Line Density Control in Screen-Space via Balanced Line Hierarchies

Mathias Kanzler[1], Florian Ferstl[1], Rüdiger Westermann[1]

Abstract

For the visualization of dense sets of 3D lines, view-dependent approaches have been proposed to avoid the occlusion of important structures. Popular concepts consider global line selection based on line importance and screen-space occupancy, and opacity optimization to resolve locally the occlusion problem. In this work, we present a novel approach to improve the spatial perception and enable the interactive visualization of large 3D line sets. Instead of making lines locally transparent, which affects a lines spatial perception and can obscure spatial relationships, we propose to adapt the line density based on line importance and screen-space occupancy. In contrast to global line selection, however, our adaptation is local and only thins out the lines where significant occlusions occur. To achieve this we present a novel approach based on minimum cost perfect matching to construct an optimal, fully balanced line hierarchy. For determining locally the desired line density, we propose a projection-based screen-space measure considering the variation in line direction, line coverage, importance, and depth. This measure can be computed in an order-independent way and evaluated efficiently on the GPU.

Keywords: Scientific Visualization, Flow Visualization, Line Fields, Focus + Context, Line Hierarchy

1. Introduction

Integral lines such as streamlines or pathlines are among the most popular means for visualizing 3D flow fields, because they can convey to the user in an intuitive way the structure of these fields. For thorough overviews of flow visualization techniques in general, and integration-based techniques such as integral lines in particular, let us refer to the state-of-the-art reports by Weiskopf and Erlebacher [1] and McLoughlin et al. [2], respectively.

However, occlusions and visual clutter are quickly introduced when too many lines are shown simultaneously. Thus, especially in three dimensions a major challenge is to select a set of lines—containing as few as possible elements—that captures all relevant flow features. A number of effective selection strategies for integral lines have been proposed, for instance, approaches which determine the line set in a preprocess via importance- or similarity-based criteria [3, 4, 5, 6].

In general, pre-selecting the lines cannot account for the problem that parts of relevant lines are occluded by less important parts of other lines being closer to the viewpoint in the rendered image. To avoid such occlusions, screen-space approaches either select the rendered lines dynamically on a frame-to-frame basis, for instance, by considering line importance and local screen-space occupancy [7, 8], or they locally adapt the line opacity to fade out those parts of foreground lines that occlude more important lines [9]. Omitting entire lines has the advantage that the remaining lines are not fragmented, meaning that properties like the line length can still be conveyed. On the other hand, this strategy can result in unnecessarily sparse depictions, since in some areas a removed line might not have caused any disturbing occlusions. Opacity adaption, in contrast, resolves the occlusion problem locally by discarding line segments instead of entire lines. This enables to emphasize relevant focus information while preserving the "surrounding" context that does not obscure relevant structures. The focus+context principle underlying this approach has been studied extensively in the context of volume rendering by Viola et al. [10].

Opacity adaption, on the other hand, can affect negatively the perception of spatial relationships between lines in the context region. Increasing transparency causes a simultaneous desaturation of the object color, which is perceived as increasing distance from the viewer. Due to this effect, which is known as aerial perspective, transparent parts of a line can seem to flatten out or even bend away from the viewer.

We propose an alternative approach which avoids this effect. Instead of using transparency, we adapt locally the screen-space density of the lines to their importance. In this way, the spatial perception of the lines remains unaffected. The spatial adjustment of the line density is achieved via the use of a precomputed, fully balanced line hierarchy, and the selection of lines from this hierarchy at run-time according to image-based density control attributes. The particular hierarchy ensures that lines are removed uniformly in the domain, and the removal of individual line segments is contiguous and does not cause fragmentation.

Our particular contributions are:

- A novel combination of line clustering and minimum cost perfect matching to construct a fully balanced line

Email addresses: `kanzler@in.tum.de` (Mathias Kanzler), `ferstlf@in.tum.de` (Florian Ferstl), `westermann@tum.de` (Rüdiger Westermann)

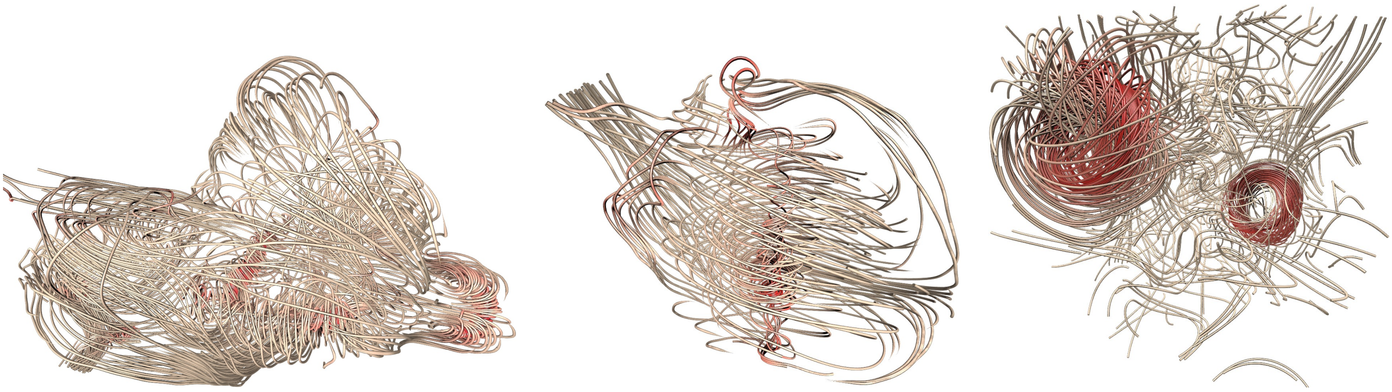


Figure 1: Visualization of flow fields (Aneurysm I/II, Rings) using line density control. Line color ranges from red (high importance) to light brown (low importance).

hierarchy.

- A number of view-dependent, yet order-independent control parameters to locally steer the line density.
- A scalable embedding of local line density control into the line rendering process.

We demonstrate our method for flow visualization in a number of real-world examples, and we compare the results of our specific modifications and extensions to other view-dependent line selection and rendering approaches. Some data sets that have been visualized by using our approach are shown in Fig. 1. Our evaluations include perceptional as well as performance and scalability issues, and they demonstrate the suitability of the proposed approach for interactive applications. On the downside, since our method splits less important lines into segments, it can become difficult to determine the length of these lines from the visualization. Furthermore, compared to opacity optimization, which smoothly fades out the less important lines, our approach generates sharper, more abrupt line endings, which, in some cases, can give a less smooth overall impression.

2. Related Work

Finding an as small as possible set of integral lines which represent a multi-dimensional flow field and its dominant structures in a comprehensive way is challenging. One way to approach this problem is to use line seeding strategies considering criteria like the line density [11, 12, 13, 14], the line distribution [15, 16, 17, 18, 19, 20, 21], the information entropy in the seeded lines [22], or the coverage of specific flow features [3, 4, 6] or geometric line features [5, 23].

Even though these techniques can be very effective in determining a good representative set of lines, they do not consider how much occlusion is produced when the seeded lines are rendered from different viewpoints. As a consequence, even though a good coverage of the domain or the relevant flow features in object-space can be achieved, lines representing relevant features may be occluded in the rendered images.

To overcome this limitation, view-dependent rendering strategies for 3D line sets have been introduced. The method by Tong et al. [24] deforms occluding lines that should not be in focus, so that the focus is revealed. Even though this approach can effectively avoid occlusions, the deformation of lines can give a wrong impression of the underlying flow field. Most alternative techniques generate an initial set of “important” lines, and determine for each new view the subset to be rendered—possibly enhanced with additional lines that are generated for this view—so that occlusions are reduced and more important lines are favored over less important ones [7, 8, 25]. Indicators for the amount of occlusion in the rendered images can be based on the “overdraw”, i.e., the number of projected line points per pixel [7, 8], or the maximum projected entropy per pixel [25].

When selecting and rendering entire lines, less important lines might be removed entirely, even though only a small part of it actually occludes some part of a more important line. In the worst case this can result in a subset in which only the most important lines are kept, yet contextual information represented by less important lines is lost (see upper right image of Fig. 2). This interferes especially with the focus+context paradigm, which suggests to show an importance-filtered fraction of the data set, i.e., the focus, embedded into context-conveying positional cues. Viola and Gröller [10] and Krüger et al. [26] discuss this paradigm in the context of volume rendering, and they propose guidelines which we also consider in our work. For integral lines, Marchesin et al. [7] and Ma et al. [8] address this problem by adding short fragments of less important lines in regions not yet occupied.

Recently, Günther et al. [9] proposed opacity optimization to circumvent the problems that are introduced when removing entire lines. Instead, the opacity along a line is modified selectively to fade out the line only in those regions where it occludes more important ones. Opacity optimization avoids the removal of lines where no occlusions occur, yet in the foreground of important lines, less important lines can fade out entirely so that contextual information is lost, and mutual spatial relationships between these lines become difficult to grasp with increasing transparency. This effect is demonstrated in the bottom left image of Fig. 2.

It is worth noting that opacity optimization computes the opacity values via a global optimization that considers the line’s importance as well as the order of occlusions. Thus, opacity

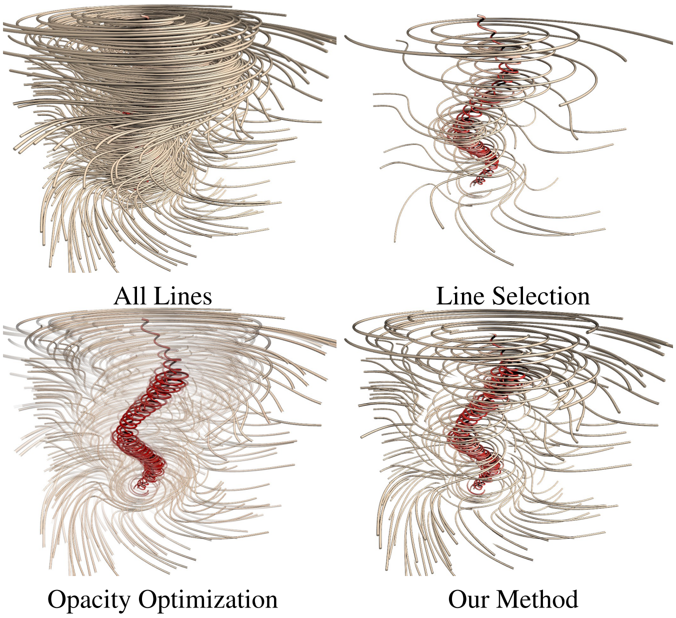


Figure 2: Different approaches to reveal the focus region in the Tornado dataset.

optimization requires to store a per-fragment linked list on the GPU, which can become very memory intensive when large and dense sets of lines are visualized.

In our work we aim at combining the strength of local approaches to resolve unwanted occlusions, with a view-dependent focus+context approach that allows keeping context-conveying positional cues even in case of occlusions (see bottom right image of Fig. 2). We address this problem by using a fully balanced hierarchical line set representation to locally adjust the density of lines in the domain, so that the important lines are revealed in the final rendering. Our hierarchical representation has similarities to the one used for flow-based line seeding by Yu et al.[6], yet the construction algorithm we propose guarantees a balanced hierarchy. Even though Yu et al. demonstrate a low standard deviation of the numbers of streamlines per cluster from the optimum, i.e., when fully balanced, their construction algorithm can result in outliers with a high deviation. This can lead to very sparse or over-populated clusters, which can counteract reaching the prescribed line density in our approach.

The construction of the line set hierarchy is based on the grouping of a set of lines into similar sub-sets, by taking into account a given similarity measure. Thus, our work is also related to clustering approaches for line sets. For a thorough overview of the field let us refer to the recent evaluation of clustering approaches for streamlines using geometry-based similarity measures by Oeltze and co-workers [27]. The comparative study by Zhang et al. [28] provides a good overview of similarity measures using geometric distances between curves. Different clustering approaches and similarity measures for fiber tracts in Diffusion Tensor Imaging (DTI) data have been evaluated by Moberts et al. [29]. We use a variant of Agglomerative Hierarchical Clustering (AHC) with single linkage [30], which groups the initial lines in bottom-up fashion and determines the distance

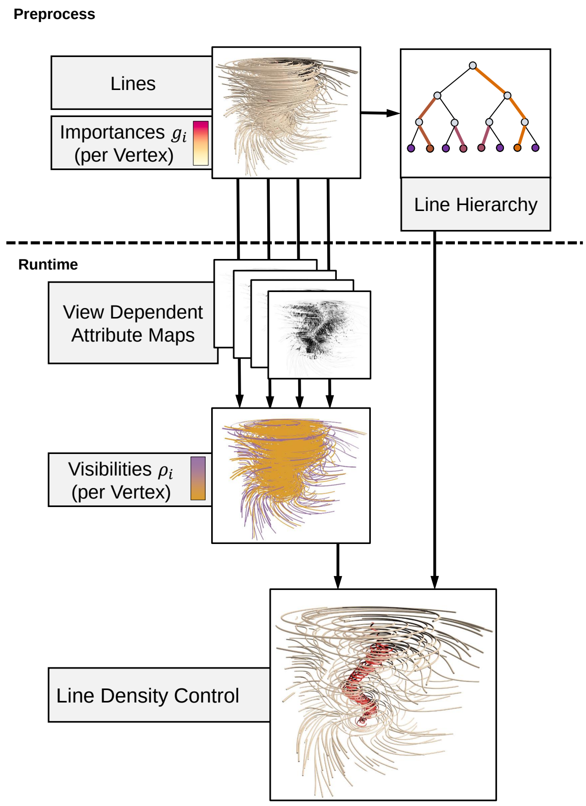


Figure 3: Overview of view-dependent importance driven line rendering.

between two clusters by the shortest distance between any pair of elements (one in each cluster). In contrast to the standard approach, where in every step exactly one pair of clusters is merged, we compute a perfect matching between the clusters in each step and simultaneously merge every cluster with some other cluster.

3. Overview

Our approach is comprised of a preprocess, and a two-stage view-dependent line rendering process that is executed repeatedly for each new view. An overview of the different parts of our approach is given in Fig. 3. Initially, we start from a set of lines which densely cover the flow domain. A line is represented by a sequence of vertices v_i, and to every vertex the line segment connecting this vertex and the next one in the sequence is associated. Every vertex is assigned an *importance* value $g_i \in [0,1]$, where higher values indicate higher importance. Throughout this work we use the local line curvature as importance measure.

On the given line set, a balanced line hierarchy is computed in a preprocess. The hierarchy is represented by a tree data structure and constructed in a bottom-up manner: Each single line is assigned to one leaf node, and level by level exactly two nodes are merged into one new node at the next coarser level. Depending on the initial number of lines, at each level at most one cluster might not find another cluster to merge with. In this case, this cluster is propagated to the next coarser level. The particular merging strategy to enforce a balanced tree is described in Sec. 4. Finally, at each inner node one line is selected. This line will be rendered as a representative for the set of lines stored at this node, if this set exceeds the locally required line density. In our work we usually select the line that is most similar to all other lines in this set, yet other choices can be incorporated as well (see Sec. 6).

In the first stage of the rendering process, we calculate for every line vertex a *visibility* value $\rho_i \in [0,1]$ based on the current view parameters and the importance information. The visibility value indicates if due to the rendering of the segment that is associated with the vertex a more important line segment is occluded, and it also takes into account additional criteria such as the importance difference as well as the overall per-pixel occupancy as proposed by Marchesin et al. [7, 8]. Günther et al. [9] have demonstrated that the visibility values can be computed automatically via opacity optimization. We found that by order-independent GPU line rendering in combination with screen-space blurring almost identical results can be achieved, yet because this approach avoids storing and sorting a per-pixel fragment list it scales better in the number of lines and the viewport resolution. Our algorithm used to compute the visibility values is described in Sec. 5.

In the second stage, the computed visibility values are used in the rendering of the line primitives: When a line segment is rendered, the visibility value of the vertex it is associated with is used to select a level in the precomputed line hierarchy. Only if at this level the line is the representative line for the cluster it belongs to, the line segment is rendered, otherwise it is discarded. This leads to an automatic local thinning of the less important occluding lines.

4. Line density control

In every frame, we seek to render a subset of all initial lines, so that this subset covers the domain as uniformly as possible for any given percentage of displayed lines. At the same time, when reducing the line density, lines with similar characteristics should be replaced by a good representative. Furthermore, since our approach does not remove entire lines but line segments, the fragmentation of lines should be kept as small as possible.

Let N denote the number of lines in the dataset. Our approach assigns to each line k a *visibility threshold* θ_k, so that for any given visibility value $\rho \in [0;1]$ the number of lines with $\theta_k < \rho$ is roughly equal to $\rho \cdot N$, i.e, $\sim 5\%$ of the lines satisfy $\theta_k < 0.05$, $\sim 25\%$ of the lines satisfy $\theta_k < 0.25$, and so on. To ensure that the lines satisfying $\theta_k < \rho$ cover the domain as uniformly as possible, hierarchical line clustering as described below is used. During rendering, for each vertex i of a line k, we compute its visibility ρ_i (see Sec. 5) and determine whether the line segment associated to it should be rendered by testing whether $\rho_i < \theta_k$.

To determine the visibility thresholds so that they adhere to the aforementioned requirements and, in particular, cause a uniform line removed, we build a fully balanced line hierarchy (i.e., a fully balanced binary tree). Our algorithm for building

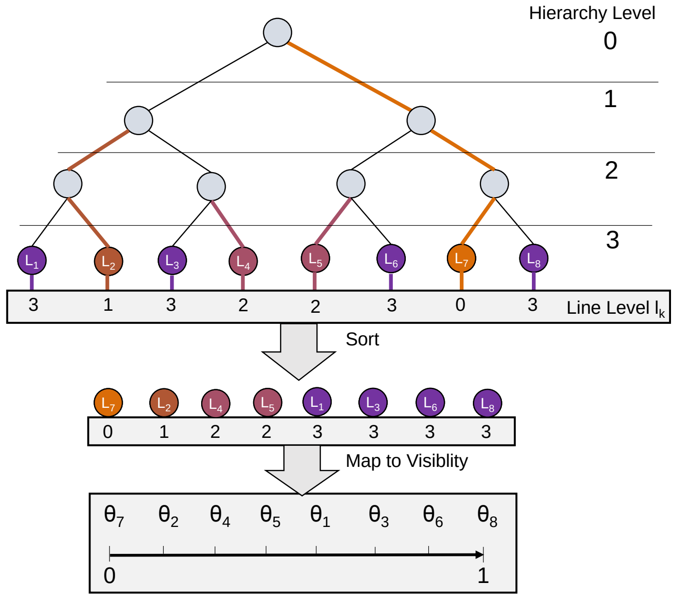


Figure 4: Construction of the line set hierarchy by pairwise node merging. Lines L_i are represented by the leaf nodes, bold edges and color indicate the selection of representative lines at each level. Nodes are finally linearized and sorted according to the coarsest level where they are representative, and the sorting order is mapped to visibility.

this hierarchy is similar to AHC—a greedy clustering approach that creates an arbitrary, unbalanced cluster hierarchy—yet it works globally and is able to produce a fully balanced hierarchy.

4.1. Balanced line hierarchy

We start by computing similarities $d_M(L_k, L_l)$ between every pair of lines L_k and L_l using the *mean of closest point distance* [31]:

$$\begin{aligned} d_M(L_k,L_l) &= \mathrm{mean}\left(d_m(L_k,L_l), d_m(L_l,L_k)\right), \\ \text{with } \; d_m(L_k,L_l) &= \operatorname*{mean}_{v_i \in L_k} \min_{v_j \in L_l} \|v_i - v_j\|. \end{aligned} \tag{1}$$

Here, v_i and v_j denote the vertices along a line. Note that any other pairwise distance metrics for lines can be used without affecting our algorithm, e.g. metrics based on Euclidean distances [32, 33], curvature and torsion signatures [34, 35], predicates for stream- and pathlines based on flow properties along these lines [36], or user-selected streamline predicates [37].

Now every line is represented by a leaf node, and pairs of nodes are grouped to generate the first coarse level of the tree hierarchy using a globally optimal approach: The similarities in Eq. (1) define a fully connected distance graph, with lines L_k as nodes and distances $d_M(L_k, L_l)$ as edge weights. On this graph, we compute a *minimum cost perfect matching*, which is a perfect matching that minimizes the sum of all included edge weights. The edges of the matching define the inner nodes on the first coarse level of the hierarchy, where each node stores the lines contained in the matched leaf nodes. In order to generate the remaining coarse levels of the line hierarchy, we recursively apply the perfect matching scheme on the remaining sets of lines. This reduces the number of sets by a factor of two in every step, until all lines are contained in one set, i.e., the root of the hierarchy. Fig. 4 illustrates the construction principle for a line set consisting of 8 elements.

For a fully connected graph with n nodes, the specific matching can be computed with a worst case runtime complexity of $O(n^3 \log n)$ using Edmond's Blossom algorithm [38]. In our implementation we use the LEMON library [39] to compute the matching. It employs priority queues to achieve a runtime complexity of $O(n^2 \log n)$. Even though this still doesn't indicate good scalability of the construction algorithm, we demonstrate in Sec. 6 that the hierarchy can be built within a few minutes for typical scenarios.

Every time a matching is computed and two line sets are merged, the distances between the sets of lines (i.e., the nodes in the distance graph) need to be updated. In classical hierarchical clustering schemes this is known as the linkage step, and several linkage criteria exist, such as single linkage, complete linkage and weighted-average. For a comparative evaluation let us refer to the work by Moberts et al. [29]. In our application, we use single linkage, i.e., the distance between two sets of lines is set to the minimum of all pairwise distances between the members from either set, since it produces the most spatial coherent merging of clusters. It is worth noting here that a fully balanced line hierarchy is computed regardless of the specific linkage used.

4.2. Hierarchical line visibility

The computed hierarchy serves as the basis for assigning visibility thresholds to each line. First, for each inner node we select a line which best represents all the lines belonging to that node (illustrated in Fig. 4 by the colored bold edges). We start at the first coarse level and select for every node a representative line from each pair of lines. Here we choose the longer one because it usually carries more information. The representatives for nodes on the remaining coarse levels are chosen in agreement with the previous choices, i.e., for each node we limit our choice to the representatives of both child nodes at the next finer level. In this case, we pick the representative which has the smaller average distance to all other lines represented by the node, i.e., according to the initial similarities d_M.

After all representatives have been selected, we assign to each line k the coarsest level l_k at which this line is a representative line in one of the nodes. We then assemble a list of 2-tuples (k, l_k), which contain one line-number/level pair for each line. This list is sorted according to the hierarchy levels l_k, and finally each line k gets assigned the visibility threshold $\theta_k = \frac{s}{N-1}$ depending on its position $s \in 0, \ldots, N-1$ in the sorted list. In the bottom part of Fig. 4 we illustrate the sorting of lines and the assignment of visibility thresholds to the lines.

Due to the particular construction scheme, the most representative lines—according to the hierarchy—are assigned very low thresholds and are very likely to be shown, whereas the less representative lines are assigned higher thresholds. Note that the sorting order is not unique because there are many lines that share the same hierarchy level. In practice, we rank all lines with equal l_k according to the similarity to other lines, such that very similar lines are removed first in the final visualization. Fig. 5 shows an example which demonstrates the use of the constructed hierarchy to uniformly thin out a given line set.

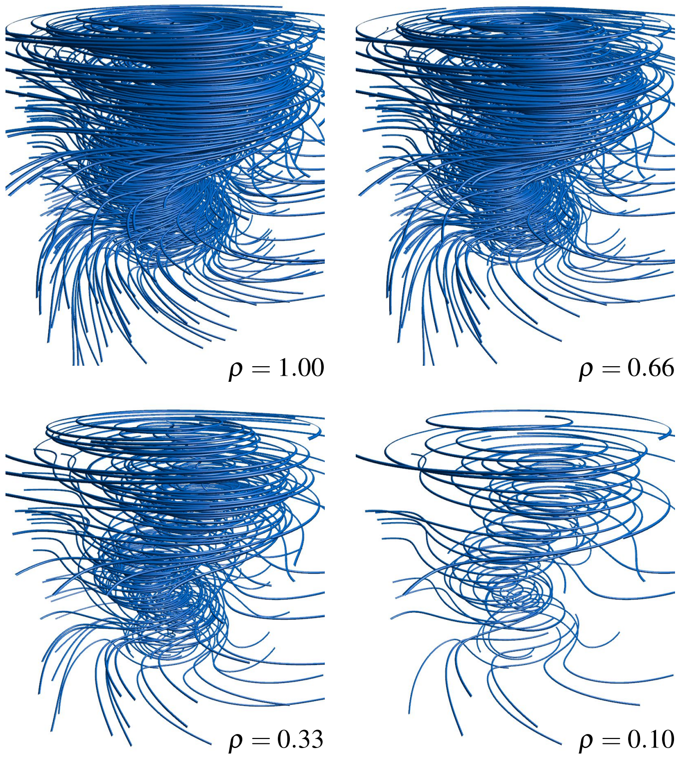


Figure 5: Line density control in the Tornado dataset using a balanced line hierarchy. Lines are thinned out uniformly according to the global visibility ρ.

5. Visibility computation via Per-Pixel Attributes

To determine the visibility of the line vertices, for each vertex a visibility value ρ_i is computed, and this value is matched against the visibility threshold of the line the vertex belongs to. If the line's threshold is larger than the vertex's visibility value, the vertex is discarded. Since the line set hierarchy already represents local and global line relationships, we can compute the vertex visibility on a per-pixel basis, by using screen-space projections of different line attributes. Conceptually, the computation is performed in three steps, which are illustrated in Fig. 6: First, by rendering all lines with attributes corresponding to importance and line direction, multiple screen-space textures are generated. These textures, respectively, contain per pixel the maximum importance, the number of fragments, the variance of line directions, and the depth of the fragment with the highest importance, along the view rays. Second, a Gaussian blur filter is applied to each texture to obtain a screen-space continuous distribution of attributes. Third, for each vertex the blurred textures are sampled at the corresponding screen-space position and the visibility attributes are used to compute a visibility value.

In the following, we describe the different visibility attributes, each in the range $[0;1]$, and their combination to form the final visibility values.

Maximum importance M: From all line fragments falling into a pixel, we find the highest importance value. After blur-

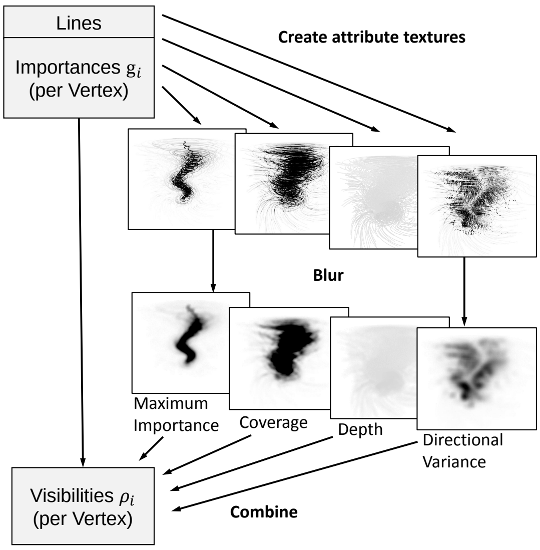


Figure 6: Visibility computation using per-pixel attributes. Four textures are generated by rasterizing different line attributes, the textures are blurred, and they are then accessed by the vertices (by sampling at their projected screen coordinates) to calculate per-vertex visibilities.

ring the resulting values, one can determine for any line vertex whether another vertex with a higher importance is in its vicinity. In this case, the visibility can be reduced accordingly to fade out less important lines in the close surrounding of important ones.

Depth D: By using the depth of the most important line fragment that maps onto a pixel, less important foreground lines can be faded out, and even cutaway views of the important structures can be realized.

Coverage C: Regions where many lines are projected onto the same location suffer from visual clutter. Therefore, in such regions the overall amount of lines should be reduced. We count the amount of fragments along each view ray and store the result, normalized by a fixed maximum count.

Directional variance V: The rational behind this measure is to thin out the foreground lines where they cause the occlusion of equally important lines with vastly different directions, and thus to emphasize the directional variance along the viewing direction. In regions where foreground and background lines follow a similar direction, their density will solely be steered by the importance, depth and coverage criteria. We thus allow the occlusion of equally important background lines if they have similar direction. Only if there is a directional variation, foreground lines are further thinned out to reveal this variation. To address this, the directional variance of all lines being projected into a pixel is computed, and in regions with a high directional variance the computed visibility values are decreased. In particular, we use the so-called *circular variance* of the (three-dimensional) view-space directions of all lines projected into a pixel. Given a set of unit-length direction vectors $\mathbf{d}_1, ..., \mathbf{d}_n$, the circular variance is defined as $\sigma(\mathbf{d}_1, ..., \mathbf{d}_n) = 1 - ||\frac{1}{n}\sum_i \mathbf{d}_i||$. If the directional variance is high, σ is close to one, while it is zero if all directions are equal.

Finally, the visibility values ρ_i are computed for each vertex i by combining the importance value g_i with the information stored in the texture maps. Let s_i denote the projected position of vertex i in screen-space, then ρ_i is calculated according to the following formula:

$$\rho_i = \frac{1}{1 + (1 - g_i^{\lambda}) \cdot P}$$
$$\text{with } P = m \cdot M(s_i) + c \cdot C(s_i) + v \cdot V(s_i) + d \cdot \text{less}(i, D(s_i)) \tag{2}$$

The positive scalar weights m, c, v and d are used to balance the contributions from the different texture maps and can be modified interactively by the user. The parameter $\lambda \geq 0$ controls the visibility of important lines. High values cause more important lines to become visible, effectively overruling the value of P. To incorporate the depth values stored in D, we perform a depth comparison, i.e., $\text{less}(i, D(s_i))$ is one if the depth of vertex i is smaller than $D(s_i)$, and zero otherwise.

6. Results and Discussion

To evaluate the quality and efficiency of our approach, we have performed a number of tests using different data sets. All our results were generated on a standard desktop computer equipped with an Intel 6×3.50 GHz processor, 32 GB RAM, and an NVIDIA GeForce GTX 970 graphics card. The viewport resolution was set to Full HD (1920×1080). The following datasets were used:

- **Tornado**: 330 randomly seeded streamlines in a synthetic flow resembling a tornado. The most interesting structure is the single vertical vortex core in the center of the domain.
- **Rings**: 450 magnetic field lines in the decay of magnetic knots, as studied by Candelaresi et al. [40]. In this specific time step, the lines assume the form of Borromean rings.
- **Heli**: 600 randomly seeded streamlines in an experimental flow of rotor wakes around a descending helicopter [41] (the helicopter geometry is not shown). The most prominent structures in this flow are the vortices formed by the helicopter blades.
- **Aneurysm I/II**: 4700/9200 streamlines in blood flows through two different aneurysms, as studied by Byrne et al. [42]. The lines were randomly seeded in the interior of the aneurysm and advected both forward and backward up to the boundary. For the hemodynamic analysis of these flows the vortices forming inside the aneurysms are of crucial interest.
- **Turbulence**: 80000 randomly seeded streamlines in a simulated turbulence field of resolution 1024^3 [43].

In Table 1, the second column gives the number of initially seeded streamlines and the average number of vertices per streamline. The line hierarchy and corresponding visibility thresholds are computed in a preprocess. The columns labeled

Table 1: Performance statistics for the precomputation of the visibility thresholds, the visibility computation via per-pixel attributes as well as the rendering of the final images for our test datasets. *In practice, we distribute the attribute map generation across several frames to favor smooth camera movements and refine the attribute map if time is available. **Edges in the graph with no influence on the result were removed in a preprocess.

		Precomputation		Visibility computation				Every frame	
Dataset	Lines (vertices per line)	Similarity [s]	Hierarchy [s]	Projections* [ms]	Blur [ms]	Visibility [ms]	Filter [ms]	Final Rendering [ms]	Sum [ms]
Tornado	330 (720)	3.2	0.16	0.9	0.27	0.04	0.5	1.6	3.31
Rings	450 (560)	3.8	0.22	0.9	0.26	0.05	0.9	2.6	4.71
Heli	600 (600)	12	0.38	1.3	0.27	0.07	1.0	1.9	4.54
Aneurysm I	4700 (410)	192	64	12.0	0.27	0.76	1.6	3.8	18.43
Aneurysm II	9200 (367)	382	562	22.5	0.27	1.23	3.2	6.5	33.7
Turbulence	80000 (220)	6923	25388 **	72	0.27	6.00	14.1	51.3	143.67

Precomputation summarize the timings for constructing the hierarchy for the test datasets. We give separate times for the computation of similarity values between line pairs (column *Similarity*, measured on the GPU) and building the line hierarchy (column *Hierarchy*, measured on the CPU), which includes perfect matching and the computation of the final visibility thresholds. In contrast to the computation of pairwise means of closest point distances between lines, which can be parallelized in a straight forward way on the GPU, perfect matching cannot easily be parallelized. Due to this, the runtime complexity of perfect matching can lead to the situation where it dominates the overall computation time, yet the timings indicate that for reasonable amounts of streamlines the overall time is still acceptable.

The remaining columns in Table 1 give additional times that are required at runtime for visualizing the datasets. Column *Projections* shows the time required for generating the visibility attribute via line rendering, *Blur* the time for blurring the attributes, and *Visibility* the time for computing the visibility values for each vertex. Since we render opaque lines from which we remove certain parts, lines can fall apart into many short pieces. To avoid the resulting visual clutter, short line segments are filtered out by a line-based, one-dimensional erosion and dilation operation in the GPU buffer storing the line geometry. Timings in column *Filter* refer to this filtering. Lastly, column *Final Rendering* shows the time required to generate the final image in each frame.

The rendering of opaque lines, which can change from frame to frame, can introduce unpleasant popping artifacts during animations. To alleviate this effect, in animations we temporarily allow for transparent lines. We smoothly blend the current per-vertex visibility values towards the new values, thus letting the solution converge when the camera stops. Let us refer the reader to the accompanying video for demonstrating this effect.

6.1. Visualization parameters

Our approach for computing the visibility values ρ_i allows the user to control several parameters (see Eq. 2), so that the visualizations can be adapted to specific datasets and requirements. Fig. 14 shows a number of examples demonstrating the effects of different attribute settings on the final visualizations. Fig. 7 demonstrates the effects that are achieved when only one of the 4 visibility attributes M, D, C, V (see Sec. 5) is used to compute per-vertex visibility values.

In the upper row of Fig. 7, we show two visualizations in which only the per-pixel maximum importance (M) and depth (D) values are used, respectively. It can be seen that in the first case the visibility of all unimportant lines around important lines is reduced, whereas only unimportant lines in front of important ones are removed in the latter case. Therefore, by using a combination of both values, the user can control the number of lines that are removed in the foreground and background of important lines.

In the bottom row, we compare the effects that can be achieved via the per-pixel coverage (C) and the directional variance (V), respectively. These values can be used to control the overall line density on the screen. On the one hand, the coverage values remove lines uniformly with the goal of achieving a uniform screen coverage, yet important lines remain unaffected due to the g_i-term in Eq. 2. On the other hand, the directional variance values can be used to reduce the line density only in regions where many lines with varying direction meet. Hence, these values can be used to reduce visual clutter in the visualizations.

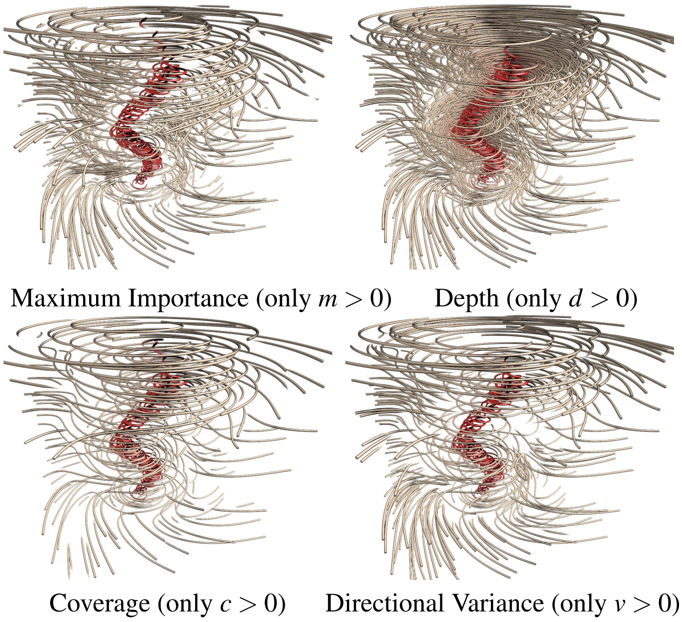


Figure 7: Line density control in the Tornado datasets via per-pixel maximum importance, depth, coverage, and directional variance. Only the corresponding weight in Eq. 2 is set to a positive value. All other weights are set to 0.

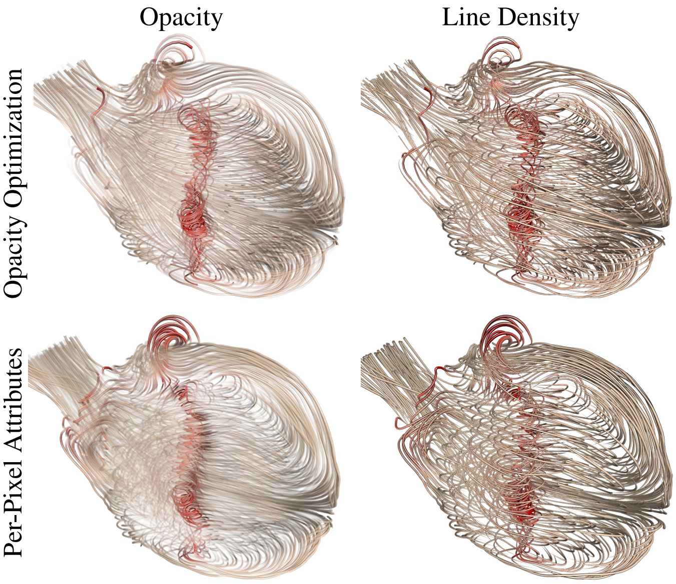


Figure 8: Visualizations of the Aneurysm I dataset. Top left: Opacity optimization + line opacity as by Günther et al. [9]. Top right: Opacity optimization [9] + line thinning as by our approach. Bottom left: Visibility attributes as by our approach and line opacity [9]. Bottom right: Visibility attributes and line thinning as by our approach.

Fig. 9 demonstrates the use of the directional variance to effectively visualize streamlines in regions where the flow field exhibits highly varying directions. When the directional variance is not used (left image), background lines are more or less entirely occluded by foreground lines with similar importance, even though they have vastly different orientations. By using the directional variance (right image), the density of both the foreground and background lines is reduced, so that the directional variance along the view rays is revealed and a good impression of the overall flow structure is achieved.

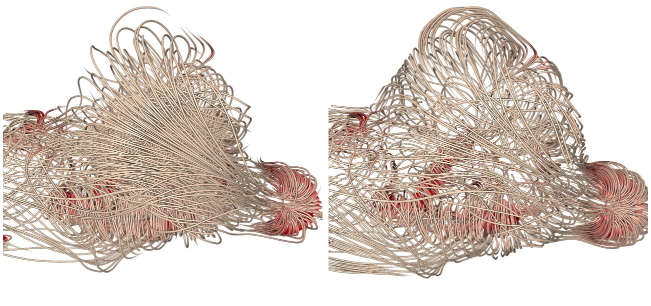


Figure 9: Effect of using the directional variance in the Aneurysm II dataset. Left: The directional variance is not used. Right: When the directional variance is used, the density of lines is reduced to emphasize the variation of the line directions along the view rays.

6.2. Cluster representatives

One major design decision underlying our approach is the use of a line hierarchy that can enforce a fairly even spatial distribution of the lines at all hierarchy levels. Therefore, at every inner node of the hierarchy the line most similar to all other lines of the corresponding cluster is usually selected as the cluster representative. Instead, however, any other selection criteria can be used, depending on the particular aspect of the data that should be retained across the hierarchy level. For instance, Fig. 10 demonstrates the difference between using the most similar and the most important line of each cluster as representative. While in the former case a better coverage of the domain and in particular the surrounding context is achieved, in the latter case some important lines are kept which are lost otherwise. Even though it is clear that many different criteria can be used and even combined in general, we did not analyze this option further due to our aforementioned design decision.

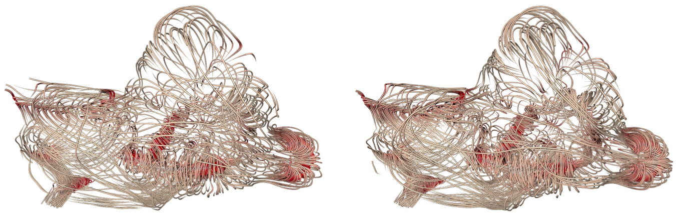


Figure 10: Different choices of cluster representatives at the inner nodes of the line hierarchy for the Aneurysm II dataset. Left: The most similar line is selected. Right: The most important line is selected.

6.3. Comparison

In Fig. 8, we compare our approach to opacity optimization [9]. We further show how the individual components of both methods can be even combined in a modular way. The computation of the visibility values is performed either via the global optimization of opacity (top row) or via our proposed approach based on screen-space projections (bottom row). Note that even though our approach does not operate globally, it generates visibility values that lead to visualizations which are very similar to the results obtained via opacity optimization.

The visibility values, computed in either way, are further processed by mapping them to opacity (left column) or using them to control locally the line density via our proposed approach (right column). In principle, both approaches reveal the important structures, but the differences are noticeable. The use of opacity makes it difficult to fully capture the spatial content, in particular the spatial relationships between the important lines and the unimportant lines in the foreground and in the vicinity of the important lines. In contrast, by adapting the line density, yet preserving line color and shading, our approach keeps unimportant lines as contextual spatial cues. Let us also refer here to the accompanying video, which demonstrates an even better 3D spatial perception when the user can interactively navigate around the line structures.

In Fig. 11, we include our results into the comparison provided by Günther et al. [9]. The approach of Marchesin et al. [7] is able to give a good overview of the flow, but cannot always reveal the important structures. Günther et al. [9] is able to emphasize important structures by locally adapting the transparency of line segments. Our approach can better reveal the spatial embedding of the focus regions into the surrounding, at the same time being able to show the focus region in a quite unobscured way.

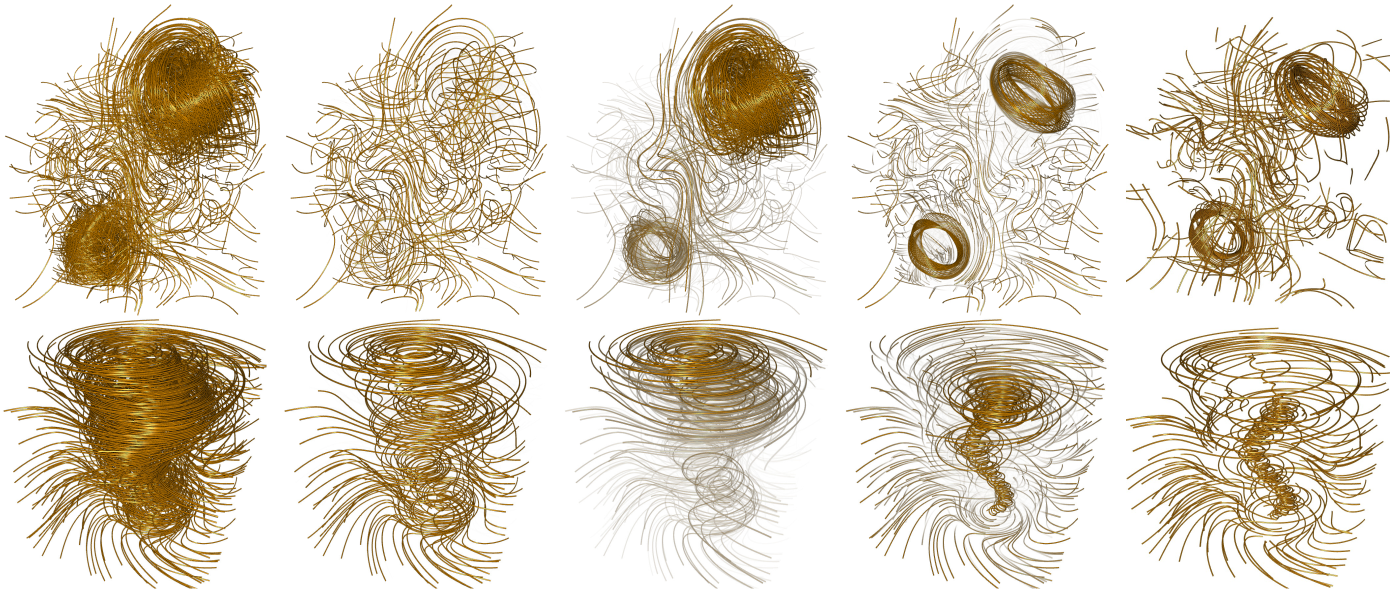


Figure 11: Rings (top) and Tornado (bottom) - left to right: all lines, Marchesin et al. [7], Günther et al. (2011) [44], Günther et al. (2013) [9], our approach

6.4. Line rendering

In the final image, lines are always rendered as shaded tubes of equal radius. The tubes are constructed on-the-fly in a geometry shader on the GPU. For every line vertex a new set of vertices is generated, and these vertices are displaced about the tube radius along the vertex's normal vector. Normals are assigned to the vertices initially as the normalized change of the unit tangent vectors.

The i-th vertex in the newly created vertex set is rotated about $360/n \cdot (i-1)$ degrees around the forward oriented tangent, where n is the number of created vertices per vertex. By constructing for every pair of consecutive vertex sets the quadrilateral connecting vertices i and $i+1$ from either set, the tube is constructed incrementally. The specific connectivity order ensures that the tubes do not twist unnecessarily. To visually separate the tubes, the angle between the surface normal and the vector along the view direction is used to draw silhouettes. Line segments with a tangent nearly parallel to the view direction are excluded from silhouette-drawing. Other rendering styles are also possible, see Stoll et al. [45] for example.

Into our visualization approach we have integrated different rendering styles for the loose line ends, which result from cutting away parts of the lines. We compare three different options in Fig. 13: a) We simply cut the lines without any further ado. b) We avoid an abrupt and visually disturbing break by letting the lines fade out over a short end-piece. This is achieved by quickly decreasing the opacity along these pieces. Regular line endings at the domain boundary are simply cut. c) We continuously narrow the lines over a short end-piece. The end-pieces are longer than the ones we use in b) to keep a smooth transition of the geometry. Of all the different possibilities, we found the last one to achieve the best spatial impression. It indicates where a line is fragmented, preserves the spatial location, and generates a smooth transition towards the line endings. Changing the line width, on the other hand, can conflict with perspective foreshortening, since a line bending away from the viewplane can appear with the same width when projected. Due to the cross-sectional tapering and the rather short line endings, however, only in very rare cases does this effect appear.

7. Conclusion and future work

In the future, one particular focus will be on the investigation of greedy algorithms for constructing an approximate balanced line hierarchy to reduce the pre-processing cost. Furthermore, we will analyse extensions of our approach to make it applicable to ensemble fields. In ensemble fields, multiple line sets are available and have to be analyzed regarding different properties. In particular, one is interested in finding similarities or outliers, which can be realized by building respective means into the construction of the line set hierarchy that is used in our work. In addition, ensemble-specific visibility attributes and density control mechanisms need to be investigated and incorporated into the line rendering approach, to be able to effectively reveal the ensemble variability.

8. Acknowledgments

We thank Steffen Oeltze and Juan Cebral for providing the bloodflow data. This work was supported by the European Union under the ERC Advanced Grant 291372 SaferVis: Uncertainty Visualization for Reliable Data Discovery.

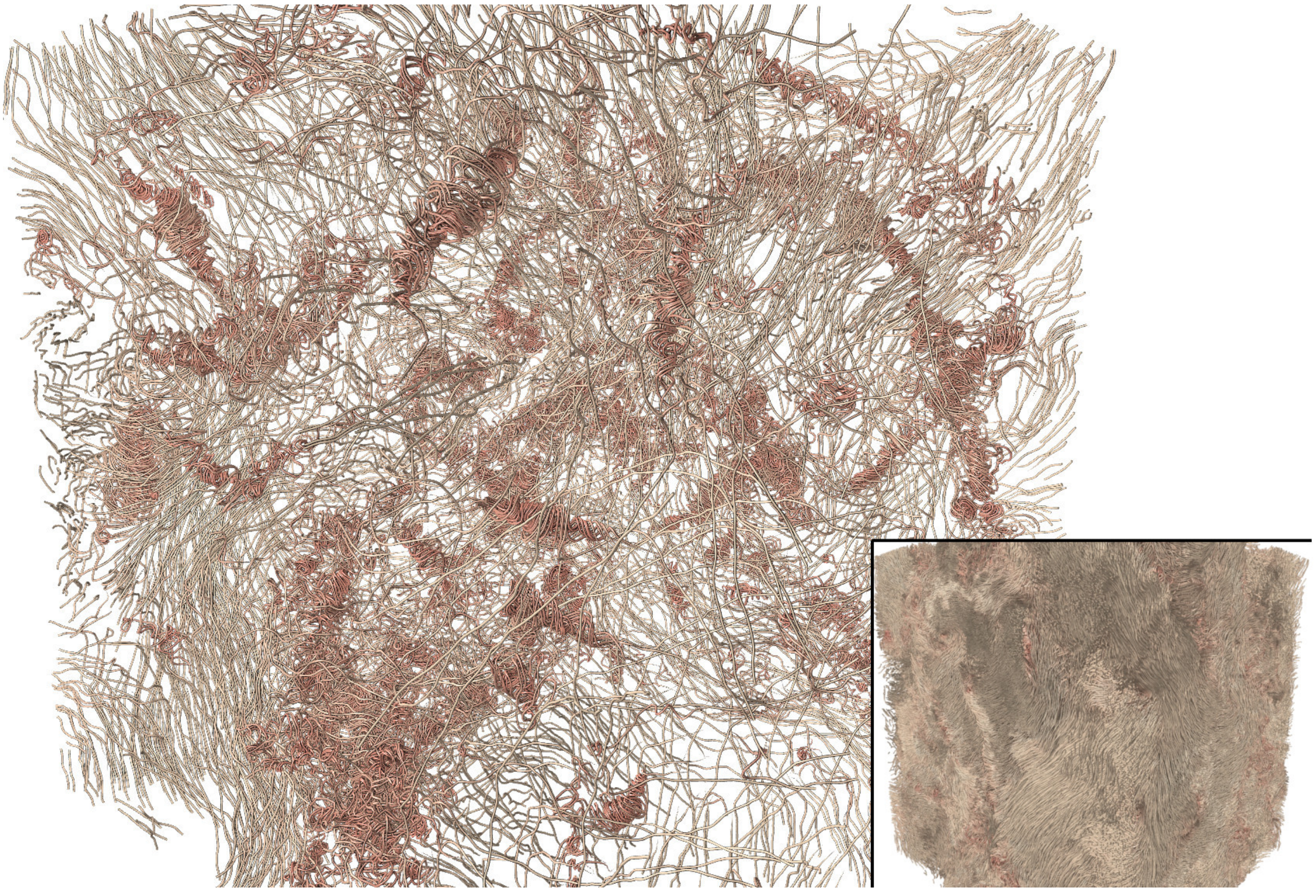


Figure 12: Controlling the density of 80.000 lines in the Turbulence dataset [43] via our approach (left). Complete line set (right).

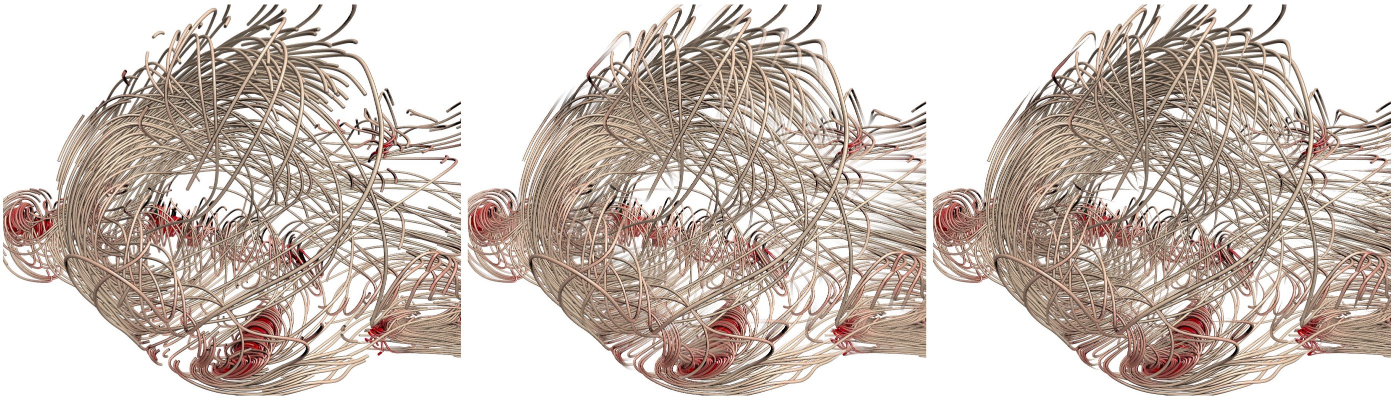
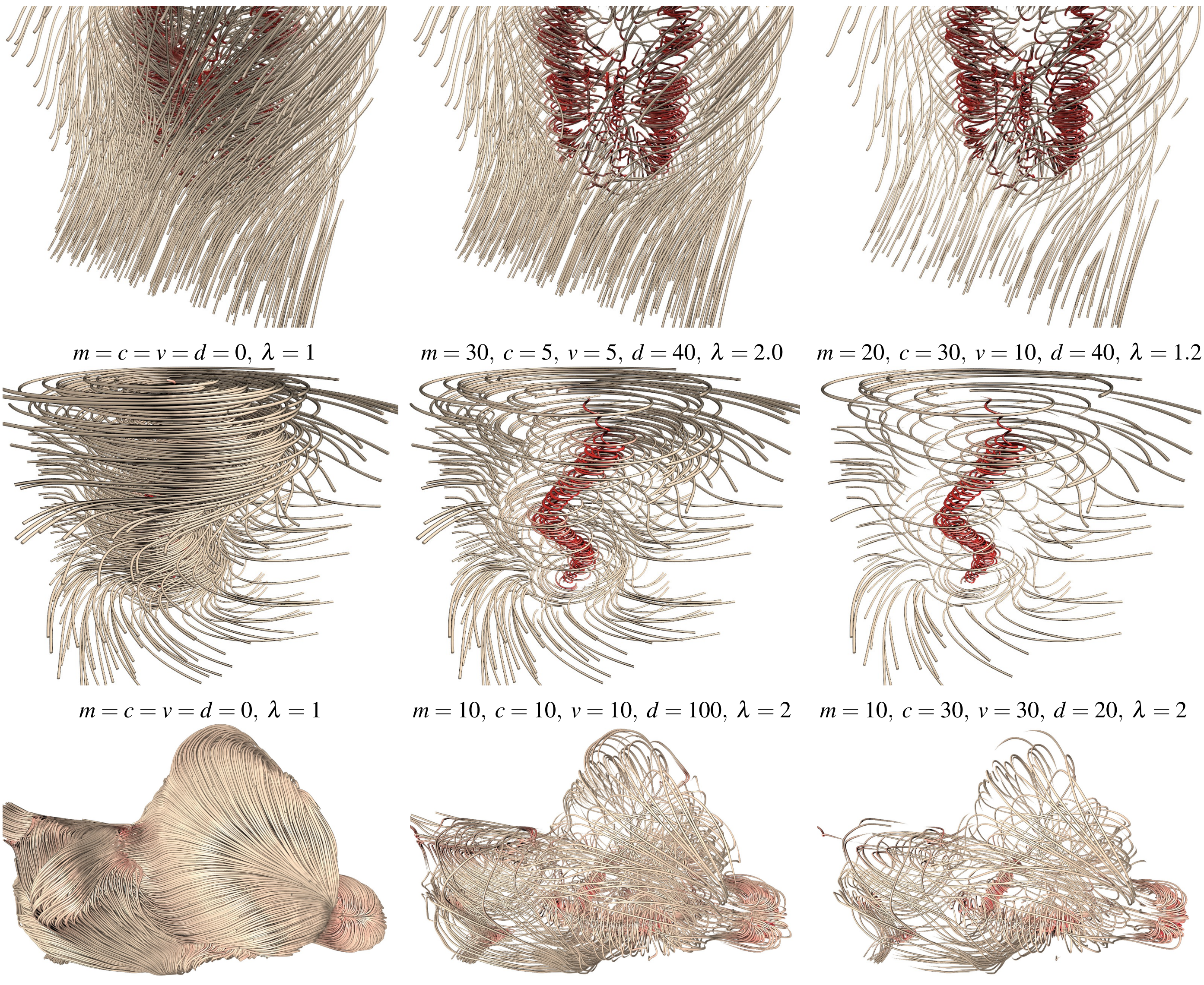


Figure 13: Visualization of the Aneurysm II dataset using different rendering styles for line endings at cut-offs resulting from partial line removal. (left) Cropping lines without transition, (middle) short transparent line ends and (right) reduced diameter at line ends—our preferred choice—.



$m = c = v = d = 0,\ \lambda = 1$ $m = 30,\ c = 5,\ v = 5,\ d = 40,\ \lambda = 2.0$ $m = 20,\ c = 30,\ v = 10,\ d = 40,\ \lambda = 1.2$

$m = c = v = d = 0,\ \lambda = 1$ $m = 10,\ c = 10,\ v = 10,\ d = 100,\ \lambda = 2$ $m = 10,\ c = 30,\ v = 30,\ d = 20,\ \lambda = 2$

$m = c = v = d = 0,\ \lambda = 1$ $m = 40,\ c = 44,\ v = 52,\ d = 170,\ \lambda = 4$ $m = 50,\ c = 35,\ v = 20,\ d = 150,\ \lambda = 4$

Figure 14: Demonstration of different parameter combinations for the visibility computation in Eq. 2. From top to bottom: Heli, Tornado, Aneurysm II.

[1] Weiskopf D, Erlebacher G. Overview of flow visualization. The Visualization Handbook 2005;:261–78.

[2] McLoughlin T, Laramee RS, Peikert R, Post FH, Chen M. Over two decades of integration-based, geometric flow visualization. Computer Graphics Forum 2010;29(6):1807–29. doi:10.1111/j.1467-8659.2010.01650.x.

[3] Verma V, Kao D, Pang A. A flow-guided streamline seeding strategy. In: Proc. IEEE Visualization. ISBN 1-58113-309-X; 2000, p. 163–70.

[4] Ye X, Kao D, Pang A. Strategy for seeding 3D streamlines. In: Proc. IEEE Visualization 2005. 2005, p. 471–8. doi:10.1109/VISUAL.2005.1532831.

[5] Chen Y, Cohen J, Krolik J. Similarity-guided streamline placement with error evaluation. IEEE Transactions on Visualization and Computer Graphics 2007;13(6):1448–55. doi:10.1109/TVCG.2007.70595.

[6] Yu H, Wang C, Shene CK, Chen JH. Hierarchical streamline bundles. IEEE Transactions on Visualization and Computer Graphics 2012;18(8):1353–67. doi:10.1109/TVCG.2011.155.

[7] Marchesin S, Chen CK, Ho C, Ma KL. View-dependent streamlines for 3D vector fields. IEEE Transactions on Visualization and Computer Graphics 2010;16(6):1578–86. doi:10.1109/TVCG.2010.212.

[8] Ma J, Wang C, Shene CK. Coherent view-dependent streamline selection for importance-driven flow visualization. Proc SPIE 8654, Visualization and Data Analysis 2013;doi:10.1117/12.2001887.

[9] Günther T, Rössl C, Theisel H. Opacity optimization for 3D line fields. Proc ACM SIGGRAPH 2013;32(4):120.

[10] Viola I, Kanitsar A, Gröller ME. Importance-driven volume rendering. In: Proc. IEEE Visualization. 2004, p. 139–45.

[11] Turk G, Banks D. Image-guided streamline placement. In: Proc. ACM SIGGRAPH. ISBN 0-89791-746-4; 1996, p. 453–60. doi:10.1145/237170.237285.

[12] Mao X, Hatanaka Y, Higashida H, Imamiya A. Image-guided streamline placement on curvilinear grid surfaces. In: Proc. IEEE Visualization. ISBN 1-58113-106-2; 1998, p. 135–42.

[13] Mattausch O, Theußl T, Hauser H, Gröller E. Strategies for interactive exploration of 3D flow using evenly-spaced illuminated streamlines. In: Proceedings of the 19th Spring Conference on Computer Graphics. New York, NY, USA: ACM. ISBN 1-58113-861-X; 2003, p. 213–22. doi:10.1145/984952.984987.

[14] Schlemmer M, Hotz I, Hamann B, Morr F, Hagen H. Priority streamlines: A context-based visualization of flow fields. In: Proc. EG/IEEE VGTC EuroVis. 2007, p. 227–34.

[15] Jobard B, Lefer W. Creating evenly-spaced streamlines of arbitrary density. In: Proc. Eurographics Workshop on Visualization in Scientific Computing. ISBN 978-3-211-83049-9; 1997, p. 43–55. doi:10.1007/978-3-7091-6876-9_5.

[16] Mebarki A, Alliez P, Devillers O. Farthest point seeding for efficient placement of streamlines. In: Proc. IEEE Visualization. 2005, p. 479–86. doi:10.1109/VISUAL.2005.1532832.

[17] Liu Z, Moorhead R, Groner J. An advanced evenly-spaced streamline placement algorithm. IEEE Transactions on Visualization and Computer Graphics 2006;12(5):965–72. doi:10.1109/TVCG.2006.116.

[18] Li L, Shen HW. Image-based streamline generation and rendering. IEEE Transactions on Visualization and Computer Graphics 2007;13(3):630–40. doi:10.1109/TVCG.2007.1009.

[19] Li L, Hsieh HH, Shen HW. Illustrative streamline placement and visualization. In: Proc. IEEE PacificVis. 2008, p. 79–86. doi:10.1109/PACIFICVIS.2008.4475462.

[20] Spencer B, Laramee RS, Chen G, Zhang E. Evenly spaced streamlines for surfaces: An image-based approach. Computer Graphics Forum 2009;28(6):1618–31.

[21] Rosanwo O, Petz C, Prohaska S, Hege HC, Hotz I. Dual streamline seeding. Proc IEEE PacificVis 2009;0:9–16.

[22] Xu L, Lee TY, Shen HW. An information-theoretic framework for flow visualization. IEEE Transactions on Visualization and Computer Graphics 2010;16(6):1216–24. doi:10.1109/TVCG.2010.131.

[23] McLoughlin T, Jones M, Laramee R, Malki R, Masters I, Hansen C. Similarity measures for enhancing interactive streamline seeding. IEEE Transactions on Visualization and Computer Graphics 2012;19(8):1342–53. doi:10.1109/TVCG.2012.150.

[24] Tong X, Chen CM, Shen HW, Wong PC. Interactive streamline exploration and manipulation using deformation. In: IEEE PacificVis. 2015, p. 1–8. doi:10.1109/PACIFICVIS.2015.7156349.

[25] Lee TY, Mishchenko O, Shen HW, Crawfis R. View point evaluation and streamline filtering for flow visualization. In: Proc. IEEE PacificVis. 2011, p. 83–90. doi:10.1109/PACIFICVIS.2011.5742376.

[26] Krüger J, Schneider J, Westermann R. Clearview: An interactive context preserving hotspot visualization technique. IEEE Transactions on Visualization and Computer Graphics 2006;12(5):941–8. doi:10.1109/TVCG.2006.124.

[27] Oeltze S, Lehmann D, Kuhn A, Janiga G, Theisel H, Preim B. Blood flow clustering and applications in virtual stenting of intracranial aneurysms. IEEE Transactions on Visualization and Computer Graphics 2014;20(5):686–701. doi:10.1109/TVCG.2013.2297914.

[28] Zhang Z, Huang K, Tan T. Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. In: Proc. International Conference on Pattern Recognition; vol. 3. 2006, p. 1135–8. doi:10.1109/ICPR.2006.392.

[29] Moberts B, Vilanova A, van Wijk J. Evaluation of fiber clustering methods for diffusion tensor imaging. In: Proc. IEEE Visualization. 2005, p. 65–72. doi:10.1109/VISUAL.2005.1532779.

[30] Jain AK. Data clustering: 50 years beyond k-means. Pattern Recogn Lett 2010;31(8):651–66.

[31] Corouge I, Gouttard S, Gerig G. Towards a shape model of white matter fiber bundles using diffusion tensor MRI. In: IEEE International Symposium on Biomedical Imaging: Nano to Macro. 2004, p. 344–347 Vol. 1. doi:10.1109/ISBI.2004.1398545.

[32] Chen Y, Cohen J, Krolik J. Similarity-guided streamline placement with error evaluation. IEEE Transactions on Visualization and Computer Graphics 2007;13(6):1448–55.

[33] Rössl C, Theisel H. Streamline embedding for 3D vector field exploration. IEEE Transactions on Visualization and Computer Graphics 2012;18(3):407–20.

[34] Yu H, Wang C, Shene CK, Chen JH. Hierarchical streamline bundles. IEEE Transactions on Visualization and Computer Graphics 2012;18(8):1353–67.

[35] McLoughlin T, Jones MW, Laramee RS, Malki R, Masters I, Hansen CD. Similarity measures for enhancing interactive streamline seeding. IEEE Transactions on Visualization and Computer Graphics 2013;19(8):1342–53.

[36] Born S, Pfeifle M, Markl M, Scheuermann G. Visual 4D MRI blood flow analysis with line predicates. In: IEEE PacificVis Symposium. 2012, p. 105–12.

[37] Salzbrunn T, Scheuermann G. Streamline predicates. IEEE Transactions on Visualization and Computer Graphics 2006;12(6):1601–12.

[38] Edmonds J. Paths, trees, and flowers. Canadian Journal of mathematics 1965;17(3):449–67.

[39] Lemon graph library. `http://lemon.cs.elte.hu/trac/lemon`; 2004 (accessed February 29, 2016).

[40] Candelaresi S, Brandenburg A. Decay of helical and nonhelical magnetic knots. Phys Rev E 2011;84. doi:10.1103/PhysRevE.84.016406.

[41] Yu YH, Tung C, van der Wall B, Pausder HJ, Burley C, Brooks T, et al. The HART-II test: Rotor wakes and aeroacoustics with higher-harmonic pitch control (HHC) inputs - the joint German/French/Dutch/US project. 58th Annual Forum of the AHS, Montreal, CN 2002;.

[42] Byrne G, Mut F, Cebral J. Quantifying the large-scale hemodynamics of intracranial aneurysms. Amer J of Neuroradiology 2014;35:333–8. doi:10.3174/ajnr.A3678.

[43] Aluie H, Eyink G, E. V, Kanov SCK, Burns R, Meneveau C, et al. Forced MHD turbulence data set. `http://turbulence.pha.jhu.edu/docs/README-MHD.pdf`; 2013 (accessed May 13, 2016).

[44] Günther T, Bürger K, Westermann R, Theisel H. A view-dependent and inter-frame coherent visualization of integral lines using screen contribution. In: Proceedings of Vision, Modeling, and Visualization. Eurographics Asssociation; 2011, p. 215–22.

[45] Stoll C, Gumhold S, Seidel HP. Visualization with stylized line primitives. In: Visualization, 2005. VIS 05. IEEE. IEEE; 2005, p. 695–702.