

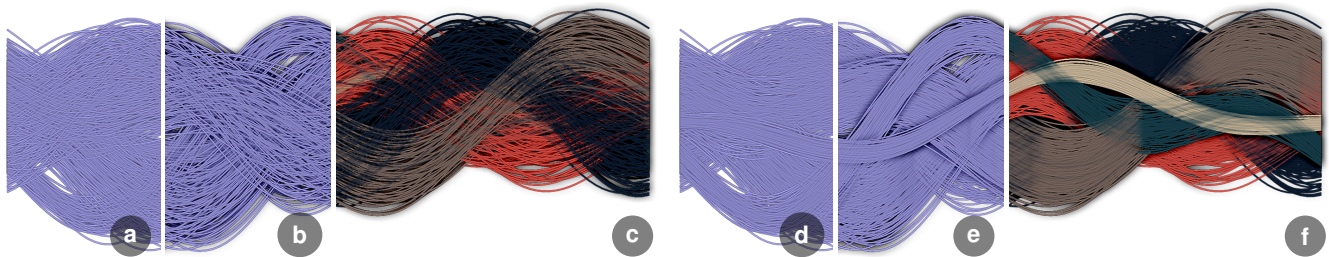


# Line Weaver: Importance-Driven Order Enhanced Rendering of Dense Line Charts

T. Trautner  and S. Bruckner 

University of Bergen, Norway



**Figure 1:** Line weaver was inspired by techniques from textile production where multiple threads are interwoven to form fabrics. If the blending order is (a) ignored or naively used, essential visual information is lost, even if (b) outlines and halos are added or (c) clusters are colored. If, however, the ordering of clusters is (d) optimized, both (e) outlines and halos as well as (f) colors can help perceiving clusters.

## Abstract

Line charts are an effective and widely used technique for visualizing series of ordered two-dimensional data points. The relationship between consecutive points is indicated by connecting line segments, revealing potential trends or clusters in the underlying data. However, when dealing with an increasing number of lines, the render order substantially influences the resulting visualization. Rendering transparent lines can help but unfortunately the blending order is currently either ignored or naively used, for example, assuming it is implicitly given by the order in which the data was saved in a file. Due to the non-commutativity of classic alpha blending, this results in contradicting visualizations of the same underlying data set, so-called "hallucinators". In this paper, we therefore present line weaver, a novel visualization technique for dense line charts. Using an importance function, we developed an approach that correctly considers the blending order independently of the render order and without any prior sorting of the data. We allow for importance functions which are either explicitly given or implicitly derived from the geometric properties of the data if no external data is available. The importance can then be applied globally to entire lines, or locally per pixel which simultaneously supports various types of user interaction. Finally, we discuss the potential of our contribution based on different synthetic and real-world data sets where classic or naive approaches would fail.

## CCS Concepts

• **Human-centered computing** → **Information visualization; Visualization techniques; Visualization theory, concepts and paradigms;**

## 1. Introduction

Line charts, also known as *line graphs*, *line plots*, or *curve charts*, are among the most frequently used forms of visual representation in statistics [Spe52]. In contrast to scatter plots, where data elements are shown as a set of two-dimensional points in space, line charts visually encode the relations between data elements. Therefore, individual points have to be ordered, for example chrono-

logically, along an axis. Unfortunately, line charts have two major weaknesses: First, the individual lines become harder to distinguish and interpret with an increasing number of lines. Second, when using standard alpha blending, occlusion between lines means that the resulting image is dependent on the rendering order. A popular remedy to distinguish individual lines is the use of color. Unfortunately, this prevents the color channel from being used to encode any other data attributes. Considering the second problem regarding

rendering order, it seems that this has not received much attention in the literature so far. In Tableau, for example, one of the most widespread visualization tools, "the drawing order is driven by the order of the members in the data source [Tab20]".

However, rendering order and the resulting occlusion relationships among graphical elements can have a significant impact on their perception, as indicated by the *Gestalt principles*. The principle of *Symmetry and Order*, for instance, sometimes also referred to as *Law of Prägnanz* or the *Law of Simplicity* describes that the human brain tends to interpret visual elements in the simplest possible way to avoid an overflow of visual stimuli. For example, when looking at a wireframe representation of a cube, where only the edges connecting the corner points are visible, our brain will tend to recognize the three-dimensional cube instead of individual primitives such as triangles, rectangles, or trapezoids that arise from intersections of lines.

In this paper, we present a novel technique for the visualization of line charts that avoids the issues caused by a global rendering order by introducing the notion of a quantitative importance function that can vary locally. Using a per-pixel blending approach controlled by this importance function, we provide explicit control over occlusion relationships, allowing us to make more efficient use of available screen space and present features such as clusters in a more coherent manner. The main contributions of our work can be summarized as follows:

- We introduce a new approach for displaying line-based data that supports the use of a quantitative importance function.
- We demonstrate how this method can be used with different types of importance functions.
- We present a simple algorithm for deriving an importance function for grouped line data.
- We show that our technique can be efficiently implemented on modern GPU architectures for high-quality rendering of line data.

## 2. Related Work

Visualization of line sets plays an important role in various domains and different scientific fields, for example, in their elementary form as classic line charts, when exploring networks or graphs to better understand their structure, when visualizing streamlines or pathlines to analyze fluid flow, when interpreting temporal changes of time series, when displaying multi-dimensional data as parallel coordinates, or directly when researching how lines can be rendered as efficiently as possible. All research fields study different challenges but one they all have in common is that visual clutter increases when more lines are displayed. For reasons of clarity, this section is therefore divided into two categories with different clutter-reduction approaches. Section 2.1 focuses on features derived from line data sets and how they can be visually encoded, and Section 2.2 presents advanced and optimized rendering techniques for dense line and curve data sets.

### 2.1. Feature Encoding

One possibility for reducing visual clutter could be to visualize a density estimate of underlying lines, instead of visualizing the

entirety of individual lines. Lampe and Hauser [DLH11b] introduce an approach based on kernel density estimation (KDE), using line kernels defined by a start and an end point. In a subsequent step, the estimated density can then be color-coded using a perceptually uniform heatmap. A similar approach can be used to estimate the density of curves in a continuous [DLH11a] or discrete manner [MF18]. Unfortunately, density representations in general are not well suited when analyzing individual lines, especially in sparse regions. Recent work by Trautner et al. [TBSB20] presents sunspot plots, an approach focusing on this challenge when visualizing scatter plots. In case of line charts, however, the problem remains unsolved.

Another visualization technique that suffers from visual clutter are parallel coordinate plots (PCPs). Assuming the multi-dimensional data originates from a continuous domain, continuous parallel coordinates [WH09] represent a related approach to KDEs and, therefore, benefit from the same advantages while also suffering from the same disadvantages. Instead of changing the visual representation of parallel coordinates in advance, an initial step can be reordering the axes. Blumenschein et al. [BZP\*20] recommend that, especially with highly cluttered data sets, axes with dissimilar data dimensions should be arranged next to each other, whereas for data sets with low clutter, similar axes should be displayed next to each other. Fua et al. [FWR99] introduce a hierarchical cluster-based enhancement for PCPs. They propose to visually encode individual clusters as variable-width opacity bands in combination with proximity-based coloring. The width of such a band represents the extent of the cluster. The center of each band is fully opaque while transparency linearly decreases towards the top and bottom edges. Unfortunately, they do not provide further specification on the blending operator or blending order used. Novotny and Hauser [NH06] introduce a technique specifically targeted at outlier and trend detection within PCPs. The authors propose to detect outliers first, providing them with a separate visual representation, and then applying aggregation techniques to the underlying data to prevent outliers from being smoothed away. Related work by Artero et al. [AdL04] uses image processing techniques to detect clusters. Work by Johansson et al. [JLJC06] introduces transfer functions to highlight different properties of clusters in PCPs. In the end, however, lines are rendered in a given order mostly using the non-commutative Porter-Duff [PD84] *over* operator for blending, neglecting that it is not order independent.

Under the assumption that lines can be viewed as three-dimensional trajectories or networks, Kwon et al. [KMLM16] introduce edge bundling using a spherical graph layout and depth routing for edges to improve legibility of graph visualizations. Subsequently, line bundles are emphasized by global illumination using real-time ambient occlusion approaches similar to the work by Eichelbaum et al. [EHS13]. Here, it is important to mention that we do not consider edge bundling as a competing approach but as a possible pre-processing step. The resulting line bundles could subsequently be displayed using our technique.

Additional inspiration comes from the work of Nakayana and Yano [NY10] who combine classic space-time cubes with KDEs by using a spatio-temporal kernel for 3D point data to emphasize both the temporal duration as well as the spatial extent, visualized using

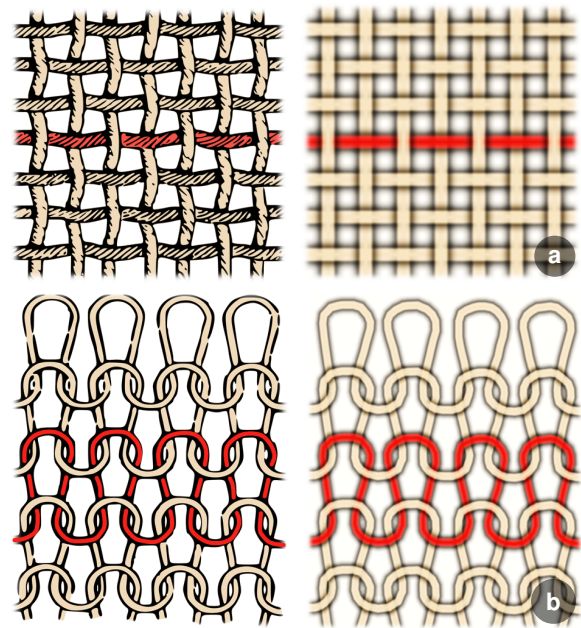
volume rendering. Subsequently, Demšar and Virrantaus [DV10] built upon this idea and applied 3D density estimations to three-dimensional polylines. The interpretation of line sets as volumes [SM04] allows for the use of transfer functions, which inspired us to use importance functions without having to convert the underlying data to a volume. Moritz and Danyel [MF18], for example, suggest normalizing the contribution of a curve to the density estimation by its arc length, correcting for the higher numbers of pixels that are needed to render strongly fluctuating curves. Instead, we use the arc length as a derived geometric property, describing curvature and frequency. This enables us to optimize overall visibility and reduce visual clutter by displaying curves with more variability in the back and less fluctuating curves in front. However, many other properties such as visual complexity, as described by Ryan et al. [RMCW18], geometric features derived from families of curves by Konyha et al. [KLM\*12], or statistical features could be used instead.

## 2.2. Line Rendering

Before the development of more advanced line rendering techniques, Spear [Spe52] recommended creating multiple graphs using the same unit scale and comparing them, for example, using juxtaposition in case the data set was too cluttered. This is still a common practice when current techniques reach their limits. Building on that, Cleveland et al. [CMM88] recommended that an average angle of 45 degrees should be kept between adjacent line segments. In the literature, this is often referred to as "banking to 45 degrees". Furthermore, line smoothing techniques can be applied, for example as proposed by Rosen and Quadri [RQ21], to initially remove high frequencies or noise from the data, which we consider an optional pre-processing step to our approach.

Rendering smooth lines in OpenGL is a basic functionality that has yet to be supported by all graphics cards. Kilgard's [Kil20] polar stroking technique is a recent approach which uses the arc length of a curve for texturing or dashing. A general overview of different CPU and GPU rendering techniques for transparent 3D line sets is provided by Kern et al. [KNM\*20]. Similar to our method, an A-buffer [Car84] combined with multi-layer alpha blending can be used to approximate transmittance and color of fragments. While rendering, A-buffers store additional fragment information, for example, by using per-pixel linked lists together with a global atomic counter, as suggested by Yang et al. [YHGT10]. A modification of A-buffers are K-buffers, introduced by Bavoil et al. [BCL\*07]. Instead of blending all fragments, only a fixed number of fragments are stored and blended. An enhancement of such a K-buffer is used by Groß and Gumhold [GG21] who introduce a GPU-based ray casting approach supporting ambient occlusion and transparency by using a billboard proxy geometry.

In their work, Hagh-Shenas et al. [HSKIH07] compare the accuracy of two distinct strategies for visualizing multivariate data using different colors, namely weaving and blending. Weaving, as presented in the work by Luboschik et al. [LRS10], refers to the selection of one colored item, such as a part of a line, which is then exclusively rendered on top at a given location. Blending, on the other hand, refers to the mixing of multiple colors, for example assigned to multiple lines which all overlap. Our approach can be



**Figure 2:** Juxtaposition of illustrations of (a) plain weaving and (b) weft knitting in comparison to visualizations retrieved with our approach (right column). Note how both examples cannot be reproduced using classic alpha blending and a globally defined render order, i.e., depth per thread.

seen as a hybrid between both, as we use blending to combine contributions of lines with similar importances, but since importances may vary along lines, their overall appearance will be reminiscent of a "weaving" pattern. Our method is related to approaches for smooth composition by Luft and Deussen [LD06] and Bruckner et al. [BRV\*10]. In addition, we want to highlight the work of Everts et al. [EBRI09] on depth-dependent halos to emphasize line bundles. Their work inspired us to use halos, implemented using the unsharp masking approach proposed by Luft et al. [LCD06].

Other related strategies consider the rendering of lines as global optimization problem. Günther et al. [GRT13] suggest rendering only a globally optimized selection of lines that are indispensable when visualizing important features, thus preventing visual clutter that would arise from rendering all lines. This approach has later on been refined in order to ensure coherence in 3D time-dependent flow visualizations [GRT14] and to support not only lines but transparency optimization in combination with points and surfaces [GTG17].

## 3. Line Weaver

Our approach addresses the fact that in many cases, not much attention is paid to the order in which lines are drawn in a chart. When rendering only solid single-colored and fully opaque lines, this does not make a significant difference, but as soon as transparency is introduced or more advanced stylization approaches such as halos are applied, it becomes important – depending on the blending operator – to also consider the effects of the rendering order.



The *algebraic model for visualization design* proposed by Kindlmann and Scheidegger [KS14] explicitly refers to such considerations. They illustrate this by using a plot of taxi pick-up and drop-off locations, which they consider as a set of points without an inherent order, and argue that using an order-dependent blending operator (such as the common *over* operator) in this case constitutes a failure of representation invariance, which they refer to as "hallucinator", i.e., a deficiency where differences in the images may arise from representational or algorithmic choices (such as the rendering order) without reflecting changes in the underlying data. Order-independent operators will, by definition, always generate the same result irrespective of the rendering order, and in their example Kindlmann and Scheidegger propose to use additive blending as a possible resolution. In additive blending, the contributions of all elements to a pixel are averaged and hence their order is irrelevant. The drawback of this approach is that it completely eliminates occlusion which in fact can be a powerful cue. As already mentioned, we have learned from the Gestalt principles such as *continuity*, *closure*, and *figure-ground* that human perception attempts to complete missing or occluded image regions.

In our approach, we therefore propose to preserve these cues by still using a blending operator that exhibits occlusion, but instead providing explicit control over occlusion relationships by introducing an importance function. While this may at first glance look like a minor semantic distinction (replacing the term order by the term importance), it opens up several interesting and, as we will demonstrate in the remainder of this paper, advantageous avenues for improving the visualization of line-based data.

We regard line data as set  $D = \{L_1, L_2, \dots, L_N\}$  of  $N$  polylines with its members  $L_i = (P_1, P_2, \dots, P_M)$  represented as tuples of  $M$  ordered two-dimensional points  $P_i = (x_i, y_i)$ . The resulting parametric curve  $l_i(u)$  of each member is normally a polyline generated by linear interpolation between its associated points  $P_1, P_2, \dots, P_M$ , but of course other interpolation functions are equally possible. We choose this formulation since it is general enough to represent common visualizations such as time-series charts and parallel coordinate plots, which simply constitute different mappings between the underlying data set and the  $x$  and  $y$  coordinates of the individual points. Our importance function  $\beta_i(u) \in [0, 1]$  now associates a scalar importance value with every position along each curve  $l_i(u)$  and has two major properties that distinguish it from order. First, our importance function is quantitative in nature, not just ordinal. This means that it is possible that two lines may have very similar or even the same importance value. Second, importance does not need to be constant along a line, but it may vary. These attributes should be represented in the resulting visualization. Specifically, our approach is based on the following requirements:

- The contributions of individual lines should be independent of their order in the data set or the order in which they are rendered.
- Line segments with similar importance contributions should contribute similarly to the pixels they cover in the final image.
- When the importance differs significantly, line segments with higher importance should occlude line segments with lower importance.

We can draw an analogy to the textile industry, for example, when comparing weaving or knitting techniques as illustrated in

Figure 2. Instead of a global ordering of individual threads, they are locally woven to interleave forming an intricate pattern that is discernible to human observers. In the same way, our aim is to thread lines in a meaningful pattern, instead of simply pasting them on top of each other. Our approach, therefore, enables the importance of individual lines to vary along their trajectory, but without imposing an ordinal relationship among them.

### 3.1. Importance-Based Blending

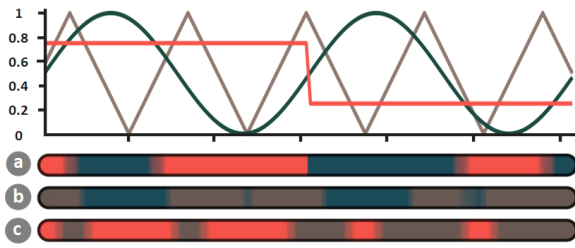
In computer graphics, the term order-independent transparency commonly refers to techniques that aim to enable the rendering of transparent polygons using the Porter-Duff [PD84] *over* operator in a way that it does not require prior sorting of the primitives. This can occur in an exact manner producing the same results as primitive sorting, e.g., by still performing sorting but rather at the fragment level (using data structures such as an A-buffer [YHGT10]) or in an approximate manner, e.g., by using different (typically depth-based [BM08]) blending operators that attempt to mimic the results of the *over* operator as closely as possible. Our goal here, however, is different. We are seeking a blending operator that allows us to visually reflect the properties of our importance function, i.e., by conveying its quantitative nature as well as its variability over polyline primitives. The latter can be resolved by changing the granularity at which the blending is performed. Switching from a per-primitive to a per-pixel resolution, as it is done in A-buffer based approaches, allows us to freely vary the importance function across a primitive. The former, however, is more challenging as the decision of whether one element is in front of another in the classic *over* operator is inherently binary and therefore prone to artifacts.

A good illustration for this is z-fighting, caused by two primitives with differences in depth that are close to the numerical precision of the z-buffer, resulting in visible artifact patterns. As stated previously, we instead want our blending to have a well-defined and meaningful behavior when importance values are equal or similar. When blending together multiple elements with equal importance values, the result should correspond to the average value of all contributing elements, while when importance values are far apart, we want to indicate higher importance values using occlusions. As importance values can vary continuously, we further want to avoid abrupt changes between these behaviors. Such smooth transitions can be achieved using a blending approach inspired by the works of Luft and Deussen [LD06], and Bruckner et al. [BRV\*10].

Conceptually, this works by having the color contribution of one element (e.g., a pixel color) influencing the contributions of other elements whenever they are within a certain importance range, using a continuous falloff function such that the contributions are equal when the respective importance values are the same. The individual contributions (which then include those of other elements within the influence range) are next blended in sequence of their importance using the conventional *over* operator. The adjusted color of an element  $c'_i$  is computed as follows:

$$c'_i = \frac{\sum_{\Delta(\beta_i, \beta_j) > 0} c_j \Delta(\beta_i, \beta_j)}{\sum_{\Delta(\beta_i, \beta_j) > 0} \Delta(\beta_i, \beta_j)}, \quad (1)$$





**Figure 3:** Pairwise comparison of two lines, each with varying importance, blended on top of each other: (a) red and turquoise, (b) brown and turquoise, and (c) red and brown, wherein red has a step function, turquoise a sine function, and brown a tent function as importance, using a smoothness of  $t = 0.15$ . Note that the visual result depends purely on the blending and not the render order.

wherein  $c_j$  are the opacity-weighted color contributions of the other elements,  $\beta_i, \beta_j$  are the respective importance values, and  $\Delta$  is a falloff function that gradually decreases to zero with the absolute difference of its arguments. In practice, we define  $\Delta$  based on a Hermite polynomial, similar to the common *smoothstep* function, such that it is 1 if the two importance values are the same and becomes zero as their difference exceeds a threshold value:

$$\Delta(a, b) = \begin{cases} 0 & \text{if } |a - b| \geq t \\ 2\left(\frac{|a-b|}{t}\right)^3 - 3\left(\frac{|a-b|}{t}\right)^2 + 1 & \text{otherwise} \end{cases}, \quad (2)$$

wherein  $t$  is the user-defined threshold that specifies the range of contributions to be considered.

This approach now allows us to flexibly vary the importance function along individual lines. If the respective importances vary continuously, two polylines may pass through each other without abrupt changes. In regions where the importances are equal, each contributing element will contribute equally. The behavior of our blending approach with different importance functions is illustrated in Figure 3. We show how the blending result of two completely overlapping lines varies based on their importance functions (a step function, a sine function, and a tent function).

### 3.2. Rendering and Stylization

For high-quality rendering of our polyline sets with variable thicknesses, we use a method that combines a geometric approach with signed distance functions (SDFs). This means that instead of generating geometry to define the exact outline, which can be difficult and expensive both due to the potentially high geometric complexity (e.g., when rounded joins between line segments are desired) as well as issues with numerical precision, we instead rasterize the line geometry conservatively and then evaluate an SDF for each fragment in order to determine the exact coverage.

In this method, we create a polygonal scaffolding for each individual polyline by extruding it on the fly into a triangle strip such that all potential pixels of the line are covered, similar to the proxy billboards used by Groß and Gumhold [GG21]. To avoid gaps in

the pixel coverage, it is important to ensure that neighboring segments are extended such that the endpoints of their outline meet at the bisector between the two segments. Then, for each fragment of the resulting geometry, we evaluate an SDF determining inside and outside of a line segment. The distance from a point  $P$  to a line segment connecting points  $A$  and  $B$  can be written as:

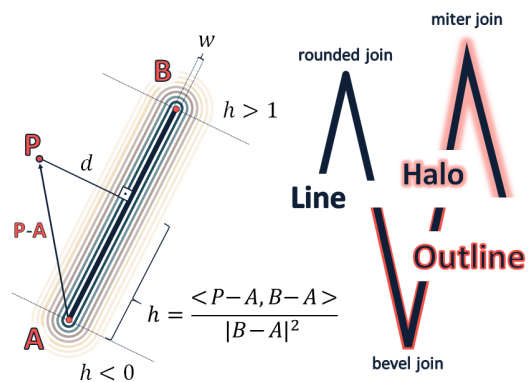
$$d(P, A, B) = \left| \vec{AP} - h \cdot \vec{AB} \right| - w, \quad \text{with} \quad (3)$$

$$h = \text{clamp} \left( \frac{\vec{AP} \cdot \vec{AB}}{\vec{AB} \cdot \vec{AB}}, 0, 1 \right), \quad (4)$$

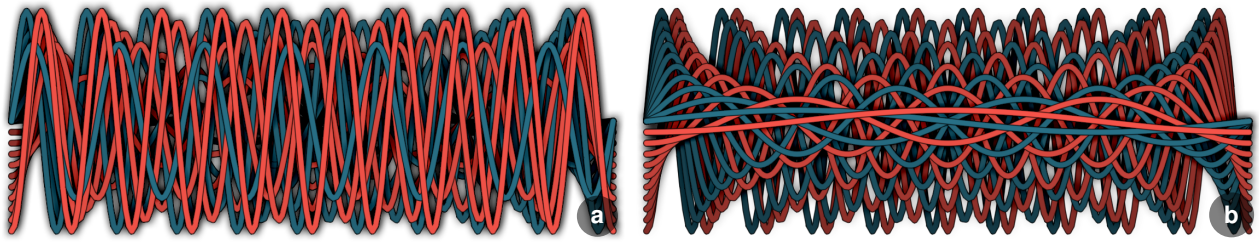
wherein  $w$  is the thickness of the line segment. Note that the boundary of the line is located at the zero level set of  $d$  and negative values correspond to positions inside the line segment. Figure 4 shows an illustration of the resulting SDF. To ensure proper handling of joins, we need to take into account the distance to the previous, current, and next line segment. This can be achieved by using different operators combining the respective distances. Per default, we use the *minimum* operator which results in rounded joins, but other types such as *miter* and *bevel* are equally possible. Thus, our distance function  $d_i$  for the  $i$ -th segment of a polyline is defined as follows:

$$d_i(P) = \min(d(P, P_{i-1}, P_i), d(P, P_i, P_{i+1}), d(P, P_{i+1}, P_{i+2})). \quad (5)$$

At the same time, this approach enables easy stylization of our lines, as the distance can be flexibly mapped to different color and/or opacity profiles. Furthermore, per-fragment derivative functions available in common APIs such as OpenGL or DirectX can be used to perform anti-aliasing. For all the images in this paper, we use solid lines with a darkened outline, generated in the described manner. In addition, we use the unsharp masking approach by Luft et al. [LCD06] to generate halos.



**Figure 4:** Calculating the distance from all points  $P$  in space to a line defined by the two points  $A$  and  $B$  results in an SDF. Note how its contour lines converge to a circle as distance  $d$  increases. The SDF can then be utilized to render anti-aliased lines with variable thicknesses, as well as additional outlines, halos, or join types.



**Figure 5:** Juxtaposition of a synthetic data set containing sine as well as cosine functions with varying amplitude and frequency. The line charts are rendered using (a) the order in which the individual lines are specified within the data set, and (b) a global ordering dependent on the overall length of the lines implicitly encoding curvature and complexity. It can be seen that (a) high-frequency curves can easily obscure low-frequency curves with a low amplitude, whereas (b) an optimized order allows for all curves to remain visible.

Once the color and opacity of a line fragment has been determined, we store it in a per-pixel linked list. To determine the final pixel color, the list is subsequently sorted based on importance and blended using the approach described in the previous Section 3.1.

### 3.3. Importance Functions

In principle, there are many different ways of how to define an importance. First, and most naturally, the importance value itself may be part of the underlying data. For instance, in case of time-series data there may be a confidence measure associated with each data point. If the resulting visualization should then prioritize high confidence values, this would be an appropriate choice. Another example could be the numerical result of a feature detection algorithm. In many cases, however, such domain-specific, explicit importance measures may unfortunately not be available.

In this case, we may instead want to use more fundamental properties of our lines themselves. Our goal is to minimize the amount of overdraw in a heuristic manner, by assigning lower importance values to those lines that take up screen space. A simple way to achieve this is by specifying the importance based on the arc length of each line. The impact of this simple yet powerful geometric property is shown in Figure 5. Extending this idea and exploiting the fact that importance values may vary locally, we propose an approach to generate importance values for the common scenario where lines grouped into different sets are depicted in a single chart. Examples include the depiction of time series grouped by a categorical variable, but our method equally applies to cases where the grouping has been established by other means, such as the application of a clustering method. In this case, each line has an associated group identifier and our goal is to reduce the amount of overdraw by assigning importances based on an estimate of the amount of screen space taken up by all the lines of a group along the dependent axis of the graph. For instance, if our chart depicts time series, the values for a particular group may vary considerably for one time step but may fall within a much narrower range as time progresses. Another group may exhibit an inverse behavior. In such a case, the importance values for each group should correspond to the estimated amount of screen space taken up by its lines at each time step, such that groups that take up less space receive higher importance values. Our blending method is well-suited for

this purpose as it allows us to interpolate between the assigned importance values along the ordinate of our graph in order to avoid discontinuities.

We propose a simple algorithm to compute the importance values for such a weaving pattern using a greedy approach (see Algorithm 1). For each value along the independent axis of the graph, we first compute the minimum and maximum values of each group, i.e., the interval on the dependent axis covered by lines belonging to this group. This allows us to determine the envelope of all lines associated with a group by connecting two subsequent intervals along the independent axis forming a trapezoid (for the last value on the axis, we simply duplicate the interval), and compute their areas. We then iterate over the calculated areas along the independent axis with the goal of establishing an ordering of all groups according to a cost function. The cost value for a group corresponds to the sum of the intersection areas with all other groups multiplied by the area covered by the group itself. Initially, all groups are marked as *active*. Using the computed envelope areas, we next determine the area of intersection between all other active groups. Then, we select the group with the lowest cost value among the active groups and mark it as *inactive*. We then repeat recomputing the cost values and select the group with the lowest cost until no active group remains. The importance values are then assigned based on this sequence of selection, i.e., the first selected group receives the highest importance, etc.

## 4. Implementation

Our approach was implemented in C++ and OpenGL. Loading as well as pre-processing the data, such as either deriving the importance based on arc length if no external importance is provided, or executing Algorithm 1 from Section 3.3, are performed on the CPU while the rendering itself runs in parallel on the GPU. However, it would be equally possible to calculate or modify importance values on the GPU allowing for dynamic data. The selection of the data set to be visualized as well as additional user-dependent parameters, such as line color or width, were implemented using an *ImGui* [Cor20] user interface. Our approach consists of the following phases:

**Line rasterization:** The input of our rendering pipeline are three buffers containing the  $x$  and  $y$  coordinates, and the impor-

**Algorithm 1:** Weaving Loom

---

```

input : A set of groups  $G = \{G_1, G_2, \dots, G_K\}$ 
output: Ordering for each value on the dependent axis
1 for all values  $x$  on the independent axis do
2   compute group coverage areas  $A(G_i) \forall G_i \in G$ 
3   compute group intersection areas  $I(G_i, G_j) \forall G_i, G_j \in G$ 
4   mark all groups as active:  $P \leftarrow G$ 
5   initialize order to zero:  $o \leftarrow 0$ 
6   while  $P \neq \emptyset$  do
7     compute cost values:
8      $C(G_i) \leftarrow A(G_i) \sum_{i \neq j} I(G_i, G_j) \forall G_i, G_j \in P$ 
9     select group with minimal cost:
10     $S \leftarrow \arg \min_{G_i \in P} C(G_i)$ 
11    output and increment order, update active groups:
12     $O_x(S) = o$ 
13     $o \leftarrow o + 1$ 
14     $P \leftarrow P \setminus S$ 

```

---

tance per tuple. It is, however, unimportant in which order the lines are allocated within the buffers as they are sorted in the blending phase. In a geometry shader, based on the desired line thickness, we construct the line scaffolding as triangle strips by using the `GL_LINE_ADJACENCY` primitive type, which provides the shader with access to neighboring vertices without requiring duplication. In the fragment shader, we evaluate the SDF from Equation 5 in order to determine the fragments covered by the line segment and evaluate their colors. The result, together with its importance value, is then added to a per-pixel linked list. We use an image object to store the index of the last list entry for each pixel, as well as a shader storage buffer object (SSBO) to store the fragment data. This buffer contains a single counter for the total number of allocated entries, as well as an array of the actual list entries.

**Fragment blending:** We render a screen-filling quad to traverse the linked list for each pixel, performing blending as described in Section 3.1. We use bubble sort to sort the list entries based on their importance and then use our blending operator to determine the final combined pixel color. If halos based on the method of Luft et al. [LCD06] are enabled, this step is preceded by an additional blurring pass which is then used as secondary input to darken surrounding pixel regions.

Our complete source code is available at:

<https://github.com/TTrautner/LineWeaver.git>

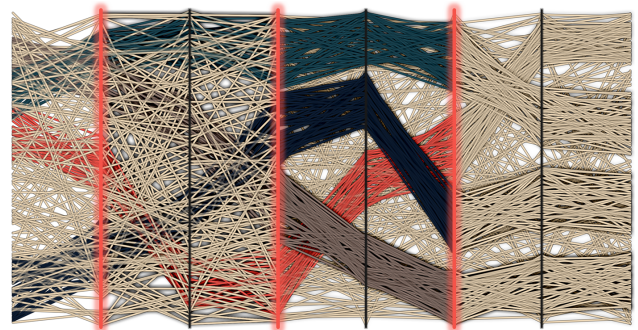
## 5. Usage Examples

In this section, we now demonstrate the strength and versatility of line weaver based on three real-world data sets and one artificially generated data set. Each data set has a different number of lines (between 20 and 1,200) and a varying number of clusters (between 3 and 5). We further illustrate the properties of our approach by using diverse types of charts such as parallel coordinate plots (PCPs), Andrews plots [And72], and time series plots. Finally, we show

that even user interaction can be efficiently implemented by presenting a usage example of a magic lens [SFB94] for focus+context [Hau06] exploration and angular brushing [HLD02].

### 5.1. Global Importance

To the best of our knowledge, there are no studies that analyze the blending order and its impact on line charts. Blumenschein et al. [BZP\*20] evaluated different axis reordering strategies in parallel coordinates and found that intersections of line bundles can help identifying clusters in cluttered data sets. Their benchmark data set therefore serves as a reasonable first test case for our technique. It allows us to compare their suggestions to our insights, e.g., that the z-ordering has a significant impact and that both orderings (axis and depth) are not independent. Figure 6 shows data set *4C.6-150N-Sim* from their study. The authors concluded that this similarity-based axis arrangement tends to be less appropriate when identifying clusters. As shown, it is difficult to distinguish clusters, even when different colors are assigned to each of them. This also does not change when lines are additionally highlighted with halos. When using a global importance, i.e., per line, the exact shapes of the clusters become visible even without using colors. There are four clusters within a relatively noisy data set. The importance of each line is determined by the cluster it belongs to. Apart from noise, the cluster with the most lines is considered as most important. Within each cluster, the importance of the lines is then determined by their arc length, whereby the shortest line is rendered on top. Although the analysis of clusters using this similarity-based axes arrangement has so far been considered unsuitable, it can now be done expressively using line weaver.

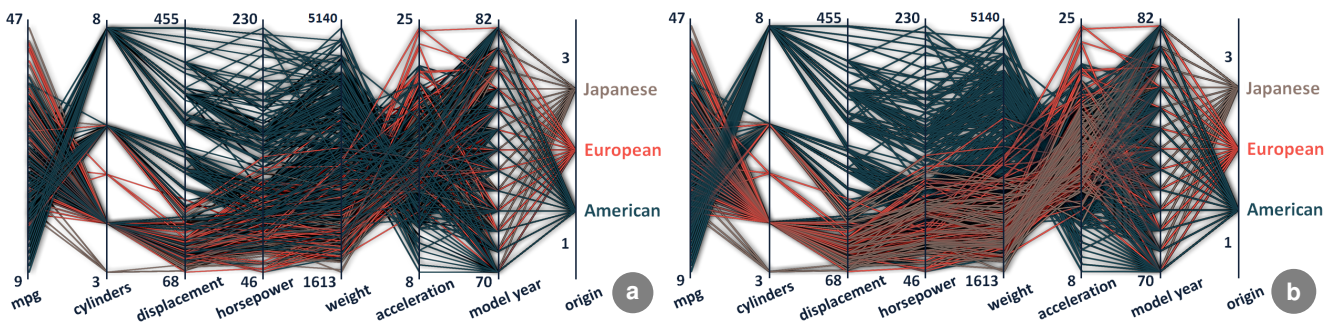


**Figure 6:** Visualization of benchmark data set *4C.6-150N-Sim* [BZP\*20] consisting of four clusters: brown (50 lines), dark blue (49 lines), turquoise (48 lines), red (46 lines), and beige noise (150 lines). From left to right, we compare the result of a randomized rendering order with outlines and additional highlights with halos, to our approach using an importance function depending on cluster size and arc length, including a monochrome version of it.

### 5.2. Local Importance

To demonstrate our technique for locally varying importance functions, we first use a PCP of the MPG [DG21] data set from the UCI Machine Learning Repository containing properties of cars, such as origin, number of cylinders, weight, horsepower, etc. from





**Figure 7:** Juxtaposition of parallel coordinates plots using (a) a randomized order and (b) line weaver of the Auto MPG [DG21] data set consisting of three clusters: brown (79 lines) Japanese cars, red (68 lines) European cars, and turquoise (245 lines) American cars.

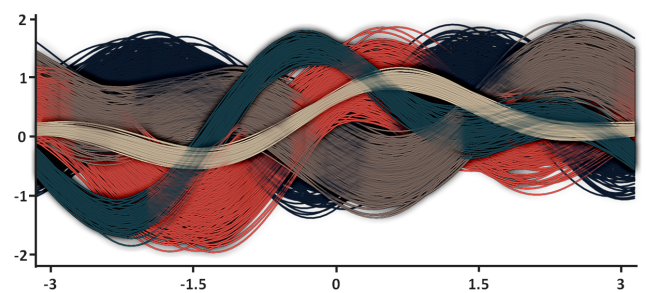
cars produced between 1970-1982. We apply our weaving loom algorithm from Section 3.3 using the origin variable to define three groups. As shown in Figure 7, Japanese and European car models have similarly low displacement, horsepower, and weight, and therefore longer acceleration times, whereas American cars cover a much wider spectrum. In contrast to the rather compact and therefore stricter ordering, looking at model year, for example, all three clusters are equally broad and therefore averaged instead. As can be seen by comparing Figure 7 (a) and (b), the randomized order results in the almost complete occlusion of the more compact group of Japanese cars, while they remain clearly visible using our approach.

Apart from classic PCPs, there are various other visualization techniques for high-dimensional data. Another well-established form of visualization are Andrews plots [And72] which are also referred to as *smooth PCPs* in the literature. Each  $n$ -dimensional data point  $x = \{x_1, x_2, \dots, x_n\}$  is defined as a finite Fourier series, visualized as curve  $f_x(t)$  within the interval  $[-\pi < t < \pi]$ :

$$f_x(t) = \frac{x_1}{\sqrt{2}} + x_2 \sin(t) + x_3 \cos(t) + x_4 \sin(2t) + x_5 \cos(2t) + \dots \quad (6)$$

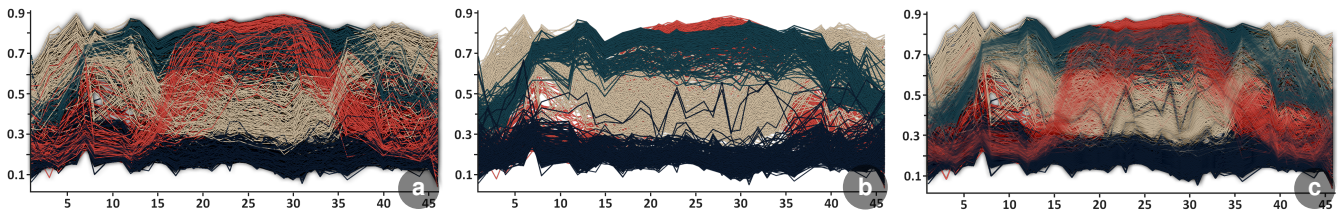
To display different facets of both the plot and our technique, Figure 8 shows the result of such a transformation using a synthetically generated data set, which was also used as our paper teaser in Figure 1. In contrast to the global importance, occlusion relationships between line bundles may change, resulting in a weaving pattern. The determining factor is the area of each bundle and how much it locally occludes other clusters. Due to this optimization, the most compact *beige* cluster becomes visible. Notice how its color and opacity are averaged when it overlaps the *turquoise* cluster with almost identical width. A similar phenomenon can be seen with the *dark blue* and *brown* clusters. In contrast, the less compact *red* cluster is alternatively woven from front to back and vice versa. Each of these changes is smooth which avoids hard cuts and discontinuities, enabling the viewer to better understand the entirety of the visualized data set.

As third example, we present time series data from Tan et al. [TWP17] derived from high-resolution satellite images, each



**Figure 8:** Andrews plot of synthetic data showing two compact beige (100 curves) and turquoise (100 curves) clusters rendered over the wider dark blue (300 curves) and brown (250 curves) clusters which are in turn interwoven with the red (250 curves) cluster.

containing 1 million pixels, with one pixel corresponding to a geographic area of  $64 \text{ m}^2$ . In total, 46 images were taken over time and corrected so that each pixel corresponds to the same geographic region. Next, the temporal change of each area was analyzed considering 24 different category classes including corn, wheat, water, sunflower, etc. We selected four classes for our visualization: soy (class 10), grassland (class 12), poplar (class 21), and mineral surface (class 22), each containing 300 time series which in turn consist of 46 time steps each. The result comparing three different rendering orders is shown in Figure 9: (a) random, (b) as stored in the file, and (c) line weaver using local importance. With random order, chance decides which lines are rendered last and are therefore best visible to the user, or how well individual outliers are preserved. Using an order defined by the data set, classes are perceived as individual bundles since they have been saved one after the other. Unfortunately, the class that was rendered first will be the least visible and the class rendered last will be visible best. Line weaver, on the contrary, works independently of the storage and rendering order, providing the same visual result even when these orderings are changed. Note that this time, the comparably high smoothness of the blending, i.e., averaging of clusters, guarantees that individual bundles are easy to identify (similarly to using a randomized order) while preserving outliers, irrespective of chance.



**Figure 9:** Comparison of four classes of the Crop data set from Tan et al. [TWP17] rendered using (a) random line order, (b) the sequence individual lines were stored, and (c) line weaver using local importance. When using (a) random order, the outliers of the dark blue cluster, otherwise in the middle of the beige cluster, are lost. An ordering that is dependent on (b) the data set results in coherent cluster bundles rendered from back to front starting with class 10 (red), followed by class 12 (beige), class 21 (turquoise), and finally class 22 (blue). This time, the blue outliers remain visible by chance, but the red cluster bundle substantially disappears. Using (c) line weaver, both the red and turquoise clusters are interwoven with the beige cluster and the most compact dark blue cluster is rendered on top, preserving its outliers.

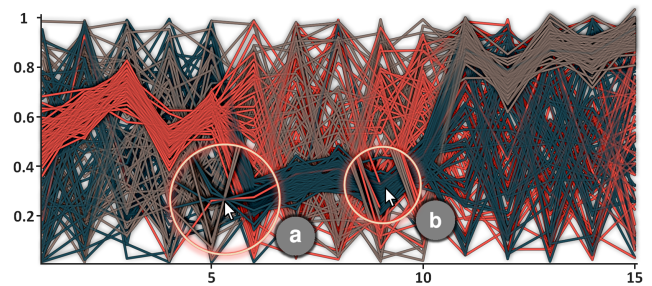
### 5.3. Highlighting and Focus Enhancement

While the previous examples demonstrate how our technique behaves using different types of importance functions derived from the data, our approach also supports dynamically changing importance values. This enables various types of user interactions that demand variable local or global changes with arbitrary granularity. Interactions that would be encoded using an attribute such as color can instead be used to modulate importance values. To illustrate this, we show a simple implementation of two well-established techniques serving as representatives of common line chart interactions: a magic lens [SFB94] that enables focus+context [Hau06] exploration and angular brushing [HLD02]. The lens, positioned at the mouse cursor, has local influence on the importance of all lines within and allows us to see lines that pass through a region even though they are covered by other lines, for instance to highlight lines with a specific axis value. Similarly, angular brushing can help to locally pull a bundle of lines with a certain angle forward – like a rubber band metaphor. As our approach supports quantitative importance functions, this can be done in a non-binary manner based, as is common, on smooth brushing operations.

For this example, we use the smooth subspace data set from Huang et al. [HYX\*16] for k-means clustering of time series data. It consists of 3 clusters, each containing 50 time series, which in turn consist of 15 time steps per series. A distinct property of this data set is that each cluster has a similar and compact pattern for 5 consecutive time steps, whereas all other steps are randomly distributed. For cluster 1 these are time steps 1-5, cluster 2: time steps 6-10, and cluster 3: time steps 11-15. The resulting visualization, where each compact pattern of a cluster is locally woven to the front once, is shown in Figure 10. Note how it additionally provides examples of (a) a focus+context lens and (b) angular brushing.

## 6. Performance

We conducted performance measurements using a desktop computer equipped with an Intel Core i7-8700K CPU (3.7 GHz), 16 GB RAM, a NVIDIA GeForce RTX 2080 graphics card with 8 GB of texture memory, and a Windows 10 Home 64-bit operating system. Overall, we analyzed two artificially generated data sets and four real-world data sets, corresponding to the figures presented in this paper. In addition, we made sure that all data sets have different



**Figure 10:** Two examples of user interaction on a time series data set from Huang et al. [HYX\*16], using a magic lens for (a) focus+context and (b) angular brushing. The lens, centered at the mouse cursor, can be used to, e.g., trace outliers, highlight lines passing through a specific point, or locally change the ordering of bundles by emphasizing lines with certain angle, e.g.  $-80^\circ$ .

densities and varying numbers of lines (between 20 and 1,200). We used two representative screen resolutions ( $1280 \times 720$  and  $1920 \times 1080$ ) and scaled each data sets so that its bounding box filled the viewport.

In addition to measuring the rendering performance, we also attempted to quantify the degree of overplotting using the following measure:

$$Overplotting = 1 - \frac{1}{|D|} \sum_{L_i \in D} \frac{\#visible\ Pixels(L_i)}{\#total\ Pixels(L_i)}, \quad (7)$$

where  $D$  is the data set the lines originate from and  $L_i$  represents a single line of which the ratio of visible to the total number of pixels that make up this line is calculated. These are then summed up and normalized. The calculated *Overplotting* measure produces a value from the interval  $[0, 1]$ , where 0 refers to no overdraw. For example, if two polylines are drawn exactly on top of one another, this corresponds to a value of 0.5, with three lines 0.66, etc.

A detailed overview of all test cases is shown in Table 1. It reveals that our technique provides interactivity for data sets containing thousands of lines. It is therefore easily possible to interact with



the visualization, for example by changing individual parameters at runtime. The analysis furthermore shows that line weaver increases information content by reducing clutter and overplotting and therefore helps to display line data in a more expressive way.

Figure	Lines	FPS		Overplotting Random/ Figure
		1280 × 720	1920 × 1080	
Sin Cos - 5	20	556.34 <sup>562.68</sup> <sub>346.85</sub>	290.53 <sup>293.77</sup> <sub>289.64</sub>	0.79 / 0.35
PCP 1 - 6	493	135.52 <sup>136.12</sup> <sub>134.87</sub>	109.34 <sup>110.72</sup> <sub>108.25</sub>	0.76 / 0.74
PCP 2 - 7	392	36.06 <sup>37.01</sup> <sub>35.63</sub>	25.15 <sup>26.42</sup> <sub>24.77</sub>	0.85 / 0.84
Andrews - 8	1,000	27.82 <sup>29.65</sup> <sub>27.58</sub>	21.91 <sup>22.27</sup> <sub>20.63</sub>	0.91 / 0.88
Crop - 9	1,200	18.43 <sup>19.41</sup> <sub>17.99</sub>	12.63 <sup>13.48</sup> <sub>11.98</sub>	0.93 / 0.84
Lens - 10	150	234.93 <sup>241.47</sup> <sub>221.16</sub>	156 <sup>158.39</sup> <sub>152.60</sub>	0.75 / 0.73

**Table 1:** Summary table of the analyzed test scenarios including name and figure number, the number of lines in the data set, the average frames per second (avg<sub>min</sub><sup>max</sup>) for two common 16:9 screen resolutions, and the result of the overplotting measure.

## 7. Discussion and Limitations

Line charts are a well-established form of representation, not only in the field of visualization but also within a variety of other research fields. Although we have focused on traditional line charts such as time series plots or parallel coordinate plots in this paper, our basic approach is also suitable for other types of two-dimensional line and trajectory data. In particular, we believe that it would be interesting to examine its application to geospatial data, as well as circular visualizations such as radar charts. Our blending and rendering approach can already be applied to such scenarios in a straight forward way, but the line weaving algorithm, as presented in Section 3.3, would have to be adapted to support general parametric curves.

Although edge bundling and line weaver are fundamentally different approaches (edge bundling reduces clutter by increasing overplotting and line weaver reduces clutter by decreasing overplotting), it would be interesting to conduct studies on how both techniques harmonize with each other. For example, by applying edge bundling as pre-processing step to initially reduce the overall number of lines, similar to techniques such as filtering or subsampling, and then smoothly blending the fewer resulting bundles using an importance function that optimizes their visibility.

We have provided examples of data sets containing up to 5 clusters. As soon as the number of clusters approaches the number of lines, for example, if lines cannot meaningfully be clustered into bundles, the ordering degenerates to an optimization of arc length only. In addition, it is recommended that clusters correspond to relatively compact bundles. Furthermore, as is valid for all techniques that rely on color mixing, the combination of several different colors may not always be easy to interpret. However, as our approach represents a hybrid between weaving and pure blending, the visual continuity due to occlusion can help to better resolve such ambiguities. To further compensate for this, alternative blending operators such as hue-preserving color blending [CWM09] could be used instead. In addition, it would be interesting to conduct studies in which various data properties are compared, in order to find an

optimal and versatile importance function derived from data properties from various domains.

Throughout the paper we have used data sets with various numbers of lines, the largest of which consists of more than thousand lines. Currently, all lines are treated equally. They are blended according to the importance function and emphasized using halos. Nonetheless, it may be advantageous to introduce an additional level of detail for even larger data sets consisting of millions of lines. For example, halos and blending could only be applied to the most important clusters and less relevant ones could be averaged, aggregated, subsampled, or only displayed as confidence bands. This would simultaneously compensate for current limitations in computing performance on even larger data sets.

## 8. Conclusion

We have presented line weaver, a novel visualization technique for dense two-dimensional line data. Line weaver allows for an optimized blending, independent of the rendering order and without costly initial sorting. The basis of line weaver is formed by a quantitative importance function which either originates from external data or is derived from (geometric) properties of the lines such as arc length, and may vary locally. Using various synthetic as well as real-world data sets, we have shown the advantages of "woven" line bundles over visualizations that ignore the non-commutativity of blending or naively assume the blending order depends on how lines were stored in a file. Furthermore, we have shown that the use of quantitative importance functions can offer additional degrees of freedom for representing focus+context information in interactive scenarios. Finally, we have demonstrated that our approach can be implemented on modern GPU architectures to provide interactive, high-quality visualizations of line-based data sets.

## Acknowledgments

The research presented in this paper was supported by the MetaVis project (#250133) funded by the Research Council of Norway.

## References

- [AdL04] ARTERO A. O., DE OLIVEIRA M. C. F., LEVKOWITZ H.: Uncovering Clusters in Crowded Parallel Coordinates Visualizations. In *Proc. IEEE Symposium on Information Visualization* (2004), pp. 81–88. doi:10.1109/INFVIS.2004.68.
- [And72] ANDREWS D. F.: Plots of High-Dimensional Data. *Biometrics* 28, 1 (1972), 125–136. doi:10.2307/2528964.
- [BCL\*07] BAVOIL L., CALLAHAN S. P., LEFOHN A., COMBA J. A. L. D., SILVA C. T.: Multi-Fragment Effects on the GPU Using the k-Buffer. In *Proc. Symposium on Interactive 3D Graphics and Games* (2007), pp. 97–104. doi:10.1145/1230100.1230117.
- [BM08] BAVOIL L., MYERS K.: Order Independent Transparency with Dual Depth Peeling. *NVIDIA* (2008).
- [BRV\*10] BRUCKNER S., RAUTEK P., VIOLA I., ROBERTS M., SOUSA M. C., GRÖLLER E.: Hybrid visibility compositing and masking for illustrative rendering. *Computers & Graphics* 34, 4 (2010), 361–369. doi:10.1016/j.cag.2010.04.003.
- [BZP\*20] BLUMENSCHNEIN M., ZHANG X., POMERENKE D., KEIM D. A., FUCHS J.: Evaluating Reordering Strategies for Cluster Identification in Parallel Coordinates. *Computer Graphics Forum* 39, 3 (2020), 537–549. doi:10.1111/cgfm.14000.



- [Car84] CARPENTER L.: The A-Buffer, an Antialiased Hidden Surface Method. *ACM SIGGRAPH Computer Graphic* 18, 3 (1984), 103–108. doi:10.1145/964965.808585.
- [CMM88] CLEVELAND W., MCGILL M. E., MCGILL R.: The Shape Parameter of a Two-Variable Graph. *Journal of the American Statistical Association* 83, 402 (1988), 289–300. doi:10.1080/01621459.1988.10478598.
- [Cor20] CORNUT O.: ImGui. <https://github.com/ocornut/imgui>, 2020. Accessed: October.
- [CWM09] CHUANG J., WEISKOPF D., MÖLLER T.: Hue-Preserving Color Blending. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 1275–1282. doi:10.1109/TVCG.2009.150.
- [DG21] DUA D., GRAFF C.: UCI Machine Learning Repository. <http://archive.ics.uci.edu/ml>, 2021. Accessed: March.
- [DLH11a] DAAE LAMPE O., HAUSER H.: Curve Density Estimates. *Computer Graphics Forum* 30, 3 (2011), 633–642. doi:10.1111/j.1467-8659.2011.01912.x.
- [DLH11b] DAAE LAMPE O., HAUSER H.: Interactive visualization of streaming data with Kernel Density Estimation. In *Proc. IEEE PacificVis* (2011), pp. 171–178. doi:10.1109/PACIFICVIS.2011.5742387.
- [DV10] DEMŠAR U., VIRRANTAUŠ K.: Space-time density of trajectories: exploring spatio-temporal patterns in movement data. *International Journal of Geographical Information Science* 24, 10 (2010), 1527–1542. doi:10.1080/13658816.2010.511223.
- [EBRI09] EVERTS M. H., BEKKER H., ROERDINK J. B. T. M., ISENBERG T.: Depth-Dependent Halos: Illustrative Rendering of Dense Line Data. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 1299–1306. doi:10.1109/TVCG.2009.138.
- [EHS13] EICHELBAUM S., HLAWITSCHKA M., SCHEUERMANN G.: LineAO—Improved Three-Dimensional Line Rendering. *IEEE Transactions on Visualization and Computer Graphics* 19, 3 (2013), 433–445. doi:10.1109/TVCG.2012.142.
- [FWR99] FUA Y.-H., WARD M. O., RUNDENSTEINER E. A.: Hierarchical Parallel Coordinates for Exploration of Large Datasets. In *Proc. IEEE Visualization* (1999), pp. 43–50. doi:10.1109/PACIFICVIS.2011.5742387.
- [GG21] GROSS D., GUMHOLD S.: Advanced Rendering of Line Data with Ambient Occlusion and Transparency. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2021), 614–624. doi:10.1109/TVCG.2020.3028954.
- [GRT13] GÜNTHER T., RÖSSL C., THEISEL H.: Opacity Optimization for 3D Line Fields. *ACM Transactions on Graphics* 32, 4 (2013), 120:1–120:8. doi:10.1145/2461912.2461930.
- [GRT14] GÜNTHER T., RÖSSL C., THEISEL H.: Hierarchical opacity optimization for sets of 3D line fields. *Computer Graphics Forum* 33, 2 (2014), 507–516. doi:10.1111/cgf.12336.
- [GTG17] GÜNTHER T., THEISEL H., GROSS M.: Decoupled Opacity Optimization for Points, Lines and Surfaces. *Computer Graphics Forum* 36, 2 (2017), 153–162. doi:10.1111/cgf.13115.
- [Hau06] HAUSER H. R.: *Generalizing Focus+Context Visualization*. Springer Berlin Heidelberg, 2006, pp. 305–327.
- [HLD02] HAUSER H., LEDERMANN F., DOLEISCH H.: Angular brushing of extended parallel coordinates. In *Proc. IEEE Symposium on Information Visualization* (2002), pp. 127–130. doi:10.1109/INFVIS.2002.1173157.
- [HSKI07] HAGH-SHENAS H., KIM S., INTERRANTE V., HEALEY C.: Weaving Versus Blending: a quantitative assessment of the information carrying capacities of two alternative methods for conveying multivariate data with color. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1270–1277. doi:10.1109/TVCG.2007.70623.
- [HYX\*16] HUANG X., YE Y., XIONG L., LAU R. Y., JIANG N., WANG S.: Time series k-means: A new k-means type smooth subspace clustering for time series data. *Information Sciences* 367-368 (2016), 1–13. doi:10.1016/j.ins.2016.05.040.
- [JLJC06] JOHANSSON J., LJUNG P., JERN M., COOPER M.: Revealing Structure in Visualizations of Dense 2D and 3D Parallel Coordinates. *Information Visualization* 5, 2 (2006), 125–136. doi:10.1057/palgrave.ivs.9500117.
- [Kil20] KILGARD M. J.: Polar Stroking: New Theory and Methods for Stroking Paths. *ACM Transactions on Graphics* 39, 4 (2020). doi:10.1145/3386569.3392458.
- [KLM\*12] KONYHA Z., LEŽ A., MATKOVIĆ K., JELOVIĆ M., HAUSER H.: Interactive Visual Analysis of Families of Curves Using Data Aggregation and Derivation. In *Proc. Conference on Knowledge Management and Knowledge Technologies* (2012), pp. 24:1–24:8. doi:10.1145/2362456.2362487.
- [KMLM16] KWON O.-H., MUELDER C., LEE K., MA K.-L.: A Study of Layout, Rendering, and Interaction Methods for Immersive Graph Visualization. *IEEE Transactions on Visualization and Computer Graphics* 22, 7 (2016), 1802–1815. doi:10.1109/TVCG.2016.2520921.
- [KNM\*20] KERN M., NEUHAUSER C., MAACK T., HAN M., USHER W., WESTERMANN R.: A Comparison of Rendering Techniques for 3D Line Sets with Transparency. *IEEE Transactions on Visualization and Computer Graphics PrePrint* (2020). doi:10.1109/TVCG.2020.2975795.
- [KS14] KINDLMANN G., SCHEIDEGGER C.: An Algebraic Process for Visualization Design. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2181–2190. doi:10.1109/TVCG.2014.2346325.
- [LCD06] LUFT T., COLDITZ C., DEUSSEN O.: Image Enhancement by Unsharp Masking the Depth Buffer. *ACM SIGGRAPH Computer Graphic* 25, 3 (2006), 1206–1213. doi:10.1145/1179352.1142016.
- [LD06] LUFT T., DEUSSEN O.: Real-Time Watercolor Illustrations of Plants Using a Blurred Depth Test. In *Proc. Symposium on Non-Photorealistic Animation and Rendering* (2006), pp. 11–20. doi:10.1145/1124728.1124732.
- [LRS10] LUBOSCHIK M., RADLOFF A., SCHUMANN H.: A New Weaving Technique for Handling Overlapping Regions. In *Proceedings of the International Conference on Advanced Visual Interfaces* (2010), pp. 25–32. doi:10.1145/1842993.1842999.
- [MF18] MORITZ D., FISHER D.: Visualizing a Million Time Series with the Density Line Chart. *CoRR abs/1808.06019* (2018).
- [NH06] NOVOTNY M., HAUSER H.: Outlier-Preserving Focus+Context Visualization in Parallel Coordinates. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 893–900. doi:10.1109/TVCG.2006.170.
- [NY10] NAKAYA T., YANO K.: Visualising Crime Clusters in a Space-time Cube: An Exploratory Data-analysis Approach Using Space-time Kernel Density Estimation and Scan Statistics. *Transactions in GIS* 14, 3 (2010), 223–239. doi:10.1111/j.1467-9671.2010.01194.x.
- [PD84] PORTER T., DUFF T.: Compositing Digital Images. *ACM SIGGRAPH Computer Graphic* 18, 3 (1984), 253–259. doi:10.1145/964965.808606.
- [RMCW18] RYAN G., MOSCA A., CHANG R., WU E.: At a Glance: Pixel Approximate Entropy as a Measure of Line Chart Complexity. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 872–881. doi:10.1109/TVCG.2018.2865264.
- [RQ21] ROSEN P., QUADRI G. J.: LineSmooth: An Analytical Framework for Evaluating the Effectiveness of Smoothing Techniques on Line Charts. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2021), 1536–1546. doi:10.1109/TVCG.2020.3030421.

- [SFB94] STONE M. C., FISHKIN K., BIER E. A.: The Movable Filter As a User Interface Tool. In *Proc. ACM CHI* (1994), pp. 306–312. doi: [10.1145/191666.191774](https://doi.org/10.1145/191666.191774).
- [SM04] SCHUSSMAN G., MA K.-L.: Anisotropic volume rendering for extremely dense, thin line data. In *Proc. IEEE Visualization* (2004), pp. 107–114. doi: [10.1109/VISUAL.2004.5](https://doi.org/10.1109/VISUAL.2004.5).
- [Spe52] SPEAR M. E.: *Charting Statistics*. McGraw-Hill, 1952, pp. 39–95.
- [Tab20] TABLEAU SOFTWARE: Control the Appearance of Marks in the View. [https://help.tableau.com/current/pro/desktop/en-gb/viewparts\\_marks\\_markproperties.htm#draw-paths-between-marks](https://help.tableau.com/current/pro/desktop/en-gb/viewparts_marks_markproperties.htm#draw-paths-between-marks), 2020. Accessed: October.
- [TBSB20] TRAUTNER T., BOLTE F., STOPPEL S., BRUCKNER S.: Sunspot Plots: Model-based Structure Enhancement for Dense Scatter Plots. *Computer Graphics Forum* 39, 3 (2020), 551–563. doi: [10.1111/cgf.14001](https://doi.org/10.1111/cgf.14001).
- [TWP17] TAN C. W., WEBB G. I., PETITJEAN F.: Indexing and classifying gigabytes of time series under time warping. In *Proc. SIAM International Conference on Data Mining* (2017), pp. 282–290. doi: [10.1137/1.9781611974973.32](https://doi.org/10.1137/1.9781611974973.32).
- [WH09] WEISKOPF D., HEINRICH J.: Continuous Parallel Coordinates. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 1531–1538. doi: [10.1109/TVCG.2009.131](https://doi.org/10.1109/TVCG.2009.131).
- [YHGT10] YANG J. C., HENSLEY J., GRÜN H., THIBIEROZ N.: Real-Time Concurrent Linked List Construction on the GPU. *Computer Graphics Forum* 29, 4 (2010), 1297–1304. doi: [10.1111/j.1467-8659.2010.01725.x](https://doi.org/10.1111/j.1467-8659.2010.01725.x).