Interactive Visualization of Streaming Data with Kernel Density Estimation

Ove Daae Lampe*
University of Bergen, Norway and
Chr. Michelsen Research AS

Helwig Hauser†
University of Bergen, Norway
www.ii.UiB.no/vis

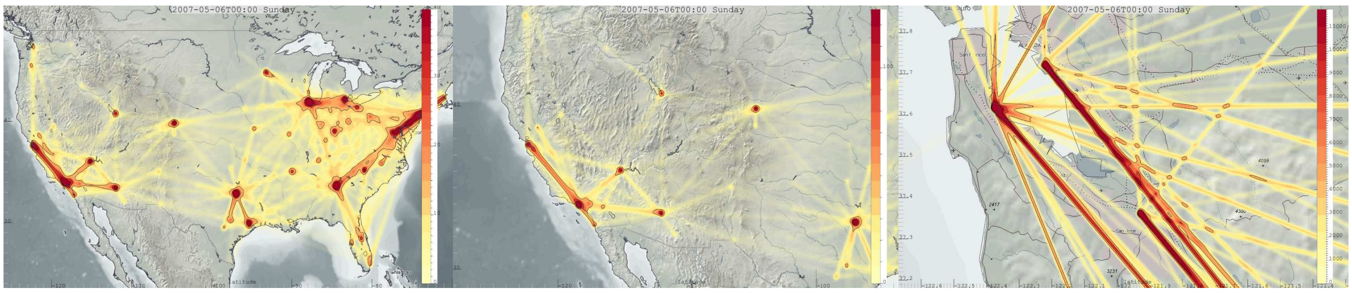


Figure 1: Interactive zooming towards SF Bay, where at first all the traffic from the Bay Area is aggregated, to a view where we can separate traffic from the three major airports, and even the distribution of traffic in each airports' cardinal direction. This interaction is enabled by automatically updating the bandwidth of the KDE when the viewport changes.

Abstract

In this paper, we discuss the extension and integration of the statistical concept of Kernel Density Estimation (KDE) in a scatterplot-like visualization for dynamic data at interactive rates. We present a line kernel for representing streaming data, we discuss how the concept of KDE can be adapted to enable a continuous representation of the distribution of a dependent variable of a 2D domain. We propose to automatically adapt the kernel bandwith of KDE to the viewport settings, in an interactive visualization environment that allows zooming and panning. We also present a GPU-based realization of KDE that leads to interactive frame rates, even for comparably large datasets. Finally, we demonstrate the usefulness of our approach in the context of three application scenarios – one studying streaming ship traffic data, another one from the oil & gas domain, where process data from the operation of an oil rig is streaming in to an on-shore operational center, and a third one studying commercial air traffic in the US spanning 1987 to 2008.

Index Terms: I.3.3 [Computing Methodologies]: Computer Graphics—Picture/Image Generation G.3 [Mathematics of Computing]: Probability and Statistics—Time series analysis;

1 Introduction

The scatterplot is one of the most prominent success stories in statistics and visualization. Scientists and practitioners have used scatterplots for more than 100 years to study the distributional characteristics of multivariate data with respect to two data attributes or dimensions [28]. However when datasets are large, scatterplots are challenged by overdraw and cluttering. There are approaches to improve this situation, e.g., by employing semi-transparency during rendering or by subsetting prior to the visualization [8]. With such approaches the number of data items that can be effectively shown in a scatterplot can be pushed by one or two orders of magnitude. Beyond a certain point, however, at least when there are many more data items to be shown than there are pixels in the scatterplot, the item-based approach is collapsing [20]. It has been shown that switching to a frequency-based visualization metaphor is a useful solution in such a case [20, 12, 19, 32]. Such frequency based visualizations are e.g., histograms or density estimations.

While histograms are straightforward to implement and interpret, the parametrization of data introduce a significant variance in appearance, e.g.,the discretization of data into buckets/bins, may cause aliasing effects. Corresponding interpretations depend on bin count and interval range along the axis. [27]. Fig. 2 illustrates one example of such a major change by showing two histograms of the same data – one computed with 9 bins and the other one with 10. To achieve a more truthful assessment of distributional data characteristics, Kernel Density Estimation (KDE) [25] is commonly used in statistics. Assuming that the distribution of the data items adheres to a certain probability density function (PDF), KDE allows estimating this PDF from the samples. The result is a function that represents the distribution of the data items in terms of their density in the data space. Years of research has made KDE into an important tool for statistical data analysis [30]. One of the major advantages of KDE is that it directly evaluates the data, without imposing a model onto it, which, consequently has the advantage that the data speak for themselves. (as Silverman says [25]).

Our goal of using interactive visual analysis on large amounts of dynamic and streaming data, demanded a real-time KDE implementation. Fast update rates for KDE is needed to highlight the coherency of the temporal correlations. To support continuously updates of streaming data, rules out techniques relying on pre-processing.

With this paper we follow up on this opportunity in utilizing KDE for visualization, and in the following: We propose a line kernel for the KDE-based visualization of streaming data, and an automatic adaptation of the bandwidth used for KDE, according to the zoom level of the visualization. We present a KDE-based interactive visualization, with real-time performance enabled by the GPU. We demonstrate how to visualize the distributional characteristics of another data attribute (instead of sample frequency) by adapting KDE accordingly. The usefulness of this approach is showed in three demonstrations, one on surveillance data from the maritime traffic domain, one on real-time drilling data from the petroleum

*e-mail: odl@cmr.no
†e-mail:Helwig.Hauser@UiB.no

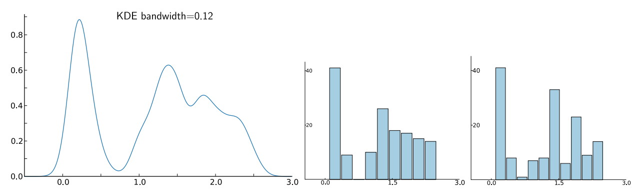


Figure 2: A kernel density estimation of *petal width* in the Iris dataset [13] and two corresponding histograms, one with 9 bins and the other with 10 bins.

industry, and one on air traffic data.

2 Related Work

Both scatterplots and histograms have become a commodity in data visualization, even for the general mass market. Ericson (from the New York Times) even said that the scatterplot is the most complex visualization technique that the general public can appreciate [9].

An interesting subset of previous work, which also is of special relevance for our work here, comes in the form of examples for this methodological change from an item-based visualization approach (as the classical scatterplot) to a frequency-based approach (such as the histogram). Fisher, for example, visualizes aggregated numbers of downloads with the hotmap approach [12]. Novotný and Hauser demonstrate how the transition to a frequency-based methodology can enable the visualization of very large datasets in parallel coordinates [20] and Muigg et al. show how this transition enables the visualization of hundreds of thousands of function graph curves [19]. Artero et al., who also use a frequency-based approach to visualizing large datasets with parallel coordinates [2], refer to kernel density estimation as an approach to compute the density values (but eventually revert to a box function as their reconstruction kernel). Kidwell et al. refer to KDE for reconstructing a smooth and space-filling heatmap visualization of a small number of data items [17]. For a more thorough discussion on the use of KDE in visualization we refer to the work by Scott [23]. Whittaker and Scott presented the use of the Average Shifted Histogram (ASH) i.e., an alternate and very efficient density estimation, that approximates KDE, for the use in a geographical context [31].

Very interesting related work is an approach called *continuous scatterplots* by Bachthaler et al. [4]. Assuming data that are continuous with respect to a spatial reference domain – such as the distribution of physical or chemical quantities over a 2D or 3D reference space as acquired through measurements or numerical simulation – a mapping is computed that represents the data in the form of an *m*-dimensional continuous histogram. KDE-based visualization, as discussed in this paper, is not a mapping from a continuous spatial domain, but rather a mapping of sparesly sampled data, which is mapped into a spatial domain. Similar work on the reconstruction of uniformly sampled data is done by Crawfis and Max [7], where they investigate the use of texture splats with normal distributed values, as means to reconstruct the continuous data field in 3D. Similarly, as in the work by Bachthaler et al. [4], a requirement for this technique is the continuous spatial domain. The work presented here also supports these continuous domains and also extends to support streaming time-dependent data, attribute reconstruction, and non-uniformly sampled data.

Jang et al. investigated the representation of non-uniform, tetrahedral volumetric datasets, by weighted Radial Basis Functions (RBF) [15, 6]. They introduce an algorithm on how to effectively render such 3D RBFs by applying a slice based technique. In this work, we investigate the use of a broader category of kernels than those available as RBFs, namely the product kernel and our extended line kernel. We furthermore show that when applying kernels to dimensions with different units or of different scale, RBFs are impractical, e.g., when plotting meters over tonnes.

Andrienko and Andrienko defined a generalized method on how to create abstractions from geospatial movement data [1]. This abstraction technique generates, from unstructured and unrestricted movement data, potential nodes, where traffic can be aggregated, similar to a node-link diagram. While, theirs and our technique both share the same type of source data, the end result portray two different images, with similar, but still, different usages. The result by Andrienko and Andrienko [1] show the total volume of traffic, and how this volume is distributed, i.e., by counting all passing vessels. With our technique we display, where the traffic spend its time, e.g., if a car stops, it will still contribute a kernel at that position. The differences in these two techniques, as well as other techniques that employ node-link diagrams for aggregation, are comparable to that of the histogram on one side, and KDE on the other side. While the aggregation techniques, similar to the histogram, provides a high level of abstraction, and clear benefits in terms of quantitative readouts, they will potentially suffer aliasing effects and hide underlying details which only a continuous representation can show.

3 Kernel Density Estimation

In the following, we first briefly define kernel density estimation (KDE), before we discuss KDE-based visualization.

KDE is a well-proven approach to achieve a non-parametric estimation of data density that has been introduced to the field of statistics by Rosenblatt and Parzen about 50 years ago [22, 21]. Given a set of n (1D) data samples x_i, $1 \leq i \leq n$, the kernel density estimator $\widehat{f}_h(x)$ is computed as

$$\widehat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right) = \frac{1}{n}\sum_{i=1}^{n} K_h(x-x_i), \qquad (1)$$

based on a kernel function K and a bandwidth parameter h. Often symmetric kernels are considered as K, with $K(x) \geq 0$ and $\int K(x)\,\mathrm{d}x = 1$, also often centered around 0. In such a case also $\widehat{f}_h(x)$ is also nonnegative and integrates to 1. This enables interpretation of $\widehat{f}_h(x)$ as a density function that approximates the PDF $f(x)$ of the data items x_i from which it has been constructed. The KDE of data attribute *petal width* in the Iris dataset, as shown in Fig. 2 on the left, is considered to be a more truthful visualization of the distribution of the considered data values, than a histogram.

A large variety of kernels has been studied, including the uniform kernel (based on the normal, Gaussian distribution), the triangle kernel, the Epanechnikov kernel [29], and many others. In many cases, however, the normal kernel,

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}, \qquad (2)$$

is used in KDE. Even though it has been concluded [30] that variations in the choice of K are less important than variations of h, there still are strong arguments for choosing the normal kernel [18], e.g., when calculating the modes of $\widehat{f}_h$.

Bandwidth h is a parameter which influences the smoothness of the density reconstruction. Fig. 3 shows four results from a 2D KDE with increasing values of h. Several authors have worked [29, 30] on (automatically) optimizing the choice of bandwidth h, e.g., Silverman describes the normal scale rule [25] to derive an optimal value for h as

$$h := 1.06 \cdot \sigma \cdot n^{-\frac{1}{5}}. \qquad (3)$$

This rule is leading to an optimal estimation if the data is normal distributed. It will lead to an over-smoothed result, however, if not [30]. There are also several other approaches to globally optimize h (several covered by Wand and Jones [30]), and in Sec. 6

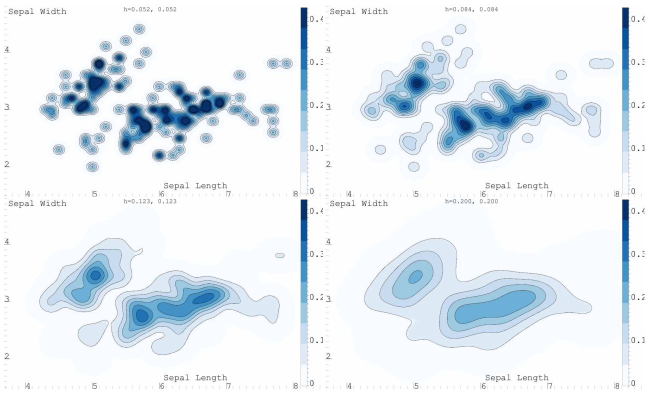


Figure 3: 2D KDE for the Iris dataset [13] with increasing bandwidth.

we briefly discuss why these are not sufficient for interactive visualization, and propose a new approach.

Altogether, it is generally agreed that KDE is a very appealing tool to investigate the distributional characteristics of data. Gray and Moore write *"In general, density estimation provides a classical basis across statistics for virtually any kind of data analysis, including clustering, classification, regression, time series analysis, active learning, ... "* [14].

Up to here, we have discussed KDE in the one-dimensional case. It is straightforward, however, to extend KDE to multiple dimensions [23]:

$$\widehat{f}_{\mathbf{H}}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x}-\mathbf{x}_i) \tag{4}$$

with $\mathbf{H}$ being a symmetric and positive definite bandwidth matrix and $K_{\mathbf{H}}$ being defined as

$$K_{\mathbf{H}}(\mathbf{x}) = |\mathbf{H}|^{-\frac{1}{2}}\mathscr{K}(\mathbf{H}^{-\frac{1}{2}}\mathbf{x}).$$

$\mathscr{K}$ is a multi-variate kernel function that integrates to 1. For the 2D case – central to all of the following –, we will consider the following simplified form of the bandwidth matrix

$$\mathbf{H}_{2\mathrm{D}} = \begin{vmatrix} h_{1,1} & 0 \\ 0 & h_{2,2} \end{vmatrix}$$

that leads to the following form of a 2D KDE:

$$\widehat{f}_{2\mathrm{D}}(x,y) = \frac{1}{nh_{1,1}h_{2,2}}\sum_{i=1}^{n}\mathscr{K}\left(\frac{(x-x_i)}{h_{1,1}}, \frac{(y-y_i)}{h_{2,2}}\right)$$

Also in 2D, the kernel function $\mathscr{K}$ is usually chosen to be a probability density function. There are two common techniques for generating a multivariate kernel $\mathscr{K}$ from a symmetric univariate reference kernel K [30]:

$$\mathscr{K}^{\mathrm{P}}(\mathbf{x}) = \prod_{i=1}^{d} K(x_i) \quad \text{and} \quad \mathscr{K}^{\mathrm{S}}(\mathbf{x}) = K(|\mathbf{x}|)/c_{k,d}$$

where $c_{k,d} = \int K(|\mathbf{x}|)\,\mathrm{d}\mathbf{x}$. $\mathscr{K}^{\mathrm{P}}$ is known as the product kernel and $\mathscr{K}^{\mathrm{S}}$ as the radially symmetric isotropic kernel. The latter of these kernels is a radial basis function (RBF), and should only be used when a single bandwidth can be devised for all the plotted dimensions. When plotting two different units, or attributes of different scale, we choose the product kernel with individual bandwidth values for the two represented data dimensions. We have now defined kernel density estimation, especially also in its 2D form, and we have compared KDE-based 2D visualization with scatterplots. Later in this paper, as a technical contribution, we present an approach to compute KDEs on the GPU, achieving a speed-up factor of about 100 (compared to existing KDE algorithms), and thereby enabling interactive frame rates needed for this visual data exploration and analysis; even for large datasets.

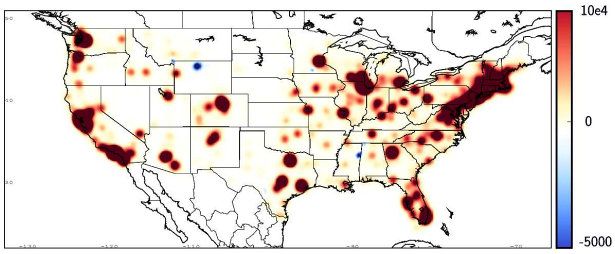


Figure 4: Visualizing over 165 000 monetary contributions to the Obama campaign. Interesting areas with negative aggregates, i.e., locations where the returned amount exceeds that of the contributed, are shown as blue.

4 Reconstructing the Distribution of a Third Attribute

In the following we discuss an extension of the KDE concept that allows the visualization of the distribution of a third data attribute (with respect to two other data attributes as in the scatterplot).

We first forgo the normalization in Eq. 4, i.e., we omit the division by the number of data items n, and thereby achieve an estimate function that will integrate to n, accordingly. Next, we introduce a weighting factor c_i to each of the accumulated kernels, that we make dependent on a third data attribute $d_{i,c}$. The new estimate is then defined as

$$\widehat{g}_{\mathbf{H}}(\mathbf{x}) = \sum_{i=1}^{n} c_i K_{\mathbf{H}}(\mathbf{x}-\mathbf{x}_i) \tag{5}$$

Visualizing $\widehat{g}_{\mathbf{H}}(\mathbf{x})$, e.g., as a height field over the 2D domain of $\mathbf{x}$, will (as a whole) communicate the accumulated sum of all values c_i of data dimension $d_{i,c}$ since

$$\int \widehat{g}_{\mathbf{H}}(\mathbf{x})\,\mathrm{d}\mathbf{x} = \sum_{i=1}^{n} c_i\,. \tag{6}$$

Due to its close relation to KDE, we achieve a continuous reconstruction of the distribution of this "value mass" with respect to the two other data attributes $d_{i,a}$ and $d_{i,b}$. This leads to very interesting visualization options for absolute quantities (not just relative densities as with KDE). In Fig. 4, for example, we visualize the distributional characteristics (here with respect to longitude and latitude) of more than 165 000 monetary contributions to the recent Obama campaign; data acknowledged FEC [10]. We achieve a continuous reconstruction of a distribution function that tells in which places how much was contributed. One strange result from this visualization is the identification of locations where the overall aggregation of all c_i values is negative (resulting in blue color), meaning that the average contribution per square mile is a negative amount of dollars. The dataset contains transactions that represent contributions that have not been accepted (and therefore returned, accordingly). One valid explanation for these negative areas is that the agencies have been more meticulously in registering the zip-code for cash returns than the initial contribution (but additional analysis would be required to fully understand this phenomenon).

5 Reconstructing Time

In many cases, and also later in our application context, we are confronted with streaming data from different types of processes. To

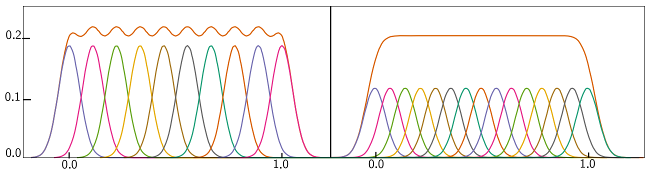


Figure 5: Gaussian kernels with a bandwidth of 0.05 and their combined integral (orange). Left 10 kernels and their sum, and right 15 kernels and their sum. These figures represents the super-sampling approach, whereas Eq. 8 calculates the sum directly.

achieve a truthful visualization of time-dependent data of this type, we need to integrate KDE with a proper representation of the continuous change over time. One approach could be to super-sample the streaming data with respect to time, resulting in a reconstruction based on a large set of kernels. Instead we suggest using a line kernel that amounts to a pre-integrated continuous solution to this problem. Fig. 7 shows the proxy geometry needed to implement this super-sampling (left) and our line kernel (right).

Accordingly, we adapt kernel density estimation to reflect this reconstruction scheme. We suggest a kernel L_k to reconstruct the contribution of a line (instead of just a point). Then, assuming a dataset of n in-streaming data items, the time reconstruction estimate $\hat{t}(\mathbf{x})$ becomes

$$\hat{t}(\mathbf{x}) = \sum_{k=1}^{n-1} L_k(\mathbf{x})\,. \tag{7}$$

For every two consecutively in-streaming data items $\mathbf{d}_i$ and $\mathbf{d}_{i+1}$, and their associated point locations $\mathbf{p}_i = \mathbf{p}(\mathbf{d}_i)$ and $\mathbf{p}_{i+1} = \mathbf{p}(\mathbf{d}_{i+1})$ in the 2D KDE domain, a line reconstruction kernel L_k is placed that is constructed as follows:

$$L_k(\mathbf{x}) = \int_0^1 c_i K_{\mathbf{H}}\big(\mathbf{x} - ((1-\phi)\mathbf{p}_1 + \phi\mathbf{p}_2)\big)\, \mathrm{d}\phi \tag{8}$$

$K_{\mathbf{H}}$ is one of the kernels that otherwise are used for point reconstruction, in our case we use the normal kernel here. And c_i is a scaling factor for each line segment, i.e., when reconstructing time, the time passed, especially also to support uneven sampling. Eq. 8 is the converged result of distributing point reconstruction kernels evenly along the line segment. The converged result of super-sampling is detailed, as a 1D example, in Fig. 5, whereas Eq. 8 directly evaluates the converged result. Fig. 6 illustrates the distribution of time, when tracing a sequence of four points, or, three edges, each weighted with one second. According to Eq. 8, these three line kernels each contribute a weight of one second, to the total integral, but since the line on top has a shorter distance between vertices, the density here is higher. Further below, in section 7, we present examples from our application case, e.g., in Fig. 9, that was also reconstructed with this approach.

6 Interactivity and Analysis

Defining interaction with a system requires one to first identify the internal parameters that can be modified, and second, on a higher level, identify the tasks that users would perform on that system. The parameters available in a KDE-based visualization are only data-samples, bandwidth, and viewport. Shneiderman listed a set of tasks [24] that fit the information visualization workflow, "Overview first, zoom and filter, then details-on-demand". Creating an overview from a KDE is simply ensuring that the shown range of the two dimensions is spanning all samples and choosing an appropriate bandwidth. Zooming and panning are direct manipulations of the viewport, and is closely related to filtering out those samples out of view. Since data investigated often have different units or scale, we suggest that zooming should be allowed individually per axis;

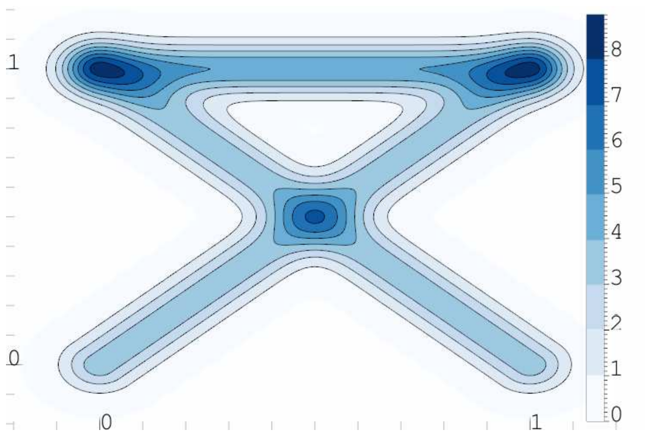


Figure 6: A line kernel density reconstruction of four samples, or three edges. Each edge is weighted by one, e.g., one second, and thus the integral of this entire figure is three. The time density at the top edge is greater than the diagonals, since this distance is smaller, and its weight is the same.

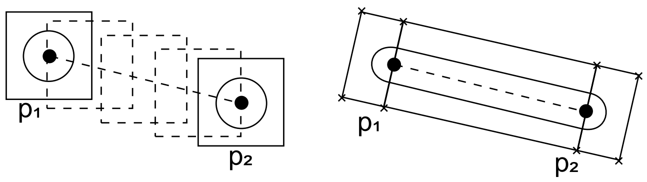


Figure 7: Reconstructing two connected samples in time, on left, super-sampling by filling the space with additional samples, and on right, by drawing a continuous rectangle and two end-caps. Both techniques produce the same result, but our line kernel density estimate does so with a significant efficiency increase.

however there are cases where an enforced aspect ratio is desired. When the unit of both axes is the same, and the scale is comparable, keeping an aspect ratio of 1:1 would help to not introduce any misleading scale impression. Another case where an enforced aspect ratio is useful is when displaying maps, or lat-lon axes. In this case we enforce an equidistant cylindrical ratio, which is ratio varying on the current viewport's latitude. This ratio ensures that at least the area around the latitude line in the center of the viewport is equal-area [26].

Often, the automatic generation of parameters is more important than interaction, and two examples of automatic parameter generation are (1) generate decent initial / default values, and (2), have parameters generated optimally, creating a nonparametric functionality, and even removing the need for user-interaction. Visualizing large datasets often makes it impossible to create an optimal viewport, showing all the data, which is why zooming and panning is introduced. There are works trying to globally optimize the bandwidth, e.g., the normal scale rule [25], but we find that this factor is highly dependent on the viewport, e.g., if the bandwidth in either dimension is less than a pixel, nothing is shown. Instead of calculating an optimal bandwidth based on the data-sample distribution, we propose a method that is tightly coupled with the viewport, that will update the bandwidth when the viewport changes. In a right-hand system, a viewport is defined by two points, the lower-left $\mathbf{p}_1$ and upper right $\mathbf{p}_2$. The range, $\mathbf{r}$, of this viewport is then $\mathbf{r} = \mathbf{p}_2 - \mathbf{p}_1$. We then define the pixel size, $\mathbf{s}$, as $\mathbf{r}$ divided by the screen size. We then have two observations. One, if the bandwidth $\mathbf{H}$ is less than $\mathbf{s}$, i.e., less than a pixel, the sample is not shown, and thus we recommend $\mathbf{H} > \mathbf{s}$. Second, if the bandwidth is larger, by a factor k, than the range $\mathbf{r}$ of the viewport, the observed result will be a near constant sum of the kernels within, and around, the view. By defining that $k \cdot \mathbf{r} > \mathbf{H} > \mathbf{s}$ we can assert a viewport independent density estimation of the prominent visible features. If we continue to enforce a bandwidth tied to $\mathbf{s}$, i.e., a pixel bound bandwidth, throughout interaction, we can zoom out to aggregate more features for an overview, and zoom in for a more detailed view. An example of this interactively changing bandwidth is shown in Fig. 1, and in the supplementary video. It our experiences, a bandwidth from approx 2 to 20 times that of a pixel, works well, and is in fact representative for all the figures in this paper, relying on line kernels.

The next task is filtering, i.e., showing only a subset of the samples. When dealing with time dependent data, the most common filter allows temporal selection and animation. Animating temporal trends can be achieved by setting three attributes, namely, *time*, *time window* and *time step*. Time is the current point in time that samples are shown until. Time window is the how far back in time from *time* samples should be visible, and time step is the increment per animation step. E.g., when showing weekday trends, the time window should be set to 24 hours, but the time step could be set to one hour, so that one would, in a video with 24 fps, get a smooth animation from day to day with a day lasting a second in the animated visualization.

The last task we facilitate is details on demand, however since KDE is not an item based visualization, selection is not available. Instead we propose a simple integration scheme where a bounding box is drawn, and the area within this box is integrated. From Eq. 6 we see that the open integral is the sum of all sample-weights, and similarly the bounded integral gives a sum of the selected region. This interaction enables accurate quantitative analysis of the distribution of this third attribute.

7 Demonstration

In this section we cover three different cases involving streaming data. The first case covers ship traffic off the coast of Norway, the second case investigates data from drilling operations in the petroleum industry, and the third all commercial air traffic in the US spanning two decades from 1987 to 2008.

7.1 AIS Ship Traffic

The Automatic Identification System (AIS) is a radio based system used by ships and other vessels for collision detection and identification. The International Maritime Organization requires all ships with a gross tonnage of 300 or more, in addition to all passenger ships, regardless of size, to be equipped with this system. With the KDE-based visualization approach described here, we enable the real-time filtering, analysis, and rendering of large sets of stored as well as of streaming AIS data. The AIS signals that we study are picked up by the Norwegian shore based network. Here we visualize 14 days of AIS data in which a total of 5000 ships are registered, sending 850 thousand position updates. Willems et al. recently presented a technique for convolving kernels along AIS ship paths [32]. Our visual results are similar to theirs in terms of AIS data. Their implementation, however, takes approx. 10 minutes to compute (data for one day, i.e., 100 000 line segments). Our technique calculates similar results for 14 days (850,000 line segments) in 43 ms (23fps). Because of the rendering speeds we achieve, and since we do not need pre-processing, we can connect to the live feed for streaming AIS data. Fig. 8 shows a small section of the area covered by the Norwegian AIS system, outside the south-western coast. These two figures clearly show the advantage of our line kernel reconstruction. On both images, the traffic close to the coast, enclosed by headlands, are clearly defined, but out in the open sea, where the radio signals are weaker, the samples become so sparse that it is hard to detect where the ships move. By zooming to a smaller region, with the sample bandwidth reduced automatically,

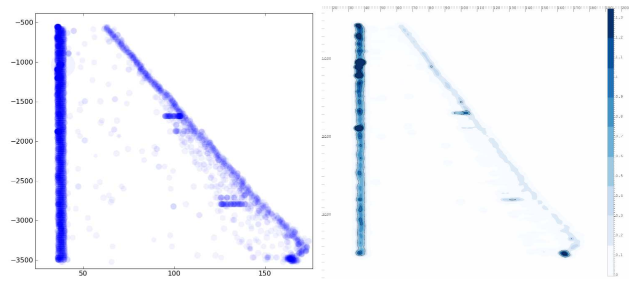


Figure 9: A side by side comparison: an overpopulated scatterplot with semi-transparent points (left) vs. our visualization with line KDE (right). Compared, the bottom of these two gives a clear overview of where time is distributed with regards to hook-load and depth. The dark blue areas to the left indicate non-productive time.

this sparseness increase even to affect the dense areas in this figure. Using this side by side visualization highlights where the dead zones of the AIS radio system is, and thus where perhaps this could be extended.

Statistics on AIS data have several times proven useful, e.g., when calculating the risk new offshore installations face with respect to collisions. Using our technique we have increased the speed of calculating these probability plots to such a degree that one can interact with them (i.e., recalculate them) at real time speeds (for this dataset, 23 fps). As Norway aims to invest in several new offshore windmill parks, our techniques will enable both manual investigations, and faster and more complex automated placement algorithms.

7.2 Drilling operations

In a project with partners from the Oil and Gas industry we investigate the distribution of time in drilling operations. The dataset that we visualize here contains several measured and derived attributes from this process. In this context we look closer at three of these, namely, *depth*, *hook load*, and *time*. Depth is the length of the drill string that is in the bore hole (and not true vertical depth) and hook load is the measured weight of this drill string. In Fig. 9 we present the visualization of these three attributes in two different versions, a regular scatterplot using transparency and a KDE-based visualization using line kernels. The vertical scale is depth, down being deeper, and the horizontal scale is hook load. The most prominent visible features are the two bands, one vertical and one diagonal. The vertical band, at approx. 35 tons, is the weight of the hook when the drill string is not attached to it, and is thus an indicator of the time spent attaching or detaching a new pipe segment to/from the string. The diagonal band is the weight when the drill string is attached to the hook, indicating weight increasing with depth, since there are more pipes attached to the hook. This dataset was acquired when the drilling crew decided to pull the entire string up, from 3500 meters down. This operation is performed every time there is something wrong, or, they want to set a new casing, or change the drill bit. It is important to do this as fast as possible, as time efficiency is paramount to have a good return on investment. When presented to the domain engineers, the first feature discussed was the visualization of unscheduled stops, shown as local peaks. To analyze further, the biggest of these, at about 1000 meters, was zoomed onto (see Fig. 10) and the integral shows a total of one hour, as compared to normally approx. two minutes for removing a 90 feet pipe. One scenario that makes good use of this tool is for the onshore team that monitors the ongoing process, or for the change of shifts, where a new team takes over the drilling, and they would need to get an overview of the recent history of progress and events.

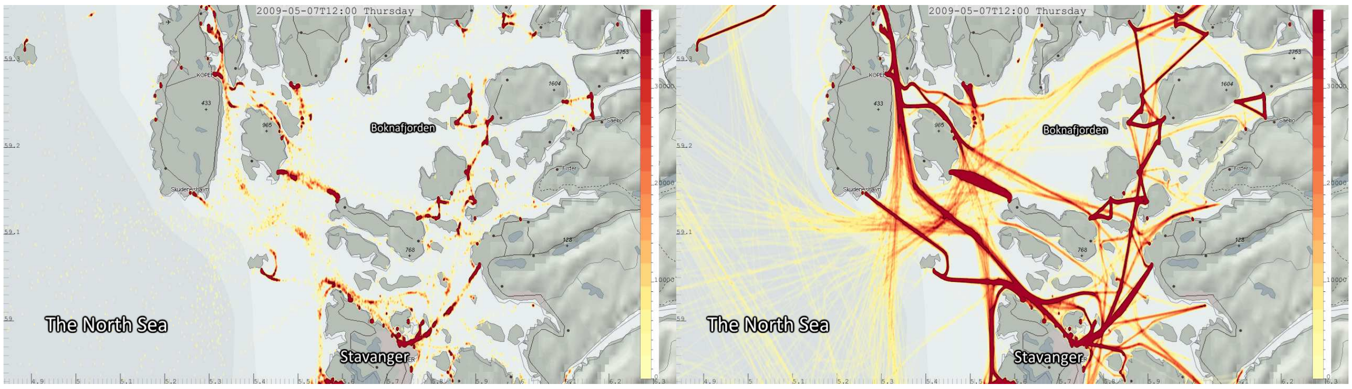


Figure 8: Two KDE-based visualizations using the same bandwidth and data, of ship traffic off the coast in western Norway. The left image shows the position samples as point kernels, and the right image shows the same data using our line reconstruction kernels.

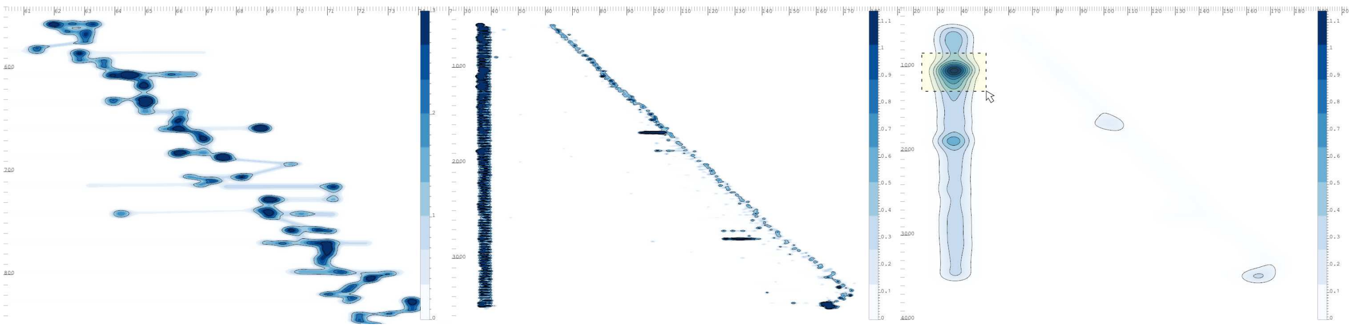


Figure 10: Three line kernel density estimates, showing the distribution of time over depth in a drilling hole (wellbore) and hook-load, the weight of the entire drill string. The leftmost image is a detailed view, with a small kernel, showing the curve with varying tons on the hook, used e.g., to calculate friction. The user then zooms out, and goes to an overview mode with a large kernel, on right, and selects an overwhelming time density, integrates and finds that over an hour was nonproductive at this depth.

7.3 Commercial Air Traffic

In this section we show how our line kernel density estimate enables insights into a dataset containing all commercial air traffic in US, from October 1987 to April 2008. This dataset [3] contains 120 million flights and makes out 12 gigabytes. The distances flown are calculated by Haversine distance from airport to airport, and goes from 16 trips to the sun and back in 1987 to 28 round-trips in 2007. One interesting note about the summary of all flights is that while the total flight hours shows an increase of 172% from 1988 to 2007, the number of takeoffs only increased by 142% in the same period, i.e., the more recent average flights travels longer.

This dataset is particularly interesting to investigate using line kernel density estimation (as opposed to regular KDE) because of both the large spatial distance between points. As defined here, one flight is a scheduled takeoff; this dataset contains the origin and destination airport of all flights. From the airport codes and all actual takeoff and landing times we created a new dataset. This dataset is a temporal line-segment dataset. A temporal line-segment consists of two points with values for latitude, longitude and time, each.

Our prototype can show temporal animations at real time, concurrently with interaction, which both require reconstruction of the KDE for every frame. An example interaction is shown in Fig. 1, where the kernel size/bandwidth of the estimate is tied to pixel size, instead of, e.g., km. This bandwidth enables the user to zoom in, while simultaneously refining spatial information. This Fig. 1, contains the automatic aggregated flight hours over the Bay Area at the initial zoom level, and after zooming in, can determine the distribution among the different airports, and their respective distributions along the different cardinal directions as such.

The top row of Fig. 11 shows hour by hour as dusk moves over the US, the air traffic picks up from east to west, a pattern that repeats itself at night, as well. The bottom row of Fig. 11 shows a more dramatic pattern, at September 11th, 2001.

8 Technical Details and Accuracy

In the following, we discuss how we implemented the above presented approach on graphics hardware and discuss performance and accuracy of this solution.

8.1 Kernel Density Estimation on the GPU

The use of modern GPU-accelerated techniques in data visualization is a promising step [11], especially since interactive visual analysis relies on interaction, and thus on interactive rendering. In our prototype we developed a two step technique for computing and visualizing KDE. The first step is to generate a floating point field by evaluating the 2D KDE equation, and the second step is to appropriately visualize this KDE field with one of several options.

Calculating KDE on the GPU requires the support of floating point, or double precision textures, as we need to store results with an appropriate precision. Evaluating the 2D KDE function to a 2D matrix (a texture), can be done in one of two ways, with one cell being one element/texel in our matrix with the properties of a value `v` and a position `p`:

```
    for c in cells:          for k in kernels:
a)   for k in kernels:   b)   for c in cells:
      c.v+=k.eval(c.p)          c.v+=k.eval(c.p)
```

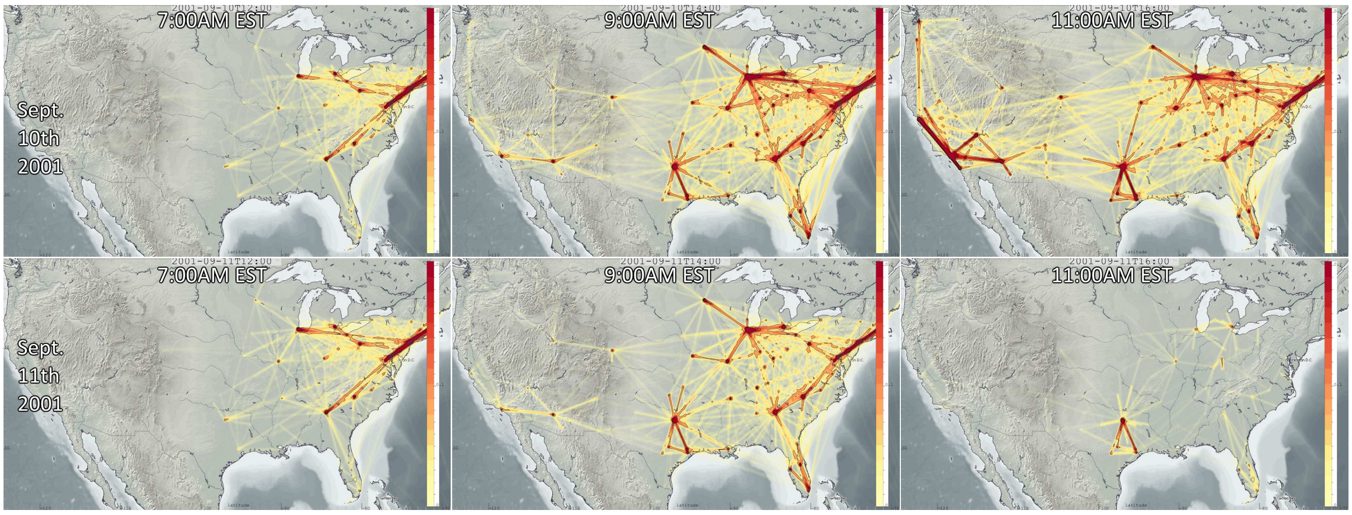


Figure 11: Temporal animation of air traffic on the 10th and 11th of September 2001. The top row shows a normal pattern of how the traffic evolves, following the timezones. These views show a time-window of the two hours leading up to the given time. The bottom row all traffic is cut short, lasting several days, due to the tragic events at this date.

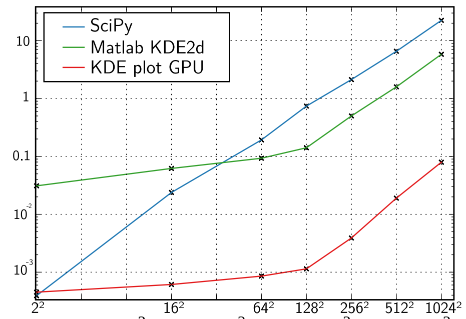
I.e., we can either first iterate over the grid cells, or over the kernels, respectively. The latter case is identical to rasterizing on the GPU, and thus this is our selected approach. To create the result, we first allocate a grid, as a 2D *frame buffer object* (FBO), with floating point precision. Then, with this FBO bound, we render all the kernels. All of them are then aggregated with an additive blend operator. To create an optimized implementation, we allow for an approximation of KDE by limiting the extent of all kernels (we will return to this subject in the next section). To further optimize this implementation as well as, to enable distinctive kernels, we precompute the kernel and store them as a floating point texture. The geometry needed for point kernels can be created by either using the point sprite extension, drawing quads, or more efficiently using geometry shaders. The use of point sprites or geometry shaders reduces the necessary vertices to one. Fig. 7 shows the necessary vertices needed to construct a line kernel, which we construct out of three quads. Here the use of a geometry shader reduces the necessary vertices to two, p_1 and p_2.

To enable a fair comparison to other KDE algorithms, we have created a Python interface, that stores the result as NumPy arrays. Fig. 12 shows the result of a comparison of three different algorithms for the 2D kernel density estimation in the Iris dataset, containing 150 samples. The three different implementations we used are the SciPy [16] implementation, a Matlab™ file implemented by Botev [5], and our implementation on the GPU. As this table shows, there is a significant, up to approx 300 times large speed-up, e.g., compared to the Matlab implementation for the 1024^2 grid.

8.2 Error Estimation

In this section we investigate the computational accuracy of our GPU-based KDE (based on a Gaussian kernel as discussed in Sec. 3), in addition to an overall discussion on the errors or drawbacks that can arise using KDE. As a kernel with infinite extent, the Gaussian is defined over the entire real line $\mathbb{R}$. As an approximation, a windowed kernel can be considered, e.g., by truncation [7]. To investigate how good bounded approximations are, we look at their integral for comparison. The finite integral of the Gaussian 2D product kernel, $N(x,y) = \frac{1}{2\pi} e^{-\left((x^2+y^2)/2\right)}$, is:

$$\int_{-n}^{n}\int_{-n}^{n} N(x,y)dydx = \operatorname{erf}\left(\frac{n}{\sqrt{2}}\right)^2 \tag{9}$$



Technique	16^2	64^2	256^2	512^2	1024^2
KDE-plot GPU	6.1 E −4	8.5 E −4	3.9 E −3	1.9 E −2	7.9 E −2
matlab KDE2D	6.2 E −2	9.3 E −2	0.5	1.6	5.8
SciPy	2.4 E −2	0.19	2.1	6.5	22.4

Figure 12: Run times (in seconds) for evaluating grids of different sizes, for three different implementations of kernel density estimation, all using same dataset, kernel and bandwidth. *KDE-plot GPU* is our proposed technique.

n/interval	1	2	3	4	5
error	0.53	8.9 E −2	5.39 E −3	1.27 E −4	1.15 E −6

Table 1: Error introduced by using a truncated Gaussian.

Technique	2^2	4^2	8^2	64^2	128^2
Central	0.97	0.163	1.33 E −5	1.12 E −6	1.14 E −6
Preintegrated	0.0	0.0	0.0	0.0	0.0

Table 2: Error introduced by integrating (summing) textures of different sizes. Results show one minus integral.

where erf is the “error function” (encountered when integrating the normal distribution). Using Eq. 9, we can calculate that the use of a texture with interval $[-n,n]^2$ will result in an error as shown in table 1. In cases were normalized kernels are used, the interval $[-5,5]^2$ with an error of $1.15E-6$ is sufficient. However since we are scaling every kernel by a factor, this error would also be scaled linearly.

When representing a kernel K as a discretized texture, the integral is the sum of all texels, multiplied by the texels' size (e.g., on an interval $[-5,5]^2$ and on an 128^2 texture: $10^2/(128^2)$). Using a discretized 2D Gaussian in the interval $[-5,5]^2$ can ideally never achieve a better integral than eq. 9, but we now look into the actual integrals using different techniques. We compare two techniques for creating and integrating kernel textures. The first, called *central*, gives every texel its value after evaluating K with its central position. In the second technique, called *preintegrated*, the integral over the the area spanned by the texel is assigned to the texel. Table 2 shows the errors introduced using different techniques and texture sizes. The errors presented for the central technique will, for larger texture sizes, converge towards the error presented in table 1.

Kernel Density Estimates, reconstruct a continuous distribution from a discrete set of samples, essentially by smoothing. In several cases, this smoothing can introduce errors. As an example of this smoothing error, we can think of a shipping lane, where the vessels are passing through a very narrow straight. If we smooth out these vessel paths, we have a low tolerance, before we introduce a probability of finding vessels on land. While not covered in this paper, there is several existing works, on variable kernel density estimation, on how to specify an individual, and optimal bandwidth, for every sample. In our implementation of the line kernel, defined in Eq. 8, we allow for an individual bandwidth per sample, enabling support for varable kernel density estimation. However, for purposes on streaming data, without pre-processing, this individual bandwidth cannot be implmented, in a trivial fashion.

Another source of errors lies in our restriction to a simple bandwidth matrix, in Eq.4. If the data modeled contains a diagonal distribution, the correct kernel to use would be one with skew, and thus cannot be modeled using our simplified bandwidth. It is however trivial to extend, the line kernel to allow the full bandwidth matrix. Our rationale for not utilizing this however, lies in the lack of preprocessing, so, we, because of streaming data, cannot pre-process to find this optimal bandwith matrix.

9 Summary and Conclusions

In this paper, we discuss the challenge of intuitively visualizing large amounts of discrete data samples. We discuss a KDE-based visualization, defined from the statistical concept of kernel density estimation (KDE), as an elegant solution. We adapt this concept to also allow for investigating the distributional characteristics of an additional, third attribute over two dimensions. Additionally, we show how KDE-based visualizations can be extended to visualize the distribution of time in the context of streaming data (with a new type of a line kernel). We explain and demonstrate how KDE-based visualizations can be computed on the GPU, leading to speed-up factors around 100 (and up to approx 300 in one of our cases). We briefly report on our prototype in the maritime, the oil & gas domain, and air traffic and show that useful results are achieved.

We demonstrate that due to our improvements to both regular and streaming KDE-based visualizations, utilizing modern GPUs, it is now possible to utilize advanced concepts from statistics for improved visual data exploration and analysis, for large data at interactive speeds. With respect to KDE, in particular, it would be great to see more interesting related future work in visualization.

Acknowledgements

The work presented here is a part of the project "e-Centre Laboratory for Automated Drilling Processes" (eLAD), participated by International Research Institute of Stavanger, Christian Michelsen Research and Institute for Energy Technology. The eLAD project is funded by grants from the Research Council of Norway (Petromaks Project 176018/S30, 2007-1010), StatoilHydro ASA and ConocoPhillips Norway. Furthermore we acknowledge the Norwegian Coastal Administration for supplying access to the AIS.

References

[1] N. Andrienko and G. Andrienko. Spatial generalization and aggregation of massive movement data. *IEEE Trans. Vis. Comput. Graph.*, 2010. (RapidPost).

[2] A. Artero, M. de Oliveira, and H. Levkowitz. Uncovering clusters in crowded parallel coordinates visualizations. *Proc. of IEEE Symp. on INFOVIS*, 2004.

[3] Asa data expo 2009. http://stat-computing.org/dataexpo/2009.

[4] S. Bachthaler and D. Weiskopf. Continuous scatterplots. *IEEE TVCG*, 14(6), 2008.

[5] Z. I. Botev. A novel nonparametric density estimator. *Postgrad. Sem. Series, Math. , The Univ. of Queensland*, 2006.

[6] M. D. Buhmann. *Radial basis functions*. Cambridge Uni. Press, 2003.

[7] R. A. Crawfis and N. Max. Texture splats for 3d scalar and vector field visualization. In *VIS '93: Proc. of the 4th conf. on Vis. '93*, 1993.

[8] G. Ellis and A. J. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE TVCG*, 13(6), 2007.

[9] M. Ericson. Keynote: Visualizing Data for the Masses: Information Graphics at The New York Times. *VisWeek*, 2007.

[10] Federal election commission. http://www.fec.gov/.

[11] J. Fekete and C. Plaisant. Interactive information visualization of a million items. In *Proc. of IEEE Symp. on INFOVIS*, 2002.

[12] D. Fisher. Hotmap: Looking at Geographic Attention. *IEEE TVCG*, 13(6), 2007.

[13] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics 7*, 1936. StatLib http://lib.stat.cmu.edu/.

[14] A. Gray and A. Moore. Nonparametric density estimation: Toward computational tractability. In *SIAM Int. Conf. on Data Mining*, 2003.

[15] Y. Jang, M. Weiler, M. Hopf, J. Huang, D. Ebert, K. Gaither, and T. Ertl. Interactively visualizing procedurally encoded scalar fields. In *Proc. of EG/IEEE TCVG Symp. on Vis. VisSym*, volume 4, 2004.

[16] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[17] P. Kidwell, G. Lebanon, and W. Cleveland. Visualizing Incomplete and Partially Ranked Data. *IEEE TVCG*, 14(6), 2008.

[18] M. C. Minnotte and D. W. Scott. The mode tree: a tool for visualization of nonparametric density features. *Journal of Computational and Graphical Statistics*, 2, 1993.

[19] P. Muigg, J. Kehrer, S. Oeltze, H. Piringer, H. Doleisch, B. Preim, and H. Hauser. A 4-level Focus+Context Approach to Interactive Visual Analysis of Temporal Features in Large Scientific Data. *Comp. Graph. Forum*, 27(3), 2008.

[20] M. Novotný and H. Hauser. Outlier-preserving focus+context visualization in parallel coordinates. *IEEE TVCG*, 12(5), 2006.

[21] E. Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3), 1962.

[22] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3), 1956.

[23] D. W. Scott. *Multivariate density estimation: theory, practice, and visualization*. Wiley-Interscience, illustrated edition, 1992.

[24] B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *IEEE Visual Languages*, 1996.

[25] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, 1986.

[26] J. P. Snyder. *Flattening the Earth: Two Thousand Years of Map Projections*. University of Chicago Press, 1993.

[27] D. Tarn. An introduction to kernel density estimation. http://school.maths.uwa.edu.au/~duongt/seminars/intro2kde/, 2001.

[28] E. Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphics Press, 1997.

[29] B. A. Turlach. Bandwidth Selection in Kernel Density Estimation: A Review. In *CORE and Institut de Statistique*, 1993.

[30] M. Wand and M. Jones. *Kernel Smoothing*. Monographs on Statistics and Applied Probability 60. Chapman & Hall, 1995.

[31] G. Whittaker and D. Scott. Nonparametric regression for analysis of complex surveys and geographic visualization. *Sankhyā: The Indian Journal of Statistics, Series B*, 1999.

[32] N. Willems, H. van de Wetering, and J. J. van Wijk. Visualization of vessel movements. *Proceedings of EuroVis*, 2009.