Vision, Modeling, and Visualization (2022)
J. Bender, M. Botsch, and D. Keim (Eds.)

Efficient High-Quality Rendering of Ribbons and Twisted Lines

Christoph Neuhauser[1], Junpeng Wang[1], Michael Kern[2] and Rüdiger Westermann[1]

[1]Technical University of Munich, Germany
[2]Advanced Micro Devices, Inc.

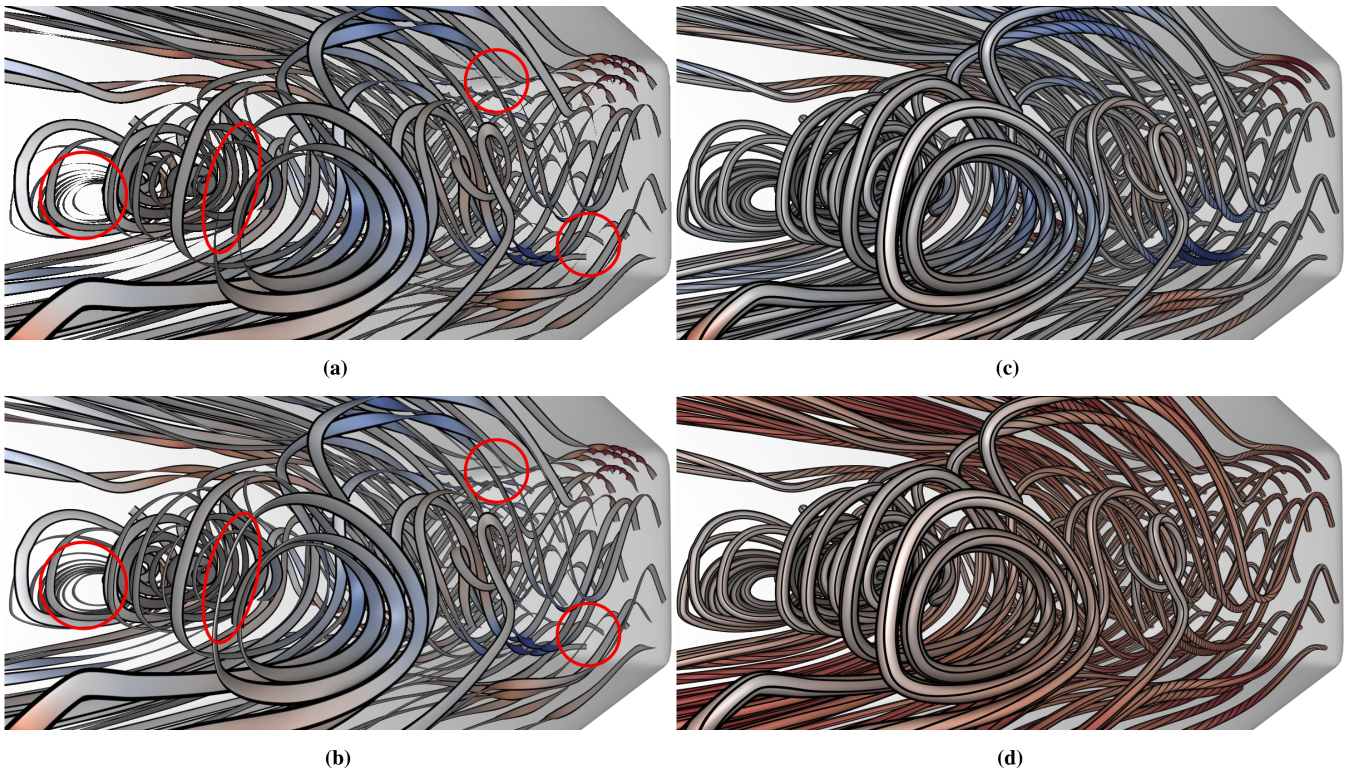

Figure 1: *Flow visualization of the Kármán vortex street using a) flat ribbons and b) our proposed elliptic ribbons. Elliptic ribbons avoid losing ribbon sections and reduce aliasing. c,d) Visualization using circular cylinders with twisting lines, to convey helicity along streamlines. c) Procedural rendering of cylinders with projected lines enforces constant line width, reduces aliasing, and enables instant changes of twist frequency and line width. Helicity is mapped to color from blue (high negative) over grey (zero) to red (high positive). d) Conveying helicity via twisting lines enables using color for showing a second quantity, i.e., velocity magnitude from grey (low) to red (high).*

Abstract

Flat twisting ribbons are often used for visualizing twists along lines in 3D space. Flat ribbons can disappear when looking at them under oblique angles, and they introduce flickering due to aliasing during animations. We demonstrate that this limitation can be overcome by procedurally rendering generalized cylinders with elliptic profiles. By adjusting the length of the cylinder's semi-minor axis, the ribbon thickness can be controlled so that it always remains visible. The proposed rendering approach further enables the visualization of twists via the projection of a line spiralling around the cylinder's center line. In contrast to texture mapping, this keeps the line width fixed, regardless of the strength of the twist, and provides efficient control over the spiralling frequency and coloring between the twisting lines. The proposed rendering approach can be performed efficiently on recent GPUs by exploiting programmable pulling, mesh shaders and hardware-accelerated ray tracing.

CCS Concepts

• ***Human-centered computing*** $\rightarrow$ *Scientific visualization;* • ***Computing methodologies*** $\rightarrow$ *Rendering;*

DOI: 10.2312/vmv.20221213

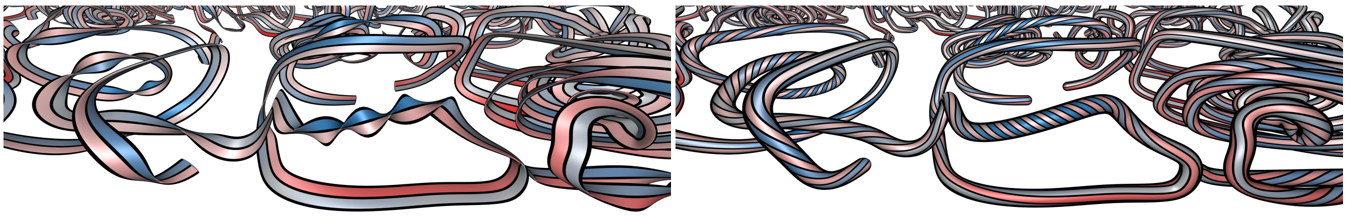

Figure 2: *Ribbon- (left) and twist-line-based (right) helicity visualization along streamlines in a Rayleigh–Bénard convection flow, including halos and visualization of temperature (red) and velocity magnitude (blue). The ribbons extend the technique by Neuhauser et al. [NHK*22] to use our silhouette point-based distance for view oriented multiparameter bands. RBC volume data set courtesy of Pandey et al. [PSS18].*

1. Introduction

In flow and tensor field visualizations, line primitives are used to graphically depict characteristic pathways in these fields, like stream- and path-lines in flow fields, or principal stress lines in stress tensor fields. While line primitives can effectively communicate the directional structure of the fields, ribbons are often used to show the local rotational motion via the vorticity or helicity in addition to the directional structure [Vol89; PW94]. Ribbons are rendered as flat, twisting quads, and, thus, can become very thin when viewed from the side. In the worst case this can lead to disappearing ribbon sections, and aliasing artifacts which are perceived as flickering during animations.

As nowadays lines are commonly rendered as generalized cylinders with a circular profile, twists along a line can also be encoded by coloring the cylinder surface [SGS05; EBRI09] or using additional geometric primitives placed along the lines [SPS06]. In particular, once the cylinder surface is available as a polygon mesh, the twist can be indicated by accordingly distorting the per-vertex texture coordinates, as, for instance, shown by Stoll et al. [SGS05]. By using a texture map containing a line pattern, this gives the impression of lines twisting around the cylinder axis and enables using color for an additional quantity (cf. Fig. 1d). A slight drawback of this approach is that at least locally a geometric surface representation is required, and the width of the twisting line varies depending on the strength of the twist, i.e., the line becomes thinner and wider with higher and lower frequency of the twist.

We address the aforementioned limitations as well as high-quality rendering of twists along cylindrical lines (see also Fig. 2) by the following contributions:

- We introduce the rendering of ribbons as generalized cylinders with an elliptic profile (cf. Subsec. 3.1). By keeping the ellipse thickness larger than zero, a minimum ribbon thickness can be enforced when viewed from the side. Thus, ribbon sections will never disappear and aliasing artifacts can be avoided. Lines and ribbons can be rendered in a unified way using generalized cylinders with circular and elliptic profiles.
- We extend the computation of silhouette points of circular cylinder profiles as introduced by Blinn [Bli89] to elliptic cylinder profiles. Thus, halos around lines and ribbons can be rendered in a single rendering pass on the GPU (cf. Subsec. 3.2).
- We propose a rendering approach for twisting lines on a generalized cylinder using procedural textures (cf. Sec. 4). This enables to maintain constant thickness of the twisting lines, avoids resolution limitations of image-based texture mapping, and enables cheap anti-aliasing using fragment shader derivatives.
- We provide an efficient implementation of all proposed rendering modes on the GPU using programmable pulling and mesh shaders. In particular for larger line sets, this implementation is significantly faster than alternative implementations using geometry shaders, e.g., by Krone et al. [KBE08], Kanzler et al. [KFW16] and Kern et al. [KNM*21]. Our implementation is open-source and available on GitHub (`https://github.com/chrismile/LineVis`).

2. Related Work

A popular approach for rendering line data in scientific visualization is via illuminated streamlines to enable an improved perception of the geometric structure of the lines. The concept was first introduced by Zöckler et al. [ZSH96], who utilized direct line rasterization with texture mapping to perform per-fragment illumination. Mattausch et al. [MTHG03] build upon this work and further add depth cues and halos along streamline silhouettes. Halos are generated by rendering the streamlines for a second time with increased line width and no color, thereby adapting the depth test to only let silhouette fragments pass. Everts et al. [EBRI09] use view-aligned quads for rendering lines with depth-dependent halos. Krueger et al. [KKKW05] and Merhof et al. [MSE*06] propose interactive and high quality visualizations of particle data and DTI fiber tracts, respectively, using triangle primitives that are textured to achieve the appearance of 3D primitives. Stoll et al. [SGS05] map line primitives to generalized cylinders with circular profile [AB76], called circular cylinders in our work. They recognize that view-aligned quads do not necessarily cover the whole area of the generalized cylinder and address this issue by using splatting in a hybrid CPU-GPU rendering pipeline. In the tessellation process, they make use of optimal cylinder silhouette point calculation by Blinn [Bli89], which we generalize in our work for elliptic cylinders.

Nowadays, most scientific visualization systems for 3D line data work entirely on the GPU and rely on geometry shader-based cylinder rendering, where line segments are extruded to circular cylinders in the geometry shader stage [KBE08; KFW16; KNM*21]. Kern et al. [KNM*21] compare geometry shader-based line rasterization to ray-tracing for rendering transparent lines using different transparency rendering approaches. Voxel ray casting [KRW18] stores discretized line segments in a regular voxel grid and traces rays through this grid. Han et al. [HWU*19] use ray tracing via OSPRay [WJA*17] to efficiently render generalized line primitives on multi-core CPU architectures. Reina et al. [RBE*06] propose a GPU-based sphere tracing technique for hyperstreamlines, and Eichelbaum et al. [EHS13] augment line rendering via fast ambient occlusions on the GPU.

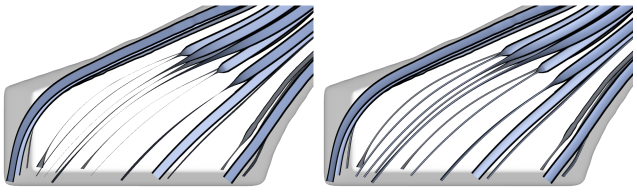

Figure 3: *Ribbons rendered as flat quads (left) and as elliptic cylinders with a minimum thickness (right).*

Ribbon-based line rendering was first introduced in the context of flow visualization by Volpe [Vol89]. Rees et al. [RLN*17] present a seeding strategy for streamribbons using the flow helicity $h = (\nabla \times v) \cdot v$ to control the seeding density. The helicity is the scalar product between the vorticity and the velocity vector and measures the local spinning motion around the tangential axis of a flow line. Ueng et al. [USM96] introduce streamribbon tracing in unstructured grids, and Zhang et al. [ZNT*18] propose Lagrangian accumulation fields for improved ribbon placement. Instead of streamribbons based on the helicity like Rees et al. [RLN*17] and Ueng et al. [USM96], Karch et al. [KSWE16] propose the use of vortex core ribbons for flow field visualization. At the center of the vortex core ribbon lies a vortex core line instead of a streamline. The direction of the ribbon is then not computed via the helicity, but by tracing two particles in the vicinity of the vortex core line to deduce the ribbon direction. Wang et al. [WNW*22] use stress ribbons to show flips in the assignment of principal stress directions to the eigenvectors of stress tensors.

3. Generalized Cylinders with Elliptic Profile

To avoid the drawbacks of ribbon rendering via flat twisting quads, we introduce a unified stylized rendering method for lines and ribbons. By using generalized cylinders with elliptic profile as the basic rendering primitives, a minimum ribbon thickness can be ensured to avoid losing ribbon sections and aliasing artifacts (see Fig. 3), and both ribbons and lines can be rendered by switching between an elliptic and circular profile.

3.1. Definitions

An elliptic cylinder of a curve $x(\tau)$ with radius r and dilation factor w in normal direction $n(\tau)$ is defined as the set of points

$$p(\tau,\varphi) = x(\tau) + rF(\tau)\begin{pmatrix} w\cdot\cos(\varphi) & \sin(\varphi) & 0\end{pmatrix}^T. \qquad (1)$$

$F(\tau) = (n(\tau), b(\tau), t(\tau)) \in SO(3)$ is a frame matrix based on the Frenet–Serret frame, with $t(\tau)$ being the tangent, $n(\tau)$ being the normal, and $b(\tau)$ being the binormal of the curve x at τ. φ is an angle in the range $[0, 2\pi)$. For discretized curves in 3D space, the tangent is approximated by the direction of the line segment between two adjacent points, and the normal and binormal are chosen as two vectors orthonormal to the tangent (cf. Stoll et al. [SGS05]). We follow the approach by Rees et al. [RLN*17] to obtain a normal and binormal vector that are rotated according to the local helicity of the flow field. For stress ribbons, two orthogonal eigenvectors of the stress tensor can be used as the normal and binormal, as proposed by Wang et al. [WNW*22] (cf. Fig. 9).

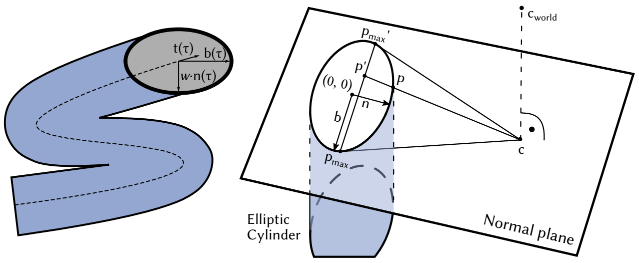

Figure 4: *Left: Generalized cylinder with elliptic profile (blue) intersected with the normal plane (gray). Right: Computation of silhouette points (p_{max}, p'_{max}) for an elliptic cylinder cross-section.*

The cross-section of an elliptic cylinder, as illustrated in Fig. 4 left, is defined as the intersection of the cylinder with a plane normal to its spanning curve. At each point along the curve, the normal plane contains the normal and binormal of the curve. Lines can be mapped to circular or elliptic cylinders by adapting the dilation factor w. For $w = 1$, the cross section of the cylinder is a circle and lines are rendered as circular cylinders. Ribbons are rendered as elliptic cylinders (i.e., the cross section of the cylinder is an ellipse). By using a dilation factor $w < 1$ in the normal direction of the ribbon, the torsion is encoded via the anisotropy of the ellipse. For $lim_{w\to 0}$, a perfectly flat ribbon geometry is obtained. By setting the cylinder thickness to a positive value, i.e., $w > 0$, it is avoided that ribbons become flat and cannot be accurately reconstructed at the given sampling frequency. In all figures in this work involving ribbons, w is set to 15% of the ribbon width. In Sec. 5, we describe how to efficiently render generalized cylinders on the GPU, by extruding a line into a polygonal representation that is then rasterized or ray traced.

3.2. Silhouette Point-based Screen-Space Outlines

Screen-space outlines (also called halos) are important visual cues to help distinguish individual lines. To render cylinders with halos in one single rendering pass, one needs to determine whether a point on a cylinder lies close to the cylinder's screen-space silhouette. When rendering elliptic cylinders, this computation becomes considerably more complicated compared to the situation where circular cylinders are used. To compute whether a cylinder point p belongs to the outline, the shortest distance of the point to the silhouette edges of the cylinder as seen from the camera position needs to be computed, and the point is drawn black if this distance is close to zero. This problem can be reduced from three to two dimensions, by projecting the camera position c_{world} into the normal plane of the cylinder for the point p with associated tangent t (cf. Fig. 4). Blinn [Bli89] shows that this dimensionality reduction does not change the distance to the cylinder silhouette edges. In the normal plane, we can now use the distance to the silhouette points p_{max} and p'_{max} of the elliptic cross-section w.r.t. the projected camera position c. The silhouette points are those points on the cylinder where the view rays touch the ellipse surface. In the following, we assume that all ellipses have unit extent in one direction, and extent w in the other direction. The extents will then be multiplied with the ribbon width.

Without loss of generality, we further assume that the center of the elliptic cross-section lies at $(0,0)^T$, and the coordinate axes are aligned with the tangent frame of the line. The world-space camera position c_{world} (assuming that the coordinate system is normalized to have its origin at the center of the ellipse) can be projected onto the normal plane, i.e.,

$$\begin{aligned} c &= c_{world} - \langle c_{world}, t \rangle \cdot t \\ &= c_{world} - \langle c_{world}, (0,0,1)^T \rangle \cdot (0,0,1)^T. \end{aligned} \tag{2}$$

The quadratic form to express that a 2D point in homogeneous coordinates $p = (x,y,1)^T$ lies on the ellipse in Fig. 4 right is

$$\frac{x^2}{w^2} + y^2 - 1 = 0. \tag{3}$$

This quadratic form can also be expressed as $p^T A p = 0$ with

$$A = \begin{pmatrix} \frac{1}{w^2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} = \operatorname{diag}\left(\frac{1}{w^2}, 1, -1\right). \tag{4}$$

The simpler case of computing silhouette points for a circular cross-section has first been described by Blinn [Bli89]. An ellipse is a special case of a conic section in two dimensions. Given the position c of the camera in the normal plane, the silhouette points p_{max} and p'_{max} are exactly those points on the conic section which are touched by the tangents of the conic section w.r.t. the camera position. For this, we can compute the polar $l = Ac$ of the camera point, which is a line that intersects the conic section in these two points [Ric11] (cf. pp. 149-154). Please note that both points and lines can be represented in 2D real projective space as homogeneous coordinate vectors with three components. In order to compute the intersection of l with the conic section, the algorithm described by Richter-Gebert [Ric11] (cf. pp. 194-196) can be employed.

First, we compute $B = M_l^T A M_l$ with $M_l = \begin{pmatrix} 0 & l_z & -l_y \\ -l_z & 0 & l_x \\ l_y & -l_x & 0 \end{pmatrix}$.
Then, with μ being the last, if non-zero, entry of l, we can compute

$$\alpha = \frac{1}{\mu} \sqrt{-\begin{vmatrix} B_{11} & B_{12} \\ B_{12} & B_{22} \end{vmatrix}}. \tag{5}$$

If the last entry of l is zero, a different sub-matrix of B needs to be selected for computing α. When μ corresponds to the n-th element of l, we need to compute the determinant from the entries of B not located at row or column n.

Let $C = B + \alpha M_l$ and (i, j) be the index of a non-zero element $C_{i,j}$ of C. Then the two intersection points p_{max} and p'_{max} are the ith row and jth column of C, respectively. Finally, we can compute $p' = \text{meet}(l, \text{join}(c, p)) = l \times (c \times p)$, which is the projection of the cylinder point p onto the silhouette. The meet operation applied on two lines yields their intersection point, and the join operation applied on two points yields their connecting line. Both operations can be expressed as cross-products on homogeneous coordinate vectors ([Ric11] (cf. pp. 52-54)). Finally, the absolute silhouette position is equal to $\lambda = \left| \frac{\|p' - p_{max}\|_2}{\|p'_{max} - p_{max}\|_2} \cdot 2 - 1 \right|$. λ becomes 1 when p coincides with the silhouette points. Thus, a black outline is drawn for $\lambda \geq 1 - \varepsilon$. The outline width ε is given in object space, thus the outline width on the screen is dependent on the size the cylinder takes up on the screen. In case it is desired, ε can also be scaled with the view distance to the cylinder point and the field of view of the camera to achieve halos with constant screen-space size.

As a side note, it is interesting that the proposed approach is also directly applicable to hyperstreamlines [DH93], where a cylinder with half-radii varying according to two mutually orthogonal direction vectors is formed along a stream line. The only difference to a ribbon is that hyperstreamlines have two separate dilation factors w_1, w_2 in the quadratic form in Equation 4 for the normal and binormal direction.

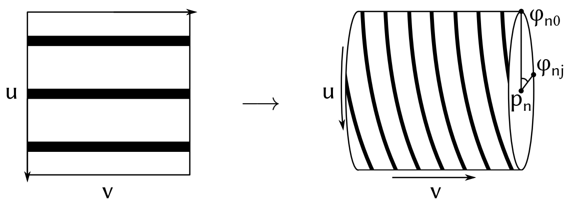

Figure 5: *Mapping stripes ($k = 3$) in texture coordinate space onto a cylinder with high rotation (without our width correction).*

4. Twist-Line Rendering

A different possibility to visualize twists along lines is to draw thin lines on the surface of an extruded cylinder with circular profile, which spin around the line with a frequency indicating the strength of the twist, e.g., the strength of the local helicity when visualizing flow fields. While helicity can also be mapped to color along the cylinder surface, an advantage of using twisting lines is that color can then be used to encode another parameter, so that relationships between twist and this parameter can be conveyed. Furthermore, the resulting visualizations can be shown effectively in black and white printing.

Twisting lines are implemented as periodic procedural line textures, which are evaluated in a fragment shader to let the lines twist on the cylinder surface (cf. Fig. 5). Procedural textures have multiple advantages over pre-computed image-based texture maps as used by Stoll et al. [SGS05]. Firstly, the user can zoom onto an arbitrarily small surface area without the finite resolution of a texture map becoming evident. Secondly, cheap anti-aliasing can be implemented using fragment shader derivatives, as, for instance, described by Schäfer [Sch15]. Thirdly, by using procedural textures, multiple additional parameters can be encoded by the colors of the regions between the twisting lines (see Fig. 2). Lastly, and most importantly, the problem of varying line thickness depending on the strength of the twist, which is paramount to using a pre-computed line texture, can be avoided. An illustration of this effect as well as the result that is obtained via a procedural texture is shown in Fig. 6.

To render lines on the cylinder surface that spiral around the cylinder axis with a frequency given by the local helicity, the accumulated helicity is used to shift the procedural texture u-coordinate according to the local twist at the cylinder axis. Therefore, the accumulated helicity $r_n = \sum_i^n h_i l_i f$ is computed for each line point in a preprocess. h_i is the helicity at the i-th line point, and l_i is the length

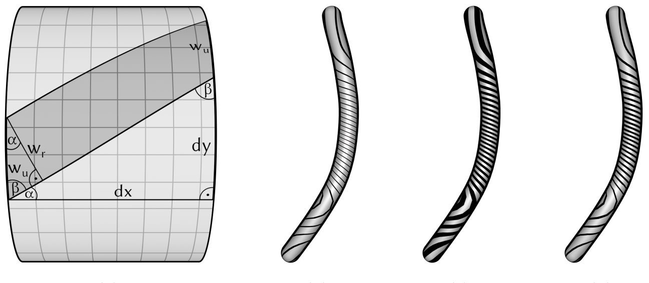

Figure 6: *a) To maintain the selected line width w_r, the texture line width w_u of the twisting line that is mapped onto the cylinder needs to be adapted. b, c) Without width correction (i.e., when using image-based texture mapping), the twisting lines become thinner in regions of high helicity. d) Constant width of twisting lines is achieved by our approach.*

of the line segment connecting the i-th line point with it's successor. f is the twist frequency factor which determines the frequency of the twist, i.e., the distance between the rendered twisting lines on the cylinder surface. Then, in the vertex shader the texture coordinate $u = (\varphi_{n,j} + r_n) \bmod \frac{2\pi}{k}$ is computed. Here, k controls the number of twisting lines, and $\varphi_{n,j}$ is the angle of the circle point on the cylinder (cf. Fig. 5), and black anti-aliased outlines are drawn if u is close to 0 using fragment shader derivatives. The texture coordinate v corresponds to the length along the cylinder. It is currently unused in the procedural texturing approach, as the stripes can also be represented as a 1D texture in u.

The described procedure results in a total of k lines twisting around the surface of the generalized cylinder with a speed depending on the helicity at the line point in which normal plane a cylinder point lies. A fragment is rendered black, i.e. as part of a twisting line, when $u \leq w_u$. We can choose the twisting line width w_u in the u-axis (i.e., along the circular cross-section) dynamically with our procedural texturing approach. However, as demonstrated in Fig. 6a), when the line twists, the real thickness of the line w_r is not equal to the width w_u of the twisting line in the u-axis. Fig. 6b,c) shows how the twisting lines appear when the line width varies depending on the strength of the twist, and Fig. 6d) shows the result of adaptive line width correction.

To counteract varying thickness depending on the twist, foreshortening of w_r is compensated by increasing w_u correspondingly. Therefore, we make use of the trigonometric equalities for the two right-angled triangles in Fig. 6a), i.e.,

$$\tan\alpha = \frac{dy}{dx} \text{ and } \cos\alpha = \frac{w_r}{w_u}, \tag{6}$$

to compute the line thickness in the tangential axis of the cylinder as

$$w_u = \frac{w_r}{\cos(\arctan\frac{dy}{dx})}. \tag{7}$$

Here, $dx = \|p_i - p_{i-1}\|_2 = l_i$ and $dy = \text{circumference} \cdot \frac{dr}{2\pi} = \frac{d_l\pi}{2\pi}dr = \frac{d_l}{2}dr$. p_i are the line points, d_l is the diameter of the cylinder (i.e., the line width), and $dr = r_i - r_{i-1}$ is the helicity difference.

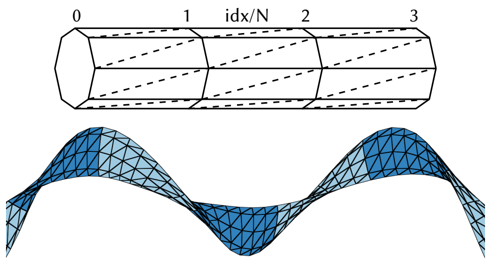

Figure 7: *Top: Indices in programmable pulling for $N = 8$. Bottom: Ribbon rendered using meshlets for $N = 8$ and $M = 7$. Segments belonging to the same meshlet are colored identically.*

dx and dy are in world-space and their direction and ratio correspond to the direction and ratio of the texture coordinate derivatives dv and du, respectively. Furthermore, it should be noted that the procedural texture coordinate u lies in the range $[0, \frac{2\pi}{k})$. If a line pattern is encoded into a texture map with coordinates in the range $[0,1)$, we can use $u = \frac{(\varphi_{n,j}+r_n) \bmod 2\pi}{2\pi}$.

5. Implementation

The proposed rendering techniques can be implemented efficiently on a wide range of graphics hardware. The open-source implementation (`https://github.com/chrismile/LineVis`) includes additional rendering options to achieve high quality, i.e., cylinders are rendered with the Blinn-Phong shading model [Bli77], and an additional depth cue is integrated by desaturating a cylinder's color slightly with increasing distance to the camera.

Many current implementations of line rendering via cylindrical shapes use geometry shaders [KFW16; KNM*21], which take a line segment as input and output the triangle geometry of the corresponding part of the cylinder. For each line, two sets of vertices on circles around each of the line endpoints are created, and triangles are formed by connecting these vertices. For the elliptic cylinders introduced in Sec. 3, we use the normal direction of the ribbon for orienting the circles before connecting the vertices on them.

Geometry shaders, however, are notoriously slow on modern GPUs [Bar15] if many vertices or polygons are generated, and have been abandoned on many mobile devices and by Apple's Metal graphics API. A replacement would be generating the cylinder triangle mesh on the CPU and uploading it to the GPU, but this would make fast updates, when changing, e.g., the cylinder thickness or radius, unfeasible. A more flexible replacement for geometry shaders are programmable pulling [Rák12] and mesh shaders. Programmable pulling stores the line data in a storage buffer in GPU memory, which can be read using random access operations. A fixed-function index buffer is used, storing the topology of the cylinder mesh. Then, in the shader, the correct line points are fetched from memory. The line point index is derived by dividing the cylinder mesh vertex index by the number of circle points (idx/N), as shown in Fig. 7 top. The circle point index can also be deduced from the vertex index ($idx\%N$). The remaining operations are analogous to those in geometry shader-based rendering, but while the geometry shader processes one line segment, the vertex shader in programmable pulling processes only one cylinder vertex.

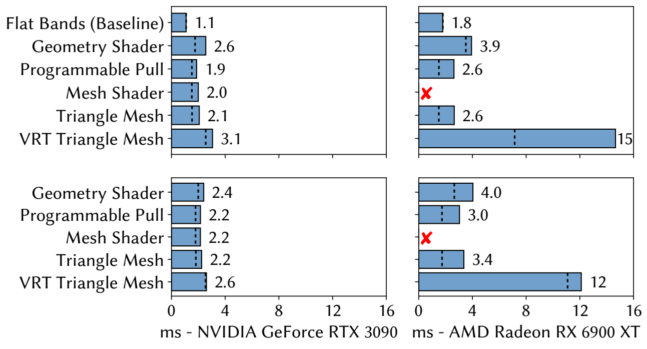

Figure 8: *Frame time in milliseconds of different implementations of elliptic ribbon (top) and twisting lines (bottom) rendering on different GPUs, measured for a centrifugal pump (Fig. 11, 986,889 line segments) at a resolution of 3840x2160 pixels. The dashed bars show the frame times when rendering plain ribbons without halos and texture-based twisting lines. Note that Vulkan mesh shaders are only supported on the NVIDIA GPU.*

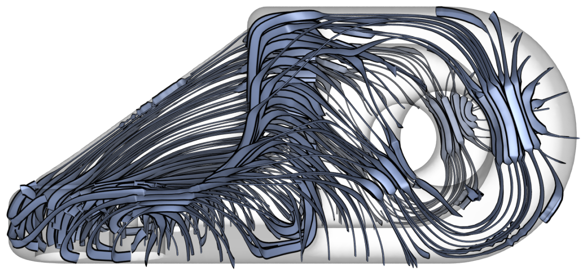

Figure 9: *Stress ribbons: Following the major principal stress direction, ribbons are oriented normal to the minor stress direction. Flips indicate where the assignment of eigenvectors to the medium and minor principal stress direction changes.*

When rendering the line data with mesh shaders, the lines are first subdivided into groups of M line segments in a preprocess on the CPU. One mesh shader work group then processes one such set of M line segments collaboratively during rendering and extrudes these line segments into a set of cylinder segments analogously to the geometry shader (cf. Fig. 7 bottom). An advantage of meshlets is that they could also be used for coarse scale visibility culling in order to improve performance. If for each meshlet the world space bounding box is stored, a so-called task shader can check whether the bounding box of the respective meshlet intersects the view frustum of the camera. If not, the whole meshlet can be immediately discarded for rendering. Another use-case of meshlets is shown in the work by Ibrahim et al. [IRR*22], where meshlets are used for fast probabilistic occlusion culling of particle data.

6. Performance Evaluation and Results

All timings are obtained on a system running Windows 10, an Intel Xeon CPU with 3.80GHz, and an NVIDIA GeForce RTX 3090. Alternatively, we have used an AMD Radeon RX 6900 XT GPU on a Xeon CPU with 3.9GHz. Rendering is to a 3840x2160 viewport.

In Fig. 8, we compare the performance of different implementations of the proposed rendering techniques for ribbons and twisting lines, based on rasterization and ray tracing. The baseline, the flat bands, are rendered using geometry shaders. All tests have been performed for the line data set in Fig. 11. The performance data is averaged for a camera path around the object. Programmable pulling and mesh shaders perform significantly better than geometry shader-based rendering. Using static triangle meshes performs similarly well when geometry does not need to be recomputed in every frame, but uses more GPU memory (approx. $\times 10$). Ray tracing-based methods provide advantages for dense data sets not tested here, as occluded geometry can be culled using an acceleration structure, while rasterization-based methods need to process the whole geometry data in every frame.

Vulkan ray tracing (VRT) builds a hardware-dependent acceleration structure on top of the generated static triangle mesh. We believe that the performance decrease when ray tracing generalized cylinders with elliptic profile is due to the used ray-tracing acceleration structure. Cylinders with elliptic profile take up more empty space in the axis-aligned bounding boxes used by the acceleration structures, resulting in more intersection candidates that do not lead to ray-geometry intersections. As can be seen in Fig. 8, the techniques presented in this work, like silhouette point-based halos and twisting lines, result in some performance decrease especially on AMD hardware, but can clearly still run in real-time.

In Fig. 9, the visualization of a stress tensor field via elliptic ribbons is shown. Ribbons can effectively encode the direction of one of the principal stress directions as well as the orientation of the two remaining principal stress directions via a ribbon's twist. In the shown example, a trajectory is computed by following the major principal stress direction in a 3D stress tensor field. At every point in this field, a stress tensor is given from which the three principal stress directions can be computed via an eigenvalue decomposition. An elliptic ribbon can then be formed by using the remaining medium and minor principal stress directions as semi-axes of the elliptic profile, and by adjusting the lengths of the axes to obtain a flat yet non-vanishing appearance.

Fig. 11 shows the use of elliptic ribbons and twisting lines to encode helicity in addition to a second variable, i.e., turbulence kinetic energy, mapped to color. The simultaneous encoding of two quantities can effectively hint to spatial correlations between them. While it is more intuitive to discern the sign of the helicity through a color mapping like in Fig. 12, it is possible to see that the ribbons and twisting lines rotate in positive direction in the inside of the rotor and in negative direction on the outer ring of the centrifugal pump due to their winding. Another advantage compared to simply mapping helicity to color is that the resulting visualizations can be shown effectively in black and white printing (cf. Fig. 13).

7. Discussion and Limitations

7.1. Performance

A slight drawback of elliptic ribbons, as show in the previous section, is the additional cost for rasterization and ray intersection tests due to the higher number of primitives. The time complexity for n triangles lies in $O(n)$ for rasterization and $O(\log n)$ for ray tracing. This means that the overhead of elliptic ribbons only contributes logarithmically to the overall cost of ray tracing, but linearly to rasterization. We believe it should be evaluated in future work whether virtual geometry approaches like *Nanite* in Unreal Engine 5 [KSW21] could be used to reduce the overhead of the rasterization of generalized cylinders. This way, the additional computational cost of the silhouette point-based halos and twisting lines also could become decoupled from the amount of geometry rendered, as the shading and lighting computations would in such a system only be performed for visible geometry.

7.2. Line Primitive Types

In this work, we have demonstrated our contributions for streamribbons and twisting lines generated from the velocity and helicity of flow fields and stressribbons generated from the eigenvectors of tensors in a stress tensor field. However, our contributions are also applicable to any other ribbon generation approach, like the vortex core ribbons by Karch et al. [KSWE16].

Streamribbons and twisting lines based on the helicity both share a common shortcoming. In Fig. 12 left, for example, we see on the leftmost side of the image many neighboring, parallel cylinders with high negative helicity, and thus twisting lines rotating with high frequency around them. The twisting lines, as well as the ribbons, may suggest the existence of particles rotating around the center streamline with a distance equal to the radius of the tube. This is, however, generally not the case. Also, the initial ribbon direction and twisting line angle is arbitrarily chosen, so the rotation of neighboring ribbons or twisting lines may be shifted. Karch et al. [KSWE16] try to solve these issues with vortex core ribbons. At the center of the vortex core ribbon lies a vortex core line instead of a streamline. The direction of the ribbon is then not computed via the helicity, but by tracing two particles in the vicinity of the vortex core line to deduce the ribbon direction. This means that the ribbon better complies with the local particle motion in close vicinity to the center line. By computing the angle between consecutive vortex core ribbon sections, the twisting lines can be generalized to use the local particle motion instead of the helicity.

In the abstract we state that the length of the cylinder's semi-minor axis can be adjusted such that the ribbons always remain visible. For a perspective projection, the cylinder can still become arbitrarily thin due to area foreshortening. If desired this problem can be solved by giving the axes of the cylinders a minimum screen space length that the ribbon thickness must not fall below.

7.3. Geometry, Shading and Lighting Continuity

In Sec. 6, we have shown how our proposed generalized cylinders with elliptic profile, silhouette point-based screen space outlines and twisting lines can be efficiently implemented by rasterizing or tracing rays against extracted triangle meshes, which are inherently only a C^0 continuous approximation of the real ribbons. By linearly interpolating the tangent vector between vertices using hardware-accelerated barycentric interpolation, C^1 continuous lighting and shading can be achieved. We would like to note that, aside from the performance evaluation, none of our contributions are dependent on the underlying geometric representation. They can also be used with implicit approaches, e.g., a generalization of the hyper-streamline sphere tracing approach by Reina et al. [RBE*06] that

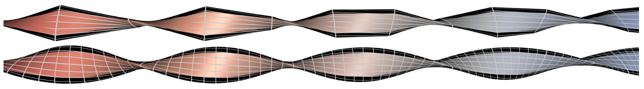
Figure 10: *Top: Triangle mesh with low number of subdivisions. Bottom: Sphere-traced elliptic ribbons.*

provides geometry, lighting and shading interpolation of higher order. As can be seen in Fig. 10, a sphere-tracing based approach can result in a smooth representation even at a low level of geometric subdivision. Recently, Pan et al. [PHB16] and Reshetov [Res22] have shown that interest exists in the development of higher-order smooth representations of flat ribbon geometry using, for example, developable surfaces and doubly ruled bilinear patches.

8. Conclusion and Future Work

In this work, we have introduced generalized cylinders with elliptic profile as a visual mapping for ribbons, to avoid aliasing and disappearing ribbon sections as occurring when ribbons are rendered as flat quads. We have shown how silhouette points of the elliptic profile can be computed, which generalizes the work by Blinn [Bli89] and enables single-pass rendering of stylized halos. A second visual mapping for twists along lines was introduced, which draws twisting lines on the surface of a circular tube to show the helicity of the underlying field. We have shown that our ribbon and twisting lines approach can be implemented efficiently on recent GPUs using features like programmable pulling and mesh shaders. In the future we want to examine how deferred shading and hierarchical culling can be used for faster rasterization of triangle meshes generated for generalized cylinders. Also, since trajectory visualization plays a significant role in meteorology [RBS*18], we plan to integrate the techniques presented in this work into Met.3D, an open-source 3D visualization tool aimed at meteorological analyses [RKSW15].

Acknowledgments

This work was supported in part by a grant from the German Research Foundation (DFG) under grant number WE 2754/10-1. The centrifugal pump flow data set is courtesy of the Institute of Applied Mechanics, Clausthal University, Germany (Dipl. Wirtsch.-Ing. Andreas Lucius).

References

[AB76] AGIN, GERALD J. and BINFORD, THOMAS O. "Computer Description of Curved Objects". *IEEE Transactions on Computers* C-25.4 (1976), 439–449. DOI: 10.1109/TC.1976.1674626 2.

[Bar15] BARCZAK, JOSHUA. "Why Geometry Shaders Are Slow (Unless you're Intel)". (2015). URL: http://www.joshbarczak.com/blog/?p=667 (visited on 06/24/2022) 5.

[Bli77] BLINN, JAMES F. "Models of Light Reflection for Computer Synthesized Pictures". *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '77. Association for Computing Machinery, 1977, 192–198. ISBN: 9781450373555. DOI: 10.1145/563858.563893 5.

[Bli89] BLINN, JAMES F. "Optimal tubes [spacecraft modelling]". *IEEE Computer Graphics and Applications* 9.5 (1989), 8–13. DOI: 10.1109/38.35533 2–4, 7.

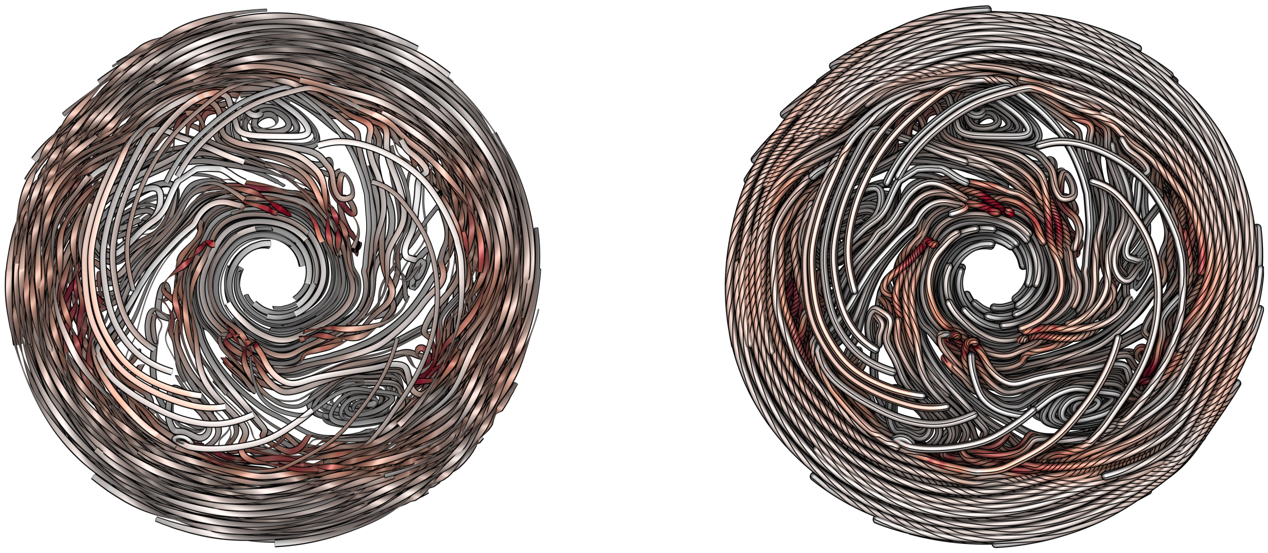

Figure 11: *Ribbons and twisting lines visualizing the flow around a centrifugal pump. Turbulence kinetic energy is mapped to color.*

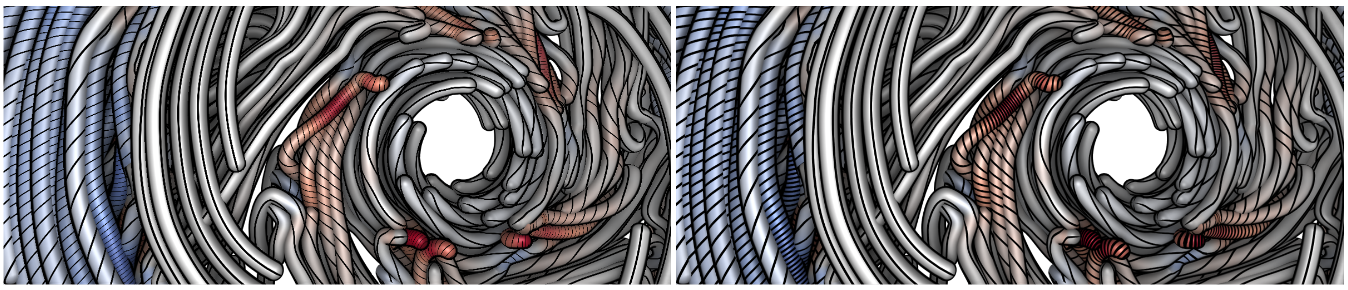

Figure 12: *Left: Image-based texture mapping. Right: Procedural texture mapping (ours) with uniform width of twisting lines and anti-aliasing via fragment shader derivatives. Helicity is mapped to color from blue (high negative) over grey (zero) to red (high positive).*

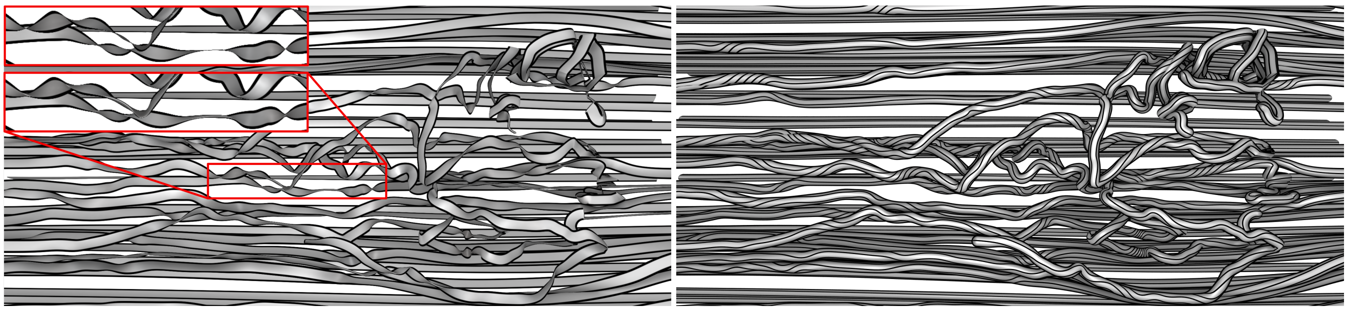

Figure 13: *Elliptic ribbons and twisting lines visualizing incompressible flow around a synthetic vessel data set [PSS04]. The close-up shows the ribbons rendered as flat quads (top) and elliptic cylinders (bottom).*

[DH93] Delmarcelle, Thierry and Hesselink, Lambertus. "Visualizing second-order tensor fields with hyperstreamlines". *IEEE Computer Graphics and Applications* 13.4 (1993), 25–33. doi: 10.1109/38.219447 4.

[EBRI09] Everts, Maarten H., Bekker, Henk, Roerdink, Jos B.T.M., and Isenberg, Tobias. "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 2.

[EHS13] Eichelbaum, Sebastian, Hlawitschka, Mario, and Scheuermann, Gerik. "LineAO—Improved Three-Dimensional Line Rendering". *IEEE Transactions on Visualization and Computer Graphics* 19.3 (2013), 433–445. doi: 10.1109/TVCG.2012.142 2.

[HWU*19] Han, Mengjiao, Wald, Ingo, Usher, Will, et al. "Ray Tracing Generalized Tube Primitives: Method and Applications". *Computer Graphics Forum* (2019). doi: 10.1111/cgf.13703 2.

[IRR*22] Ibrahim, Mohamed, Rautek, Peter, Reina, Guido, et al. "Probabilistic Occlusion Culling using Confidence Maps for High-Quality Rendering of Large Particle Data". *IEEE Transactions on Visualization and Computer Graphics* 28.1 (2022), 573–582. doi: 10.1109/TVCG.2021.3114788 6.

[KBE08] KRONE, MICHAEL, BIDMON, KATRIN, and ERTL, THOMAS. "GPU-based Visualisation of Protein Secondary Structure". *Theory and Practice of Computer Graphics*. Ed. by LIM, IK SOO and TANG, WEN. The Eurographics Association, 2008. DOI: 10.2312/LocalChapterEvents/TPCG/TPCG08/115-122 2.

[KFW16] KANZLER, MATHIAS, FERSTL, FLORIAN, and WESTERMANN, RÜDIGER. "Line density control in screen-space via balanced line hierarchies". *Computers & Graphics* 61 (2016), 29–39. DOI: http://dx.doi.org/10.1016/j.cag.2016.08.001 2, 5.

[KKKW05] KRÜGER, JENS, KIPFER, PETER, KONDRATIEVA, POLINA, and WESTERMANN, RÜDIGER. "A Particle System for Interactive Visualization of 3D Flows". *IEEE Transactions on Visualization and Computer Graphics* 11.6 (2005), 744–756. DOI: 10.1109/TVCG.2005.87 2.

[KNM*21] KERN, MICHAEL, NEUHAUSER, CHRISTOPH, MAACK, TORBEN, et al. "A Comparison of Rendering Techniques for 3D Line Sets With Transparency". *IEEE Transactions on Visualization and Computer Graphics* 27.8 (2021), 3361–3376. DOI: 10.1109/TVCG.2020.2975795 2, 5.

[KRW18] KANZLER, MATHIAS, RAUTENHAUS, MARC, and WESTERMANN, RÜDIGER. "A Voxel-based Rendering Pipeline for Large 3D Line Sets". *IEEE Transactions on Visualization and Computer Graphics* (2018), 1–1. ISSN: 1077-2626. DOI: 10.1109/TVCG.2018.2834372 2.

[KSW21] KARIS, BRIAN, STUBBE, RUNE, and WIHLIDAL, GRAHAM. "A Deep Dive into Nanite Virtualized Geometry". *In ACM SIGGRAPH 2021 Courses, Advances in Real-Time Rendering in Games, Part 1* (2021). URL: https://advances.realtimerendering.com/s2021/index.html (visited on 08/24/2022) 7.

[KSWE16] KARCH, GRZEGORZ K., SADLO, FILIP, WEISKOPF, DANIEL, and ERTL, THOMAS. "Visualization of 2D Unsteady Flow Using Streamline-Based Concepts in Space-Time". *J. Vis.* 19.1 (Feb. 2016), 115–128. DOI: 10.1007/s12650-015-0284-z 3, 7.

[MSE*06] MERHOF, DORIT, SONNTAG, MARKUS, ENDERS, FRANK, et al. "Hybrid Visualization for White Matter Tracts using Triangle Strips and Point Sprites". *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), 1181–1188. DOI: 10.1109/TVCG.2006.151 2.

[MTHG03] MATTAUSCH, OLIVER, THEUSSL, THOMAS, HAUSER, HELWIG, and GRÖLLER, EDUARD. "Strategies for Interactive Exploration of 3D Flow Using Evenly-Spaced Illuminated Streamlines". *Proceedings of the 19th Spring Conference on Computer Graphics*. SCCG '03. Association for Computing Machinery, 2003, 213–222. DOI: 10.1145/984952.984987 2.

[NHK*22] NEUHAUSER, CHRISTOPH, HIERONYMUS, MAICON, KERN, MICHAEL, et al. *Visualization of Model Parameter Sensitivity along Trajectories in Numerical Weather Predictions*. 2022. DOI: 10.48550/ARXIV.2205.02268 2.

[PHB16] PAN, ZHERONG, HUANG, JIN, and BAO, HUJUN. "Modelling Developable Ribbons Using Ruling Bending Coordinates". *CoRR* abs/1603.04060 (2016). arXiv: 1603.04060 7.

[PSS04] POPINET, STÉPHANE, SMITH, MURRAY, and STEVENS, CRAIG. "Experimental and Numerical Study of the Turbulence Characteristics of Airflow around a Research Vessel". *Journal of Atmospheric and Oceanic Technology* 21.10 (2004), 1575–1589. DOI: 10.1175/1520-0426(2004)021<1575:EANSOT>2.0.CO;2 8.

[PSS18] PANDEY, AMBRISH, SCHEEL, JANET D., and SCHUMACHER, JÖRG. "Turbulent superstructures in Rayleigh-Bénard convection". *Nature Communications* 9.1 (May 2018). ISSN: 2041-1723. DOI: 10.1038/s41467-018-04478-0 2.

[PW94] PAGENDARM, HANS-GEORG and WALTER, BIRGIT. "Feature detection from vector quantities in a numerically simulated hypersonic flow field in combination with experimental flow visualization". *Proceedings Visualization '94*. 1994, 117–123. DOI: 10.1109/VISUAL.1994.346329 2.

[Rák12] RÁKOS, DANIEL. "Programmable Vertex Pulling". *OpenGL Insights*. Ed. by COZZI, PATRICK and RICCIO, CHRISTOPHE. http://www.openglinsights.com/. CRC Press, July 2012, 293–301 5.

[RBE*06] REINA, GUIDO, BIDMON, KATRIN, ENDERS, FRANK, et al. "GPU-Based Hyperstreamlines for Diffusion Tensor Imaging". *EUROVIS - Eurographics /IEEE VGTC Symposium on Visualization*. The Eurographics Association, 2006. DOI: 10.2312/VisSym/EuroVis06/035-042 2, 7.

[RBS*18] RAUTENHAUS, MARC, BÖTTINGER, MICHAEL, SIEMEN, STEPHAN, et al. "Visualization in Meteorology—A Survey of Techniques and Tools for Data Analysis Tasks". *IEEE Transactions on Visualization and Computer Graphics* 24.12 (2018), 3268–3296. DOI: 10.1109/TVCG.2017.2779501 7.

[Res22] RESHETOV, ALEXANDER. "Ray/Ribbon Intersections". *Proc. ACM Comput. Graph. Interact. Tech.* 5.3 (July 2022). DOI: 10.1145/3543862. URL: https://doi.org/10.1145/3543862 7.

[Ric11] RICHTER-GEBERT, JÜRGEN. "Perspectives on projective geometry. A guided tour through real and complex geometry". Springer Science & Business Media, Jan. 2011. ISBN: 978-3-642-17285-4. DOI: 10.1007/978-3-642-17286-1 4.

[RKSW15] RAUTENHAUS, M., KERN, M., SCHÄFLER, A., and WESTERMANN, R. "Three-dimensional visualization of ensemble weather forecasts – Part 1: The visualization tool Met.3D (version 1.0)". *Geoscientific Model Development* 8.7 (2015), 2329–2353. DOI: 10.5194/gmd-8-2329-2015 7.

[RLN*17] REES, DYLAN, LARAMEE, ROBERT S., NGUYEN, DUONG, et al. "A Stream Ribbon Seeding Strategy". *EuroVis 2017 - Short Papers*. The Eurographics Association, 2017. DOI: 10.2312/eurovisshort.20171135 3.

[Sch15] SCHÄFER, SEBASTIAN. "Using fwidth for distance based anti-aliasing". (2015). URL: http://www.numb3r23.net/2015/08/17/using-fwidth-for-distance-based-anti-aliasing/ (visited on 06/27/2022) 4.

[SGS05] STOLL, CARSTEN, GUMHOLD, STEFAN, and SEIDEL, HANS-PETER. "Visualization with stylized line primitives". *VIS 05. IEEE Visualization, 2005.* 2005, 695–702. DOI: 10.1109/VISUAL.2005.1532859 2–4.

[SPS06] SADLO, FILIP, PEIKERT, RONALD, and SICK, MIRJAM. "Visualization Tools for Vorticity Transport Analysis in Incompressible Flow". *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), 949–956. DOI: 10.1109/TVCG.2006.199 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 (1996), 100–110. DOI: 10.1109/2945.506222 3.

[Vol89] VOLPE, G. "Streamlines and streamribbons in aerodynamics". *27th Aerospace Sciences Meeting*. 1989. DOI: 10.2514/6.1989-140 2, 3.

[WJA*17] WALD, INGO, JOHNSON, GREGORY P., AMSTUTZ, JEFFERSON, et al. "OSPRay - A CPU Ray Tracing Framework for Scientific Visualization". *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017), 931–940. DOI: 10.1109/TVCG.2016.2599041 2.

[WNW*22] WANG, JUNPENG, NEUHAUSER, CHRISTOPH, WU, JUN, et al. "3D-TSV: The 3D Trajectory-based Stress Visualizer". *Advances in Engineering Software* 170 (2022), 103144. DOI: https://doi.org/10.1016/j.advengsoft.2022.103144 3.

[ZNT*18] ZHANG, LEI, NGUYEN, DUONG, THOMPSON, DAVID, et al. "Enhanced vector field visualization via Lagrangian accumulation". *Computers & Graphics* 70 (2018). CAD/Graphics 2017, 224–234. DOI: https://doi.org/10.1016/j.cag.2017.07.014 3.

[ZSH96] ZÖCKLER, MALTE, STALLING, DETLEV, and HEGE, HANS-CHRISTIAN. "Interactive Visualization of 3D-Vector Fields Using Illuminated Stream Lines". VIS '96. IEEE Computer Society Press, 1996, 107–ff. 2.