STAG: Smart Tools and Applications in Graphics (2022)
D. Cabiddu, T. Schneider, G. Cherchi, and R. Scateni (Editors)

Versatile Geometric Flow Visualization by Controllable Shape and Volumetric Appearance

M. Zeidan[1,2], C. Peters[1], T. Rapp[1], and C. Dachsbacher[1]

[1]Computer Graphics Group, Karlsruhe Institute of Technology, Germany
[2]Faculty of Computer and Information Sciences, Ain Shams University, Egypt

Abstract

We present a novel visualization technique for geometry-based visualization of vector fields. Our approach generalizes and combines several existing approaches in a flexible framework using a scalable GPU-accelerated implementation. We map characteristic lines to a variety of glyphs. The user can define multiple cross-sectional shapes that will be used for extrusion. Our system interpolates between these shapes as requested, either based on attributes of the vector field and the characteristic lines or using global user-controlled parameters. Thus, a single characteristic line can use different cross-sectional shapes in different parts to aid the visualization of different phenomena. Transitions can be smooth or discrete and we support highlighting of silhouettes. Additionally, we track and visualize the rotation in the vector field and offer full control of the color mapping, the opacity and the radii along the characteristic lines. Texture-based approaches such as 3D line integral convolution (3D LIC) offer another avenue to vector field visualization. In 3D, they typically rely on sparsely placed seed points. We emulate their appearance with our geometry-based approach through an approximation of the volume integral within our glyphs. Combined with fast order-independent transparency, our GPU implementation achieves fast rendering, even at high resolutions, while keeping the memory footprint moderate.

CCS Concepts

• ***Human-centered computing*** → ***Visualization systems and tools;***

1. Introduction

A large variety of techniques for 3D flow visualization has been developed over the last decades, categorized into direct, geometry-based (e.g. streamlines or streamsurfaces), texture-based (such as 3D-LIC), and feature-based techniques [LHD*04].

Geometry-based techniques create many individual geometric glyphs which can easily convey more information than just direction, e.g. rotation (streamribbons) or divergence (streamtubes). Different glyphs are suitable for the visualization of different phenomena that often occur together in a single vector field. Therefore, we propose to combine different methods by giving the user fine-grained control of cross-sectional shapes along the characteristic lines and by interpolating between them as needed. Other aspects such as radii and colors are also controllable.

Texture-based approaches, which use volume rendering to generate the final images, offer another intuitive flow visualization. 3D line-integral convolution (LIC) [FW08] is a prominent example. They convey directionality in the volume using transparency and directionally smoothed texture features. However, volume computation and rendering are costly and do not scale well to high screen resolutions, and the storage demands are non-negligible if the volume is not computed on-the-fly [FW08]. Although dense visualizations do not deteriorate performance further, sparse visualizations are preferred to avoid excessive visual clutter.

We use our geometric glyphs to mimic the appearance of sparse texture-based approaches. Traditionally, handling transparency is challenging for geometry-based approaches. Our system uses moment-based order-independent transparency [MKKP18] for high-quality results at high performance. We also propose an approximation for the transmittance through glyphs, which allows us to mimic the volumetric appearance of 3D LIC.

Thus, our system generalizes a variety of geometry-based and texture-based flow visualizations and enables users to combine them as needed and to transition between them in a smooth fashion. All steps rely on GPU-acceleration and rendering benefits from rasterization hardware. Therefore, our method scales well to high geometric complexity and large screen resolutions.

2. Related Work

Geometry-based flow visualization techniques [MLP*10] place discrete geometry in the velocity field to visualize the behavior of the underlying flow. In particular, Ueng et al. [USM96] discuss the efficient construction of streamlines, streamribbons, and

DOI: 10.2312/stag.20221259

streamtubes. Streamsurfaces [ELC*12] are another standard approach to explore flow behavior in 3D. To address increasingly large and complex data sets, illustrative visualization [BCP*12] aims at visualizing the data using effective visual abstractions similar to handcrafted illustrations. View-dependent streamline selection and placement [MCHM10; GRT13] is another promising research direction to visualize complex flows. Günther et al. [GTG17] propose a global optimization that balances occlusion and meaningful geometry. These methods limit flow geometry exploration to shading colors and opacity values.

Streamlines and streamtubes are widely used for diffusion MRI visualization where streamline trajectories follow the main eigenvector of diffusion MRI data, and the cross-sectional shapes along the streamlines encode the medium and minor eigenvectors [RBE*06]. In high-angular resolution diffusion imaging (HARDI) visualization, a continuous placement of ellipsoidal glyphs over streamlines is used [PPvA*11]. Vos et al. [VVL13] use hyperstreamlines and streamribbons to visualize local orientation of MRI tracts. Wien et al. [WSSS14] use interpolation along streamlines from circular to superquadratic cross-sections with sharp edges to convey the ratio and orientation of the second and third eigenvectors in diffusion MRI data. Our framework generalizes such techniques as it allows interpolation between arbitrary cross-sectional shapes in various ways.

The introduction of line integral convolution (LIC) [CL93] spawned a lot of follow-up work [LHD*04; LEG*08]. Notably, Wegenkittel et al. [WGP97] introduce oriented LIC (OLIC) to visualize the direction of flow in still images by using a sparse texture of spots that is smeared in the direction of the local vector field.

LIC can be extended to 3D but it is computationally expensive and causes perceptual and occlusion problems when displaying a dense 3D volume. Interrante and Grosch [IG98] were the first to apply LIC in 3D and emphasize the significance of sparseness in 3D. Suzuki et al. [SFCN02] propose the use of a significance map, to select interesting parts of the flow. Falk and Weiskopf [FW08] introduce output-sensitive 3D LIC, which tightly couples the LIC generation with volume rendering to avoid unnecessary evaluations of the LIC integral. This motivates us to present a fast way to visualize directional properties and volumetric appearance of vector field streaks using a rasterization approach.

3. Geometry Generation

We propose a novel approach for visualization of characteristic lines that enables a flexible and intuitive way to generate diverse geometric glyphs along characteristic lines. Fig. 1 shows an overview of our vector field visualization pipeline. To encode local attribute properties, our method offers a simple way to generate arbitrary geometric structures along characteristic lines using 2D cross-sectional glyphs (Sec. 3.2). Then, the user gets to choose in which way cross-sectional glyphs are interpolated (Sec. 3.3). Finally, we allow the user to change the width of geometric structures along characteristic lines to efficiently visualize flow direction (Sec. 3.4). The flexible generation of geometry naturally enables us to reproduce techniques from geometry-based vector field visualization (e.g. streamribbons, and streamtubes), and also enables us to mimic the appearance of texture-based/volumetric techniques such as 3D LIC [FW08]. We extensively rely on GPU acceleration and implement all stages of geometry generation using NVIDIA CUDA.

In Section 4, we present a rasterization approach to render extruded geometric structures. In addition, our technique approximates volumetric absorption and composes semi-transparent geometric structures along the view ray to convey volumetric appearance using order-independent transparency techniques [YHGT10; MKKP18].

3.1. Integration of Characteristic Lines

We generate characteristic lines by tracing independent particles through a vector field using a fourth-order Runge-Kutta method [BDV*04]. Our current implementation places initial positions at stratified random locations inside the vector field, and each particle trace stops when it reaches the boundary of the volume or a user-defined maximal number of integration steps. After tracing, each characteristic line is represented as a sequence of 3D points $p_0, p_1, \ldots, p_n \in \mathbb{R}^3$, where each point has corresponding attributes such as the magnitude of the velocity that can be used later during geometry extrusion and final visualization. In order to keep track of vector field local direction, we always trace two particles at spatially equidistant positions perpendicular to the tracing direction [Tel07].

To generate characteristic lines in parallel, we bind the vector field as a 3D texture to benefit from hardware-accelerated texture filtering. We preallocate a GPU storage buffer for all lines to handle the maximum number of integration steps. After particle tracing, lines might have a different number of points. Thus, we perform a parallel stream compaction [SHZO07] on all lines. We compute aggregate attributes for lines such as the number of points per line, the length, or the curvature. Optionally, the user can define thresholds for length and curvature to filter out uninteresting lines. Filtering is followed by another stream compaction. Then, we prepare geometric buffers needed for shape extrusion and extrude cross-sectional glyphs over line vertices in parallel.

3.2. Design and Placement of Cross-Sectional Glyphs

Transfer functions that map input attributes (e.g., velocity, pressure, or heat) onto color and opacity play a crucial role in modern visualization pipelines. However, transfer functions do not provide a suitable way to encode local attributes such as local rotation or global attributes such as flow direction. Our visualization method provides an efficient way to encode local flow attributes by using cross-sectional glyphs along characteristic lines, and directional attributes using a longitudinal kernel. From now and throughout the paper, we assume that input attributes are normalized to the range from 0 to 1. We allow the user to place diverse 2D cross-sectional glyphs along the attribute range. Each cross-sectional glyph is defined by a closed polygon and fills a certain range of input attributes. This closed polygon is extruded throughout each streamlet that passes through the corresponding input range (see Fig. 2 and 3).

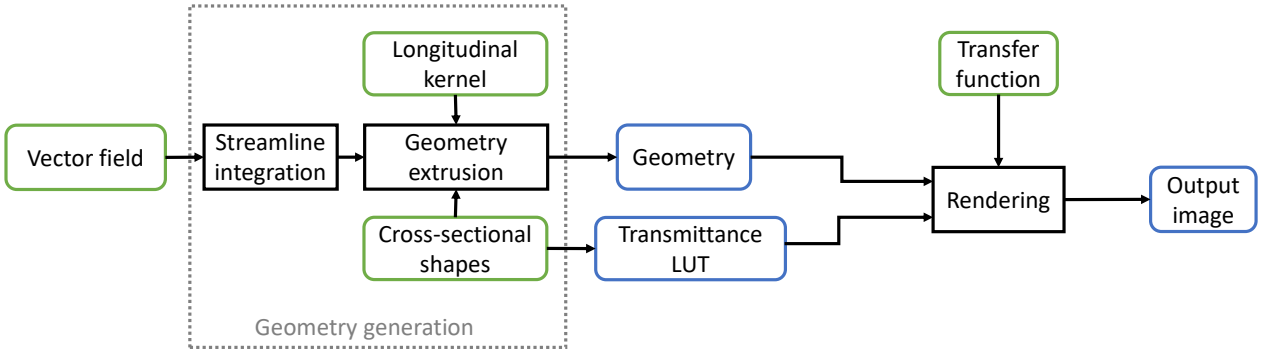

Figure 1: *An overview of our pipeline for vector field visualization. Data in green boxes is controlled by the user interactively. Data in blue boxes is updated at run-time. All operations (black rectangles) are GPU-accelerated.*

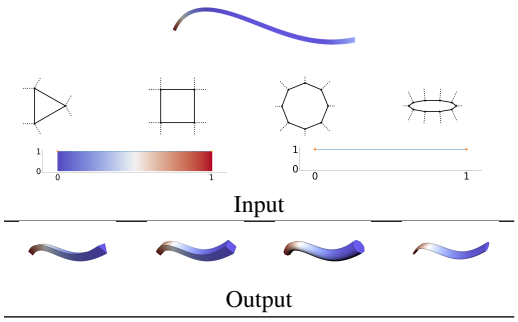

Figure 2: *Extrusion of cross-sectional glyphs using different shapes. We use flat normals for triangle and quad glyphs, and smooth normals for circle and ellipse glyphs. We use a constant radius as shown in the kernel in the bottom right of top row.*

Our framework offers the user a 2D glyph control panel to create polygons defined by a fixed number of vertices. Additionally, we provide commonly used standard shapes such as circles and ribbons. Our GUI panel also supports basic 2D vertex editing operations using either a mouse pointer or a position attribute edit box. Fig. 2 demonstrates an illustrative example of a characteristic line with different cross-sectional glyphs.

3.3. Morphing Cross-Sectional Glyphs

To combine the strengths of different glyphs, our extrusion method supports 2D shape morphing using linear interpolation of vertices with fixed correspondences. The user assigns different cross-sectional glyphs at certain data values in the range $[0,1]$. This is similar in spirit to assigning different colors and opacities to different values through a transfer function. Consider two neighboring glyphs G_a, G_b, assigned to two values $a, b \in [0,1]$. During extrusion for a vertex of a characteristic line with a value $x \in [a,b]$, we perform linear interpolation between the vertices of the glyphs G_a and G_b to obtain the cross-sectional glyph used for this line vertex. This method exploits that all glyphs are designed with the same vertex count and with fixed correspondences. Finding correspondences could be automated [YF09] but we opt for a simpler approach.

The user can choose between three modes of interpolation as shown in Fig. 3. Firstly, attribute values along a streamline can be used for continuous interpolation between neighboring shapes as shown in row A. Secondly, a specific attribute value can be used to select an interpolation instant for the whole streamline extrusion, e.g. to explore different glyphs conveniently as shown in row B. And finally, a threshold attribute value can be used to obtain a hard transition between different glyphs as shown in row C.

During shape morphing, we use a special strategy for handling shape normals. As an input to our technique, each 2D glyph vertex is marked with a flag as either flat or smooth. For flat vertices, we use edge normals and duplicate vertices during extrusion. For smooth vertices, we compute the per-vertex normal as the average of both edge normals. Our interpolation technique merges vertices marked with two different flags as flat.

3.4. Controlling Longitudinal Kernel

In addition, our system can control the geometric width across the extruded geometry. During geometry extrusion, we bind the longitudinal kernel (see Sec. 3.5) as a 1D texture and sample it using a certain attribute. By doing so, we can encode directional properties of characteristic lines using the integration step t for kernel sampling (see Fig. 4), or highlight certain features within vector fields using a certain attribute range.

Spatially Varying Opacity. In oriented LIC, the filter kernel $k(s)$ is an asymmetric function such that the directionality of the vector field is encoded in the density. Varying the density in this way is easily accomplished with our method. A factor for the opacity is taken from a user-defined function that depends on the time variable of the characteristic line.

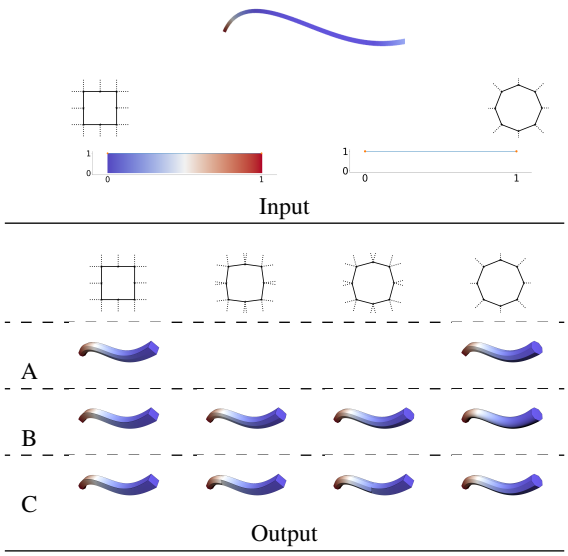

Figure 3: *Using circle and quad glyphs, we allow several modes of interpolations along extruded geometry. In row A, we extrude all interpolation instants between quad and circle (left), and between circle and quad (right). In row B, we display four separate example instants of interpolation, where we selectively extrude an intermediate shape between quad and circle over the whole streamline. In row C, we use a user-specified threshold t along a streamline to selectively extrude the first glyph shape to all attribute values below t, and extrude the other glyph shape to the rest of the input range.*

3.5. Generation of Triangle Meshes

The geometry generation step is done on the GPU using several parallel kernel calls across vertices and characteristic lines. The output of this stage is a triangle soup of vertex and index buffers, which are subsequently bound to an OpenGL rasterizer for final rendering (Sec. 4). We define a segment as a user-defined region along the attribute range with two cross-sectional glyphs with the same number of vertices. In the beginning, we triangulate the closed polygon of cross-section shape to be mapped at the start and end boundaries of each extruded segment. For each pair of consecutive vertices, we generate side quad faces between each two adjacent points of the cross-sectional glyphs. To do so, we launch a parallel kernel for each characteristic line vertex to count the exact triangle budget needed for the corresponding geometric faces. Then we count the aggregate number of triangles for each characteristic line and use a prefix sum scan [SHZO07] to emit triangles of each vertex in another parallel pass over all characteristic line vertices. Streamribbon glyphs have special but predefined handling in this stage since we display such glyphs as geometric quads between each adjacent pair of vertices along the characteristic line.

4. Rendering

In this section, we discuss the main steps to render geometric structures produced by our framework. Firstly, we describe our shading scheme in Sec. 4.1. Next, in Sec. 4.2, we show how to achieve a volumetric appearance in spite of the geometry-based representation of input structures. Finally, we discuss the use of state-of-the-art order-independent transparency techniques for the correct composition of the characteristic lines in Sec. 4.3. Our renderer uses OpenGL 4.5 with programmable shaders to render the opaque or volumetric streaks.

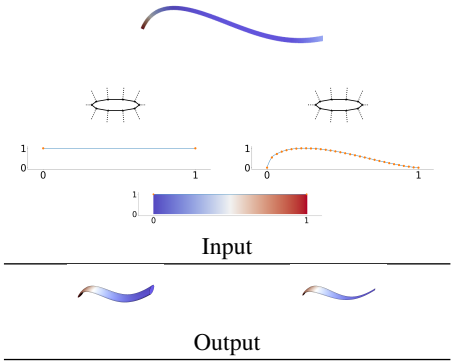

Figure 4: *Extruding an ellipse cross-section shape using a constant kernel for extrusion (left), and ramp kernel (right).*

4.1. Shading

Shading contributes substantially to the perception of the shape of geometric structures. We use a physically based reflection model for shading [Wol18] with point light sources. The material colors are chosen by the user or taken from a transfer function. The final rendering color is subject to change through shading (Sec. 4.1) and the absorption coefficient is integrated along view rays to obtain opacity (Sec. 4.2).

To aid the understanding of silhouettes as well as edges with flat shading, we highlight them with black outlines in the spirit of non-photorealistic rendering. We detect silhouettes and edges in screen space using normal and depth buffers [ND03].

4.2. Opacity Computation along Volumetric Cross Sections

From the transfer function, we only obtain an absorption coefficient $\sigma \geq 0$. However, for volumetric lines, each fragment that we render corresponds to a whole line segment where the ray passes through the volume. We assume that the absorption coefficient is constant along these line segments. Thus, the only missing quantity to compute the opacity $\alpha \in [0,1]$ of a fragment is the distance $d > 0$ that a ray travels through the extruded geometry. Then according to Beer's law, the opacity is

$$\alpha = 1 - e^{-\sigma d}.$$

An exact computation of the distance traveled would require ray-triangle intersections. To avoid this cost, we make one simplifying assumption: We assume that the extruded lines are thin enough to be considered straight over the distances that the rays travel within them. Thus, we can project the ray direction to the plane orthogonal to the tube direction and compute the distance traveled for this

modified ray. Subsequently, we divide the distance by the sine of the angle between the ray direction and the tube direction to obtain d for the original ray. The assumption above fails at grazing angles. In practice and to avoid extreme results, we clamp the reciprocal of the sine above by 5.

Hence, we have turned the computation of the distance traveled into a planar problem. The set of all rays that we need to consider is only two-dimensional and it is viable to create a complete lookup table. This two-dimensional lookup table is only constructed once per cross-sectional shape. In the following, we fix definitions for how to construct and use the table.

Recall from Section 3.2 that the cross-sectional shape is defined in a coordinate frame where the yz-plane is orthogonal to the tube direction. It is natural to perform the computations for the distance traveled in this coordinate frame as well. We begin by computing a bounding circle around the cross-sectional shape. Each point on this circle is characterized by an angle, namely $atan2(z,y)$. A ray in the yz-plane is characterized by the points where it enters and leaves this circle, or equivalently by two angles $\psi_i, \psi_o \in [-\pi, \pi]$ (disregarding rays that start within the tube) (see Fig. 5). These two angles are the values that we use to make lookups in the table. To store the lookup table in a texture, the angles are mapped to texture coordinates in $[0,1]$ linearly. During lookups, we only have to perform a planar ray-circle intersection to compute the two intersection points and thus the angles.

For construction of the lookup table, we iterate over all pairs of angles, e.g. at a resolution of 512^2. For each pair, we construct the ray in Cartesian coordinates and perform intersection tests with all line segments that constitute the cross-sectional shape. From the sorted list of intersection points, the distance that the ray travels within the cross-sectional shape is trivial to compute. This computation is implemented in a CUDA kernel using one thread per entry of the lookup table.

To support volumetric cross-sectional transmittance approximation with 2D cross-sectional glyphs interpolation (Sec. 3.2), we sample n intermediate shapes in between each pair of neighbors glyphs and store the corresponding n lookup tables in a 3D texture. During rendering, we use trilinear interpolation.

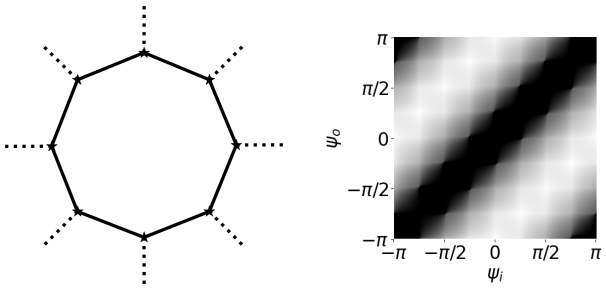

Figure 5: *Precomputed traveled distance (right) over cross-sectional disk shape (left).*

Controlling Opacity. Although we compute opacity across geometric primitives using approximate transmittance, we also give the user full control to edit opacity for important regions using a transfer function widget. Using this widget, the user can edit opacity and color to explore the effects on the data set at run time. In our framework, we optionally, and according to user input, modulate the approximate transmittance by the transfer function opacity. This way we give the user focus-plus-context functionality by emphasizing regions of interest using distinct opacity and color values.

4.3. Order-Independent Transparency

Since we rely on rasterization rather than ray marching, geometry is not processed front to back. Therefore, compositing of semi-transparent fragments requires a technique for order-independent transparency. Moment-based order-independent transparency (MBOIT) [MKKP18] is well-suited for our use case. The technique scales well to situations with high overdraw, guarantees smooth and plausible results and approximates the ground truth more accurately than competing techniques. A full explanation of MBOIT is beyond the scope of this paper but we briefly discuss the steps we performed to integrate it into our renderer using the published code [MKKP18].

MBOIT renders the transparent geometry in two passes with simple additive blending. The first pass renders to so-called moment buffers, which encode the information needed for correct compositing. The second pass renders to color buffers. For each fragment it reads from the moment buffers and uses a special heuristic to compute how much the fragments in front occlude the currently rendered fragment. Multiplying the color value by this transmittance factor and adding it to the other color values yields an estimate of the pixel color. A subsequent full screen pass normalizes the sum of the colors and composites them with the background (a constant color in our case). Opaque geometry is rendered separately with depth buffer writes enabled.

5. Results and Evaluation

In this section, we demonstrate multiple visualizations to analyze and highlight interesting features within real and synthetic 3D vector field data sets. We also discuss several visualization parameters and evaluate performance using our method. For all experiments, we used a computer with an Intel i7-6700 CPU with 3.40 GHz and 32 GB of main memory, and an NVIDIA GTX 1080 Ti GPU with 11 GB VRAM. In all rendered examples, and if not mentioned otherwise, we map the velocity magnitude to color using a transfer function. All example images are rendered at a resolution of 1920×1080. Table 1 reports the size of input and output geometry for our test data sets as well as timings for geometry generation and rendering. We achieve real-time rendering and interactive editing of geometry in all cases.

5.1. Applications

Flow Visualization using a Directional Kernel. By using a directional kernel, we visualize flow direction by extruding an isotropic cross-sectional glyph with a varying width over each streamline. In Fig. 6, we use a multiple ramps kernel with a circular cross-sectional glyph to encode flow direction inside the synthetic tornado vector field [MCHM10]. In order to apply the longitudinal

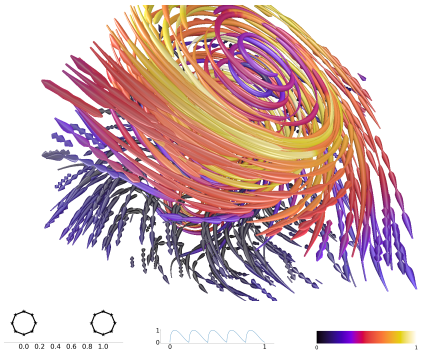

Figure 6: *The tornado dataset with a circular cross-sectional glyph and several ramps as a longitudinal kernel.*

kernel over streamlines, we keep a parametric distance t for each streamline vertex during particle tracing. This parametric distance is proportional to the integration step and is normalized per streamline to be in $[0, 1]$.

Our method lets us mimic the volumetric appearance of 3D directional LIC. In Fig. 7 we compare rendering the geometric structures using a constant opacity of 0.7 for all structures against our approach using approximated volumetric traveled distance (see Sec. 4.2). As shown in Table 1, our volumetric cross-section opacity approximation has a low overhead on rendering performance.

An important aspect of our method is its scalability in terms of geometry generation and rendering performance with respect to the number of integrated characteristic lines. Fig. 8 shows the TrefoilKnot data set with diverse LIC-like structures using a varying number of lines to emulate the sparsity of noise texture integration with 3D volumetric LIC visualization [FW08]. Short lines are filtered out. In Table 1, we list the corresponding geometry size, geometry generation time, and rendering performance. As we increase the number of characteristic lines, rendering remains real-time and geometry generation remains interactive.

In contrast to 3D LIC, our approach does not require an expensive pre-processing step [FW08] or on-the-fly integration of characteristic lines. After our geometry generation step, we do not have to perform any additional operations besides rendering. Lastly, our approach is well-suited for high resolutions since we do not perform volume rendering. Our system provides real-time feedback avoiding the complexity of volumetric ray casting inside 3D volumes [RHTE99; HA04].

Region of Interest Selection. In Fig. 9, we map velocity to cross-sectional glyphs. Low velocities are mapped to tube structures and high velocity values are mapped to streamribbons. In this scenario, we used ribbon-to-circular shape continuous interpolation. As the flow becomes faster and more turbulent, we smoothly transition to a visualization that highlights rotational components.

Encoding Vector Field Rotational Attributes. Figure 10 shows simulated flow around an airplane model [ST69; SGH06]. The flow around the airplane wings has strong rotational components. This region is of great importance for studying fluid dynamic phenomena. We used streamribbons to highlight local rotation of streamlines. To do so, we used streamline curvature as an input parameter to interpolate from a circular cross-section (green inset) to a ribbon (orange and blue insets). In this case, we used curvature as a threshold value to select our region of interest so that regions with high curvature values are mapped to streamribbons and other areas are mapped to circular cross sections.

We sample equidistant vertices over the streamribbon sides to match the number of vertices of the other glyph. Only the top and bottom vertices at the streamribbon are marked with flat flags, and all other vertices are marked with smooth flags.

6. Conclusions and Future Work

In this paper, we presented a geometry-based flow visualization technique whose appearance can be controlled intuitively by the user and mimic a variety of well-established techniques (e.g. streamribbons) as well as "blending" between them. The core of our technique is the creation of geometric glyphs of genus-0 along characteristic lines with specific radii, arbitrary cross-sections, color, and opacity mapping. Combined with an approximation of the volume integrals of view rays intersecting the resulting geometry, we also reproduce the appearance of texture-based/volumetric techniques such as 3D line integral convolution. Despite its versatility, our method is easy to implement, well-suited for GPU implementations, and easily achieves real-time frame rates even at high screen resolutions.

Our method shares the limitations of most flow visualization techniques in 3D. Displaying many lines results in cluttered renderings that might hide important regions. On the other hand, it is difficult to find a representative set of characteristic lines that emphasizes important features. Since our method is a geometric rendering technique motivated by volume rendering, it becomes problematic when geometric primitives, such as tubes, are clipped at the near viewing plane.

In the future, we would like to use an automatic way to find correspondences between vertices of different 2D cross-sectional glyphs [YF09]. We also want to employ an automatic way to select and place geometric glyphs on streamlines in a way similar to opacity optimization techniques [GTG17; ZRPD20].

7. Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments. The tornado dataset is courtesy of Marchesinoet al. [MCHM10], and TrefoilKnot dataset is courtesy of Candelaresi and Brandenburg [CB11]. Thanks to Mahmoud Salem, Hisanari Otsu, and Max Piochowiak for the proofreading. The first author is partially supported by the Cultural Affairs and Missions Sector in Egypt, and the German Academic Exchange Service in Germany.

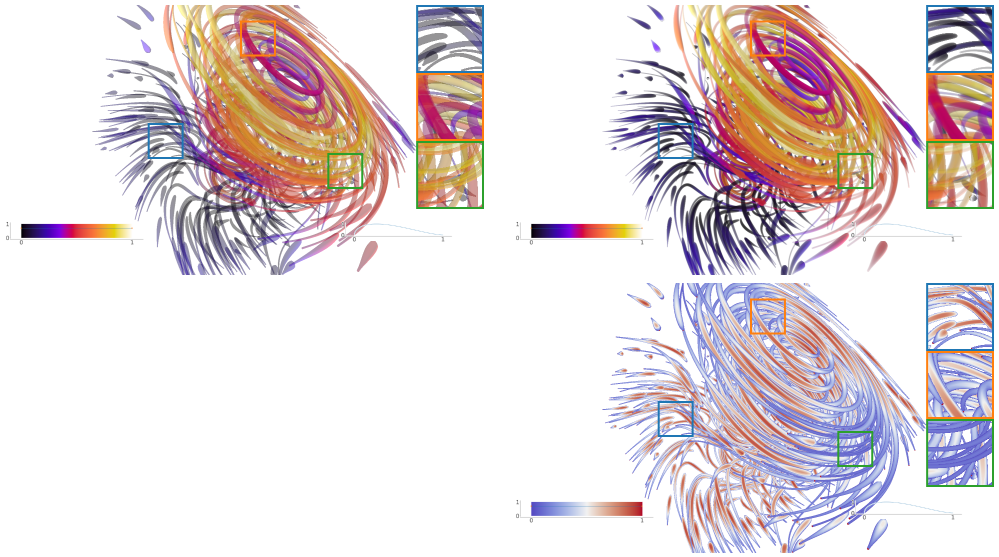

Figure 7: *The tornado dataset with circular cross-section glyph and a ramp longitudinal kernel. Left: Geometry is rendered with a constant opacity value of 0.7 with all geometry. Right top: Approximate cross-sectional transmittance along geometry using our method. Right bottom: Approximate traveled distance using our cross-sectional transmittance approximation.*

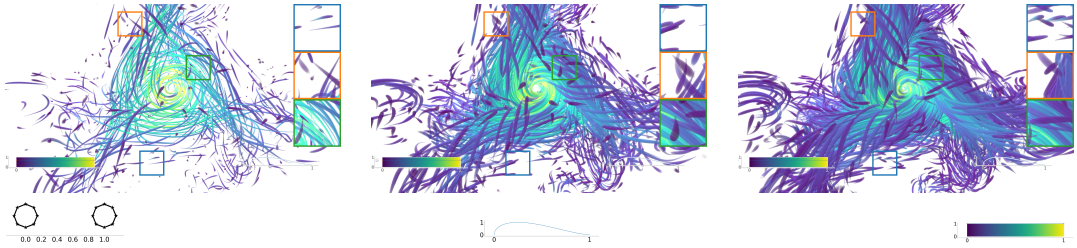

Figure 8: *A ramp kernel and a circular cross-sectional shape are used to emulate texture sparsity with the TrefoilKnot dataset. From left to right we increase the number of streamlines.*

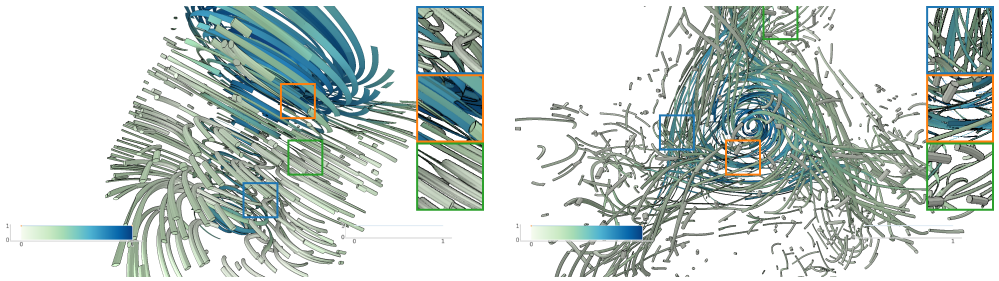

Figure 9: *Encoding vector field attributes using different cross-section glyphs and interpolation instants. High magnitude velocities are marked with streamribbons and low values are marked with interpolation instants between a disk shape and a streamribbon.*

Table 1: *Performance measurements for all figures. Geometry is only regenerated when it changes. Render times are total frame times.*

Dataset	Input geometry		Output geometry		Run-time	
	Num. streamlines	Num. vertices (k)	Num. vertices (k)	Num. indices (k)	Geometry (ms)	Rendering (ms)
Tornado Fig. 6	353	21.7	179.4	1039.3	8.5	6.2
Tornado Fig. 7 top left	353	21.7	179.4	1039.3	8.4	6.8
Tornado Fig. 7 top right						7.3
TrefoilKnot Fig. 8 left	924	87	172	4179	32	7.7
TrefoilKnot Fig. 8 middle	1845	175	1435	8415	62	12
TrefoilKnot Fig. 8 right	2935	282	2302	13502	102	16
Tornado Fig. 9 left	353	21.7	288.4	844.6	12.4	6.4
TrefoilKnot Fig. 9 right	3084	298.1	4740.8	14037.5	198	9
DeltaWing Fig. 10	356	307.3	4922.7	14747.2	209	9.2

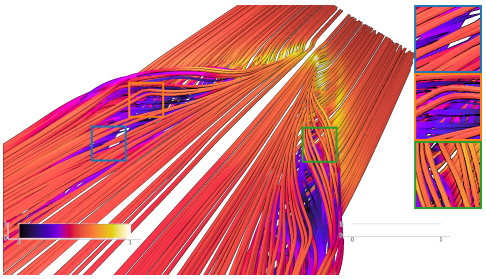

Figure 10: *Encoding the rotational component of the vector field using different cross-sectional glyphs. We use a vertex curvature threshold ($t = 0.015$) so that high curvature values are marked with streamribbons and low values are extruded with circular cross-sectional glyphs.*

References

[BCP*12] BRAMBILLA, ANDREA, CARNECKY, ROBERT, PEIKERT, RONALD, et al. "Illustrative Flow Visualization: State of the Art, Trends and Challenges". *Eurographics 2012 - State of the Art Reports*. Ed. by CANI, MARIE-PAULE and GANOVELLI, FABIO. The Eurographics Association, 2012. DOI: 10.2312/conf/EG2012/stars/075-094 2.

[BDV*04] BOYCE, WILLIAM E, DIPRIMA, RICHARD C, VILLAGÓMEZ VELÁZQUEZ, HUGO, et al. *Elementary differential equations and boundary value problems. Ecuaciones diferenciales y problemas con valores en la frontera*. 2004 2.

[CB11] CANDELARESI, SIMON and BRANDENBURG, AXEL. "Decay of helical and nonhelical magnetic knots". *Physical Review E* 84.1 (2011), 016406 6.

[CL93] CABRAL, BRIAN and LEEDOM, LEITH CASEY. "Imaging Vector Fields Using Line Integral Convolution". *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA: ACM, 1993, 263–270. ISBN: 0-89791-601-8. DOI: 10.1145/166117.166151 2.

[ELC*12] EDMUNDS, MATT, LARAMEE, ROBERT S., CHEN, GUONING, et al. "Surface-based flow visualization". *Computers & Graphics* 36.8 (2012). Graphics Interaction Virtual Environments and Applications 2012, 974–990. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2012.07.006 2.

[FW08] FALK, MARTIN and WEISKOPF, DANIEL. "Output-Sensitive 3D Line Integral Convolution". *IEEE Transactions on Visualization and Computer Graphics* 14.4 (July 2008), 820–834. ISSN: 1077-2626. DOI: 10.1109/TVCG.2008.25 1, 2, 6.

[GRT13] GÜNTHER, TOBIAS, RÖSSL, CHRISTIAN, and THEISEL, HOLGER. "Opacity Optimization for 3D Line Fields". *ACM Trans. Graph.* 32.4 (July 2013), 120:1–120:8. ISSN: 0730-0301. DOI: 10.1145/2461912.2461930 2.

[GTG17] GÜNTHER, TOBIAS, THEISEL, HOLGER, and GROSS, MARKUS. "Decoupled Opacity Optimization for Points, Lines and Surfaces". *Computer Graphics Forum* 36.2 (2017), 153–162. DOI: 10.1111/cgf.13115 2, 6.

[HA04] HELGELAND, A. and ANDREASSEN, O. "Visualization of vector fields using seed LIC and volume rendering". *IEEE Transactions on Visualization and Computer Graphics* 10.6 (Nov. 2004), 673–682. ISSN: 1077-2626. DOI: 10.1109/TVCG.2004.49 6.

[IG98] INTERRANTE, V. and GROSCH, C. "Visualizing 3D flow". *IEEE Computer Graphics and Applications* 18.4 (July 1998), 49–53. ISSN: 0272-1716. DOI: 10.1109/38.689664 2.

[LEG*08] LARAMEE, ROBERT S., ERLEBACHER, GORDON, GARTH, CHRISTOPH, et al. "Applications of Texture-Based Flow Visualization". *Engineering Applications of Computational Fluid Mechanics* 2.3 (2008), 264–274. DOI: 10.1080/19942060.2008.11015227 2.

[LHD*04] LARAMEE, ROBERT S., HAUSER, HELWIG, DOLEISCH, HELMUT, et al. "The State of the Art in Flow Visualization: Dense and Texture-Based Techniques". *Computer Graphics Forum* 23.2 (2004), 203–221. DOI: 10.1111/j.1467-8659.2004.00753.x 1, 2.

[MCHM10] MARCHESIN, S., CHEN, C., HO, C., and MA, K. "View-Dependent Streamlines for 3D Vector Fields". *IEEE Transactions on Visualization and Computer Graphics* 16.6 (Nov. 2010), 1578–1586. ISSN: 1077-2626. DOI: 10.1109/TVCG.2010.212 2, 5, 6.

[MKKP18] MÜNSTERMANN, CEDRICK, KRUMPEN, STEFAN, KLEIN, REINHARD, and PETERS, CHRISTOPH. "Moment-Based Order-Independent Transparency". *Proc. ACM Comput. Graph. Interact. Tech. (Proc. i3D)* 1.1 (July 2018), 7:1–7:20. ISSN: 2577-6193. DOI: 10.1145/3203206 1, 2, 5.

[MLP*10] MCLOUGHLIN, TONY, LARAMEE, ROBERT S., PEIKERT, RONALD, et al. "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 1.

[ND03] NIENHAUS, MARC and DÖLLNER, JÜRGEN. "Edge-Enhancement-An Algorithm for Real-Time Non-Photorealistic Rendering." *WSCG*. Vol. 11. 2003, 1–3 4.

[PPvA*11] PRCKOVSKA, VESNA, PEETERS, TIM H. J. M., van ALMSICK, MARKUS, et al. "Fused DTI/HARDI Visualization". *IEEE Transactions on Visualization and Computer Graphics* 17.10 (2011), 1407–1419. DOI: 10.1109/TVCG.2010.244 2.

[RBE*06] REINA, G., BIDMON, K., ENDERS, F., et al. "GPU-Based Hyperstreamlines for Diffusion Tensor Imaging". *EUROVIS - Eurographics /IEEE VGTC Symposium on Visualization*. Ed. by SANTOS, BEATRIZ SOUSA, ERTL, THOMAS, and JOY, KEN. The Eurographics Association, 2006. ISBN: 3-905673-31-2. DOI: 10.2312/VisSym/EuroVis06/035-042 2.

[RHTE99] REZK-SALAMA, C., HASTREITER, P., TEITZEL, C., and ERTL, T. "Interactive Exploration of Volume Line Integral Convolution Based on 3D-texture Mapping". *Proceedings of the Conference on Visualization '99: Celebrating Ten Years*. VIS '99. San Francisco, California, USA: IEEE Computer Society Press, 1999, 233–240. ISBN: 0-7803-5897-X. URL: http://dl.acm.org/citation.cfm?id=319351.319379 6.

[SFCN02] SUZUKI, Y., FUJISHIRO, I., CHEN, L., and NAKAMURA, H. "Case study: hardware-accelerated selective LIC volume rendering". *IEEE Visualization, 2002. VIS 2002.* Oct. 2002, 485–488. DOI: 10.1109/VISUAL.2002.1183811 2.

[SGH06] SCHWAMBORN, DIETER, GERHOLD, THOMAS, and HEINRICH, RALF. "The DLR TAU-code: recent applications in research and industry". (2006) 6.

[SHZO07] SENGUPTA, SHUBHABRATA, HARRIS, MARK, ZHANG, YAO, and OWENS, JOHN D. "Scan primitives for GPU computing". 2007 2, 4.

[ST69] SCHLICHTING, HERMANN and TRUCKENBRODT, ERICH. "Tragflügel endlicher Spannweite bei inkompressibler Strömung". *Aerodynamik des Flugzeuges: Zweiter Band: Aerodynamik des Tragflügels (Teil II), des Rumpfes, der Flügel-Rumpf-Anordnung und der Leitwerke*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1969, 1–132. ISBN: 978-3-662-05619-6. DOI: 10.1007/978-3-662-05619-6_1. URL: https://doi.org/10.1007/978-3-662-05619-6_1 6.

[Tel07] TELEA, ALEXANDRU C. *Data visualization: principles and practice*. 2nd Edition. AK Peters/CRC Press, 2007 2.

[USM96] UENG, SHYH-KUANG, SIKORSKI, C., and MA, KWAN-LIU. "Efficient streamline, streamribbon, and streamtube constructions on unstructured grids". *IEEE Transactions on Visualization and Computer Graphics* 2.2 (June 1996), 100–110. ISSN: 1077-2626. DOI: 10.1109/2945.506222 1.

[VVL13] VOS, SJOERD B., VIERGEVER, MAX A., and LEEMANS, ALEXANDER. "Multi-Fiber Tractography Visualizations for Diffusion MRI Data". *PLOS ONE* 8.11 (Nov. 2013), null. DOI: 10.1371/journal.pone.0081453. URL: https://doi.org/10.1371/journal.pone.0081453 2.

[WGP97] WEGENKITTL, R., GROLLER, E., and PURGATHOFER, W. "Animating flow fields: rendering of oriented line integral convolution". *Proceedings. Computer Animation '97 (Cat. No.97TB100120)*. June 1997, 15–21. DOI: 10.1109/CA.1997.601035 2.

[Wol18] WOLFF, DAVID. *OpenGL 4 Shading Language Cookbook: Build High-Quality, Real-Time 3D Graphics with OpenGL 4.6, GLSL 4.6 and C++17, 3rd Edition*. Packt Publishing, 2018. ISBN: 9781789342253 4.

[WSSS14] WIENS, VITALIS, SCHLAFFKE, LARA, SCHMIDT-WILCKE, TOBIAS, and SCHULTZ, THOMAS. "Visualizing Uncertainty in HARDI Tractography Using Superquadric Streamtubes". *EuroVis - Short Papers*. Ed. by ELMQVIST, N., HLAWITSCHKA, M., and KENNEDY, J. The Eurographics Association, 2014. ISBN: 978-3-905674-69-9. DOI: 10.2312/eurovisshort.20141154 2.

[YF09] YANG, WENWU and FENG, JIEQING. "2D shape morphing via automatic feature matching and hierarchical interpolation". *Computers & Graphics* 33.3 (2009), 414–423 3, 6.

[YHGT10] YANG, JASON C., HENSLEY, JUSTIN, GRÜN, HOLGER, and THIBIEROZ, NICOLAS. "Real-Time Concurrent Linked List Construction on the GPU". *Proceedings of the 21st Eurographics Conference on Rendering*. EGSR'10. Saarbrücken, Germany: Eurographics Association, 2010, 1297–1304. DOI: 10.1111/j.1467-8659.2010.01725.x. URL: https://doi.org/10.1111/j.1467-8659.2010.01725.x 2.

[ZRPD20] ZEIDAN, MAHMOUD, RAPP, TOBIAS, PETERS, CHRISTOPH, and DACHSBACHER, CARSTEN. "Moment-Based Opacity Optimization". *Eurographics Symposium on Parallel Graphics and Visualization*. Ed. by FREY, STEFFEN, HUANG, JIAN, and SADLO, FILIP. The Eurographics Association, 2020. ISBN: 978-3-03868-107-6. DOI: 10.2312/pgv.20201072 6.