Distance Field Visualization and 2D Abstraction of Vessel Tree Structures with on-the-fly Parameterization

Nils Lichtenberg\(^1\), Bastian Krayer\(^1\), Christian Hansen\(^2\), Stefan Müller\(^1\) and Kai Lawonn\(^1\)

\(^1\)Institute for Computational Visualistics, University of Koblenz, Germany
\(^2\)Computer Assisted Surgery group, University of Magdeburg, Germany

Abstract

In this paper, we make contributions to the visualization of vascular structures. Based on skeletal input data, we provide a combined 2D and implicit 3D visualization of vasculature, that is parameterized on-the-fly for illustrative visualization. We use an efficient algorithm that creates a distance field volume from triangles and extend it to handle skeletal tree data. 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, modifying an object-based algorithm, 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 the 3D surface visualization, we propose a simple layout 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, that highlights the branching topology of the skeleton. We transfer measurements, done in 3D space, to the 2D plot in order to avoid 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 potential of our pipeline is shown in several prototypical application scenarios.

1. Introduction

Visualization methods for the investigation of vasculature have been actively researched in the past and are still a present 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. [GKS*93], more modern and more controllable methods have been developed [OP05]. A common goal of advanced visualization techniques is to communicate complex information about the vasculature in an understandable and comprehensible way. For this, concepts of human perception are considered [PBC*16a]. Illustrative techniques are also often applied in this context [LVP118]. 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 [RLL*06] 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 [SLKL11]. The reduction of spatial dimensions 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 [KMM*18].

Our goal is to visualize vascular tree data and associated information by addressing the aforementioned topics. Recent vascular visualization techniques are wrapped up in the summary chapter by Lichtenberg and Lawonn [LL19]. The discussed methods aim to provide better comprehension of the presented vasculature, by enhancing the perception of depth and distances and therefore the perception of the overall structure. At the same time visual channels are left open to encode clinical data on the rendered surface. However, the problem of self-occlusion and visual clutter is not addressed. Thus, we combine a 2D and 3D view of the vasculature to highlight clinically relevant properties in a clearer way. For example, the distance of vessel segments to tumor tissue inside a liver, which may aid analysis prior a tumor resection, or hemodynamics data [BBPS17]. A crucial factor that determines what kind of visualization techniques can be applied, is the type of data in which the vasculature is represented. Mesh data can be nicely pre-processed to obtain texture parameters but skeletal data is also common, as it is more efficient to store and encodes additional information, such as the vessel’s tree structure. We therefore decided to work on skeletal tree data, because it can be used as input to our 3D and 2D visualization. At the same time we aim to incorporate the advantages of triangulated data, by employing an efficient rendering method and allowing to parameterize the rendered surface on-the-fly.
For the 3D depiction, we use an implicit representation of the surface, based on signed distance fields (SDF), which expose some beneficial properties. The 2D representation will be handled by a graph-layout algorithm. In order to apply illustrative patterns that encode data, e.g., as done by Ritter et al. [RHD+06] or Lawonn et al. [LLPH15], we require a parameterization. As the 3D surface is rendered implicitly, we cannot apply a pre-processing to parameterize it. Instead, we apply a screen-space parameterization (SSP) that is solved for each frame and only takes visible surface information into account. The 2D graph representation will consist of line-segments, thus, no surface to apply illustrative patterns to is available. We circumvent this by applying such patterns to the 2D space next to associated line segments. Our motivation to apply the above mentioned methods is wrapped up in Fig. 1. Looking at the 3D part (Fig. 1, left) a reason why we propose to use a SDF is that we get additional information from it, like shadows, for free. The subsequent SSP provides the basis to encode illustrative shadows and, e.g., stippling can be used to encode tumor proximity. We additionally benefit from the screen-space nature of the parameterization, since it allows to apply patterns to the void space next to the surface as well. Thus, even filigree structures can be highlighted more prominently. For the 2D view, we want to achieve a clean representation, free from overlaps. Patterns used in the 3D depiction should also be visible here, for which we use the free space next to the tree segments (Fig. 1, bottom).

While the SDF helps to rapidly trace the implicit surface, the generation of the SDF itself can also be done at highly interactive rates [KM19]. The properties of the SDF allow simplified implementations of otherwise more complex visual effects. We extend the work by Krayer et al. [KM19] in order to generate a SDF from skeletal vascular data and to render an exact surface from it. The original approach is designed for triangles and cannot be trivially transferred to line segments. Hence, this extension is our first contribution. Second, we propose a simple layout algorithm to transform the 3D skeleton to an abstract 2D graph representation. The graph layout will highlight the branch-topology of the vasculature and allows to view related properties of the whole model in one gaze. This contrasts common layout algorithms that usually try to find a most compact representation without overlaps. For the 3D surface parameterization, we utilize the algorithm by Lichtenberg et al. [LSHL18] and modify it. The original algorithm works on multi-resolution mesh data. Our contribution is to adapt the algorithm such that an implicitly rendered surface can be frame-coherently parameterized to obtain periodic texture coordinates. This 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 show how the main contributions can be combined to provide a tool-set and the basis for different vascular visualization applications that employ a shared 2D and 3D view. The individual features are presented to visualize vasculature within a liver-tumor scenario, but can as well be utilized individually for different tasks. Especially the SSP method generalizes and is applicable to other kinds of surfaces.

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 [KGPS13], with the model-based methods dominating the literature. A reason for this may be that most hemodynamics simulation applications require an explicit surface representation [OJM†17]. For example, the method by Oeltze et al. [OP05] 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 [BS91], 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. [SOB†07] 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. [KGPS13] 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. [SSPOJ16], 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 [FSG03], surface reconstruction [OBA*03] to general rendering applications. Two-dimensional 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 [Gret07]. 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 [Wret15]. An overview over three-dimensional distance fields can be found in Jones et al. [JBS06].

To render structured patterns on the implicit surface, a parameterization is required. While several techniques that employ implicit representations for vascular visualization exist [SOB*07, P008, SSP0116], we are not aware of any approach that would employ a parameterization in order to enable illustrative visualization of the surface. We therefore close this gap in the literature by providing a technique that generates texture coordinates for implicit surfaces on-the-fly. The range of parameterization applications is very wide and well covered in the summary by Sheffer et al. [SPR*07]. As mentioned earlier, parameterizations can be aligned to an input guiding field. Vaxman et al. [VCD*16] 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. [PFH00] 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. [RLL*06]. 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. [KCP15] and Son et al. [SLKL11]. 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. [LSHL18] (triangles), where they moved an approach, similar to the work by Jakob et al. [JTPH15] (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. The hierarchy structure will then be an image pyramid. A recent method by Lichtenberg and Lawonn [LL18] is tailored specifically to tree-like structures and parameterizes the surface partly in object space and partly in image space. As our implicitly rendered surface is given in image space, we adapt the idea of both, the hierarchical and the image based approaches. The goal of the method by Lichtenberg and Lawonn is close to ours and uses the screen space to obtain a texture parameter whose gradient is orthogonal to a structure’s contour. A second parameter, almost orthogonal to the first one, is obtained in object-space. It has to be noted that almost orthogonal is only true for regions close to the object’s contour. Both parameters are continuous, as they have a distinct starting frame (contour or vessel root) in their respective space (screen or object space). In order to achieve a parameterization solely based on screen space information, the computation of the second parameter cannot just be transferred to screen space, because an adequate reference (such as the contour for the first parameter) is missing. One could as well propose to parameterize the skeletal tree in object space, but that would lead to discontinuities on the implicit surface. These problems can be avoided by using a periodic parameterization and we fall back to the method by Lichtenberg et al. [LSHL18]. This also allows to maintain orthogonality of the two parameters and patterns can be displayed with low distortion. Further, the method is not restricted to tree-like input data, as several example figures throughout this paper will show. On the downside, no global information is encoded in the obtained parameters because the periodic intervals cannot be ordered (e.g. from vessel root to end-points). Such a hierarchical approach is highly suitable for parallel execution of the local approximations in one hierarchy level. 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. [GGSC96] 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 [MKC07] 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. [KHR18] shows. Alternative methods to apply structured patterns were proposed: Kim et al. [KYYL08] 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. [BSSM*07] 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 [PBC*16b] have been made by Lawonn et al. [LPH15, LHL17], Lichtenberg et al. [LHL17] or Kreiser et al. [KHR18]. Further, flattening techniques [KMM*18] 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.
3. Method

The vascular trees to be visualized are given as a directed graph \( T \) with nodes \( V \) and edges \( E \). A directed edge \((i, j) \in E\) connects two nodes \((i, j) \in V\) and a node \( i \) is associated with a point \( \mathbf{p}_i \in \mathbb{R}^3 \) and a radius \( r_i \in \mathbb{R} \). We assume a binary, directed, acyclic graph and that the directed edges point away from the root (i.e., each node has at most two children and one parent). 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 \( T_r \) which represents the branching structure of the vascular tree.

This section can be wrapped up as follows: First, the input graphs \( T_r \) and \( T \) are used to generate a 2D (Sec. 3.1) and 3D (Sec. 3.2) depiction of the vasculature. Then, a recap of the PPA (Sec. 3.3) is given, which is then used for a background reconstruction (see Sec. 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. This is similar to the Void Space Surfaces (VSS) by Kreiser et al. (KHR18). However, in our pipeline the background reconstruction is differently motivated and solved with a faster, yet simple algorithm. It ensures that the successive SSP does not need to handle void pixels. Before the SSP is applied to the 3D view (see Sec. 3.6), we generate a guiding field (see Sec. 3.5). To ensure a frame coherent parameterization, successive frames are optimized based on the results of the previous frame (see Sec. 3.7).

Fig. 2 depicts the main sections of the method.

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 input tree. We use the reduced tree \( T_r \) and arrange the nodes in a way that allows to easily identify different sub-trees. Common graph layout algorithms try to provide a compact representation, that optimizes the required space and treats nodes equally. In our scenario, we have different constraints: we want the user to be able to visually extract individual sub-trees of the graph. Hence, different sub-trees shall not overlap, i.e., if the tree-depth is plotted along the vertical axis, sub-trees do not overlap across the vertical axis. Further, we want sub-trees to be more or less prominent, based on some assigned weight per node. For example, nodes in a small sub-tree, that may be negligible for more or less prominent, based on some assigned weight per node. The h-space is subdivided by individual sub-trees of \( T_r \). A larger \( p_h \) amplifies the weight difference and therefore a larger fraction of the \( h \)-parameter is assigned to the node with the larger weight. 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 \( T_r \) 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 \([0, 1]\) to draw the vessel graph. Using \((h, s)\) as is, a linear layout (see Fig. 2 (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. 10 (bottom). The \((h, s)\) parameter can also be directly used 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 \( T_r \) the coloring is able to highlight the respective sub-trees.

Algorithm 1 2D Node Layout

1: procedure CHILDLAYOUT\((n_i, r_1, r_2)\)
2: if NumberOfChildren\((n_i) = 2\) then
3: \( c_1, c_2 = \text{getChildNodes}\(n_i)\)
4: \( w_1 = \text{weight}(c_1), w_2 = \text{weight}(c_2)\)
5: \( s(c_1) = n_i - \sqrt{\min(w_1/w_2, 1)}\)
6: \( s(c_2) = n_i - \sqrt{\max(w_2/w_1, 1)}\)
7: \( r_i = (w_2 r_1 + w_1 r_2)/(w_1 + w_2)\)
8: \( w_r = \min\left(\frac{w_1}{w_2}, \frac{w_2}{w_1}\right)\left(p_h, \max\left(\frac{2\pi}{3}, \frac{\pi}{2}\right)\right)\)
9: \( m_1 = (w_r - 1)\left(\frac{1}{r_1} + \frac{1}{r_2} - h(n_i)\right)\)
10: \( m_2 = (w_r - 1)\left(\frac{1}{r_1} + \frac{1}{r_2} - h(n_i)\right)\)
11: \( h(c_1) = \text{mean}\left(r_1 + m_1, r_1 + m_1\right)\)
12: \( h(c_2) = \text{mean}\left(r_2 + m_2, r_2 + m_2\right)\)
13: \( \text{ChildLayout}(c_1, r_1 + m_1, r_1 + m_1)\)
14: \( \text{ChildLayout}(c_2, r_2 + m_2, r_2 + m_2)\)

to the range \([0, 1]\) to draw the vessel graph. Using \((h, s)\) as is, a linear layout (see Fig. 2 (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. 10 (bottom). The \((h, s)\) parameter can also be directly used 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 \( T_r \) the coloring is able to highlight the respective sub-trees.

3.2. Signed Distance Field Generation

We use an algorithm to prepare a fast sampled signed distance field for skeletal data, which is a modification to the work by Krayer et al. (KM19). This and other related techniques operate on triangle data. We utilize the same idea of taking advantage of the hardware rasterization unit, but extend it to use skeletal input without prior triangulation. Due to the exact distance function described below,
The signed distance field for skeletal segments is derived by geometric considerations, similar to Barbier et al. [BG04]. In contrast to them we use a full SDF instead of using zero for all points inside 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. 3, left). We fix the origin at \( p_i \) and orient the \( x \) axis to coincide with the direction \( p_j - p_i \). We define the distance between both points \( s = |p_j - p_i| \) and the relative radius \( r_d = r_j - r_i \). A second coordinate system \((i, j)\) is located around the point where the outer tangents of both circles coincide. Its second axis \( j \) is parallel to the vector \( e - p_i \), which is perpendicular to the outer tangent line, thus determining the first axis \( i \). With geometric considerations, the center point \( c \) is found by computing the offsets from \( p_j \) in the \( x \) and \( y \) directions along with the length between \( c \) and the tangent intersection with the second circle:

\[
e = \left( \frac{d_i}{h_i} \right) = \frac{1}{s} \left( \frac{r_d r_i}{r_i l} \right), \quad \text{with } l = \begin{cases} \sqrt{s^2 - r_d^2} & \text{if } s \geq |r_d| \\ \max(r_i, r_j) & \text{otherwise} \end{cases}
\]

This formulation is valid regardless of whether \( r_i \) or \( r_j \) is larger. A given query point \( q \) in the first coordinate system can be transformed to the second coordinate system with

\[
q' = \left( \frac{q_i}{q_j} \right) (q - e), \quad \text{with } i = \frac{1}{s} \left( \frac{l}{-d_i} \right), \quad j = \frac{1}{s} \left( \frac{d_j}{l} \right)
\]

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

\[
s(q) = \begin{cases} |q - r_i| & , \text{if } q_i' < 0 \\
|q - p_j| - r_j & , \text{if } q_i' > l \\
q_i' & , \text{else} \end{cases}
\]

The distance field of the surface of revolution in 3D is obtained by projecting a query point \( p \) into the two-dimensional space and then evaluating the field there. The two-dimensional coordinates \( q \) are given as the projection of a query point onto the line segment and its distance to the line through \( p_i \) and \( p_j \).

**Fast Signed Distance Field Generation** Our GPU algorithm is a variant of the unsigned distance field generation used in [KM19], which is modified to support the special non-circle 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 al. [HAMO05]. Since we deal with an implicit representation based on line segments (the actual surface is the result of the segment and associated radii), we have to extend the original technique, which will be described next. 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. We project the primitive in either x, y or z direction, such that the largest amount of fragments is occupied by the projection. 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. 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. A around this projected line a bounding box, oriented along the line direction, is constructed. Its size is determined by the segment length and the larger 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. 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 occupied cell. This is used in a last step, to refine distances by computing the exact values of a center cell to the geometry stored in the nearest 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 observation was made by Hart [Har96] 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 found, as otherwise that closer geometry would provide a smaller distance. This is shown in Fig. 3 (right). 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, as in our case. 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, which may be used to associate data with the traced surface. Fig. 4 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 p, the closest line segment’s index ∈ E and the normal, given by the SDF’s gradient.

3.3. Pull-push algorithm

The PPA was introduced by Gortler et al. [GGSC96] and makes use of a multi-resolution image pyramid to interpolate scattered data of an input image. As stated by Rosenfeld [Ros13] 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. [GGSC96] 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 Ai−v is the set of neighbors of i in the same pyramid level (the four horizontal and vertical neighbors of a pixel). The sets Ai−h and Ai−v 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 [MKC07].

The Pull-phase constructs the image pyramid from the input image by iteratively computing the coarser levels:

\[
\begin{align*}
w_i &= \sum_{j \in A_i^-} h_{j,i} \min(w_j, 1) \\
\end{align*}
\]

\[
\begin{align*}
g_i &= \frac{1}{w_i} \sum_{j \in A_i^-} h_{j,i} \min(w_j, 1) g_j
\end{align*}
\]

where \( h_{j,i} \in [0, 1] \) (default: \( \frac{1}{|A_i^-|} \)).
The Push-phase iteratively reconstructs data from coarse to fine levels in the pyramid, where \( h_{ji} \in [0, 1] \) (default: \( \frac{1}{|A_i|} \)):

\[
\hat{w}_i = \sum_{j \in A_i^c} h_{ji} \min(w_j, 1) \tag{6}
\]

\[
\hat{g}_i = \frac{1}{w_j} \sum_{j \in A_i^c} h_{ji} \min(w_j, 1) g_j \tag{7}
\]

The above \( \hat{w}_i \) and \( \hat{g}_i \) 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:

\[
\alpha = 1 - (1 - w_i)^k \tag{8}
\]

\[
g_i \leftarrow \hat{g}_i (1 - \alpha) + w_i g_i, \quad w_i \leftarrow \hat{w}_i (1 - \alpha) + w_i \tag{9}
\]

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. The \( k \) parameter has a similar effect as the exponent to an inverse distance weight. The result is more sensitive to local data when using a larger exponent (see Fig 5). Note, that the result is not exactly the same, because of the multi-resolution hierarchy of the PPA.

### 3.4. Background reconstruction

This section describes the reconstruction of the background for the 3D view. The SSP (see Sec. 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. [KHR18] 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 \( p_i \), 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 depth of a pixel. This yields a smooth height profile that connects to the foreground. During the push phase, we set \( h_{ji} \) to bi-quadratic b-spline weights as used by [MKC07] and the parameter \( k \) in Eq. 8 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 \( p_i \) for the given depth value and location of \( i \) in screen space [vdLGS09]. A normal \( n_i \) can be obtained by the second derivative of the newly obtained positions. An example result is given in Fig. 5. 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.

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. As opposed to the 3D view, the positional information is interpolated directly and not based on the height profile. The reconstructed background of the 2D view will help to visualize 3D-based magnitudes, as will be shown in Sec. 5.

### 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 \( Z \). We assume that no field is defined on the input structure and construct it from the available data. For this, we look at the depth image produced by the input surface and apply the Sobel filter to obtain a gradient \( \tau_i \) 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. If \( |\tau_i| \) is larger than a threshold \( t \) (default: 0.2, i.e., 20% of the bounding box), then the pixel is considered as a contour generator and we set \( x_i \in Z \) 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 non-contour pixels. The variable \( w_f \) (default: 0.2) is used to introduce a smoothing into the existing data. Note that after the PPA has run, the weight of each individual pixel reaches 1. Thus, by setting \( w_f = 0.2 \), only 20% percent of the original value remain, while the rest is introduced by the surrounding area. Note that \( Z \) is considered in screen space and we process it as a 2D field. In the pull and push Eqs. 4 through 7 the element \( g \) (i.e. the element being reconstructed) is a matrix representation of the 2D vector \( z \in Z \). If \( z = (a, b) \), then \( g = \begin{pmatrix} a & b \\ a & b \end{pmatrix} \). We obtain the resulting 2D vector as the first eigenvector of the averaged matrices. This matrix representation handles the ambiguity of \( Z \): it only defines an orientation and not a direction, i.e., \( Z \sim \mathbb{Z} \). 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 \( h_{ji} = 1 \) during the pull phase. During the push phase, \( h_{ji} \) is additionally scaled by \( \frac{1}{1+w_i} \), 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

![Figure 5: Background (pseudo chromadepth [RSH06] colors) reconstructed from a tree. Isolines highlight the depth profile, resembling VSS [KHR18]. The k-parameter set to 1 (left) and 25 (right).](image.png)
foreground. The result is a guiding field that lets us produce texture coordinates that are aligned to the contour of the input structure and are also smooth away from the contour. This contrasts the method by Lichtenberg et al. [LL18], where the background is parameterized as a distance field of the contour whose gradient is not guaranteed to be smooth. Fig. 6 shows two example images of parameterizations using the SPP, which will be described in the next section.

3.6. Screen space parameterization

This section describes how we employ the method by Lichtenberg et al. [LSHL18] to obtain 2D periodic texture coordinates \((U, V)\) in screen-space that are aligned to \(Z\), 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. The original algorithm aims to minimize a global energy function by employing a divide-and-conquer technique. Thus, a solution close to a global minimum can be found while executing individual chunks of the optimization in parallel. The input to the method is a tangent vector field and the algorithm produces a periodic scalar field, such that the gradient of the scalar field is equal, or very close, to the input vector field. At the bottom-line, we simply take the formulation from Lichtenberg et al. [LSHL18] and apply it to the topology of the image pyramid, instead of the topology of a triangulated mesh. Recall that the topology within one hierarchy level of the PPA pyramid is given by \(A^*\) and the topology across hierarchy levels is given by \(A^{-}\). Since we are now working in screen space and cannot store the result of the parameterization for the implicit surface, additional obstacles have to be overcome. We now initialize the top hierarchy level (consisting of a single pixel) with \(w_1 = 1\) 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 \(Z\). Since the field \(Z\) is defined in 2D screen space, but the parameterization algorithm takes a 3D field as input, we project each \(z \in Z\) into the tangent plane of the respective pixel.

The parameter push operation reconstructs \(u_i \in U\) based on the values at \(A^+_1\). The element \(g\) in Eq. 7 (i.e., the element being propagated through the hierarchy levels) now refers to the Cartesian target coordinate \(\phi_{ij}\) as in Eq. 8 in [LSHL18]. 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 [LSHL18]. The energy of two neighboring pixels is zero, if \(||z_i - p_j|| = |u_i - u_j|\), where \(f\) is a scale factor that finally determines the grid size of the parameterization (see the grid visualizing the result in Fig. 6). 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 \(p_i\) and \(p_j\), projected to the guiding direction \(z\). The weight \(h_{ij}\) to obtain \(\phi_{ij}\) from a higher hierarchy level is assembled by several measures:

\[
h_{ij} = \frac{1}{(p_i - p_j)^2} \cdot m_j \cdot |\langle z_i, z_j \rangle|
\]

where \(p_i\) is the 3D position represented by a pixel, \(z\) the direction of the guiding field at that pixel. Here, \(m_j\) is a penalty applied to background pixels, being 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. With this modification, we can apply Eq. 7 to compute an average Cartesian target coordinate and obtain \(u_i = \tan^2(\phi_{ij})\), where \(\phi\) is the average optimal parameter for \(i\) for all neighbors \(j\).

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 [LSHL18]) but use the \(A^*\) neighborhood instead of the \(A^{-}\) 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 Fig. 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. This is due to the hierarchical approach. If the input to the PPA causes a slight change in the coarser pyramid levels during the pull phase, the subsequent push phase may amplify this change towards the finer levels. 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 smooth the changes to the background morphology over the frames. This is easily achieved by a small adjustment to the procedure in Sec. 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 \(w_f\) for the background pixels is set to a parameter \(w_f \in [0, 1]\) (default: 0.01). Thus, 1% of the previous frame’s information, (i.e., 3D position per pixel), is pulled into the current frame.
4. Implementation

This section briefly implements the computation time of the individual pipeline steps. The 2D graph layout (Sec. 3.1) is computed in a C++ application, based on the input graph \( \mathcal{T} \), the node weights and the user parameters \( p_s \) and \( p_b \). The 2D node positions are then loaded onto the GPU and edges of \( \mathcal{T} \) 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 3.2)
- S2: Compute 2D graph layout (Section 3.1)
- S3: Render input images (Section 3.2)
- S4: Reconstruct background (Section 3.4)
- S5: Compute SSP (Section 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 or the weights that are assigned to \( \mathcal{T} \) 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

<table>
<thead>
<tr>
<th>Step</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixels</td>
<td>Step</td>
<td>512²</td>
<td>6-12</td>
</tr>
<tr>
<td>1024²</td>
<td>20-30</td>
<td>1.15</td>
<td>15.09</td>
</tr>
</tbody>
</table>

Performance  To assess 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 64³ took on average 2.1 ms for input trees consisting of about 1000 edges. The generation of a 128³ volume took 13.1 ms on average. Executing the 2D graph layout algorithm is in sub-millisecond range. Table 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 \( p_p = 3 \).

5. Applications and benefits

This section depicts potential application areas of our method and highlights the benefits over state-of-the-art techniques.

Visualization of vascular trees  First of all, the SDF can be used to simply visualize a given tree structure. Fig. 7 compares our result with the convolution surfaces by Oeltze et al. [OP05]. Please note, that we used an example skeleton consisting of about 6000 edges, provided by the MeVisLab [mev] WEMVascularSystem module.

It is neither a binary tree, nor is it passed to the convolution surface extraction algorithm without pre-processing, so our representation may deviate from the original. We can state, however, that the generation of the convolution surface for this tree took about
vessel segments to the tumor surfaces is encoded. Regions that fall
primitive (see Fig. 9, left). In Fig. 2 (right) the minimal distance of
Son to the guiding field, which allows us to produce results similar to
periodic texture coordinates directly to run a stippling and hatching
Illustrative rendering styles for implicit surfaces
Texture coordinates for implicit surfaces
Background reconstruction for medical visualization
Recently, the work by Kreiser et al. [KHR18] used the so called Void Space Surfaces (VSS) around projections of medical vascular data to enhance depth perception. They use IDW [She68] to interpolate depth values from the vessel contour into the background. A similar result can be obtained by our background reconstruction described in Section 3.4, as shown in Fig. 2 (center, left) and Fig. 5. 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 1.5 ms. Of course, rendering the extracted triangle surface is less costly than the sphere tracing that needs to be applied to the SDF. As a compensation, the SDF visualization provides the basis for additional features, that will be explained later in this section.

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 3). 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 $d_{\min} < d_i$, where $d_i$ 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. This allows to obtain $d_{\min}$ for a passing surface, even if the ray finally hits another part of the surface. 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. [RHD*06]. If $d_{\text{shadow}}$ is the distance of the traced surface to the point associated with $d_{\min}$, we draw a depth shadow if $d_{\min} < \min(m \cdot d_{\text{shadow}}, a)$ and $d_{\min}$ is less, a shadow, which is not the case for ray B (left). The depth dependent shadows support depth perception as in [RHD*06] (right).

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 3). 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 $d_{\min} < d_i$, where $d_i$ 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. This allows to obtain $d_{\min}$ for a passing surface, even if the ray finally hits another part of the surface. 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. [RHD*06]. If $d_{\text{shadow}}$ is the distance of the traced surface to the point associated with $d_{\min}$, we draw a depth shadow if $d_{\min} < \min(m \cdot d_{\text{shadow}}, a)$ and $d_{\min}$ is less, a shadow, which is not the case for ray B (left). The depth dependent shadows support depth perception as in [RHD*06] (right).

Surgical risk assessment
We propose a risk assessment visualization to highlight the strength of the combined 2D and 3D view. Fig. 10 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 Fig. 2 (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.

SDF features As described previously, the SDF can be used to visualize vascular trees based on an implicit representation. Though the additional complexity of the approach may not pay off for simple surface generation, the SDF enables additional features that would otherwise be more difficult to achieve. 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 [Har96]. For instance, primitive combinations can be applied by calculating the union, intersection or subtraction of different primitives’ SDFs. As shown in Fig. 11, Boolean operations can be applied to carve portions from a surface. Note that a cut always produces a valid surface and the implicit surface appears like a solid object (see Fig. 11, right). As shown in Fig. 8 the additional information encoded in the SDF can be used to immediately obtain shadow information, with the camera being the source of light, which can then be used to implement depth-shadows [RHD*06].

The surface tracing described in Sec. 3.2 employs a quick look-up of the segment of T closest to a traced surface point in order to avoid 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 Fig. 11, where the tumor surface is colored based on the closest vessel segment. 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 viewer’s attention. A default and an alternative layout are shown in Fig. 12, with a color-code depicting the distance of graph segments 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 (Fig. 12, left). In contrast (Fig. 12, 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. 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 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 re-initialization of the successive frame, using $l_p$ pyramid levels may prevent the algorithm from convergence. An alternative would be to set $l_p = 0$ if the scene does not change, then convergence is achieved. Further, the recycling depth $l_p$ (see Section 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. Fig. 13 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. A practical solution for this issue would be to employ a safe-margin, i.e., to copy only a centered portion of the processed image to the display. The background, i.e., the void space [KHR18] 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
Figure 10: 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.

Figure 11: 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.

Figure 12: Graph layout with default weights, based on the number of sub-tree nodes (left). Alternative based on scalar field (right).

Figures 13: Left to right: Sequence of bunny entering the view port.

Tracing are implemented, the Boolean operations are a way to create focus and context applications with low effort and high visual quality. In the future, we would like to investigate to which extent hypertextures [PH89] 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.
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.

References


[mev] MeVisLab, MeVis Medical Solutions AG. [Online; accessed 23-July-2019]. URL: www.mevislab.de, 9, 10


Section 3.1

- $p_k$: the impact of different node weights on the final layout. The default uses an empirical value $p_k = \frac{1}{2}$. Larger values result in a layout where nodes with larger weights are assigned an even larger portion of the available h-space. Thus, with a larger $p_k$, whole sub-trees may vanish in the 2D layout, if the contained nodes have low weights.

Section 3.3

- $A^*, A^+, A^-*$: the pixel neighborhood within the image hierarchy as depicted in Fig. 14.
- $w_i$: weight of the new (coarser) pixel during pull phase.
- $\hat{w}_i$: weight of the new (finer) pixel during push phase.

### Appendix A: Method parameters

Here, we summarize the various method parameters for a quick overview. Except for $k$, $p_k$, $l_p$ and $f$, all parameters are empirically determined and should not require any changes. The four mentioned above may require tuning from the user, but are also set to empirically feasible values.

Section 3.5

- $r$: threshold $\in [0..1]$ within the object bounding box, that determines what depth difference of neighboring pixels is required to consider the pixels as contour pixels.
- $w_i$: initial weight of contour pixels before application of the PPA. The guiding field is smoothed for $w_i < 1$, because (smoothed) information from the coarser hierarchy levels is used with a factor $1 - w_f$.

Section 3.6

- $f$: scaling factor to the periodic interval. It determines the size of the grid given by $[U, V] = x$, where $x$ is any value of the periodic interval.
- $m$: fraction of foreground pixels in the lowest hierarchy level, represented by a pixel in any hierarchy level.

Section 3.7

- $w_k$: weight to initialize the frame coherent background reconstruction. With the default value $w_k = 0.01$, the background information (3D position per pixel) is taken with a weight of 1% into the computation of the new frame’s background. Due to sufficiently high frame rates, this small impact is sufficient to achieve a smooth update of the background across frames.
- $l_p$: reconstruction depth for the SSP. The SSP result of the previous frame is pulled into a the $l_p$th hierarchy level for initialization of the current frame. The larger $l_p$, the less local information remains. Therefore a balance between the visualized structures detail, screen resolution and $l_p$ need to be found. We propose an empirically determined default value of $l_p = 3$. 

![Figure 14: Hierarchy level adjacency for a given pixel i.](image-url)