On-Tube Attribute Visualization for Multivariate Trajectory Data

Benjamin Russig, David Groß, Raimund Dachselt, Stefan Gumhold

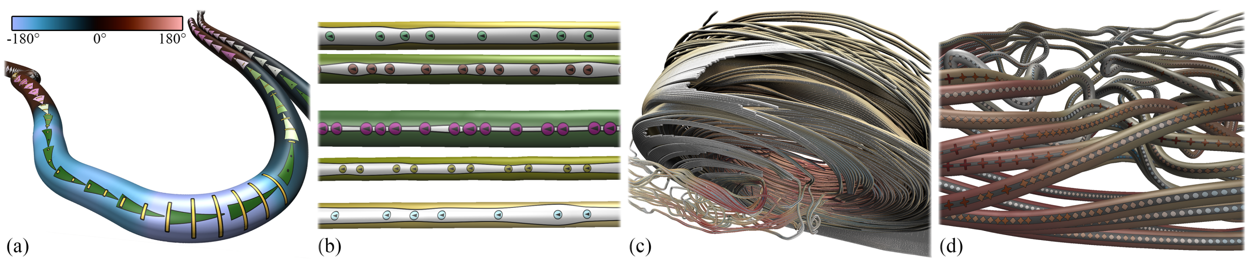


Fig. 1. Examples of multivariate on-tube visualizations. **(a) Vehicles:** steering angle is mapped to color (cf. color scale); *Triangles* show velocity mapped to length and acceleration mapped to color and inversely to base width (wide and purple depict braking actions); thin yellow *Rectangles* are used to show the throttle amount. **(b) Delivery Drones:** using color to visualize velocity (yellow = faster); *Line* plot shows power usage; *Circles* used as labels to identify drones. **(c,d) Fisch Wehr:** overview and detail visualizations of streamlines. Color shows velocity (red = faster); a *Line* plot is used to visualize high pressure areas; *Sign* glyph shows angular velocity (shape and color). Ambient occlusion emphasizes the vortex structure in (c).

Abstract— Stylized tubes are an established visualization primitive for line data as encountered in many scientific fields, ranging from characteristic lines in flow fields, fiber tracks reconstructed from diffusion tensor imaging, to trajectories of moving objects as they arise from cyber-physical systems in many engineering disciplines. Typical challenges include large data set sizes demanding for efficient rendering techniques as well as a large number of attributes that cannot be mapped simultaneously to the basic visual attributes provided by a tube-based visualization. In this work, we tackle both challenges with a new on-tube visualization approach. We improve recent work on high-quality GPU ray casting of Hermite spline tubes supporting ambient occlusion and extend it by a new layered procedural texturing technique. In the proposed framework, a large number of data set attributes can be mapped simultaneously to a variety of glyphs and plots that are embedded in texture space and organized in layers. Efficient rendering with minimal data transfer is achieved by generating the glyphs procedurally and drawing them in a deferred shading pass. We integrated these techniques in a prototype visualization tool that facilitates flexible mapping of data set attributes to visual tube and glyph attributes. We studied our approach on a variety of example data from different fields and found it to provide a highly adaptable and extensible toolbox to quickly craft tailor-made tube-based trajectory visualizations.

Index Terms—Visualization, Rendering, Line Data, Trajectories, Multivariate Data.

1 Introduction

Modern tracking and sensor technology facilitates the collection of large amounts of movement data to capture the ever increasing complexity of systems with non-stationary entities. Acquiring such trajectory data is common in a number of different fields. Typical data sources include the movement of land [1, 24], air [2, 3] and water vehicles [25], migratory animals [44] or entities in scientific simulations [35]. The path of an object is recorded over time as a sequence of (geo-)spatial positions that form a two- or three-dimensional trajectory.

Stylized tubes have become the de-facto standard for visualizing this kind of data as demonstrated by a large body of research applying them for this purpose, but they are inherently limited in their capacity to show more than spatial aspects of the data. However, trajectories are often enriched with additional data originating from sensor measurements, manual annotations or contextual cues, or data simply derived from other attributes. Examples include physical properties such as velocity, acceleration and orientation, internal states like battery status and engine load, or semantic data like object type and events. In general, a multitude of additional internal or external attributes specific to certain research fields are possible. As the amount of dense multivariate trajectory data increases, data analysts and domain experts employ diverse visualization techniques that help them understand the complexity of the data and gain insights into underlying structures. For analysis tasks like finding anomalies or inspecting trends and correlations with the spatial data, it is essential to visualize a subset of the available attributes simultaneously with the spatial behavior of trajectories.

This work extends the idea of using color as a visual attribute by mapping additional data attributes to various representations rendered on the tube surface, including glyphs for discrete samples along the tube and various continuous mappings like heatmaps and plots. We will refer to this concept as *On-Tube Visualization* throughout this paper. We believe this concept to be extremely powerful in a wide range of scientific fields where trajectories and other line data are commonly studied. Consequently, we must be able to deal with the diverse and sometimes conflicting requirements they pose. Advanced multivariate visualization capabilities need to be integrated without compromising the spatial intuition provided by existing tube-based visualizations. We especially do not want to sacrifice recent advancements in terms of visual fidelity that enable improved spatial perception.

In this work, we present a domain-agnostic toolbox that offers the flexibility to quickly create use-case driven multivariate trajectory visualizations for both small and large scale data sets, and we lay the technical foundations necessary to achieve this. We provide a smart view-dependent parameterization approach to minimize distortion of glyphs and to maximize their visibility at the same time. Light-weight data management on the GPU and procedural generation of glyphs ensures high performance and enhances the flexibility of the system.

High quality textures are achieved by generating them on-the-fly in the shading stage. For this, glyph rendering is based on the evaluation of signed distance functions, opening up the use of a near-infinite

• *Benjamin Russig, David Groß and Stefan Gumhold are with the Chair of Computer Graphics and Visualization, TU Dresden. E-mail: benjamin.russig|david.gross1|stefan.gumhold@tu-dresden.de*
• *Raimund Dachselt is with the Interactive Media Lab, TU Dresden. E-mail: raimund.dachselt@tu-dresden.de*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

range of shapes to render glyphs and plots from, while at the same time enabling effects such as pre-filter anti-aliasing and outlines. We limit the performance impact of this procedural approach by employing a deferred shading pipeline, which also enables other advanced per-fragment effects like ambient occlusion to be performed efficiently. Geometry anti-aliasing for deferred shading is supported via standard temporal- and screen-space methods. The geometry pass of the pipeline can be switched to an image-order ray caster that utilizes hardware ray tracing cores, further extending the scalability to massive data sets.
To summarize, our main contributions are:

- A powerful approach to building use-case driven, high quality tube-based visualizations of multivariate line data sets:
  - *Efficient*: a large number of attributes can be visualized directly in their spatial context, reducing the need for additional views.
  - *Flexible*: layers and procedural glyphs allow for quick adaptation to use cases.
  - *Extensible*: any glyph or plot that can be expressed in terms of a signed distance function could be incorporated.
- A high-performance spline tube rendering architecture:
  - efficient anti-aliased deferred shading pipeline for spline tubes with ambient occlusion, either using a highly optimized rasterization strategy or hardware RT cores to cast rays, enabling best possible performance for small and large data sets alike
  - requires few pre-computations, all of which are fast (ambient occlusion density field, parameterization, glyph placement, construction of acceleration structure when using hardware RT cores)

We also provide an interactive prototype that demonstrates both the flexibility and realtime performance of the methods we present. We used this prototype in the validation of our approach with various data sets and found that already few procedural glyphs and plots in combination with the layer concept provide the power to easily create expressive visualizations for many different contexts. Examples of such visualizations can be seen in Fig. 1. The large configuration space combined with the interactive speeds of all involved calculations make us confident that both visualization researchers and domain experts will benefit from this approach.

2 Related Work

2.1 Multivariate Data Visualization

Ware defines data as a set of entities, their attributes and relations [43], with the number of attributes possessed by an entity determining its complexity. Entities with a complexity higher than two are referred to as *multivariate data*. Finding suitable visual mappings for the given attributes to allow for good perception of value and relations is the main concern of visualizing such data. Different authors have investigated visual properties and their mappings to attributes of varying types. The works from Cleveland and McGill [5] and Mackinlay [27] provide the groundwork for the usage of graphical primitives in visual representations of data. Beyond design guidelines, they rank the effectiveness of visual properties in perceptual tasks based on the type of data (quantitative, ordinal, or nominal).

For the visualization of multivariate data, several surveys and overview works have been conducted [18, 26, 45, 47]. These reviews provide valuable summaries and categorizations of the various existing techniques and state of the art. Early works like the one by Wing and Chang [45] have focused on the various aspects of visual encodings. Ward [42] presents an overview of glyphs that are especially well-suited to represent multivariate data. The author defines glyphs as graphical objects or symbols that represent data entities and map their attributes to graphical properties. Later surveys aim to provide general summaries of the possibilities of high-dimensional data visualization as a whole. Liu et al. [26] provide a coherent overview including classifications of methods for data transformation, visual mapping and view transformation. Methods for feature classification, fusion visualization and correlation analysis specific to spatial data are presented by He et al. [18]. Additional to the more general surveys, Zhou and Weiskopf [47] review methods for the visualization of multivariate particle data sets. They also evaluate the usage of general multivariate visualization methods like scatter plots and parallel coordinates in that context. Staib et al. [35] employ so-called flow ribbons, which combine animation with abstract views of the data while preserving relation to the spatial domain. For general time-dependent data in multidimensional state spaces, Grottel et al. [11] adapt well-known tools for multivariate visualization and propose continuous-time scatter plot matrices and parallel coordinates plots. Since these do not visualize time information directly, they further introduce temporal heat maps and combine these modalities into coordinated views. We adapt their temporal heat map as color bands on the tube surface.

2.2 3D Trajectory Visualization

A multitude of methods to visualize trajectory data have been proposed, a detailed review of which is given in a recent survey by He et al. [17]. The spatial extent of three-dimensional trajectories is typically conveyed using lines, tubes, ribbons or consecutive arrangements of primitives, as presented by Buschmann et al. [4]. Their hardware accelerated attribute mapping enables changes to the visualization parameters in real-time. Data attributes are mapped directly to visual properties of the geometric representations, and texture mapping is used to display additional attributes by varying pattern density, stretch or animation speed. In a follow-up work, they applied this approach to visualize animated air-traffic data [3], where arrows depict the movement direction of aircraft. Tominski et al. [37] propose using stacked trajectory bands to visualize space, time and attribute values simultaneously. They use color mapping to depict attribute values and integrate time through the ordering of bands. For vector fields, Stoll et al. [36] also utilize color, tube radius and texture mapping to visualize streamline attributes. Both approaches indicate the general direction of movement using arrow textures. Contrary to that, Vrotsou et al. [39] indicate the direction of paragliding trajectories using tapered segments. Further, they propose simplification methods for the visualized trajectories. One drawback of this approach is that it does not allow for texture mapping in a coherent manner, thus one has to rely only on color mapping.

For trajectories attributed with orientation values, ribbons are a suitable choice of representation. Ware et al. [44] use such ribbons to visualize the underwater movement of humpback whales. Texture mapping on the flat sides of the ribbon conveys the swimming direction with a 2D chevron pattern. To visualize the angular velocities, they use 3D sawtooth glyphs that are placed onto the ribbons. More involved use of texture mapping for encoding spatial properties of the trajectories is possible as well, as shown by Ritter et al. [32]. Here, the density of procedurally generated circle or stroke patterns is varied to indicate relative distances between trajectories or to the viewer.

Common to all these techniques is a strong focus on the spatial aspects of the data, with usually one or two additional characteristic attributes included for context. On the other hand, methods dedicated to multivariate data visualization have limited capability to provide intuitive spatial insights, especially on larger scales. We see a strong opportunity for improvement here by combining techniques from both fields in an economic manner.

2.3 Tube Rendering

Various methods have been proposed to generate tube-like geometry from discrete polyline representations of trajectories. In the standard approach, a fully tessellated tubular mesh is generated around a given trajectory. Cross-section vertex rings located around the sample points are connected to form a generalized cylinder, as described by Ueng et al. [38] and Zhang et al. [46], implying a *swept-disc* type tube volume which – depending on the use case – requires special care for points with extreme inflection or G^1-discontinuous segment connections. With shading, realistic results are produced that allow for good depth perception and, with appropriate parameterization, easy texturing.

As the amount of geometry quickly becomes a bottleneck for large data sets, other approaches use imposters that give the impression of tubes when combined with a lighting approximation [8, 23, 28, 30, 34, 36]. Perhaps most relevant to us from this family of methods is work done by

Everts et al. [9]. They present stylized wide lines with depth dependent halos and depth cueing, and later extend this work to allow the mapping of various texture patterns modulated by data attributes [10].

Simple impostors without depth information might fall short of delivering realistic results at geometry intersections. As an alternative, direct ray casting of tubes can be used. Reshetov and Luebke [31] proposed a fast iterative algorithm for computing the approximate intersection of a ray with a tube following a polynomial curve. Their algorithm is fast and delivers high-quality results using a *swept-disc* type tube model. Han et al. [16] introduced a piece-wise linear tube primitive defined as a union of spheres interconnected by tangential cones, and provide a method for efficiently composing this union within a raytracing framework. Being effectively *swept-sphere*, their model naturally handles junctions and corners. Groß and Gumhold [12] proposed a rasterization-based pipeline for GPU ray casting of a similar piece-wise linear tube primitive with support for correct transparency and ambient occlusion. They achieve high geometry throughput by using screen-space proxy quadrilaterals for casting rays, while the fast GPU visibility sorting necessary for transparency increases fill-rate utilization. Russig et al. [33] proposed a *swept-sphere* Hermite spline tube primitive that can be efficiently ray casted, enabling smooth, continuous tubes similar to [31]. They also propose a rasterization-based pipeline, but use oriented bounding boxes as proxy geometry, negatively affecting geometry throughput.

Recently, it has been shown that image-order ray casting has a clear scalability advantage over rasterization. Kanzler et al. [20] proposed a voxel-based ray-marching pipeline suitable for GPUs that can easily cope with extremely large data sets, although their piece-wise linear tube model cannot generally represent data sets exactly due to the inherent discretization, and is thus best suited for overview visualizations. Kern at al. [22] compared the efficiency of several image-order ray casting and rasterization-based tube rendering strategies in the context of transparency. Notably, they confirmed that utilizing the hardware ray tracing (RT) cores available in the latest generation of GPUs offers superior performance at scale, especially when only the primary rays are required. Specifically in the context of tubes, Wald et al. [40] have shown that utilization of hardware RT cores can be optimized even further by exploiting the GPU ray transformation facilities that are used for supporting hierarchies of acceleration data structures to essentially enable free oriented bounding box rejection tests, greatly benefiting user-defined line/curve primitives.

3 On-Tube Visualization

To our knowledge, there are currently no methods described in the literature that allow for a significant number of additional scalar attributes to be visualized for multi-variate trajectories without assuming restrictions on their spatial complexity or using additional linked views. On-tube visualization tackles this challenge by making use of the space offered by the tube primitives to display additional visualizations. In this section, we will elaborate on this concept.

3.1 Design Goals

Our main objective is to enable interactive visualization of as many additional scalar variables as possible from highly multivariate 3D trajectories (also called *traces*) directly in their spatial context. The defining feature of trajectories over more general line data is their parameterization in terms of time. While we explicitly target trajectories, the method should be general enough to be applicable to all kinds of line data from many scientific fields, including those that typically deal with very large data set sizes. For this, rendering performance is as important as aiding structural perception, which are potentially conflicting goals.

The additional attributes visualized alongside the spatial behavior of a trace should mainly provide better context, but ideally they should also help analysts identify interesting developments in the data that would otherwise remain hidden when inspecting a purely spatial view. Nevertheless, we do target analysis tasks for which spatial aspects are of prime importance, so the added elements must not cause clutter or introduce occlusion issues. Finally, we do not aim for the ability to get

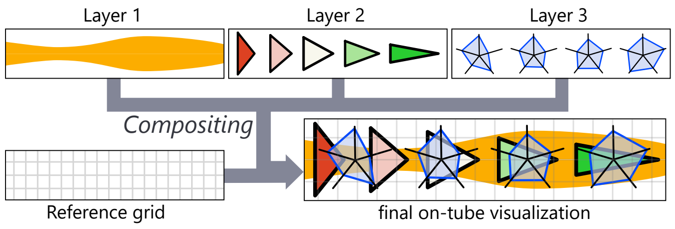


Fig. 2. Schematic view of on-tube visualization including the optional procedural reference grid texture. This example includes: *(1)* a *Line* plot in the bottom layer. *(2) Triangle* glyphs placed as densely as possible at the original sample locations of the mapped attributes. For contrasting the white areas of the color map with the white background, a thick outline is applied. *(3)* equidistantly spaced *Star* glyphs sampling additional five data attributes. Compositing is performed by alpha blending the layers in the order of their definition on top of the reference grid.

exact readings of quantities from the visualization (we feel this is best served by on-demand linked detail views, the integration of which we deem out of scope in this work). They should instead help revealing local trends and anomalies, correlations or divergences between spatial and non-spatial behavior, as well as relative quality. In this work, we develop the technical framework needed to craft such visualizations.

3.2 Design Rationale

Tube primitive. The choice of line primitive forms the main dimension of the design space for any 3D trajectory visualization. We consider the use of tubes a premise because of the widespread familiarity many domain experts have with them. We do believe their popularity is warranted though, as they are a direct and intuitive representation of the volume a moving object has covered along its path, and in some fields even directly correspond to the physical object they represent (e.g. blood vessels or nerve fiber tracts). While ribbon-like primitives trivially support correct texturing, they either favor certain viewing directions or require non-trivial view-alignment that can still break down in areas where trajectory and viewing direction become parallel. Nevertheless, we strive in this work to adapt the desirable properties of ribbon parameterization to tube texturing. The curved surface of a tube does pose a challenge when using it as a canvas for visualization, but we feel confident that the surface parameterization we present in Sect. 4.3 sufficiently mitigates the inherent distortions to make the design goals laid out in the previous section achievable.

The particular choice of tube primitive is limited by the need for a continuous canvas to draw on. The piecewise linear cone-sphere combination prevalent in line data visualizations is problematic here, since it inherently introduces singularities to cylinder-like parameterizations, as the spheres connecting two segments correspond to single points on the axis. On the other hand, recent work has shown that ray casting of tubes extruded from parametric curves can be implemented fast enough for real-time applications, so they are a viable choice [31, 33]. As a curve representation, Hermite splines in particular lend themselves well, as velocity vectors are a common trace attribute, which in turn opens up possibilities of major data-guided reduction for the purpose of rendering (which we did not look at in this work).

Glyph and plot-based visualization. Trace data is usually discretized. Attributes are being sampled at finite time or space intervals, and often the rate at which they are being sampled varies. Additionally, some data types such as events are meaningless to interpolate. Glyphs are an adequate tool for such data as they can be placed exactly at the point on the trajectory where the sample was taken. However, care must be taken in the case of on-tube visualization. While the usual considerations discussed in the literature regarding perception of geometric features like size, area or angles apply, the issues they pose are exacerbated by the curved tube surface. Additional limitations apply to the use of orientation: Directions other than forward or backward become somewhat disconnected from the space they are embedded in. Smooth view-alignment of the visualization around the tube inherently requires that there be no preferred reference direction, thus relative

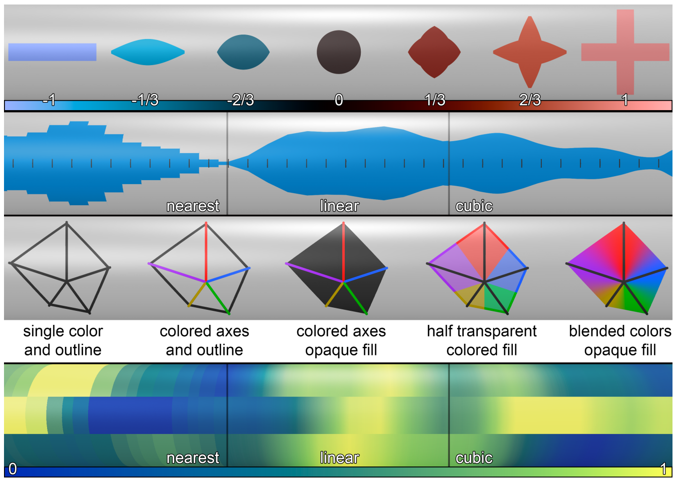


Fig. 3. Example configurations of the more complex glyph types and continuous mappings we implemented. **1st row:** *Sign* glyph with diverging color (see color scale), showing the full value range from -1 to 1. **2nd row:** *Line* plot with nearest, linear and cubic attribute interpolation. **3rd row:** five configurations of a 5-axis *Star* glyph showing the colorization capabilities. **4th row:** *Temporal heat map*, again with the three interpolation modes, showing values from 0 to 1. All interpolation takes place before mapping. Color maps used in 1st and 4th from [6] (berlin, imola).

directions like "left" or "right", "up" or "down" in texture space do not necessarily agree with the corresponding directions in the original 3D reference frame of the traced object.

The selection of glyphs we implemented in our prototype takes these considerations into account. In addition, we implemented continuous mappings as an alternative for scalar attributes that are very densely sampled or which are safe to interpolate. Table 1 lists the glyphs and plots we have implemented so far. Our selection is in no way exhaustive and we do not claim an optimal choice at this point. Rather, we regard it as part of a proof-of-concept effort for demonstrating the feasibility and usefulness of on-tube visualization. As we will show in Sect. 4.4, the framework allows for easy integration of new glyph and plot types.

Layers. We propose to stack glyphs and plots in layers as a way to organize the visualization and group together related attributes or those that are especially well-suited for a certain visualization type. Furthermore, they provide a clear way of defining a z-order for compositing, for which we currently employ simple alpha blending. However, they also provide a clear singular extension point for adding more advanced compositing techniques later on (e.g. into other channels like surface normal or reflectance properties of physically based lighting models).

Layers also serve to configure the glyph or plot they are set to draw. Table 2 lists the general options we currently support.

Reference grid. To counter both perspective distortions when embedding visualizations in 3D space as well as enabling comparison of glyph proportions over longer distances, we added the option of a reference grid. Since the color channel can easily become overloaded when superimposing several layers, the grid can be purely normal-mapped, or use a tunable combination of color and normal-mapping. Fig. 2 gives a schematic overview of the on-tube visualization concept.

3.3 Renderer Design

We opted to include both object-order and image-order ray casting using the OpenGL rasterization pipeline and the NVIDIA OptiX hardware ray tracing API [29], respectively. While ray tracing will scale better for massive data sets, rasterization still has a performance advantage for a relevant range of small to medium data set sizes in our experiments. Thus, we argue that it is an important option to have, even when hardware support for ray tracing is present.

[1]In general, only one attribute per glyph can be sampled at its original location. The remaining ones will be interpolated.

Circle	1 generic quantity (radius) or marking of sample positions for other layers.
Rectangle	2 generic quantities (width, height) or constant-size tick marks.
Triangle	indicate movement direction and visualize 2 related quantities (width, height) like speed and acceleration.
Sign	1 quantity that oscillates around zero (directly supports negative values), changes shape smoothly from '$-$'-like to '$+$'-like (see Fig. 3)
Star	simple implementation of star coordinates [19] for multiple comparable scalar attributes (see Fig. 3).
Line plot [35]	maps up to n quantities to thickness of n lines (we currently support $n = 1..4$), ideal for densely sampled comparable scalar attributes (see Fig. 3).
Temporal heat map [11]	maps up to 4 quantities to tube color. Useful even for a single attribute as it bypasses the limited sampling rate (at the spline nodes) of the tube color visual attribute (see Fig. 3).

Table 1. Glyph and continuous mapping types implemented for this paper and their recommended usage. Each glyph also supports color mapping in addition to its explicitly mentioned visual attributes.

sampling	original[1], uniform time, equidistant
interpolation type	nearest, linear, cubic
outline	on/off, thickness
color	fixed or color-mapped
attribute $1..n$	fixed value or data source, windowing params

Table 2. Configurable options per layer. We made two exceptions: neither *Line* plots nor the *Star* glyph can use a color scale.

For rasterization, we build upon the Hermite spline tube pipeline by Russig et al. [33] and improve it with established GPU ray casting optimizations to further increase render performance. Regardless of the choice of render method, we expect their *swept-sphere* tube model to be more robust with real-life trajectory data due to the natural support for G^1 discontinuities and tolerance to areas of extreme inflection that sweeping a sphere offers, but we also include the highly efficient intersection routine by Reshetov and Luebke [31] when *swept-disc* tubes are unproblematic and rendering speed is deemed most important.

Since the shading operations required to realize on-tube visualization can become quite complex depending on the number of layers and glyph types used, we must take care that no unnecessary work is done on fragments that are later discarded. This is not an issue for image-order ray casting, but cannot be avoided in rasterization unless the work is deferred to a separate shading pass. Since deferred shading prevents use of multi-sampling by the GPU, we use *Temporal Anti-Aliasing* (TAA, [21]) as a standard approach.

Finally, we opted against using two completely separated renderers. OptiX allows for near-seamless integration into the OpenGL-based renderer, which in turn greatly simplifies the overall software architecture by reducing duplicate code. Furthermore, we do not see a clear benefit from simulating global illumination effects other than ambient occlusion, which can be adequately implemented in rasterization. While on-tube visualization targets closer distances, AO is strongly required to aid spatial perception at overview scales (see Fig. 1 (c)). Future work may investigate combining on-tube-visualizations with transparency, which might warrant a purely ray tracing-based renderer.

4 Concept and Implementation

This section details the technical aspects of our approach. We start with an overview of the rendering process and then discuss the most important components in detail.

4.1 Overview

Our pipeline, as illustrated in Fig. 4, has two render paths, enabling the use of both rasterization and ray tracing hardware to perform ray casting (the latter we will refer to as the RT path for short). The rasterization path shares all pre-processing steps with the RT path, while the latter additionally requires building an acceleration bounding volume hierarchy (BVH), which is implemented within the OptiX API.

First, loaded data is converted to an internal representation. We assume spatial trajectories sampled over time with tangent vectors that are automatically estimated with finite differences if not contained in the input data. Tube radius is kept constant to avoid scaling of glyphs.[2] Each trajectory is organized into nodes storing position plus tangent, and Hermite segments indexing their start and end node. An arbitrary number of additional data series is supported. Their sampling is not constrained to the position nodes but can have an independent, often denser and also varying sampling rate. When the RT path is utilized, we need to provide axis-aligned bounding boxes for each segment to the ray tracing API, which we compute from the node data according to the chosen radius (see Sect. 4.2). The rasterization path consumes the node data without further pre-processing in the form of line lists; the necessary proxy geometry (illustrated in Fig. 5 (a)) is generated on-the-fly. Once all geometry is uploaded to the GPU, the scene is voxelized into a density volume, which is later used to apply ambient occlusion (AO) using a technique similar to [7]. We implemented this process as a highly parallel GPU algorithm to allow for fast re-computation when loading a new data set.

Glyphs are sampled at discrete time values and are specified in a local 2D coordinate system over the range $[-1,1]^2$. To map the glyphs onto the tube surface, we locate the glyph center based on its time value and our (u,v) surface parameterization. The parameterization is described in Sect. 4.3, and the process of creating and placing glyphs is detailed in Sect. 4.4. Once the parameterization is calculated on the CPU, it is transferred into a GPU-side buffer for access by the shading stage. Glyph data and information about which glyphs overlap a segment are also uploaded to the GPU in two buffers that need updating after every change in the mapping configuration.

With all pre-processing completed, rendering is started via the chosen render path. Rasterization and RT paths share a common output interface and are thus exchangeable modules for the geometry stage in our deferred shading architecture. The per-pixel attributes *surface color* (as emitted from the tube primitive), *hit position*, *hit normal*, *spline tangent*, *segment index* and *texture coordinates* (calculated with the help of the parameterization within each render path) are stored in a G-Buffer. The shading pass determines the final unlit color for each pixel by drawing up to four glyph layers as well as the optional and potentially normal-mapped reference grid on top of the surface color. Glyph drawing happens by evaluating their signed distance function (see Sect. 4.4). Shading is finalized with local lighting, with the AO term being evaluated globally using the previously computed density volume. The resulting image is then anti-aliased using standard TAA.

4.2 Ray Casting

In the following, we will describe important implementation details for each render path.

Rasterization path. As is typical for object-order GPU ray casting, the primitives are approximated by proxy geometry that, in the best case, tightly encloses the silhouette of the original shape. The complexity of this proxy geometry greatly impacts the efficiency of the method, and herein lies the main optimization opportunity compared to [33]. It is important to note that unlike for the RT path, splitting curves into smaller segments has a direct negative impact on GPU geometry load, so emitting as little proxy geometry as possible is key to make rasterization scale well with data set size.

Following the original approach, the geometry shader is responsible to generate proxy geometry for the quadratic sub-segments within a Hermite segment. Our major optimization is to compute the extents of a view-aligned oriented bounding box instead of one that is fitted in object-space. This enables using the front-facing side (which will cover

[2]We initially set the tube radius to 1/4 of the average distance between position samples and make this factor user-adjustable.

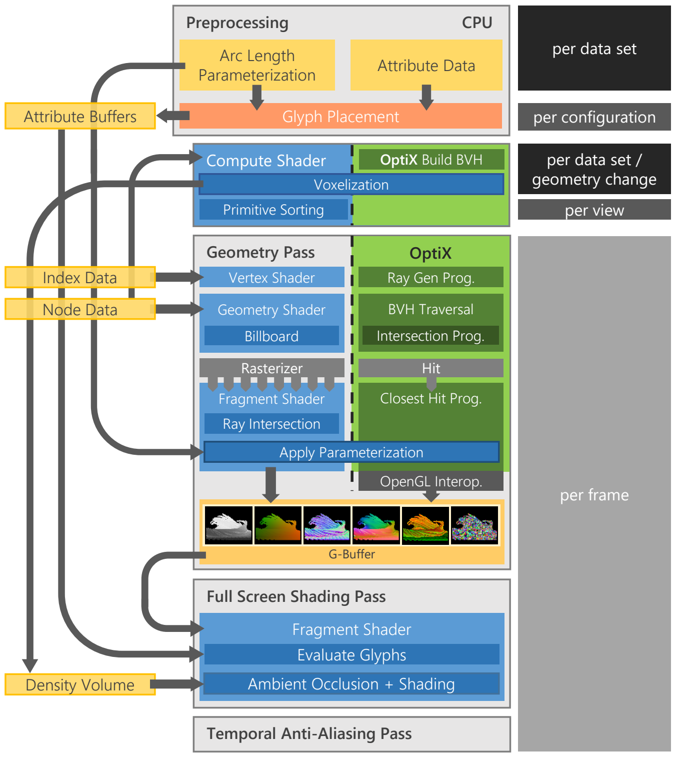


Fig. 4. Schematic illustration of the data flow and two-pass rendering process with its alternative render paths using either highly optimized rasterization or hardware-accelerated ray tracing.

the whole tube) as a view-aligned quadrilateral, reducing the number of emitted triangle strip vertices from 14 to 4.

Let $\underline{b}_1, \underline{b}_2, \underline{b}_3$ be the control points of a sub-segment positional Bézier curve, $\underline{c}_b$ their centroid and $\hat{d}_b \propto \underline{b}_3 - \underline{b}_1$ their main normalized direction. The orthonormal basis vectors of a view-aligned coordinate system can then be defined as ($\hat{\cdot} \propto$ denoting normalization):

$$\hat{z} \propto \underline{c}_b - \underline{e}\,, \quad \hat{y} \propto \vec{d}_b \times \hat{z} \quad \text{and} \quad \hat{x} \propto \hat{z} \times \hat{y}, \tag{1}$$

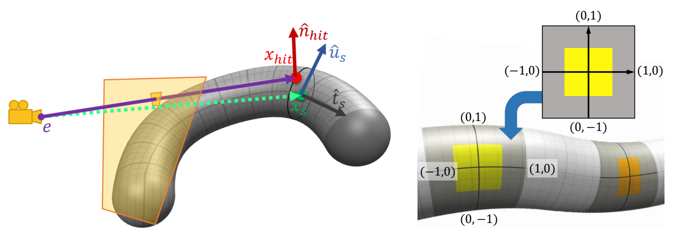
where $\underline{e}$ is the eye position. Using these basis vectors, we construct a 3×3 rotation matrix $R = [\hat{x}, \hat{y}, \hat{z}]$. The control points are then transformed into the local coordinate system by moving the centroid to the origin and applying the inverse rotation:

$$\underline{b}'_{1,2,3} = R^T \cdot (\underline{b}_{1,2,3} - \underline{c}_b). \tag{2}$$

Calculating the bounding box extents can then be performed on $\underline{b}'_{1,2,3}$ following the procedure detailed in [33], however, only the face normal to the local $\hat{z}$ direction needs to be emitted.

The modified proxy geometry still lies completely in front of the actual primitive geometry. Thus, the *conservative depth* extension of the rasterization pipeline can be used to reclaim the early-z test. To capitalize on this, primitives are drawn front-to-back by sorting them in visibility order. Sorting is performed whenever the view changes a certain amount using a GPU-based compute shader implementation of a fast parallel radix sort [14]. We found that 36 degrees of orbital movement around the camera focus are a reasonable threshold for triggering a re-sort. To enable TAA with rasterization, we apply sub-pixel jitter to the projection matrix each frame using a 2D Halton pattern [15, 21].

RT path. Thanks to OpenGL interoperability, OptiX enables accelerated image-order ray casting to be used as a drop-in replacement for the rasterization path in our pipeline. However, it requires axis-aligned bounding boxes to be provided when custom primitives are used. We can use the same procedure as outlined above, but we use the world-coordinate system as the target reference frame instead.



(a) (b)

Fig. 5. **(a) Object-order ray casting.** A ray through a fragment of the silhouette quad intersects the tube in $\underline{x}_{hit}$, which corresponds to spline point x_s on tube center line. Normalized direction vectors are surface normal $\hat{n}_{hit}$, spline tangent $\hat{t}_s$ and screen up direction $\hat{u}_s$. **(b) Glyph mapping.** Glyphs are specified over the domain $[-1,1]^2$ and mapped to a tube surface location based on our parameterization.

Ray casting results are transferred into the G-Buffer using fast GPU-internal memory transfers. To enable TAA, we choose jittered sub-pixel locations for casting rays using the same pattern as with rasterization.

4.3 Spline Tube Parameterization

For tube surface parameterization, our goals are to minimize distortion and to maximize glyph visibility from an arbitrary viewing direction. There are three major sources of distortion: tube parameterization, tube curvature, and perspective projection. To minimize impact of parameterization, we re-parameterize according to trajectory arc length. The orthogonal direction around the tube could also employ circular arc length as shown in Fig. 6 on the right side, but tube curvature results in a distortion that increases towards the silhouette. We compensate for this with the screen aligned parameterization shown on the left of Fig. 6. Glyph visibility can be maximized by aligning the v-origin on the surface with the center of arcs corresponding to fixed u parameter values as illustrated by the dark brown dotted line in Fig. 7 (a).

v coordinate. The center of Fig. 6 illustrates the computation of v on an orthogonal cut through the tube, where the tube tangent points towards the reader. Imagine that the eye is located to the left along the x-axis. We can compute the screen-aligned normalized up direction $\hat{u}_s$ (see also Fig. 5) according to

$$\hat{u}_s \propto \hat{t}_s \times (\underline{x}_s - \underline{e}). \tag{3}$$

From the screen aligned up direction, v computes to

$$v = \langle \hat{n}_{hit}, \hat{u}_s \rangle\,. \tag{4}$$

v can be easily computed in the fragment shader.

u coordinate. To compute the u coordinate from the tube parameter t_{hit} at a hit point, we need fast access to an arc length parameterization $s_j(t)$ for each tube segment j, which we assume to be parameterized over $t \in [0,1]$. We follow the same strategy as proposed by Walter and Fournier [41]. They observed that the arc length parameterization can be accurately approximated by optionally splitting the domain into two parts and representing each part of the parameterization by a cubic polynomial computed by a fit to four equidistant samples t_i of a precise numeric approximation of $s(t_i)$. They propose to split the domain at potential inflection points. Here, we take a more GPU-friendly approach that avoids conditional execution by splitting the domain equidistantly into four spans $g^{j,k=0\dots3}$ as illustrated in Fig. 7 (b). To compute the polynomials $g^{j,k}(l)$ in Bernstein basis,

$$g^{j,k}(l) = (1-l)^3 \cdot b_1^{j,k} + 3l(1-l)^2 \cdot b_2^{j,k} + 3l^2(1-l) \cdot b_3^{j,k} + l^3 \cdot b_4^{j,k} \tag{5}$$

each parameterized over $l \in [0,1]$, we first evaluate $s_i = s(t = {}^{i}/_{12})$ for $i = 0\dots12$ using 10-point Gauss-Legendre quadrature. From this the Bernstein coefficients compute to

$$b_1^{j,k} = s_{3k} + \sigma_j, \qquad b_4^{j,k} = s_{3k+3} + \sigma_j \tag{6}$$
$$b_2^{j,k} = {}^1/_6(18 s_{3k+1} - 9 s_{3k+2} + 2) + \sigma_j \tag{7}$$
$$b_3^{j,k} = {}^1/_6(-9 s_{3k+1} + 18 s_{3k+2} - 5) + \sigma_j. \tag{8}$$

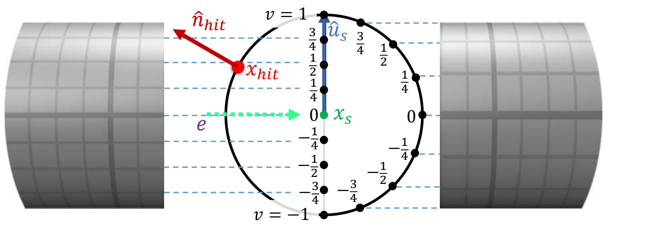


Fig. 6. Illustration of the two strategies to parameterize tube surface around center curve. Left: equidistant in screen space; right: arc length parameterization.

where σ_j is the sum over all segments before segment j. Finally, u is computed from

$$u = g^{j,\lfloor 4t \rfloor}\big(4(t \bmod {}^1/_4)\big)/R, \tag{9}$$

where we divide by tube radius R to ensure that the on-tube glyph coordinate systems illustrated in Fig. 5 (b) are uniformly scaled. We upload the coefficients $b_{1\dots4}^{j,k}$ in one 4x4 matrix per segment to the GPU.

Additional Considerations. Even though we do not expect this to negatively impact the effectiveness of on-tube visualizations, we note that Hermite splines are only C^1 continuous at segment transitions, causing the continuity of surface normal and v parameterization to reduce to C^0. This can lead to noticeable kinks in v-parameter iso lines and mapped glyphs.

The parameterization currently ignores the end caps at the beginning and end of a trajectory, which can cause artefacts like in Fig. 9, where the yellow *Star* glyph axis is mapped to the complete end cap hemisphere due to the u singularity there. The end caps of *swept-disc* tubes will instead take on a single color as their planar nature causes a singularity in v. We leave the resolution of both effects to future work.

4.4 Attribute Visualization

As in most visualization tools, we define a windowing function for each mapped attribute based on the value range in the data and an output range defaulting to $[0,1]$. The user can adjust both ranges and if desired, reverse the output range for a windowing function with negative slope.

Glyph placement. Each layer supports a different glyph type. As each glyph combines several attributes that can be sampled differently, we support nearest, linear and cubic interpolation for re-sampling before mapping to visual attributes. While we support placing glyphs equidistant in time or trajectory arc length, we argue for a variable spacing that maximizes space utilization and reduces the need for interpolation, as illustrated in Fig. 8 for a moderately complex setup.

The basic idea is to instantiate glyphs per layer in a single pass through each trajectory such that no two glyphs overlap on the tube, iterating over all unique timestamps of all attributes (A1,A2,A3 in Fig. 8) mapped in the current layer. By default, glyphs are placed in a greedy manner: The first glyph is placed at the earliest possible timestamp, and subsequent glyphs are placed at the next timestamp among all involved attributes where no overlap in u-extent with the previous glyph occurs. For equidistant spacing, the glyph locations are simply chosen independently of attribute sample locations. Even without parallelization over trajectories, this $O(n)$ algorithm executes in less than ${}^1/_3$ of a second for all but our two largest test cases, where our benchmark configuration (see Sect. 6) causes a noticeable one-time lag of 0.8 and 2 seconds respectively. This would pose a minor annoyance for very large datasets, but an optimized parallel GPU-implementation could drastically reduce execution time.

Regardless of the chosen placement strategy, the result of the placement algorithm is **a)** an array of glyph information per layer consisting of glyph center u_{ctr} and extent $[u_{\min}, u_{\max}]$ as well as the vector of involved data attribute values with interpolation and windowing applied, and **b)** annotations for every segment storing a range of indices into **a)** that mark which glyphs overlap with it. Both arrays are provided in GPU-side buffers for shading.

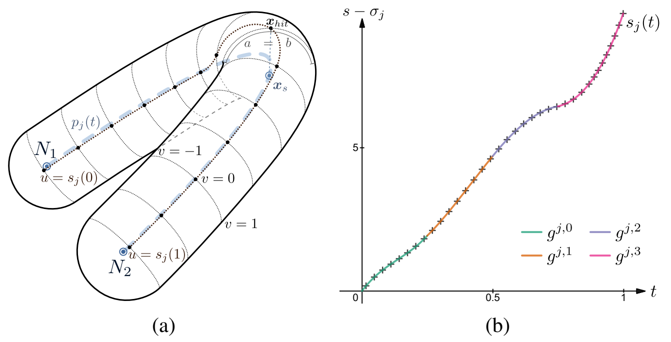


Fig. 7. **(a) Parameterization.** Behavior of the parameterization for some Hermite tube segment j, defined by the tube axis/position curve $p_j(t)$ (central dashed line) connecting nodes N_1, N_2. The dotted line going through the point x_{hit} marks where $v = 0$. It bisects all cross-section disk arcs on the tube surface into 2 arcs of equal length (a and b for the disk around x_s). v runs from -1 to 1 between the opposing tube silhouette lines. Note that the screen-space direction of v depends on the screen-space direction of the curve tangent at that point. **(b) Arc length approximation.** Plot of $s_j(t)$ vs. precise measurements of the arc length of some Hermite segment j. The graph is colored according to the polygonal spans $g^{j,k}$ being used in each range of t. The Hermite control point matrix of the curve used for this example is $K_j = \left[(-1.5,0,-1)^T \;\; (-1.5,0,12)^T \;\; (-16,0,21)^T \;\; (2,0,1)^T\right]$

Parameters u and v are computed as described in Sect. 4.3 for every ray intersection and are provided to the fragment shader in the G-Buffer (see Sect. 4.1). The shader can now check $u_{\min} < u < u_{\max}$ on the two closest glyphs associated with the segment to find which, if any, is covering the fragment. The closest glyphs are in turn found on-the-fly via binary search. In case a glyph is found, glyph-local coordinates $\underline{\tau} = (u - u_{\text{ctr}}, v)$ are computed and passed on to glyph instance evaluation, which provides color and opacity of the glyph at the fragment. Continuous mappings are handled slightly differently:

Here, ”glyphs” represent control points and have zero width. Thus, the algorithm generates an attribute vector at all unique timestamps. For shading, either two or four adjacent attribute vectors are collected depending on the requested interpolation scheme.

Procedural glyphs and plots. Fig. 3 shows the two discrete glyphs (*Sign* and *Star*) as well as two continuous mappings (*Line* and *Heat map*) illustrating the different interpolation strategies. Continuous mappings are evaluated in (u,v) coordinates in the same way as if drawn in a 2D visualization.

The discrete glyphs are procedurally constructed without generation of any geometry data. Most glyphs are based on evaluation and combination of signed distance functions (SDFs) together with a color selection strategy. SDFs provide algebraic distance from the glyph outline and indicate whether the sampling point is inside through their sign. Drawing of outlines is also straight-forward by selecting the glyph color inside and the outline color around the zero values (see Fig. 8). For glyph internal antialiasing of color edges we exploit the `fwidth` and the `smoothstep` functions of GLSL. In the following, we detail the evaluation of the *Sign* and *Star* glyphs, since they exemplify the power and flexibility that SDFs award to the framework.

Let $f_-(\underline{\tau})$, $f_\circ(\underline{\tau})$ and $f_+(\underline{\tau})$ be SDFs describing a rectangular shaped minus symbol, a circular shape zero symbol and a cross shaped plus symbol. Simple linear interpolation with parameter $\alpha \in [-1,1]$ allows mapping of a signed attribute:

$$S_a(\underline{\tau}) = \begin{cases} -af_-(\underline{\tau}) \,+\, (a+1)f_\circ(\underline{\tau}), & \text{if } a \leq 0 \\ (1-a)f_\circ(\underline{\tau}) \,+\, af_+(\underline{\tau}) & \text{otherwise} \end{cases} \tag{10}$$

Perceptual scaling can be added by a simple re-parameterization of α, which we leave for investigation in future work.

The *Star* glyph is an example of a complex cascade of *union* operations on lines and optionally, a polygon filling the area enclosed by the star coordinates. The union can simply be implemented by a minimum over the SDFs of the composing shapes. We additionally use the distance to the line primitives for selecting different colors per axis and for the interior, as well as for defining a blending area within which these colors should smoothly blend into each other. When rasterization is used, deferred shading effectively limits the performance penalties such complex constructs incur, as the measurements in Sect. 6 confirm.

Reference grid. To help judging relative sizes and distances, we implemented a procedural 2-level reference grid. The spacing of the grid lines are user-configurable, but by default they are chosen such that they create square-looking cells, with a multiple of 4 minor ticks for every major tick. For normal mapping, we implemented two relief models. The simpler of the two creates V-shaped troughs, the other attains smooth-looking results via sine-wave shaped normal perturbation at the cost of a small but measurable performance impact (see Sect. 6). The configurations in Fig. 10 (c and f) show an example of the grid being employed in an on-tube visualization.

4.5 Prototype Tool

To test our approach, we created a prototype implementation capable of loading data sets from various formats. The tool was implemented as a plugin to the CGV-Framework [13] using the OpenGL API, following the flow detailed in Sect. 4.1. We also already leverage the framework's plugin architecture to enable quick extension with newly supported data formats, and it could be leveraged to enable extension with new glyph and plot types as well.

Fig. 9 shows a screenshot of the tool interface. Data sets are loaded using the graphical user interface (GUI) or simply via drag & drop, which triggers arc length re-parameterization and voxelization. The GUI is further used to manage glyph and plot layers. To support the large amount of possible visual mapping combinations, the implementation programmatically generates and compiles the fragment shader on demand, which is fast and only necessary when changing layers or glyph types. Glyph placement is also done whenever these actions occur. Value changes like adjusting color maps or mapping parameters are interactive and directly reflected in real-time on the visualization.

5 Case Studies

Multivariate trajectory data emerges from a variety of applications. The tool just presented has the potential to create rich and diverse on-tube visualizations for such data. In the following, we examine three use cases, including land and air vehicles and a flow simulation, covering a wide area of possible applications of our approach. The example configurations presented are intended to show the flexibility of the tool; we do not yet claim optimality for the data sets at hand.

Car Movement Data. Computerization has transformed the way manufacturers control the state of modern motor vehicles. Nowadays, combustion engines need precise controlling to meet the strict emission regulations imposed by governments. Ongoing digitalization enables the read-out of sensor data and internal state in the form of on-board diagnostics (OBD). The *NOx Emissions* (NOx = Nitrogen Oxides) data set is a recording of OBD data, originating from a 100 km long test drive through cities, rural areas and over highways. Fig. 10 (a, b and c)

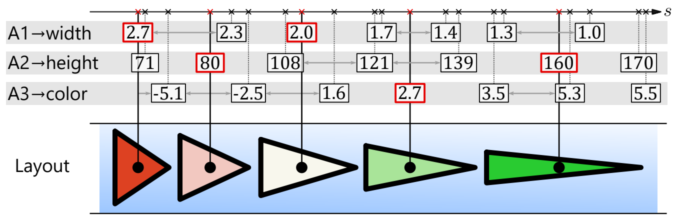


Fig. 8. Glyph placement at original samples: the first sample among mapped attributes A1, A2 and A3 that corresponds to a point on the trajectory where the glyph would fit decides the location of the glyph. The remaining attributes are interpolated (indicated by the grey arrows) before computing the value for the visual attribute.

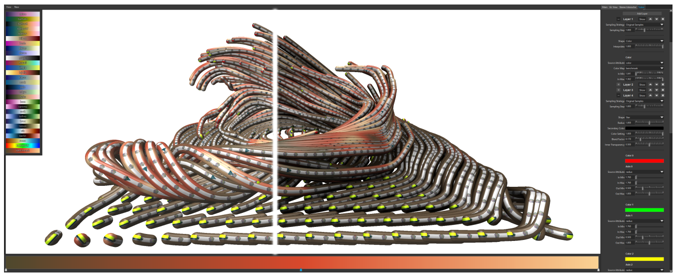


Fig. 9. Our tool in action, showing the benchmark glyph configuration on *Hot Room* (left side is without and right side with AO). To the right is the UI, showing glyph setups for the color and *Star* layer. Custom color maps can be added directly in the tool using the editor at the bottom. The widget to the left shows the preset color maps, mostly from [6].

show different glyph configurations applied to this data set. The small box in (b) shows an overview of the whole route.

In (a), we studied the relations between throttle (white *Rectangle* height), fuel flow and emissions (blue and red *Line* plots). Tube color shows velocity, with yellow being faster. Fuel flow usually increases after throttle is applied, which is to be expected, producing a subsequent rise of emissions. However, the bottom row shows significantly more exhaust gases produced for a similar amount of throttle and fuel used. Since the speed is similar, we guess this results from higher engine speed in a lower gear – unfortunately these attributes were not included. Depending on traffic, the driver or automatic transmission could have chosen a higher gear in the bottom scenario to reduce emissions.

Another mapping was applied in Fig. 10 (b and c), again showing emissions but this time using both available sensors. A *Star* is used to allow quick shape-based perception of correlations between throttle amount (red), fuel flow (blue) and emission (grey). The *Temporal heat map* displays the NOx aftertreatment state (black means active) as a single attribute. Movement direction is depicted by small white *Triangles*. Color helps to quickly find interesting areas in (b), where a close inspection can be performed. It seems that the fuel flow and subsequently the emissions are delayed, probably because throttle response is not instantaneous. Still, emissions rise with the amount of burned fuel, albeit sometimes with a discrepancy between the two sensors. However, we have also seen areas with longer time spans of throttle applied and no significant fuel flow or NOx gases produced. Overall, aftertreatment happens more often than expected – especially in contrast to the regeneration of the particulate filter – and is usually performed in small intervals, sometimes with very short time in between.

Fig. 10 (d) shows *Vehicles*, which is another data set of vehicle movement data (see also Fig. 1 (a)). Here we employed the *Temporal heat map* to simultaneously show engine rpm, throttle amount and brake action. It is interesting to see that the driver did not break when entering the sharp turn, probably because the car was already driving slow but also there must have been no immediate oncoming traffic on the entered lane. Throttle abruptly stops when preparing a gear change, while engine rpm stayed high until the shift action was executed, as illustrated by the color-coded *Circle* glyphs. Given that the car just entered a road after a sharp turn, this might not be the best time to interrupt acceleration (cf. the white arrow at the same spot in Fig. 1 (a)). The sharp white *Triangles* show the indicator usage, which is visualized using a *Star* glyph, using its "left" and "right" axes (relative to the tube direction). While this is not a typical use case for a *Star* glyph, it once more shows the flexibility of our tool and the provided glyph types.

Drone Movement Data. The *Delivery Drones* data set compares various speed settings for parcel drones flying along a test trajectory (payload is equal for all drones). An example visualization is shown in Fig. 1 (b), using tube color to map velocity (yellow is faster). A *Line* plot shows the used current of the battery. We did not aim at a deeper analysis of the data nor did we find special features, yet this example illustrates the use of the *Circle* glyph as a marker for identification. Further it shows the importance of the outline to visually separate the individual mappings which improves perception.

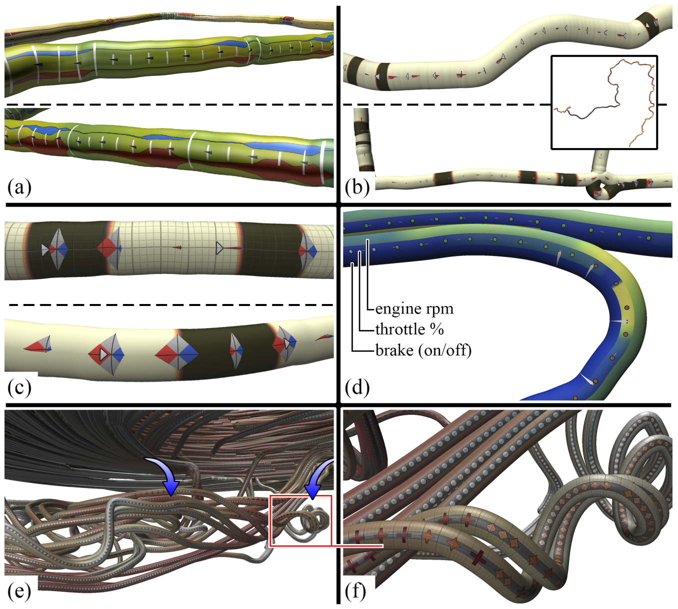


Fig. 10. Various examples of layer configurations supported by our tool. **(a, b and c)** are from *NOx Emissions*. **(d)** shows the *Vehicles* data set. **(e and f)** are views on the turbulent region of *Fisch Wehr*.

Fluid Simulation Data. The *Fisch Wehr* data set is a collection of streamlines originating from an etho-hydraulics (comparative behavior research in hydraulics) simulation of a fish friendly wier combined with a turbine generator. The goal is to enable safe up and down passage of small fish over a 1 meter height difference. This requires low velocities and minimal turbulence in the outflow regions of the lower turbine part. Fig. 1 (c) shows an overview of the whole swirl structure with color coded velocities. White depicts slow velocities, while red shows fast moving currents (3 m/s). The outer part of the swirl is rather slow and can be deemed safe for passage. The turbine outflow shows turbulent areas, which are shown up-close in Fig. 10 (e, f) (blue arrows loosely depict the outflow walls). A grey *Line* plot maps the pressure, while the *Sign* glyph visualizes angular velocity. The outflow region has fluctuating and, as is expected, overall high velocities combined with frequent changes in angular momentum. Note especially (f), where a vortex is formed with strong angular velocities to the left. Based on the visualization we judge that fish may be able to easily swim through the upper part of the structure, yet have trouble migrating upwards when entering the difficult outflow passage.

6 Performance Analysis

Performance tests are being conducted to evaluate the real-time capabilities of our approach to render potentially large amounts of segments with complex glyph mappings. The test system was equipped with an Intel Core i9-9900K with 8×3.6 GHz, 64 GB RAM and an NVIDIA GeForce RTX 2080 Ti with 11 GB of VRAM. To evaluate both overview and close inspection scenarios, we use a *far* and *near* view configuration – *far* showing the whole data set in the viewport and *near* close up views. Average frame times were recorded for a 5 s long full camera rotation around the data set using a Full HD (1920×1080) viewport. Local illumination was enabled for all measurements.

We first compare the base performance of the different rendering methods without any visualization mappings. To evaluate our improvements to the rasterization path, a comparison to the previous method [33] using their intersection routine and original OBB proxy without sorting was conducted. Two of the data sets used in the original work were included for reference: *HotRoom* (convection streamlines) and *Furball* (synthetic). To test scalability, we additionally included two larger synthetic data sets *2 Mil* and *5 Mil* which are structurally very similar to *Furball*. The results are shown in Fig. 11. Following the re-triggering strategy, our rasterization path performed sorting a total of 10 times for the full camera rotation. Single sorting times per data set

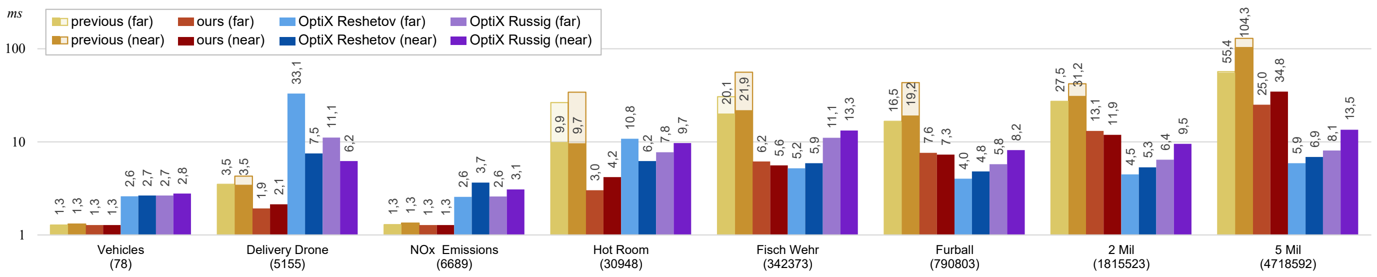
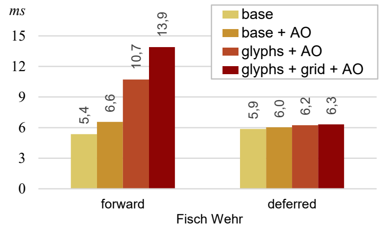


Fig. 11. Average render times (in milliseconds) of the eight evaluated data sets for far and near views using a logarithmic scale. Segment counts are given in parentheses. Numbers for previous state times with early-z culling enabled, while the transparent area depicts frame times without.



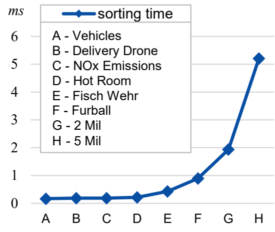


Fig. 12. **Left:** Comparing the influence of AO, glyph and grid mapping on the frame times for *Fisch Wehr* (averaged over both view configurations). A complex configuration of 4 glyph layers was used with a color- and normal-mapped grid. **Right:** Sort timings in milliseconds.

are plotted in Fig. 12 (right). Our rasterization method tends towards a 3× times speedup over the previous method for sufficiently large data sets, even with conservative depth enabled for both. This can be mainly attributed to reduced geometry load due to the simplified billboard, improving performance especially in zoomed out views of large data sets. Differences between far and near views are far less pronounced and only become noticeable for the *5 Mil* data set. We believe this to be the result of the optimized draw order due to visibility sorting in conjunction with early-z culling, which helps reducing the impact of fill rate limitations. Additionally, more consistent frame times over all view directions are achieved.

The OptiX render path shows mostly consistent scaling across all data set sizes. However, for smaller data sets up to *Hot Room*, our rasterization path performs consistently better, with *Delivery Drone* and *Hot Room* being a particularly challenging scenario for the former. We believe this is caused by a suboptimal BVH layout due to many small overlapping segments. The Reshetov intersection routine is up to twice as fast as the one by Russig, but shows severely degraded performance in some data sets, which we hypothesize contain adversely shaped segments negatively affecting its convergence. The advantage of rasterization on up-to medium-sized data sets can likely be attributed to overhead caused by BVH traversal. *Fisch Wehr* marks the break-even-point, where our rasterization and hardware ray tracing (Reshetov) achieve almost equal render times. For even larger data sets, the inherent scalability advantage of ray tracing becomes the dominant factor. In general, we believe our rasterization approach to be well-suited to lower-end hardware without RT support.

With the performance baseline determined, the influence of our shading operations was measured to validate the choice of a deferred pipeline. The results are shown in Fig. 12 (left) for *Fisch Wehr*; other data sets showed comparable behavior. For the glyph mappings, a complex configuration consisting of 4 layers with *Temporal heat map*, *Line plot*, *Triangle* and 5-axis *Star* glyph was chosen for a total of 8 mapped attributes (cf. Fig. 9). A color- and normal-mapped grid was used. Deferred shading shows superior scaling with each additional effect enabled, causing only a slight increase by 12% on average for our test viewport size. We conclude from this that early-z culling cannot fully prevent overdraw, causing forward rendering performance to degrade with each additional effect, resulting in up to 120% increased frame times with all effects enabled.

Temporal anti-aliasing is applied after the shading pass and takes around 0.9 ms to evaluate for our viewport configuration.

7 Discussion and Outlook

The prototype tool shows that the flexibility of our approach enables quick adaptation to multivariate data from diverse fields. In this work it was not our intention to create optimal visualizations for given data and tasks, but to lay the foundations that allow visualization experts in collaboration with domain experts to create effective, tailored on-tube visualizations. For future work, our tool facilitates the rigorous evaluation of such specific solutions. The performance measurements show that our method is applicable to large data sets comprising millions of segments, and that glyphs and plots do not introduce performance issues. This makes us confident that more computationally involved and/or specialized on-tube visualizations can be realized in the future.

A few issues remain unresolved, however. On-tube visualization in its current form breaks down for trajectories that contain longer stationary periods. Land vehicle trajectories are especially prone to this, and the *Vehicles* and *NOx Emissions* data sets contain several such instances. For the future, we propose superimposing these parts with a billboard showing a view-aligned time series plot of the evolution of mapped attributes within the affected period.

While our parameterization mitigates visibility and length distortion problems associated with texturing a 3D tube for visualization, perceptual issues related to perspective distortions remain unsolved. Here, orthographically projecting glyphs onto the tube in screen space could be a viable alternative in some situations. Additionally, the kinks in the parameterization that can appear at C^2 discontinuities between segments could be eliminated by blending tangents of neighboring segments within a small blending zone when calculating v.

Finally, we observed that it is still possible to quickly overload visualizations with too many attributes even in just four layers. Utilizing surface features other than color (e.g. normals similar to the reference grid, or physically based material properties) has the potential to push the boundaries a little further, which may warrant an extensive user study verifying the perceptual limits of the layer-based approach.

8 Conclusion

We proposed *On-Tube Visualization*, a flexible and extensible approach to tube-based multivariate trajectory visualization. We contributed a technical framework realizing this approach based on a high-performance spline tube rendering architecture. We were able to validate the approach by creating useful visualizations for diverse data sets, thereby demonstrating the powerful solution space. Performance measurements confirm that the technical foundation provides a sound basis to apply and further develop the approach in the future, where we hope it will help visualization- and domain experts to fully realize the potential of rich and expressive multivariate trajectory visualizations.

Acknowledgments

This work received funding from the Deutsche Forschungsgemeinschaft through DFG grant 389792660 as part of TRR 248, the Clusters of Excellence CeTI (EXC2050/1 grant 390696704) and PoL (EXC2068 grant 390729961), and from the German Federal Ministry of Education and Research via the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI) Dresden/Leipzig (BMBF, 01/S18026A-F).

References

[1] S. Al-Dohuki, Y. Wu, F. Kamw, J. Yang, X. Li, Y. Zhao, X. Ye, W. Chen, C. Ma, and F. Wang. Semantictraj: A new approach to interacting with massive taxi trajectories. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):11–20, 2017. doi: 10.1109/TVCG.2016.2598416

[2] N. Andrienko, G. Andrienko, J. M. C. Garcia, and D. Scarlatti. Analysis of flight variability: a systematic approach. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):54–64, 2019. doi: 10.1109/TVCG.2018.2864811

[3] S. Buschmann, M. Trapp, and J. Döllner. Animated visualization of spatial–temporal trajectory data for air-traffic analysis. *The Visual Computer*, 32:371–381, Mar. 2016. doi: 10.1007/s00371-015-1185-9

[4] S. Buschmann, M. Trapp, P. Lühne, and J. Döllner. Hardware-accelerated attribute mapping for interactive visualization of complex 3d trajectories. In *2014 International Conference on Information Visualization Theory and Applications (IVAPP)*, pp. 356–363, 2014.

[5] W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

[6] F. Crameri. Scientific colour maps (7.0.1). Zenodo, Sept. 2021. doi: 10.5281/zenodo.5501399

[7] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann. Interactive indirect illumination using voxel cone tracing: A preview. In *Proc. Symp. Interactive 3D Graph. and Games*, p. 207. Association for Computing Machinery, New York, NY, USA, 2011. doi: 10.1145/1944745.1944787

[8] J. Díaz-García and P. P. Vázquez. Fast illustrative visualization of fiber tracts. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7431 LNCS, pp. 698–707. Springer, Berlin, Heidelberg, 2012. doi: 10.1007/978-3-642-33179-4_66

[9] 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*, 15(6):1299–1306, Nov./Dec. 2009. doi: 10.1109/TVCG.2009.138

[10] 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*, abs/1503.05787, Mar. 2015.

[11] S. Grottel, J. Heinrich, D. Weiskopf, and S. Gumhold. Visual analysis of trajectories in multi-dimensional state spaces. *Computer Graphics Forum*, 33(6):310–321, 2014. doi: 10.1111/cgf.12352

[12] D. Groß and S. Gumhold. Advanced rendering of line data with ambient occlusion and transparency. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):614–624, 2021. doi: 10.1109/TVCG.2020.3028954

[13] S. Gumhold. The computer graphics and visualization framework. `https://github.com/sgumhold/cgv`. Accessed: 25-March-2022.

[14] L. K. Ha, J. H. Krüger, and C. T. Silva. Fast four-way parallel radix sorting on GPUs. *Comput. Graph. Forum*, 28(8):2368–2378, Dec. 2009. doi: 10.1111/j.1467-8659.2009.01542.x

[15] J. H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM*, 7(12):701–702, dec 1964. doi: 10.1145/355588.365104

[16] M. Han, I. Wald, W. Usher, Q. Wu, F. Wang, V. Pascucci, C. D. Hansen, and C. R. Johnson. Ray tracing generalized tube primitives: Method and applications. *Computer Graphics Forum*, 38(3):467–478, jul 2019. doi: 10.1111/cgf.13703

[17] J. He, H. Chen, Y. Chen, X. Tang, and Y. Zou. Diverse visualization techniques and methods of moving-object-trajectory data: A review. *ISPRS International Journal of Geo-Information*, 8(2):63, Jan. 2019. doi: 10.3390/ijgi8020063

[18] X. He, Y. Tao, Q. Wang, and H. Lin. Multivariate spatial data visualization: A survey. *Journal of Visualization*, 22:897–912, Oct. 2019. doi: 10.1007/s12650-019-00584-3

[19] E. Kandogan. Star coordinates: A multi-dimensional visualization technique with uniform treatment of dimensions. In *Proceedings of the IEEE information visualization symposium*, vol. 650, p. 22. Citeseer, 2000.

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

[21] B. Karis. High quality temporal anti-aliasing. *Advances in Real-Time Rendering for Games, SIGGRAPH Courses*, 2014.

[22] M. Kern, C. Neuhauser, T. Maack, M. Han, W. Usher, and R. Westermann. A comparison of rendering techniques for 3d line sets with transparency. *IEEE Transactions on Visualization and Computer Graphics*, 27(8):3361–3376, aug 2021. doi: 10.1109/TVCG.2020.2975795

[23] A. Kuhn, N. Lindow, T. Günther, A. Wiebel, H. Theisel, and H.-C. Hege. Trajectory density projection for vector field visualization. In M. Hlawitschka and T. Weinkauf, eds., *Proc. EuroVis - Short Papers*. The Eurographics Association, 2013. doi: 10.2312/PE.EuroVisShort.EuroVisShort2013.031-035

[24] C. Liu, K. Qin, and C. Kang. Exploring time-dependent traffic congestion patterns from taxi trajectory data. In *2015 2nd IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services (ICSDM)*, pp. 39–44, 2015. doi: 10.1109/ICSDM.2015.7298022

[25] H. Liu, X. Chen, Y. Wang, B. Zhang, Y. Chen, Y. Zhao, and F. Zhou. Visualization and visual analysis of vessel trajectory data: A survey. *Visual Informatics*, 5(4):1–10, 2021. doi: 10.1016/j.visinf.2021.10.002

[26] S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, and V. Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE Transactions on Visualization and Computer Graphics*, 23(3):1249–1268, 2017. doi: 10.1109/TVCG.2016.2640960

[27] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, Apr. 1986. doi: 10.1145/22949.22950

[28] D. Merhof, M. Sonntag, F. Enders, C. Nimsky, P. Hastreiter, and G. Greiner. Hybrid visualization for white matter tracts using triangle strips and point sprites. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1181–1188, 2006. doi: 10.1109/TVCG.2006.151

[29] NVIDIA. Optix™ ray tracing engine. `https://developer.nvidia.com/rtx/ray-tracing/optix`, 2009. Accessed: 2022-06-30.

[30] V. Petrovic, J. Fallon, and F. Kuester. Visualizing whole-brain dti tractography with gpu-based tuboids and lod management. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1488–1495, 2007. doi: 10.1109/TVCG.2007.70532

[31] A. Reshetov and D. Luebke. Phantom ray-hair intersector. *Proc. ACM Comput. Graph. Interact. Tech.*, 1(2), aug 2018. doi: 10.1145/3233307

[32] F. Ritter, C. Hansen, V. Dicken, O. Konrad, B. Preim, and H.-o. Peitgen. Real-time illustration of vascular structures. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):877–884, 2006. doi: 10.1109/TVCG.2006.172

[33] B. Russig, M. Salm, and S. Gumhold. GPU-based raycasting of hermite spline tubes. In *2020 IEEE Visualization Conference (VIS)*, pp. 26–30, 2020. doi: 10.1109/VIS47514.2020.00012

[34] M. Schirski, T. Kuhlen, M. Hopp, P. Adomeit, S. Pischinger, and C. Bischof. Efficient visualization of large amounts of particle trajectories in virtual environments using virtual tubelets. In *Proc. ACM SIGGRAPH Int. Conf. VRCAI*, pp. 141–147. Association for Computing Machinery, New York, NY, USA, 2004. doi: 10.1145/1044588.1044615

[35] J. Staib, S. Grottel, and S. Gumhold. Temporal focus+context for clusters in particle data. VMV '17, p. 85–93. Eurographics Association, Goslar, DEU, 2017. doi: 10.2312/vmv.20171263

[36] C. Stoll, S. Gumhold, and H.-P. Seidel. Visualization with stylized line primitives. In *Proc. IEEE Visualization*, pp. 695–702, 2005. doi: 10.1109/VISUAL.2005.1532859

[37] C. Tominski, H. Schumann, G. Andrienko, and N. Andrienko. Stacking-based visualization of trajectory attribute data. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2565–2574, 2012. doi: 10.1109/TVCG.2012.265

[38] S.-K. Ueng, C. Sikorski, and K.-L. Ma. Efficient streamline, streamribbon, and streamtube constructions on unstructured grids. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):100–110, 1996. doi: 10.1109/2945.506222

[39] K. Vrotsou, H. Janetzko, C. Navarra, G. Fuchs, D. Spretke, F. Mansmann, N. Andrienko, and G. Andrienko. Simplifly: A methodology for simplification and thematic enhancement of trajectories. *IEEE Transactions on Visualization and Computer Graphics*, 21(1):107–121, 2015. doi: 10.1109/TVCG.2014.2337333

[40] I. Wald, N. Morrical, S. Zellmann, L. Ma, W. Usher, T. Huang, and V. Pascucci. Using hardware ray transforms to accelerate ray/primitive intersections for long, thin primitive types. *Proc. ACM Comput. Graph. Interact. Tech.*, 3(2), aug 2020. doi: 10.1145/3406179

[41] M. Walter and A. Fournier. Approximate arc length parametrization. In *Proceedings of the 9th Brazilian Symposium on Computer Graphics and Image Processing*, pp. 143–150. Caxambu, Minas Gerais, Brazil, 29 Oct.–

1 Nov. 1996.

[42] M. O. Ward. Multivariate data glyphs: Principles and practice. In *Handbook of Data Visualization*, chap. II.7, pp. 179–198. Springer Berlin Heidelberg, Dec. 2008. doi: 10.1007/978-3-540-33037-0_8

[43] C. Ware. *Information Visualization: Perception for Design*. Elsevier Inc., 3 ed., 2012. doi: 10.1016/C2009-0-62432-6

[44] C. Ware, R. Arsenault, M. Plumlee, and D. Wiley. Visualizing the underwater behavior of humpback whales. *IEEE Computer Graphics and Applications*, 26(4):14–18, 2006. doi: 10.1109/MCG.2006.93

[45] W. Wing and Y. Chan. A survey on multivariate data visualization. pp. 1–29, Jan. 2006.

[46] S. Zhang, C. Demiralp, and D. Laidlaw. Visualizing diffusion tensor mr images using streamtubes and streamsurfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):454–462, 2003. doi: 10.1109/TVCG.2003.1260740

[47] L. Zhou and D. Weiskopf. Multivariate visualization of particle data. *The European Physical Journal Special Topics*, 227:1741–1755, Mar. 2019. doi: 10.1140/epjst/e2019-800158-6