels from the image pyramid constructed in Section 6.3.4. In a fragment shader, the extracted positions can be fed into a buffer with the aid of an atomic counter variable that guarantees unique write indices. An example showing point primitives rendered at extracted sample locations can be found in Figure 6.10 (right). These positions could be used in glyph based information visualization or other methods that require a near-uniform sampling [149]. We have to point out that if the parameterization

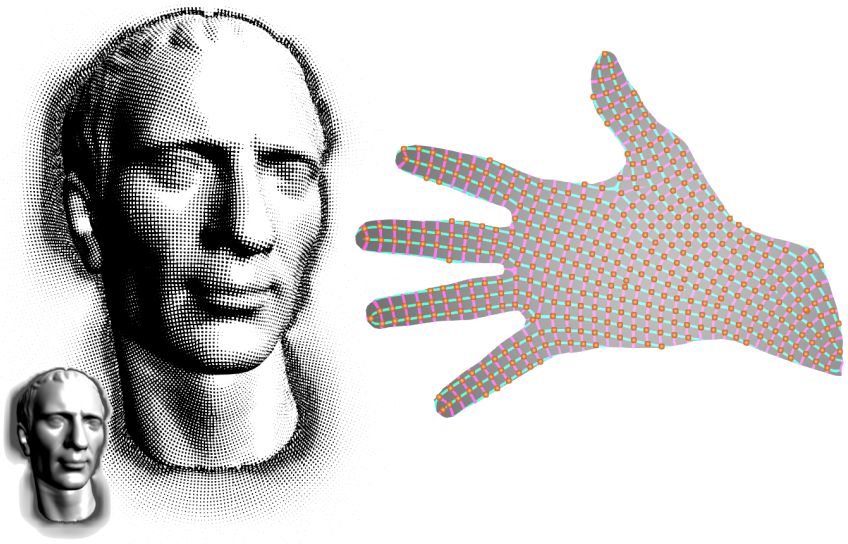


Figure 6.10: Stippling and hatching shader by the formulation of Son *et al.* [206] applied to a shaded surface (left). Sample points extracted from the U, V pattern (right). The orange dots are drawn as point primitives at the extracted 3D locations.

changes (e.g. under rotation), sample points appear and disappear due to the creation and collapse of periodic intervals in U and V. This leads to a visually unstable appearance that may be removed by tracking sample points in order to apply a blending. We leave this for future work. An advantage of this approach may be, that the space of extracted samples is optimized for the view of the current frame.

SURGICAL RISK ASSESSMENT We propose a risk assessment visualization to highlight the strength of the combined 2D and 3D view. Fig 6.11 shows the same vasculature three times in 2D and 3D view with different example access paths. The minimal distance of the vessel segments to the respective path is color-coded in the 2D view and an additional stippling pattern points out regions that are closer to the path than a threshold (20 mm). While it is not obvious in the 3D view, the 2D view clearly reveals which parts of the vasculature are affected. The scalar field which is color-coded in the 2D view is additionally smoothed. This helps to achieve smoother and more expressive isolines. The parts which fall into the risk area, however, are not smoothed and precision is maintained. Another assessment example has already been given in Figure 6.1 (right). In this depiction, the color- and pattern-overlay simply encode the distance of the vasculature to the tumor tissue. Again, the 2D view allows for a quick determination of which parts of the vessel are close - or within a certain range - to the tumor surfaces.

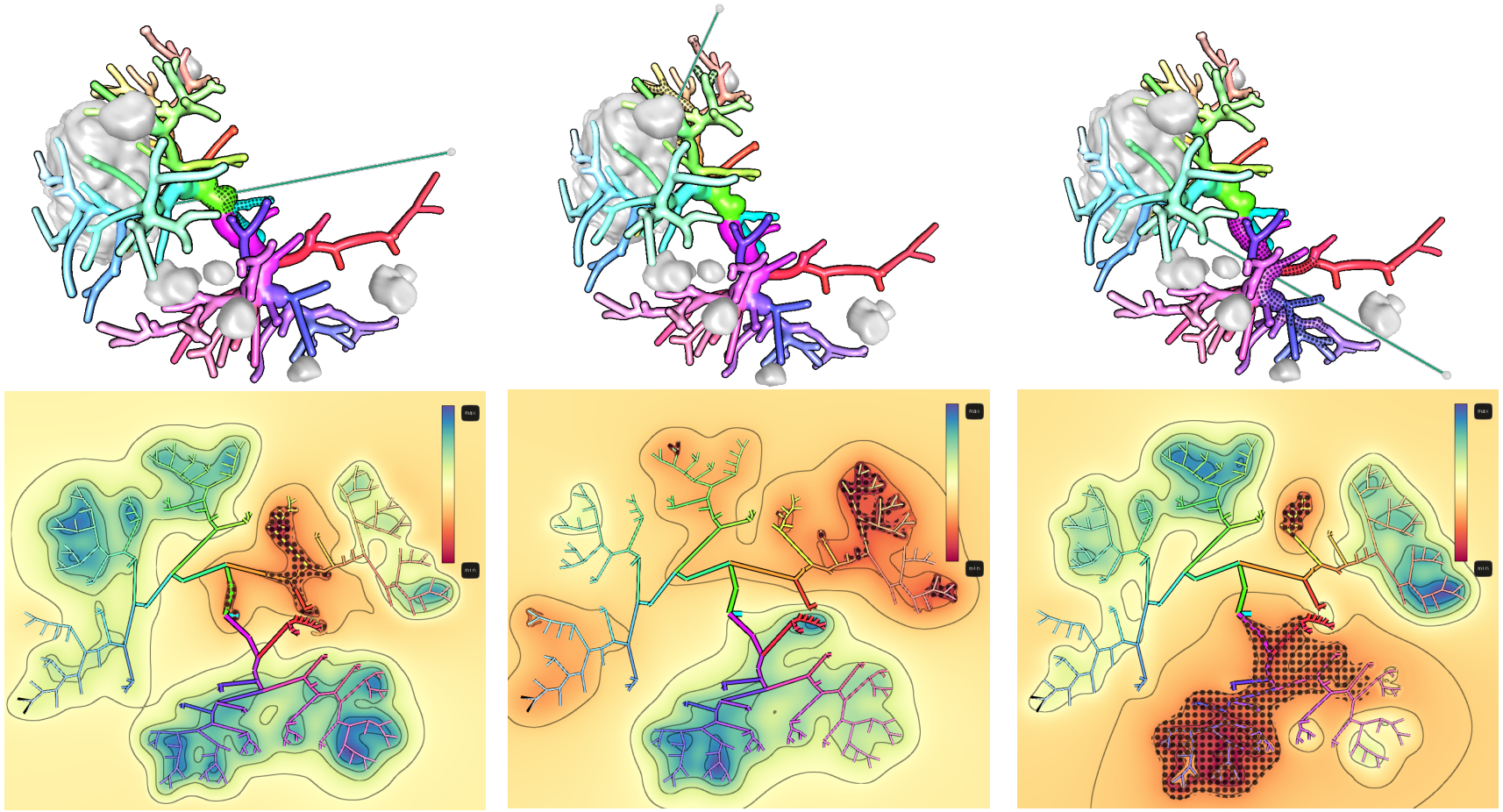


Figure 6.11: An example path to the largest tumor tissue is depicted by a green line (top). The minimal distance of each tree segment to the path is color-coded in the 2D view (bottom). A stippling pattern is applied to all regions (2D and 3D) that are closer than 20 mm to the path).

SURGICAL NAVIGATION With the methods proposed in this work, segment borders within organs, e.g. segments of the liver or lung, could be approximated and visualized in real time. Up to now, these segment borders have only been simulated during the surgical planning process and a change of the object geometry is associated with a considerable computing effort [63]. Using the SDF generation and tracing (See Section 6.3.2), it is possible to adapt the segment boundaries in real time, taking into account the intra-operative deformation of the vascular tree and the organ. Furthermore, the presented visualization method reduces the 3D complexity of the vascular tree to a 2D map. Through the improved overview and reduction of visual clutter, the method has the potential to reduce the risk of accidental vascular injury for future vascular interventions. It enhances existing approaches in the field of 2D map visualization for surgical navigation[65, 120].

BACKGROUND RECONSTRUCTION FOR MEDICAL VISUALIZATION Recently, the work by Kreiser *et al.* [108] used the so called *Void Space Surfaces* (VSS) around projections of medical vascular data to enhance depth perception. They use IDW [203] to interpolate depth values from the vessel contour into the background. A similar result can be obtained by our background reconstruction described in Section 6.3.4, as shown in Figure 6.1 (center, left) and Figure 6.6. The depth approximation of our approach is executed via the PPA on the GPU and therefore distinctly faster than the reference method. The authors report execution times of $25 - 165$ ms on a 1024×768 image, while our method executes at relatively constant $1.15ms$ for a 1024^2 image. This allows to freely rotate the vasculature, while maintaining a smooth background reconstruction. Nevertheless, qualitative differences remain to be evaluated. Further, while Kreiser *et al.* [108] used color-codes and isolines to encode information on the VSS, our SSP could be used to add additional information channels that employ texture patterns.

SDF MODIFICATION The SDF volume holds information about the distance to the closest surface point for each voxel and thus accelerates the tracing of the implicit surface. If the voxel information is modified while maintaining crucial properties of the SDF, the implicit surface can be modified [68]. For instance, primitive combinations can be applied by calculating the union, intersection or subtraction of different primitives' SDFs. Figure 6.12 shows an example, where vessel segments are cut in dependence of their distance to tumor tissue. Note that a cut always produces a valid surface and the implicit surface appears like a solid object (see Figure 6.13, right).

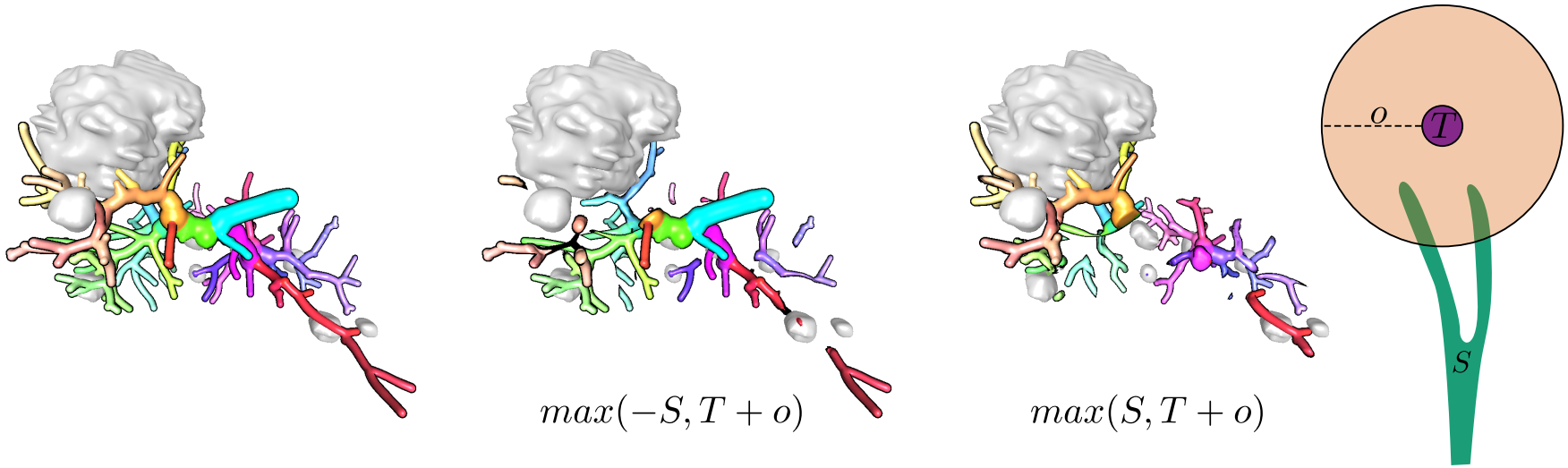


Figure 6.12: Boolean operations, where o = 25 mm is added to the tumor SDF T and combined with the vessel SDF S (right). Plain S (left). T + o subtracted from S (center, left). Intersection of T + o and S (center, right). The mathematical operations are displayed below the figures.

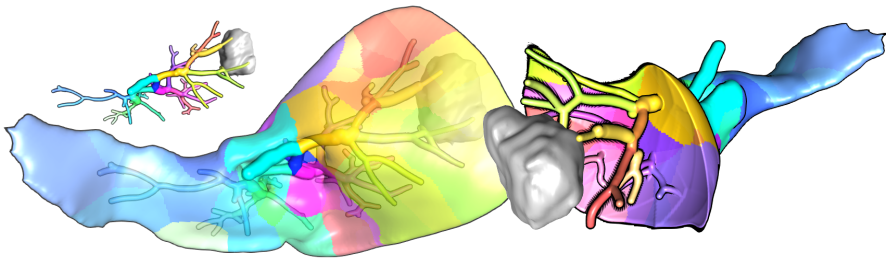


Figure 6.13: Coloring the tumor surface according to the closest vessel segment (left). Combining Boolean operations to expose the vasculature close to the tumor, by carving the liver SDF.

LABEL TRANSITION The surface tracing described in Section 6.3.2 employs a quick look-up of the segment of $\mathcal{T}$ closest to a traced surface point in order to trace the surface without approximation errors. This step further allows to obtain information from the respective segment and to associate it with the traced surface point, as done in the figures throughout this paper to color the 3D surface and 2D graph segments. It can also be applied to transition information to other surfaces as depicted in Figure 6.13, where the tumor surface is colored based on the closest vessel segments. Admittedly, this is a very simple example, but we want to point out that such a parameter look-up is part of our pipeline and can be done with practically no overhead. However, the look-up precision is restricted by the SDF volume resolution.

2D LAYOUT FOCUS The layout of the 2D graph view can be used to guide a viewers attention. A default and an alternative layout are shown in Figure 6.14, with a color-code depicting the distance of graph segments of $\mathcal{T}_r$ to a reference point in 3D world space. Assume the user is interested in parts of the vessel that are close to this reference point. The default layout tries to achieve a balanced distribution of the sub-trees (Figure 6.14, left). In contrast, to obtain the alternative layout (Figure 6.14, left) we assigned higher weights to the sub-trees whose nodes are closer to the above mentioned reference point in 3D. This allows the closer nodes to take more space of the 2D layout (see the sub-trees that are color-coded in a red hue, indicating minimal distance values). Hence, the user's attention can be brought to the important parts of the tree by modifying the tree layout.

6.6 DISCUSSION AND CONCLUSION

We have presented a pipeline that takes a graph representation of a vascular structure as input and creates a combined 2D and 3D view. The 3D surface is implicitly represented as a SDF and efficiently and exactly traced by our proposed procedure. The 2D view is a simple

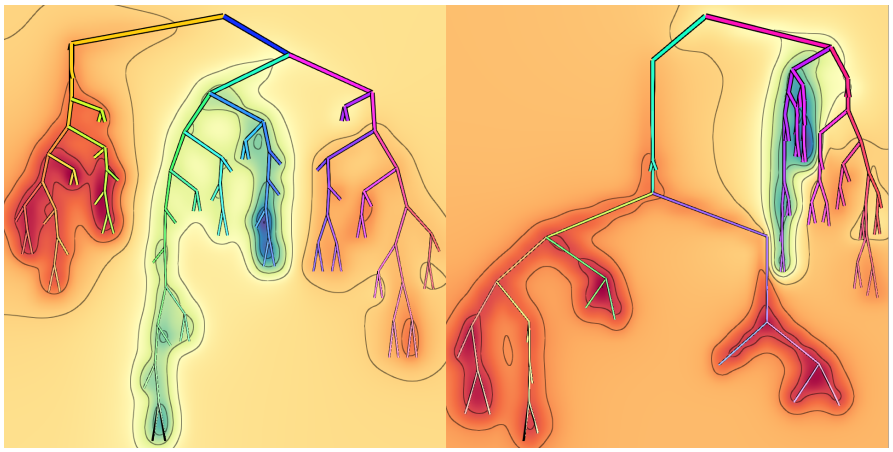


Figure 6.14: 2D graph layout with default weights, based on the number of sub-tree nodes (left). Alternative layout based on scalar field (right).

abstraction of the input graph's branching topology and can be employed to gain a quick overview of measured magnitudes obtained from the medical data. Using different weights for the nodes, different layouts can be generated. As the 3D surface is generated implicitly, we cannot store texture coordinates in order to support the visualization with illustrative styles. As a consequence, we propose a screen-space based parameterization method, that finds periodic and frame-coherent texture coordinates for each rendered frame. It has to be noted that we currently stop the SSP optimization as long as the input parameters do not change. We do this to bring the optimization to a halt, because actual convergence is at the earliest reached after several hundreds of iterations. However, the visual results are already satisfying with this workaround. Further, the recycling depth l_p (see Section 6.3.6) is an important factor to the coherence. A high value will abruptly introduce changes, because updates in a very coarse hierarchy level are propagated to the final image with large impact. A low value may fail to update changes of the input image quickly enough. This is linked to a notable limitation, which is an issue that occurs as soon as the surface to be parameterized leaves and re-enters the view port. Figure 6.15 shows a sequence of frames while the bunny model enters the view port. As no reprojection information is available for the entering part of the surface, the parameterization takes several frames to adapt. A higher value for l_p reduces this effect, but may introduce other coherency artifacts.

The background, i.e., the void space [108] around a vessel, is reconstructed as part of the SSP process. It can be further utilized with the aid of texture coordinates, as shown in the applications section.

We were able to show some introductory examples of the abilities of SDF rendering in visualization. Once the SDF creation and tracing are implemented, the Boolean operations are a way to cre-

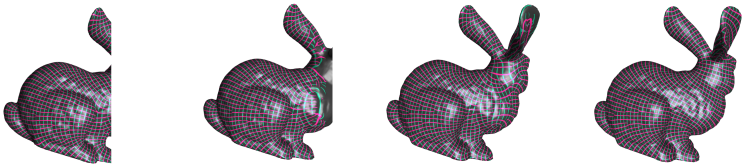


Figure 6.15: Left to right: Sequence of the bunny entering the view port.

ate focus and context applications with low effort and high visual quality. In the future, we would like to investigate to which extent *hypertextures* [169] can be employed to encode data. Hypertextures can be used to modify the SDF in order to deform and alter surfaces. For example, a rough/noisy surface could indicate segmentation uncertainty, while a smooth/even surface indicates high segmentation confidence. The concept of magic lenses, to spatially hide or highlight user-defined parts of the data, could also be implemented on SDF basis. What our SDF procedure currently lacks is the ability to handle input data with non-circular cross-sections. This is an important feature for the faithful representation of vascular data that we would like to tackle in the future.

Finally, we hope that the proposed algorithms and the ability to generate and trace a SDF from model-based input data, as well as to parameterize the traced surface on-the-fly, will motivate further development into this direction.

7

AUXILIARY TOOLS FOR ENHANCED DEPTH PERCEPTION IN VASCULAR STRUCTURES

ABSTRACT This chapter discusses the concept of *Auxiliary Tools* in depth perception. Four recent techniques are considered, that apply the concept in the domain of liver vasculature visualization. While an improvement is evident, the evaluations and conducted studies are found to be biased and not general enough to provide a convincing assessment. The chapter provides background information about human visual perception and a brief history on vascular visualization. Then the four state-of-the-art methods are discussed. Finally, a comparative discussion points out objectives for future follow-up work.

7.1 INTRODUCTION

Comprehensive visualizations of medical data are assumed to improve the accuracy and success of existing surgical and interventional approaches. For instance, 3D images conveying a patient's anatomy or the morphology of organs can be used as a basis to refine surgery planning or to detect anomalies. Such volumetric data, obtained from CT, MRI or ultrasound devices can be visualized on a monitor directly, using volume-rendering techniques, or indirectly. For the indirect approach, the data is segmented, e.g., in order to extract a specific organ. The segmentation yields a surface representation of the object in question, which can be triangulated for efficient storage and rendering. No matter what technique is used, if the morphological properties of the data are of interest they have to be represented faithfully by the computer-generated images. This means that cues, which support the spatial perception in the human visual system, have to be employed. To address this challenge, recent methods employ advanced rendering techniques to communicate information through auxiliary geometry and illustrative styles. The survey by [175] covers a wide range of perceptually motivated visualization techniques in the medical field. They draw a clear line between shape and depth perception but state that both aspects can reinforce each other to support the perception of 3D structures. Both classes are further subdivided to distinguish between different techniques. One of these categories are the *Auxiliary Tools* that were introduced to the medical visualization domain with the work of [122]. *Auxiliary Tools* describe techniques that populate the free space in 3D visualizations with additional geometric constructs. These constructs encode further data or activate and enhance existing depth cues. While the literature often speaks

of depth cues and depth perception, it has to be pointed out that several depth cues are combined by the visual system to derive a three-dimensional perception of an object [80]. In that manner, the predominant motivation of depth perception techniques is usually to improve the spatial perception of a presented 3D object and the depth cue is only a tool for that. However, it can generally be stated, that the encoding of depth in computer graphics occupies parts of the available information channels. Hence, its encoding competes with the encoding of other magnitudes in a visualization. The goal of recent work was therefore to combine depth and parameter encoding in a task-oriented, efficient way. While a common choice is the use of color codes, shading would influence the color map. Renouncing shading means to give up on a natural depth cue, i.e., the influence of light sources. Thus, other depth cues have been brought to attention. In this course, new methods have come up with complex, task-oriented 3D scenes, that more and more creatively utilize the unoccupied virtual space. While more complex scenes allow to pack more information into a generated image (i.e., depth information and medical parameters), questions about the practical feasibility and usability, real-time applicability and extensibility arise.

This chapter will pursue the topic of *Auxiliary Tools* for the enhancement of depth perception in the context of blood vessel visualization. This basically extends the survey by [175], focusing on the *Auxiliary Tools* category which is underrepresented in their survey due to the lack of existing works at their time of publication. At first, a recap of basic information on spatial perception and a short history of blood vessel visualization will be provided. Then, recent work will be presented and discussed in order to provide an overview of *Auxiliary Tools* in depth enhancement. The chapter concludes with a discussion of open question in order to motivate further research in this area.

7.2 HUMAN PERCEPTION

The spatial perception in human vision is the result of the combination of a range of visual cues [80]. These cues are evaluated by the visual system and brain to, for example, estimate the distance between two objects. Hubona et al. [80] state that different models, to describe how depth cues interact, exist. While these models aim to provide a general understanding of the processes of combining several depth cues, one common aspect is important: the posture of the eyes is one of the plenty depth cues. This leads to the assumption that 2D monitors cannot induce natural spatial perception as familiar from within the real world. Even if a real world scene and a 2D image would produce the same projective image on the retina, the posture of the eyes would still be different. In the 2D image, the focus point is always within the image plane. Even stereovision in

head mounted displays (HMD) cannot fully compensate for this, as the study by [212] shows. The work by [80] further wraps up several important depth cues that are also known from natural perception: while *stereopsis* does not apply to the context of this chapter, *motion*, *shadows*, *occlusion* and *partial-occlusion* are important monovision depth cues. Partial-occlusion refers to a transparency effect, that reduces an object's contrast if it is situated further away (e.g., the perception of an object through fog). Of course, even if natural depth cues are present, a subject may not be able to perfectly estimate distances or depth. Therefore the utilization of depth cues in visualization does not only aim to reproduce natural perception, but also to enhance it. With this, the estimation of spatial properties which are prone to human error are sought to be reduced. As mentioned earlier, apart from encoding spatial properties of a 3D scene, it is desired to encode and visualize further magnitudes. A computer-generated scene that is observed by a subject is processed in two phases [186]. The visual cues that are processed in these phases are categorized in the survey by [188] and can also be applied to the context of auxiliary objects. The formulation is brought into a larger context by [18]. Namely, a *pre-attentive* and *attentive* phase is considered in the human perception process. During the pre-attentive phase, which covers a short time span after visual cues are exposed to the viewer, stimuli like color, size, shape and orientation are perceived in a cumulative manner. These can trigger a *pop-out effect*, influencing the viewers attention. During the attentive phase, the viewer is more focused on details and interactive exploration of the data. In depth perception, for example, the attentive process may cover the explicit comparison of two points in a scene w.r.t. their depth. More information on human perception in the context of visualization can be found in the work by [71].

7.3 BRIEF HISTORY OF DEPTH ENHANCEMENT IN VASCULAR VISUALIZATION

The area of enhanced vascular visualization is actively being dealt with in the medical visualization community, because of the significance of vascular morphology in different medical subjects. For example, the blood vessels around a liver tumor may significantly influence the intervention strategy for an ablation or resection [66]. The visualization of blood vessels has emerged from magnetic resonance angiography (MRA) imaging. Due to the high intensity values assigned to vasculature, the maximum intensity projection (MIP) is a common way to visualize the obtained image data. A first method to extract and visualize 3D surface data obtained from MRA was introduced by [52]. After that, more extraction and visualization techniques emerged and also the incorporation of depth cues received at-

tention. A concise overview with further references is given by [189]. Their work also embodies the first implementation of methods to enhance depth perception in 3D angiographic data, that rely on monoscopic features only. Their depth cues are inspired by natural depth cues, but are realized in an exaggerated way to improve their effect. The *pseudo chromadepth* (PCD) introduced by [189] can be seen as a variation of the *chromaDepth* (CD) [207]. These techniques induce depth perception through chromatic aberration.

From then on, the monoscopic depth cues that received the most attention are the following: Overlap, PCD, aerial perspective and kinetic depth. These cues have been evaluated and compared against stereopsis in the work by [96]. The overlap is usually magnified by visually enhancing edges or rendering halos and the aerial perspective is induced by reducing the contrast of points that are further away from the viewer. The kinetic depth cannot be employed in still images and is restricted to interactive applications. The techniques to this point can be directly applied to the surface of a rendered vascular structure. More recent publications have designed more complex but, to some extent, less general approaches. Instead of applying variations to the visual appearance of the vasculature in question, they use additional geometry to encode information.

Additional geometry can contribute to various depth cues. It can introduce further overlapping cues or be used to simulate perspective distortion. If the shape of the geometry is very simple (e.g. a circle, rectangle or bar), variations in size are easier to perceive. For example, if the circles in Figure 7.2 (left) are assumed to have the same real world size, then the green circle must be assumed to be further away, because of its smaller projection. Another advantage here is that such objects can be decoupled from the vasculature in the graphics pipeline. Therefore orthographic projection can be used to depict the medical data in a common manner, while helper objects are resized to simulate perspective projection, expressing the related depth cues. Additional objects in a scene to encode information are known as glyphs (surveyed in the medical to domain by [188]). However, the following section will discuss techniques that utilize additional geometry whose concepts exceed the common goals of glyphs.

The use of enhanced depth perception in vascular visualization can be divided into three categories. The most basic category covers the *natural cues*, such as occlusion, fog, shadows and perspective distortion. Examples can be found in [52, 60, 73, 97]. At the next stage, authors began to *alter the appearance* of depicted vascular models in order to encode depth information. Prominent examples are CD [207] and PCD [189], that applied a color function to the surface. [24] used halos (i.e., exaggerated contours) to enhance depth perception and [184] drew illustrative shadows on vascular surfaces to highlight overlaps. Recently, the category of *Auxiliary Tools* has emerged. Here, additional

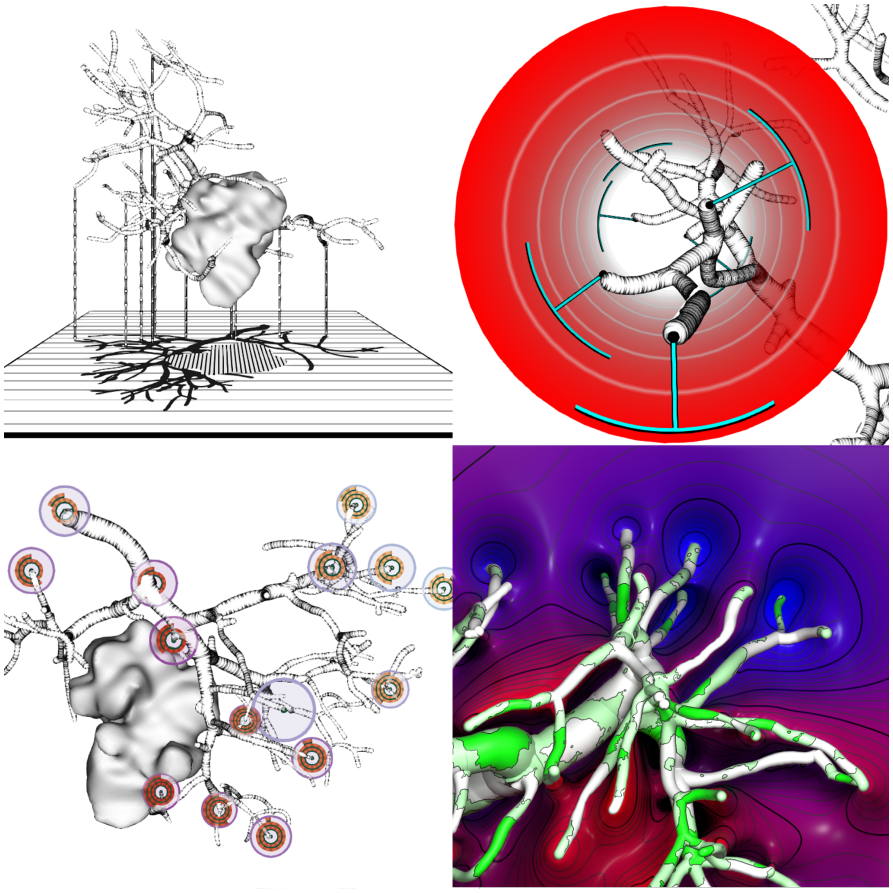


Figure 7.1: Overview of the four techniques: SL (left, top), SA (right, top), CCG (left, bottom) and VSS (right, bottom, *courtesy of Julian Kreiser, Ulm University*).

geometric entities are used to convey depth. These entities can be data glyphs, but may also extend to more complex designs. The next section will discuss four techniques from this field.

7.4 AUXILIARY TOOLS

This section describes four techniques that utilize *Auxiliary Tools* to enhance depth perception. The examined methods are the *Supporting Lines* (SL) and *Supporting Anchors* (SA) by [124], the *Concentric Circle Glyphs* (CCG) by [132] and the *Void Space Surfaces* (VSS) by [108]. All techniques are depicted in Figure 7.1. Though introduced as a category of depth perception techniques by [175], *Auxiliary Tools* are not properly defined as such. A definition can be phrased as follows:
Auxiliary Tools in depth perception describe visual entities, (i.e., geometric objects) that augment a generated image of spatial data in order to encode depth information or to trigger and/or exaggerate depth cues.
Figure 7.2 shows an example of how an auxiliary object can trigger a depth cue. While the illustrative shadows by [184] can technically

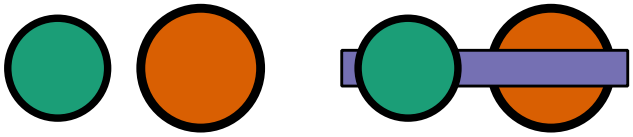


Figure 7.2: The depth difference of the two circles can not be perceived (left). The depth difference of the two circles becomes evident after adding an auxiliary object (purple bar) that triggers an overlapping depth cue (right).

be achieved by drawing geometry based hatching strokes, we refrain from including this work to auxiliary tools. The rationale is, that the hatching strokes are rather perceived as a shading style than as additional objects. The fact that additional objects are added to a scene leads to the problem that the *Auxiliary Tool* may negatively interfere with or obstruct the actually visualized data. Therefore a careful and task-oriented design of *Auxiliary Tools* is crucial. This also means that certain *Auxiliary Tools* may only be suitable for a limited range of types of data. For example, vascular trees cover the 3D and screen space rather sparsely, hence, room remains to add further objects. Additionally, the vessel branch- and end-points provide expressive landmarks that can be used to attach auxiliary objects to. The techniques that will be presented in the following subsections all aim at the improvement of depth perception in the visualization of liver vascular trees.

7.4.1 *Supporting Lines*

In the given medical context, SL [122] embody the first occurrence of *Auxiliary Tools* [175] (Figure 7.1, left top). The supporting geometry used in this method comprises of a plane, situated below the vascular structure, and supporting lines that connect user-defined points on the vasculature with the plane. Further, a depth dependent contour is drawn. The plane works as a canvas to cast a shadow of the 3D structure, triggering an additional depth cue. Further, the shadow can be drawn in various styles to encode information like the source of the shadow (e.g., vessel or tumor tissue). A grid on the plane enhances the capability to sort multiple supporting lines by their depth. The supporting lines (inspired by [54]) provide a further link between the vessel and its shadow to simplify the reading and are drawn in a ruler-like style to allow assessment of the distance to the plane. At the same time, the lines trigger overlap cues, so that depth differences of distant vascular branches can also be estimated. The contour width (inspired by [24]) decreases quadratically with increasing depth. This exaggerates the effect that a perspective distortion would have on the width, therefore amending depth perception. The combination of

these cues supports a precise (support lines) and a global (contour) perception of depth.

7.4.2 *Supporting Anchors*

The SA [124] can be seen as a follow-up work on SL (Figure 7.1, right top). Instead of a shadow plane, a cylinder is used as a reference object. The height of the cylinder is aligned with the view direction. The cylinder can be placed anywhere, also intersecting the vasculature, and therefore allows to define a focus-region. The cylinder is a simple structure and the user looks along the height of the cylinder, hence it is intuitive to determine the relative depth of points on the cylinder. As the depth of a point increases, the perspective distortion will move the point closer to the center of the cylinder's projection. Similar to SL, points on the blood vessel are linked to that cylinder. This link has an anchor-like shape that clings to the cylinder and helps to trace the point's depth along the cylinder's circumference. The surrounding vessel branches may intersect the cylinder, which also indicates the depth of the intersection. In this way, SA helps to lift the complex vascular shape to a more comprehensive geometry and finally enhances depth perception. However, it is difficult to infer information about parts of the vascular structure that are not attached to anchors and do not intersect the cylinder.

The work further addresses the issue of overlapping auxiliary objects, as the authors provide an algorithm that automatically finds a pre-defined number of vessel end-points that are linked to the cylinder. The selection of the end-points is done in a way that balances the distribution across the depth and the cylinder so that a clean visual result is achieved.

7.4.3 *Concentric Circle Glyphs*

The CCG [132] are an attempt to encode depth information without relying on spacious reference geometry like the previous shadow plane or cylinder (Figure 7.1, left bottom). Semi-transparent disc objects are attached to selected vessel end-points. The discs are gradually filled with up to three concentric circles, as the object's distance to the viewer increases. Further, overlapping cues are created in the close proximity of selected points. The filling of the concentric circles is done by completing the circles one after another in a clock-wise manner. The subdivision into three circles therefore allows a very fine-grained distinction of mapped depth values. It is also stated that the disc shape itself can be altered (e.g., to a rectangle) in order to encode additional data. Moreover, the size of the discs is dependent on the depth, therefore an exaggerated effect of perspective distortion

is possible. As a drawback, structures away from CCG instances do not benefit from the technique.

The method includes, similar to SA, an approach to avoid overlaps. In this case, the disc objects should not interfere and the selected points are chosen so that the screen space is evenly covered. Additionally, pairs of vessel end-points with a low depth difference are preferred to help with specifically difficult situations.

For this article, a further analysis of the data obtained during the evaluation of [132] is done to augment the descriptive results presented in the original paper and to bring it in line with the evaluations of the other techniques. Two subjects were removed from the data, as they achieved less than 40% of correct answers for at least one of the tested techniques. A Shapiro-Wilk test reveals that the obtained measurements are not normally distributed. Therefore a Friedman's ANOVA to test for the statistical significance is conducted with a post-hoc Wilcoxon signed-rank test, as recommended by [195]. Results are obtained as $\chi^2(2) = 18.9, p < 0.0001$ for the precision, $\chi^2(2) = 7.96, p = 0.0187$ for the reaction time and $\chi^2(2) = 18.9, p < 0.0001$ for the confidence. This means, that statistically significant differences among the tested groups (SL, SA, CCG) are present. As suggested by [175], the effect size that describes the difference between groups is also reported. Because of the non-parametric nature of the measured data, the Wilcoxon signed-rank test is applied to obtain a z-score for each pair of techniques. From that, effect sizes are derived for the precision measure: CCG-SL = 0.122, CCG-SA = 0.157, SL-SA = 0.042. This reveals that the actual difference in performance of the tested techniques ranges from small to very small, while the CCG is still ahead of the other approaches.

7.4.4 *Void Space Surfaces*

The VSS [108] follow a very different approach (Figure 7.1, right bottom). Instead of attaching supportive objects to selected points on the vasculature, they augment the whole free background (the void space) of a scene with a surface as an *Auxiliary Tool*. The surface is attached to the contour of the 3D structure and smoothly interpolates between sections of different depth. For this, *Inverse Distance Weighting* [203] is used, that allows to control the VSS smoothness by a user parameter. It is further equipped with isolines that allow the user to trace regions of similar depth. Illumination of the VSS introduces extra depth cues and amplifies the improved depth perception. This approach is by its concept free of any overlaps. Also, it is not required to pre-select any points of interest from the vessel. Instead it applies to the whole contour. Another advantage is that the method does not interfere with the projection of the vessel tree. The vascular surface can therefore be used to encode other parameters. A difficulty arises when two

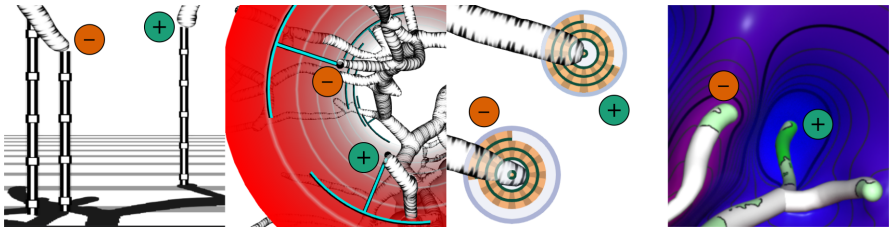


Figure 7.3: Closeup of all four techniques (SL, SA, CCG, VSS) with two vessel end-points marked each. The green (+) label indicates the point with a higher depth.

points that one wants to compare are not directly connected by the VSS. However, in such a case the surface itself can be color mapped, for example with PCD colors, to allow a comparison.

7.5 EVALUATION

This section examines the quantitative evaluations conducted in the previously described works w.r.t. their study and task setup. After that, a comparative discussion of the results is provided to derive question for future work.

7.5.1 *Evaluation Overview*

All evaluations follow the scheme presented by [96]: the subject is shown a vascular structure, with two points on that structure clearly marked. The subject is then asked to determine the point that is closer or further away. Examples with the correct answer indicated are depicted in Figure 7.3. This task is repeated several times with alternating point sets, vascular structures and visualization techniques. The quantitative measures include the *percentage of correct decisions* (i.e., precision) and the *time required for a decision* (i.e., reaction time). These quantities were then used to derive a sound ranking of the tested depth perception enhancement techniques. Table 7.1 gives an overview of the average precision for the tested techniques per paper, along with the number of participating subjects and stimuli per subject. It can be observed that the performances for basic Phong shading and PCD vary significantly across the papers, indicating that the tasks were notably different. The overview excludes the results from [96], because they tested a range of classical depth cues, but here we focus on *Auxiliary Tools*. A link to their findings can be established via the results for PCD, which was rated best by [96] and therefore serves as the reference visualization.

More considerations were made w.r.t. the precise task and stimuli setup. The first is that in all publications, the tested point pairs were located at vessel end-points. However, while the auxiliary elements

Table 7.1: Average precision for evaluated techniques in percent, number of subjects and number of stimuli per subject for each paper.

	Phong	CD	PCD	SL	SA	CCG	VSS	Subjects	Stimuli
SL [122]	26		54	84				50	24
SA [124]	48		79		87			81	24
CCG [132]				92.2	87.2	93.6		24	45
VSS [108]	73	91	94				92	20	150

in SL, SA and CCG are directly attached to individual end-points, the VSS is attached to the whole vessel outline and fills the whole background. Therefore the VSS evaluation utilizes additional crosshair pointers to indicate the points in question. All works describe criteria that were used to come up with a set of tasks with a balanced difficulty level.

[96] found that the screen-space and depth distance of two points affect the decision performance. As a consequence, this has been considered in the surveyed evaluations, however, not in a mutual way. [122] (SL) used labels XY, where X describes the screen-space distance and Y the depth distance. The labels can be set as F (far), if the individual distance is more than half of the maximal possible distance for a given vascular model and N otherwise. In the follow-up work [124] (SA) this labelling was omitted. Instead, the tasks were restricted to point pairs with a depth distance of less than 20 mm. The work by [132] (CCG) used the F and N labels again. As an additional challenge, point pairs were chosen such that the circle glyphs would overlap, making readings more difficult. This is a design choice quite specific to the visualization method. [108] (VSS) assured that the depth distance of point pairs was at least 10% of the total depth range in the image. The screen-space distance was covered such that each point was clearly assignable to either the left or right part of the screen. A more specific criterion applied was whether the point pairs were directly connected through the void space or not. In summary, all papers did an attempt to reduce the data selection bias [173].

Intuitively, the complexity of the vascular structures used during the individual tasks is also expected to have an effect on the results. [124] employed eight models, while [132] and [108] had, respectively, five and six data sets available. Unfortunately, the complexity of the structures was not considered in the analysis in any of the publications. Supposedly, properties like number of end-points, branch segment length or the ratio of vessel thickness and vasculature size may provide a rudimentary description of structural complexity. Such could then be incorporated into further statistical analyses.

The actual tasks were all performed in an automated application. Before, training questions were completed to familiarize the subjects with the task. Then, stimuli with different visualization techniques,

point pairs or vessel trees were shown to the subjects. For SL, SA and CCG the subjects hat to mouse-click a radio button in order to submit their decision. Additionally, they were asked to estimate their decision confidence. The VSS paper describes that the subjects were to press one out of two keys to choose between the presented points. This is a cleaner design when it comes to the reaction time, because the interactive and cognitive overhead is minimized. However, asking for the perceived confidence for each stimulus allows to better interpret the perceived usefulness of individual techniques. An aspect that is missing in the present studies is to compare user confidence separately for correct and wrong decisions. It may occur that some visualization techniques induce an overly high confidence while suggesting wrong decisions. Therefore, confidence ratings have to be treated carefully. Another difference in the conducted studies concerns the type of projection used to display the 3D data. Orthographic projection is common in the medical context because of the physicians being familiar with CT or MRI slice views, that resemble parallel projections. While CCG and VSS used orthographic projections, the methods SL and SA are bound to perspective projection. Therefore an additional depth cue was present in the latter methods.

The subject selection bias [173] is of importance if subjects from different domains and with varying abilities or knowledge are selected. For the medical context of the presented techniques it would be desirable to access a pool of subjects from a physicians or surgeons domain, but specialists are not always available. For the SL study, five out of 50 subjects were physicians and 19 had experience with vascular visualization. The SA study contained 81 subjects, with 15 being physicians and 25 with vascular visualization experience. There were no physicians in the CCG study and seven out of 24 had experience with vascular visualization. The VSS study was also conducted without medical experts. Based on the results presented by [96], we can state that experts do not necessarily perform better than lay people, which leads to the assumption that the tested techniques apply well to the general perception of depth, regardless of additional knowledge. However, there are differences in performance among different subject groups, indicating that certain groups better accept certain techniques than others. Regardless of this observation, the studies of the four techniques in this chapter do not distinguish between subject groups.

7.5.2 *Comparative discussion*

As described previously, the setup of the four evaluations have a common denominator but are nonetheless quite different. Therefore a direct comparison of the results is not possible. Instead, a comparative discussion of the methods follows.

The methods SL, SA and CCG can be grouped together, as they all utilize additional geometry that is attached to pre-selected points on the vasculature. From their evaluation results, it can be concluded that this positively affects the precision of the subjects. The evaluations, however, do not cover the performance when deriving information about points that are not attached to the auxiliary objects. It is to be expected that the precision drops drastically in this case. In this aspect the VSS can be assumed to be superior, as it supports the whole vessel contour. Unfortunately, the VSS evaluation does not underline this strength. In return VSS lacks the ability to provide a distinct perception of differences if two compared points are not connected by the VSS. Here, the other methods prevail, because a visual link can always be established. The necessity to examine differently difficult tasks is underlined by the evaluation for SL and CCG and should be taken into account in future works.

The precision of the first three methods comes at the cost of additional visual load and overlaps. Depending on the situation, this may lead to unwanted obstruction of the visualized structure and introduce visual clutter. Hence, the VSS has the cleanest appearance of all the methods. This should also be beneficial if the VSS is combined with the encoding of other parameters on the vessel surface.

Another aspect to be considered is the region of interest (ROI) that a user might focus on. While SA nicely defines such a ROI through the cylinder, SL always needs to keep the shadow plane visible. Therefore, SL is not suitable for zooming in to a structure. CCG as well as VSS do not suffer from this restriction, as they are not bound to exterior context geometry.

It would further be interesting to examine whether combinations of the above methods can achieve generally better results. Augmenting the VSS's global depth overview with the CCG's precision might be a viable option.

From the above observations, the necessity for extended evaluation setups arises: while all methods claim to leave visual information channels open to encode additional data, none of the publications addresses this aspect in their analysis. VSS exemplifies this by displaying wall shear stress on the vessel using a color map. This mapping may interfere with the color mapping of depth, so a combined evaluation should be conducted. This also applies to the other techniques Generally, applying a color mapping to the vascular surface is problematic due to employed shading effects. If the shading is omitted, structural features are lost and the overall spatial perception suffers. Further, the range of differently difficult tasks should be more carefully designed and evaluated. In particular, global comparisons (i.e., comparison of any arbitrary point pairs) and precise comparisons (i.e., comparisons of pre-selected point pairs with just noticeable depth differences) should be considered. The task difficulty should

further incorporate vascular models of varying complexity. However, a definition for this complexity would be required beforehand. As stated in Section 7.2, depth cues are combined to a spatial impression of a 3D scene. Therefore, tasks in a study should not only test the performance of depth perception, but also of spatial perception. For this, subjects should be asked to estimate distances in all spatial dimensions. With respect to the two phases of human visual perception (pre-attentive and attentive), tasks should be created to target one of these specifically in order to determine how well a visualization technique exploits each phase. The F and N labels introduced by [122] may be suitable to support this as one would expect attentive, precise readings for NN configurations and quick determinations for FF configurations. From the results presented by [96], it can be concluded that different groups of subjects perform differently well with presented depth enhancement methods. This should be considered in future studies as well.

The above suggestions are only concerned to consolidate the study results w.r.t. the depth and space perception task. However, as pointed out by [173], research in the medical visualization domain often lacks a direct connection to actual clinical requirements and potential to be integrated into clinical work flows. This also applies to the methods presented in this chapter, what becomes already apparent in the low number of subjects with clinical knowledge. In order to design studies that better reflect a potential clinical application of the techniques, the focus needs to be shifted away from the sole improvement of depth perception, towards integrated and task-oriented applications. It should be an application that heavily relies on spatial comprehension of the presented vasculature such as a needle guiding scenario for liver ablations [2]. Finally, all techniques report statistical significance for their improvement in depth perception in comparison to previous methods Such reports would be more convincing when covering the above mentioned aspects in order to more formally depict the strengths and weaknesses of each technique.

7.6 CONCLUSION

A recent category for the improvement of depth perception, *Auxiliary Tools*, has been surveyed in this chapter. The methods show that advanced rendering and visualization techniques can contribute to an improved depth perception. This improvement of depth perception itself is motivated by the necessity to comprehensively visualize complex structures, such as vascular trees. Evaluation results, however, indicate that the studies were conducted under significantly differing circumstances, which includes the task difficulty, selection of data and selection of subjects. To align the methods' performance a more extensive study is required. Aspects to consider for this were

suggested earlier. The framework proposed by [150] may serve as the basis for such. Subjects with clinical knowledge could be tested under lab conditions, while lay subjects can be reached via the online questionnaire generated by their tool.

In summary, the existing methods prove to fulfill the task of improved depth perception. However, this is only a small piece of the cake and insights into their performance in real world clinical applications are still to be gained. Thus, the topic of *Auxiliary Tools* in depth perception should be further pursued, especially with the aid of clinicians.

Part IV

DIMENSION REDUCTION

The previous part made a contribution to the dimensional abstraction of vascular tress. This part is further about data abstraction through dimension reduction, but covers different data sets and goals. In MD simulation data, complex 3D structures are given over a time range. Reducing this 3D+t data to a 2D map allows domain experts to gain novel insights into their data (Chapter 8). A second contribution is presented in the context of mitral valve analysis (Chapter 9). Convoluted 3D surface data of the valve is unfolded to a 2D representation. The new abstraction object allows to display existing and to derive novel physiological parameters that may be interesting for clinical analysis.

This part consists of the following papers:

Lichtenberg, N., Menges, R., Ageev, V., George, A. P., Heimer, P., Imhof, D., Lawonn, K., "Analyzing Residue Surface Proximity to Interpret Molecular Dynamics." In: *Computer Graphics Forum*. Vol. 37. 3. Wiley Online Library. 2018, pp. 379–390. DOI: 10.1111/cgf.13427

Lichtenberg, N., Eulzer, P., Romano, G., Brčić, A., Karck, M., Lawonn, K., De Simone, R., Engelhardt, S., "Mitral valve flattening and parameter mapping for patient-specific valve diagnosis." In: *International Journal of Computer Assisted Radiology and Surgery* (2020). DOI: 10.1007/s11548-019-02114-w

8

ANALYZING RESIDUE SURFACE PROXIMITY TO INTERPRET MOLECULAR DYNAMICS

ABSTRACT The surface of a molecule holds important information about the interaction behavior with other molecules. In dynamic folding or docking processes, residues of amino acids with different properties change their position within the molecule over time. The atoms of the residues that are accessible to the solvent can directly contribute to binding interactions, while residues buried within the molecular structure contribute to the stability of the molecule. Understanding patterns and causality of structural changes is important for experts in the pharmaceutical domain, e.g., in the process of drug design. We apply an iterative computation of the Solvent Accessible Surface in order to extract virtual layers of a molecule. The extraction allows to track the movement of residues in the body of the molecule, with respect to the distance of the residue to the surface or the core during dynamics simulations. We visualize the obtained layer information for the complete time span of the molecular dynamics simulation as a 2D-map and for individual time-steps as a 3D-representation of the molecule. The data acquisition has been implemented alongside with further analysis functionality in a prototypical application, which is available to the public domain. We underline the feasibility of our approach with a study from the pharmaceutical domain, where our approach has been used for novel insights into the folding behavior of μ-conotoxins.

8.1 INTRODUCTION

The surface of a molecule provides important information about its binding behavior. In 1971, Lee and Richards described the topology of the surface as directly related to the function and interaction of a molecule with other molecules [129]. The topology is of high interest in fields like drug design or medical research where molecules are arranged, simulated and evaluated to investigate their interaction behavior with existing molecules. Since the outcome of these evaluations is an important indicator for the scientists about their design decisions, the process is worth to be supported by data acquisition methods and advanced visualization techniques.

Molecular Dynamics (MD) simulations have become a staple ingredient to any drug discovery pipeline used by both academics and pharmaceutical companies [39]. MD simulations produce data that contains hundreds of thousands of time-steps, covering the time span

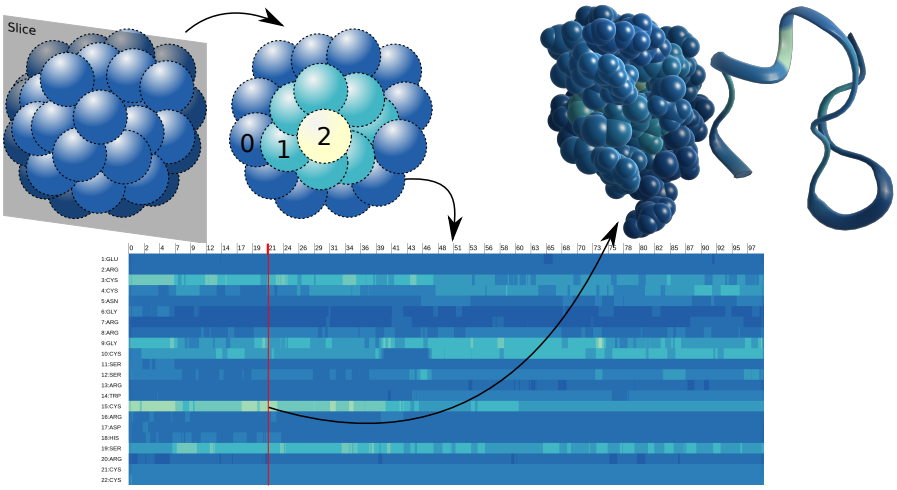


Figure 8.1: From an input molecule (top, left), we derive a surface proximity term (top, center, left) that represents the closeness of an atom to the molecule surface. We apply this method to a Molecular Dynamics simulation to obtain a color-coded map, representing the trajectory for each atom or residue (bottom). Single time-steps can be visualized in 3D (top, right), e.g., VdW-spheres or in a combined backbone-helix form.

of a molecular reaction or folding process. The time-resolution of the data may vary in dependence of the application. The amount of data and possibilities for altering the simulation conditions is constantly growing with the advance of computer hardware [67]. Working with large data requires great effort for analysis by the experts. Thus, methods exist that reduce the acquired data to lower dimensions in order to make it more accessible to the user. A common example is the application of the *Root Mean Squared Distance* (RMSD) of an atom to an individual reference position. The RMSD of all atoms can be averaged to obtain a single scalar value that represents the spatial similarity of an assembly of atoms to a reference assembly. Also popular is the *Radius of Gyration* (RG), which is the average distance of atoms to the molecule's center of mass. However, RMSD or RG treat atoms individually (i.e., the molecule is processed as a cloud of unrelated points) and therefore do not capture spatial relations within the molecule. The computation of such magnitudes can be time consuming, depending on the data size. However, for an application, duration of precomputations and display of the computed data is a key factor for an efficient expert's work flow. Especially, when parameters of data evaluation are dynamic, time consuming recomputations for the visualization are not desired.

This work introduces a new perspective on changes in molecular structure and tackles the above mentioned issues. Experts from the pharmaceutical domain, who tested our approach, stated that it reveals additional insights and serves as a glue between measures

like RMSD or RG. We employ an analytic data acquisition method and an appropriate visualization of the obtained data. The results have been implemented in a stand-alone application that allows the user to investigate MD simulation data with our approach. A surface atom extraction algorithm builds on previous work by Totrov and Abagyan [220] and we further incorporate the idea to form virtual surface layers, which is inspired by the work by Karampudi *et al.* [94]. We visualize the layer information in a 2D-map that provides an overview of the complete time span of a MD simulation or we display the information directly in a 3D-representation of the individual time-steps. The user is able to modify and filter the acquired data for further analysis, by employing a flexible filter syntax. This filtering supports the search for specific features. The main contributions of this paper are:

- A feature extraction approach, called *Residue Surface Proximity*, applied to trajectory data from Molecular Dynamics simulations.
- An interactive, flexible and extensible filtering framework for the analysis of the extracted data.
- An implementation of our approach as an open-source stand-alone application, which is based on modern OpenGL and the Qt-Framework [216]. The application allows for convenient data and visualization export and has been evaluated in a pharmaceutical workflow.

8.2 BACKGROUND

This section shortly describes the background of the chemical domain, covering proteins and the analysis of trajectory data, as well as a common molecular surface representation that is crucial for our contribution.

PROTEINS The smallest building blocks of a molecule that we consider in this work are atoms. Certain groups of atoms of different elements form the so called *amino acids*, where instances are called *residues*. Chains of these amino acids (*primary sequence)* form macromolecules, known as *proteins*. In a dynamic folding process, the primary sequence folds and settles down into a state of energetic equilibrium. The resulting spatial structure of the sequence is also referred to as its *conformation*. Depending on the conditions under which the folding occurs, different conformations are possible for the same primary sequence, which is crucial for its biological activity. Atoms of residues that are on the surface play a significant role in determining the folding behavior and consequently the function of a protein [143]. The residues that form the core of the protein affect the stability of

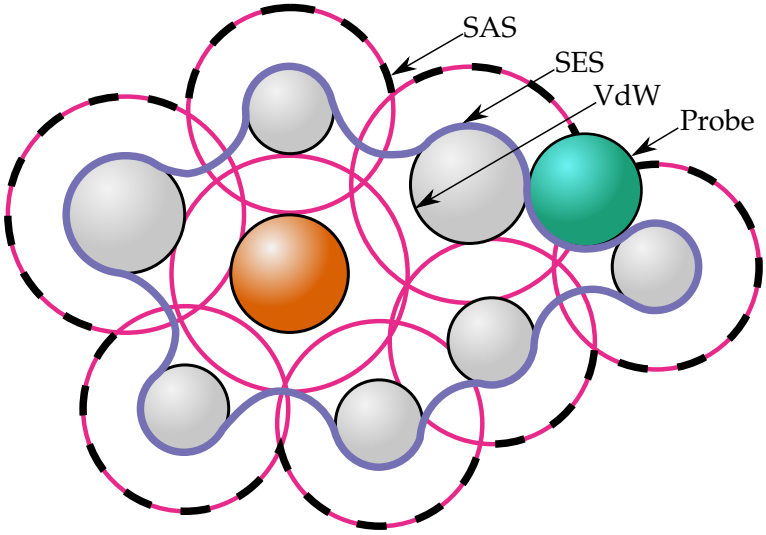


Figure 8.2: Definitions of molecular surface. VdW-spheres (gray). Probe sphere (green). Extended hull (magenta). SAS (dashed hull). SES (purple). Internal atom (orange).

the protein. Usually, a stable protein core is the consequence of a tightly packed combination of hydrophobic amino acids [156]. Surface residues with charge have also been found to influence protein stability [209]. In nature, a vast amount of different proteins and conformations can be found. Depending on their respective bioactivity, they can be utilized in medical treatment. In such a case, it is desired to synthesize these proteins, in order to make them available in sufficient quantities. Pharmacists study these potential sequences, aiming to understand their folding behavior. This is necessary to find the optimal folding conditions that result in the desired conformation, with the desired bioactivity. The obtained knowledge enables synthesization of the bio-active proteins. Identification and tracking of the behavior, which involves protein-protein interactions, is a task of significance in elucidation of mechanisms that contribute to biological function. This has been done right from the primary sequence level [90]. Such studies are commonly conducted by employing MD simulations, which output trajectory data. The trajectory data contains the movements of each atom throughout a simulated folding process under predefined conditions. The course of understanding a protein's folding behavior is what we aim to support with the presented work.

SOLVENT ACCESSIBLE SURFACE The surface of a molecule is a relevant factor of its bioactivity. Therefore, we shortly describe a common and well accepted formulation for the surface of a molecule. A basic representation for atoms is to draw them as spheres with their respective *Van der Waals* (VdW) radius (see Figure 8.1, top, second from the right). We therefore refer to atoms as spheres throughout

this work. The *Solvent Accessible Surface* (SAS) is a representation derived from the VdW-spheres. For each atom, an extended hull is created by adding a probe radius to the VdW radius. The union of these extended hulls yields the SAS. A structure that is no larger than a probe sphere can access (i.e., get in contact with) the atoms that are part of the SAS. A 2D-illustration is given in Figure 8.2, where the SAS is displayed as a dashed curve. The orange atom can not be reached from the outside through the extended hulls and we call it an *internal* atom. The gray atoms that are accessible to the probe are labeled *external*. The SAS representation has first been described by Lee and Richards [129]. An efficient algorithm that yields the external atoms for a given probe radius as a by-product has been proposed by Totrov and Abagyan [220] and is called *Contour-Buildup Algorithm* (CBA).

8.3 RELATED WORK

The acquisition and visualization of properties of molecular data is a field of extensive research. Here we cover existing work that handles the extraction of molecular surface atoms, as this is an important aspect of our software prototype. Then, we shortly outline publications that describe the work with surface or surface-derived data.

SURFACE EXTRACTION Since the original publication about the SAS in 1971 [129] and the extension to the SES [179], several methods have been proposed to calculate the surface of a molecule in an efficient manner. *Voxel-based* algorithms place a molecule inside a 3D-grid and determine surface atoms based on cells that are occupied by atoms. The algorithm by Lee *et al.* [130] does not yield an exact result, since it depends on empirically derived parameters that may vary for different input molecules. A *sample-based* algorithm has been proposed by Byungjoo *et al.* [98]. They create samples on spheres that represent atoms. Samples that are inside the spheres of neighboring atoms are discarded. Surviving samples refer to surface atoms. The accuracy of this approach depends on the number of samples. An approach that disassociates from the common surface definitions is introduced by Karampudi and Bahadur [94]. They spawn cylinders along the x, y, z directions of the 3D-space for each atom center. Periphery atoms within these cylinders are potential surface atoms. The granularity of the surface representation can be controlled by the cylinder radius. Earlier, Connolly [33] has proposed an analytical algorithm to compute the SES area of a molecule. Following their definition, an exact algorithm to extract the actual SES is described by Totrov and Abagyan [220]. Their definition of the CBA yields an exact result with respect to the SAS definition and. The CBA can be parallelized [142] and has also been brought to the GPU by Krone *et al.* [114]. An alternative algorithm by Sanner *et al.* [196] obtains the

reduced surface of a molecule, which also contains the information that is necessary to build the SAS. A thorough list of algorithms that deal with the computation and construction of molecular surfaces can be found in the report by Kozlíková *et al.* [105] and in the work by Daenda *et al.* [41]. Because of the correctness of the CBA, we use the approach by Totrov and Abagyan as the basis for the implementation in our application.

STRUCTURE VISUALIZATION Several approaches that improve the acquisition and visualization of dynamic molecular surfaces exist [113, 141, 142]. However, to understand the behavior of biomolecules, the necessity arises to visualize structural changes. Comparing complex 3D structures is a difficult task and may be solved by switching to a more simple representation. Recent work by Kocincová *et al.* [104], for instance, tackles this task by bridging between 1D amino acid sequences and their 3D-representations. This is useful to compare time-steps of a MD simulation of the same sequence, but fails to capture global correlations between the changes of individual elements of the sequence. An approach by Malzahn *et al.* [145] unfolds proteins to allow the view on parts of the molecule that are usually buried in tunnels. Krone *et al.* [115] compute 2D projections of molecular surfaces and use the time-steps of a MD simulation as the third dimension in a space-time cube. A proper visualization of the cube allows to quickly find surface- and time-dependent features. A disadvantageous property is that information from inside the molecule is disregarded.

This paper contrasts these approaches, by attempting to capture structural changes on the surface and on the inside of a molecule over time and visualizing the results in a single 2D map. We stick to a surface-based formulation, but take the whole molecule into account by defining virtual surface layers inside the molecule. Such advances can also be found in the chemical domain, e.g., Saunders *et al.* [197] infer information from the relative surface participation of individual residues. Further, Xu *et al.* [232] compute the distance of an atom to the SAS. This magnitude reflects how deep an atom is buried inside a molecule and it is shown to be an interesting, additional feature, besides RMSD or RG. A similar, but very coarse approach was proposed by Karampudi and Bahadur [94]. Instead of obtaining a precise distance of each atom to the surface, they strictly define three layers, independent of the size of the molecule. Despite of the precision, both methods could be applied to track movements of atoms or residues inside the molecule, independent of absolute spatial movements. The depth of an atom has also been used in the context of cavity prediction by Tan *et al.* [213]. We argue that Karampudi's approach lacks the incorporation of the SAS, while Xu's methods misses the atomistic structure and order within a molecule. Hence, the work at hand is influenced by both these methods and transfers their ideas to MD

trajectory data. By constructing virtual inner layers, we basically introduce a way to capture the entire structure of a molecule.

8.4 REQUIREMENTS

Here, we describe the requirements from the point-of-view of a pharmaceutical expert and derive goals in order to support the scientific work flow. This work was carried out in the context of research of μ-conotoxins, which are taken from the venom of the marine cone snail [1]. For example, these small proteins (10-50 amino acids) are of interest in the area of drug design (e.g., for the fabrication of pain killers). Studying their folding behavior and applications is an active field of research [218, 219]. MD simulations may contain several hundreds of thousands of time-steps, each representing the conformation of the molecule in a folding process. Thus, the obtained data has to be broken down and simplified in order to make it seizable and comprehensive in practice. We divide the workflow into four main steps:

1. *Acquisition* Novel data is extracted from input trajectory data.
2. *Presentation* The extracted data is presented to the user in a way that allows immediate insights into the data.
3. *Analysis* The extracted data is filtered in order to reveal individual features and to allow in-depth interpretation.
4. *Distribution* The extracted data is prepared for publication as a basis for discussion with the expert community.

In the following, we describe the above steps in more detail with respect to our research contribution.

ACQUISITION RMSD or RG measures help identifying sections of the MD data, in which a molecule's conformation is similar to a reference conformation and where larger deviations occur. Experts can draw conclusions from these values to further analyze certain time-steps. The drawback is that each atom is treated individually (i.e., the molecule is treated as a cloud of unrelated points). The spatial relation of atoms is, however, very important for further interpretation. In general, the surface of a protein determines its interaction behavior with other molecules. In conotoxins, *Structure Activity Relationship* studies indicate that the residues that are important for the toxin's binding and interaction with its ion channel target are found on the surface of the molecule [1], while the core determines its stability [156]. This last statement yields the basis for the approach proposed in this work that introduces the concept of *Residue Surface Proximity* (RSP). We design the RSP as a scalar measure for atoms and residues that reflects the proximity of an atom or residue to the molecular surface. Doing so, the relative ordering of atoms and residues

within the molecule will be taken into account. Details on the RSP can be found in Section 8.5.1.

PRESENTATION With a large amount of data at hand, it is necessary to present the data in a clear manner. A good representation aids the user in finding certain features within the data. In our case, the experts are interested in the change of RSP per residue. During a folding process, some residues are expected to be more mobile and also correlations between pairs of residues are expected. Thus, we want to give the user an overview in which the RSP value of residues is easily perceivable and comparable to other residues. We additionally map the derived RSP data to the original 3D-structure to create a visual link between them. The 3D-visualization helps the user to mentally integrate the derived RSP data into the original MD simulations data (see Section 8.5.2).

ANALYSIS When working with new data, it is not always clear what to focus on or what to search for. In such a case, it is beneficial for the user to quickly filter the data in order to manually extract certain features. Predefined filters may not always cover the needs for individual data sets and exporting the data for filtering in external programs delays the workflow. Hence, we want to provide the user with a flexible filtering framework that can be tailored inside the application. The framework allows for an immediate visual feedback through the visual data representation, as described in Section 8.5.3.

DISTRIBUTION After the analysis, the user might want to share the extracted data with colleagues and the research community. Hence, it is imperative to provide convenient modes to export the visual representation of the RSP in publication quality (see Section 8.5.4). Exporting the raw data does also allow further processing by external software, if required. The four workflow steps described above yield the main features that we want to cover with our application. In the next section we describe our approaches to address the listed requirements.

8.5 APPLICATION CONCEPT

This section describes the approaches with which we address the requirements established in Section 8.4. The user workflow and the associated feature flow of our application is depicted in Figure 8.3. The concrete steps are described next.

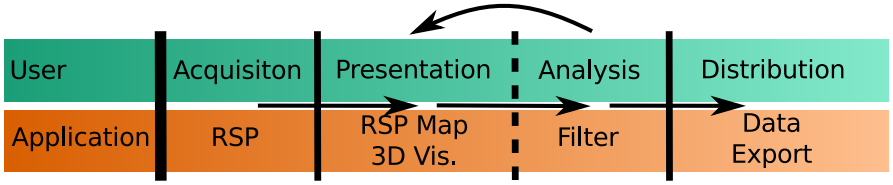


Figure 8.3: User workflow and the application feature flow. The Presentation and Analysis steps form a repetitive process, where filtered data is visualized, motivating further analysis and filtering.

8.5.1 *Acquisition: Residue Surface Proximity*

We aim to provide an algorithm that is capable of capturing the arrangement of atoms. The relation we are looking for is the ordering of atoms with respect to their proximity to the molecule surface. The concept is delineated in Figure 8.1 (top, second from the left), where atoms are represented by their extended hull. The outer layer (blue) is made up of the atoms that are part of the SAS. The next (cyan) layer can be computed by removing the first layer and extracting the SAS of the remaining atoms. This process is repeated until no more atoms remain (yellow core layer). Let $\mathcal{A}$ be the set of all atoms of a molecule. Then, atom $i \in \mathcal{A}$ is labeled with a layer number $\mathcal{L}_i \in \{0, N-1\}$, where N is the total number of layers. Such a layering approach has initially been used by Karampudi and Bahadur [94]. Their method is based on their own definition of the surface of a molecule and restricts the number of layers to three distinct classes for both, atoms and residues. We argue that the SAS is a more suitable representation to create layers, because it is an accepted method to determine the accessibility of atoms. Further, we allow a continuous layer spectrum for residues. Let o be a residue consisting of atoms $\mathcal{A}_o$, then we compute the averaged residue layer $\mathcal{L}_o$ as the arithmetic mean of the layers of atoms $\mathcal{A}_o$. In this way, we obtain a scalar value $\mathcal{L}_o \in [0, N-1] \in \mathbb{R}$ for each residue and each time-step. We see this approach as a trade off between Xu *et al.* [232] and Karampudi and Bahadur [94]. While the algorithm by Xu *et al.* yields a precise distance value of each atom to the surface, it neglects the SAS property that we maintain through the layer approach. In contrast to Karampudi and Bahadur, the averaging of atom layers allows us to faithfully process small proteins. Their approach uses the minimum atom layer to define the residue layer. For small molecules, most residues are likely to have at least one atom in the outermost (lowest) layer, which would make the representation less discriminative.

As the layer membership is based on the SAS definition, it also depends on the applied probe radius. An increased probe radius leads to less accessible atoms per layer, increasing the maximum number of layers. This way the user can control the granularity of the assigned layers. A standard radius of 1.4 Å represents the size of a solvent

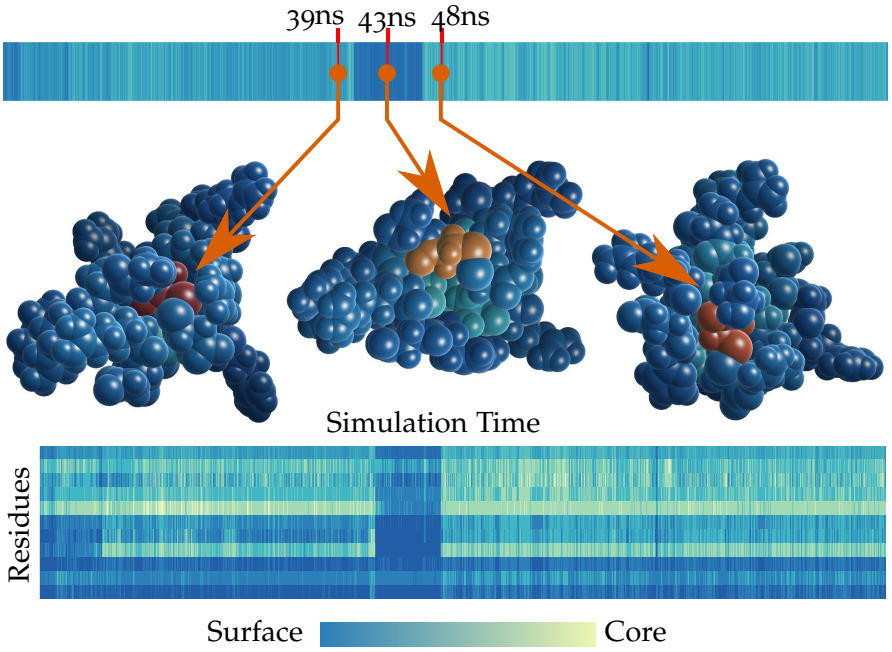


Figure 8.4: Link between RSP-map (top, showing one residue) and 3D-view (center). The highlighted residue (inverted colors) is part of the core until 39ns (center, left). It then emerges for a short period (center) before going back to the core after 48ns (center, right). Information per atom can also be obtained in a detail view (bottom). A tracker (top, red bar) selects a time-step from the RSP-map.

water molecule, which is usually applied for the SAS calculation. In our experiments a radius of 2.8 Å was observed to yield better results and has been used for the figures and experiments in this work, if not stated otherwise. It was chosen based on testing different radii with the trajectories at hand. The optimal radius may vary in dependence of the data set. We describe the SAS layer extraction in detail in Section 8.6.1.

8.5.2 *Presentation: RSP-map and 3D-visualization*

The data presentation in our application is divided in two parts: A RSP-map (based on the heat-map concept) and a 3D-visualization of the molecule with RSP data mapped to it. The RSP-map contains a row for each residue and each row is made up of the time-steps of the MD data. An example RSP-map is depicted in Figure 8.4. To provide an integration with the 3D-structure, a time-step tracker can be used to select time-steps that are then rendered in a 3D-view. The residues (or atoms) of a molecule are then colored accordingly to the RSP-map. The example in Figure 8.4 shows that the RSP-map is able to reflect conformational changes of the molecule. Having the 3D-visualization of the molecule helps the experts to quickly make

judgments from a certain time-step of the simulation. The integration within our tool negates the need to have the trajectory playing in another external application to follow the progress of the simulation along with the RSP-map perspective. One of the main components of MD simulation trajectory analysis is to see changes in the molecule at different time-steps in the simulation. A usual way of doing this is to collect snapshots from interesting time-steps from the trajectory and to visually compare them. In our tool, from looking at the RSP-map, experts can identify interesting time-steps, e.g., points where the layer membership of a residue changes. Given this, the tool allows to visualize these multiple time-steps in side-by-side 3D views for comparison. The figures shown in this work use a blue to yellow color map, where blue represents the molecule surface and yellow the core. In our application, the colors can be customized.

8.5.3 *Analysis: Filter Expressions*

Manipulation and filtering of data is a crucial step when searching for certain features, or preparing the data for further tasks. We provide a flexible and extensible basis, which allows the user to directly process the data through a filtering interface inside the application. The user can submit C-like expressions through a command-line interface for filtering. Currently, two types of expressions are supported, distinguished by their return type:

- *Boolean* Expressions return `true` or `false`. They hide an item or items from the data if `true`. (E.g., *AtomElement* == *H* hides all hydrogen atoms from the RSP-map and 3D-visualization)
- *Numerical* Expressions return a numerical value. They set the displayed RSP values with the returned value. (E.g., the difference of RSP for each time-step (frame) and time-step 600 is returned by *ResidueLayer(frame) - ResidueLayer(600)*.)

Hiding certain amino acids may be useful when working with larger molecules. In that case, the RSP-map can be reduced to the set of residues of interest. Multiple expressions can be chained together to form more complex expressions, which accept standard arithmetic, relational and logical operators. Basically, the input to an expression is either a variable or a numerical value. Variables are defined inside our application and return the associated value. The previously mentioned variable `frame`, for example, returns the time-step of each data element that the filter is applied to. Chained expressions might, however, become too complex for the user to express. Such expressions can be wrapped in functions that are implemented in the source code of the application and are then available through the filtering interface. For instance, Figure 8.5 (top) shows a raw and a filtered version of a RSP-map. The modified version (center) makes it much easier to

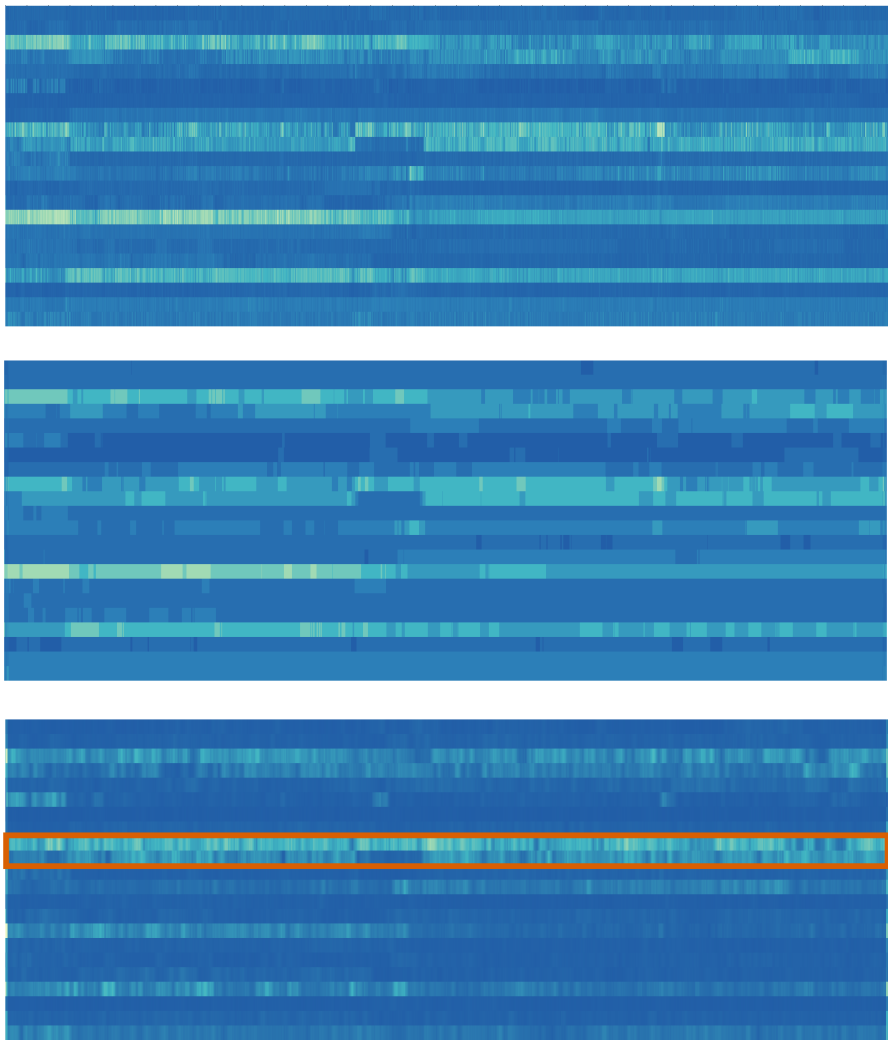


Figure 8.5: The raw RSP-map data (top) has been modified by a filter expression (center) to highlight significant changes in the representation. The following expression was used: *Quantize(Sum(-100, 100, ResidueLayer(Frame + x), x) / 201, 0.5)*. This command smooths the residue layer in a window of 201 time-steps and rounds them to the next multiple of 0.5. Another filter has been applied (bottom) to display the smoothed layer difference of adjacent time steps.

detect significant changes in the RSP values. It might also be feasible to compute the layer difference of adjacent time-steps. Figure 8.5 (bottom) reveals that the marked (orange box) residues change their layer membership frequently throughout the simulation. For further details and more possible filter expressions, we refer to the application prototype and source code that we supply in the supplementary files. Details on the processing and extensibility of the filter expressions can be found in Section 8.6.3.

8.5.4 *Distribution: Data Export*

Data that has been generated by our application can be exported in several ways. RSP values can be stored as *Comma Separated Values* (.csv) file, which is suitable for loading into external software for further processing. A binary format *Binary Atom Layer Data* (.bald) is also available. It reduces the memory consumption and is faster to load, and should be preferred to store data that will later be loaded again by our application. Additionally, it is possible to output the RSP data in an *.html* file, which allows plotting the RSP-map in an interactive representation in the browser (requires the Plotly API [170]). Further, the RSP-map as well as the 3D-visualization can be written to disk as images with user defined resolution to provide high quality renderings.

8.6 IMPLEMENTATION

This section describes the extraction of surface atoms, the visualization and the implementation of filter expressions in more detail.

8.6.1 *Surface Atom Extraction*

In order to compute the RSP as described in Section 8.5.1, we extract the atoms that are part of the SAS. The basis of our implementation is the CBA [220]. While the CBA can be employed to find the SES, we only need to compute a part of the algorithm's possible output, which yields the external atoms. This is similar to the work by Zhang and Shi [235]. As described in Section 8.2, the SAS is defined with respect to the radius of a probe sphere. Atoms $i \in \mathcal{A}$ of a molecule are treated as a sphere cloud, where each sphere radius is given by the respective VdW radius, extended by the probe radius (p_r): $r_i = r_{i,VdW} + p_r$. Each extended sphere is then tested for overlapping with neighboring spheres. If the surface of a sphere i resides completely within neighboring spheres, that sphere is not part of the SAS. The algorithm is applied to each sphere $i \in \mathcal{A}$ individually and classifies i as internal or external. The process is subdivided into four steps:

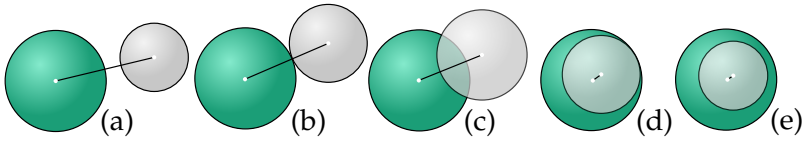


Figure 8.6: Neighbor configurations for two spheres.

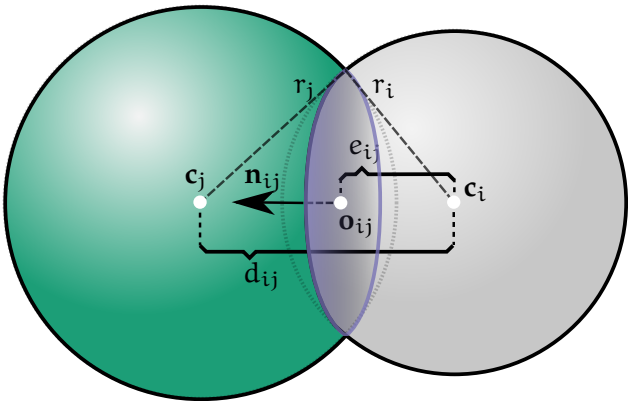


Figure 8.7: Illustration of the intersection distance e_{ij} in dependence of the cutting plane $(\mathbf{x} - \mathbf{o}_{ij}) \cdot \mathbf{n}_{ij} = 0$.

S1 Build cutting face list.

S2 Filter cutting face list.

S3 Build end-points.

S4 Filter end-points.

We closely follow the original algorithm, but utilize a geometric property to speed up the computation of S2 and S3. In order to derive that geometric property, we provide a detailed recap of the required parts of the CBA.

S1 Let $\mathcal{N}_i \subset \mathcal{A}$ be the set of neighbors of sphere i. Then, $j \in \mathcal{N}_i$ if and only if $d_{ij} < r_i + r_j$ and $d_{ij} > |r_i - r_j|$, where d_{ij} is the Euclidean distance between the centers of spheres i, j. Hence, $\mathcal{N}_i$ contains all spheres that intersect with i, as depicted in Figure 8.6(c). A cutting face is computed for each intersection and represented in Hessian Normal Form: $(\mathbf{x} - \mathbf{o}_{ij}) \cdot \mathbf{n}_{ij} = 0$, where $\mathbf{n}_{ij} = \frac{\mathbf{c}_j - \mathbf{c}_i}{|\mathbf{c}_j - \mathbf{c}_i|}$ is the normalized vector between the two sphere centers $\mathbf{c}$. The support vector for the plane is given by $\mathbf{o}_{ij} = \mathbf{c}_i + e_{ij} \cdot \mathbf{n}_{ij}$. Here, e_{ij} is the *intersection distance* of two spheres (w.r.t. $\mathbf{c}_i$, see Figure 8.7):

$$e_{ij} = \frac{d_{ij}^2 + r_i^2 - r_j^2}{2 \cdot d_{ij}} \tag{8.1}$$

S2 The next step is to filter the set of cutting faces, as there are cases where a cutting face can be discarded or the classification of sphere

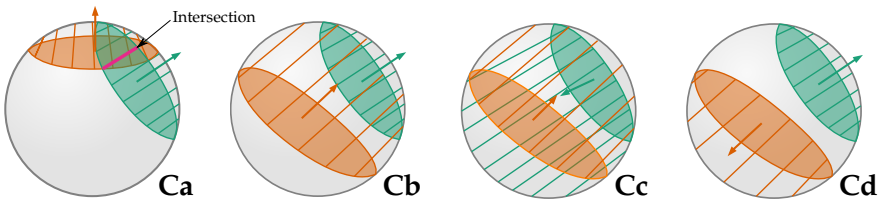


Figure 8.8: Cutting face configurations. The striped area is cut from the gray sphere by the respective cutting face.

i can already be terminated. For this, we require the normalized intersection distance (i.e., on the unit sphere), given by $\hat{e}_{ij} = \frac{e_{ij}}{r_i}$. The value $\hat{e}_{ij}$ resides within $(-1, 1)$. The excluded bounds 1 and -1 are depicted by Figure 8.6(b) and (d), respectively (where i is shown as the gray and j as the green sphere).

Let $A, B \in \mathcal{N}_i, A \neq B$ be two neighboring spheres of i. In the following figures, the observed sphere i is shown in gray, A in green and B in orange. Four general configurations are possible as illustrated in Figure 8.8:

Ca The cutting planes intersect within the observed sphere i.

Cb The green face is cut away by the orange face and is discarded (this covers an analogue case, where orange is discarded).

Cc The observed sphere i can be classified as internal, since it is completely cut away by both faces.

Cd Both faces survive, but no intersection is found inside the observed sphere.

Instead of explicitly calculating the intersections between each pair of cutting faces, and checking for the given configurations as suggested by the original algorithm, we speed up the computation, as follows: The configurations in Figure 8.8 can be described by the orientations $\mathbf{n}_{iA}, \mathbf{n}_{iB}$ and the normalized intersection distances $\hat{e}_{iA}, \hat{e}_{iB}$. The relative orientation of A and B is depicted by the included angle α and the resulting dot product: $\cos(\alpha) = \langle \mathbf{n}_{iA}, \mathbf{n}_{iB} \rangle$. Thus, the cutting face configurations can be described by three degrees of freedom: $\{\alpha, \hat{e}_{iA}, \hat{e}_{iB}\}$. From that we derive a formula, which immediately determines one of the cutting face configurations. As illustrated in Figure 8.9 (left), we can obtain the half of the normalized intersection range

$$\hat{h}_{i,AB} = \hat{r}_{iA} \cdot \sin(\alpha) \tag{8.2}$$

where,

$$\hat{r}_{iA} = \sqrt{1 - \hat{e}_{iA}^2} \tag{8.3}$$

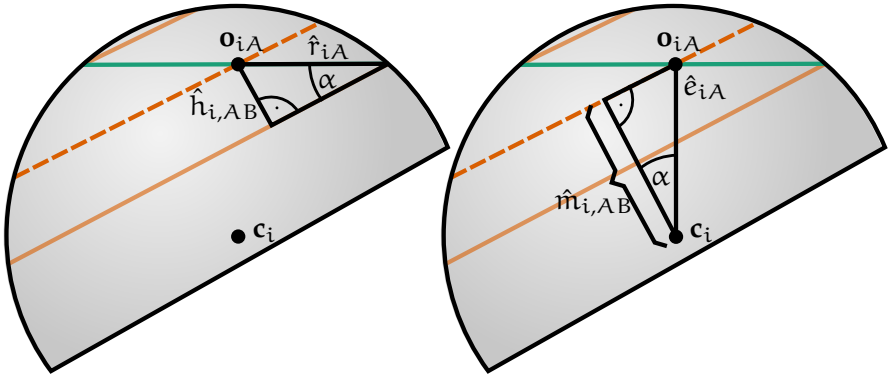


Figure 8.9: Determination of the variables $\hat{h}_{i,AB}$ and $\hat{m}_{i,AB}$, where A (green) and B (orange) cut i (gray sphere). The orange cutting planes represent the minimal and maximal cutting distance for B. The dashed orange lines depict the center of the range.

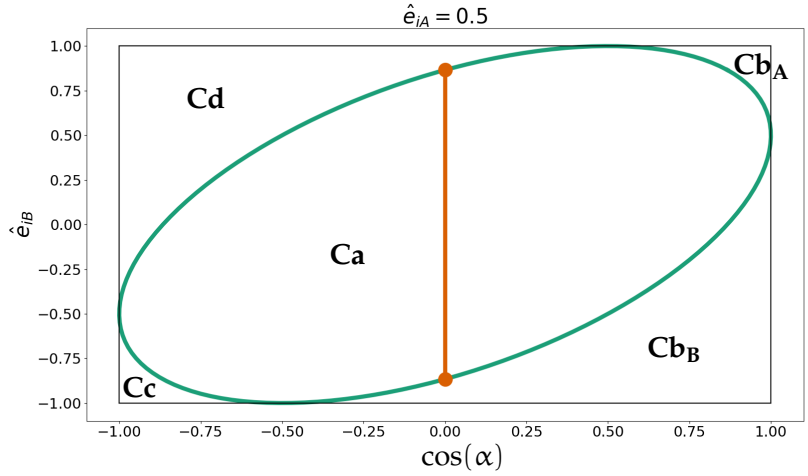


Figure 8.10: Plot for a fixed intersection depth $\hat{e}_{iA} = 0.5$. The labels refer to the cutting face configurations in Figure 8.8, where Cb_A, Cb_B cover the case *Cb* (A or B survives, respectively). The orange line depicts the possible values for $\hat{e}_{iB}$, if $\mathbf{n}_A$ and $\mathbf{n}_B$ are orthogonal.

is the radius of the cutting plane of A with respect to i, mapped to the unit sphere. Further, we compute the normalized shift of the intersection range as in Figure 8.9 (right):

$$\hat{m}_{i,AB} = \hat{e}_{iA} \cdot \cos(\alpha). \tag{8.4}$$

Finally, if spheres A and B intersect with i for a given intersection distance $\hat{e}_{iA}$, we obtain the minimal ($\hat{e}_{iB}^{min}$) and maximal ($\hat{e}_{iB}^{max}$) intersection distances for B, such that A and B intersect within i:

$$\hat{e}_{iB}^{min} = \hat{m}_{i,AB} - \hat{h}_{i,AB}, \qquad \hat{e}_{iB}^{max} = \hat{m}_{i,AB} + \hat{h}_{i,AB} \tag{8.5}$$

In Figure 8.10 we plot the respective minimum and maximum values for a fixed $\hat{e}_{iA}$ with varying $\cos(\alpha)$. The resulting graph has the shape of an ellipse, which subdivides the parameter range $(-1, 1)$ into

```
IF ê_iB <= ê_iB^min THEN:
  IF cos(α) >= −ê_iA THEN:
    case Cb_B;
  ELSE:
    case Cc;
ELSE IF ê_iB >= ê_iB^max THEN:
  IF cos(α) > ê_iA THEN:
    case Cb_A;
  ELSE:
    case Cd;
ELSE:
  case Ca;
```

Listing 8.1: The five possible configurations of two cutting faces

five segments. From these segments we can immediately derive the configurations from Figure 8.8, by looking up the point $(\cos(\alpha), \hat{e}_{iB})$ within the graph:

Ca Inside the ellipse

Cb Bottom right (B survives), Top right (A survives)

Cc Bottom left

Cd Top left

The segments can be quickly determined as in Listing 8.1. After optimizing the cutting faces for i in the above described manner, only configurations *Ca* and *Cd* remain. If only *Cd* occurs, i can be classified as external, otherwise we have to further process the intersections of *Ca*, which will be covered by S3 and S4.

S3 For each pair A, B that results in configuration *Ca* we compute the intersection line of both cutting faces. The previously computed values $\hat{e}_{iB}^{min}$, $\hat{e}_{iB}^{max}$ and $\hat{e}_{iB}$ allow us to do this efficiently.

$$\mathbf{p}_0 = \mathbf{k} + \mathbf{v}_0 \cdot l, \qquad \mathbf{p}_1 = \mathbf{k} - \mathbf{v}_0 \cdot l \tag{8.6}$$

where $\mathbf{p}_{0,1}$ are the intersection points, $\mathbf{k}$ is the closest point on the intersection line of A, B to the sphere center $\mathbf{c}_i$, $\mathbf{v}_0 = \mathbf{n}_{iB} \times \mathbf{n}_{iA}$ is the direction of the intersection line and $l^2 = r_A^2 - q^2$ is the distance of the intersection points to $\mathbf{k}$, with $r_A^2 = r_i^2 - e_{iA}^2$. Here,

$$q = 2 \cdot r_A \cdot \left(0.5 - \frac{\hat{e}_{iB} - \hat{e}_{iB}^{min}}{\hat{e}_{iB}^{max} - \hat{e}_{iB}^{min}}\right) \tag{8.7}$$

is the distance of the intersection line to the $\mathbf{c}_i$. With that, we obtain

$$\mathbf{k} = \mathbf{o}_{iA} + \mathbf{v}_1 \cdot q \tag{8.8}$$

where $\mathbf{v}_1 = \mathbf{n}_{iA} \times \mathbf{v}_0$. See Figure 8.11 for an illustration of the computation. The intersection points for all pairs A, B, $A \neq B$ in configuration *Ca* are maintained in the set $\mathcal{P}_i$.

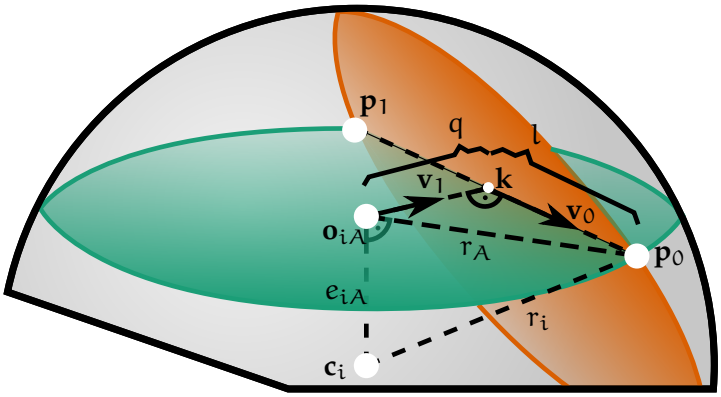


Figure 8.11: Determination of the intersection points $\mathbf{p}_{0,1}$ on sphere i in dependence of the intersection planes of spheres A (green) and B (orange).

S4 The last step is equal to the original algorithm. All intersection points in $\mathcal{P}_i$ are tested against all cutting planes of configuration *Ca*. If an intersection point $\mathbf{p} \in \mathcal{P}$ lies in the positive half-space (i.e., in the portion that gets cut away) of a cutting plane, it is removed from $\mathcal{P}$. Finally, if $\mathcal{P} = \emptyset$, then i is an internal atom, otherwise it is external.

To compute the layer membership for all atoms of the molecule, multiple iterations of the following algorithm are employed, where $k = 0$ is the index of the first iteration and $\mathcal{A}_r \leftarrow \mathcal{A}$ is the set of remaining atoms:

1. The steps *S1*-*S4* are executed for each atom $i \in \mathcal{A}_r$ to obtain the set of external atoms $\mathcal{A}_s \subseteq \mathcal{A}_r$.
2. The atoms $j \in \mathcal{A}_s$ are assigned the layer membership $\mathcal{L}_j = k$.
3. The external atoms are removed from further processing and the iteration index is increased: $\mathcal{A}_r \leftarrow \mathcal{A}_r \setminus \mathcal{A}_s$, $k \leftarrow k + 1$.
4. The iteration stops when $\mathcal{A}_r = \emptyset$.

In our implementation, the above process is executed in parallel for individual time-steps of the MD simulation data, utilizing full multi-processor CPU power.

VALIDATION Theoretically, the CBA yields an exact result with respect to the SAS definition. As a proof, we validate our implementation with a sampling algorithm by covering the extended sphere of an atom with sample points. The sample points inherit the classification of the associated atom (internal or external). Then we test each sample against the extended hulls of all other atoms. If all internal samples are inside of at least one extended hull, then all internal atoms are classified correctly. If at least one sample of an external atom is not included in any extended hull, then the associated atom is correctly classified as external. We applied this test to several data

Table 8.1: Number of atoms in the outermost SAS layer, for 1.4Å and 2.8Å probe radii. $|\mathcal{A}|$ is the total number of atoms of the input molecule.

PDB ID	$\lvert\mathcal{A}\rvert$	# 1.4Å	# 2.8Å
1A19	1438	951	548
2PLT	804	498	358
1AF6	10050	6153	3001

sets and simulation time-steps to ensure the correctness of our implementation. Table 8.1 shows the number of atoms in the outermost layer for three sample molecules from the Protein Data Bank [12].

8.6.2 *3D Visualization*

The integration of the RSP value and the actual 3D-structure is an important part of the analysis process. To achieve this, the user can choose from a set of standard techniques to render the molecule in 3D, with the RSP values mapped to the atoms or residues. The techniques are *Space-fill, Ball & Stick, Licorice* and *Backbone*. While the backbone gives an overview of the structure, its pattern is usually highlighted by a *secondary structure* visualization, whose basic elements are α-helices and β-sheets [105, 180]. A recent algorithm for the efficient display of such has been proposed by Hermosilla *et al.* [72], who extend to the idea by Krone *et al.* [112]. However, these approaches require a classification per atom, which determines the secondary structure element that it belongs to [48]. This classification may change with every time-step in a trajectory. Thus, we decided to combine the backbone representation with a light-weight helix approximation. The approximation allows us to accentuate helical sections of the backbone on the fly.

We draw the backbone as a tube, based on a B-spline. Let $\mathbf{b}_i \in \mathcal{B}$ be the support points of the backbone spline $\mathcal{B}$ (i.e., the C_α atoms). Then we compute a weight w_i for each support point, which depends on the convolutedness of a spline section around $\mathbf{b}_i$ with

$$\mathbf{h}_i = \sum_{j=i-k}^{i+k-1} \frac{\mathbf{b}_{j+1} - \mathbf{b}_j}{|\mathbf{b}_{j+1} - \mathbf{b}_j|}, \tag{8.9}$$

where $\mathbf{h}_i$ is the convolution vector at $\mathbf{b}_i$. Note that if the spline from $\mathbf{b}_{i-k}$ to $\mathbf{b}_{i+k}$ is a straight line, $|\mathbf{h}_i| = 2k$ holds. For any convoluted spline section, we obtain $|\mathbf{h}_i| < 2k$. The relation $w_i = 1 - \frac{|\mathbf{h}_i|}{2k}$ can then be used to express the convolutedness of the section around $\mathbf{b}_i$. We use w_i to flatten the tubular spline as depicted in Figure 8.12 (right). It can be observed that helical sections are flattened, while straight sections are tubular. In our implementation, we chose $k = 10$. This representation approximates the depiction of α-helices and can

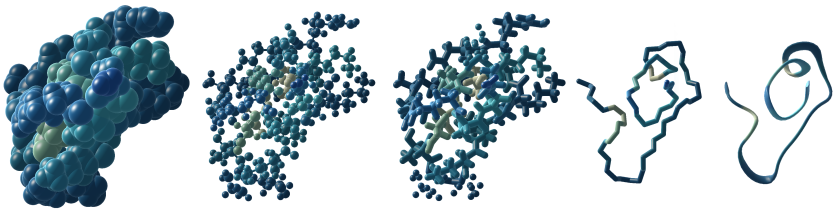


Figure 8.12: 3D representations, from left to right: Space-fill, Ball & Stick, Licorice, Backbone, Helix approximation.

easily be updated on the fly. It is therefore suitable for a cost efficient animation of trajectory data.

8.6.3 *Filter Expressions*

Filter expressions that are submitted by the user are parsed and split into tokens, which are inserted into a tree structure. Five different types of tokens exist (see Figure 8.13, top). After extracting a list of tokens from the filter expression, a processing tree is constructed. Each node in that tree has a type, which defines the number of children it can have. The type also defines a pattern that must be present in the token list to create that node. If a pattern matches the token list, a node is created and the list is divided into sublists (a sublist for each child node). The sublists are then processed in the same manner, until leaf nodes are reached. There are five types of nodes:

- *Bracket Pair* Pattern: Brackets enclosing n tokens. Children: 1.
- *Operator* Pattern: x tokens left and y tokens right. Children: 2.
- *Function* Pattern: Function name, followed by x parameters, divided by commas and enclosed by brackets. Children: x.
- *Variable* Pattern: String token with attached value. Leaf node.
- *Number* Pattern: Single numerical token. Leaf node.

An example tree is given in Figure 8.13 (bottom). The tree is then evaluated, beginning at the leaf nodes, to obtain the final result. The functions and variables used in the filter expression determine whether the filter is called by each atom, each residue, or both. For example the variable `AtomIndex` can only be called by atoms, while `frame` can be called by atoms and residues. If the result is a `boolean` value, it is used to toggle the visibility of the calling data item in the RSP-map and the 3D-visualization. A `numerical` value is used to overwrite the respective RSP value. This basic structure allows to combine the available tokens to complex expressions. Furthermore, the interpretation of function and variable strings is well extensible. Developers are able to define new function names that take a number of parameters and

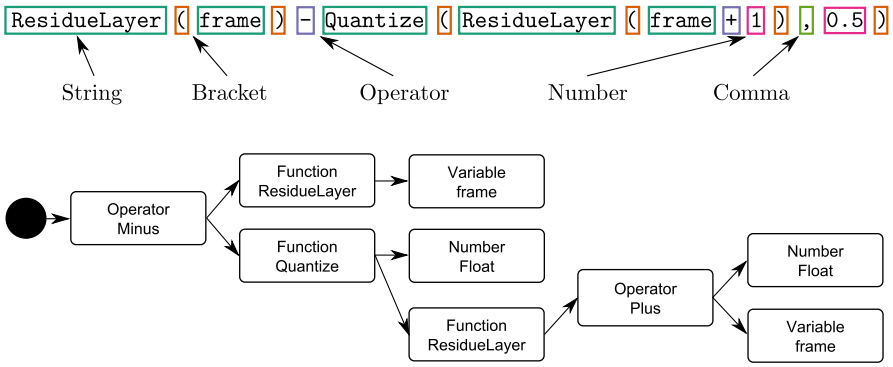


Figure 8.13: A filter expression can contain up to five different token types (top). The tokens are then evaluated and inserted into a tree structure (bottom).

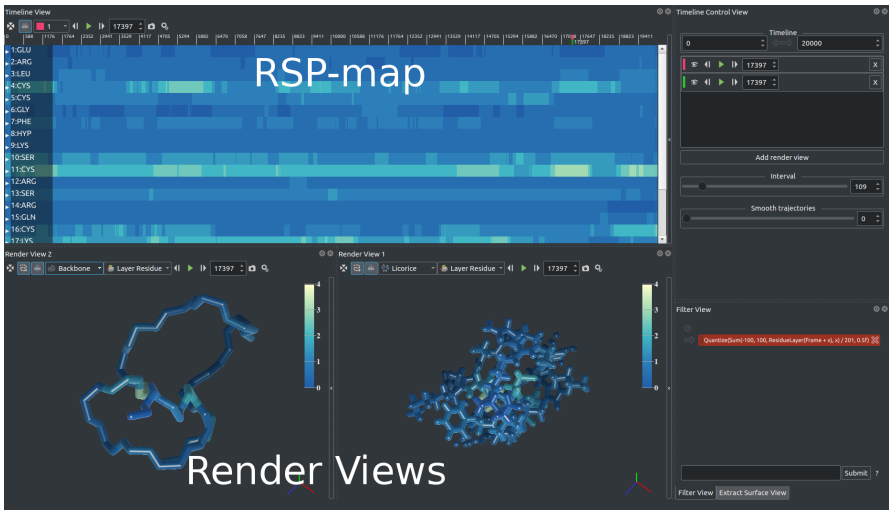


Figure 8.14: Screen-shot of our application with a PIIIA protein loaded.

map these to an actual C++ implementation. In this manner, the filter interface can be augmented with arbitrary functionality.

8.7 RESULTS

In this section, we examine our application from a performance's and the domain experts' point-of-view, focusing on the expert user acceptance, based on the questionnaire by Davis [37]. A screen-shot of the application is shown in Figure 8.14, where the RSP-map is displayed along with two 3D-render-views. The render-views can be used to compare different time-steps or to apply different visualizations or color styles. The filter panel is shown at the bottom right (the applied filter expression is marked in red).

Table 8.2: Timings of our application to compute the RSP for 20k time-steps. Results are shown in seconds for a probe radius of 1.4Å and 2.8Å. The maximum number of extracted atom layers was 3 and 5, respectively.

Protein	# Atoms	s 1.4Å	s 2.8Å
GIIIA	352	41	104
KIIIA	235	25	60
PIIIA	350	39	99
SIIIA	274	31	79
SmIIIA	337	40	98

8.7.1 *Performance*

To assess the performance of our implementation, we computed the RSP values for five MD simulation trajectories of known μ-conotoxins. The trajectories contained 100k time-steps each and were sampled down to 20k steps. This down-sampling was applied, as our domain experts usually work on the lower-resolution data, in order to speed up their workflow. The probe radius for the layer calculation was set to 1.4Å and 2.8Å. The performance tests were executed on a desktop computer environment with a 4-core 4.00 GHz i7-6400 processor, a NVIDIA GTX 1070 GPU and 16GB DDR4 RAM. As depicted in Table 8.2, the probe radius has a significant impact on the computational workload. A larger radius requires the observation of a larger neighborhood per atom. The size of the neighborhood also depends on how compact the molecule is (i.e., the number of layers). However, the timings to precompute the RSP values are in the range of few minutes, which is feasible for an efficient work flow, since the values only have to be calculated once. Note that the time to obtain the MD trajectories is in the range of a couple of hours on consumer hardware, hence, our timings are negligible.

8.7.2 *User Experience*

To validate the expert user acceptance of the software prototype, a survey was conducted with domain experts that used the application during their professional work. The experts' task was to investigate the effect of breaking disulfide bonds on conformational changes of the respective molecule during an MD simulation. To obtain insights into conformational changes, they commonly use magnitudes like RMSD. Now, they used our software prototype in addition to gain more information through the RSP-maps. This includes loading trajectory data, setting up the probe radius for the RSP-map calculation and executing the algorithm. The experts were then able to use the

RSP-map, filter expressions and 3D views of our prototype to locate interesting sections of the MD trajectory for further analysis. More detail about the incorporation of the RSP-maps during the workflow is given in Section 8.7.3. The significance of our study is underlined by the fact that it was conducted within a professional work environment and not in an artificial survey setup. The survey contained the 12 questions proposed by Davis [37]. The questions are separated into two categories. One measures the perceived usefulness and the other the perceived ease of use. Both can be correlated to powerfulness and simplicity. A useful application is powerful as it has all the features needed to perform a task efficiently. While simplicity is not directly ease of use, it does result in ease of use.

Three subjects participated in the survey. The subjects were classified into three categories, namely novice, intermediate and advanced user, based on their previous experience of conducting molecular simulations and using related software. The ratings of Usefulness and Ease of Use after Davis for each user are displayed in Table 8.3. The *novice user* was a pure experimentalist with minimal experience in usage of computational tools related to molecular simulations. Initial problems arose due to lack of knowledge of how simulation files are generated and stored. Once the concept of structure and trajectory files were explained, the application was much easier to handle and the user could independently navigate through the application in an intuitive way. Because of the missing experience with MD simulation data, the Usefulness was rated with 3 out of 5 points. However, the Ease of Use rating was rated 4.6. The *intermediate user* was again an experimentalist but had occasionally used online tools for protein structure predictions. The user also had some knowledge on molecular simulations and rated the Usefulness with a score of 4. The additional experience with online tools supported the Ease of Use, resulting in a rating of 5. The *advanced user* was a computational chemist who was highly accustomed to running molecular simulations, analyzing trajectory data and presenting them to experimentalists to make further biological deductions from the computational findings. Both, overall Usefulness and the Ease of Use, were rated with 5 points.

The experts stated that the RSP approach is a new and intuitive perspective on MD simulations, which allows to extract informations from complex trajectory data more easily. They see potential that our approach serves as a tool to study molecular motion and surface properties, adding to more common approaches. The advanced user found that the tool is a must-have in the analysis pipeline of any computational dynamic simulation of biomolecules. In general, it was stated that one of the biggest advantages that our prototype provides, is that RSP values are displayed as a color coded map. This enables the user to skip the redundant time-consuming process of playing the trajectory multiple times in an animation to observe and

Table 8.3: Ratings of the Usefulness and Ease of Use after Davis [37]. The maximum possible ratings in the questionnaire were 5.

User	Usefulness	Ease of Use
Novice	3	4.6
Intermediate	4	5
Advanced	5	5

interpret the movement or layer membership of individual amino acid residues or individual parts of the protein. The additional display of 3D-visualizations of multiple selected time-steps has also been pointed out as an important aspect for a convenient work-flow. The ability to super-impose these selected time-steps would be an additional improvement. Further benefits that the domain experts see are summarized in the following section.

8.7.3 *Accompanying Study using RSP*

A pharmaceutical study (currently unpublished in submission) aims to clarify the factors that contribute to the conformational stability of μ-conotoxins. Here we describe how this work could benefit from the RSP method.

In general, the experts employ RMSD and the variance of atom positions (*Root Mean Squared Fluctuation*, RMSF) throughout MD simulations to obtain insights into how disulfide bonds affect the stability of the observed proteins. For instance, in certain cases the RMSF reveals that the removal of disulfide bonds may lead to increased structural stability (i.e., the RSMF is lowered). The removal leads to the formation of α-helices that could be visualized by our helix approximation visualization.

The RSP method adds information to the analysis of residue movements and reveals that it is also important to consider global, relative movements within a protein that are not captured by fast thermal vibrations (reflected by RMSF). For instance, our approach supports the experts in finding which residues form the core of the molecule. The RSP-map, which covers the whole simulation time, then allows the user to determine whether core residues stay within the core during the entire course of the simulation. This is important for determination and retaining of the 3D-structure characteristics of the bioactive protein. For example, core residues moving to the surface can be an indicator for increased solvent accessibility and at times loss of molecular compactness and rigidity. It is also straightforward to determine in which part of the simulation a rearrangement of residues occurs. This is of interest, because residues that move to the surface at later stages of a simulation have stronger implications on the be-

havior of the protein in solution. The RSP-map being plotted as a function of the simulation time makes this sort of useful interpretations possible. The above statements also apply to surface residues and further to correlated movements of core and surface residues. Correlated movements reveal, for example, whether the rearrangement of a core residue has impact on the surface residues in a way that they tend to move towards the core. It has already been shown by Das and Mukhopadhyay [36] that information about directionality and amplitude of residue movements contains valuable information about folding processes. The mentioned benefits contrast common magnitudes like RMSD, RMSF or RG, which rely on euclidean space and consider atoms or groups of atoms individually.

The domain experts see the RSP-map as a glue between the other measures mentioned above. Since RMSD and the RSP-map use the same time scale, it is easy to visually compare the RSP and RMSD plots to interpret atomistic events from the trajectory. If a particular region of the trajectory has a noticeable spike or dip in RMSD, the RSP-map can be looked up at the same time-step to check which residual movements contribute to this RMSD change. Since our prototype, in its current state, only computes the RSP-map, comparisons with other magnitudes were conducted by using the export functions of our tool. The same kind of intuitive visual comparison can be made between the RG graph and RSP-map in identifying which residual movements are responsible for an increase or decrease in RG values that quantify molecular compactness. The first derivative (i.e., the magnitude of change, as in Figure 8.5, bottom) of the RSP values and RMSF plots can be visually compared to find out if the thermal fluctuations quantified by the RMSF for each residue have an influence on the residue's movement between the surface and the core. As a simple example, residues such as *Arg* with long side chains and found on the surface show high RMSF values due to thermal fluctuations. However, these residues still remain on the surface of the bioactive molecule. In the chance that this residue tends to move towards the core, this would be considered as a key event captured by the RSP-map.

8.8 DISCUSSION AND CONCLUSION

In this work, we have integrated existing algorithms[220] and ideas [94, 232] to a novel measurement, Residue Surface Proximity, which reflects the closeness of the atoms or residues of a molecule to its surface or core. The novel aspect is that this is done with respect to virtual layers of the Solvent Accessible Surface definition and applied to MD simulation trajectories, providing a domain expert with a quick overview over the whole trajectory. We developed an application that extracts the data as described in our definition (Section 8.6) and which further allows the filtering of the obtained data, in order to

support the analysis step. The RSP values were successfully utilized in a real-world pharmaceutical workflow and have proven to be a useful addition to available analysis tools (see Section 8.7.3 and 8.7.2).

The timings obtained for the computation of the RSP values of the μ-conotoxins used in this work were in the range of few minutes. Larger datasets (containing more atoms or time-steps) will quickly exceed these timings, which impairs the Usefulness of the application (as pointed out by the subjects of our study). Hence, a desired addition to the current implementation is the ability to define time-ranges or sub-sampling of the input data, to reduce computational effort. We would also like to investigate the extraction of further morphology based properties that are available through our RSP algorithm. E.g., the ellipse in Figure 8.10 is the result of a fixed intersection depth $\hat{e}_{i,A}$. Adding $\hat{e}_{i,A}$ as a third dimension to the plot would result in a volume representation of all possible intersection configurations. Gathering the intersection configurations that occur for a given molecule and probe radius would yield a structure profile, which reflects the spatial relations among neighboring atoms in a new way.

The source code of our application is publicly available (see supplementary files). While there are already many free tools available for molecular processing, we do not aim to compete with these. Existing solutions (e.g., VMD [81], YASARA [111] and GROMACS [10]) have been developed over a long time and already have established a big community. However, we argue that these programs are too restrictive for the development of modern visualization techniques (only partially open source or based on legacy libraries, software and programming interfaces). The use of modern OpenGL and Qt as the basis for the user interface allows programmers to develop visualization techniques without any legacy restrictions. Further, the filtering system is extensible and can be adjusted to fit the needs of expert users. We consider connecting the filtering back-end to a graphical node interface, which we expect to be more user friendly. Extending the output of filters to change multiple different visualization parameters would also be conceivable. In this context, it is not necessary that a programmer defines a fixed set of functionalities. More likely, filters can be used to provide the expert user with a number of high level commands, which the user can then further combine to complex expressions. We think that this has the potential to accelerate the process of defining the right set of functionalities or parameters that are required for an effective usage of new developed techniques. Hence, making the interdisciplinary work process more efficient. A final step could then be to translate successful techniques to existing and more sophisticated applications with a larger community.

In summary, we consider our application prototype as a basis for interdisciplinary work between experts from the chemical and pharmaceutical domain and experts from the visualization community. Our

accompanying work from the pharmaceutical domain (currently unpublished in submission) shows that our approach is a useful augmentation to the domain experts' tools. We see potential that the given functionalities are capable of accelerating the development of further visualization or data acquisition approaches in the context of molecular research. Finally, we are confident that the open source character will benefit the exchange of applicable algorithms among the molecular visualization community and that our software prototype is valuable for further research in the pharmaceutical domain.

ACKNOWLEDGEMENTS This project was partly funded by the DFG: LA 3855/1-1 and DFG SFB 813 (to D.I.), University of Bonn (to D.I.).

8

9

MITRAL VALVE FLATTENING AND PARAMETER-MAPPING FOR PATIENT-SPECIFIC VALVE QUANTIFICATION

ABSTRACT Intensive planning and pre-operative analysis from echocardiography is a crucial step before reconstructive surgeries are applied to malfunctioning mitral valves. Volume visualizations of 3D echocardiographic data are often used in clinical routine, however, despite being very helpful, they lack a clear visualization of the crucial factors for decision making.

We are building upon patient-specific geometric mitral valve surface models segmented from echocardiography. These models include rich information about the valve's geometry, but suffer from self-occlusions due to their complex 3D-shape. Therefore, we transfer their 3D-structures to 2D-maps by unfolding their geometry to a planar space. Customized pathophysiological mappings are applied, which facilitate comprehension of the underlying pathology by representing distribution of coaptation zone area or prolapsed regions. This helps the physicians to grasp information about the mitral valve anomaly in one gaze without the need for further scene interaction.

Quality and effectiveness of the proposed methods were evaluated through a user survey conducted with domain experts. We assessed pathology detection accuracy using 3D valve models in comparison to the novel visualizations. Classification accuracy increased by 5.3% across all tested valves and by 10.0% for prolapsed valves. Further, the participants' understanding of the relation between 3D- and 2D-view was evaluated. A questionnaire, that focused on the utility of our method in a clinical work flow, was answered by the subjects and underlines the potential of the method.

In summary, our survey shows that pathology detection can be improved in comparison to simple 3D-surface visualizations of the mitral valve. The correspondence between the 2D and 3D representations is comprehensible and pathophysiological magnitudes, depicted by color-codes, further support the clinical assessment.

9.1 INTRODUCTION

Surgeries and catheter-based interventions to fix mitral valve (MV) defects are complex and require thorough planning and post-operative evaluation. Transesophageal echocardiography (TEE) is a standard clinical modality to obtain image data of the MV, which can suffer from multiple and very complex pathologies that alter the geometry

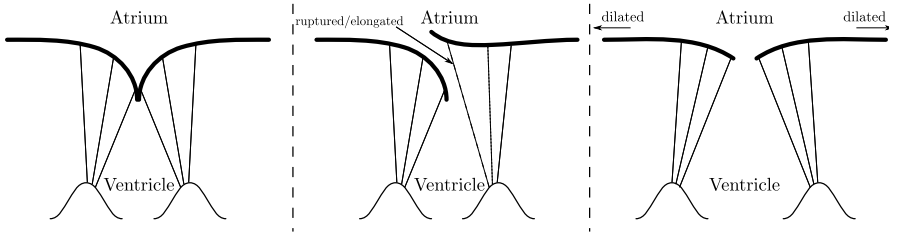


Figure 9.1: A healthy and prolapsed MV and functional MI during systole. The healthy valve separates the ventricle from the atrium (left). The prolapsed valve does not close due to ruptured chrodae tendineae (center). The functional MI prevents closure due to dilation (right).

of the valve and that ultimately affect their function. Especially 3D probes allow physicians to obtain an insightful view on the valve. Most clinical workstations offer a direct volume visualization of the captured data. However, they lack the ability to highlight important clinical pathology indicators at a glance. Thus, more enhanced visualization techniques should be added to the available tools for improved clinical assessment and surgical planning.

The MV consists of two leaflets, embedded into the mitral annulus and connected to the papillary muscles by chordae tendineae. The function of the MV is to prevent the back flow of blood into the left atrium during systole. Different pathologies can hamper this functionality, resulting in mitral regurgitation (MR). Our approach primarily supports the analysis of prolapsed MVs and valves with functional mitral insufficiency (MI). In prolapsed MVs, which is a degenerative form of MR, the chordae tendineae are either prolonged or ruptured and fail guide the MV leaflets into a closed state during systole. The prolapse can either affect a single leaflet segment or multiple segments, making qualitative and quantitative assessment on 2D-TEE data error-prone. Functional MI is caused by a dilated left ventricle, i.e., the MV leaflets are not large enough to close the valve, resulting in a small coaptation area. An illustration of a healthy MV and the two pathologies during systole is given in Figure 9.1.

The 3D convoluted surfaces of the MV leaflets are difficult to be fully understood on volume rendered TEE images, hence, advanced visualization strategies are a valuable asset. For analysis tasks, flattened representations of anatomic structures are becoming increasingly popular in the domain of medical visualizations. Such 2D representations allow the assessment of a whole object in a single view. Flattening techniques are predominantly projection-based and rely on mesh parameterization. This requires the creation of bijective mappings between a parameter domain in $\mathbb{R}^2$ and a triangulated surface embedded in $\mathbb{R}^3$. In this way, every point in the parameter domain is uniquely associated with a point in the target domain. This is a well known problem in the computer graphics field, where 2D texture

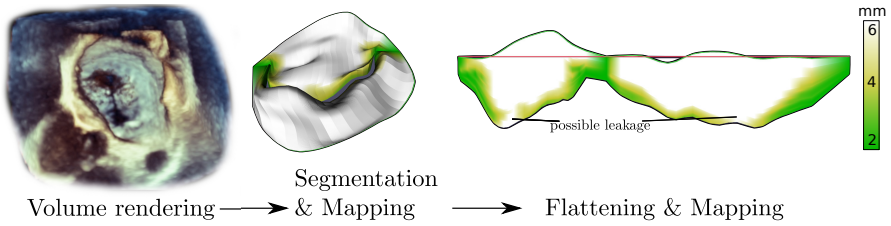


Figure 9.2: Image-based MV assessment: Volume rendering of TEE image allows for initial assessment. More detailed analysis is obtained by means of a MV segmentation. The 2D representation allows for a novel perspective onto the data and new insights. Model-based visualizations are shown with coaptation mapping (cf. Section 9.3.3) and allow thorough evaluation of the data.

images are mapped to 3D surfaces [9]. A bijective mapping applied to the whole target domain is also referred to as a global mapping, because the whole target domain is considered in the process of mapping to the parameter domain.

In this work, we will transfer the MV models to a disc-like topology and apply a global mapping. We propose a flattened view of patient specific MVs through global parameterization that preserves original structure in terms of tissue area and shape. We use a boundary-free approach that allows us to fix certain landmarks in the parameter space, enhancing overall comparability. This is, for instance, crucial as a basis for the pre- and post-operative comparison of the same patient. As a novel depiction of the MV, the resulting 2D view of a MV should be used in connection with the original 3D representation. Pathophysiological parameters can then be mapped to both, 2D and 3D view, in order to facilitate the visualization of the MV properties. The performance of this combined representation will be evaluated in a user study. Our core objectives can be wrapped up as follows:

SECTION 9.3.2: A bijective, landmark-based, equiareal mapping between the 3D and 2D representation.

SECTION 9.3.3: Application of pathophysiological mappings and localization supporting mappings in order to facilitate comprehension of the MV.

SECTION 9.4: A user study that evaluates the performance of our visualization approach w.r.t. the application in a clinical workflow.

In this paper, we build upon our previous publication [46] by providing additional detail on the background, related work and method and conduct a more comprehensive user study. With our contribution, we envision to augment the existing image-based MV assessment capabilities provided by clinical work stations. An overview of the envisioned work flow is given in Figure 9.2. Here, the volume rendering

provided by a clinical work station is used for initial assessment of the valve. An extracted 3D model and its 2D projection are then used for further in-depth quantification and decision-making. The flattening technique proposed in this paper (Fig 9.2, right) builds upon a MV segmentation based on Engelhardt et al. [45] (Fig 9.2, center) and provides a novel, supportive view of the MV to enhance diagnosis and therapy decisions.

9.2 RELATED WORK

This section will cover previous work from the mesh parameterization domain and visualization of MV data sets. A general overview of mesh parameterization techniques can be found in [77]. This field is based on early work in graph theory [222] and most commonly applied to computer graphics in purposes of texture mapping [9, 144]. Here, a one-to-one mapping from 3D surfaces to the 2D domain is sought in order to map textures to the surface with minimized or user-controlled distortion. Other approaches can reduce distortion even better, even for arbitrary topology, but do this at the cost of the one-to-one mapping property. Then, bijectivity is only given locally, as in Ray et al. [176].

The visualization of 3D MV models has been applied in the context of MV simulation. For example, Rim et al. [182] studied the effect of leaflet-to-chordae contact interaction and visualize the result as a color map. In a virtual leaflet resection scenario Rim et al.[183] color code leaflet stress measurements before and after the virtual resection. Similar depictions of mapped magnitudes can be found in the work by Zhang et al. [234]. While the applied color maps allow to perceive distinct differences of individual data sets, a precise comparison may be hampered by the convoluted leaflet morphology. In such a case, a flattened representation, exposing the whole surface in one view, may provide an advantage. Thus, a comprehensible overview of the MV is not only required in the clinical context, but also in the domain of simulation and modelling.

Flattened depictions have been proposed for the circulatory system, the colon, the brain and the bones. A 2D representation of aortic valve prostheses has been introduced as well [21]. Properties like stent compression were assessed after implantation in order to analyze complication co-occurrences. A recent state of the art report [109] reviews a variety of medical visualization techniques focused on planar representations.

9.3 MATERIALS AND METHODS

We target patient-specific maps and rely on an already existing semi-automatic method to extract MV surface models from 4D ultrasound

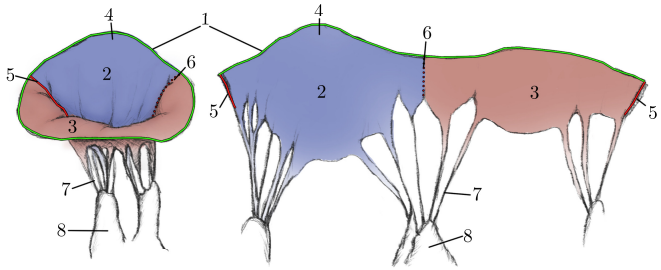


Figure 9.3: MV similar to depictions in anatomy books with closed (left) and flattened valve (right), cut along the lateral commissure. Important anatomical features are marked.

scans [45], consisting of annulus and leaflets. This segmentation algorithm provides separate triangulations for both MV leaflets and the annulus. Anatomical markers (cf. Figure 9.3) are already embedded in the representation (cf. Figure 9.4, left) and can be utilized during the flattening. Chordae tendineae and papillary muscles are not part of these models. Please note that TEE images are not capable of depicting the small branching structures of the chordae tendineae, therefore, our work mainly focuses on leaflets and annuli. We will now provide details concerning the notation and requirements. Then, the following section describes the flattening process. Finally, color- and texture mapping approaches are depicted, that help to grasp the MV morphology and 3D to 2D correspondence.

9.3.1 *Requirements and notation*

The mathematical notation of the valve models is described as follows: each 3D valve mesh consists of vertices $\mathcal{V} \subset \mathbb{R}^3$. A 3D vertex is always denoted as $\mathbf{p}_i = (x_i, y_i, z_i)$ and its counterpart in 2D as $\mathbf{q}_i = (u_i, v_i)$. The points $\mathbf{q}_i$ make up the parameter domain $\Omega \subset \mathbb{R}^2$, i.e., represent the coordinates of the flattened 2D model. An important subset consists of vertices lying on the annulus $\mathcal{V}_a = \{\mathbf{p}_1, \ldots, \mathbf{p}_m\} \subset \mathcal{V}$, where m is the number of annulus vertices (cf. Figure 9.4, green points, left). Each annulus point is also the first point in a subset $\mathcal{V}_l = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \subset \mathcal{V}, l \in \{1, \ldots, m\}$, which forms a single line from an annulus vertex $(\mathbf{p}_l \subset \mathcal{V}_a) = (\mathbf{p}_1 \subset \mathcal{V}_l)$ to the coaptation (cf. Figure 9.4, red points, left). Moreover, the topology of the MV mesh is described by a set of triangles $t \in \mathcal{T}, t = \{\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k\}$, with the corresponding set $\bar{t} \in \bar{\mathcal{T}}, \bar{t} = \{\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k\}$. An illustration of $\mathcal{V}$ and Ω is given in Figure 9.4 (right), where the red edge represents the cut along the lateral commissure. The goal of our parameterization will be to find a solution for the function $\mathcal{X}$, to map the points $\mathbf{p}_i \in \mathcal{V}$ to $\mathbf{q} \in \Omega$. The quality of a flattening technique can be determined through metrics describing the amount of (inevitable) distortions. Usually, parameterization methods either focus on preserving angles (conformal) or

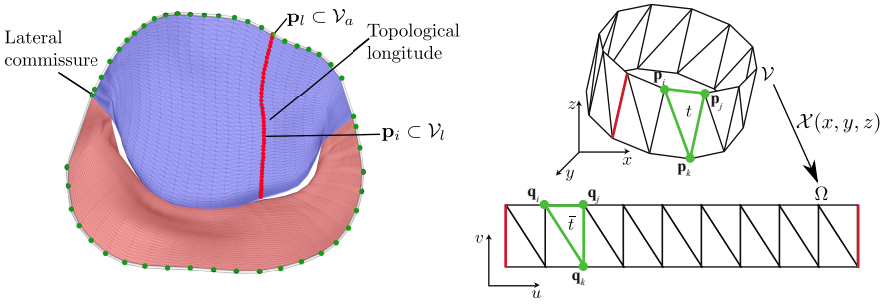


Figure 9.4: The annulus points $\mathcal{V}_a$ (green), the first one being the point on the lateral commissure (left). Vertices of one topological longitude $\mathcal{V}_l$ (red, left). In the 3D domain, $\mathcal{V}$ consists of points $\mathbf{p}$ and triangles $t \in \mathcal{T}$ and is mapped via $\mathcal{X}$ to Ω in the 2D domain with points $\mathbf{q}$ and triangles $\bar{t} \in \bar{\mathcal{T}}$ (right). Red edges depict where the original model is cut to form a disc-like topology.

area (equiareal) of an input mesh [58]. Fulfillment of both characteristics would result in an isometric or length-preserving parameterization. It is desirable that a 2D-view of the MV is close to isometric or at least equiareal. Retaining the proportions of the MV is a primary goal of our method, which should enable the possibility of area and length quantification on the flattened surface. Further, a physician using the 2D-view should develop an intuition for its orientation and scale. Hence, apart from minimizing distortions, the 2D-view should also target comparability across different data sets through a uniform appearance. Lastly, spatial context should be preserved, i.e., the relation between the 3D- and 2D-domain should be clear.

The general idea of the proposed flattening algorithm is to cut the MV along its lateral commissure and to unroll it along its diameter. This results in a perspective similar to the valve's depiction in standard textbooks [25] (cf. Figure 9.3, right). In our algorithm we split the flattening process into three steps:

SECTION 9.3.2.1: Annulus parameterization.

SECTION 9.3.2.2: Leaflet initialization.

SECTION 9.3.2.3: Leaflet relaxation optimization.

The shape of the annulus can give important hints during pathology analysis, therefore we parameterize it as a curve, independent from the MV leaflets.

9.3.2 *Flattening*

The three flattening steps will be detailed in this section. After the annulus parameterization, its configuration will remain unchanged during all further steps, preserving the annulus shape and arc length

and increasing comparability across different data sets. The subsequent steps for the leaflet parameterization are then based on the annulus curve.

9.3.2.1 *Annulus parameterization*

The annulus' height is plotted along the v- and its length along the u-axis of the 2D-view (cf. Figure 9.5). The correspondence of the u-axis in 3D is a reference plane through the annulus curve. An intuitive approach to compute the reference plane would be the least-squares (LS) method, resulting in a plane with minimal distances to points on the annulus. However, this approach does not always lead to good results, i.e., the annulus' height is over- or underestimated (cf. Figure 9.5, top). Further, the anterior saddle horn is an important feature that we would like to accentuate. Height of the saddle horn can be different in dependence of pathologies, e.g., functional MI can be associated with a more planar annular shape [159], while normal or prolapsed valve have a non-planar saddle shaped shape. Therefore, we employ a landmark-based approach, defining the annulus plane through three points: the two commissure points on the annulus, which form a natural axis through the MV and the barycenter of the posterior annulus, which usually approximates a planar layout. This yields an annulus plane $\mathcal{P}_a$ with normal $\mathbf{n}_a$. The annulus parameterization is then obtained as:

$$\begin{aligned} v_i &= \langle \mathbf{n}_a, (\mathbf{p}_i - \mathrm{proj}(\mathbf{p}_i, \mathcal{P}_a)) \rangle, \\ u_i &= \begin{cases} 0 & \text{if } i = 1, \\ u_{i-1} + \sqrt{\|\mathbf{p}_i - \mathbf{p}_{i-1}\|^2 - (v_i - v_{i-1})^2} & \text{otherwise.} \end{cases} \end{aligned} \tag{9.1}$$

where $\mathrm{proj}(\mathbf{p}, \mathcal{P})$ is the projection of a point $\mathbf{p}$ onto the plane $\mathcal{P}$ and $\langle \cdot, \cdot \rangle$ denotes the dot-product. In this way, the distance of two adjacent points in $\mathcal{V}_l$ is equal to the distance of the corresponding parameterization in Ω: $\|\mathbf{q}_i - \mathbf{q}_{i-1}\| = \|\mathbf{p}_i - \mathbf{p}_{i-1}\|$. The resulting 2D view now allows to compare the annulus curve in relation to the u-axis, i.e., the location and height of the anterior saddle horn can be assessed in relation to the rest of the curve. Figure 9.5 compares the result of the landmark-based approach (bottom) with the LS approach (top). It can be observed that the landmark-based method achieves a more intuitive representation of the 3D annuls shape and the anterior saddle horn can be clearly pointed out in the 2D depiction.

9.3.2.2 *Leaflet initialization*

The leaflet geometry is placed below the annulus, similar to the appearance of Figure 9.3. First, a valid configuration is initialized, i.e., a leaflet layout without self-intersections or triangle-flips. We exploit

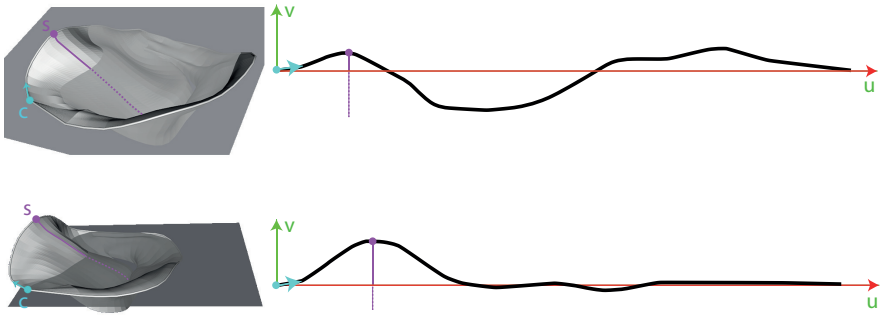


Figure 9.5: 3D MV model and annulus plane (left) constructed by a LS method (top) and the landmark-based method (bottom). Parameterization of annulus (right). Lateral commissure (c) and saddle horn (s) including its iso-u line are marked.

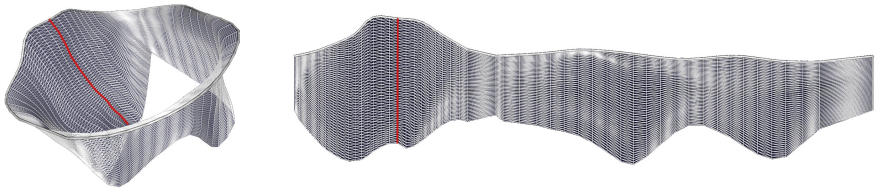


Figure 9.6: Wireframe of the initial layout of the mapping from 3D (left) to 2D (right) below the already parameterized annulus. An example iso-u curve is shown in red.

the spline-like lines $\mathcal{V}_l$ which are interpreted as iso-u curve approximations and points of each line $\mathcal{V}_l$ are mapped to a shared u-coordinate while the distance between points is preserved in v-direction:

$$\begin{aligned} u_i &= u_1 \qquad \forall i \in [2, n] \\ v_i &= v_{i-1} - \|\mathbf{p}_i - \mathbf{p}_{i-1}\| \qquad \forall i \in [2, n]. \end{aligned} \tag{9.2}$$

where $\mathbf{q}_1 = (u_1, v_1)$ corresponds to the previously parameterized vertex $\mathbf{p}_1 \in \mathcal{V}_a$ on the annulus. Note that this approach requires the vertices in $\mathcal{V}_l$ to be ordered from annulus to coaptation. An intermediate result of this initialization step is shown in Figure 9.6.

9.3.2.3 *Leaflet relaxation optimization*

As all lines $\mathcal{V}_l$ are now parallel in parameter space, the parameterization does not faithfully reflect the area and morphology of the original 3D valve. Therefore, we optimize towards a more equiareal parameterization. We aim to minimize an energy term describing the distortion amount of the mesh's edge lengths. If the 3D mesh consists of vertices $\mathbf{p}_i$, corresponding to uv-coordinates $\mathbf{q}_i$ and the set N_i

contains the indices of all neighbors of $\mathbf{p}_i$, a per-vertex edge length energy can be described as

$$\mathcal{E}_l = \frac{1}{|N_i|} \sum_{j \in N_i} \frac{\|\mathbf{q}_i - \mathbf{q}_j\|}{\|\mathbf{p}_i - \mathbf{p}_j\|} + \frac{\|\mathbf{p}_i - \mathbf{p}_j\|}{\|\mathbf{q}_i - \mathbf{q}_j\|}. \tag{9.3}$$

Note that this energy reaches its minimum $\mathcal{E}_l = 2$, if and only if all observed edge lengths in the 3D mesh are equal to their counterpart in the parameter domain and therefore both fractions evaluate to 1.

We use an iterative Euler method to minimize this energy, modelling the mesh edges in the parameter space Ω as a network of springs. The points $\mathbf{q}_i$ are displaced each iteration in the direction of a summed spring force $\mathcal{F}$ calculated based on the model's edge lengths:

$$\mathcal{F}_i = \sum_{j \in N_i} k\Big(\|\mathbf{p}_j - \mathbf{p}_i\| - \|\mathbf{q}_j - \mathbf{q}_i\|\Big) \frac{\mathbf{q}_j - \mathbf{q}_i}{\|\mathbf{q}_j - \mathbf{q}_i\|}, \tag{9.4}$$

with $k = 1$ being a constant stiffness parameter. A similar method was proposed to simulate MV closure [61]. The above force vector calculation does not take the angles of the edges between the points into account. However, due to our initialization (cf. Eq. 9.2), the layout of Ω is already in a shape that allows to omit this aspect.

9.3.3 *Mappings*

After the parameter space has been established, a variety of parameters can be color-mapped onto the 2D and 3D surface. To support a clinical work flow, we address two kinds of mappings:

- *Localization mappings* help to understand the 3D/2D correspondence, by applying the same color or texture pattern in both views.
- *Pathophysiology mappings* display medical information, which can facilitate MV analysis.

We implemented two pathophysiology mappings. The first one shows an approximated coaptation zone, where we used a 2 mm distance threshold between anterior and posterior leaflet to determine the parts of the surface that collide. The other one is similar to a height-map. It marks areas of the MV which are above and below the annulus-plane. All mappings are applied to both, the 3D- and 2D-view, which are always rendered side-by-side. To support localization, we delineate anterior and posterior leaflet segments on the surface as proposed by Carpentier et al. [25] (cf. Figure 9.7). The subdivision is based on the parameterization Ω. For the posterior leaflet, all three segments occupy an equal range of the u parameter. The anterior

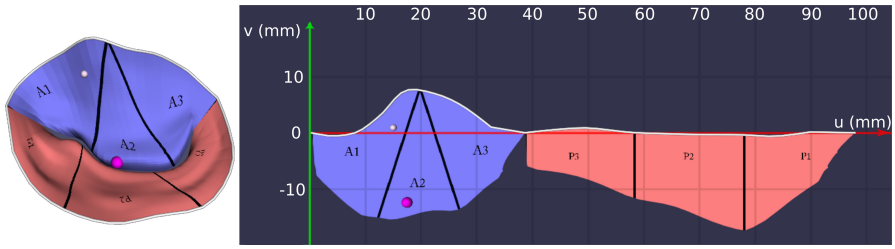


Figure 9.7: Leaflet subdivision improves 3D-2D correspondence. User-defined points (magenta and white sphere) allow a look-up of points of interest. The grid overlay supports size estimation.

leaflet is split up such that the three segments join at the vertex representing the center of the anterior annulus curve, while at the coaptation, the segments occupy an equal range of the u parameter. Figure 9.7 additionally shows two spheres (magenta and white), placed on the 2D and 3D surface. The spheres can be placed by the user to get a precise feedback of the point correspondence and, e.g., to measure the distance of the selected points, or to compare local magnitudes derived from the model data. Finally, a grid overlay with 10 mm spacing allows for a quick estimation of the leaflet size.

9.4 EVALUATION

The evaluation of our technique is split into a technical and practical part. The technical part covers the performance of the parameterization algorithm and the practical part covers the evaluation of our visualization approach in the context of MV assessment user study.

9.4.1 *Parameterization Evaluation*

To evaluate the employed optimization step, we performed measurements concerning area, angle and edge-length deformation (Eq. 9.3) using 50 MV models and compared the results before and after the spring relaxation method. The area distortion was measured as:

$$\mathcal{E}_{A,i} = \frac{A(t_i)}{A(\bar{t}_i)} + \frac{A(\bar{t}_i)}{A(t_i)} \tag{9.5}$$

where $A(t)$ is the area of a triangle. The angle distortion is computed according to the formulation found in [78]:

$$\mathcal{E}_{M,i} = \frac{\cot\alpha|a|^2 + \cot\beta|b|^2 + \cot\gamma|c|^2}{2A(t_i)}. \tag{9.6}$$

where a, b and c are the edge lengths of a triangle t_i and α, β and γ are the interior angles of $\bar{t}_i$, opposite of the respective edges. The minimum of $\mathcal{E}_M = 2$ is reached if t and $\bar{t}$ have equal angle-to-edge proportions.

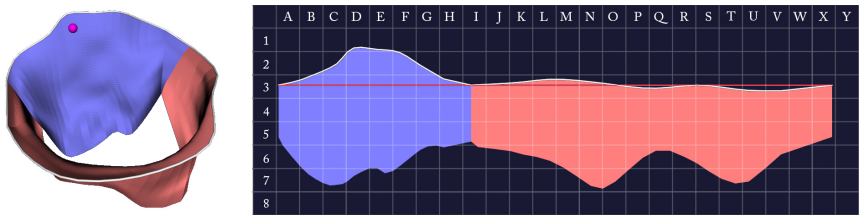


Figure 9.8: Localization task without localization mapping. In this example, the correct corresponding grid cell for the point shown in 3D would be *D2*.

9.4.2 *User Study*

We assessed the capabilities of the proposed visualization in a user study conducted with one visualization expert, three cardiac surgeons and one anesthetist. After an introductory video, subjects were given a point-localization task, where they were asked to mark corresponding points in the 3D- and 2D-view. The tasks were given with the leaflet segments shown as in Figure 9.7 and without the segments. In each task, a point was given either in the 3D or 2D view and the 2D or 3D view was augmented with a grid layout. Subjects were then asked to point out the cell in the 2D grid that corresponds to the marked point in 3D (cf. Figure 9.8).

The next task was designed to simulate clinical decision making. Within an interactive prototype of our implementation participants were subsequently shown 20 MV models, each in two alternating formats, resulting in 40 tasks: half of the tasks displayed models only in 3D (without color-coding) and the other half in a combined 3D/2D-view (with color-coding). In the latter, participants could access the coaptation and the height map (cf. Figure 9.9). Participants were asked to assign each valve to a category: normal, prolapsed or functional mitral regurgitation. The tasks were presented in a randomized order, making direct comparison possible. Participants were further instructed to mark their confidence in their classification on a Likert-scale from one (not confident) to five (very confident).

After the hands-on part of the evaluation, participants filled out a questionnaire, which mainly consisted of providing approval-ratings for specific statements concerning the 2D-view. Again, a Likert-scale from one (strongly disagree) to five (strongly agree) was given. Each participant had to rate the following statements:

C1 The correspondence between the 3D- and 2D-view is clear.

C2 Different valves result in distinguishable 2D-views.

C3 Pathologies of the MV are easier to identify when the 2D-view is provided.

C4 Analysis of the MV is facilitated through the 2D-view.

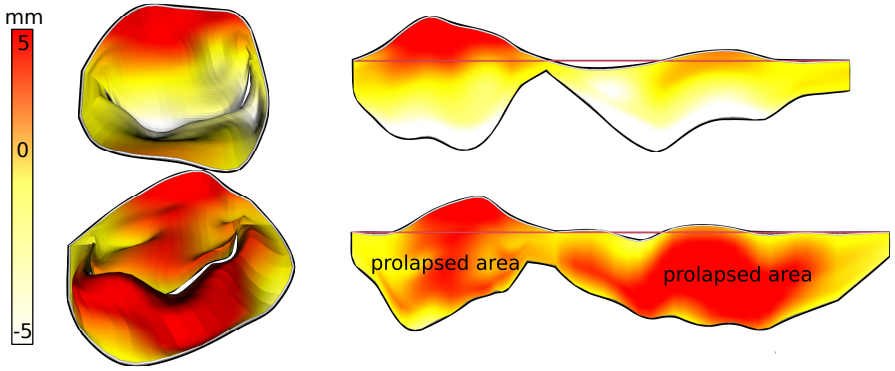


Figure 9.9: Combined 3D (left) and 2D (right) view with pathophysiological mapping for a healthy valve (top) and prolapsed valve (bottom). The depicted prolapse (height) mapping shows the signed distance of a leaflet point to the annulus plane.

9.5 RESULTS

Evaluation of the relaxation step showed that we were able to optimize the average edge-length energy $\mathcal{E}_l$ from $2 + 0.78$ before the relaxation to $2 + 0.07$ afterwards. The area distortion dropped from $2 + 1.9$ to $2 + 0.2$. This is at the cost of the angular distortion, which was not addressed in the optimization formulation and remained almost equal at a value of $2 + 1.4$. With this, the goal of achieving an equiareal mapping of the MV is met to a satisfying amount. Remember, that the minimum of each magnitude is 2 and that distortion is inevitable due to the transformation from the complex MV morphology to a planar representation.

The point localization task revealed that the 3D-2D correspondence can be well understood, especially with the leaflet segments being displayed. Without leaflet segments, the average distance of the correctly mapped 2D point to the point selected by the subject (measured from correct grid cell center to selected grid cell center) was approximately 4 mm (or 0.74 cells). The average distance with leaflet segments on was zero, w.r.t. the grid cell size. This means that all subjects were able to select the correct grid cell in every task with segments displayed.

Two examples of the resulting visualization can be seen in Figure 9.2 (right) and Figure 9.9. The visual analysis of the 2D maps provide a much more detailed understanding of the pathology: in the 3D-view it is not clearly visible whether the valve fully closes or not. In contrast to that, the 2D-coaptation view (Figure 9.2, right) indicates a part where the leaflets do not touch. Beyond that, the height map (Figure 9.9) illustrates the extent of prolapsing areas very well, i.e., tissue which surpasses the annulus plane in systole is marked in red.

The pathology assessment survey showed an increase in the pathology detection rate in 4 out of 5 participants when they had access to

Table 9.1: Average of the pathology identification task per participant P regarding accuracy A_i, confidence C_i and time T_i for 3D-only-view ($i = 0$) and 3D/2D combination ($i = 1$).

P	A_0	A_1	C_0	C_1	T_0	T_1
1	78.9%	84.2%	3.89	4.16	11.5s	15.4s
2	68.4%	73.7%	4.37	4.53	14.0s	15.7s
3	78.9%	89.5%	4.58	4.84	15.9s	12.6s
4	84.2%	73.7%	3.00	3.00	13.9s	13.2s
5	84.2%	100%	4.53	4.68	15.9s	16.0s
Total	78.8%	84.2%	4.07	4.24	14.3s	14.6s

the 3D surface and the flattened MV, including the coaptation and prolapse color maps (cf. Tab. 9.1). A total average of 5.3% increased accuracy was measured. Most noticeably, global detection accuracy for prolapsed valves rose from 85% to 95%. The time required for one classification averaged at about 14s, regardless of the view mode. Participants were slightly more confident in their decisions when using the combined view. Average confidence (discrete scale from 1 to 5) rose from 4.07 to 4.24. When making incorrect classifications, participants reported an average confidence of 3.53 in both view-modes. For correct classifications, average confidence increased from 4.08 (3D-only) to 4.28 (3D/2D).

The scores obtained for the four categories of the questionnaire underline that the subjects are in favor of the proposed visualization technique and see the potential for improvement of MV assessment. The average scores from 1 (strongly disagree) to 5 (strongly agree) were: ($C1$, 4.9), ($C2$, 4.2), ($C3$, 4.6), ($C4$, 4.6).

9.6 DISCUSSION

We presented an approach for flattening patient-specific 3D MV models, that results in an expressive 2D depiction across different data sets. The visualization targets clinical MV analysis, a process that appears to benefit from the proposed 2D-view with color-maps. This is underlined by the increased pathology detection rate measured in our survey, which holds especially true for prolapsed valves. The coaptation zone can be assessed at a glance, as well as the prolapsed valve area. A landmark-based parameterization of the annulus makes comparison of height deviations possible. Low area and edge-length distortions of the leaflet geometry allow size quantification of the flattened MV. The evaluation shows that spatial context is preserved as the domain experts had no difficulties understanding 3D/2D correspondence. Participants claimed pathologies were easier to identify and MV analysis was facilitated when the 2D-view was provided.

They pointed out that determining the valve part affected by a prolapse was aided by the flattened representation, even though the study was not particularly designed for this aspect. The comparison of pre- and post-operative leaflet stress as done by Rim et al. [183] would also benefit from our approach. While the pathophysiological mappings used in our study were derived from 3D data, it may be interesting to evaluate whether measurements obtained from the 2D representation can also be employed for the quantification and comparison of MVs. The planar MV view could also be extended by inclusion of functional information.

Our leaflet relaxation optimization (cf. Section 9.3.2.3) treats all parts of the MV equally and does not optimize angular distortion. Hence, the final distortion is predominantly influenced by the leaflet initialization step (cf. Section 9.3.2.2). A future extension may involve a more user-controlled parameterization, that favors regions of interest and relocates the distortion to less important regions. This would support the faithful visualization for in-depth analyses. Another interesting aspect would be incorporation of the time variable. Using a stacked view of 2D representations or applying a suitable form of dimensional reduction, samples over the whole cardiac cycle could be visualized in 2D at once, simplifying assessment of annulus and leaflet shape variation. Our landmark-based parameterization might aid in the registration of multiple time-steps in order to establish meaningful correspondences over time.

A possible drawback of our method is its low generalizability. The approach is tailored to the MV and our implementation relies on the structure of the MV model by Engelhardt et al. [45]. However, our results should motivate the extension to more generic representations of the MV and the incorporation of more controllable parameterization methods, as well as additional clinically relevant magnitudes.

ACKNOWLEDGEMENTS This work was supported by the German Research Foundation (DFG) project 398787259, EN1197/2-1, DE 2131/2-1 and LA 3855/1-1.

Part V

CONCLUSION

This part wraps up the contributions of this thesis to the visualization community and discusses open research directions.

10

SUMMARY

This thesis made several contributions down along the *Visualization Pipeline*, with a focus on data abstraction. With respect to the concept of interactive, visual reasoning, the proposed methods were designed with a secondary focus on quick, responsive execution. Multiple technical disciplines come together to reach from data preparation to a final visualization. These disciplines can be examined individually, to find generalized solutions. In contrast, they can be designed w.r.t. the whole pipeline formulation or a certain task, which was attempted throughout this thesis. Therefore, the work at hand stepped into each part of the *Visualization Pipeline* and thus provided a broad overview of challenges, solutions and opportunities at the example of bio-medical data visualization.

Previous works already describe the visualization pipeline in other ways, that allow for more interactivity or larger datasets [8, 200]. From the contributions of Part III an augmented formulation of the *Visualization Pipeline* by Haber and McNabb [59] can be extracted (recall the *Data Enrichtment*, *Visualization Mapping* and *Rendering* steps). If the results of the *Rendering* step, i.e., the generated image or the applied transformations, are used to adjust, optimize or adapt the visualization in the successive frame, then the *Visualization Pipeline* forms a loop. This was shown in Chapter 4, where data glyphs are placed in dependence of the view parameters and distributed in an optimized way. Further, Chapters 5 and 6 used the result of a rendered image to augment the screen-space with a parameterization of the depicted surface. The parameterizations can be directly employed for visualization tasks. Thus, the linear pipeline property of the definition by Haber and McNabb [59] dissolves. The baseline for the loop property was given by the view-dependent interactivity of the proposed techniques. Future work could pick up this track to formulate a visualization pipeline definition that is suitable to describe such visualization concepts. Figure 10.1 depicts the *Visualization Pipeline* with an additional step *Render Adaption* that closes the loop. It further lists all contributing Chapters for each part of the extended pipeline.

The main focus of this work was on the visual abstraction of spatial or spatio-temporal bio-medical data. Part II dealt with continuous visual abstraction for focus-and-contact visualization, using line-drawing techniques. Especially for hatching, the question on where (surface sampling) and along which direction (field alignment) to put lines arose. As a solution, an algorithm of the mesh processing domain was picked up and made suitable for pixel-precise generation

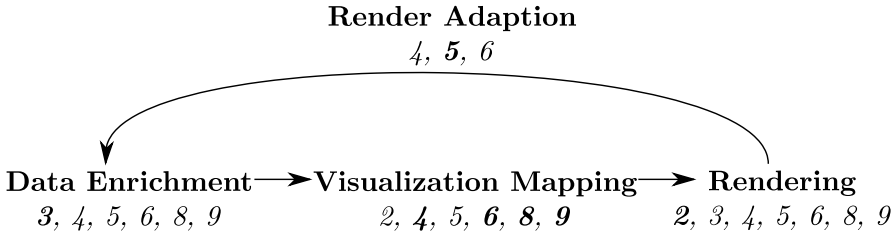


Figure 10.1: Affiliation of each chapter with the steps of the extended *Visualization Pipeline*. Major affiliations marked in bold face.

of periodic texture coordinates that are field-aligned. It was further shown that underlying parameters can be changed and the resulting texture coordinates be frame-coherently updated. This enables interactive visualizations that encode dynamic information using patterns based on the texture coordinates. Also, the texture coordinates can be utilized to extract surface samples with a user-defined spacing and orientation. These sample locations may be used for placement of data glyphs, as done by Hombeck et al. [76].

Part III focused on the visualization of liver vasculature. An active topic regarding such data sets is the improvement of depth perception. Novel techniques were presented to contribute to this area. As a part of this, feature-extraction methods were introduced to extract vascular end- or branch-locations from surface data, that are useful to support subsequent visualization strategies. Using additional geometric objects that are more complex than data glyphs was identified as the concept of *Auxiliary Tools*. The term has been used by Preim et al. [175] in the context of depth perception, but without a definition, which is now provided with this thesis based on the current state-of-the art. This state-of-the-art was further examined and goals for future work were extracted, that focus on how to conduct reproducible user studies in this area. The part proceeded with another contribution to the visual abstraction of vascular structures. Here, a linear or radial 2D graph depicts the branching topology of the vasculature. The background of the graph plot can be color coded or augmented with abstract patterns in order to encode data that corresponds to the nearby tree segments. In this way, the whole tree-structure can be analyzed in one gaze to point out critical information. Latterly, the utility of SDFs for visualization was studied. Once calculated, an SDF provides a handy framework for the implementation of different illustrative effects or focus-and-context applications. To provide a parameterization that fits the implicitly rendered SDF, a screen-space parameterization method was introduced. It is based on the parameterization described in the previous part, but does not require an explicit surface representation.

Part IV further pursued the visualization of data through data abstraction and dimension reduction, applied to data from the biomedical domain. Here the spatio-temporal data of MD simulations

were reduced to a 2D heatmap. In this course, the spatial information for a single time-step is highly limited. In return, spatial correlations over time can be better perceived. The resultant overview enabled the domain experts to narrow their analysis to time-frames of interest. For the mitral valve, a convoluted 3D structure was unfolded to make it displayable in the 2D domain. This maintains spatial relations and allows to view the full surface at one gaze in a single view. At the same time, the novel representation yields parameters that may support clinical analysis tasks or serves as a canvas for common data-derived magnitudes. The incorporated domain experts assumed great potential for these purposes.

10

11

FUTURE WORK

The contributions of this thesis are already discussed w.r.t. future work in the respective chapters. However, this chapter wraps up possible future goals for each part of the thesis and therefore gives a broader overview on what can be done to build up on this thesis.

Part II presented a screen- and sample-based line drawing technique in combination with existing feature line methods to achieve continuous visual abstraction. The sampling used to initialize the line-drawing was not found to be optimal and the parameterization-based sampling technique presented later in this part could be applied as a substitute. This would allow for a more controllable distribution of line samples and therefore a better hatching quality would be expected. Further, objects that are presented with a high abstraction level are depicted only with a few lines and occupy only a fraction of the visual foreground. It needs to be investigated how this can be combined with transparency effects in order to allow for a better visibility of nested objects. The proposed periodic texture coordinates could be as well combined with this task, as exemplified with the *binary transparency* in Chapter 3. A broad user-study should be conducted that investigates on how people perceive different levels of abstraction and whether there are limits w.r.t. the comparability and granularity of different levels (compare the study by Chung et al. [31]). A strong limitation of the parameterization approach is the resolution of the input mesh. While a hierarchical structure speeds up the computation and improves the quality of results significantly, adaption to dynamic mesh input can not easily be handled. Dynamic update of resolution hierarchies has already been investigated [198], but should be applied to the parameterization task in order to proof the applicability of the respective method.

Part III dealt with the visualization and enhancement of depth perception of vascular structures. The presented glyph-based method is already recapped and compared against other existing methods in Chapter 7. The most important open questions in the evaluation of these methods is how difficult the study tasks actually were. This seems to differ greatly among the individual papers. Thus, the study setup needs to be more carefully designed. Phases of human visual perception, vascular complexity, sample selection, just noticeable depth differences (JNDD) and the generalizability of an approach have to be taken into account. Only then can different methods be faithfully compared. Another contribution of this part is the parameterization of tubular structures. The proposed hybrid approach allows to combine

an object-space static parameter, with a dynamic view-dependent, screen-space parameter. This enables a convenient way to align a texture coordinates-based visualization with the user's view. On the down-side, frame-coherency of the screen-space part is an issue, which should be addressed to make this approach more sophisticated. While a full screen-space approach is already covered in this part, the two methods differ in an important aspect: the hybrid approach yields global parameters, i.e., each 2D parameter is unique, but the full screen-space approach yields periodic texture coordinates. A very interesting direction would be to remove the periodicity of the parameterization (globally or patch-wise) while adjusting or maintaining the spacing of the mapped parameters. Moreover, the screen-space parameterization can be used for sample extraction on implicit surfaces. Since the sampling runs in real-time and in screen-space, the technique has the potential to maintain frame-coherent surface samples, even if the surface morphology changes. For this, a tracking of samples over multiple frames would be required. Concerning the SDF visualization, this chapter should be able to motivate further work in this direction. Here, future work should cover questions on how more elaborate illustrative techniques can be applied to implicitly rendered surfaces, based on the information given by an SDF. While sophisticated line drawing techniques exist that can be applied to implicit surfaces extracted from volume data [168, 231], SDFs can be expected to enable new solutions. This is due to the fact that an SDF holds information in a *3D volume extracted from a single surface*. Therefore, much more specific information about that surface is available. In contrast, for common volume visualizations the result is a *surface extracted from a 3D volume*. Also, SDFs may greatly support the ease of implementation and possibilities for focus-and-context as well as magic-lens interaction paradigms via Boolean operations on multiple SDFs. SDFs could also prove useful for surface feature extraction, for example, based on morphological operations.

Part IV presented two task-oriented data abstraction approaches. Both developed techniques remain as the first iteration of a prototypical solution. Application in clinical practice and real-world workflows should be considered to extract more capabilities and limitations of the methods. In both cases, a focus should be lain at the data analysis part of the software prototypes. As the previous focus was on data abstraction and visualization, the analytics part has come short. Integrating the novel methods with existing analysis tools would allow to asses the benefit of the respective method more faithfully. The mitral valve unfolding should further be investigated w.r.t. the susceptibility to and control of distortion [58]. Moving the inevitable distortion to less interesting anatomical landmarks may further improve the feasibility of the approach. Additionally, the time variable should be considered to capture whole cardiac cycles in one visualization.

More generally, the methods presented in this thesis should be considered within the light of *Visual Analytics*, i.e., the concept of *Visually Enabled Reasoning*. As mentioned in the introduction, the visualization alone is only one step on the path to gaining insight into data. The interaction of the user with the data or with the visualization represents another important aspect. Therefore, advanced interaction concepts should be investigated that are enabled by the novel foundations provided by this dissertation.

BIBLIOGRAPHY

[1] Akondi, K. B., Muttenthaler, M., Dutertre, S., Kaas, Q., Craik, D. J., Lewis, R. J., Alewood, P. F., "Discovery, Synthesis, and Structure-Activity Relationships of Conotoxins." In: *Chemical Reviews* 114.11 (2014), pp. 5815–5847.

[2] Alpers, J., Hansen, C., Ringe, K., Rieder, C., "CT-Based Navigation Guidance for Liver Tumor Ablation." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. 2017, pp. 83–92.

[3] Au, O. K.-C., Tai, C.-L., Chu, H.-K., Cohen-Or, D., Lee, T.-Y., "Skeleton Extraction by Mesh Contraction." In: *ACM Transactions on Graphics (TOG)* 27.3 (Aug. 2008), 44:1–44:10.

[4] Auer, C., Stripf, C., Kratz, A., Hotz, I., "Glyph- and Texture-Based Visualization of Segmented Tensor Fields." In: *GRAPP/IVAPP*. 2011, pp. 670–677.

[5] Barbier, A., Galin, E., "Fast Distance Computation Between a Point and Cylinders, Cones, Line-Swept Spheres and Cone-Spheres." In: *Journal of Graphics Tools* 9.2 (2004), pp. 11–19.

[6] Behrendt, B., Berg, P., Preim, B., Saalfeld, S., "Combining Pseudo Chroma Depth Enhancement and Parameter Mapping for Vascular Surface Models." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. 2017, pp. 159–68.

[7] Belkin, M., Sun, J., Wang, Y., "Discrete laplace operator on meshed surfaces." In: *Proceedings of Symposium on Computational Geometry*. ACM, 2008, pp. 278–287.

[8] Benger, W., Ritter, G., Ritter, M., Schoor, W., "Beyond the visualization pipeline: The visualization cascade." In: *Proc. of High-End Visualization Workshop. Lehmanns Media GmbH*. 2009, pp. 35–49.

[9] Bennis, C., Vézien, J. M., Iglésias, G., "Piecewise Surface Flattening for non-distorted Texture Mapping." In: *ACM SIGGRAPH Computer Graphics* 25.4 (1991), pp. 237–246.

[10] Berendsen, H., Spoel, D., Drunen, R., "GROMACS: A Message-Passing Parallel Molecular Dynamics Implementation." In: *Computer Physics Communications* 91.1 (1995), pp. 43 –56. ISSN: 0010-4655.

[11] Berg, P., Roloff, C., Beuing, O., Voss, S., Sugiyama, S.-I., Aristokleous, N., Anayiotos, A. S., Ashton, N., Revell, A., Bressloff, N. W., "The computational fluid dynamics rupture challenge 2013 - phase II: variability of hemodynamic simulations in two intracranial aneurysms." In: *Journal of biomechanical engineering* 137.12 (2015), p. 121008.

[12] Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., Bourne, P. E., "The Protein Data Bank." In: *Nucleic Acids Research* 28 (2000), pp. 235–242. URL: www.rcsb.org.

[13] Bichlmeier, C., Heining, S. M., Feuerstein, M., Navab, N., "The virtual mirror: a new interaction paradigm for augmented reality environments." In: *IEEE Transactions on Medical Imaging* 28.9 (2009), pp. 1498–1510.

[14] Bloomenthal, J., Shoemake, K., "Convolution surfaces." In: *ACM SIGGRAPH Computer Graphics* 25.4 (1991), pp. 251–256.

[15] Bommes, D., Zimmer, H., Kobbelt, L., "Mixed-integer quadrangulation." In: *ACM Transactions on Graphics (TOG)* 28.3 (2009), p. 77.

[16] Bommes, D., Lévy, B., Pietroni, N., Puppo, E., Silva, C., Tarini, M., Zorin, D., "Quad-mesh generation and processing: A survey." In: *Computer Graphics Forum*. Vol. 32. 6. Wiley Online Library. 2013, pp. 51–76.

[17] Borgo, R., Kehrer, J., Chung, D. H. S., Maguire, E., Laramee, R. S., Hauser, H., Ward, M., Chen, M., "Glyph-based Visualization: Foundations, Design Guidelines, Techniques and Applications." In: *Eurographics State of the Art Reports*. 2013, pp. 39–63.

[18] Borgo, R., Kehrer, J., Chung, D. H., Maguire, E., Laramee, R. S., Hauser, H., Ward, M., Chen, M., "Glyph-based Visualization: Foundations, Design Guidelines, Techniques and Applications." In: *Eurographics State of the Art Reports*. 2013, pp. 39–63.

[19] Borland, D., "Ambient Occlusion Opacity Mapping for Visualization of Internal Molecular Structure." In: *Journal of WSCG* 19.1 (2011), pp. 17–24.

[20] Born, S., Markl, M., Gutberlet, M., Scheuermann, G., "Illustrative Visualization of Cardiac and Aortic Blood Flow from 4D MRI Data." In: *IEEE Pacific Visualization*. 2013, pp. 129 –136.

[21] Born, S., Sündermann, S. H., Russ, C., Hopf, R., Ruiz, C. E., Falk, V., Gessat, M., "Stent maps - Comparative visualization for the prediction of adverse events of transcatheter aortic valve implantations." In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2704–2713.

[22] Boyell, R. L., Ruston, H., "Hybrid techniques for real-time radar simulation." In: *Proceedings of the November 12-14, 1963, fall joint computer conference*. ACM. 1963, pp. 445–458.

[23] Breslav, S., Szerszen, K., Markosian, L., Barla, P., Thollot, J., "Dynamic 2D patterns for shading 3D scenes." In: *ACM Transactions on Graphics (TOG)*. Vol. 26. ACM. 2007, p. 20.

[24] Bruckner, S., Gröller, E., "Enhancing depth-perception with flexible volumetric halos." In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1344–1351.

[25] Carpentier, A, Adams, D. H., Filsoufi, F, *Carpentier's Reconstructive Valve Surgery*. Maryland Heights, Missouri: Saunders/Elsevier, 2010.

[26] Carr, H., Snoeyink, J., Axen, U., "Computing contour trees in all dimensions." In: *Computational Geometry* 24.2 (2003), pp. 75–94.

[27] Chen, G., Kwatra, V., Wei, L.-Y., Hansen, C. D., Zhang, E., "Design of 2d time-varying vector fields." In: *IEEE Transactions on Visualization and Computer Graphics* 18.10 (2012), pp. 1717–1730.

[28] Chen, M., Floridi, L., "An analysis of information visualisation." In: *Synthese* 190.16 (2013), pp. 3421–3438.

[29] Choong, A., Beidas, R., Zhu, J., "Parallelizing simulated annealing-based placement using GPGPU." In: *Proceedings - 2010 International Conference on Field Programmable Logic and Applications, FPL 2010* (2010), pp. 31–34. ISSN: 1946-1488.

[30] Chu, A., Chan, W.-Y., Guo, J., Pang, W.-M., Heng, P.-A., "Perception-aware Depth Cueing for Illustrative Vascular Visualization." In: *BioMedical Engineering and Informatics, International Conference on* 1 (2008), pp. 341–346.

[31] Chung, D. H., Archambault, D., Borgo, R., Edwards, D. J., Laramee, R. S., Chen, M., "How ordered is it? On the perceptual orderability of visual channels." In: *Computer Graphics Forum*. Vol. 35. 3. Wiley Online Library. 2016, pp. 131–140.

[32] Cleary, K., Peters, T., "Image-guided interventions: technology review and clinical applications." In: *Annual Review of Biomedical Engineering* 12 (2010), pp. 119–142. ISSN: 1523-9829.

[33] Connolly, M. L., "Analytical Molecular Surface Calculation." In: *Journal of Applied Crystallography* 16.5 (1983), pp. 548–558.

[34] Corsini, M., Cignoni, P., Scopigno, R., "Efficient and flexible sampling with blue noise properties of triangular meshes." In: *IEEE transactions on visualization and computer graphics* 18.6 (2012), pp. 914–924.

[35] Crane, K., Weischedel, C., Wardetzky, M, "Geodesics in heat: A new approach to computing distance based on heat flow." In: *ACM Transactions on Graphics (TOG)* 3 (2013), p. 10.

[36] Das, A., Mukhopadhyay, C., "Application of Principal Component Analysis in Protein Unfolding: An All-Atom Molecular Dynamics Simulation Study." In: *The Journal of Chemical Physics* 127.16 (2007), 10B625.

[37] Davis, F. D., "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology." In: *Management Information Systems Quarterly* (1989), pp. 319–340.

[38] De Moura Pinto, F., Freitas, C. M., "Importance-aware composition for illustrative volume rendering." In: *Proceedings - 23rd SIBGRAPI Conference on Graphics, Patterns and Images* (2010), pp. 134–141.

[39] De Vivo, M., Masetti, M., Bottegoni, G., Cavalli, A., "Role of Molecular Dynamics and Related Methods in Drug Discovery." In: *Journal of medicinal chemistry* 59.9 (2016), pp. 4035–4061.

[40] DeCarlo, D., Finkelstein, A., Rusinkiewicz, S., Santella, A., "Suggestive Contours for Conveying Shape." In: *Proceedings of SIGGRAPH* (2003), pp. 848–855.

[41] Deanda, F., Pearlman, R. S., "A Novel Approach for Identifying the Surface Atoms of Macromolecules." In: *Journal of Molecular Graphics and Modelling* 20.5 (2002), pp. 415–425.

[42] Dong, F., Clapworthy, G. J., Lin, H., Krokos, M. A., "Nonphotorealistic Rendering of Medical Volume Data." In: *IEEE Computer Graphics and Applications* 23.4 (2003), pp. 44–52.

[43] Ebke, H.-C., Schmidt, P., Campen, M., Kobbelt, L., "Interactively controlled quad remeshing of high resolution 3D models." In: *ACM Transactions on Graphics (TOG)* 35.6 (2016), p. 218.

[44] Elmqvist, N., Assarsson, U., Tsigas, P., "Employing Dynamic Transparency for 3D Occlusion Management: Design Issues and Evaluation." In: *Human-Computer Interaction – INTERACT 2007: 11th IFIP TC 13 International Conference*. Springer Berlin Heidelberg, 2007, pp. 532–545.

[47] Floater, M. S., Hormann, K., "Surface Parameterization: a Tutorial and Survey." In: *Advances in Multiresolution for Geometric Modelling*. Springer, 2005, pp. 157–186. ISBN: 978-3-540-26808-6.

[48] Frishman, D., Argos, P., "STRIDE: Protein Secondary Structure Assignment from Atomic Coordinates." In: *Proteins* 23 (1995), pp. 455–479.

[49] Fuchs, R., Hauser, H., "Visualization of Multi-Variate Scientific Data." In: *Computer Graphics Forum* 28.6 (2009), pp. 1670–1690.

[50] Fuhrmann, A., Sobotka, G., Groß, C., "Distance fields for rapid collision detection in physically based modeling." In: *Proceedings of GraphiCon 2003*. Citeseer. 2003, pp. 58–65.

[51] Gasteiger, R., Tietjen, C., Baer, A., Preim, B., "Curvature- and Model-Based Surface Hatching of Anatomical Structures Derived from Clinical Volume Datasets." In: *Smart Graphics*. 2008, pp. 255–262.

[52] Gerig, G., Koller, T., Székely, G., Brechbühler, C., Kübler, O., "Symbolic description of 3-D structures applied to cerebral vessel tree obtained from MR angiography volume data." In: *Biennial International Conference on Information Processing in Medical Imaging*. Springer. 1993, pp. 94–111.

[53] Gibson, J. J., *The Perception Of The Visual World*. Boston: Houghton Mifflin, 1950.

[54] Glueck, M., Crane, K., Anderson, S., Rutnik, A., Khan, A., "Multiscale 3D reference visualization." In: *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. ACM. 2009, pp. 225–232.

[55] Gortler, S. J., Grzeszczuk, R., Szeliski, R., Cohen, M. F., "The lumigraph." In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM. 1996, pp. 43–54.

[56] Green, C., "Improved Alpha-tested Magnification for Vector Textures and Special Effects." In: *ACM SIGGRAPH 2007 Courses*. SIGGRAPH '07. ACM, 2007, pp. 9–18. ISBN: 978-1-4503-1823-5. URL: http://doi.acm.org/10.1145/1281500.1281665.

[57] Griffin, W., Wang, Y., Berrios, D., Olano, M., "GPU curvature estimation on deformable meshes." In: *Symposium on Interactive 3D Graphics and Games*. ACM. 2011, pp. 159–166.

[58] Grossmann, N., Köppel, T., Gröller, M. E., Raidou, R. G., "Visual Analysis of Distortions in the Projection of Biomedical Structures." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. Eurographics Association. 2018.

[59] Haber, R. B., McNabb, D. A., "Visualization idioms: A conceptual model for scientific visualization systems." In: *Visualization in scientific computing* 74 (1990), p. 93.

[60] Hahn, H. K., Preim, B., Selle, D., Peitgen, H.-O., "Visualization and interaction techniques for the exploration of vascular structures." In: *Visualization, 2001. VIS'01. Proceedings*. IEEE. 2001, pp. 395–578.

[61] Hammer, P. E., Perrin, D. P., Pedro, J, Howe, R. D., "Image-based mass-spring model of mitral valve closure for surgical planning." In: *Proc. SPIE*. Vol. 6918. International Society for Optics and Photonics. 2008, 69180Q.

[62] Hansen, C., Zidowitz, S., Hindennach, M., Schenk, A., Hahn, H., Peitgen, H.-O., "Interactive Determination of Robust Safety Margins for Oncologic Liver Surgery." In: *International Journal of Computer Assisted Radiology and Surgery* 4.5 (2009), pp. 469–474.

[63] Hansen, C., Zidowitz, S., Hindennach, M., Schenk, A., Hahn, H., Peitgen, H. O., "Interactive determination of robust safety margins for oncologic liver surgery." In: *International Journal of Computer Assisted Radiology and Surgery* 4.5 (2009), pp. 469–474.

[64] Hansen, C., Wieferich, J., Ritter, F., Rieder, C., Peitgen, H.-O., "Illustrative Visualization of 3D Planning Models for Augmented Reality in Liver Surgery." In: *International Journal of Computer Assisted Radiology and Surgery* 5.2 (2010), pp. 133–141. ISSN: 1861-6410.

[65] Hansen, C., Zidowitz, S., Ritter, F., Lange, C., Oldhafer, K., Hahn, H. K., "Risk maps for Liver Surgery." In: *International Journal of Computer Assisted Radiology and Surgery* 8.3 (2013), pp. 419–428.

[66] Hansen, C., Zidowitz, S., Preim, B., Stavrou, G., Oldhafer, K. J., Hahn, H. K., "Impact of model-based risk analysis for liver surgery planning." In: *International journal of computer assisted radiology and surgery* 9.3 (2014), pp. 473–480.

[67] Hansson, T., Oostenbrink, C., Van Gunsteren, W. F., "Molecular Dynamics Simulations." In: *Current Opinion in Structural Biology* 12.2 (2002), pp. 190–196.

[68] Hart, J. C., "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces." In: *The Visual Computer* 12.10 (1996), pp. 527–545. ISSN: 1432-2315. URL: https://doi.org/10.1007/s003710050084.

[69] Hasselgren, J., Akenine-Möller, T., Ohlsson, L., "Conservative Rasterization." In: *GPU Gems 2*. Ed. by Matt Pharr. Addison-Wesley, 2005, pp. 677–690.

[70] Hauser, H., "Generalizing focus+ context visualization." In: *Scientific visualization: The visual extraction of knowledge from data*. Springer, 2006, pp. 305–327.

[71] Healey, C., Enns, J., "Attention and visual memory in visualization and computer graphics." In: *IEEE transactions on visualization and computer graphics* 18.7 (2012), pp. 1170–1188.

[72] Hermosilla, P, Guallar, V, Vinacua, A, Vázquez, P. P., "Instant Visualization of Secondary Structures of Molecular Models." In: *Eurographics Workshop on Visual Computing for Biology and Medicine* (2015), pp. 51–60.

[73] Hernández-Hoyos, M., Anwander, A., Orkisz, M., Roux, J.-P., Douek, P., Magnin, I. E., "A Deformable Vessel Model with Single Point Initialization for Segmentation, Quantification, and Visualization of Blood Vessels in 3D MRA." In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2000, pp. 735–745.

[74] Hertzmann, A., "Introduction to 3D Non-Photorealistic Rendering : Silhouettes and Outlines." In: *Non-Photorealistic Rendering. SIGGRAPH* 99 (1999), pp. 1–14.

[75] Hertzmann, A., Zorin, D., "Illustrating smooth surfaces." In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 2000, pp. 517–526.

[77] Hormann, K, Polthier, K, Sheffer, A, *Mesh Parameterization: Theory and Practice*. 2008.

[78] Hormann, K., Greiner, G., "MIPS: An Efficient Global Parameterization Method." In: *Curve and Surface Design: Saint-Malo 1999* (2000), pp. 153–162.

[79] Huang, J., Pei, W., Wen, C., Chen, G., Chen, W., Bao, H., "Output-coherent image-space lic for surface flow visualization." In: *Visualization Symposium (PacificVis), IEEE Pacific*. 2012, pp. 137–144.

[80] Hubona, G. S., Wheeler, P. N., Shirah, G. W., Brandt, M., "The relative contributions of stereo, lighting, and background scenes in promoting 3D depth visualization." In: *ACM Transactions on Computer-Human Interaction* 6 (1999), pp. 214–242. ISSN: 10730516.

[81] Humphrey, W., Dalke, A., Schulten, K., "VMD – Visual Molecular Dynamics." In: *Journal of Molecular Graphics* 14 (1996), pp. 33–38.

[82] Interrante, V., Anderson, L., Ries, B., "Distance Perception in Immersive Virtual Environments, Revisited." In: *In proceedings of IEEE Virtual Reality*. 2006, pp. 3–10.

[83] Interrante, V., Fuchs, H., Pizer, S. M., "Conveying the 3D shape of smoothly curving transparent surfaces via texture." In: *IEEE Transactions on Visualization and Computer Graphics* 3.2 (1997), pp. 98–117.

[84] Interrante, V., Fuchs, H., Pizer, S., "Enhancing Transparent Skin Surfaces with Ridge and Valley Lines." In: *IEEE Transactions on Visualization and Computer Graphics* (1995), pp. 52–59.

[85] Jainek, W. M., Born, S., Bartz, D., Straßer, W., Fischer, J., "Illustrative Hybrid Visualization and Exploration of Anatomical and Functional Brain Data." In: *Computer Graphics Forum* 27.3 (2008), pp. 855–862.

[86] Jakob, W., Tarini, M., Panozzo, D., Sorkine-Hornung, O., "Instant Field-Aligned Meshes." In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)* 34.6 (2015).

[87] Janiga, G, Berg, P, Sugiyama, S, Kono, K, Steinman, D., "The Computational Fluid Dynamics Rupture Challenge 2013—Phase I: prediction of rupture status in intracranial aneurysms." In: *American Journal of Neuroradiology* 36.3 (2015), pp. 530–536.

[88] Jenkins, B., *Algorithm Alley: Hash Functions*. `http://www.drdobbs.com/database/algorithm-alley/184410284`. Accessed: 2017-03-31. 1997.

[89] Jones, M. W., Baerentzen, J. A., Sramek, M., "3D distance fields: a survey of techniques and applications." In: *IEEE Transactions on Visualization and Computer Graphics* 12.4 (2006), pp. 581–599. ISSN: 1077-2626.

[90] Jordan, R. A., Yasser, E.-M., Dobbs, D., Honavar, V., "Predicting Protein-Protein Interface Residues using Local Surface Structural Similarity." In: *BMC Bioinformatics* 13.1 (2012), p. 41.

[91] Judd, T., Durand, F., Adelson, E., "Apparent ridges for line drawing." In: *ACM Transactions on Graphics (TOG)* 26.3 (2007), p. 19.

[92] Kälberer, F., Nieser, M., Polthier, K., "QuadCover - Surface parameterization using branched coverings." In: *Computer Graphics Forum* 26.3 (2007), pp. 375–384.

[93] Kälberer, F., Nieser, M., Polthier, K., "Stripe parameterization of tubular surfaces." In: *Mathematics and Visualization* 202489 (2011), pp. 13–26. ISSN: 2197666X.

[94] Karampudi, N. B. R., Bahadur, R. P., "Layers: A Molecular Surface Peeling Algorithm and its Applications to Analyze Protein Structures." In: *Scientific Reports* 5 (2015).

[95] Kehrer, J., Hauser, H., "Visualization and visual analysis of multifaceted scientific data: A survey." In: *IEEE Transactions on Visualization and Computer Graphics* 19.3 (2013), pp. 495–513.

[96] Kersten-Oertel, M., Chen, S. J. S., Collins, D. L., "An evaluation of depth enhancing perceptual cues for vascular volume visualization in neurosurgery." In: *IEEE Transactions on Visualization and Computer Graphics* 20.3 (2014), pp. 391–403. ISSN: 10772626.

[97] Kersten, M., Stewart, J., Troje, N., Ellis, R., "Enhancing depth perception in translucent volumes." In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006).

[98] Kim, B., Kim, K.-J., Seong, J.-K., "GPU Accelerated Molecular Surface Computing." In: *Applied Mathematics & Information Sciences* 6 (2012), pp. 185–194.

[99] Kim, Y., Yu, J., Yu, X., Lee, S., "Line-art illustration of dynamic and specular surfaces." In: *ACM Transactions on Graphics (TOG)* 27.5 (2008), p. 156.

[100] Kindlmann, G., Westin, C.-F., "Diffusion tensor visualization with glyph packing." In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006).

[101] Kindlmann, G., Whitaker, R., Tasdizen, T., Moller, T., "Curvature-Based Transfer Functions for Direct Volume Rendering: Methods and Applications." In: *Proceedings of the 14th IEEE Visualization 2003*. 2003, pp. 67–.

[102] Kirby, R. M., Marmanis, H., Laidlaw, D. H., "Visualizing Multivalued Data from 2D Incompressible Flows Using Concepts from Painting." In: *Proceedings of Visualization'99*. IEEE. 1999, pp. 333–540.

[103] Knöppel, F., Crane, K., Pinkall, U., Schröder, P., "Stripe patterns on surfaces." In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), 39:1–39:11.

[104] Kocincová, L., Jarešová, M., Byška, J., Parulek, J., Hauser, H., Kozlíková, B., "Comparative Visualization of Protein Secondary Structures." In: *BMC Bioinformatics* 18.Suppl 2 (2017), pp. 1–12.

[105] Kozlikova, B., Krone, M., Lindow, N., Falk, M., Baaden, M., Baum, D., Viola, I., Parulek, J., Hege, H.-C., "Visualization of biomolecular structures: State of the art." In: *Eurographics Conference on Visualization (EuroVis)-STARs*. The Eurographics Association. 2015, pp. 061–081.

[106] Kraima, A, Smit, N., Jansma, D, West, N., Quirke, P, Rutten, H., Vilanova, A, Velde, C., DeRuiter, M., "62. The virtual surgical pelvis: A highly-detailed 3D pelvic model for anatomical education and surgical simulation." In: *European Journal of Surgical Oncology* 40.11 (2014), S32.

[107] Krayer, B., Müller, S., "Generating signed distance fields on the GPU with ray maps." In: *The Visual Computer* (2019).

[108] Kreiser, J., Hermosilla, P., Ropinski, T., "Void Space Surfaces to Convey Depth in Vessel Visualizations." In: *ArXiv e-prints* (2018). arXiv: 1806.07729 [cs.GR].

[109] Kreiser, J, Meuschke, M, Mistelbauer, G, Preim, B, Ropinski, T, "A Survey of Flattening-Based Medical Visualization Techniques." In: *Computer Graphics Forum*. Vol. 37. 3. Wiley Online Library. 2018, pp. 597–624.

[110] Kretschmer, J., Godenschwager, C., Preim, B., Stamminger, M., "Interactive patient-specific vascular modeling with sweep surfaces." In: *IEEE transactions on visualization and computer graphics* 19.12 (2013), pp. 2828–2837.

[111] Krieger, E., *Yet Another Scientific Artificial Reality Application*. 1993. URL: www.yasara.org.

[112] Krone, M., Bidmon, K., Ertl, T., "GPU-based Visualisation of Protein Secondary Structure." In: *Theory and Practice of Computer Graphics* 8 (2008), pp. 115–122.

[113] Krone, M., Bidmon, K., Ertl, T., "Interactive Visualization of Molecular Surface Dynamics." In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1391–1398.

[114] Krone, M., Grottel, S., Ertl, T., "Parallel Contour-Buildup Algorithm for the Molecular Surface." In: *Biological Data Visualization*. IEEE. 2011, pp. 17–22.

[115] Krone, M., Frieß, F., Scharnowski, K., Reina, G., Fademrecht, S., Kulschewski, T., Pleiss, J., Ertl, T., "Molecular Surface Maps." In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 701–710.

[116] Kruijff, E., Swan II, J. E., Feiner, S., "Perceptual Issues in Augmented Reality Revisited." In: *Proc. of IEEE International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2010, pp. 3–12.

[117] Laan, W. J., Green, S., Sainz, M., "Screen space fluid rendering with curvature flow." In: *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. ACM. 2009, pp. 91–98.

[118] Lai, Y.-K., Jin, M., Xie, X., He, Y., Palacios, J., Zhang, E., Hu, S.-M., Gu, X., "Metric-driven rosy field design and remeshing." In: *IEEE Transactions on Visualization and Computer Graphics* 16.1 (2010), pp. 95–108.

[119] Laidlaw, D. H., Ahrens, E. T., Kremers, D., Avalos, M. J., Jacobs, R. E., Readhead, C., "Visualizing Diffusion Tensor Images of the Mouse Spinal Cord." In: *Proceedings of the Conference on Visualization '98*. Research Triangle Park, North Carolina, USA, 1998, pp. 127–134.

[120] Lamata, P., Lamata, F., Sojar, V., Makowski, P., Massoptier, L., Casciaro, S., Ali, W., Stüdeli, T., Declerck, J., Elle, O. J., "Use of the Resection Map system as guidance during hepatectomy." In: *Surgical endoscopy* 24.9 (2010), pp. 2327–2337.

[121] Lawonn, K., Moench, T., Preim, B., "Streamlines for illustrative real-time rendering." In: *Computer Graphics Forum* 32.3 (2013), pp. 321–330. ISSN: 01677055.

[122] Lawonn, K., Luz, M., Preim, B., Hansen, C., "Illustrative Visualization of Vascular Models for Static 2D Representations." In: *Proceedings of Medical Image Computing and Computer-Assisted Intervention (MICCAI)* 9350.Pt 2 (2015), pp. 399–406.

[123] Lawonn, K, Smit, N., Bühler, K, Preim, B, "A Survey on Multimodal Medical Data Visualization." In: *Computer Graphics Forum*. Vol. 37. 1. Wiley Online Library. 2018, pp. 413–438.

[124] Lawonn, K., Luz, M., Hansen, C., "Improving spatial perception of vascular models using supporting anchors and illustrative visualization." In: *Computers and Graphics* 63 (2017), pp. 37–49.

[125] Lawonn, K., Mönch, T., Preim, B., "Streamlines for Illustrative Real-Time Rendering." In: *Computer Graphics Forum*. Vol. 32. 3pt3. Wiley Online Library. 2013, pp. 321–330.

[126] Lawonn, K., Preim, B., "Feature Lines for Illustrating Medical Surface Models: Mathematical Background and Survey." In: *Visualization in Medicine and Life Sciences III* (2015), to appear.

[127] Lawonn, K., Krone, M., Ertl, T., Preim, B., "Line Integral Convolution for Real-Time Illustration of Molecular Surface Shape and Salient Regions." In: *Computer Graphics Forum* 33.3 (2014), pp. 181–190.

[128] Lawonn, K., Viola, I., Preim, B., Isenberg, T., "A Survey of Surface-Based Illustrative Rendering for Visualization." In: *Computer Graphics Forum*. Wiley Online Library. 2018.

[129] Lee, B., Richards, F. M., "The Interpretation of Protein Structures: Estimation of Static Accessibility." In: *Journal of Molecular Biology* 55.3 (1971), pp. 379–400.

[130] Lee, L. W., Bargiela, A., "Protein Surface Atoms Extraction: Voxels as an Investigative Tool." In: *Engineering Letters* 20.3 (2012).

[131] Lefebvre, S., Hoppe, H., "Appearance-space texture synthesis." In: *ACM Transactions on Graphics (TOG)* 25.3 (2006), p. 541.

[141] Lindow, N., Baum, D., Hege, H. C., "Ligand Excluded Surface: A new Type of Molecular Surface." In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2486–2495.

[142] Lindow, N., Baum, D., Prohaska, S., Hege, H. C., "Accelerated Visualization of Dynamic Molecular Surfaces." In: *Computer Graphics Forum* 29.3 (2010), pp. 943–952.

[143] Lins, L., Thomas, A., Brasseur, R., "Analysis of Accessible Surface of Residues in Proteins." In: *Protein Science* 12.7 (2003), pp. 1406–1417.

[144] Maillot, J., Yahia, H., Verroust, A., "Interactive Texture Mapping." In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (1993), pp. 27–34.

[145] Malzahn, J, Kozlíková, B, Ropinski, T, "Protein Tunnel Reprojection." In: *Eurographics Workshop on Visual Computing for Biology and Medicine* (2017), pp. 1–10.

[146] Marcias, G., Pietroni, N., Panozzo, D., Puppo, E., Sorkine-Hornung, O., "Animation-Aware Quadrangulation." In: *Computer Graphics Forum (proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing)* 32.5 (2013), pp. 167–175.

[147] Marroquim, R., Kraus, M., Cavalcanti, P. R., "Efficient Point-Based Rendering Using Image Reconstruction." In: *SPBG*. 2007, pp. 101–108.

[148] McLoughlin, T., Laramee, R. S., Peikert, R., Post, F. H., Chen, M., "Over Two Decades of Integration-Based, Geometric Flow Visualization." In: *Computer Graphics Forum* 29.6 (2010), pp. 1807–1829.

[149] Meuschke, M., Voß, S., Beuing, O., Preim, B., Lawonn, K., "Glyph-Based Comparative Stress Tensor Visualization in Cerebral Aneurysms." In: *Computer Graphics Forum* 36.3 (2017), pp. 99–108. ISSN: 1467-8659.

[152] Meuschke, M., Voß, S., Preim, B., Lawonn, K., "Exploration of blood flow patterns in cerebral aneurysms during the cardiac cycle." In: *Computers & Graphics* 72 (2018), pp. 12 –25. ISSN: 0097-8493.

[153] Meyer-Spradow, J., Stegger, L., Döring, C., Ropinski, T., Hinrichs, K., "Glyph-based SPECT visualization for the diagnosis of coronary artery disease." In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1499–1506. ISSN: 10772626.

[154] Meyer, J., Thomas, J., Diehl, S., Fisher, B., Keim, D. A., "From Visualization to Visually Enabled Reasoning." In: *Scientific Visualization: Advanced Concepts*. Ed. by Hans Hagen. Vol. 1. Dagstuhl Follow-Ups. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010, pp. 227–245.

[155] Milnor, J. W., Spivak, M., Wells, R., *Morse theory*. Princeton university press, 1963.

[156] Munson, M., Balasubramanian, S., Fleming, K. G., Nagi, A. D., O'Brien, R., Sturtevant, J. M., Regan, L., "What Makes a Protein a Protein? Hydrophobic Core Designs that Specify Stability and Structural Properties." In: *Protein Science* 5.8 (1996), pp. 1584–1593.

[157] Myles, A, Pietroni, N, Zorin, D, "Robust field-aligned global parametrization." In: *ACM Transactions on Graphics (TOG)* 33 (2014), pp. 1–14.

[158] Naumov, M, Castonguay, P, Cohen, J, *Parallel graph coloring with applications to the incomplete-LU factorization on the GPU*. Tech. rep. NVIDIA, Tech. Rep, 2015.

[159] Nguyen, T. C., Itoh, A., Carlhäll, C. J., Bothe, W., Timek, T. A., Ennis, D. B., Oakes, R. A., Liang, D., Daughters, G. T., Ingels Jr, N. B., "The effect of pure mitral regurgitation on mitral annular geometry and three-dimensional saddle shape." In: *The Journal of thoracic and cardiovascular surgery* 136.3 (2008), pp. 557–565.

[160] Oeltze-Jafra, S, Meuschke, M., Neugebauer, M., Saalfeld, S., Lawonn, K., Janiga, G., Hege, H.-C., Zachow, S., Preim, B., "Generation and Visual Exploration of Medical Flow Data: Survey, Research Trends and Future Challenges." In: *Computer Graphics Forum*. Wiley Online Library. 2018.

[161] Oeltze, S., Preim, B., "Visualization of vasculature with convolution surfaces: method, validation and evaluation." In: *IEEE Transactions on Medical Imaging* 24.4 (2005), pp. 540–548.

[162] Oeltze, S., Hennemuth, A., Glasser, S., Kühnel, C., Preim, B., "Glyph-Based Visualization of Myocardial Perfusion Data and Enhancement with Contractility and Viability Information." In: *Eurographics Workshop on Visual Computing for Biomedicine*. The Eurographics Association, 2008, pp. 11–20.

[163] Ohtake, Y., Belyaev, A., Seidel, H.-P., "Ridge-Valley Lines on Meshes via Implicit Surface Fitting." In: *Proceedings of SIGGRAPH* 23 (2004), pp. 609–612.

[164] Ohtake, Y., Belyaev, A., Alexa, M., Alexa, M., Turk, G., Seidel, H.-P., "Multi-level Partition of Unity Implicits." In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. ACM, 2003, pp. 463–470. ISBN: 1-58113-709-5. URL: http://doi.acm.org/10.1145/1201775.882293.

[165] Palacios, J., Zhang, E., "Interactive visualization of rotational symmetry fields on surfaces." In: *IEEE transactions on visualization and computer graphics* 17.7 (2011), pp. 947–955.

[166] Parulek, J., Ropinski, T., Viola, I., "Seamless Visual Abstraction of Molecular Surfaces." In: *Spring Conference*. ACM, 2013, pp. 107–114.

[167] Pascucci, V., Scorzelli, G., Bremer, P.-T., Mascarenhas, A., "Robust on-line computation of Reeb graphs: simplicity and speed." In: *Acm transactions on graphics (tog)*. Vol. 26. 3. ACM. 2007, p. 58.

[168] Pelt, R., Bartroli, A. V., Wetering, H., "Illustrative volume visualization using GPU-based particle systems." In: *IEEE transactions on visualization and computer graphics* 16.4 (2010), pp. 571–582.

[169] Perlin, K., Hoffert, E. M., "Hypertexture." In: *ACM Siggraph Computer Graphics*. Vol. 23. 3. ACM. 1989, pp. 253–262.

[170] Plotly Technologies Inc. *Collaborative Data Science*. 2015. URL: https://plot.ly.

[171] Praun, E., Finkelstein, A., Hoppe, H., "Lapped textures." In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH oo* 1 (2000), pp. 465–470.

[172] Praun, E., Hoppe, H., Webb, M., Finkelstein, A., "Real-time hatching." In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, p. 581.

[173] Preim, B., Ropinski, T., Isenberg, P., "A Critical Analysis of the Evaluation Practice in Medical Visualization." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2018.

[174] Preim, B., Saalfeld, P., "A survey of virtual human anatomy education systems." In: *Computers & Graphics* 71 (2018), pp. 132–153.

[175] Preim, B., Baer, A., Cunningham, D., Isenberg, T., Ropinski, T., "A survey of perceptually motivated 3d visualization of medical image data." In: *Computer Graphics Forum*. Vol. 35. 3. Wiley Online Library. 2016, pp. 501–525.

[176] Ray, N., Li, W. C., Lévy, B., Sheffer, A., Alliez, P., "Periodic global parameterization." In: *ACM Transactions on Graphics (TOG)* 25.4 (2006), pp. 1460–1485.

[177] Reeb, G., "Sur les points singuliers d'une forme de pfaff completement integrable ou d'une fonction numerique [on the singular points of a completely integrable pfaff form or of a numerical function]." In: *Comptes Rendus Acad. Sciences Paris* 222 (1946), pp. 847–849.

[178] Reichelt, S., Häussler, R., Fütterer, G., Leister, N., "Depth cues in human visual perception and their realization in 3D displays." In: *Proc. SPIE*. Vol. 7690. 2010, 76900B–76900B–12.

[179] Richards, F. M., "Areas, Volumes, Packing, and Protein Structure." In: *Annual Review of Biophysics and Bioengineering* 6.1 (1977), pp. 151–176.

[180] Richardson, J. S., "The Anatomy and Taxonomy of Protein Structure." In: *Advances in Protein Chemistry* 34 (1981), pp. 167–339.

[181] Rieder, C., Ritter, F., Raspe, M., Peitgen, H. O., "Interactive visualization of multimodal volume data for neurosurgical tumor treatment." In: *Computer Graphics Forum* 27.3 (2008), pp. 1055–1062.

[182] Rim, Y., McPherson, D. D., Kim, H., "Effect of leaflet-to-chordae contact interaction on computational mitral valve evaluation." In: *Biomedical engineering online* 13.1 (2014), p. 31.

[183] Rim, Y., Choi, A., McPherson, D. D., Kim, H., "Personalized computational modeling of mitral valve prolapse: virtual leaflet resection." In: *PloS one* 10.6 (2015), e0130906.

[184] Ritter, F., Hansen, C., Dicken, V., Konrad, O., Preim, B., Peitgen, H.-O., "Real-time illustration of vascular structures." In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 877–884.

[185] Rocha, A., Alim, U., Silva, J. D., Sousa, M. C., "Decal-maps: Real-Time Layering of Decals on Surfaces for Multivariate Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 1 (2017), pp. 821–830.

[186] Rodrigues, J. F., Traina, A. J., Oliveira, M. C. F., Traina, C, "Reviewing data visualization: an analytical taxonomical study." In: *Tenth International Conference on Information Visualisation (IV'06)*. IEEE. 2006, pp. 713–720.

[187] Rong, G., Tan, T.-S., "Jump flooding in GPU with applications to Voronoi diagram and distance transform." In: *Proceedings of the 2006 symposium on Interactive 3D graphics and games - SI3D '06* (2006), p. 109.

[188] Ropinski, T., Oeltze, S., Preim, B., "Survey of glyph-based visualization techniques for spatial multivariate medical data." In: *Computers and Graphics* 35.2 (2011), pp. 392–401.

[189] Ropinski, T., Steinicke, F., Hinrichs, K., "Visually Supporting Depth Perception in Angiography Imaging." In: *Smart Graphics: 6th International Symposium, SG 2006*. 2006, pp. 93–104.

[190] Ropinski, T., Specht, M., Meyer-Spradow, J., Hinrichs, K., Preim, B., "Surface glyphs for visualizing multimodal volume data." In: *Proceedings of the 12th International Fall Workshop on Vision Modeling and Visualization* D (2007), pp. 3–12.

[191] Rosenfeld, A., *Multiresolution image processing and analysis*. Vol. 12. Springer Science & Business Media, 2013.

[192] Roy, L., Kumar, P., Golbabaei, S., Zhang, Y., Zhang, E., "Interactive Design and Visualization of Branched Covering Spaces." In: *IEEE Transactions on Visualization and Computer Graphics* 2626.c (2017).

[193] Rusinkiewicz, S., "Estimating Curvatures and Their Derivatives on Triangle Meshes." In: *Symposium on 3D Data Processing, Visualization, and Transmission*. Sept. 2004.

[194] Saalfeld, P., Stojnic, A., Preim, B., Oeltze-Jafra, S., "Semi-Immersive 3D Sketching of Vascular Structures for Medical Education." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. 2016, pp. 123–132.

[195] Saalfeld, P., Luz, M., Berg, P., Preim, B., Saalfeld, S., "Guidelines for quantitative evaluation of medical visualizations on the example of 3d aneurysm surface comparisons." In: *Computer Graphics Forum*. Vol. 37. 1. Wiley Online Library. 2018, pp. 226–238.

[196] Sanner, M. F., Olson, A. J., Spehner, J.-C., "Reduced Surface: An Efficient Way to Compute Molecular Surfaces." In: *Biopolymers* 38.3 (1996), pp. 305–320.

[197] Saunders, C. T., Baker, D., "Evaluation of Structural and Evolutionary Contributions to Deleterious Mutation Prediction." In: *Journal of Molecular Biology* 322.4 (2002), pp. 891–901.

[198] Schertler, N., Tarini, M., Jakob, W., Kazhdan, M., Gumhold, S., Panozzo, D., "Field-aligned Online Surface Reconstruction." In: *ACM Transactions on Graphics (TOG)* 36.4 (July 2017), 77:1–77:13. ISSN: 0730-0301.

[199] Schroeder, D., Keefe, D. F., "Visualization-by-Sketching: An Artist's Interface for Creating Multivariate Time-Varying Data Visualizations." In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2016), pp. 877–885.

[200] Schulz, H.-J., Angelini, M., Santucci, G., Schumann, H., "An enhanced visualization process model for incremental visualization." In: *IEEE transactions on visualization and computer graphics* 22.7 (2016), pp. 1830–1842.

[201] Schumann, C., Oeltze, S., Bade, R., Preim, B., Peitgen, H.-O., "Model-free surface visualization of vascular trees." In: *EuroVis*. Citeseer. 2007, pp. 283–290.

[202] Sheffer, A., Praun, E., Rose, K., "Mesh parameterization methods and their applications." In: *Foundations and Trends® in Computer Graphics and Vision* 2.2 (2007), pp. 105–171.

[203] Shepard, D., "A two-dimensional interpolation function for irregularly-spaced data." In: *Proceedings of the 1968 23rd ACM national conference*. ACM. 1968, pp. 517–524.

[204] Shinagawa, Y., Kunii, T. L., "Constructing a Reeb graph automatically from cross sections." In: *IEEE Computer Graphics and Applications* 6 (1991), pp. 44–51.

[205] Singh, R. V. P., Namboodiri, A. M., "Efficient texture mapping by homogeneous patch discovery." In: *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing - ICVGIP '12* (2012), pp. 1–8.

[206] Son, M., Lee, Y., Kang, H., Lee, S., "Structure grid for directional stippling." In: *Graphical Models* 73.3 (2011), pp. 74–87.

[207] Steenblik, R. A., "The Chromostereoscopic Process: A Novel Single Image Stereoscopic Process." In: *Proc. SPIE*. Vol. 0761. 1987, pp. 27–34.

[208] Steinicke, F., Bruder, G., Hinrichs, K. H., Steed, A., "Gradual Transitions and their Effects on Presence and Distance Estimation." In: *Computers & Graphics* 34.1 (2010), pp. 26–33.

[209] Strickler, S. S., Gribenko, A. V., Gribenko, A. V., Keiffer, T. R., Tomlinson, J., Reihle, T., Loladze, V. V., Makhatadze, G. I., "Protein Stability and Surface Electrostatics: A Charged Relationship." In: *Biochemistry* 45.9 (2006), pp. 2761–2766.

[210] Svakhine, N. A., Ebert, D. S., Andrews, W. M., "Illustration-Inspired Depth Enhanced Volumetric Medical Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 15.1 (2009), pp. 77–86.

[211] Svetachov, P., Everts, M. H., Isenberg, T., "DTI in Context: Illustrating Brain Fiber Tracts In Situ." In: *Computer Graphics Forum* 29.3 (2010), pp. 1023–1032.

[212] Swan, J. E., Singh, G., Ellis, S. R., "Matching and Reaching Depth Judgments with Real and Augmented Reality Targets." In: *IEEE Transactions on Visualization and Computer Graphics* 21.11 (2015), pp. 1289–1298.

[213] Tan, K. P., Nguyen, T. B., Patel, S., Varadarajan, R., Madhusudhan, M. S., "Depth: A Web Server to Compute Depth, Cavity Sizes, Detect Potential Small-Molecule Ligand-Binding Cavities and Predict the pKa of Ionizable Iesidues in Proteins." In: *Nucleic Acids Research* 41.Web Server issue (2013), pp. 314–321.

[214] Tarini, M., Cignoni, P., Montani, C., "Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 1237–1244.

[215] Tarini, M., Hormann, K., Cignoni, P., Montani, C., "Polycube-maps." In: *ACM transactions on graphics (TOG)* 23.3 (2004), pp. 853–860.

[216] The Qt Company, *Complete Software Development Framework - Qt*. 2017. URL: https://www.qt.io/.

[217] Tietjen, C., Isenberg, T., Preim, B., "Combining Silhouettes, Surface, and Volume Rendering for Surgery Education and Planning." In: *Computer Graphics Forum*. 2005, pp. 303–310.

[218] Tietze, A. A., Tietze, D., Ohlenschläger, O., Leipold, E., Ullrich, F., Kühl, T., Mischo, A., Buntkowsky, G., Görlach, M., Heinemann, S. H., "Structurally Diverse μ-Conotoxin PIIIA Isomers Block Sodium Channel NaV1. 4." In: *Angewandte Chemie International Edition* 51.17 (2012), pp. 4058–4061.

[219] Tosti, E., Boni, R., Gallo, A., "μ-Conotoxins Modulating Sodium Currents in Pain Perception and Transmission: A Therapeutic Potential." In: *Marine Drugs* 15.10 (2017), p. 295.

[220] Totrov, M, Abagyan, R, "The Contour-Buildup Algorithm to Calculate the Analytical Molecular Surface." In: *Journal of Structural Biology* 116.1 (1996), pp. 138–143.

[221] Turk, G., "Texture Synthesis on Surfaces." In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. ACM, 2001, pp. 347–354.

[222] Tutte, W. T., "Convex Representations of Graphs." In: *Proceedings of the London Mathematical Society* 10 (1960), pp. 304–320.

[223] Van Der Zwan, M., Lueks, W., Bekker, H., Isenberg, T., "Illustrative molecular visualization with continuous abstraction." In: *Computer Graphics Forum* 30.3 (2011), pp. 683–690.

[224] Van Pelt, R., Gasteiger, R., Lawonn, K., Meuschke, M., Preim, B., "Comparative blood flow visualization for cerebral aneurysm treatment assessment." In: *Computer Graphics Forum* 33.3 (2014), pp. 131–140.

[225] Vasilakis, A. A., Fudos, I., "k+-buffer: Fragment Synchronized k-buffer." In: *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 2014, pp. 143–150.

[226] Vasilakis, A. A., Papaioannou, G., Fudos, I., "K-buffer: An efficient, memory-friendly and dynamic K-buffer framework." In: *IEEE Transactions on Visualization and Computer Graphics* 21.6 (2015), pp. 688–700.

[227] Vaxman, A., Campen, M., Diamanti, O., Panozzo, D., Bommes, D., Hildebrandt, K., Ben-Chen, M., "Directional field synthesis, design, and processing." In: *Computer Graphics Forum*. Vol. 35. 2. Wiley Online Library. 2016, pp. 545–572.

[228] Ward, M. O., "A Taxonomy of Glyph Placement Strategies for Multidimensional Data Visualization." In: *Information Visualization* 1.3-4 (2002), pp. 194–210.

[229] Weber, J. R., "ProteinShader: illustrative rendering of macromolecules." In: *BMC Structural Biology* 9:19 (May 2009).

[230] Wright, D., "Dynamic occlusion with signed distance fields." In: *ACM SIGGRAPH*. 2015.

[231] Xie, X., He, Y., Tian, F., Seah, H.-S., Gu, X., Qin, H., "An effective illustrative visualization framework based on photic extremum lines (PELs)." In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1328–1335.

[232] Xu, D., Li, H., Zhang, Y., "Protein Depth Calculation and the Use for Improving Accuracy of Protein Fold Recognition." In: *Journal of Computational Biology* 20.10 (2013), pp. 805–816.

[233] Zander, J., Isenberg, T., Schlechtweg, S., Strothotte, T., "High Quality Hatching." In: *Computer Graphics Forum* 23.3 (2004), pp. 421–430.

[234] Zhang, F., Kanik, J., Mansi, T., Voigt, I., Sharma, P., Ionasec, R. I., Subrahmanyan, L., Lin, B. A., Sugeng, L., Yuh, D., "Towards patient-specific modeling of mitral valve repair: 3D transesophageal echocardiography-derived parameter estimation." In: *Medical image analysis* 35 (2017), pp. 599–609.

[235] Zhang, J., Shi, Z., "Triangulation of Molecular Surfaces based on Extracting Surface Atoms." In: *Computers & Graphics* 38 (2014), pp. 291–299.

[236] Zhang, L., He, Y., Seah, H. S., "Real-time computation of photic extremum lines (PELs)." In: *The Visual Computer* 26.6-8 (2010), pp. 399–407.

[237] Zhang, L., He, Y., Xia, J., Xie, X., Chen, W., "Real-Time Shape Illustration Using Laplacian Lines." In: *IEEE Transactions on Visualization and Computer Graphics* 17 (2011), pp. 993–1006.

[238] Zwan, M., Lueks, W., Bekker, H., Isenberg, T., "Illustrative Molecular Visualization with Continuous Abstraction." In: *Computer Graphics Forum* 30.3 (2011), pp. 683–690.

[45] Engelhardt, S., **Lichtenberg, N.**, Al-Maisary, S., De Simone, R., Rauch, H., Roggenbach, J., Müller, S., Karck, M., Meinzer, H.-P., Wolf, I., "Towards Automatic Assessment of the Mitral Valve Coaptation Zone from 4D Ultrasound." In: *Functional Imaging and Modeling of the Heart*. Vol. 9126. Springer, 2015, pp. 137–145. DOI: 10.1007/978-3-319-20309-6.

[46] Eulzer, P., **Lichtenberg, N.**, Arif, R., Brcic, A., Karck, M., Lawonn, K., De Simone, R., Engelhardt, S., "Mitral Valve Quantification at a Glance." In: *Bildverarbeitung für die Medizin 2019*. Springer, 2019, pp. 296–301. DOI: 10.1007/978-3-658-25326-4.

[76] Hombeck, J. N., **Lichtenberg, N.**, Lawonn, K., "Evaluation of Spatial Perception in Virtual Reality within a Medical Context." In: *Bildverarbeitung für die Medizin 2019*. Springer, 2019, pp. 283–288. DOI: 10.1007/978-3-658-25326-4.

[132] **Lichtenberg, N.**, Hansen, C., Lawonn, K., "Concentric Circle Glyphs for Enhanced Depth-Judgment in Vascular Models." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2017, pp. 178–188. DOI: 10.2312/vcbm.20171252.

[133] **Lichtenberg, N.**, Lawonn, K., "Parameterization and feature extraction for the visualization of tree-like structures." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. Eurographics Association. 2018, pp. 145–155. DOI: 10.2312/vcbm.20181240.

[134] **Lichtenberg, N.**, Lawonn, K., "Auxiliary Tools for Enhanced Depth Perception in Vascular Structures." In: *Biomedical Visualisation*. Ed. by Paul M. Rea. Springer International Publishing, 2019, pp. 103–113. DOI: 10.1007/978-3-030-14227-8.

[135] **Lichtenberg, N.**, Lawonn, K., "Parameterization, Feature Extraction and Binary Encoding for the Visualization of Tree-Like Structures." In: *Computer Graphics Forum* (2019). DOI: 10.1111/cgf.13888.

[136] **Lichtenberg, N.**, Smit, N., Hansen, C., Lawonn, K., "Sline: Seamless Line Illustration for Interactive Biomedical Visualization." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2016, pp. 133–142. DOI: 10.2312/vcbm.20161281.

[137] **Lichtenberg, N.**, Menges, R., Ageev, V., George, A. P., Heimer, P., Imhof, D., Lawonn, K., "Analyzing Residue Surface Proximity to Interpret Molecular Dynamics." In: *Computer Graphics Forum*. Vol. 37. 3. Wiley Online Library. 2018, pp. 379–390. DOI: 10.1111/cgf.13427.

[138] **Lichtenberg, N.**, Smit, N., Hansen, C., Lawonn, K., "Real-time field aligned stripe patterns." In: *Computers & Graphics* 74 (2018), pp. 137–149. DOI: 10.1016/j.cag.2018.04.008.

[139] **Lichtenberg, N.**, Krayer, B., Hansen, C., Müller, S., Lawonn, K., "Distance Field Visualization and 2D Abstraction of Vessel Tree Structures with on-the-fly Parameterization." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2019. DOI: 10.2312/vcbm.20191251.

[140] **Lichtenberg, N.**, Eulzer, P., Romano, G., Brčić, A., Karck, M., Lawonn, K., De Simone, R., Engelhardt, S., "Mitral valve flattening and parameter mapping for patient-specific valve diagnosis." In: *International Journal of Computer Assisted Radiology and Surgery* (2020). DOI: 10.1007/s11548-019-02114-w.

[150] Meuschke, M., Smit, N.**, Lichtenberg, N.**, Preim, B., Lawonn, K., "Automatic Generation of Web-Based User Studies to Evaluate Depth Perception in Vascular Surface Visualizations." In: *Proceedings of the Eurographics Workshop on Visual Computing for Biology and Medicine*. Vol. 3. 2018. DOI: 10.2312/vcbm.20181227.

[151] Meuschke, M., Smit, N. N.**, Lichtenberg, N.**, Preim, B., Lawonn, K., "EvalViz – Surface visualization evaluation wizard for depth and shape perception tasks." In: *Computers & Graphics* (2019). DOI: https://doi.org/10.1016/j.cag.2019.05.022.

ERKLÄRUNG

Hiermit erkläre ich gemäß § 10 Abs. 3 Punkt 4 der Promotionsordnung des Fachbereichs 4: Informatik der Universität Koblenz-Landau,

- dass ich die vorliegende Dissertation mit dem Titel *Abstraction of bio-medical surface data for enhanced comprehension and analysis* selbst angefertigt und alle benutzten Hilfsmittel in der Arbeit angegeben habe,
- dass ich die Dissertation oder Teile der Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht haben, und
- dass ich weder diese noch eine andere Abhandlung bei einer anderen Hochschule als Dissertation eingereicht habe,
- dass ich meine individuellen Beiträge kooperativ an kooperativ erzielten Forschungsergebnissen in der Dissertation an den entsprechenden Stellen gekennzeichnet habe und dass meine Koautoren diese Einschätzung meines Beitrages teilen (siehe gesonderte Bestätigung der Koautoren nach § 10 Abs. 4).

Koblenz, den _____

_____

Nils Lichtenberg

CURRICULUM VITÆ

NILS LICHTENBERG

PERSONAL INFORMATION

Born in Germany, 06 April 1989

Email lichtenberg.nils@gmail.com

WORK EXPERIENCE

04/2015–07/2019 PhD student and Research Associate

Conducting lecture exercises and supervising final theses. Worked on thesis related DFG-project — *University of Koblenz-Landau*

05/2014–01/2015 Research Assistant

Worked on Master's Thesis — DKFZ, *Heidelberg*

07/2012–07/2013 Research Assistant

Supported teaching and lab demonstrations for the Computer Graphics Group — *University of Koblenz-Landau*

07/2011–12/2011 Research Assistant

Supported project work for the Image Recognition Lab — *University of Koblenz-Landau*

EDUCATION

05/2014–01/2015 DKFZ, Heidelberg

Thesis supervised by the Medical and Biological Informatics department of the German Cancer Research Center. Thesis title: *Semi-Automatic Segmentation of the Mitral Valve Leaflets on 4D Ultrasound Images* — *Master of Science, Thesis*

08/2013–12/2013 University of Georgia, USA

DAAD student exchange programme — *Exchange semester*

10/2012–03/2015 University of Koblenz-Landau

Grade 1.2 — *Master of Science*

04/2009–09/2012 University of Koblenz-Landau

Grade 2.0 — *Bachelor of Science*

Engelhardt, S., **Lichtenberg, N.**, Al-Maisary, S., De Simone, R., Rauch, H., Roggenbach, J., Müller, S., Karck, M., Meinzer, H.-P., Wolf, I., "Towards Automatic Assessment of the Mitral Valve Coaptation Zone from 4D Ultrasound." In: *Functional Imaging and Modeling of the Heart*. Vol. 9126. Springer, 2015, pp. 137–145. DOI: 10.1007/978-3-319-20309-6.

Eulzer, P., **Lichtenberg, N.**, Arif, R., Brcic, A., Karck, M., Lawonn, K., De Simone, R., Engelhardt, S., "Mitral Valve Quantification at a Glance." In: *Bildverarbeitung für die Medizin 2019*. Springer, 2019, pp. 296–301. DOI: 10.1007/978-3-658-25326-4.

Hombeck, J. N., **Lichtenberg, N.**, Lawonn, K., "Evaluation of Spatial Perception in Virtual Reality within a Medical Context." In: *Bildverarbeitung für die Medizin 2019*. Springer, 2019, pp. 283–288. DOI: 10.1007/978-3-658-25326-4.

Lichtenberg, N., Hansen, C., Lawonn, K., "Concentric Circle Glyphs for Enhanced Depth-Judgment in Vascular Models." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2017, pp. 178–188. DOI: 10.2312/vcbm.20171252.

Lichtenberg, N., Lawonn, K., "Parameterization and feature extraction for the visualization of tree-like structures." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. Eurographics Association. 2018, pp. 145–155. DOI: 10.2312/vcbm.20181240.

Lichtenberg, N., Lawonn, K., "Auxiliary Tools for Enhanced Depth Perception in Vascular Structures." In: *Biomedical Visualisation*. Ed. by Paul M. Rea. Springer International Publishing, 2019, pp. 103–113. DOI: 10.1007/978-3-030-14227-8.

Lichtenberg, N., Lawonn, K., "Parameterization, Feature Extraction and Binary Encoding for the Visualization of Tree-Like Structures." In: *Computer Graphics Forum* (2019). DOI: 10.1111/cgf.13888.

Lichtenberg, N., Smit, N., Hansen, C., Lawonn, K., "Sline: Seamless Line Illustration for Interactive Biomedical Visualization." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2016, pp. 133–142. DOI: 10.2312/vcbm.20161281.

Lichtenberg, N., Menges, R., Ageev, V., George, A. P., Heimer, P., Imhof, D., Lawonn, K., "Analyzing Residue Surface Proximity to Interpret Molecular Dynamics." In: *Computer Graphics Forum*. Vol. 37. 3. Wiley Online Library. 2018, pp. 379–390. DOI: 10.1111/cgf.13427.

Lichtenberg, N., Smit, N., Hansen, C., Lawonn, K., "Real-time field aligned stripe patterns." In: *Computers & Graphics* 74 (2018), pp. 137–149. DOI: 10.1016/j.cag.2018.04.008.

Lichtenberg, N., Krayer, B., Hansen, C., Müller, S., Lawonn, K., "Distance Field Visualization and 2D Abstraction of Vessel Tree Structures with on-the-fly Parameterization." In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2019. DOI: 10.2312/vcbm.20191251.

Lichtenberg, N., Eulzer, P., Romano, G., Brčić, A., Karck, M., Lawonn, K., De Simone, R., Engelhardt, S., "Mitral valve flattening and parameter mapping for patient-specific valve diagnosis." In: *International Journal of Computer Assisted Radiology and Surgery* (2020). DOI: 10.1007/s11548-019-02114-w.

Meuschke, M., Smit, N.**, Lichtenberg, N.**, Preim, B., Lawonn, K., "Automatic Generation of Web-Based User Studies to Evaluate Depth Perception in Vascular Surface Visualizations." In: *Proceedings of the Eurographics Workshop on Visual Computing for Biology and Medicine*. Vol. 3. 2018. DOI: 10.2312/vcbm.20181227.

Meuschke, M., Smit, N. N.**, Lichtenberg, N.**, Preim, B., Lawonn, K., "EvalViz – Surface visualization evaluation wizard for depth and shape perception tasks." In: *Computers & Graphics* (2019). DOI: https://doi.org/10.1016/j.cag.2019.05.022.

QUALIFICATIONS

C++, OpenGL, GLSL, Matlab	*Computer Skills*
Advanced English (C1), native German	*Language Skills*

OTHER INFORMATION

2018 · Honorable Mention Award at VCBM 2018 for *Parameterization and Feature Extraction for the Visualization of Tree-like Structures.*	*Awards*