KD-Box: Line-segment-based KD-tree for Interactive Exploration of Large-scale Time-Series Data

Yue Zhao, Jian Zhang, Chi-Wing Fu, Mingliang Xu, Dominik Moritz, Yunhai Wang

Abstract—Time-series data—usually presented in the form of lines—plays an important role in many domains such as finance, meteorology, health, and urban informatics. Yet, little has been done to support interactive exploration of large-scale time-series data, which requires a clutter-free visual representation with low-latency interactions. In this paper, we contribute a novel line-segment-based KD-tree method to enable interactive analysis of many time series. Our method enables not only fast queries over time series in selected regions of interest but also a line splatting method for efficient computation of the density field and selection of representative lines. Further, we develop KD-Box, an interactive system that provides rich interactions, e.g., *timebox*, *attribute filtering*, and *coordinated multiple views*. We demonstrate the effectiveness of KD-Box in supporting efficient line query and density field computation through a quantitative comparison and show its usefulness for interactive visual analysis on several real-world datasets.

Index Terms—Many time series, density-based visualization, interactive visualization for large-scale data

1 Introduction

An increasingly large amount of time-series data are being collected, stored, and analyzed in a broad range of domains such as finance, health, and urban informatics. To gain insights into these time series, analysts usually need to explore, compare, and relate the data of multiple entities whose numbers can range from dozens to millions, e.g., multiple stocks, machine's power consumption, etc. Thus, interactive analysis systems that scale to large time-series data are in great demand.

Typical methods [19] visualize time-series data as individual lines in a line graph. These methods usually provide dynamic query tools for users to interactively filter a subset of lines that satisfy certain patterns to enable interactive exploration. As the number of time series increases, such line graphs often suffer from overplotting and slow querying, hindering analysts to interactively explore the data. To reduce visual clutter, researchers have proposed density-based methods [11, 37] to visualize many time series together as a density field rather than as individual lines. However, density-based visualizations do not support the interactive exploration of time-series data mainly due to three limitations. First, the calculation and rendering of a density visualization on the CPU are too heavy to provide interactive feedback. Second, querying a subset of lines from a density visualization is almost impossible since the visualization is merely a raster image, which omits the individual lines. Third, while density visualizations give analysts a sense of the overall density pattern of the data, the detailed information of individual lines is lost, especially the representative continuous trends. Therefore, density visualizations hinder analysis tasks that involve individual time series, e.g., *what are the representative stocks in a given time interval?* Clustering of time-series lines might be possible for identifying representative lines, however the involved computational cost is too high [1] for interactive exploration. These limitations motivate us to study *how to enable interactive analysis of many time series while combining the advantages of line graphs and density fields*.

Supporting interactive exploration of large-scale time-series data is a non-trivial task. The challenges lie in three fundamental requirements that existing density-based visualizations cannot satisfy:

(i) **fast rendering**—the calculation and rendering of the density field must be fast enough to give responsive feedbacks to users;

(ii) **fast querying**—the visualization must allow low-latency interactions to query line subsets in time-series exploration; and

(iii) **representative lines**—the visualization must also present major trends by showing representative lines in regions of interest.

These challenges become more serious, as the number of lines (time series) increases. We need novel approaches that can simultaneously present global trends and local representative time series, while supporting fast rendering and querying.

In this paper, we contribute a novel K-Dimensional(KD)-tree method to enable interactive analysis of many time series. At the core of our method is a line-segment-based KD-tree [31] we adapted for time-series visualization. The KD-tree is constructed by first segmenting each time series into a few line segments and then building a KD-tree for all line segments. To support various query operations for interactive exploration, e.g., rectangular brush and angular query [19], we build a 3D KD-tree in the time-value-slope space. Such a KD-tree can be built in less than 30 seconds for a dataset with 100K time series and 13.5 million data points. With our incremental query, line search can be completed in less than 15 milliseconds on average with a search accuracy of over 98.5% for the various datasets we tested in this work.

With an efficient radius nearest line (RNL) search, our KD-tree enables efficient computation of density field via line gathering [6, 58]. For each pixel in a given display, we perform an RNL search to find all line segments within a given radius r and quickly obtain the combined density of these segments. Our method enables fast rendering of the density of many lines in an order of magnitude less time than the state-of-the-art implementation [30]. Once the density field is available, we provide a heuristic method to locate representative time series in a selected region of interest. Our method finds diverse time series, while considering the density and shape of the lines. Overlaying such representative time-series lines on the density field or all the selected lines facilitates users to interactively explore local patterns, while showing the global trend of the entire data.

We develop KD-Box[1], an interactive system with flexible interactions with Timebox [19] and coordinated views for visual exploration of large-scale time-series data. To evaluate our method, we compare it with existing approaches of density field computation and dynamic

- *Y. Zhao and Y. Wang are with Shandong University, Qingdao, China. E-mail: {jack.zhao9802,cloudseawang}@gmail.com.*
- *J. Zhang is with CNIC, CAS. E-mail: zhangjian@sccas.cn.*
- *C.-W. Fu is with the Chinese University of Hong Kong. E-mail: cwfu@cse.cuhk.edu.hk.*
- *M. Xu is with Zhengzhou University, China. E-mail: iexumingliang@zzu.edu.cn.*
- *D. Moritz is with Carnegie Mellon University, United States. E-mail: domoritz@cmu.edu.*
- *Y. Wang is the corresponding author.*

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

[1] https://kd-box.github.io

query. The experimental results show that even with 100 K time series, our method can deliver real-time interactivity on high-resolution (1200×600px) visualizations. Also, we demonstrate how KD-Box enables analysts to quickly test different hypotheses on their large real-world time-series data. Our main contributions are listed below:

- An efficient line-segment-based 3D KD-tree built in the time-value-slope space that enables incremental query update for interactive exploration of many time series;
- Fast density field computation by KD-tree-based line gathering and a heuristic method for selecting representative lines; and
- An interactive exploration system with rich interactions that demonstrates the effectiveness of our method via a quantitative comparison and applications on large real-world datasets.

2 Related Work

2.1 Visualization for Many Time Series

Analysts in many domains need to visualize time-series data with interactive exploration. To reduce overplotting, researchers have proposed clutter-free visualizations and interactive query methods.

Clutter-free visualization. Many visualization approaches have been introduced for visualizing time-series data, such as calendar-based [49], circle-based [54], symbol-based [25], and tile-based [44]. Among them, line graph is the most common way to present time-series data. Javed et al. [21] summarized the trade-offs among the line-graph-based techniques. All these methods, however, suffer from overplotting when the underlying data is large. Lopez-Hernandez et al. [29] developed a 2.5D layer-based technique for many time series, but it is designed mainly for time series with very small ranges in the vertical direction.

Instead of visualizing individual lines, Muigg et al. [40] compute a frequency binmap [42] (a discrete density field) by counting the number of time series that pass through each pixel. With three additional degree-of-interest (DOI) variables per data item, their four-level focus+context approach composites four density fields of a time series. Later, Lampe and Hauser introduced continuous line kernel [23] and further proposed *Curve Density Estimates* (CDE) [11] to visualize the density of many lines, aiming to resolve the visual clutter issue. Such density field enables users to see the overall trends in the data. However, the calculation of CDE is too slow to support interactive exploration. Inspired by CDE, Moritz and Fisher present DenseLines [37] that aggregates the data into bins and generates a discrete density representation of the time-series data. They further suggest normalizing density by arc length to remove artifacts in the density fields. Though DenseLines can be computed faster than CDE, it is still not fast enough to generate smooth density in real-time for interactive exploration of 100k lines (see Sect. 5.1). In this work, our method ensures a smooth interaction with the help of a pre-computed line-segment-based KD-tree.

Interactive query. Instead of visualizing all time series at once, an alternative approach to reduce visual clutter is to allow the user to interactively query a subset of lines for display. For instance, Timeboxes [19] and QuerySketch [53] provide interaction widgets that allow users to query specific time series based on their shapes. Zhao et al. [57] propose ChronoLenses, which uses a lens metaphor to support exploration of local regions in time-series data. However, these methods do not scale well to large time-series data because the queries supported in these systems are too slow for interactive exploration.

Researchers have also developed techniques to speed up the query process for large-scale data. Most of these techniques use advanced data management techniques, such as data cubes [26, 28], approximation [8, 38], and prefetching [3, 39]. Some of them have been optimized to support large time-series data. ATLAS [7] combines predictive caching and level-of-detail management to ensure smooth interactions in exploring massive time-series data. Time Lattice [35] customizes a data-cube structure for time-series data with an implicit temporal hierarchy, resulting in multi-resolution queries on time series at an interactive rate. However, these methods visualize time series with line graphs and cannot be adapted for density-based visualizations. The four-level focus+context framework [40] supports query in density fields but displays the results as another DOI variable's density field, losing the details of the selected (original) lines. In contrast, our KD-Box allows users to interactively query large-scale time-series data with flexible interactions introduced by Timebox [19] and simultaneously supports density-based visualization and representative-line selection.

Representative-line selection. Clustering is a common solution for picking representative lines from time-series data in scientific visualization. Moberts et al. [36] compared different clustering methods and distance measures for DTI clustering, and Yu et al. [56] applied hierarchical clustering to streamlines and created a hierarchy of streamline bundles. However, most of these techniques are not interactive due to the heavy computational cost of clustering. For time-series data, representative series can also be selected by directly clustering the time series [1], but doing so is computationally expensive. To bypass this issue, we advocate a heuristic method based on the density field to efficiently identify representative time series.

2.2 Density-based Visualization

Density-based visualizations are commonly used to declutter charts with many data points. A naive way to show the density is to make individual marks semitransparent and employ alpha blending to show dense regions as more pronounced [10]. More sophisticated methods aggregate data points that are close to one another and visualize the computed density field. Heatmaps show density as colors using scales that can be tuned to highlight large ranges (e.g., using a log scale) or highlight the difference between no data and some data [22].

Typical examples of density visualizations are density-based scatterplots. Mayorga and Gleicher introduced Splatterplots [34], which automatically groups dense data points into density contours and samples the remaining points. Splatterplots combine data points and density contours through perceptually accurate color blending. Other methods blend data points and density contours by reducing the opacity of densities [33] or by smoothing [55]. Density visualization has also been used in other chart types, such as trajectory maps [43], parallel coordinates [2, 13], and graph visualization [24, 48].

Most existing techniques estimate density via splatting that convolves each primitive sample (node or edge) with various kernels and accumulates the contributions from all samples. For example, Graph splatting [48] convolves each node with a Gaussian kernel for rendering large graphs without visual clutter. Telea and Ersoy [45] and Heinrich et al. [17] use edge splatting to produce image-based edge bundling and continuous parallel coordinates, respectively. As an object-order approach, the complexity of the splatting techniques depends on the number of visual primitives, so they might be slow for many time series on the CPU. In contrast, our density field computation is an image-order approach. Its complexity depends on the screen space, or the number of pixels. Accordingly, our method gains more advantages as the number of objects increases.

For data points sampled from time-series data, Lampe and Hauser-proposed Curve Density Estimates (CDE) [11], which shows distributional characteristics along the time axis enabled by a moving, column-based normalization. By showing the density field, CDE achieves clutter-free visualizations but it is difficult to implement and slow to compute. Moritz and Fisher [37] introduced DenseLines, a special case of CDE that groups data into bins and generates discrete density, which is multiple orders of magnitude faster than CDE on the GPU. However, both methods omit the information of individual lines, so users cannot explore details in the time-series data. Our method improves over existing density-field-based time-series data visualization in three aspects. First, we design a 3D line-segment-based KD-tree in time-value-slope space to enable interactive query of time-series lines in density fields. Second, we propose fast computation of density fields for time series via KD-tree-based RNL search. Last, we present both the density and representative lines together, giving analysts details about the individual lines to help explore local regions of interest.

3 Background: Timebox and Curve Density Estimate

In this section, we introduce nearest line queries and timebox queries, review the background of Curve Density Estimate (CDE), and discuss how lines query technique can be applied to CDE.

Nearest Line Queries. Given a set of curves, Lu et al. [31] proposed two neighbor-based line query techniques: radius-nearest line (RNL) query and k-nearest line (KNL) query. Given a query point $\mathbf{q}$, RNL finds all curves within a distance of r from $\mathbf{q}$, whereas KNL finds the k curves nearest to $\mathbf{q}$. Fig. 1a illustrates RNL and KNL queries using query points $\mathbf{q}_1$ and $\mathbf{q}_2$, respectively. Note that the point in a curve with the shortest distance to $\mathbf{q}$ might not be a sample point on the curve.

TimeBox and Angular Queries. Given a line graph with a set of time series, users can select the series that cross a region of interest specified by brushing with timeboxes [18] or attribute-based filtering. Denoting a box query as a 4-tuple $(t_{\min}, t_{\max}, v_{\min}, v_{\max})$, which specifies ranges in the time and value dimensions, we locate curve c_i (the i-th time series) with the condition that all points in c_i within time range $[t_{\min}, t_{\max}]$ should lie within the given value range $[v_{\min}, v_{\max}]$. Fig. 1b shows an example, in which curves c_1 and c_2 are excluded in the query results, as some of their sample points are above $v_{\max}$ or below $v_{\min}$ in the given time range. For queries with multiple timeboxes, the retrieved series need to satisfy the conditions of all the timeboxes.

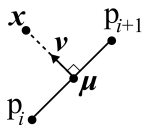
The angular query works in the same way as the timebox query, but the range is specified in the time-angle instead of the time-value domain. For each line segment in a curve, calculating its slope angle involves an expensive arctan function, so we convert the angle-range constraint into a slope-range constraint. Fig. 1c shows an example: the angle-range constraint is specified by $(t_{\min}, t_{\max}, \theta_{\min}, \theta_{\max})$ shown on the left, whereas the converted slope-range constraint is shown as piece-wise line segments in the time-slope space shown on the right. From the right figure, we can see that only curves c_2 (yellow) and c_3 (green) are retrieved as the query results.

A timebox can be regarded as a two-dimensional rectangular-range query, and thus, some efficient search methods [47], e.g., orthogonal range trees and grid structures, have been suggested. However, Hochheiser and Shneiderman [19] quantitatively compared the performance of various methods and found that sequential search still outperforms others. With a careful analysis, they attributed the success of sequential search to the early termination, i.e., the search can stop whenever one data point is found out of range, whereas the other methods need to examine every point in the timebox. Since the box is movable and resizable, they further developed an optimized sequential search algorithm [18] for providing rapid and incremental queries.

Curve Density Estimate (CDE). Given a list of 1D points $\{p_1, \cdots, p_n\}$ sampled from an unknown probability density, the density at position x estimated by kernel density estimation (KDE) is

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} K_r(x, p_i) \tag{1}$$

where K_r is a kernel function and r is the bandwidth. A common kernel function is the normal kernel $N_r(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x/r)^2}$. For two discrete points, the normal kernel produces a distribution with two peaks (Fig. 2a). However, when the two consecutive points are sampled from a continuous time series, this distribution cannot faithfully reveal the continuous change between successive points.



To tackle this problem, CDE introduces a 2D line kernel along each line segment, i.e., consecutive points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ in curve c_i. Given an arbitrary point $\mathbf{x}$ in the 2D space, let $\boldsymbol{\mu}$ be the foot of the perpendicular when projecting $\mathbf{x}$ onto the line segment, i.e., $\boldsymbol{\mu} = \mathbf{x} - |\boldsymbol{\nu} \cdot (\mathbf{x} - \mathbf{p}_i)|\boldsymbol{\nu}$, where $\boldsymbol{\nu}$ is the unit vector perpendicular to the line segment and towards $\mathbf{x}$. The 2D line kernel at $\mathbf{x}$ is the product of a 1D line kernel along the line segment (L_r^{1D}) and a normal kernel (N_r) along the $\boldsymbol{\nu}$ direction perpendicular to the line segment:

$$L_r(\mathbf{x}, \mathbf{p}_i, \mathbf{p}_{i+1}) = L_r^{1D}(\mu, p_i, p_{i+1}) \cdot N_r(|\mathbf{x} - \boldsymbol{\mu}|) \tag{2}$$

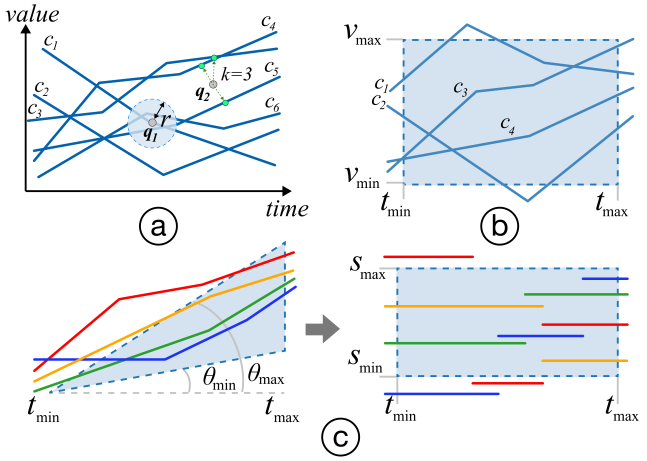


Fig. 1. Illustrations of four different lines query methods: (a) radius nearest lines (RNL) query locates lines c_1, c_5, and c_6 for query point $\mathbf{q}_1$ and radius r, where k-nearest lines (KNL) query locates lines c_3, c_4, and c_5 for query point $\mathbf{q}_2$ and $k = 3$; (b) timebox query locates lines c_3 and c_4 within the box; and (c) angular query: user brushes in the time-angle domain (left) and the computation is done in the time-slope domain (right), in which continuous lines become piecewise line segments.

where μ, p_i, and p_{i+1} are the 1D equivalents (defined along the straight line through $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$) of the 2D points $\boldsymbol{\mu}$, $\mathbf{p}_i$, and $\mathbf{p}_{i+1}$, respectively. The 1D line kernel L_r^{1D} is a subtraction of two cumulative distribution functions (CDFs) of the normal kernel:

$$L_r^{1D}(\mu, p_i, p_{i+1}) = \frac{\mathrm{CDF}_r(\mu, p_i) - \mathrm{CDF}_r(\mu, p_{i+1})}{|p_{i+1} - p_i|} \tag{3}$$

where $\mathrm{CDF}_r(\mu, p_i)$ is the integral of the normal kernel:

$$\mathrm{CDF}_r(\mu, p_i) = \int_{p_i}^{\mu} N_r(|t - p_i|)\, dt. \tag{4}$$

As a result, the estimated density is distributed evenly from one point to the next (see Fig. 2b). The density at each point $\mathbf{x}$ is the summation of the density from all individual line segments.

KDE often uses only the points within radius range r to estimate the density while ignoring the other points. Similarly, computing the line density needs to consider only the lines within the radius distance r from $\mathbf{x}$. Although KD-trees have been widely used for r-nearest neighbor (RNN) search [41], only curve complexity heuristic (CCH) [31] KD-trees developed for the search of nearest lines from a set of 3D curves. In this work, we extend such trees to further support timebox and angular queries of many time series.

4 Method

In this section, we first present an adapted CCH KD-tree [31], a line-segment-based KD-tree, for exploring time-series data. Like the original CCH KD-tree, we also fit each line (curve) with a small number of straight-line segments, but we build our line-segment-based KD-tree in the time-value-slope (TVS) space. Based on such a tree, we can perform KNL and RNL queries similar to nearest points queries in traditional point-based KD-trees [31]. In addition, we extend the line-segment-based KD-tree in the TVS space with three techniques for exploring large-scale time-series data: (i) an efficient density computation, (ii) incremental timebox queries, and (iii) representative line

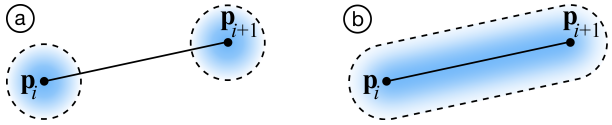


Fig. 2. Given a line defined by two points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$, its density calculated by (a) normal kernel and by (b) line kernel.

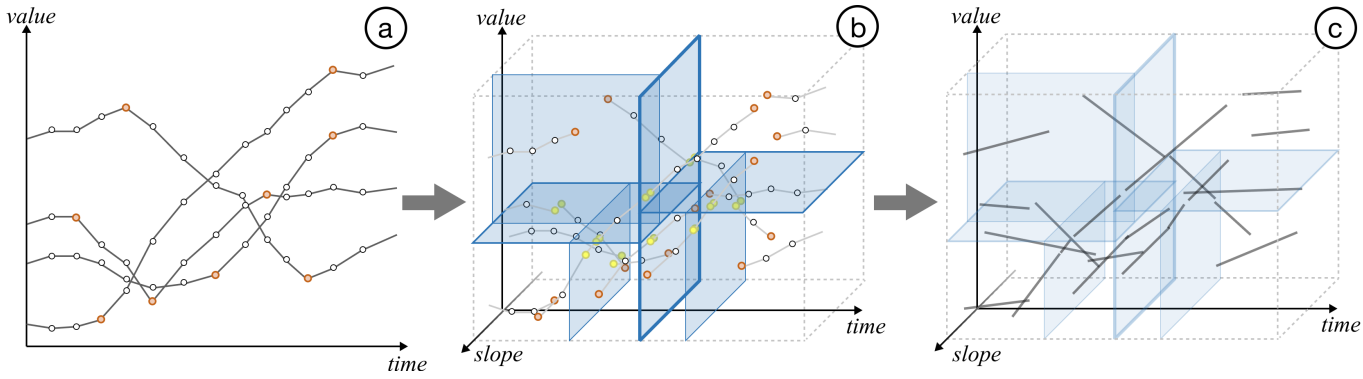


Fig. 3. Overview of our tree construction: (a) a set of input curves with point samples (white) and split points (orange); (b) the KD-tree in TVS space built for the input curves with the newly-added split points (yellow), the thickness of split planes' borders (blue) indicates the level, where split planes of thicker borders are in the upper (higher) levels of the KD-tree; and (c) the curve segment in each grid cell volume is fit by a straight-line segment.

selection. In the following, we detail the tree construction then present the exploration techniques.

4.1 Pre-Processing

Given n time-series lines, our line-segment-based KD-tree is constructed in three steps, as illustrated in Fig. 3: (i) decompose each line into segments; (ii) construct line-segment-based KD-trees; and (iii) fit each segment with a straight-line segment.

Curve Decomposition. Considering each line in a given time-series data as an ordered list of points $\{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_m\}$, we first apply the classic Ramer-Douglas-Peucker algorithm [50] with threshold ε to approximate each line with a small number of straight-line segments. We proceed by finding point $\boldsymbol{p}_i$, which has the largest distance deviation from the straight-line segment l that connects endpoints $\boldsymbol{p}_1$ and $\boldsymbol{p}_m$. If the distance deviation is less than threshold ε, all points between $\boldsymbol{p}_1$ and $\boldsymbol{p}_m$ can be removed, as the whole line can be approximated by l. Otherwise, we keep $\boldsymbol{p}_i$ and repeat the process recursively on $\{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_i\}$ and on $\{\boldsymbol{p}_i, \cdots, \boldsymbol{p}_m\}$, in which $\boldsymbol{p}_i$ is referred to as a *split point*; see the orange points in Fig. 3a. A larger ε tends to remove more points and the curve would have larger changes in shape, and vice versa. By default, ε is set to one pixel unit in our experiment.

Tree Construction. We build our line-segment-based KD-tree in TVS space by recursively bisecting the 3D axis-aligned bounding volume of all line segments. Taking the whole TVS volume as the tree's root, the bisection has three steps: (i) find a candidate split plane for each dimension in the volume using spatial median [51]; (ii) use the split plane with the least query cost (defined by Lu et al. [31]) to divide the current node into two child nodes; and (iii) divide the line segments that intersect the split plane, and lengthen each of them to reach the associated intersection point on the split plane. Such intersections (see the yellow points in Fig. 3b) are also regarded as split points for better curve fitting. The recursion stops when the cost of all candidate split planes is less than zero. Since the split-plane selection accounts for the query cost, the resulting tree is optimized as a nearest-line query.

Curve Fitting. After the tree is constructed, each time-series line is represented by multiple straight line segments, each stored in a corresponding cell. Such segment can be obtained by connecting the two endpoints of the time-series line in the grid cell, or by projecting the 2D point samples to a line segment with a principal component analysis. We use the former method, since it is faster and induces less error in most cases. Lastly, each leaf node contains six values per line segment (i.e., two endpoints in the time-value space, a slope value, and a line index), while each internal node records the split dimension and the node's bounding volume range (min and max) in the split dimension. Fig. 3c shows all the fitted line segments in the TVS space.

4.2 Efficient Density Computation

Once a KD-tree is built, we can perform a per-pixel nearest line (NL) query to construct the density field. For each pixel $\mathbf{x}$ in the time-value space, we compute its density by performing an RNL query to gather the line segments within radius r (see Fig. 1a) and then summing up the contribution of each line segment calculated by Equation 2. Thanks to the efficient RNL query, our method is even faster than the classical splatting-based methods [6, 37] (see Sect. 5).

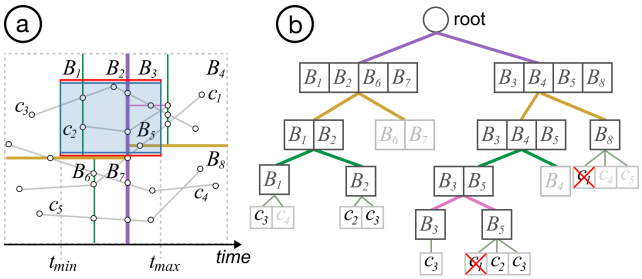


Fig. 4. Illustrating tree-traversal query and boundary-based filtering. (a) Decompose the space covered by time series c_1 to c_5 with the tree shown in (b); B_1 to B_7 denote the bounding volumes (grid cells) associated with the tree's leaf nodes. The blue box denotes a timebox query with a pair of ghost ranges (red) on top and bottom. (b) Traversing the tree can efficiently find all leaf nodes and lines (c_1, c_2, and c_3) that intersect the timebox. Next, boundary-based filtering employs the two ghost ranges to filter the lines: we discard c_1, since it intersects the bottom ghost range, meaning that some of its points in $[t_{\min}, t_{\max}]$ must be out of the timebox.

4.3 Rapid Incremental Range Query

To support timebox query and angular query, we define the associated range query R as a six-tuple $(t_{\min}, t_{\max}, v_{\min}, v_{\max}, s_{\min}, s_{\max})$ in the TVS space. Since timebox queries do not constrain angles, we set $s_{\min}$ as negative infinity and $s_{\max}$ as positive infinity to skip the slope dimension in timebox queries. Similarly, we set $v_{\min}$ as negative infinity and $v_{\max}$ as positive infinity for angular queries.

Tree-Traversal Query. Given a six-tuple range query R, the tree-traversal query aims to find all time series (lines) that intersect with the TVS volume specified by R. Such a traversal starts from the root and can be done very efficiently. At each internal node, we only need to compare the node's bounding volume range with R's corresponding range in the split dimension. Suppose the split dimension is in "value," we only need to compare the "value" range of the node's bounding volume and the "value" range of R: There are three cases:

- Case 1: If the node's range is entirely in R's range, we directly go to all its leaf nodes and gather all the stored curve indices;
- Case 2: If the node's range is partially in R's range, we traverse to each of its child nodes and further check each child node; and
- Case 3: If the node's range is completely out of R's range, we stop the traversal of this node.

In Case 2, if a children node is a leaf node, we further need to check if each of its contained line segments is inside R. Fig. 4a shows an example that the bounding volume of leaf node B_1 intersects R but not

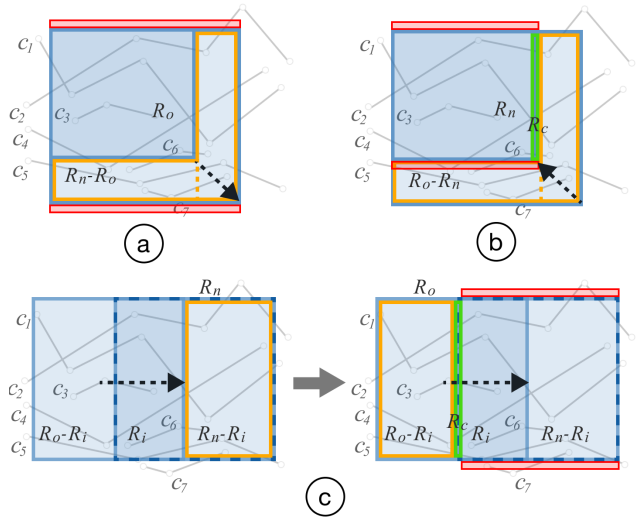


Fig. 5. Illustrations of the incremental query update for three different cases of modifying the query widget of a range query, from R_o to R_n. (a) Case 1: when enlarging the query widget, we insert the lines in range $R_n - R_o$. (b) Case 2: when reducing the query widget, we remove the lines in range $R_o - R_n$ but retain those that pass through the green ghost range. (c) Case 3: when moving the query widget, we need to insert the lines in $R_n - R_i$ following Case 1 and then remove the lines in $R_o - R_i$ following Case 2, where $R_i = R_o \cap R_n$. For all three cases, we need to remove lines that cross the red ghost ranges in the end.

all curves in B_1 really intersect R, e.g., c_4. Fig. 4b illustrates the whole tree-traversal procedure. Once the traversal is done, we obtain a set of lines (denoted as $\bar{\Omega}$) that intersects the given range query R.

Boundary-based Filtering. With tree-traversal query, we can obtain $\bar{\Omega}$ for a given range query R. Yet, to fulfill a timebox/angular query, we need to find lines that lie within R's query volume for the entire time range $[t_{\min}, t_{\max}]$ of R. Hence, we need to filter $\bar{\Omega}$ by discarding the unqualified lines, e.g., c_1 for the blue timebox shown in Fig. 4a.

Our key idea is to make use of the efficient tree-traversal query to quickly filter $\bar{\Omega}$; that is, we define a pair of ghost ranges (parallel to the time axis) next to R's query volume and discard any line in $\bar{\Omega}$ that intersects these ghost ranges. For the case of timebox query (see Fig. 4a for an example), we define ghost ranges $(t_{\min}, t_{\max}, v_{\max}, v_{\max} + \varepsilon, -\infty, \infty)$ and $(t_{\min}, t_{\max}, v_{\min} - \varepsilon, v_{\min}, -\infty, \infty)$ with a small threshold ε; see the two small red boxes in Fig. 4a. Then, we use tree-traversal query to locate lines that intersect the ghost ranges. If a line in $\bar{\Omega}$ is found to cross these ghost ranges, it means that a portion of the line is outside the timebox for time range $[t_{\min}, t_{\max}]$. Hence, such line should be excluded from the timebox's query result. By doing so, we can obtain the final query result Ω, which includes the desired lines in $\bar{\Omega}$. Though the time complexity is $O(\log N + k)$ (N being the number of all line segments and k, the number of line segments in $\bar{\Omega}$), the procedure is still very fast, especially when configured with the incremental query update to be presented later. Also, we can jointly query the lines in both ghost ranges to improve performance. Fig. 4b shows an example, where c_1 is discarded after this boundary-based filtering, since c_1 crosses the bottom ghost range (in red).

For the case of angular query, the above ghost ranges cannot work, since the time-series lines in the slope dimension are discrete rather than continuous. Hence, rather than having small ghost ranges attached on the bounding volume of the range query R, we define the two ghost ranges as $(t_{\min}, t_{\max}, -\infty, \infty, s_{\max}, \infty)$ and $(t_{\min}, t_{\max}, -\infty, \infty, -\infty, s_{\min})$, such that they extend to infinity and help to check if any line in $\bar{\Omega}$ go out of R for the given time range of R.

Incremental Query Update. When the user resizes or moves the query widget in the visualization interface that marks a range query (timebox/angular query), we can perform an incremental query to rapidly update Ω. Suppose the previous range is R_o and the new range is R_n, their spatial relationship can be one of the following three cases (see the corresponding illustrations in Fig. 5a-c):

- Case 1: $R_o \subset R_n$ when enlarging the query widget;
- Case 2: $R_n \subset R_o$ when reducing the query widget; and
- Case 3: R_o and R_n intersect by range R_i when moving the query widget.

Assuming that the line segments are uniformly distributed in space [31], searching for lines in the whole range R_n is slower than searching in a small sub-range. Hence, we design a family of incremental query mechanisms to handle the three cases between R_o and R_n.

For Case 1, we first find all lines that cross the range $R_n - R_o$ (see Fig. 5a) and add them to Ω. Since $R_n - R_o$ is L-shaped, we break its region into two regular boxes (by the dashed orange line shown in Fig. 5a) before finding lines that intersect the boxes. Also, we set a pair of ghost ranges (see the red boxes in Fig. 5a) to ensure the lines are strictly inside R_n for the whole time range.

For Case 2, we find lines that cross $R_o - R_n$ (again divided into two boxes) and remove the retrieved lines from Ω. However, such a removal might rule out some valid lines. Taking Fig. 5b as an example, it will remove lines c_1, c_2, c_4, c_5, c_6, and c_7; among these lines, c_1 and c_2 fulfill R_n. To resolve this issue, we construct ghost range R_c at R_n's left or right border next to $R_o - R_n$; see the green box in Fig. 5b; then, we can add back the lines that cross R_c but not the red ghost ranges as in Case 1. In this way, we can retain c_1 and c_2.

Case 3 is a combination of Cases 1 and 2. Incremental query update is done in two steps. First, we find the lines in range $R_n - R_i$ and insert them into Ω, following Case 1 (see the left side of Fig. 5c), where $R_i = R_o \cap R_n$. Then, we follow Case 2 to remove the lines in range $R_o - R_i$ but retain those that pass through ghost range R_c. Note again that for all three cases, we use the red ghost ranges to ensure all resulting lines are strictly in R_n for the whole time range.

The time complexity is $\log(N) + k$, which includes the cost of querying N line segments and k set operations to insert or remove lines from the query result Ω (k is the number of lines involved in the query).

4.4 Representative Lines Selection

Analysts often need to see representative time series (as individual lines) to explore the local patterns in ranges of interest, whereas density-based visualization inherently loses the information of individual lines. One straightforward solution to address this issue is to cluster the time-series lines in terms of similarity metrics [1]; however, such computation is too expensive for interactive query response times.

Our goal is to find representative lines that follow major trends in the data. Inspired by density-based edge-bundling [24], we assume that a line can represent many time series well if it passes through high-density regions as much as possible. On the other hand, each representative line should be unique, in terms of revealing diverse trends of all the lines. Here, we use the curvature differences to characterize the variation between lines [52]. Also, a representative line should not be too short. Accordingly, we propose a heuristic method to select k representative lines by considering both their density scores and curvatures.

For each line retrieved from a given range query R, we compute its density score by summing the density of all its occupied pixels:

$$\rho_i = \frac{1}{\mathrm{L}_i} \sum_k \rho(\mathbf{x}_{i,k}) \tag{5}$$

where $\mathbf{x}_{i,k}$ is the kth pixel located at the ith line, and L_i is the length of the line segment in the time axis. Meanwhile, we compute the shape difference between two lines c_i and c_j by:

$$\beta_{i,j} = \frac{1}{\mathrm{L}_{i,j}} \sum_k |\kappa(\mathbf{x}_{i,k}) - \kappa(\mathbf{x}_{j,k})| \tag{6}$$

where $\kappa(\mathbf{x}_{i,k})$ is the curvature at the kth pixel $\mathbf{p}_k$ of the ith line, and $\mathrm{L}_{i,j}$ is the common length range of the ith and jth lines in the time axis.

To enable fast subsequent queries, we pre-compute the curvature $\kappa(\mathbf{x}_{i,k})$ for each pixel point in every line and calculate ρ_i for each line

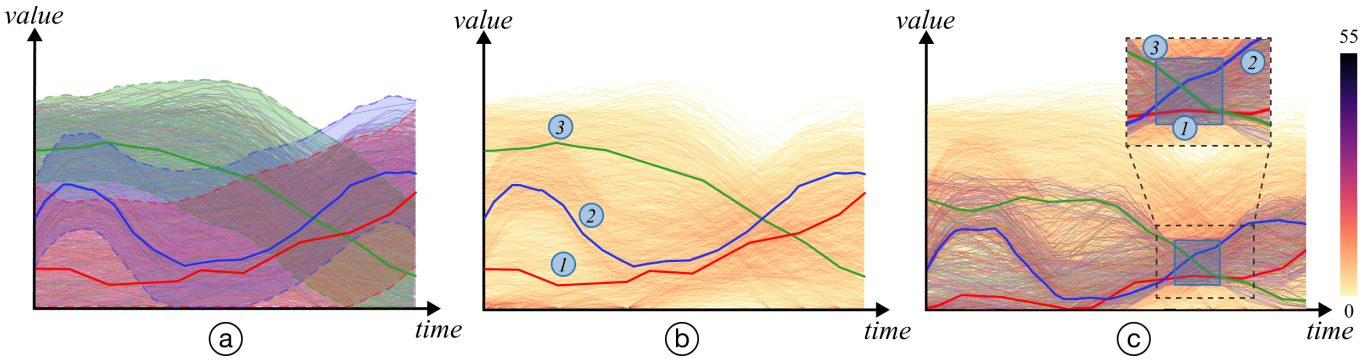


Fig. 6. Querying representative lines on a synthetic dataset with many time series. (a) An overplotting line graph consists of lines coming from three groups whose representative lines are highlighted; (b) a density visualization reduces the visual clutter with three representative lines aligned with the overall trends in the density field; and (c) we use the timebox widget (in blue) to query a set of lines and then find three representative ones. In (b & c), the line indices (one to three) indicate a decreasing density order of the lines.

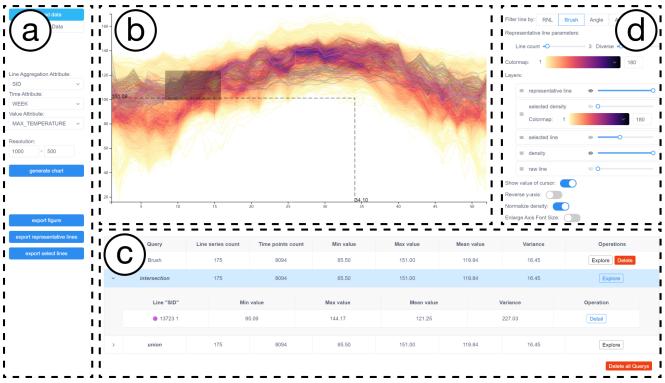


Fig. 7. The interface of our interactive exploration system KD-Box: (a) Data panel for loading the data and specifying the data fields; (b) Main view for showing the resulted density field; (c) Detail View for showing the statistical information of the query results; and (d) Control Panel for users to adjust the parameters in interaction and rendering.

segment, once the KD-tree is generated. With two user-provided parameters, i.e., length threshold τ and shape difference threshold λ, we find representative lines for a given range R in two steps. After computing ρ_i and $\beta_{i,j}$ for all the lines in R, we sort the lines in descending order of their density scores and initialize a queue with the line that has the highest score. Then, we pick the next line subject to two conditions: (i) the curvature differences $\beta_{i,j}$ between the next line and those already in the queue should be larger than λ; and (ii) the length of the next line should be larger than τ. We iterate this process until finding the desired number of representative lines, which is three by default.

In our experiments, we found that setting τ and λ to $(t_{\max} - t_{\min})/2$ and 0.1 respectively works well for most data, and both parameters can be interactively adjusted by users. Fig. 6 shows an example on a data set synthesized by moving each of three specified trend lines (see the highlighted lines in Fig. 6a) vertically by a random value drawn from a Gaussian distribution while perturbing each data point $\mathbf{x}_{i,k}$ with small Gaussian noise. The representative lines identified by our method shown in Fig. 6b are the same as the ground truth in Fig. 6a. Fig. 6c shows the representative lines for the timebox query, for which we overlay the selected lines on the density field.

4.5 Interactive Exploration System

We implemented the methods presented above in KD-Box, a JavaScript-based prototype web application for interactive exploration of time-series data without any server-side components. Fig. 7 shows the user interface of KD-Box, which has four parts. When a data set is loaded from the *Data Panel* (Fig. 7a), an overview of the data is shown using two coordinated views: the *Main View* (Fig. 7b) and the *Detail View* (Fig. 7c), which, respectively, show the density field and the detail of the corresponding representative curves and query widgets if users specified. Also, users can hover on any information in the *Detail View*, and the corresponding curve or widget will be highlighted in the *Main View*, and vice versa. Further, users can customize the rendering layers, specify parameters, and adjust color maps in the *Control Panel* (Fig. 7d), while interactively exploring the data with the provided query tools. In the following, we describe the rendering layers and query tools.

Rendering Layers. There are five rendering layers in the *Main View*: the density field, the line graph of all time series, the line graph of the queried lines, the density field of the queried lines, and the representative lines. For each query operation, the default layer is the density field, while users can combine and order any layer with adjustable transparency. For the selected density field, users can apply a different color map selected in the control panel. After the user specifies a query, the system will overlay the representative lines on the density field and show the corresponding details in the *Detail View* by default.

Query Tools. We support querying a subset of lines by using four well-known interactions, namely, timebox, angular query, mouse hover, and attribute filtering. For the first two tools, please see the details in Sect. 3; attribute filtering allows users to filter time series by specifying multiple attribute values, whereas mouse hover allows the user to interact with individual lines. To implement this feature, we reuse the efficient RNL query with a radius of 3×3 pixels centered at the user's cursor, while allowing the user to enlarge the radius with the mouse wheel.

5 Comparative Evaluation

We evaluate our technique in two aspects: *line query* and *density rendering*, on a computer with an Intel Core i5-8400 CPU @ 2.8GHz, 32GB memory, running Google Chrome 89.

Datasets. We follow existing practices [16, 21] to generate synthetic datasets with varied distributions. Based on the non-seasonal time-series model [15], we synthesize a time series with two components:

$$v_t = \mathbf{T}_t + \beta \mathbf{W}_t, \tag{7}$$

where v_t is the value at the tth time step, $\mathbf{T}_t$ is the trend component and $\mathbf{W}_t$ is the white noise. The trend component $\mathbf{T}_t$ is computed by randomly selecting one from a list of seven trend distributions, such as Gaussian, exponential, Poisson, linear, log, sine, and cosine functions, and the parameters for white noise $\mathbf{W}_t$ and β are randomly generated.

To investigate how the performance of our method varies with the different time-series data, we generated datasets by varying two parameters: the number of time series and number of time points in each time series. It is done by fixing one parameter to 100 and varying the other from 1 K to 10 K with a step size of 1 K and from 10 K to 100 K with a step size of step 10 K. In total, we generated 38 datasets.

5.1 Line Query

We conducted a comparative evaluation of four selected methods by performing (i) RNL, (ii) timebox query, and (iii) angular query using

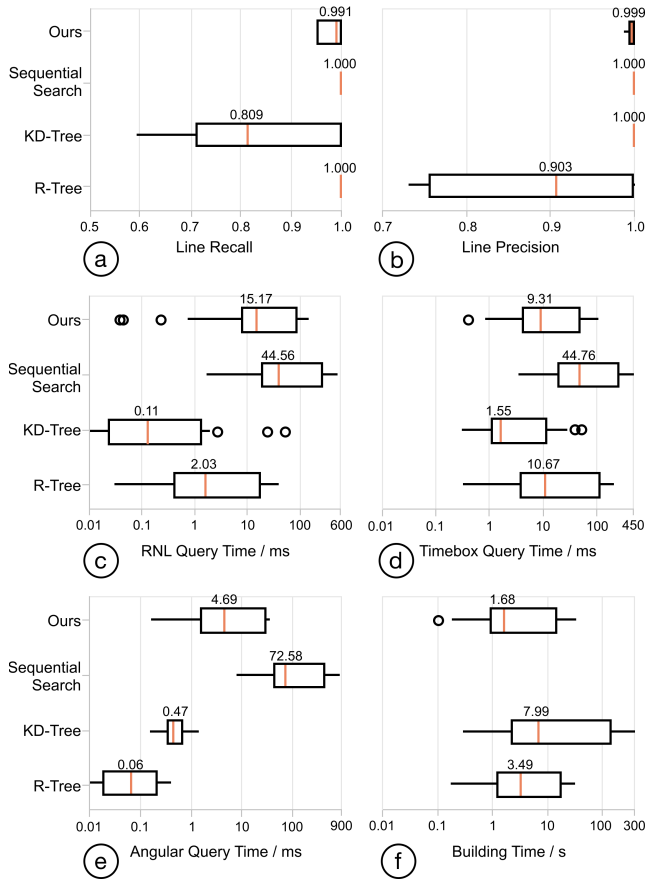


Fig. 8. Boxplots on the line recall (a), line precision (b), query time (c,d,e), and tree-building time (f) for the RNL search (r=0.02), timebox query, and angular query on all datasets using the four query methods.

M randomly-generated query points for RNL, timeboxes, and angular ranges for each data set, respectively. Here, we set M as 1,000 and tested different r values for the RNL search, i.e., $\{0.01, 0.02, \cdots, 0.1\}$ relative to the normalized screen space ranged $[0, 1]$.

Methods. We compared our method with three methods: KD-Tree, R-Tree, and sequential search. Among them, sequential search was shown to perform better than range trees and hash tables for timebox search [19], so we chose to implement it. To adapt point-based KD-Tree [5] and R-Tree [4] for nearest line (NL) search, we pre-stored the line index of each point in the leaf node and implemented them based on the state-of-the-art JavaScript implementations [12, 32].

Measures. We quantitatively measure the quality and performance of line query in three aspects: tree-building time, query processing time, and query accuracy. For query accuracy, we follow Lu et al. [31] by using two numerical measures *line recall* and *line precision*. Suppose the set of retrieved lines is $\hat{\Omega}$ and the ground truth is Ω, *line recall* measures the proportion of lines retrieved in the ground truth Ω:

$$\textit{line recall } (LR) = \frac{|\hat{\Omega} \cap \Omega|}{|\Omega|},$$

which ranges from 0 to 1. A larger value indicates that more ground-truth lines are found. Likewise, *line precision* measures the proportion of correctly-retrieved lines in $\hat{\Omega}$:

$$\textit{line precision } (LP) = \frac{|\hat{\Omega} \cap \Omega|}{|\hat{\Omega}|},$$

which also ranges from 0 to 1. A larger value indicates that more retrieved lines in $\hat{\Omega}$ are true nearest lines.

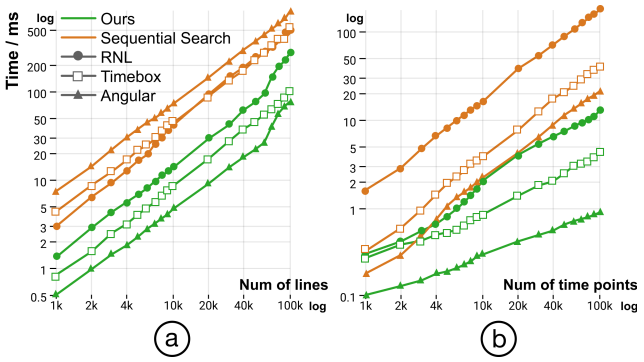


Fig. 9. Comparing the query time of our method and sequential search in performing the RNL, timebox query, and angular query with different parameters: (a) number of time series and (b) number of time sample points in each line.

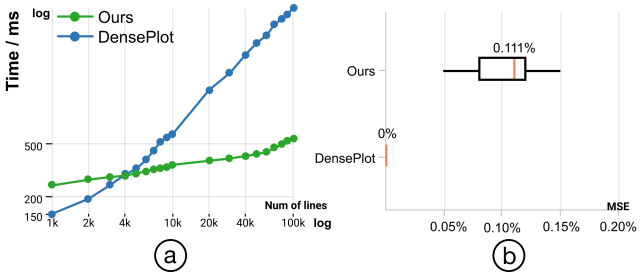


Fig. 10. Comparing the rendering time (a) and MSE (b) of our method and DensePlot in rendering density fields with different parameters.

Result. To quantitatively evaluate the four methods, we computed the averaged *LR*, *LP*, and *query time* with M query operations issued by each of the four query methods on each dataset. The boxplots shown in Figs. 8a-e summarize the three measures for the four methods.

The results show that our method achieves mean accuracies of 99.1% and 99.9% for line recall and line precision, respectively, whereas the mean line recall of KD-tree is only 80.9% and the mean line precision of R-tree is only 90.3%. The major reason for the poor accuracy of KD-tree is that it is built on sample points without considering the continuous line information. R-tree has a similar issue that its stored bounding boxes do not have accurate line ranges, thus including more unqualified lines in the query result. In contrast, sequential search always returns the ground truth, but it is 3 times slower for RNL query, 5 times slower for timebox query, and 15 times slower for angular query than our method. Although our method is slower than KD-tree and R-tree in the three query strategies, it is sufficiently fast for supporting effective exploration [27], where the mean query time of the three query strategies are all less than 16ms. Also, our method takes less time to build the tree than KD-tree and R-tree, as shown in Fig. 8(f). Considering all these results, we conclude that our method is the best in achieving high accuracy with less computation cost in offline pre-processing and online query.

As shown in Fig. 9, the query time of our method and sequential search both increases with the number of lines and also the number of time points, no matter which query strategy is used. Compared to sequential search, our method is around 10x faster for RNL search, 10x for timebox query, and 20x for angular query, when the number of lines or the number of time points is 100 K. All of our queries finished within 300ms, which suffices efficient exploration of large time-series data.

Furthermore, comparing the query time of our three query strategies reveals that angular query ranks the first (fastest), followed by timebox query, and RNL query is the last. This is reasonable, since our RNL query involves computing the distance from the query point to lines and timebox query needs to check the intersection between lines and the query box, while angular query does not need these operations.

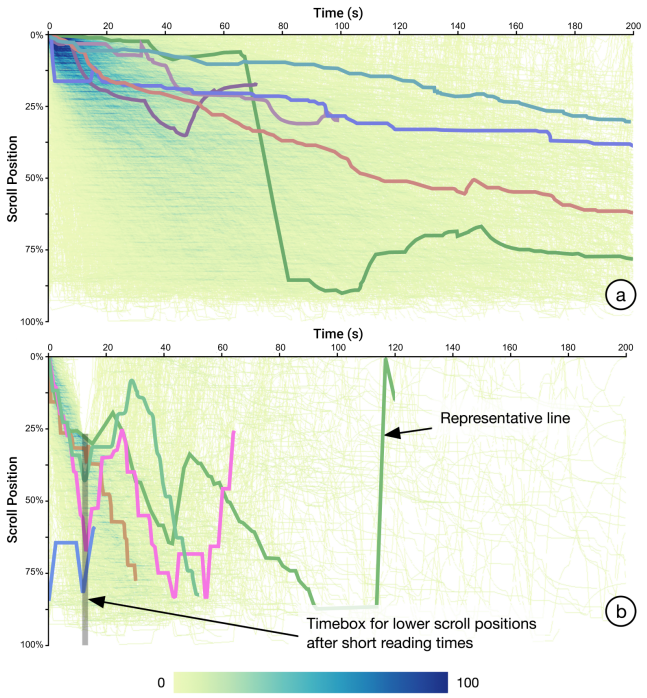


Fig. 11. (a) Density visualization and representative lines of reading behavior of all readers in an interactive article on the web. (b) Putting a timebox for lower scroll position early in the overall time range, we can reveal readers who scrolled down very early and quickly produce a density field for the reading behavior of these readers. Note that both density fields (a & b) are encoded in the same color scale.

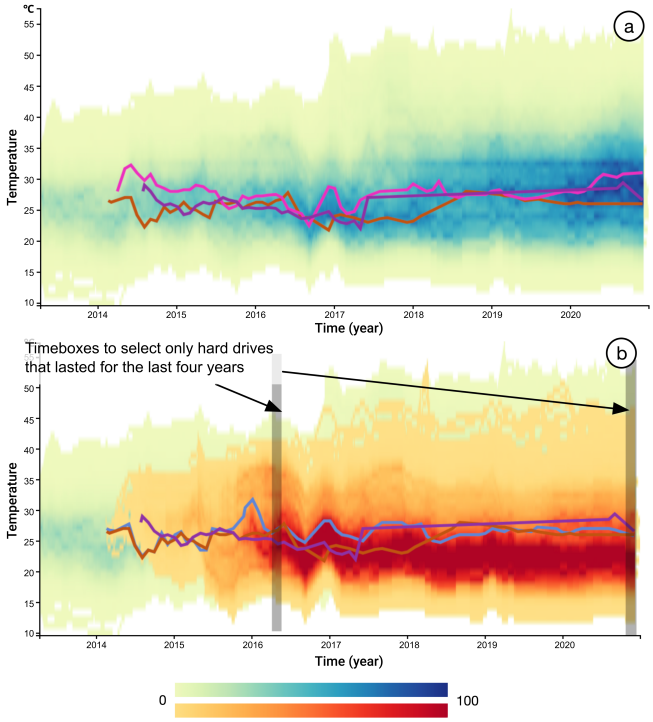


Fig. 12. (a) Density field visualization and representative lines for the temperature of 92 K hard drives over six years. (b) Overlaid density and representative lines for long-lived hard drives selected by the gray boxes. The overall density field and the selected ones are encoded by the yellow-green-blue and yellow-orange-red color scales, respectively.

5.2 Density Rendering

To evaluate the rendering performance and quality of our density field, we select DensePlot [30], the state-of-the-art CPU implementation of DenseLine [37], as the rendering groundtruth. As the tree construction is necessary for supporting interaction, we only measure the rendering time and mean squared error (MSE) between our results and those from DensePlot. The display size of all density fields is 1200×600 pixels.

Results. Fig. 10a summarizes the rendering time of the two methods. We can see that our method is only a bit slower than DensePlot when the number of time series is less than 5 K but then it is much faster. Moreover, its rendering time increases very slowly, where the maximal rendering time is around 500ms. The reason is that our method is an image-order approach and its complexity is regardless of the number of time series or time points. Nonetheless, our density field is highly accurate, where the MSE error is less than 0.15% as shown in Fig. 10(b). Thus, we conclude that our method can produce almost accurate and high-quality density fields much faster.

6 Demonstration of KD-Box

In this section, we present three demonstrations on real-world data to show the interactions in KD-Box. We explore these demonstrations through hypothetical user scenarios to show how one may use our tool. We study large time-series data from online reading patterns (Sect. 6.1), hard-drive statistics (Sect. 6.2), and temperature data (Sect. 6.3).

While KD-Box only supports a subset of interactions in previous systems such as TimeSearcher [18], it is able to handle larger datasets. Therefore, we focus the demonstration cases on the data aspects that are enabled by KD-Box's scalability.

6.1 Discover Reading Patterns of *Beat Basics*

This demonstration case explores the reading behaviors of *Beat Basics*[2], an interactive article that explains the differences between 3/4 and 6/8 beats. Conlen et al. [10] characterized the interaction patterns in this article using visualizations of recorded activities. They published the data they collected to conduct their analysis. The dataset has 4430 time series, each representing the reading behavior of a user in a session and recording their scroll position from opening to closing the article. Conlen et al. used a density visualization overlaying lines in low opacity. In this naive representation, each line has the same opacity. Therefore, if it is set too low, low-density areas disappear. If it is set too high, the density tops out too quickly. Moreover, this representation leads to visual artifacts of steep lines as we discussed in Sect. 3.

We load the dataset into KD-Box and generate the density visualization shown in Fig. 11. The horizontal axis denotes the reading time and the vertical axis denotes the scroll position in the article. The reading time ranges from 0 to 3.3 minutes and the scroll position ranges from 0% (top of the article) to 100% (bottom of the article). The density visualization (Fig. 11a) shows a high-density pattern on top-left, which makes sense, as we expect readers to start reading from the top.

Interestingly, some high-density areas appear at lower scroll positions early in the reading. Investigating the matter further using a brush query (Fig. 11b) shows that some readers simply scroll down immediately and some start from the middle of the article. The brush query only selects a few hundred lines, so we can render all selected lines (and hide the full density using the control panel shown in Fig. 7d). As Fig. 11b shows, there are too many readers with such behavior ($\sim$1K), so we may switch to a density representation. The representative lines can reveal some other common reading patterns, e.g., a user who scrolls down before going back up (also identified by Conlen et al. [10].

6.2 Sensor Data from Hard Drives

Sensors are one of the most common sources of time-series data, providing measurements on environments and electronic systems over time. As an example of sensors in electronic systems, in the second demonstration case, we explore the statistics of more than 92 K hard drives. Backblaze, a cloud storage provider, publishes detailed statistics about the hard drives in their data centers every quarter of the year[3].

[2] idyll-lang.org/gallery/beat-basics

[3] www.backblaze.com/b2/hard-drive-test-data.html

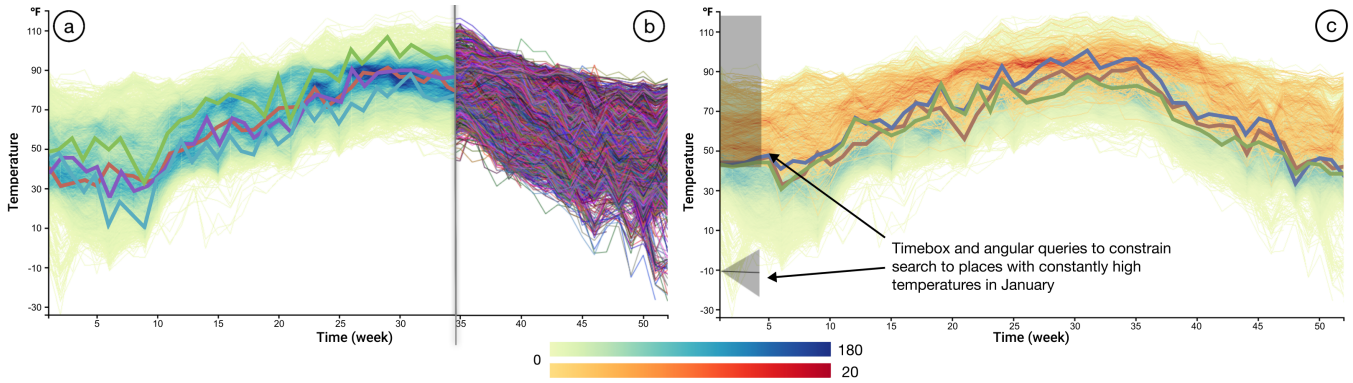


Fig. 13. (a) A density visualization of a weather dataset in the U.S. obtained from the ACIS web services. (b) The raw lines in the dataset when all are visualized together. (c) Density visualization for places with consistently high temperatures in January selected by a Timebox and an angular query; see the gray boxes overlaid on the left. The overall density field and the selected ones are encoded by the yellow-green-blue and yellow-orange-red color scales, respectively.

Fig. 12a shows the time series of the hard drive temperature (SMART 194) for a random sample of 92 K time series out of over 160 K in the full dataset. Note that our method can handle larger amount of data but it is currently limited by the browser (see Sect. 7). Yet, the visualization in Fig. 12a already displays an aggregation of ~2.4 M individual records, from which we can see that no hard drive goes above 55°C and most stay between 20°C and 30°C. A non-density version of the same data would be a highly-cluttered visualization.

By drawing a brush at the beginning of the recording in 2014, we can see that none of the first-generation hard drives are still in operation. We may want to see hard drives that have been in operation for the last four years. Setting a pair of timeboxes four years apart can filter the data and locate 14 K time series (Fig. 12b). The density of these selected lines shows that most of these long-lived hard drives stayed within a narrower temperature band than the rest. Also, the majority of these hard drives stayed at lower temperatures. This insight could provide an indicator for better maintenance of the hard drives.

6.3 Patterns in Large-Scale Temperature Data

In this case, we explore an environmental sensor dataset obtained from the Applied Climate Information System (ACIS) Web Services [4], which records the daily temperature of 6187 public weather stations in the United States in 2019.

The density visualization without filters in Fig. 13a shows a clear seasonal pattern where average temperatures are higher in the summer. This pattern is not too surprising, as the United States is in the northern hemisphere. While this trend is also visible when simply rendering the lines (Fig. 13b), the density visualization reveals that the majority of weather stations have higher variability than how the envelope implies. The representative lines also show this broader trend in the data.

We may be interested in the temperature trends of places with consistently high temperatures in the first few months of the year. To do so, we can create an angular query to look for time series with small steepness (slope) in the first month of the year and a brush query at high temperatures. The density visualization (Fig. 13c) shows that the temperature of these places is generally higher and stays rather consistently warm throughout the year, which is also confirmed by the representative line overlaid to the selected density field. During the summer months, the temperatures for these places are similar to the majority of all temperatures in the US. The *Detail View* (Fig. 7c) shows that the representative lines in this selection are for places in the south or near the ocean (i.e., Florida, California coast, and Hawaii). Being close to the ocean receive huge heat capacity, leading to a more constant temperature.

Similarly, we might want to see the temperature curves for places with consistent (without restricting the range) temperatures in other months. We can move the angular filter along the time axis to update the query. Since KD-Box gives a fast query response within 40 ms, we can see how temperature develops for the selected places and notice that places with lower temperatures generally have higher temperature variability. The interactions are fast because we can update the query results incrementally (Sect. 4.3).

[4] www.rcc-acis.org/docs_webservices.html

7 Discussion and Future Work

Hochheiser et al. [18] showed that timebox queries are a valuable tool for exploratory analysis of time-series data. Their system TimeSearcher [18, 19] offers interactive response times for small time-series datasets. However, there are two issues related to scalability for large time-series data. First, overdraw and visual clutter can make the data overview less meaningful. Constraints can help to filter time series in the display but doing so only addresses part of the problem, as many interesting queries may not be easily selectable. In this work, we address the issues through an efficient density visualization that shows all time series and the time series that match the query constraints. Second, Hochheiser et al. [18] note that 75% of query time is spent on query evaluation as opposed to rendering. The line-segment-based KD-tree we incorporated into KD-Box enables fast queries over larger data while still being highly responsive.

There are three major limitations related to our current implementation of KD-Box. First, due to the memory limit imposed by browsers, we can at most support efficient exploration of 100 K time series. To address this issue, we plan to extend our line-segment-based KD-tree to work in a progressive way [46], in which data is streamed into memory in chunks. Second, our representative line selection is based on a heuristic method that might not capture all interesting trends for complex data. We plan to extend a fast density-based clustering method, mean-shift clustering [9, 14] into curve-based density fields for identifying representative lines. Last, our KD-Box only focuses on the core functionality of timebox and angular queries; while future versions of our system should support the full gamut of features in TimeSearcher [19], including the leaders and laggards, query inversion, and variable-time boxes.

There are two future work possibilities. One is to extend our system to support interactive analysis of streaming time-series data in various domains such as finance, mobile, and IoT. Second, density-based representations have been used for visualizing various types of data [20] such as trajectory, graph, and high-dimensional data, whereas most of them suffer from similar challenges as time-series data. In the future, we will extend our line-segment-based KD-tree to explore these different data for some other analysis tasks.

Acknowledgments

This work is supported by the grants of the NSFC (61772315, 61861136012), the Open Project Program of State Key Laboratory of Virtual Reality Technology and Systems, Beihang University (No. VRLAB2020C08), and the CAS grant (GJHZ1862).

References

[1] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah. Time-series clustering–a decade review. *Information Systems*, 53:16–38, Oct. 2015. doi: 10.1016/j.is.2015.04.007

[2] A. O. Artero, M. C. Ferreira De Oliveira, and H. Levkowitz. Uncovering clusters in crowded parallel coordinates visualizations. In *IEEE Symposium on Information Visualization*, pp. 81–88, Oct. 2004. doi: 10.1109/INFVIS.2004.68

[3] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. *Proceedings of the 2016 International Conference on Management of Data*, pp. 1363–1375, June 2016. doi: 10.1145/2882903.2882919

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pp. 322–331, 1990. doi: doi/10.1145/93597.98741

[5] J. L. Bentley and J. H. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, Dec. 1979. doi: 10.1145/356789.356797

[6] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf. Parallel edge splatting for scalable dynamic graph visualization. *IEEE Trans. Vis. & Comp. Graphics*, 17:2344–2353, Nov. 2011. doi: 10.1109/TVCG.2011.226

[7] S.-M. Chan, L. Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. In *IEEE Symposium on Visual Analytics Science and Technology*, pp. 59–66, Oct. 2008. doi: 10.1109/vast.2008.4677357

[8] S. Chaudhuri, B. Ding, and S. Kandula. Approximate Query Processing: No Silver Bullet. In *ACM International Conference on Management of Data*, SIGMOD '17, pp. 511–519. Association for Computing Machinery, May 2017. doi: 10.1145/3035918.3056097

[9] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, 2002.

[10] M. Conlen, A. Kale, and J. Heer. Capture & Analysis of Active Reading Behaviors for Interactive Articles on the Web. *Computer Graphics Forum*, 38:687–698, June 2019. doi: 10.1111/cgf.13720

[11] O. Daae Lampe and H. Hauser. Curve density estimates. *Computer Graphics Forum*, 30:633–642, June 2011. doi: 10.1111/j.1467-8659.2011.01912.x

[12] Eronana. RBush3D JavaScript Library. `https://github.com/Eronana/rbush-3d`. [Online; accessed 6-Feb.-2021].

[13] D. Feng, L. Kwock, Y. Lee, and R. Taylor. Matching visual saliency to confidence in plots of uncertain data. *IEEE Trans. Vis. & Comp. Graphics*, 16:980–989, Oct. 2010. doi: 10.1109/tvcg.2010.176

[14] D. Freedman and P. Kisilev. Fast mean shift by compact density representation. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1818–1825. IEEE, 2009.

[15] E. Ghysels, D. R. Osborn, and T. J. Sargent. *The econometric analysis of seasonal time series*. Cambridge University Press, 2001. doi: 10.1017/cbo9781139164009

[16] J. Heer, N. Kong, and M. Agrawala. Sizing the horizon: the effects of chart size and layering on the graphical perception of time series visualizations. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1303–1312, Apr. 2009. doi: 10.1145/1518701.1518897

[17] J. Heinrich, S. Bachthaler, and D. Weiskopf. Progressive splatting of continuous scatterplots and parallel coordinates. *Computer Graphics Forum*, 30:653–662, June 2011. doi: 10.1111/j.1467-8659.2011.01914.x

[18] H. Hochheiser and B. Shneiderman. *Interactive Graphical Querying of Time Series and Linear Sequence Data Sets*. PhD thesis, USA, 2003. AAI3094494.

[19] H. Hochheiser and B. Shneiderman. Dynamic Query Tools for Time Series Data Sets: Timebox Widgets for Interactive Exploration. *Information Visualization*, 3:1–18, Mar. 2004. doi: 10.1057/palgrave.ivs.9500061

[20] C. Hurter. Image-based visualization: Interactive multidimensional data exploration. *Synthesis Lectures on Visualization*, Dec. 2015. doi: 10.2200/s00688ed1v01y201512vis006

[21] W. Javed, B. McDonnel, and N. Elmqvist. Graphical perception of multiple time series. *IEEE Trans. Vis. & Comp. Graphics*, 16:927–934, Oct. 2010. doi: 10.1109/TVCG.2010.162

[22] S. Kandel, R. Parikh, A. Paepcke, J. Hellerstein, and J. Heer. Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment. In *Proc. of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pp. 547–554, May 2012. doi: 10.1145/2254556.2254659

[23] O. D. Lampe and H. Hauser. Interactive visualization of streaming data with Kernel Density Estimation. In *IEEE Pacific Visualization Symposium*, pp. 171–178, Mar. 2011. doi: 10.1109/pacificvis.2011.5742387

[24] A. Lhuillier, C. Hurter, and A. Telea. State of the art in edge and trail bundling techniques. *Computer Graphics Forum*, 36:619–645, June 2017. doi: 10.1111/cgf.13213

[25] J. Lin, E. Keogh, S. Lonardi, J. P. Lankford, and D. M. Nystrom. Visually mining and monitoring massive time series. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 460–469, Aug. 2004. doi: 10.1145/1014052.1014104

[26] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Vis. & Comp. Graphics*, 19:2456–2465, Oct. 2013. doi: 10.1109/TVCG.2013.179

[27] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE Trans. Vis. & Comp. Graphics*, 20:2122–2131, Nov. 2014. doi: 10.1109/tvcg.2014.2346452

[28] Z. Liu, B. Jiang, and J. Heer. ImMens: Real-time visual querying of big data. *Computer Graphics Forum*, 32:421–430, June 2013. doi: 10.1111/cgf.12129

[29] R. Lopez-Hernandez, D. Guilmaine, M. J. McGuffin, and L. Barford. A layer-oriented interface for visualizing time-series data from oscilloscopes. In *IEEE Pacific Visualization Symposium*, pp. 41–48, Mar. 2010. doi: 10.1109/PACIFICVIS.2010.5429607

[30] T. Ltd. Time Series Density Plot. `https://observablehq.com/@twitter/time-series-density-plot`. [Online; accessed 5-Mar.-2021].

[31] Y. Lu, L. Cheng, T. Isenberg, C.-W. Fu, G. Chen, H. Liu, O. Deussen, and Y. Wang. Curve Complexity Heuristic KD-trees for Neighborhood-based Exploration of 3D Curves. *Computer Graphics Forum*, 40(2), May 2021.

[32] M. Lysenko. Static KDTree JavaScript Library. `https://github.com/mikolalysenko/static-kdtree`. [Online; accessed 6-Feb.-2021].

[33] J. Matejka, F. Anderson, and G. Fitzmaurice. Dynamic opacity optimization for scatter plots. In *ACM Conference on Human Factors in Computing Systems*, CHI '15, pp. 2707–2710. Association for Computing Machinery, Apr. 2015. doi: 10.1145/2702123.2702585

[34] A. Mayorga and M. Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE Trans. Vis. & Comp. Graphics*, 19:1526–1538, Mar. 2013. doi: 10.1109/TVCG.2013.65

[35] F. Miranda, M. Lage, H. Doraiswamy, C. Mydlarz, J. Salamon, Y. Lockerman, J. Freire, and C. T. Silva. Time lattice: A data structure for the interactive visual analysis of large time series. *Computer Graphics Forum*, 37:23–35, June 2018. doi: 10.1111/cgf.13398

[36] B. Moberts, A. Vilanova, and J. J. van Wijk. Evaluation of fiber clustering methods for diffusion tensor imaging. In *IEEE Visualization*, pp. 65–72. Los Alamitos, Oct. 2005. doi: 10.1109/VISUAL.2005.1532779

[37] D. Moritz and D. Fisher. Visualizing a Million Time Series with the Density Line Chart. *arXiv*, 2018.

[38] D. Moritz, D. Fisher, B. Ding, and C. Wang. Trust, but Verify: Optimistic visualizations of approximate queries for exploring big data. In *ACM Conference on Human Factors in Computing Systems*, CHI '17, pp. 2901–2915. Association for Computing Machinery, May 2017. doi: 10.1145/3025453.3025456

[39] D. Moritz, B. Howe, and J. Heer. Falcon: Balancing interactive latency and resolution sensitivity for scalable linked visualizations. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–11, May 2019. doi: 10.1145/3290605.3300924

[40] P. Muigg, J. Kehrer, S. Oeltze, H. Piringer, H. Doleisch, B. Preim, and H. Hauser. A four-level focus+context approach to interactive visual analysis of temporal features in large scientific data. *Computer Graphics Forum*, 27:775–782, May 2008. doi: 10.1111/j.1467-8659.2008.01207.x

[41] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 36:2227–2240, May 2014. doi: 10.1109/tpami.2014.2321376

[42] M. Novotny and H. Hauser. Outlier-preserving focus+context visualization in parallel coordinates. *IEEE Trans. Vis. & Comp. Graphics*, 12:893–900, Nov. 2006. doi: 10.1109/tvcg.2006.170

[43] R. Scheepens, N. Willems, H. van De Wetering, G. Andrienko, N. Andrienko, and J. J. van Wijk. Composite density maps for multivariate trajectories. *IEEE Trans. Vis. & Comp. Graphics*, 17:2518–2527, Nov. 2011. doi: 10.1109/TVCG.2011.181

[44] B. J. Swihart, B. Caffo, B. D. James, M. Strand, B. S. Schwartz, and

N. M. Punjabi. Lasagna plots: A saucy alternative to spaghetti plots. *Epidemiology*, 21:621–625, Sept. 2010. doi: 10.1097/EDE.0b013e3181e5b06a

[45] A. Telea and O. Ersoy. Image-based edge bundles: Simplified visualization of large graphs. *Computer Graphics Forum*, 29:843–852, June 2010. doi: 10.1111/j.1467-8659.2009.01680.x

[46] C. Turkay, N. Pezzotti, C. Binnig, H. Strobelt, B. Hammer, D. A. Keim, J.-D. Fekete, T. Palpanas, Y. Wang, and F. Rusu. Progressive data science: Potential and challenges. *arXiv*, 2018.

[47] M. van Kreveld, O. Schwarzkopf, M. de Berg, and M. Overmars. *Computational geometry algorithms and applications*. Springer, 2000. doi: 10.1007/978-3-540-77974-2

[48] R. van Liere and W. De Leeuw. GraphSplatting: Visualizing graphs as continuous fields. *IEEE Trans. Vis. & Comp. Graphics*, 9:206–212, Apr. 2003. doi: 10.1109/tvcg.2003.1196007

[49] J. J. van Wijk and E. R. van Selow. Cluster and calendar based visualization of time series data. In *IEEE Symposium on Information Visualization*, pp. 4–9, Oct. 1999. doi: 10.1109/infvis.1999.801851

[50] M. Visvalingam and J. D. Whyatt. The Douglas-Peucker Algorithm for Line Simplification: Re-evaluation through Visualization. *Computer Graphics Forum*, 9:213–225, Sept. 1990. doi: 10.1111/j.1467-8659.1990.tb00398.x

[51] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in *O(N log N)*. In *IEEE Symposium on Interactive Ray Tracing*, pp. 61–69. Los Alamitos, Sept. 2006. doi: 10.1109/RT.2006.280216

[52] R. J. Watt and D. P. Andrews. Contour curvature analysis: hyperacuities in the discrimination of detailed shape. *Vision research*, 22:449–460, 1982. doi: 10.1016/0042-6989(82)90193-6

[53] M. Wattenberg. Sketching a graph to query a time-series database. In M. M. Tremaine, ed., *Extended Abstracts on Human Factors in Computing Systems*, CHI EA '01, pp. 381–382. Association for Computing Machinery, Mar. 2001. doi: 10.1145/634067.634292

[54] M. Weber and W. Müller. Visualizing Time-Series on Spirals. In *IEEE Symposium on Information Visualization*, pp. 7–13, Oct. 2001. doi: 10.1109/INFVIS.2001.963273

[55] H. Wickham. Bin-summarise-smooth : A framework for visualising large data. Technical report, had.co.nz, 2013.

[56] H. Yu, C. Wang, C.-K. Shene, and J. H. Chen. Hierarchical streamline bundles. *IEEE Trans. Vis. & Comp. Graphics*, 18:1353–1367, Sept. 2012. doi: 10.1109/TVCG.2011.155

[57] J. Zhao, F. Chevalier, E. Pietriga, and R. Balakrishnan. Exploratory analysis of time-series with chronolenses. *IEEE Trans. Vis. & Comp. Graphics*, 17:2422–2431, Nov. 2011. doi: 10.1109/TVCG.2011.195

[58] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *Proc. of the ACM in Computer Graphics and Interactive Techniques*, pp. 371–378, Aug. 2001. doi: 10.1145/383259.383300