Visualization of Big Spatial Data using Coresets for Kernel Density Estimates

Yan Zheng, Yi Ou, Alexander Lex, Jeff M. Phillips,

Abstract—The size of large, geo-located datasets has reached scales where visualization of all data points is inefficient. Random sampling is a method to reduce the size of a dataset, yet it can introduce unwanted errors. We describe a method for subsampling of spatial data suitable for creating kernel density estimates from very large data and demonstrate that it results in less error than random sampling. We also introduce a method to ensure that thresholding of low values based on sampled data does not omit any regions above the desired threshold when working with sampled data. We demonstrate the effectiveness of our approach using both, artificial and real-world large geospatial datasets.

Index Terms—Spatial data visualization, sampling, big data, coresets.

1 Introduction

Data is collected at ever-increasing sizes, and for many datasets, each data point has geo-spatial locations (e.g., either (x,y)-coordinates, or latitudes and longitudes). Examples include population tracking data, geo-located social media contributions, seismic data, crime data, and weather station data. The availability of such detailed datasets enables analysts to ask more complex and specific questions. These have applications in wide ranging areas including biosurveillance, epidemiology, economics, ecology environmental management, public policy and safety, transportation design and monitoring, geology, and climatology. Truly large datasets, however, cannot be simply plotted, since they typically exceed the number of pixels available for plotting, the available storage space, and/or the available bandwidth necessary to transfer the data.

A common way to manage and visualize such large, complex spatial data is to represent it using a continuous, smoothed function, typically a kernel density estimate [20], [21] (KDE). A KDE is a statistically and spatially robust method to represent a continuous density using only a discrete set of sample points. Informally, this can be thought of as a continuous average over all choices of histograms, which avoid some instability issues that arise in histograms due to discretization boundaries. Or it is a convolution of all data points with a continuous smoothing function. For a formal definition, we first require a kernel $K : \underline{\mathrm{R}}^2 \times \underline{\mathrm{R}}^2 \to \underline{\mathrm{R}}$; we will use the Gaussian kernel $K(p,x) = e^{-\|p-x\|^2}$, the most common and pervasive kernel. Then, given a planar point set $P \subset \underline{\mathrm{R}}^2$, the kernel density estimate is defined at any query point $x \in \underline{\mathrm{R}}^2$ as

$$\mathrm{KDE}_P(x) = \frac{1}{|P|} \sum_{p \in P} K(p,x).$$

This allows regions with more points nearby (i.e., points x with a large value $K(p,x)$ for many p in P) to have a large density value, and this function is smooth and in general nicely behaved in many contexts. Using this function summarizes the data, and avoids the over-plotting and obfuscation issues demonstrated in Figure 1(left). However, just computing $\mathrm{KDE}_P(x)$ for a single value x requires $O(|P|)$ time; it iterates over all data points summing their contributions. While these values can be precomputed and mapped to a bitmap, visually interacting with a KDE e.g., to query and filter, would then require expensive reaggregating. For instance, as a user zooms in on a region of interest, ideally the visual interface should increase the resolution, and possibly shift the grid boundaries. This would require recomputing each of these visible pixel values in $O(|P|)$ time each.

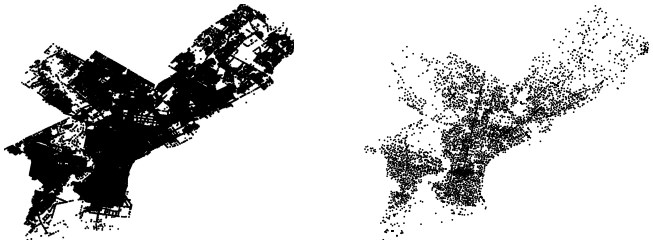

Fig. 1. Crimes from 2006 to 2013 in Philadelphia, the full dataset (left) with 0.7 million points and a coreset (right) with only 5300 points.

Towards alleviating these issues, we propose to use **coresets** for KDEs. In general, a *coreset* Q is a proxy for a large set P; it is a carefully designed small subset of a very large dataset P where Q retains properties from P as accurately as possible. In particular, in many cases the size of Q depends only on a desired minimum level of accuracy, not the size of the original dataset P. This implies that even if the full dataset grows, the size of the coreset required to represent a phenomenon stays fixed. This also holds when P represents a continuous quantity (like the locus of points along a road network, or a spread of particulates from a forest fire) and Q constitutes some carefully placed representative points [24]. Figure 1 shows a dataset P with 700 thousand points and its coreset from all reported crimes in Philadelphia from 2005-2014. For more details on variations and constructions, refer to recent surveys [3], [17].

In particular, a coreset for a kernel density estimate is a subset $Q \subset P$ [16], [31], with $|Q| \ll |P|$ so that for some error parameter ε

$$L_\infty(\mathrm{KDE}_P, \mathrm{KDE}_Q) = \max_{x \in \underline{\mathrm{R}}^2} |\mathrm{KDE}_P(x) - \mathrm{KDE}_Q(x)| \leq \varepsilon. \quad (1)$$

Y. Zheng is with Visa Research, e-mail: yanzh.cs@gmail.com; Much of this work was completed while at the University of Utah.
Y. Ou is with Expedia, Inc, e-mail: olly93219@outlook.com; Much of this work was completed while at the University of Utah.
A. Lex is with University of Utah, e-mail: alex@sci.utah.edu
J. Phillips is with University of Utah, e-mail: jeffp@cs.utah.edu

This means that at any and all evaluation points x, the kernel density estimates are guaranteed to be close. In particular, such a bound on the *worst case error* is essential when attempting to find outlier or anomalous regions; in contrast an average case error bound (e.g. $L_1(\text{KDE}_P, \text{KDE}_Q)$) would allow for false positives and false negatives even with small overall error. Thus, with such a worst-case bounded coreset Q, we can use KDE_Q efficiently without misrepresenting the data.

In the rest of this paper we demonstrate two properties of coresets used for KDEs that make them pertinent for visual analysis. In Section 3, we first demonstrate that we can create a coreset that is more accurate than the naive but common approach of random sampling. Second, very sparse subsets (e.g., from random sampling) tend to cause anomalous regions of low, but noticeable density; we introduce a method to counteract this problem in Section 5, by carefully adjusting the smallest non-zero layer of the corresponding transfer function. Towards demonstrating these insights we design and present an interactive system for visualizing large, complex spatial data with coresets of kernel density estimates. Based on these insights, we believe that coresets and kernel density estimates can become an important tool for interactive visual analysis of large spatial data.

2 Related Work

Visualizing large spatial datasets is an important challenge attracting a lot of attentions among the visualization community. Visualization researchers widely consider encoding data with position, as it is e.g., used in scatterplots or bar charts, to be the most powerful visual channel [6]. Yet, when visualizing geospatial data, position is needed to indicate the position of a data point on a map. Consequently, spatial data visualizations have to rely on other channels such as color, e.g., by color-coding data points or regions on a map (choropleth mapping), or size, e.g., by displaying proportionally scaled circles on top of maps. For multidimensional data vectors that are associated with geospatial positions, complex glyphs drawn on top of the map are also an option [22], [25], [26].

All of these approaches, however, have significant drawbacks: color-coding either of individual data points or regions is effective, yet limited to one variable at a time, making it difficult to communicate temporal trends, or complex data vectors. Adding symbols for displaying uncertainty or other information is likely to increase the visual complexity for map users and most of mapping projects still disregarded data uncertainty [23]. Maps using symbols or glyphs also commonly suffer from occlusion, as data is rarely evenly distributed in geospatial datasets.

This uneven distribution of data points in geopspatial datasets is also a significant problem for choropleth maps: in the case of population-related data, for example, large, sparsely populated areas, are over-represented on choropleth maps with respect to the pixels on the screen. For example, while Manhattan has a population larger than Alaska, the island of Manhattan is not even visible on a typical map of the USA, while Alaska has a significant share of the screen space. Some approaches address this by sizing regions proportional to the data [8], but typically compromise the readability of the graphs significantly.

While choropleth maps are suitable for representing data that can naturally be aggregated into spatial units, such as vote share per state, they are not ideal to represent data where no meaningful spatial boundaries are present. Also, choropleth maps are problematic for representing data at multiple zoom levels, for example, on a country, state, city, and city block level. Pixel-based encodings, i.e., representing each data point on a map, do not suffer from these drawbacks, and are also well suited for interaction, e.g., based on data-driven filtering [29].

However, representing each data point on a map has obvious limitations when analyzing very large datasets, especially when real-time interaction is desired. The data somehow needs to be compressed, either as a subset, or by some statistical summarization. Common remedies for visualizing large data are sampling [7] and aggregation [9].

With respect to large spatial datasets, this has led to the development of a variety of research platforms including, inMens [15], Nanocubes [14] and Gaussian cubes [27]. These systems all provide a variety of ways to explore, interact, and analyze spatial datasets at scale. For interacting with such spatial data purely based on its density, a kernel density estimate is a necessary and often default tool; it is the statistical premise behind a heat map.

Related approaches are also taken by database projects, such as BlinkDB [1] and STORM [5] (shown in Figure 2). In these systems, random sampling of data is the core tool since it is efficient and preserves to some degree most relevant statistical properties of the data, including error in kernel density estimates [12].

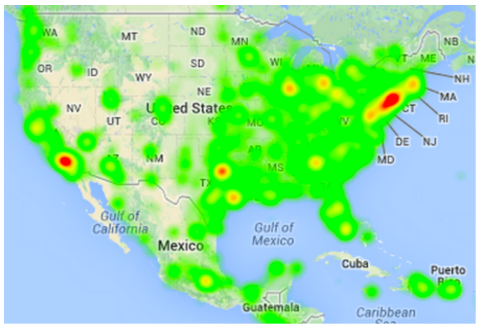

Fig. 2. Screenshot image from STORM [5] showing heatmap/KDE of tweet density in the USA.

Numerous other sampling schemes have been proposed to reduce the dataset size for visualization [13], [30]. However, these approaches do not directly address the preservation of kernel density estimates. Park et.al. [30] develop heuristics to optimize a measure related to the inverse of a KDE, and consider mainly data from long strands of data along road networks. Kim et.al. [13] focus on techniques for binned, one-dimensional data. Moreover, these approaches are considerably more complicated than the ones we consider and do not allow for efficient and stable updates in the parameters of the KDE.

3 Coresets Constructions

When tracking tweets or when analyzing crime in an area, a high frequency of such events in a sparsely populated area can be an important pattern to analyze further. If a subset has low error on average, but has locations with large deviations from the truth, analysis based on that subset can lead to both false positives and false negatives. In particular, the allowable error, that is the average error is small, but may create a region which itself appears to be much more dense than expected, thus prompting a user to spend valuable time chasing down a potential anomaly. And the similar effect can happen with a dense region (say around a population center) which appears in the low-average-error approximation to have far less density than expected.

On the other hand, an L_∞ error, as formalized in equation (1), ensures that *all regions* have small error. And as a result, if the L_∞ error is small, then we can be assured that no such false negatives or false positives as outlined above can be present in the approximate KDE. Both the advanced coresets techniques [31] and random sampling [12], can make such strong L_∞ guarantees and are thus suitable for efficient summaries for visual analytics.

We next describe the two most useful ways to create such L_∞-error KDE coresets:

1) A random sample Q of size $O((1/\varepsilon^2)\log(1/\delta))$ from a large set P creates a coreset for kernel density estimates with probability at least $1-\delta$ [12]. We refer such a method as RS. This can be implemented in $O(|P|)$ time.
2) There are several techniques to create coresets for kernel density estimates [4], [12], [16], [18], [31]. The one we use [31] (labeled Z-order, described below) results in a coreset of size $O((1/\varepsilon)\log^{2.5}(1/\varepsilon)\log(1/\delta))$, that succeeds with probability at least $1-\delta$, and runs in time $O(|P|\log|P|)$ time. This is roughly a $1/6$ of the size of the random sample technique. Note that other techniques [16], [18], can in theory reduce the coreset size to $O((1/\varepsilon)\log^{0.5}(1/\varepsilon))$; the Z-order method mimics this approach with something more efficient and with better constant factors, but a bit worse "in theory."

3.1 Coreset method

To generate the coresets, we use the two-dimensional technique based on space filling curves [31]. A space filling curve [2] puts a single order on two- (or higher-) dimensional points that preserves spatial locality. They have many uses in databases for approximate high-dimensional nearest-neighbor queries and range queries. The single order can be used for a (one-dimensional) B$^+$-tree, which provides extremely efficient queries even on massive datasets that do not fit in memory.

In particular, the Z-order curve is a specific type of space filling curve that can be interpreted as implicitly ordering points based on the traversal order of a quad tree. That is if all of the points are in the range $[0,1]^2$ (or normalized to be so), then the top level of the quad tree has 4 children over the domains $c_1 = [0,\frac{1}{2}]\times[0,\frac{1}{2}]$, $c_2 = [\frac{1}{2},1]\times[0,\frac{1}{2}]$, $c_3 = [0,\frac{1}{2}]\times[\frac{1}{2},1]$, and $c_4 = [\frac{1}{2},1]\times[\frac{1}{2},1]$. Each child's four children itself divide symmetrically, and so on recursively. Then the Z-order curve visits all points in the child c_1, then all points in c_2, then all points in c_3, and all points in c_4 (in the shape of a'Z'); and all points within each child are also visited in such a Z-shaped order. Thus given a domain containing all points, this defines a complete order on them, and the order preserves spatial locality roughly as well as a quad tree does. Usefully, the order of two points can be directly compared without knowing all of the data, so plugging in such a comparison operation, any efficient comparison-based sorting algorithm can be used to sort points in this order.

To generate the coreset based on the Z-order curve, set $k = O(\frac{1}{\varepsilon}\log^{2.5}\frac{1}{\varepsilon})$ and randomly select one point from each Z-order rank range $[(i-1)\frac{|P|}{k}, i\frac{|P|}{k}]$. The resulting set Q gives an ε-sample of KDE. Note that this asymptotic value of k is what is needed to provide an adversarial guarantee of small L_∞ error for *any* query point, and will work with *any* data set and *any* choice of kernel. In many cases, this error is indeed much small.

Moreover, this approach is oblivious to the parameters and type of the kernel density estimate (the type of kernel, the choice of bandwidth, the bitmap on which it is visualized), so it does not need to be updated if we change these parameters. For instance, in an interactive visualization using such a coreset that changes from a Gaussian kernel to an Epanechnikov kernel could use the exact same corset Q for the new choice of kernel and be assured that the same strong error guarantee would hold.

3.2 Pre-ordering points

The one parameter that will cause the above coresets to change is the error parameter ε, or symmetrically the size parameter k. Clearly if we want to increase the size of the coreset to decrease the error, or we want to decrease the size to increase the speed with which we can handle changes in resolution, we will need to change the coreset. It seems this would require completely redoing the entire coreset construction. This would take $O(|P|\log|P|)$ time to sort the points P, and even if the data is pre-sorted selecting the coreset would take at least another $O(|P|)$ time. Such a process would not meet the demands of an interactive visualization, so changing these parameters would not be feasible within such a system.

Rather we propose a more useful way to preprocess the data. In particular, we can reorder the original dataset P (from the Z-order to a different ordering) to what we call a *priority ordering*, so that the first k points in that order, which we call a *priority subset*, are precisely the points to choose as a coreset of size k. For instance, such a priority ordering can be created via random sampling: assign each point a random number, and sort on the points by these random numbers. This priority ordering has several enticing properties.

- The coreset construction only needs to be done once, and this can be done offline and in code that lives outside of an interactive visualization system. For instance, in our implementation, this is realized extremely efficiently in low-level C, but we have built our visualization in JavaScript, Canvas, and D3. This also makes the visualization system modular, separating the coreset construction technique, which only needs to provide a (priority) ordered set of points.
- If we increase the size of the coreset, the new larger coreset necessarily contains the old smaller one. This increases the stability of the result, since for instance increasing the size k by one point, only changes the coreset by 1 point. This means adjusting this parameter makes the visual interface more efficient and less jarring. Also, for small updates, it can allow for some caching in recomputing various quantities. In contrast, for a coreset Q_1 constructed directly from a Z-order, if the size parameter is changed slightly, we may recompute a new coreset Q_2 to satisfy this parameter change with no overlap with Q_1. This could cause the visualization to appear unstable and require that everything is completely recomputed.

For the Z-order approach, we can simply describe this priority reordering using a bit reversal. Given all of the points sorted by the Z-order, label each point as a binary number starting from $0\ldots00$, $0\ldots01$, $0\ldots10$, $0\ldots11$, …. Pad the dataset with dummy points so the total number is a power of 2; i.e., all binary numbers of a fixed length are included. Then reverse the order of the bits, so 101011 becomes 110101. Next randomize this by taking a random mask M and XORing the mask with all flipped numbers; basically

Input: Z-order index	1	2	3	4	5	6	7	x
binary representation	000	001	010	011	100	101	110	111
reverse bits	000	100	010	110	001	101	011	111
after random mask $M = 101$	101	001	111	011	100	000	110	010
new binary ordering index	6	2	8	4	5	1	7	3
priority ordering index	5	2	7	3	4	1	6	x

TABLE 1
An example demonstration of using bit-reversal to create a priority ordering. The first line is the input Zordering index, based on this sorted order. There are 7 points and one dummy point designated as x. The final line indicates the resulting priority ordering after removing the dummy point.

this randomly flips half of the bits. Then sort these points by these new binary numbers. Remove the dummy points, and this is the new order. This is illustrated in a small example in Table 1.

An alternative way of understanding this approach is via a binary tree. For the original data P, assign each point an index i based on the order in the Z-order. Then we construct a binary tree over these points based on this sorted order. Next, we fill up the binary tree with dummy points at the end of the ordering so that the size is a power of 2, and the binary tree is a perfectly-balanced tree; see Figure 3 for an example with 14 points.

Then we re-order these points by selecting points from the tree in a random way, so the number of selected points in each subtree is as balanced as possible; Algorithm 1 provides psuedocode for this priority re-ordering algorithm. At each step, at each internal node, we keep track of how many points have been selected from each subtree. If the two subtrees have the same number of selected points, choose one at random, and recurse. If the two subtrees have imbalanced counts of selected points, then recurse on the subtree (which will be `unmarked`) that has fewer selected points. This randomizes the process while ensuring that the selection is as balanced as possible with respect to the original ordering. The new priority order of the points $S = \langle s_1, s_2, \ldots, s_n \rangle$ is the order in which they are selected, ignoring dummy points. The dummy points ensure not to over-select from the existing points on the right subtree which can have fewer points than the left subtree.

Algorithm 1 priority reordering

```
1:  i = 1
2:  loop
3:    node = root
4:    while (node is not leaf node and not marked) do
5:      if (node→left and node→right are both unmarked)
        then
6:        generate a random number r from {0,1}
7:        if r = 0 then node = node→left and mark node
8:        if r = 1 then node = node→right and mark node
9:      else if (node→left is marked) then
10:       reset node→left as unmarked
11:       node = node→right
12:     else if (node→right is marked) then
13:       reset node→right as unmarked
14:       node = node→left
15:     else if (both children are marked) then
16:       return        [all nodes have been processed]
17:     if (leaf node and not dummy) then
18:       output node as s_i
19:       i = i + 1
```

Theorem 1. *The two constructions for a priority ordering on a set P are equivalent. For any priority subset of size k, where $k = O(\frac{1}{\varepsilon_k}\log^{2.5}\frac{1}{\varepsilon_k})$ for an appropriate chosen ε_k, provides a coreset Q such that $\|\mathrm{KDE}_P - \mathrm{KDE}_Q\|_\infty \leq \varepsilon_k$.*

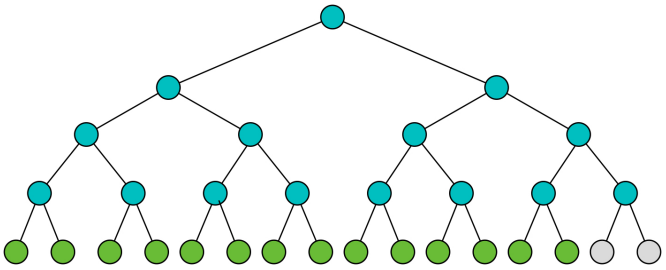

Fig. 3. Index tree of a dataset of 14 points (green). Dummy nodes are shown in grey.

Proof. We will describe how the bit-reversal approach is equivalent to the binary tree approach, and then show how the binary-tree approach preserves the coreset error guarantee. Let there be n points in P, and assume that $n = 2^m$ is a power of 2. As explained above, the dummy points ensure that under-filled trees are not oversampled.

By reversing the bits of each binary representation, the new reversed-bit ordering ensures that the first 2^ℓ points (for any integer ℓ) includes the $(1 + (n/2^\ell)j)$th point in the original order, for all integers $j = 0, 1, \ldots, 2^\ell - 1$. In the binary tree this is the leftmost point from each subtree at level ℓ. Since the first ℓ bits uniquely determine which of the 2^ℓ subtrees a point falls in, then a random mask keeps the subsets of points in the same subtree; however, it changes the order of the subtrees, and it changes the order of the points within the subtrees. It can be viewed as a way to randomly exchange the order of two children of all subtrees, with the same decision made for all subtrees at the same level. In particular, the left most point within a subtree is now random. So the selection of points is one from each subtree, but otherwise random – exactly as in the binary tree approach when k is a power of 2. When k is not a power of k (between 2^ℓ and $2^{\ell+1}$), then each of the first 2^ℓ points selected is also in a unique subtree at level $\ell + 1$, and the remaining points are chosen from other and distinct trees, at random, at level $\ell + 1$, as desired. So these methods are equivalent.

To see that the binary tree approach retains the coreset error guarantee, we again start with $k = 2^\ell$, a power of 2. In this case, there are again k easily defined intervals (defined by the original k subtrees at level ℓ), and we randomly select one point from each of them. To handle the case where k is not a power of two, we need to use the argument of why a coreset of size $O(1/\varepsilon)$ can be found in 1-dimension [16]; the 2-dimensional coreset construction [31] loses an extra factor $\log^{2.5}(1/\varepsilon)$ using the Z-order approximation, which carries through here unchanged. The 1-d argument works by saying for any KDE query can be reduced [12] to two one-sided interval $(-\infty, a]$ queries, with free parameter a, and each one-sided interval query can be shown to induce ε-error. This works since each query completely contains all points from P and Q in some intervals, no points in other intervals, and in only a single interval does it contain a partial subset of its points (it may or may not contain the point from Q). This interval contributes at most ε

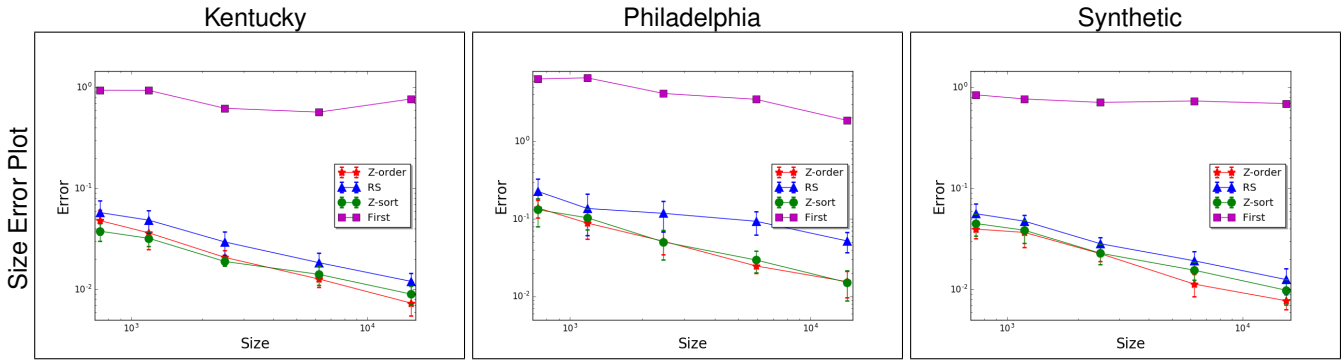

Fig. 4. Error comparison of random sample (RS), coreset (Z-order) and priority-ordered (Z-sort), or first points as presented in dataset (First). The marks represent the average value of ten random trials with the error bar indicating the standard deviation of the errors.

error, and hence since that is the only error, it is the overall error. What changes in our setting when k is not a power of 2, is that not all of the intervals are the same size; some smaller ones contain half the number of points from P as other ones. However, within each subtree, for each small interval on the left, up to 1, there are an equal number of small intervals on the right. Hence the error still comes down to the single interval being split, and its error is at most 2ε, so we get the same asymptotic bound. □

We note that random sampling can also be made into a priority ordering. We assign a random number $u_p \in [0,1]$ to each data point $p \in P$. Then we sort these points by this random value u_p, and this is the priority ordering. To see why this works, just observe that any priority subset k is a Poisson uniform random sample, based on the randomness in the u_p random values.

4 Experimental Evaluation

In our experiments we use two large real datasets and one synthetic dataset. The first dataset (*Kentucky*) is of size 199,163 and consists of the longitude and latitude of all highway data points from OpenStreetMap data in the state of Kentucky. The second dataset (*Philadelphia*) contains 683,499 geolocated data points; it consists of the longitude and latitude of all crime incidents reported in the city of Philadelphia by the Philadelphia Police Department between 2005 and 2014.

Our *Synthetic* dataset mimics a construction of Zheng and Phillips [32] meant to create density features at many different scales using a recursive approach inside a unit square $[0,1]^2$. The dataset contains 532,900 data points. At the top level it generates 4 points $p_1 = (0,0), p_2 = (0,1), p_3 = (1,0), p_4 = (1,1)$. We recurse into 9 new rectangles by splitting the x- and y-coordinates into 3 intervals each and taking the cross-product of these intervals. The intervals are defined non-uniformly, splitting the x-range (and y-range) into pieces $[0,0.5]$, $[0.5,0.8]$, and $[0.8,1.0]$. We also add 4 new points at $(0.5,0.5)$, $(0.5,0.8)$, $(0.8,0.5)$, and $(0.8,0.8)$ to the created dataset. In recursing on the 9 new rectangles we further split each of these and add points proportional to the length of their sides, and at the same non-uniform proportions. Since each rectangle is split into more pieces in the large part of its range, and each piece gets proportionally the same number of points, there are more points in the larger value ranges. This pattern is recursively true in each range, and can be seen in Figure 5.

4.1 Effective of Coreset Method

To guarantee ε-error coresets we require $O((1/\varepsilon)\log^{2.5}\frac{1}{\varepsilon})$ size, while random sampling requires $O(1/\varepsilon^2)$ size. In other words, coresets with the same error as random sampling can be about a square root of the size. While these theoretical bounds are useful guidance for effectiveness of these techniques, we also demonstrate them empirically in Figure 4 using the three datasets. In general we consider Error $= \text{KDE}_P - \text{KDE}_Q$, where P is the original coreset and Q the sample. Visually we will also show the relative error $\frac{\text{KDE}_P - \text{KDE}_Q}{\text{KDE}_P}$. We evaluate coresets using the Z-order method and the priority ordering (Z-sort), which reorders the Z-order coreset. Our baseline algorithms are random sampling (RS) and simply use the points in the order they exist in the original dataset (First). We consider subsets from size 700 to 15,000. We evaluate the maximum error by randomly evaluating the KDEs at 10,000 random points in the range of the image, as advocated here [32]. Our experiments fix an appropriate bandwidth for all the samples and the original dataset. The plots show the average value of ten random trials with the error bar indicating the standard deviation of the errors. The method of simply using the points in the order they exist in the original dataset doesn't have error bars since it is a deterministic baseline method. From the experiment, we can see that only choosing the first few points from the original dataset gets the highest error – dramatically worse than any other approaches. For other methods, as the number of points in the sample increases, the error decreases fairly consistently. For the same size of sample points, Z-order and Z-sort perform much better than the random sampling method; note that these are log-log plots. For instance on the Kentucky data set, at size about 2,500, the two coreset approaches result in error about 0.02, while RS generates 0.035 error. Another interpretation, is that RS requires size about 6,300 samples to observe 0.02 which the coresets obtained with a 40% of the sample size. While these numbers vary across data sets, and show some random variation, this sort of pattern in error ratios is typical.

4.2 Visual Demonstration on Data

To demonstrate the advantage of the coreset method over the random sampling method, we show the visualizations of KDEs on these three datasets in Figure 5. In this figure we show the KDE of the original dataset, the coreset, and a random sample (we use Z-sort coreset as the representative of coreset methods due to not too much difference between Z-order and Z-sort method from Figure 4). Since simply getting the first few points from original dataset are getting much worse error, it is not necessary to visually evaluate it. We set the size of the coreset in Kentucky to 7,675, in Philadelphia to 7,675, and in Synthetic to 69,077. A transfer function colors each pixel with respect to the largest $\text{KDE}(x)$ value

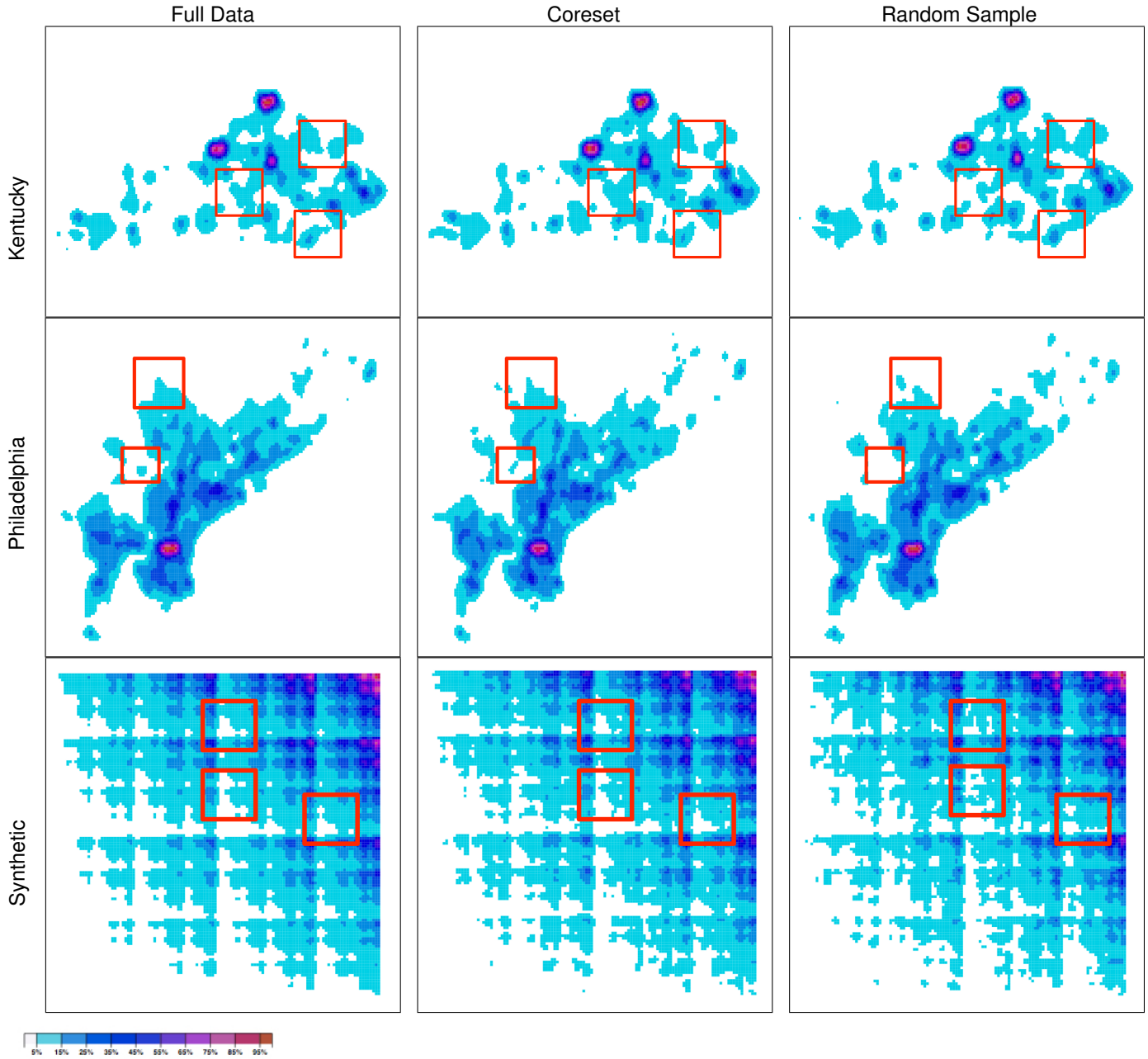

Fig. 5. Comparison of ground truth KDE (left), coreset KDE (middle), random sample KDE (right), on three datasets. Regions of high error in the random sampling are highlighted with red frames across all conditions.

observed in the full dataset (a dark red), transitioning to a light blue and then white for values less than 5% of this value.

The high-level structure for both the coreset and random sample visualizations are preserved in each case; however, for each dataset there are many subtle differences where the random sample captures some area incorrectly. We highlight a few of these differences in red boxes in Figure 5.

Another way to understand the error is by directly plotting the error values, as we have done in Figure 6. We plot both the absolute and relative error. Here the transfer function is normalized based on the largest difference observed for each dataset and error measure, but held the same between conditions, to allow for the direct comparison of coreset error and random sample error. The resulting color scale is a diverging color map: when the sample has a larger value than the true dataset, the area is shown in increasingly saturated shades of red; and when the true dataset has a larger value, it is shown in increasingly saturated blue. When similar, white is shown. We visually observe darker colors (and more error) for random sampling than with coresets.

Note that the theory has guarantees only for the additive error; e.g., smaller for coresets than random sampling. But we also plot the relative error to highlight relative differences that may have more effect both in quantitative anomaly detection and observed visual artifacts. Indeed we observe larger relative error for random sampling as well, compared to the coreset.

5 Thresholding Iso-Levels

A common pattern for interactive data visualization is to show an overview of all data and enable analysts to zoom in to investigate regions of interest. For geospatial data, nano-cubes is a recent system that delivers such an experience [14] for large datasets.

A critical aspect of such overviews is hence that they faithfully represent the data in any region above some density of interest,

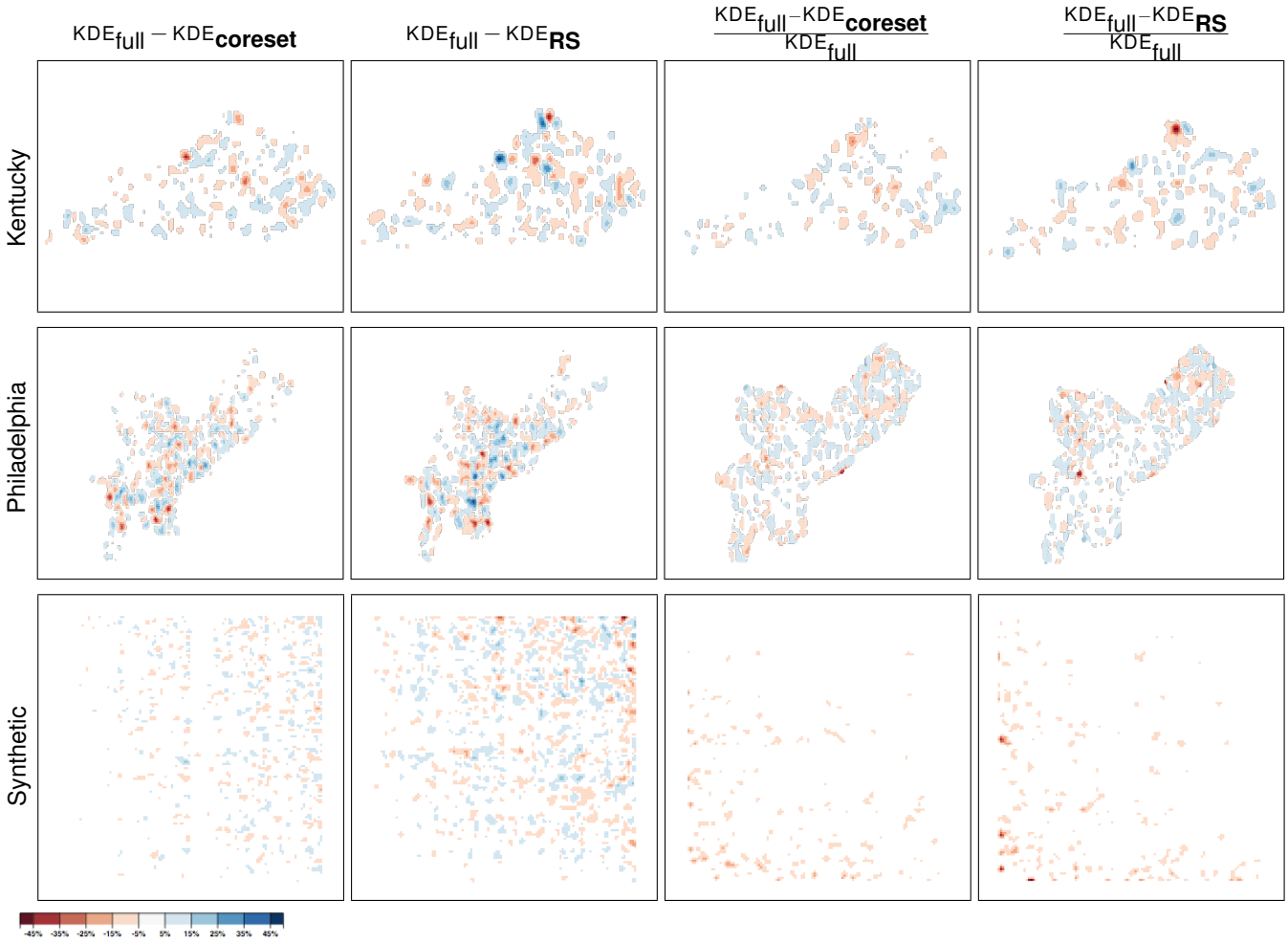

Fig. 6. Comparison of the differences between original KDEs and coreset KDEs (first column) and the difference between original KDEs and random sampling KDEs (second column). The last two columns show the corresponding relative differences.

i.e., that wherever there is data above a threshold there should be a visible mark that can be investigated in detail. In fact there is a well-developed theory around random sampling regarding this property called and ε-net. It says if we sample $O((1/\varepsilon)\log(1/\varepsilon))$ points, then any geometric region (like a circle or rectangle) with more than ε-fraction of the points (a density value larger than ε) will contain at least one point [11].

However, this desire to show **all** possibly interesting features runs into another problem. If we set the minimum threshold for coloring pixels as non-white too low, then the visualization ends up displaying a lot of noise. That is, there may be regions which should have low (or almost 0) density, which are shown with a visible mark. In contrast to the other sampling results mentioned above (which require larger, $O(1/\varepsilon^2)$-size, samples), the guarantees for ε-nets provide no protection against false positives. Moreover, simple random sampling is used heavily in many big data systems, such as STORM [5].

To address this problem, we will build on a more recent adaption of ε-nets specific to kernel density estimates, called (τ,ε)-*nets* [19]. This coreset $Q \subset P$ ensures that for any point $x \in \mathbb{R}^d$ such that $\text{KDE}_P(x) \geq \varepsilon$, there exists a point $q \in Q$ such that $K(x,q) \geq \tau$. That is, for any query point x above some density threshold ε, there is some *witness* point in the coreset point $q \in Q$ that is nearby (its similarity $K(x,q)$, is at least τ). Although such guarantees can be derived from the coresets we discussed earlier, this (τ,ε)-net only requires a random sample of size $O(\frac{1}{\varepsilon-\tau}\log\frac{1}{\varepsilon-\tau})$, which for $\tau=\varepsilon/2$ is $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$, i.e., roughly the same as the previous and slightly more complex coreset.

So how can we use this idea of a (τ,ε)-net to aid in choosing a color threshold of our transfer function? One approach is to make that threshold adaptive. Our proposed method will only color low-density regions (at some threshold taking the place of τ) if they are close to some higher density region (defined by another parameter ε). This means spurious regions far from the main data will not be illustrated as they are likely noise. But near a high density region our visualization will draw the lowest density layer. Data near a high-density regions is less likely to be noise, and so our method displays this part as accurately as possible.

In detail, we implement this using two values. The first value ε (= percentage) is the minimum observed value to represent a "high density region." The second value r (= radius) is the minimum distance an interesting point must be to a high-density region. Then if a pixel x is not within a distance r of some other pixel y such that $\text{KDE}_Q(y) \geq \varepsilon$, then it is not drawn, as if there is no appreciable density there. If $\text{KDE}_Q(x) \geq \varepsilon$ or if x is within distance r of some pixel y such that $\text{KDE}_Q(y) \geq \varepsilon$, then it will be drawn as specified by the transfer function.

Figure 7 demonstrates this approach on our three datasets. For each dataset, it shows the kernel density estimate for the full data, a random sample of that data, and a de-noised version of the random sample. In the random sample, some anomalous regions appear due to sampling noise (examples are highlighted with red circles in Figure 7), which disappear in the de-noised version. The denoised

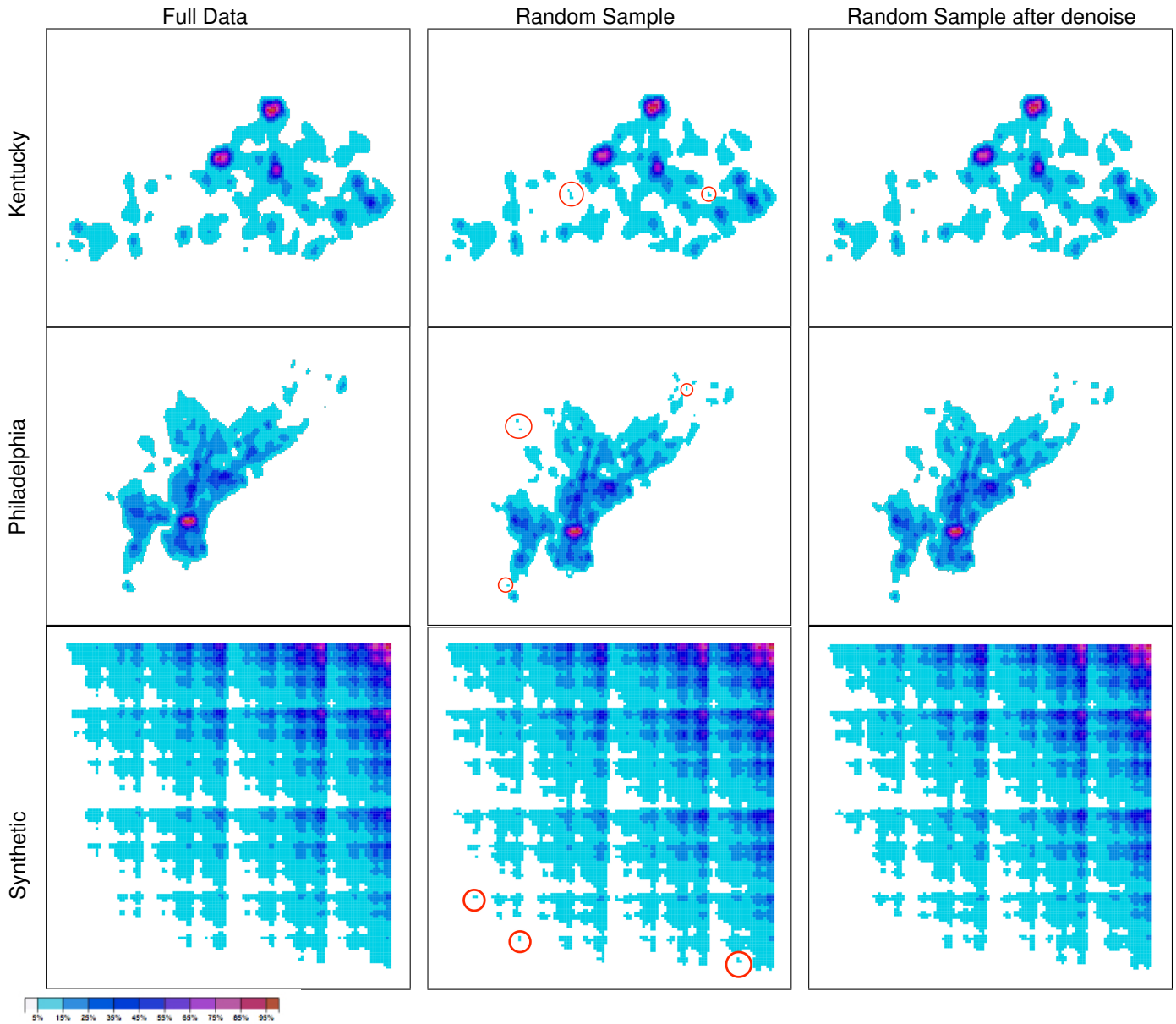

Fig. 7. Visualization of random sample KDEs of all three datasets. Showing all isolevels of a random sample (middle) shows false anomalous regions, circled, compared to ground truth (left). After zapping process, (right) still preserves the rough shape of the data–enough to know where to explore more–without any of the false positive regions.

version is a more accurate representation of the original data as it does not show various anomalous bumps of density.

6 System

To demonstrate our approach and compare it to both, ground truth and random sampling we build an interactive system to display kernel density estimates of very large spatial data. It enables analysts to interactively explore such large data while avoiding false positives. To enable a direct comparison of various approaches, we show two windows showing the same dataset using different methods—the KDE of the full dataset, coreset KDE, random sample KDE, coreset error, coreset relative error, random sampling error and random sampling relative error. Analysts can specify the error threshold ε, based on which the system automatically generates a coreset or a random sample based on ε.

Zooming and panning is synchronized between views, so that analysts can navigate and compare the views at various scales and positions. To provide geospatial context, the KDE visualization is rendered on top of a customized Google Map widget, which shows the geographic features as grayscale to avoid interference with the colors used to display the KDE.

We also provide various color maps options from ColorBrewer [10]. We allow users to dynamically change the choice of color map, and its scaling within the colorbar.

6.1 Interactive De-noising

When applying the de-noising process that alters the low end of the color scale with ε, τ-nets, we found that the choice of these parameters can be difficult for a user to select. To address this, we designed a feature where an analyst can highlight a region that appears to be an anomalous region, and the system will suggest a pair of minimal percentage and radius values that can be set to remove the noise in that region. Figure 8 illustrates this process within our system.

Analysts select an isolated regions to get rid of, then a tips message will give the suggestions of “percentage” and “radius”,

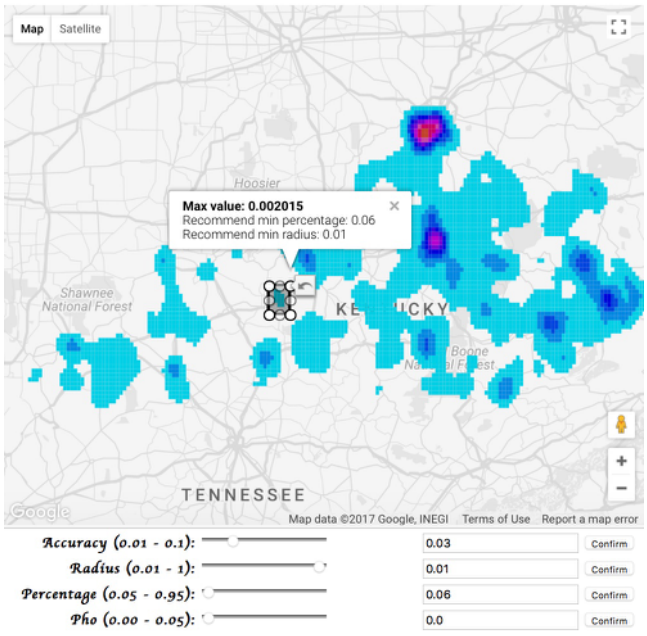

Fig. 8. Illustration of the interactive de-noising process. Analysts select a region in the visualization they suspect to contain an artifact. The algorithm suggests parameters that can be used to remove that artifact and applies them to the input fields.

so $\tau =$ "percentage" $\times$ the largest KDE within "radius" of the objective point. These values can then be applied to the de-noising process, eliminating the noisy spot and others like it.

We suggest to users to attempt this with a few isolated dots and see the effects. Then if desired, they can also manually tune these parameters directly and quickly see the effect.

6.2 Implementation

The front end of our technology demonstration is implemented in HTML/JavaScript and uses D3 for axis, scales and user interface elements, Canvas for the rendering of the KDEs and the Google Maps API for the background maps.

The backend that generates the coresets is an extension of work from SIGMOD 2013 [31] and is written in C. This can take any large spatial dataset as a text file, a error parameter ε, and output a coreset. We modify this to generate a priority reordering of the entire dataset so that every initial subset of the data is a coreset, with error parameter effectively decreasing as the chosen coreset size increases. This process is also written in C, and generates a text file sufficient for the HTML/Javascript to use as its input.

The implementation of the visualization system (https://github.com/SayingsOlly/kernel_vis_d3) and the back-end code (http://www.cs.utah.edu/~jeffp/students/kde/kde.html) is available under the BSD 3-clause license. We invite others to download and interact with it.

7 Discussion and Limitations

We study the specific but ubiquitous visualization tool of kernel density estimates, with the goal of how best to integrate them into a large-scale visualization system — specifically those making the increasingly common design choice to approximate massive datasets. In this context we demonstrate that coresets provide better and more efficient estimates than simple random sampling. We also develop a new way to preprocess the coresets so that their size resolution can be easily updated without redoing expensive computation. Additionally, we introduced a new tool for dealing with spatial noise at low densities — a common nuisance that distracts the user to explore potential outliers which are not present in the full dataset. This provides an easy way to "zap" these unfortunate events with a simple rule that will apply to all similar visual (but not statistical) anomalies. Our simple system demonstrates the usefulness of all of these insights through interaction with real and synthetic datasets.

Our interactive visualization system, however, is designed as a prototype to demonstrate the strengths of the underlying technique and is not designed to be a fully-fledged geospatial data analysis system. Several improvements with respect to data loading and usability are conceivable to make the system useful for actual analysis tasks. We would also like to explore the effects of different coreset constructions (e.g., [4], [16]) and types of kernels other than Gaussians (e.g., Laplace or Epanechnikov).

With any interactive visualization tool, it is important to be cognizant of the potential for *visual p-hacking* [28]: where a user tweaks the visual parameters until he/she finds the interpretation of the data he/she wants to see, but unwittingly just discovers artifacts of data noise. Our technique moderates this by allowing users to identify noise (perhaps using expert knowledge) and remove it. Moreover, it enforces the same pruning criteria for all isolated parts of the dataset, so it is not possible to design pruning criteria separately for different areas — an easy way to overfit.

In general, one should complement this with a query-and-filter strategy to verify abnormal or interesting aspects of the data beyond just the visual patterns. Our tool is meant to help users quickly determine where to take these closer looks.

7.1 Evaluation Strategies

We considered various strategies to evaluate our methods, including a quantitative comparison to baseline algorithms and an evaluation with human subjects. One potential human subjects study design would be to let users pick whether our method or an alternative, such as random sampling, is closer to the ground truth. We ultimately decided against this design, because tangible differences with respect to the ground truth are readily apparent in our demonstration (see Section 4.1), and in the provided difference images (see Figure 6). Moreover, our quantitative evaluation in Section 4.2 provides a clear and quantitative measurement of the error improvement of the coreset techniques versus common baselines. An alternative question to ask would be whether the techniques presented in this paper have an impact on correctness and performance of a human analysts in real analysis scenarios. Such a study design, however, is beyond the scope of this article.

8 Conclusion and Future Work

We have demonstrated the use of coresets for kernel density estimates, ways to preprocess them for easy parameter updates, and how to prune a certain type of low-density noise. We believe these are techniques that should be integrated into many visualization systems for large spatial datasets.

However, our system itself is only a prototype. We would like to actually map these ideas into more complex systems (e.g., nanocubes [14] or STORM [5]) which already deal with and approximate various datasets and allow for other richer types of interactions.

ACKNOWLEDGMENTS

Thanks for NSF CCF-1350888, IIS-1251019, ACI1443046, CNS-1514520, CNS-1564287 and NIH U01 CA198935.

REFERENCES

[1] S. Agarwal, B. Mozfari, A. PAnda, H. Milner, S. Madden, and I. Stoica. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[2] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer. Space-filling curves and their use in the design of geometric data structures. *Theoretical Computer Science*, 181:3–15, 1997.

[3] O. Bachem, M. Lucic, and A. Krause. Practical coreset construction for machine learning. Technical report, arXiv: 1703.06476, 2017.

[4] Y. Chen, M. Welling, and A. Smola. Supersamples from kernel-herding. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2010.

[5] R. Christensen, L. Wang, F. Li, K. Yi, J. Tang, and N. Villa. STORM: Spatio-Temporal Online Reasoning and Management of large spatio-temporal data. In *Proceedings of 34th ACM SIGMOD International Conference on Management of Data*, 2015.

[6] W. S. Cleveland and R. McGill. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

[7] A. Dix and G. Ellis. By chance: Enhancing interaction with large data sets through statistical sampling. In *Proceedings of the ACM Conference on Advanced Visual Interfaces (AVI '02)*, pages 167–176. ACM Press, 2002.

[8] M. T. Gastner and M. E. J. Newman. Diffusion-based method for producing density-equalizing maps. *Proceedings of the National Academy of Sciences of the United States of America*, 101(20):7499–7504, May 2004.

[9] J. Goldstein and S. F. Roth. Using aggregation and dynamic queries for exploring large data sets. In *CHI 1994: Proceedings of the SIGCHI*, pages 23–29, New York, NY, USA, 1994. ACM.

[10] M. Harrower and C. A. Brewer. Colorbrewer.org: An online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40:27–37, 2003.

[11] D. Haussler and E. Welzl. epsilon-nets and simplex range queries. *Disc. & Comp. Geom.*, 2:127–151, 1987.

[12] S. Joshi, R. V. Kommaraju, J. M. Phillips, and S. Venkatasubramanian. Comparing distributions and shapes using the kernel distance. *Proceedings 27th Annual Symposium on Computational Geometry*, 2011.

[13] A. Kim, E. Blais, A. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. In *Proceedings VLDB Endowment*, 2015.

[14] L. Lins, C. Scheidegger, and J. Klosowski. Nanocubes for real-time exploratioin of spatiotemporal datasets. *IEEE TVCG*, 2013.

[15] Z. Liu, B. Jiang, and J. Heer. inmens: Realt-time visual querying of big data. In *Eurographics Conference on Visualization*, 2013.

[16] J. M. Phillips. eps-samples for kernels. *Proceedings 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2013.

[17] J. M. Phillips. Coresets and sketches. In *Handbook of Discrete and Computational Geometry*, chapter 49. CRC Press, 2016.

[18] J. M. Phillips and W. Tai. Improved coresets for kernel density estimates. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2018.

[19] J. M. Phillips and Y. Zheng. Subsampling in smoothed range spaces. In *Algorithmic Learning Theory*, pages 224–238. Springer, 2015.

[20] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley, 1992.

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

[22] B. Speckmann and K. Verbeek. Necklace Maps. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):881 –889, nov.-dec. 2010.

[23] M. Sun, D. W. Wong, and B. J. Kronenfeld. A Classification Method for Choropleth Maps Incorporating Data Reliability Information. *The Professional Geographer*, 67(1):72–83, Jan. 2015.

[24] C. Sung, D. Feldman, and D. Rus. Trjacetory clustering for motion prediction. In *IEEE/RSJ International Converence on Intelligent Robots and Systems*, 2012.

[25] C. Tominski and H.-J. Schulz. The Great Wall of Space-Time. In *Proceedings of the Workshop on Vision, Modeling and Visualization (VMV'12)*, 2012. to appear.

[26] C. Tominski, H. Schumann, G. Andrienko, and N. Andrienko. Stacking-Based Visualization of Trajectory Attribute Data. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2565 –2574, Dec. 2012.

[27] Z. Wang, N. Ferreira, Y. Wei, A. Bhaskar, and C. Scheidegger. Gaussian cubes: Real-time modeling for visual exploration of large multidimensional datasets. In *IEEE InfoVis*, 2016.

[28] H. Wickham, D. Cook, H. Hofman, and A. Buja. Graphical inference for infovis. *IEEE Transactions of Visualization and Computer Graphics*, 16:973–979, 2010.

[29] C. Williamson and B. Shneiderman. The Dynamic HomeFinder: Evaluating Dynamic Queries in a Real-estate Information Exploration System. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '92, pages 338–346, New York, NY, USA, 1992. ACM.

[30] B. M. Yongjoo Park, Michael Cafarella. Visualization-aware sampling for very large databases. In *IEEE International Conference on Data Engineering*, 2016.

[31] Y. Zheng, J. Jestes, J. M. Phillips, and F. Li. Quality and efficiency in kernel density estimates for large data. In *Proceedings ACM Conference on the Management of Data (SIGMOD)*, 2012.

[32] Y. Zheng and J. M. Phillips. L_infty error and bandwidth selection for kernel density estimates of large data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1533–1542. ACM, 2015.

Yan Zheng is a Research Scientist of Visa Research in Data Analytics. Before joining Visa, she received a PhD from University of Utah in computer science. Her primary research interests are Data Analytics, Machine Learning and Deep Learning

Yi Ou is a Software Engineer at Expedia. Before joining Expedia he was a master student at the University of Utah and received his MS in 2017. His research interests mainly are data mining, data visualization and machine learning.

Alexander Lex is an Assistant Professor of Computer Science at the Scientific Computing and Imaging Institute and the School of Computing at the University of Utah. Before joining Utah he was a lecturer and a post-doctoral visualization researcher at the Harvard School of Engineering and Applied Sciences. He received his PhD from the Graz University of Technology in 2012. His primary research interests are data visualization, especially applied to molecular biology, and human computer interaction.

Jeff M. Phillips is an Associate Professor in the School of Computing at the University of Utah. He received a BS degree in computer science and the BA degree in math from Rice University, and a PhD from Duke University in computer science. He was an NSF Graduate Research Fellow at Duke University, an NSF CI postdoctoral fellow at the University of Utah, and received an NSF CAREER Award in 2014.