Eurographics Workshop on Visual Computing for Biology and Medicine (2018)
A. Puig Puig, T. Schultz, and A. Vilanova (Editors)

Parameterization and Feature Extraction for the Visualization of Tree-like Structures

N. Lichtenberg and K. Lawonn

Institute for Computational Visualistics, University of Koblenz-Landau, Koblenz, Germany

Abstract

The study and visualization of vascular structures, using 3D models obtained from medical data, is an active field of research. Illustrative visualizations have been applied to this domain in multiple ways. Researchers have tried to make the geometric properties of vasculature more comprehensive and to augment the surface with representations of multivariate clinical data. Techniques that head beyond the application of color-maps or simple shading approaches require a sort of 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, yet effective, parameterization approach that is suitable for tree-like structures. The parameterization is done w.r.t. to a pre-defined source vertex. For this, we present an automatic algorithm, that detects the root of a tree-structure.
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, the display of decals, as an additional depth cue and feature extraction.

CCS Concepts
•***Computing methodologies*** $\rightarrow$ ***Non-photorealistic rendering;*** *Mesh geometry models;*

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 [LSBP18] and also physical simulations [OJMN*18] 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 [PBC*16]. This also plays an important role in medical education [PS18], 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 [MVPL18]. 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 [LVPI18]. 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. 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.

The results are brought to use in several examples.

DOI: 10.2312/vcbm.20181240

2. Related Work

The parameterization of surface meshes has been researched for a long time [FH05]. In the past this field was highly motivated by the topic of remeshing algorithms [BLP*13]. 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. [KCPS15] and Lichtenberg et al. [LSHL18]. 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. [KNP11]. 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. [RASS17]. 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. [KOCC14] 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. [RSH06] 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. [BBPS17] 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 [ROP11]. 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 placement, as done by Lichtenberg et al. [LHL17]. With their work, they followed up on previous methods by Lawonn et al. [LLPH15,LLH17], 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. [KMM*18].

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 [HZ00] who computed integral lines along principal curvature directions. Praun et al. [PHWF01] followed with a texture based approach. Further insights into this illustrative rendering area can be obtained from the survey by Lawonn et al. [LVPI18] while an example of illustrative rendering in the context of vascular models is given by Ritter et al. [RHD*06] .

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. [CWW13] and the *Jump Flood Algorithm* (JFA) by Rong and Tan [RT06]. These two algorithms have a very general domain of application and provide the basis for our parameterization of U and V.

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 [Rus04]. 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 [CWW13]. 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:

$$\mathbf{z}_i = \begin{cases} \mathbf{c}_i, & \text{if } \langle \mathbf{c}_i, \mathbf{g}_i \rangle \geq 0 \\ -\mathbf{c}_i, & \text{otherwise} \end{cases} \quad (1)$$

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

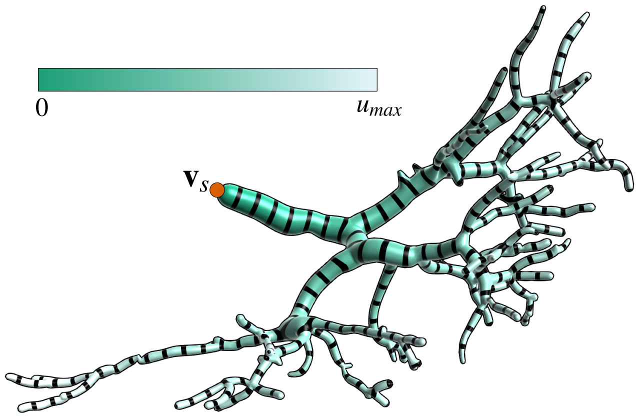

Figure 1: *Resulting U for a given source vertex (marked orange). Isolines underline the smoothness of the result.*

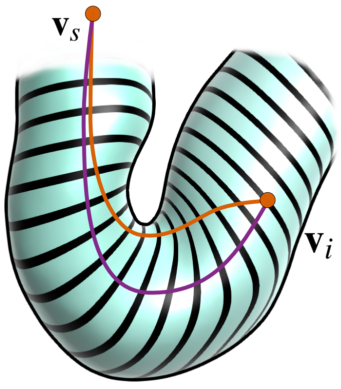

Figure 2: *Delineated geodesic from a vertex* $\mathbf{v}_i$ *to a source* $\mathbf{v}_s$ *(orange) compared with the path along the gradient of U (purple).*

compute U, such that $\nabla U = \mathcal{Z}$. Applying the divergence on both sides yields the Poisson equation

$$\Delta U = \nabla \cdot \mathcal{Z}. \tag{2}$$

Solving Eq. 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 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. to 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 2 shows the final U compared to an 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 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. Constructing a Reeb graph [SK91] based on U may resolve this ambiguity by introducing a segmentation to the mesh with unique isolines per segment.

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 [LLH17, LHL17] presented methods to find end-points, we propose a faster approach to identify these points. Lawonn et al. [LLH17] 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. [LHL17] 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 of our method we apply a thinning to the entire mesh, following the approach by Au et al. [ATC*08]. Their method is based on an iterative shrinking determined by

$$\begin{pmatrix} \mathbf{W_H L} \\ \mathbf{W}_H \end{pmatrix} \mathbf{p}' = \begin{pmatrix} 0 \\ \mathbf{W_H p}, \end{pmatrix} \tag{3}$$

where $\mathbf{W_L}$ and $\mathbf{W_H}$ are diagonal weighting matrices that change per iteration. For every iteration, Eq. 3 is solved in a least squares sense, resulting in Figure 3 (center). 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 $\mathbf{p}^i_j$ on the contracted mesh that lie inside the radius of each sphere B_i (see Figure 3, right). Finally, we take the vector $\mathbf{n}_i = \mathbf{p}^i_j - \mathbf{p}_i$, with $j = \mathrm{argmax}_{\mathbf{p}^i_j \in B_i} \|\mathbf{p}^i_j - \mathbf{p}_i\|$. If all points $\mathbf{p}^i_j$ 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 3, right). 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 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

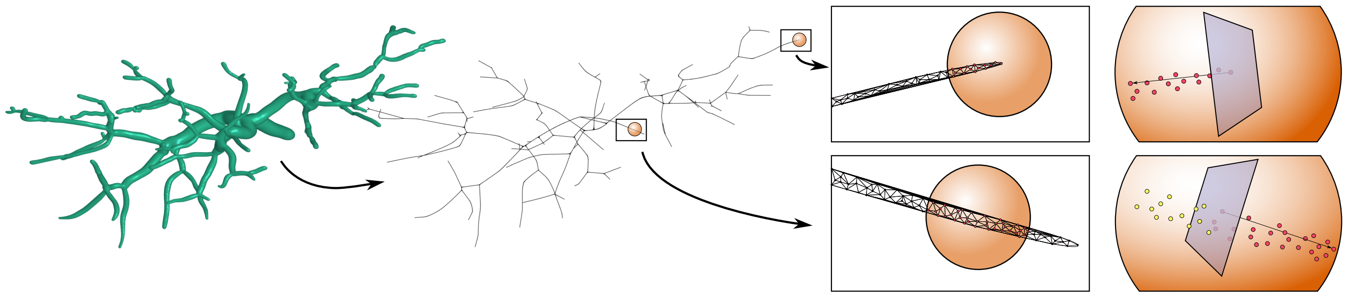

Figure 3: *First, we apply a thinning to the mesh (center). Afterwards, a sphere is created for every point to test if the created plane separates the points inside the sphere (right). If this is not the case, an end-point is found.*

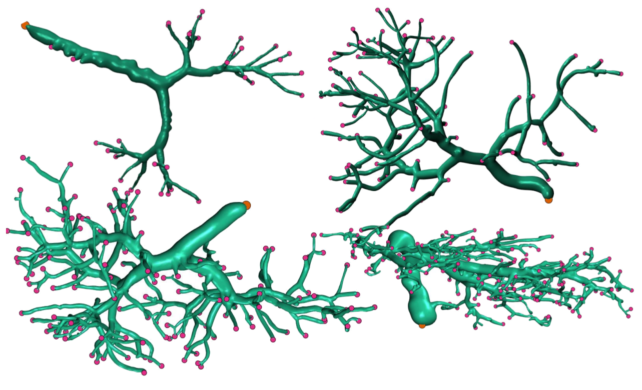

Figure 4: *Result of our root-point (orange) detection for various meshes. Other end-points are marked magenta.*

minimum and maximum values of V meet. Previous work by Kälberer et al. [KNP11] 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. [RLL*06] and followed up upon by Lichtenberg et al. [LSHL18] 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). 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 [RT06] 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 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 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 6 (right). Note that more complex overlaps result in less meaningful approximations of V, which we will address in Section 6. Finally, we rescale V so that (U,V) is isotropic:

$$V \leftarrow V \cdot d_p \tag{4}$$

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 7.

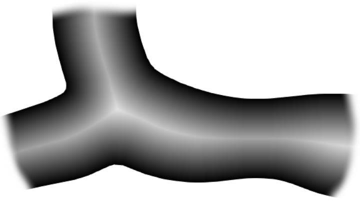

Figure 5: *Minimal distance of each pixel on the mesh projection to the projected mesh contour, mapped to a gray scale.*

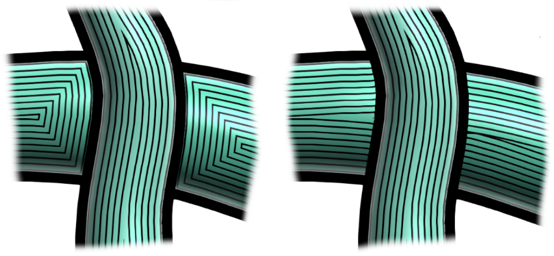

Figure 6: *Unmodified JFA (left) and modified JFA, incorporating pixel depth (right).*

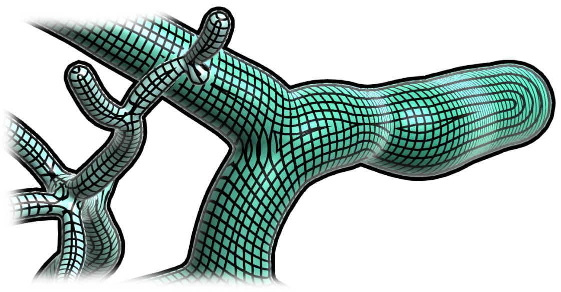

Figure 7: *Example result showing isolines for (U,V).*

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 4.2 shows how this can be utilized.

4. Applications

This sections describes various applications of the (U,V) parameters that we obtained in Section 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.

4.1. Branch and end-points

We can use U as obtained in Section 3 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 3, which creates a sink in the vector field $\mathcal{Z}$ at local maxima of $\mathcal{G}$. As depicted in Figure 8, 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 8, 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 8, 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

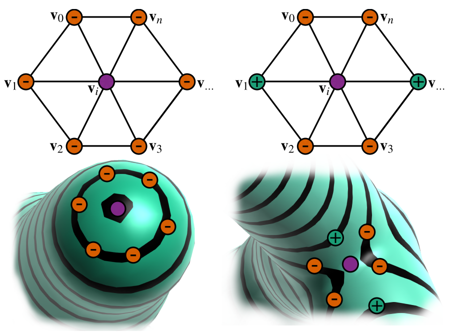

Figure 8: *Configuration of U to detect vessel end-points (maximum, left) and branch locations (saddle, right). The reference vertex is colored purple and vertices of the one-ring with higher or lower values U are marked with a + or - sign, respectively.*

the work by Lichtenberg et al. [LHL17], since the vessel end-points and branch locations are structurally significant features.

4.2. Using V to enhance depth perception

In Section 3 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 9. The result appears similar to the *Void Space Surfaces* by Kreiser et al. [KHR18]. 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 10 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.

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.

$$b_i = \text{floor}((U+o)\cdot f) \tag{5}$$

$$\hat{u}_i = (U+o)\cdot f - b_i; \tag{6}$$

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.

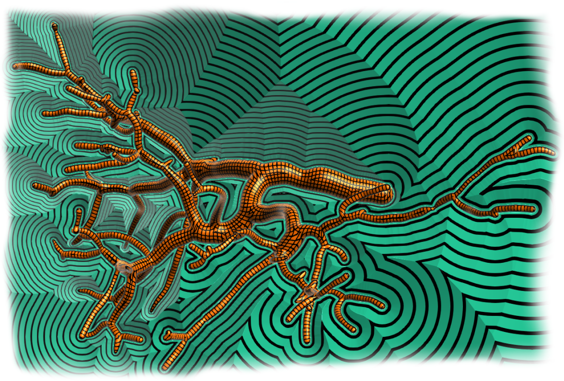

Figure 9: *The parameterization of the background pixels in dependence of the contour depth values. Spacing of the isolines is larger in regions that are closer to the viewer.*

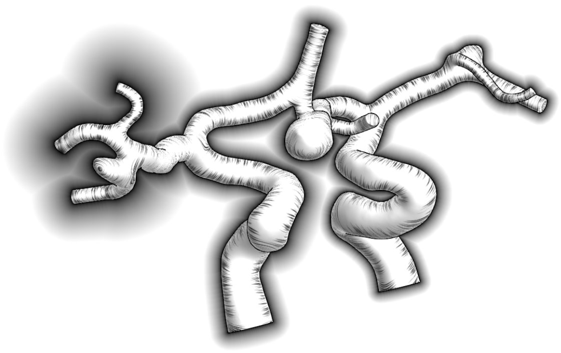

Figure 10: *The parameterization of the background pixels is used to draw a depth dependent glow. The surface itself is rendered with varying hatching strokes.*

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

$$h_i = \cos(\hat{u}_i \cdot 2\pi)0.5 + 0.5 + w_i) \tag{7}$$

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:

$$t = \text{clamp}_{0,1}(\frac{v_i}{l_i})^s \tag{8}$$

$$h_i \leftarrow (1-t)h_i + t \tag{9}$$

where s controls the falloff of the stroke intensity. An initial rendering of equally shapes strokes is shown in Figure 11 (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 11, center, left) and the stroke width $w_i \leftarrow r_{i2}$ (see Figure 11, 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 dependence of V to achieve tilted strokes (see Figure 11, 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 dataset rendered with this technique is shown in Figure 10. The variation of strokes contrasts a reference method by Lawonn et al. [LMP13] shown in Figure 20. Figure 12 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 application, this representation could be used to encode the distance to a reference object (e.g. a tumor or surgical instrument).

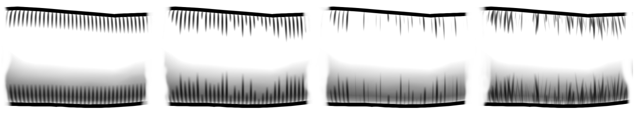

Figure 11: *Different generic variations to the hatching strokes: no variation (left), varying length (center, left), varying width (center, right), final result after overlay of multiple strokes (right).*

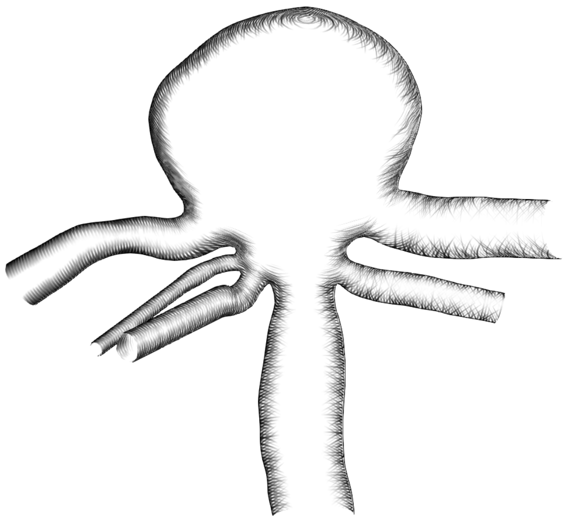

Figure 12: *The variation of hatching strokes is shown in dependence of a scalar field (increasing from left to right).*

4.4. Contour parameterization

If a visualization should be used to encode multiple scalar fields the number of available information channels is often quickly reached. 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. [BBPS17], 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 13), and therefore their scale matches the scale of V as computed in Section 3. The V-value of a point on the contour is then basically its distance to the generating surface. We can then display small texture patches on the contour as shown in Figure 14 (top). In this example, the general direction of blood flow is depicted by an arrow decal at the vessel boundary. In Figure 15 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 16, leaving the surface free for other encodings. This addresses an issue which has been tackled by Behrendt et al. [BBPS17]. 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 17). 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. Implementation

In this section we describe details of our implementation.

Offline calculation of U Since U is defined in object-space, it only has to be calculated once. Therefore we implemented the calculation of U in Matlab (the code is provided in the supplementary material). In order to solve Eq. 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.

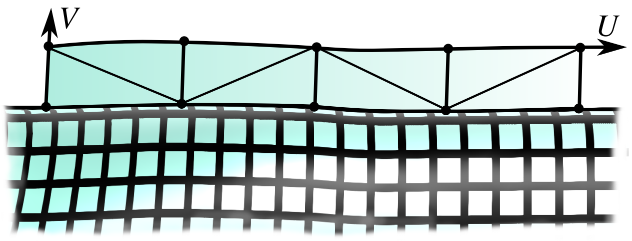

Figure 13: *Additional contour geometry (triangles delineated at the top) is created from the data of the input mesh.*

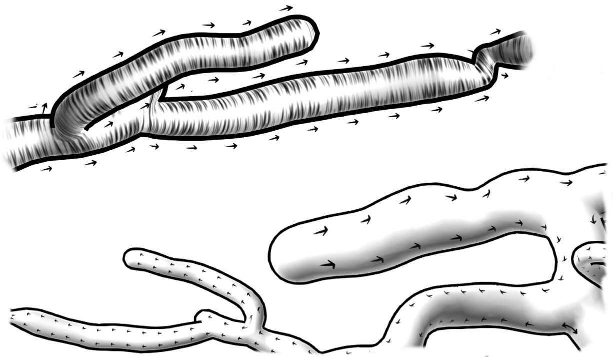

Figure 14: *The contour of the mesh is labeled with decals that emphasize the direction of blood flow (top). The same example using the (U,V) coordinates of the initial parameterization to draw decals directly on the surface (bottom).*

Online calculation of V The evaluation of 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 background, contour and surface pixels (see Figure 18, left, top) and a depth texture (see Figure 18, left, bottom) that is used to implement the depth awareness of the the JFA as depicted earlier in Figure 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 in the supplementary material). For more details on the JFA we refer to the original paper by Rong et al. [RT06]. 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. [ATC*08] and an iterative search

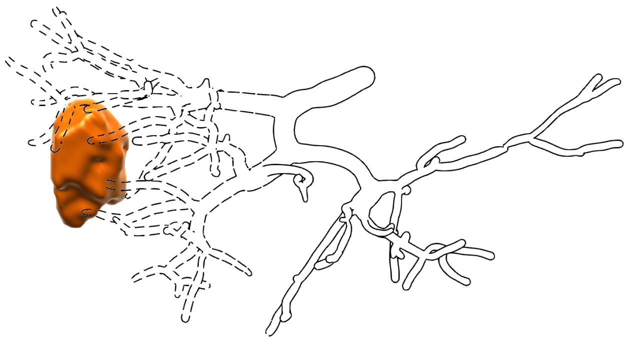

Figure 15: *U is used to draw a dashed contour. The spacing of the dashes decreases with the distance from the tumor.*

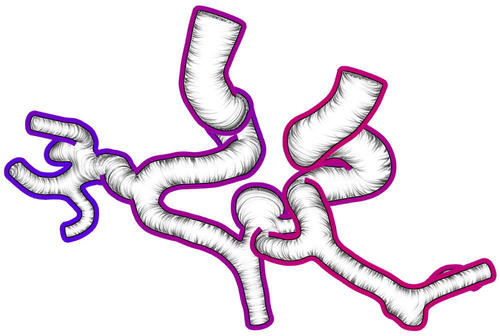

Figure 16: *Contour used to display pseudo chroma-depth (right).*

Table 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

over each vertex as described in Section 3 (the code is provided in the supplementary material).

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 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 10,12 and16). The JFA execution time has not changed in these cases, since always all pixels are processed. The mesh shown in Figure 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. [LHL17]. 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 3, paragraph about root-point detection). A larger radius yields less end-points, detecting only more protruding branches as compared in Figure 19. 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 $\sim$ 110 seconds, while the reference approach took 154 seconds. Especially the iterative execution of vertices for the sphere test (see Figure 3, right) could be improved by a parallel implementation.

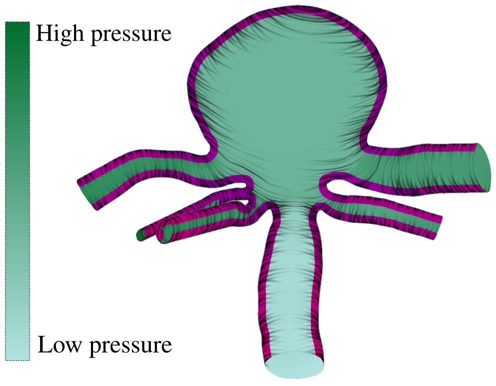

Figure 17: *Pseudo chroma-depth applied to a contour margin. The remaining surface color codes the blood pressure on the surface.*

Contour generation The contours that we used for the contour parameterization in Section 4.4 (see Figure 13) and for the contour mask (see Figure 18, 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 edge 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. to 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.

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 [KNP11] 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 10. 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 23).

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

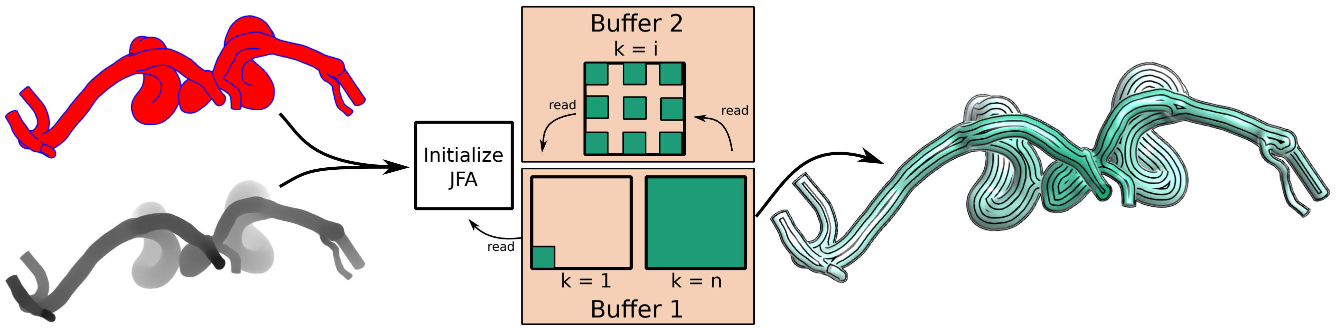

Figure 18: *Process of the JFA execution. A mask image (left, top) determines whether a pixel belongs to background (white), contour (blue) or surface (red). A depth image (left, bottom) is used for our depth aware modification of the JFA. The result of each JFA iteration k is read in the next iteration by swapping two buffers. After the final iteration, the last write-buffer is used to assemble a final image based on V.*

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) 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 21. 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 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 3 that we smooth the vector-field $\mathcal{Z}$ 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 22 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.

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 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. [LSHL18]. Further, resolving the ambiguity of U across different branches of a tree structure would result in a segmentation of the same, e.g. by using Reeb graphs [SK91]. 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.

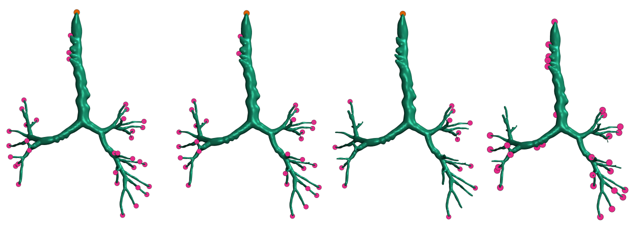

Figure 19: *Detected vessel end-points from left to right: Changing factor $k = 3, 9, 25$ to the search sphere radius for our method and the reference method by Lichtenberg et al. [LHL17] on the far right.*

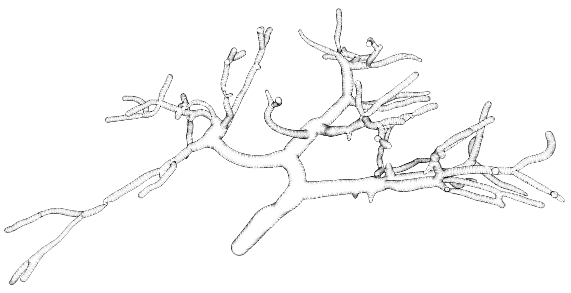

Figure 20: *The ConFis method by Lawonn et al. [LMP13]. Compare with Figures 10,16,14,12.*

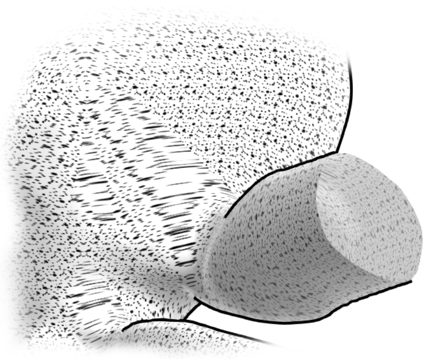

Figure 21: *Closeup of an aneurysm data set. Our approach has issues with the spherical shape, introducing high distortion in V.*

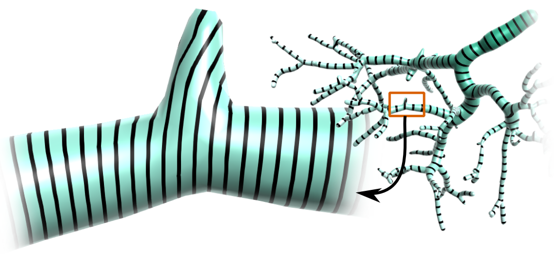

Figure 22: *A tiny branch on the vessel structure is omitted by U.*

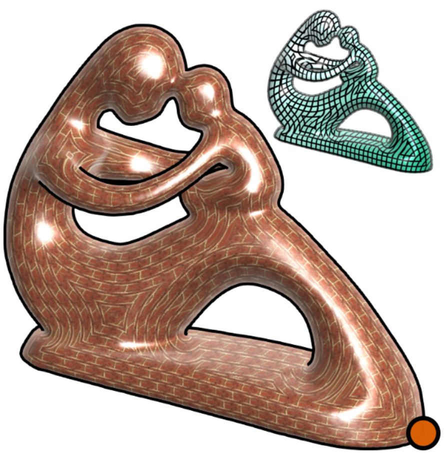

Figure 23: *Our method applied to the fertility model, using a brick texture. A root point (orange) was also found by our method in a reasonable location.*

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

References

[ATC*08] Au O. K.-C., Tai C.-L., Chu H.-K., Cohen-Or D., Lee T.-Y.: Skeleton extraction by mesh contraction. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 44:1–44:10. 3, 7

[BBPS17] 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. 2, 7

[BLP*13] 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* (2013), vol. 32, Wiley Online Library, pp. 51–76. 2

[CWW13] Crane K., Weischedel C., Wardetzky M.: Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics*, 3 (2013), 10. 2

[FH05] Floater M. S., Hormann K.: Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modelling*, Dodgson N. A., Floater M. S., Sabin M. A., (Eds.). Springer, 2005, pp. 157–186. 2

[HZ00] Hertzmann A., Zorin D.: Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 517–526. 2

[KCPS15] Knöppel F., Crane K., Pinkall U., Schröder P.: Stripe patterns on surfaces. *ACM Transactions on Graphics 34*, 4 (2015), 39:1–39:11. 2

[KHR18] Kreiser J., Hermosilla P., Ropinski T.: Void Space Surfaces to Convey Depth in Vessel Visualizations. *ArXiv e-prints* (2018). 5

[KMM*18] Kreiser J., Meuschke M., Mistelbauer G., Preim B., Ropinski T.: A survey of flattening-based medical visualization techniques. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 597–624. 2

[KNP11] Kälberer F., Nieser M., Polthier K.: Stripe parameterization of tubular surfaces. *Mathematics and Visualization*, 202489 (2011), 13–26. 2, 4, 8

[KOCC14] Kersten-Oertel M., Chen S. J. S., Collins D. L.: An evaluation of depth enhancing perceptual cues for vascular volume visualization in neurosurgery. *IEEE Transactions on Visualization and Computer Graphics 20*, 3 (2014), 391–403. 2

[LHL17] 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* (2017), The Eurographics Association. 2, 3, 5, 8, 9

[LLH17] Lawonn K., Luz M., Hansen C.: Improving spatial perception of vascular models using supporting anchors and illustrative visualization. *Computers and Graphics 63* (2017), 37–49. 2, 3

[LLPH15] Lawonn K., Luz M., Preim B., Hansen C.: Illustrative visualization of vascular models for static 2D representations. In *Lecture Notes in Computer Science*, vol. 9350. Springer International Publishing, 2015, pp. 399–406. 2

[LMP13] Lawonn K., Mönch T., Preim B.: Streamlines for illustrative real-time rendering. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 321–330. 6, 9

[LSBP18] Lawonn K., Smit N., Bühler K., Preim B.: A survey on multimodal medical data visualization. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 413–438. 1

[LSHL18] Lichtenberg N., Smit N., Hansen C., Lawonn K.: Real-time field aligned stripe patterns. *Computers & Graphics 74* (2018), 137–149. 2, 4, 9

[LVPI18] LAWONN K., VIOLA I., PREIM B., ISENBERG T.: A survey of surface-based illustrative rendering for visualization. In *Computer Graphics Forum* (2018), Wiley Online Library. 1, 2

[MVPL18] MEUSCHKE M., VOSS S., PREIM B., LAWONN K.: Exploration of blood flow patterns in cerebral aneurysms during the cardiac cycle. *Computers & Graphics 72* (2018), 12 – 25. 1

[OJMN*18] 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* (2018), Wiley Online Library. 1

[PBC*16] 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* (2016), vol. 35, Wiley Online Library, pp. 501–525. 1

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-time hatching. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, p. 581. 2

[PS18] PREIM B., SAALFELD P.: A survey of virtual human anatomy education systems. *Computers & Graphics 71* (2018), 132–153. 1

[RASS17] ROCHA A., ALIM U., SILVA J. D., SOUSA M. C.: Decal-maps: Real-time layering of decals on surfaces for multivariate visualization. *IEEE transactions on visualization and computer graphics 23*, 1 (2017), 821–830. 2

[RHD*06] RITTER F., HANSEN C., DICKEN V., KONRAD O., PREIM B., PEITGEN H. O.: Real-time illustration of vascular structures. *IEEE Trans. Vis. Comput. Graphics 12*, 5 (2006), 877–884. 2

[RLL*06] RAY N., LI W. C., LÉVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. *ACM Transactions on Graphics 25*, 4 (2006), 1460–1485. 4

[ROP11] ROPINSKI T., OELTZE S., PREIM B.: Survey of glyph-based visualization techniques for spatial multivariate medical data. *Computers and Graphics 35*, 2 (2011), 392–401. 2

[RSH06] 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. 2

[RT06] RONG G., TAN T.-S.: Jump flooding in GPU with applications to Voronoi diagram and distance transform. *Proceedings of the 2006 symposium on Interactive 3D graphics and games - SI3D '06* (2006), 109. 2, 4, 7

[Rus04] RUSINKIEWICZ S.: Estimating curvatures and their derivatives on triangle meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission* (Sept. 2004). 2

[SK91] SHINAGAWA Y., KUNII T. L.: Constructing a reeb graph automatically from cross sections. *IEEE Computer Graphics and Applications*, 6 (1991), 44–51. 3, 9