# SLAM: Efficient Sweep Line Algorithms for Kernel Density Visualization

Tsz Nam Chan
Hong Kong Baptist University
edisonchan@comp.hkbu.edu.hk

Leong Hou U
University of Macau
SKL of Internet of Things for Smart City
ryanlhu@um.edu.mo

Byron Choi
Hong Kong Baptist University
bchoi@comp.hkbu.edu.hk

Jianliang Xu
Hong Kong Baptist University
xujl@comp.hkbu.edu.hk

## ABSTRACT

Kernel Density Visualization (KDV) has been extensively used in a wide range of applications, including traffic accident hotspot detection, crime hotspot detection, disease outbreak detection, and ecological modeling. However, KDV is a computationally expensive operation, which is not scalable to large datasets (e.g., million-scale data points) and high resolution sizes (e.g., $1920 \times 1080$). To significantly improve the efficiency for generating KDV, we develop two efficient Sweep Line AlgorithMs (SLAM), which can theoretically reduce the time complexity for generating KDV. By incorporating the resolution-aware optimization (RAO) into SLAM, we can further achieve the lowest time complexity for generating KDV. Our extensive experiments on four large-scale real datasets (up to 4.33 million data points) show that all our methods can achieve one to two-order-of-magnitude speedup in many test cases and efficiently support KDV with exploratory operations (e.g., zooming and panning) compared with the state-of-the-art solutions.

## CCS CONCEPTS

• **Theory of computation** → **Computational geometry**; • **Human-centered computing** → **Heat maps**; • **Information systems** → **Geographic information systems**.

## KEYWORDS

Kernel density visualization, hotspot detection, SLAM

**ACM Reference Format:**
Tsz Nam Chan, Leong Hou U, Byron Choi, and Jianliang Xu. 2022. SLAM: Efficient Sweep Line Algorithms for Kernel Density Visualization. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3514221.3517823

## 1 INTRODUCTION

Kernel-density-estimation-based visualization (or Kernel Density Visualization (KDV)) [16, 17, 57] has become a de facto visual analytic tool for various applications, including hotspot detection [15, 34, 37, 38, 53, 62, 65, 69, 72] and ecological modeling [14, 27, 28, 60, 66]. Criminologists [15, 34, 37, 53, 72] and transportation experts [62, 65, 69] utilize KDV to detect the crime and traffic accident hotspots in different regions, respectively. Epidemiologists [39, 45, 56, 59, 68] utilize KDV to detect the disease outbreak in different cities and countries. Ecologists [14, 28, 60] utilize KDV to model the distribution of environmental incidents, e.g., pollution [60]. Due to a wide range of applications of KDV, many geographical and scientific software packages, including QGIS [52], ArcGIS [1], Scikit-learn [47], and KDV-Explorer [19], have been developed to support this operation. Figure 1 illustrates an example usage of KDV to identify the traffic accident hotspots in two regions, namely Upper Manhattan and Lower Manhattan, of New York City, using the New York traffic accident dataset [3] (with 1.5 million location data points).

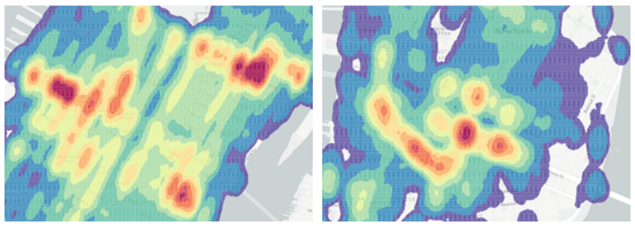

(a) Upper Manhattan (b) Lower Manhattan

**Figure 1: Generate the traffic accident hotspot maps, based on KDV, in two regions, namely Upper Manhattan and Lower Manhattan, of New York City, using the software KDV-Explorer [19], where each pixel with red color represents the high density value (i.e., hotspot/ traffic accident blackspot).**

Despite the usefulness of KDV, generating KDV is very time-consuming, which takes $O(XYn)$ time in the worst case, where $X \times Y$ and $n$ denote the screen resolution size (e.g., $1920 \times 1080$) and the number of location data points (e.g., 1.5 million), respectively. The huge amounts of computational costs restrict the applicability of this operation to small-scale datasets and low resolution sizes. Worse still, domain experts [19, 25, 36, 41–43, 70] need to generate massive amounts of KDVs for a single dataset, by using some exploratory tools, including zooming, panning, bandwidth selection (i.e., control the smoothness of the hotspot map), attribute-based filtering (e.g., generate KDV for only robbery crime events), and time-based filtering (e.g., generate KDV for those crime events

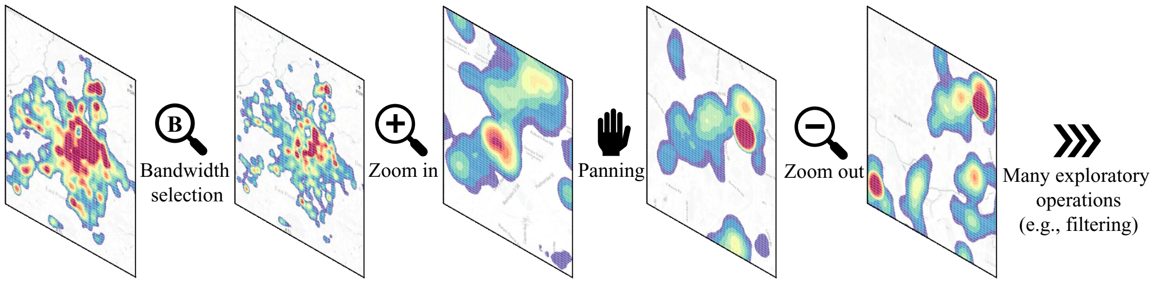


**Figure 2: Some visual analytic tasks need to undergo many exploratory operations in order to thoroughly understand the hotspots of a given dataset.**

**Table 1: Theoretical results of different exact methods for generating KDV.**

| Method | Time complexity | Space complexity |
|---|---|---|
| RQS (cf. Section 2.2) | $O(XYn)$ | $O(XY+n)$ |
| $\text{SLAM}_{\text{SORT}}$ (cf. Section 3.4) | $O(Y(X+n\log n))$ (cf. Theorem 1) | $O(XY+n)$ (cf. Theorem 4) |
| $\text{SLAM}_{\text{BUCKET}}$ (cf. Section 3.5) | $O(Y(X+n))$ (cf. Theorem 2) | |
| $\text{SLAM}^{(\text{RAO})}_{\text{SORT}}$ (cf. Sections 3.4 and 3.6) | $O(\min(X,Y)\times(\max(X,Y)+n\log n))$ (cf. Theorem 3) | |
| $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ (cf. Sections 3.5 and 3.6) | $O(\min(X,Y)\times(\max(X,Y)+n))$ (cf. Theorem 3) | |

from 1st Jan 2018 to 1st Jan 2019), in order to thoroughly understand the hotspots (cf. Figure 2). Therefore, many research studies also complain about the inefficiency issues for using KDV. Some representative ones are quoted as follows.

- *"KDV cannot scale well to handle many data points and display of color maps on high-resolution screens."* [16]
- *"...the total runtime cost of density estimation is quadratic in dataset size..."* [31]
- *"However, many (or even most) of the practical algorithms and solutions designed in the context of KDE are very time-consuming with quadratic computational complexity being a commonplace."* [32]

Even though many solutions [16, 21, 22, 31, 73, 75] have been developed to efficiently generate KDV, these algorithms either cannot reduce the time complexity [16, 21, 22, 31] or cannot provide the exact solution [16, 21, 22, 31, 73, 75] for generating KDV. Therefore, we ask a question: *Can we reduce the time complexity of generating KDV, without degrading the quality of KDV theoretically (i.e., exact solution)?*

To provide an affirmative answer to this question, we first develop two Sweep Line AlgorithMs (SLAM), which are the simple sorting-based sweep line algorithm ($\text{SLAM}_{\text{SORT}}$) and the advanced bucket-based sweep line algorithm ($\text{SLAM}_{\text{BUCKET}}$). These two algorithms can theoretically reduce the time complexity for generating KDV compared with the state-of-the-art solutions (e.g., QUAD [16]). In addition, we further develop the resolution-aware optimization (RAO) and incorporate this method into $\text{SLAM}_{\text{SORT}}$ and $\text{SLAM}_{\text{BUCKET}}$, which are called $\text{SLAM}^{(\text{RAO})}_{\text{SORT}}$ and $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$, respectively. Theoretically, both $\text{SLAM}^{(\text{RAO})}_{\text{SORT}}$ and $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ can further reduce the time complexity for generating KDV compared with $\text{SLAM}_{\text{SORT}}$ and $\text{SLAM}_{\text{BUCKET}}$, respectively. Table 1 summarizes the theoretical results for generating KDV in different exact methods, where $X \times Y$ and $n$ denote the resolution size of a given geographical region and the number of data points, respectively. To the best of our knowledge, this is the first research work that can reduce the time complexity for generating exact KDV. Our experiments on four large-scale real datasets show that all our methods can achieve one to two-order-of-magnitude speedup in many test cases compared with the state-of-the-art solutions. In addition, we can efficiently support KDV generation with exploratory tools (e.g., zooming and panning), using large-scale datasets and high resolution sizes, which cannot be achieved by other software packages, including Scikit-learn [47], QGIS [52], ArcGIS [1], and KDV-Explorer [19] (slow with high resolution sizes).

The rest of the paper is organized as follows. We first revisit the concept of KDV and discuss the exact method, called range-query-based solution (RQS), in Section 2. Then, we present our sweep line algorithms (SLAM) in Section 3. After that, we provide the experimental evaluation in Section 4. Then, we review the existing work in Section 5. Lastly, we conclude our paper in Section 6.

## 2 PRELIMINARIES

In this section, we revisit the concepts of KDV in Section 2.1. Then, we discuss the exact solution, called range-query-based solution (RQS), in Section 2.2, which is mostly related to this work.

### 2.1 Revisitation of KDV

In order to generate KDV for each geographical region (e.g., Upper Manhattan in Figure 1a), we need to color each pixel $\mathbf{q}$, based on the kernel density function value of this pixel. Here, we formally define KDV in Problem 1.

PROBLEM 1. *Given a geographical region with size $X \times Y$, where $X$ and $Y$ are the numbers of pixels in the $x$-axis and $y$-axis, respectively, and a set $P$ of $n$ location data points, we compute the kernel density function $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 1) for each pixel $\mathbf{q}$.*

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p}\in P} w \cdot K(\mathbf{q},\mathbf{p}) \tag{1}$$

*where $w$ and $K(\mathbf{q},\mathbf{p})$ denote the normalization constant and the kernel function, respectively.*

Table 2 summarizes the representative kernel functions. Here, we denote $b$ and $dist(\mathbf{q},\mathbf{p})$ as the bandwidth value of the kernel function and the Euclidean distance, respectively.

**Table 2: Representative kernel functions.**

| Kernel | $K(\mathbf{q},\mathbf{p})$ | Used in |
|---|---|---|
| Uniform | $\begin{cases} \frac{1}{b} & \text{if } dist(\mathbf{q},\mathbf{p}) \le b \\ 0 & \text{otherwise} \end{cases}$ | [34, 64] |
| Epanechnikov | $\begin{cases} 1-\frac{1}{b^2}dist(\mathbf{q},\mathbf{p})^2 & \text{if } dist(\mathbf{q},\mathbf{p}) \le b \\ 0 & \text{otherwise} \end{cases}$ | [10, 60] |
| Quartic | $\begin{cases} (1-\frac{1}{b^2}dist(\mathbf{q},\mathbf{p})^2)^2 & \text{if } dist(\mathbf{q},\mathbf{p}) \le b \\ 0 & \text{otherwise} \end{cases}$ | [38, 65] |

In this paper, we denote the $x$ and $y$ coordinates of each location point $\mathbf{p}$ to be $\mathbf{p}.x$ and $\mathbf{p}.y$, respectively, i.e., $\mathbf{p} = (\mathbf{p}.x, \mathbf{p}.y)$, and mainly use the Epanechnikov kernel in $\mathcal{F}_P(\mathbf{q})$ for discussion (i.e., Equation 2). We leave the discussion for using our methods to support other kernel functions (cf. Table 2) in Section 3.7.

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p}\in P} w \cdot \begin{cases} 1-\frac{1}{b^2}dist(\mathbf{q},\mathbf{p})^2 & \text{if } dist(\mathbf{q},\mathbf{p}) \le b \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

## 2.2 Range-Query-based Solution (RQS)

In Equation 2, observe that only those data points $\mathbf{p}$ with $dist(\mathbf{q},\mathbf{p}) \le b$ can contribute to the kernel density function value $\mathcal{F}_P(\mathbf{q})$ of the pixel $\mathbf{q}$. Therefore, we can first find the range query solution set $R(\mathbf{q})$ for each pixel $\mathbf{q}$, where:

$$R(\mathbf{q}) = \{\mathbf{p} \in P : dist(\mathbf{q},\mathbf{p}) \le b\} \quad (3)$$

Then, we can find the kernel density function value $\mathcal{F}_P(\mathbf{q})$ for pixel $\mathbf{q}$, based on this range query solution set $R(\mathbf{q})$, where:

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p}\in R(\mathbf{q})} w \cdot \left(1-\frac{1}{b^2}dist(\mathbf{q},\mathbf{p})^2\right) \quad (4)$$

Even though many well-known and commonly-used index structures, e.g., kd-tree [9], and ball-tree [44], have been proposed to improve the efficiency for solving range query problem, which can also boost the efficiency for generating exact KDV, the worst-case time complexity for using this approach to solve this problem remains in $O(XYn)$.

## 3 SWEEP LINE ALGORITHMS (SLAM)

In this section, we first summarize the core ideas of our solutions in Section 3.1. Then, we discuss two concepts, that are adopted in our solutions, namely (1) envelope point set and (2) lower and upper bound functions, in Section 3.2 and Section 3.3, respectively. Based on these two concepts, we propose two Sweep Line AlgorithMs (SLAM) for efficiently generating KDV, which are sorting-based sweep line algorithm (SLAM$_{\text{SORT}}$) and bucket-based sweep line algorithm (SLAM$_{\text{BUCKET}}$) in Section 3.4 and Section 3.5, respectively. Then, we develop the resolution-aware optimization (RAO) for SLAM$_{\text{SORT}}$ and SLAM$_{\text{BUCKET}}$ (i.e., SLAM$^{(\text{RAO})}_{\text{SORT}}$ and SLAM$^{(\text{RAO})}_{\text{BUCKET}}$, respectively) in Section 3.6. After that, we discuss how to extend our methods to support other representative kernel functions in Section 3.7. Lastly, we discuss the space complexity of SLAM in Section 3.8.

### 3.1 Core Ideas

The main disadvantage for using RQS is that we need to compute the range query solution set $R(\mathbf{q})$ (cf. Equation 3) for each pixel $\mathbf{q}$, in order to compute the kernel density function $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 4). However, two consecutive pixels $\mathbf{q}$ and $\mathbf{q}'$ can share a large amount of data points (e.g., white circles in Figure 3). Therefore, we ask a question: Suppose that we have obtained the range query solution set $R(\mathbf{q})$, can we efficiently update this set to $R(\mathbf{q}')$? In Figure 3, we observe that the most efficient approach to update from $R(\mathbf{q})$ to $R(\mathbf{q}')$ is to remove all yellow circles and insert all green circles.

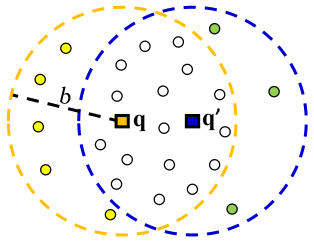


**Figure 3: The range query solution sets, i.e., $R(\mathbf{q})$ (inside the dashed orange circle) and $R(\mathbf{q}')$ (inside the dashed blue circle), of two consecutive pixels, i.e., q and q', respectively, share a large amount of data points (white circles).**

On the other hand, computing $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 4) from scratch for each pixel $\mathbf{q}$ can be time-consuming. By expanding Equation 4, we have:

$$\begin{aligned}\mathcal{F}_P(\mathbf{q}) &= \sum_{\mathbf{p}\in R(\mathbf{q})} w \cdot \left(1-\frac{1}{b^2}dist(\mathbf{q},\mathbf{p})^2\right) \\ &= w|R(\mathbf{q})| - \frac{w}{b^2}\sum_{\mathbf{p}\in R(\mathbf{q})} (\mathbf{q}-\mathbf{p})^T(\mathbf{q}-\mathbf{p}) \\ &= w|R(\mathbf{q})| - \frac{w}{b^2}\sum_{\mathbf{p}\in R(\mathbf{q})} (||\mathbf{q}||_2^2 - 2\mathbf{q}^T\mathbf{p} + ||\mathbf{p}||_2^2) \\ &= w|R(\mathbf{q})| - \frac{w}{b^2}\left(|R(\mathbf{q})| \times ||\mathbf{q}||_2^2 - 2\mathbf{q}^T\Big(\sum_{\mathbf{p}\in R(\mathbf{q})}\mathbf{p}\Big) + \sum_{\mathbf{p}\in R(\mathbf{q})}||\mathbf{p}||_2^2\right)\end{aligned}$$

Once we let $\mathbf{A}_{R_\mathbf{q}} = \sum_{\mathbf{p}\in R(\mathbf{q})}\mathbf{p}$ and $S_{R_\mathbf{q}} = \sum_{\mathbf{p}\in R(\mathbf{q})}||\mathbf{p}||_2^2$, we can represent $\mathcal{F}_P(\mathbf{q})$ by the following expression:

$$\mathcal{F}_P(\mathbf{q}) = w|R(\mathbf{q})| - \frac{w}{b^2}\left(|R(\mathbf{q})| \times ||\mathbf{q}||_2^2 - 2\mathbf{q}^T\mathbf{A}_{R_\mathbf{q}} + S_{R_\mathbf{q}}\right) \quad (5)$$

Based on Equation 5, suppose that we can efficiently maintain these aggregate terms, i.e., $|R(\mathbf{q})|$, $\mathbf{A}_{R_\mathbf{q}}$, and $S_{R_\mathbf{q}}$, we can efficiently obtain $\mathcal{F}_P(\mathbf{q})$.

### 3.2 Envelope Point Set

We consider how to evaluate all pixels, $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_X$, with the same $y$-coordinate (i.e., pixels in the green region in Figure 4). Here, we let the $y$-coordinate of all these pixels to be $k$, i.e.,

$$\mathbf{q}_1.y = \mathbf{q}_2.y = \cdots = \mathbf{q}_X.y = k$$

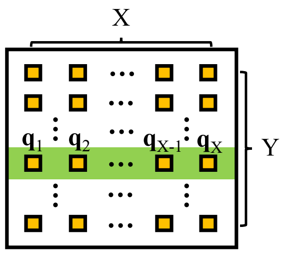


**Figure 4: All pixels, $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_X$, in the green region have the same $y$-coordinate ($= k$).**

We also define the concept, called envelope point set, for this $y$-coordinate $k$ in Definition 1.

Definition 1. *Given a set $P$ of location data points, bandwidth $b$ and $y$-coordinate $k$, we define the envelope point set of $k$ to be $E(k)$, where:*

$$E(k) = \{\mathbf{p} \in P : |k - \mathbf{p}.y| \leq b\} \tag{6}$$

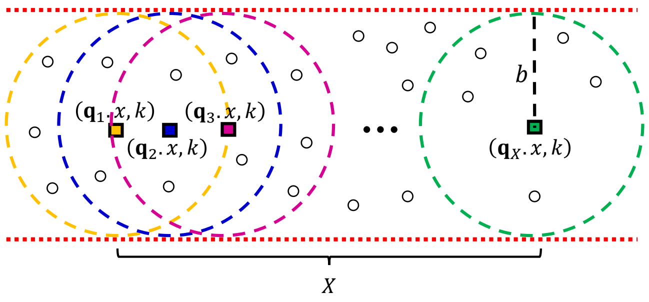


**Figure 5: The white circles, which are covered by the red dotted lines, are in the envelope point set $E(k)$. This $E(k)$ covers the range query solution sets $R(\mathbf{q}_i)$ for all pixels $\mathbf{q}_i$ with the same $y$-coordinate $k$.**

In Figure 5, observe that this envelope point set $E(k)$ covers all range query solution sets for all pixels with the same $y$-coordinate $k$, i.e.,

$$R(\mathbf{q}_i) \subseteq E(k), i = 1, 2, ..., X$$

Lemma 1 states that we can use $O(n)$ time to find the envelope point set $E(k)$.

Lemma 1. *The time complexity of finding $E(k)$ (cf. Equation 6) is $O(n)$.*

Proof. To find the envelope point set $E(k)$, we only need to scan the point dataset $P$ with size $n$ and check the condition in Equation 6. As such, the time complexity of finding $E(k)$ is $O(n)$. □

## 3.3 Lower and Upper Bound Functions

In order to find the range query solution set $R(\mathbf{q})$ for each pixel $\mathbf{q}$ with the same $y$-coordinate $k$ (cf. Figure 5), we check whether the condition $dist(\mathbf{q}, \mathbf{p}) \leq b$ holds for the data point $\mathbf{p}$ (i.e., white circle). By expanding this expression, we have:

$$\begin{aligned} dist(\mathbf{q}, \mathbf{p}) &\leq b \\ (\mathbf{q}.x - \mathbf{p}.x)^2 &\leq b^2 - (k - \mathbf{p}.y)^2 \end{aligned}$$

Based on Equation 6, we ensure that each data point $\mathbf{p}$ (cf. Figure 5) in the envelope point set $E(k)$ fulfills $b^2 - (k - \mathbf{p}.y)^2 \geq 0$. As such, we can take the square root for the above inequality to obtain the following expression.

$$\mathbf{p}.x - \sqrt{b^2 - (k - \mathbf{p}.y)^2} \leq \mathbf{q}.x \leq \mathbf{p}.x + \sqrt{b^2 - (k - \mathbf{p}.y)^2} \tag{7}$$

Therefore, after we have obtained the envelope point set $E(k)$, we can then store the lower bound and upper bound values, $LB_k(\mathbf{p})$ and $UB_k(\mathbf{p})$, for each data point $\mathbf{p}$, where:

$$LB_k(\mathbf{p}) = \mathbf{p}.x - \sqrt{b^2 - (k - \mathbf{p}.y)^2} \tag{8}$$

$$UB_k(\mathbf{p}) = \mathbf{p}.x + \sqrt{b^2 - (k - \mathbf{p}.y)^2} \tag{9}$$

With these bound values for each data point $\mathbf{p}$ in $E(k)$, we can conclude that the data point $\mathbf{p}$ is in the range query solution set $R(\mathbf{q})$ if $LB_k(\mathbf{p}) \leq \mathbf{q}.x \leq UB_k(\mathbf{p})$ (cf. Lemma 2). Using Figure 6 as an example, since $\mathbf{q}.x$ is inside the bound intervals of $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$, the range query solution set $R(\mathbf{q})$ contains these three data points.

Lemma 2. *Given the lower and upper bound values, i.e., $LB_k(\mathbf{p})$ and $UB_k(\mathbf{p})$, respectively, for each data point $\mathbf{p}$ in the envelope point set $E(k)$, this data point $\mathbf{p}$ is in the range query solution set $R(\mathbf{q})$ if $\mathbf{q}.x$ is within the bound interval $[LB_k(\mathbf{p}), UB_k(\mathbf{p})]$.*

Proof. Since we use $dist(\mathbf{q}, \mathbf{p}) \leq b$ to derive the bound functions $LB_k(\mathbf{p})$ and $UB_k(\mathbf{p})$ (cf. Equation 8 and Equation 9, respectively), we can conclude that $LB_k(\mathbf{q}) \leq \mathbf{q}.x \leq UB_k(\mathbf{q}) \implies dist(\mathbf{q}, \mathbf{p}) \leq b \implies \mathbf{p} \in R(\mathbf{q})$ (cf. Equation 3). □

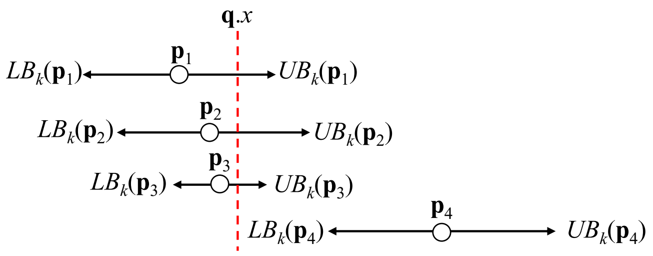


**Figure 6: The range query solution set of $R(\mathbf{q}) = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$.**

## 3.4 A Simple Sorting-based Sweep Line Algorithm (SLAM$_{\text{SORT}}$)

Once we have obtained the lower and upper bound values for each data point in the envelope point set $E(k)$, our method SLAM$_{\text{SORT}}$ needs to create the list $\mathcal{L}$, by sorting these bound values and the x-coordinates of all pixels with the same $y$-coordinate $k$ (cf. Figure 5), i.e., $\mathbf{q}_1.x$, $\mathbf{q}_2.x$,..., $\mathbf{q}_X.x$, in an increasing order. Using Figure 7 as an example, the list $\mathcal{L} = [LB_k(\mathbf{p}_1), LB_k(\mathbf{p}_2), LB_k(\mathbf{p}_3), LB_k(\mathbf{p}_4), \mathbf{q}_{i-1}.x, UB_k(\mathbf{p}_2), UB_k(\mathbf{p}_4), ...]$. With this list $\mathcal{L}$, the core idea of SLAM$_{\text{SORT}}$ is to utilize the sweep line algorithm to process these bound values in order to compute the kernel density function $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 5) for each pixel $\mathbf{q}$. In Figure 7, we maintain two point sets, namely $\mathbb{L}_\ell$ (cf. Equation 10) and $\mathbb{U}_\ell$ (cf. Equation 11), for the sweep line $\ell$, which has passed through the lower and upper bound values of those data points, respectively.

$$\mathbb{L}_\ell = \{\mathbf{p} \in E(k) : LB_k(\mathbf{p}) \leq \ell.x\} \tag{10}$$

$$\mathbb{U}_\ell = \{\mathbf{p} \in E(k) : UB_k(\mathbf{p}) \leq \ell.x\} \tag{11}$$

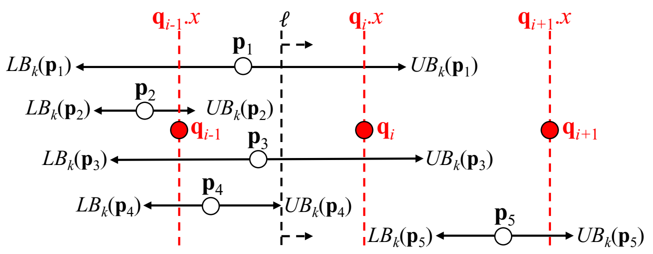
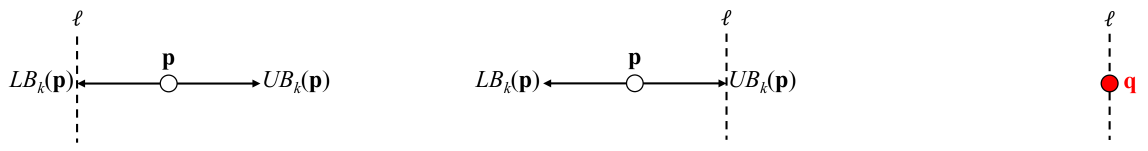


**Figure 7: Sweep line $\ell$ moves from left to right (by scanning the list $\mathcal{L}$), $\mathbb{L}_\ell = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4\}$ and $\mathbb{U}_\ell = \{\mathbf{p}_2, \mathbf{p}_4\}$.**



Case 1: $\ell$ intersects with $LB_k(\mathbf{p})$, i.e., $\ell.x = LB_k(\mathbf{p})$ (insert $\mathbf{p}$ in $\mathbb{L}_\ell$, add $A_{\mathbb{L}_\ell}$ by $\mathbf{p}$ , add $S_{\mathbb{L}_\ell}$ by $||\mathbf{p}||_2^2$)

Case 2: $\ell$ intersects with $UB_k(\mathbf{p})$, i.e., $\ell.x = UB_k(\mathbf{p})$ (insert $\mathbf{p}$ in $\mathbb{U}_\ell$, add $A_{\mathbb{U}_\ell}$ by $\mathbf{p}$ , add $S_{\mathbb{U}_\ell}$ by $||\mathbf{p}||_2^2$)

Case 3: $\ell$ intersects with $\mathbf{q}.x$, i.e., $\ell.x = \mathbf{q}.x$ (compute $\mathcal{F}_P(\mathbf{q})$)

**Figure 8: Three cases of the sweep line $\ell$ for scanning the list $\mathcal{L}$.**

We also maintain the aggregate values for these two sets, i.e., $\mathbf{A}_{\mathbb{L}_\ell}, \mathbf{A}_{\mathbb{U}_\ell}$ (cf. Equation 12) and $S_{\mathbb{L}_\ell}, S_{\mathbb{U}_\ell}$ (cf. Equation 13), which can be used to efficiently compute the kernel density function $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 5).

$$\mathbf{A}_{\mathbb{L}_\ell} = \sum_{\mathbf{p}\in\mathbb{L}_\ell} \mathbf{p} \quad \text{and} \quad \mathbf{A}_{\mathbb{U}_\ell} = \sum_{\mathbf{p}\in\mathbb{U}_\ell} \mathbf{p} \tag{12}$$

$$S_{\mathbb{L}_\ell} = \sum_{\mathbf{p}\in\mathbb{L}_\ell} ||\mathbf{p}||_2^2 \quad \text{and} \quad S_{\mathbb{U}_\ell} = \sum_{\mathbf{p}\in\mathbb{U}_\ell} ||\mathbf{p}||_2^2 \tag{13}$$

Figure 8 shows three possible cases of the sweep line $\ell$, which are: (1) $\ell$ intersects with the lower bound $LB_k(\mathbf{p})$ of the data point $\mathbf{p}$ (i.e., $\ell.x = LB_k(\mathbf{p})$), (2) $\ell$ intersects with the upper bound $UB_k(\mathbf{p})$ of the data point $\mathbf{p}$ (i.e., $\ell.x = UB_k(\mathbf{p})$), and (3) $\ell$ intersects with $\mathbf{q}.x$ (i.e., $\ell.x = \mathbf{q}.x$). Regarding the first two cases, we need to update the corresponding point sets ($\mathbb{L}_\ell$ or $\mathbb{U}_\ell$) and aggregate values, e.g., we insert $\mathbf{p}$ in $\mathbb{L}_\ell$, add $\mathbf{A}_{\mathbb{L}_\ell}$ by $\mathbf{p}$ and add $S_{\mathbb{L}_\ell}$ by $||\mathbf{p}||_2^2$, once the sweep line $\ell$ intersects with the lower bound $LB_k(\mathbf{p})$. Therefore, the time complexity of these two cases is $O(1)$. Regarding the third case, we claim that we can calculate $\mathcal{F}_P(\mathbf{q})$ in $O(1)$ time (cf. Lemma 3), using these two point sets ($\mathbb{L}_\ell$ and $\mathbb{U}_\ell$) and their aggregate values (cf. Equations 12 and 13).

Lemma 3. *If the sweep line $\ell$ intersects with $\mathbf{q}.x$ (i.e., case 3), we can compute $\mathcal{F}_P(\mathbf{q})$ in $O(1)$ time, using Equation 5 with:*

$$|R(\mathbf{q})| = |\mathbb{L}_\ell| - |\mathbb{U}_\ell| \tag{14}$$

$$\mathbf{A}_{R(\mathbf{q})} = \mathbf{A}_{\mathbb{L}_\ell} - \mathbf{A}_{\mathbb{U}_\ell} \tag{15}$$

$$S_{R(\mathbf{q})} = S_{\mathbb{L}_\ell} - S_{\mathbb{U}_\ell} \tag{16}$$

Proof. The sweep line $\ell$ is within the bound interval $[LB_k(\mathbf{p}), UB_k(\mathbf{p})]$, if $\ell$ passes through $LB_k(\mathbf{p})$, i.e., $\mathbb{L}_\ell$ contains $\mathbf{p}$, and $\ell$ does not pass through $UB_k(\mathbf{p})$, i.e., $\mathbb{U}_\ell$ does not contain $\mathbf{p}$. Therefore, once the line $\ell$ intersects with $\mathbf{q}.x$, $|\mathbb{L}_\ell| - |\mathbb{U}_\ell|$ denotes the number of bound intervals that contain $\mathbf{q}.x$. Based on Lemma 2, we prove Equation 14. By adopting the similar argument, we can also prove Equation 15 and Equation 16. □

Algorithm 1 describes how our method SLAM$_{\text{SORT}}$ finds the kernel density function values for all pixels $\mathbf{q}_1, \mathbf{q}_2,..., \mathbf{q}_X$ with the same $y$-coordinate (cf. Figure 4). In Lemma 4, we show that Algorithm 1 takes $O(n \log n + X)$ time to process all pixels $\mathbf{q}_1$, $\mathbf{q}_2$,...., $\mathbf{q}_X$ with the same $y$-coordinate.

Lemma 4. *Given the pixels $\mathbf{q}_1$, $\mathbf{q}_2$,...., $\mathbf{q}_X$ with the same $y$-coordinate, SLAM$_{SORT}$ computes the kernel density function values for all these pixels with $O(n \log n + X)$ time.*

Proof. Recall that the time complexity of finding $E(k)$ (line 2) is $O(n)$ (cf. Lemma 1) and the sweep line $\ell$ only takes at most $O(n+X)$

**Algorithm 1** Sorting-based Sweep Line Algorithm (SLAM$_{\text{SORT}}$)

```
1: procedure SLAM_SORT(Point set $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$, bandwidth $b$, pixels $\mathbf{q}_1, \mathbf{q}_2,..., \mathbf{q}_X$ with $y$-coordinate $k$)
2:     Find the envelope point set $E(k)$                    ▷ Equation 6
3:     Create the sorted list $\mathcal{L}$                  ▷ Increasing order
4:     $\mathbb{L}_\ell \leftarrow \phi$, $\mathbb{U}_\ell \leftarrow \phi$
5:     $\mathbf{A}_{\mathbb{L}_\ell} \leftarrow \mathbf{0}$, $\mathbf{A}_{\mathbb{U}_\ell} \leftarrow \mathbf{0}$
6:     $S_{\mathbb{L}_\ell} \leftarrow 0$, $S_{\mathbb{U}_\ell} \leftarrow 0$
7:     $\ell.x \leftarrow -\infty$
8:     while $\ell.x \leq \mathbf{q}_X.x$ do
9:         $\ell$ sweeps the next element in $\mathcal{L}$
10:        if $\ell.x = LB_k(\mathbf{p})$ then                ▷ Case 1
11:            $\mathbb{L}_\ell \leftarrow \mathbb{L}_\ell \cup \{\mathbf{p}\}$
12:            $\mathbf{A}_{\mathbb{L}_\ell} \leftarrow \mathbf{A}_{\mathbb{L}_\ell} + \mathbf{p}$
13:            $S_{\mathbb{L}_\ell} \leftarrow S_{\mathbb{L}_\ell} + ||\mathbf{p}||_2^2$
14:        if $\ell.x = UB_k(\mathbf{p})$ then                ▷ Case 2
15:            $\mathbb{U}_\ell \leftarrow \mathbb{U}_\ell \cup \{\mathbf{p}\}$
16:            $\mathbf{A}_{\mathbb{U}_\ell} \leftarrow \mathbf{A}_{\mathbb{U}_\ell} + \mathbf{p}$
17:            $S_{\mathbb{U}_\ell} \leftarrow S_{\mathbb{U}_\ell} + ||\mathbf{p}||_2^2$
18:        if $\ell.x = \mathbf{q}_i.x$ then         ▷ Case 3 (where $1 \leq i \leq X$)
19:            Compute $\mathcal{F}_P(\mathbf{q}_i)$                ▷ Lemma 3
20:    Return $\{\mathcal{F}_P(\mathbf{q}_1), \mathcal{F}_P(\mathbf{q}_2), ..., \mathcal{F}_P(\mathbf{q}_X)\}$
```

iterations for sweeping all elements in the list $\mathcal{L}$ (which contains at most $2|E(k)| + X$ elements and $|E(k)| \to n$ in the worst case), where each iteration (lines 9 - 19) takes $O(1)$ time. Therefore, the main bottleneck of this algorithm is to sort the list $\mathcal{L}$ (line 3), which takes $O(n \log n + X)$ time.[1] Therefore, the worst-case time complexity of Algorithm 1 is $O(n \log n + X)$. Hence, this lemma is proved. □

Based on Lemma 4, we conclude that SLAM$_{\text{SORT}}$ takes $O(Y(n \log n + X))$ time to generate KDV, i.e., computes the kernel density values for all $X \times Y$ pixels, (cf. Theorem 1).

Theorem 1. *The time complexity of SLAM$_{SORT}$ is $O(Y(n \log n + X))$.*

Proof. Since there are $Y$ y-coordinates in the geographical region (cf. Figure 4) and SLAM$_{\text{SORT}}$ takes $O(n \log n + X)$ time to compute the kernel density function values for all pixels with the same $y$-coordinate (cf. Lemma 4), the time complexity of this method is $O(Y(n \log n + X))$. □

[1]Since the pixels $\mathbf{q}_1$, $\mathbf{q}_2$,..., $\mathbf{q}_X$ are sorted in advance ($\mathbf{q}_1.x < \mathbf{q}_2.x < \ldots < \mathbf{q}_X.x$), we only need to sort the data points in $E(k)$, which takes $O(n \log n)$ time, and merge them into the list $\mathcal{L}$, which takes $O(n + X)$ time.

Compared with the method RQS (cf. Section 2.2), which takes $O(XYn)$ time to generate each KDV, our method SLAM$_{\text{SORT}}$ is theoretically faster than RQS, if $X > \log n$. In practice, since $X$ is normally large, the inequality $X > \log n$ usually holds (e.g., the resolution size can be $1280 \times 960$ (i.e., $X = 1280$) in modern screens and $n$ is 1.5 million for the New York traffic accident dataset [3]).

## 3.5 An Advanced Bucket-based Sweep Line Algorithm (SLAM$_{\text{BUCKET}}$)

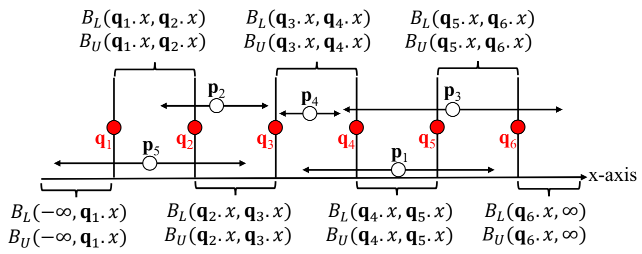
Even though SLAM$_{\text{SORT}}$ can be theoretically faster than the method RQS (cf. Section 2.2) in most of the cases (with $X > \log n$), this method cannot be faster than RQS in all cases (e.g., the cases with $X \leq \log n$). Therefore, we further ask a question: Can we further improve the theoretical result (e.g., remove the $\log n$ factor in Theorem 1)? Here, we provide an affirmative answer to this question.

Recall that the main bottleneck of Algorithm 1 is to sort the list $\mathcal{L}$ (line 3), which incurs $O(n \log n + X)$ time. Suppose that we can avoid this sorting operation, it is possible to remove the $\log n$ factor in Theorem 1. In Figure 4, observe that the x-coordinates of all pixels in the green region follow an increasing order, i.e., $\mathbf{q}_0.x < \mathbf{q}_1.x < \mathbf{q}_2.x < \cdots < \mathbf{q}_{X-1}.x < \mathbf{q}_X.x < \mathbf{q}_{X+1}.x$, where we denote two dummy pixels as $\mathbf{q}_0$ and $\mathbf{q}_{X+1}$ with $\mathbf{q}_0.x = -\infty$ and $\mathbf{q}_{X+1}.x = \infty$, respectively. Therefore, we conclude that the lower (upper) bound value of each data point $\mathbf{p}$, i.e., $LB_k(\mathbf{p})$ ($UB_k(\mathbf{p})$) must be within any unique interval $[\mathbf{q}_{i-1}.x, \mathbf{q}_i.x]$ (cf. Figure 9), where $1 \leq i \leq X + 1$.

Based on this property, we augment the lower and upper bound buckets, denoted as $B_L(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$ and $B_U(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$, respectively, between two consecutive pixels $\mathbf{q}_{i-1}$ and $\mathbf{q}_i$, where:

$$B_L(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x) = \{\mathbf{p} \in E(k) : \mathbf{q}_{i-1}.x < LB_k(\mathbf{p}) \leq \mathbf{q}_i.x\} \quad (17)$$

$$B_U(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x) = \{\mathbf{p} \in E(k) : \mathbf{q}_{i-1}.x < UB_k(\mathbf{p}) \leq \mathbf{q}_i.x\} \quad (18)$$



$B_L(\mathbf{q}_1.x, \mathbf{q}_2.x)$
$B_U(\mathbf{q}_1.x, \mathbf{q}_2.x)$
$B_L(\mathbf{q}_3.x, \mathbf{q}_4.x)$
$B_U(\mathbf{q}_3.x, \mathbf{q}_4.x)$
$B_L(\mathbf{q}_5.x, \mathbf{q}_6.x)$
$B_U(\mathbf{q}_5.x, \mathbf{q}_6.x)$
$\mathbf{p}_2$
$\mathbf{p}_4$
$\mathbf{p}_3$
$\mathbf{q}_1$
$\mathbf{p}_5$
$\mathbf{q}_2$
$\mathbf{q}_3$
$\mathbf{q}_4$
$\mathbf{p}_1$
$\mathbf{q}_5$
$\mathbf{q}_6$
x-axis
$B_L(-\infty, \mathbf{q}_1.x)$
$B_U(-\infty, \mathbf{q}_1.x)$
$B_L(\mathbf{q}_2.x, \mathbf{q}_3.x)$
$B_U(\mathbf{q}_2.x, \mathbf{q}_3.x)$
$B_L(\mathbf{q}_4.x, \mathbf{q}_5.x)$
$B_U(\mathbf{q}_4.x, \mathbf{q}_5.x)$
$B_L(\mathbf{q}_6.x, \infty)$
$B_U(\mathbf{q}_6.x, \infty)$

**Figure 9: Augment the buckets for each pair of consecutive pixels with $X = 6$.**

Each lower (upper) bound bucket stores the data points $\mathbf{p}$ with their lower (upper) bounds $LB_k(\mathbf{p})$ ($UB_k(\mathbf{p})$) lying in this bucket. Using Figure 9 as an example, Table 3 shows the data points in these buckets.

**Table 3: Data points in the lower and upper bound buckets (cf. Figure 9).**

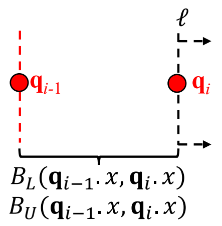
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $B_L(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$ | $\{\mathbf{p}_5\}$ | $\{\mathbf{p}_2\}$ | $\phi$ | $\{\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_4\}$ | $\phi$ | $\phi$ | $\phi$ |
| $B_U(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$ | $\phi$ | $\phi$ | $\{\mathbf{p}_2, \mathbf{p}_5\}$ | $\{\mathbf{p}_4\}$ | $\phi$ | $\{\mathbf{p}_1\}$ | $\{\mathbf{p}_3\}$ |

Since the $x$-coordinate gap $g_x$ between each pair of two consecutive pixels (except for $\mathbf{q}_1.x - \mathbf{q}_0.x$ and $\mathbf{q}_{X+1}.x - \mathbf{q}_X.x$) is the same, i.e., $\mathbf{q}_2.x - \mathbf{q}_1.x = \cdots = \mathbf{q}_X.x - \mathbf{q}_{X-1}.x = g_x$, we can use $O(1)$ time to determine which lower and upper bound buckets ($B_L(\mathbf{q}_{i_l-1}.x, \mathbf{q}_{i_l}.x)$ and $B_U(\mathbf{q}_{i_u-1}.x, \mathbf{q}_{i_u}.x)$) cover the data point $\mathbf{p}$, where:

$$i_l = \max\left(\left\lceil \frac{LB_k(\mathbf{p}) - \mathbf{q}_1.x}{g_x} \right\rceil, 0\right) \quad (19)$$

$$i_u = \min\left(\left\lceil \frac{UB_k(\mathbf{p}) - \mathbf{q}_1.x}{g_x} \right\rceil, X + 1\right) \quad (20)$$

Once we have obtained these buckets, we utilize the sweep line $\ell$ to process each pixel $\mathbf{q}_i$. Suppose that $\ell$ intersects with $\mathbf{q}_i.x$ (cf. Figure 10), we insert the bucket $B_L(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$ and $B_U(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$ into $\mathbb{L}_\ell$ and $\mathbb{U}_\ell$, respectively, and also update the aggregate values $\mathbf{A}_{\mathbb{L}_\ell}$, $\mathbf{A}_{\mathbb{U}_\ell}$ (cf. Equation 12) and $S_{\mathbb{L}_\ell}$, $S_{\mathbb{U}_\ell}$ (cf. Equation 13) for these two sets.



$\ell$
$\mathbf{q}_{i-1}$
$\mathbf{q}_i$
$\mathbf{q}_{i+1}$
$B_L(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$
$B_U(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$

**Figure 10: Sweep line $\ell$ intersects with $\mathbf{q}_i.x$ (We need to insert $B_L(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$ and $B_U(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$ into $\mathbb{L}_\ell$ and $\mathbb{U}_\ell$, respectively, and add $\mathbf{A}_{\mathbb{L}_\ell}$ ($S_{\mathbb{L}_\ell}$) and $\mathbf{A}_{\mathbb{U}_\ell}$ ($S_{\mathbb{U}_\ell}$) by $\mathbf{p}$ ($||\mathbf{p}||_2^2$) with each $\mathbf{p}$ in $B_L(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$ and $B_U(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$, respectively).**

Hence, if $\ell$ intersects with $\mathbf{q}_i.x$, we have:

$$\mathbb{L}_\ell = \bigcup_{j=1}^{i} B_L(\mathbf{q}_{j-1}.x, \mathbf{q}_j.x) \quad (21)$$

$$\mathbb{U}_\ell = \bigcup_{j=1}^{i} B_U(\mathbf{q}_{j-1}.x, \mathbf{q}_j.x) \quad (22)$$

In Lemma 5, we claim that we can correctly compute $\mathcal{F}_P(\mathbf{q}_i)$, using these two sets $\mathbb{L}_\ell$ (cf. Equation 21) and $\mathbb{U}_\ell$ (cf. Equation 22) and their aggregate values (cf. Equations 12 and 13).

Lemma 5. *If the sweep line $\ell$ intersects with $\mathbf{q}_i.x$ (cf. Figure 10), we can correctly compute $\mathcal{F}_P(\mathbf{q}_i)$ (cf. Equation 5), using Equations 14, 15 and 16 as $|R_{\mathbf{q}_i}|$, $\mathbf{A}_{R_{\mathbf{q}_i}}$ and $S_{R_{\mathbf{q}_i}}$, respectively.*

Proof. We first prove $R_{\mathbf{q}_i} = \mathbb{L}_\ell \backslash \mathbb{U}_\ell$ and then prove $\mathbb{U}_\ell \subseteq \mathbb{L}_\ell$. Based on these two expressions, we have:

$$|R_{\mathbf{q}_i}| = |L_\ell \backslash U_\ell| = |L_\ell| - |U_\ell|$$

$$\mathbf{A}_{R_{\mathbf{q}_i}} = \sum_{\mathbf{p} \in R_{\mathbf{q}_i}} \mathbf{p} = \sum_{\mathbf{p} \in L_\ell \backslash U_\ell} \mathbf{p} = \sum_{\mathbf{p} \in L_\ell} \mathbf{p} - \sum_{\mathbf{p} \in U_\ell} \mathbf{p} = \mathbf{A}_{\mathbb{L}_\ell} - \mathbf{A}_{\mathbb{U}_\ell}$$

$$S_{R_{\mathbf{q}_i}} = \sum_{\mathbf{p} \in R_{\mathbf{q}_i}} ||\mathbf{p}||_2^2 = \sum_{\mathbf{p} \in L_\ell \backslash U_\ell} ||\mathbf{p}||_2^2 = \sum_{\mathbf{p} \in L_\ell} ||\mathbf{p}||_2^2 - \sum_{\mathbf{p} \in U_\ell} ||\mathbf{p}||_2^2 = S_{\mathbb{L}_\ell} - S_{\mathbb{U}_\ell}$$

which means Equations 14, 15 and 16 hold if $\ell$ intersects with $\mathbf{q}_i.x$.

**Proof of $R_{\mathbf{q}_i} = \mathbb{L}_\ell \backslash \mathbb{U}_\ell$:** In Figure 11, observe that there are three possible cases for the bound intervals.

Consider the (blue) bound interval of $\mathbf{p}_{\text{left}}$, i.e., $LB_k(\mathbf{p}_{\text{left}}) \leq \mathbf{q}_i.x$ and $UB_k(\mathbf{p}_{\text{left}}) \leq \mathbf{q}_i.x$. Since $-\infty = \mathbf{q}_0.x < \mathbf{q}_1.x < \cdots < \mathbf{q}_i.x$, we conclude that there exists one unique interval $[\mathbf{q}_{j-1}.x, \mathbf{q}_j.x]$,

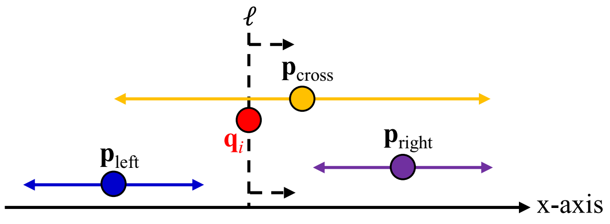

$\ell$
$\mathbf{p}_{\text{cross}}$
$\mathbf{q}_i$
$\mathbf{p}_{\text{left}}$
$\mathbf{p}_{\text{right}}$
x-axis

**Figure 11: Three possible cases for the bound intervals. (1) The (blue) bound interval of $\mathbf{p}_{\text{left}}$ is in the left side of $\mathbf{q}_i.x$. (2) The (orange) bound interval of $\mathbf{p}_{\text{cross}}$ crosses $\mathbf{q}_i.x$. (3) The (purple) bound interval of $\mathbf{p}_{\text{right}}$ is in the right side of $\mathbf{q}_i.x$.**

where $1 \le j \le i$, that contains $LB_k(\mathbf{p}_{\text{left}})$ ($UB_k(\mathbf{p}_{\text{left}})$). Based on Equations 21 and 22, we have: $\mathbf{p}_{\text{left}} \in \mathbb{L}_\ell$ and $\mathbf{p}_{\text{left}} \in \mathbb{U}_\ell$.

Consider the (purple) bound interval of $\mathbf{p}_{\text{right}}$, since both $LB_k(\mathbf{p}_{\text{right}})$ and $UB_k(\mathbf{p}_{\text{right}})$ are larger than $\mathbf{q}_i.x$, none of the intervals $[\mathbf{q}_{j-1}.x, \mathbf{q}_j.x]$ with $1 \le j \le i$ contains these two values. Therefore, we have: $\mathbf{p}_{\text{right}} \notin \mathbb{L}_\ell$ and $\mathbf{p}_{\text{right}} \notin \mathbb{U}_\ell$.

Consider the (orange) bound interval of $\mathbf{p}_{\text{cross}}$, we can easily conclude that $\mathbf{p}_{\text{cross}} \in \mathbb{L}_\ell$ and $\mathbf{p}_{\text{cross}} \notin \mathbb{U}_\ell$.

Therefore, $\mathbb{L}_\ell \backslash \mathbb{U}_\ell$ denotes the set of data points $\mathbf{p}$ with (orange) bound intervals, i.e., $LB_k(\mathbf{p}) \le \mathbf{q}_i.x \le UB_k(\mathbf{p})$. Based on Lemma 2, we conclude that $R_{\mathbf{q}_i} = \mathbb{L}_\ell \backslash \mathbb{U}_\ell$.

**Proof of $\mathbb{U}_\ell \subseteq \mathbb{L}_\ell$:** If the data point $\mathbf{p} \in \mathbb{U}_\ell$, there exists an interval $[\mathbf{q}_{j-1}.x, \mathbf{q}_j.x]$ with $1 \le j \le i$ that covers $UB_k(\mathbf{p})$. Since $LB_k(\mathbf{p}) \le UB_k(\mathbf{p})$, there exists another interval $[\mathbf{q}_{j'-1}.x, \mathbf{q}_{j'}.x]$ with $j' \le j$ that covers $LB_k(\mathbf{p})$. Therefore, $\mathbf{p} \in \mathbb{L}_\ell$ and we have: $\mathbb{U}_\ell \subseteq \mathbb{L}_\ell$. □

Algorithm 2 shows how our method SLAM$_{\text{BUCKET}}$ computes the kernel density function values for all pixels $\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_X$ with the same $y$-coordinate $k$ (cf. Figure 4). We claim that Algorithm 2 takes $O(n+X)$ time for computing the kernel density function values for all pixels $\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_X$ (cf. Lemma 6).

Lemma 6. *Given the pixels $\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_X$ with the same $y$-coordinate $k$, the time complexity for using Algorithm 2 to compute the kernel density function values $\mathcal{F}_P(\mathbf{q}_1)$, $\mathcal{F}_P(\mathbf{q}_2)$,..., $\mathcal{F}_P(\mathbf{q}_X)$ is $O(n+X)$ time.*

Proof. In lines 6-9, we assign each data point $\mathbf{p}$ in $E(k)$ in the lower and upper bound buckets, using Equations 19 and 20 (with $O(1)$ time), respectively. Therefore, the time complexity is $O(|E(k)|)$ time. In lines 13-20, we utilize the sweep line to scan each pixel. Once the sweep line $\ell$ touches the pixel $\mathbf{q}_{i_q}$, this algorithm needs to scan those data points in $B_L(\mathbf{q}_{i_q-1}.x, \mathbf{q}_{i_q}.x)$ and $B_U(\mathbf{q}_{i_q-1}.x, \mathbf{q}_{i_q}.x)$ to update $\mathbb{L}_\ell$, $\mathbf{A}_{\mathbb{L}_\ell}$, $S_{\mathbb{L}_\ell}$ and $\mathbb{U}_\ell$, $\mathbf{A}_{\mathbb{U}_\ell}$, $S_{\mathbb{U}_\ell}$, respectively. Therefore, the time complexity is $O(X + \sum_{i_q=1}^{X} |B_L(\mathbf{q}_{i_q-1}.x, \mathbf{q}_{i_q}.x)| + \sum_{i_q=1}^{X} |B_U(\mathbf{q}_{i_q-1}.x, \mathbf{q}_{i_q}.x)|) = O(X + |E(k)|)$ time (The reason is that each data point can only be in one unique lower bound bucket and one unique upper bound bucket). Therefore, the worst-case time complexity of this algorithm is $O(n+X)$ time with $|E(k)| = n$. □

Based on Lemma 6, we conclude that SLAM$_{\text{BUCKET}}$ takes $O(Y(X+n))$ time to generate KDV (cf. Theorem 2). We omit the proof of this theorem, since it is similar to the proof of Theorem 1.

Theorem 2. *The time complexity of SLAM$_{\text{BUCKET}}$ is $O(Y(n+X))$.*

**Algorithm 2** Bucket-based Sweep Line Algorithm (SLAM$_{\text{BUCKET}}$)

```
1: procedure SLAM_BUCKET(Point set $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$, bandwidth $b$, pixels $\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_X$ with $y$-coordinate $k$)
2:     Find the envelope point set $E(k)$                          ▷ Equation 6
3:     $B_L(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x) \leftarrow \phi$, where $1 \le i \le X+1$
4:     $B_U(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x) \leftarrow \phi$, where $1 \le i \le X+1$
5:     //Assign each data point $\mathbf{p} \in E(k)$ in different buckets
6:     for each $\mathbf{p} \in E(k)$ do
7:         Obtain $i_l$ and $i_u$          ▷ Equations 19 and 20, respectively
8:         $B_L(\mathbf{q}_{i_l-1}.x, \mathbf{q}_{i_l}.x) \leftarrow B_L(\mathbf{q}_{i_l-1}.x, \mathbf{q}_{i_l}.x) \cup \{\mathbf{p}\}$
9:         $B_U(\mathbf{q}_{i_u-1}.x, \mathbf{q}_{i_u}.x) \leftarrow B_U(\mathbf{q}_{i_u-1}.x, \mathbf{q}_{i_u}.x) \cup \{\mathbf{p}\}$
10:    //Sweep line $\ell$
11:    $\mathbf{A}_{\mathbb{L}_\ell} \leftarrow \mathbf{0}$, $\mathbf{A}_{\mathbb{U}_\ell} \leftarrow \mathbf{0}$
12:    $S_{\mathbb{L}_\ell} \leftarrow 0$, $S_{\mathbb{U}_\ell} \leftarrow 0$
13:    for $i_q \leftarrow 1$ to $X$ do
14:        $\mathbb{L}_\ell \leftarrow \mathbb{L}_\ell \cup B_L(\mathbf{q}_{i_q-1}.x, \mathbf{q}_{i_q}.x)$
15:        $\mathbb{U}_\ell \leftarrow \mathbb{U}_\ell \cup B_U(\mathbf{q}_{i_q-1}.x, \mathbf{q}_{i_q}.x)$
16:        $\mathbf{A}_{\mathbb{L}_\ell} \leftarrow \mathbf{A}_{\mathbb{L}_\ell} + \sum_{\mathbf{p} \in B_L(\mathbf{q}_{i_q-1}.x, \mathbf{q}_{i_q}.x)} \mathbf{p}$
17:        $\mathbf{A}_{\mathbb{U}_\ell} \leftarrow \mathbf{A}_{\mathbb{U}_\ell} + \sum_{\mathbf{p} \in B_U(\mathbf{q}_{i_q-1}.x, \mathbf{q}_{i_q}.x)} \mathbf{p}$
18:        $S_{\mathbb{L}_\ell} \leftarrow S_{\mathbb{L}_\ell} + \sum_{\mathbf{p} \in B_L(\mathbf{q}_{i_q-1}.x, \mathbf{q}_{i_q}.x)} ||\mathbf{p}||_2^2$
19:        $S_{\mathbb{U}_\ell} \leftarrow S_{\mathbb{U}_\ell} + \sum_{\mathbf{p} \in B_U(\mathbf{q}_{i_q-1}.x, \mathbf{q}_{i_q}.x)} ||\mathbf{p}||_2^2$
20:        Compute $\mathcal{F}_P(\mathbf{q}_{i_q})$                          ▷ Lemma 5
21:    Return $\{\mathcal{F}_P(\mathbf{q}_1), \mathcal{F}_P(\mathbf{q}_2), ..., \mathcal{F}_P(\mathbf{q}_X)\}$
```

### 3.6 Resolution-Aware Optimization (RAO)

Although our methods SLAM$_{\text{SORT}}$ and SLAM$_{\text{BUCKET}}$ significantly reduce the time complexity for generating KDV to $O(Y(X+n\log n))$ time (cf. Theorem 1) and $O(Y(X+n))$ time (cf. Theorem 2), respectively, the worst-case time complexity for these two methods can also be high with $Y >> X$ (the large $Y$ multiplies with the large $n$).

Therefore, we incorporate the resolution-aware optimization (RAO) into our methods SLAM$_{\text{SORT}}$ and SLAM$_{\text{BUCKET}}$, which are renamed as SLAM$_{\text{SORT}}^{(\text{RAO})}$ and SLAM$_{\text{BUCKET}}^{(\text{RAO})}$, respectively. If $Y > X$, these two methods compute the kernel density function values for the pixels with the same $x$-coordinate, i.e., $\mathbf{h}_1$, $\mathbf{h}_2$,..., $\mathbf{h}_Y$ (cf. yellow region in Figure 12), instead of the pixels with the same $y$-coordinate (cf. Figure 4). On the other hand, if $X \ge Y$, we adopt the default solutions, i.e., SLAM$_{\text{SORT}}$ and SLAM$_{\text{BUCKET}}$, for these two methods.

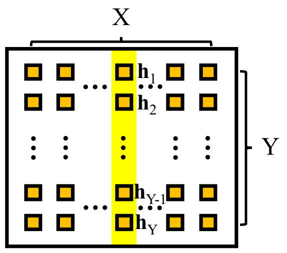


**Figure 12: Evaluate all pixels, $\mathbf{h}_1$, $\mathbf{h}_2$,..., $\mathbf{h}_Y$, which have the same $x$-coordinate, in the yellow region.**

Even though the concept of RAO is simple, these two methods SLAM$_{\text{SORT}}^{(\text{RAO})}$ and SLAM$_{\text{BUCKET}}^{(\text{RAO})}$ further reduce the time complexity for generating KDV (cf. Theorem 3).

Theorem 3. *The time complexity of $SLAM_{SORT}^{(RAO)}$ and $SLAM_{BUCKET}^{(RAO)}$ for generating KDV is $O(\min(X,Y)\times(\max(X,Y)+n\log n))$ and $O(\min(X,Y)\times(\max(X,Y)+n))$, respectively.*

Proof. If $X \geq Y$, the time complexity of $\text{SLAM}_{\text{BUCKET}}^{\text{(RAO)}}$ (default setting) is $O(Y(X+n))$ (cf. Theorem 2). If $Y > X$, the time complexity of $\text{SLAM}_{\text{BUCKET}}^{\text{(RAO)}}$ is $O(X(Y+n))$. Hence, we conclude that the time complexity of $\text{SLAM}_{\text{BUCKET}}^{\text{(RAO)}}$ is $O(\min(X,Y)\times(\max(X,Y)+n))$. Similarly, we can also prove that the time complexity of $\text{SLAM}_{\text{SORT}}^{\text{(RAO)}}$ is $O(\min(X,Y)\times(\max(X,Y)+n\log n))$. □

## 3.7 Other Kernels

In previous sections, we only focus on the Epanechnikov kernel. Here, we further extend our methods to support other kernel functions (cf. Table 2). Recall that one of the core ideas for efficiently computing the kernel density function $\mathcal{F}_P(\mathbf{q})$ with the Epanechnikov kernel is that $\mathcal{F}_P(\mathbf{q})$ is composed of the aggregate terms $|\mathrm{R}(\mathbf{q})|$, $\mathbf{A}_{R_\mathbf{q}}$ and $S_{R_\mathbf{q}}$ (cf. Equation 5), which can be efficiently maintained by our sweep line algorithms. Therefore, if we can decompose $\mathcal{F}_P(\mathbf{q})$ with other kernel functions into the aggregate terms, our methods can support these kernel functions.

***Uniform kernel:*** We can decompose $\mathcal{F}_P(\mathbf{q})$ with uniform kernel in the following expression:

$$\mathcal{F}_P(\mathbf{q}) = \frac{w}{b}|R(\mathbf{q})|$$

***Quartic kernel:*** We can decompose $\mathcal{F}_P(\mathbf{q})$ with quartic kernel in the following expression:

$$\mathcal{F}_P(\mathbf{q}) = w\Big(1-\frac{2}{b^2}||\mathbf{q}||_2^2+\frac{1}{b^4}||\mathbf{q}||_2^4\Big)|R(\mathbf{q})| + w\Big(\frac{4}{b^2}-\frac{4||\mathbf{q}||_2^2}{b^4}\Big)\mathbf{q}^T\mathbf{A}_{R_\mathbf{q}}$$
$$+ w\Big(\frac{2||\mathbf{q}||_2^2-2}{b^4}\Big)S_{R_\mathbf{q}} - \frac{4w}{b^4}\mathbf{q}^T\mathbf{C}_{R_\mathbf{q}} + \frac{w}{b^4}Q_{R_\mathbf{q}} - \frac{4w}{b^2}\mathbf{q}^T M_{R_\mathbf{q}}\mathbf{q}$$

where $\mathbf{C}_{R_\mathbf{q}} = \sum_{\mathbf{p}\in R(\mathbf{q})}||\mathbf{p}||_2^2\mathbf{p}$, $Q_{R_\mathbf{q}} = \sum_{\mathbf{p}\in R(\mathbf{q})}||\mathbf{p}||_2^4$ and $M_{R_\mathbf{q}} = \sum_{\mathbf{p}\in R(\mathbf{q})}\mathbf{p}\cdot\mathbf{p}^T$.

Since we can decompose the kernel density function $\mathcal{F}_P(\mathbf{q})$ with the above aggregate values, which are summarized in Table 4, we can extend our methods for supporting these representative kernel functions (cf. Table 2) with the same time complexity.

**Table 4: Aggregate values for all kernel functions.**

| Kernel | Aggregate values |
|---|---|
| Uniform | $\lvert R(\mathbf{q})\rvert$ |
| Epanechnikov | $\lvert R(\mathbf{q})\rvert$, $\mathbf{A}_{R_\mathbf{q}}$, $S_{R_\mathbf{q}}$ |
| Quartic | $\lvert R(\mathbf{q})\rvert$, $\mathbf{A}_{R_\mathbf{q}}$, $S_{R_\mathbf{q}}$, $\mathbf{C}_{R_\mathbf{q}}$, $Q_{R_\mathbf{q}}$, $M_{R_\mathbf{q}}$ |

As a remark, our methods can only support the kernel functions in Table 2, which cannot support some kernel functions (e.g., the Gaussian kernel). The main reason is that we cannot decompose the kernel density function $\mathcal{F}_P(\mathbf{q})$ in terms of the aggregate values (cf. Table 4) for those kernel functions. Nevertheless, the kernel functions in Table 2 are representative in the GIS community (e.g., quartic kernel is the default kernel function in the QGIS [52] and ArcGIS [1] software packages) and have been extensively adopted in different applications (cf. Table 2).

## 3.8 Space Complexity of SLAM

In this section, we proceed to discuss the space complexity for using our methods to generate KDV.

Here, we first consider the method $\text{SLAM}_{\text{SORT}}$ (cf. Algorithm 1). Recall that this method only needs to maintain the additional variables, which are the envelope point set $E(k)$, $\mathbb{L}_\ell$, $\mathbb{U}_\ell$, $\mathbf{A}_{\mathbb{L}_\ell}$, $\mathbf{A}_{\mathbb{U}_\ell}$, $S_{\mathbb{L}_\ell}$ and $S_{\mathbb{U}_\ell}$. Since the sets $\mathbb{L}_\ell$ and $\mathbb{U}_\ell$ contain at most $|E(k)|$ points and $\mathbf{A}_{\mathbb{L}_\ell}$, $\mathbf{A}_{\mathbb{U}_\ell}$, $S_{\mathbb{L}_\ell}$ and $S_{\mathbb{U}_\ell}$ take $O(1)$ space, the worst-case space complexity for using $\text{SLAM}_{\text{SORT}}$ to process $\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_X$ (cf. Figure 4) is at most $O(n)$ (with $|E(k)| \rightarrow n$). Moreover, we can clear and reuse these variables to process each row of pixels (cf. Figure 4) and we need to store the visualization results ($X \times Y$ pixels). Therefore, the space complexity of $\text{SLAM}_{\text{SORT}}$ is $O(XY+n)$ for processing all pixels.

Then, we consider the method $\text{SLAM}_{\text{BUCKET}}$ (cf. Algorithm 2). Even though this method needs to store additional buckets $B_L(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$ and $B_U(\mathbf{q}_{i-1}.x, \mathbf{q}_i.x)$ (where $1 \leq i \leq X+1$) compared with $\text{SLAM}_{\text{SORT}}$, the additional space for storing these buckets is at most $O(n)$. Therefore, the space complexity of $\text{SLAM}_{\text{BUCKET}}$ is the same as $\text{SLAM}_{\text{SORT}}$, which is $O(XY+n)$ for processing all pixels.

Since the method RAO only needs to determine the processing order of the pixels (cf. Figures 4 and 12), which does not incur additional space overhead, the space complexity of $\text{SLAM}_{\text{SORT}}^{\text{(RAO)}}$ and $\text{SLAM}_{\text{BUCKET}}^{\text{(RAO)}}$ remains in $O(XY+n)$.

Based on the above discussion, we conclude the space complexity of our methods in Theorem 4. Observe that all our methods do not incur additional space compared with the method RQS (cf. Section 2.2).

Theorem 4. *The space complexity of $SLAM_{SORT}$, $SLAM_{SORT}^{(RAO)}$, $SLAM_{BUCKET}$, and $SLAM_{BUCKET}^{(RAO)}$ is $O(XY+n)$.*

## 4 EXPERIMENTAL EVALUATION

In this section, we first introduce the experimental settings in Section 4.1. Then, we compare our methods with the state-of-the-art KDV methods, using the default Epanechnikov kernel function, in Section 4.2. Lastly, we also evaluate the efficiency of all methods for other kernel functions, including the uniform and quartic kernels (cf. Table 2), in Section 4.3.

## 4.1 Experimental Settings

We adopt four large-scale real datasets for evaluating the efficiency of all methods, which are summarized in Table 5. All these datasets are the open data from the local governments of different cities, which can be classified into three categories, namely crime events, traffic accidents, and 311 calls. In our experiments, we follow the existing studies [16, 31] and utilize the Scott's rule [57] to determine the default bandwidth value $b$. In addition, we also follow [16] and choose the default resolution size to be $1280 \times 960$.

**Table 5: Datasets.**

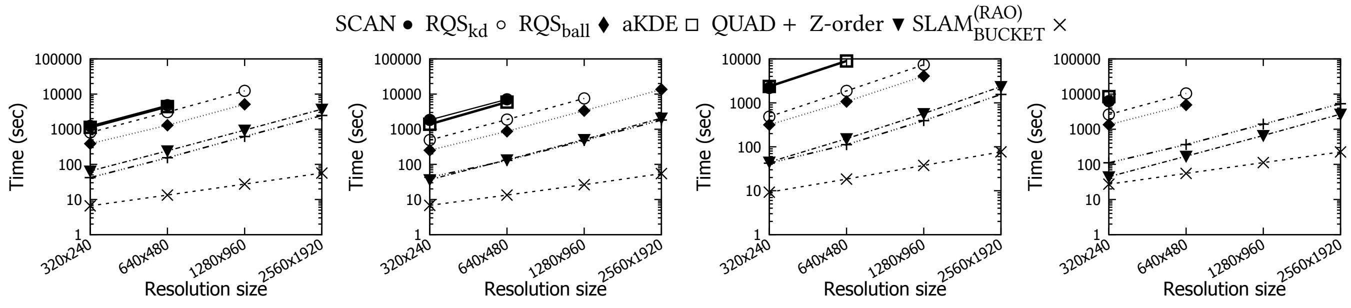
| Dataset name | Dataset size $n$ | Category | Bandwidth $b$ (meters) |
|---|---|---|---|
| Seattle [5] | 862873 | Crime events | 671.39 |
| Los Angeles [2] | 1255668 | Crime events | 1588.47 |
| New York [3] | 1499928 | Traffic accidents | 1062.53 |
| San Francisco [4] | 4333098 | 311 calls | 279.27 |

Table 6: All KDV Methods.

| Method | SCAN | $RQS_{kd}$ | $RQS_{ball}$ | Z-order | aKDE | QUAD | $SLAM_{SORT}$ | $SLAM_{BUCKET}$ | $SLAM_{SORT}^{(RAO)}$ | $SLAM_{BUCKET}^{(RAO)}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Ref. | [57] | Section 2.2 with [9] | Section 2.2 with [44] | [73] | [33] | [16, 19] | Section 3.4 | Section 3.5 | Section 3.4 Section 3.6 | Section 3.5 Section 3.6 |

Table 7: Response time (sec) of all methods, using the default setting of parameters.

| Method | SCAN | $RQS_{kd}$ | $RQS_{ball}$ | Z-order | aKDE | QUAD | $SLAM_{SORT}$ | $SLAM_{BUCKET}$ | $SLAM_{SORT}^{(RAO)}$ | $SLAM_{BUCKET}^{(RAO)}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Seattle | > 14400 | 12351.1 | 5117.44 | 942.43 | > 14400 | 620.68 | 57.77 | 34.99 | 45.19 | **27.47** |
| Los Angeles | > 14400 | 7538.65 | 3375.3 | 509.69 | > 14400 | 478.31 | 54.56 | 33.56 | 42.99 | **26.42** |
| New York | > 14400 | 7388.31 | 4083.54 | 570.77 | > 14400 | 394.49 | 74.49 | 47.41 | 58.83 | **37.63** |
| San Francisco | > 14400 | > 14400 | > 14400 | 662.92 | > 14400 | 1397.53 | 232.44 | 142.9 | 182.69 | **112.29** |



(a) Seattle (b) Los Angeles (c) New York (d) San Francisco

Figure 13: Response time for generating KDV with default bandwidth value, varying the resolution size.

Table 6 shows all KDV methods that are used for efficiency evaluation in our experiments. SCAN is the baseline method, which scans all data points for each pixel to generate KDV. Both $RQS_{kd}$ and $RQS_{ball}$ are the range-query-based solutions (cf. Section 2.2), using the kd-tree and ball-tree, respectively. Z-order [73] is the data sampling method, which achieves the probabilistic error guarantee for the kernel density function $\mathcal{F}_P(\mathbf{q})$. Both aKDE [33] and QUAD [16] incorporate the lower and upper bound functions into the indexing framework to boost the efficiency for evaluating $\mathcal{F}_P(\mathbf{q})$. Compared with these methods, our methods $SLAM_{SORT}$ and $SLAM_{BUCKET}$ can significantly reduce the time complexity for generating exact KDV. By combining the resolution-aware optimization (cf. Section 3.6) with $SLAM_{SORT}$ and $SLAM_{BUCKET}$, both $SLAM_{SORT}^{(RAO)}$ and $SLAM_{BUCKET}^{(RAO)}$ can further reduce the time complexity compared with $SLAM_{SORT}$ and $SLAM_{BUCKET}$, respectively.

We implemented all these methods[2] with C++ and conducted experiments on an Intel i7 3.19GHz PC with 32GB memory. In our experiments, we perform the efficiency evaluation of all methods and only report the response time which is smaller than 14400 sec (i.e., 4 hours).

## 4.2 Efficiency Comparison of KDV Methods

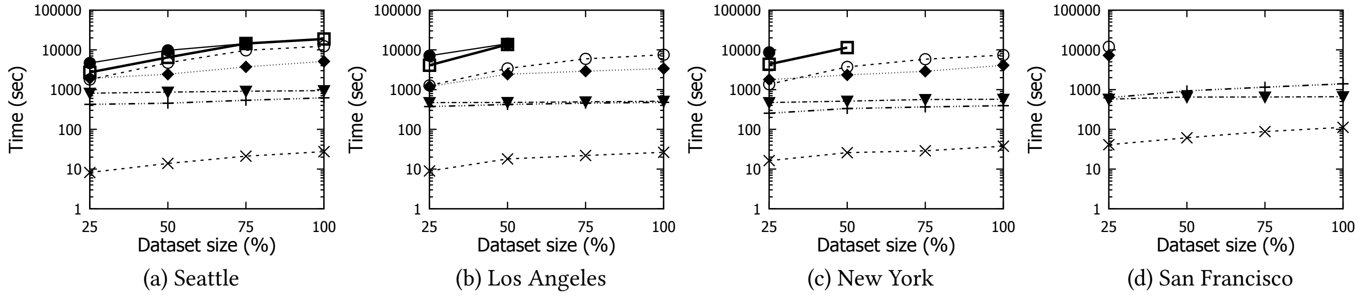
Although all our methods $SLAM_{SORT}$, $SLAM_{BUCKET}$, $SLAM_{SORT}^{(RAO)}$ and $SLAM_{BUCKET}^{(RAO)}$ can significantly reduce the time complexity for generating KDV without increasing the space complexity, we do not know the practical improvement of our methods compared with the existing methods (cf. Table 6). In this section, we investigate the following six questions, which are related to the time and space efficiency issues for generating KDV.

(1) What is the response time of all methods under the default setting of parameters?
(2) How does the resolution size (i.e., $X \times Y$) affect the response time of all methods?
(3) How does the dataset size (i.e., $n$) affect the response time of all methods?
(4) How does the bandwidth value (i.e., $b$) affect the response time of all methods?
(5) What is the response time of all methods for supporting exploratory operations (e.g., zooming and panning)?
(6) How does the dataset size (i.e., $n$) affect the space consumption of all methods?

***Response time of all methods under the default setting of parameters:*** In this experiment, we test the efficiency of all methods, using the default parameters, i.e., we set the default resolution size to be $1280 \times 960$ and adopt the Scott's rule [57] to choose the default bandwidth value $b$ for each dataset. Table 7 shows the response time of all methods. Observe that our methods $SLAM_{SORT}$, $SLAM_{BUCKET}$, $SLAM_{SORT}^{(RAO)}$ and $SLAM_{BUCKET}^{(RAO)}$ can achieve 2.85x to more than two-order-of-magnitude speedup in different datasets compared with the existing methods. The main reason is that all our methods can reduce the time complexity for generating KDV. Since $SLAM_{BUCKET}$ is theoretically more efficient than $SLAM_{SORT}$, $SLAM_{BUCKET}$ can outperform $SLAM_{SORT}$ by 1.57x to 1.65x in all these datasets. By incorporating the resolution-aware optimization (cf. Section 3.6)

[2]The source codes of all methods can be found in the Github repository https://anonymous.4open.science/r/SLAM-F8D8/.

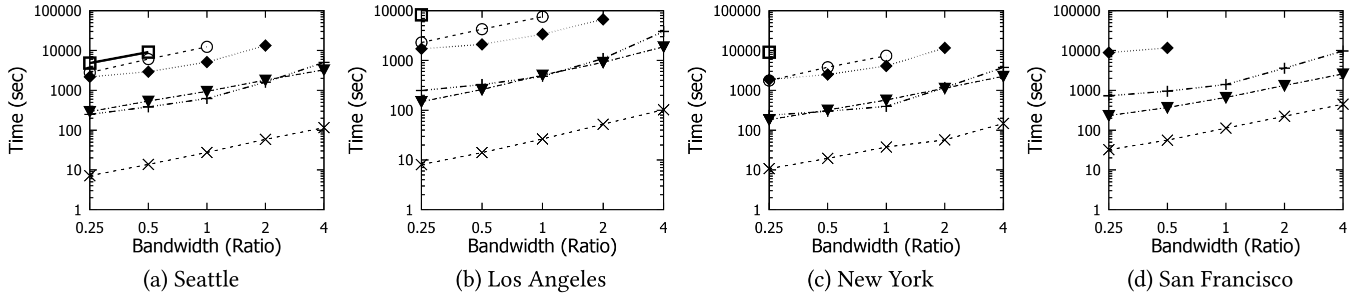
SCAN ● $\text{RQS}_{\text{kd}}$ ○ $\text{RQS}_{\text{ball}}$ ♦ aKDE □ QUAD + Z-order ▼ $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ ×



(a) Seattle (b) Los Angeles (c) New York (d) San Francisco

**Figure 14: Response time for generating KDV with default resolution size and bandwidth value, varying the dataset size.**

SCAN ● $\text{RQS}_{\text{kd}}$ ○ $\text{RQS}_{\text{ball}}$ ♦ aKDE □ QUAD + Z-order ▼ $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ ×



(a) Seattle (b) Los Angeles (c) New York (d) San Francisco

**Figure 15: Response time for generating KDV with default resolution size, varying the bandwidth value.**

into the method $\text{SLAM}_{\text{BUCKET}}$, the method $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ achieves the smallest response time in practice. In the following experiments, we omit the results of $\text{SLAM}_{\text{SORT}}$, $\text{SLAM}_{\text{BUCKET}}$ and $\text{SLAM}^{(\text{RAO})}_{\text{SORT}}$, since these methods are consistently inferior compared with the best method $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$.

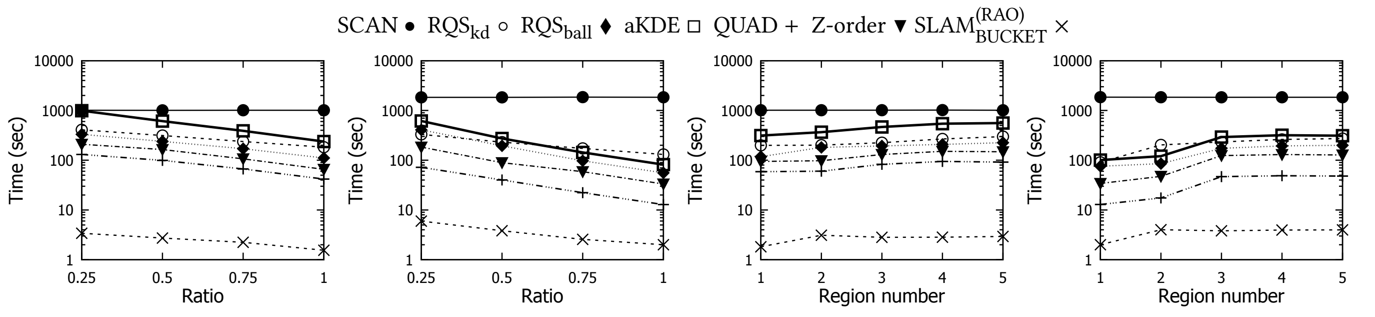
***Response time of all methods with different resolution sizes:*** In this experiment, we investigate how the resolution size affects the response time of all methods. We follow [16] and choose these four resolution sizes, which are $320 \times 240$, $640 \times 480$, $1280 \times 960$ and $2560 \times 1920$, for testing. In Figure 13, the larger the resolution size, the higher the response time of all methods. In addition, since the time complexity of existing methods is $O(XYn)$, the response time of these existing methods can increase by roughly four times, after we adopt the next larger resolution size (e.g., change from $640 \times 480$ to $1280 \times 960$). As our method $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ can reduce the time complexity for generating KDV (with $O(\min(X, Y) \times (\max(X, Y) + n))$), we expect that the response time of $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ increases smaller (i.e., two times) for using the next larger resolution size compared with these existing methods. Therefore, once the resolution size is larger, the time gaps between $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ and all existing methods are also larger.

***Response time of all methods with different dataset sizes:*** We proceed to test how the dataset size affects the response time of all methods. To conduct this experiment, we adopt the random sampling (without replacement) approach to sample 25%, 50%, 75% and 100% (original one) of data points from each dataset (cf. Table 5) for testing. Figure 14 shows the results for all methods. Observe that once we increase the dataset size, all methods need to process more data points. As such, the response time of all methods is also higher. Here, we notice that $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ can outperform the existing methods by a visible margin, no matter which dataset size we adopt.

***Response time of all methods with different bandwidth values:*** Here, we further investigate how the bandwidth value $b$ affects the response time of all methods. To conduct this experiment, we choose five bandwidth values, by multiplying the default bandwidth value with five ratios, which are 0.25, 0.5, 1 (original one), 2 and 4, for testing in each dataset. In Figure 15, observe that once we increase the bandwidth value $b$, the response time of all methods increases. The main reason is that all methods need to scan more data points with the large bandwidth $b$ (e.g., large range value $b$ for the methods $\text{RQS}_{\text{kd}}$ and $\text{RQS}_{\text{ball}}$ (cf. Section 2.2)). In addition, we notice that $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ can consistently outperform the top-2 best competitors, i.e., Z-order and QUAD, by 5.76x to 34.77x.

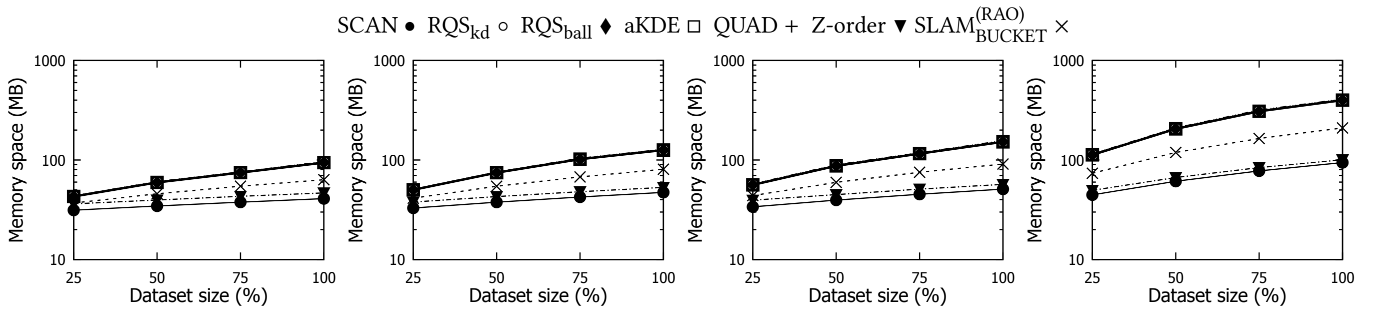
***Response time of all methods with different exploratory operations:*** In practice, users (e.g., Geoscientists, criminologists) need to perform the exploratory operations (cf. Figure 2) to visualize hotspots in different regions (e.g., zooming and panning) and time intervals (e.g., time-based filtering). Here, we investigate the efficiency of all KDV methods for supporting zooming and panning operations, using the Seattle and Los Angeles datasets, where the event time of each data point is from 1st Jan 2019 to 31st Dec 2019. As a remark, we fix the resolution size to be $1280 \times 960$.

In the first experiment, we test the efficiency for the zooming operation. First, we use the minimum bounding rectangle to cover



(a) Seattle (Zooming) (b) Los Angeles (Zooming) (c) Seattle (Panning) (d) Los Angeles (Panning)

**Figure 16: Response time for generating KDV with zooming (a and b) and panning (c and d) operations in the Seattle (a and c) and Los Angeles (b and d) datasets, where the event time of each data point is from 1st Jan 2019 to 31st Dec 2019.**



(a) Seattle (b) Los Angeles (c) New York (d) San Francisco

**Figure 17: Space consumption for generating KDV with default resolution size and bandwidth value, varying the dataset size.**

each city, e.g., Seattle. Then, we generate four visualized regions, by multiplying the height and width of this rectangle with four ratios, i.e., 0.25, 0.5, 0.75 and 1 (the original one). Here, the smaller ratio denotes that we zoom in to this city. Figure 16a and Figure 16b show the response time of all methods. Since we fix the resolution size to be $1280 \times 960$ for each visualized region and the zoomed regions contain data points with higher density, each pixel in the zoomed region with smaller ratio (e.g., 0.25) can have a larger number of data points that are within the bandwidth $b$ (i.e., each method, except SCAN, needs to process more data points). Therefore, the smaller the ratio, the higher the response time of each method. Observe that $\text{SLAM}_{\text{BUCKET}}^{(\text{RAO})}$ can achieve at least one to two-order-of-magnitude speedup in most of the cases compared with the existing methods, regardless of the visualized regions. As a remark, since the competitors (e.g., QUAD) adopt the filter-and-refinement approach to improve the efficiency for generating KDV and the filtering performance in the Los Angeles dataset is better, the time gap between $\text{SLAM}_{\text{BUCKET}}^{(\text{RAO})}$ and the best competitor (i.e., QUAD) in the Seattle dataset is larger compared with the Los Angeles dataset.

In the second experiment, we test the efficiency for the panning operation. Like the first experiment, we also use the minimum bounding rectangle to cover each city (let the size be $H \times W$). Then, we randomly generate five rectangles, which are inside the minimum bounding rectangle, with the same size $0.5H \times 0.5W$, as the visualized regions. In Figure 16c and Figure 16d, we observe that $\text{SLAM}_{\text{BUCKET}}^{(\text{RAO})}$ can achieve one-order-of-magnitude speedup in most of the cases compared with the best competitor.

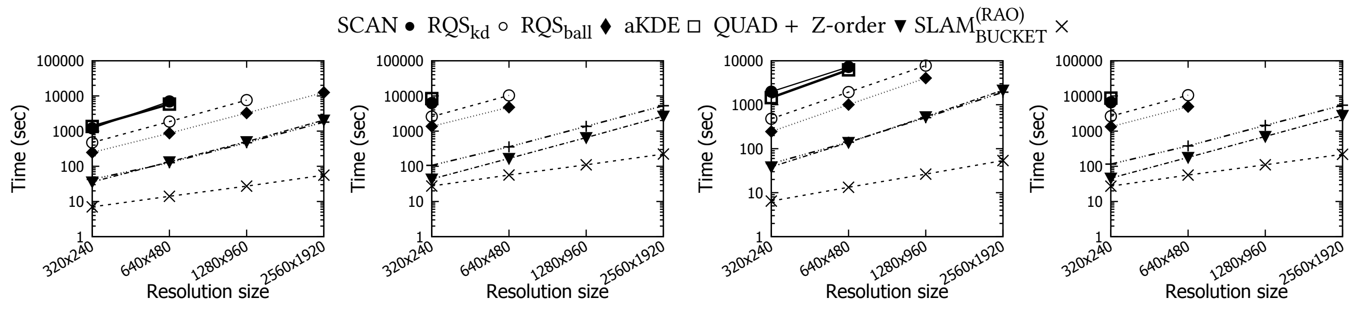
Based on the results of these two experiments, we further observe that $\text{SLAM}_{\text{BUCKET}}^{(\text{RAO})}$ only takes less than 6 sec (i.e., near real-time) for generating each KDV with these exploratory operations, which cannot be achieved by other state-of-the-art methods.

***Space consumption of all methods with different dataset sizes:*** In this experiment, we investigate how the dataset size affects the space consumption of all methods (by sampling 25%, 50%, 75% and 100% (original one) of data points from each dataset). Since $\text{SLAM}_{\text{BUCKET}}^{(\text{RAO})}$ does not increase the worst-case space complexity (cf. Theorem 4) compared with existing methods, the space consumption of all methods is similar (cf. Figure 17), no matter which dataset size we adopt.

## 4.3 Efficiency Evaluation for Other Kernels

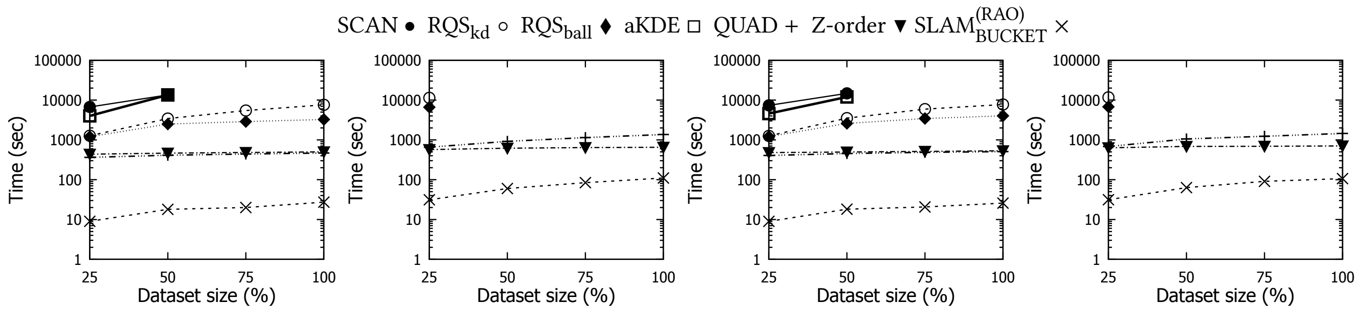
In this section, we proceed to test the efficiency for generating KDV with other kernel functions, including uniform and quartic kernels. Here, we adopt the Los Angeles and San Francisco datasets for testing.

In the first experiment, we test how the resolution size affects the response time for generating KDV, using the uniform and quartic kernels. Figure 18 shows the results for all methods. Since the existing methods and our method $\text{SLAM}_{\text{BUCKET}}^{(\text{RAO})}$ does not incur the large time overhead for supporting these kernel functions, the response time of all methods is similar to the results in Figure 13b and Figure 13d. Here, we can observe that $\text{SLAM}_{\text{BUCKET}}^{(\text{RAO})}$ can consistently outperform the existing methods by a visible margin in many test cases. In addition, once we increase the resolution size, the time



(a) Los Angeles (Uniform) (b) San Francisco (Uniform) (c) Los Angeles (Quartic) (d) San Francisco (Quartic)

**Figure 18: Response time for generating KDV in Los Angeles (a and c) and San Francisco (b and d) datasets with uniform (a and b) and quartic (c and d) kernels, varying the resolution size.**



(a) Los Angeles (Uniform) (b) San Francisco (Uniform) (c) Los Angeles (Quartic) (d) San Francisco (Quartic)

**Figure 19: Response time for generating KDV in Los Angeles (a and c) and San Francisco (b and d) datasets with uniform (a and b) and quartic (c and d) kernels, varying the dataset size.**

gaps between SLAM$^{(RAO)}_{BUCKET}$ and the existing methods are larger, due to the lower time complexity for our methods.

In the second experiment, we test how the dataset size affects the response time for generating KDV, using the uniform and quartic kernels. In Figure 19, we can observe that SLAM$^{(RAO)}_{BUCKET}$ can achieve one to two-order-of-magnitude speedup in many test cases for these two kernel functions compared with different state-of-the-art methods.

## 5 RELATED WORK

Kernel Density Visualization (KDV) has been widely used in different types of applications. Some representative examples include traffic accident hotspot detection [62, 65], crime hotspot detection [34, 37], disease outbreak detection [19, 56] and ecological modeling [27, 60]. However, KDV is a computationally expensive operation [16, 31, 73], which is not scalable to large-scale datasets (e.g., > 1 million data points) and high resolution size (e.g., $1280 \times 960$). In this section, we review six camps of research studies, which are mostly related to this work.

***Function approximation methods:*** Some researchers propose to approximate the kernel density function $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 1). Raykar et al. [54] and Yang et al. [67] adopt the fast Gauss transform to approximately compute $\mathcal{F}_P(\mathbf{q})$. On the other hand, Chan et al. [16, 19, 21, 22], Gan et al. [31] and Gray et al [33] develop the lower and upper bound functions for $\mathcal{F}_P(\mathbf{q})$ and further incorporate these bound functions into the indexing framework to approximately compute $\mathcal{F}_P(\mathbf{q})$. Although these function approximation methods can improve the practical efficiency for generating KDV, these methods cannot reduce the time complexity for this operation and can only provide the approximate result for each density value $\mathcal{F}_P(\mathbf{q})$.

***Data sampling methods:*** In this camp of research studies, Zheng et al. [73–75] and Phillips et al. [49–51] propose the advanced data sampling methods to sample the given original dataset and then evaluate the modified kernel density function, based on the reduced dataset. Since the size of the reduced dataset can be much smaller than the original dataset, these methods can significantly improve the efficiency for generating KDV. However, like the function approximation methods, these methods can only provide the approximate result for each density value $\mathcal{F}_P(\mathbf{q})$. In addition, these methods still need to evaluate the exact KDV for the reduced dataset, which can still be time-consuming. As a remark, since our methods can reduce the time-complexity for evaluating exact KDV, our methods can seamlessly combine with these data sampling methods to further improve the efficiency for generating approximate KDV.

***Range-query-based methods:*** In Section 2.2, we recall that computing the kernel density function $\mathcal{F}_P(\mathbf{q})$ can be cast as solving the range query problem for each pixel $\mathbf{q}$. Therefore, the efficient methods for solving the range query problem can be used to improve the efficiency for generating KDV. In the literature, many efficient index structures have been developed to improve the performance for solving the range query problem. Among most of these index structures, both kd-tree [9] and ball-tree [44] are the representative and efficient solutions for the low dimensional datasets [47]. Although these methods can improve the practical efficiency for generating KDV, they cannot theoretically reduce the time complexity for this operation. Therefore, these methods can normally provide inferior efficiency compared with our methods, especially for large bandwidth $b$.

***Sweep line methods:*** In both spatial database, computational geometry and GIS communities, sweep line methods (or plane sweep methods) have been widely adopted to improve the efficiency for different query processing or data analysis tasks. Some representative examples include skyline queries [35, 58], spatial join queries [7, 12, 13], range sum queries [23, 61], delaunay triangulation [6, 26], Voronoi diagram [29, 30] and line segmentation intersection [11, 24, 63]. Compared with our work, these research studies do not consider the complex kernel density function $\mathcal{F}_P(\mathbf{q})$. Therefore, their sweep line methods cannot be directly applied for generating KDV with smaller time complexity.

***Parallel/distributed and hardware-based methods:*** In the literature, there are also many parallel/distributed and hardware-based methods to boost the efficiency for generating KDV. Some representative examples include MapReduce [73], GPU [40, 48, 71] and FPGA [32]. In this work, we mainly focus on the single CPU setting and leave the combination of our methods and these methods in the future work.

***Visual analytic systems:*** Recently, many visual analytic systems [25, 36, 41–43] have been developed to analyze hotspots, based on KDV, for different types of applications, including crime hotspot detection [25, 42], and disease outbreak detection [36, 41, 43]. These systems offer different types of exploratory operations, e.g., zooming, panning, bandwidth selection, attribute-based filtering, and time-based filtering, for generating KDV, in order to accomplish the more complicated visual analytic tasks for understanding hotspots. Although these visual analytic tasks normally involve massive KDV operations, which are not scalable to large datasets and high resolution sizes, these research studies do not develop efficient algorithms for supporting KDV. By incorporating our SLAM into these systems, we reckon that this can significantly save the precious waiting time of domain experts.

## 6 CONCLUSION

In this paper, we study kernel density visualization (KDV), which has become a de facto visual analytic tool in many applications, including hotspot detection and ecological modeling. Since KDV is a time-consuming operation, which is not scalable to large datasets and high resolution sizes, many research studies [16, 31, 32] have complained about its efficiency issues. Although many recent research studies propose to improve the efficiency for this operation, all these studies can only provide approximate results for generating KDV in order to improve the efficiency. Worse still, some of these studies cannot reduce the time complexity for this operation. To address this issue, we develop two sweep line algorithms (SLAM), namely $\text{SLAM}_{\text{SORT}}$ and $\text{SLAM}_{\text{BUCKET}}$, which can theoretically reduce the time complexity for generating exact KDV. By incorporating the resolution-aware optimization (RAO) into these two methods, namely $\text{SLAM}^{(\text{RAO})}_{\text{SORT}}$ and $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$, we can further achieve the lowest time complexity for generating exact KDV. Compared with the state-of-the-art solutions, all our methods can achieve one to two-order-of-magnitude speedup in many test cases and our best method $\text{SLAM}^{(\text{RAO})}_{\text{BUCKET}}$ can efficiently support KDV with exploratory operations in practice.

In the future, we plan to support other types of GIS operations, e.g., $K$-function [8] and network $K$-function [46]. Furthermore, we will extend our methods to support other commonly used kernel functions (e.g., Gaussian kernel) and other types of KDV (e.g., NKDV [20], STKDV [18], and STNKDV [55]). In addition, we will develop the real-time KDV system, based on SLAM, to support some meaningful applications with large-scale location datasets, e.g., visualizing the distribution of COVID-19 cases, using high resolution sizes (e.g., $2560 \times 1920$). Moreover, we will develop the new plugin, which adopts SLAM to support efficient KDV generation, for both QGIS [52] and ArcGIS [1].

## ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Plan of China (No.2019YFB2102100), the Science and Technology Development Fund Macau (SKL-IOTSC-2021-2023, 0015/2019/AKP), University of Macau (MYRG2019-00119-FST), IRCMS/19-20/H01, Hong Kong RGC Projects 12201518, 12202221, C2004-21GF, and C6030-18GF.

## REFERENCES

[1] [n. d.]. ArcGIS. http://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/how-kernel-density-works.htm.

[2] [n. d.]. Los Angeles Open Data. https://data.lacity.org/A-Safe-City/Crime-Data-from-2010-to-2019/63jg-8b9z.

[3] [n. d.]. NYC Open Data. https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95.

[4] [n. d.]. San Francisco Open Data. https://data.sfgov.org/City-Infrastructure/311-Cases/vw6y-z8j6.

[5] [n. d.]. Seattle Open Data. https://data.seattle.gov/Public-Safety/SPD-Crime-Data-2008-Present/tazs-3rd5.

[6] Pankaj K. Agarwal, Lars Arge, and Ke Yi. 2005. I/O-Efficient Construction of Constrained Delaunay Triangulations. In *ESA*. 355–366. https://doi.org/10.1007/11561071_33

[7] Lars Arge, Octavian Procopiuc, Sridhar Ramaswamy, Torsten Suel, and Jeffrey Scott Vitter. 1998. Scalable Sweeping-Based Spatial Join. In *VLDB*. 570–581. http://www.vldb.org/conf/1998/p570.pdf

[8] Adrian Baddeley, Ege Rubak, and Rolf Turner. 2015. *Spatial point patterns: methodology and applications with R*. CRC press.

[9] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (1975), 509–517.

[10] Michal Bíl, Richard Andrášik, and Zbyněk Janoška. 2013. Identification of hazardous road locations of traffic accidents by means of kernel density estimation and cluster significance evaluation. *Accident Analysis & Prevention* 55 (2013), 265–273. https://doi.org/10.1016/j.aap.2013.03.003

[11] Jean-Daniel Boissonnat and Franco P. Preparata. 2000. Robust Plane Sweep for Intersecting Segments. *SIAM Journal of Computing* 29, 5 (2000), 1401–1421. https://doi.org/10.1137/S0097539797329373

[12] Panagiotis Bouros and Nikos Mamoulis. 2017. A Forward Scan based Plane Sweep Algorithm for Parallel Interval Joins. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1346–1357. https://doi.org/10.14778/3137628.3137644

[13] Panagiotis Bouros, Nikos Mamoulis, Dimitrios Tsitsigkos, and Manolis Terrovitis. 2021. In-Memory Interval Joins. *The VLDB Journal* 30, 4 (2021), 667–691. https://doi.org/10.1007/s00778-020-00639-0

[14] Ellen E. Brandell, Nicholas M. Fountain-Jones, Marie L. J. Gilbertson, Paul C. Cross, Peter J. Hudson, Douglas W. Smith, Daniel R. Stahler, Craig Packer, and Meggan E. Craft. 2021. Group density, disease, and season shape territory size and overlap of social carnivores. *Journal of Animal Ecology* 90, 1 (2021), 87–101. https://doi.org/10.1111/1365-2656.13294 arXiv:https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/1365-2656.13294

[15] Spencer Chainey, Lisa Tompson, and Sebastian Uhlig. 2008. The Utility of Hotspot Mapping for Predicting Spatial Patterns of Crime. *Security Journal* 21, 1 (01 Feb 2008), 4–28. https://doi.org/10.1057/palgrave.sj.8350066

[16] Tsz Nam Chan, Reynold Cheng, and Man Lung Yiu. 2020. QUAD: Quadratic-Bound-based Kernel Density Visualization. In *SIGMOD*. 35–50. https://doi.org/10.1145/3318464.3380561

[17] Tsz Nam Chan, Pak Lon Ip, Leong Hou U, Byron Choi, and Jianliang Xu. 2022. SAFE: A Share-and-Aggregate Bandwidth Exploration Framework for Kernel Density Visualization. *Proc. VLDB Endow.* 15, 3 (2022), 513–526.

[18] Tsz Nam Chan, Pak Lon Ip, Leong Hou U, Byron Choi, and Jianliang Xu. 2022. SWS: A Complexity-Optimized Solution for Spatial-Temporal Kernel Density Visualization. *Proc. VLDB Endow.* 15, 4 (2022), 814–827.

[19] Tsz Nam Chan, Pak Lon Ip, Leong Hou U, Weng Hou Tong, Shivansh Mittal, Ye Li, and Reynold Cheng. 2021. KDV-Explorer: A Near Real-Time Kernel Density Visualization System for Spatial Analysis. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2655–2658. http://www.vldb.org/pvldb/vol14/p2655-chan.pdf

[20] Tsz Nam Chan, Zhe Li, Leong Hou U, Jianliang Xu, and Reynold Cheng. 2021. Fast Augmentation Algorithms for Network Kernel Density Visualization. *Proc. VLDB Endow.* 14, 9 (2021), 1503–1516. http://www.vldb.org/pvldb/vol14/p1503-chan.pdf

[21] Tsz Nam Chan, Leong Hou U, Reynold Cheng, Man Lung Yiu, and Shivansh Mittal. To appear. Efficient Algorithms for Kernel Aggregation Queries. *IEEE Transactions on Knowledge and Data Engineering* (To appear).

[22] Tsz Nam Chan, Man Lung Yiu, and Leong Hou U. 2019. KARL: Fast Kernel Aggregation Queries. In *ICDE*. 542–553. https://doi.org/10.1109/ICDE.2019.00055

[23] Dong-Wan Choi, Chin-Wan Chung, and Yufei Tao. 2014. Maximizing Range Sum in External Memory. *ACM Transactions on Database Systems* 39, 3 (2014), 21:1–21:44. https://doi.org/10.1145/2629477

[24] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. 2008. *Computational geometry: algorithms and applications, 3rd Edition.* Springer. https://www.worldcat.org/oclc/227584184

[25] Jose Florencio de Queiroz Neto, Emanuele Marques dos Santos, Creto Augusto Vidal, and David S. Ebert. 2020. A Visual Analytics Approach to Facilitate Crime Hotspot Analysis. *Computer Graphics Forum* 39, 3 (2020), 139–151. https://doi.org/10.1111/cgf.13969

[26] Vid Domiter and Borut Zalik. 2008. Sweep-line algorithm for constrained Delaunay triangulation. *International Journal of Geographical Information Science* 22, 4 (2008), 449–462. https://doi.org/10.1080/13658810701492241

[27] Jianquan Dong, Jian Peng, Yanxu Liu, Sijing Qiu, and Yinan Han. 2020. Integrating spatial continuous wavelet transform and kernel density estimation to identify ecological corridors in megacities. *Landscape and Urban Planning* 199 (2020), 103815. https://doi.org/10.1016/j.landurbplan.2020.103815

[28] C. H. Fleming, W. F. Fagan, T. Mueller, K. A. Olson, P. Leimgruber, and J. M. Calabrese. 2015. Rigorous home range estimation with movement data: a new autocorrelated kernel density estimator. *Ecology* 96, 5 (2015), 1182–1188. https://doi.org/10.1890/14-2010.1 arXiv:https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.1890/14-2010.1

[29] Steven Fortune. 1986. A Sweepline Algorithm for Voronoi Diagrams. In *SoCG*. 313–322. https://doi.org/10.1145/10515.10549

[30] Steven Fortune. 1987. A Sweepline Algorithm for Voronoi Diagrams. *Algorithmica* 2 (1987), 153–174. https://doi.org/10.1007/BF01840357

[31] Edward Gan and Peter Bailis. 2017. Scalable Kernel Density Classification via Threshold-Based Pruning. In *SIGMOD*. 945–959.

[32] A. Gramacki. 2017. *Nonparametric Kernel Density Estimation and Its Computational Aspects*. Springer International Publishing. https://books.google.com.hk/books?id=PCpEDwAAQBAJ

[33] Alexander G. Gray and Andrew W. Moore. 2003. Nonparametric Density Estimation: Toward Computational Tractability. In *SDM*. 203–211.

[34] Timothy Hart and Paul Zandbergen. 2014. Kernel density estimation and hotspot mapping: examining the influence of interpolation method, grid cell size, and bandwidth on crime forecasting. *Policing: An International Journal of Police Strategies and Management* 37 (2014), 305–323. https://doi.org/10.3390/s80603601

[35] Casper Kejlberg-Rasmussen, Yufei Tao, Konstantinos Tsakalidis, Kostas Tsichlas, and Jeonghun Yoon. 2013. I/O-efficient planar range skyline and attrition priority queues. In *PODS*. 103–114. https://doi.org/10.1145/2463664.2465225

[36] Seokyeon Kim, Seongmin Jeong, Insoo Woo, Yun Jang, Ross Maciejewski, and David S. Ebert. 2018. Data Flow Analysis and Visualization for Spatiotemporal Statistical Data without Trajectory Information. *IEEE Transactions on Visualization and Computer Graphics* 24, 3 (2018), 1287–1300. https://doi.org/10.1109/TVCG.2017.2666146

[37] Ourania Kounadi, Alina Ristea, Adelson Araujo, and Michael Leitner. 2020. A systematic review on spatial crime forecasting. *Crime Science* 9, 7 (2020), 1–22.

[38] Pei-Fen Kuo, Dominique Lord, and Troy Duane Walden. 2013. Using geographical information systems to organize police patrol routes effectively by grouping hotspots of crash and crime data. *Journal of Transport Geography* 30 (2013), 138–148. https://doi.org/10.1016/j.jtrangeo.2013.04.006

[39] P.C. Lai, Chit-Ming Wong, Anthony Hedley, S.V. Lo, P.Y. Leung, J Kong, and G.M. Leung. 2004. Understanding the Spatial Clustering of Severe Acute Respiratory Syndrome (SARS) in Hong Kong. *Environmental Health Perspectives* 112, 15 (2004), 1550–1556. https://doi.org/10.1289/ehp.7117

[40] Ove Daae Lampe and Helwig Hauser. 2011. Interactive visualization of streaming data with Kernel Density Estimation. In *PacificVis*. 171–178. https://doi.org/10.1109/PACIFICVIS.2011.5742387

[41] Ross Maciejewski, Ryan Hafen, Stephen Rudolph, Stephen G. Larew, Michael A. Mitchell, William S. Cleveland, and David S. Ebert. 2011. Forecasting Hotspots - A Predictive Analytics Approach. *IEEE Transactions on Visualization and Computer Graphics* 17, 4 (2011), 440–453. https://doi.org/10.1109/TVCG.2010.82

[42] Ross Maciejewski, Stephen Rudolph, Ryan Hafen, Ahmad M. Abusalah, Mohamed Yakout, Mourad Ouzzani, William S. Cleveland, Shaun J. Grannis, and David S. Ebert. 2010. A Visual Analytics Approach to Understanding Spatiotemporal Hotspots. *IEEE Transactions on Visualization and Computer Graphics* 16, 2 (2010), 205–220. https://doi.org/10.1109/TVCG.2009.100

[43] Ross Maciejewski, Stephen Rudolph, Ryan Hafen, Ahmad M. Abusalah, Mohamed Yakout, Mourad Ouzzani, William S. Cleveland, Shaun J. Grannis, Michael Wade, and David S. Ebert. 2008. Understanding syndromic hotspots - a visual analytics approach. In *VAST*. 35–42. https://doi.org/10.1109/VAST.2008.4677354

[44] Andrew W. Moore. 2000. The Anchors Hierarchy: Using the Triangle Inequality to Survive High Dimensional Data. In *UAI*. 397–405.

[45] Norihiko Muroga, Yoko Hayama, Takehisa Yamamoto, Akihiro Kurogi, Tomoyuki Tsuda, and Toshiyuki Tsutsui. 2011. The 2010 Foot-and-Mouth Disease Epidemic in Japan. *The Journal of Veterinary Medical Science* 74, 4 (2011), 399–404. https://doi.org/10.1292/jvms.11-0271

[46] A. Okabe and K. Sugihara. 2012. *Spatial analysis along networks: statistical and computational methods*. Wiley. https://books.google.com.hk/books?id=48GRqj51_W8C

[47] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[48] Alexandre Perrot, Romain Bourqui, Nicolas Hanusse, Frédéric Lalanne, and David Auber. 2015. Large Interactive Visualization of Density Functions on Big Data Infrastructure. In *LDAV*. 99–106. https://doi.org/10.1109/LDAV.2015.7348077

[49] Jeff M. Phillips. 2013. $\epsilon$-Samples for Kernels. In *SODA*. 1622–1632. https://doi.org/10.1137/1.9781611973105.116

[50] Jeff M. Phillips and Wai Ming Tai. 2018. Improved Coresets for Kernel Density Estimates. In *SODA*. 2718–2727. https://doi.org/10.1137/1.9781611975031.173

[51] Jeff M. Phillips and Wai Ming Tai. 2018. Near-Optimal Coresets of Kernel Density Estimates. In *SOCG*. 66:1–66:13. https://doi.org/10.4230/LIPIcs.SoCG.2018.66

[52] QGIS Development Team. 2009. *QGIS Geographic Information System*. Open Source Geospatial Foundation. http://qgis.osgeo.org

[53] Jerry Ratcliffe and Spencer Chainey. 2005. *GIS and crime mapping*. John Wiley.

[54] Vikas C. Raykar, Ramani Duraiswami, and Linda H. Zhao. 2010. Fast Computation of Kernel Estimators. *Journal of Computational and Graphical Statistics* 19, 1 (2010), 205–220. https://doi.org/10.1198/jcgs.2010.09046

[55] Benjamin Romano and Zhe Jiang. 2017. Visualizing Traffic Accident Hotspots Based on Spatial-Temporal Network Kernel Density Estimation. In *SIGSPATIAL*. ACM, 98:1–98:4. https://doi.org/10.1145/3139958.3139981

[56] Natalya Rybnikova, Richard G Stevens, David I Gregorio, Holly Samociuk, and Boris A Portnov. 2018. Kernel density analysis reveals a halo pattern of breast cancer incidence in Connecticut. *Spatial and Spatio-temporal Epidemiology* 26 (2018), 143–151.

[57] D. W. Scott. 1992. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley. https://books.google.com.hk/books?id=7crCUS_F2ocC

[58] Cheng Sheng and Yufei Tao. 2011. On finding skylines in external memory. In *PODS*. 107–116. https://doi.org/10.1145/1989284.1989298

[59] Xun Shi. 2010. Selection of bandwidth type and adjustment side in kernel density estimation over inhomogeneous backgrounds. *International Journal of Geographical Information Science* 24, 5 (2010), 643–660. https://doi.org/10.1080/13658810902950625

[60] Xun Shi, Meifang Li, Olivia Hunter, Bart Guetti, Angeline Andrew, Elijah Stommel, Walter Bradley, and Margaret Karagas. 2019. Estimation of environmental exposure: interpolation, kernel density estimation or snapshotting. *Annals of GIS* 25, 1 (2019), 1–8. https://doi.org/10.1080/19475683.2018.1555188

[61] Yufei Tao, Xiaocheng Hu, Dong-Wan Choi, and Chin-Wan Chung. 2013. Approximate MaxRS in Spatial Databases. *Proceedings of the VLDB Endowment* 6, 13 (2013), 1546–1557. https://doi.org/10.14778/2536258.2536266

[62] Lalita Thakali, Tae J. Kwon, and Liping Fu. 2015. Identification of crash hotspots using kernel density estimation and kriging methods: a comparison. *Journal of Modern Transportation* 23, 2 (2015), 93–106. https://doi.org/10.1007/s40534-015-0068-0

[63] Jan Vahrenhold. 2007. Line-segment intersection made in-place. *Computational Geometry* 38, 3 (2007), 213–230. https://doi.org/10.1016/j.comgeo.2006.09.001

[64] Zuyuan Wang, Christian Ginzler, and Lars T. Waser. 2020. Assessing structural changes at the forest edge using kernel density estimation. *Forest Ecology and Management* 456 (2020), 117639. https://doi.org/10.1016/j.foreco.2019.117639

[65] Kun Xie, Kaan Ozbay, Abdullah Kurkcu, and Hong Yang. 2017. Analysis of Traffic Crashes Involving Pedestrians Using Big Data: Investigation of Contributing Factors and Identification of Hotspots. *Risk Analysis* 37, 8 (2017), 1459–1476. https://EconPapers.repec.org/RePEc:wly:riskan:v:37:y:2017:i:8:p:1459-1476

[66] Liting Xu, Shuhe Zhao, Sophia Shuang Chen, Cheng Yu, and Buyun Lei. 2020. Analysis of arable land distribution around human settlements in the riparian area of Lake Tanganyika in Africa. *Applied Geography* 125 (2020), 102344. https://doi.org/10.1016/j.apgeog.2020.102344

[67] Changjiang Yang, Ramani Duraiswami, and Larry S. Davis. 2004. Efficient Kernel Machines Using the Improved Fast Gauss Transform. In *NIPS*. 1561–1568. http://papers.nips.cc/paper/2550-efficient-kernel-machines-using-the-improved-fast-gauss-transform

[68] Suyan Yi, Hongwei Wang, Shengtian Yang, Ling Xie, Yibo Gao, and Chen Ma. 2021. Spatial and Temporal Characteristics of Hand-Foot-and-Mouth Disease and Its Response to Climate Factors in the Ili River Valley Region of China. *International Journal of Environmental Research and Public Health* 18, 4 (2021).

[69] Hao Yu, Pan Liu, Jun Chen, and Hao Wang. 2014. Comparative analysis of the spatial analysis methods for hotspot identification. *Accident Analysis and Prevention* 66 (2014), 80 – 88. https://doi.org/10.1016/j.aap.2014.01.017

[70] Kunxiaojia Yuan, Xiaoqiang Cheng, Zhipeng Gui, Fa Li, and Huayi Wu. 2019. A quad-tree-based fast and adaptive Kernel Density Estimation algorithm for heat-map generation. *International Journal of Geographical Information Science* 33, 12 (2019), 2455–2476. https://doi.org/10.1080/13658816.2018.1555831

[71] Guiming Zhang, A-Xing Zhu, and Qunying Huang. 2017. A GPU-accelerated adaptive kernel density estimation approach for efficient point pattern analysis on spatial big data. *International Journal of Geographical Information Science* 31, 10 (2017), 2068–2097. https://doi.org/10.1080/13658816.2017.1324975

[72] Xiangyu Zhao and Jiliang Tang. 2018. Crime in Urban Areas: A Data Mining Perspective. *SIGKDD Explorations* 20, 1 (2018), 1–12. https://doi.org/10.1145/3229329.3229331

[73] Yan Zheng, Jeffrey Jestes, Jeff M. Phillips, and Feifei Li. 2013. Quality and efficiency for kernel density estimates in large data. In *SIGMOD*. 433–444.

[74] Yan Zheng, Yi Ou, Alexander Lex, and Jeff M. Phillips. 2021. Visualization of Big Spatial Data Using Coresets for Kernel Density Estimates. *IEEE Transactions on Big Data* 7, 3 (2021), 524–534. https://doi.org/10.1109/TBDATA.2019.2913655

[75] Yan Zheng and Jeff M. Phillips. 2015. $L_\infty$ Error and Bandwidth Selection for Kernel Density Estimates of Large Data. In *SIGKDD*. 1533–1542. https://doi.org/10.1145/2783258.2783357