Visualization of Big Spatial Data using Coresets for Kernel Density Estimates

Yan Zheng* Visa Research
Yi Ou† Expedia, Inc.
Alexander Lex‡ University of Utah
Jeff M. Phillips§ University of Utah

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.

Keywords: 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 aggregate it using a kernel density estimate [16, 15] (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. For a formal definition, we first require a kernel $K : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$; we will use the Gaussian kernel $K(p,x) = e^{-\|p-x\|^2}$. Then, given a planar point set $P \subset \mathbb{R}^2$, the kernel density estimate is defined at any query point $x \in \mathbb{R}^2$ as

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

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 several contexts. Using this function summarizes the data, and avoids the over-plotting and obfuscation issues demonstrated in Figure 1(left). However, just computing $\text{KDE}_P(x)$ for a single value x requires $O(|P|)$ time. 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.

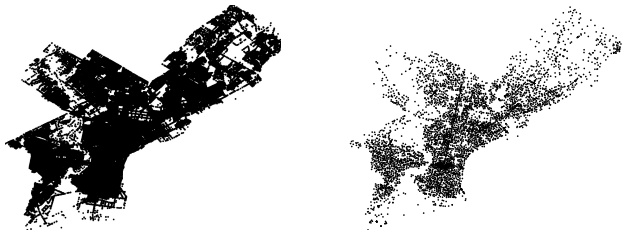


Figure 1: Crimes from 2006 to 2013 in Philadelphia, the full dataset (left) with 0.7 million points and a coreset (right) with only 5300 points.

Towards alleviating these issues, we propose to use **coresets** for KDEs. In general, a *coreset* Q is a 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) and Q constitutes some carefully placed way-points [18]. 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 [13, 3].

In particular, a coreset for a kernel density estimate is a subset $Q \subset P$ [12, 22], with $|Q| \ll |P|$ so that for some error parameter ε

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

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 4, 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.

*e-mail: yanzh.cs@gmail.com; Much of this work was completed while at the University of Utah.
†e-mail: olly93219@outlook.com; Much of this work was completed while at the University of Utah.
‡e-mail: alex@sci.utah.edu
§e-mail: jeffp@cs.utah.edu

2 Related Work

Visualizing large spatial datasets is a challenge attracting a lot of attention among the visualization community. This has led to the development of a variety of research platforms including Polaris [17], inMens [11], Nanocubes [10] and Gaussian cubes [19]. These systems all provide a variety of ways of to explore, interact, and analyze spatial datasets. For interacting with such spatial data purely based on its density, a kernel density estimate is a necessary and often the default tool; it is the statistical premise behind a heat map.

Another common theme among visualization systems for large data is that in order to allow real-time interaction, every single data point cannot be rendered. The data somehow needs to be compressed, either as a subset, or by some statistical summarization. This trend denominates efficiency and scalability focused database projects, such as BlinkDB [1] and STORM [5]. In these systems, random sampling of data is the core tool since it can be done efficiently and preserves to some degree most relevant statistical properties of the data.

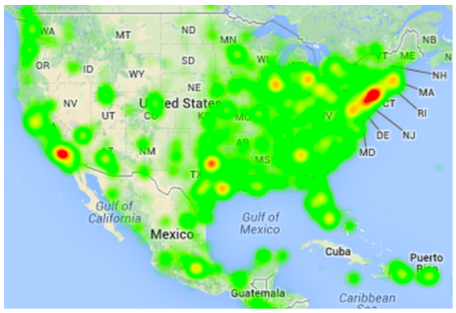


Figure 2: Screenshot image from STORM [5] showing heatmap/KDE of tweet density in the USA.

Numerous other sampling schemes have been proposed to reduce the dataset size for visualization [21, 9]. However, these approaches do not directly address the preservation of kernel density estimates. Park et.al. [21] 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. [9] 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 negatives. This is why the L_∞ error, as in equation (1) is the right way to measure accuracy. Both coresets techniques for KDEs [22] and random sampling can make such guarantees, but the ones for coresets are stronger.

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 $1-\delta$ [8]. 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 [8, 22, 12, 4]. The one we use [22] (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 square-root of the size of the random sample technique. Note that other techniques [12], 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."

While these theoretical bounds are useful guidance for effectiveness of these techniques, we also demonstrate them empirically in Figure 3 using Open Street Map Utah highway data. We observer that indeed Z-order produces a coreset roughly a square-root of the size of the one produced by RS for the same observed error.

Size	RS Err	Coreset Err
830	0.035	0.01
1890	0.023	0.005
5000	0.014	0.002
10000	0.01	0.001

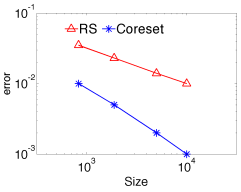


Figure 3: Error comparison of random sample (RS) and coresets.

3.1 Coreset method

To generate the coresets, we use the two-dimensional technique based on space filling curves [22]. 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 generally preserves spatial locality 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 approach is oblivious to the parameters in 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.

3.2 Pre-ordering points

One downside of the above method, is that if we would like to change the resolution of the coreset, that is increase or decrease its accuracy by increasing or decreasing its size, we need to repeat much of the computation. Sorting the $|P|$ points takes $O(|P|\log|P|)$ time, and selecting a coreset from the sorted list would take $O(|P|)$ time under most implementations and ways of preprocessing the data.

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*,

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 describes 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.

so that the first k points in that order 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 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 to illustrate it using a binary tree. For the original data P, we give each point an index i based on the order of the points 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 4 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 purpose of the dummy points it to make sure that we don't over-select from the existing points on the right subtree if they have fewer points than the left subtree.

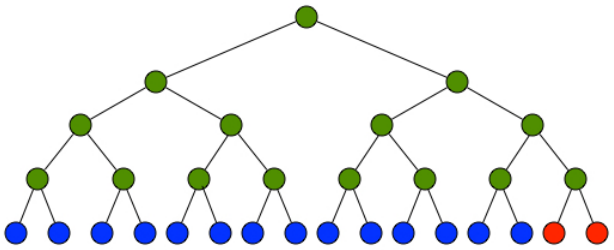


Figure 4: Index tree of a dataset of 14 points (blue). Dummy nodes are shown in red.

Algorithm 1 priority reordering

```
1:  i = 1
2:  loop
3:      node = root
4:      while (node→left and node→right are both unmarked) 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
```

3.3 Comparing Coresets with Random Sampling

To guarantee ε-error coresets 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 (see Figure 3). We will compare two kind of errors: general error and relative error between original data KDE and coreset KDE as well as original data KDE and random sample KDE. Suppose the original dataset is P, coreset or random sample is defined as Q, then absolute error is defined as $\text{KDE}_P - \text{KDE}_Q$ and relative

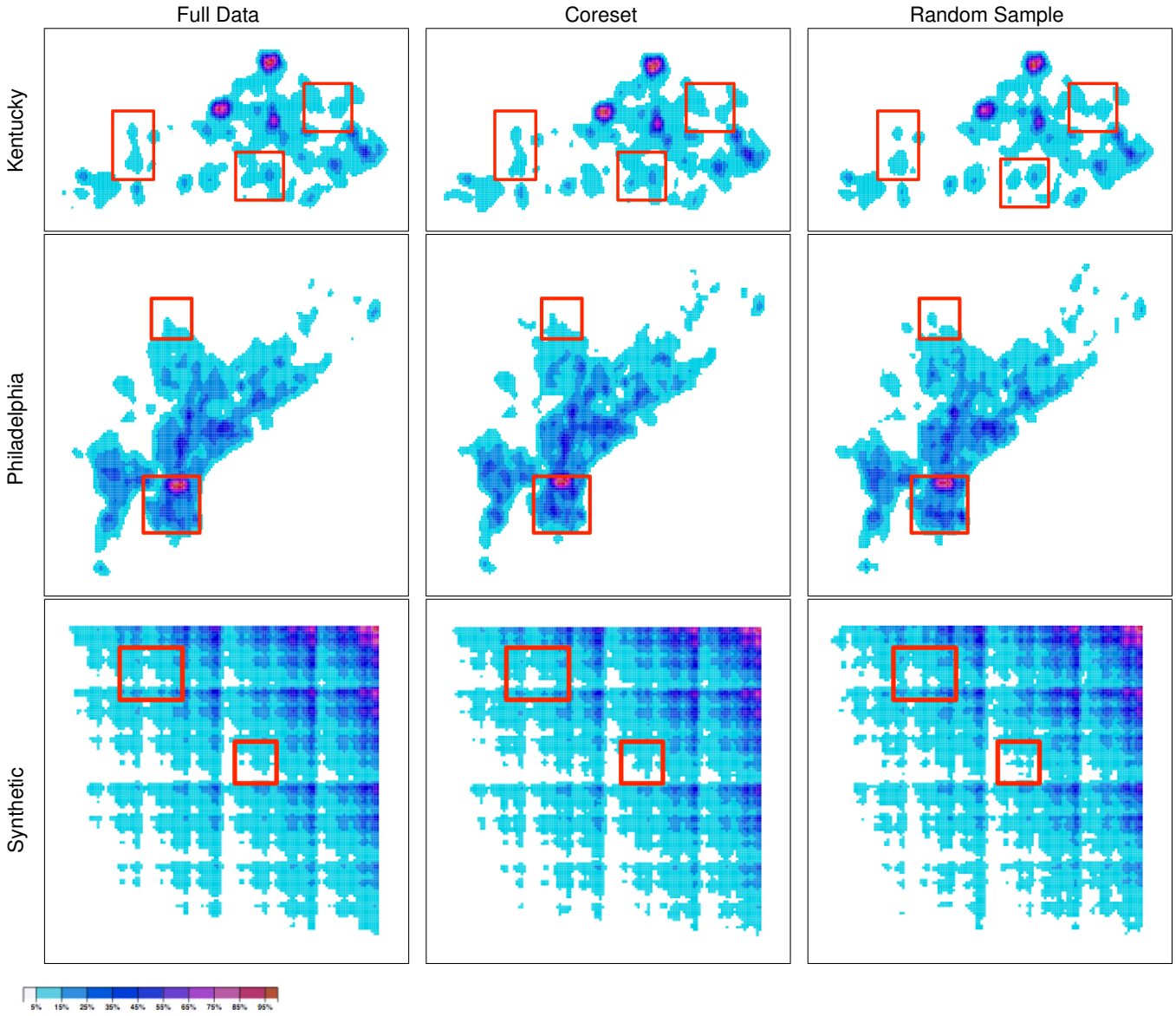


Figure 5: Comparison of ground truth KDE (left), coreset KDE (middle), random sample KDE (right), on three datasets. Regions of high error in the random sampling are highlighted with red frames across all conditions.

error is defined as $\frac{\text{KDE}_P - \text{KDE}_Q}{\text{KDE}_P}$.

3.3.1 Datasets

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 [23] 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.

3.3.2 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 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 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 have highlighted a few of these differences across the 3 visualizations 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 for the same dataset. 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 coreset or random 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 smaller value, the area is shown in increasingly saturated blue. When they are similar white is shown. We can visually observe darker colors (and hence more error) for the random sampling approach than the coreset approach.

Note that the theory specifically guarantees the additive error should be smaller for coresets, but we plotted the relative error as well since it seems that such relative differences may have more effect both in quantitative anomaly detection as well as in an observed visual artifact. Indeed we observe larger relative error for random sampling as well.

4 Avoiding Error when Thresholding Iso-Levels

A common pattern for interactive data visualization is to show an overview of all the data and then enable analysts to zoom in to investigate regions of interest. For geospatial data, nano-cubes is a recent system that delivers such an experience [10] 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, 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 [7].

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

5 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 [6]. We allow users dynamically change the choice of color map, and its scaling within the colorbar (Figure 9).

5.1 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 the a pair of minimal percentage and radius values that can be set to remove the noise in that region. Figure 10 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", so $\tau =$ "percentage" $\times$ the largest KDE within "radius" of the objective point. These values can then be applied to the parametrization of the de-noising process, eliminating the noisy spot and other 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.

5.2 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.

Figure 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.

The backend that generates the coresets is an extension of work from SIGMOD 2012 [22] 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/~yanzheng/kde/`) is available under the BSD 3-clause license. We invite others to download and interact with it.

6 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 event 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 dataset.

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., [12, 4]) 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* [20]: where a user tweaks the visual parameters until he/she finds the interpretation of the data he/she wants to see, but unwittingly has just discovered artifacts of the noise in the data. 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 compliment this with a query-and-filter strategy to verify abnoromal 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.

Figure 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.

7 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 [10] or STORM [5]) which already deal with and approximate various datasets and allow for other richer types of interactions.

We also believe coresets [13, 3] can potentially be a very useful tool for efficiently visually interacting with many types of massive datasets. We hope to explore more of these applications in the future.

Acknowledgements

Thanks to support by NSF CCF-1350888, IIS-1251019, ACI-1443046, 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.

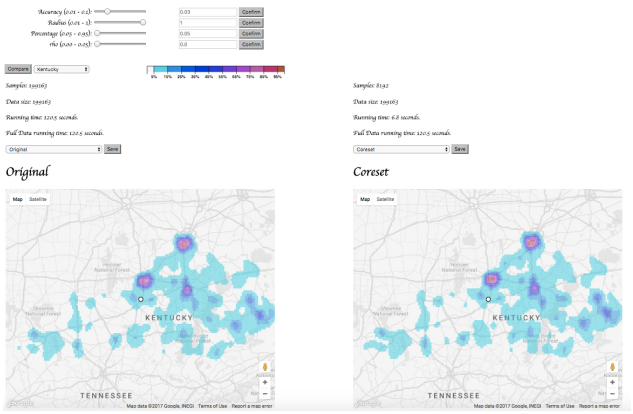


Figure 8: A snapshot of the system.

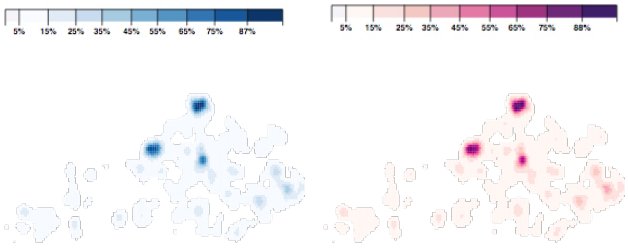
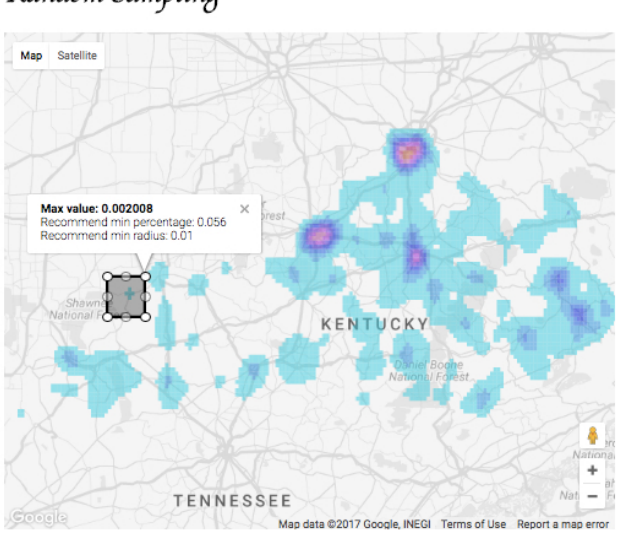


Figure 9: Visualization of KDEs with different colorbars.



(a) KDE of a random sampling and selected zapping area.

(b) Input the selected parameters.

Figure 10: 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 (a) and applies them to the input fields (b).

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

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

[8] 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.

[9] 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.

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

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

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

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

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

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

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

[17] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions of Visualization and Computer Graphics*, 8(1), 2002.

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

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

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

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

[22] 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.

[23] 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.