SWS: A Complexity-Optimized Solution for Spatial-Temporal Kernel Density Visualization

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

Pak Lon Ip
University of Macau
paklonip@um.edu.mo

Leong Hou U
University of Macau
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

Spatial-temporal kernel density visualization (STKDV) has been extensively used in a wide range of applications, e.g., disease outbreak analysis, traffic accident hotspot detection, and crime hotspot detection. While STKDV can provide accurate and comprehensive data visualization, computing STKDV is time-consuming, which is not scalable to large-scale datasets. To address this issue, we develop a new sliding-window-based solution (SWS), which theoretically reduces the time complexity for generating STKDV, without increasing the space complexity. Moreover, we incorporate SWS with the progressive visualization framework, which can continuously output partial visualization results to users (from coarse to fine), until users satisfy the visualization. Our experimental studies on five large-scale datasets show that SWS achieves 1.71x to 24x speedup compared with the state-of-the-art methods.

1 INTRODUCTION

Data visualization [18, 51, 56] is an important tool for understanding a dataset. Among most of the data visualization tools, kernel-density-estimation-based visualization (or kernel density visualization (KDV)) [14, 51] has been extensively used in a wide range of applications, including disease outbreak analysis [4, 15, 20, 26, 50, 65], traffic accident hotspot detection [28, 33, 35, 61], crime hotspot detection [12, 13, 25, 27, 31, 39, 66], health informatics [29, 60], and resource management [70, 71]. Therefore, different types of scientific/ geographical software, including QGIS [9], ArcGIS [1], CrimeStat [2], KDV-Explorer [15], and Scikit-learn [41], can also support this operation. Figure 1 illustrates an example usage of KDV for visualizing the density distribution of the COVID-19 cases in Hong Kong from February 2020 to February 2021.

To generate the hotspot map (cf. Figure 1), existing studies in KDV [14, 17, 21, 25, 61, 67] utilize the following kernel density function $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 1) to determine the color of each pixel $\mathbf{q}$, where P, w and $K(\mathbf{q}, \mathbf{p})$ denote the set of two dimensional spatial data points (e.g., latitude and longitude values of COVID-19 cases), the positive weight value (i.e., constant) and the kernel function (e.g., Epanechnikov kernel), respectively.

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

However, one major drawback for using KDV is that this method does not incorporate the occurrence time of data points, which may generate misleading visualization to the domain experts (e.g., geographical users). Using Figure 2 as a simplified example of COVID-19 cases over time, we can observe that the data points near $\mathbf{q}_2$ have similar time, which indicates a community outbreak, while the time gap of those data points near $\mathbf{q}_1$ is large, which only indicates the sporadic cases. Therefore, the position $\mathbf{q}_2$ in July should require the attention of epidemiologists rather than $\mathbf{q}_1$. However, based on Equation 1, since both the pixels $\mathbf{q}_1$ and $\mathbf{q}_2$ are surrounded by the same number of (i.e., three) data points with similar distances, both $\mathbf{q}_1$ and $\mathbf{q}_2$ would have the similar density values (or color).

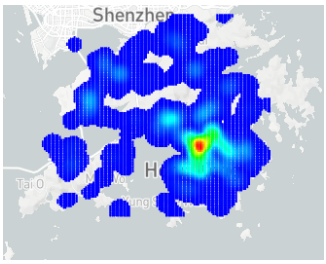


Figure 1: A hotspot map (generated by KDV) for the density distribution of COVID-19 cases in Hong Kong (obtained from [3]), where the red color denotes the high density region.

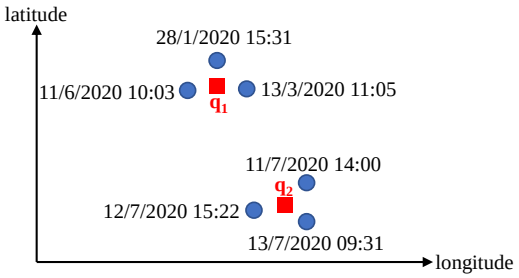


Figure 2: A simplified example of COVID-19 cases over time.

In addition to the above example, many recent studies in different applications (e.g., crime hotspot detection [27], disease outbreak analysis [20], and traffic accident hotspot detection [33]) also point out the same drawback, i.e., ignoring the time of each data point for using KDV, which are quoted as follows.

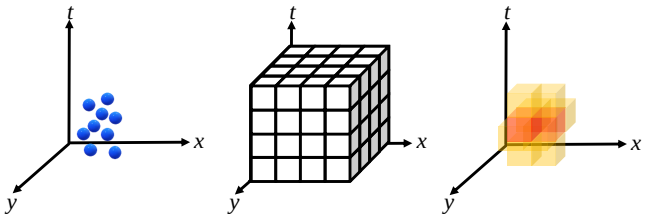
- *"Ignoring the temporal component of crime deprives researchers and practitioners of the opportunity to target specific time periods with elevated crime risks."* [27]

Table 1: Commonly-used spatial and temporal kernel functions.

Kernel	$K_{\text{space}}(\mathbf{q}, \mathbf{p})$	$K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p})$	Representative application(s)
Triangular	$\begin{cases} 1-\gamma_s dist(\mathbf{q},\mathbf{p}) & \text{if } dist(\mathbf{q},\mathbf{p}) \le \frac{1}{\gamma_s} \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} 1-\gamma_t dist(t_\mathbf{q},t_\mathbf{p}) & \text{if } dist(t_\mathbf{q},t_\mathbf{p}) \le \frac{1}{\gamma_t} \\ 0 & \text{otherwise} \end{cases}$	Resource management [70]
Epanechnikov	$\begin{cases} 1-\gamma_s^2 dist(\mathbf{q},\mathbf{p})^2 & \text{if } dist(\mathbf{q},\mathbf{p}) \le \frac{1}{\gamma_s} \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} 1-\gamma_t^2 dist(t_\mathbf{q},t_\mathbf{p})^2 & \text{if } dist(t_\mathbf{q},t_\mathbf{p}) \le \frac{1}{\gamma_t} \\ 0 & \text{otherwise} \end{cases}$	Disease outbreak analysis [20, 65] Traffic accident hotspot detection [28] Health informatics [60]
Quartic	$\begin{cases} (1-\gamma_s^2 dist(\mathbf{q},\mathbf{p})^2)^2 & \text{if } dist(\mathbf{q},\mathbf{p}) \le \frac{1}{\gamma_s} \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} (1-\gamma_t^2 dist(t_\mathbf{q},t_\mathbf{p})^2)^2 & \text{if } dist(t_\mathbf{q},t_\mathbf{p}) \le \frac{1}{\gamma_t} \\ 0 & \text{otherwise} \end{cases}$	Crime hotspot detection [12, 31]

- *"... ignoring the temporal aspect (or considering it secondary to the geographic component) would undermine our ability to analyze the underlying dynamics and/or to visualize the likelihood of re-occurrence of the disease..."* [20]
- *"... the STKDE space-time cube made it easier to detect the spatio-temporal patterns of traffic violations than did the traditional hotspots map."* [33]

Due to the importance for incorporating the time component into KDV, many existing studies [20, 27, 28, 33, 39, 65, 71] propose to adopt spatial-temporal kernel density visualization (STKDV), in which they aim to visualize the colored space-time cube (cf. Figure 3c), instead of the hotspot map (like Figure 1). In practice, we display the space-time cube as the time-evolving hotspot map to users. For details, please refer to Section 6.5.



(a) Data points (b) Space-time cube (c) STKDV

Figure 3: Generate STKDV for a set of (blue) data points, where red, orange and white colors denote the high, middle and low density values of each voxel, respectively.

In order to generate STKDV, we need to first divide the cube into a set of voxels (i.e., small cubes in Figure 3b), denoted as $(\mathbf{q}, t_\mathbf{q})$, where $\mathbf{q}$ and $t_\mathbf{q}$ represent the two-dimensional spatial position and the time of the voxel, respectively. Then, we color each voxel based on the spatial-temporal kernel density function [26, 33, 39] (cf. Equation 2), given a set $\widehat{P}$ of spatial-temporal data points $(\mathbf{p}, t_\mathbf{p})$.

$$\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q}) = \sum_{(\mathbf{p},t_\mathbf{p})\in\widehat{P}} w \cdot K_{\text{space}}(\mathbf{q},\mathbf{p}) \cdot K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p}) \quad (2)$$

where $K_{\text{space}}(\mathbf{q}, \mathbf{p})$ and $K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p})$ denote the spatial kernel and temporal kernel, respectively. Table 1 summarizes different types of commonly-used kernel functions for STKDV, which are supported in the famous QGIS and ArcGIS software packages.

Even though STKDV has been extensively used in different domains, computing STKDV is very time-consuming. Using the space-time cube with size $128\times128\times128$ and the New York traffic accident dataset [6] (with nearly 1.5 million data points) as an example, generating STKDV for this dataset takes 9.43 trillion operations in the worst case. As such, this operation cannot scale well to handle large-scale datasets with high visualization quality (i.e., large number of voxels), especially for the usage of the exploratory analysis [20, 26, 39]. Many existing studies also complain about this inefficiency issue for computing STKDV.

- *"... on computing the first step of the visualization pipeline, space-time kernel density estimation (STKDE), which is most computationally expensive."* [50]
- *"The temporal extension of the KDE is known as the space-time kernel density estimation (STKDE) and essentially maps a volume of disease intensity along the space-time domain (Nakaya and Yano, 2010). However, the above methods are computationally intensive..."* [26]
- *"Expanding the KDE algorithm to integrate the temporal dimension is computationally demanding..."* [20]

In this paper, we develop an efficient sliding-window-based solution (SWS), which, to the best of our knowledge, is **the first solution that theoretically reduces the time complexity for generating STKDV, without increasing the space complexity**. In addition, we further develop a general progressive visualization framework, which can continuously output partial STKDV (from coarse to fine) to users. Experimental results show that our method SWS **achieves 1.71x-24x speedup** compared with the state-of-the-art methods.

The rest of the paper is organized as follows. We first discuss the background in Section 2. Then, we present our method SWS in Section 3. Next, we extend SWS to other kernel functions in Section 4. After that, we illustrate the progressive visualization framework for STKDV in Section 5. Later, we show our experimental results in Section 6. Then, we discuss the related work in Section 7. Lastly, we conclude our paper in Section 8. The appendix of proofs, pseudocode and implementation details can be found in Section 9.

2 PRELIMINARIES

In this section, we formally define our problem for STKDV in Section 2.1. Then, we illustrate how to adapt the range-query-based solution (RQS) as the baseline method in Section 2.2.

2.1 Problem Statement for STKDV

Recall from Section 1, we need to determine the color of each voxel, using the spatial-temporal kernel density function (cf. Equation 2), in the 3D cube (cf. Figure 3b) with size $X \times Y \times T$, where X, Y and T are the numbers of voxels in x-axis, y-axis and t-axis, respectively.

Problem 1. *Given a cube, with size $X \times Y \times T$, of voxels and a dataset $\widehat{P} = \{(\mathbf{p}_1, t_{\mathbf{p}_1}), (\mathbf{p}_2, t_{\mathbf{p}_2}), ..., (\mathbf{p}_n, t_{\mathbf{p}_n})\}$ with n spatial-temporal data points, we compute the kernel density value $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q})$ (cf. Equation 2) for each voxel $(\mathbf{q}, t_\mathbf{q})$.*

Observe from Equation 2, the kernel density function $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q})$ depends on both the spatial kernel $K_{\text{space}}(\mathbf{q}, \mathbf{p})$ and the temporal kernel $K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p})$. Since most of the existing studies (cf. Table 1) mainly utilize either the triangular, Epanechnikov or quartic kernels to generate the STKDV (especially for Epanechnikov kernel), we specifically focus on these kernel functions in this paper.

2.2 Range-Query-based Solution (RQS)

Different types of scientific and geographical software, e.g., Scikit-learn [41], QGIS [46], and ArcGIS [1], implement the range-query-based solution (RQS) to boost the efficiency for generating KDV. Here, we illustrate how to extend RQS for generating STKDV, i.e., solving Problem 1. Observe from Table 1, we find that only those data points $(\mathbf{p}, t_\mathbf{p})$ with $dist(\mathbf{q}, \mathbf{p}) \leq \frac{1}{\gamma_s}$ and $dist(t_\mathbf{q}, t_\mathbf{p}) \leq \frac{1}{\gamma_t}$ can contribute to $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q})$ (cf. Equation 2) for a given voxel $(\mathbf{q}, t_\mathbf{q})$. Therefore, we can first obtain the reduced set $R_\mathbf{q}$ of data points (cf. Equation 3), which can be cast as the range query problem, and then evaluate the kernel density function $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q})$ (cf. Equation 4), based on the reduced set $R_\mathbf{q}$.

$$R_\mathbf{q} = \left\{(\mathbf{p}, t_\mathbf{p}) \in \widehat{P} \;\middle|\; dist(\mathbf{q}, \mathbf{p}) \leq \frac{1}{\gamma_s} \text{ and } dist(t_\mathbf{q}, t_\mathbf{p}) \leq \frac{1}{\gamma_t}\right\} \quad (3)$$

$$\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q}) = \sum_{(\mathbf{p}, t_\mathbf{p}) \in R_\mathbf{q}} w \cdot K_{\text{space}}(\mathbf{q}, \mathbf{p}) \cdot K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p}) \quad (4)$$

In existing work [19, 24, 41], different types of index approaches, e.g., kd-tree and ball-tree, can be used to boost the efficiency for obtaining $R_\mathbf{q}$, which are summarized in Table 2.

Table 2: Worst case time and space complexity for finding the reduced set $R_\mathbf{q}$, i.e., solving range query, using different types of index structures.

Index structure	Time complexity	Space complexity
kd-tree [19, 41]	$O(n^{\frac{2}{3}} + \lvert R_\mathbf{q}\rvert)$	$O(n)$
ball-tree [24, 41]	$O(n + \lvert R_\mathbf{q}\rvert)$	$O(n)$

Even though RQS can be possible to improve the efficiency for generating STKDV, the response time can still be long once the size of set $R_\mathbf{q}$ is large, i.e., large values for $\frac{1}{\gamma_s}$ and $\frac{1}{\gamma_t}$ in Equation 3. Theoretically, once $\gamma_s \to 0$ and $\gamma_t \to 0$, the size $|R_\mathbf{q}| \to n$. In this case, the time complexity for generating STKDV remains the same as the basic approach (i.e., scan without filtering), which is $O(XYTn)$.

Since the size of the cube is $X \times Y \times T$ and both kd-tree and ball-tree take $O(n)$ space, the space complexity of the method RQS is $O(XYT + n)$.

3 SLIDING-WINDOW-BASED SOLUTION (SWS)

Even though the method RQS can improve the efficiency for calculating $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q})$, RQS cannot reduce the time complexity for generating STKDV (cf. Problem 1), which remains in $O(XYTn)$ time.

In this section, we propose a sliding-window-based solution (SWS) that only takes $O(XY(T + n))$ time to generate STKDV, using the commonly-used Epanechnikov kernel for $K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p})$. Here, we do not assume any kernel type for $K_{\text{space}}(\mathbf{q}, \mathbf{p})$.

3.1 Sliding Window for Temporal Dimension

In our method SWS, the core idea is to maintain the sliding window in the temporal dimension (cf. Figure 4) around the voxel $(\mathbf{q}, t_\mathbf{q})$. Here, we sort the data points in $\widehat{P}$ such that $t_{\mathbf{p}_1} \leq t_{\mathbf{p}_2} \leq ... \leq t_{\mathbf{p}_n}$. Observe that this sliding window $W(t_\mathbf{q})$ needs to cover the data points $(\mathbf{p}, t_\mathbf{p})$ such that $dist(t_\mathbf{q}, t_\mathbf{p}) \leq \frac{1}{\gamma_t}$, i.e., $K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p}) > 0$ (cf. Table 1), where:

$$W(t_\mathbf{q}) = \left\{(\mathbf{p}, t_\mathbf{p}) \in \widehat{P} \;\middle|\; dist(t_\mathbf{q}, t_\mathbf{p}) \leq \frac{1}{\gamma_t}\right\}$$

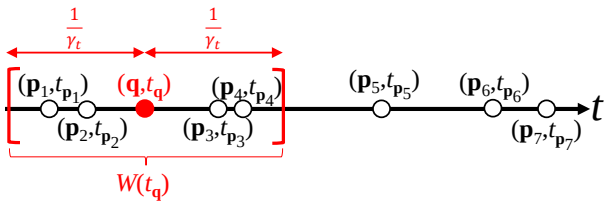


Figure 4: The sliding window $W(t_\mathbf{q})$ for the voxel $(\mathbf{q}, t_\mathbf{q})$.

Since we can ensure that those data points in $W(t_\mathbf{q})$ should have $K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p}) > 0$ (we use Epanechnikov kernel here), we can conclude that:

$$\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q}) = \sum_{(\mathbf{p}, t_\mathbf{p}) \in W(t_\mathbf{q})} w \cdot K_{\text{space}}(\mathbf{q}, \mathbf{p}) \cdot (1 - {\gamma_t}^2 dist(t_\mathbf{q}, t_\mathbf{p})^2)$$

By adopting some simple algebraic operations, we can express $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q})$ as:

$$\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q}) = w(1-{\gamma_t}^2 t_\mathbf{q}^2) \cdot S^{(0)}_{W(t_\mathbf{q})}(\mathbf{q}) + 2w{\gamma_t}^2 t_\mathbf{q} \cdot S^{(1)}_{W(t_\mathbf{q})}(\mathbf{q}) - w{\gamma_t}^2 \cdot S^{(2)}_{W(t_\mathbf{q})}(\mathbf{q}) \quad (5)$$

where:

$$S^{(i)}_{W(t_\mathbf{q})}(\mathbf{q}) = \sum_{(\mathbf{p}, t_\mathbf{p}) \in W(t_\mathbf{q})} t_\mathbf{p}^i \cdot K_{\text{space}}(\mathbf{q}, \mathbf{p}) \quad (6)$$

The sliding window $W(t_\mathbf{q})$ maintains/stores these three statistical terms $S^{(i)}_{W(t_\mathbf{q})}(\mathbf{q})$, where $i = 0, 1, 2$.

3.2 SWS: An Incremental Algorithm

After we have illustrated the concept of sliding window, we propose an efficient incremental algorithm, namely SWS, for improving the efficiency to evaluate the kernel density function $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_{\mathbf{q}_n})$ for the next voxel $(\mathbf{q}, t_{\mathbf{q}_n})$ in the temporal dimension, i.e., fixing the spatial position $\mathbf{q}$ and change the temporal coordinate from $t_\mathbf{q}$ to $t_{\mathbf{q}_n}$.

Observe from Figure 5, once we shift from the voxel $(\mathbf{q}, t_\mathbf{q})$ to $(\mathbf{q}, t_{\mathbf{q}_n})$, we need to insert the green point $(\mathbf{p}_5, t_{\mathbf{p}_5})$ and delete the yellow points $(\mathbf{p}_1, t_{\mathbf{p}_1})$ and $(\mathbf{p}_2, t_{\mathbf{p}_2})$ in order to update to the next window $W(t_{\mathbf{q}_n})$ (i.e., blue window). Here, we denote these two sets of points as $I(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))$ (cf. Equation 7) and $D(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))$ (cf. Equation 8), where:

$$I(W(t_\mathbf{q}), W(t_{\mathbf{q}_n})) = W(t_{\mathbf{q}_n}) \setminus W(t_\mathbf{q}) \quad (7)$$

$$D(W(t_\mathbf{q}), W(t_{\mathbf{q}_n})) = W(t_\mathbf{q}) \setminus W(t_{\mathbf{q}_n}) \quad (8)$$

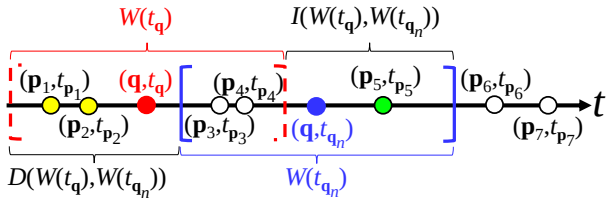


Figure 5: Movement of the sliding window from $W(t_{\mathbf{q}})$ (red dashed window) to $W(t_{\mathbf{q}_n})$ (blue window), after we change from the voxel $(\mathbf{q}, t_{\mathbf{q}})$ to $(\mathbf{q}, t_{\mathbf{q}_n})$. Green and yellow points denote the newly inserted and deleted points, respectively.

Recall from Section 3.1, each sliding window $W(t_{\mathbf{q}})$ needs to maintain the statistical terms $S^{(0)}_{W(t_{\mathbf{q}})}(\mathbf{q})$, $S^{(1)}_{W(t_{\mathbf{q}})}(\mathbf{q})$ and $S^{(2)}_{W(t_{\mathbf{q}})}(\mathbf{q})$ (cf. Equation 6). Therefore, we also need to update these terms to $S^{(0)}_{W(t_{\mathbf{q}_n})}(\mathbf{q})$, $S^{(1)}_{W(t_{\mathbf{q}_n})}(\mathbf{q})$ and $S^{(2)}_{W(t_{\mathbf{q}_n})}(\mathbf{q})$, once we have updated the window from $W(t_{\mathbf{q}})$ to $W(t_{\mathbf{q}_n})$. Lemma 1 shows how we can incrementally obtain $S^{(i)}_{W(t_{\mathbf{q}_n})}(\mathbf{q})$, given the statistical terms $S^{(i)}_{W(t_{\mathbf{q}})}(\mathbf{q})$, with $i = 0, 1, 2$.

Lemma 1. *Given two windows $W(t_{\mathbf{q}})$ and $W(t_{\mathbf{q}_n})$ for the voxels $(\mathbf{q}, t_{\mathbf{q}})$ and $(\mathbf{q}, t_{\mathbf{q}_n})$, respectively, and the statistical terms $S^{(i)}_{W(t_{\mathbf{q}})}(\mathbf{q})$, where $i = 0, 1, 2$, for the window $W(t_{\mathbf{q}})$, we can represent $S^{(i)}_{W(t_{\mathbf{q}_n})}(\mathbf{q})$ with the following equation.*

$$S^{(i)}_{W(t_{\mathbf{q}_n})}(\mathbf{q}) = S^{(i)}_{W(t_{\mathbf{q}})}(\mathbf{q}) - \sum_{(\mathbf{p},t_{\mathbf{p}})\in D(W(t_{\mathbf{q}}),W(t_{\mathbf{q}_n}))} t^i_{\mathbf{p}} \cdot K_{space}(\mathbf{q},\mathbf{p}) + \sum_{(\mathbf{p},t_{\mathbf{p}})\in I(W(t_{\mathbf{q}}),W(t_{\mathbf{q}_n}))} t^i_{\mathbf{p}} \cdot K_{space}(\mathbf{q},\mathbf{p}) \quad (9)$$

Observe from Equation 9, once we update the statistical terms $S^{(i)}_{W(t_{\mathbf{q}_n})}(\mathbf{q})$ of the window $W(t_{\mathbf{q}_n})$, we only need to scan additional data points in $I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))$ and remove those points in $D(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))$ (cf. Figure 5), which only take $O(|I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))| + |D(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))|)$ time. Therefore, we can also obtain the kernel density value $\mathcal{F}_{\mathcal{P}}(\mathbf{q}, t_{\mathbf{q}_n})$ (cf. Equation 5, replace $t_{\mathbf{q}}$ by $t_{\mathbf{q}_n}$) in $O(|I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))| + |D(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))|)$ time (cf. Lemma 2).

Lemma 2. *Given two windows $W(t_{\mathbf{q}})$ and $W(t_{\mathbf{q}_n})$ for the voxels $(\mathbf{q}, t_{\mathbf{q}})$ and $(\mathbf{q}, t_{\mathbf{q}_n})$, respectively, and the statistical terms $S^{(i)}_{W(t_{\mathbf{q}})}(\mathbf{q})$, where $i = 0, 1, 2$, for the window $W(t_{\mathbf{q}})$, we can compute the kernel density function $\mathcal{F}_{\mathcal{P}}(\mathbf{q}, t_{\mathbf{q}_n})$ for the voxel $(\mathbf{q}, t_{\mathbf{q}_n})$ and update the statistical terms $S^{(i)}_{W(t_{\mathbf{q}_n})}(\mathbf{q})$ for the window $W(t_{\mathbf{q}_n})$ with $O(|I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))| + |D(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))|)$ time.*

With the above concepts, we discuss how to efficiently compute all the kernel density values $\mathcal{F}_{\mathcal{P}}(\mathbf{q}, t_{\mathbf{q}})$ (cf. Equation 5) for all voxels which are along the time-axis (or t-axis) with the same spatial position $\mathbf{q}$ (cf. Figure 6), in which we denote these T voxels as $(\mathbf{q}, t_{\mathbf{q}_1})$, $(\mathbf{q}, t_{\mathbf{q}_2})$,..., $(\mathbf{q}, t_{\mathbf{q}_T})$.

Figure 7 illustrates multiple sliding windows along the t-axis which correspond to different voxels. Suppose that we have computed the density value $\mathcal{F}_{\mathcal{P}}(\mathbf{q}, t_{\mathbf{q}_1})$ for the voxel $(\mathbf{q}, t_{\mathbf{q}_1})$ and also maintained the statistical terms $S^{(i)}_{W(t_{\mathbf{q}_1})}(\mathbf{q})$ (cf. Equation 6) for the

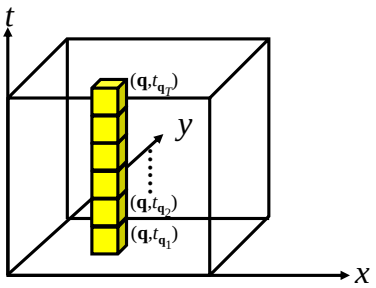


Figure 6: Compute the kernel density values for all yellow voxels that are along the time axis (or t-axis) with the same spatial position $\mathbf{q}$.

red window $W_{t_{\mathbf{q}_1}}$, which take $O(|W_{t_{\mathbf{q}_1}}|)$ time, we can then update the statistical terms of the consecutive window $W_{t_{\mathbf{q}_2}}$, i.e., $S^{(i)}_{W(t_{\mathbf{q}_2})}(\mathbf{q})$ and compute $\mathcal{F}_{\mathcal{P}}(\mathbf{q}, t_{\mathbf{q}_2})$ with $O(|I(W(t_{\mathbf{q}_1}), W(t_{\mathbf{q}_2}))| + |D(W(t_{\mathbf{q}_1}), W(t_{\mathbf{q}_2}))|)$ time, based on Lemma 2. By adopting the same approach for other windows (e.g., pink and black windows), we can conclude that the time complexity for obtaining the density values for all voxels with the same spatial position along the time axis, i.e., all yellow voxels in Figure 6, is[1]:

$$O\Big(|W_{t_{\mathbf{q}_1}}| + \sum_{i=1}^{T-1} |I(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}}))| + \sum_{i=1}^{T-1} |D(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}}))| + T\Big) \quad (10)$$

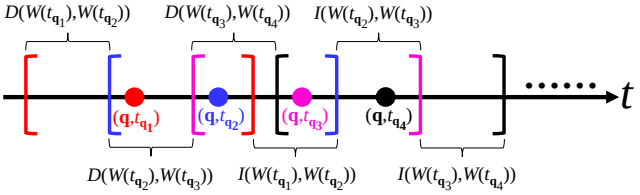


Figure 7: Illustration of multiple sliding windows, i.e., red, blue, pink, and black, with the voxels $(\mathbf{q}, t_{\mathbf{q}_1})$, $(\mathbf{q}, t_{\mathbf{q}_2})$, $(\mathbf{q}, t_{\mathbf{q}_3})$ and $(\mathbf{q}, t_{\mathbf{q}_4})$, respectively.

In Lemma 3, we state that this time complexity (cf. Equation 10) for computing the density values of all yellow voxels is $O(T + n)$. We include the formal proof of this lemma in the appendix (cf. Section 9.1).

Lemma 3. *The time complexity (i.e., Equation 10) for computing the density values of all T voxels $(\mathbf{q}, t_{\mathbf{q}_1})$, $(\mathbf{q}, t_{\mathbf{q}_2})$,..., $(\mathbf{q}, t_{\mathbf{q}_T})$ is $O(T+n)$.*

Once we can use $O(T + n)$ time to obtain all density values for all voxels along the t-axis with the same spatial position, we can also conclude that the method SWS only takes $O(XY(T + n))$ time to generate STKDV, as there are XY two-dimensional spatial positions in the visualization, as stated in Theorem 1.

Theorem 1. *Sliding-window-based solution (SWS) takes $O(XY(T + n))$ time to generate STKDV (cf. Problem 1), using the Epanechnikov kernel for $K_{time}(t_{\mathbf{q}}, t_{\mathbf{p}})$.*

As a remark, our method SWS only stores one sliding window for processing all voxels $(\mathbf{q}, t_{\mathbf{q}_1})$, $(\mathbf{q}, t_{\mathbf{q}_2})$,..., $(\mathbf{q}, t_{\mathbf{q}_T})$ (i.e., yellow voxels

[1]The $+T$ term is the access cost of the T voxels.

in Figure 6). This sliding window can be cleared and then reused for the next T voxels with another spatial position. Therefore, SWS only incurs $O(n)$ additional space for maintaining this sliding window and its statistical terms, which does not increase the worst case space complexity for generating STKDV, i.e., $O(XYT + n)$ space (cf. Lemma 4).

Lemma 4. *The space complexity of sliding-window-based solution (SWS) is $O(XYT + n)$ for generating STKDV (cf. Problem 1), using the Epanechnikov kernel for $K_{time}(t_\mathbf{q}, t_\mathbf{p})$.*

The pseudocode and the implementation details for SWS can be found in the appendix (cf. Section 9.4).

4 SWS FOR OTHER TEMPORAL KERNELS

In Section 3, we have illustrated how to utilize SWS to improve the efficiency for generating STKDV using Epanechnikov kernel as $K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p})$. Here, we ask a question, can we extend this method to other kernel functions in Table 1 with similar time and space efficiency guarantee (cf. Theorem 1 and Lemma 4, respectively)? In this section, we give an affirmative answer for this question.

4.1 Quartic kernel

We consider the following kernel density function with quartic kernel as $K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p})$.

$$\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q}) = \sum_{(\mathbf{p}, t_\mathbf{p}) \in W(t_\mathbf{q})} w \cdot K_{\text{space}}(\mathbf{q}, \mathbf{p}) \cdot (1 - \gamma_t{}^2 dist(t_\mathbf{q}, t_\mathbf{p})^2)^2$$

Observe that we can also decompose this kernel density function as:

$$\begin{aligned}\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q}) &= w(1 - 2\gamma_t^2 t_\mathbf{q}^2 + \gamma_t^4 t_\mathbf{q}^4) \cdot S^{(0)}_{W(t_\mathbf{q})}(\mathbf{q}) \\ &+ w(4\gamma_t^2 t_\mathbf{q} - 4\gamma_t^4 t_\mathbf{q}^3) \cdot S^{(1)}_{W(t_\mathbf{q})}(\mathbf{q}) \\ &+ w(6\gamma_t^4 t_\mathbf{q}^2 - 2\gamma_t^2) \cdot S^{(2)}_{W(t_\mathbf{q})}(\mathbf{q}) \\ &- 4w\gamma_t^4 t_\mathbf{q} \cdot S^{(3)}_{W(t_\mathbf{q})}(\mathbf{q}) + w\gamma_t^4 \cdot S^{(4)}_{W(t_\mathbf{q})}(\mathbf{q})\end{aligned}$$

Once we maintain the statistical terms $S^{(i)}_{W(t_\mathbf{q})}(\mathbf{q})$ (cf. Equation 6), where $0 \le i \le 4$, in the sliding window $W(t_\mathbf{q})$ for each voxel $(\mathbf{q}, t_\mathbf{q})$, using the similar idea in Section 3.2, we can directly extend both Lemmas 1, 2, 3, 4 and Theorem 1 for quartic kernel, i.e., $O(XY(T+n))$ time and $O(XYT + n)$ space for generating STKDV.

4.2 Triangular kernel

We proceed to consider the kernel density function with triangular kernel as $K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p})$.

$$\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q}) = \sum_{(\mathbf{p}, t_\mathbf{p}) \in W(t_\mathbf{q})} w \cdot K_{\text{space}}(\mathbf{q}, \mathbf{p}) \cdot (1 - \gamma_t dist(t_\mathbf{q}, t_\mathbf{p}))$$

However, unlike the Epanechnikov and quartic kernels, we cannot decompose the kernel density function into the linear combination of the statistical terms (like Equation 5), since we cannot simply expand the Euclidean distance $dist(t_\mathbf{q}, t_\mathbf{p})$. Nevertheless, we notice that:

$$dist(t_\mathbf{q}, t_\mathbf{p}) = \begin{cases} t_\mathbf{q} - t_\mathbf{p} & \text{if } t_\mathbf{q} > t_\mathbf{p} \\ t_\mathbf{p} - t_\mathbf{q} & \text{otherwise} \end{cases}$$

Therefore, once we have maintained the left and right sliding windows, $W_L(t_\mathbf{q})$ and $W_R(t_\mathbf{q})$, respectively, for the voxel $(\mathbf{q}, t_\mathbf{q})$ (cf. Figure 8), we can obtain:

$$\begin{aligned}\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q}) &= wS^{(0)}_{W(t_\mathbf{q})}(\mathbf{q}) - w\gamma_t \Big(t_\mathbf{q} S^{(0)}_{W_L(t_\mathbf{q})}(\mathbf{q}) - S^{(1)}_{W_L(t_\mathbf{q})}(\mathbf{q}) \\ &+ S^{(1)}_{W_R(t_\mathbf{q})}(\mathbf{q}) - t_\mathbf{q} S^{(0)}_{W_R(t_\mathbf{q})}(\mathbf{q})\Big) \qquad (11)\end{aligned}$$

where $S^{(0)}_{W(t_\mathbf{q})}(\mathbf{q})$, $S^{(0)}_{W_L(t_\mathbf{q})}(\mathbf{q})$, $S^{(1)}_{W_L(t_\mathbf{q})}(\mathbf{q})$, $S^{(0)}_{W_R(t_\mathbf{q})}(\mathbf{q})$ and $S^{(1)}_{W_R(t_\mathbf{q})}(\mathbf{q})$ are the statistical terms (cf. Equation 6) with respect to either $W(t_\mathbf{q})$, $W_L(t_\mathbf{q})$ or $W_R(t_\mathbf{q})$.

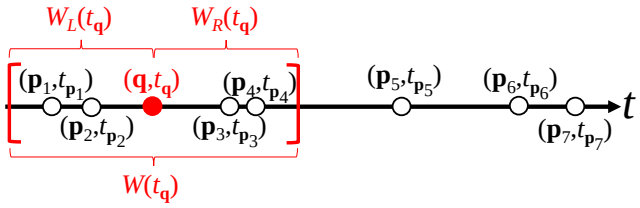


Figure 8: Left sliding window $W_L(t_\mathbf{q})$ and right sliding window $W_R(t_\mathbf{q})$ for the voxel $(\mathbf{q}, t_\mathbf{q})$.

Recall from Section 3.2, we remain to discuss how to efficiently update these statistical terms for the next voxel $(\mathbf{q}, t_{\mathbf{q}_n})$. Here, we claim that it takes $O(|I(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))| + |D(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))| + |C(t_\mathbf{q}, t_{\mathbf{q}_n})|)$ to obtain all these statistical terms and also the density value $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_{\mathbf{q}_n})$ for the next voxel $(\mathbf{q}, t_{\mathbf{q}_n})$ in Lemma 5, where $C(t_\mathbf{q}, t_{\mathbf{q}_n})$ denotes the set of points $(\mathbf{p}, t_\mathbf{p})$ in $\widehat{P}$ with the time $t_\mathbf{p}$ inside the interval $[t_\mathbf{q}, t_{\mathbf{q}_n}]$ (cf. Equation 12 and Figure 9), $I(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))$ and $D(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))$ are defined in Equations 7 and 8, respectively. We leave the proof of Lemma 5 in the appendix (cf. Section 9.2).

$$C(t_\mathbf{q}, t_{\mathbf{q}_n}) = \{(\mathbf{p}, t_\mathbf{p}) \in \widehat{P} | t_\mathbf{q} \le t_\mathbf{p} \le t_{\mathbf{q}_n}\} \qquad (12)$$

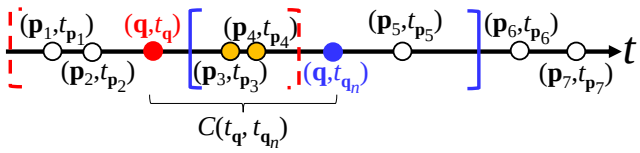


Figure 9: The orange points $(\mathbf{p}_3, t_{\mathbf{p}_3})$ and $(\mathbf{p}_4, t_{\mathbf{p}_4})$ are inside the set $C(t_\mathbf{q}, t_{\mathbf{q}_n})$.

Lemma 5. *Given two windows $W(t_\mathbf{q})$ and $W(t_{\mathbf{q}_n})$ for the voxels $(\mathbf{q}, t_\mathbf{q})$ and $(\mathbf{q}, t_{\mathbf{q}_n})$, respectively, and the statistical terms for the window $W(t_\mathbf{q})$, we can compute the kernel density function $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_{\mathbf{q}_n})$, using the triangular kernel, for the voxel $(\mathbf{q}, t_{\mathbf{q}_n})$ and update the statistical terms for the window $W(t_{\mathbf{q}_n})$ in $O(|I(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))| + |D(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))| + |C(t_\mathbf{q}, t_{\mathbf{q}_n})|)$ time.*

Compared with Lemma 2, even though we need to spend the additional cost $|C(t_\mathbf{q}, t_{\mathbf{q}_n})|$ for obtaining the density value $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_{\mathbf{q}_n})$, we claim that we can still use $O(T + n)$ time to compute all the density values of all T voxels $(\mathbf{q}, t_{\mathbf{q}_1})$, $(\mathbf{q}, t_{\mathbf{q}_2})$,..., $(\mathbf{q}, t_{\mathbf{q}_T})$ (cf. Figure 6) in Lemma 6. We leave the proof of this lemma in the appendix (cf. Section 9.3).

Lemma 6. *The time complexity for evaluating the density values, using the triangular kernel as $K_{time}(t_q, t_p)$, of all T voxels $(\mathbf{q}, t_{q_1})$, $(\mathbf{q}, t_{q_2})$,..., $(\mathbf{q}, t_{q_T})$ is $O(T + n)$.*

Based on Lemma 6, we can extend Theorem 1, i.e., $O(XY(T + n))$ time for generating STKDV, to the triangular kernel. In addition, since the method SWS only needs to maintain $W_L(t_q)$, $W_R(t_q)$, $W(t_q)$ and their statistical terms, the space complexity of this method remains in $O(XYT + n)$ (i.e., Lemma 4 holds for the triangular kernel).

5 PROGRESSIVE VISUALIZATION FRAMEWORK FOR STKDV

Even though SWS can significantly reduce the time complexity for generating STKDV (from $O(XYTn)$ to $O(XY(T + n))$), SWS can still be time-consuming, especially for large-scale datasets. Instead of generating the visualization with all data points, many existing studies [30, 40, 42, 44, 45, 47, 67–69] adopt the data sampling methods to further improve the efficiency in different visualization tasks. In particular, Perrot et al. [42] propose to first divide the dataset into different subsets (with different sizes) in different levels and then progressively generate the visualization to users (e.g., data scientists). The general idea of this method is to first provide a rough visualization to users and then further refine it until users are satisfied with the visualization quality. In this section, we extend this idea for generating progressive STKDV (cf. Figure 10).

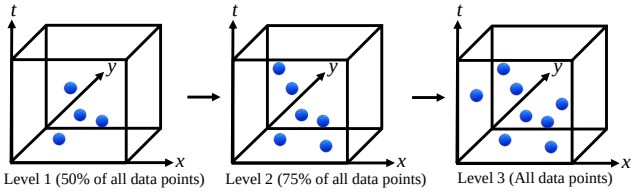


Figure 10: Progressive visualization for STKDV from lower to higher levels, i.e., smaller to larger subsets of the dataset, respectively.

One straightforward approach to support progressive STKDV is to compute the density values of each level from scratch. However, we observe that each pair of consecutive levels shares many data points, e.g., level 2 and level 3 in Figure 10 can share six data points. As such, this approach can waste the density computations of the previous level. Here, we ask a question, can we reuse the information from the previous level to generate STKDV for the next level in order to further boost the efficiency for progressive visualization?

Here, we let $\widehat{P}_\ell$ and $\widehat{P}_{\ell+1}$ be two sets of data points in the ℓ^{th} and $(\ell + 1)^{\text{th}}$ levels, respectively. Moreover, we denote $\mathcal{I}_\ell$ as the set of new data points that are in $\widehat{P}_{\ell+1}$ but not in $\widehat{P}_\ell$, i.e., $\mathcal{I}_\ell = \widehat{P}_{\ell+1} \setminus \widehat{P}_\ell$. Based on Equation 2, we have[2]:

$$\mathcal{F}_{\widehat{P}_{\ell+1}}(\mathbf{q}, t_q) = \mathcal{F}_{\widehat{P}_\ell}(\mathbf{q}, t_q) + \mathcal{F}_{\mathcal{I}_\ell}(\mathbf{q}, t_q) \quad (13)$$

Suppose that we have already stored the exact result $\mathcal{F}_{\widehat{P}_\ell}(\mathbf{q}, t_q)$ for each voxel $(\mathbf{q}, t_q)$ in the ℓ^{th} level, we can obtain $\mathcal{F}_{\widehat{P}_{\ell+1}}(\mathbf{q}, t_q)$, based on computing $\mathcal{F}_{\mathcal{I}_\ell}(\mathbf{q}, t_q)$ and then adding the precomputed value $\mathcal{F}_{\widehat{P}_\ell}(\mathbf{q}, t_q)$ for each voxel $(\mathbf{q}, t_q)$. Therefore, we can use $O(XY(T + |\mathcal{I}_\ell|))$ time (based on our method SWS) to generate STKDV for the level $\ell + 1$, which can be much faster than generating STKDV from scratch. Since we only maintain at most two cubes (for $\mathcal{F}_{\widehat{P}_\ell}(\mathbf{q}, t_q)$ and $\mathcal{F}_{\mathcal{I}_\ell}(\mathbf{q}, t_q)$) with size $X \times Y \times T$ and at most n data points (for $\widehat{P}_1$, $\mathcal{I}_2$, $\mathcal{I}_3$,...), the space complexity remains in $O(XYT + n)$. As a remark, this progressive visualization framework can combine with different types of data sampling methods (e.g., random sampling [43]).

6 EXPERIMENTAL EVALUATION

In this section, we first introduce the experimental settings in Section 6.1. Then, we investigate the efficiency improvement of our methods against the existing methods in Section 6.2, using the Epanechnikov kernel. After that, we further compare the efficiency of all methods in Section 6.3, using other kernel functions. Next, we demonstrate the efficiency for using the progressive visualization framework to generate STKDV in Section 6.4. Lastly, we provide the practical use case in Section 6.5 to visualize the time-evolving hotspots, by displaying STKDV as the time-evolving hotspot map.

6.1 Experimental Settings

We use five large-scale datasets for conducting the experiments, which are summarized in Table 3. All these datasets are the open data from the local governments of different cities/provinces. We follow [14, 21] and utilize the Scott's rule [51] to obtain the default parameters γ_s and γ_t. Moreover, we set the default resolution to be $128 \times 128 \times 128$ for generating STKDV.

Table 3: Datasets.

Dataset	n	Category	Ref.
Ontario	560,856	COVID-19	[8]
Seattle	839,504	Crime	[10]
Los Angeles	1,255,668	Crime	[5]
New York	1,499,928	Traffic accident	[6]
$\text{New York}_{\text{taxi}}$	13,596,055	Pickup location	[7]

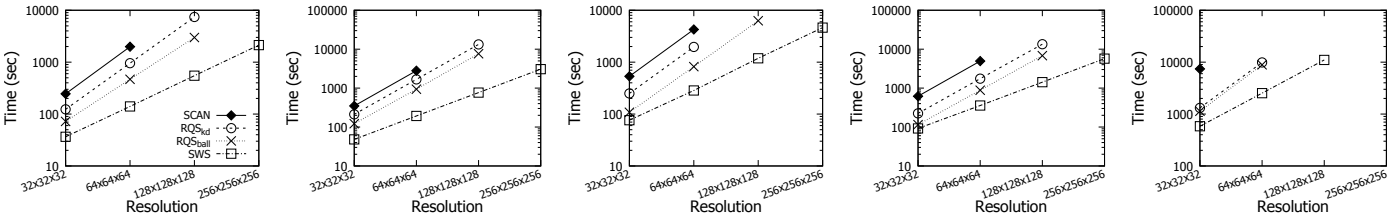
In our experiments, we compare our method SWS with different methods (cf. Table 4). SCAN is the scanning-based approach for generating STKDV, which does not adopt any type of filtering. RQS_{kd} and RQS_{ball} are the range-query-based solutions (cf. Section 2.2), which adopt the kd-tree and ball-tree, respectively. Our method SWS has a lower worst case time complexity compared with other methods. We implemented all methods[3] with C++ and conducted experiments on an Intel i7 3.19GHz PC with 32GB memory. In this paper, we use the response time (sec) to measure the efficiency of all methods and only report the response time which is smaller than 14400 sec (i.e., 4 hours).

Table 4: Methods for generating STKDV.

Method	Time complexity	Space complexity	Ref.
SCAN	$O(XYTn)$	$O(XYT + n)$	NIL
RQS_{kd}			[19, 41]
RQS_{ball}			[24, 41]
SWS	$O(XY(T + n))$		Sections 3-5

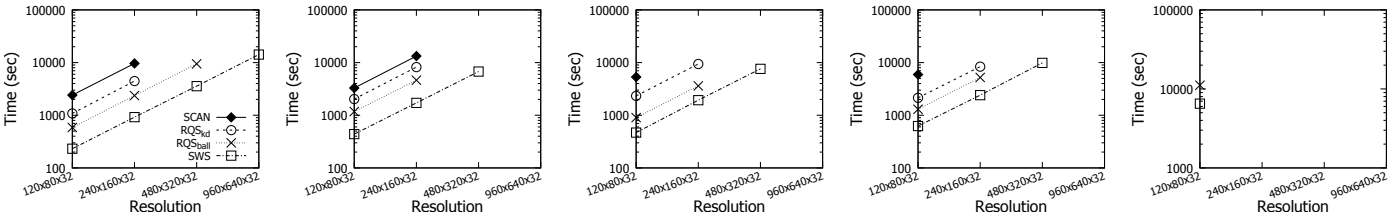
[2]Kernel density functions (cf. Equation 2) with different sizes of datasets can have different weights (constants) w [27]. Here, we omit the details to simplify the presentation.

[3]The source codes of all methods can be found in this Github repository https://github.com/STKDV/STKDV.



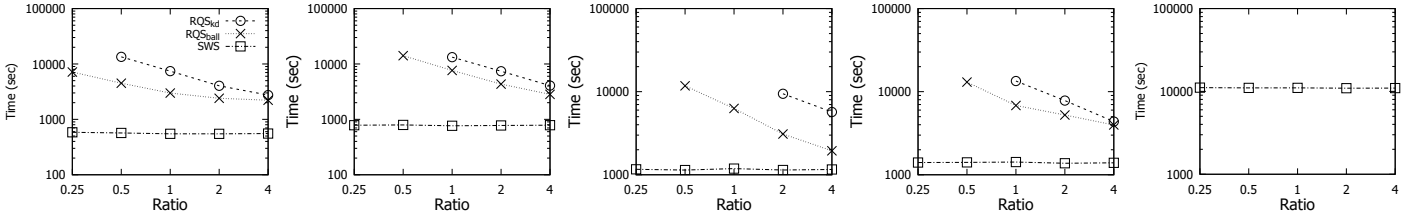
(a) Ontario (b) Seattle (c) Los Angeles (d) New York (e) New York$_{taxi}$

Figure 11: Response time for computing STKDV, varying the resolution size (from $32 \times 32 \times 32$ to $256 \times 256 \times 256$).



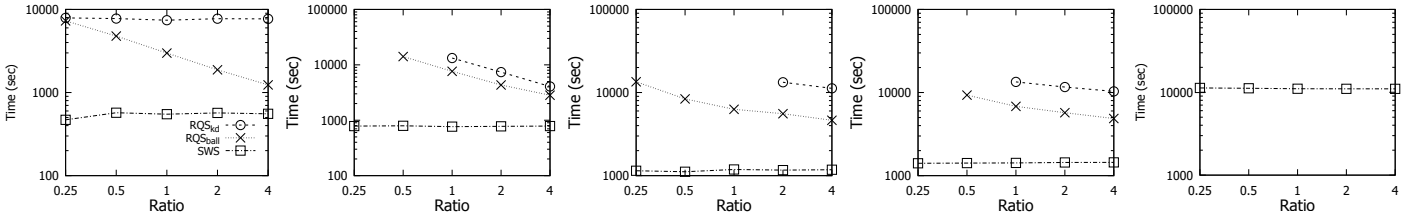
(a) Ontario (b) Seattle (c) Los Angeles (d) New York (e) New York$_{taxi}$

Figure 12: Response time for computing STKDV, varying the resolution size (from $120 \times 80 \times 32$ to $960 \times 640 \times 32$).



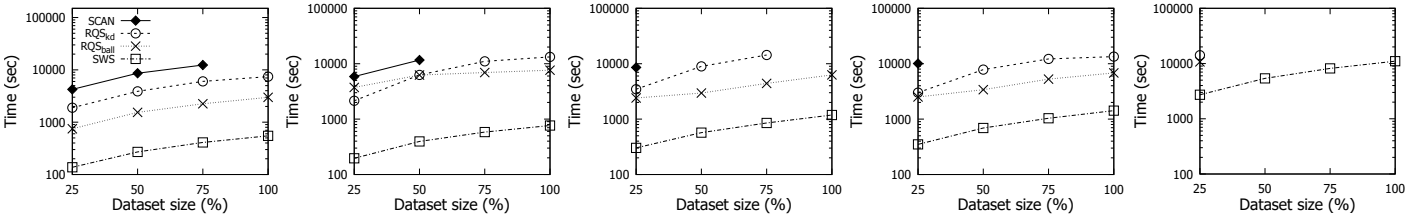
(a) Ontario (b) Seattle (c) Los Angeles (d) New York (e) New York$_{taxi}$

Figure 13: Response time for computing STKDV with default resolution $128 \times 128 \times 128$, varying the parameter γ_s (by multiplying the default value with different values of ratio).



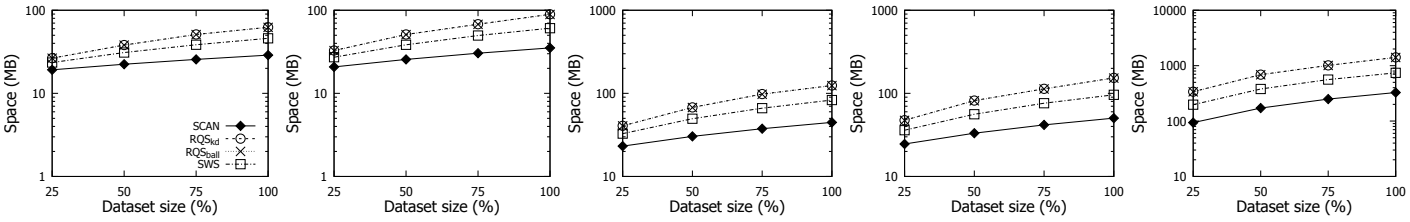
(a) Ontario (b) Seattle (c) Los Angeles (d) New York (e) New York$_{taxi}$

Figure 14: Response time for computing STKDV with default resolution $128 \times 128 \times 128$, varying the parameter γ_t (by multiplying the default value with different values of ratio).



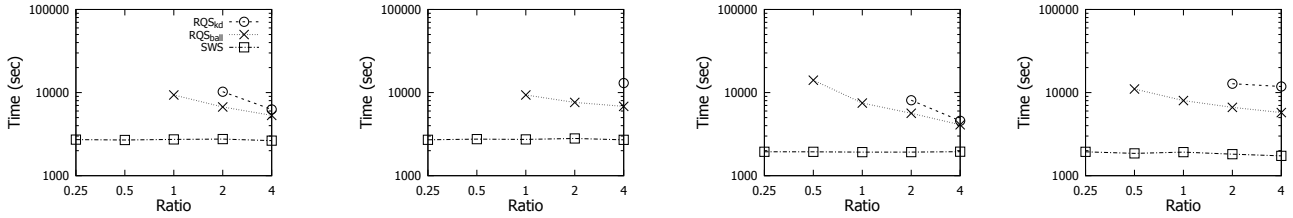
(a) Ontario (b) Seattle (c) Los Angeles (d) New York (e) New York$_{taxi}$

Figure 15: Response time for computing STKDV with default resolution $128 \times 128 \times 128$, varying the dataset size (by sampling different percentages of data points in each dataset).



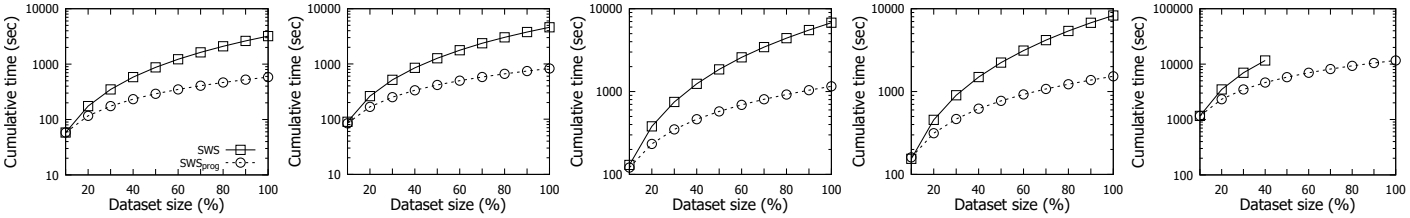
(a) Ontario (b) Seattle (c) Los Angeles (d) New York (e) New York$_{taxi}$

Figure 16: Space consumption (MB) for computing STKDV with default resolution $128 \times 128 \times 128$, varying the dataset size (by sampling different percentages of data points in each dataset).



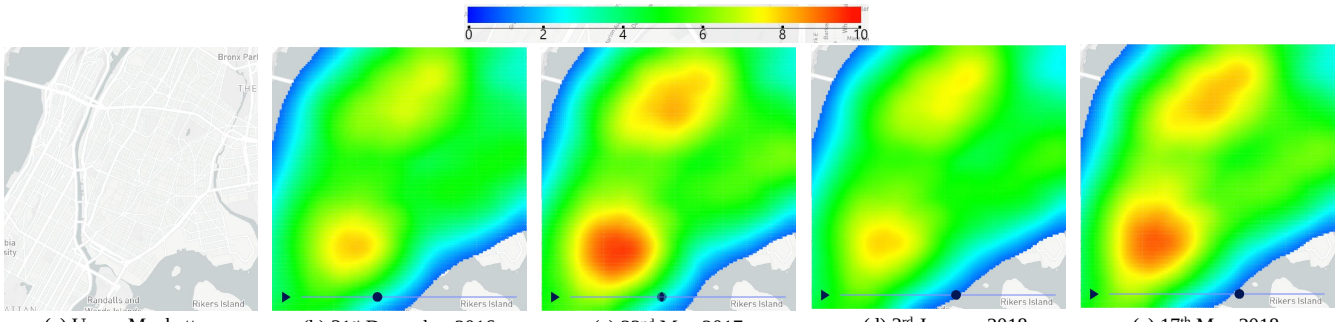
(a) Triangular kernel (varying γ_s) (b) Triangular kernel (varying γ_t) (c) Quartic kernel (varying γ_s) (d) Quartic kernel (varying γ_t)

Figure 17: Response time for computing STKDV in the New York dataset with default resolution $128 \times 128 \times 128$, using the triangular ((a) and (b)) and quartic ((c) and (d)) kernels, varying the parameters γ_s ((a) and (c)) and γ_t ((b) and (d)).



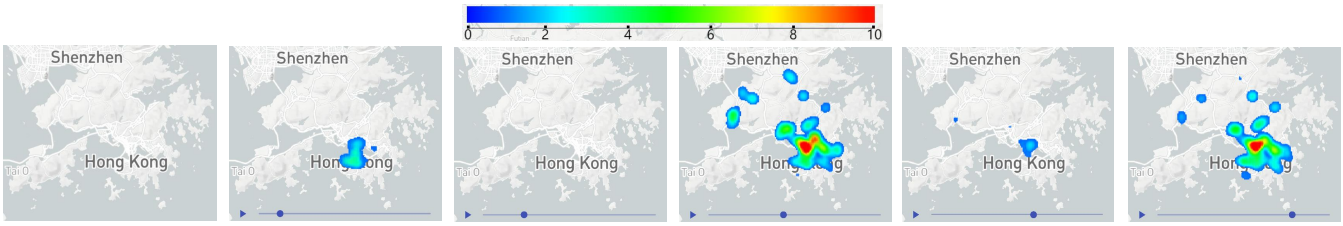
(a) Ontario (b) Seattle (c) Los Angeles (d) New York (e) New York$_{taxi}$

Figure 18: Cumulative response time for progressively computing STKDV with default resolution $128 \times 128 \times 128$, using a sequence of subsets of dataset.



(a) Upper Manhattan (b) 31st December 2016 (c) 22nd May 2017 (d) 3rd January 2018 (e) 17th May 2018

Figure 19: Time-evolving hotspot map (based on STKDV) in the Upper Manhattan region with four timestamps, using the New York traffic accident dataset.



(a) Hong Kong (b) 1st April 2020 (c) 15th May 2020 (d) 30th July 2020 (e) 1st Oct 2020 (f) 10th Dec 2020

Figure 20: Time-evolving hotspot map (based on STKDV) in Hong Kong with five timestamps, using the Hong Kong COVID-19 dataset (confirmed cases).

6.2 Efficiency Evaluation with Epanechnikov Kernel

Even though our method SWS is theoretically more efficient than the existing methods without additional space overhead (cf. Table 4), it is yet to compare the time and space efficiency of our methods with these methods in practice. In this section, we investigate the following four research questions of efficiency issues for generating STKDV.

(1) How does the resolution size affect the response time of each method?
(2) How does the parameter γ_s affect the response time of each method?
(3) How does the parameter γ_t affect the response time of each method?
(4) How does the size of the dataset affect the response time and space consumption of each method?

Varying the resolution size: In the first experiment, we choose four resolution sizes, which are 32×32×32, 64×64×64, 128×128×128 and $256 \times 256 \times 256$, and measure the response time of different methods. In Figure 11, we observe that our method SWS consistently outperforms the existing methods with different resolutions. Since the (worst case) time complexity of existing methods is $O(XYTn)$ (cf. Table 4), the response time of these methods can increase by 8 times, once we use the next larger resolution (e.g., from $32 \times 32 \times 32$ to $64 \times 64 \times 64$). However, since the time complexity of SWS is $O(XY(T+n))$, the response time of SWS only increases by 4 times, using the next larger resolution. As such, the larger the resolution size, the larger the time gap between SWS and the existing methods (cf. Figure 11).

In the second experiment, we further choose four resolution sizes, which are $120 \times 80 \times 32$, $240 \times 160 \times 32$, $480 \times 320 \times 32$ and $960 \times 640 \times 32$, for testing. Since we only vary the spatial resolution $X \times Y$ and fix the temporal resolution T, the time gap between our method SWS and the best method RQS_{ball} does not significantly change for using the next larger resolution (e.g., from $120 \times 80 \times 32$ to $240 \times 160 \times 32$). Nevertheless, our method SWS still achieves at least 1.71x to 2.69x speedup (cf. Figure 12) compared with the existing methods.

Varying the parameter γ_s: We proceed to investigate how the parameter γ_s affects the response time of all methods. Here, we adopt the default resolution 128×128×128 and the default parameter γ_t (obtained by Scott's rule). In this experiment, we multiply the default value of γ_s by different values of ratio, including 0.25, 0.5, 1, 2 and 4, and measure the response time for generating STKDV. Recall that the data points that are within $\frac{1}{\gamma_s}$ can have non-zero values for $K_{\text{space}}(\mathbf{q}, \mathbf{p})$ (cf. Table 1). Therefore, once the value γ_s is smaller (i.e., the range $\frac{1}{\gamma_s}$ is larger), all the range-query-based methods RQS_{kd} and RQS_{ball} need to scan more data points and index nodes. As such, all these range-query-based methods can be slower with the smaller value of ratio (cf. Figure 13). Since our method SWS is not sensitive to γ_s, our method can be significantly more efficient than the existing methods, especially for small value of γ_s.

Varying the parameter γ_t: We further conduct the experiment for measuring the response time of different methods by using different values of γ_t (i.e., multiplying the default value by different values of ratio), while we adopt the default value for γ_s and the default resolution $128 \times 128 \times 128$. In Figure 14, we observe that our method SWS outperforms the existing methods by a visible margin, no matter which γ_t (or ratio) we adopt. Moreover, unlike the range-query-based methods, SWS is not sensitive to the parameter γ_t.

Varying the dataset size: In this experiment, we randomly sample each dataset (in Table 3) with different percentages, including 25%, 50%, 75%, and 100% (original one), and measure the response time and memory space consumption of each method for each sampled dataset. In Figure 15, we observe that SWS consistently outperforms the existing methods by 5x to 16x speedup in different dataset sizes. On the other hand, since our method SWS has the same space complexity as the existing methods (cf. Table 4), the space consumption of all methods are similar (cf. Figure 16).

6.3 Efficiency for Other Kernels

We investigate the response time for generating STKDV with other kernels. In this experiment, we adopt the New York dataset for testing and follow the same settings in Section 6.2 for varying the parameters γ_s and γ_t. Figure 17 illustrates that our method SWS can also consistently outperform the state-of-the-art methods, regardless of the chosen kernel types. On the other hand, since both the parameters γ_s and γ_t cannot affect the efficiency of our method SWS for generating STKDV with triangular and quartic kernels, we observe that the response time of SWS is similar, no matter which γ_s and γ_t we adopt (cf. Figure 17).

6.4 Progressive Visualization Framework

In this section, we proceed to test the efficiency for using the progressive visualization framework. To conduct this experiment, we randomly sample each dataset with different levels of subsets (cf.

Figure 10), where the subset at a larger level covers the subset at a smaller level. Here, we choose a sequence of percentages, which are 10%, 20%,..., and 100%, to represent the size of subsets in each level compared with the original dataset. In this experiment, we measure the cumulative time for generating STKDV, following the above sequence of levels. Since our method SWS is consistently more efficient than the previous methods, we only compare the efficiency of this method with the progressive version of this method SWS_{prog} in this experiment. In Figure 18, we observe that SWS_{prog} achieves smaller cumulative time, since this method does not need to recompute all the density values from scratch.

6.5 Use Case: Display STKDV as Time-Evolving Hotspot Map

After we generate STKDV for a dataset, we can display this space-time cube as a time-evolving hotspot map, which can facilitate users (e.g., geoscientists) to visualize the time-evolving hotspots (based on STKDV). Here, we show two examples in this section.

Traffic accident hotspot detection: Figure 19 shows the time-evolving traffic accident hotspot map in the Upper Manhattan region of New York with four timestamps, using the New York traffic accident dataset (cf. Table 3). Observe that the hotspots (with orange color) can change in different timestamps. For example, the sizes of traffic accident hotspots are larger in May compared with December and January. This phenomenon indicates that more traffic police officers should be assigned for these two hotspot regions in May in order to reduce the number of traffic accident events. Moreover, the transportation experts also need to investigate the underlying reasons for this phenomenon.

COVID-19 hotspot detection: In the second example, we show the time-evolving COVID-19 hotspot map in Hong Kong with five timestamps, using the COVID-19 open dataset [3] from the Hong Kong government, which stores the location and time of each COVID-19 confirmed case in Hong Kong. In Figure 20, observe that the hotspots can significantly change in different timestamps. For example, there can be no hotspot on 15^{th} May 2020 and a large hotspot on 30^{th} July 2020. In addition, this tool can correctly show different waves in Hong Kong, which are in line with the trend of COVID-19 confirmed cases in Hong Kong (cf. Figure 21).

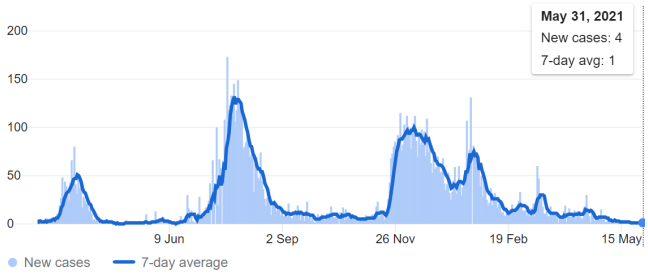


Figure 21: The trend of COVID-19 confirmed cases in Hong Kong (cropped from Google statistics).

For details, please refer to the Github repository https://github.com/STKDV/STKDV, in which we provide these two time-evolving hotspot maps (cf. Figures 19 and 20) with 128 timestamps and their implementation. Moreover, we also discuss the zoom-in operation to explore the hotspots in different regions and time ranges.

7 RELATED WORK

Kernel density visualization (KDV) [14, 51] has been extensively used in different domains, including traffic accident hotspot detection [61], crime hotspot detection [13, 25, 66], and disease outbreak analysis [4, 15]. However, KDV only generates the visualization based on the spatial positions of the geographical events (cf. Equation 1), which ignores the event time. As such, many recent studies [20, 27, 33, 71] in different domains also complain about the effectiveness for using KDV. To overcome the weakness of KDV, many research studies [12, 20, 26–28, 31, 33, 39, 65, 71] utilize the spatial-temporal kernel density visualization (STKDV), which incorporates the temporal kernel for estimating the density (cf. Equation 2), to color the cube (cf. Figure 3). These studies also indicate that STKDV can achieve superior effectiveness compared with the traditional visualization tools (e.g., KDV), under different case studies. However, with the high time complexity, i.e., $O(XYTn)$, for generating STKDV, existing methods cannot be scalable to large-scale datasets. To the best of our knowledge, this is the first research work that theoretically reduces the time complexity for this time-consuming operation (cf. Table 4). In this section, we summarize six camps of research studies, which are mostly related to this work.

Range-query-based methods: Recall from Section 2.2, we can compute the kernel density function of STKDV $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_{\mathbf{q}})$ (cf. Equation 4), based on obtaining the reduced set $R_{\mathbf{q}}$ (cf. Equation 3). Therefore, computing STKDV can be also cast as solving the range query problem for each voxel $(\mathbf{q}, t_{\mathbf{q}})$. Range queries [19, 49, 63] have been extensively studied in the literature. Among most of the existing methods, kd-tree [11] and ball-tree [38] are the most efficient and popular methods, which have been widely used for efficiently solving the range queries in low-dimensional datasets [41]. Even though range-query-based solutions can improve the efficiency for generating STKDV, these methods (cf. RQS_{kd} and RQS_{ball}) cannot theoretically reduce the time complexity for generating STKDV (cf. Table 4). As shown in our experiments, these methods are not scalable to large resolution size (cf. Figure 11), small γ_s and γ_t (cf. Figures 13 and 14, respectively) and large dataset size (cf. Figure 15) compared with our method SWS.

Sliding-window-based methods: In both database and data mining communities, many efficient sliding-window-based methods have been developed to support different query processing tasks, including aggregation (e.g., sum, count, max, min, etc.) [22, 32, 52–54, 57], skyline [55], and top-k queries [59, 72], over streaming data. However, none of the existing methods focuses on the complex spatial-temporal kernel density function $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_{\mathbf{q}})$ (cf. Equation 2). Therefore, existing studies cannot be easily extended to efficiently compute $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_{\mathbf{q}})$.

Function approximation methods: Many researchers have proposed to approximate the kernel density function $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 1) in order to improve the efficiency for generating KDV. Raykar et al. [48] and Yang et al. [62] propose using fast Gauss transform to efficiently and approximately compute $\mathcal{F}_P(\mathbf{q})$. On the other hand, Chan et al. [14, 16, 17], Gan et al. [21] and Gray et al. [24] develop the lower and upper bound functions to accurately approximate

$\mathcal{F}_P(\mathbf{q})$ (cf. Equation 1). However, unlike KDV, the kernel density function for STKDV $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q})$ (cf. Equation 2) is more complex, which involves the multiplication of both spatial kernel $K_{\text{space}}(\mathbf{q}, \mathbf{p})$ and temporal kernel $K_{\text{time}}(t_\mathbf{q}, t_\mathbf{p})$. Therefore, it remains unknown whether these methods can be modified to support the fast computation of $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q})$ with non-trivial approximation guarantee.

Data sampling methods: To efficiently generate KDV, Zheng et al. [67–69] and Phillips et al. [43–45] have developed advanced algorithms to first sample the original dataset and then evaluate the modified kernel density function, based on the reduced dataset. They further show that this approach can provide the non-trivial approximation guarantee between the original kernel density function value $\mathcal{F}_P(\mathbf{q})$ and their output result for each pixel $\mathbf{q}$. However, it remains unknown whether this approach can be extended to support STKDV with non-trivial approximation guarantee. As a remark, our progressive visualization framework (cf. Section 5) can combine with different types of data sampling methods.

Parallel/distributed computation and hardware-based methods: There are also many research studies that utilize the parallel/distributed computation, e.g., MapReduce [67] and hardware-based methods, e.g., GPU [30, 42, 64] and FPGA [23], to further boost the efficiency for computing KDV. Recently, Saule et al. [50], Hohl et al. [26] and Delmelle et al. [20] further adopt the parallel computation to improve the efficiency for generating STKDV. Like [20, 26, 50], our method SWS can also be highly parallelized to further improve the efficiency for generating STKDV (cf. Section 9.5).

Other visualization methods: In both database and visualization communities, some research studies adopt other simple visualization methods to visualize data. Two representative approaches include scatter plot [36, 37] and histogram [34, 58]. However, scatter plot suffers from the overplotting issues [42] (i.e., this method cannot clearly show the density of data points in different regions with limited resolution size and large number of data points.). Although histogram-based visualization can overcome the overplotting issues, it is hard for domain experts to simultaneously determine the locations of all bins and the bin size, which leads to incorrect interpretation of data (as pointed out by [41]). Furthermore, each data point can only be assigned to one bin for histogram-based visualization, even though it can possibly affect larger region (or larger number of bins). Therefore, histogram-based visualization is not smooth [2], which achieves low visualization quality. Compared with these two visualization methods, both KDV and STKDV methods can overcome these drawbacks, which can therefore be the de facto visualization tools in different domains [20, 27, 31, 33].

8 CONCLUSION

In this paper, we study spatial-temporal kernel density visualization (STKDV), which has been extensively used in different types of applications, including disease outbreak analysis [26, 65], traffic accident hotspot detection [28, 33], and crime hotspot detection [27]. However, STKDV is a computational expensive operation (with $O(XYTn)$ time), which is not scalable to large-scale dataset. To improve the efficiency for computing STKDV, we develop the sliding-window-based solution (SWS), which can theoretically reduce the time complexity to $O(XY(T+n))$, without increasing the space complexity (i.e., $O(XYT+n)$). By combining SWS with the progressive visualization framework, we can further reduce the response time for supporting progressive visualization with different types of data sampling methods (e.g., random sampling [43]). Our experimental results show that our method SWS can consistently outperform the state-of-the-art methods by 2x to 24x.

In the future, we will extend SWS to other commonly-used kernel functions. In addition, we will develop the visualization system for STKDV to support many data analytics tasks. Furthermore, we will also exploit the opportunity for combining the hardware-based approach with SWS, which can further improve the efficiency for generating STKDV.

9 APPENDIX

9.1 Proof of Lemma 3

Proof. In this proof, our goal is to show that $|W_{t_{\mathbf{q}_1}}| + \sum_{i=1}^{T-1} |I(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}}))| + \sum_{i=1}^{T-1} |D(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}}))|$ is $O(n)$ in Equation 10.

We consider the first term $|W_{t_{\mathbf{q}_1}}|$. Since the window $W_{t_{\mathbf{q}_1}}$ can be arbitrary large (by setting the value γ_t of the Epanechnikov kernel (cf. Table 1) to be as small as possible), we can find that $|W_{t_{\mathbf{q}_1}}| = O(n)$ in the worst case theoretically.

Observe from Figure 7, we have:

$$I(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}})) \cap I(W(t_{\mathbf{q}_j}), W(t_{\mathbf{q}_{j+1}})) = \phi$$

$$D(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}})) \cap D(W(t_{\mathbf{q}_j}), W(t_{\mathbf{q}_{j+1}})) = \phi$$

where $1 \le i, j \le T-1$ and $i \ne j$.

Since the number of points in the dataset is at most n, we can conclude that:

$$\sum_{i=1}^{T-1} |I(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}}))| \le n \text{ and } \sum_{i=1}^{T-1} |D(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}}))| \le n$$

Hence, we have proved Lemma 3. □

9.2 Proof of Lemma 5

In this proof, we focus on how to update the statistical terms in both left and right windows[4], i.e., $W_L(t_{\mathbf{q}_n})$ and $W_R(t_{\mathbf{q}_n})$, respectively, for the next voxel $(\mathbf{q}, t_{\mathbf{q}_n})$. Then, we can compute $\mathcal{F}_{\widehat{P}}(\mathbf{q}, t_{\mathbf{q}_n})$ in $O(1)$ time by adopting Equation 11, given the statistical terms. Here, we need to consider the following three possible cases.

Case 1 $t_{\mathbf{q}_n} - t_\mathbf{q} \le \frac{1}{\gamma_t}$: Observe from Figure 22, we find that:

$$\begin{aligned} W_L(t_{\mathbf{q}_n}) &= (W_L(t_\mathbf{q}) \setminus D(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))) \cup C(t_\mathbf{q}, t_{\mathbf{q}_n}) \\ W_R(t_{\mathbf{q}_n}) &= (W_R(t_\mathbf{q}) \setminus C(t_\mathbf{q}, t_{\mathbf{q}_n})) \cup I(W(t_\mathbf{q}), W(t_{\mathbf{q}_n})) \end{aligned}$$

Therefore, we can update the statistical terms for both left and right windows of $(\mathbf{q}, t_{\mathbf{q}_n})$ using the following equations:

$$\begin{aligned} S^{(i)}_{W_L(t_{\mathbf{q}_n})}(\mathbf{q}) &= S^{(i)}_{W_L(t_\mathbf{q})}(\mathbf{q}) - \sum_{(\mathbf{p}, t_\mathbf{p}) \in D(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))} t^i_\mathbf{p} \cdot K_{\text{space}}(\mathbf{q}, \mathbf{p}) \\ &\quad + \sum_{(\mathbf{p}, t_\mathbf{p}) \in C(t_\mathbf{q}, t_{\mathbf{q}_n})} t^i_\mathbf{p} \cdot K_{\text{space}}(\mathbf{q}, \mathbf{p}) \\ S^{(i)}_{W_R(t_{\mathbf{q}_n})}(\mathbf{q}) &= S^{(i)}_{W_R(t_\mathbf{q})}(\mathbf{q}) + \sum_{(\mathbf{p}, t_\mathbf{p}) \in I(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))} t^i_\mathbf{p} \cdot K_{\text{space}}(\mathbf{q}, \mathbf{p}) \\ &\quad - \sum_{(\mathbf{p}, t_\mathbf{p}) \in C(t_\mathbf{q}, t_{\mathbf{q}_n})} t^i_\mathbf{p} \cdot K_{\text{space}}(\mathbf{q}, \mathbf{p}) \end{aligned}$$

[4] We omit the proof for the fast update for the statistical term $S^{(0)}_{W(t_\mathbf{q})}$ in Equation 11, as we can reuse the result in Lemma 2 to infer that the time complexity is $O(|I(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))| + |D(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))|)$.

As such, we can obtain the above statistical terms after we scan those data points in $D(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))$, $I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))$ and $C(t_{\mathbf{q}}, t_{\mathbf{q}_n})$. Therefore, the time complexity for this case is $O(|I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))| + |D(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))| + |C(t_{\mathbf{q}}, t_{\mathbf{q}_n})|)$.

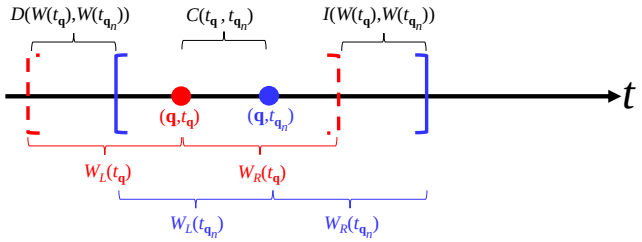


Figure 22: The case for $t_{\mathbf{q}_n} - t_{\mathbf{q}} \leq \frac{1}{\gamma_t}$.

Case 2 $t_{\mathbf{q}_n} - t_{\mathbf{q}} > \frac{1}{\gamma_t}$ and $t_{\mathbf{q}_n} - t_{\mathbf{q}} \leq \frac{2}{\gamma_t}$: Observe from Figure 23, we find that:

$$W_L(t_{\mathbf{q}_n}) = (W_R(t_{\mathbf{q}}) \setminus S) \cup A$$
$$W_R(t_{\mathbf{q}_n}) = I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n})) \setminus A$$

Therefore, we have:

$$S^{(i)}_{W_L(t_{\mathbf{q}_n})}(\mathbf{q}) = S^{(i)}_{W_R(t_{\mathbf{q}})}(\mathbf{q}) - \sum_{(\mathbf{p},t_{\mathbf{p}})\in S} t^i_{\mathbf{p}} \cdot K_{\text{space}}(\mathbf{q},\mathbf{p}) + \sum_{(\mathbf{p},t_{\mathbf{p}})\in A} t^i_{\mathbf{p}} \cdot K_{\text{space}}(\mathbf{q},\mathbf{p})$$

$$S^{(i)}_{W_R(t_{\mathbf{q}_n})}(\mathbf{q}) = \sum_{(\mathbf{p},t_{\mathbf{p}})\in I(W(t_{\mathbf{q}}),W(t_{\mathbf{q}_n}))} t^i_{\mathbf{p}} \cdot K_{\text{space}}(\mathbf{q},\mathbf{p}) - \sum_{(\mathbf{p},t_{\mathbf{p}})\in A} t^i_{\mathbf{p}} \cdot K_{\text{space}}(\mathbf{q},\mathbf{p})$$

Since both the sets S and A are inside the sets $D(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))$ and $I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))$, respectively, we can compute these statistical terms by scanning these two sets, which take $O(|I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))| + |D(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))|)$ time.

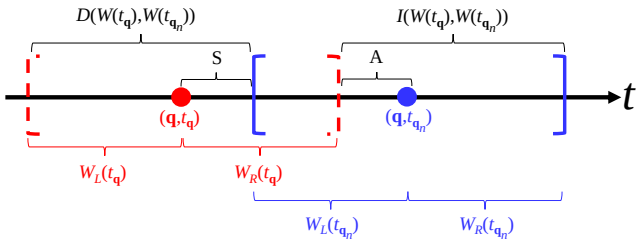


Figure 23: The case for $t_{\mathbf{q}_n} - t_{\mathbf{q}} > \frac{1}{\gamma_t}$ and $t_{\mathbf{q}_n} - t_{\mathbf{q}} \leq \frac{2}{\gamma_t}$.

Case 3 $t_{\mathbf{q}_n} - t_{\mathbf{q}} > \frac{2}{\gamma_t}$: Observe from Figure 24, we only need to scan all data points from $I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))$ in order to obtain both $S^{(i)}_{W_L(t_{\mathbf{q}_n})}(\mathbf{q})$ and $S^{(i)}_{W_R(t_{\mathbf{q}_n})}(\mathbf{q})$, which takes $O(|I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))|)$ time. In addition, we also need to scan the additional data points in the yellow region, which is at most $O(|C(t_{\mathbf{q}}, t_{\mathbf{q}_n})|)$ time, in order to determine the start position of the $W(t_{\mathbf{q}_n})$ (blue window). Therefore, it takes at most $O(|I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))| + |C(t_{\mathbf{q}}, t_{\mathbf{q}_n})|)$ in this case.

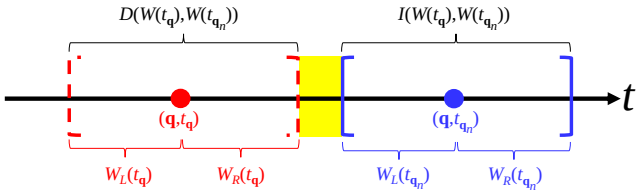


Figure 24: The case for $t_{\mathbf{q}_n} - t_{\mathbf{q}} > \frac{2}{\gamma_t}$.

Hence, based on the time complexity for these three cases, we have proved this lemma.

9.3 Proof of Lemma 6

Since the proof of this lemma follows the same concept in Section 3.2 (just after Lemma 2), we omit the details for this part. Based on the result of Lemma 5, we can conclude that it takes the following time complexity to evaluate all voxels $(\mathbf{q}, t_{\mathbf{q}_1})$,$(\mathbf{q}, t_{\mathbf{q}_2})$,...,$(\mathbf{q}, t_{\mathbf{q}_T})$:

$$O\Big(|W_{t_{\mathbf{q}_1}}| + \sum_{i=1}^{T-1} |I(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}}))| + \sum_{i=1}^{T-1} |D(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}}))| + \sum_{i=1}^{T-1} |C(t_{\mathbf{q}_i}, t_{\mathbf{q}_{i+1}})| + T\Big)$$

Based on the proof of Lemma 3, we know that $|W_{t_{\mathbf{q}_1}}| + \sum_{i=1}^{T-1} |I(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}}))| + \sum_{i=1}^{T-1} |D(W(t_{\mathbf{q}_i}), W(t_{\mathbf{q}_{i+1}}))| = O(n)$. Here, once we can show that $\sum_{i=1}^{T-1} |C(t_{\mathbf{q}_i}, t_{\mathbf{q}_{i+1}})| = O(n)$, we can prove this lemma.

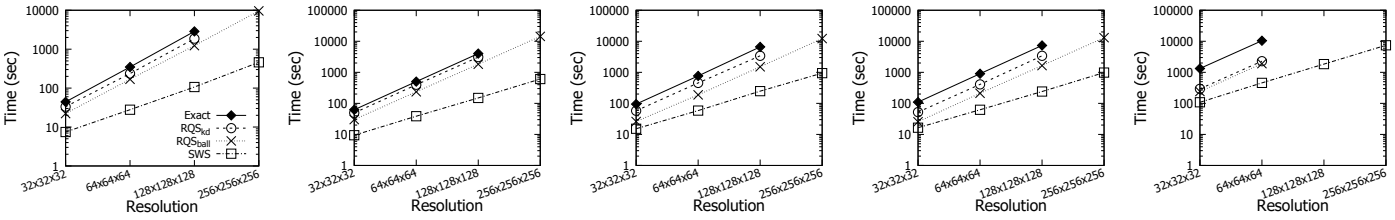
Observe from both Figure 7 and Equation 12, we can conclude that $C(t_{\mathbf{q}_i}, t_{\mathbf{q}_{i+1}}) \cap C(t_{\mathbf{q}_j}, t_{\mathbf{q}_{j+1}}) = \phi$, where $1 \leq i, j \leq T-1$ and $i \neq j$. Therefore, we also have:

$$\sum_{i=1}^{T-1} |C(t_{\mathbf{q}_i}, t_{\mathbf{q}_{i+1}})| \leq n = O(n)$$

9.4 Pseudocode of SWS and Its Implementation Details

Pseudocode of SWS: Recall from Figure 3b, we have divided the cube into a set of voxels (i.e., small cubes), where each voxel can be represented by the spatial and temporal coordinates $(\mathbf{q}, t_{\mathbf{q}})$. Here, we let x_s, y_s and t_s be the smallest values of x-coordinate, y-coordinate and t-coordinate, respectively. Moreover, we also denote the distance value for shifting to the next voxel with positive direction along the x-axis, y-axis and t-axis to be Δ_x, Δ_y and Δ_t, respectively. Furthermore, we also represent the cube C (cf. Figure 3b) with $X \times Y \times T$ voxels. Based on the above information, we provide the pseudocode (cf. Algorithm 1) for our method SWS with the Epanechnikov kernel. In line 12 to line 16 of Algorithm 1, we adopt the incremental algorithm in Section 3.2 to update the statistical terms and compute the kernel density function $\mathcal{F}_{\mathcal{P}}(\mathbf{q}, t_{\mathbf{q}})$. This pseudocode can be extended to support other kernel functions in Section 4.

Implementation details: In order to efficiently obtain $I(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))$ (line 13) and $D(W(t_{\mathbf{q}}), W(t_{\mathbf{q}_n}))$ (line 14), we



(a) Ontario (b) Seattle (c) Los Angeles (d) New York (e) New York$_{taxi}$

Figure 25: Response time for computing STKDV with six cores, varying the resolution size (from $32 \times 32 \times 32$ to $256 \times 256 \times 256$).

Algorithm 1 Sliding-Window-based Solution (with Epanechnikov kernel)

procedure SWS(Point set $\widehat{P}$, x_s, y_s, t_s, Δ_x, Δ_y, Δ_t, X, Y, T)
  Define cube C with size $X \times Y \times T$
  for $u \leftarrow 1$ to X **do**
    $x \leftarrow x_s + (u-1)\Delta_x$
    for $v \leftarrow 1$ to Y **do**
      $y \leftarrow y_s + (v-1)\Delta_y$
      $\mathbf{q} \leftarrow (x, y)$, $t_\mathbf{q} \leftarrow t_s$
      Obtain $W(t_\mathbf{q})$
      Obtain $S^{(i)}_{W(t_\mathbf{q})}(\mathbf{q})$ $(i = 0, 1, 2)$ ▷ Equation 6
      $C(\mathbf{q}, t_\mathbf{q}) \leftarrow \mathcal{F}_{\widehat{P}}(\mathbf{q}, t_\mathbf{q})$ ▷ Equation 5
      for $w \leftarrow 2$ to T **do**
        $t_{\mathbf{q}_n} \leftarrow t_s + (w-1)\Delta_t$
        Obtain $I(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))$ ▷ Equation 7
        Obtain $D(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))$ ▷ Equation 8
        Obtain $S^{(i)}_{W(t_{\mathbf{q}_n})}$ $(i = 0, 1, 2)$ ▷ Equation 9
        $C(\mathbf{q}, t_{\mathbf{q}_n}) \leftarrow \mathcal{F}_{\widehat{P}}(\mathbf{q}, t_{\mathbf{q}_n})$ ▷ Equation 5
        $t_\mathbf{q} \leftarrow t_{\mathbf{q}_n}$
  Return the cube C

also need to maintain the starting and ending data points for each sliding window. Using Figure 5 as an example, the red dashed window $W(t_{\mathbf{q}_n})$ should store $(\mathbf{p}_1, t_{\mathbf{p}_1})$ and $(\mathbf{p}_4, t_{\mathbf{p}_4})$ as the starting and ending data points, respectively. Once we shift to the next sliding window $W(t_\mathbf{q})$ (blue window), we can identify $I(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))$ and $D(W(t_\mathbf{q}), W(t_{\mathbf{q}_n}))$ without incurring the additional costs for finding the positions of these points (green and yellow points). After we scan these data points, we can then obtain the starting and ending data points for the next window $W(t_{\mathbf{q}_n})$, i.e., $(\mathbf{p}_3, t_{\mathbf{p}_3})$ and $(\mathbf{p}_5, t_{\mathbf{p}_5})$, respectively for the blue window.

For more details, please refer to our implementation, which is available in the Github repository https://github.com/STKDV/STKDV.

9.5 Parallel Approach for Computing STKDV

Due to the high time complexity for generating STKDV, some research studies [20, 26, 50] adopt the parallel approach to boost the efficiency for computing STKDV. Here, we discuss how to combine the parallel approach with different methods and compare the efficiency of these methods.

Parallel approach for SWS: In Figure 6, observe that only those (yellow) voxels along the t-axis need to access the same computational resources. As an example, we need to use the statistical terms in $W(t_{\mathbf{q}_1})$ to compute the statistical terms in $W(t_{\mathbf{q}_2})$ (cf. Lemma 1). However, those voxels with different spatial positions do not need to share the same computational resources. Therefore, we can assign each thread to handle different spatial positions in parallel.

Parallel approach for existing methods: Since different voxels do not need to access the same computational resources for SCAN, RQS$_{kd}$ and RQS$_{ball}$, we can assign each thread to handle different voxels in parallel (like [20, 26, 50]).

Efficiency Comparison: We adopt an Intel i7 3.19GHz PC with six cores to conduct this experiment. Here, we vary the resolution size and measure the response time of each method. Compared with the single-thread experiment (cf. Figure 11), all these methods can be highly parallelized, which further achieve 4.5x to 5.6x speedup (cf. Figure 25). In addition, we can further observe that our method SWS still achieves the smallest response time among all these methods under this setting (cf. Figure 25).

REFERENCES

[1] Arcgis. http://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/how-kernel-density-works.htm.
[2] CrimeStat: Spatial statistics program for the analysis of crime incident locations. https://nij.ojp.gov/topics/articles/crimestat-spatial-statistics-program-analysis-crime-incident-locations.
[3] Hong Kong GeoData Store. https://geodata.gov.hk/gs/view-dataset?uuid=d4ccd9be-3bc0-449b-bd27-9eb9b615f2db&sidx=0.
[4] Ikcest: Disaster risk reduction. http://drr.ikcest.org/knowledge_service/ncp.html.
[5] Los Angeles open data. https://data.lacity.org/A-Safe-City/Crime-Data-from-2010-to-2019/63jg-8b9z.
[6] NYC open data. https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95.
[7] NYC yellow taxi trip data. https://data.cityofnewyork.us/Transportation/2014-Yellow-Taxi-Trip-Data/gkne-dk5s.
[8] Ontario open data. https://data.ontario.ca/dataset/confirmed-positive-cases-of-covid-19-in-ontario.
[9] QGIS. https://docs.qgis.org/2.18/en/docs/user_manual/plugins/plugins_heatmap.html.
[10] Seattle open data. https://data.seattle.gov/Public-Safety/SPD-Crime-Data-2008-Present/tazs-3rd5.
[11] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.
[12] C. Brunsdon, J. Corcoran, and G. Higgs. Visualising space and time in crime patterns: A comparison of methods. *Comput. Environ. Urban Syst.*, 31(1):52–75, 2007.
[13] S. Chainey, L. Tompson, and S. Uhlig. The utility of hotspot mapping for predicting spatial patterns of crime. *Security Journal*, 21(1):4–28, Feb 2008.
[14] T. N. Chan, R. Cheng, and M. L. Yiu. QUAD: Quadratic-bound-based kernel density visualization. In *SIGMOD*, pages 35–50, 2020.
[15] T. N. Chan, P. L. Ip, L. H. U, W. H. Tong, S. Mittal, Y. Li, and R. Cheng. KDV-Explorer: A near real-time kernel density visualization system for spatial analysis. *Proc. VLDB Endow.*, 2021, (To appear).
[16] T. N. Chan, L. H. U, R. Cheng, M. L. Yiu, and S. Mittal. Efficient algorithms for kernel aggregation queries. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.
[17] T. N. Chan, M. L. Yiu, and L. H. U. KARL: Fast kernel aggregation queries. In *ICDE*, pages 542–553, 2019.

[18] W. Chen, F. Guo, and F. Wang. A survey of traffic data visualization. *IEEE Trans. Intelligent Transportation Systems*, 16(6):2970–2984, 2015.

[19] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition.* Springer, 2008.

[20] E. Delmelle, C. Dony, I. Casas, M. Jia, and W. Tang. Visualizing the impact of space-time uncertainties on dengue fever patterns. *International Journal of Geographical Information Science*, 28(5):1107–1127, 2014.

[21] E. Gan and P. Bailis. Scalable kernel density classification via threshold-based pruning. In *ACM SIGMOD*, pages 945–959, 2017.

[22] T. M. Ghanem, M. A. Hammad, M. F. Mokbel, W. G. Aref, and A. K. Elmagarmid. Incremental evaluation of sliding-window queries over data streams. *IEEE Trans. Knowl. Data Eng.*, 19(1):57–72, 2007.

[23] A. Gramacki. *Nonparametric Kernel Density Estimation and Its Computational Aspects.* Studies in Big Data. Springer International Publishing, 2017.

[24] A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *SDM*, pages 203–211, 2003.

[25] T. Hart and P. Zandbergen. 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*, 37:305–323, 2014.

[26] A. Hohl, E. Delmelle, W. Tang, and I. Casas. Accelerating the discovery of space-time patterns of infectious diseases using parallel computing. *Spatial and Spatio-temporal Epidemiology*, 19:10 – 20, 2016.

[27] Y. Hu, F. Wang, C. Guin, and H. Zhu. A spatio-temporal kernel density estimation framework for predictive crime hotspot mapping and evaluation. *Applied Geography*, 99:89 – 97, 2018.

[28] Y. Kang, N. Cho, and S. Son. Spatiotemporal characteristics of elderly population's traffic accidents in Seoul using space-time cube and space-time kernel density estimation. *PLOS ONE*, 13(5):1–17, 05 2018.

[29] Y. Kestens, A. Lebel, M. Daniel, M. Thériault, and R. Pampalon. Using experienced activity spaces to measure foodscape exposure. *Health & Place*, 16(6):1094 – 1103, 2010.

[30] O. D. Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *PacificVis*, pages 171–178, 2011.

[31] J. Lee, J. Gong, and S. Li. Exploring spatiotemporal clusters based on extended kernel estimation methods. *Int. J. Geogr. Inf. Sci.*, 31(6):1154–1177, 2017.

[32] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. Semantics and evaluation techniques for window aggregates in data streams. In *SIGMOD*, pages 311–322, 2005.

[33] Y. Li, M. Abdel-Aty, J. Yuan, Z. Cheng, and J. Lu. Analyzing traffic violation behavior at urban intersections: A spatio-temporal kernel density estimation approach using automated enforcement system data. *Accident Analysis & Prevention*, 141:105509, 2020.

[34] Q. Liu, Y. Shen, and L. Chen. Lhist: Towards learning multi-dimensional histogram for massive spatial data. In *ICDE*, pages 1188–1199. IEEE, 2021.

[35] J. Lukasczyk, R. Maciejewski, C. Garth, and H. Hagen. Understanding hotspots: a topological visual analytics approach. In *SIGSPATIAL*, pages 36:1–36:10, 2015.

[36] A. Mayorga and M. Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1526–1538, Sept 2013.

[37] L. Micallef, G. Palmas, A. Oulasvirta, and T. Weinkauf. Towards perceptual optimization of the visual design of scatterplots. *IEEE Trans. Vis. Comput. Graph.*, 23(6):1588–1599, 2017.

[38] A. W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *UAI*, pages 397–405, 2000.

[39] T. Nakaya and K. Yano. Visualising crime clusters in a space-time cube: An exploratory data-analysis approach using space-time kernel density estimation and scan statistics. *Transactions in GIS*, 14(3):223–239, 2010.

[40] Y. Park, M. J. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. In *ICDE*, pages 755–766, 2016.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[42] A. Perrot, R. Bourqui, N. Hanusse, F. Lalanne, and D. Auber. Large interactive visualization of density functions on big data infrastructure. In *LDAV*, pages 99–106, 2015.

[43] J. M. Phillips. ϵ-samples for kernels. In *SODA*, pages 1622–1632, 2013.

[44] J. M. Phillips and W. M. Tai. Improved coresets for kernel density estimates. In *SODA*, pages 2718–2727, 2018.

[45] J. M. Phillips and W. M. Tai. Near-optimal coresets of kernel density estimates. In *SOCG*, pages 66:1–66:13, 2018.

[46] QGIS Development Team. *QGIS Geographic Information System.* Open Source Geospatial Foundation, 2009.

[47] X. Qin, Y. Luo, N. Tang, and G. Li. Making data visualization more efficient and effective: a survey. *VLDB J.*, 29(1):93–117, 2020.

[48] V. C. Raykar, R. Duraiswami, and L. H. Zhao. Fast computation of kernel estimators. *Journal of Computational and Graphical Statistics*, 19(1):205–220, 2010.

[49] H. Samet. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann. 2006.

[50] E. Saule, D. Panchananam, A. Hohl, W. Tang, and E. Delmelle. Parallel space-time kernel density estimation. In *ICPP*, pages 483–492, 2017.

[51] D. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization.* A Wiley-interscience publication. Wiley, 1992.

[52] A. Shein and P. K. Chrysanthis. Multi-query optimization of incrementally evaluated sliding-window aggregations. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[53] K. Tangwongsan, M. Hirzel, and S. Schneider. Optimal and general out-of-order sliding-window aggregation. *PVLDB*, 12(10):1167–1180, 2019.

[54] K. Tangwongsan, M. Hirzel, S. Schneider, and K. Wu. General incremental sliding-window aggregation. *PVLDB*, 8(7):702–713, 2015.

[55] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *IEEE Trans. Knowl. Data Eng.*, 18(2):377–391, 2006.

[56] A. C. Telea. *Data Visualization: Principles and Practice, Second Edition.* A. K. Peters, Ltd., Natick, MA, USA, 2nd edition, 2014.

[57] Á. Villalba, J. L. Berral, and D. Carrera. Constant-time sliding window framework with reduced memory footprint and efficient bulk evictions. *IEEE Trans. Parallel Distributed Syst.*, 30(3):486–500, 2019.

[58] C. Wang, H. Yu, and K. Ma. Importance-driven time-varying data visualization. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1547–1554, 2008.

[59] X. Wang, Y. Zhang, W. Zhang, X. Lin, and Z. Huang. SKYPE: top-k spatial-keyword publish/subscribe over sliding window. *PVLDB*, 9(7):588–599, 2016.

[60] Q. Wei, J. She, S. Zhang, and J. Ma. Using individual GPS trajectories to explore foodscape exposure: A case study in beijing metropolitan area. *International journal of environmental research and public health*, 15, 02 2018.

[61] K. Xie, K. Ozbay, A. Kurkcu, and H. Yang. Analysis of traffic crashes involving pedestrians using big data: Investigation of contributing factors and identification of hotspots. *Risk Analysis*, 37(8):1459–1476, 2017.

[62] C. Yang, R. Duraiswami, and L. S. Davis. Efficient kernel machines using the improved fast gauss transform. In *NIPS*, pages 1561–1568, 2004.

[63] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach.* Advances in Database Systems. Springer US, 2006.

[64] G. Zhang, A. Zhu, and Q. Huang. 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):2068–2097, 2017.

[65] Z. Zhang, D. Chen, W. Liu, J. Racine, S.-H. Ong, Y. Chen, G. Zhao, and Q. Jiang. Nonparametric evaluation of dynamic disease risk: A spatio-temporal kernel approach. *PloS one*, 6:e17381, 03 2011.

[66] X. Zhao and J. Tang. Crime in urban areas: A data mining perspective. *SIGKDD Explorations*, 20(1):1–12, 2018.

[67] Y. Zheng, J. Jestes, J. M. Phillips, and F. Li. Quality and efficiency for kernel density estimates in large data. In *SIGMOD*, pages 433–444, 2013.

[68] Y. Zheng, Y. Ou, A. Lex, and J. M. Phillips. Visualization of big spatial data using coresets for kernel density estimates. In *IEEE Symposium on Visualization in Data Science (VDS '17), to appear*. IEEE, 2017.

[69] Y. Zheng and J. M. Phillips. L∞ error and bandwidth selection for kernel density estimates of large data. In *SIGKDD*, pages 1533–1542, 2015.

[70] Y. Zhou, J. Li, H. Chen, Y. Wu, J. Wu, and L. Chen. A spatiotemporal attention mechanism-based model for multi-step citywide passenger demand prediction. *Inf. Sci.*, 513:372–385, 2020.

[71] Z. Zhou and D. S. Matteson. Predicting ambulance demand: a spatio-temporal kernel approach. In *SIGKDD*, pages 2297–2303, 2015.

[72] R. Zhu, B. Wang, X. Yang, B. Zheng, and G. Wang. SAP: improving continuous top-k queries over streaming data. *IEEE Trans. Knowl. Data Eng.*, 29(6):1310–1328, 2017.