Technische Universität München
Fakultät für Informatik
Lehrstuhl für Computer Grafik und Visualisierung

Interactive Visualization of Large 3D Line Sets

Mathias Kanzler

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende:	Prof. Dr.-Ing. L. Leal-Taixé
Prüfer der Dissertation:	1. Prof. Dr. R. Westermann
	2. J.-Prof. Dr.-Ing. habil. K. Lawonn, Friedrich-Schiller-Universität Jena

Die Dissertation wurde am 16.10.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20.01.2020 angenommen.

To my family and friends

Abstract

Three-dimensional vector fields can be found in many research areas, be it medicine, meteorology, or fluid mechanics. Curved lines are an effective visualization tool to display these vector fields indirectly. Due to the increasing resolution of vector fields, it is necessary to increase the density of lines in order to capture as many of the contained details as possible. However, a high number of lines presents new challenges in three-dimensional visualization. In this thesis, we propose methods that dynamically generate visualizations from the given line data sets to illustrate the relationship of flow trends and relevant flow features.

In order to increase the proportion of relevant information conveyed through the visualization, we introduce a method that allows the camera to be moved interactively and freely, while at the same time showing or hiding line segments in a smart way: line segments are selected which, depending on the current viewpoint, are most representative and at the same time contain interesting flow features while limiting coverage of other lines. To thin out a certain number of lines depending on their relevance, a hierarchy of all lines is set up in a pre-processing step. Based on the geometric distance of the lines to each other, we present a graph-based global matching algorithm that generates this hierarchical structure. In addition to the hierarchical order of the lines, line segments are weighted according to their importance. At runtime, the projection of the line data onto the image plane is first evaluated taking into account various features, such as coverage, variance, and importance. Then, for the final representation, the line segments are thinned out depending on the determined projection characteristics and the precalculated hierarchy.

Another approach for reducing coverage of lines without removing line segments is to visualize them semi-transparently. That way, covered lines can still be observed visually. The classification of the lines with regard to their spatial position within the semi-transparent layers, however, is impeded. We therefore suggest the use of global illumination effects such as indirect illumination or shadows. These effects come with a high computational effort, especially for complex scenes. To counteract, we propose an algorithm that discretizes the line data segment by segment and stores them in a data structure that allows for fast access at voxel level. Location-based access to line segments enables efficient ray casting accompanied by advanced illumination effects. As line

segments along the ray are implicitly sorted by distance to the viewer, efficient alpha compositing of semi-transparent line segments is rendered possible.

When analyzing a large number of lines, lines are often clustered to simplify the data and to help identifying trends. However, the relationships between clusters are often unclear. We present a system that allows for interactive visual exploration of hierarchically structured clusters. The generated clusters are displayed in a tree structure, from which clusters can be selected for a three-dimensional comparison. Different characteristics of the lines from the respective clusters are measured and stored as scalar volumes. Hulls are derived from these scalar volumes, which provide a spatial overview of the extent of the line clusters. Starting from this, additional lines can be queried in selected areas taking into account the local properties such as density or variance. This makes it possible to visually compare different clusters in a targeted and detailed way.

Zusammenfassung

Dreidimensionale Vektorfelder sind In den unterschiedlichsten Anwendungsbereichen zu finden, sei es Medizin, Meteorologie oder Strömungsmechanik. Gekrümmte Linien sind hierbei ein effektives Werkzeug der Visualisierung, um diese Vektorfelder indirekt zu veranschaulichen. Eine zunehmend höhere Auflösung der Vektorfelder erfordert es, die Dichte der Linien zu erhöhen, um möglichst viele der enthaltenen Details zu erfassen. Dies bringt in der dreidimensionalen Darstellung allerdings neue Herausforderungen mit sich. In dieser Arbeit werden Möglichkeiten vorgestellt, die dynamisch aus den gegebenen Liniendatensätzen Visualisierungen erzeugen, die die Zusammenhänge der Strömungen und besondere Strömungseigenschaften veranschaulichen.

Um den Anteil der enthaltenen relevanten Informationen in der Visualisierung zu erhöhen, schlagen wir eine Methode vor, die es erlaubt, die Kamera interaktiv frei zu bewegen, gleichzeitig aber Liniensegmente aus- oder einblendet. Es werden Liniensegmente bestimmt, die, abhängig von der gewählten Ansicht, möglichst repräsentativ sind und gleichzeitig interessante Strömungseigenschaften beinhalten. Um eine bestimmte Anzahl an Linien, abhängig von deren Relevanz, ausdünnen zu können, wird in einem Vorverarbeitungsschritt eine Hierarchie unter allen Linien bestimmt. Basierend auf der geometrischen Distanz der Linien zueinander, präsentieren wir einen Graph-basierten globalen Matching-Algorithmus, der die hierarchische Struktur erzeugt. Zusätzlich zur hierarchischen Einordnung der Linien können Liniensegmente hinsichtlich ihrer Wichtigkeit gewichtet werden. Zur Laufzeit wird zuerst, unter Berücksichtigung verschiedener Merkmale, die Projektion der Linieneigenschaften auf die Bildebene ausgewertet. Anschließend wird für die finale Darstellung eine Ausdünnung der Liniensegmente in Abhängigkeit der ermittelten Projektionsmerkmale und der vorberechneten Hierarchie vorgenommen.

Um Verdeckungen von Linien zu reduzieren ohne Liniensegmente zu entfernen, können diese auch halbtransparent dargestellt werden. Auch tiefer liegende Linien können so noch wahrgenommen werden. Die visuelle Einordnung der Linien hinsichtlich deren räumlicher Position innerhalb der halbtransparenten Schichten wird hierbei jedoch erschwert. Daher schlagen wir die Anwendung von globalen Beleuchtungseffekten wie indirekter Beleuchtung oder Schatten vor, welche insbesondere für komplexe Szenen einen hohen Rechenaufwand bedeuten. Hierfür haben wir einen spezialisierten Algorithmus entwickelt, der die Liniendaten segmentweise

diskretisiert und in einer Datenstruktur ablegt, die einen schnellen Zugriff auf Voxel-Ebene erlaubt. Durch die ortsbezogene Zugriffsart auf Liniensegmente ist eine effiziente Durchführung von Raycasting möglich. Da entlang des Strahls die Liniensegmente implizit nach Abstand zum Betrachter sortiert sind, ist ein effizientes Alpha-Compositing halbtransparenter Liniensegmente möglich.

Bei der Analyse einer großen Anzahl an Linien werden häufig Linien zu Clustern zusammengefasst, um die Darstellung zu vereinfachen und Trends zu erkennen. Die Beziehungen der Cluster zueinander sind jedoch häufig unklar. Deshalb stellen wir ein System vor, das eine interaktive visuelle Exploration von hierarchisch aufgebauten Clustern ermöglicht. Die erzeugten Cluster werden in einer Baumstruktur dargestellt, aus der Cluster für den Vergleich ausgewählt werden können. Aus den Linien der jeweiligen Cluster werden verschiedene Beschaffenheiten gemessen und als Skalarvolumina abgespeichert. Aus diesen werden wiederum Hüllen abgeleitet, die einen räumlichen Überblick über die Ausdehnung der Liniencluster verschaffen. Davon ausgehend können in ausgewählten Bereichen zusätzliche Linien unter Berücksichtigung der lokalen Beschaffenheit eingeblendet werden. Dadurch ist es möglich, unterschiedliche Cluster gezielt und detailliert visuell zu vergleichen.

Acknowledgments

First and foremost, I would like to thank my doctoral advisor Prof. Dr. Rüdiger Westermann. I knew him from the beginning of my computer science study when he was teaching Java. He has given me the possibility to learn more about computer graphics and visualization during my bachelor's and master's thesis. Later, he made it possible for me to start research in the field of scientific visualization. During my PhD, I always felt supported and I enjoyed the exchange of ideas with him. Whenever I had a problem, he was available, anytime.

I also want to thank my co-authors Florian Ferstl and Marc Rautenhaus who contributed to the success of my papers and were always up for interesting discussions.

During my time at the chair, I not only met intelligent researchers who provided useful suggestions and valuable ideas, but also friends.

I specially thank Marie-Lena Eckert and Alexander Kumpf for helping me particularly during the final phase of my PhD. Furthermore, I want to thank my colleagues Johannes Kehrer, Ismail Demir, Mihaela Jarema, Michael Kern, Henrik Masbruch, Bianca Tost, Seyedbehdad Ghaffari, Christian Reinbold, Sebastian Weiss, Steffen Wiewel, Lukas Prantl, Kiwon Um, Rachel Chu as well as Susanne Weitz and Sebastian Wohner. Sebastian Wohner not only helped me with technical problems, he also offered his gardening skills and embellished my office with a banana plant.

I also would like to thank Steffen Oeltze-Jafra for providing me with interesting medical data sets.

Finally, I am enormously thankful to my girlfriend Sarah, my parents, and my sister for always supporting and encouraging me.

This work was funded by the European Union under the ERC Advanced Grant 291372—SaferVis: Uncertainty Visualization for Reliable Data Discovery.

Contents

1 Introduction

Fluids are omnipresent in our everyday lives. Be it wine in a glass, fuel in combustion engines, or the air we breathe — complex swirls and turbulent flows but also regular patterns can be found. Even if these complex processes are usually not actively perceived (as long as the wine remains in the glass), they are part of our environment.

Usually the transport routes cannot be seen in homogeneous gases or liquids. Only when media with different optical properties meet, vortices and streams become visible. When water meets air, one can see eddies on the water surface, for example. In smoke, microscopically small particles are usually responsible for the visibility of the structures. Based on this, tracking of artificial particles was discovered to be a useful tool in flow visualization. By recording paths of these particles, the resulting trajectories intuitively reveal structural patterns in flow fields. These flow fields are the subject of study in a wide variety of scientific disciplines and also have very different characteristics and extents. For example, wind speed fields in climate research span many kilometers, whereas in medical research blood flow in aneurysms of few millimeters in diameter are investigated. When visualizing flow fields, however, the underlying resolution is also relevant as it determines the potential amount of information contained. As advances in simulation and measurement allow ever increasing resolution of flow fields, the number of particle trajectories needed to cover as many aspects of the flow field as possible increases as well. However, the representation of a densely seeded line set poses new challenges, since there are increasing occlusions. Lines located in the inside of the domain may never become visible to the viewer. Even partially occluded lines can pose problems when the view gets cluttered resulting in a visually complex image that is hard to comprehend and paths of lines become difficult to follow. Then again, a reduced number of lines involves the risk of missing important structures in the flow field. Focusing only on interesting structures, however, leads to an incomplete picture of

the overall flow structure. Additionally, generated visualizations still have to be understood by humans and have to be processed by their visual system. By creating artificial images containing elements that are not common in real world, special care has to be taken by replicating familiar visual impressions. Although humans differ in their perception, there are similarities in the psychological process of gaining knowledge from the images seen. Thus, visualizations can be designed more effectively and more expressively if human perception is taken into account.

Altogether, having this in mind, we propose methods that allow visualizations of dense line sets comprising reduced occlusion while conserving context information around important structures. In our visualisations, human perception is considered based on depth cues, expressive illumination effects, and by giving continuous visual feedback during interactive exploration.

Our methods are generally applicable to line data sets of various forms. Trajectories described by massless particles that are transported in a flow field are called *pathlines*. Pathlines therefore can be used to illustrate possible transport paths. However, to observe the current flow direction inside the domain, massless particles can be transported in a snapshot of the flow field. These paths are called *streamlines* and show a single timestep of the time varying flow. Although visualizations comprising lines are common in different scientific domains, there are variations regarding the relevance of certain line patterns and attributes. For example, fast ascending particles can represent interesting formations in meteorology, whereas in the context of the classification of aneurysms the existence of vortices can be significant for the identification of shear-wall stress. Therefore, our methods are designed to improve the overall information gain independent of the underlying data and to additionally process a weighting of line segments based on domain-specific attributes as following. Our first method, presented in Chapter 5 [KFW16], on the one hand generates a line hierarchy, which is based purely on geometric line properties, on the other hand evaluates attributes per line segment and takes them into account when thinning out the lines. Furthermore, in Chapter 6 [KRW19] we introduce a general method that reinterprets line data and stores it in voxels. The new line arrangement enables efficient rendering of semi-opaque lines while simultaneously applying ambient occlusion and shading effects. The method for interactive exploration of line clusters presented in Chapter 7 [KW18] primarily considers the geometric properties of lines. Different needs of various application areas can be addressed by means of a user-controlled brush. If a special requirement demands support for interactive exploration based on attributes attached to line segments, the developed framework can be adapted to the extent that consistency volumes can be derived, thus supporting brush guidance based on these attributes.

1.1. Contributions

In this thesis, we propose a set of novel interactive visualization techniques for large line sets. These techniques comprise improvements considering line density control based on a hierarchical grading of lines in combination with view dependent measurements as well as manual adjustments of line density using smart brushes. Large line sets pose a problem when costly effects like transparency or global illumination effects are included. Considering this, our ray casting algorithm provides an alternative to commonly used rasterization-based rendering approaches, enabling these effects in interactive framerates. Specifically, we present the following visualization techniques:

- We show how *line density control in screen-space* helps to reduce visual clutter and at the same time brings out important lines by reducing overlap. This can be accomplished by combining a precomputed *balanced line hierarchy* with a view dependent grading. A progressively applied graph-based perfect matching algorithm produces a line hierarchy suitable for varying density, ensuring representative lines ranked higher in the hierarchy. To control the density automatically and interactively for a large number of lines, we show how a set of projections, measuring maximum importance, directional variance, coverage, and depth of important structures helps to rank line segments.
- We show how our *voxel-based line ray casting framework* enables transparency and advanced illumination effects in interactive framerates for a large number of lines. In a preprocess, lines are split at voxel boundaries into line segments that are stored using the discretized offset of intersection points inside the voxel to ensure a compact representation. At runtime, the voxel grid is traversed by a ray caster, enabling transparency through implicitly sorted voxels. Shadows and ambient occlusions greatly improve spatial understanding of line sets and can be calculated at runtime due to the possibility of accessing line segments in world space.
- We show how our *interactive visualization of large 3D line sets* gives insights through a combination of derived cluster consistency fields and an automatically refined brush. Mayor trends in large line sets are often identified by splitting the line set into multiple clusters. Cluster consistency fields comprising information about line density, trend direction, and directional variance are used to show information about clusters in 3D by deriving hulls. This summarizing visualization serves as a basis for manual queries for detailed line representations by using a brush-like tool. Based on this, we present a mechanism which refines and steers that line selection in a way that interesting structures are emphasized.

1.2. Thesis Outline

The structure of this cumulative thesis is as follows. In Chapter 1, we present our main contributions after a short introduction and list the publications that are part of this thesis. Related state of the art works with focus on streamline selection techniques, explorative techniques, voxel-based rendering, and human perception are discussed in detail in Chapter 2. Chapter 3 explains the fundamentals of line rendering and illumination in computer graphics. Moreover, relevant depth perception cues are listed and it is shown how these are incorporated to gain effective visualization techniques. Further applications and use cases by customizing presented approaches are depicted in Chapter 4. Our work of [KFW16, KRW19, KW18] is listed in Chapter 5, 6, and 7, respectively. Finally, Chapter 8 concludes this thesis and presents some ideas for future work. The published/accepted version of the papers can be found in the appendix.

1.3. List of Publications

- Mathias Kanzler, Florian Ferstl, Rüdiger Westermann:
  Line Density Control in Screen-Space via Balanced Line Hierarchies.
  Computers & Graphics, 61:29-39, 2016.
  doi:10.1016/j.cag.2016.08.001

- Mathias Kanzler, Marc Rautenhaus, Rüdiger Westermann:
  A Voxel-Based Rendering Pipeline for Large 3D Line Sets.
  IEEE Transactions on Visualization and Computer Graphics, 25(7):2378-2391, 2019.
  doi:10.1109/TVCG.2018.2834372

- Mathias Kanzler, Rüdiger Westermann:
  Interactive Visual Exploration of Line Clusters.
  Proceedings of the Conference on Vision, Modeling, and Visualization, EG VMV '18:155-163, 2018.
  doi:10.2312/vmv.20181265

2 Related Work

Vector or flow fields can be visualized in various ways. Direct rendering of velocity fields is done either by applying traditional volume rendering of derived attributes like velocity or by drawing glyphs like arrows. While these visualizations provide intuitive and fast renderings of 2D vector fields, 3D vector fields require more advanced techniques due to occlusion and cluttering. Deriving a texture that reveals flow structure is another class of flow field visualization, which is more suitable for 3D data than direct techniques. An overview over dense texture-based flow visualization techniques is provided by Laramee et al. [LHD*04]. In their work, different forms of Line Integral Convolution (LIC) methods are discussed. LIC methods are typically based on applying a filter kernel, aligned to the vector field, on a noise texture.

Dividing the flow field into different sub-areas taking common structures or features into account is useful for larger data as a high-level overview is acquired by giving an abstract representation. Such partition-based techniques are discussed in detail by Salzbrunn et al. [SWJS08]. First analyzing flow fields and extracting features is another approach to generate abstract visualizations by highlighting certain flow structures. Post et al. [PVH*02] give an overview of feature-based visualizations. Pobitzer et al. [PPF*11] list different topological methods for flow visualization and a general description of topological features in computational fluid dynamics is given by Asimov [Asi03].

Integrating the vector field and deriving geometric objects over a long distance results in streamlines for steady vector fields and pathlines, streaklines, or timelines in the unsteady case. Geometry-based visualization is especially suitable for 3D visualization of dense flow fields. Hence, we base our work on this technique and discuss specialized placement, seeding, and rendering techniques in more detail in subsequent sections. An overview over geometry-based visualization of vector fields is presented by Post et al. [PVH*02]. These derived geometric

objects can be rendered in different ways to show indirectly the vector field. Visualizaitons comprising densely seeded streamlines, however, suffer from clutter and occlusion. Therefore, view-independent and view-dependent streamline selection techniques are presented in the following sections. Additionally, illustrative visualizations advance the expressiveness of traditional flow visualization techniques by considering human perception and imitating visual physical phenomena as well as artistic ideas. This field of illustrative flow visualization is classified and summarized by Brambilla et al. [BCP*12].

2.1. View-Independent Streamline Selection

Finding the balance between a large number of lines that represent the flow field and a few lines that are clearly visible, is difficult. The techniques for seeding and selecting lines presented in this section find a line set that reflects the flow field as accurately as possible while being small enough to avoid clutter and occlusion. The current camera position is not taken into account, i.e., the resulting line set is constant with regard to different viewing angles.

Turk and Banks [TB96] show an approach to evenly place streamlines in 2D. This is done by moving or joining streamlines and by creating new streamlines in empty areas. A visually similar result is obtained by Jobard and Lefer [JL97]. They present an iterative streamline seeding strategy, where candidate streamline seeds have a defined offset to already placed streamlines. The greedy algorithm described by Mebarki et al. [MAD05] uses a Delaunay triangulation to significantly speed up the streamline placement process while preserving image quality. Liu et al. [LMG06] further speed up the placement and enhanced the coverage of features through adaptive distance control. By measuring the dissimilarity between streamlines, Li et al. [LHS08] place streamlines in such a way that flow patterns are shown with a minimum set of streamlines.

While these techniques all operate in 2D, the following presented methods outline methods suitable for 3D line sets. Mao et al. [MHHI98] extend the streamline placement technique of Turk and Banks to uniformly distribute streamlines on 3D curvilinear grid surfaces. Evenly spaced streamlines on 3D surfaces can be generated efficiently by using an image-based approach as described by Spencer et al. [SLCZ09]. Mattausch et al. [MTHG03] describe an algorithm to generate evenly-spaced streamlines in 3D. The dual streamline seeding strategy, which is searching for seeds orthogonal to the vector field direction, presented by Rosanwo et al. [RPP*09] also allows streamline placement on 3D surfaces while covering critical points with a higher priority. Verma et al. [VKP00] avoid missing interesting regions by identifying critical points and applying specific seeding patterns dependent on the type of the critical point. However, the result is focused on critical points, neglecting other flow patterns. Ye et al. [YKP05] extend the

principle of Verma et al. to be applicable to 3D fields. As the visualization of streamlines in 3D introduces cluttering and occlusion, the streamlines are filtered based on their geometric features after applying the seeding templates on critical points.

The approach described by Schlemmer et al. [SHH*07] includes a preprocess to measure flow features and uses the resulting scalar map to steer the density of seeded streamlines. Accentuating regions of interest is done by Chen et al. [CCK07a] by placing streamlines with an alternative distance metric. For this, statistical measures of streamline shape and directional similarity are used to avoid seeding of related streamlines. McLoughlin et al. [MJL*12] suggest streamline seeding using a similarity measure that is based on curve-based attributes. As not all geometric properties but only a signature needs to be stored with this approach, streamlines can be compared fast, which enables interactive adjustments. Xu et al. [XLS10] use the distributions of vectors in the domain to calculate entropy fields. This entropy field is afterwards used to seed streamlines in 2D or 3D, which maximizes the communicated information. By grouping streamlines based on geometric distance, Yu et al. [YWSC12] derive hierarchical streamline bundles. These bundles are used to select streamlines which are framing regions of similar flow.

Clustering The selection of streamlines representing the flow field is related to clustering problems. Lines depicting a similar flow structure are grouped, making it possible to reduce the number of group members by calculating representative lines which capture the structure of their cluster. Oeltze et al. [OLK*14] evaluate common clustering techniques based on streamline geometry in the context of blood flow analysis. Furthermore, they evaluate representatives of different clustering techniques and conclude that a small number of representatives is not enough to capture all parts of the flow This is why the cluster itself should be considered as well. Moberts et al. [MVvW05] studied different clustering methods for diffusion tensor imaging and present a measure based on manual classification of fiber tracts. In their application, the hierarchical clustering technique *single-linkage* in combination with the mean distance between fibers performed best. Zhang et al. [ZHT06] compare different similarity measures concerning quality and computational cost for outdoor surveillance trajectories. A general and extensive overview of different data clustering techniques is given by Jain [Jai10]. These clustering approaches focus on the extraction of main flow characterisitcs. However, to be able to control line density, the complete cluster hierarchy is important. Therefore, we propose a graph-based clustering strategy that generates a completely balanced line hierarchy.

2.2. View-Dependent Streamline Selection

Carefully selected streamlines are able to communicate the flow field in 2D quite well. Visualizations of 3D streamlines, however, have to take into account that not every streamline is completely visible from every viewpoint. View-independent methods reduce occlusions by placing as few streamlines as possible. Nevertheless, important regions might not be represented well and context information in the neighborhood might be missed, which impedes the general understanding of the flow. There is no guarantee to have an unobscured view to important flow features. For some viewpoints, a streamline can be added without impairing the visualization of the present streamlines while this streamline could occlude important streamlines for other viewpoints. To take this into account, a category of techniques arose that is considering the current viewpoint.

Instead of seeding streamlines evenly in object space, Li and Shen [LS07] propose an algorithm that seeds streamlines in screen space. These streamlines are integrated as long as they keep a specified distance in screen space to previously placed streamlines. In their work, occlusions of streamlines are considered but interesting structures are not prioritized and hence, interesting structures can only be visualized by manually adjusting a parameter that controls a clipping plane, which removes line segments. Marchesin et al. [MCHM10] propose a streamline selection technique, which dynamically adds and removes lines reducing the overall overdraw for the current view. In this approach, the entropy of lines is measured and an occupancy buffer identifies empty or overfull regions. A similar approach is proposed by Ma et al. [MWS13] where seeded lines are evaluated for a set of sample viewpoints in a preprocess concerning their relationship between the 3D line representation and their corresponding 2D projection. Another entropy based technique is proposed by Lee et al. [LMSC11]. The maximum value of an entropy volume derived from the underlying vector field is projected to screen-space. This entropy map is used on the one hand to find a viewpoint that maximizes the visibility of interesting regions and on the other hand, is used in combination with a streamline score to reduce coverage of those regions. Annen et al. [ATR*08] reduce clutter and overdraw only indirectly by filtering streamlines based on their orientation to the view plane. Their algorithm generates a set of streamlines which imitate a contour in 3D vector fields. Günther et al. [GBWT11] suggest the use of semi-transparent streamlines to fade them in and out during exploration of the data set and to hint at lines while not completely blocking the sight towards streamlines further in the back. The level of transparency is determined by the streamline's screen contribution. Their work is putting the viewer in charge of an expressive streamline selection by prioritizing streamlines dominating the current view and by allowing interactive brushing, that fades in lines similar to a selected line. However, a

streamline near the viewer, resulting in a high screen contribution, may hide lines that are more expressive.

Later, Günther et al. [GRT13] propose an algorithm that determines alpha values for line segments to balance shown information and coverage. A precalculated set of streamlines ensures frame coherence and reproducible images. To balance context and main information, lines have to be graded. While other approaches prioritize lines regarding their geometric features or use specialized entropy measures, the approach by Günther et al. is not limited to these attributes, but allows to input arbitrary importance values per line segment. With graded line segments, the algorithm finds a globally optimal alpha value regarding information transport for each line segment for the given view. This means, important line segments are faded out if they block a higher number of equally important line segments or one line segment that has a higher importance value. Moreover, unimportant line segments are faded in if they are not blocking the view of other line segments. Figure 2.1 shows the application of opacity optimization by Günther et al. [GRT13] on a medical data set. Line segments marked as important are clearly visible, nevertheless understanding of the global flow is difficult due to many semi-opaque layers. Opacity optimization is realized by solving a quadratic energy function that contains information about overlapping line segments weighted by their importance. As solving this energy function per frame is computationally expensive, Günther et al. [GRT13] introduce a hierarchical opacity optimization that additionally solves transitions between lines of different time steps.

Since this technique fades out lines that are classified as unimportant, the relationship between those context lines and important lines, however, remains unclear in areas where context lines are removed near important lines. A refined version of the opacity optimization technique proposed by Günther et al. [GTG17] solves the minimization problem per pixel and therefore is able to combine the visualization of lines with arbitrary geometry types like points and surfaces. Rojo et al. [RGG19] extend that framework to process and render geometry order-independent. Using a Fourier approximation, visually similar results can be obtained while increasing performance, which allows for visualization of larger data sets. However, a high number of semi-opaque lines that overlap leads to the problem that individual lines can no longer be recognized. Therefore, we control the density of lines by a selection from our line hierarchy.

2.3. Explorative Techniques for Flow Analysis

Explorative techniques are used to investigate data sets where it is not yet clear what is actually being searched for or which features are contained. The viewer can influence the visualization and emphasize or hide areas of the data. Through manual control, structures can thus be revealed that

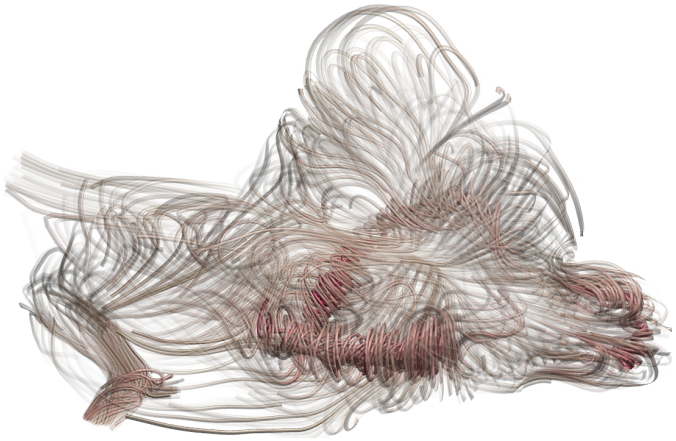

Figure 2.1.: The visualization of streamlines representing the blood flow inside an aneurysm by using opacity optimization by Günther et al. [GRT13] shows internal structures while preserving context lines. Nevertheless, depth perception is limited due to many layers of semi-opaque lines.

are otherwise hidden by selection through automatic mechanisms. Fuhrmann and Gröller [FG98] propose an interactive 2D lens to analyze flow structures. They also show how boxes whose shape can be interactively modified are a view-independent alternative to inspect the scene by showing a higher level of detail inside the volume of the box. This interactive box approach is refined by Mattausch et al. [MTHG03] by better integrating focus regions in context regions.

A good balance between context and focus information is also contributing to an effective visualization in scalar volume rendering. Viola and Gröller [VKG04] discuss different compositions of previously segmented and classified regions in volumes. To show as much as possible of context structures while the locations of regions of interest are not known beforehand, Krüger et al. [KSW06] propose a method that clears the view only at user defined locations. In order to reveal hidden line structures, Tong et al. [TCSW15] propose a method for line data sets that removes layers blocking the view at user defined regions. Using touch-based devices, the user is able to define a lens where the front layer of lines is deformed to uncover structures located deeper. Yu et al. [YEII12] provide a technique specialized for direct-touch interfaces. By encircling a target region, a 3D selection of particles is realized by considering the particle density. In flow visualization, Jackson et al. [JCK12] suggest a haptic device to select a line in 3D and to include more lines based on this initial selection. Behrendt et al. [BBB*18] propose an indirect pathline selection technique for blood flow based on the surface of the vessel. Features shown at the vessel wall hint towards insightful selections. The relation between a selected area and pathlines generated beforehand at that area determine the visibility of those lines.

A different approach for exploring blood flow in aneurysms is presented by Gasteiger et al. [GNBP11]. They explicitly divide relevant attributes into focus and context information. In a preprocess, the surface of the vessel and anatomical landmarks are extracted. Context information is presented in form of a semi-opaque surface and a few streamlines. Manually placing the 2D lens on the surface shows either vorticity, which is displayed by a color coded LIC on a view-aligned plane, pathlines to show time dependent data, or iso-surfaces of flow pressure. The shape of the lens is not adaptive and the focus region has to be defined manually, requiring knowledge about the approximate region where relevant structures are located.

2.4. Human Perception and Rendering Styles

How visualizations are translated into human understanding of the underlying data set is linked to human perception. To communicate data effectively, this translation process has to be considered when designing visualization approaches. The following section presents studies that show how observance of human perception increases the effectiveness of visualizations. Our methods are drafted with human perception in mind and utilize concepts of these studies.

Wanger [Wan92] conducted experiments to evaluate the effect of shadows of simple objects. He observes an overall improvement on task performance regarding shape, position, and size when using shadows. Object shape matching is more difficult when using soft shadows in contrast to renderings with sharp shadow edges. Wanger et al. [WFG92] evaluated perceptional cues like perspective, shadows, texture gradients, and motion regarding spatial relationships between simple objects. Shadow and perspective were identified as important cues for identifying object locations. In these experiments, the task setup was comparatively simple since the shape of the objects were ordinary (e.g. spheres and cubes) and shadows were projected onto an even ground plane.

To understand the effects of surface shading of objects further, Kleffner and Ramachandran [AKR92] conducted experiments to investigate the interaction of shading with other visual processes. They inferred from the results that the extraction of shape from shading is done early in the human visual system. In these studies, simple illumination models were used to conduct the experiments. However, the assumption of a single light source and no interreflection is not a good approximation of reality. Hence, depth perception considering a more realistic light model by adding diffuse illumination is studied by Langer and Bülthoff [LB00]. A user study on image comprehension comprising a number of different illumination models applied to direct volume rendering was conducted by Lindemann and Ropinski [LR11]. Their survey shows an enhanced depth perception for images rendered with global shadows. The survey by Jönsson et

al. [JSYR14] gives an overview of different interactive volume rendering techniques comprising global illumination effects.

Eichelbaum et al. [EHS13] make use of ambient occlusion effects to improve spatial discrimination of dense line sets. For this, they adapt a technique that imitates effects similar to ambient occlusion by using screen-space measurements [LCD06] to be applicable for thin structures like lines. Another technique to enhance spatial perception of dense line sets is proposed by Everts et al [EBRI15]. By using depth-dependent halos around lines, bundles of lines are visually merged and single lines are separated better.

Transparency For dense data sets, semi-transparently rendered geometry helps to reduce occlusion of structures located in deeper layers. However, perceptual limits of transparency-based visualizations exist, which are described by Mishchenko and Crawfis in their study [MC14]. They conclude that simple renderings of semi-transparent streamlines exceed the understanding of depth as too many similar layers are composed.

Specialized blending techniques help improve conveying structures contained in the data set [CFM*13]. By unordered accumulating fragments, abstract visualizations show features through density projection [KLG*13]. Alpha-blending, however, requires the fragments falling onto the same pixel to be in correct visibility order. Common techniques accurately accomplishing this without the need for prior sorting of geometric objects are *depth peeling* [EW] or *fragment linked lists* [TG10, YHGT10]. Depth peeling renders the scene multiple times, for each layer that has to be blended once, which diminishes performance when rendering scenes comprising a high depth complexity. Fragment linked lists gather all fragments in one pass, which are then sorted in parallel on the GPU and blended afterwards. Fragment linked lists have an unbound memory requirement, leading to lost fragments when the scene complexity is too high.

To overcome performance and memory limitations, approximate techniques emerged. Bavoil et al. [BCL*07], for example, introduce the *k-buffer* that stores only a fixed number of entries. Salvi and Vaidyanathan [SV14] propose with *Multi-Layer Alpha Blending* a technique that achieves bound memory requirements with a scalable approximation of the compositing equation by merging fragments into a fixed number of layers. A stochastic transparency approach is proposed by Enderton et al. [ESSL11] that samples the fragment's contribution. *Moment-Based Order-Independent Transparency* introduced by Münstermann et al. [MKKP18] enables progressive compositing by using power moments to approximate the original transmittance function.

2.5. Voxel-Based Rendering

In contrast to rasterization-based rendering, ray casting approaches do not neccessarily process every geometric object in the view but calculate intersections of those objects with the viewing ray. An essential component for ray casting are hence fast queries of geometric objects in world coordinates. Therefore, advanced space partitioning and traversal schemes emerged [WIK*06, WMS06, WJA*17]. Objects along the viewing ray are implicitly sorted by depth, which enables efficient blending of semi-transparent objects.

For complex geometry, high resolution voxel-based representations were chosen to improve visual quality with respect to geometric details [CNLE09, LK11, RCBW]. Cohen-Or and Kaufman [COK97] show how line data sets can be converted to voxel models for ray casting applications. Wald et al. [WKJ*15] propose a ray casting approach for particle visualization that stores particles in a tree-based structure to enable fast world-space queries.

Access to information of geometric objects in world space facilitates global illumination effects as structures influencing the light distribution can efficiently be evaluated. Wald et al. [WKJ*15] show how ambient occlusion can be included in particle renderings by using ray tracing to enhance shape perception of particle groups. By converting geometric objects into voxel-based representations, Crassin et al. [CNS*11] outline how ambient occlusion effects can efficiently be calculated, and Thiedemann et al. [THGM11] demonstrate global illumination effects in interactive framerates.

Schussman and Ma [SM04] propose a specialized technique to render line data sets as given a huge amount of lines, rasterization-based approaches easily reach a memory or performance limit due to their linear scalability. The proposed rendering technique has a constant rendering time and a constant overall storage requirement by saving spherical harmonics in voxels and employing traditional volume rendering. In a preprocess, for each voxel, the average lighting color and opacity is calculated, summarizing many line segments and light sources inside one voxel. For this, ambient, diffuse, and specular illumination terms of cylindrical elements are used. To obtain a compact representation of the lighting distribution, spherical harmonics are used to store samples located on a unit sphere. At runtime, the encoded lighting information is evaluated for the current viewing direction and converted to traditional voxels, which can then be blended using front to back ray casting. As a drawback, individual lines can no longer be identified and the resolution of the image dependends on the grid resolution. Zooming in on an individual region requires swapping the preprocessed data. Nevertheless, adjustments to the transfer function can be done interactively.

3

Fundamentals of Expressive Line Visualization

In this chapter, the fundamentals of line rendering on modern GPUs are reviewed briefly. Furthermore, as line visualizations in 3D highly depend on depth perception to communicate contained information effectively, relevant depth cues are explained in detail. For this reason, the technical background on simulation of illumination as a major depth cue is introduced thereafter. Finally, we show, how these depth cues are considered and incorporated in our approaches.

3.1. Line Rendering Techniques on GPUs

In the context of modern GPU development, small programs called shaders can be used to modify data along the graphics pipeline. Rendering involves at least a vertex and a fragment shader, which have to be specified. The vertex shader operates on vertices and transforms them usually from world space to view space and via perspective projection to normalized device coordinates. From the output vertices and a defined geometrical topology, the not programmable rasterizer creates a number of fragments . Afterwards, the fragment shader operates on the created fragments and is used to define a color for the final output. An optional geometry shader is placed between the vertex and fragment shader. In a geometry shader, the defined primitive topology can be changed and vertices can be discarded or additional vertices can be created.

A curved line approximated by a set of points of support can be submitted to the graphics pipeline via vertices. Graphics APIs are able to draw lines directly, though resulting in a thickness of one pixel. Lines of one pixel in diameter are difficult to assess spatially as there is limited space to communicate illumination and there is no variation of projected thickness possible. To display lines with a defined thickness, one possibility is to double all vertices and shift the connected vertices away from each other to create a ribbon. The ribbon can be oriented specifically to

encode information like nearby flow properties i.e. vortices or turbulences. Another possibility is orienting the ribbon towards the camera. This results in a consistent projected thickness of the ribbons most of the time. To create a view oriented ribbon out of a line, the line's tangents have to be known for each point of support. To span this ribbon, perpendicular to this tangent and perpendicular to the view direction, one vertex is shifted in positive direction and one vertex in the opposite direction away from the line's center. View oriented line ribbons are used by Everts et al. [EBRI15] including a contour that is shifted away from the viewer by a small offset. This has the effect that the line ribbons form a connected surface as the contour isolating individual lines is hidden when lines are close together.

One downside of line ribbons is the limited spacial extent in 3D. This gets apparent when the direction of line segemnts coincides with the view direction. To get a real three dimensional representation, the lines can be extruded to tubes. This can be done by generating vertices on a circle orthogonal to the line's tangent at the points of support. Using the generated vertices and the vertices located at the circle of the subsequent point of support, triangles can be spanned to get the surface of the tube. One has to track an additional normal along the line, which ensures that the tubes are not winding from one point of support to the subsequent one when connecting vertices from the circles. As the circles are aligned with the tangent given at the point of support, the tubes are squeezed slightly when a curve is rendered. Though, this ensures a closed geometry along the tube in contrast to cylinders that are generated per line segment. Another possibility to ensure a closed surface is to combine cylinders aligned with the line segments and spheres at the joints. This composition of spheres and cylinders has an outer surface with a constant distance towards the original line at the cost of additional geometrical complexity.

Technically, the location of the triangles approximating tubes or cylinders and spheres can be calculated on the CPU in a preprocess and transfered to the GPU on which a simple rendering of triangles is performed. This comes with drawbacks comprising a limited flexibility regarding the radius of the tubes as the geometry has to be reconstructed. In terms of handling a large number of lines, constructed tubes need a multiple of memory, not only reducing available GPU memory, but decreasing rendering performance due to memory bandwidth limitations. For this, the programmability of the rendering pipeline is used by only keeping points of support in memory and constructing tubes dynamically at each frame. This can be done by using a geometry shader which gets a line segment as input and changes the primitive topology to triangles forming the tube segment. Only vertices representing points of support are submitted to the GPU, while the geometry shader generates additional vertices to render the final tube. To avoid the use of geometry shaders, tubes can also be extruded in the vertex shader. The final number of vertices has to be specified in advance but passing position information with inputted vertices is dispensable.

The input of the vertex shader includes vertices consisting of just an unique sequential ID. This ID is used to access the correct point of support in a bound buffer. Multiple vertices get assigned to a line segment and are extruded to a tube segment as described earlier.

3.2. Illumination in Computer Graphics

It was recognized early on that illumination effects are decisive for a realistic image. Starting with a brief description of real world light transport, basic illumination simulation techniques are shown in this section. Light transport is a complex phenomenon that can only be approximated in visualizations. Generally, a better approximation of light phenomena comes typically with higher computational load. In computer graphics, techniques emerged that resemble partial aspects of complete light transport by using simplified assumptions in their model, enabling realistic looking effects in real time.

Light Transport Photons between a light source and a viewer can travel multiple paths, since the photons interact in many ways in mediums and on surfaces. In our visualizations, not all optical phenomena are relevant, such as refraction, fluorescence, or several interactions in air, to name a few. When light hits an opaque object, it is partly reflected and partly absorbed. In the case of semi transparent materials, light may also transmit through the object. Depending on the roughness of surfaces, starting from a perfect reflection on smooth surfaces, light is scattered into all directions when encountering rough materials. At least partly diffusely reflective materials can be found commonly in real world, wherefore this kind of reflective characteristic is essential when imitating light transport. Realistic looking images can be simulated easily when only light from a single light source reflected by a diffusely reflective material into a specified direction, e.g. into the direction of the viewer, is considered. Multiple reflections, i.e., interreflections, can only be roughly approximated due to light scattering in all directions, resulting in an increased number of possible pathways. In the following sections, illumination models are covered that can be evaluated at interactive framerates and enhance perceivability of line visualizations.

Local Illumination Local illumination models process only information given at the respective surface point, e.g., normals or color. Without considering light arriving from other reflective objects or casted shadows, the computation is independent of other objects and the computational effort is low. The well-known Phong illumination model [Pho75] approximates light intensity by adding a constant ambient term, diffuse reflection, and specular reflection. To evaluate these terms, the surface normal, the light direction, and the view direction have to be known, besides material

properties. This model is refined by Blinn [Bli77] to better approximate specular reflections. While directly lit diffuse reflections of these models look plausible, surfaces not directed to the light source are only represented by the constant ambient term, making it hard to recognize shapes at those regions.

Ambient Occlusion An improved approximation of the ambient term is given by simulating ambient occlusion effects. Ambient occlusion imitates the shadowing of ambient light through nearby structures. Especially surfaces only illuminated by ambient light can be visually improved in comparison to a constant ambient term. As nearby structures are considered, the simulation is more complex compared to a simple local model, but still faster compared to global illumination simulations, which consider distant objects as well. A common technique to calculate ambient occlusion for a surface point is to cast rays into sample directions. The number of rays hitting nearby objects is set into relation to rays without a hit, determining the occlusion factor. A simplified method uses screen space depth values and approximates ambient occlusion on the basis of them. Since samples are only taken from the depth buffer around the projected surface pixel, this approach is fast, however, the quality is reduced as the region behind the depth imprint is not considered. Eichelbaume et al. [EHS13] present a specialized screen-space ambient occlusion approach for dense line data. As it relies on the depth buffer, semi-opaque lines can not be processed. In contrast, our voxel-based line ray casting approach (Chapter 6) calculates ambient occlusion values in 3D world space, enabling considerations of semi-opaque structures.

In our rasterization-based visualizations, however, we use a simple local method to imitate ambient light, which does not affect performance. As shown in Figure 3.1, the illumination along a curve is darkened at the inside and brightened at the outer side by comparing the length between vertices along the tube segment that are generated in the geometry shader. This roughly imitates self occlusion at the inside of the curve but does not consider nearby lines.

Shadows A surface is shadowed when an object is blocking the path towards the light source. In this case, the surface is darker compared to directly lit surfaces. Since other objects are involved, spatial relationships between objects are visually enhanced as described in the previous section. Though, for each surface point a ray has to be casted towards the light source to check for intersections. This is the reason, why shadows have to be simulated globally. With an increasing number of geometric objects in the scene, these intersection checks get computationally expensive. Considering this, our voxel-based line rendering algorithm has two advantages. First, the presence of geometric structures can be queried in world space since they are stored in a regular grid. Second, derived per-voxel density values obviate the need for accurate line intersection checks.

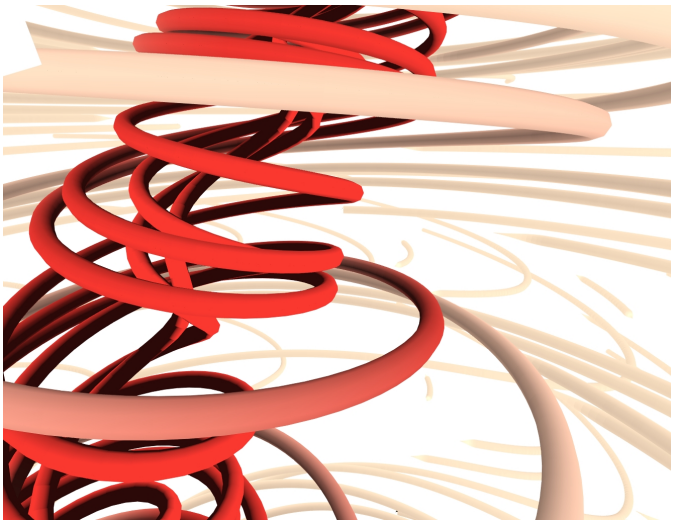

Figure 3.1.: By locally reducing the brightness at the inside of the curve, the path can be followed more easily. This is based on the assumption that less ambient light reaches the inside of the curve.

Using this method, an area light source is imitated, casting smooth shadows that bring the general line distribution into focus.

3.3. Perceptional Aspects in Scientific Visualization

Visualizations are interpreted by people. An effective visualization is able to communicate a high amount of information, lets people draw correct conclusions, and enables fast understanding of the data. To achieve this, human factors have to be taken into account. Including considerations on human perception in visualization techniques contributes to whether the underlying data can be communicated as intended.

Additionally, the chosen visualization technique influences the communicated aspects of the data. As an example, a 3D computed tomography (CT) scan can be visualized by displaying all slices next to each other. This allows a thorough presentation of the data. Neighborly relations between pathologic structures and anatomic landmarks may be apparent within one slice but not across slices. A 3D rendering may show coherences across slices but is not always able to resolve occlusions and thus hides structures. Therefore, it is not only important to think about the right visualization approach for the right data, it is also important to keep in mind, which aspects of the data should be exposed when designing visualizations. Nevertheless, independent of the intended target group and the chosen visualization approach, the amount and quality of information communicated can be increased when human perceptional aspects are taken into

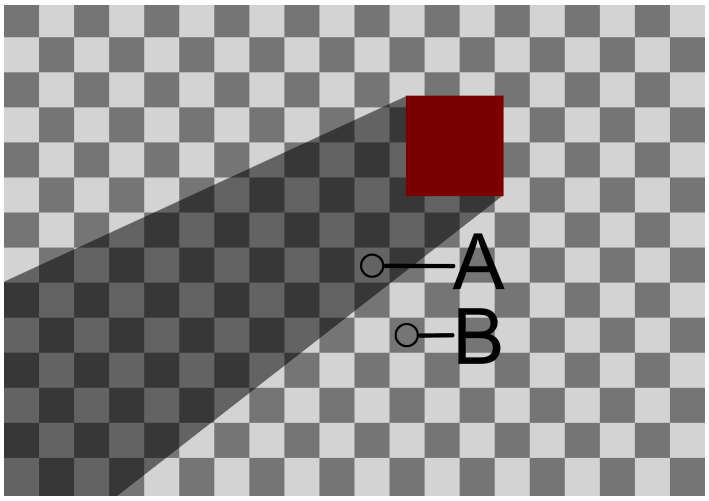

Figure 3.2.: Brightness of shadowed regions is automatically adapted by the human vision system. Therefore, region A is perceived brighter than region B, whereas the actual color of region A is darker than the color of region B.

account. As every visualization inevitable communicates with differences in hue, chroma, and luminance, the effects of choosing a specific color map, the influence of shading, and blending multiple layers have to be considered. Attributes like attenuation in CT scans or velocity in wind fields are often mapped to a color scheme. By doing this, one has to be careful to choose a color scheme that transports the information as meant. The RGB color model consisting of individual contributions of the additively mixed color channels red, green, and blue is the standard for the output in graphics frameworks DirectX and OpenGL. Blending linearly inside this color model from one color to another results in a color gradient with a luminosity that is not linearly perceived [SMDZ14]. Furthermore, Royer et al. [RH12] point out that perceived brightness is dependent on the age of the viewer.

For this reason, it makes sense to use transfer functions that map attributes into a linearly perceived color scheme from which the color is converted to RGB color values for further processing, afterwards. The human vision system is trained to interpret illumination effects like shadows and light reflections to identify shapes, absolute positions in the domain, and relative positions between objects. The reflected color of surfaces is determined by the surface character and the incoming light.

Attributes are commonly mapped to a color, which in turn is mapped to surface characteristics. By including simulated illumination in scientific visualizations, reflective behavior of the surface is chosen so that the chosen color is represented. However, the mapped color can have various

manifestations under different lighting conditions, e.g., the color is darker in shadowed regions. Then again, the human vision system is expecting illumination effects and automatically adapts perception accordingly. Figure 3.2 shows that the human brain compensates brightness of regions affected by illumination. A region in shadow can therefore be perceived brighter than a region directly illuminated, even so if this is not the case. This shows that it makes sense for our proposed visualizations to keep human perception in mind and to make use of unconscious processing of illumination effects.

Interferences between illumination and color recognition can be compensated by the human vision system up to a certain degree. Therefore, illumination effects do not necessarily have to be excluded from visualizations that contain attributes mapped to colors. This means, the positive impact of illumination on depth perception can be utilized in our visualizations.

3.4. Depth Perception of Curved Line Renderings

In this work, visualization techniques specialized for 3D line data sets are presented. These techniques are operating in a 3D domain which is projected onto a 2D image plane for output. With this 2D projection, some cues used in human depth perception like binocular parallax (dependent on the depth, differently displaced images are seen by the eyes) or accommodation (gaining depth information by changing the focal length) are not available. Rendering of curved lines has to convey as much information stored in the lines as possible. Understanding the pathways of the lines is thereby a critical aspect. With longer lines or an increasing number of lines, the visual appearance suffers from clutter and occlusion due to projection to the image plane. To get an effective visualization of 3D line data, the communication of shapes, positions, and pathways in the domain has to be supported by depth cues, nevertheless. However, some depth cues, e.g., occlusion, compete with the presentation of lines in an unobscured way. On the one hand, illumination effects enhance depth perception, on the other hand they can be confusing when applied to filigree structures like lines. Therefore, special care has to be taken when utilizing depth cues when developing methods for line visualization. An extensive and general overview of depth perception is given by Howard [How12]. In the following, relevant depth cues are listed and their application in our methods is shown.

Perspective Projection Through perspective projection, objects of the same size have a different size after projection onto the 2D image plane depending on their distance to this plane. All of our proposed 3D visualization systems are using perspective projection, serving as a basic depth cue. To achieve the effect of projected lines getting thinner with increasing distance, lines have to

be rendered with a minimum thickness of a few pixels that allows changes in thickness along the depth axis to be perceivable.

Motion Through motion of the observer or a virtual camera, the relative depth of objects can be identified as objects far away move slower along the image plane than near objects. Additionally, the effect of perspective projection is enhanced as objects change their projected shape. Movement was identified as a powerful depth cue when navigating through line data sets as their relative position to each other was noticed more easily, wherefore interactivity is crucial. This has implications on the performance of the 3D visualization systems, as for smooth movement, the images have to be rendered in a high frequency (more than 30 frames per second are desirable). In the proposed applications, the virtual camera can be navigated freely and interactively by continuously evaluating the scene and processing complex operations in parallel.

Occlusion When objects cover other objects, the conclusion can be drawn that those covered objects have a higher distance. The perception of coverage can be enhanced through motion as objects with different distances to the camera can be separated more clearly. However, occlusion directly competes with the general goal of showing as many lines and as much of the lines as possible. Furthermore, as lines are usually rendered as a thin structure, the overlapping region between two crossing lines is small, which produces a hardly visible depth cue. In our application many lines are rendered that occlude each other. Nevertheless, our priority is to balance the amount of occlusion to maximize conveyed information and to provide enough depth cues. The view towards important structures is not affected much by showing a small number of particularly representative lines, which are selected by our line hierarchy. At the same time, those representative lines hint towards other lines that are passing in the front. Another possibility to indicate occlusion while showing structures behind is to employ semi-opaque rendering. While multiple layers of semi-opaque lines clutter the view, we show how global illumination effects help to understand the scene.

Illumination Illumination as a deviation of the initial color of the object needs a surface where these illumination effects can be displayed. Lines are thin structures resulting in a small projected area that can be affected by lighting. For this, a thickness of lines that is able to transport illumination effects is used in our visualizations, for what reason other mechanics, i.e., line density control, are used to counteract occlusion.

Local illumination effects are commonly used in realtime graphics applications. Therefore, at least a local illumination model is used for rendering the lines which have been extruded to tubes, showing this geometric property. While local illumination reveals the orientation of the surface

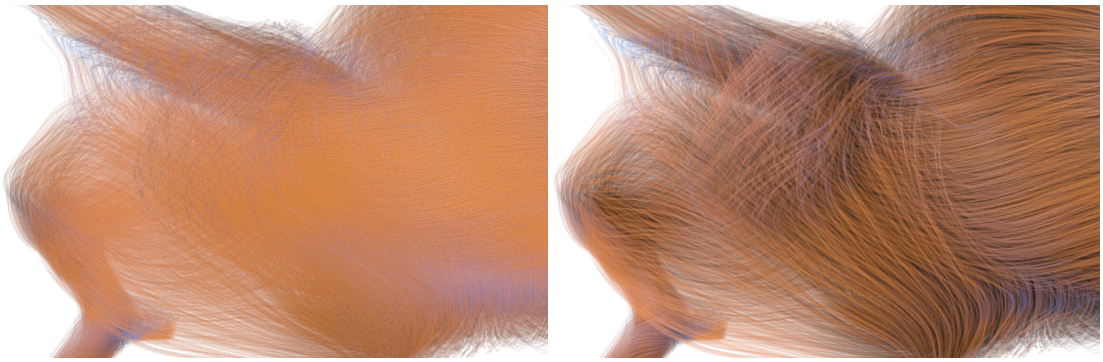

Figure 3.3.: Semi-opaque rendered streamlines implicate reduced depth perception as shown on the left. By including shadows and ambient occlusion, line bundles can be followed easily and depth perception of layered structures is enhanced as shown on the right.

towards the light source, globally calculated shadows include information about the relationship between the light source, the shadowing object, and the object the shadow is casted onto. This allows conclusions to be drawn about their relative position to each other.

Global illumination effects, however, are computationally expensive, especially when complex geometrical structures like a large number of lines have to be processed. Figure 3.3 shows the necessity to apply global illumination effects when rendering dense semi-opaque lines, though. Which is why we have combined semi-opaque lines with ambient occlusion and shadow effects in our voxel-based renderer. Furthermore, filigree structures like lines produce complex shadows that clutter the view when casted onto other lines. Therefore, hard shadows are replaced by soft shadows that diminish the influence of individual lines and emphasize line bundles.

Defocus Blur In the human eye, a lens system is bending light rays and projecting them onto the retina. This lens system results in a limited depth of field. Depending on the distance to the eye, objects are either sharply projected if they are in the focus plane, or they are blurred. Following this concept, in our application, only lines near the focus plane would be drawn sharply. This makes it less suitable for our application for the following reasons. Lines are filigree structures that are hard to distinguish when blurred. Moreover, tracking of individual lines is impeded as lines ranging in a wide area are located in defocused regions as well as focused regions. Nevertheless, we make use of combining surfaces comprising smooth gradients with detailed line renderings in our line brushing method. Lines are in the focus, surrounded by hulls derived from cluster consistency fields. With this, these two information channels can be distinguished easily.

4

Demonstration Cases

In this chapter, we demonstrate how our approaches can be used for different application cases. We show how a specialized metric used as importance criterion gives insight into cores of convection rolls and how manual tagging of lines is integrated. In addition, we describe how direct volume rendering of a scalar field can be integrated into our voxel-based ray casting framework. At last, internal functionality of proposed brush-based volumetric line selection is visualized and modified to operate in inverse mode, i.e., starting with a full set of lines, the view towards structures is cleared by application of the brush functionality. This allows further possibilities in the inspection of line sets with minimal modifications to the proposed brushing procedure.

4.1. Line Density Results of Different Line Attributes

When applying techniques that automatically remove line segments, there is a chance of missing flow features. Then again, flow features might be missed when too many lines are cluttering the view. For these cases, we add the possibility to manually change initially set importance values by selecting lines interactively on the screen. The selection of a specific line is done by including a rendering pass that writes unique IDs of the lines to a screen buffer. At the selected position, the chosen line can then be looked up at the respective buffer position. By setting importance values for all line segments to the maximum, the complete line gets visible and other lines preventing a clear view are faded out according to their hierarchy. Figure 4.1 shows a line highlighted after selecting a line segment. The path of that line now can be clearly followed. Even so, our method automatically enhances visibility according to given importance values, a

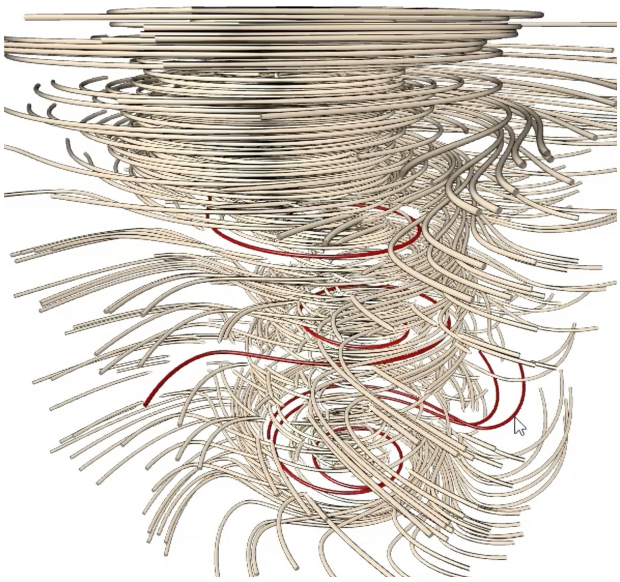

Figure 4.1.: After picking a distinct line, the importance attributes of corresponding line segments are set to the maximum. The density of nearby lines is modified according to projected metrics. This mechanic enables detailed analysis of manually selected lines.

manual revision is doable. Our algorithm immediately adapts the current line density to changed visibility constraints.

Our screen space line density control approach maximizes visibility according to importance values given per line segment. Best visual results are therefore achieved when problem-specific importance values are given. An example of specifically tailored importance values is shown by Frasson et al. [FEW*19]. They present an algorithm to extract centerlines of roll-like structures in Rayleigh-Bénard-type convective heat transport. Curvature-based importance criterions are not applicable for enhancing the interior of the circulation rolls, which is of particular interest, as streamlines near the center are elongated along the roll. Therefore, an importance criterion considering the closeness to extracted theoretical centerlines is mapped to the streamlines. Figure 4.2 shows these convection rolls with enhanced streamlines that are close to the centerlines. By setting a high importance value along the whole line, different phases can be identified since the density of surrounding areas is automatically adapted.

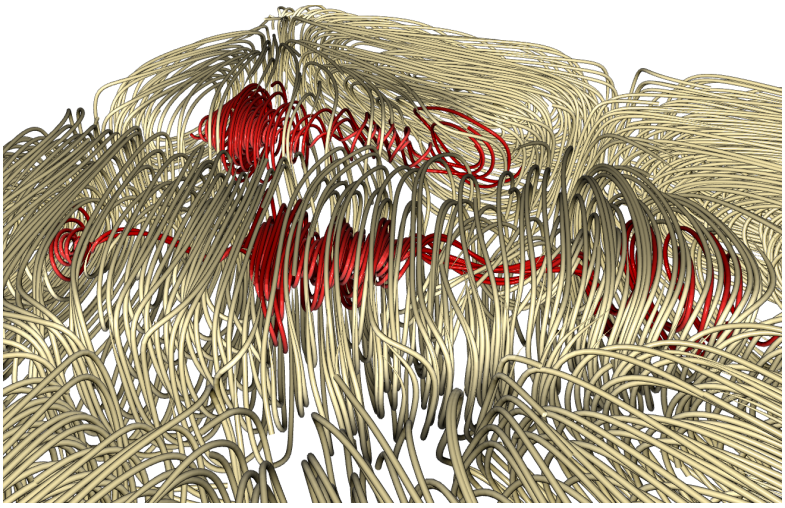

Figure 4.2.: Importance values along the lines are set according to an application-specific metric. Circulation rolls in convective heat transport were segmented in a preprocess, allowing the identification of lines near their center. Mapping high importance values to identified lines enables visualizations that make the study of inner transport behaviors possible.

4.2. Integration of Direct Volume Rendering Into Voxel-Based Line Rendering

Direct volume rendering is a common technique in scientific visualization to show 3D scalar fields in an intuitive way. By analyzing line data sets, additional information given as a 3D scalar field can be beneficial for understanding the properties of the domain when the relationship between different characteristics becomes visible. Rasterization based rendering of semi-opaque geometric objects struggles when including volumetric data sets like 3D scalar fields. An approach using linked lists for semi-opaque objects would need additional traversal steps for each gap between fragments falling onto the same pixel, diminishing performance. A regular grid as underlying data structure in combination with a ray based image generation enables the possibility to adapt and integrate other ray based techniques. Our voxel-based framework for rendering line data sets uses similar data structures and traversal schemes as used by direct volume rendering. Therefore, additional time needed for processing 3D scalar values can be neglected.

A straightforward solution is as follows. While striding the ray, at each intersection with a voxel, a lookup of the 3D scalar field is performed at the voxel's center. After mapping the scalar value to a color and an alpha value using a transfer function, a blend operation is queued in addition to the blend operations required by the voxel's line segments.

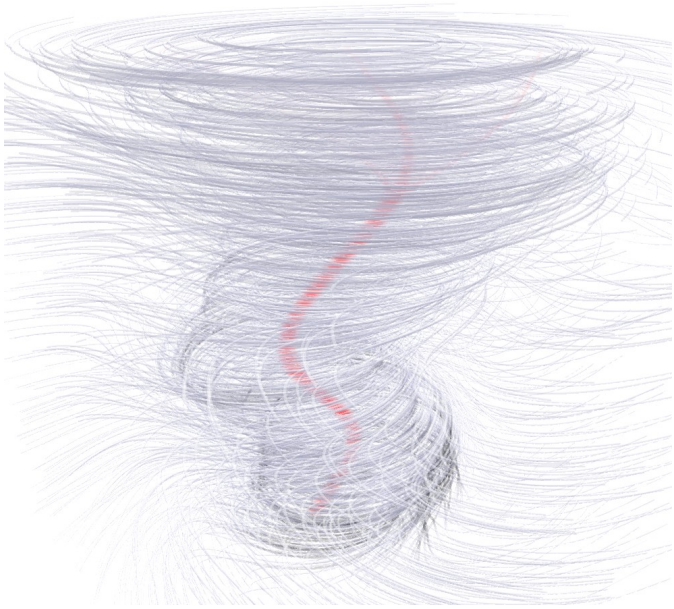

Figure 4.3.: Direct volume rendering of a 3D scalar field is combined with a voxel-based rendering of semi-opaque lines. The scalar field containing vorticity values was mapped to opacity and red color. By this technique, lines and corresponding scalar fields can be fused into one visualization including semi-opaque elements.

Figure 4.3 shows semi-opaque lines integrated in the context of a 3D scalar field. Streamlines resembling a tornado are rendered with a constant opacity value. Subtle addition of ambient occlusion and shadow effects help to preserve spatial perception even though many semi-opaque layers are blended. A scalar field containing vorticity values of the underlying vector field is included in this rendering by mapping the values to opacity and red color. By including vorticity information to the streamline rendering, the core of the tornado is clearly visible. This rendering combines a scalar field and streamlines, allowing to visually analyze correlations.

Furthermore, an embedding of opaquely rasterized geometric objects is possible. For this, the objects have to be rendered in a preceding pass with enabled depth testing. Afterwards, the volume is transmigrated until the opaque object is reached, i.e., the termination of the ray is given by evaluating the depth buffer. Altogether, this allows for a combination of rasterized opaque objects of arbitrary shape, semi-opaque lines, and common rendering of 3D scalar field.

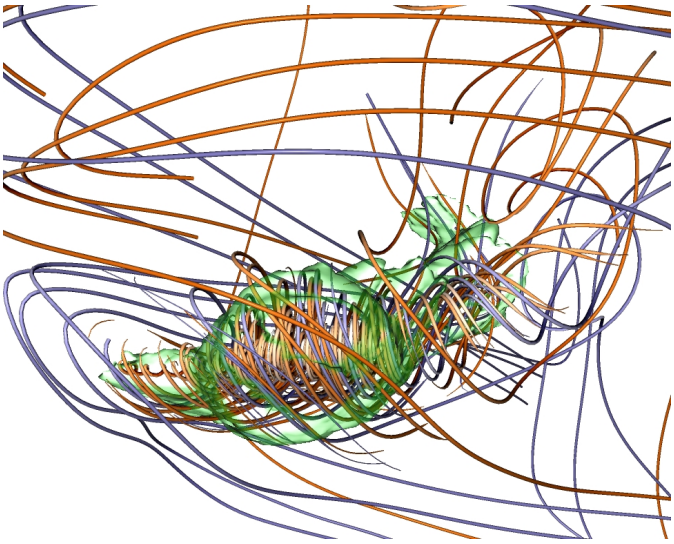

Figure 4.4.: Visibility values that determine the density of lines are usually not presented to the viewer. In this image, visibility values are rendered by direct volume ray casting for demonstration cases. Visibility values are mostly located near the projected brush and gathering around the vortex. Starting from there, visibility values are extended along the vortex core to the left and along the dissipation of the vortex to the right.

4.3. Interactive Brushing of Line Clusters

Our proposed technique for interactively exploring line clusters does not rely on an accurate targeting of special flow features as the initial selection is refined in a smart way. This refined selection shows potential interesting features that one can further analyze by panning the brush. This mechanism is realized by converting the 2D screen space brush into a voxel-based volumetric selection — the visibility volume — containing scalar values determining the local line density. Line density control is again realized by our graph-based perfect matching algorithm that assigns a hierarchy value to each line. Therefore, gradations targeted by our visibility volume can be converted to line density, showing only most representative lines in areas consisting of low visibility values. When expanding the initial visibility selection towards interesting structures, the front boundary comprising low visibility values queries a small number of lines that resemble the flow best to not clutter the image. This allows the viewer to select further analysis paths. Figure 4.4 was rendered including the outer iso-contour of the visibility volume in green for illustrative purposes. The primary selection is located in the middle of the vortex. The brush is extended along the central core at the left and towards undefined structures at the right. Two line clusters (colored orange and blue) seem to be involved in the vortex. The viewer can recognize a

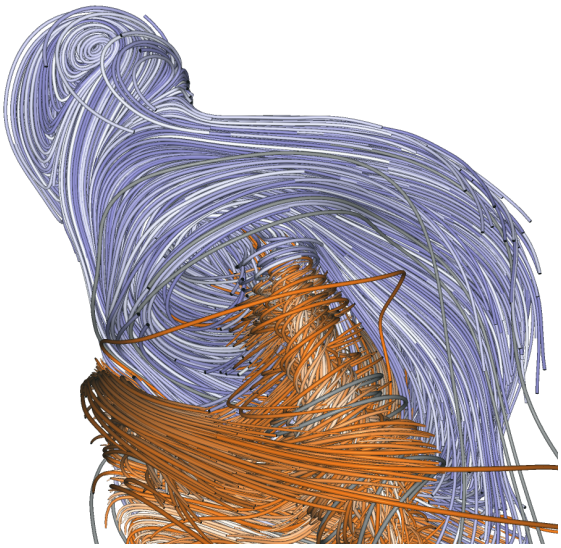

Figure 4.5.: Starting with a visualization of all lines belonging to selected clusters, layers of lines can be peeled away after minor modifications to the brush mechanic. With this, a detailed and complete visualization of all lines is given at the outer side while interfering lines can be removed.

clear vortex core and a visual hint at the right side towards a dissipation into the two line clusters comprising different flow characteristics. The viewer now can choose how analysis proceeds.

When drawing all lines belonging to a cluster, global outer structures and outliers in distance to the majority of lines become visible. Besides our proposed cluster hulls that are derived from cluster consistency fields, a view including all lines can also serve as a starting point for further exploration. With all lines drawn simultaneously, occlusion of inner structures has to be resolved. With minor modifications of our proposed framework, the brush can be used to remove interfering lines. For this, our visibility volume is used to gradually exclude unimportant lines, instead of joining representative lines. The original method gathers high visibility values around regions identified as interesting by our cluster consistency fields. An inverted interpretation of visibility values would lead to reductions of lines in areas with, e.g., high directional variance, wherefore transportation also has to be inverted, i.e., away from relevant areas. Placement of visibility values again has to be inverted, to achieve a gradient from a high number of removed lines to a high number of visible lines. Figure 4.5 shows an example, how our proposed alternative technique allows exploration of two clusters. The vortex in the middle can be uncovered while preserving most lines as context information. This alternative technique is suitable for explorations where most lines should be visible to be able to compare clusters, e.g., outliers have to be considered.

5

Paper A: Line Density Control in Screen-Space via Balanced Line Hierarchies [1]

Summary 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.

Contribution Theoretical work especially the introduction of screen-space metrics to control line density and the implementation of a prototype was done by the main author. This approach was refined based on discussions with all co-authors. Discussions with the co-authors also lead to the idea of using a graph-based clustering. Further developement of that idea and the implementation was done by the main author.

[1] This article was published in Computers & Graphics, 61, Mathias Kanzler, Florian Ferstl, Rüdiger Westermann, Line density control in screen-space via balanced line hierarchies, 29-39, Copyright Elsevier 2016

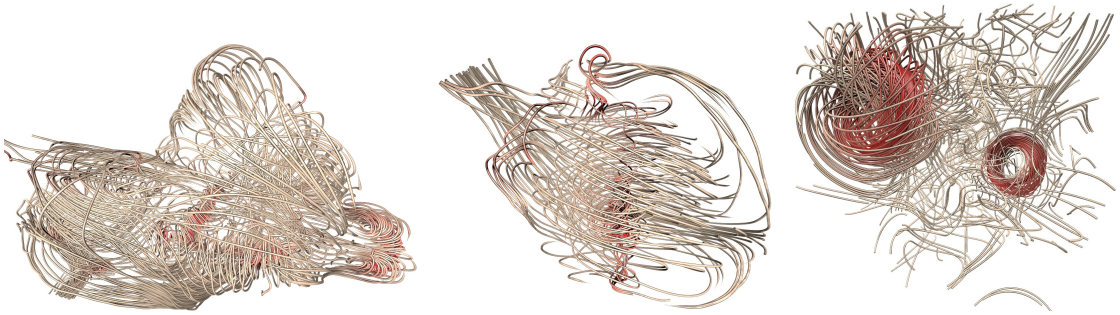

Figure 5.1.: Visualization of flow fields (Aneurysm I/II, Rings) using line density control. Line color ranges from red (high importance) to light brown (low importance).

5.1. 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 [WE05] and McLoughlin et al. [MLP*10], 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 [VKP00, YKP05, CCK07a, YWSC12].

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 [MCHM10, MWS13], or they locally adapt the line opacity to fade out those parts of foreground lines that occlude more important lines [GRT13]. 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. [VKG04].

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

We propose a remedy for this shortcoming, by avoiding the use of transparency and, instead, adapting locally the number of lines that are shown. Our approach leaves the spatial perception of the lines unaffected, yet it relates locally the screen-space density of the lines to their importance. 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 parameters. 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 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. 5.1. 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.

5.2. 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 [TB96, MHHI98, MTHG03, SHH*07], the line distribution [JL97, MAD05, LMG06, LS07, LHS08, SLCZ09, RPP*09], the information entropy in the seeded lines [XLS10], or the coverage of specific flow features [VKP00, YKP05, YWSC12] or geometric line features [CCK07a, MJL*12].

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. [TCSW15] 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 [MCHM10, MWS13, LMSC11]. 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 [MCHM10, MWS13], or the maximum projected entropy per pixel [LMSC11].

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. 5.2). 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 [VKG04] and Krüger et al. [KSW06] 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. [MCHM10, MWS13] address this problem by adding short fragments of less important lines in regions not yet occupied.

Recently, Günther et al. [GRT13] 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 can effectively avoid the removal of lines where no occlusions occur. However, in the foreground of important lines, less important lines may fade out entirely so that any contextual information is lost. Furthermore, increasing transparency affects negatively the line's spatial perception, and their shapes as well as mutual spatial relationships become more and more obscured. This effect is demonstrated in the bottom left image of Fig. 5.2.

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 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. 5.2). 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.[YWSC12], 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 [OLK*14]. The comparative study by Zhang et al. [ZHT06] 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. [MVvW05]. We use a variant of Agglomerative Hierarchical Clustering (AHC) with single linkage [Jai10], which groups the initial lines in bottom-up fashion and determines the distance 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.

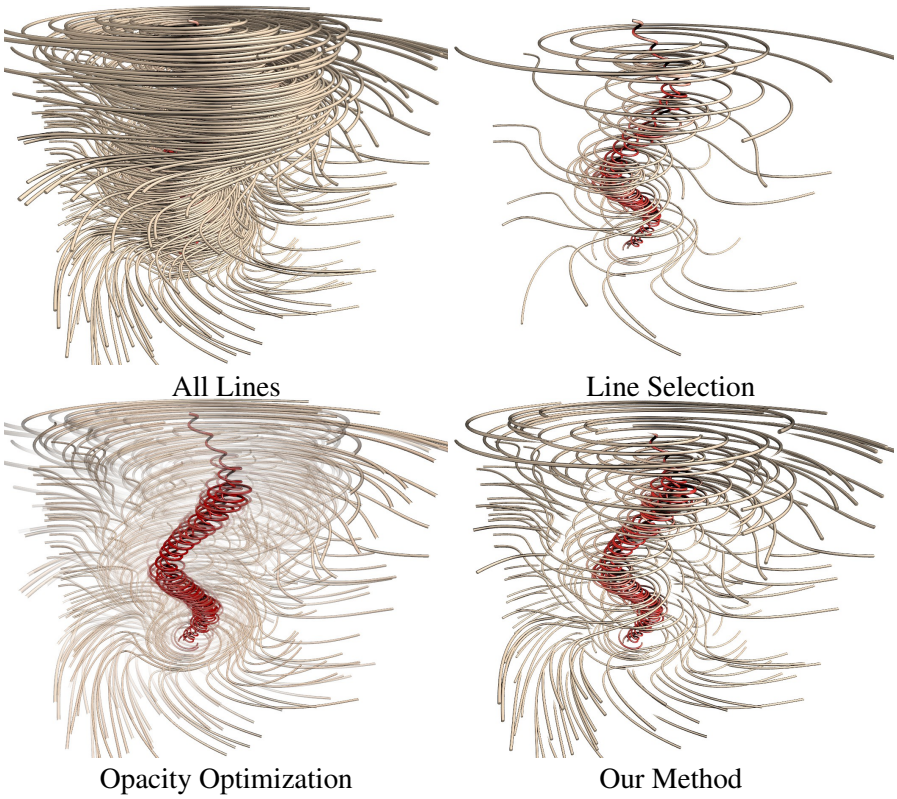

Figure 5.2.: Different approaches to reveal the focus region in the Tornado dataset.

5.3. Overview

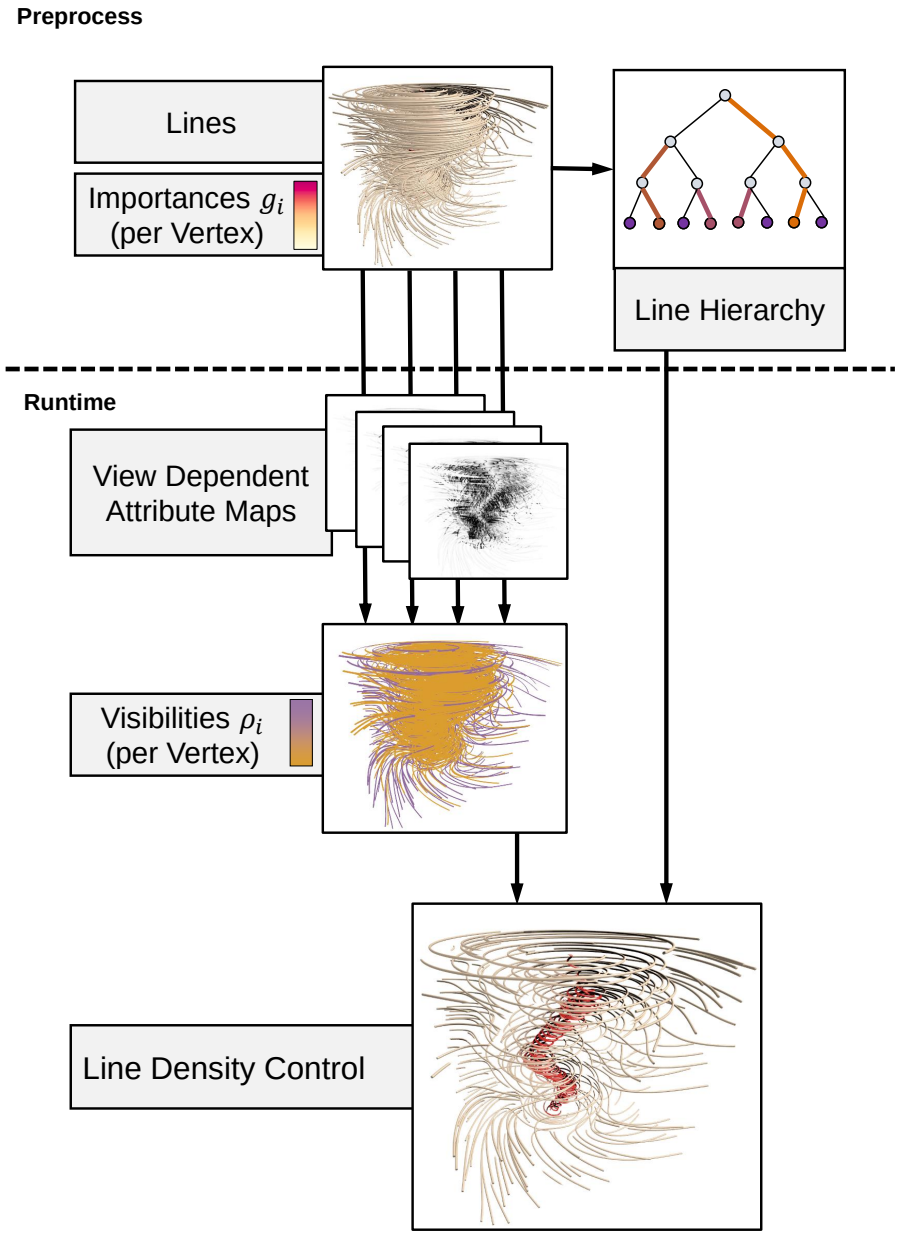

Figure 5.3.: Overview of view-dependent importance driven line rendering.

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. 5.3. 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. 5.4. 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. 5.6).

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. [MCHM10, MWS13]. Günther et al. [GRT13] 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. 5.5.

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.

5.4. 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. 5.5) 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 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.

5.4.1. 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* [CGG04]:

$$\begin{aligned} d_M(L_k, L_l) &= \text{mean}\left(d_m(L_k, L_l), d_m(L_l, L_k)\right), \\ \text{with } \; d_m(L_k, L_l) &= \underset{v_i \in L_k}{\text{mean}} \; \underset{v_j \in L_l}{\min} \|v_i - v_j\|. \end{aligned} \tag{5.1}$$

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 [CCK07b, RT12], curvature and torsion signatures [YWSC12, MJL*12], predicates for stream- and pathlines based on flow properties along these lines [BPMS12], or user-selected streamline predicates [SS06].

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

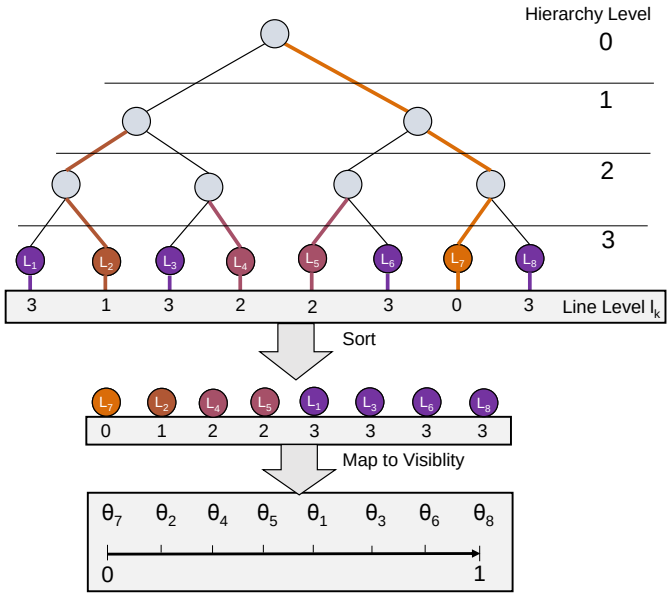

Figure 5.4.: Construction of the line set hierarchy by pairwise node merging. Lines L_i are represented by the leaf nodes, bold edges and color indicate the selection of representative lines at each level. Nodes are finally linearized and sorted according to the coarsest level where they are representative, and the sorting order is mapped to visibility.

Eq. (5.1) 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. 5.4 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 [Edm65]. In our implementation we use the LEMON library [lem16] 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. 5.6 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. [MVvW05]. 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.

5.4.2. 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. 5.4 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. 5.4 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. 5.5 shows an example which demonstrates the use of the constructed hierarchy to uniformly thin out a given line set.

5.5. 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. 5.6: 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.

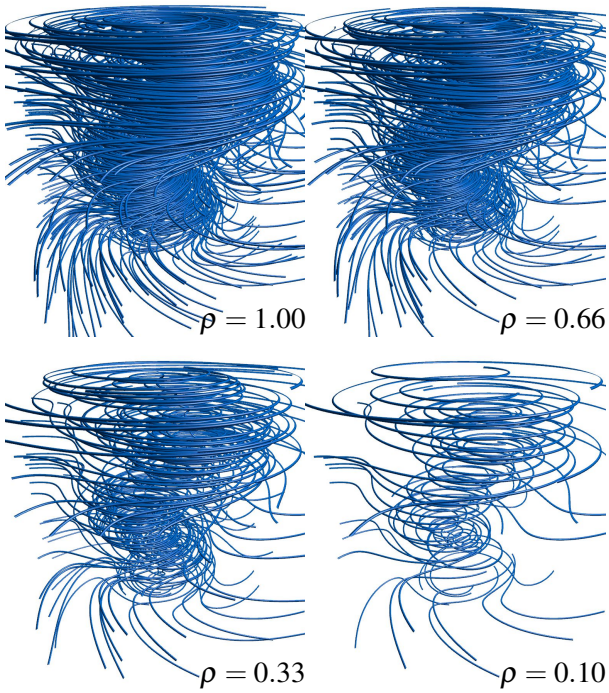

Figure 5.5.: Line density control in the Tornado dataset using a balanced line hierarchy. Lines are thinned out uniformly according to the global visibility ρ.

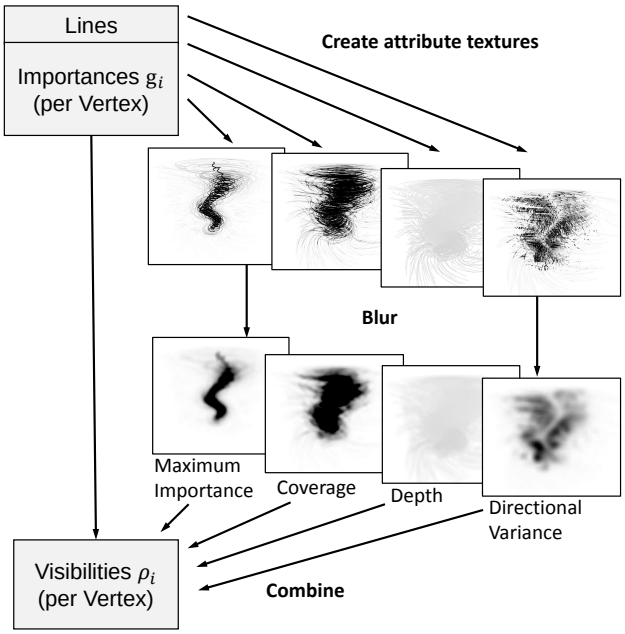

Figure 5.6.: Visibility computation using per-pixel attributes. Four textures are generated by rasterizing different line attributes, the textures are blurred, and they are then accessed by the vertices (by sampling at their projected screen coordinates) to calculate per-vertex visibilities.

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 blurring 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: To avoid that dense sets of lines with similar importance but vastly different direction can occlude each other, we introduce the directional variance as an additional control parameter. The rational behind this measure is to thin out the foreground lines in regions exhibiting different directions, so that the variation in direction to the background lines can be seen. In regions were foreground and background lines follow a similar direction, their density will solely be steered by the importance, depth and coverage criteria. 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:

$$\rho_i = \frac{1}{1+(1-g_i^{\lambda})\cdot P} \qquad \text{with} \quad P = m\cdot M(s_i) + c\cdot C(s_i) + v\cdot V(s_i) + d\cdot \text{less}(i, D(s_i)) \tag{5.2}$$

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

Table 5.1.: 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

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.

5.6. 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. [CB11]. 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 [YTvdW*02] (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. [BMC14]. 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 [AEE*16].

In Table 5.1, 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 *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 5.1 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.

5.6.1. Visualization parameters

Our approach for computing the visibility values ρ_i allows the user to control several parameters (see Eq. 5.2), so that the visualizations can be adapted to specific datasets and requirements.

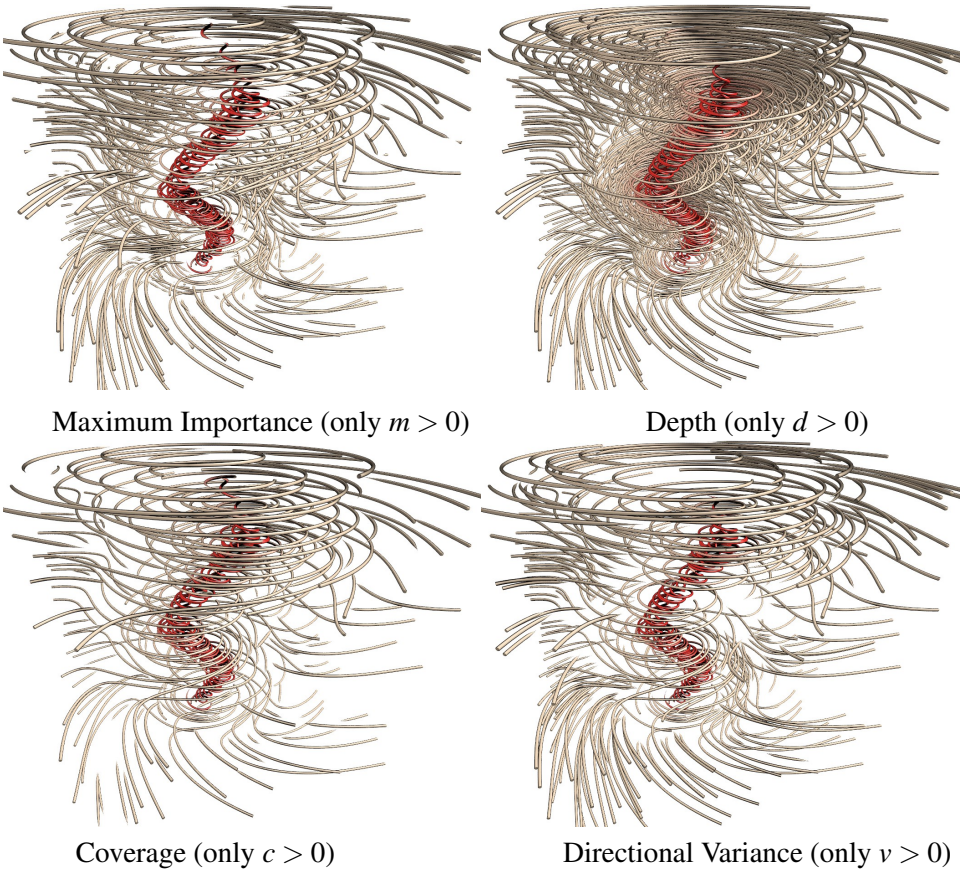

Figure 5.7.: Line density control in the Tornado datasets via per-pixel maximum importance, depth, coverage, and directional variance. Only the corresponding weight in Eq. 5.2 is set to a positive value. All other weights are set to 0.

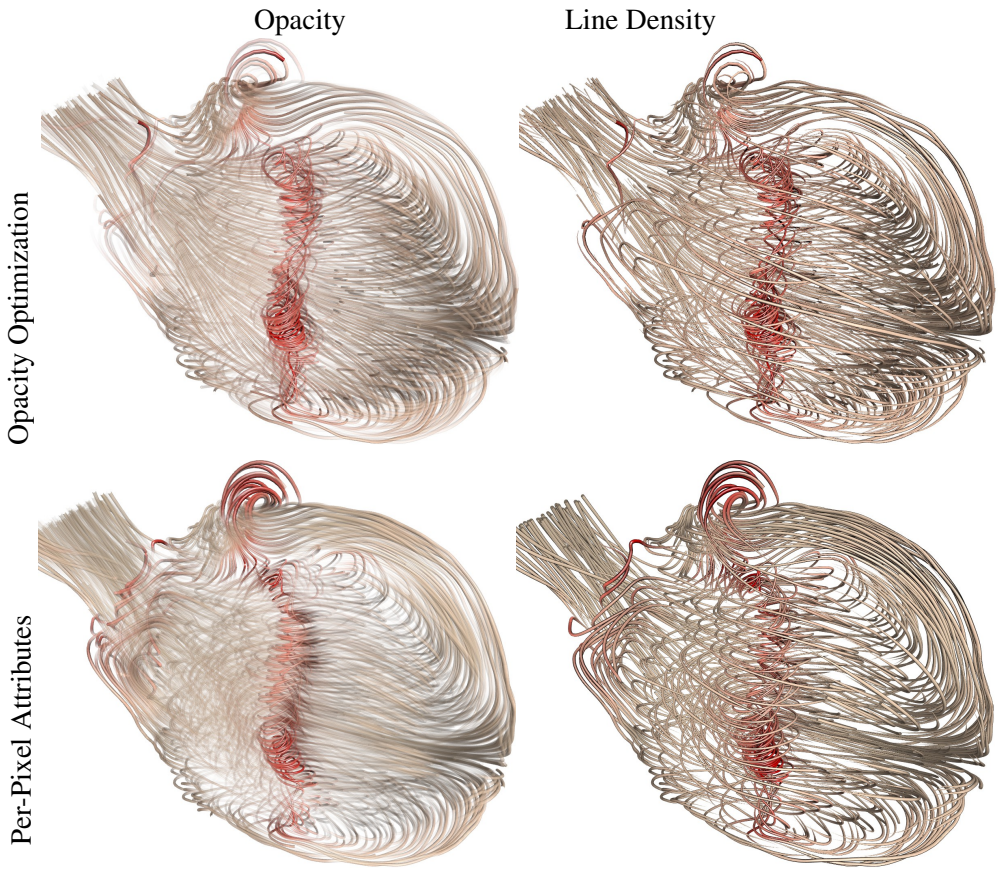

Figure 5.8.: Visualizations of the Aneurysm I dataset. Top left: Opacity optimization + line opacity as by Günther et al. [GRT13]. Top right: Opacity optimization [GRT13] + line thinning as by our approach. Bottom left: Visibility attributes as by our approach and line opacity [GRT13]. Bottom right: Visibility attributes and line thinning as by our approach.

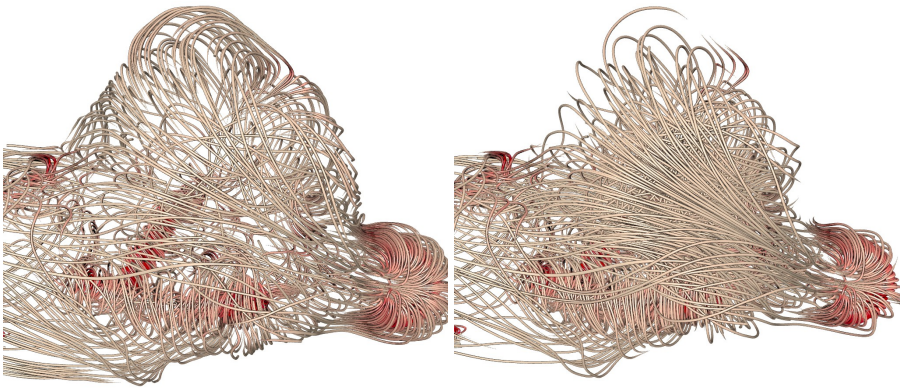

Figure 5.9.: Line density control in the Aneurysm II dataset via the directional variance (left). Right: the same visualization but the directional variance is not used.

Fig. 5.14 shows a number of examples demonstrating the effects of different attribute settings on the final visualizations. Fig. 5.7 demonstrates the effects that are achieved when only one of the 4 visibility attributes M, D, C, V (see Sec. 5.5) is used to compute per-vertex visibility values.

In the upper row of Fig. 5.7, 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. 5.2. 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.

Fig. 5.9 demonstrates the use of the directional variance to effectively visualize streamlines in regions were the flow field exhibits highly varying directions. When the directional variance is not used (first image), the background lines are more or less entirely occluded by the foreground lines, even though they have vastly different orientations. By using the directional variance (second image), the density of both the foreground and background lines is reduced, so that a good impression of the overall flow structure is achieved.

Figure 5.10.: 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.

5.6.2. 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. 5.10 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.

5.6.3. Comparison

In Fig. 5.8, we compare our approach to opacity optimization [GRT13]. 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

Images from page 6 in [GRT13]
removed in this thesis due to copyright reasons.

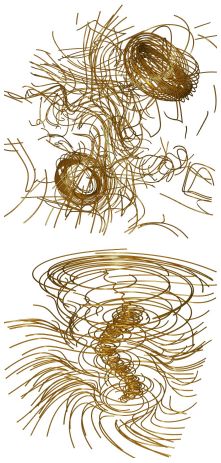

Figure 5.11.: Rings (top) and Tornado (bottom) - left to right: all lines, Marchesin et al. [MCHM10], Günther et al. (2011) [GBWT11], Günther et al. (2013) [GRT13], our approach

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. 5.11, we include our results into the comparison provided by Günther et al. [GRT13]. The approach of Marchesin et al. [MCHM10] is able to give a good overview of the flow, but cannot always reveal the important structures. Günther et al. [GRT13] 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.

5.6.4. 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. [SGS05] 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. 5.13: 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 produce the best spatial impression, since it indicates where a line is fragmented due to occlusion avoidance and simultaneously preserves the spatial location and orientation as good as possible.

5.7. Conclusion and future work

We have presented a interactive approach for the visualization of 3D flow fields using integral lines like stream- and pathlines. Our approach combines an efficient computation of per-pixel visibility attributes with a balanced line-set hierarchy to adaptively control the line density depending on the importance of lines and their occlusions. We have described the construction of such a hierarchy and its use for automatic density control. Our approach is local in that it discards line segments rather than entire lines, yet it uses line density control instead of opacity optimization to keep important contextual cues even where lines are classified as unimportant. Compared to opacity optimization, our results demonstrate enhanced spatial impression and better scalability in the number of lines. A drawback is the time that is required to construct the balanced line hierarchy, which prohibits the seeding of new lines at runtime.

In the future, one particular focus will be on the investigation of extensions to make the approach 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

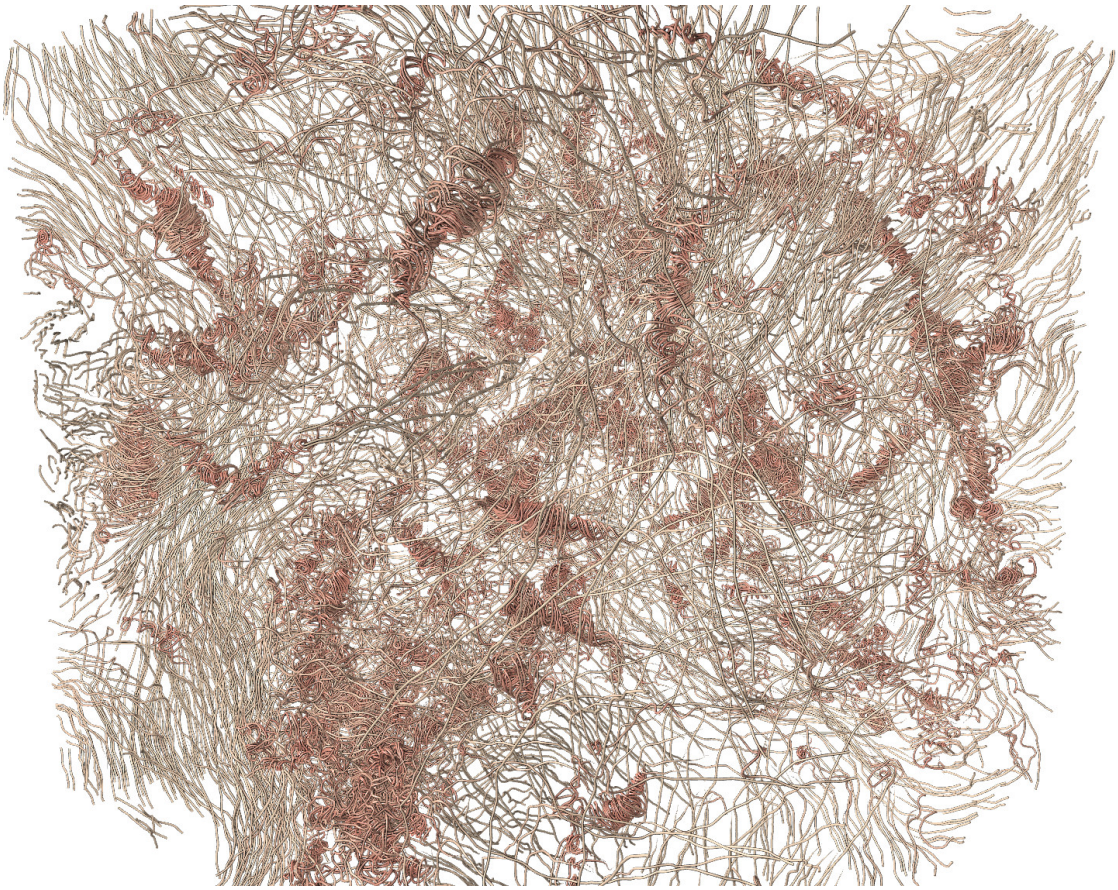

Figure 5.12.: Controlling the density of 80.000 lines in the Turbulence dataset [AEE*16] via our approach.

of the line set hierarchy that is introduced in our work. Furthermore, 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.

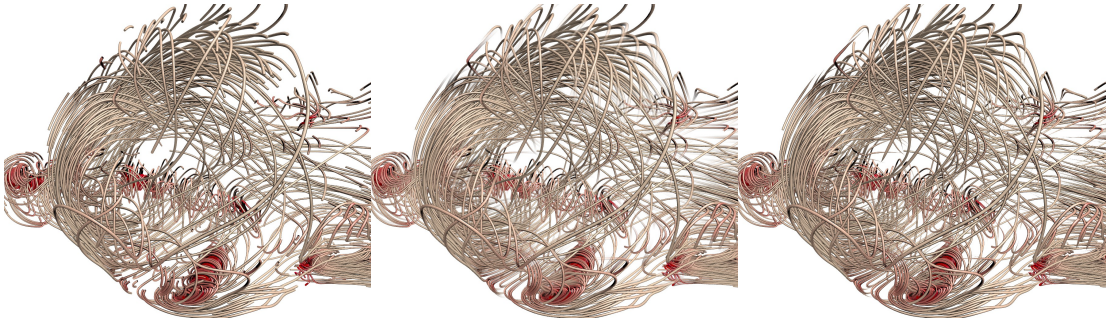

Figure 5.13.: Visualization of the Aneurysm II dataset using different rendering styles for line endings at cut-offs resulting from partial line removal. (left) Cropping lines without transition, (middle) short transparent line ends and (right) reduced diameter at line ends—our preferred choice—.

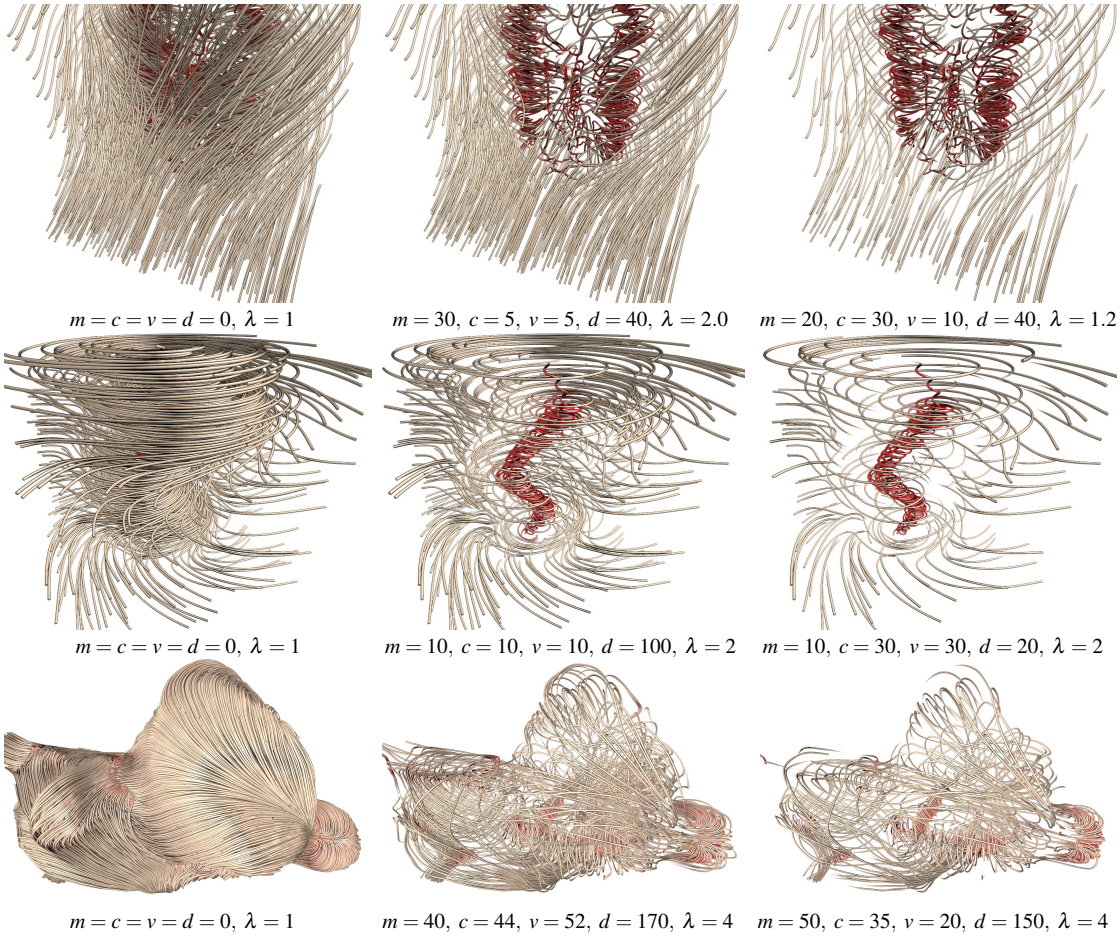

Figure 5.14.: Demonstration of different parameter combinations for the visibility computation in Eq. 5.2. From top to bottom: Heli, Tornado, Aneurysm II.

5.8. Acknowledgements

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 forR eliable Data Discovery.

6

Paper B: A Voxel-based Rendering Pipeline for Large 3D Line Sets [1]

Summary We present a voxel-based rendering pipeline for large 3D line sets that employs GPU ray-casting to achieve scalable rendering including transparency and global illumination effects. Even for opaque lines we demonstrate superior rendering performance compared to GPU rasterization of lines, and when transparency is used we can interactively render amounts of lines that are infeasible to be rendered via rasterization. We propose a direction-preserving encoding of lines into a regular voxel grid, along with the quantization of directions using face-to-face connectivity in this grid. On the regular grid structure, parallel GPU ray-casting is used to determine visible fragments in correct visibility order. To enable interactive rendering of global illumination effects like low-frequency shadows and ambient occlusions, illumination simulation is performed during ray-casting on a level-of-detail (LoD) line representation that considers the number of lines and their lengths per voxel. In this way we can render effects which are very difficult to render via GPU rasterization. A detailed performance and quality evaluation compares our approach to rasterization-based rendering of lines.

Contribution Theoretical work and development of the voxel-based rendering pipeline for 3D line sets was done by the main author. Discussions with the co-authors lead to subsequent enhancements with respect to the algorithm.

[1] ©2018 IEEE. Reprinted, with permission, from Mathias Kanzler, Marc Rautenhaus, Rüdiger Westermann, A Voxel-Based Rendering Pipeline for Large 3D Line Sets, IEEE Transactions on Visualization and Computer Graphics, July 2019

6.1. Introduction

3D line sets can be rendered efficiently via the rasterization-based rendering pipeline on GPUs, for instance, by constructing tubes around each line on-the-fly in a geometry shader. For the rendering of very large 3D line sets, however, GPU rasterization introduces the following limitations: a) When using transparency, which requires the rendered fragments to be blended in correct visibility order, all fragments need to be stored in a per-pixel linked list [TG10] on the GPU and sorted with respect to the current camera position. This approach works efficient for moderate sized line sets, yet for large sets the sorting operation significantly slows down performance, and in some cases the lists can even exceed the GPU's memory (see Fig. 6.17). b) Operations requiring spatial adjacency queries cannot be embedded efficiently into rasterization-based rendering. For instance, shadow simulation or ambient occlusion calculations to enhance the spatial perception of rendered lines. Even though hard shadows of point lights can be simulated via shadow mapping, this requires a second rendering pass using the full set of geometry, and the resulting high-frequency illumination variations are rather disturbing from a perceptual point of view (see Fig. 6.12).

An option to overcome these limitations is a rendering technique that does not build upon the order-independent projection of primitives, but is ray-guided and can efficiently traverse the line set along an arbitrary direction. For the rendering of polygon models, GPU voxel ray-casting has been established as a powerful alternative to rasterization. Voxel ray-casting employs a voxel-based object representation in combination with a regular sampling grid that can be traversed efficiently on the GPU [CNLE09, LK11]. This approach is particular useful because it can generate the sample points along a ray in an order-dependent way, and provides the ability to perform adaptive LoD selection. Furthermore, the regular sampling grid gives rise to efficient search operations, which have been used to simulate global illumination effects via ray-guided visibility computations [CNS*11, THGM11].

Mimicking voxel ray-casting for line primitives, however, is challenging, since no voxel representation of lines is known that can effectively encode spatial occupancy *and* direction. A solid voxelization of line primitives—or tubes generated from them—into a regular sampling grid [COK97] is not feasible due to the following reasons: Firstly, for the line sets we consider, too many voxels must be generated to differentiate between individual lines. Secondly, upon voxelization, changes of rendering parameters like line width and line removal require exhaustive processing. Thirdly, already at moderate zoom factors the lines will appear blocky. Finally, since voxels do not encode any directional information, direction-guided shading and filtering as well as direction-preserving LoD construction becomes infeasible.

6.1.1. Contribution

In this work we propose an alternative rendering technique for large 3D line sets on the GPU, which builds upon the concept of voxel ray-casting to overcome some of the limitations of rasterization-based line rendering. We present a new voxel model for 3D lines and demonstrate its use for GPU line rendering including transparency and ambient occlusion effects. We further show that the voxel model can be used in combination with rasterization-based rendering, by performing the simulation of volumetric effects on this model and letting the rendered fragments access the computed results.

Our specific contributions are:

- A novel voxel-based representation of lines, consisting of a macro voxel grid, and a per-voxel quantization structure using voxel face-to-face connectivity.
- A LoD line representation that considers per voxel the number of lines, their orientations, and lengths to estimate an average line density and direction at ever coarser resolutions.
- An implementation of GPU ray-casting on the voxel-based representation, including the computation of line, respectively tube intersections on the voxel level, to efficiently simulate transparency as well as local and global illumination effects.

We see our rendering approach as an alternative to existing rendering techniques for 3D line sets when there are many lines introducing massive overdraw, and when transparency or global illumination effects are used to enhance the visual perception (see Fig. 6.19). In this case, our method can significantly accelerate the rendering process, and it can even be used when the memory requirements of rasterization-based approaches exceed what is available on current GPUs. The possibility to construct a LoD line representation enables trading flexibly between quality and speed, and efficiently performing search operations required for simulating advanced illumination effects.

This paper is organized as follows: After reviewing work that is related to ours, in Sec. 6.3 we introduce the particular voxel-model underlying our approach and discuss the model generation process. Here we show images of voxelized line data using classical GPU line rendering to emphasize the effects of different voxelization parameters on the quality of the resulting representation. In Sec. 6.4 we describe the efficient realization of voxel ray-casting on the voxelized line representation, and we demonstrate the simulation of advanced rendering effects like transparency and global illumination using voxel-based ray-casting. Sec. 6.5 provides a thorough evaluation of our approach regarding quality, speed, and memory requirement, and we

demonstrate its benefits and limitations with data sets from different applications. We conclude our work with some ideas on future extensions and applications.

6.2. Related work

Our approach is related to established techniques in visualization and rendering, namely line-based rendering of flow fields and voxel-based ray-tracing.

Line-based rendering of flow fields

Today, integral curves in 3D vector fields are usually visualized via the rasterization-based rendering pipeline on GPUs, either as illuminated line primitives [ZSH96, MPSS05], or as ribbons or tubes which are constructed around each line on-the-fly in the GPU's shader units [SGS05]. For dense sets of opaque lines, illustrative rendering [EBRI09, EBRI15] has been proposed to keep separate lines perceptually visible. Other techniques abstract from single primitives and show flow features via line density projections[PYH*06, KLG*13]. Automatic view-dependent selection of transparency has been introduced by Günther et al. [GRT13], to selectively fade out lines in those regions where they occlude more important ones. A user study on perceptual limits of transparency-based line rendering for flow visualization has been conducted by Mishchenko and Crawfis [MC14]. They also suggest a number of specific usages of transparency to effectively avoid visual clutter and high levels of occlusions.

When transparent lines are rendered, generated line fragments need to be blended in correct visibility order. On the GPU, this can be realized by using either depth peeling [EW] or per-pixel fragment lists [TG10, YHGT10]. Depth peeling does not require storing and sorting fragments, yet it requires rendering the data set as many times as the maximum depth complexity of the scene, i.e., the maximum number of fragments falling into the same pixel. For the data sets we consider, where along the majority of view rays the depth complexity is in the order of many hundred or even thousand, the resulting increase in rendering times is not acceptable. Per-pixel fragment lists, on the other hand, require only one single rendering pass, yet they require storing all fragments and can quickly run out of memory on current GPUs. This also holds when depth peeling is applied to fragments bucketed by depth [LHLW09], even though the number of rendering passes can be reduced. As an alternative to the exact simulation of transparency, stochastic transparency [ESSL11] uses sub-pixel masks to stochastically sample the fragments' transparency contributions. Stochastic transparency requires only a rather small and fix number of rendering passes, yet it transforms directional structures into noise. This is especially undesirable

in our scenarios, where only lines are rendered and even in transparent regions their directional structure should be preserved.

The technique most closely related to ours is the one by Schussman and Ma [SM04]. They voxelize streamlines into a regular grid, and compute a spherical harmonics representation of the lighting distribution caused by the line segments in every voxel. Voxel-based rendering with sub-pixel illumination can then be performed, yet single line primitives cannot be determined any more, for instance, to render the initial lines or compute exact occlusions.

Voxel-based ray-tracing

Advances in hardware and software technology have shown the potential of ray-tracing as an alternative to rasterization, especially for high-resolution models with many inherent occlusions. Developments in this field include advanced space partitioning and traversal schemes [WIK*06, WMS06, WJA*17], and optimized GPU implementations [AL09, LK11, PBD*10], to name just a few. All these approaches can be classified as "conventional ray-tracing approaches", since they operate on the polygon object representation and perform classical ray-polygon intersection tests. Recently, Wald et al. [WKJ*15] proposed the use of ray-tracing for particle rendering, by using a tree-based search structure for particle locations to efficiently find those particles a ray has to be intersected with. The integration of global illumination effects like ambient occlusion into particle visualizations has been demonstrated by Wald et al. [WKJ*15] and Staib et al [SGG15].

Voxel models have been first introduced 1993 by Kaufman in the seminal paper [KCY93]. Compared to polygonal representations, they provide an interesting set of advantages like easier level of detail computation and combined storage of surface and geometry information. A detailed investigation of algorithms for line voxelization has been provided by Cohen-Or and Kaufman [COK97]. Interestingly, a voxel model has also been used in the very first published work on iso-surface visualization: *Cuberilles* [HL79]. The Cuberille method—or opaque cubes—works by computing the set of grid cells that contain a selected iso-surface and rendering those as small cubes.

In the last years, there has been a lot of research into large octrees to render high-resolution voxel models [CNLE09, LK11, RCBW]. All these approaches subdivide the model into a sparse volume, storing only small volume "bricks" along the initial model surface, and use a compute-based octree traversal to render the contained surface. The potential of voxel-based rendering approaches to efficiently simulate global illumination effects has been demonstrated, for instance, in [CNS*11, THGM11]. The survey by Joenssen et al [JSYR14] provides a general overview of approaches for simulating global illumination effects on volumetric data sets.

6.3. Voxel-based curve discretization

The input of our method consists of a set of curves, e.g. streamlines, where each curve is approximated by a connected line set. Each line set is represented by a sequence of vertices v_i and corresponding attributes a_i. The attributes can be specified optionally, for instance, to enable the use of a transfer function to interactively change the curves' colors or transparencies. The curve discretization is performed in a pre-process on the CPU, and the generated data structure is then uploaded to the GPU where rendering is performed. This pre-process can also be performed on the GPU, enabling instant update operations when curves are removed or new curves are added. This, however, requires some modifications, the discussion of which we delay until the end of the current section. We propose a two-level grid structure for discretizing the initial curves, as illustrated in Fig. 6.1.

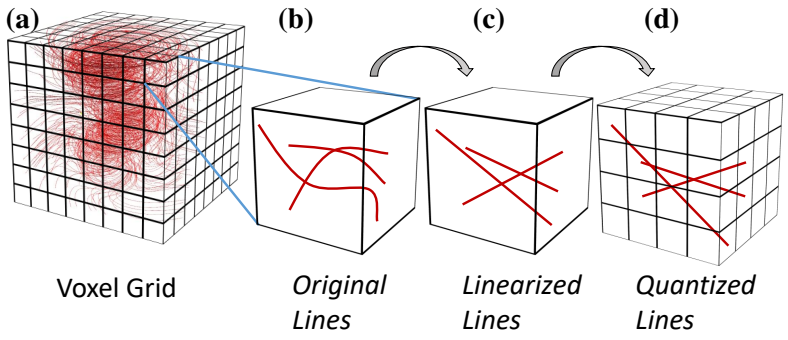

Figure 6.1.: Two-level grid structure. First level: Voxel grid, curves are clipped to the voxel faces (b), and per-voxel linear line segments are generated (c). Second level: Line vertices are quantized based on a uniform subdivision of voxel faces (d).

The first level is given by a voxel grid, in which each curve is approximated by a set of per-voxel linear line segments. A regular subdivision of each voxel face yields the second level, where the endpoints of the per-voxel line segments are quantized to the center points of sub-faces.

The use of a two-level structure allows controlling the approximation quality of the voxelization and storing the line segments per voxel in a compact form. The resolution of the voxel grid controls the size of the geometric features that can get lost when approximating these features by linear segments in every voxel. With increasing grid resolution better approximation quality is achieved, yet at the cost of increasing memory requirement. The end points of each per-voxel line segment can either be stored exactly using floating point values, or they can be quantized to a set of points on the voxel faces. This allows for a compact encoding of the line segments per voxel, and it can be used to control how many lines passing through one voxel are collapsed to one

single line. While we let the user select the quantization resolution, more advanced quantization strategies can generate a locally adaptive quantization to assure that local line features are well preserved [FG69].

6.3.1. Curve voxelization

Initially, the resolution $r_x \times r_y \times r_z$ of the 3D voxel grid into which the curves are voxelized is set. We use cube-shaped voxels of side length 1 and set the resolution so that the aspect ratio of the bounding box of the initial curves is maintained. The vertex coordinates v_i are then transformed to local object coordinates in the range from $(0,0,0)$ to (r_x, r_y, r_z).

For each curve and starting with the first vertex, every pair of vertices v_i and v_{i+1} is processed consecutively. A line through v_i and v_{i+1} is clipped against the voxel boundaries, via line-face intersection tests in the order of their occurrence from v_i to v_{i+1}. If v_i and v_{i+1} are located in the same voxel, no new intersection point is generated. This gives a sequence of voxel-face intersections, and every pair of consecutive intersections represents a line that enters into a voxel and exits that voxel. In general the first and last vertex of a line do not lie on a face, we hence omit the segments from the first vertex to the first face intersection and from the last face intersection to the last vertex.

Fig. 6.2 illustrates the voxelization process, demonstrating increasing approximation quality with increasing grid resolution, as well as limitations of the piecewise linear curve approximation when the grid resolution is too low.

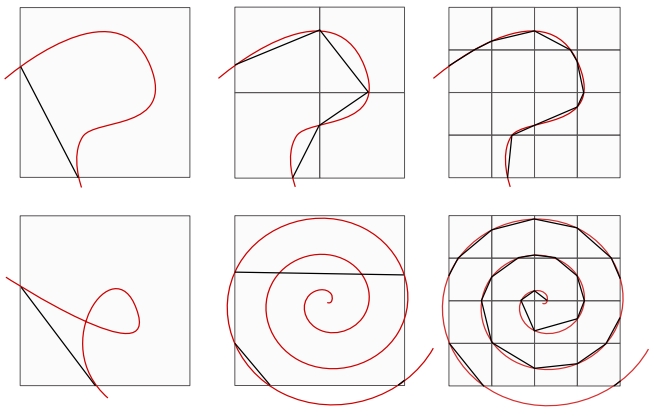

Figure 6.2.: Curve voxelization: Initial curve in red and piecewise linear approximation in black. Top: Decreasing approximation error with increasing voxel grid resolution. Bottom: Geometric details can be missed if the voxel grid resolution is too low.

For every voxel in the voxel grid, a linked list is used to store the lines intersecting that voxel. Line attributes, interpolated to the line-face intersection points, and IDs that identify to which curve a line belongs can be stored in addition.

6.3.2. Line quantization

For quantizing the coordinates of the curve-voxel intersection points, we use a small number of bins per face: As illustrated in Fig 6.1d, every square voxel face is subdivided regularly into $N \times N$ smaller squares. The coordinates of the intersection points are then quantized to the centers of these squares. Thus, every vertex can be encoded via a $2N$ bit pattern (indicating the sub-square in a voxel face) and a 3 bit face ID (indicating on which face the vertex is located). The bits are packed into a single word that is just large enough to store them. Since voxels have unit size and each voxel face is subdivided equally, the vertex location in object coordinates (quantized to the per-face square centers) can be computed from a bit pattern stored per-vertex. Fig. 6.3 illustrates the quantization process and the effects of different values of N on the curve approximations.

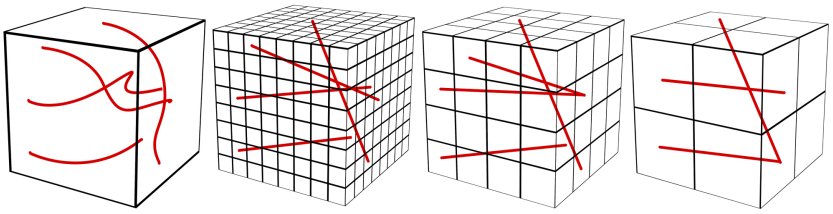

Figure 6.3.: Line quantization. From left to right: Original curves, per-voxel linear approximation using vertex quantization to $8^2, 4^2$, and 2^2 sub-faces (bins) per voxel face, respectively. If the number of bins is too low, lines can fall onto each other.

The quantization process introduces an additional approximation error that depends on the selected subdivision of voxel faces. This error does not cause any geometric details to be lost, yet it slightly jitters the vertex locations and, thus, affects a line's orientation. In particular, different lines might be mapped onto the same quantized line because their vertices are quantized to the same locations.

Fig. 6.4 shows a direct comparison between the original curves and the voxelized curves using a voxel grid resolution and per-face subdivision at which the discretization errors can only just be perceived. In both cases the lines are rendered via GPU rasterization; by starting at the first vertex and then either by traversing the original set of vertices and constructing tubes around each line on-the-fly in a geometry shader, or by performing the same construction on the discretized line

set. Especially the straight curves in the foreground show some subtle bumpiness that is caused by the quantization of line vertices.

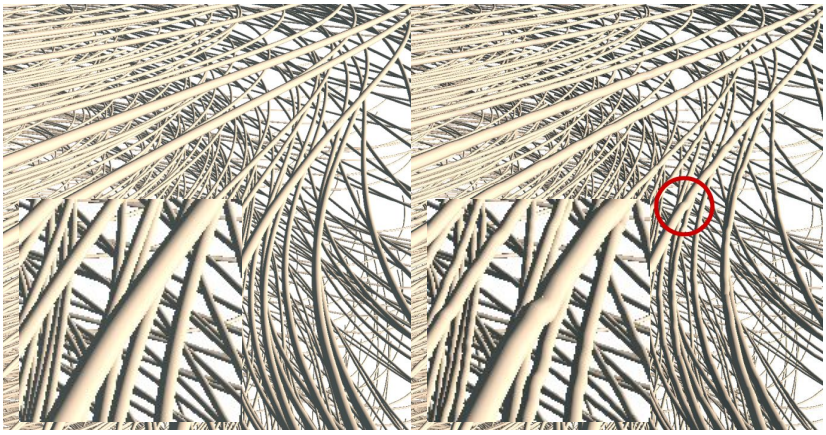

Figure 6.4.: GPU rasterization of initial lines (left) vs. rasterization of voxelized lines using 256^3 voxel grid and curve-face intersections quantized to 16^2 bins (right).

To demonstrate how, in general, the voxel grid resolution and the voxel face subdivision affect the final reconstruction quality, Fig. 6.5 shows some extreme cases which also reveal the interplay between both resolutions. A detailed analysis of the dependencies between resolution, quality and memory consumption for different resolutions is given in Sec. 6.5. While a low resolution of the voxel grid affects in particular the per-voxel approximation error, a low degree of face subdivision can introduce additional C^1-discontinuities at voxel transitions, i.e., by jittering a vertex that is shared by two lines with the same orientation. Especially high frequent variations that occur when a short line is jittered are perceptually noticeable, as indicated by the last example in Fig. 6.5.

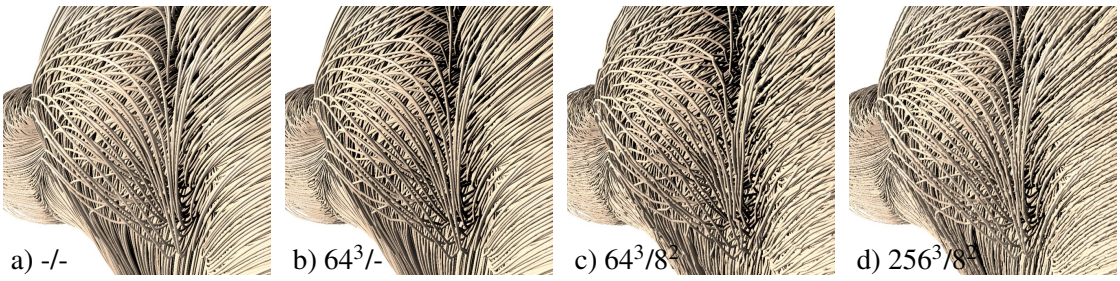

Figure 6.5.: Rasterization of streamlines in the 256^3 Aneurysm II dataset. (a) Initial lines. (b) Line encoding in 64^3 voxel grid with exact curve-voxel intersections. (c) Line encoding in 64^3 voxel grid with curve-voxel intersections quantized to 8^2 bins. (d) Line encoding in 256^3 voxel grid with curve-voxel intersections quantized to 8^2 bins.

6.3.3. GPU implementation

The per-voxel linked list representation used to store the lines is neither memory efficient, as each line stores a pointer to the following element, nor can subsequent lines be accessed quickly, as they are not stored consecutively in memory and, thus, cached reads are prohibited. Therefore, before uploading the data to the GPU, the lines are reordered in memory so that all lines passing through the same voxel are stored in consecutive elements of a linear array. The final data structure stores for each voxel a header, which stores the number of lines for that voxel, and a pointer to the array element storing the first line passing through it.

Constructing the voxel model on the GPU consists of three stages: Firstly, in parallel for every initial curve we compute the lines per voxel and write them into a linear append buffer. Every segment is assigned the unique ID of the voxel it is contained in. Next, the buffer is sorted with respect to the voxel IDs, so that all lines falling into the same voxel are located consecutively in the buffer. An exclusive parallel prefix sum is then computed in-place over the buffer to count the number of line per voxel. Now, for every voxel the start index of its line set can be determined from the content of the buffer, and stored in a separate voxel buffer at the corresponding location. This buffer is used at render time to look up at which position in the global buffer the lines for a particular voxel are stored.

6.3.4. LoD construction

One important property of classical voxel models for surfaces is that a LoD structure can be generated efficiently by simple averaging operations on the voxel values to aggregate information. For a line set that is voxelized as proposed in our work, such an averaging operation cannot be applied immediately because a) an averaging operator for lines first needs to be defined, and b) every voxel might store not only one but many lines. It is worth noting that regardless of how a) and b) are addressed, it is impossible, in general, to represent the lines in one voxel by one single average line so that continuity with the average lines in adjacent voxels is ensured.

We address the problem of LoD construction for a voxelized line set as follows: Since we intend to use the LoD structure in particular to accelerate the simulation of global illumination effects, we derive a scalar indicator for the amount of light that is blocked by the lines in a single voxel. By using standard averaging operators, an octree LoD structure can then be generated in a

straight forward way from the indicator field. For every voxel, we compute an average *density* value ρ from the lines passing through it, by taking into account the lines' lengths and opacities:

$$\rho = \sum_{l_i \in L} length(l_i)\sigma_i, \tag{6.1}$$

where L is the set of lines in the voxel, and l_i and σ_i are the length and opacity of the i-th line in that voxel, respectively. The lines' opacities are either set to a constant values or assigned individually via a transfer function. Finally, an octree is build bottom-up by averaging the density values in 2^3 voxels into one voxel at the next coarser level (see Fig. 6.6).

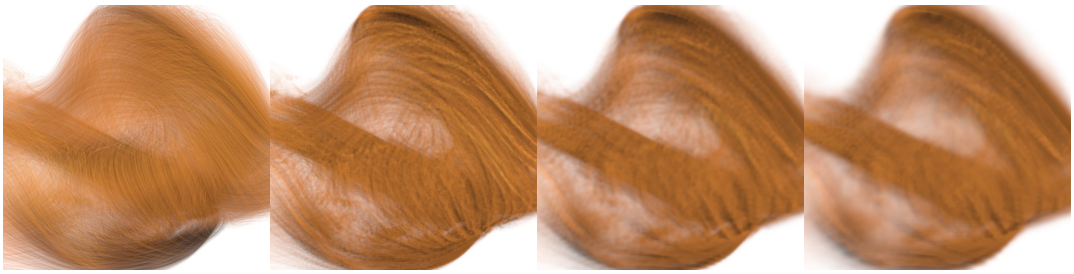

Figure 6.6.: Volume rendering of the initial line density per voxel as computed via eqn. (6.1), and the line density values at the first, second, and third LoD.

The density values can be interpreted as per-voxel opacities telling how much light is blocked by a voxel. Since the amount of blocking depends also on how the lines are oriented with respect to the incoming light, direction-dependent opacity values are favorable in principle [GM05]. Even though they can be generated in a straight forward way, by simulating the attenuation from a number of selected directions via ray-casting, we did not consider this option in the current work to avoid storing many values per voxel and, thus, increasing the memory consumption significantly.

In addition to the per-voxel opacity values, one representative average line is computed for every voxel; by averaging separately the start points and end points of all lines in a voxel. This gives two average points which are snapped to the closest bin on the voxel boundaries, and from which the average line is computed. Care has to be taken regarding the orientation of lines, i.e., when two averaged lines have vastly opposite directions. Therefore, before considering a new pair of start and end point, we first test whether the corresponding line has an angle of more than 90 with the previous line, and we flip the line if this is the case. We also attempt to establish connectivity between representative lines in adjacent voxels if their endpoints are in the same voxel face, by snapping the endpoints to the quantization bin into which the average of both

points is falling. In this way, we can often achieve continuous representative curves, even though it is clear that in general such a continuous fit is not possible. Fig. 6.7 illustrates a multi-resolution representation of a set of curves. Note in particular how well in certain regions the average curves even on the coarser resolution levels represent the initial curves.

Figure 6.7.: Rasterization of the initial curves (left) compared to 1st (middle) and 2nd (right) LoD.

The representative lines are stored in addition to the per-voxel line set, and a LoD representation can be computed by propagating these lines to ever coareser resolution levels. The average per-voxel opacity values at different resolution levels are stored in a set of 3D texture maps. By using these quantities we can efficiently realize a number of rendering accelerations and effects, which we will introduce in Sec. 6.4.

6.4. Voxel-based Line Raycasting

Once the voxel-based line representation has been constructed and the data is residing in GPU memory, volume ray-casting can be used to render the lines in front-to-back order. For every pixel a ray is cast through the voxel grid, thereby going from voxel face to voxel face using a digital differential analyzer algorithm. The voxel grid serves as a search structure to efficiently determine those lines that need to be tested for an intersection with the ray.

Whenever a voxel is hit, the voxel header is read to determine how many lines are stored in that voxel, and if a voxel is empty, it is skipped. Otherwise, the lines are read consecutively and intersected with the ray. Here it is assumed that the lines are in fact tubes with a user-defined radius, so that the intersection test becomes a ray-tube intersection test. This test yields an entry and exit point, from which the distance the ray travels inside the tube can be computed and used, for instance, to simulate attenuation effects. If more than one intersection with tubes are determined, the intersections are first computed and then sorted in place in the correct visibility order.

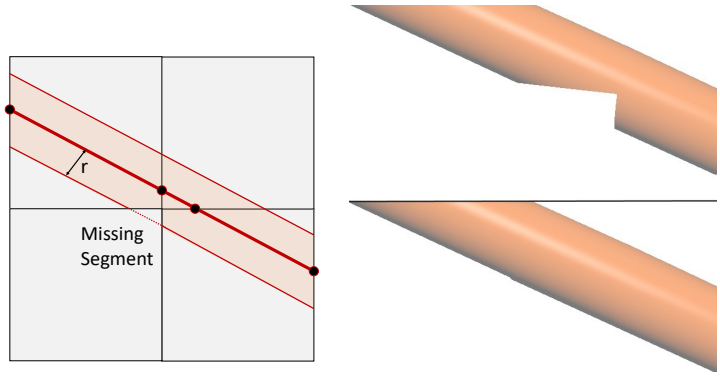

Figure 6.8.: When a tube extends into a voxel, and a ray intersects that voxel but none of the voxels in which the tube's center line is encoded (left), no intersection is found and a piece of the tube is missing (right top). By including neighbouring voxels in the intersection test, missing intersection points are found (right bottom).

6.4.1. Ray-tube intersections

During ray-casting, a problematic case can occur if a tube stands out of the voxel in which its center line is defined. Since the tube expands into a voxel which may not know the center line, the piece in this voxel cannot be rendered if the ray doesn't intersect any of the voxels in which the center line is encoded. This situation is illustrated in Fig. 6.8. A straightforward solution to this problem is for a given ray and voxel intersection to also test for intersections with lines from adjacent voxels, a solution that unfortunately decreases rendering performance due to the increased number of intersection tests. However, our experiments have shown that the artifacts that occur when neglecting the missing segments are only rarely visible. In particular when rendering tubes with transparency the artifacts are hardly perceivable. As a compromise, we hence during interactive navigation restrict our method to testing against the lines in the voxel hit by the ray. As soon as the camera stands still, adjacent voxels are also taken into account.

Another problematic situation occurs at the joint between adjacent lines, i.e., a gap is produced if the lines do not have the same direction (see Fig. 6.9). The gap is filled by rendering spheres at the end points of the line segments with a radius identical to the tube's radius. To avoid blending the same line twice, we keep track of the IDs of intersected lines using bit operations and consider an intersection point only once.

At every entering ray-tube intersection point, we calculate the tube normal and evaluate a local illumination model. If the tubes are rendered opaque, this only has to be done for the first intersection point. The resulting color value is combined with the tube color, and this color is

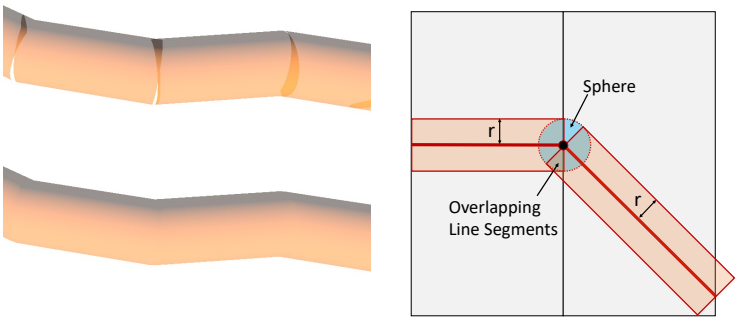

Figure 6.9.: Left top: Overlapping tubes are blended incorrectly and parts are missing. Rendering a sphere at line joints (right) continuously closes the gaps (left bottom).

Figure 6.10.: Opaque line rendering. Left: GPU ray-casting of voxelized lines (8 ms, 256^3 voxel grid, curve-face intersections quantized to 16^2 bins). Right: GPU line rasterization (12 ms).

used as pixel color. Fig. 6.10 compares the rendering of opaque tubes via GPU ray-casting to GPU line rasterization.

Interestingly, in terms of rendering quality rasterization and ray-casting do not seem to show any significant differences at the selected discretization resolution, yet even for opaque lines ray-casting renders already faster than rasterization. The main reason is that ray-casting can effectively employ early-ray termination once the first intersection with a tube is determined. Since rasterization renders the lines in the order they are stored, which is not the visibility order in general, it needs to generate a considerably larger amount of fragments.

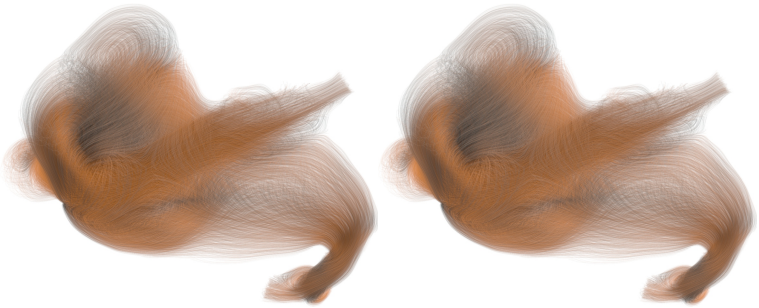

Figure 6.11.: Left: Transparency rendering using fragment linked lists on the GPU (160 ms, 600 MB fragment list). Right: GPU ray-casting (24 ms, 256^3 voxel grid, curve-face intersections quantized to 16^2 bins, 60 MB voxel representation).

Transparency rendering

If the tubes are rendered semi-transparent, at every entering ray-tube intersection an opacity value α is either read from the voxel header or assigned via a transfer function. The opacity value is then used to modulate the tube color, and this color is blended with the pixel color using front-to-back α-compositing, i.e., in the order in which the tube intersections are determined along the ray. The described way of handling opacity is exactly how the final pixel color is computed when semi-transparent lines are rendered via GPU rasterization. In contrast to ray-casting, however, in GPU rasterization all generated fragments first have to be stored and finally sorted per-pixel with respect to increasing depth. Only then can the fragments' colors be blended in correct order.

Fig. 6.11 shows colored and semi-transparent lines, once rendered via GPU rasterization and once via GPU ray-casting. In this situation, the advantage of ray-casting comes out most significantly: Since the ray-tube intersection points are computed in correct visibility order, there is no need to store and finally sort these points for blending. Due to this, the performance gain of ray-casting compared to rasterization now becomes significant; about a factor of 7 for the used dataset.

Furthermore, additional acceleration and quality improvement strategies can be integrated into ray-casting in a straight forward way. Firstly, α-termination, i.e., the termination of a ray once the accumulated opacity exceeds a user-defined threshold, can be used to reduce the number of ray-tube intersection points. If a LoD representation is available, even opacity-acceleration[DH92] can be employed, i.e., increasing the step size along the ray and simultaneously sampling the opacity from ever coarser resolution levels with increasing optical depth. Secondly, instead of considering a constant opacity per tube, the opacity can be made dependent on the distance the ray travels within the tube. Since together with the ray entry point also the ray exit point is computed,

this distance is immediately available. Even the handling of penetrating tubes does not impose any conceptual problem, and only requires to sort the ray-tube intersections locally per voxel.

Shadow simulation

The possibility to efficiently trace arbitrary rays through the voxelized 3D curves can be employed to efficiently simulate global illumination effects such as shadows. Shadows provide additional depth and shape cues, and can significantly enhance the visual perception of the curves geometry and their spatial relationships.

The simulation of hard shadows of a point light source can be realized by sending out shadow rays and testing if the ray hits another tube before it hits a light source. However, as demonstrated in Fig. 6.12b, due to the high frequency shadow patterns that are caused by a dense set of curves, hard shadows rather disturb the visual perception than help to improve it.

We propose the following two approaches to incorporate shadows into the rendering of large line sets without introducing high-frequency shadow patterns. The first approach is to test the shadow rays against the representative lines at a coarser LoD, thus reducing the number of lines that throw a shadow and making the shadows wider and more contiguous (see Fig. 6.12c). The second approach replaces hard shadows by soft shadows, by sampling the line density values along the shadow rays to measure the amount of blocking. (see Fig. 6.12d). Both approaches require to traverse only one single shadow ray towards the light source and can be performed efficiently on the proposed LoD representation.

In the first approach, the shadow rays traverse the voxels of a selected LoD and test for intersections with the representative line in every voxel. Even though we do not avoid hard shadows in this way, by using one single yet thicker line to represent many thinner lines, the frequency of variations from shadow to non-shadow can be reduced significantly. The second approach mimics the effect of an area light source, requiring, in general, to use many rays to estimate how much of the light leaving the area light is blocked. Instead, we sample the line density values with one ray in a way similar to cone-tracing [CNS*11], i.e., by sampling from ever coarser resolution levels with increasing distance from the illuminated point. In particular, we simulate the amount of light that falls onto the point along a cone with an opening angle that subtends one voxel at the finest level.

When comparing the results of both approaches to the rendering of hard shadows in Fig. 6.12a, one can see that the shadow frequency is considerably reduced and the visual perception is improved. The major shadowing effects, on the other hand, are still present in the final renderings, and the spatial relationships between the lines are effectively revealed. When using the first

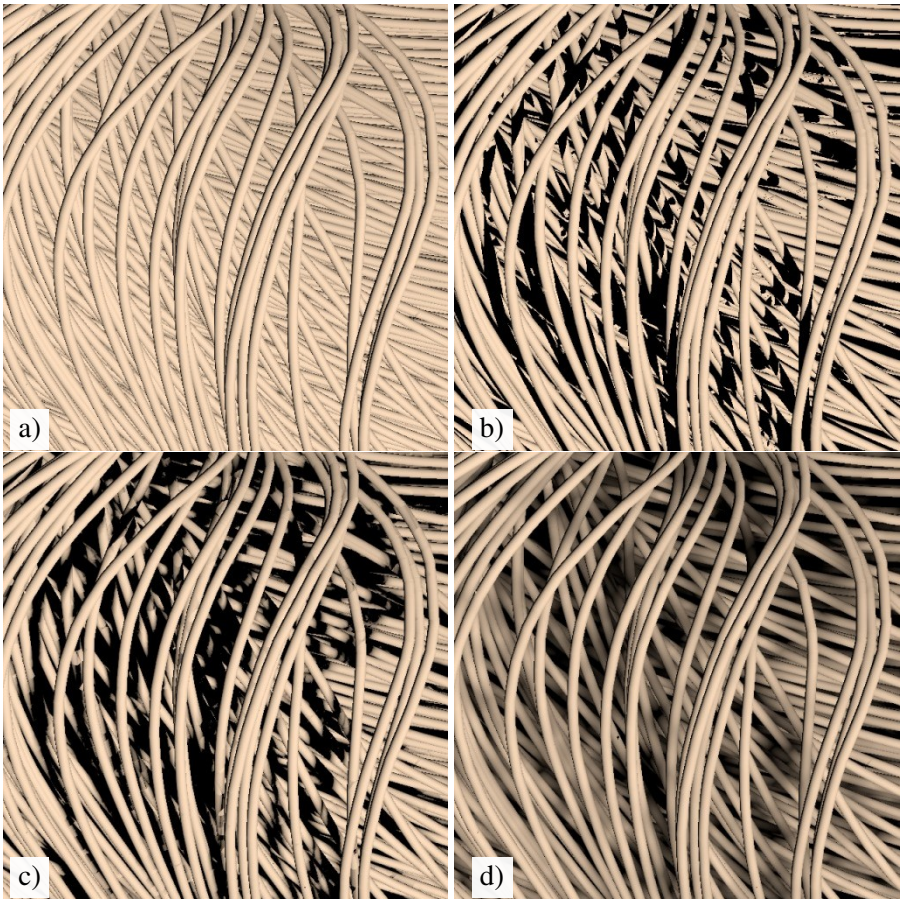

Figure 6.12.: Voxel-based line ray-casting with (a) local illumination (10 ms), (b) point light shadows (22 ms), (c) point light shadows from representative lines at first LoD (12 ms), (d) soft shadows from line density values via cone-tracing (11 ms).

approach, the render time increases about 20% compared to the rendering without shadows; the second approach yields a decrease of about 10%. These only marginal decreases are due to the use of the LoD representation, which requires testing against only one single line per voxel when using the first approach, and interpolating trilinearly in the line density fields at different resolution levels when using the second approach. Since a texture lookup operation takes less time than an explicit ray-tube intersection test, the second approach performs even faster than the first one at almost similar quality.

Ambient occlusion

Both rendering approaches for low-frequency shadows can also be used to simulate ambient occlusions (AO), i.e., soft shadows that occur in the cavities of a 3D object when indirect lighting is cast out onto the scene. As demonstrated in Fig. 6.13, the soft shadows from ambient occlusions help in particular to enhance the spatial separation between individual lines or bundles of lines.

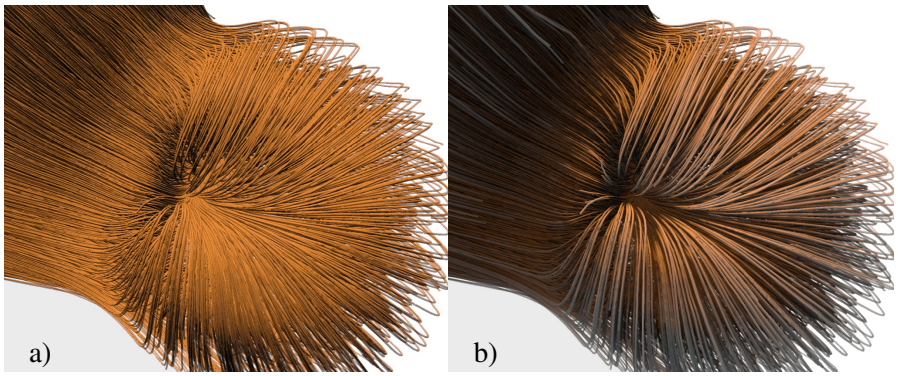

Figure 6.13.: Local lighting (a) vs. ambient occlusion (b).

AO is simulated by casting out rays to sample the surrounding geometry, and computing how much light from an environment map is blocked by this geometry. For simulating shadows we restrict the direction along which we send out rays to the direction of a directional light or towards a point light source, yet ambient occlusion requires to send out rays into the entire upper hemisphere with respect to the normal direction at a surface point. AO can be integrated in a straight forward way into the ray-based rendering pipeline, by spawning at every visible point a number of secondary rays into the hemisphere and calculating whether the environment light is blocked or not. In Fig. 6.14a, 2500 rays per visible point were used to uniformly sample the hemisphere, and these rays were intersected against the tube segments in every voxel to determine

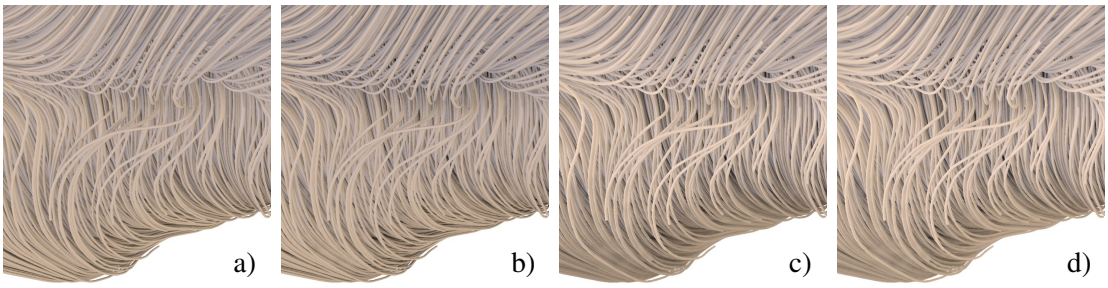

Figure 6.14.: Top: Hemisphere ambient occlusion using 2500 shadow rays. (a) Rays are tested against line geometry (>10 s). (b) Occlusions are estimated from line density values at first LoD (600 ms). Bottom: Pre-computed per voxel spherical occlusions, sampled trilinearly along 2500 (c) and 25 (d) shadow rays at 12 ms and 1130 ms, respectively.

whether the light is blocked or not. The higher variance in the ambient occlusion values when less rays are used can clearly be seen.

Figure 6.15.: The datasets we have used in our experiments: Tornado, Aneurysm I, Aneurysm II, Turbulence, Weather Forecast.

To enable interactive updates of AO values when the scene or light situation changes, the number of rays as well as the number of objects against which the shadow rays are tested need to be reduced. Screen-space approaches [BSD08] compute a rough AO estimation by using a few rays in 2D screen-space (between 8 and 20 in realtime applications), and by testing these rays against rendered surface points in a short radius of influence.

To overcome visual shortcomings of screen-space calculation, we compute AO values in 3D space, and present two acceleration strategies to efficiently approximate the amount of occlusion per tube point. Similar to screen-space ambient occlusion, we restrict the sampling of occluding structures to a radius of influence (in our case 15 voxels at the finest voxel resolution).

The first approach is to approximate the amount of occlusion along a ray by sampling the line density values at the finest level voxel grid via trilinear interpolation (Fig. 6.14b), instead of testing against the tube geometries at this level (Fig. 6.14a). The values along a ray are accumulated until either full blocking is reached (line density ≥ 1) or the ray reaches beyond the radius of influence of leaves the domain. The occlusion values are finally integrated over all rays and normalized by dividing through the number of rays.

The renderings in Figs. 6.14a and b show that the AO values vary strongly around the tube axis, which is due the hemisphere sampling of occlusions wrt to the varying normal direction. Due to the many tubes that are rendered, this adds high frequent intensity variations which rather disturb the visual impression than help to enhance the spatial relationships between the tubes. To avoid this effect, we propose a second approach which computes point-wise AO values independent of the surface orientation by considering occlusions in the entire sphere around each visible point (see Fig. 6.14c and d). The AO values are first approximated per voxel in a pre-process, and at runtime these values are trilinearly interpolated at the locations of the visible tube points. In this way the AO values reflect the local spherical surrounding of a line rather than the surrounding in the normal direction at the tube points. This emphasizes in a far better way the embedding of a line in the surrounding set of lines.

6.5. Results and Discussion

In this section, we analyze the quality, memory consumption, and performance of our approach. All times were measured on a standard desktop PC, equipped with an Intel Xeon E5-1650 v3 CPU with 6×3.50 GHz, 32 GB RAM, and an NVIDIA GeForce GTX 970 graphics card with 4 GB VRAM. For all renderings, the view port was set to 1920×1080.

We used the following datasets to analyze the quality, memory requirements, and performance of the proposed voxel-based rendering approach for line sets (see Fig. 6.15):

- **Tornado**: 1000 randomly seeded streamlines in a flow forming a tornado.
- **Aneurysm I / II**: 4700 / 9200 randomly seeded streamlines in the interior of two aneurysms [BMC14]. Streamlines were advected up to the vascular wall, resulting in empty space up to the cuboid domain boundaries.
- **Turbulence**: 50000 domain-filling streamlines advected in a forced turbulence field of resolution 1024^3 as described by Aluie et al. [AEE*16].
- **Weather Forecast**: 212000 domain-filling path lines computed over 96 hours each on the wind field of a forecast by the European Centre for Medium-Range Weather Forecasts.

6.5.1. Quality analysis

To analyze the effect of the resolution of the voxel grid and the quantization structure on reconstruction quality, we performed a number of experiments with the datasets listed above. The subjective visual quality of the rendered images is hard to measure, wherefore we attempt to

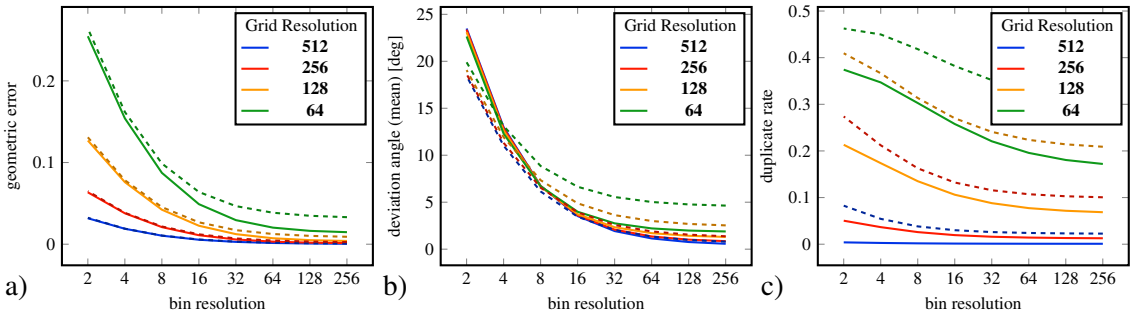

Figure 6.16.: Comparing the Aneurysm II (solid line) with the Weather Forecast (dashed line) dataset using different metrics: (a) Distance Error (in units of voxel size at a grid of 64^3), (b) tangent deviation, (c) Number of duplicates in proportion to the number of generated segments

quantify the quality objectively by measuring the following quantities: For different resolutions we measured the mean Hausdorff distance between the original curves and their piecewise linear approximations (Fig. 6.16a), the mean angle between the tangents at the original curve points and the curve's piecewise linear approximations (Fig. 6.16b), and the number of line segments falling onto each other due to the quantization of vertex coordinates (Fig. 6.16c).

Fig. 6.16a indicates that already at a voxel grid resolution of 128^3 and a quantization resolution of 32, suitable reconstruction accuracy is achieved. At a voxel grid resolution of 64^3, geometric features get lost and cannot be faithfully reconstructed even at high quantization resolution. Fig. 6.16b shows essentially the same dependencies, yet one can observe a stronger effect of the quantization resolution. The local directional changes of the curve tangents due to displacements of the vertex coordinates is scale independent and depends mainly on the quantization resolution. From Fig. 6.16c it can be seen that at a voxel grid resolution of 128^3 and higher, and starting at a quantization resolution of 32, the number of duplicate lines is below 0.1% of all encoded lines and doesn't change significantly beyond these resolutions. The maximum Hausdorff distance, on the other hand, is always bounded by the voxel diagonal, and the directional error can be up to 180 degrees, if a curve makes a loop in a voxel.

Supported by our analysis, and further verified by comparing the visual quality of the original curves and the voxel-based curve representations, we found that a voxel grid resolution of 256^3 and a quantization level of 32 is for all cases the resolution at which a further increase in voxel resolution or bin size only causes little improvement in error quantities This is also evidenced by the close-up views in Fig. 6.18, where no apparent differences between the original curves and their voxelized counterparts can be perceived. Due to this, we decided to use these resolutions in the performance analysis and the analysis of illumination effects below.

6.5.2. Memory statistics

Our approach requires a 5 byte header for every voxel, to indicate how many lines are encoded per voxel (1 byte, restricting to a maximum of $2^8 - 1$ lines per voxel) and to reference the memory address where the lines are stored (4 bytes). In addition, every line is encoded by specifying at which of the 6 voxel faces the two endpoints are located ($2 \cdot 3$ bits), and addressing the sub-face on each voxel face to which the endpoints are quantized ($2 \cdot 10$ bits for a quantization resolution of 32^2). Furthermore, we use 1 byte per line to map to a transparency or color value and 5 bits to encode a local line ID. For a selected quantization resolution, the minimum number of bytes that is required to encode this information is used. Overall, our approach requires 4, 5, and 6 bytes per line, respectively, when quantization resolutions of (4,8), (16,32) and (64,128) are used.

Table 6.1 compares the memory consumption for different resolution and quantization levels with the memory that is required to store the original lines on the GPU, using 32 bit float values per vertex component and 1 byte per vertex to map to transparency or color. In the second column we show in brackets the memory that is required in average over multiple views to store the per-pixel fragment lists when transparent lines are rendered. It can be seen that for all but the Tornado dataset the voxel-based representation at our selected resolution level ($256^3/32^3$) has a significantly lower memory footprint. Even when opaque lines are rendered and the extra memory for storing the fragment lists is not required, a rather moderate increase of about a factor of 4 to 5 is observed. One exception is the Weather Forecast dataset, which is comprised of many curves with only few long lines. These lines are split into many smaller lines and stored in the voxel representation, so that the memory requirement is significantly increased compared to the original line representation.

It is in particular interesting that due to memory limitations on the GPU the Weather Forecast dataset cannot be rendered via GPU rasterization if transparency is used (see Fig. 6.17). Our approach requires only 144 MB to store the view-independent voxel-based representation on the GPU.

Table 6.1.: Statistics on memory consumption and preprocessing time. The original geometry is encoded in three float values per vertex. In the second column, we give in brackets the memory that is required to store the per-pixel fragment lists on the GPU. The remaining columns show the memory that is used by the voxel-based representation—with voxel grid resolution V^3 and quantization resolution Q^2 given as V^3/Q^2—, and in brackets the preprocessing time to generate the representations. Per vertex and per quantized line a 1 byte index is stored.

Dataset	Lines (vertices per line)	Original Geometry (Linked List)	$512^3/256^2$ (Preprocess)	$256^3/128^2$	$256^3/32^2$	$128^3/128^2$	$128^3/32^2$	$64^3/8^2$
Tornado	1000 (250)	3 MB (100 MB)	811 MB (1.4 s)	103 MB (0.2 s)	103 MB (0.2 s)	14 MB (0.1 s)	14 MB (0.1 s)	2 MB (0.1 s)
Aneurysm I	4700 (410)	25 MB (600 MB)	731 MB (1.6 s)	111 MB (0.4 s)	107 MB (0.4 s)	25 MB (0.2 s)	22 MB (0.2 s)	6 MB (0.3 s)
Aneurysm II	9200 (367)	44 MB (750 MB)	702 MB (1.9 s)	116 MB (0.6 s)	109 MB (0.6 s)	29 MB (0.5 s)	26 MB (0.5 s)	8 MB (0.4 s)
Turbulence	50000 (220)	143 MB (2500 MB)	1087 MB (3.5 s)	221 MB (1.4 s)	201 MB (1.4 s)	72 MB (1.3 s)	62 MB (1.3 s)	21 MB (1.0 s)
Forecast	212000 (13)	36 MB (>4000 MB)	573 MB (5.6 s)	209 MB (4.6 s)	177 MB (4.6 s)	98 MB (5.0 s)	82 MB (5.0 s)	31 MB (4.7 s)

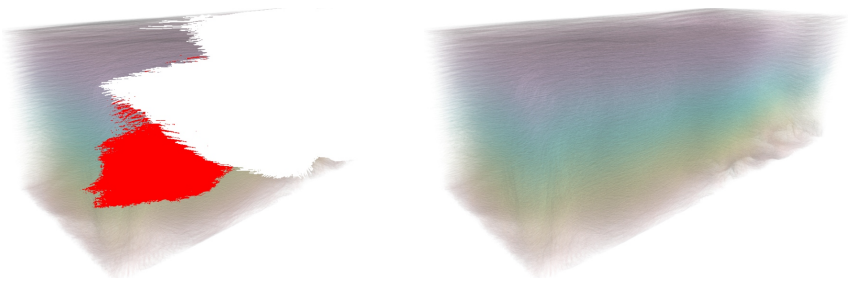

Figure 6.17.: Left: GPU rasterization using fragment linked lists fails due to memory limitations. Red indicates 512 or more fragments fall onto a pixel, when not all fragments could be stored due to memory limitations the pixel is set to white. Right: Voxel-based ray-casting renders at 23 ms.

6.5.3. Performance analysis

For all datasets, we compare the performance of rasterization-based line rendering with and without fragment lists to voxel-based line ray-casting (see Tab. 6.2). For rasterization-based rendering, which unfolds a given line primitive into a number of triangles approximating a cylinder, we can also unfold the line into a quadrilateral and let the generated fragments perform an analytical ray-cylinder intersection test. This approach, even though it frees the geometry shader, requires some additional stitching geometry to achieve the appearance of a continuous tube. Furthermore, it increases the load in the pixel shader, which is already the bottleneck when using transparency; the use case we focus on in this work. Since even for opaque lines we observed only a slight performance increase when using this approach, we decided to refrain from using it. In all of our experiments we let the geometry shader generate 8×2 triangles per line segment.

To also compare the performance of our approach to that of triangle-based GPU ray-tracing, for all datasets we saved the generated triangles to a file and used them as input geometry for the OptiX ray-tracing framework [PBD*10]. We selected Spatial Splits in Bounding Volume Hierarchies (SBVH) [SFD09] as acceleration structure. Besides the fact that both the Turbulence and Weather Forecast datasets could not be rendered due to memory limitations, for the smaller datasets (Aneurysm I and II) we observed almost similar frame rates when rendering opaque tubes. When rendering transparent tubes, however, the performance dropped significantly, of up to a factor of 10. We attribute this to the fact that SBVH allows skipping empty space surrounding the line sets efficiently, yet when the lines are dense in the interior (as in Aneurysm I and II), repeated traversal operations slow down the performance. If more space in the interior is empty, this limitation is reduced: OptiX renders the opaque lines in Tornado at 160 fps, while our approach renders at 83 fps. However, it is fair to say that in this case also our approach can be

Table 6.2.: Performance statistics for different datasets. Rendering performance was evaluated using a constant per-line opacity value of 25 percent and 100 percent, respectively.

	Timings Semi Opaque		Timings Opaque	
Dataset	Linked List	Raycasting	Rasterization	Raycasting
Tornado	4 ms/250 fps	15 ms/67 fps	2 ms/500 fps	12 ms/83 fps
Aneurysm I	46 ms/22 fps	25 ms/40 fps	8 ms/125 fps	9 ms/111 fps
Aneurysm II	160 ms/6 fps	27 ms/37 fps	14 ms/71 fps	12 ms/83 fps
Turbulence	380 ms/3 fps	24 ms/42 fps	65 ms/15 fps	6 ms/167 fps
Forecast	overflow	23 ms/43 fps	29 ms/34 fps	4 ms/250 fps

speeded up significantly by using the octree voxel grid to skip empty space. At a per-line opacity of 25 percent, OptiX and our approach (w/o empty space skipping) render at 35 fps and 67 fps, respectively.

In Fig. 6.21 we show the use of transparency to reveal interior structures that are occluded when opaque lines are rendered. Even when opaque lines are rendered, so that fragment lists and sorting is not required in GPU rasterization, the larger datasets can be rendered at higher rates using voxel-based ray-casting. The main reason is that a ray can be terminated immediately when the first ray-tube intersection is computed, while GPU rasterization always needs to generate all fragments even if the early depth-test can discard many of them before entering the fragment stage. On the other hand, for the Tornado dataset, where the line density is rather low so that many rays need to be traversed through the entire voxel grid, voxel-based ray-casting performs slower than rasterization-based rendering. This still holds when transparency is used. The fact that in rasterization-based rendering using transparency all fragments need to be sorted explains the over-linear increase of the render times, while the render times remain almost constant when ray-casting is used on the voxel-based representation.

The worst case scenario for the ray-caster is a very dense line set with very low line opacity. This prevents rays from terminating early, so that all lines along the rays have to be accessed and tested for intersections. Even though this scenario is quite unusual, we analyzed the rendering performance in this situation for the Weather Forecast and Turbulence dataset. In this case, the Forecast dataset can still be ray-cast in roughly 144 ms, while the dataset cannot be rendered via rasterization because the fragment lists exceed the available GPU memory. When using the Turbulence dataset with very low line opacity, GPU rasterization and voxel-based ray-casting render at 380 ms and 124 ms, respectively, demonstrating the efficiency or our approach even in this extreme situation.

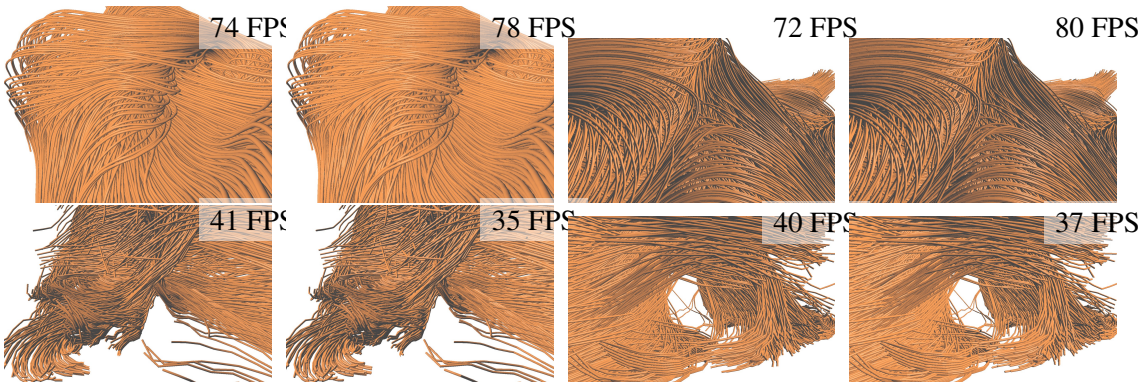

Figure 6.18.: Comparison between rasterization-based rendering (left) and voxel-based line raycasting using 256^3 voxels partitioned into 32^2 bins per face (right), for the Aneurysm II and the Weather Forecast dataset. The lines are rendered opaque with local illumination.

6.5.4. Illumination effects

The secondary rays used to simulate global illumination effects like shadows and ambient occlusions are solely traversed on the first LoD level of the voxel grid. Therefore, these effects cause only a moderate increase in the render time, yet they can significantly improve the spatial perception of the rendered line structures. This is demonstrated in Fig. 6.19, where opaque lines are rendered with soft shadows and ambient occlusions (right).

For instance, for 100 rays sampling in a sphere with a radius of roughly 5 times the voxel size (with a step size of 1 voxel size), ambient occlusions for all 256^3 voxels can be computed in roughly 9 ms. Soft shadows can be computed in about 2 ms. The computed illumination values can be stored inside our voxel representation and only have to be updated if the scene changes.

Fig. 6.18 shows a comparison between reference images produced by a rasterizer using fragment linked lists (left) and images produced using our ray-casting approach using a memory efficient representation (right). In all images an attribute stored for each line segment was mapped by a transfer function to a color value. Opacity values were manually predefined. The memory efficient representation introduces some artifacts like jittered line segments and visible gradations at the voxel boundaries as attributes are not interpolated in between. Comparing the reference image and the high resolution represenation, no significant differences are observable.

In Fig. 6.19, lines are rendered with local illumination (left) and soft shadows in combination with ambient occlusions (right). When rendering huge amounts of semi-transparent lines, single lines cannot be distinguished any longer. Ambient occlusions help to distinguish individual curves, since less light is reaching occluded curves. Shadow effects enhance the overall structure

of the dataset and indicate the relative position of curves to each other. In combination with ambient occlusions, details in shadowed regions are preserved.

6.6. Conclusion and future work

We have introduced a new approach for visualizing large 3D line sets, by using GPU ray-casting on a novel voxel-based line representation. For large sets of transparent lines we have shown significant performance improvements over rasterization-based approaches. The voxel-model gives rise to an efficient integration of local and global illumination effects.

A limitation of our approach is the lack of hardware accelerated anti-aliasing. While anti-aliasing in triangle rasterization does not have a significant effect on performance, it is expensive in ray-casting since several rays have to be traced per pixel. Another limitation is with respect to the addition of new lines. Since the voxels are densely packed in one linear array on the GPU, the whole memory needs to be restructured when new lines are added. On the other hand, since generating the voxel model is sufficiently fast, even a complete re-voxelization of all lines can be performed at high speed.

In the future we will further optimize the voxelization process so that even time-varying line sets can be visualized at interactive rates, and we will incorporate empty space skipping using the voxel hierarchy. Moreover, it will be interesting to investigate the use of adaptive line quantization strategies, for instance, based on line curvature, to generate an adaptively refined voxel model.

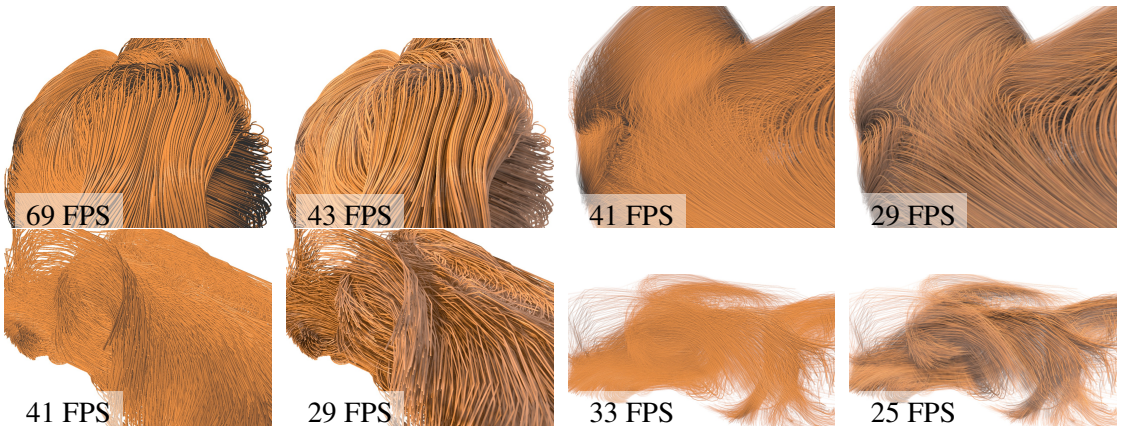

Figure 6.19.: Lines are rendered with local illumination (left) and soft shadows in combination with ambient occlusions (right). When rendering huge amounts of semi-transparent curves, single curves cannot be distinguished any longer. Ambient occlusions help to distinguish individual curves, since less light is reaching occluded curves. Shadow effects enhance the overall structure of the dataset and indicate the relative position of curves to each other. In combination with ambient occlusions, details in shadowed regions are preserved.

Figure 6.20.: First: Local illumination. Second: Soft shadows from line density values via cone-tracing. Third: Soft shadows in combination with ambient occlusion using 50 sample rays per voxel. The Aneurysm II dataset is shown, while continuously updating corresponding illumination values.

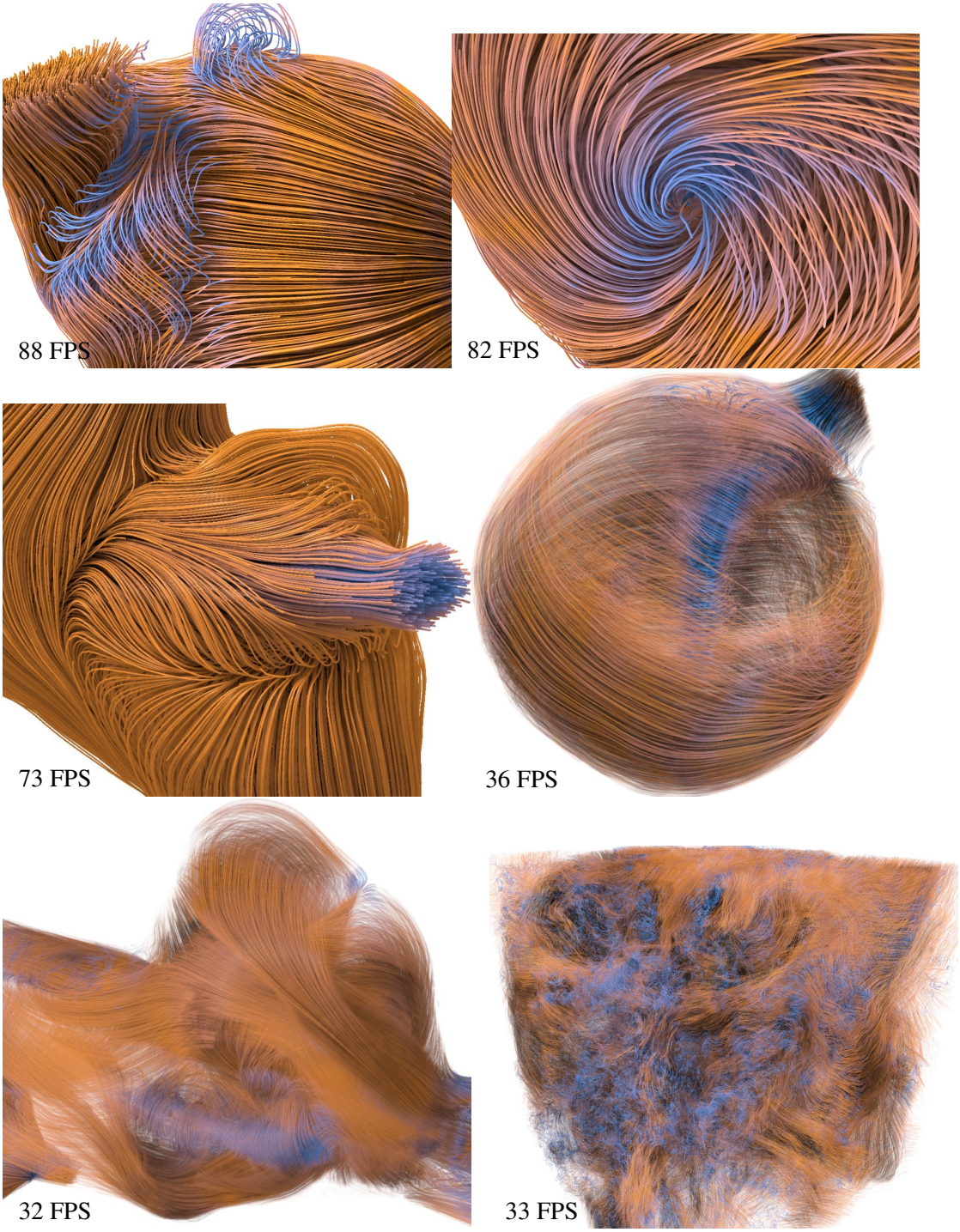

Figure 6.21.: Renderings of Aneurysm I (top and middle right), Aneurysm II (middle right and bottom left), and Turbulence (bottom right). Soft shadows and ambient occlusions are enhancing the depth perception. Semi opaque renderings are revealing inner structures.

7

Paper C:
Interactive Visual Exploration of Line Clusters[1]

Summary We propose a visualization approach to interactively explore the structure of clusters of lines in 3D space. We introduce cluster consistency fields to indicate the local consistency of the lines in a cluster depending on line density and dispersion of line directions. Via brushing the user can select a focus region where lines are shown, and the consistency fields are used to automatically control the density of displayed lines according to information content. The brush is automatically continued along the gradient of the consistency field towards high information regions, or along a derived mean direction field to reveal major pathways. For a given line clustering, visualizations of cluster hulls are added to preserve context information.

Contribution The main author provided the theoretical principles to derive cluster consistency fields from a line data set. He furthermore developed a smart brush that makes use of derived cluster consistency fields. Finally, he devised a smart selection refinement mechanic based on the current line selection. The author developed a tool including these principles to show the effectiveness. This approach was refined based on discussions with the co-author.

[1]This article was published in Vision, Modeling and Visualization, Mathias Kanzler, Rüdiger Westermann, Interactive Visual Exploration of Line Clusters

7.1. Introduction

Clustering of lines is used in many applications [MVvW05, AT06, AMP10, YWSC12] to condense large line sets to few representative lines that expose the major geometric trends in the initial set. Here, we refer to a line as a sequence of vertices where every pair of consecutive vertices is connected by a straight line segment. While the representative lines serve as a basis for the application-specific analysis process, the distribution and spatial coverage of the lines per cluster, and the variation of these characteristics among clusters is usually not examined any further. Instead, it is quietly assumed that the lines in one cluster follow the representative line closely.

For large line sets and a reasonable number of clusters, however, the lines in one cluster usually show significant local deviations from the representative, requiring indicators of where deviations occur and how these deviations reflect in the lines' geometries. Computing a large enough set of clusters so that the per-cluster deviations are small, on the other hand, contradicts the requirement to effectively reduce the number of representatives for the upcoming analysis process. Thus, one cannot avoid significant local deviations in general, yet one can try to analyze these deviations in 3D space to better interpret the consistency of grouped elements. Such an analysis is also important to compare different clusterings to each other, and to eventually shed light on specific properties of the used clustering algorithm and distance metric.

To perform a consistency analysis of a given cluster, the lines belonging to that cluster can be shown in addition to the representative line, either entirely or only to a certain percent-

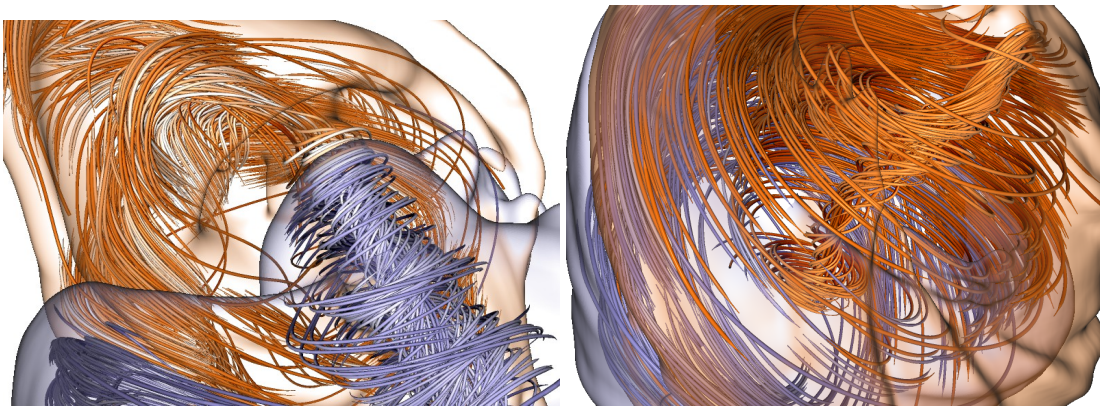

Figure 7.1.: Clusters of lines are analyzed by visualizing cluster-specific contextual information in the form of cluster hulls (colored surfaces), and using a measure of local line consistency within a cluster to adapt the line density automatically so that important structures are conveyed. Via brushing the user can interactively select where automated density control is applied.

age [YWSC12, OLK*14]. While the first approach can quickly introduce occlusions, the second one can withhold important information if the lines are thinned out randomly. Yu et al. [YWSC12] address these shortcomings by showing in addition to the representative line also lines indicating the cluster boundaries. This can reveal the spatial extent of a cluster, but it cannot effectively reveal a cluster's consistency in the interior.

Our contribution: Building upon these previous works, our approach enables an interactive visual exploration of the local consistency of a cluster of lines, in context of the clusters in the current clustering. We introduce a measure of cluster consistency in 3D space, which considers the local density of lines as well as the variance of the local line directions. Boundary surfaces in the scalar-valued consistency field are used to indicate a cluster's shape.

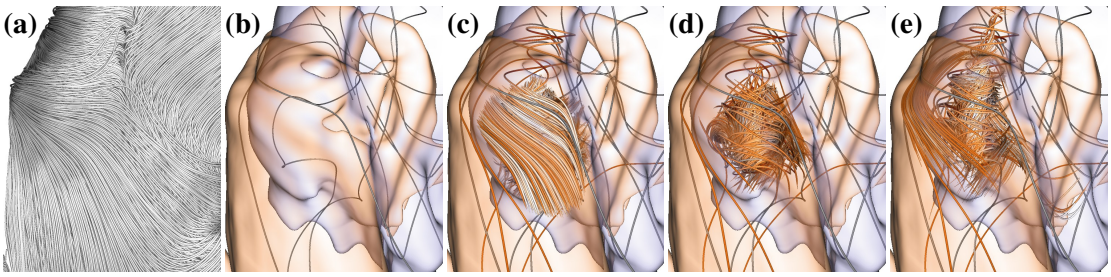

Figure 7.2.: (a) Initial line set. (b) Representative lines of 8 clusters and hulls of two clusters. (c) All lines going through a brushed focus region. (d) View-dependent line density control using local consistency. (e) Automatic refinement and continuation of focus region towards important structures in the surrounding.

Starting with a context view showing cluster boundaries and representatives, we let the user brush a region where lines should be added. By using a hierarchy of line clusters [KFW16], the line density in the brushed region is controlled to emphasize regions where lines are strongly diverging, at the same time showing only few representative lines in regions where lines behave similar. The object-space cluster consistency is used to determine the line density. To further guide the user towards regions of low consistency, the brush is continued automatically along the gradient of the consistency field. Alternatively, the brush can be continued along the major trends in the line set by following bundles of similar lines.

Our specific contributions are:

- *Cluster consistency fields* as spatial indicators for the line consistency in a cluster.
- The use of cluster consistency fields to reveal the most representative lines in a brushed focus region.
- A method to continue a user-selected brush along major pathways and towards regions of high information content.

We demonstrate our approach by visualizing a number of real-world examples comprised of large line sets. Some of these data sets are shown in Fig. 7.1, hinting towards the specific visualizations targeted in this work. Equipped with a cluster panel that shows an abstract view of a given line clustering, the user can interactively select clusters based on their consistency, and compare clusters using their hulls and overlaps.

7.2. Related Work

Our approach uses a hierarchical clustering of a line set. Let us refer to the book by Jain [Jai10] for an exhaustive summary of available clustering algorithms. An overview of similarity measures for lines is given in the comparative study by Zhang et al. [ZHT06]. Oeltze and co-workers [OLK*14] evaluate clustering approaches for streamlines using geometry-based similarity measures. Different clustering approaches and similarity measures for fiber tracts in diffusion tensor imaging data have been evaluated by Moberts et al. [MVvW05].

Yu et al. [YWSC12] use hierarchical clustering to generate a small representative set of streamlines in a flow field. They further select streamlines close to the cluster boundaries to convey the spatial extent of computed clusters. Kanzler et al. [KFW16] use a line cluster hierarchy for view-dependent line density control. Every segment of a line determines a level in the pre-computed line hierarchy depending on a screen-space importance measure, and only if the line is the representative line at this level the segment is drawn. For the construction of a line cluster hierarchy we follow the work by Kanzler et al., which uses a variant of Agglomerative Hierarchical Clustering (AHC) with single linkage and graph matching to construct a fully balanced cluster tree.

Both aforementioned approaches try to find the minimum number of lines that represent the important structures in a given line set. This process can be performed either via importance- or similarity-based criteria in object-space, or in screen-space by selecting the rendered lines dynamically on a frame-to-frame basis. Screen-space approaches determine for each new view the subset of lines to be rendered so that occlusions are reduced and more important lines are favored over less important ones [MCHM10, LMSC11, MWS13]. Indicators for the amount of occlusion in the rendered images are based on the "overdraw", i.e., the number of projected line points per pixel [MCHM10, MWS13], or the maximum projected entropy per pixel [LMSC11]. View-dependent opacity control was introduced by Guenther et al. [GRT13] to fade out those parts of foreground lines that occlude more important ones.

For the selection of representative lines in object-space a number of different criteria have been proposed, for instance, based on the line density [TB96, MHHI98, MTHG03, SHH*07], the line

distribution [JL97, LMG06, LHS08, SLCZ09, RPP*09], or the coverage of specific flow or line features [VKP00, YKP05, CCK07a, MJL*12]. Behrendt et al. [BBB*18] suggest an explorative approach to select lines based on surface features. The explorative selection of point clouds can be accomplished by encircling a target location as described by Yu et al. [YEII12]. An interactive focus and context approach using a 2D lens to analyze flow structures is described by Gasteiger et al. [GNBP11] and Fuhrmann et al. [FG98]. While Jackson et al. [JCK12] are using a dedicated haptic device to explore flow structures in 3D, our method is designed for standard input devices (mouse and keyboard). Our proposed consistency measure to control the line density is related to the measure of information entropy that was introduced by Xu et al. [XLS10]. They use the local directional variation of a given vector field as entropy measure, and generate an entropy field that is then used to guide the placement of streamlines. We introduce a similar measure using the local directional variations of lines in a cluster, and we further extend this measure to also consider the line density.

7.3. Method Overview

The input of our method consists of a set of lines, e.g., streamlines, where each line is represented by a sequence of vertices. If no initial clustering is given, we use the mean-of-closest-point distance [CGG04] as distance metric and compute a balanced cluster tree using graph matching [KFW16].

Every node in the computed cluster tree represents a cluster of lines that are grouped together based on the used distance metric. For selecting a single representative line at each node, we perform a top-down sweep through the tree as follows: We traverse the tree in breadth-first order, and whenever we encounter a node that has not been visited we determine the representative line, i.e., the line that has the smallest average mean-of-closest-point distance to all lines in the cluster at that node. This line becomes the representative line for every cluster at the nodes along the path from the current node to the leaf node where the line is the only cluster member. In this way we build a hierarchy of representatives which explain as much as possible of the cluster variation at the highest (coarsest) levels. In addition, the hierarchy is nested in that at every inner node one of the child nodes has the same representative than that node. This enables to keep the current representatives and progressively add the new ones when going from one level to the next finer one. Fig. 7.2 (a) and (b), respectively, show a line set and the 8 representative lines (as well as two cluster hulls) at level three of the computed cluster hierarchy.

For every cluster down to a user-selected level a cluster consistency field (CCF) is pre-computed (c.f. Section 7.4). If at run-time a CCF is requested that has not been computed, it is generated

in turn within less than a second on the GPU. A CCF is similar to a visitation volume [Jon08, BCM*08, BFMW12], which is generated by rasterizing lines into a 3D voxel grid and counting the number of lines going through every voxel. Level sets in this field can then be visualized to show the spatial extend of the lines in one cluster, i.e., the cluster hull (see Fig. 7.2 (b)), and multiple hulls can be displayed concurrently to enable a comparative cluster analysis. In addition to line density, we also store a measure of the directional variance of the lines going through a voxel.

While cluster hulls provide context information about a cluster's shape and spatial extent, they do not, in general, show the lines' geometries. To select a region in 3D space where lines are shown, the user can brush a region on a cluster hull, and this region is extruded to 3D in a specific way (c.f. Section 7.5). Via brushing the user increases a visibility parameter in the selected region, requesting additional lines besides the cluster representatives. If all lines passing through the selected region are shown concurrently, however, occlusions are quickly introduced and important structures are hidden (see Fig. 7.2 (c)).

To adapt the density of displayed lines in the selected focus region, we employ the CCF as a measure of information content (c.f. Section 7.5), as shown in Fig. 7.2 (d). Furthermore, since regions conveying high information can be missed when brushing manually, the visibility is transported automatically along the gradient in the CCF towards regions of low consistency (Fig. 7.2 (e)), or along bundles with low directional variance to reveal major trends in the line set. In either case, line density control is active to focus only on the most relevant lines.

7.4. Cluster Consistency Fields

CCFs are computed on the GPU using rasterization of lines into a Cartesian 3D voxel grid. Initially, we compute the resolution $r_x \times r_y \times r_z$ of the voxel grid into which the lines are voxelized. If the resolution is too high, a voxel carries merely information about one single line passing through it, yet if it is too low, the CCF cannot serve the purpose of a local indicator of consistency. In our implementation we link the average distance of the cluster representative to all lines in the cluster to the side length of cube-shaped voxels. Our experiments have shown that this choice gives a good balance between locality and information content. The grid size is set so that the aspect ratio of the bounding box of the lines per cluster is maintained. The vertex coordinates v_i are then transformed to local object coordinates in the range from $(0,0,0)$ to (r_x, r_y, r_z).

The voxel values can be optionally filtered using a low-pass filter with adjustable extent. In this way, high frequencies in whatever values are sampled can be removed, and a smooth consistency representation is obtained. In particular, if a feature is present in one voxel the smoothing process

distributes this information into the surrounding region. In combination with our proposed brush-based focus visualization this leads to an improved continuation of features beyond the brush extent. For this reason we store a separate low-pass filtered version for every CCF we generate.

For each line in parallel, and starting with the first vertex, every pair of vertices v_i and v_{i+1} is processed consecutively. A line through v_i and v_{i+1} is clipped against the voxel boundaries, via line-face intersection tests in the order of their occurrence from v_i to v_{i+1}. If v_i and v_{i+1} are located in the same voxel, no new intersection point is generated. This gives a sequence of voxel-face intersections, and every pair of consecutive intersections represents a line segment that enters into a voxel and exits that voxel. From all line segments that are generated for a single voxel, a number of different quantities are derived to generate different variants of CCFs. These variants are explained in the following.

7.4.1. Line density fields

For each line segment, the real length of the line from the entry to the exit point is computed and added into the CCF. Since in general the first and last vertex of a line do not lie on a face, we consider the length from the first vertex to the first face intersection and from the last face intersection to the last vertex in these cases.

Adding a length value to the CCF means to add this value to a counter at the voxel the line is currently passing through. Note that this is different to the construction of visitation volumes, where every segment adds the same contribution to a voxel regardless of its length. Our approach, in contrast, keeps track of the voxels' fill rates to obtain a more accurate measure of the line density per voxel.

The line density distribution within one cluster provides information about where and how many lines come close together (see Fig. 7.3 (left)). The level set to a threshold just above zero gives an approximate hull enclosing all lines. Regions in which only few isolated lines occur appear as low value regions which can be filtered out by slightly increasing the threshold. Then again, these regions can also be emphasized to show outliers which have been assigned to a cluster even though at least in some region they deviate strongly from the rest of the lines in that cluster. Fig. 7.3 (right) shows the cluster hulls that were rendered as level sets in multiple cluster density fields.

Level sets in the line density field are visualized using direct isosurface raycasting. For shading, gradients are calculated on-the-fly using central differences. The cluster hulls should be as transparent as possible without failing to communicate the outer shape. We therefore

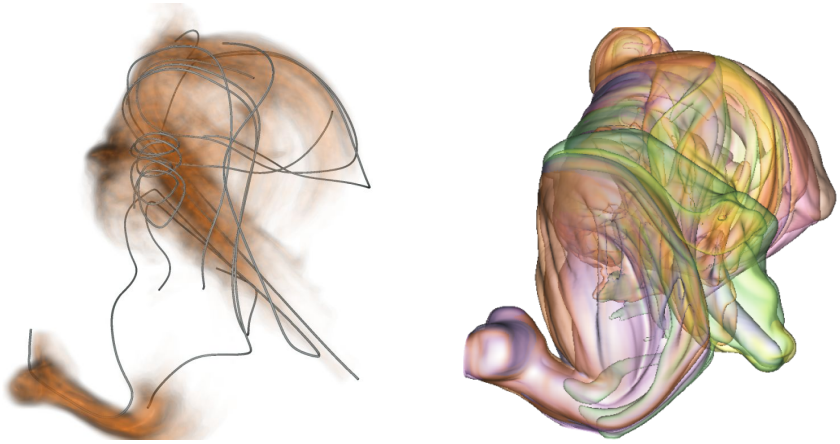

Figure 7.3.: Left: Representative lines of 8 clusters are shown in combination with summed up density of all clusters mapped linearly to opacity. Right: Hulls of 8 clusters using line density fields.

adapt the opacity of the isosurface depending on the angle between the surface normal and view direction [HGH*10]. In this way, the silhouettes of cluster hulls are enhanced while a non-obscured view into the cluster interior is provided otherwise.

7.4.2. Directional variance fields

Nearby lines in one single cluster that follow the same direction are in most cases less interesting than lines following different directions, they even carry redundant information and can thus be condensed. Xu et al. [XLS10] have motivated this statement in the context of vector fields via information entropy as a measure of how much information is carried by a region, depending on the directional uniformity of the vectors in this region. Measuring the variation of lines per volumetric unit allows to directly communicate locations which are potentially relevant for the understanding of the underlying line structure, at the same time determining regions where less lines than available can be shown. In complex line sets, like turbulent structures, lines can deviate strongly in direction. By solely filtering line segments of low directional variance, the resulting image can still suffer from visual clutter. A measure of variation is therefore used as an additional filtering criterion for the visualization algorithm, and we use it in particular in combination with interactive brushing to balance the number of shown lines with respect to the information they carry. Furthermore, the measure is used to guide towards regions of high information content which have not been selected by the user. Similar in spirit to the approach by Xu et al. [XLS10], we calculate per voxel the directional variance of all line segments passing through that voxel.

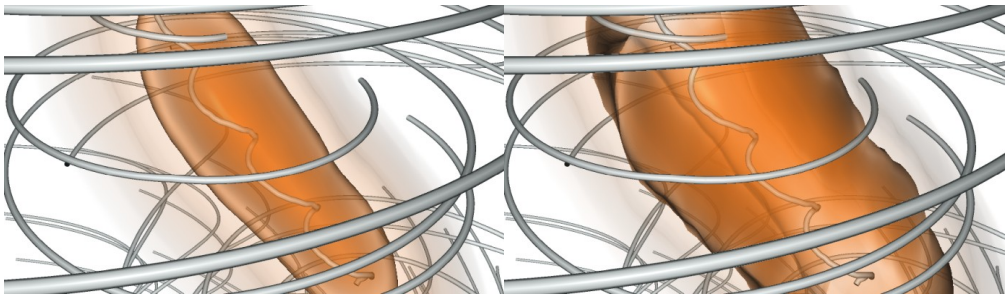

Figure 7.4.: Left: Cluster variance field of the Tornado data set. Color and opacity mapping is as for cluster density. Right: Magnitude of the gradient of the cluster variance field. The maximum gradients are located around the core of the tornado.

Let the line segments in a certain voxel be given as a set of direction vectors $\mathbf{d}_1, ..., \mathbf{d}_n$. The directional variance, weighted by the length of each line segment, is defined as following:

$$\sigma(\mathbf{d}_1, ..., \mathbf{d}_n) = 1 - ||\frac{\sum_i \mathbf{d}_i}{\sum_i ||\mathbf{d}_i||}|| \tag{7.1}$$

If the directional variance is high, σ is close to one, while it is zero if all directions are equal. This is demonstrated in Fig. 7.4, where the directional variance around the core of a tornado is shown. As can be seen, the variance is strongly increasing towards the core, with the magnitude of the gradient of the variance starting to grow strongly with decreasing distance from the core.

7.4.3. Mean direction fields

The mean direction volume of a cluster stores at every voxel one representative average line segment; by averaging separately the start points and end points of all flow-aligned line segments in a voxel. This gives two average points which define the average line direction that is stored per voxel. The mean direction volume is not visualized directly but used for accessing the main flow direction per voxel in constant time.

7.5. Line Density Control via Brushing

We use the pre-computed CCFs to visualize cluster context information, i.e., cluster hulls, and to adapt the density of lines that are shown in a region that is brushed by the user. Cluster boundaries are visualized as isocontours in the cluster density field, and they support the user in identifying the boundaries of the brushable zone. At the same time they hint towards prominent regions, for instance, regions where the consistency of lines is extraordinarily low or high. This forms

the context in which the user queries additional lines, and by this, analyses interactively the internal cluster structure. The user can request a more detailed representation of the line set by interactively using a brush in screen-space. Therefore, the 2D brush region is extruded to 3D along the viewing direction, defining a 3D brush volume in which the visibility is increased. Per default, the visibility is zero everywhere, meaning that only the cluster representatives are visualized. Where the visibility is increased, additional lines are shown, yet their density is controlled by the content of the CCFs as well as occlusion information that is computed for that location. The hierarchical decomposition of a selected cluster is utilized in combination with the visibility values to adapt the line density so that a sparse set of lines is shown where lines are close together, yet enough lines are shown to emphasize the important trends in regions where the cluster consistency is low.

Whenever the brush is moved a visibility volume is filled with visibility values $\in (0, 1)$. Voxels in the brush volume get assigned a high visibility value. Simultaneously, via raycasting from every voxel center towards the viewplane in the cluster density field, the amount of occlusion, i.e., the accumulated density, a voxel receives due to lines in front of it is estimated. Voxels in the visibility volume that are not in the brush volume keep their values from previous frames, yet these values are continuously faded out over time if not queried once again. This allows one to move the camera while brushing without losing the previously brushed location, a mechanism that is demonstrated in Fig. 7.5, where the brush is applied and frozen before moving the camera to another viewpoint.

7.5.1. Local line density control

Based on the cluster tree every line gets assigned a *level-id*, which indicates the lowest level of the tree at which the line is chosen as representative. Note that for every line there is at least one level where this line is a representative line, at latest at the leaf node where the line is the only cluster member. The level-id is used during rendering to decide whether a line segment should be rendered in the selected focus region or not. Note that level-ids are divided by the depth of the cluster tree to obtain values in (0,1).

3D line sets can be rendered efficiently via the rasterization-based rendering pipeline on the GPU, by constructing for every line segment a tube on-the-fly in a geometry shader resulting in 8×2 triangles per line segment. Every vertex that is generated when rendering a tube can access the visibility value at its object-space location, and this value can then be used to modify the rendering style of the tube segment or to discard the segment entirely. Every vertex compares the visibility value with the level-id of the line it belongs to. Only if the visibility value is larger

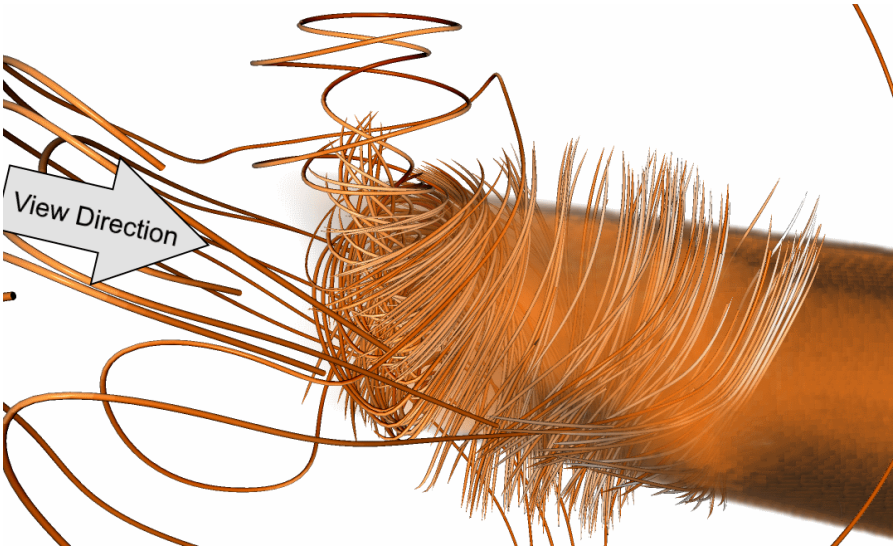

Figure 7.5.: The visibility volume is visualized using raycasting for demonstration purposes. The camera was originally positioned at the top left while using the brush. After brushing, the visibility values were frozen and the camera position was changed. With accumulated density samples, the visibility values are higher.

than the level-id, the vertex is rendered, and it is discarded otherwise. In this way, the line density can be adapted automatically to the local object-space visibility, i.e., in regions of high visibility far more lines are rendered than in low visibility regions. In principle, also other rendering parameters can be controlled by in this way, for instance, opacity or color.

After all visibility values are resolved to per-vertex properties, the actual rendering is performed in two passes: In the first pass all lines are rasterized into a texture including their screen-space depth. We are using an intermediate vertex property for fading lines in or out to avoid sudden changes at the rendered image while moving the brush. Fading in or out is done by modifying the radius of the segment. The second pass performs raycasting and blends the previously rendered lines with the cluster hulls.

Visibility values for each voxel (x,y,z) in the brush volume (a cone with its center axis going through the camera position and the selected point in screen-space) are set by a compute shader according to

$$V_{x,y,z} = 1 - min(max(((\lambda + r) - p_d * D_{Acc} + p_v * Var), 0), 1) \tag{7.2}$$

With this formula, we provide a number of options to accentuate specific structures in the line set: λ defines a threshold that has to be exceeded before visibility values are set. By setting λ

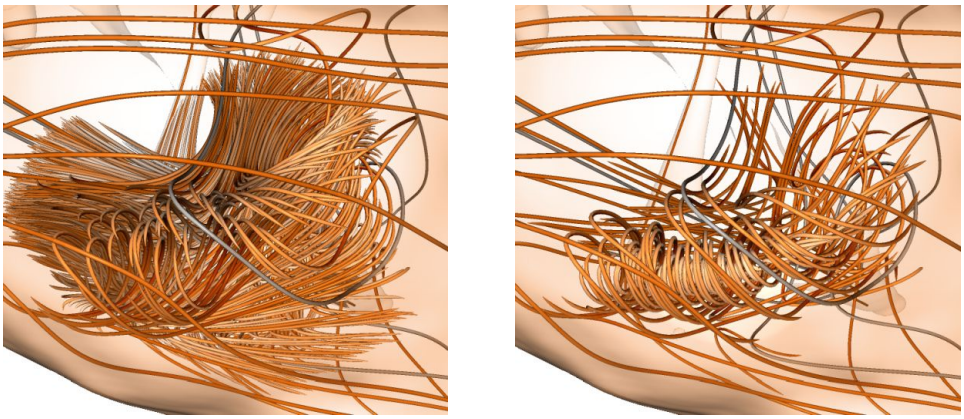

Figure 7.6.: By modifying parameters of the brush, different structures of the Aneurysm II data set are brought out. Left: Lines which are usually covered can be inspected. Right: Lines in regions of high variance are emphasized.

to a high value, more density values along the ray have to be accumulated or a region of high directional variance has to be passed, before visibility values are increased. Increasing λ enables a deeper insight into the data set. $r \in [0,1]$ is the distance of the ray to the center of the brush in screen-space. With this, threshold λ is increased with increasing distance to the center of the brush, leading to a reduced density of lines at the border of the brush. D_{Acc} is the accumulated density along each ray from the camera to the corresponding voxel. The visibility values are increased with accumulated density, to not only show the nearest lines but to allow a placement of visibility values behind a number of lines as can be seen in Fig. 7.6 (left). The impact of this quantity is controlled by parameter p_d. Regions showing a high directional variance *Var* can be explicitly emphasized by increasing parameter p_v, as can be seen in Fig. 7.6 (right).

7.5.2. Brush guidance

By relying solely on the screen-space location of the used brush, the user can easily miss significant structures which are not in the brush volume, yet located very close to the brushed region in 3D space. In order to guide the user towards such structures, we propose a mechanism to automatically continue the selected brush volume anisotropically into their direction. This is achieved by letting visibility values being transported into the surrounding, either along gradient vectors in the CCF or lines passing through the selected cluster as illustrated in Fig. 7.7:

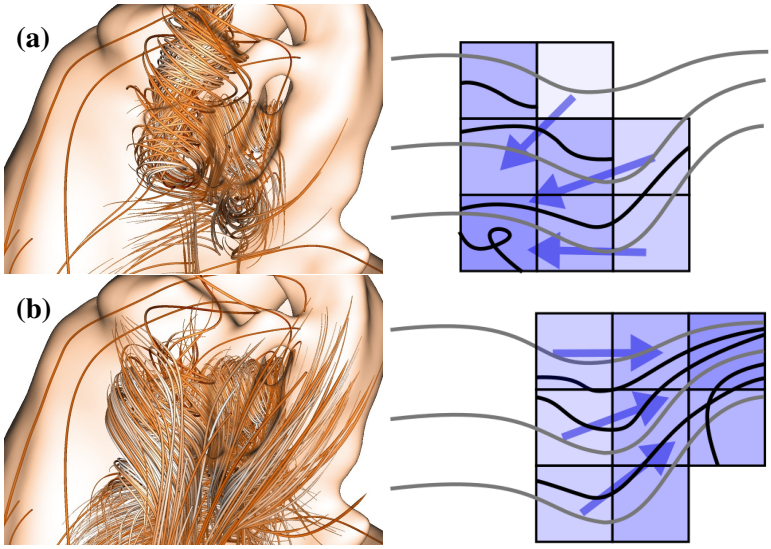

Figure 7.7.: Visibility values (blue) are controlling the line density. (a) Visibility values are transported to areas of higher variation. (b) Visibility values are transported along mean direction.

Mean Direction The visibility values are transported along the mean direction of lines, which is encoded in the mean direction field (see Fig. 7.7 (b)). By continuing the visibility values along the mean direction, crossing lines can be detected and the main path can be identified.

Variability Gradient By using the gradient field of the directional variance field, and transporting the visibility values along the gradient directions, nearby locations with high visibility will be revealed. This effect is shown in Fig. 7.7 (a), where the transport is towards the center of the vortex.

A mixture of both transport mechanisms is possible, by using them in combination but with different priorities. For instance, in the current implementation we use a linear blend between the gradient direction and the mean line direction according to the gradient magnitude. Altogether, visibility values are increased in the brush volume, transported into the dominating direction, and faded out over time. Thus, we can control the distance the visibility values can be continued via a global parameter that controls the life time of continuation.

The transport of visibility values is performed per frame in a compute shader. For each voxel, the visibility value and the vector indicating the location to which this value should be continued is computed. Instead of writing the visibility value to that location, we gather the visibility value from the location indicated by the inverse vector via tri-linear interpolation in the field of current visibility values.

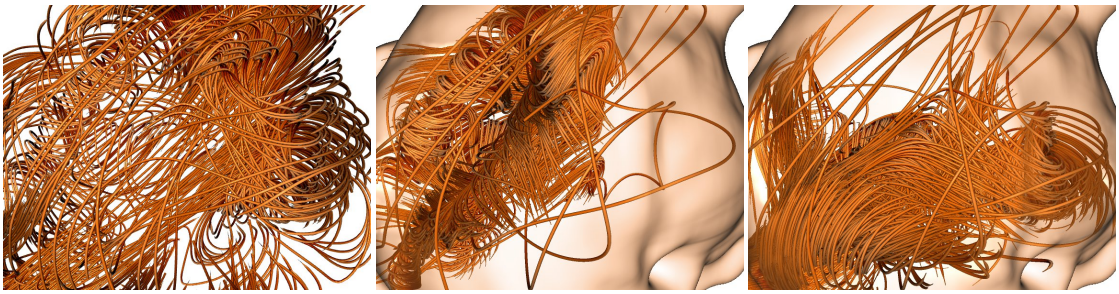

Figure 7.8.: Comparison of importance-based density control as proposed by Kanzler et al. [KFW16] (left) and our brush based visualization (center and right). The vortex at the bottom of the image is not covered by Kanzler et al. but can be brought out using our brush (right).

7.6. Results

In the following, we show the quality of our method based on four data sets. All images were generated on a standard desktop computer (Intel 6x3.50 GHz processor, 32 GB RAM, and an NVIDIA GeForce GTX 1070 Ti graphics card). We have tested our method with following data sets:

Tornado 10000 streamlines were randomly seeded in a 256^3 vector field representing a tornado (Fig. 7.13).

Aneurysm I and II 4070 (Fig. 5.1 (right)) and 9200 streamlines (Fig. 7.11) describing the blood flow in two different aneurysms. [BMC14].

Turbulence 10000 streamlines were randomly seeded in a simulated turbulent vector field of size 1024^3 [AEE*16] (Fig. 7.12).

We compare our approach to importance-based line density control as proposed by Kanzler et al. [KFW16]. The lines of one cluster were extracted and set as input. The curvatures along the lines were mapped to importance values as proposed by the authors. The shading of lines was adjusted to focus on the distribution of line density. In Fig. 7.8 (left), we can see the overall density of lines adapted in a way to have an overview of the data set and to have an unobscured view towards the main vortex located at the left. With our brush-based approach in combination with refinement and transport of visibility values, the vortex at the left can be inspected (Fig. 7.8 (center)). By further inspecting the data set, another vortex at the bottom is revealed, which is not

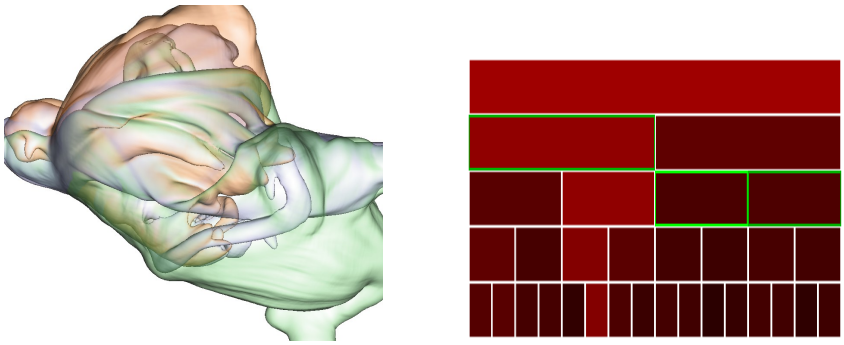

Figure 7.9.: On the left, 3 cluster hulls are visualized, which are selected in the cluster tree panel on the right.

visible when using the importance criterion. The vortex is stretched resulting in low curvature and hence a low importance value. Since our approach works in object-space, 3D flow structures are better preserved than the approach using a screen-space measure to adapt the line density.

The typical overall frame time (22 ms) includes line rendering (3 ms), rendering of cluster hulls using raycasting (13 ms), brushing at the focus region (4 ms), and visibility transport (2 ms) for a visibility volume of size 256^3. In all of our experiments the frame rate was consistently above 20 frames per second.

In the following, we describe a typical workflow using our method for the Aneurysm II data set. At first, the user selects a number of clusters for which the hulls are shown. This provides an overview of the clusters's extends and overlaps. Clusters can then be selected and further split into multiple clusters. A cluster panel represents the first levels of the cluster tree with its root at the top as shown in Fig. 7.9 on the right, and acts as control interface. The user is able to select and deselect different clusters at different levels, while the visual representation of the corresponding cluster hulls is updated immediately. The cluster panel communicates an estimate about the consistency of the cluster by using a mapping to a color gradient from black to red, i.e., from higher to lower consistency. The measure for consistency in this panel is calculated by summing up the distances of all lines of the cluster to the centroid of the cluster. By using this measure, a higher sum of distances indicates a cluster which might be a candidate for splitting up into two distinct clusters. After refining the clusters until a desired level, one can focus on one or two clusters by hiding the other clusters. To investigate one cluster in detail, the brush is applied as can be seen in Fig. 7.10. The vortex cores are now clearly visible while the hull in combination with auxiliary lines allows an estimate about the surroundings of the brushed location. By adapting the radius of the brush, the density of lines is increased in a larger region. By the automated continuation of the brush volume, additional structures become visible.

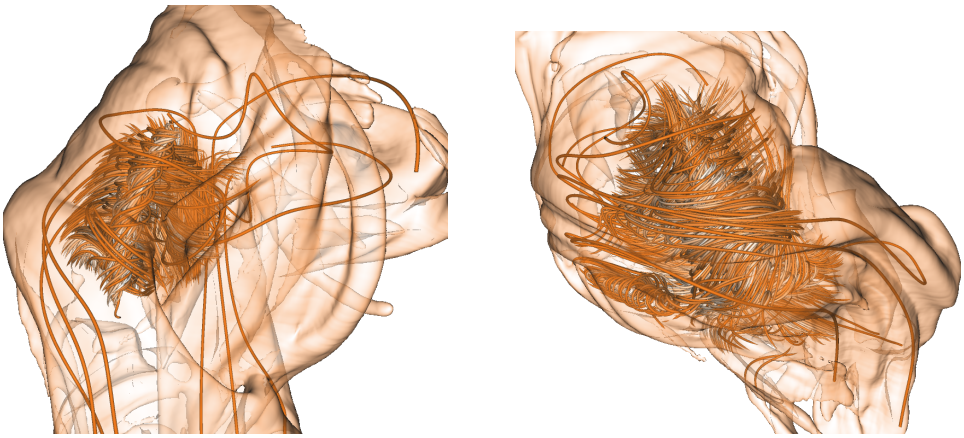

Figure 7.10.: The brush is used to add details by adding lines. Visibility values are placed by using primarily accumulated density and are extended in the direction of higher directional variance.

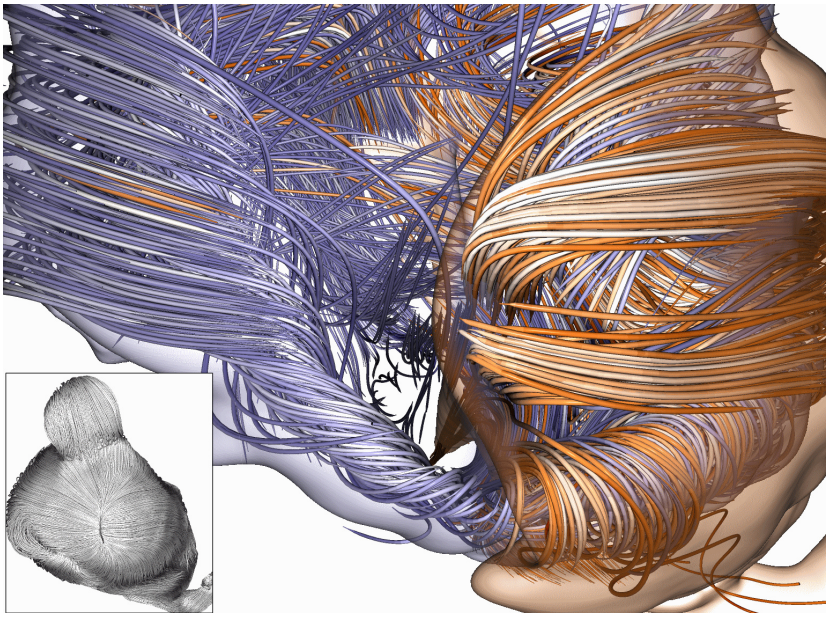

Figure 7.11.: The boundaries of two clusters are inspected by brushing. The vortex core at the bottom is split into two clusters. While the left part is consisting only of lines of the blue cluster, lines of the blue cluster are also interweaved in the orange cluster. Bottom left: All lines of the Aneurysm II data set.

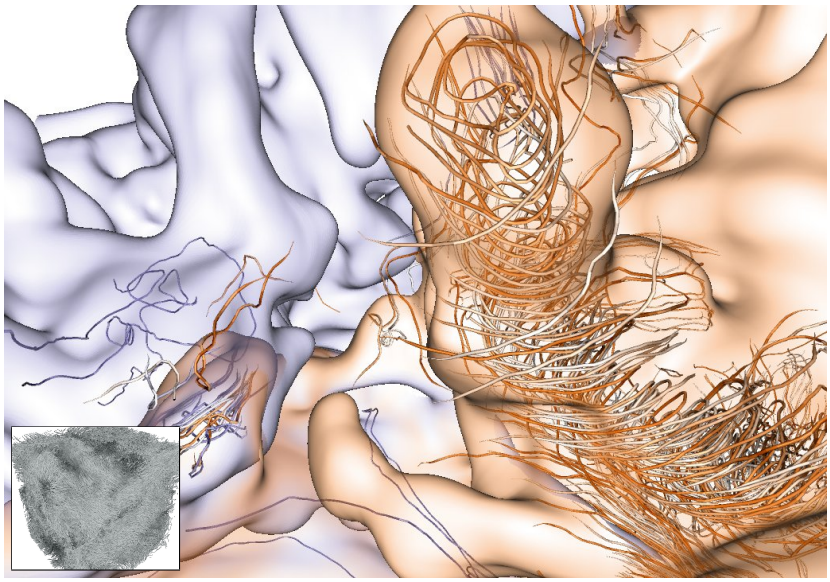

Figure 7.12.: Two clusters of the turbulence data set are inspected. A vortex structure is revealed by placing visibility values at regions of high directional variance. Bottom left: All lines of the Turbulence data set.

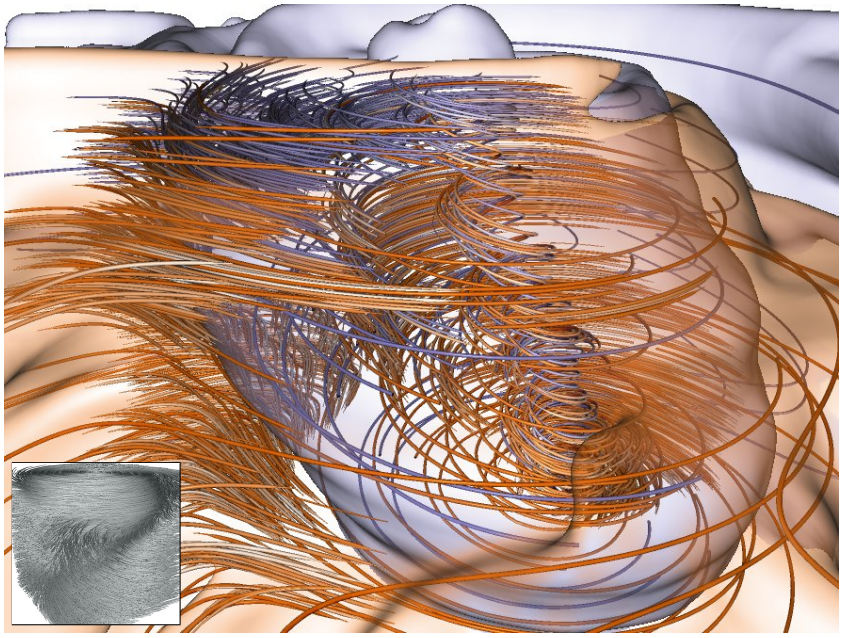

Figure 7.13.: Boundaries of two clusters are inspected by enhancing regions with high accumulated density inside the brush region. Transport of visibility values is showing the core of the tornado. Bottom left: All lines of the Tornado data set.

The Aneurysm II data set consists of long lines, resulting in cluster hulls that overlap in many regions. To further study at which location a cluster shares a feature with a nearby cluster, a second cluster can be enabled. The brush can now be applied to both clusters, revealing information about the underlying clustering. By transporting visibility values along the mean direction field and towards regions of high directional variance, cohesive structures can be identified: Fig. 7.11 shows the boundary between the vortex and the other cluster. In this example lines were not faded out to include the continuation of the lines forming the vortex.

Placing visibility values at regions of high directional variance by increasing parameter p_{var} and extending the selection along the lines, vortices embedded in turbulent structures can be revealed as can be seen in Fig. 7.12. By increasing λ, the coverage of lines located at the background is reduced. Combining this setting with a visibility transport along the directional variance gradient, vortices in front of background lines are still emphasized as can be seen in Fig. 7.13 and Fig. 5.1 (right). The core vortex of the aneurysm is dominated by lines of one cluster while we can see a mixing of clusters at the core of the tornado.

7.7. Conclusion

In this paper, we have presented a novel approach to interactively explore clusters of lines via consistency-guided visualization techniques on the GPU. We have introduced cluster consistency fields and employed them to apply focus+context visualization techniques. We have shown the combination of cluster hulls with interactive and automatic techniques to select focus regions and adapt the line density to cluster consistency. This enables an in-depth comparison of cluster overlap regions and the line consistency of a selected cluster.

In future work, we will compare different clustering methods by matching similar clusters and simultaneously visualizing matches between two clustering methods. Specific characteristics of clustering methods can be studied directly on a given data set. We will further examine time-dependent data sets by finding a transition between brushed regions at different time steps. The development and movement of features is of interest in this case. Guiding a brush by exploiting time-dependent information can further enhance the understanding of the data set.

8

Conclusion and Future Work

Summary Visualizations of lines are a powerful tool to analyze flow fields. In this thesis, we have shown several contributions to improve visualizations of line data sets. In this section, we summarize our presented techniques and their benefits. Finally, we present some ideas for potential future work.

Vector fields with ever increasing resolution require advanced tools to enable analysis. Densely seeded streamlines, which cover the vector fields, however, introduce visual clutter and occlusion if not visualized appropriately. In this thesis, we have introduced a technique that considers markings for important segments along the streamlines to automatically ensure a detailed representation of important structures and an inclusion of an appropriate amount of context structure. To gain the possibility to define the density of lines in screen space, a line hierarchy is determined in a preprocess by using a graph-based perfect matching algorithm. Instead of rendering lines semi-opaque or with a thin diameter to reveal hidden structures, lines classified lower in the hierarchy are removed. The amount of lines that have to be removed for the current view is determined by screen-space measurements, for what reason our approach can optimize the visibility of a large number of lines in interactive framerates. Spacial coherences of lines can be better understood and depth perception is enhanced when illumination effects such as ambient occlusion and shadows are used. We have shown, how a specialized technique enables rendering of semi-opaque lines with advanced illumination effects in interactive framerates for large data sets. This can be accomplished by dividing the initial line geometry in voxel bounded discretized line segments and saving them in a volumetric regular grid. This preprocess enables a ray casting approach to efficiently blend the lines in front-to-back order. Due to our regular grid structure, further attributes such as average density or average line direction can be derived efficiently. Lastly, we have shown how these derived local consistency measurements can be used

to enable interactive exploration of line clusters. By visualizing iso-surfaces of these consistency fields, an overview of the underlying line data set is presented, which serves as a starting point for interactive exploration. A brush-like gesture steers the density of shown lines, enabled by our graph-based line hierarchy. This brush queries calculated consistency fields to emphasize interesting regions along the viewing ray. The shape of the brush also adapts smartly to hint towards nearby structures that could be of interest. This method is especially useful for visually inspecting borders of clusters in detail.

Future Work In future research, we plan to extend our focus-and-context based idea and include mechanics to handle time dependent data sets. One possibility is the use of animations, i.e., in each time step corresponding streamlines are displayed that are similar to the previous time step. Therefore, our line hierarchy has to be extended to ensure temporal coherence. Further, the selection of the brush must also be continuously adapted to the current time step. For this, it is necessary to search for structures in the vector field that are present in the vicinity in the subsequent time step as well, to update the selection accordingly.

The quality of clusters can be measured by various metrics. However, different line data sets and application cases have different requirements, which often can not be represented only by using metrics, but which become evident through visualizations of the cluster members. In order to be able to compare the quality of different line cluster algorithms with regard to the respective application, we will extend our framework for interactive exploration of clusters by an analysis of different cluster algorithms in the future. The intention is to determine common features of the algorithms and to highlight differences by means of distance fields calculated from cluster members. These measurements will be processed in new cluster consistency fields. In combination with these new fields it can be shown where certain line structures are handled differently under different cluster algorithms.

We have shown how global illumination contributes to the spatial understanding of semi-opaque line data sets. Following, we want to investigate to what extent opacity optimization as proposed by Günther et al. [GRT13] can be enhanced by global illumination effects through our voxel structure. An approximation of occlusions could be obtained by casting rays and summing up voxel density values. One difficulty here is that line segments stored in the voxels no longer have a connection to each other. A line can therefore no longer be processed as a whole. For static geometry, however, only the alpha-value for the respective line segment would have to be updated. Thus, an ordinary line data structure could be used in addition to interpolate opacity along the line. Afterwards, the results are transferred to the respective line segments in the voxels.

Bibliography

[AEE*16] Aluie H., Eyink G., E. V., Kanov S. C. K., Burns R., Meneveau C., Szalay A.: Forced MHD turbulence data set. `http://turbulence.pha.jhu.edu/docs/README-MHD.pdf`, 2013 (accessed May 13, 2016). 46, 53, 76, 100

[AKR92] A Kleffner D., Ramachandran V.: On the perception of shape from shading. percept psychophys. *Perception & psychophysics 52* (08 1992), 18–36. 11

[AL09] Aila T., Laine S.: Understanding the efficiency of ray traversal on gpus. In *Proc. High-Performance Graphics 2009* (2009), pp. 145–149. 61

[AMP10] Atev S., Miller G., Papanikolopoulos N. P.: Clustering of vehicle trajectories. *IEEE Transactions on Intelligent Transportation Systems 11*, 3 (Sept 2010), 647–657. `doi:10.1109/TITS.2010.2048101`. 88

[Asi03] Asimov D.: Notes on the topology of vector fields and flows. 5

[AT06] Antonini G., Thiran J. P.: Counting pedestrians in video sequences using trajectory clustering. *IEEE Transactions on Circuits and Systems for Video Technology 16*, 8 (Aug 2006), 1008–1020. `doi:10.1109/TCSVT.2006.879118`. 88

[ATR*08] Annen T., Theisel H., Rössl C., Ziegler G., Seidel H.-P.: Vector field contours. In *Proceedings of Graphics Interface 2008* (Toronto, Ont., Canada, Canada, 2008), GI '08, Canadian Information Processing Society, pp. 97–105. 8

[BBB*18] Behrendt B., Berg P., Beuing O., Preim B., Saalfeld S.: Explorative blood flow visualization using dynamic line filtering based on surface features. 10, 91

[BCL*07] Bavoil L., Callahan S. P., Lefohn A., Comba J. a. L. D., Silva C. T.: Multi-fragment effects on the gpu using the k-buffer. In *Proceedings of the 2007 Symposium on Interactive 3D*

Graphics and Games (New York, NY, USA, 2007), I3D '07, ACM, pp. 97–104. URL: http://doi.acm.org/10.1145/1230100.1230117, doi:10.1145/1230100.1230117. 12

[BCM*08] Berman J. I., Chung S., Mukherjee P., Hess C. P., Han E. T., Henry R. G.: Probabilistic streamline q-ball tractography using the residual bootstrap. *NeuroImage 39*, 1 (2008), 215 – 222. doi:https://doi.org/10.1016/j.neuroimage.2007.08.021. 92

[BCP*12] Brambilla A., Carnecky R., Peikert R., Viola I., Hauser H.: Illustrative flow visualization: State of the art, trends and challenges. pp. 75–94. doi:10.2312/conf/EG2012/stars/075-094. 6

[BFMW12] Bürger K., Fraedrich R., Merhof D., Westermann R.: Instant visitation maps for interactive visualization of uncertain particle trajectories, 2012. URL: https://doi.org/10.1117/12.906872, doi:10.1117/12.906872. 92

[Bli77] Blinn J. F.: Models of light reflection for computer synthesized pictures. *SIGGRAPH Comput. Graph. 11*, 2 (July 1977), 192–198. URL: http://doi.acm.org/10.1145/965141.563893, doi:10.1145/965141.563893. 18

[BMC14] Byrne G., Mut F., Cebral J.: Quantifying the large-scale hemodynamics of intracranial aneurysms. *Amer. J. of Neuroradiology 35* (2014), 333–338. doi:10.3174/ajnr.A3678. 45, 76, 100

[BPMS12] Born S., Pfeifle M., Markl M., Scheuermann G.: Visual 4D MRI blood flow analysis with line predicates. In *IEEE PacificVis Symposium* (2012), pp. 105–112. 39

[BSD08] Bavoil L., Sainz M., Dimitrov R.: Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 Talks* (2008), SIGGRAPH '08, pp. 22:1–22:1. doi:10.1145/1401032.1401061. 75

[CB11] Candelaresi S., Brandenburg A.: Decay of helical and nonhelical magnetic knots. *Phys. Rev. E 84* (2011). doi:10.1103/PhysRevE.84.016406. 45

[CCK07a] Chen Y., Cohen J., Krolik J.: Similarity-guided streamline placement with error evaluation. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1448–1455. doi:10.1109/TVCG.2007.70595. 7, 32, 34, 91

[CCK07b] Chen Y., Cohen J., Krolik J.: Similarity-guided streamline placement with error evaluation. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1448–1455. 39

[CFM*13] Carnecky R., Fuchs R., Mehl S., Jang Y., Peikert R.: Smart transparency for illustrative visualization of complex flow surfaces. *Visualization and Computer Graphics, IEEE Transactions on 19*, 5 (May 2013), 838–851. doi:10.1109/TVCG.2012.159. 12

[CGG04] 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* (April 2004), pp. 344–347 Vol. 1. doi:10.1109/ISBI.2004.1398545. 39, 91

[CNLE09] Crassin C., Neyret F., Lefebvre S., Eisemann E.: Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (2009), I3D '09, ACM, pp. 15–22. doi:10.1145/1507149.1507152. 13, 58, 61

[CNS*11] Crassin C., Neyret F., Sainz M., Green S., Eisemann E.: Interactive indirect illumination using voxel cone tracing: A preview. In *Symposium on Interactive 3D Graphics and Games* (2011), I3D '11, pp. 207–207. doi:10.1145/1944745.1944787. 13, 58, 61, 72

[COK97] Cohen-Or D., Kaufman A.: 3d line voxelization and connectivity control. *IEEE Comput. Graph. Appl. 17*, 6 (1997), 80–87. doi:10.1109/38.626973. 13, 58, 61

[DH92] Danskin J., Hanrahan P.: Fast algorithms for volume ray tracing. In *Proceedings of the 1992 Workshop on Volume Visualization* (1992), pp. 91–98. doi:10.1145/147130.147155. 71

[EBRI09] Everts M. H., Bekker H., Roerdink J. B. T. M., Isenberg T.: Depth-dependent halos: Illustrative rendering of dense line data. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1299–1306. doi:10.1109/TVCG.2009.138. 60

[EBRI15] Everts M. H., Bekker H., Roerdink J. B. T. M., Isenberg T.: Interactive illustrative line styles and line style transfer functions for flow visualization. *CoRR abs/1503.05787* (2015). 12, 16, 60

[Edm65] Edmonds J.: Paths, trees, and flowers. *Canadian Journal of mathematics 17*, 3 (1965), 449–467. 40

[EHS13] Eichelbaum S., Hlawitschka M., Scheuermann G.: Lineao - improved three-dimensional line rendering. *IEEE Transactions on Visualization and Computer Graphics 19*, 3 (Mar. 2013), 433–445. URL: http://dx.doi.org/10.1109/TVCG.2012.142, doi:10.1109/TVCG.2012.142. 12, 18

[ESSL11] Enderton E., Sintorn E., Shirley P., Luebke D.: Stochastic transparency. *IEEE Transactions on Visualization and Computer Graphics 17*, 8 (2011), 1036–1047. doi:10.1109/TVCG.2010.123. 12, 60

[EW] Everitt C., Williams L.: Interactive order-independent transparency. In *White paper available online at http://developer.nvidia.com, 2001*. 12, 60

[FEW*19] Frasson A., Ender M., Weiss S., Kanzler M., Pandrey A., Schumacher J., Westermann R.: Visual exploration of circulation rolls in convective heat flows. pp. 202–211. doi:10.1109/PacificVis.2019.00031. 26

[FG69] Freeman H., Glass J. M.: On the quantization of line-drawing data. *IEEE Transactions on Systems Science and Cybernetics 5*, 1 (1969), 70–79. 63

[FG98] Fuhrmann A., Gröller E.: Real-time techniques for 3d flow visualization. In *Proceedings of the conference on Visualization'98* (1998), IEEE Computer Society Press, pp. 305–312. 10, 91

[GBWT11] 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* (2011), Eurographics Assosciation, p. to appear. 8, 51

[GM05] Gobbetti E., Marton F.: Far voxels: A multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. *ACM Trans. Graph. 24*, 3 (2005), 878–885. doi:10.1145/1073204.1073277. 67

[GNBP11] Gasteiger R., Neugebauer M., Beuing O., Preim B.: The flowlens: A focus-and-context visualization approach for exploration of blood flow in cerebral aneurysms. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (Dec. 2011), 2183–2192. 11, 91

[GRT13] Günther T., Rössl C., Theisel H.: Opacity optimization for 3D line fields. *Proc. ACM SIGGRAPH 32*, 4 (2013), 120. doi:10.1145/2461912.2461930. 9, 10, 32, 34, 38, 48, 50, 51, 60, 90, 106

[GTG17] Günther T., Theisel H., Gross M.: Decoupled opacity optimization for points, lines and surfaces. *Computer Graphics Forum (Proc. Eurographics) 36*, 2 (2017), 153–162. 9

[HGH*10] Hummel M., Garth C., Hamann B., Hagen H., Joy K. I.: Iris: Illustrative rendering for integral surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (Nov 2010), 1319–1328. doi:10.1109/TVCG.2010.173. 94

[HL79] Herman G. T., Liu H. K.: Three-dimensional display of human organs from computed tomograms. *Computer Graphics and Image Processing 9*, 1 (1979), 1–21. doi:10.1016/0146-664X(79)90079-0. 61

[How12] Howard I.: *Perceiving in depth. Volume 3. Other mechanisms of depth perception*. 03 2012. doi:10.1093/acprof:oso/9780199764167.001.0001. 21

[Jai10] Jain A. K.: Data clustering: 50 years beyond k-means. *Pattern Recogn. Lett. 31*, 8 (2010), 651–666. 7, 35, 90

[JCK12] Jackson B., Coffey D., Keefe D. F.: Force Brushes: Progressive Data-Driven Haptic Selection and Filtering for Multi-Variate Flow Visualizations. In *EuroVis - Short Papers* (2012), Meyer M., Weinkaufs T., (Eds.), The Eurographics Association. 10, 91

[JL97] Jobard B., Lefer W.: Creating evenly-spaced streamlines of arbitrary density. In *Proc. Eurographics Workshop on Visualization in Scientific Computing*. 1997, pp. 43–55. 6, 34, 91

[Jon08] Jones D. K.: Tractography gone wild: Probabilistic fibre tracking using the wild bootstrap with diffusion tensor mri. *IEEE Transactions on Medical Imaging 27*, 9 (Sept 2008), 1268–1274. doi:10.1109/TMI.2008.922191. 92

[JSYR14] Jönsson D., Sundén E., Ynnerman A., Ropinski T.: A survey of volumetric illumination techniques for interactive volume rendering. *Computer Graphics Forum 33*, 1 (2014), 27–51. doi:10.1111/cgf.12252. 12, 61

[KCY93] Kaufman A., Cohen D., Yagel R.: Volume graphics. *Computer 26*, 7 (July 1993), 51–64. doi:10.1109/MC.1993.274942. 61

[KFW16] Kanzler M., Ferstl F., Westermann R.: Line density control in screen-space via balanced line hierarchies. *Computers & Graphics 61* (2016), 29 – 39. URL: http://www.sciencedirect.com/science/article/pii/S0097849316300899, doi:https://doi.org/10.1016/j.cag.2016.08.001. 2, 4, 89, 90, 91, 100

[KLG*13] Kuhn A., Lindow N., Günther T., Wiebel A., Theisel H., Hege H.-C.: Trajectory density projection for vector field visualization. In *EuroVis - Short Papers 2013* (2013), pp. 31–35. 12, 60

[KRW19] Kanzler M., Rautenhaus M., Westermann R.: A voxel-based rendering pipeline for large 3d line sets. *IEEE Transactions on Visualization and Computer Graphics 25*, 7 (July 2019), 2378–2391. URL: https://doi.org/10.1109/TVCG.2018.2834372, doi:10.1109/TVCG.2018.2834372. 2, 4

[KSW06] Krüger J., Schneider J., Westermann R.: Clearview: An interactive context preserving hotspot visualization technique. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 941–948. doi:10.1109/TVCG.2006.124. 10, 34

[KW18] Kanzler M., Westermann R.: Interactive visual exploration of line clusters. In *Proceedings of the Conference on Vision, Modeling, and Visualization* (2018), EG VMV '18, Eurographics Association, pp. 155–163. URL: https://doi.org/10.2312/vmv.20181265, doi:10.2312/vmv.20181265. 2, 4

[LB00] Langer M., Bülthoff H.: Depth discrimination from shading under diffuse lighting. *Perception 29* (02 2000), 649–60. doi:10.1068/p3060. 11

[LCD06] Luft T., Colditz C., Deussen O.: Image enhancement by unsharp masking the depth buffer. In *ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), SIGGRAPH '06, ACM, pp. 1206–1213. URL: http://doi.acm.org/10.1145/1179352.1142016, doi:10.1145/1179352.1142016. 12

[lem16] Lemon graph library. http://lemon.cs.elte.hu/trac/lemon, 2004 (accessed February 29, 2016). 40

[LHD*04] Laramee R. S., Hauser H., Doleisch H., Vrolijk B., Post F. H., Weiskopf D.: The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum 23*, 2 (2004), 203–221. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2004.00753.x, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2004.00753.x, doi:10.1111/j.1467-8659.2004.00753.x. 5

[LHLW09] Liu F., Huang M.-C., Liu X.-H., Wu E.-H.: Efficient depth peeling via bucket sort. In *Proceedings of the Conference on High Performance Graphics 2009* (2009), HPG '09, pp. 51–57. doi:10.1145/1572769.1572779. 60

[LHS08] Li L., Hsieh H.-H., Shen H.-W.: Illustrative streamline placement and visualization. In *Proc. IEEE PacificVis* (2008), pp. 79–86. doi:10.1109/PACIFICVIS.2008.4475462. 6, 34, 91

[LK11] Laine S., Karras T.: Efficient sparse voxel octrees. *IEEE Transactions on Visualization and Computer Graphics 17*, 8 (2011), 1048–1059. doi:10.1109/TVCG.2010.240. 13, 58, 61

[LMG06] Liu Z., Moorhead R., Groner J.: An advanced evenly-spaced streamline placement algorithm. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 965–972. doi:10.1109/TVCG.2006.116. 6, 34, 91

[LMSC11] Lee T.-Y., Mishchenko O., Shen H.-W., Crawfis R.: View point evaluation and streamline filtering for flow visualization. In *Proc. IEEE PacificVis* (2011), pp. 83–90. doi:10.1109/PACIFICVIS.2011.5742376. 8, 34, 90

[LR11] Lindemann F., Ropinski T.: About the influence of illumination models on image comprehension in direct volume rendering. *IEEE transactions on visualization and computer graphics 17* (12 2011), 1922–31. doi:10.1109/TVCG.2011.161. 11

[LS07] Li L., Shen H.-W.: Image-based streamline generation and rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (2007), 630–640. doi:10.1109/TVCG.2007.1009. 8, 34

[MAD05] Mebarki A., Alliez P., Devillers O.: Farthest point seeding for efficient placement of streamlines. In *Proc. IEEE Visualization* (2005), pp. 479–486. doi:10.1109/VISUAL.2005.1532832. 6, 34

[MC14] Mishchenko O., Crawfis R.: On perception of semi-transparent streamlines for three-dimensional flow visualization. *Computer Graphics Forum 33*, 1 (2014), 210–221. doi:10.1111/cgf.12268. 12, 60

[MCHM10] Marchesin S., Chen C.-K., Ho C., Ma K.-L.: View-dependent streamlines for 3D vector fields. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1578–1586. doi:10.1109/TVCG.2010.212. 8, 32, 34, 38, 51, 90

[MHHI98] Mao X., Hatanaka Y., Higashida H., Imamiya A.: Image-guided streamline placement on curvilinear grid surfaces. In *Proc. IEEE Visualization* (1998), pp. 135–142. 6, 34, 90

[MJL*12] 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 19*, 8 (2012), 1342–1353. doi:10.1109/TVCG.2012.150. 7, 34, 39, 91

[MKKP18] Münstermann C., Krumpen S., Klein R., Peters C.: Moment-based order-independent transparency. *Proc. ACM Comput. Graph. Interact. Tech. 1*, 1 (July 2018), 7:1–7:20. URL: http://doi.acm.org/10.1145/3203206, doi:10.1145/3203206. 12

[MLP*10] McLoughlin T., Laramee R. S., Peikert R., Post F. H., Chen M.: Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum 29*, 6 (2010), 1807–1829. doi:10.1111/j.1467-8659.2010.01650.x. 32

[MPSS05] Mallo O., Peikert R., Sigg C., Sadlo F.: Illuminated lines revisited. In *VIS 05. IEEE Visualization, 2005.* (2005), pp. 19–26. doi:10.1109/VISUAL.2005.1532772. 60

[MTHG03] 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, 2003), ACM, pp. 213–222. doi:10.1145/984952.984987. 6, 10, 34, 90

[MVvW05] Moberts B., Vilanova A., van Wijk J.: Evaluation of fiber clustering methods for diffusion tensor imaging. In *Proc. IEEE Visualization* (Oct 2005), pp. 65–72. doi:10.1109/VISUAL.2005.1532779. 7, 35, 41, 88, 90

[MWS13] Ma J., Wang C., Shene C.-K.: Coherent view-dependent streamline selection for importance-driven flow visualization. *Proc. SPIE 8654, Visualization and Data Analysis* (2013). doi:10.1117/12.2001887. 8, 32, 34, 38, 90

[OLK*14] 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 20*, 5 (2014), 686–701. doi:10.1109/TVCG.2013.2297914. 7, 35, 89, 90

[PBD*10] Parker S. G., Bigler J., Dietrich A., Friedrich H., Hoberock J., Luebke D., McAllister D., McGuire M., Morley K., Robison A., Stich M.: Optix: a general purpose ray tracing engine. *ACM Trans. Graph. 29*, 4 (2010), 66:1–66:13. 61, 80

[Pho75] Phong B. T.: Illumination for computer generated pictures. *Commun. ACM 18*, 6 (June 1975), 311–317. URL: http://doi.acm.org/10.1145/360825.360839, doi:10.1145/360825.360839. 17

[PPF*11] Pobitzer A., Peikert R., Fuchs R., Schindler B., Kuhn A., Theisel H., Matkovic K., Hauser H.: The state of the art in topology-based visualization of unsteady flow. *Comput. Graph. Forum 30* (2011), 1789–1811. 5

[PVH*02] Post F., Vrolijk B., Hauser H., Laramee R., Doleisch H.: Feature extraction and visualisation of flow fields. *Proceedings of the Eurographics Conference 2002* (08 2002). 5

[PYH*06] Park S. W., Yu H., Hotz I., Kreylos O., Linsen L., Hamann B.: Structure-accentuating dense flow visualization. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization* (2006), pp. 163–170. doi:10.2312/VisSym/EuroVis06/163-170. 60

[RCBW] Reichl F., Chajdas M. G., Bürger K., Westermann R.: Hybrid Sample-based Surface Rendering. In *VMV 2012 - Vision, Modeling and Visualization*, pp. 47–54. doi:10.2312/PE/VMV/VMV12/047-054. 13, 61

[RGG19] Rojo I. B., Gross M., Günther T.: Fourier opacity optimization for scalable exploration. *IEEE Transactions on Visualization and Computer Graphics* (2019). 9

[RH12] Royer M. P., Houser K. W.: Spatial brightness perception of trichromatic stimuli. *LEUKOS 9*, 2 (2012), 89–108. doi:10.1582/LEUKOS.2012.09.02.002. 20

[RPP*09] Rosanwo O., Petz C., Prohaska S., Hege H.-C., Hotz I.: Dual streamline seeding. *Proc. IEEE PacificVis 0* (2009), 9–16. 6, 34, 91

[RT12] Rössl C., Theisel H.: Streamline embedding for 3D vector field exploration. *IEEE Transactions on Visualization and Computer Graphics 18*, 3 (2012), 407–420. 39

[SFD09] Stich M., Friedrich H., Dietrich A.: Spatial splits in bounding volume hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG '09, ACM, pp. 7–13. URL: http://doi.acm.org/10.1145/1572769.1572771, doi:10.1145/1572769.1572771. 80

[SGG15] Staib J., Grottel S., Gumhold S.: Visualization of particle-based data with transparency and ambient occlusion. *Computer Graphics Forum 34*, 3 (2015), 151–160. doi:10.1111/cgf.12627. 61

[SGS05] Stoll C., Gumhold S., Seidel H.-P.: Visualization with stylized line primitives. In *Visualization, 2005. VIS 05. IEEE* (2005), IEEE, pp. 695–702. 52, 60

[SHH*07] 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), pp. 227–234. 7, 34, 90

[SLCZ09] Spencer B., Laramee R. S., Chen G., Zhang E.: Evenly spaced streamlines for surfaces: An image-based approach. *Computer Graphics Forum 28*, 6 (2009), 1618–1631. 6, 34, 91

[SM04] Schussman G., Ma K.-L.: Anisotropic volume rendering for extremely dense, thin line data. In *Proceedings of the Conference on Visualization '04* (2004), VIS '04, pp. 107–114. doi:10.1109/VISUAL.2004.5. 13, 61

[SMDZ14] Stauffer R., Mayr G., Dabernig M., Zeileis A.: Somewhere over the rainbow: How to make effective use of colors in meteorological visualizations. *Bulletin of the American Meteorological Society 96* (07 2014), 140710055335002. doi:10.1175/BAMS-D-13-00155.1. 20

[SS06] Salzbrunn T., Scheuermann G.: Streamline predicates. *IEEE Transactions on Visualization and Computer Graphics 12*, 6 (2006), 1601–1612. 39

[SV14] Salvi M., Vaidyanathan K.: Multi-layer alpha blending. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2014), I3D '14, ACM, pp. 151–158. URL: http://doi.acm.org/10.1145/2556700.2556705, doi:10.1145/2556700.2556705. 12

[SWJS08] Salzbrunn T., Wischgoll T., Jänicke H., Scheuermann G.: The state of the art in flow visualization: Partition-based techniques. In *In Simulation and Visualization 2008 Proceedings* (2008). 5

[TB96] Turk G., Banks D.: Image-guided streamline placement. In *Proc. ACM SIGGRAPH* (1996), pp. 453–460. doi:10.1145/237170.237285. 6, 34, 90

[TCSW15] Tong X., Chen C.-M., Shen H.-W., Wong P. C.: Interactive streamline exploration and manipulation using deformation. In *IEEE PacificVis* (2015), pp. 1–8. doi:10.1109/PACIFICVIS.2015.7156349. 10, 34

[TG10] Thibieroz N., Gruen H.: Oit and indirect illumination using dx11 linked lists. In *Game Developers Conference*. 2010. 12, 58, 60

[THGM11] Thiedemann S., Henrich N., Grosch T., Müller S.: Voxel-based global illumination. In *Symposium on Interactive 3D Graphics and Games* (2011), I3D '11, pp. 103–110. doi:10.1145/1944745.1944763. 13, 58, 61

[VKG04] Viola I., Kanitsar A., Gröller M. E.: Importance-driven volume rendering. In *Proc. IEEE Visualization* (2004), pp. 139–145. 10, 33, 34

[VKP00] Verma V., Kao D., Pang A.: A flow-guided streamline seeding strategy. In *Proc. IEEE Visualization* (2000), pp. 163–170. 6, 32, 34, 91

[Wan92] Wanger L.: The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics* (New York, NY, USA, 1992), I3D '92, ACM, pp. 39–42. URL: http://doi.acm.org/10.1145/147156.147161, doi:10.1145/147156.147161. 11

[WE05] Weiskopf D., Erlebacher G.: Overview of flow visualization. *The Visualization Handbook* (2005), 261–278. 32

[WFG92] Wanger L. R., Ferwerda J. A., Greenberg D. P.: Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications 12*, 3 (May 1992), 44–58. doi:10.1109/38.135913. 11

[WIK*06] Wald I., Ize T., Kensler A., Knoll A., Parker S. G.: Ray tracing animated scenes using coherent grid traversal. *ACM Transactions on Graphics* (2006), 485–493. 13, 61

[WJA*17] Wald I., Johnson G., Amstutz J., Brownlee C., Knoll A., Jeffers J., Gunther J., Navratil P.: Ospray - a cpu ray tracing framework for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2017), 931–940. doi:10.1109/TVCG.2016.2599041. 13, 61

[WKJ*15] Wald I., Knoll A., Johnson G. P., Usher W., Pascucci V., Papka M. E.: Cpu ray tracing large particle data with balanced p-k-d trees. In *IEEE SciVis* (2015). 13, 61

[WMS06] Woop S., Marmitt G., Slusallek P.: B-KD Trees for hardware accelerated ray tracing of dynamic scenes. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (New York, NY, USA, 2006), ACM, pp. 67–77. 13, 61

[XLS10] Xu L., Lee T.-Y., Shen H.-W.: An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1216–1224. doi:10.1109/TVCG.2010.131. 7, 34, 91, 94

[YEII12] Yu L., Efstathiou K., Isenberg P., Isenberg T.: Efficient structure-aware selection techniques for 3D point cloud visualizations with 2DOF input. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (Dec. 2012), 2245–2254. 10, 91

[YHGT10] Yang J. C., Hensley J., Grün H., Thibieroz N.: Real-time concurrent linked list construction on the gpu. In *Proceedings of the 21st Eurographics Conference on Rendering* (2010), EGSR'10, pp. 1297–1304. doi:10.1111/j.1467-8659.2010.01725.x. 12, 60

[YKP05] Ye X., Kao D., Pang A.: Strategy for seeding 3D streamlines. In *Proc. IEEE Visualization 2005* (2005), pp. 471–478. doi:10.1109/VISUAL.2005.1532831. 6, 32, 34, 91

[YTvdW*02] Yu Y. H., Tung C., van der Wall B., Pausder H.-J., Burley C., Brooks T., Beaumier P., Delrieux Y., Mercker E., Pengel K.: 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). 45

[YWSC12] Yu H., Wang C., Shene C.-K., Chen J. H.: Hierarchical streamline bundles. *IEEE Transactions on Visualization and Computer Graphics 18*, 8 (2012), 1353–1367. doi:10.1109/TVCG.2011.155. 7, 32, 34, 35, 39, 88, 89, 90

[ZHT06] Zhang Z., Huang K., Tan T.: Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. In *Proc. International Conference on Pattern Recognition* (2006), vol. 3, pp. 1135–1138. doi:10.1109/ICPR.2006.392. 7, 35, 90

[ZSH96] Zöckler M., Stalling D., Hege H.-C.: Interactive visualization of 3d-vector fields using illuminated stream lines. In *Proceedings of the 7th Conference on Visualization '96* (1996), VIS '96, pp. 107–ff. 60

Appendix

A

Line Density Control in Screen-Space via Balanced Line Hierarchies

Mathias Kanzler, Florian Ferstl, Rüdiger Westermann:
Line Density Control in Screen-Space via Balanced Line Hierarchies.
Computers & Graphics, 61:29-39, 2016.
doi:10.1016/j.cag.2016.08.001

Computers & Graphics 61 (2016) 29–39

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Line density control in screen-space via balanced line hierarchies ☆

Mathias Kanzler *, Florian Ferstl, Rüdiger Westermann

Computer Graphics and Visualization Group, Technische Universität München, Boltzmannstr. 3, 85748 Garching bei München, Germany

ARTICLE INFO

Article history:
Received 1 March 2016
Received in revised form
4 July 2016
Accepted 1 August 2016
Available online 23 September 2016

Keywords:
Scientific visualization
Flow visualization
Line fields
Focus + Context
Line hierarchy

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.

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 [3–6].

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

☆This article was recommended for publication by Stefan Brukner

* Corresponding author.
E-mail addresses: kanzler@in.tum.de (M. Kanzler), ferstlf@in.tum.de (F. Ferstl), westermann@tum.de (R. Westermann).

http://dx.doi.org/10.1016/j.cag.2016.08.001
0097-8493/

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 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 [11–14], the line distribution [15–21], 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 the 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.

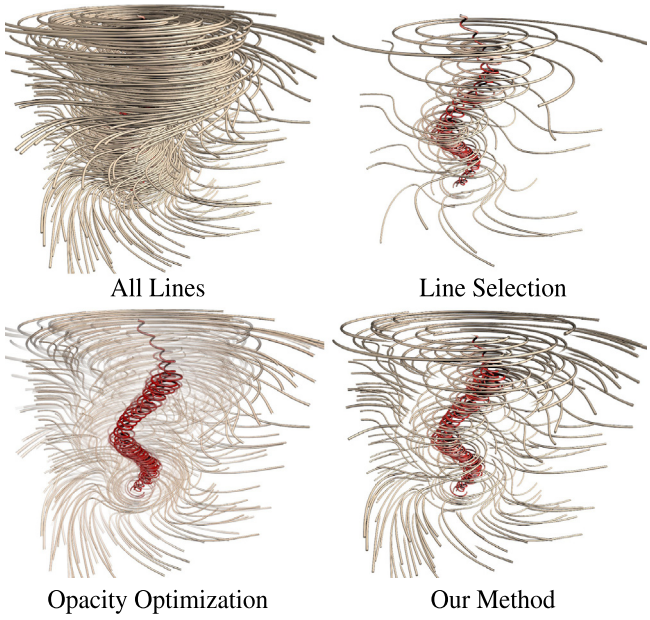

Fig. 2. Different approaches to reveal the focus region in the Tornado dataset.

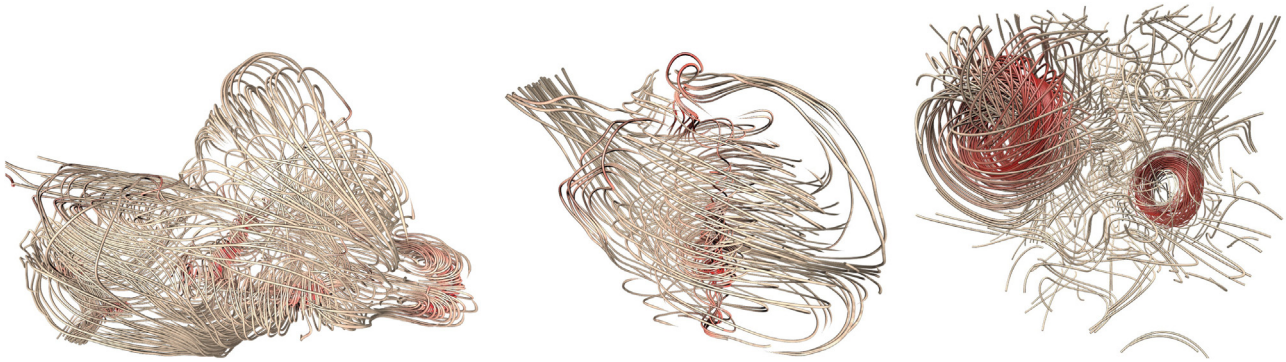

Fig. 1. Visualization of flow fields (Aneurysm I/II, Rings) using line density control. Line color ranges from red (high importance) to light brown (low importance). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

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 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 the 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 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 composed 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.

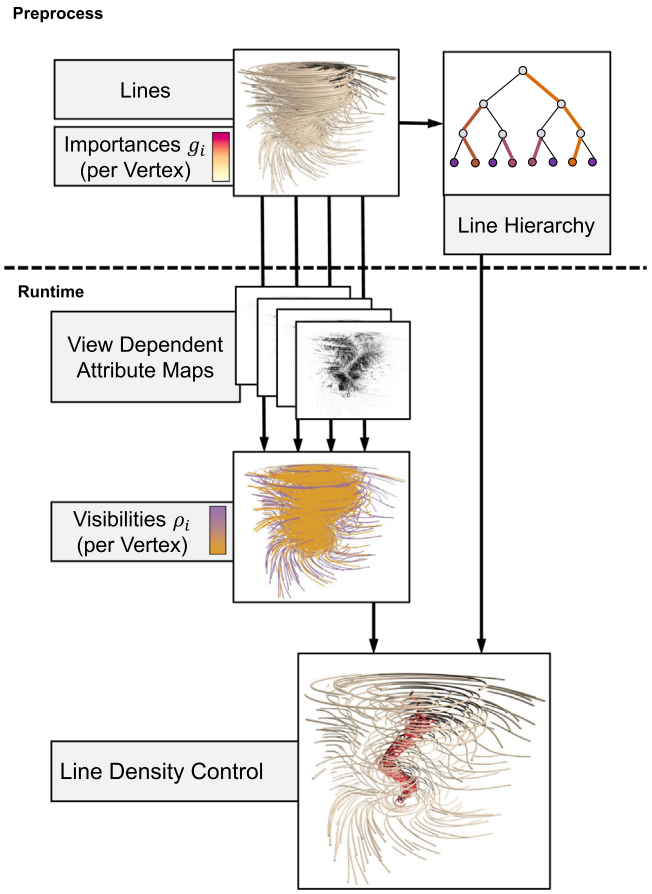

Fig. 3. Overview of view-dependent importance driven line rendering.

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 Section 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 Section 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 Section 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 Section 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 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]:

$$d_M(L_k, L_l) = \text{mean}(d_m(L_k, L_l), d_m(L_l, L_k)),$$
$$\text{with } d_m(L_k, L_l) = \underset{v_i \in L_k}{\text{mean}} \min_{v_j \in L_l} \| v_i - v_j \|. \quad (1)$$

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.

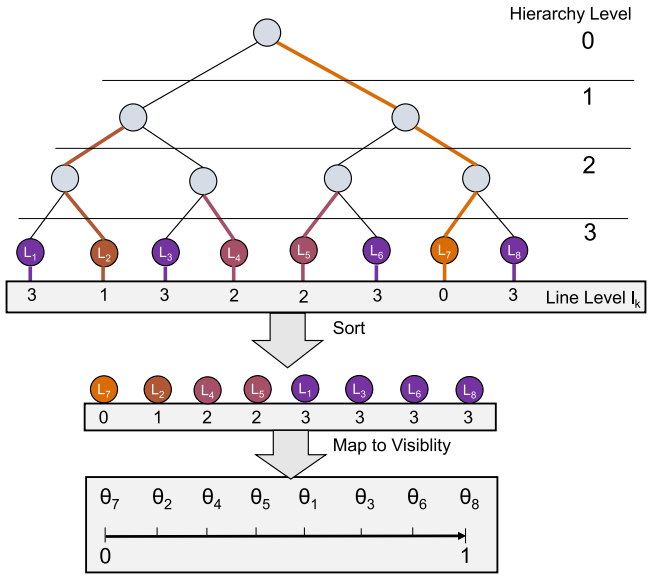

Fig. 4. Construction of the line set hierarchy by pairwise node merging. Lines L_i are represented by the leaf nodes, bold edges and color indicate the selection of representative lines at each level. Nodes are finally linearized and sorted according to the coarsest level where they are representative, and the sorting order is mapped to visibility. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

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 does not indicate good scalability of the construction algorithm, we demonstrate in Section 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

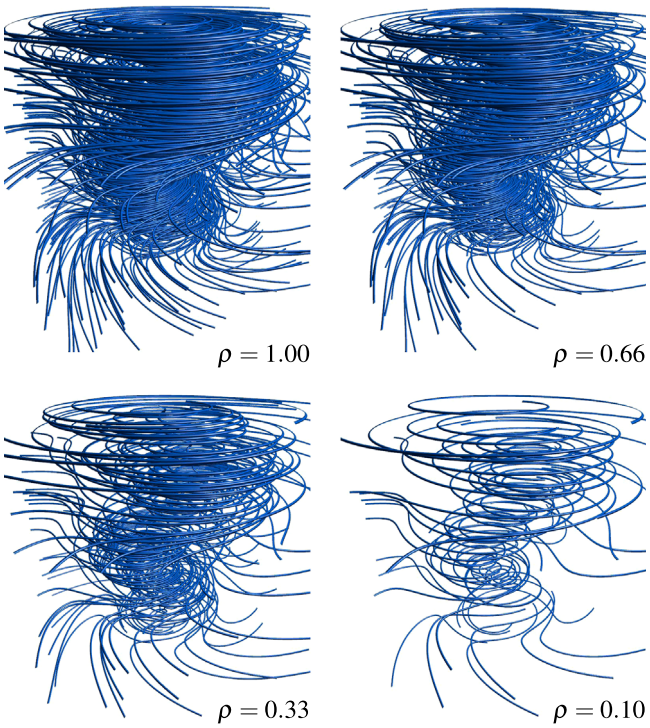

Fig. 5. Line density control in the Tornado dataset using a balanced line hierarchy. Lines are thinned out uniformly according to the global visibility ρ.

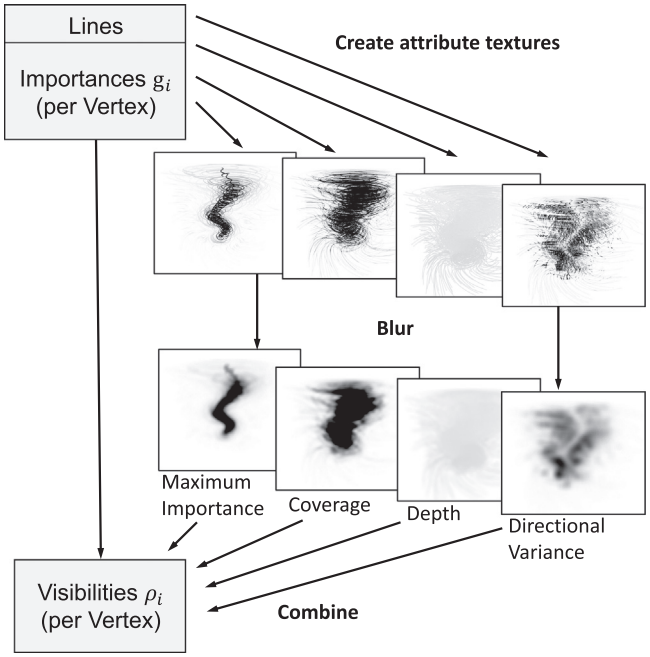

Fig. 6. Visibility computation using per-pixel attributes. Four textures are generated by rasterizing different line attributes, the textures are blurred, and they are then accessed by the vertices (by sampling at their projected screen coordinates) to calculate per-vertex visibilities.

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.

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 blurring 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 $\mathbf{d}_1, \ldots, \mathbf{d}_n$, the

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.

Dataset	Lines (vertices per line)	Precomputation		Visibility computation				Every frame	Sum (ms)
		Similarity (s)	Hierarchy (s)	Projections[a] (ms)	Blur (ms)	Visibility (ms)	Filter (ms)	Final rendering (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	80,000 (220)	6923	25,388[b]	72	0.27	6.00	14.1	51.3	143.67

[a] 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.
[b] Edges in the graph with no influence on the result were removed in a preprocess.

circular variance is defined as $\sigma(\mathbf{d}_1,\ldots,\mathbf{d}_n) = 1 - \|\frac{1}{n}\sum_i \mathbf{d}_i\|$. 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:

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

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*: 80,000 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 *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 straightforward 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 Section 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

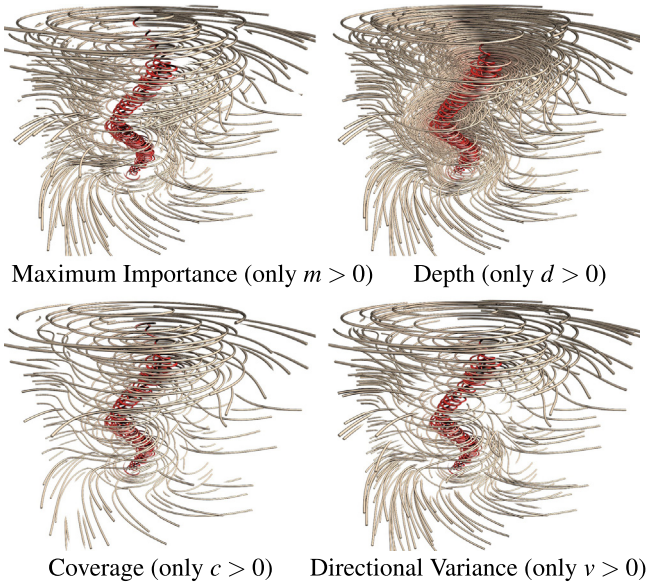

Fig. 7. Line density control in the Tornado datasets via per-pixel maximum importance, depth, coverage, and directional variance. Only the corresponding weight in Eq. (2) is set to a positive value. All other weights are set to 0.

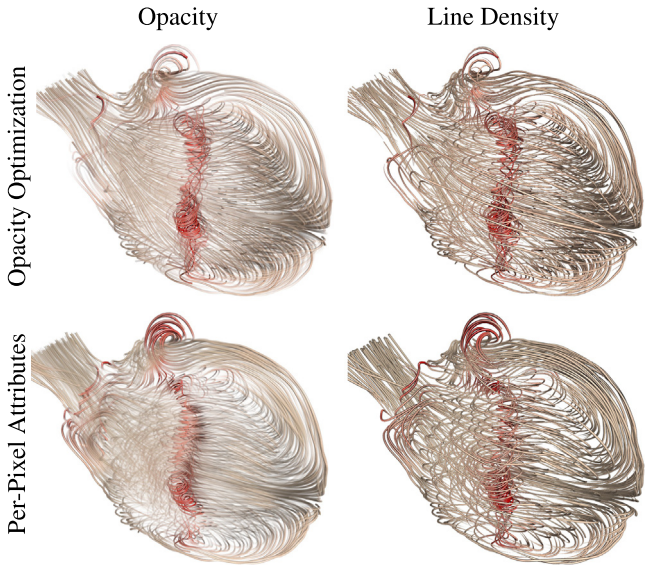

Fig. 8. Visualizations of the Aneurysm I dataset. Top left: opacity optimization + line opacity as by Günther et al. [9]. Top right: opacity optimization [9] + line thinning as by our approach. Bottom left: visibility attributes as by our approach and line opacity [9]. Bottom right: visibility attributes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

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.

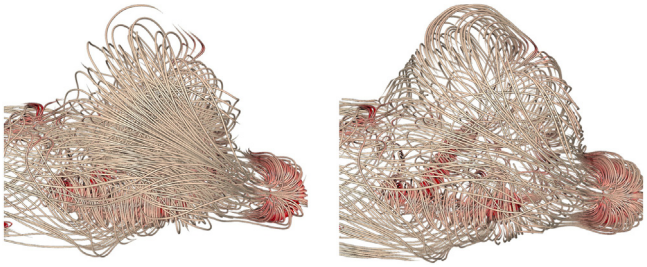

Fig. 9. Effect of using the directional variance in the Aneurysm II dataset. Left: the directional variance is not used. Right: when the directional variance is used, the density of lines is reduced to emphasize the variation of the line directions along the view rays.

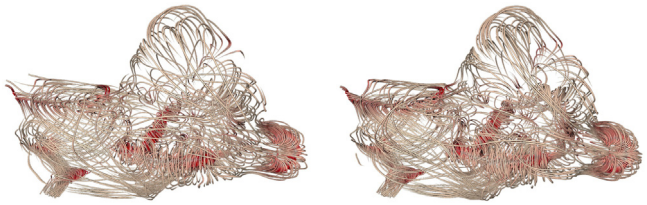

Fig. 10. Different choices of cluster representatives at the inner nodes of the line hierarchy for the Aneurysm II dataset. Left: the most similar line is selected. Right: the most important line is selected.

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.

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.

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

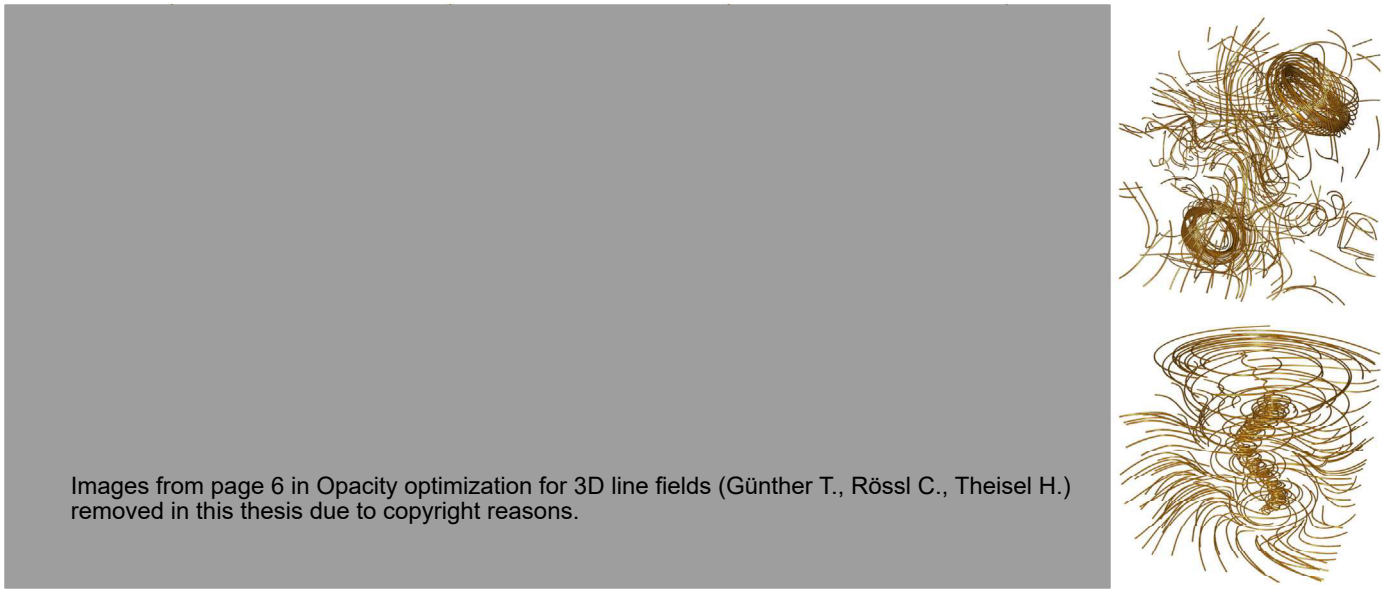

Fig. 11. Rings (top) and Tornado (bottom) – left to right: all lines, Marchesin et al. [7], Günther et al. [44], Günther et al. [9], and our approach.

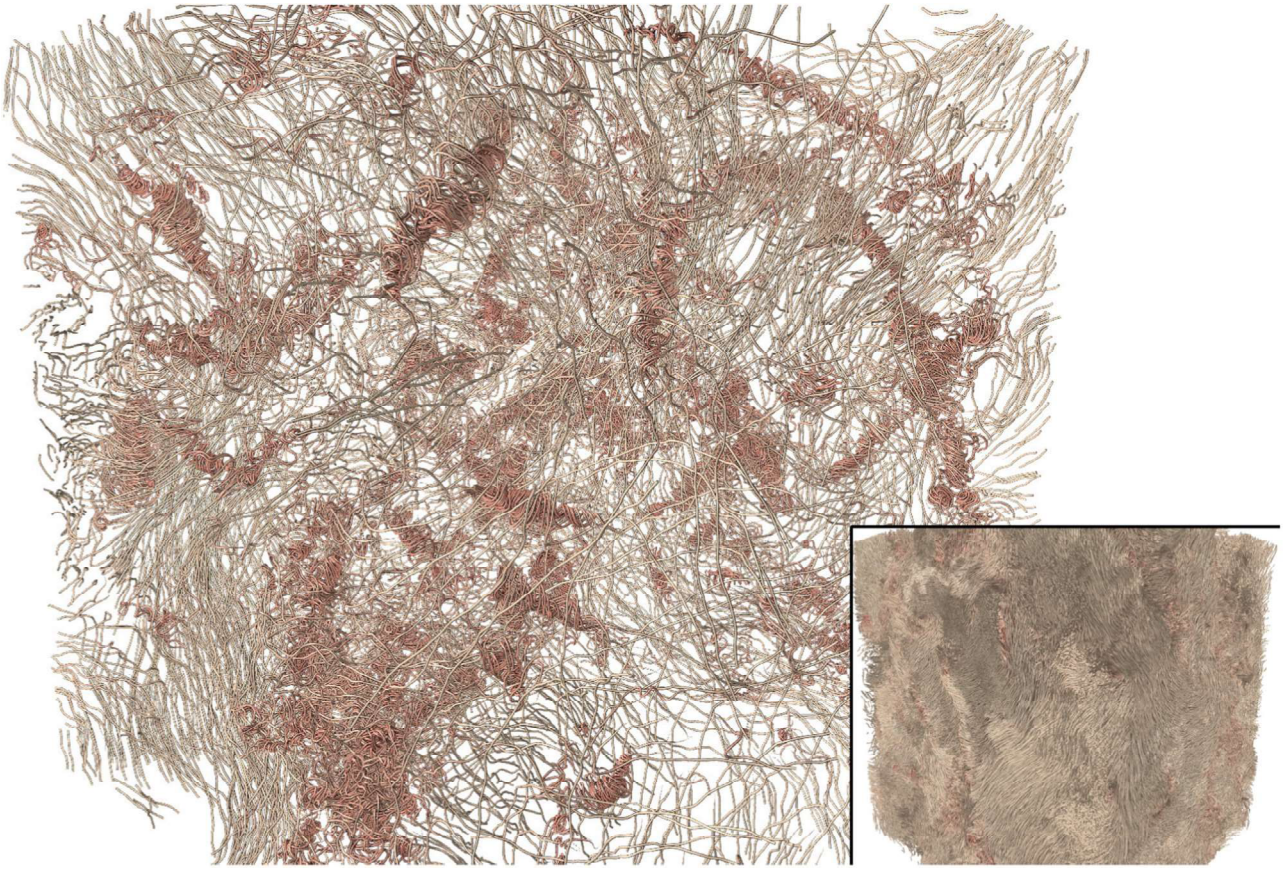

Fig. 12. Controlling the density of 80,000 lines in the Turbulence dataset [43] via our approach (left). Complete line set (right).

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. In Fig 12, we show the visual result of our approach using an input of 80,000 lines.

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

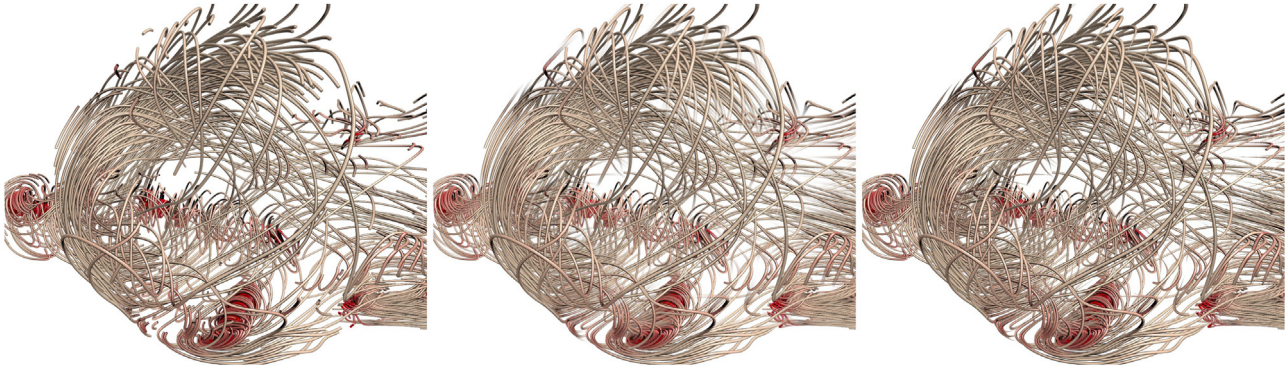

Fig. 13. Visualization of the Aneurysm II dataset using different rendering styles for line endings at cut-offs resulting from partial line removal. (left) Cropping lines without transition, (middle) short transparent line ends and (right) reduced diameter at line ends—our preferred choice.

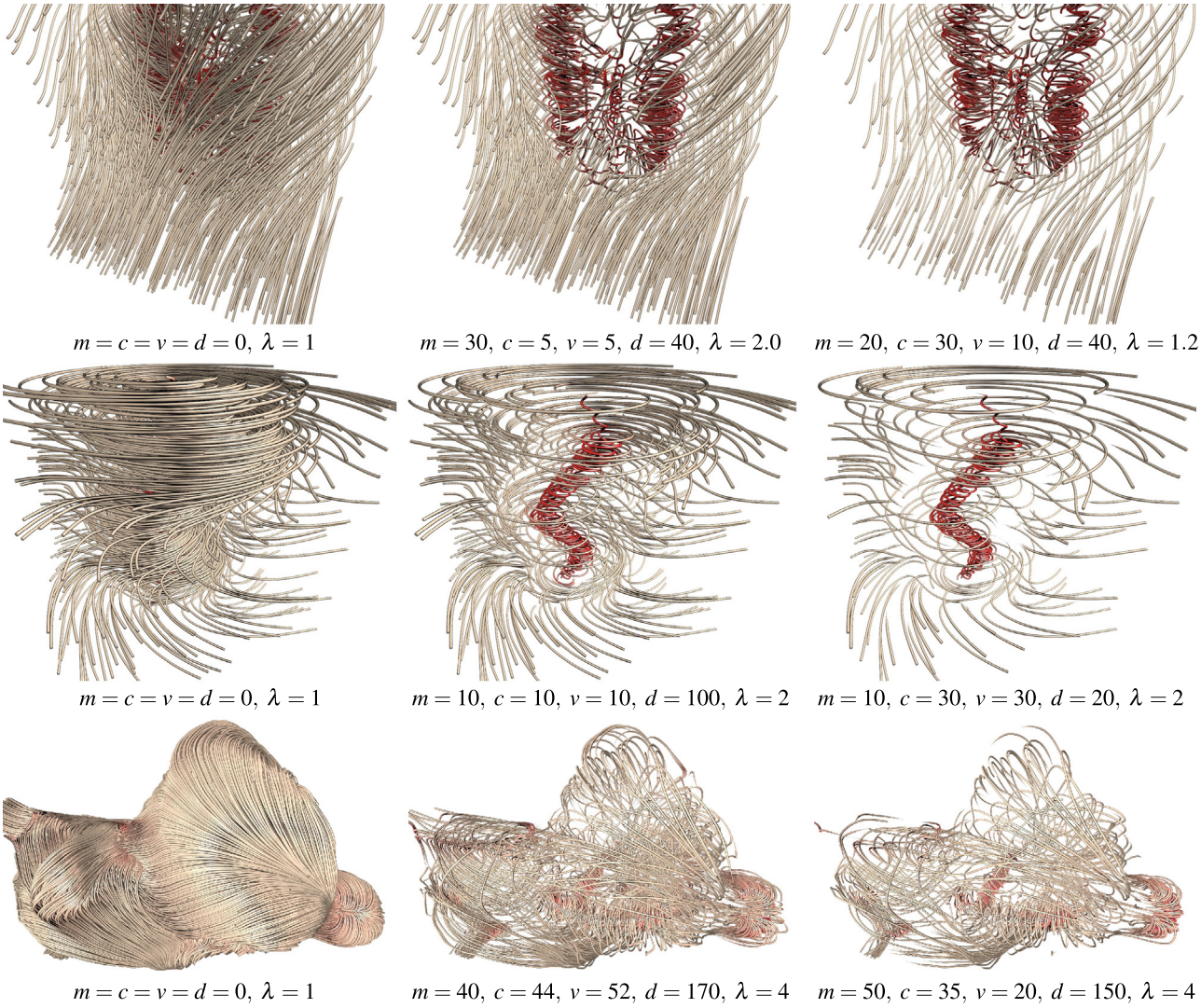

Fig. 14. Demonstration of different parameter combinations for the visibility computation in Eq. (2). From top to bottom: Heli, Tornado, Aneurysm II.

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

We have presented an interactive approach for the visualization of 3D flow fields using integral lines like stream- and pathlines. Our approach combines an efficient local computation of per-pixel visibility attributes with a balanced line-set hierarchy to adaptively control the line density depending on the importance of lines and their occlusions. We have described the construction of such a hierarchy and its use for automatic density control. Our approach discards line segments rather than entire lines, yet it uses line density control to keep important contextual cues even where lines are classified as unimportant. A limitation of our approach is the time-consuming pre-process that is required to construct the balanced line hierarchy, which prohibits the seeding of new lines at runtime.

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 analyze 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.

Acknowledegments

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.

Appendix A. Supplementary material

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.jlumin.2016.09.056.

References

[1] Weiskopf D, Erlebacher G. Overview of flow visualization. In: The visualization handbook; 2005. p. 261–78.
[2] McLoughlin T, Laramee RS, Peikert R, Post FH, Chen M. Over two decades of integration-based, geometric flow visualization. Comput Graph Forum 2010;29(6):1807–29. http://dx.doi.org/10.1111/j.1467-8659.2010.01650.x.
[3] Verma V, Kao D, Pang A. A flow-guided streamline seeding strategy. In: Proceedings of IEEE visualization; 2000. p. 163–70. ISBN 1-58113-309-X.
[4] Ye X, Kao D, Pang A. Strategy for seeding 3D streamlines. In: Proceedings of IEEE visualization 2005; 2005. p. 471–8. http://dx.doi.org/10.1109/VISUAL.2005.1532831.
[5] Chen Y, Cohen J, Krolik J. Similarity-guided streamline placement with error evaluation. IEEE Trans Vis Comput Graph 2007;13(6):1448–55. http://dx.doi.org/10.1109/TVCG.2007.70595.
[6] Yu H, Wang C, Shene CK, Chen JH. Hierarchical streamline bundles. IEEE Trans Vis Comput Graph 2012;18(8):1353–67. http://dx.doi.org/10.1109/TVCG.2011.155.
[7] Marchesin S, Chen CK, Ho C, Ma KL. View-dependent streamlines for 3D vector fields. IEEE Trans Vis Comput Graph 2010;16(6):1578–86. http://dx.doi.org/10.1109/TVCG.2010.212.
[8] Ma J, Wang C, Shene CK. Coherent view-dependent streamline selection for importance-driven flow visualization. In: Proceedings of SPIE conference on visualization and data analysis, vol. 8654; 2013 http://dx.doi.org/10.1117/12.2001887.
[9] Günther T, Rössl C, Theisel H. Opacity optimization for 3D line fields. In: Proceedings of ACM SIGGRAPH, vol. 32, no. 4; 2013. p. 120.
[10] Viola I, Kanitsar A, Gröller ME. Importance-driven volume rendering. In: Proceedings of IEEE visualization; 2004. p. 139–45.
[11] Turk G, Banks D. Image-guided streamline placement. In: Proceedings of ACM SIGGRAPH; 1996. p. 453–60. http://dx.doi.org/10.1145/237170.237285. ISBN 0-89791-746-4.
[12] Mao X, Hatanaka Y, Higashida H, Imamiya A. Image-guided streamline placement on curvilinear grid surfaces. In: Proceedings of IEEE visualization; 1998. p. 135–42. ISBN 1-58113-106-2.
[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; 2003. p. 213–22. http://dx.doi.org/10.1145/984952.984987. ISBN 1-58113-861-X.
[14] Schlemmer M, Hotz I, Hamann B, Morr F, Hagen H. Priority streamlines: a context-based visualization of flow fields. In: Proceedings of EG/IEEE VGTC EuroVis; 2007. p. 227–34.
[15] Jobard B, Lefer W. Creating evenly-spaced streamlines of arbitrary density. In: Proceedings of eurographics workshop on visualization in scientific computing; 1997. p. 43–55. http://dx.doi.org/10.1007/978-3-7091-6876-9_5. ISBN 978-3-211-83049-9.
[16] Mebarki A, Alliez P, Devillers O. Farthest point seeding for efficient placement of streamlines. In: Proceedings of IEEE visualization; 2005. p. 479–86. http://dx.doi.org/10.1109/VISUAL.2005.1532832.
[17] Liu Z, Moorhead R, Groner J. An advanced evenly-spaced streamline placement algorithm. IEEE Trans Vis Comput Graph 2006;12(5):965–72. http://dx.doi.org/10.1109/TVCG.2006.116.
[18] Li L, Shen HW. Image-based streamline generation and rendering. IEEE Trans Vis Comput Graph 2007;13(3):630–40. http://dx.doi.org/10.1109/TVCG.2007.1009.
[19] Li L, Hsieh HH, Shen HW. Illustrative streamline placement and visualization. In: Proceedings of IEEE PacificVis; 2008. p. 79–86. http://dx.doi.org/10.1109/PACIFICVIS.2008.4475462.
[20] Spencer B, Laramee RS, Chen G, Zhang E. Evenly spaced streamlines for surfaces: an image-based approach. Comput Graph Forum 2009;28(6):1618–31.
[21] Rosanwo O, Petz C, Prohaska S, Hege HC, Hotz I. Dual streamline seeding. In: Proceedings of IEEE PacificVis; 2009. p. 9–16.
[22] Xu L, Lee TY, Shen HW. An information-theoretic framework for flow visualization. IEEE Trans Vis Comput Graph 2010;16(6):1216–24. http://dx.doi.org/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 Trans Vis Comput Graph 2012;19(8):1342–53. http://dx.doi.org/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. http://dx.doi.org/10.1109/PACIFICVIS.2015.7156349.
[25] Lee TY, Mishchenko O, Shen HW, Crawfis R. View point evaluation and streamline filtering for flow visualization. In: Proceedings of IEEE PacificVis; 2011. p. 83–90. http://dx.doi.org/10.1109/PACIFICVIS.2011.5742376.

[26] Krüger J, Schneider J, Westermann R. Clearview: an interactive context preserving hotspot visualization technique. IEEE Trans Vis Comput Graph 2006;12(5):941–8. http://dx.doi.org/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 Trans Vis Comput Graph 2014;20(5):686–701. http://dx.doi.org/10.1109/TVCG.2013.2297914.
[28] Zhang Z, Huang K, Tan T. Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. In: Proceedings of international conference on pattern recognition, vol. 3; 2006, p. 1135–8. http://dx.doi.org/10.1109/ICPR.2006.392.
[29] Moberts B, Vilanova A, van Wijk J. Evaluation of fiber clustering methods for diffusion tensor imaging. In: Proceedings of IEEE visualization; 2005. p. 65–72. http://dx.doi.org/10.1109/VISUAL.2005.1532779.
[30] Jain AK. Data clustering: 50 years beyond *k*-means. Pattern Recognit 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, vol. 1; 2004. p. 344–7. http://dx.doi.org/10.1109/ISBI.2004.1398545.
[32] Chen Y, Cohen J, Krolik J. Similarity-guided streamline placement with error evaluation. IEEE Trans Vis Comput Graph 2007;13(6):1448–55.
[33] Rössl C, Theisel H. Streamline embedding for 3D vector field exploration. IEEE Trans Vis Comput Graph 2012;18(3):407–20.
[34] Yu H, Wang C, Shene CK, Chen JH. Hierarchical streamline bundles. IEEE Trans Vis Comput Graph 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 Trans Vis Comput Graph 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 Trans Vis Comput Graph 2006;12(6):1601–12.
[38] Edmonds J. Paths, trees, and flowers. Can J Math 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. http://dx.doi.org/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. In: 58th Annual forum of the AHS, Montreal, Canada; 2002.
[42] Byrne G, Mut F, Cebral J. Quantifying the large-scale hemodynamics of intracranial aneurysms. Am Neuroradiol 2014;35:333–8. http://dx.doi.org/10.3174/ajnr.A3678.
[43] Aluie H, Eyink G, Vishniac E, Chen S, Kanov K, Burns R, 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 VMV 2011: Vision, Modeling and Visualization. Eurographics Association; 2011. p. 215–22.
[45] Stoll C, Gumhold S, Seidel HP. Visualization with stylized line primitives. In: IEEE visualization, 2005. VIS 05. IEEE; 2005. p. 695–702.

A Voxel-based Rendering Pipeline for Large 3D Line Sets

Mathias Kanzler, Marc Rautenhaus, Rüdiger Westermann:
A Voxel-Based Rendering Pipeline for Large 3D Line Sets.
IEEE Transactions on Visualization and Computer Graphics, 25(7):2378-2391, 2019.
doi:10.1109/TVCG.2018.2834372

A Voxel-based Rendering Pipeline for Large 3D Line Sets

Mathias Kanzler, Marc Rautenhaus, and Rüdiger Westermann

Abstract—We present a voxel-based rendering pipeline for large 3D line sets that employs GPU ray-casting to achieve scalable rendering including transparency and global illumination effects. Even for opaque lines we demonstrate superior rendering performance compared to GPU rasterization of lines, and when transparency is used we can interactively render amounts of lines that are infeasible to be rendered via rasterization. We propose a direction-preserving encoding of lines into a regular voxel grid, along with the quantization of directions using face-to-face connectivity in this grid. On the regular grid structure, parallel GPU ray-casting is used to determine visible fragments in correct visibility order. To enable interactive rendering of global illumination effects like low-frequency shadows and ambient occlusions, illumination simulation is performed during ray-casting on a level-of-detail (LoD) line representation that considers the number of lines and their lengths per voxel. In this way we can render effects which are very difficult to render via GPU rasterization. A detailed performance and quality evaluation compares our approach to rasterization-based rendering of lines.

Index Terms—Ray-casting, large 3D line sets, transparency, global illumination

1 Introduction

3D line sets can be rendered efficiently via the rasterization-based rendering pipeline on GPUs, for instance, by constructing tubes around each line on-the-fly in a geometry shader. For the rendering of very large 3D line sets, however, GPU rasterization introduces the following limitations: a) When using transparency, which requires the rendered fragments to be blended in correct visibility order, all fragments need to be stored in a per-pixel linked list [1] on the GPU and sorted with respect to the current camera position. This approach works efficient for moderate sized line sets, yet for large sets the sorting operation significantly slows down performance, and in some cases the lists can even exceed the GPU's memory (see Fig. 17). b) Operations requiring spatial adjacency queries cannot be embedded efficiently into rasterization-based rendering. For instance, shadow simulation or ambient occlusion calculations to enhance the spatial perception of rendered lines. Even though hard shadows of point lights can be simulated via shadow mapping, this requires a second rendering pass using the full set of geometry, and the resulting high-frequency illumination variations are rather disturbing from a perceptual point of view (see Fig. 12).

An option to overcome these limitations is a rendering technique that does not build upon the order-independent projection of primitives, but is ray-guided and can efficiently traverse the line set along an arbitrary direction. For the rendering of polygon models, GPU voxel ray-casting has been established as a powerful alternative to rasterization. Voxel ray-casting employs a voxel-based object representation in combination with a regular sampling grid that can be traversed efficiently on the GPU [2], [3]. This approach is particular useful because it can generate the sample points along a ray in an order-dependent way, and provides the ability to perform adaptive LoD selection. Furthermore, the regular sampling grid gives rise to efficient search operations, which have been used to simulate global illumination effects via ray-guided visibility computations [4], [5].

Mimicking voxel ray-casting for line primitives, however, is challenging, since no voxel representation of lines is known that can effectively encode spatial occupancy *and* direction. A solid voxelization of line primitives—or tubes generated from them—into a regular sampling grid [6] is not feasible due to the following reasons: Firstly, for the line sets we consider, too many voxels must be generated to differentiate between individual lines. Secondly, upon voxelization, changes of rendering parameters like line width and line removal require exhaustive processing. Thirdly, already at moderate zoom factors the lines will appear blocky. Finally, since voxels do not encode any directional information, direction-guided shading and filtering as well as direction-preserving LoD construction becomes infeasible.

1.1 Contribution

In this work we propose an alternative rendering technique for large 3D line sets on the GPU, which builds upon the concept of voxel ray-casting to overcome some of the limitations of rasterization-based line rendering. We present a new voxel model for 3D lines and demonstrate its use for GPU line rendering including transparency and ambient occlusion effects. We further show that the voxel model can be used in combination with rasterization-based rendering, by performing the simulation of volumetric effects on this model and letting the rendered fragments access the computed results.

Our specific contributions are:

- A novel voxel-based representation of lines, consisting of a macro voxel grid, and a per-voxel quantization structure using voxel face-to-face connectivity.

M. Kanzler, M. Rautenhaus and R. Westermann are with the Computer Graphics & Visualization Group, Technische Universität München, Garching, Germany.
E-mail: {mathias.kanzler, marc.rautenhaus, westermann}@tum.de

- A LoD line representation that considers per voxel the number of lines, their orientations, and lengths to estimate an average line density and direction at ever coarser resolutions.
- An implementation of GPU ray-casting on the voxel-based representation, including the computation of line, respectively tube intersections on the voxel level, to efficiently simulate transparency as well as local and global illumination effects.

We see our rendering approach as an alternative to existing rendering techniques for 3D line sets when there are many lines introducing massive overdraw, and when transparency or global illumination effects are used to enhance the visual perception (see Fig. 19). In this case, our method can significantly accelerate the rendering process, and it can even be used when the memory requirements of rasterization-based approaches exceed what is available on current GPUs. The possibility to construct a LoD line representation enables trading flexibly between quality and speed, and efficiently performing search operations required for simulating advanced illumination effects.

This paper is organized as follows: After reviewing work that is related to ours, in Sec. 3 we introduce the particular voxel-model underlying our approach and discuss the model generation process. Here we show images of voxelized line data using classical GPU line rendering to emphasize the effects of different voxelization parameters on the quality of the resulting representation. In Sec. 4 we describe the efficient realization of voxel ray-casting on the voxelized line representation, and we demonstrate the simulation of advanced rendering effects like transparency and global illumination using voxel-based ray-casting. Sec. 5 provides a thorough evaluation of our approach regarding quality, speed, and memory requirement, and we demonstrate its benefits and limitations with data sets from different applications. We conclude our work with some ideas on future extensions and applications.

2 Related Work

Our approach is related to established techniques in visualization and rendering, namely line-based rendering of flow fields and voxel-based ray-tracing.

Line-based rendering of flow fields

Today, integral curves in 3D vector fields are usually visualized via the rasterization-based rendering pipeline on GPUs, either as illuminated line primitives [7], [8], or as ribbons or tubes which are constructed around each line on-the-fly in the GPU's shader units [9]. For dense sets of opaque lines, illustrative rendering [10], [11] has been proposed to keep separate lines perceptually visible. Other techniques abstract from single primitives and show flow features via line density projections [12], [13]. Automatic view-dependent selection of transparency has been introduced by Günther et al. [14], to selectively fade out lines in those regions where they occlude more important ones. A user study on perceptual limits of transparency-based line rendering for flow visualization has been conducted by Mishchenko and Crawfis [15]. They also suggest a number of specific usages of transparency to effectively avoid visual clutter and high levels of occlusions.

When transparent lines are rendered, generated line fragments need to be blended in correct visibility order. On the GPU, this can be realized by using either depth peeling [16] or per-pixel fragment lists [1], [17]. Depth peeling does not require storing and sorting fragments, yet it requires rendering the data set as many times as the maximum depth complexity of the scene, i.e., the maximum number of fragments falling into the same pixel. For the data sets we consider, where along the majority of view rays the depth complexity is in the order of many hundred or even thousand, the resulting increase in rendering times is not acceptable. Per-pixel fragment lists, on the other hand, require only one single rendering pass, yet they require storing all fragments and can quickly run out of memory on current GPUs. This also holds when depth peeling is applied to fragments bucketed by depth [18], even though the number of rendering passes can be reduced. As an alternative to the exact simulation of transparency, stochastic transparency [19] uses sub-pixel masks to stochastically sample the fragments' transparency contributions. Stochastic transparency requires only a rather small and fix number of rendering passes, yet it transforms directional structures into noise. This is especially undesirable in our scenarios, where only lines are rendered and even in transparent regions their directional structure should be preserved.

The technique most closely related to ours is the one by Schussman and Ma [20]. They voxelize streamlines into a regular grid, and compute a spherical harmonics representation of the lighting distribution caused by the line segments in every voxel. Voxel-based rendering with sub-pixel illumination can then be performed, yet single line primitives cannot be determined any more, for instance, to render the initial lines or compute exact occlusions.

Voxel-based ray-tracing

Advances in hardware and software technology have shown the potential of ray-tracing as an alternative to rasterization, especially for high-resolution models with many inherent occlusions. Developments in this field include advanced space partitioning and traversal schemes [21], [22], [23], and optimized GPU implementations [24], [25], [26], to name just a few. All these approaches can be classified as "conventional ray-tracing approaches", since they operate on the polygon object representation and perform classical ray-polygon intersection tests. Recently, Wald et al. [27] proposed the use of ray-tracing for particle rendering, by using a tree-based search structure for particle locations to efficiently find those particles a ray has to be intersected with. The integration of global illumination effects like ambient occlusion into particle visualizations has been demonstrated by Wald et al. [27] and Staib et al [28].

Voxel models have been first introduced 1993 by Kaufman in the seminal paper [29]. Compared to polygonal representations, they provide an interesting set of advantages like easier level of detail computation and combined storage of surface and geometry information. A detailed investigation of algorithms for line voxelization has been provided by Cohen-Or and Kaufman [6]. Interestingly, a

voxel model has also been used in the very first published work on iso-surface visualization: *Cuberilles* [30]. The Cuberille method—or opaque cubes—works by computing the set of grid cells that contain a selected iso-surface and rendering those as small cubes.

In the last years, there has been a lot of research into large octrees to render high-resolution voxel models [2], [3], [31]. All these approaches subdivide the model into a sparse volume, storing only small volume "bricks" along the initial model surface, and use a compute-based octree traversal to render the contained surface. The potential of voxel-based rendering approaches to efficiently simulate global illumination effects has been demonstrated, for instance, in [5], [32]. The survey by Joenssen et al [33] provides a general overview of approaches for simulating global illumination effects on volumetric data sets.

3 Voxel-based curve discretization

The input of our method consists of a set of curves, e.g. streamlines, where each curve is approximated by a connected line set. Each line set is represented by a sequence of vertices v_i and corresponding attributes a_i. The attributes can be specified optionally, for instance, to enable the use of a transfer function to interactively change the curves' colors or transparencies. The curve discretization is performed in a pre-process on the CPU, and the generated data structure is then uploaded to the GPU where rendering is performed. This pre-process can also be performed on the GPU, enabling instant update operations when curves are removed or new curves are added. This, however, requires some modifications, the discussion of which we delay until the end of the current section. We propose a two-level grid structure for discretizing the initial curves, as illustrated in Fig. 1.

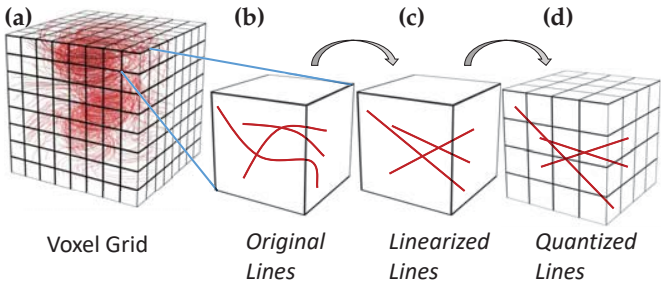

Fig. 1: Two-level grid structure. First level: Voxel grid, curves are clipped to the voxel faces (b), and per-voxel linear line segments are generated (c). Second level: Line vertices are quantized based on a uniform subdivision of voxel faces (d).

The first level is given by a voxel grid, in which each curve is approximated by a set of per-voxel linear line segments. A regular subdivision of each voxel face yields the second level, where the endpoints of the per-voxel line segments are quantized to the center points of sub-faces.

The use of a two-level structure allows controlling the approximation quality of the voxelization and storing the line segments per voxel in a compact form. The resolution of the voxel grid controls the size of the geometric features that can get lost when approximating these features by linear segments in every voxel. With increasing grid resolution better approximation quality is achieved, yet at the cost of increasing memory requirement. The end points of each per-voxel line segment can either be stored exactly using floating point values, or they can be quantized to a set of points on the voxel faces. This allows for a compact encoding of the line segments per voxel, and it can be used to control how many lines passing through one voxel are collapsed to one single line. While we let the user select the quantization resolution, more advanced quantization strategies can generate a locally adaptive quantization to assure that local line features are well preserved [34].

3.1 Curve voxelization

Initially, the resolution $r_x \times r_y \times r_z$ of the 3D voxel grid into which the curves are voxelized is set. We use cube-shaped voxels of side length 1 and set the resolution so that the aspect ratio of the bounding box of the initial curves is maintained. The vertex coordinates v_i are then transformed to local object coordinates in the range from $(0, 0, 0)$ to (r_x, r_y, r_z).

For each curve and starting with the first vertex, every pair of vertices v_i and v_{i+1} is processed consecutively. A line through v_i and v_{i+1} is clipped against the voxel boundaries, via line-face intersection tests in the order of their occurrence from v_i to v_{i+1}. If v_i and v_{i+1} are located in the same voxel, no new intersection point is generated. This gives a sequence of voxel-face intersections, and every pair of consecutive intersections represents a line that enters into a voxel and exits that voxel. In general the first and last vertex of a line do not lie on a face, we hence omit the segments from the first vertex to the first face intersection and from the last face intersection to the last vertex.

Fig. 2 illustrates the voxelization process, demonstrating increasing approximation quality with increasing grid resolution, as well as limitations of the piecewise linear curve approximation when the grid resolution is too low. The maximal deviation between the initial curve and the generated line segments is bound by the length of a voxel's diagonal.

For every voxel in the voxel grid, a linked list is used to store the lines intersecting that voxel. Line attributes, interpolated to the line-face intersection points, and IDs that identify to which curve a line belongs can be stored in addition.

3.2 Line quantization

For quantizing the coordinates of the curve-voxel intersection points, we use a small number of bins per face: As illustrated in Fig 1d, every square voxel face is subdivided regularly into $N \times N$ smaller squares. The coordinates of the intersection points are then quantized to the centers of these squares. Thus, every vertex can be encoded via a $2N$ bit pattern (indicating the sub-square in a voxel face) and a 3 bit face ID (indicating on which face the vertex is located). The bits are packed into a single word that is just large enough to store them. Since voxels have unit size and each voxel face is subdivided equally, the vertex location in object coordinates (quantized to the per-face square centers) can be computed

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2018.2834372

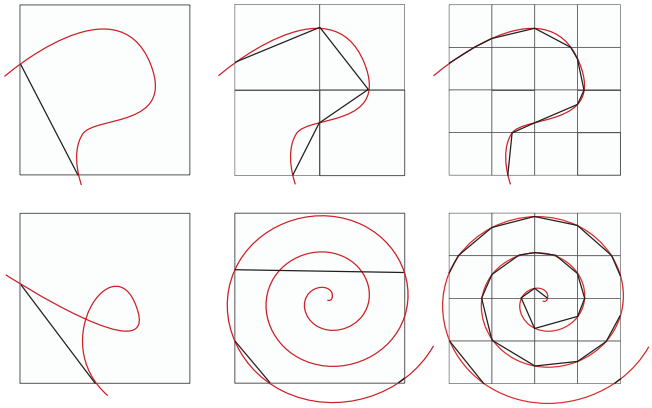

Fig. 2: Curve voxelization: Initial curve in red and piecewise linear approximation in black. Top: Decreasing approximation error with increasing voxel grid resolution. Bottom: Geometric details can be missed if the voxel grid resolution is too low.

from a bit pattern stored per-vertex. Fig. 3 illustrates the quantization process and the effects of different values of N on the curve approximations.

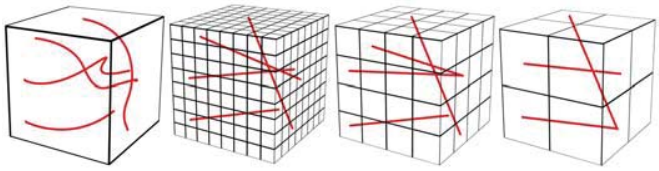

Fig. 3: Line quantization. From left to right: Original curves, per-voxel linear approximation using vertex quantization to $8^2, 4^2$, and 2^2 sub-faces (bins) per voxel face, respectively. If the number of bins is too low, lines can fall onto each other.

The quantization process introduces an additional approximation error that depends on the selected subdivision of voxel faces. This error does not cause any geometric details to be lost, yet it slightly jitters the vertex locations and, thus, affects a line's orientation. In particular, different lines might be mapped onto the same quantized line because their vertices are quantized to the same locations.

Fig. 4 shows a direct comparison between the original curves and the voxelized curves using a voxel grid resolution and per-face subdivision at which the discretization errors can only just be perceived. In both cases the lines are rendered via GPU rasterization; by starting at the first vertex and then either by traversing the original set of vertices and constructing tubes around each line on-the-fly in a geometry shader, or by performing the same construction on the discretized line set. Especially the straight curves in the foreground show some subtle bumpiness that is caused by the quantization of line vertices.

To demonstrate how, in general, the voxel grid resolution and the voxel face subdivision affect the final reconstruction quality, Fig. 5 shows some extreme cases which also reveal the interplay between both resolutions. A detailed analysis of the dependencies between resolution, quality and memory consumption for different resolutions is given in Sec. 5.

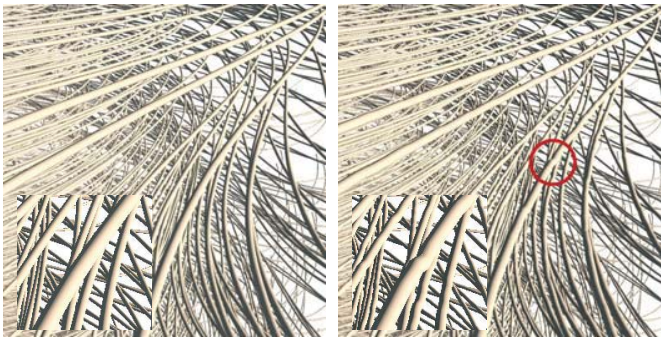

Fig. 4: GPU rasterization of initial lines (left) vs. rasterization of voxelized lines using 256^3 voxel grid and curve-face intersections quantized to 16^2 bins (right).

While a low resolution of the voxel grid affects in particular the per-voxel approximation error, a low degree of face subdivision can introduce additional C^1-discontinuities at voxel transitions, i.e., by jittering a vertex that is shared by two lines with the same orientation. Especially high frequent variations that occur when a short line is jittered are perceptually noticeable, as indicated by the last example in Fig. 5.

3.3 GPU implementation

The per-voxel linked list representation used to store the lines is neither memory efficient, as each line stores a pointer to the following element, nor can subsequent lines be accessed quickly, as they are not stored consecutively in memory and, thus, cached reads are prohibited. Therefore, before uploading the data to the GPU, the lines are reordered in memory so that all lines passing through the same voxel are stored in consecutive elements of a linear array. The final data structure stores for each voxel a header, which stores the number of lines for that voxel, and a pointer to the array element storing the first line passing through it.

Constructing the voxel model on the GPU consists of three stages: Firstly, in parallel for every initial curve we compute the lines per voxel and write them into a linear append buffer. Every segment is assigned the unique ID of the voxel it is contained in. Next, the buffer is sorted with respect to the voxel IDs, so that all lines falling into the same voxel are located consecutively in the buffer. An exclusive parallel prefix sum is then computed in-place over the buffer to count the number of line per voxel. Now, for every voxel the start index of its line set can be determined from the content of the buffer, and stored in a separate voxel buffer at the corresponding location. This buffer is used at render time to look up at which position in the global buffer the lines for a particular voxel are stored.

3.4 LoD construction

One important property of classical voxel models for surfaces is that a LoD structure can be generated efficiently by simple averaging operations on the voxel values to aggregate information. For a line set that is voxelized as proposed in our work, such an averaging operation cannot be applied immediately because a) an averaging operator

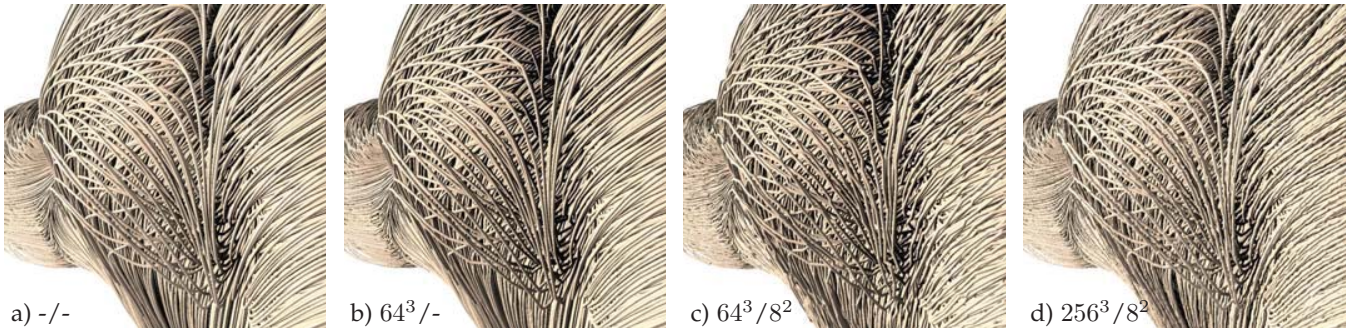

Fig. 5: Rasterization of streamlines in the 256^3 Aneurysm II dataset. (a) Initial lines. (b) Line encoding in 64^3 voxel grid with exact curve-voxel intersections. (c) Line encoding in 64^3 voxel grid with curve-voxel intersections quantized to 8^2 bins. (d) Line encoding in 256^3 voxel grid with curve-voxel intersections quantized to 8^2 bins.

for lines first needs to be defined, and b) every voxel might store not only one but many lines. It is worth noting that regardless of how a) and b) are addressed, it is impossible, in general, to represent the lines in one voxel by one single average line so that continuity with the average lines in adjacent voxels is ensured.

We address the problem of LoD construction for a voxelized line set as follows: Since we intend to use the LoD structure in particular to accelerate the simulation of global illumination effects, we derive a scalar indicator for the amount of light that is blocked by the lines in a single voxel. By using standard averaging operators, an octree LoD structure can then be generated in a straight forward way from the indicator field. For every voxel, we compute an average *density* value ρ from the lines passing through it, by taking into account the lines' lengths and opacities:

$$\rho = \sum_{l_i \in L} length(l_i)\sigma_i, \tag{1}$$

where L is the set of lines in the voxel, and l_i and σ_i are the length and opacity of the i-th line in that voxel, respectively. The lines' opacities are either set to a constant values or assigned individually via a transfer function. Finally, an octree is build bottom-up by averaging the density values in 2^3 voxels into one voxel at the next coarser level (see Fig. 6).

Fig. 6: Volume rendering of the initial line density per voxel as computed via eqn. (1), and the line density values at the first, second, and third LoD.

The density values can be interpreted as per-voxel opacities telling how much light is blocked by a voxel. Since the amount of blocking depends also on how the lines are oriented with respect to the incoming light, direction-dependent opacity values are favorable in principle [35]. Even though they can be generated in a straight forward way, by simulating the attenuation from a number of selected directions via ray-casting, we did not consider this option in the current work to avoid storing many values per voxel and, thus, increasing the memory consumption significantly.

In addition to the per-voxel opacity values, one representative average line is computed for every voxel; by averaging separately the start points and end points of all lines in a voxel. This gives two average points which are snapped to the closest bin on the voxel boundaries, and from which the average line is computed. Care has to be taken regarding the orientation of lines, i.e., when two averaged lines have vastly opposite directions. Therefore, before considering a new pair of start and end point, we first test whether the corresponding line has an angle of more than 90 with the previous line, and we flip the line if this is the case. We also attempt to establish connectivity between representative lines in adjacent voxels if their endpoints are in the same voxel face, by snapping the endpoints to the quantization bin into which the average of both points is falling. In this way, we can often achieve continuous representative curves, even though it is clear that in general such a continuous fit is not possible. Fig. 7 illustrates a multi-resolution representation of a set of curves. Note in particular how well in certain regions the average curves even on the coarser resolution levels represent the initial curves.

Fig. 7: Rasterization of the initial curves (left) compared to 1st (middle) and 2nd (right) LoD.

The representative lines are stored in addition to the per-voxel line set, and a LoD representation can be computed by propagating these lines to ever coareser resolution levels. The average per-voxel opacity values at different resolution levels are stored in a set of 3D texture maps. By using these quantities we can efficiently realize a number of rendering accelerations and effects, which we will introduce in Sec. 4.

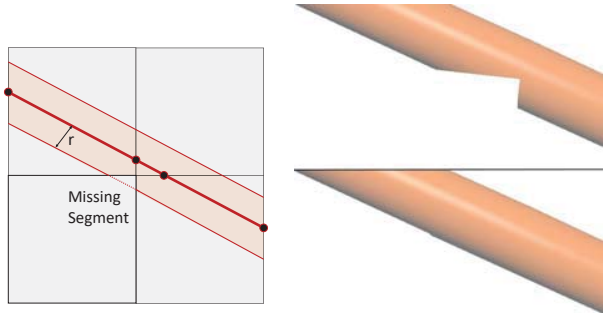

Fig. 8: When a tube extends into a voxel, and a ray intersects that voxel but none of the voxels in which the tube's center line is encoded (left), no intersection is found and a piece of the tube is missing (right top). By including neighbouring voxels in the intersection test, missing intersection points are found (right bottom).

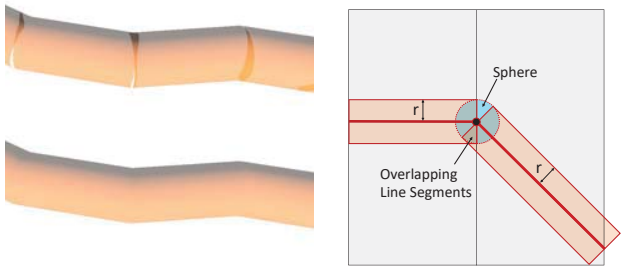

Fig. 9: Left top: Overlapping tubes are blended incorrectly and parts are missing. Rendering a sphere at line joints (right) continuously closes the gaps (left bottom).

4 Voxel-based Line Raycasting

Once the voxel-based line representation has been constructed and the data is residing in GPU memory, volume ray-casting can be used to render the lines in front-to-back order. For every pixel a ray is cast through the voxel grid, thereby going from voxel face to voxel face using a digital differential analyzer algorithm. The voxel grid serves as a search structure to efficiently determine those lines that need to be tested for an intersection with the ray.

Whenever a voxel is hit, the voxel header is read to determine how many lines are stored in that voxel, and if a voxel is empty, it is skipped. Otherwise, the lines are read consecutively and intersected with the ray. Here it is assumed that the lines are in fact tubes with a user-defined radius, so that the intersection test becomes a ray-tube intersection test. This test yields an entry and exit point, from which the distance the ray travels inside the tube can be computed and used, for instance, to simulate attenuation effects. If more than one intersection with tubes are determined, the intersections are first computed and then sorted in place in the correct visibility order.

4.1 Ray-tube intersections

During ray-casting, a problematic case can occur if a tube stands out of the voxel in which its center line is defined. Since the tube expands into a voxel which may not know the center line, the piece in this voxel cannot be rendered if the ray doesn't intersect any of the voxels in which the center line is encoded. This situation is illustrated in Fig. 8. A straightforward solution to this problem is for a given ray and voxel intersection to also test for intersections with lines from adjacent voxels, a solution that unfortunately decreases rendering performance due to the increased number of intersection tests. However, our experiments have shown that the artifacts that occur when neglecting the missing segments are only rarely visible. In particular when rendering tubes with transparency the artifacts are hardly perceivable. As a compromise, we hence during interactive navigation restrict our method to testing against the lines in the voxel hit by the ray. As soon as the camera stands still, adjacent voxels are also taken into account.

Another problematic situation occurs at the joint between adjacent lines, i.e., a gap is produced if the lines do not have the same direction (see Fig. 9). The gap is filled by rendering spheres at the end points of the line segments with a radius identical to the tube's radius. To avoid blending the same line twice, we keep track of the IDs of intersected lines using bit operations and consider an intersection point only once.

At every entering ray-tube intersection point, we calculate the tube normal and evaluate a local illumination model. If the tubes are rendered opaque, this only has to be done for the first intersection point. The resulting color value is combined with the tube color, and this color is used as pixel color. Fig. 10 compares the rendering of opaque tubes via GPU ray-casting to GPU line rasterization.

Fig. 10: Opaque line rendering. Left: GPU ray-casting of voxelized lines (8 ms, 256^3 voxel grid, curve-face intersections quantized to 16^2 bins). Right: GPU line rasterization (12 ms).

Interestingly, in terms of rendering quality rasterization and ray-casting do not seem to show any significant differences at the selected discretization resolution, yet even for opaque lines ray-casting renders already faster than rasterization. The main reason is that ray-casting can effectively employ early-ray termination once the first intersection with a tube is determined. Since rasterization renders the lines in the order they are stored, which is not the visibility order in general, it needs to generate a considerably larger amount of fragments.

4.1.1 Transparency rendering

If the tubes are rendered semi-transparent, at every entering ray-tube intersection an opacity value α is either read from the voxel header or assigned via a transfer function. The opacity value is then used to modulate the tube color, and this color is blended with the pixel color using front-to-back α-compositing, i.e., in the order in which the tube intersections are determined along the ray. The described way of handling opacity is exactly how the final pixel color is computed when semi-transparent lines are rendered via GPU rasterization. In contrast to ray-casting, however, in GPU rasterization all generated fragments first have to be stored and finally sorted per-pixel with respect to increasing depth. Only then can the fragments' colors be blended in correct order.

Fig. 11 shows colored and semi-transparent lines, once rendered via GPU rasterization and once via GPU ray-casting. In this situation, the advantage of ray-casting comes out most significantly: Since the ray-tube intersection points are computed in correct visibility order, there is no need to store and finally sort these points for blending. Due to this, the performance gain of ray-casting compared to rasterization now becomes significant; about a factor of 7 for the used dataset.

Fig. 11: Left: Transparency rendering using fragment linked lists on the GPU (160 ms, 600 MB fragment list). Right: GPU ray-casting (24 ms, 256^3 voxel grid, curve-face intersections quantized to 16^2 bins, 60 MB voxel representation).

Furthermore, additional acceleration and quality improvement strategies can be integrated into ray-casting in a straight forward way. Firstly, α-termination, i.e., the termination of a ray once the accumulated opacity exceeds a user-defined threshold, can be used to reduce the number of ray-tube intersection points. If a LoD representation is available, even opacity-acceleration [36] can be employed, i.e., increasing the step size along the ray and simultaneously sampling the opacity from ever coarser resolution levels with increasing optical depth. Secondly, instead of considering a constant opacity per tube, the opacity can be made dependent on the distance the ray travels within the tube. Since together with the ray entry point also the ray exit point is computed, this distance is immediately available. Even the handling of penetrating tubes does not impose any conceptual problem, and only requires to sort the ray-tube intersections locally per voxel.

4.1.2 Shadow simulation

The possibility to efficiently trace arbitrary rays through the voxelized 3D curves can be employed to efficiently simulate global illumination effects such as shadows. Shadows provide additional depth and shape cues, and can significantly enhance the visual perception of the curves geometry and their spatial relationships.

The simulation of hard shadows of a point light source can be realized by sending out shadow rays and testing if the ray hits another tube before it hits a light source. However, as demonstrated in Fig. 12b, due to the high frequency shadow patterns that are caused by a dense set of curves, hard shadows rather disturb the visual perception than help to improve it.

We propose the following two approaches to incorporate shadows into the rendering of large line sets without introducing high-frequency shadow patterns. The first approach is to test the shadow rays against the representative lines at a coarser LoD, thus reducing the number of lines that throw a shadow and making the shadows wider and more contiguous (see Fig. 12c). The second approach replaces hard shadows by soft shadows, by sampling the line density values along the shadow rays to measure the amount of blocking. (see Fig. 12d). Both approaches require to traverse only one single shadow ray towards the light source and can be performed efficiently on the proposed LoD representation.

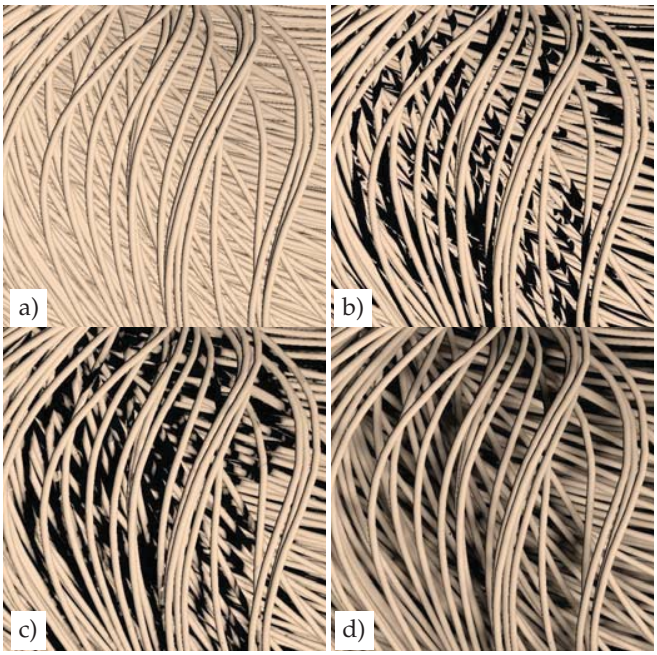

Fig. 12: Voxel-based line ray-casting with (a) local illumination (10 ms), (b) point light shadows (22 ms), (c) point light shadows from representative lines at first LoD (12 ms), (d) soft shadows from line density values via cone-tracing (11 ms).

In the first approach, the shadow rays traverse the voxels of a selected LoD and test for intersections with the representative line in every voxel. Even though we do not avoid hard shadows in this way, by using one single yet thicker line to represent many thinner lines, the frequency of variations from shadow to non-shadow can be reduced significantly. The second approach mimics the effect of an area light source, requiring, in general, to use many rays

to estimate how much of the light leaving the area light is blocked. Instead, we sample the line density values with one ray in a way similar to cone-tracing [4], i.e., by sampling from ever coarser resolution levels with increasing distance from the illuminated point. In particular, we simulate the amount of light that falls onto the point along a cone with an opening angle that subtends one voxel at the finest level.

When comparing the results of both approaches to the rendering of hard shadows in Fig. 12a, one can see that the shadow frequency is considerably reduced and the visual perception is improved. The major shadowing effects, on the other hand, are still present in the final renderings, and the spatial relationships between the lines are effectively revealed. When using the first approach, the render time increases about 20% compared to the rendering without shadows; the second approach yields a decrease of about 10%. These only marginal decreases are due to the use of the LoD representation, which requires testing against only one single line per voxel when using the first approach, and interpolating trilinearly in the line density fields at different resolution levels when using the second approach. Since a texture lookup operation takes less time than an explicit ray-tube intersection test, the second approach performs even faster than the first one at almost similar quality.

4.1.3 Ambient occlusion

Both rendering approaches for low-frequency shadows can also be used to simulate ambient occlusions (AO), i.e., soft shadows that occur in the cavities of a 3D object when indirect lighting is cast out onto the scene. As demonstrated in Fig. 13, the soft shadows from ambient occlusions help in particular to enhance the spatial separation between individual lines or bundles of lines.

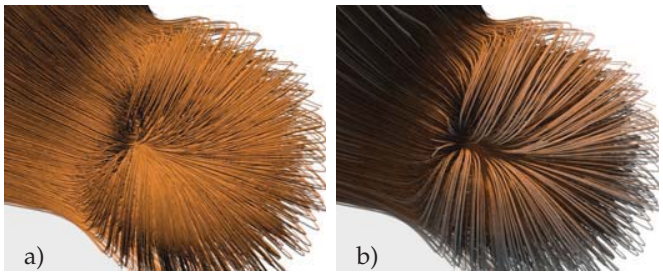

Fig. 13: Local lighting (a) vs. ambient occlusion (b).

AO is simulated by casting out rays to sample the surrounding geometry, and computing how much light from an environment map is blocked by this geometry. For simulating shadows we restrict the direction along which we send out rays to the direction of a directional light or towards a point light source, yet ambient occlusion requires to send out rays into the entire upper hemisphere with respect to the normal direction at a surface point. AO can be integrated in a straight forward way into the ray-based rendering pipeline, by spawning at every visible point a number of secondary rays into the hemisphere and calculating whether the environment light is blocked or not. In Fig. 14a, 2500 rays per visible point were used to uniformly sample the hemisphere, and these rays were intersected against the tube segments in every voxel to determine whether the light is

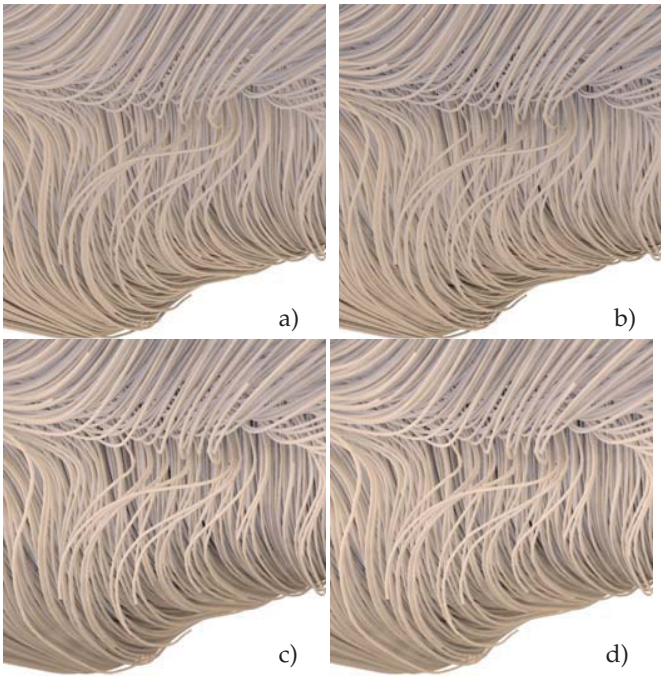

Fig. 14: Top: Hemisphere ambient occlusion using 2500 shadow rays. (a) Rays are tested against line geometry (>10 s). (b) Occlusions are estimated from line density values at first LoD (600 ms). Bottom: Pre-computed per voxel spherical occlusions, sampled trilinearly along 2500 (c) and 25 (d) shadow rays at 12 ms and 1130 ms, respectively.

blocked or not. The higher variance in the ambient occlusion values when less rays are used can clearly be seen.

To enable interactive updates of AO values when the scene or light situation changes, the number of rays as well as the number of objects against which the shadow rays are tested need to be reduced. Screen-space approaches [37] compute a rough AO estimation by using a few rays in 2D screen-space (between 8 and 20 in realtime applications), and by testing these rays against rendered surface points in a short radius of influence.

To overcome visual shortcomings of screen-space calculation, we compute AO values in 3D space, and present two acceleration strategies to efficiently approximate the amount of occlusion per tube point. Similar to screen-space ambient occlusion, we restrict the sampling of occluding structures to a radius of influence (in our case 15 voxels at the finest voxel resolution).

The first approach is to approximate the amount of occlusion along a ray by sampling the line density values at the finest level voxel grid via trilinear interpolation (Fig. 14b), instead of testing against the tube geometries at this level (Fig. 14a). The values along a ray are accumulated until either full blocking is reached (line density ≥ 1) or the ray reaches beyond the radius of influence of leaves the domain. The occlusion values are finally integrated over all rays and normalized by dividing through the number of rays.

The renderings in Figs. 14a and b show that the AO values vary strongly around the tube axis, which is due the hemisphere sampling of occlusions wrt to the varying normal direction. Due to the many tubes that are rendered,

Fig. 15: The datasets we have used in our experiments: Tornado, Aneurysm I, Aneurysm II, Turbulence, Weather Forecast.

this adds high frequent intensity variations which rather disturb the visual impression than help to enhance the spatial relationships between the tubes. To avoid this effect, we propose a second approach which computes point-wise AO values independent of the surface orientation by considering occlusions in the entire sphere around each visible point (see Fig. 14c and d). The AO values are first approximated per voxel in a pre-process, and at runtime these values are trilinearly interpolated at the locations of the visible tube points. In this way the AO values reflect the local spherical surrounding of a line rather than the surrounding in the normal direction at the tube points. This emphasizes in a far better way the embedding of a line in the surrounding set of lines.

5 Results and Discussion

In this section, we analyze the quality, memory consumption, and performance of our approach. All times were measured on a standard desktop PC, equipped with an Intel Xeon E5-1650 v3 CPU with 6×3.50 GHz, 32 GB RAM, and an NVIDIA GeForce GTX 970 graphics card with 4 GB VRAM. For all renderings, the view port was set to 1920×1080.

We used the following datasets to analyze the quality, memory requirements, and performance of the proposed voxel-based rendering approach for line sets (see Fig. 15):

- **Tornado**: 1000 randomly seeded streamlines in a flow forming a tornado.
- **Aneurysm I/II**: 4700 / 9200 randomly seeded streamlines in the interior of two aneurysms [38]. Streamlines were advected up to the vascular wall, resulting in empty space up to the cuboid domain boundaries.
- **Turbulence**: 50000 domain-filling streamlines advected in a forced turbulence field of resolution 1024^3 as described by Aluie et al. [39].
- **Weather Forecast**: 212000 domain-filling path lines computed over 96 hours each on the wind field of a forecast by the European Centre for Medium-Range Weather Forecasts.

5.1 Quality analysis

To analyze the effect of the resolution of the voxel grid and the quantization structure on reconstruction quality, we performed a number of experiments with the datasets listed above. The subjective visual quality of the rendered images is hard to measure, wherefore we attempt to quantify the quality objectively by measuring the following quantities: For different resolutions we measured the mean Hausdorff distance between the original curves and their piecewise linear approximations (Fig. 16a), the mean angle between the tangents at the original curve points and the curve's piecewise linear approximations (Fig. 16b), and the number of line segments falling onto each other due to the quantization of vertex coordinates (Fig. 16c).

Fig. 16a indicates that already at a voxel grid resolution of 128^3 and a quantization resolution of 32, suitable reconstruction accuracy is achieved. At a voxel grid resolution of 64^3, geometric features get lost and cannot be faithfully reconstructed even at high quantization resolution. Fig. 16b shows essentially the same dependencies, yet one can observe a stronger effect of the quantization resolution. The local directional changes of the curve tangents due to displacements of the vertex coordinates is scale independent and depends mainly on the quantization resolution. From Fig. 16c it can be seen that at a voxel grid resolution of 128^3 and higher, and starting at a quantization resolution of 32, the number of duplicate lines is below 0.1% of all encoded lines and doesn't change significantly beyond these resolutions. The maximum Hausdorff distance, on the other hand, is always bounded by the voxel diagonal, and the directional error can be up to 180 degrees, if a curve makes a loop in a voxel.

Supported by our analysis, and further verified by comparing the visual quality of the original curves and the voxel-based curve representations, we found that a voxel grid resolution of 256^3 and a quantization level of 32 is for all cases the resolution at which a further increase in voxel resolution or bin size only causes little improvement in error quantities This is also evidenced by the close-up views in Fig. 18, where no apparent differences between the original curves and their voxelized counterparts can be perceived. Due to this, we decided to use these resolutions in the performance analysis and the analysis of illumination effects below.

5.2 Memory statistics

Our approach requires a 5 byte header for every voxel, to indicate how many lines are encoded per voxel (1 byte, restricting to a maximum of $2^8 - 1$ lines per voxel) and to reference the memory address where the lines are stored (4 bytes). In addition, every line is encoded by specifying at which of the 6 voxel faces the two endpoints are located ($2 \cdot 3$ bits), and addressing the sub-face on each voxel face to which the endpoints are quantized ($2 \cdot 10$ bits for a quantization resolution of 32^2). Furthermore, we use 1 byte per line to map to a transparency or color value and 5 bits to encode a local line ID. For a selected quantization resolution, the minimum number of bytes that is required to encode this

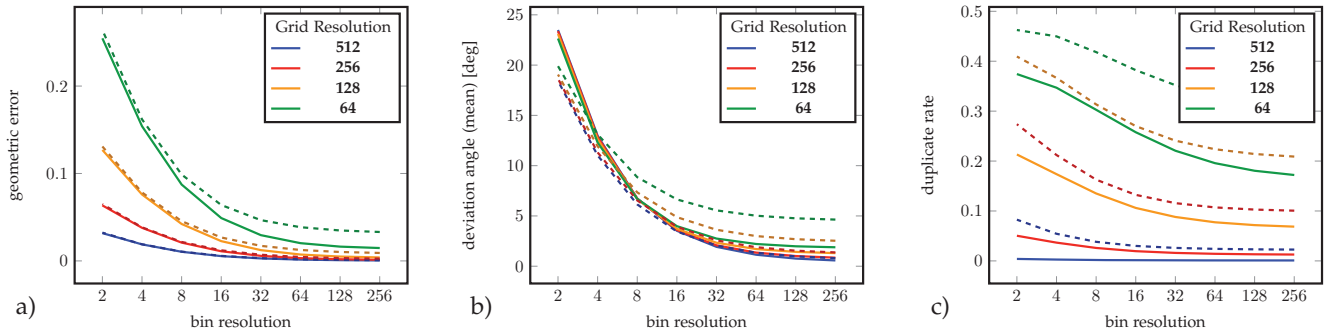

Fig. 16: Comparing the Aneurysm II (solid line) with the Weather Forecast (dashed line) dataset using different metrics: (a) Distance Error (in units of voxel size at a grid of 64^3), (b) tangent deviation, (c) Number of duplicates in proportion to the number of generated segments

information is used. Overall, our approach requires 4, 5, and 6 bytes per line, respectively, when quantization resolutions of (4,8), (16,32) and (64,128) are used.

Table 1 compares the memory consumption for different resolution and quantization levels with the memory that is required to store the original lines on the GPU, using 32 bit float values per vertex component and 1 byte per vertex to map to transparency or color. In the second column we show in brackets the memory that is required in average over multiple views to store the per-pixel fragment lists when transparent lines are rendered. It can be seen that for all but the Tornado dataset the voxel-based representation at our selected resolution level ($256^3/32^3$) has a significantly lower memory footprint. Even when opaque lines are rendered and the extra memory for storing the fragment lists is not required, a rather moderate increase of about a factor of 4 to 5 is observed. One exception is the Weather Forecast dataset, which is comprised of many curves with only few long lines. These lines are split into many smaller lines and stored in the voxel representation, so that the memory requirement is significantly increased compared to the original line representation.

It is in particular interesting that due to memory limitations on the GPU the Weather Forecast dataset cannot be rendered via GPU rasterization if transparency is used (see Fig. 17). Our approach requires only 144 MB to store the view-independent voxel-based representation on the GPU.

Fig. 17: Left: GPU rasterization using fragment linked lists fails due to memory limitations. Red indicates 512 or more fragments fall onto a pixel, when not all fragments could be stored due to memory limitations the pixel is set to white. Right: Voxel-based ray-casting renders at 23 ms.

5.3 Performance analysis

For all datasets, we compare the performance of rasterization-based line rendering with and without fragment lists to voxel-based line ray-casting (see Tab. 2). For rasterization-based rendering, which unfolds a given line primitive into a number of triangles approximating a cylinder, we can also unfold the line into a quadrilateral and let the generated fragments perform an analytical ray-cylinder intersection test. This approach, even though it frees the geometry shader, requires some additional stitching geometry to achieve the appearance of a continuous tube. Furthermore, it increases the load in the pixel shader, which is already the bottleneck when using transparency; the use case we focus on in this work. Since even for opaque lines we observed only a slight performance increase when using this approach, we decided to refrain from using it. In all of our experiments we let the geometry shader generate 8×2 triangles per line segment.

To also compare the performance of our approach to that of triangle-based GPU ray-tracing, for all datasets we saved the generated triangles to a file and used them as input geometry for the OptiX ray-tracing framework [40]. We selected Spatial Splits in Bounding Volume Hierarchies (SBVH) [41] as acceleration structure. Besides the fact that both the Turbulence and Weather Forecast datasets could not be rendered due to memory limitations, for the smaller datasets (Aneurysm I and II) we observed almost similar frame rates when rendering opaque tubes. When rendering transparent tubes, however, the performance dropped significantly, of up to a factor of 10. We attribute this to the fact that SBVH allows skipping empty space surrounding the line sets efficiently, yet when the lines are dense in the interior (as in Aneurysm I and II), repeated traversal operations slow down the performance. If more space in the interior is empty, this limitation is reduced: OptiX renders the opaque lines in Tornado at 160 fps, while our approach renders at 83 fps. However, it is fair to say that in this case also our approach can be speeded up significantly by using the octree voxel grid to skip empty space. At a per-line opacity of 25 percent, OptiX and our approach (w/o empty space skipping) render at 35 fps and 67 fps, respectively.

In Fig. 21 we show the use of transparency to reveal interior structures that are occluded when opaque lines are rendered. Even when opaque lines are rendered, so that fragment lists and sorting is not required in GPU rasterization, the larger datasets can be rendered at higher rates using voxel-based ray-casting. The main reason is that a ray can be terminated immediately when the first

TABLE 1: Statistics on memory consumption and preprocessing time. The original geometry is encoded in three float values per vertex. In the second column, we give in brackets the memory that is required to store the per-pixel fragment lists on the GPU. The remaining columns show the memory that is used by the voxel-based representation—with voxel grid resolution V^3 and quantization resolution Q^2 given as V^3/Q^2—, and in brackets the preprocessing time to generate the representations. Per vertex and per quantized line a 1 byte index is stored.

Dataset	Lines (vertices per line)	Original Geometry (Linked List)	$512^3/256^2$ (Preprocess)	$256^3/128^2$	$256^3/32^2$	$128^3/128^2$	$128^3/32^2$	$64^3/8^2$
Tornado	1000 (250)	3 MB (100 MB)	811 MB (1.4 s)	103 MB (0.2 s)	103 MB (0.2 s)	14 MB (0.1 s)	14 MB (0.1 s)	2 MB (0.1 s)
Aneurysm I	4700 (410)	25 MB (600 MB)	731 MB (1.6 s)	111 MB (0.4 s)	107 MB (0.4 s)	25 MB (0.2 s)	22 MB (0.2 s)	6 MB (0.3 s)
Aneurysm II	9200 (367)	44 MB (750 MB)	702 MB (1.9 s)	116 MB (0.6 s)	109 MB (0.6 s)	29 MB (0.5 s)	26 MB (0.5 s)	8 MB (0.4 s)
Turbulence	50000 (220)	143 MB (2500 MB)	1087 MB (3.5 s)	221 MB (1.4 s)	201 MB (1.4 s)	72 MB (1.3 s)	62 MB (1.3 s)	21 MB (1.0 s)
Forecast	212000 (13)	36 MB (>4000 MB)	573 MB (5.6 s)	209 MB (4.6 s)	177 MB (4.6 s)	98 MB (5.0 s)	82 MB (5.0 s)	31 MB (4.7 s)

ray-tube intersection is computed, while GPU rasterization always needs to generate all fragments even if the early depth-test can discard many of them before entering the fragment stage. On the other hand, for the Tornado dataset, where the line density is rather low so that many rays need to be traversed through the entire voxel grid, voxel-based ray-casting performs slower than rasterization-based rendering. This still holds when transparency is used. The fact that in rasterization-based rendering using transparency all fragments need to be sorted explains the over-linear increase of the render times, while the render times remain almost constant when ray-casting is used on the voxel-based representation.

The worst case scenario for the ray-caster is a very dense line set with very low line opacity. This prevents rays from terminating early, so that all lines along the rays have to be accessed and tested for intersections. Even though this scenario is quite unusual, we analyzed the rendering performance in this situation for the Weather Forecast and Turbulence dataset. In this case, the Forecast dataset can still be ray-cast in roughly 144 ms, while the dataset cannot be rendered via rasterization because the fragment lists exceed the available GPU memory. When using the Turbulence dataset with very low line opacity, GPU rasterization and voxel-based ray-casting render at 380 ms and 124 ms, respectively, demonstrating the efficiency or our approach even in this extreme situation.

5.4 Illumination effects

The secondary rays used to simulate global illumination effects like shadows and ambient occlusions are solely traversed on the first LoD level of the voxel grid. Therefore, these effects cause only a moderate increase in the render time, yet they can significantly improve the spatial perception of the rendered line structures. This is demonstrated in Fig. 19, where opaque lines are rendered with soft shadows and ambient occlusions (right).

For instance, for 100 rays sampling in a sphere with a radius of roughly 5 times the voxel size (with a step size of 1 voxel size), ambient occlusions for all 256^3 voxels can be computed in roughly 9 ms. Soft shadows can be computed in about 2 ms. The computed illumination values can be stored inside our voxel representation and only have to be updated if the scene changes.

Fig. 18 shows a comparison between reference images produced by a rasterizer using fragment linked lists (left) and images produced using our ray-casting approach using a memory efficient representation (right). In all images an attribute stored for each line segment was mapped by a transfer function to a color value. Opacity values were manually predefined. The memory efficient representation introduces some artifacts like jittered line segments and visible gradations at the voxel boundaries as attributes are not interpolated in between. Comparing the reference image and the high resolution represenation, no significant differences are observable.

In Fig. 19, lines are rendered with local illumination (left) and soft shadows in combination with ambient occlusions (right). When rendering huge amounts of semi-transparent lines, single lines cannot be distinguished any longer. Ambient occlusions help to distinguish individual curves, since less light is reaching occluded curves. Shadow effects enhance the overall structure of the dataset and indicate the relative position of curves to each other. In combination with ambient occlusions, details in shadowed regions are preserved.

6 Conclusion and Future Work

We have introduced a new approach for visualizing large 3D line sets, by using GPU ray-casting on a novel voxel-based line representation. For large sets of transparent lines we have shown significant performance improvements over rasterization-based approaches. The voxel-model gives rise to an efficient integration of local and global illumination effects.

A limitation of our approach is the lack of hardware accelerated anti-aliasing. While anti-aliasing in triangle rasterization does not have a significant effect on performance, it is expensive in ray-casting since several rays have to be traced per pixel. Another limitation is with respect to the addition of new lines. Since the voxels are densely packed in one linear array on the GPU, the whole memory needs to be restructured when new lines are added. On the other hand, since generating the voxel model is sufficiently fast, even a complete re-voxelization of all lines can be performed at high speed.

In the future we will further optimize the voxelization process so that even time-varying line sets can be visualized at interactive rates, and we will incorporate empty space skipping using the voxel hierarchy. Moreover, it will be interesting to investigate the use of adaptive line quantization strategies, for instance, based on line curvature, to generate an adaptively refined voxel model.

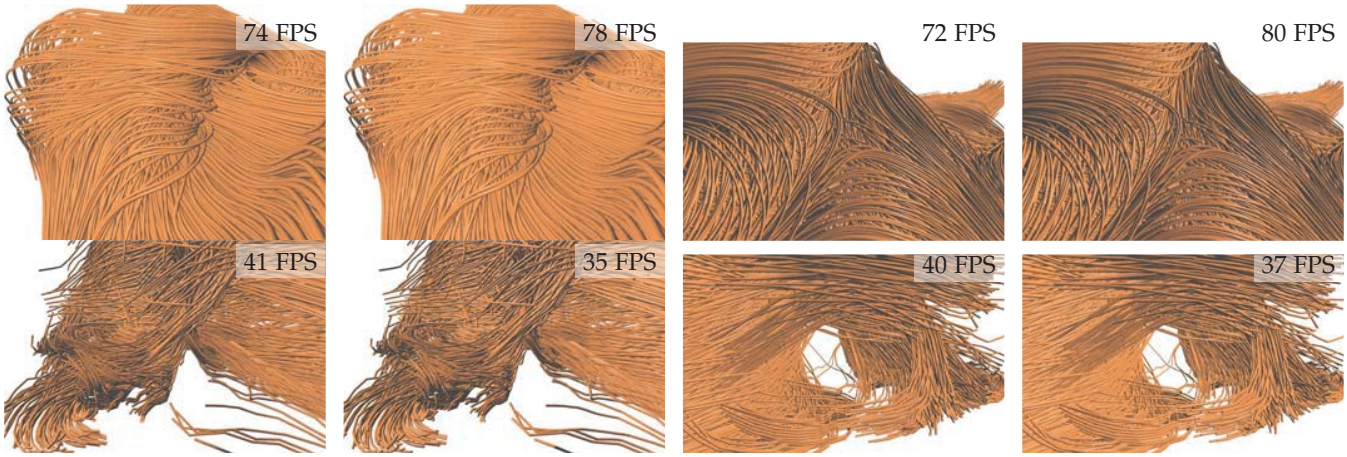

Fig. 18: Comparison between rasterization-based rendering (left) and voxel-based line raycasting using 256^3 voxels partitioned into 32^2 bins per face (right), for the Aneurysm II and the Weather Forecast dataset. The lines are rendered opaque with local illumination.

Fig. 19: Lines are rendered with local illumination (left) and soft shadows in combination with ambient occlusions (right). When rendering huge amounts of semi-transparent curves, single curves cannot be distinguished any longer. Ambient occlusions help to distinguish individual curves, since less light is reaching occluded curves. Shadow effects enhance the overall structure of the dataset and indicate the relative position of curves to each other. In combination with ambient occlusions, details in shadowed regions are preserved.

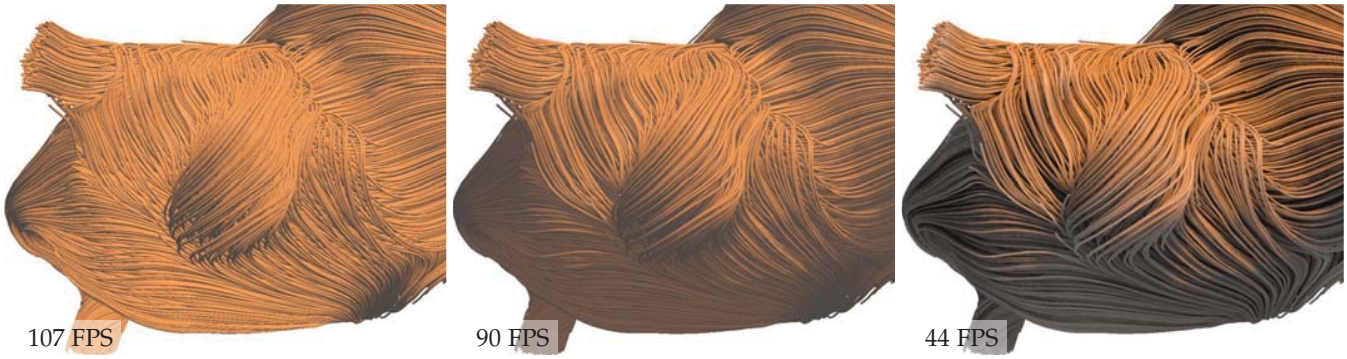

Fig. 20: First: Local illumination. Second: Soft shadows from line density values via cone-tracing. Third: Soft shadows in combination with ambient occlusion using 50 sample rays per voxel. The Aneurysm II dataset is shown, while continuously updating corresponding illumination values.

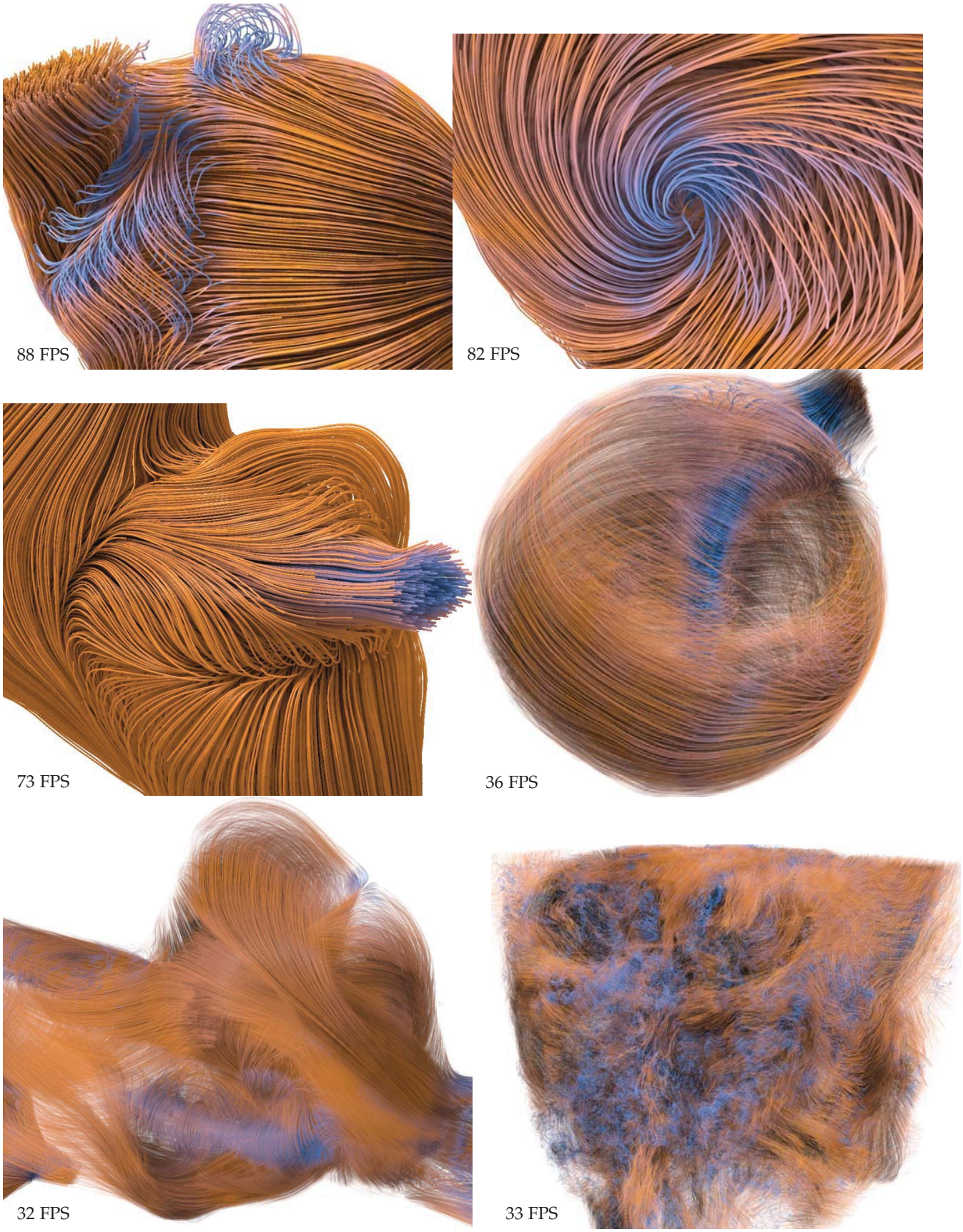

Fig. 21: Renderings of Aneurysm I (top and middle right), Aneurysm II (middle right and bottom left), and Turbulence (bottom right). Soft shadows and ambient occlusions are enhancing the depth perception. Semi opaque renderings are revealing inner structures.

TABLE 2: Performance statistics for different datasets. Rendering performance was evaluated using a constant per-line opacity value of 25 percent and 100 percent, respectively.

	Timings Semi Opaque		Timings Opaque	
Dataset	Linked List	Raycasting	Rasterization	Raycasting
Tornado	4 ms/250 fps	15 ms/67 fps	2 ms/500 fps	12 ms/83 fps
Aneurysm I	46 ms/22 fps	25 ms/40 fps	8 ms/125 fps	9 ms/111 fps
Aneurysm II	160 ms/6 fps	27 ms/37 fps	14 ms/71 fps	12 ms/83 fps
Turbulence	380 ms/3 fps	24 ms/42 fps	65 ms/15 fps	6 ms/167 fps
Forecast	overflow	23 ms/43 fps	29 ms/34 fps	4 ms/250 fps

REFERENCES

[1] N. Thibieroz and H. Gruen, "Oit and indirect illumination using dx11 linked lists," in *Game Developers Conference*, 2010.

[2] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann, "Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2009, pp. 15–22.

[3] S. Laine and T. Karras, "Efficient sparse voxel octrees," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 8, pp. 1048–1059, 2011.

[4] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann, "Interactive indirect illumination using voxel cone tracing: A preview," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2011, pp. 207–207.

[5] S. Thiedemann, N. Henrich, T. Grosch, and S. Müller, "Voxel-based global illumination," in *Symposium on Interactive 3D Graphics and Games*, 2011, pp. 103–110.

[6] D. Cohen-Or and A. Kaufman, "3d line voxelization and connectivity control," *IEEE Comput. Graph. Appl.*, vol. 17, no. 6, pp. 80–87, 1997.

[7] M. Zöckler, D. Stalling, and H.-C. Hege, "Interactive visualization of 3d-vector fields using illuminated stream lines," in *Proceedings of the 7th Conference on Visualization '96*, ser. VIS '96, 1996, pp. 107–ff.

[8] O. Mallo, R. Peikert, C. Sigg, and F. Sadlo, "Illuminated lines revisited," in *VIS 05. IEEE Visualization, 2005.*, 2005, pp. 19–26.

[9] C. Stoll, S. Gumhold, and H. P. Seidel, "Visualization with stylized line primitives," in *IEEE Visualization*, 2005.

[10] M. H. Everts, H. Bekker, J. B. T. M. Roerdink, and T. Isenberg, "Depth-dependent halos: Illustrative rendering of dense line data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1299–1306, 2009.

[11] M. H. Everts, H. Bekker, J. B. T. M. Roerdink, and T. Isenberg, "Interactive illustrative line styles and line style transfer functions for flow visualization," *CoRR*, vol. abs/1503.05787, 2015.

[12] S. W. Park, H. Yu, I. Hotz, O. Kreylos, L. Linsen, and B. Hamann, "Structure-accentuating dense flow visualization," in *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization*, 2006, pp. 163–170.

[13] A. Kuhn, N. Lindow, T. Günther, A. Wiebel, H. Theisel, and H.-C. Hege, "Trajectory density projection for vector field visualization," in *EuroVis - Short Papers 2013*, 2013, pp. 31–35.

[14] T. Günther, C. Rössl, and H. Theisel, "Opacity optimization for 3d line fields," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 120:1–120:8, 2013.

[15] O. Mishchenko and R. Crawfis, "On perception of semi-transparent streamlines for three-dimensional flow visualization," *Computer Graphics Forum*, vol. 33, no. 1, pp. 210–221, 2014.

[16] C. Everitt and L. Williams, "Interactive order-independent transparency," in *White paper available online at http://developer.nvidia.com, 2001*.

[17] J. C. Yang, J. Hensley, H. Grün, and N. Thibieroz, "Real-time concurrent linked list construction on the gpu," in *Proceedings of the 21st Eurographics Conference on Rendering*, ser. EGSR'10, 2010, pp. 1297–1304.

[18] F. Liu, M.-C. Huang, X.-H. Liu, and E.-H. Wu, "Efficient depth peeling via bucket sort," in *Proceedings of the Conference on High Performance Graphics 2009*, 2009, pp. 51–57.

[19] E. Enderton, E. Sintorn, P. Shirley, and D. Luebke, "Stochastic transparency," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 8, pp. 1036–1047, 2011.

[20] G. Schussman and K.-L. Ma, "Anisotropic volume rendering for extremely dense, thin line data," in *Proceedings of the Conference on Visualization '04*, ser. VIS '04, 2004, pp. 107–114.

[21] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker, "Ray tracing animated scenes using coherent grid traversal," *ACM Transactions on Graphics*, pp. 485–493, 2006.

[22] S. Woop, G. Marmitt, and P. Slusallek, "B-KD Trees for hardware accelerated ray tracing of dynamic scenes," in *GH '06: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*. New York, NY, USA: ACM, 2006, pp. 67–77.

[23] I. Wald, G. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Gunther, and P. Navratil, "Ospray - a cpu ray tracing framework for scientific visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 931–940, 2017.

[24] T. Aila and S. Laine, "Understanding the efficiency of ray traversal on gpus," in *Proc. High-Performance Graphics 2009*, 2009, pp. 145–149.

[25] S. Laine and T. Karras, "Efficient sparse voxel octrees," in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 2010, pp. 55–63.

[26] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "Optix: a general purpose ray tracing engine," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 66:1–66:13, 2010.

[27] I. Wald, A. Knoll, G. P. Johnson, W. Usher, V. Pascucci, and M. E. Papka, "Cpu ray tracing large particle data with balanced p-k-d trees." in *IEEE SciVis*, 2015.

[28] J. Staib, S. Grottel, and S. Gumhold, "Visualization of particle-based data with transparency and ambient occlusion," *Computer Graphics Forum*, vol. 34, no. 3, pp. 151–160, 2015.

[29] A. Kaufman, D. Cohen, and R. Yagel, "Volume graphics," *Computer*, vol. 26, no. 7, pp. 51–64, Jul. 1993.

[30] G. T. Herman and H. K. Liu, "Three-dimensional display of human organs from computed tomograms," *Computer Graphics and Image Processing*, vol. 9, no. 1, pp. 1–21, 1979.

[31] F. Reichl, M. G. Chajdas, K. Bürger, and R. Westermann, "Hybrid Sample-based Surface Rendering," in *Vision, Modeling and Visualization*, 2013, pp. 47–54.

[32] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann, "Interactive indirect illumination using voxel cone tracing," *Computer Graphics Forum (Proceedings of Pacific Graphics 2011)*, vol. 30, no. 7, 2011.

[33] D. Jönsson, E. Sundén, A. Ynnerman, and T. Ropinski, "A survey of volumetric illumination techniques for interactive volume rendering," *Computer Graphics Forum*, vol. 33, no. 1, pp. 27–51, 2014.

[34] H. Freeman and J. M. Glass, "On the quantization of line-drawing data," *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, no. 1, pp. 70–79, 1969.

[35] E. Gobbetti and F. Marton, "Far voxels: A multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 878–885, 2005.

[36] J. Danskin and P. Hanrahan, "Fast algorithms for volume ray tracing," in *Proceedings of the 1992 Workshop on Volume Visualization*, 1992, pp. 91–98.

[37] L. Bavoil, M. Sainz, and R. Dimitrov, "Image-space horizon-based ambient occlusion," in *ACM SIGGRAPH 2008 Talks*, 2008, pp. 22:1–22:1.

[38] G. Byrne, F. Mut, and J. Cebral, "Quantifying the large-scale hemodynamics of intracranial aneurysms," *Amer. J. of Neuroradiology*, vol. 35, pp. 333–338, 2014.

[39] H. Aluie, G. Eyink, V. E., S. C. K. Kanov, R. Burns, C. Meneveau, and A. Szalay, "Forced MHD turbulence data set," http://turbulence.pha.jhu.edu/docs/README-MHD.pdf, 2013 (accessed March 22, 2017).

[40] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "Optix: A general purpose ray tracing engine," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 66:1–66:13, 2010.

[41] M. Stich, H. Friedrich, and A. Dietrich, "Spatial splits in bounding volume hierarchies," in *Proceedings of the Conference on High Performance Graphics*, 2009, pp. 7–13.

Mathias Kanzler is a PhD candidate in the Computer Graphics and Visualization Group at the Technical University of Munich (TUM). He received the M.Sc. in computer science from TUM in 2015. His research interests include visualization and real-time rendering.

Marc Rautenhaus is a postdoctoral researcher in the Computer Graphics and Visualization Group at the TUM. He received the M.Sc. in atmospheric science from the University of British Columbia, Vancouver, in 2007, and the Ph.D. in computer science from TUM in 2015. Prior to joining TUM, Marc worked as a research associate at the German Aerospace Center's Institute for Atmospheric Physics. His research interests focus on the intersection of visualization and meteorology.

Rüdiger Westermann studied computer science at the Technical University Darmstadt and received the Ph.D. in computer science from the University of Dortmund, both in Germany. In 2002, he was appointed the chair of Computer Graphics and Visualization at TUM. His research interests include scalable data visualization and simulation algorithms, GPU computing, real-time rendering of large data, and uncertainty visualization.

C

Interactive Visual Exploration of Line Clusters

Mathias Kanzler, Rüdiger Westermann:
Interactive Visual Exploration of Line Clusters.
Proceedings of the Conference on Vision, Modeling, and Visualization, EG VMV '18:155-163, 2018.
doi:10.2312/vmv.20181265

Vision, Modeling, and Visualization (2018)
F. Beck, C. Dachsbacher, and F. Sadlo (Eds.)

Interactive Visual Exploration of Line Clusters

Mathias Kanzler and Rüdiger Westermann

Technical University of Munich (TUM), Germany

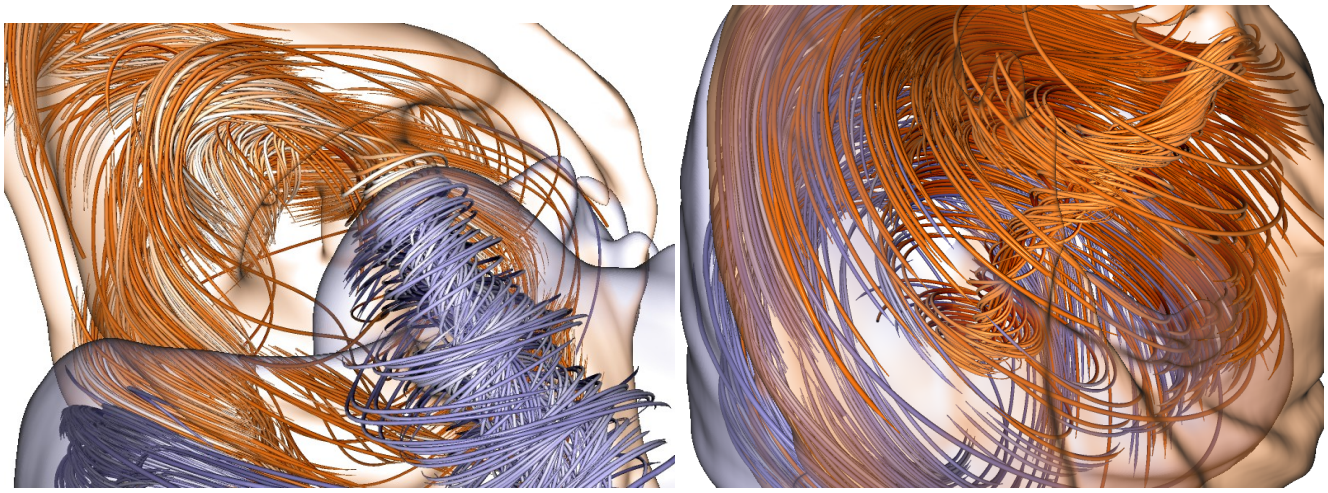

Figure 1: *Clusters of lines are analyzed by visualizing cluster-specific contextual information in the form of cluster hulls (colored surfaces), and using a measure of local line consistency within a cluster to adapt the line density automatically so that important structures are conveyed. Via brushing the user can interactively select where automated density control is applied.*

Abstract

We propose a visualization approach to interactively explore the structure of clusters of lines in 3D space. We introduce cluster consistency fields to indicate the local consistency of the lines in a cluster depending on line density and dispersion of line directions. Via brushing the user can select a focus region where lines are shown, and the consistency fields are used to automatically control the density of displayed lines according to information content. The brush is automatically continued along the gradient of the consistency field towards high information regions, or along a derived mean direction field to reveal major pathways. For a given line clustering, visualizations of cluster hulls are added to preserve context information.

1. Introduction

Clustering of lines is used in many applications [MVvW05, AT06, AMP10, YWSC12] to condense large line sets to few representative lines that expose the major geometric trends in the initial set. Here, we refer to a line as a sequence of vertices where every pair of consecutive vertices is connected by a straight line segment. While the representative lines serve as a basis for the application-specific analysis process, the distribution and spatial coverage of the lines per cluster, and the variation of these characteristics among clusters is usually not examined any further. Instead, it is quietly assumed that the lines in one cluster follow the representative line closely.

For large line sets and a reasonable number of clusters, however, the lines in one cluster usually show significant local deviations from the representative, requiring indicators of where deviations occur and how these deviations reflect in the lines' geometries. Computing a large enough set of clusters so that the per-cluster deviations are small, on the other hand, contradicts the requirement to effectively reduce the number of representatives for the upcoming analysis process. Thus, one cannot avoid significant local deviations in general, yet one can try to analyze these deviations in 3D space to better interpret the consistency of grouped elements. Such an analysis is also important to compare different clusterings

DOI: 10.2312/vmv.20181265

to each other, and to eventually shed light on specific properties of the used clustering algorithm and distance metric.

To perform a consistency analysis of a given cluster, the lines belonging to that cluster can be shown in addition to the representative line, either entirely or only to a certain percentage [YWSC12, OLK*14]. While the first approach can quickly introduce occlusions, the second one can withhold important information if the lines are thinned out randomly. Yu et al. [YWSC12] address these shortcomings by showing in addition to the representative line also lines indicating the cluster boundaries. This can reveal the spatial extent of a cluster, but it cannot effectively reveal a cluster's consistency in the interior.

Our contribution: Building upon these previous works, our approach enables an interactive visual exploration of the local consistency of a cluster of lines, in context of the clusters in the current clustering. We introduce a measure of cluster consistency in 3D space, which considers the local density of lines as well as the variance of the local line directions. Boundary surfaces in the scalar-valued consistency field are used to indicate a cluster's shape.

Starting with a context view showing cluster boundaries and representatives, we let the user brush a region where lines should be added. By using a hierarchy of line clusters [KFW16], the line density in the brushed region is controlled to emphasize regions where lines are strongly diverging, at the same time showing only few representative lines in regions where lines behave similar. The object-space cluster consistency is used to determine the line density. To further guide the user towards regions of low consistency, the brush is continued automatically along the gradient of the consistency field. Alternatively, the brush can be continued along the major trends in the line set by following bundles of similar lines.

Our specific contributions are:

- *Cluster consistency fields* as spatial indicators for the line consistency in a cluster.
- The use of cluster consistency fields to reveal the most representative lines in a brushed focus region.
- A method to continue a user-selected brush along major pathways and towards regions of high information content.

We demonstrate our approach by visualizing a number of real-world examples comprised of large line sets. Some of these data sets are shown in Fig. 1, hinting towards the specific visualizations targeted in this work. Equipped with a cluster panel that shows an abstract view of a given line clustering, the user can interactively select clusters based on their consistency, and compare clusters using their hulls and overlaps.

2. Related Work

Our approach uses a hierarchical clustering of a line set. Let us refer to the book by Jain [Jai10] for an exhaustive summary of available clustering algorithms. An overview of similarity measures for lines is given in the comparative study by Zhang et al. [ZHT06]. Oeltze and co-workers [OLK*14] evaluate clustering approaches for streamlines using geometry-based similarity measures. Different clustering approaches and similarity measures for fiber tracts in diffusion tensor imaging data have been evaluated by Moberts et al. [MVvW05].

Yu et al. [YWSC12] use hierarchical clustering to generate a small representative set of streamlines in a flow field. They further select streamlines close to the cluster boundaries to convey the spatial extent of computed clusters. Kanzler et al. [KFW16] use a line cluster hierarchy for view-dependent line density control. Every segment of a line determines a level in the pre-computed line hierarchy depending on a screen-space importance measure, and only if the line is the representative line at this level the segment is drawn. For the construction of a line cluster hierarchy we follow the work by Kanzler et al., which uses a variant of Agglomerative Hierarchical Clustering (AHC) with single linkage and graph matching to construct a fully balanced cluster tree.

Both aforementioned approaches try to find the minimum number of lines that represent the important structures in a given line set. This process can be performed either via importance- or similarity-based criteria in object-space, or in screen-space by selecting the rendered lines dynamically on a frame-to-frame basis. Screen-space approaches determine for each new view the subset of lines to be rendered so that occlusions are reduced and more important lines are favored over less important ones [MCHM10, LMSC11, MWS13]. Indicators for the amount of occlusion in the rendered images are based on the "overdraw", i.e., the number of projected line points per pixel [MCHM10, MWS13], or the maximum projected entropy per pixel [LMSC11]. View-dependent opacity control was introduced by Guenther et al. [GRT13] to fade out those parts of foreground lines that occlude more important ones.

For the selection of representative lines in object-space a number of different criteria have been proposed, for instance, based on the line density [TB96, MHHI98, MTHG03, SHH*07], the line distribution [JL97, LMG06, LHS08, SLCZ09, RPP*09], or the coverage of specific flow or line features [VKP00, YKP05, CCK07, MJL*12]. Behrendt et al. [BBB*18] suggest an explorative approach to select lines based on surface features. The explorative selection of point clouds can be accomplished by encircling a target location as described by Yu et al. [YEII12]. An interactive focus and context approach using a 2D lens to analyze flow structures is described by Gasteiger et al. [GNBP11] and Fuhrmann et al. [FG98]. While Jackson et al. [JCK12] are using a dedicated haptic device to explore flow structures in 3D, our method is designed for standard input devices (mouse and keyboard). Our proposed consistency measure to control the line density is related to the measure of information entropy that was introduced by Xu et al. [XLS10]. They use the local directional variation of a given vector field as entropy measure, and generate an entropy field that is then used to guide the placement of streamlines. We introduce a similar measure using the local directional variations of lines in a cluster, and we further extend this measure to also consider the line density.

3. Method Overview

The input of our method consists of a set of lines, e.g., streamlines, where each line is represented by a sequence of vertices. If no initial clustering is given, we use the mean-of-closest-point dis-

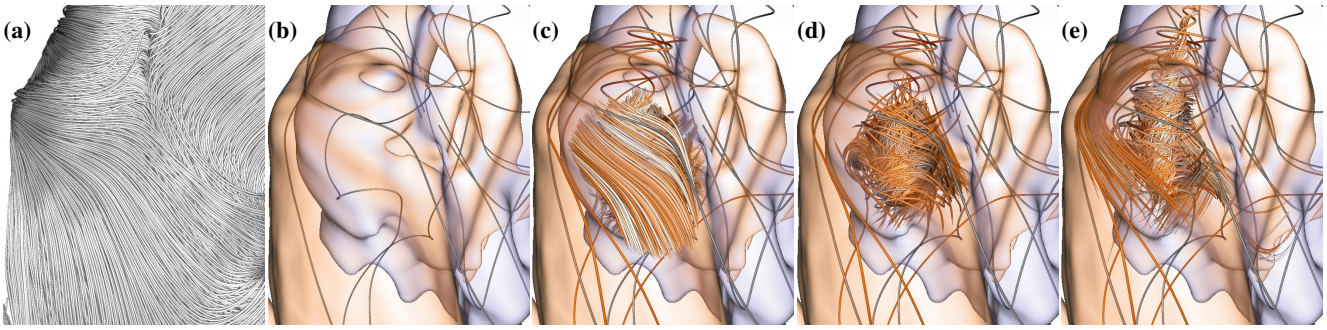

Figure 2: *(a) Initial line set. (b) Representative lines of 8 clusters and hulls of two clusters. (c) All lines going through a brushed focus region. (d) View-dependent line density control using local consistency. (e) Automatic refinement and continuation of focus region towards important structures in the surrounding.*

tance [CGG04] as distance metric and compute a balanced cluster tree using graph matching [KFW16].

Every node in the computed cluster tree represents a cluster of lines that are grouped together based on the used distance metric. For selecting a single representative line at each node, we perform a top-down sweep through the tree as follows: We traverse the tree in breadth-first order, and whenever we encounter a node that has not been visited we determine the representative line, i.e., the line that has the smallest average mean-of-closest-point distance to all lines in the cluster at that node. This line becomes the representative line for every cluster at the nodes along the path from the current node to the leaf node where the line is the only cluster member. In this way we build a hierarchy of representatives which explain as much as possible of the cluster variation at the highest (coarsest) levels. In addition, the hierarchy is nested in that at every inner node one of the child nodes has the same representative than that node. This enables to keep the current representatives and progressively add the new ones when going from one level to the next finer one. Fig. 2 (a) and (b), respectively, show a line set and the 8 representative lines (as well as two cluster hulls) at level three of the computed cluster hierarchy.

For every cluster down to a user-selected level a cluster consistency field (CCF) is pre-computed (c.f. Section 4). If at run-time a CCF is requested that has not been computed, it is generated in turn within less than a second on the GPU. A CCF is similar to a visitation volume [Jon08, BCM*08, BFMW12], which is generated by rasterizing lines into a 3D voxel grid and counting the number of lines going through every voxel. Level sets in this field can then be visualized to show the spatial extend of the lines in one cluster, i.e., the cluster hull (see Fig. 2 (b)), and multiple hulls can be displayed concurrently to enable a comparative cluster analysis. In addition to line density, we also store a measure of the directional variance of the lines going through a voxel.

While cluster hulls provide context information about a cluster's shape and spatial extent, they do not, in general, show the lines' geometries. To select a region in 3D space where lines are shown, the user can brush a region on a cluster hull, and this region is extruded to 3D in a specific way (c.f. Section 5). Via brushing the user increases a visibility parameter in the selected region, requesting additional lines besides the cluster representatives. If all lines passing through the selected region are shown concurrently, however, occlusions are quickly introduced and important structures are hidden (see Fig. 2 (c)).

To adapt the density of displayed lines in the selected focus region, we employ the CCF as a measure of information content (c.f. Section 5), as shown in Fig. 2 (d). Furthermore, since regions conveying high information can be missed when brushing manually, the visibility is transported automatically along the gradient in the CCF towards regions of low consistency (Fig. 2 (e)), or along bundles with low directional variance to reveal major trends in the line set. In either case, line density control is active to focus only on the most relevant lines.

4. Cluster Consistency Fields

CCFs are computed on the GPU using rasterization of lines into a Cartesian 3D voxel grid. Initially, we compute the resolution $r_x \times r_y \times r_z$ of the voxel grid into which the lines are voxelized. If the resolution is too high, a voxel carries merely information about one single line passing through it, yet if it is too low, the CCF cannot serve the purpose of a local indicator of consistency. In our implementation we link the average distance of the cluster representative to all lines in the cluster to the side length of cube-shaped voxels. Our experiments have shown that this choice gives a good balance between locality and information content. The grid size is set so that the aspect ratio of the bounding box of the lines per cluster is maintained. The vertex coordinates v_i are then transformed to local object coordinates in the range from $(0,0,0)$ to (r_x, r_y, r_z).

The voxel values can be optionally filtered using a low-pass filter with adjustable extent. In this way, high frequencies in whatever values are sampled can be removed, and a smooth consistency representation is obtained. In particular, if a feature is present in one voxel the smoothing process distributes this information into the surrounding region. In combination with our proposed brush-based focus visualization this leads to an improved continuation of features beyond the brush extent. For this reason we store a separate low-pass filtered version for every CCF we generate.

For each line in parallel, and starting with the first vertex, every pair of vertices v_i and v_{i+1} is processed consecutively. A line

through v_i and v_{i+1} is clipped against the voxel boundaries, via line-face intersection tests in the order of their occurrence from v_i to v_{i+1}. If v_i and v_{i+1} are located in the same voxel, no new intersection point is generated. This gives a sequence of voxel-face intersections, and every pair of consecutive intersections represents a line segment that enters into a voxel and exits that voxel. From all line segments that are generated for a single voxel, a number of different quantities are derived to generate different variants of CCFs. These variants are explained in the following.

4.1. Line density fields

For each line segment, the real length of the line from the entry to the exit point is computed and added into the CCF. Since in general the first and last vertex of a line do not lie on a face, we consider the length from the first vertex to the first face intersection and from the last face intersection to the last vertex in these cases.

Adding a length value to the CCF means to add this value to a counter at the voxel the line is currently passing through. Note that this is different to the construction of visitation volumes, where every segment adds the same contribution to a voxel regardless of its length. Our approach, in contrast, keeps track of the voxels' fill rates to obtain a more accurate measure of the line density per voxel.

The line density distribution within one cluster provides information about where and how many lines come close together (see Fig. 3 (left)). The level set to a threshold just above zero gives an approximate hull enclosing all lines. Regions in which only few isolated lines occur appear as low value regions which can be filtered out by slightly increasing the threshold. Then again, these regions can also be emphasized to show outliers which have been assigned to a cluster even though at least in some region they deviate strongly from the rest of the lines in that cluster. Fig. 3 (right) shows the cluster hulls that were rendered as level sets in multiple cluster density fields.

Level sets in the line density field are visualized using direct isosurface raycasting. For shading, gradients are calculated on-the-fly using central differences. The cluster hulls should be as transparent as possible without failing to communicate the outer shape. We therefore adapt the opacity of the isosurface depending on the angle between the surface normal and view direction [HGH*10]. In this way, the silhouettes of cluster hulls are enhanced while a non-obscured view into the cluster interior is provided otherwise.

4.2. Directional variance fields

Nearby lines in one single cluster that follow the same direction are in most cases less interesting than lines following different directions, they even carry redundant information and can thus be condensed. Xu et al. [XLS10] have motivated this statement in the context of vector fields via information entropy as a measure of how much information is carried by a region, depending on the directional uniformity of the vectors in this region. Measuring the variation of lines per volumetric unit allows to directly communicate locations which are potentially relevant for the understanding of the underlying line structure, at the same time determining regions where less lines than available can be shown. In complex line

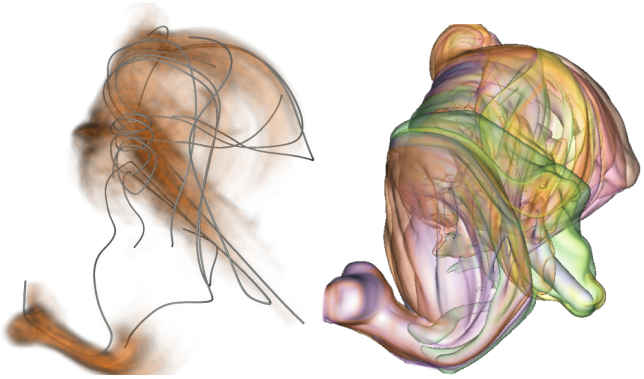

Figure 3: *Left: Representative lines of 8 clusters are shown in combination with summed up density of all clusters mapped linearly to opacity. Right: Hulls of 8 clusters using line density fields.*

sets, like turbulent structures, lines can deviate strongly in direction. By solely filtering line segments of low directional variance, the resulting image can still suffer from visual clutter. A measure of variation is therefore used as an additional filtering criterion for the visualization algorithm, and we use it in particular in combination with interactive brushing to balance the number of shown lines with respect to the information they carry. Furthermore, the measure is used to guide towards regions of high information content which have not been selected by the user. Similar in spirit to the approach by Xu et al. [XLS10], we calculate per voxel the directional variance of all line segments passing through that voxel. Let the line segments in a certain voxel be given as a set of direction vectors $\mathbf{d}_1,...,\mathbf{d}_n$. The directional variance, weighted by the length of each line segment, is defined as following:

$$\sigma(\mathbf{d}_1,...,\mathbf{d}_n) = 1 - ||\frac{\sum_i \mathbf{d}_i}{\sum_i ||\mathbf{d}_i||}|| \quad (1)$$

If the directional variance is high, σ is close to one, while it is zero if all directions are equal. This is demonstrated in Fig. 4, where the directional variance around the core of a tornado is shown. As can be seen, the variance is strongly increasing towards the core, with the magnitude of the gradient of the variance starting to grow strongly with decreasing distance from the core.

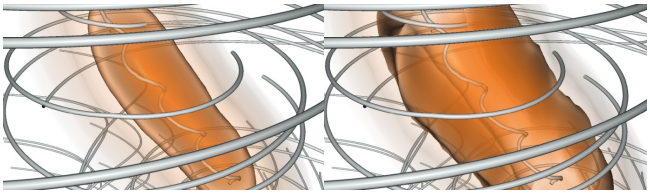

Figure 4: *Left: Cluster variance field of the Tornado data set. Color and opacity mapping is as for cluster density. Right: Magnitude of the gradient of the cluster variance field. The maximum gradients are located around the core of the tornado.*

4.3. Mean direction fields

The mean direction volume of a cluster stores at every voxel one representative average line segment; by averaging separately the start points and end points of all flow-aligned line segments in a voxel. This gives two average points which define the average line direction that is stored per voxel. The mean direction volume is not visualized directly but used for accessing the main flow direction per voxel in constant time.

5. Line Density Control via Brushing

We use the pre-computed CCFs to visualize cluster context information, i.e., cluster hulls, and to adapt the density of lines that are shown in a region that is brushed by the user. Cluster boundaries are visualized as isocontours in the cluster density field, and they support the user in identifying the boundaries of the brushable zone. At the same time they hint towards prominent regions, for instance, regions where the consistency of lines is extraordinarily low or high. This forms the context in which the user queries additional lines, and by this, analyses interactively the internal cluster structure. The user can request a more detailed representation of the line set by interactively using a brush in screen-space. Therefore, the 2D brush region is extruded to 3D along the viewing direction, defining a 3D brush volume in which the visibility is increased. Per default, the visibility is zero everywhere, meaning that only the cluster representatives are visualized. Where the visibility is increased, additional lines are shown, yet their density is controlled by the content of the CCFs as well as occlusion information that is computed for that location. The hierarchical decomposition of a selected cluster is utilized in combination with the visibility values to adapt the line density so that a sparse set of lines is shown where lines are close together, yet enough lines are shown to emphasize the important trends in regions where the cluster consistency is low.

Whenever the brush is moved a visibility volume is filled with visibility values $\in (0,1)$. Voxels in the brush volume get assigned a high visibility value. Simultaneously, via raycasting from every voxel center towards the viewplane in the cluster density field, the amount of occlusion, i.e., the accumulated density, a voxel receives due to lines in front of it is estimated. Voxels in the visibility volume that are not in the brush volume keep their values from previous frames, yet these values are continuously faded out over time if not queried once again. This allows one to move the camera while brushing without losing the previously brushed location, a mechanism that is demonstrated in Fig. 5, where the brush is applied and frozen before moving the camera to another viewpoint.

5.1. Local line density control

Based on the cluster tree every line gets assigned a *level-id*, which indicates the lowest level of the tree at which the line is chosen as representative. Note that for every line there is at least one level where this line is a representative line, at latest at the leaf node where the line is the only cluster member. The level-id is used during rendering to decide whether a line segment should be rendered in the selected focus region or not. Note that level-ids are divided by the depth of the cluster tree to obtain values in (0,1).

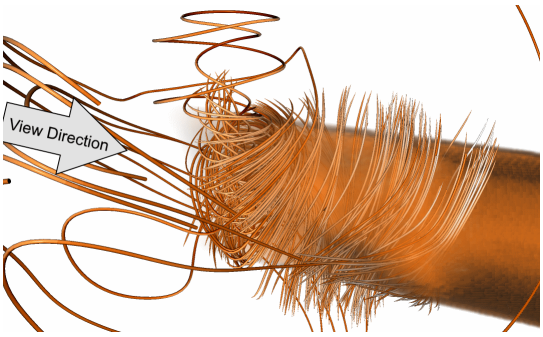

Figure 5: *The visibility volume is visualized using raycasting for demonstration purposes. The camera was originally positioned at the top left while using the brush. After brushing, the visibility values were frozen and the camera position was changed. With accumulated density samples, the visibility values are higher.*

3D line sets can be rendered efficiently via the rasterization-based rendering pipeline on the GPU, by constructing for every line segment a tube on-the-fly in a geometry shader resulting in 8×2 triangles per line segment. Every vertex that is generated when rendering a tube can access the visibility value at its object-space location, and this value can then be used to modify the rendering style of the tube segment or to discard the segment entirely. Every vertex compares the visibility value with the level-id of the line it belongs to. Only if the visibility value is larger than the level-id, the vertex is rendered, and it is discarded otherwise. In this way, the line density can be adapted automatically to the local object-space visibility, i.e., in regions of high visibility far more lines are rendered than in low visibility regions. In principle, also other rendering parameters can be controlled by in this way, for instance, opacity or color.

After all visibility values are resolved to per-vertex properties, the actual rendering is performed in two passes: In the first pass all lines are rasterized into a texture including their screen-space depth. We are using an intermediate vertex property for fading lines in or out to avoid sudden changes at the rendered image while moving the brush. Fading in or out is done by modifying the radius of the segment. The second pass performs raycasting and blends the previously rendered lines with the cluster hulls.

Visibility values for each voxel (x,y,z) in the brush volume (a cone with its center axis going through the camera position and the selected point in screen-space) are set by a compute shader according to

$$V_{x,y,z} = 1 - min(max(((\lambda + r) - p_d * D_{Acc} + p_v * Var), 0), 1) \quad (2)$$

With this formula, we provide a number of options to accentuate specific structures in the line set: λ defines a threshold that has to be exceeded before visibility values are set. By setting λ to a high value, more density values along the ray have to be accumulated or a region of high directional variance has to be passed, before visibility values are increased. Increasing λ enables a deeper insight into the data set. $r \in [0,1]$ is the distance of the ray to the center of the brush in screen-space. With this, threshold λ is increased with

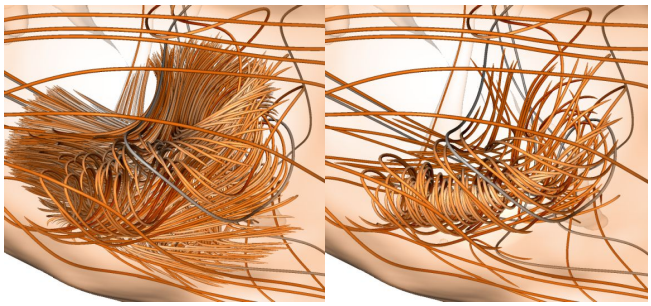

Figure 6: *By modifying parameters of the brush, different structures of the Aneurysm II data set are brought out. Left: Lines which are usually covered can be inspected. Right: Lines in regions of high variance are emphasized.*

increasing distance to the center of the brush, leading to a reduced density of lines at the border of the brush. D_{Acc} is the accumulated density along each ray from the camera to the corresponding voxel. The visibility values are increased with accumulated density, to not only show the nearest lines but to allow a placement of visibility values behind a number of lines as can be seen in Fig. 6 (left). The impact of this quantity is controlled by parameter p_d. Regions showing a high directional variance *Var* can be explicitly emphasized by increasing parameter p_v, as can be seen in Fig. 6 (right).

5.2. Brush guidance

By relying solely on the screen-space location of the used brush, the user can easily miss significant structures which are not in the brush volume, yet located very close to the brushed region in 3D space. In order to guide the user towards such structures, we propose a mechanism to automatically continue the selected brush volume anisotropically into their direction. This is achieved by letting visibility values being transported into the surrounding, either along gradient vectors in the CCF or lines passing through the selected cluster as illustrated in Fig. 7:

Mean Direction The visibility values are transported along the mean direction of lines, which is encoded in the mean direction field (see Fig. 7 (b)). By continuing the visibility values along the mean direction, crossing lines can be detected and the main path can be identified.

Variability Gradient By using the gradient field of the directional variance field, and transporting the visibility values along the gradient directions, nearby locations with high visibility will be revealed. This effect is shown in Fig. 7 (a), where the transport is towards the center of the vortex.

A mixture of both transport mechanisms is possible, by using them in combination but with different priorities. For instance, in the current implementation we use a linear blend between the gradient direction and the mean line direction according to the gradient magnitude. Altogether, visibility values are increased in the brush volume, transported into the dominating direction, and faded out over time. Thus, we can control the distance the visibility values can be continued via a global parameter that controls the life time of continuation.

The transport of visibility values is performed per frame in a compute shader. For each voxel, the visibility value and the vector indicating the location to which this value should be continued is computed. Instead of writing the visibility value to that location, we gather the visibility value from the location indicated by the inverse vector via tri-linear interpolation in the field of current visibility values.

6. Results

In the following, we show the quality of our method based on four data sets. All images were generated on a standard desktop computer (Intel 6x3.50 GHz processor, 32 GB RAM, and an NVIDIA GeForce GTX 1070 Ti graphics card). We have tested our method with following data sets:

Tornado 10000 streamlines were randomly seeded in a 256^3 vector field representing a tornado (Fig. 13).

Aneurysm I and II 4070 (Fig. 1 (right)) and 9200 streamlines (Fig. 11) describing the blood flow in two different aneurysms. [BMC14].

Turbulence 10000 streamlines were randomly seeded in a simulated turbulent vector field of size 1024^3 [AEE*16] (Fig. 12).

We compare our approach to importance-based line density control as proposed by Kanzler et al. [KFW16]. The lines of one cluster were extracted and set as input. The curvatures along the lines were mapped to importance values as proposed by the authors. The shading of lines was adjusted to focus on the distribution of line density. In Fig. 8 (left), we can see the overall density of lines adapted in a way to have an overview of the data set and to have an unobscured view towards the main vortex located at the left. With our brush-based approach in combination with refinement and transport of visibility values, the vortex at the left can be inspected (Fig. 8 (center)). By further inspecting the data set, another vortex at the bottom is revealed, which is not visible when using the importance criterion. The vortex is stretched resulting in low curvature and hence

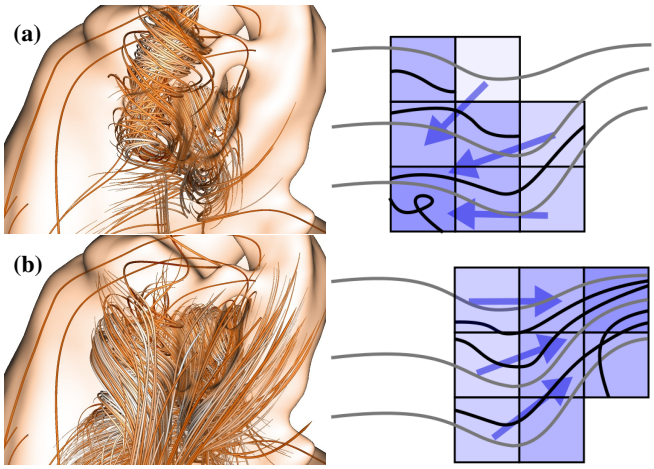

Figure 7: *Visibility values (blue) are controlling the line density. (a) Visibility values are transported to areas of higher variation. (b) Visibility values are transported along mean direction.*

Figure 8: *Comparison of importance-based density control as proposed by Kanzler et al. [KFW16] (left) and our brush based visualization (center and right). The vortex at the bottom of the image is not covered by Kanzler et al. but can be brought out using our brush (right).*

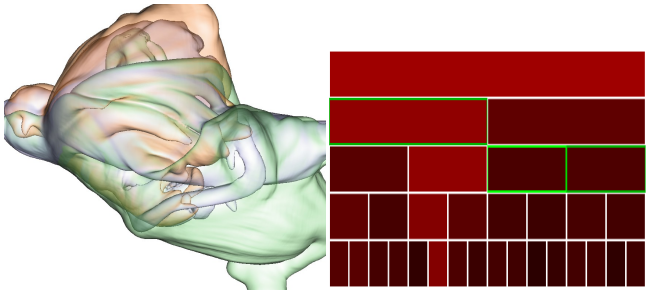

Figure 9: *On the left, 3 cluster hulls are visualized, which are selected in the cluster tree panel on the right.*

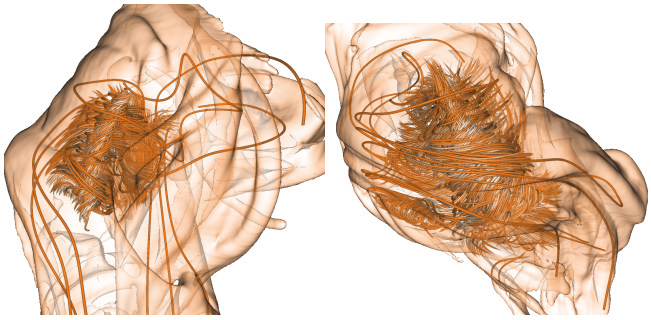

Figure 10: *The brush is used to add details by adding lines. Visibility values are placed by using primarily accumulated density and are extended in the direction of higher directional variance.*

a low importance value. Since our approach works in object-space, 3D flow structures are better preserved than the approach using a screen-space measure to adapt the line density.

The typical overall frame time (22 ms) includes line rendering (3 ms), rendering of cluster hulls using raycasting (13 ms), brushing at the focus region (4 ms), and visibility transport (2 ms) for a visibility volume of size 256^3. In all of our experiments the frame rate was consistently above 20 frames per second.

In the following, we describe a typical workflow using our method for the Aneurysm II data set. At first, the user selects a number of clusters for which the hulls are shown. This provides an overview of the clusters's extends and overlaps. Clusters can then be selected and further split into multiple clusters. A cluster panel represents the first levels of the cluster tree with its root at the top as shown in Fig. 9 on the right, and acts as control interface. The user is able to select and deselect different clusters at different levels, while the visual representation of the corresponding cluster hulls is updated immediately. The cluster panel communicates an estimate about the consistency of the cluster by using a mapping to a color gradient from black to red, i.e., from higher to lower consistency. The measure for consistency in this panel is calculated by summing up the distances of all lines of the cluster to the centroid of the cluster. By using this measure, a higher sum of distances indicates a cluster which might be a candidate for splitting up into two distinct clusters. After refining the clusters until a desired level, one can focus on one or two clusters by hiding the other clusters. To investigate one cluster in detail, the brush is applied as can be seen in Fig. 10. The vortex cores are now clearly visible while the hull in combination with auxiliary lines allows an estimate about the surroundings of the brushed location. By adapting the radius of the brush, the density of lines is increased in a larger region. By the automated continuation of the brush volume, additional structures become visible.

The Aneurysm II data set consists of long lines, resulting in cluster hulls that overlap in many regions. To further study at which location a cluster shares a feature with a nearby cluster, a second cluster can be enabled. The brush can now be applied to both clusters, revealing information about the underlying clustering. By transporting visibility values along the mean direction field and towards regions of high directional variance, cohesive structures can be identified: Fig. 11 shows the boundary between the vortex and the other cluster. In this example lines were not faded out to include the continuation of the lines forming the vortex.

Placing visibility values at regions of high directional variance by increasing parameter p_{var} and extending the selection along the lines, vortices embedded in turbulent structures can be revealed as can be seen in Fig. 12. By increasing λ, the coverage of lines located at the background is reduced. Combining this setting with a

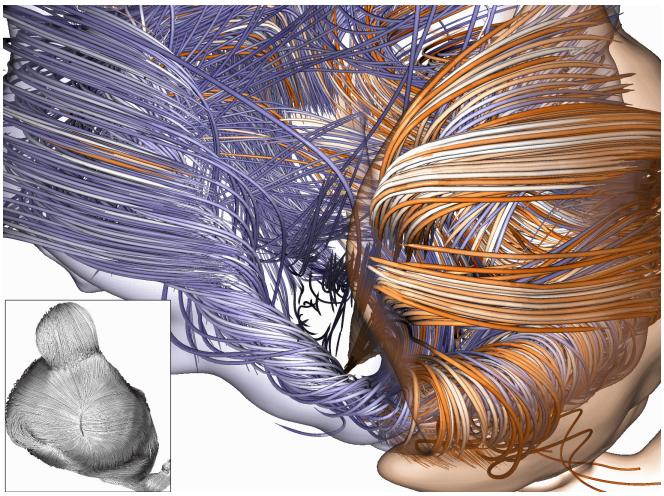

Figure 11: *The boundaries of two clusters are inspected by brushing. The vortex core at the bottom is split into two clusters. While the left part is consisting only of lines of the blue cluster, lines of the blue cluster are also interweaved in the orange cluster. Bottom left: All lines of the Aneurysm II data set.*

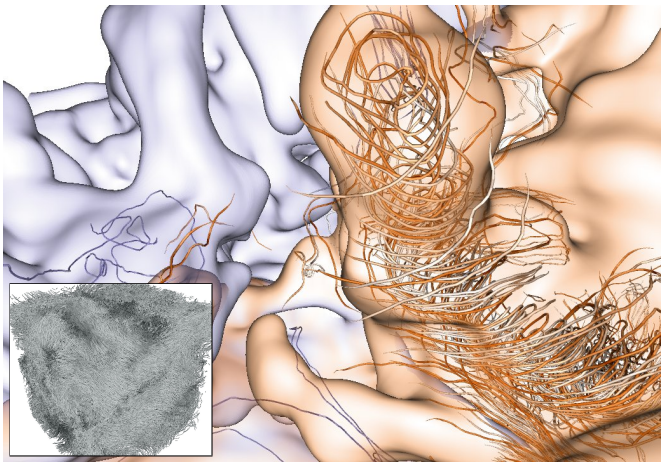

Figure 12: *Two clusters of the turbulence data set are inspected. A vortex structure is revealed by placing visibility values at regions of high directional variance. Bottom left: All lines of the Turbulence data set.*

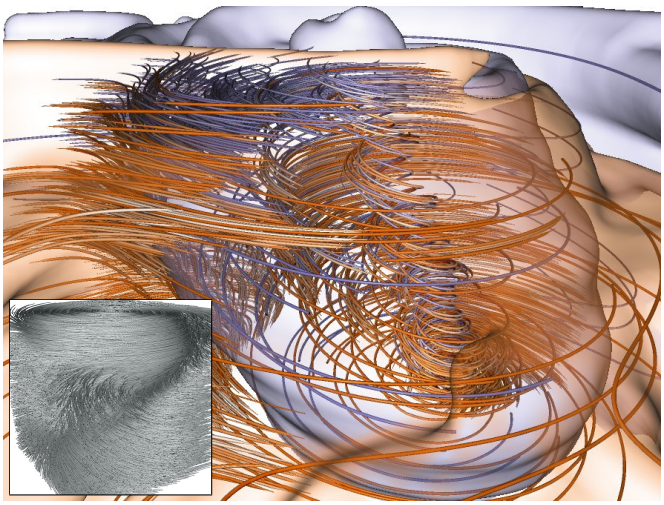

Figure 13: *Boundaries of two clusters are inspected by enhancing regions with high accumulated density inside the brush region. Transport of visibility values is showing the core of the tornado. Bottom left: All lines of the Tornado data set.*

visibility transport along the directional variance gradient, vortices in front of background lines are still emphasized as can be seen in Fig. 13 and Fig. 1 (right). The core vortex of the aneurysm is dominated by lines of one cluster while we can see a mixing of clusters at the core of the tornado.

7. Conclusion

In this paper, we have presented a novel approach to interactively explore clusters of lines via consistency-guided visualization techniques on the GPU. We have introduced cluster consistency fields and employed them to apply focus+context visualization techniques. We have shown the combination of cluster hulls with interactive and automatic techniques to select focus regions and adapt the line density to cluster consistency. This enables an in-depth comparison of cluster overlap regions and the line consistency of a selected cluster.

In future work, we will compare different clustering methods by matching similar clusters and simultaneously visualizing matches between two clustering methods. Specific characteristics of clustering methods can be studied directly on a given data set. We will further examine time-dependent data sets by finding a transition between brushed regions at different time steps. The development and movement of features is of interest in this case. Guiding a brush by exploiting time-dependent information can further enhance the understanding of the data set.

References

[AEE*16] ALUIE H., EYINK G., E. V., KANOV S. C. K., BURNS R., MENEVEAU C., SZALAY A.: Forced MHD turbulence data set. http://turbulence.pha.jhu.edu/docs/README-MHD.pdf, 2013 (accessed May 13, 2016). 6

[AMP10] ATEV S., MILLER G., PAPANIKOLOPOULOS N. P.: Clustering of vehicle trajectories. *IEEE Transactions on Intelligent Transportation Systems 11*, 3 (Sept 2010), 647–657. 1

[AT06] ANTONINI G., THIRAN J. P.: Counting pedestrians in video sequences using trajectory clustering. *IEEE Transactions on Circuits and Systems for Video Technology 16*, 8 (Aug 2006), 1008–1020. 1

[BBB*18] BEHRENDT B., BERG P., BEUING O., PREIM B., SAALFELD S.: Explorative blood flow visualization using dynamic line filtering based on surface features. 2

[BCM*08] BERMAN J. I., CHUNG S., MUKHERJEE P., HESS C. P., HAN E. T., HENRY R. G.: Probabilistic streamline q-ball tractography using the residual bootstrap. *NeuroImage 39*, 1 (2008), 215 – 222. 3

[BFMW12] BÜRGER K., FRAEDRICH R., MERHOF D., WESTERMANN R.: Instant visitation maps for interactive visualization of uncertain particle trajectories, 2012. 3

[BMC14] BYRNE G., MUT F., CEBRAL J.: Quantifying the large-scale hemodynamics of intracranial aneurysms. *Amer. J. of Neuroradiology 35* (2014), 333–338. 6

[CCK07] CHEN Y., COHEN J., KROLIK J.: Similarity-guided streamline placement with error evaluation. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1448–1455. 2

[CGG04] 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* (April 2004), pp. 344–347 Vol. 1. 3

[FG98] FUHRMANN A., GRÖLLER E.: Real-time techniques for 3d flow visualization. In *Proceedings of the conference on Visualization'98* (1998), IEEE Computer Society Press, pp. 305–312. 2

[GNBP11] GASTEIGER R., NEUGEBAUER M., BEUING O., PREIM B.: The flowlens: A focus-and-context visualization approach for exploration of blood flow in cerebral aneurysms. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (Dec. 2011), 2183–2192. 2

[GRT13] GÜNTHER T., RÖSSL C., THEISEL H.: Opacity optimization for 3D line fields. *Proc. ACM SIGGRAPH 32*, 4 (2013), 120. 2

[HGH*10] HUMMEL M., GARTH C., HAMANN B., HAGEN H., JOY K. I.: Iris: Illustrative rendering for integral surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (Nov 2010), 1319–1328. 4

[Jai10] JAIN A. K.: Data clustering: 50 years beyond k-means. *Pattern Recogn. Lett. 31*, 8 (2010), 651–666. 2

[JCK12] JACKSON B., COFFEY D., KEEFE D. F.: Force Brushes: Progressive Data-Driven Haptic Selection and Filtering for Multi-Variate Flow Visualizations. In *EuroVis - Short Papers* (2012), Meyer M., Weinkaufs T., (Eds.), The Eurographics Association. 2

[JL97] JOBARD B., LEFER W.: Creating evenly-spaced streamlines of arbitrary density. In *Proc. Eurographics Workshop on Visualization in Scientific Computing*. 1997, pp. 43–55. 2

[Jon08] JONES D. K.: Tractography gone wild: Probabilistic fibre tracking using the wild bootstrap with diffusion tensor mri. *IEEE Transactions on Medical Imaging 27*, 9 (Sept 2008), 1268–1274. 3

[KFW16] KANZLER M., FERSTL F., WESTERMANN R.: Line density control in screen-space via balanced line hierarchies. *Computers & Graphics 61* (2016), 29 – 39. 2, 3, 6, 7

[LHS08] LI L., HSIEH H.-H., SHEN H.-W.: Illustrative streamline placement and visualization. In *Proc. IEEE PacificVis* (2008), pp. 79–86. 2

[LMG06] LIU Z., MOORHEAD R., GRONER J.: An advanced evenly-spaced streamline placement algorithm. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 965–972. 2

[LMSC11] LEE T.-Y., MISHCHENKO O., SHEN H.-W., CRAWFIS R.: View point evaluation and streamline filtering for flow visualization. In *Proc. IEEE PacificVis* (2011), pp. 83–90. 2

[MCHM10] MARCHESIN S., CHEN C.-K., HO C., MA K.-L.: View-dependent streamlines for 3D vector fields. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1578–1586. 2

[MHHI98] MAO X., HATANAKA Y., HIGASHIDA H., IMAMIYA A.: Image-guided streamline placement on curvilinear grid surfaces. In *Proc. IEEE Visualization* (1998), pp. 135–142. 2

[MJL*12] 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 19*, 8 (2012), 1342–1353. 2

[MTHG03] 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, 2003), ACM, pp. 213–222. 2

[MVvW05] MOBERTS B., VILANOVA A., VAN WIJK J.: Evaluation of fiber clustering methods for diffusion tensor imaging. In *Proc. IEEE Visualization* (Oct 2005), pp. 65–72. 1, 2

[MWS13] MA J., WANG C., SHENE C.-K.: Coherent view-dependent streamline selection for importance-driven flow visualization. *Proc. SPIE 8654, Visualization and Data Analysis* (2013). 2

[OLK*14] 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 20*, 5 (2014), 686–701. 2

[RPP*09] ROSANWO O., PETZ C., PROHASKA S., HEGE H.-C., HOTZ I.: Dual streamline seeding. *Proc. IEEE PacificVis 0* (2009), 9–16. 2

[SHH*07] 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), pp. 227–234. 2

[SLCZ09] SPENCER B., LARAMEE R. S., CHEN G., ZHANG E.: Evenly spaced streamlines for surfaces: An image-based approach. *Computer Graphics Forum 28*, 6 (2009), 1618–1631. 2

[TB96] TURK G., BANKS D.: Image-guided streamline placement. In *Proc. ACM SIGGRAPH* (1996), pp. 453–460. 2

[VKP00] VERMA V., KAO D., PANG A.: A flow-guided streamline seeding strategy. In *Proc. IEEE Visualization* (2000), pp. 163–170. 2

[XLS10] XU L., LEE T.-Y., SHEN H.-W.: An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1216–1224. 2, 4

[YEII12] YU L., EFSTATHIOU K., ISENBERG P., ISENBERG T.: Efficient structure-aware selection techniques for 3D point cloud visualizations with 2DOF input. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (Dec. 2012), 2245–2254. 2

[YKP05] YE X., KAO D., PANG A.: Strategy for seeding 3D streamlines. In *Proc. IEEE Visualization 2005* (2005), pp. 471–478. 2

[YWSC12] YU H., WANG C., SHENE C.-K., CHEN J. H.: Hierarchical streamline bundles. *IEEE Transactions on Visualization and Computer Graphics 18*, 8 (2012), 1353–1367. 1, 2

[ZHT06] ZHANG Z., HUANG K., TAN T.: Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. In *Proc. International Conference on Pattern Recognition* (2006), vol. 3, pp. 1135–1138. 2