LARGE: A Length-Aggregation-based Grid Structure for Line Density Visualization

Tsz Nam Chan
Shenzhen University
edisonchan@szu.edu.cn

Bojian Zhu
Hong Kong Baptist University
csbjzhu@comp.hkbu.edu.hk

Dingming Wu
Shenzhen University
dingming@szu.edu.cn

Yun Peng
Institute of Artificial Intelligence and Blockchain, Guangzhou University
yunpeng@gzhu.edu.cn

Leong Hou U
University of Macau
ryanlhu@um.edu.mo

ABSTRACT

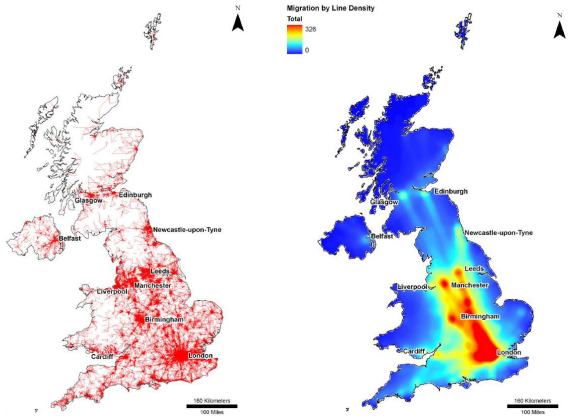
Line Density Visualization (LDV) is an important operation of geospatial analysis, which has been extensively used in many application domains, e.g., urban planning, criminology, and transportation science. However, LDV is computationally demanding. Therefore, existing exact solutions are not scalable (or even not feasible) to support large-scale datasets and high resolution sizes for generating LDV. To handle the efficiency issues, we develop the first solution to approximately compute LDV with an ϵ-relative error guarantee, which consists of two main parts. First, we develop the new indexing structure, called length-aggregation-based grid structure (LARGE). Second, based on LARGE, we develop two types of fast bound functions, namely (1) square-shaped lower and upper bound functions and (2) arbitrary-shaped lower and upper bound functions, which can filter a large portion of unnecessary computations. By theoretically analyzing the tightness of our bound functions and experimentally comparing our solution with existing exact solutions on four large-scale datasets, we demonstrate that our solution can be scalable to generate high-resolution LDVs using large-scale datasets. In particular, our solution achieves up to 291.8x speedups over the state-of-the-art solutions.

PVLDB Reference Format:
Tsz Nam Chan, Bojian Zhu, Dingming Wu, Yun Peng, and Leong Hou U. LARGE: A Length-Aggregation-based Grid Structure for Line Density Visualization. PVLDB, 17(13): 4585 - 4598, 2024.
doi:10.14778/3704965.3704968

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/edisonchan2013928/LARGE.

1 INTRODUCTION

Density visualization [80, 84] is an important field in many communities. Among most of these tools, Line Density Visualization (LDV) [84] is a de facto tool for analyzing the density of flow data/trajectory data, which has been extensively used in many application domains. Urban planners and ecologists adopt LDV to (1) analyze human mobility and animal mobility, respectively [11, 31, 38, 41, 61, 74, 75, 77], (2) conduct urban analysis [62, 94, 95], (3) conduct disaster analysis [9, 39], and (4) conduct environmental analysis [83, 90]. Criminologists utilize LDV to understand crime patterns in different geographical regions [85, 86]. Transportation experts utilize LDV to analyze traffic flows/trajectories in different cities [41, 50, 60, 64, 69, 93]. Figure 1 shows how the domain expert [75] uses LDV to analyze the density of migration flows in the UK mobility dataset. Observe that the flows mainly concentrate on the south part of the United Kingdom.



(a) Line segment (flow) dataset (b) Line density visualization (LDV)

Figure 1: Generate LDV (in (b)) for the UK mobility dataset, where the red line segments in (a) denote the migration flows that move from one place to another place (Obtained from [75]).

Due to the wide applicability of LDV, many contemporary and famous software platforms, e.g., QGIS [5] and ArcGIS [1], can also support LDV. However, LDV is a slow operation, which takes $O(XYn)$ time for generating a single visualization, where $X \times Y$ and n denote the resolution size and the number of line segments in a dataset, respectively. Using the New York taxi dataset (with 13.6 million line segments) as an example, generating a single LDV with the resolution size 1280×960 based on the naïve method takes 16.7 trillion operations. As such, many domain experts have pointed out the efficiency issues to adopt the LDV tool for performing geospatial analysis [26, 66]. Despite this, there is still a lack of research studies to develop efficient algorithms for supporting this tool.

Dingming Wu is the corresponding author of this paper.

Proceedings of the VLDB Endowment, Vol. 17, No. 13 ISSN 2150-8097.
doi:10.14778/3704965.3704968

To boost the efficiency of using LDV, we ask a question in this paper. *Can we develop efficient algorithms for generating LDV, without degrading the visualization quality?* In order to provide an affirmative answer to this question, we have proposed the first approximate solution that can efficiently compute LDV with an ϵ-relative error guarantee, by (1) developing the indexing structure, called Length-AggRegation-based Grid structurE (LARGE), (2) developing the tight lower and upper bound functions that can efficiently filter a large portion of unnecessary computations (based on LARGE), and (3) discovering that our bound functions can be theoretically tighter if the pixel size is smaller (i.e., high resolution) or the bandwidth value (will be discussed in Section 2.1) is larger, which indicates that LARGE can be scalable to high resolution sizes and large bandwidth values. To the best of our knowledge, this theoretical result has not been achieved by any previous work. In practice, our experiment results on four large-scale datasets (up to 14.3 million line segments) show that LARGE can achieve speedups of 2.35x to 291.8x over the state-of-the-art methods, without incurring huge space overhead. Furthermore, our case study also verifies that LARGE does not degrade the visualization quality compared with the exact approach. Moreover, we have also developed the new QGIS plugin (based on LARGE), called Fast Line Density Analysis [2], for efficiently supporting LDV.

The rest of the paper is structured as follows. We first formally define LDV and discuss two baseline (exact) solutions in Section 2. Then, we discuss our solution, LARGE, in Section 3. Next, we provide the experimental evaluation in Section 4. After that, we review the related work in Section 5. Lastly, we conclude this paper in Section 6.

2 PRELIMINARIES

In this section, we first discuss the problem definition of LDV in Section 2.1. Then, we discuss two baseline solutions, which are (1) sequential scan and (2) hierarchical indexing framework, in Section 2.2 and Section 2.3, respectively.

2.1 Problem Definition

In order to generate LDV (cf. Figure 1b) for a line segment dataset (cf. Figure 1a), we need to color each pixel $\mathbf{q} = (q_x, q_y)$ based on the line density function $\mathcal{L}(\mathbf{q})$, which counts the accumulated length of all line segments that are within the search range (or the bandwidth) b from $\mathbf{q}$ per unit area. Using Figure 2 as an example, the line density function value $\mathcal{L}(\mathbf{q})$ (for the pixel $\mathbf{q}$) is $\frac{L_1+L_2+L_3}{\pi b^2}$.

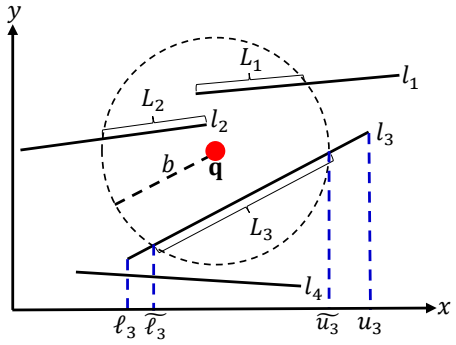


Figure 2: The line density function value for the pixel $\mathbf{q}$ is $\frac{L_1+L_2+L_3}{\pi b^2}$.

Based on the above discussion, we formally define LDV. Here, we first define the concept of line segment in Definition 1.

Definition 1. *Each line segment l_i is represented by the equation $y = m_i x + k_i$, where $\ell_i \le x \le u_i$.*

To find the length L_i of each line segment l_i that is within the bandwidth b from $\mathbf{q}$ (i.e., the dashed circle in Figure 2), we need to find $\widetilde{\ell_i}$ and $\widetilde{u_i}$ (e.g., $\widetilde{\ell_3}$ and $\widetilde{u_3}$ in Figure 2) such that:

$$\widetilde{\ell_i} = \min(\{x \in [\ell_i, u_i] : y = m_i x + k_i, (x-q_x)^2 + (y-q_y)^2 \le b^2\}) \quad (1)$$

$$\widetilde{u_i} = \max(\{x \in [\ell_i, u_i] : y = m_i x + k_i, (x-q_x)^2 + (y-q_y)^2 \le b^2\}) \quad (2)$$

As a remark, we denote $\widetilde{\ell_i} = \phi$ and $\widetilde{u_i} = \phi$ to indicate that the line segment is never inside the bandwidth b from $\mathbf{q}$ (e.g., $\widetilde{\ell_4} = \phi$ and $\widetilde{u_4} = \phi$ in Figure 2).

Based on simple mathematical operations, the length L_i is stated in Equation 3.

$$L_i = \begin{cases} \sqrt{1+m_i^2} \cdot |\widetilde{u_i} - \widetilde{\ell_i}| & \text{if } \widetilde{\ell_i} \neq \phi \text{ and } \widetilde{u_i} \neq \phi \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

With the above concepts, LDV is formally stated in Definition 2.

Definition 2. *[1, 5] Given a line segment dataset $\mathbb{L} = \{l_1, l_2, ..., l_n\}$ with size n, a resolution size $X \times Y$, and a bandwidth parameter b, we need to compute $\mathcal{L}(\mathbf{q})$ for each pixel $\mathbf{q} = (q_x, q_y)$, where*

$$\mathcal{L}(\mathbf{q}) = \frac{1}{\pi b^2} \sum_{i=1}^{n} L_i \quad (4)$$

2.2 Baseline Solution 1: Sequential Scan

To generate LDV, a simple approach, which has been adopted in QGIS [5] and ArcGIS [1], is to first scan each line segment l_i in the line segment dataset $\mathbb{L}$ and then evaluate the length L_i (cf. Equation 3) in order to compute the line density function $\mathcal{L}(\mathbf{q})$ for each pixel $\mathbf{q}$ (cf. Equation 4). By considering four possible cases of the endpoints, i.e., (ℓ_i, y_{ℓ_i}) and (u_i, y_{u_i}), of each line segment (cf. Figure 3), L_i can be computed in $O(1)$ time (i.e., this simple approach can generate LDV in $O(XYn)$ time.).

Case 1: Observe from Figure 3a that these two endpoints are within the bandwidth b from the pixel $\mathbf{q}$. Therefore, we can simply compute L_i by setting $\widetilde{\ell_i} = \ell_i$ and $\widetilde{u_i} = u_i$ in Equation 3, which takes $O(1)$ time.

Case 2: In Figure 3b, note that (ℓ_i, y_{ℓ_i}) and (u_i, y_{u_i}) are inside and outside the search range, respectively. Therefore, this line segment l_i must intersect the black dashed circle. Once we substitute the equation $y = m_i x + k_i$ into $(x-q_x)^2 + (y-q_y)^2 = b^2$, there must exist the solution $x = \widetilde{u_i}$ for the following quadratic equation.

$$Ax^2 + Bx + C = 0$$

where $A = 1 + m_i^2$, $B = 2m_i k_i - 2q_x - 2m_i q_y$, and $C = q_x^2 + k_i^2 - 2k_i q_y + q_y^2 - b^2$. As such, we can compute $\widetilde{u_i}$ based on the following equation.

$$\widetilde{u_i} = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \quad (5)$$

Since the parameters A, B, and C can be computed in $O(1)$ time, we can obtain $\widetilde{u_i}$ and the length L_i (by using $\widetilde{\ell_i} = \ell_i$ and this $\widetilde{u_i}$ in Equation 3) in $O(1)$ time.

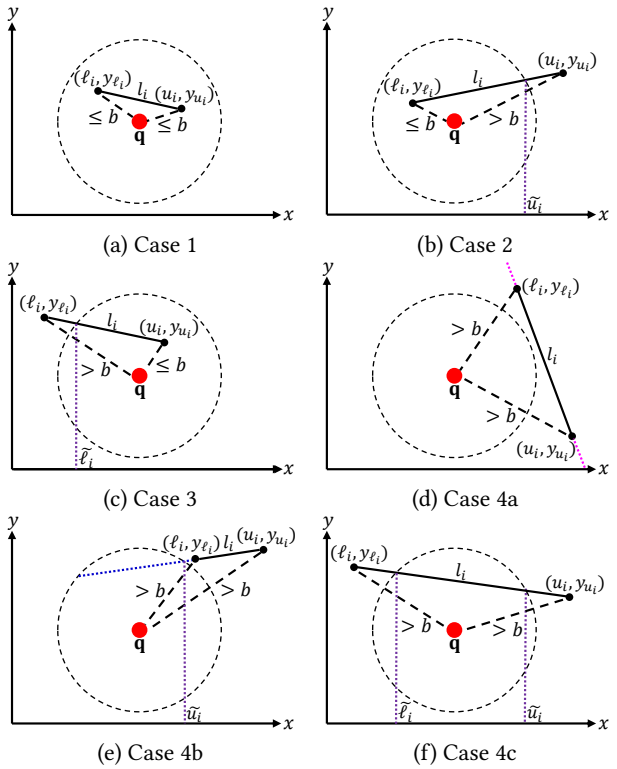


Figure 3: Four possible cases for the endpoints of each line segment.

Case 3: In Figure 3c, observe that (ℓ_i, y_{ℓ_i}) and (u_i, y_{u_i}) are outside and inside the search range, respectively. This case is similar to Case 2. Instead, we need to find $\widetilde{\ell_i}$ where

$$\widetilde{\ell_i} = \frac{-B - \sqrt{B^2 - 4AC}}{2A} \tag{6}$$

As such, the length L_i can be computed in $O(1)$ time.

Case 4: In this case, both (ℓ_i, y_{ℓ_i}) and (u_i, y_{u_i}) are not within the search range b from $\mathbf{q}$. This case is the most complicated one, which consists of three subcases (cf. Figures 3d-f).

Case 4a: Observe from Figure 3d that the extended line (purple dotted line) never intersects the black dashed circle. We can identify this subcase (i.e., the length $L_i = 0$) if $B^2 - 4AC < 0$, which can be computed in $O(1)$ time.

Case 4b: In this subcase (cf. Figure 3e), we have $B^2 - 4AC \geq 0$. Once $\widetilde{u_i} < \ell_i$, we can conclude $L_i = 0$ (with $O(1)$ time).[1]

Case 4c: In Figure 3f, note that we also have $B^2 - 4AC \geq 0$. Once $\ell_i \leq \widetilde{\ell_i} \leq \widetilde{u_i} \leq u_i$, we can directly compute L_i based on Equation 3 in $O(1)$ time.

2.3 Baseline Solution 2: Hierarchical Indexing Framework

Observe from Figure 4 that once the minimum bounding rectangle R does not intersect the search region (black dashed circle) of $\mathbf{q}$ (i.e., $mindist(\mathbf{q}, R) > b$), all the line segments that are inside R cannot contribute to the line density function $\mathcal{L}(\mathbf{q})$ (like Figure 3d and Figure 3e). Therefore, we can avoid the computation of the length L_i for all those line segments in R.

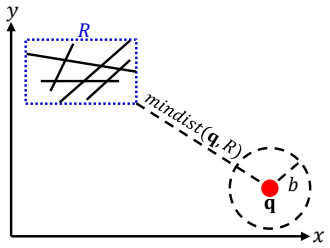


Figure 4: Those line segments in the minimum bounding rectangle R can be filtered since all those line segments are far away from the pixel $\mathbf{q}$ (i.e., the minimum distance between $\mathbf{q}$ and R is larger than b ($mindist(\mathbf{q}, R) > b$)).

Inspired by this idea, one basic method is to extend the hierarchical indexing framework for line segments [45–47] (cf. Figure 5) to improve the performance for evaluating the line density function $\mathcal{L}(\mathbf{q})$. Some representative indexing structures include R-tree [54] and PMR quadtree [45–47]. Given a pixel $\mathbf{q}$ (e.g., the red point in Figure 5), this method iteratively traverses each node (starting from the root node). If the minimum bounding rectangle of each node intersects the search region, it continuously traverses its child nodes. As an example, since the blue dotted rectangle of the node R_{root} intersects the black dashed circle, this method needs to traverse R_1 and R_2. Otherwise, this method avoids traversing its child nodes. For example, since $mindist(\mathbf{q}, R_1) > b$, we do not need to traverse R_3 and R_4. Once this method reaches the leaf node, it adopts the sequential scan method (cf. Section 2.2) to evaluate the length L_i for each line segment l_i in this node if the corresponding minimum bounding rectangle intersects the search range. For example, we need to compute the lengths L_5 and L_6 (for the line segments l_5 and l_6, respectively) since the minimum bounding rectangle of the node R_5 intersects the search range of $\mathbf{q}$. Otherwise, this method avoids the computation of the length for each line segment in the leaf node (e.g., l_7 and l_8 in the leaf node R_6).

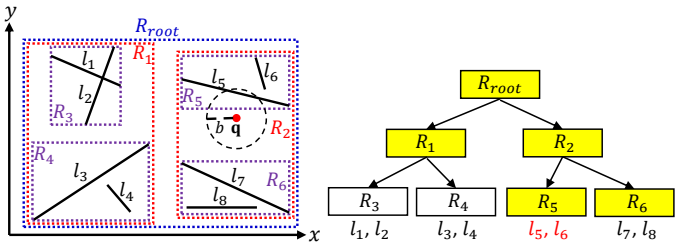


Figure 5: A hierarchical indexing framework. This method traverses the yellow nodes and compute the lengths, L_5 and L_6, for the line segments (in red), l_5 and l_6, respectively, in order to evaluate $\mathcal{L}(\mathbf{q})$ for the pixel $\mathbf{q}$.

Although this method can efficiently filter those line segments that are far away from the pixel $\mathbf{q}$ (cf. Figure 4), it cannot improve the efficiency for handling those line segments that are (1) close to the pixel $\mathbf{q}$ (e.g., Figures 3a, b, c, and f) and (2) long (which results in large minimum bounding rectangles). Hence, if we consider the large line segment dataset (i.e., more line segments can possibly

[1] For sake of simplicity, we omit the subcase $\widetilde{\ell_i} > u_i$, which also leads to $L_i = 0$.

intersect the search range of each pixel $\mathbf{q}$ and can be possibly long.), this method can still be slow.

3 OUR SOLUTION

In this section, we first discuss the core idea of our solution in Section 3.1. Then, we propose the new indexing structure, called Length-AggRegation-based Grid structurE (LARGE), in Section 3.2. Next, we further propose the square-shaped and arbitrary-shaped (lower and upper) bound functions in Section 3.3 and Section 3.4, respectively. After that, we illustrate how to incorporate these lower and upper bound functions into the filter and refinement framework for efficiently generating LDV in Section 3.5. Lastly, we further investigate the tightness of our bound functions in Section 3.6.

3.1 Core Idea

Observe from Figure 6 that all line segments intersect the search range (i.e., the black dashed circle) of the pixel $\mathbf{q}$. As such, the state-of-the-art indexing methods (cf. Section 2.3) cannot filter all these line segments, which can incur huge computational overhead. However, not the full portion of each line segment is important for the pixel $\mathbf{q}$. For example, the red portion of each line segment only intersects the black dashed circle with a small length, which only contributes a small value for the line density function $\mathcal{L}(\mathbf{q})$ (cf. Equation 4). Hence, a core idea is that it is possible to achieve a good approximation of $\mathcal{L}(\mathbf{q})$ if we only compute the majority of density values for those line segments (i.e., with black portion), which can further improve the efficiency without significantly degrading the visualization quality.

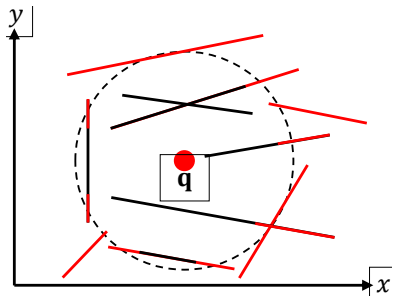


Figure 6: All line segments can contribute to the line density function $\mathcal{L}(\mathbf{q})$. However, the black portion of each line segment is more important, which can contribute more density values.

Therefore, instead of adopting exact solutions (like Section 2.2 and Section 2.3) for generating LDV, we aim to generate approximate LDV with a small relative error ϵ (cf. Definition 3) in this paper.

Definition 3. *Given a line segment dataset $\mathbb{L} = \{l_1, l_2, ..., l_n\}$ with size n, a resolution size $X \times Y$, a bandwidth parameter b, and a relative error ϵ, we need to obtain the result $R(\mathbf{q})$ for each pixel $\mathbf{q}$ so that*

$$(1-\epsilon)\mathcal{L}(\mathbf{q}) \leq R(\mathbf{q}) \leq (1+\epsilon)\mathcal{L}(\mathbf{q}) \quad (7)$$

3.2 LARGE: A Length-Aggregation-based Grid Structure

In order to efficiently compute LDV with an ϵ-relative error guarantee (cf. Definition 3), we construct the new indexing structure, called LARGE (i.e., length-aggregation-based grid structure), which consists of three steps, namely (1) obtain the extended region for the plane with $X \times Y$ pixels, (2) aggregate the accumulated length of each grid in the extended region, and (3) build the prefix-sum grid structure based on the accumulated lengths of all grids.

Step 1: Recall from Figure 2 that we need to count the accumulated length for all line segments that are within the bandwidth b from each pixel $\mathbf{q}$ (where each pixel $\mathbf{q}$ covers the grid with size $\delta_x \times \delta_y$ in Figure 7). Therefore, we append the additional grids (i.e., the blue grids in Figure 7) so that this extended region can cover the search range (the black dashed circle) of every pixel $\mathbf{q}$.

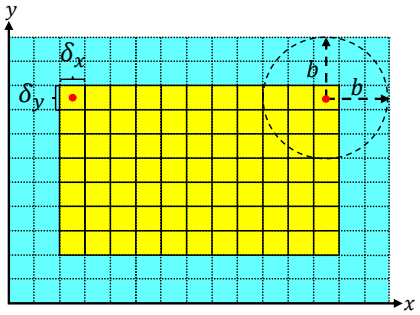


Figure 7: Obtain the extended region (the yellow and blue regions) from the original plane with $X \times Y$ pixels (the yellow region), given the bandwidth value b.

In practice, if the bandwidth b is very large, LDV can assign high density values for all pixels (cf. Equations 1 to 4), which can provide meaningless visualization. Hence, we have an assumption that $b \leq \min((X-0.5)\delta_x, (Y-0.5)\delta_y)$ (i.e., the search range in Figure 7 cannot cover all X yellow grids in a row and all Y yellow grids in a column) in this paper. With this assumption, we state in Lemma 1 that the number of grids in the extended region remains in $O(XY)$.

Lemma 1. *Consider the pixel plane with size $X \times Y$ and each pixel covers the grid with size $\delta_x \times \delta_y$. If the bandwidth $b \leq \min((X-0.5)\delta_x, (Y-0.5)\delta_y)$, there are at most $O(XY)$ grids in the extended region.*

Proof. Since each pixel has size $\delta_x \times \delta_y$ and $b \leq \min((X-0.5)\delta_x, (Y-0.5)\delta_y)$, this bandwidth value can cover at most $\min(X, Y)$ pixels. Observe from Figure 7 that the x-axis and the y-axis can cover at most $O(X + \min(X, Y))$ and $O(Y + \min(X, Y))$ pixels, respectively. Therefore, the extended region still covers at most $O(XY)$ grids. □

Step 2: After we have obtained the extended region (i.e., the colored grids in Figure 8), we need to augment the accumulated length of each grid in this region (e.g., augment $\rho_3 + \rho_4$ for the pink grid in Figure 8). To achieve this goal, we need to first find the intersection point that is closest to the starting point (ℓ_i, y_{ℓ_i}) (i.e., with the minimum x-coordinate) in the extended region for each line segment l_i (cf. red dashed circles in Figure 8).

In Figure 9, note that the starting point (ℓ_i, y_{ℓ_i}) (i.e., the black point) of any line segment l_i can lie on seven possible regions (I to VII), e.g., both (ℓ_2, y_{ℓ_2}) and (ℓ_3, y_{ℓ_3}) in Figure 8 belong to Region II of Figure 9. We examine, in each of these seven regions, the potential intersections of line segments (indicated by dashed lines)

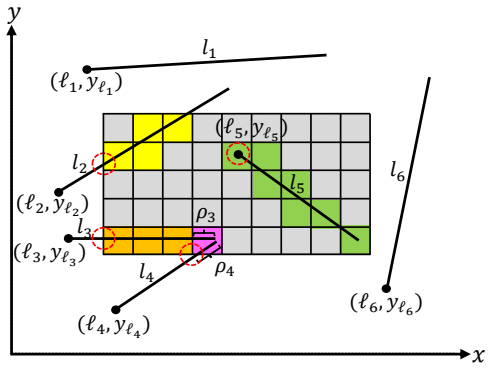


Figure 8: Obtain the accumulated length of each grid in the extended region (i.e., the colored grids) for this example of six line segments, where the accumulated length for the pink grid is $\rho_3 + \rho_4$.

originating from a starting point with the extended region (and the corresponding intersection points that are closest to this starting point).

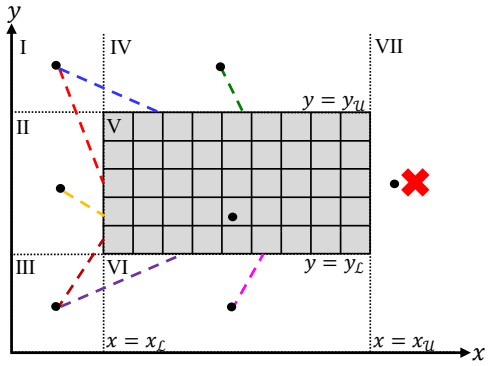


Figure 9: Given the extended region (i.e., the grey region), the starting point (ℓ_i, y_{ℓ_i}) of each line segment l_i (the black data point) with equation $y = m_i x + k_i$, where $\ell_i \leq x \leq u_i$, can be in seven possible regions (from I to VII). The dashed lines (from I to VI) denote the possible line segments that can intersect the extended region, while the cross symbol (from VII) indicates that all line segments (with the corresponding starting points) never intersect the extended region.

Consider the starting point (ℓ_i, y_{ℓ_i}) that is in the region I, region II, or region III. The line segment from this starting point can intersect the left boundary (e.g., the red dashed line segment, the orange dashed line segment, or the brown dashed line segment in Figure 9) of the extended region with the intersection point $(x_{\mathcal{L}}, m_i x_{\mathcal{L}} + k_i)$ if $y_{\mathcal{L}} \leq m_i x_{\mathcal{L}} + k_i \leq y_{\mathcal{U}}$ and the Euclidean distance between $(x_{\mathcal{L}}, m_i x_{\mathcal{L}} + k_i)$ and (ℓ_i, y_{ℓ_i}) is less than the bandwidth b.

Consider the starting point (ℓ_i, y_{ℓ_i}) that is in the region I or region IV. The line segment from this starting point can intersect the upper boundary (e.g., the blue dashed line segment or the green dashed line segment in Figure 9) with the intersection point $(\frac{y_{\mathcal{U}} - k_i}{m_i}, y_{\mathcal{U}})$ if $x_{\mathcal{L}} \leq \frac{y_{\mathcal{U}} - k_i}{m_i} \leq x_{\mathcal{U}}$ and the Euclidean distance between $(\frac{y_{\mathcal{U}} - k_i}{m_i}, y_{\mathcal{U}})$ and (ℓ_i, y_{ℓ_i}) is less than the bandwidth b.

Consider the starting point (ℓ_i, y_{ℓ_i}) that is in the region III and the region VI. The line segment from this starting point can possibly intersect the lower boundary (e.g., the purple dashed line segment or the pink dashed line segment in Figure 9) with the intersection point $(\frac{y_{\mathcal{L}} - k_i}{m_i}, y_{\mathcal{L}})$ if $x_{\mathcal{L}} \leq \frac{y_{\mathcal{L}} - k_i}{m_i} \leq x_{\mathcal{U}}$ and the Euclidean distance between $(\frac{y_{\mathcal{L}} - k_i}{m_i}, y_{\mathcal{L}})$ and (ℓ_i, y_{ℓ_i}) is less than the bandwidth b.

Consider the starting point (ℓ_i, y_{ℓ_i}) that is in the region V. The intersection point is the same as the starting point (ℓ_i, y_{ℓ_i}).

Consider the starting point (ℓ_i, y_{ℓ_i}) that is in the region VII. The line segment never intersects the extended region (i.e., no intersection point) as the x-coordinate of the line segment l_i must fulfill $\ell_i \leq x \leq u_i$ (cf. Definition 1).

After we have found the intersection point (that is closest to (ℓ_i, y_{ℓ_i})), we then illustrate how to aggregate the length for each grid that is intersected by this line segment l_i (cf. Figure 8). Observe from Figure 10 that there are five possible cases for the line segment to intersect the boundaries of this grid, which are (1) intersecting the upper boundary, (2) intersecting the upper right corner, (3) intersecting the right boundary, (4) intersecting the lower right corner, and (5) intersecting the lower boundary. By substituting the correct equation of the boundary (e.g., $y = y_L + (v+1)\delta_y$) into the equation of the line segment $y = m_i x + k_i$, we can find the intersection point (e.g., (1)) in $O(1)$ time. However, not all line segments can be long enough to intersect the boundaries (e.g., the green dashed line in Figure 10). In this case, the length between the "intersection point" (i.e., green dashed circle) and the initial intersection point (x_I, y_I) must be larger than the length between the endpoint (u_i, y_{u_i}) and (x_I, y_I), which can also be computed in $O(1)$ time. Once we have the new intersection point (e.g., blue dashed circle) or the end point (u_i, y_{u_i}) (e.g., green point), we can use $O(1)$ time to aggregate the distance for this grid. This process iteratively moves to (and aggregate the distance for) the next grid (e.g., the upper grid for (1)) with the new intersection point until it reaches the rightmost boundary (i.e., $x = x_{\mathcal{U}}$ in Figure 9), the uppermost boundary (i.e., $y = y_{\mathcal{U}}$ in Figure 9), or the lowermost boundary (i.e., $y = y_{\mathcal{L}}$ in Figure 9), or it reaches the end point (u_i, y_{u_i}) (e.g., the green point in Figure 10).

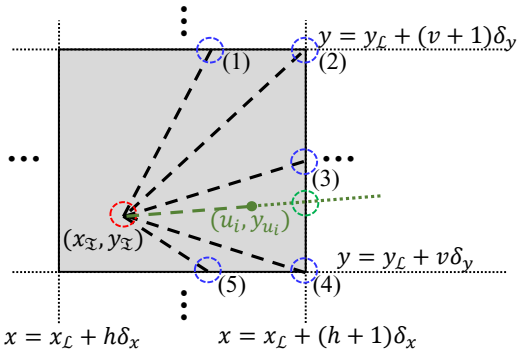


Figure 10: Five possible cases (blue dashed circles) for the next intersection point.

Here, we state in Lemma 2 that obtaining the accumulated lengths of all grids in the extended region takes $O((X+Y)n)$ time, given a line segment dataset $\mathbb{L}$ with size n.

LEMMA 2. *Given a line segment dataset $\mathbb{L} = \{l_1, l_2, ..., l_n\}$ with size n, the time complexity for obtaining the accumulated lengths of all grids in the extended region is $O((X+Y)n)$.*

Proof. In this proof, we aim to show that obtaining the accumulated lengths of all grids for only a single line segment l_i takes $O(X+Y)$ time (i.e., $O((X+Y)n)$ time for n line segments). Moreover, we also assume that l_i has the slope $m_i > 0$ (i.e., the case (4) and case (5) never happen in Figure 10). However, we can easily extend this proof for the line segment l_i with $m_i \leq 0$.

Observe from Figure 10 that the line segment l_i with $m_i > 0$ can possibly intersect three boundaries, i.e., the upper boundary (case (1)), the upper-right corner (case (2)), and the right boundary (case (3)), which indicates that the next intersection point is in the upper grid (i.e., move one grid up in the y-axis), the upper right grid (i.e., move one grid right in the x-axis and move one grid up in the y-axis), and the right grid (i.e., move one grid right in the x-axis), respectively. Therefore, this process follows these three cases to aggregate the distance for the next grid during the iteration. Since there are at most $O(X)$ grids and $O(Y)$ grids in the x-axis and the y-axis, respectively (cf. Lemma 1), finding the accumlated lengths of all grids for a single line segment takes $O(X+Y)$ time. Therefore, the time complexity for using n line segments is $O((X+Y)n)$. □

Step 3: Once we have obtained the grid structure G, where $G[\alpha, \beta]$ denotes the accumulated length for the grid in which the corresponding pixel has the coordinates $(x_{\mathcal{L}} + (\alpha - 0.5)\delta_x, y_{\mathcal{L}} + (\beta - 0.5)\delta_y)$ $(1 \leq \alpha \leq X$ and $1 \leq \beta \leq Y)$, in step 2 (cf. Figure 11a), we can construct the prefix-sum grid structure PG (cf. Figure 11b), based on the concept of prefix-sum array [44], where $PG[\alpha^*, \beta^*]$ represents the sum of all values in the grid structure G with $1 \leq \alpha \leq \alpha^*$ and $1 \leq \beta \leq \beta^*$ (e.g., the value 24.8 in the light blue grid of Figure 11b denotes the sum of all values (i.e., $5+5+5+9.8$) in the red rectangle of Figure 11a), which is stated in Equation 8.

$$PG[\alpha^*, \beta^*] = \sum_{\alpha=1}^{\alpha^*} \sum_{\beta=1}^{\beta^*} G[\alpha, \beta] \tag{8}$$

0	2.9	4.2	0	0	0	0	0	0
6.5	0.8	0	0	1.6	0.7	0	0	0
0	0	0	0	0	3.1	3	0	0
0	0	0	0	0	0	0.6	5.8	0
5	5	5	9.8	0	0	0	0	2.7

(a) Grid structure with the accumulated length for each grid

11.5	20.2	29.4	39.2	40.8	44.6	48.2	54	56.7
11.5	17.3	22.3	32.1	33.7	37.5	41.1	46.9	49.6
5	10	15	24.8	24.8	27.9	31.5	37.3	40
5	10	15	24.8	24.8	24.8	25.4	31.2	33.9
5	10	15	24.8	24.8	24.8	24.8	24.8	27.5

(b) Prefix-sum grid structure (based on length-aggregation)

Figure 11: Build the indexing structure, called LARGE, which is the prefix-sum grid structure, for the extended region in Figure 8.

With this prefix-sum grid structure PG, we can compute the aggregation of all lengths in any rectangular region of grids in G, $\alpha_L \leq \alpha \leq \alpha_U$ and $\beta_L \leq \beta \leq \beta_U$, by accessing at most four grids in the prefix-sum grid structure. Using Figure 11a as an example, the aggregation of all lengths in the blue rectangle is 9, which can be computed based on four green grids in Figure 11b (i.e., $41.1 - 24.8 - 22.3 + 15$).

In Lemma 3, we state that constructing the prefix-sum grid structure PG and obtaining the aggregation of all lengths in any rectangular region of grids in G take $O(XY)$ time and $O(1)$ time, respectively, after we have obtained the grid structure G from step 2. This conclusion can be easily extended from [44], which is omitted in this paper. More details about the prefix-sum grid structure PG are available in Section I of the supplementary document [25] of this paper.

Lemma 3. *Given an extended region and its grid structure G, the time complexities for constructing the prefix-sum grid structure PG and obtaining the aggregation of all lengths in any rectangular region of grids in G are $O(XY)$ and $O(1)$, respectively.*

Time and space complexities of LARGE: In order to construct the indexing structure, LARGE, we need to obtain the extended region (i.e., step 1), which takes $O(XY)$ time (cf. Lemma 1), obtain the accumulated lengths of all grids in the extended region (i.e., step 2), which takes $O((X+Y)n)$ time (cf. Lemma 2), and construct the prefix-sum grid structure (i.e., step 3), which takes $O(XY)$ time (cf. Lemma 3). As such, we state that the time complexity for constructing LARGE is $O((X+Y)n + XY)$ time (cf. Theorem 1), which is much faster than the time complexity for computing LDV (with $O(XYn)$ time).

Theorem 1. *Given a line segment dataset $\mathbb{L} = \{l_1, l_2, ..., l_n\}$ with size n and a resolution size $X \times Y$, the time complexity for constructing LARGE is $O((X+Y)n + XY)$.*

Note that computing LDV needs to access XY pixels and n line segments. Therefore, every algorithm takes at least $O(XY+n)$ space. Since obtaining the extended region and constructing the prefix-sum grid structure only take $O(XY)$ additional space, the space complexity of LARGE remains in $O(XY+n)$ (cf. Theorem 2).

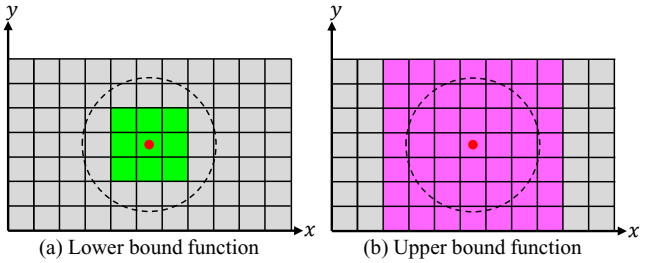
Theorem 2. *Given a line segment dataset $\mathbb{L} = \{l_1, l_2, ..., l_n\}$ with size n and a resolution size $X \times Y$, the space complexity for constructing LARGE is $O(XY+n)$.*

3.3 Square-shaped Lower and Upper Bound Functions

After we have obtained the indexing structure, LARGE, we aim to efficiently compute the lower and upper bound functions of $\mathcal{L}(\mathbf{q})$ (cf. Equation 4) for all pixels. Observe from Figure 12a that those green grids, which form the square-shaped region, are fully covered by the search region (i.e., black dashed circle). As such, the total length values that are assigned for all these green grids $LB_{\square}(\mathbf{q})$ can contribute to $\mathcal{L}(\mathbf{q})$ (i.e., the red pixel $\mathbf{q}$). Hence, we have $\mathcal{L}(\mathbf{q}) \geq LB_{\square}(\mathbf{q})$. On the other hand, the pink square-shaped region can fully cover the dashed circle (cf. Figure 12b). Therefore, the total length values of all these pink grids $UB_{\square}(\mathbf{q})$ must act as the upper bound value for $\mathcal{L}(\mathbf{q})$. Therefore, we have $\mathcal{L}(\mathbf{q}) \leq UB_{\square}(\mathbf{q})$.

However, if we directly compute the lower bound function $LB_{\square}(\mathbf{q})$ and the upper bound function $UB_{\square}(\mathbf{q})$, we need to scan all these green grids and pink grids, respectively, which can take $O(XY)$ time in the worst case for each pixel $\mathbf{q}$ (i.e., $O(X^2Y^2)$ time for all pixels). Recall that LARGE is the prefix-sum grid structure (cf. Figure 11b), which can be used to obtain the aggregation of all lengths in any rectangular region of grids with $O(1)$ time (cf. Lemma 3). Therefore, we can compute $LB_{\square}(\mathbf{q})$ and $UB_{\square}(\mathbf{q})$ in $O(1)$ time once LARGE has been built (cf. Theorem 3).

Theorem 3. *Suppose that the indexing structure LARGE has been built for a line segment dataset $\mathbb{L} = \{l_1, l_2, ..., l_n\}$ with size n, computing the lower bound function $LB_{\square}(\mathbf{q})$ and the upper bound function $UB_{\square}(\mathbf{q})$ for each pixel takes $O(1)$ time.*

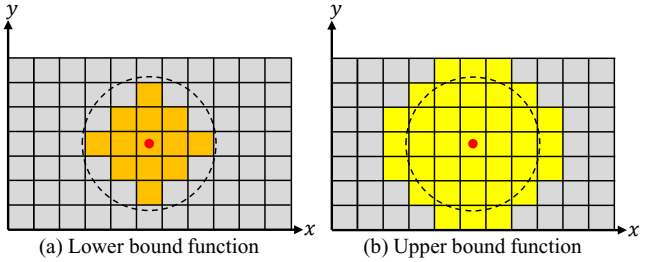


(a) Lower bound function (b) Upper bound function

Figure 12: Illustration of the square-shaped lower and upper bound functions, i.e., $LB_{\square}(\mathbf{q})$ and $UB_{\square}(\mathbf{q})$, respectively, for the red pixel $\mathbf{q}$.

3.4 Arbitrary-shaped Lower and Upper Bound Functions

Although the square-shaped lower and upper bound functions can be computed in $O(1)$ time (cf. Theorem 3), these bound functions may not be tight enough. Observe from Figure 12a that four grey grids are fully covered by the dashed circle, which are omitted by $LB_{\square}(\mathbf{q})$. In addition, some pink grids do not intersect the dashed circle (e.g., those pink grids near the corners in Figure 12b), in which their accumulated lengths are still aggregated in $UB_{\square}(\mathbf{q})$.



(a) Lower bound function (b) Upper bound function

Figure 13: Illustration of the arbitrary-shaped lower and upper bound functions, i.e., $LB_a(\mathbf{q})$ and $UB_a(\mathbf{q})$, respectively, for the red pixel $\mathbf{q}$.

To further tighten the bound values, we propose the arbitrary-shaped lower and upper bound functions, namely $LB_a(\mathbf{q})$ and $UB_a(\mathbf{q})$, respectively. In $LB_a(\mathbf{q})$, we aim to aggregate the accumulated lengths of all grids that are fully covered by the dashed circle (cf. the orange grids in Figure 13a). In $UB_a(\mathbf{q})$, we only aggregate the accumulated lengths of those grids that partly intersect or are fully covered by the dashed circle (cf. the yellow grids in Figure 13b). Therefore, we can ensure that $\mathcal{L}(\mathbf{q}) \geq LB_a(\mathbf{q}) \geq LB_{\square}(\mathbf{q})$ and $\mathcal{L}(\mathbf{q}) \leq UB_a(\mathbf{q}) \leq UB_{\square}(\mathbf{q})$.

Compared with the square-shaped lower and upper bound functions, both $LB_a(\mathbf{q})$ and $UB_a(\mathbf{q})$ are not based on the rectangular-shaped regions. As such, we cannot use $O(1)$ time to obtain the aggregate values of all grids in the orange region and the yellow region (cf. Figures 13a and b, respectively). Instead, by regarding each horizontal stripe (or vertical stripe) as the rectangular region of grids, we can compute $LB_a(\mathbf{q})$ and $UB_a(\mathbf{q})$ (based on LARGE) in $O(\min(X, Y))$ time[2] in the worst case (cf. Theorem 4).

[2]If $X \leq Y$, we adopt the vertical stripe. Otherwise, we adopt the horizontal stripe. By using this strategy, we can achieve this worst-case time complexity.

Theorem 4. *Suppose that the indexing structure LARGE has been built for a line segment dataset $\mathbb{L} = \{l_1, l_2, ..., l_n\}$ with size n, computing the lower bound function $LB_a(\mathbf{q})$ and the upper bound function $UB_a(\mathbf{q})$ for each pixel takes $O(\min(X, Y))$ time.*

3.5 Filter and Refinement Framework

With these lower and upper bound functions (cf. Section 3.3 and Section 3.4), we can further utilize the filter and refinement framework [15, 24, 36] to boost the efficiency for generating LDV with an ϵ-relative error guarantee (cf. Definition 3). Note that if any lower and upper bound functions (namely $LB(\mathbf{q})$ and $UB(\mathbf{q})$, respectively) can fulfill the condition $UB(\mathbf{q}) \leq (1+\epsilon)LB(\mathbf{q})$, we can ensure that any value in between $LB(\mathbf{q})$ and $UB(\mathbf{q})$ can be a valid value for the result $R(\mathbf{q})$. Hence, we can let $R(\mathbf{q}) = \frac{LB(\mathbf{q})+UB(\mathbf{q})}{2}$. However, if $UB(\mathbf{q}) \leq (1+\epsilon)LB(\mathbf{q})$ does not hold, we need to adopt the sequential scan method (cf. Section 2.2) or the hierarchical indexing framework (cf. Section 2.3) as a refinement method to compute $\mathcal{L}(\mathbf{q})$. As a remark, since $LB_{\square}(\mathbf{q})$ and $UB_{\square}(\mathbf{q})$ are faster (but looser) compared with $LB_a(\mathbf{q})$ and $UB_a(\mathbf{q})$, respectively, our implementation first checks whether the condition is fulfilled by the $(LB_{\square}(\mathbf{q}), UB_{\square}(\mathbf{q}))$-pair, and then the $(LB_a(\mathbf{q}), UB_a(\mathbf{q}))$-pair (if the first pair is failure).

3.6 Tightness of Bound Functions

In this section, we further investigate the tightness of our bound functions (cf. Section 3.3 and Section 3.4). In order to achieve a tight lower (or upper) bound value for each pixel $\mathbf{q}$, the occupied area of a bound function (e.g., the green area in Figure 12a) should be close to the area of the search region (e.g., the area that is covered by the black dashed circle in Figure 12a), i.e., πb^2. Therefore, we use the ratio between these two areas to measure the tightness of bound functions.

Here, we let $A_{LB_{\square}(\mathbf{q})}$ and $A_{UB_{\square}(\mathbf{q})}$ be the occupied areas of the bound functions, $LB_{\square}(\mathbf{q})$ and $UB_{\square}(\mathbf{q})$, respectively, where (based on the $\sqrt{\delta_x^2+\delta_y^2} \leq 2b$ assumption[3] and Section II of the supplementary document [25])

$$A_{LB_{\square}(\mathbf{q})} = \left(2 \times \left\lfloor \frac{b - \frac{1}{2}\sqrt{\delta_x^2+\delta_y^2}}{\sqrt{\delta_x^2+\delta_y^2}} \right\rfloor + 1\right)^2 \delta_x \delta_y \quad (9)$$

$$A_{UB_{\square}(\mathbf{q})} = \left(2 \times \left\lceil \frac{b - \frac{1}{2}\min(\delta_x, \delta_y)}{\min(\delta_x, \delta_y)} \right\rceil + 1\right)^2 \delta_x \delta_y \quad (10)$$

As a remark, $A_{LB_{\square}(\mathbf{q})} \leq \pi b^2$ and $A_{UB_{\square}(\mathbf{q})} \geq \pi b^2$. We state in Theorem 5 that the bound functions can be tight if this ratio $\frac{\min(\delta_x,\delta_y)}{b}$ is small. This theorem indicates that $LB_{\square}(\mathbf{q})$ and $UB_{\square}(\mathbf{q})$ can be more useful for supporting small pixel sizes and large bandwidth values, which cannot be efficiently handled by state-of-the-art methods (cf. Section 2.3).

Theorem 5. *If $\sqrt{\delta_x^2+\delta_y^2} \leq 2b$ and $\frac{\min(\delta_x,\delta_y)}{b} \to 0$, $\frac{A_{LB_{\square}(\mathbf{q})}}{\pi b^2}$ and $\frac{A_{UB_{\square}(\mathbf{q})}}{\pi b^2}$ attain the maximum value and the minimum value, respectively.*

[3]Suppose that $\sqrt{\delta_x^2+\delta_y^2} > 2b$. The region of a pixel is larger than the search region, which is not meaningful for the visualization. Therefore, we have the assumption $\sqrt{\delta_x^2+\delta_y^2} \leq 2b$ in this paper.

Proof. We first consider the expression $\frac{A_{LB_\square(\mathbf{q})}}{\pi b^2}$ and let $\delta_y = c\delta_x$, where $c \geq 1$ (i.e., $\delta_x \leq \delta_y$). We have

$$\frac{A_{LB_\square(\mathbf{q})}}{\pi b^2} = \frac{c\left(2\left\lfloor \frac{b}{\sqrt{1+c^2}\delta_x} - \frac{1}{2}\right\rfloor + 1\right)^2 \delta_x^2}{\pi b^2} \geq \frac{c\left(2\left(\frac{b}{\sqrt{1+c^2}\delta_x} - \frac{3}{2}\right) + 1\right)^2 \delta_x^2}{\pi b^2} = \frac{4c}{\pi(1+c^2)} - \frac{8c}{\pi\sqrt{1+c^2}}\left(\frac{\delta_x}{b}\right) + 4c\left(\frac{\delta_x}{b}\right)^2$$

Hence, the lower bound of $\frac{A_{LB_\square(\mathbf{q})}}{\pi b^2}$ can be represented by the quadratic equation in terms of $\frac{\delta_x}{b}$. Since we have $\frac{\delta_x}{b} > 0$ (both $\delta_x > 0$ and $b > 0$) and $\frac{\delta_x}{b} \leq \frac{2}{\sqrt{1+c^2}}$ (based on $\sqrt{\delta_x^2 + \delta_y^2} \leq 2b$ and $\delta_y = c\delta_x$), we can conclude that the lower bound of $\frac{A_{LB_\square(\mathbf{q})}}{\pi b^2}$ attains the maximum value $\frac{4c}{\pi(1+c^2)}$ when $\frac{\min(\delta_x,\delta_y)}{b} = \frac{\delta_x}{b}$ tends to 0. Moreover, we also have

$$\frac{A_{LB_\square(\mathbf{q})}}{\pi b^2} \leq \frac{c\left(2\left(\frac{b}{\sqrt{1+c^2}\delta_x} - \frac{1}{2}\right) + 1\right)^2 \delta_x^2}{\pi b^2} = \frac{4c}{\pi(1+c^2)}$$

which indicates that the upper bound of $\frac{A_{LB_\square(\mathbf{q})}}{\pi b^2}$ is at most $\frac{4c}{\pi(1+c^2)}$ no matter which $\frac{\delta_x}{b}$ we use. As such, based on the squeeze theorem [8], we can conclude that $\frac{A_{LB_\square(\mathbf{q})}}{\pi b^2}$ can also attain the maximum value when $\frac{\min(\delta_x,\delta_y)}{b} = \frac{\delta_x}{b}$ tends to 0.

Then, we consider the expression $\frac{A_{UB_\square(\mathbf{q})}}{\pi b^2}$ and let $\delta_y = c\delta_x$, where $c \geq 1$ (i.e., $\delta_x \leq \delta_y$). We have

$$\frac{A_{UB_\square(\mathbf{q})}}{\pi b^2} = \frac{c\left(2\left\lfloor \frac{b}{\delta_x} - \frac{1}{2}\right\rfloor + 1\right)^2 \delta_x^2}{\pi b^2} \leq \frac{c\left(2\left(\frac{b}{\delta_x} + \frac{1}{2}\right) + 1\right)^2 \delta_x^2}{\pi b^2} = \frac{4c}{\pi}\left(1 + \frac{\delta_x}{b}\right)^2$$

Therefore, if $\frac{\delta_x}{b}$ tends to 0, the upper bound of $\frac{A_{UB_\square(\mathbf{q})}}{\pi b^2}$ attains the minimum value $\frac{4c}{\pi}$. Furthermore, we also have

$$\frac{A_{UB_\square(\mathbf{q})}}{\pi b^2} \geq \frac{c\left(2\left(\frac{b}{\delta_x} - \frac{1}{2}\right) + 1\right)^2 \delta_x^2}{\pi b^2} = \frac{4c}{\pi}$$

Therefore, the lower bound of $\frac{A_{UB_\square(\mathbf{q})}}{\pi b^2}$ is $\frac{4c}{\pi}$ regardless of which $\frac{\delta_x}{b}$ we choose. By the squeeze theorem [8], we can conclude that $\frac{A_{UB_\square(\mathbf{q})}}{\pi b^2}$ can also attain the minimum value if $\frac{\delta_x}{b}$ tends to 0.

By adopting the same concept, we can also have the same conclusion for the case $\delta_x = c\delta_y$, where $c \geq 1$ (i.e., $\delta_y \leq \delta_x$). □

Unlike $A_{LB_\square(\mathbf{q})}$ (cf. Equation 9) and $A_{UB_\square(\mathbf{q})}$ (cf. Equation 10), there is no closed-form expression for the occupied areas of $LB_a(\mathbf{q})$ and $UB_a(\mathbf{q})$, i.e., $A_{LB_a(\mathbf{q})}$ and $A_{UB_a(\mathbf{q})}$, respectively. However, suppose that the pixel size, $\delta_x \times \delta_y$, and the bandwidth parameter b are known, we can calculate the corresponding values of $A_{LB_a(\mathbf{q})}$ and $A_{UB_a(\mathbf{q})}$ (based on counting the total areas of those orange and yellow grids, respectively, in Figure 13). Therefore, in order to analyze the tightness of the arbitrary-shaped bound functions, we first set the parameters to be $\delta_x = \delta_y = \delta$ (i.e., adopt the commonly used square-shaped pixel) and then plot the values of $\frac{A_{LB_a(\mathbf{q})}}{\pi b^2}$ and $\frac{A_{UB_a(\mathbf{q})}}{\pi b^2}$ with respect to different ratios $\frac{\delta}{b}$ (cf. Figure 14). Observe that the tightness of these bound functions (i.e., $\frac{A_{LB_a(\mathbf{q})}}{\pi b^2}$ and $\frac{A_{UB_a(\mathbf{q})}}{\pi b^2}$) tend to 1 (i.e., the occupied areas of $LB_a(\mathbf{q})$ and $UB_a(\mathbf{q})$ are close to the search region) if we adopt the small ratio $\frac{\delta}{b}$, which indicates that the bound functions are tight for small pixel size (i.e., small $\delta = \delta_x = \delta_y$) or large bandwidth value (i.e., large b) in practice.

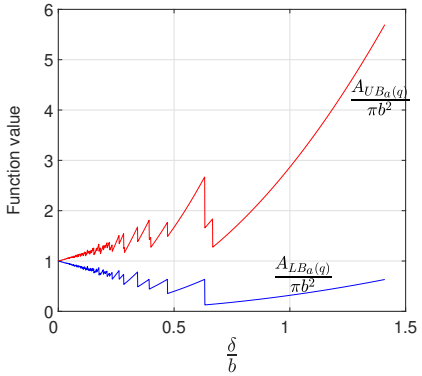


Figure 14: Tightness of the arbitrary-shaped bound functions, $LB_a(\mathbf{q})$ and $UB_a(\mathbf{q})$, varying the ratio $\frac{\delta}{b}$ (where $\delta_x = \delta_y = \delta$).

4 EXPERIMENTAL EVALUATION

In this section, we first discuss the experimental settings in Section 4.1. Then, we test the efficiency of all methods for generating LDV in Section 4.2. Next, we measure the space consumption of all methods in Section 4.3. After that, we compare the practical accuracy of exact and approximation methods in Section 4.4. Then, we discuss the effectiveness of bound functions in Section 4.5. Lastly, we conduct a case study in the Los Angeles bicycle mobility dataset for testing the visualization quality of the exact and approximation methods in Section 4.6. Some additional experiments can also be found in Section III of the supplementary document [25].

4.1 Experimental Settings

We adopt four large-scale trajectory datasets for testing, where we regard two consecutive trajectory points as a line segment for each dataset (which follows the same setting as [75]). Table 1 shows the details of all datasets.

Table 1: Datasets.

Dataset	n	Category	Ref.
Los Angeles	402,171	Bicycle mobility	[4]
San Francisco	402,602	Taxi mobility	[6]
Chicago	2,237,135	Taxi mobility	[7]
Beijing	14,263,241	Human mobility	[3]

In our experiments, we compare our method, LARGE, with four baseline methods, namely SCAN, $\text{SCAN}_{\text{line}}$, R-tree, and PMR quadtree, which are summarized in Table 2. SCAN is the sequential scan method for computing the line density function of each pixel (cf. Section 2.2). $\text{SCAN}_{\text{line}}$ is the variant of SCAN, which first finds all pixels that are within the bandwidth b from each line segment and then updates the density values for those pixels. Both R-tree [54][4] and PMR quadtree [45–47] are the representative methods for indexing line segment data so that they can also be extended

[4]Since all line segments are available in advance (i.e., static data), we adopt the advanced bulk loading technique [54] to construct a compact R-tree for each dataset.

to compute the line density function (cf. Section 2.3). We implemented all these methods with C++ and conducted experiments on an Intel i7 2.9GHz PC with 32GB memory. In this paper, we use the response time (sec) and the memory space (MB) to measure the time efficiency and the space efficiency of each method, respectively, and only report the response time that is smaller than one day (i.e., 86,400 sec). Moreover, we also adopt the mean squared error (MSE) as a metric, which indicates the average line density value deviation of each pixel, to measure the practical accuracy of our approximation method, LARGE, compared with any exact method (e.g., R-tree). As a remark, we use R-tree as the refinement method of LARGE (cf. Section 3.5) in our experiment.

Table 2: All methods.

Method	SCAN	$\text{SCAN}_{\text{line}}$	R-tree	PMR quadtree	LARGE
Ref.	[1, 5]	[1, 5]	[54]	[45–47]	Section 3

4.2 Efficiency of All Methods

In this section, we conduct the following four experiments to measure the response time of each method for generating LDV (cf. Definition 2), which are (1) varying the resolution size, (2) varying the bandwidth parameter b, (3) varying the dataset size, and (4) varying the relative error ϵ (cf. Definition 3). By default, we set the resolution size, the bandwidth parameter, and the relative error to be 320×240, 1000m, and 0.1, respectively.

Varying the resolution size. In this experiment, we test the efficiency of all methods with respect to different resolution sizes, namely 320×240, 480×360, 720×540, and 1080×810. Figure 15 shows the results of all methods. Recall that the larger the resolution size (i.e., δ_x and δ_y are smaller), the tighter the lower and upper bound functions of LARGE (cf. Theorem 5 and Figure 14), which leads to the higher filtering power for generating LDV (cf. Section 3.5). Therefore, the response time of LARGE is not very sensitive to the resolution size compared with that of existing methods (which are proportional to the resolution size). Since the time complexity of constructing LARGE (cf. Theorem 1) and computing the bound functions (cf. Theorem 3 and Theorem 4) are also small, our method can achieve speedups of 4.87x to 288.25x compared with the existing methods.

Varying the bandwidth parameter b. Here, we test the efficiency of all methods with respect to different bandwidth parameters b, which are 500m, 1000m, 1500m, 2000m, and 2500m. Observe from Figure 16 that the response time of SCAN is not sensitive to the bandwidth parameter b since this method does not adopt any filtering technique. Moreover, all tree-based indexing methods (cf. Section 2.3), including R-tree and PMR quadtree, take long response time if the bandwidth parameter b is large. The main reason is that the search range of each pixel (e.g., the black dashed circle in Figure 5) is large, which can cover more line segments and nodes in a tree index, given a large bandwidth value b. Similarly, the variant of the sequential scan method, $\text{SCAN}_{\text{line}}$, also takes long response time with the large bandwidth parameter b since more pixels should be accessed for each line segment. In contrast, since our lower and upper bound functions are tight with a large bandwidth value b (cf. Theorem 5 and Figure 14), the response time of our method, LARGE, is not proportional to the bandwidth value b. As a remark, LARGE can even achieve small response time with large bandwidth values for the Los Angeles, San Francisco, and Chicago datasets.[5] Due to the lower time complexity of constructing LARGE (cf. Theorem 1) and computing bound functions (cf. Theorem 3 and Theorem 4), our method can achieve speedups of 2.35x to 291.8x for generating LDV compared with all existing methods.

Varying the dataset size. We proceed to investigate how the dataset size affects the response time of each method. To conduct this experiment, we first sample those line segments in each dataset with four sampling ratios, which are 25%, 50%, 75%, and 100% (no sampling). Then, we measure the response time of all methods with respect to each reduced dataset. Since all methods need to access more line segments given a larger sampling ratio, the response time of all methods is proportional to this parameter (cf. Figure 17). Note that our method, LARGE, achieves speedups of 4.72x to 55.93x over the state-of-the-art methods for generating LDV.

Varying the relative error ϵ. In this experiment, we further test how the relative error ϵ affects the response time of each method for supporting LDV (by varying ϵ from 0.05 to 0.2). Figure 18 shows the results of all methods. Since SCAN, $\text{SCAN}_{\text{line}}$, R-tree, and PMR quadtree are the exact methods for generating LDV, the response time of all these methods is not sensitive to this parameter. Note that our methods can achieve speedups of 3.13x to 105.84x compared with the state-of-the-art methods no matter which ϵ we choose.

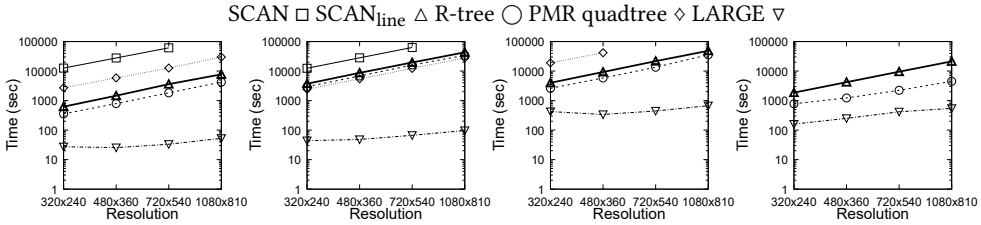
4.3 Space Consumption of All Methods

In this section, we further investigate the space consumption of all methods for generating LDV by conducting the following two experiments, which are (1) varying the dataset size and (2) varying the bandwidth parameter b. By default, we set the resolution size, the bandwidth parameter, and the relative error to be 320×240, 1000m, and 0.1, respectively.

Varying the dataset size. Here, we examine how the dataset size affects the memory space consumption of each method. To conduct this experiment, we first sample each dataset using four sampling ratios, which are 25%, 50%, 75%, and 100% (no sampling), and then measure the memory space consumption of each method. Figure 19 shows the results of all methods. Observe that the larger the dataset size, the higher the memory space consumption of each method. The main reason is that all methods need to access more data points with larger dataset sizes. Note that R-tree, PMR quadtree, and LARGE (using R-tree as the refinement method) need to construct tree-based indexing structures, which consume larger memory space compared with SCAN and $\text{SCAN}_{\text{line}}$. Since our method, LARGE, still retains the small space complexity (cf. Theorem 2), LARGE does not incur significant memory space overhead compared with the state-of-the-art methods, R-tree and PMR quadtree.

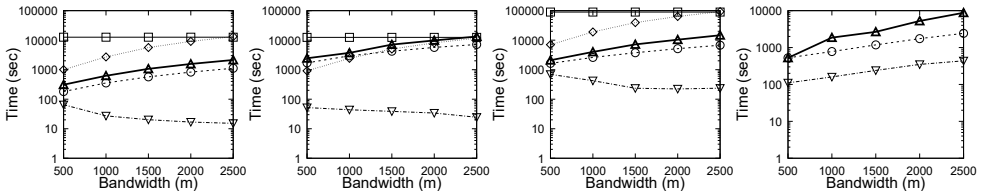
Varying the bandwidth parameter b. We proceed to investigate how the bandwidth parameter b affects the memory space consumption of each method, by choosing five bandwidth values, namely 500m, 1000m, 1500m, 2000m, and 2500m. Figure 20 shows the results of all methods. Since the sequential scan methods, SCAN and

[5] Theoretically, LARGE needs to access more line segments in the refinement phase for each pixel $\mathbf{q}$ (i.e., $UB(\mathbf{q}) \leq (1+\epsilon)LB(\mathbf{q})$ does not hold in the filter phase.) with a large bandwidth value b since we adopt the R-tree in this phase. However, with the high filtering power of bound functions for a large bandwidth value b, there must be less pixels in the refinement phase. Therefore, the response time depends on the final trade off between these two factors (e.g., smaller response time for the Los Angeles, San Francisco, and Chicago datasets and higher response time for the Beijing dataset with the larger bandwidth value b).



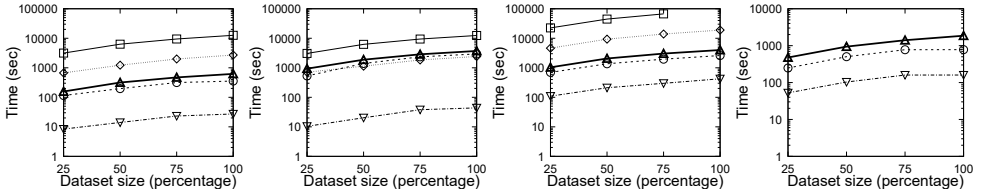
(a) Los Angeles (b) San Francisco (c) Chicago (d) Beijing

Figure 15: Response time for generating LDV, varying the resolution size.



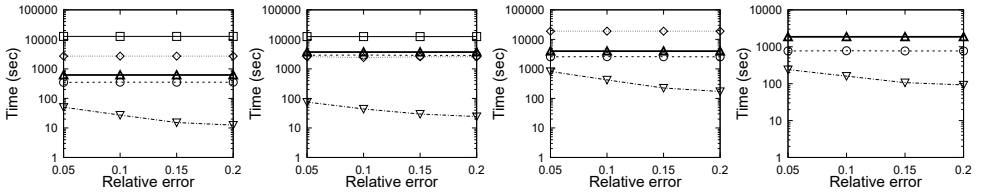
(a) Los Angeles (b) San Francisco (c) Chicago (d) Beijing

Figure 16: Response time for generating LDV, varying the bandwidth parameter b.



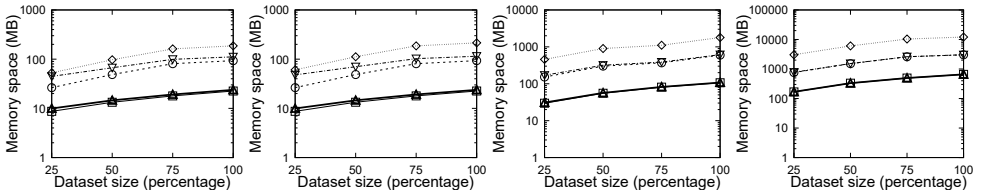
(a) Los Angeles (b) San Francisco (c) Chicago (d) Beijing

Figure 17: Response time for generating LDV, varying the dataset size.



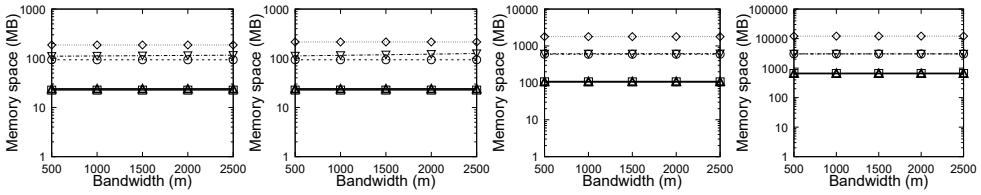
(a) Los Angeles (b) San Francisco (c) Chicago (d) Beijing

Figure 18: Response time for generating LDV, varying the relative error ϵ.



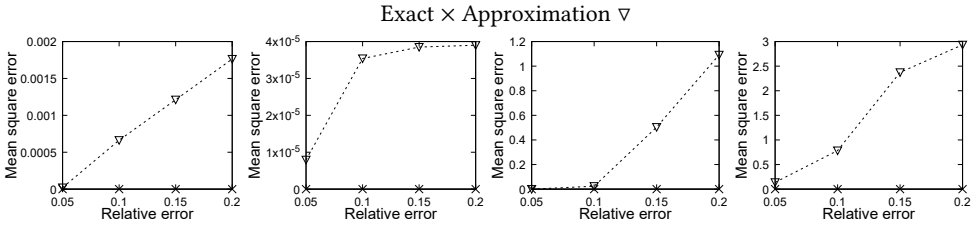
(a) Los Angeles (b) San Francisco (c) Chicago (d) Beijing

Figure 19: Memory space consumption (MB) for generating LDV, varying the dataset size.



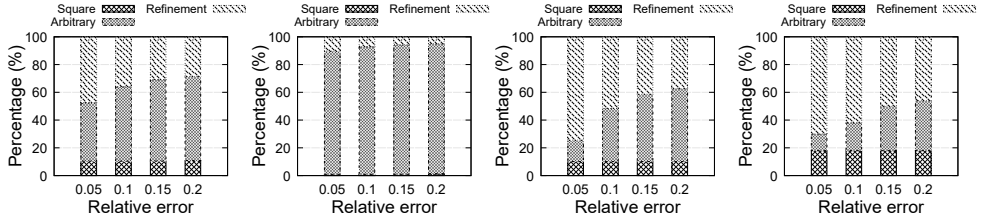
(a) Los Angeles (b) San Francisco (c) Chicago (d) Beijing

Figure 20: Memory space consumption (MB) for generating LDV, varying the bandwidth parameter b.



(a) Los Angeles (b) San Francisco (c) Chicago (d) Beijing

Figure 21: Accuracy (mean squared error) for generating LDV, varying the relative error ϵ.



(a) Los Angeles (b) San Francisco (c) Chicago (d) Beijing

Figure 22: Percentages of pixels that (1) can be handled by the square-shaped bound functions (Square), (2) can be handled by the arbitrary-shaped bound functions (Arbitrary) and (3) cannot be handled by our bound functions (Refinement).

$\text{SCAN}_{\text{line}}$, and the construction of all tree-based indexing structures, R-tree and PMR quadtree, are independent to the bandwidth parameter, all existing methods, SCAN, $\text{SCAN}_{\text{line}}$, R-tree, and PMR quadtree, are not sensitive to this parameter. Recall that the space consumption of our method, LARGE, should be proportional to the bandwidth parameter b, due to a larger extended region in Figure 7. However, the space consumption of our method, LARGE, is still not sensitive to this parameter since the number of line segments n is much larger than the size of the extended region of each dataset (which is the main bottleneck of the space complexity $O(XY + n)$ in Theorem 2).

4.4 Accuracy of All Methods

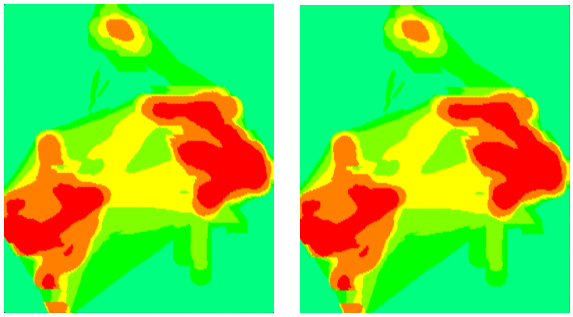
In this section, we proceed to investigate how the relative error ϵ affects the practical accuracy (i.e., the mean squared error) of exact and approximation methods, by using four values of ϵ, which are 0.05, 0.1, 0.15, and 0.2. Observe from Figure 21 that the larger the relative error ϵ, the larger the mean squared error. Note that our approximation method, i.e., LARGE, retains the small mean squared error (which ranges from 7.95×10^{-6} to 2.93) compared with any exact method (with zero error). Therefore, the results indicate that our approximation method can achieve accurate performance for generating LDV.

4.5 Effectiveness of Bound Functions

In this section, we further investigate the effectiveness (i.e., the filtering power) of our bound functions in the filter and refinement framework (cf. Section 3.5) by choosing four values of ϵ, which are 0.05, 0.1, 0.15, and 0.2. Figure 22 shows the percentages of pixels that (1) can be handled by the square-shaped bound functions, (2) can be handled by the arbitrary-shaped bound functions and (3) cannot be handled by our bound functions (i.e., need to undergo the refinement stage.). Observe that the larger the relative error ϵ, the higher the percentage of pixels (i.e., filtering power) that can be handled by our bound functions (especially for arbitrary-shaped bound functions). In addition, our bound functions can handle 25.55% to 95.02% pixels in these datasets. Therefore, LARGE can significantly improve the efficiency for generating LDV compared with exact methods.

4.6 Case Study

In this section, we conduct a case study for testing the visualization quality of the exact (i.e., SCAN, $\text{SCAN}_{\text{line}}$, R-tree, and PMR quadtree) and approximation (i.e., LARGE) methods using the Los Angeles bicycle mobility dataset. By default, we set the resolution size, the bandwidth parameter b, and the relative error ϵ to be 320×240, 1000m, and 0.1, respectively, for conducting this case study. Figure 23 shows the LDVs of using the exact and approximation methods. Observe that these two plots also reveal that the bicycle flows are mainly between the west part of Los Angeles (i.e., Santa Monica) and the east part of Los Angeles (i.e., Downtown Los Angeles). In addition, we also note that our approximation method, i.e., LARGE, (cf. Figure 23b) provides the similar visualization compared with the one of the exact method (cf. Figure 23a), which indicates that LARGE achieves significant efficiency improvement, without degrading the visualization quality in practice.



(a) Exact (b) Approximation

Figure 23: Generate LDVs using the exact (i.e., SCAN, $\text{SCAN}_{\text{line}}$, R-tree, and PMR quadtree) and approximation (i.e., LARGE) methods.

5 RELATED WORK

In this section, we review five camps of research studies, namely (1) efficient algorithms for spatial join and line-segment-based queries, (2) efficient algorithms for kernel density visualization, (3) efficient algorithms for moving object queries, (4) tree-based indexing structures, and (5) grid-based indexing structures, which are mostly related to this work.

Efficient algorithms for spatial join and line-segment-based queries. In the first camp, many researchers in the database [32, 37, 42, 43, 45–49, 51, 68, 79, 96], computational geometry [13, 27, 29, 65], and theoretical computer science [59] communities have developed efficient algorithms for supporting different types of spatial join queries. Among most of these queries, line-segment-based queries [13, 27, 29, 42, 45–49, 51, 59, 65] (e.g., finding the nearest line segment from a query point in a line segment database [45, 46]) are one of the representative classes. However, unlike our problem (cf. Definition 2), all these research studies mainly focus on relatively simple line-segment-based queries, which do not need to compute the more complex density function (cf. Equation 4). Therefore, most of these methods (and their theoretical results) cannot be easily extended for solving our problem.

Efficient algorithms for kernel density visualization. In the second camp, many researchers [15–21, 23, 36, 71–73, 99–101] have proposed efficient algorithms for kernel density visualization (KDV), which is another important density visualization tool that is based on location data points. However, since LDV (cf. Definition 2) is based on line segments (rather than location data points), all these efficient algorithms (and their theoretical results) in KDV cannot be extended for generating LDV.

Efficient algorithms for moving object queries. In the third camp, a plethora of research studies [52, 53, 57, 67, 70, 87, 88, 91] have been proposed for solving different types of queries that are related to moving objects (or trajectories), which can also be regarded as a set of line segments in some research studies (e.g., [67, 70, 87, 88]). However, unlike our work, none of these research studies focuses on computing the line density function (cf. Equation 4), which cannot be used for improving the efficiency of generating LDV.

Tree-based indexing structures. In the fourth camp, many tree-based indexing structures [78, 97] have been proposed to boost the efficiency of spatial/multidimensional similarity search queries. To the best of our knowledge, using the indexing framework with the lower bound that is based on minimum bounding rectangles (cf. Section 2.3) is also the state-of-the-art approach for solving our problem. However, since generating LDV is based on line segments, many point-based indexing structures (e.g., kd-tree [12, 97], ball-tree [63, 78], and metric-tree [28, 78]) cannot be easily extended for solving this problem. Among most of these indexing structures, rectangle-based indexing structures (e.g., R-tree [40, 54], R^+-tree [33], and R^*-tree [10]) and PMR quadtree [45–47] are the most representative ones for generating LDV. However, all these solutions cannot efficiently handle line segments that are close to a pixel (cf. Figure 6), which provide inferior efficiency performance compared with LARGE (cf. Section 4.2).

Grid-based indexing structures. In the fifth camp, many researchers [14, 30, 34, 35, 53, 55, 57, 76, 81, 82, 88, 89, 98] have adopted grid-based indexing structures to handle different types of query processing problems (e.g., k-nearest neighbor search and range search problems). However, most of these research studies [14, 34, 35, 55, 57, 81, 82, 89, 98] mainly focus on location point data rather than line-segment data, which cannot be extended for solving the LDV problem (cf. Definition 2). Although some research studies [30, 53, 76, 88, 98] also build grid-based indexing structures for line-segment data in order to support various types of spatial queries (e.g., k-nearest neighbor search), all of them do not consider the complex line density function (cf. Equation 4). Therefore, these research studies cannot be extended for solving the LDV problem.

6 CONCLUSION

In this paper, we study line density visualization (LDV), which has been widely used in different applications, including mobility analysis, traffic flow analysis, and crime pattern analysis, and has been extensively supported by many commonly used software platforms, including QGIS and ArcGIS. However, LDV is a computationally expensive operation, which cannot be scalable to handle large-scale line segment datasets or high resolution sizes. To overcome the efficiency issues of this operation, we first propose the indexing structure, called length-aggregation-based grid structure (LARGE). Then, based on LARGE, we further develop two types of efficient bound functions, namely (1) square-shaped lower and upper bound functions and (2) arbitrary-shaped lower and upper bound functions, which can filter a large portion of unnecessary computations for approximately computing LDV with an ϵ-relative error guarantee. Theoretically, we also show that our bound functions can be tight if the ratio between the pixel size and the bandwidth value is small, indicating that our solution, LARGE, can be more scalable to compute LDV with high resolution sizes (i.e., small pixel sizes) and large bandwidth values compared with the existing solutions (e.g., R-tree and PMR quadtree). In practice, our experiment results on four large-scale line segment datasets also show that LARGE yields up to 291.8x speedups over the state-of-the-art exact solutions, without degrading the visualization quality and incurring the significant space overhead. Furthermore, we have also developed the new plugin [2] (based on LARGE) for QGIS users to efficiently generate LDV in a line segment dataset.

In the future, we plan to investigate the parallel, distributed, and hardware-based approaches to further boost the efficiency of LARGE in order to achieve real-time performance for generating LDV. Moreover, we will investigate how to develop another fast and accurate line density visualization tool for domain experts. Furthermore, we will develop efficient solutions for other types of geospatial analysis tools [22], including Moran's I [58], Geary's c [58], inverse distance weighting [56], and Kriging [92].

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China 2023YFC3321300, the Natural Science Foundation of China under grants 62202401, 62372308, and 62472116, the Natural Science Foundation of Guangdong Province of China under grants 2023A1515011619 and 2023A1515030273, the Science and Technology Development Fund Macau SAR (0003/2023/RIC, 0052/2023/RIA1, 0031/2022/A, 001/2024/SKL for SKL-IOTSC), the Research Grant of University of Macau (MYRG2022-00252-FST), Shenzhen-Hong Kong-Macau Science and Technology Program Category C (SGDX20230821095159012), and Wuyi University Hong Kong and Macau joint Research Fund (2021WGALH14).

REFERENCES

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

[2] [n. d.]. Fast Line Density Analysis. https://plugins.qgis.org/plugins/fast_line_density_analysis/.

[3] [n. d.]. GeoLife GPS Trajectories. https://www.microsoft.com/en-us/download/details.aspx?id=52367.

[4] [n. d.]. Los Angeles Bike Trip Data. https://bikeshare.metro.net/about/data/.

[5] [n. d.]. QGIS. https://docs.qgis.org/3.28/en/docs/user_manual/processing_algs/qgis/interpolation.html#line-density.

[6] [n. d.]. San Francisco taxi trajectories. https://figshare.com/articles/dataset/San_Francisco_taxi_trajectories/12302243.

[7] [n. d.]. Taxi Trips 2020. https://data.cityofchicago.org/Transportation/Taxi-Trips-2020/r2u4-wwk3.

[8] Robert A Adams and Christopher Essex. 2018. *Calculus: a complete course*. Pearson.

[9] Michael J Allen, Thomas R Allen, Christopher Davis, and George McLeod. 2021. Exploring spatial patterns of virginia tornadoes using kernel density and space-time cube analysis (1960–2019). *ISPRS International Journal of Geo-Information* 10, 5 (2021), 310.

[10] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*. 322–331.

[11] J. Adam Beeco, Jeffrey C. Hallo, William 'Rockie' English, and Gary W. Giumetti. 2013. The importance of spatial nested data in understanding the relationship between visitor use and landscape impacts. *Applied Geography* 45 (2013), 147–157. https://doi.org/10.1016/j.apgeog.2013.09.001

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

[13] Sergei Bespamyatnikh and Jack Snoeyink. 2000. Queries with segments in Voronoi diagrams. *Comput. Geom.* 16, 1 (2000), 23–33. https://doi.org/10.1016/S0925-7721(99)00055-3

[14] Bin Cao, Chenyu Hou, Suifei Li, Jing Fan, Jianwei Yin, Baihua Zheng, and Jie Bao. 2018. SIMkNN: A Scalable Method for in-Memory kNN Search over Moving Objects in Road Networks. *IEEE Trans. Knowl. Data Eng.* 30, 10 (2018), 1957–1970. https://doi.org/10.1109/TKDE.2018.2808971

[15] 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

[16] 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.

[17] 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. *Proc. VLDB Endow.* 14, 12 (2021), 2655–2658.

[18] Tsz Nam Chan, Pak Lon Ip, Kaiyan Zhao, Leong Hou U, Byron Choi, and Jianliang Xu. 2022. LIBKDV: A Versatile Kernel Density Visualization Library for Geospatial Analytics. *Proc. VLDB Endow.* 15, 12 (2022), 3606–3609. https://www.vldb.org/pvldb/vol15/p3606-chan.pdf

[19] Tsz Nam Chan, Leong Hou U, Reynold Cheng, Man Lung Yiu, and Shivansh Mittal. 2022. Efficient Algorithms for Kernel Aggregation Queries. *IEEE Trans. Knowl. Data Eng.* 34, 6 (2022), 2726–2739. https://doi.org/10.1109/TKDE.2020.3018376

[20] Tsz Nam Chan, Leong Hou U, Byron Choi, and Jianliang Xu. 2022. SLAM: Efficient Sweep Line Algorithms for Kernel Density Visualization. In *SIGMOD*. ACM, 2120–2134. https://doi.org/10.1145/3514221.3517823

[21] Tsz Nam Chan, Leong Hou U, Byron Choi, Jianliang Xu, and Reynold Cheng. 2023. Kernel Density Visualization for Big Geospatial Data: Algorithms and Applications. In *MDM*. IEEE, 231–234. https://doi.org/10.1109/MDM58254.2023.00046

[22] Tsz Nam Chan, Leong Hou U, Byron Choi, Jianliang Xu, and Reynold Cheng. 2023. Large-scale Geospatial Analytics: Problems, Challenges, and Opportunities. In *SIGMOD Companion*. ACM, 21–29. https://doi.org/10.1145/3555041.3589401

[23] 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

[24] Tsz Nam Chan, Man Lung Yiu, and Leong Hou U. 2021. The Power of Bounds: Answering Approximate Earth Mover's Distance with Parametric Bounds. *IEEE Trans. Knowl. Data Eng.* 33, 2 (2021), 768–781. https://doi.org/10.1109/TKDE.2019.2931969

[25] Tsz Nam Chan, Bojian Zhu, Dingming Wu, Yun Peng, and Leong Hou U. 2024. Supplementary Document for "LARGE: A Length-Aggregation-based Grid Structure for Line Density Visualization". https://github.com/edisonchan2013928/LARGE_supplementary_document/blob/main/LARGE_supplementary.pdf.

[26] Changjie Chen, Jasmeet Judge, and David Hulse. 2022. PyLUSAT: An open-source Python toolkit for GIS-based land use suitability analysis. *Environmental Modelling & Software* 151 (2022), 105362. https://doi.org/10.1016/j.envsoft.2022.105362

[27] Danny Ziyi Chen and Haitao Wang. 2015. Weak visibility queries of line segments in simple polygons. *Comput. Geom.* 48, 6 (2015), 443–452. https://doi.org/10.1016/j.comgeo.2015.02.001

[28] Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB*. 426–435.

[29] Ovidiu Daescu and Robert Serfling. 2005. Extremal point queries with lines and line segments and related problems. *Comput. Geom.* 32, 3 (2005), 223–237. https://doi.org/10.1016/j.comgeo.2005.03.002

[30] Jian Dai, Bin Yang, Chenjuan Guo, and Zhiming Ding. 2015. Personalized route recommendation using big trajectory data. In *ICDE*. IEEE, 543–554. https://doi.org/10.1109/ICDE.2015.7113313

[31] Urška Demšar, Kevin Buchin, Francesca Cagnacci, Kamran Safi, Bettina Speckmann, Nico Van de Weghe, Daniel Weiskopf, and Robert Weibel. 2015. Analysis and visualisation of movement: an interdisciplinary review. *Movement ecology* 3, 1 (2015), 1–24.

[32] Ahmed Eldawy and Mohamed F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *ICDE*. IEEE, 1352–1363. https://doi.org/10.1109/ICDE.2015.7113382

[33] Christos Faloutsos, Timos K. Sellis, and Nick Roussopoulos. 1987. Analysis of Object Oriented Spatial Access Methods. In *SIGMOD*. ACM, 426–439. https://doi.org/10.1145/38713.38758

[34] Ziquan Fang, Lu Chen, Yunjun Gao, Lu Pan, and Christian S. Jensen. 2021. Dragoon: a hybrid and efficient big trajectory management system for offline and online analytics. *VLDB J.* 30, 2 (2021), 287–310. https://doi.org/10.1007/s00778-021-00652-x

[35] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. 2000. Vector Approximation based Indexing for Non-uniform High Dimensional Data Sets. In *CIKM*. ACM, 202–209. https://doi.org/10.1145/354756.354820

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

[37] Thanasis Georgiadis and Nikos Mamoulis. 2023. Raster Intervals: An Approximation Technique for Polygon Intersection Joins. *Proc. ACM Manag. Data* 1, 1 (2023), 36:1–36:18. https://doi.org/10.1145/3588716

[38] Anita Graser. 2021. An exploratory data analysis protocol for identifying problems in continuous movement data. *Journal of Location Based Services* 15, 2 (2021), 89–117.

[39] Richard L Graw and Bret A Anderson. 2022. Strategies to reduce wildfire smoke in frequently impacted communities in south-western Oregon. *International Journal of Wildland Fire* 31, 12 (2022), 1155–1166.

[40] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. 47–57.

[41] Kazumasa Hanaoka, Tomoki Nakaya, Keiji Yano, and Shigeru Inoue. 2014. Network-based spatial interpolation of commuting trajectories: Application of a university commuting management project in Kyoto, Japan. *Journal of Transport Geography* 34 (2014), 274–281.

[42] Gísli R. Hjaltason and Hanan Samet. 2002. Speeding up construction of PMR quadtree-based spatial indexes. *VLDB J.* 11, 2 (2002), 109–137. https://doi.org/10.1007/s00778-002-0067-8

[43] Gísli R. Hjaltason, Hanan Samet, and Yoram J. Sussmann. 1997. Speeding up Bulk-Loading of Quadtrees. In *GIS*. ACM, 50–53. https://doi.org/10.1145/267825.267839

[44] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. 1997. Range Queries in OLAP Data Cubes. In *SIGMOD*. 73–88.

[45] Erik G. Hoel and Hanan Samet. 1991. Efficient Processing of Spatial Queries in Line Segment Databases. In *SSD*. Springer, 237–256. https://doi.org/10.1007/3-540-54414-3_41

[46] Erik G. Hoel and Hanan Samet. 1992. A Qualitative Comparison Study of Data Structures for Large Line Segment Databases. In *SIGMOD*. ACM, 205–214. https://doi.org/10.1145/130283.130316

[47] Erik G. Hoel and Hanan Samet. 1995. Benchmarking Spatial Join Operations with Spatial Output. In *VLDB*. Morgan Kaufmann, 606–618. http://www.vldb.org/conf/1995/P606.PDF

[48] Qiang Huang, Yifan Lei, and Anthony K. H. Tung. 2021. Point-to-Hyperplane Nearest Neighbor Search Beyond the Unit Hypersphere. In *SIGMOD*. ACM, 777–789. https://doi.org/10.1145/3448016.3457240

[49] Qiang Huang and Anthony K. H. Tung. 2023. Lightweight-Yet-Efficient: Revitalizing Ball-Tree for Point-to-Hyperplane Nearest Neighbor Search. In *ICDE*. IEEE, 436–449. https://doi.org/10.1109/ICDE55515.2023.00040

[50] Tabassum Zarina Insaf, Temilayo Adeyeye, Catherine Adler, Victoria Wagner, Anisa Proj, Susan McCauley, and Jacqueline Matson. 2022. Road traffic density and recurrent asthma emergency department visits among Medicaid enrollees in New York State 2005–2015. *Environmental Health* 21, 1 (2022), 73.

[51] H. V. Jagadish. 1990. On Indexing Line Segments. In *VLDB*, Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek (Eds.). Morgan Kaufmann, 614–625.

http://www.vldb.org/conf/1990/P614.PDF
[52] Christian S. Jensen, Dan Lin, and Beng Chin Ooi. 2004. Query and Update Efficient B+-Tree Based Indexing of Moving Objects. In *VLDB*. Morgan Kaufmann, 768–779. https://doi.org/10.1016/B978-012088469-8.50068-1
[53] Fengmei Jin, Wen Hua, Boyu Ruan, and Xiaofang Zhou. 2022. Frequency-based Randomization for Guaranteeing Differential Privacy in Spatial Trajectories. In *ICDE*. IEEE, 1727–1739. https://doi.org/10.1109/ICDE53745.2022.00175
[54] Scott T. Leutenegger, Mario Alberto López, and J. M. Edgington. 1997. STR: A Simple and Efficient Algorithm for R-Tree Packing. In *ICDE*. 497–506. https://doi.org/10.1109/ICDE.1997.582015
[55] Haoda Li, Qiyang Song, Guoliang Li, Qi Li, and Rengui Wang. 2022. GPSC: A Grid-Based Privacy-Reserving Framework for Online Spatial Crowdsourcing. *IEEE Trans. Knowl. Data Eng.* 34, 11 (2022), 5378–5390. https://doi.org/10.1109/TKDE.2021.3055623
[56] Jin Li and Andrew D. Heap. 2014. Spatial interpolation methods applied in the environmental sciences: A review. *Environmental Modelling & Software* 53 (2014), 173–189. https://doi.org/10.1016/j.envsoft.2013.12.008
[57] Jiajia Li, Cancan Ni, Dan He, Lei Li, Xiufeng Xia, and Xiaofang Zhou. 2023. Efficient kNN query for moving objects on time-dependent road networks. *VLDB J.* 32, 3 (2023), 575–594. https://doi.org/10.1007/s00778-022-00758-w
[58] Jie Lin. 2023. Comparison of Moran's I and Geary's c in Multivariate Spatial Pattern Analysis. *Geographical Analysis* 55, 4 (2023), 685–702. https://doi.org/10.1111/gean.12355 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/gean.12355
[59] Michael Lindenbaum, Hanan Samet, and Gísli R. Hjaltason. 2005. A Probabilistic Analysis of Trie-Based Sorting of Large Collections of Line Segments in Spatial Databases. *SIAM J. Comput.* 35, 1 (2005), 22–58. https://doi.org/10.1137/S0097539700368527
[60] Zuopeng Ma, Chenggu Li, Pingyu Zhang, Jing Zhang, Daqian Liu, and Mingke Xie. 2023. The impact of transportation on commercial activities: The stories of various transport routes in Changchun, China. *Cities* 132 (2023), 103979. https://doi.org/10.1016/j.cities.2022.103979
[61] Harvey J. Miller, Somayeh Dodge, Jennifer A. Miller, and Gil Bohrer. 2019. Towards an integrated science of movement: converging research on animal movement ecology and human mobility science. *Int. J. Geogr. Inf. Sci.* 33, 5 (2019), 855–876. https://doi.org/10.1080/13658816.2018.1564317
[62] Masoud Minaei. 2020. Evolution, density and completeness of OpenStreetMap road networks in developing countries: the case of Iran. *Applied geography* 119 (2020), 102246.
[63] Andrew W. Moore. 2000. The Anchors Hierarchy: Using the Triangle Inequality to Survive High Dimensional Data. In *UAI*. 397–405.
[64] Fazzami Othman, Zaharah M. Yusoff, and Siti Aekbal Salleh. 2020. Assessing the visualization of space and traffic volume using GIS-based processing and visibility parameters of space syntax. *Geo-spatial Information Science* 23, 3 (2020), 209–221. https://doi.org/10.1080/10095020.2020.1811781
[65] Mark H. Overmars. 1985. Range searching in a set of line segments. In *SCG*. ACM, 177–185. https://doi.org/10.1145/323233.323257
[66] Cesar Palomo, Zhan Guo, Cláudio T. Silva, and Juliana Freire. 2016. Visually Exploring Transportation Schedules. *IEEE Trans. Vis. Comput. Graph.* 22, 1 (2016), 170–179. https://doi.org/10.1109/TVCG.2015.2467592
[67] Costas Panagiotakis, Nikos Pelekis, Ioannis Kopanakis, Emmanuel Ramasso, and Yannis Theodoridis. 2012. Segmentation and Sampling of Moving Object Trajectories Based on Representativeness. *IEEE Trans. Knowl. Data Eng.* 24, 7 (2012), 1328–1343. https://doi.org/10.1109/TKDE.2011.39
[68] Jignesh M. Patel and David J. DeWitt. 1996. Partition Based Spatial-Merge Join. In *SIGMOD*. ACM, 259–270. https://doi.org/10.1145/233269.233338
[69] Parsa Pezeshknejad, Saeed Monajem, and Hamid Mozafari. 2020. Evaluating sustainability and land use integration of BRT stations via extended node place model, an application on BRT stations of Tehran. *Journal of Transport Geography* 82 (2020), 102626. https://doi.org/10.1016/j.jtrangeo.2019.102626
[70] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. 2000. Novel Approaches to the Indexing of Moving Object Trajectories. In *VLDB*. Morgan Kaufmann, 395–406. http://www.vldb.org/conf/2000/P395.pdf
[71] Jeff M. Phillips. 2013. ϵ-Samples for Kernels. In *SODA*. 1622–1632. https://doi.org/10.1137/1.9781611973105.116
[72] 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
[73] 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
[74] Steven D. Prager and R. Paul Wiegand. 2014. Modeling Use of Space from Social Media Data Using a Biased Random Walker. *Trans. GIS* 18, 6 (2014), 817–833. https://doi.org/10.1111/tgis.12069
[75] Alasdair Rae. 2009. From spatial interaction data to spatial interaction information? Geovisualisation and spatial structures of migration from the 2001 UK census. *Computers, Environment and Urban Systems* 33, 3 (2009), 161–178.
[76] Dimitris Sacharidis, Kostas Patroumpas, Manolis Terrovitis, Verena Kantere, Michalis Potamias, Kyriakos Mouratidis, and Timos K. Sellis. 2008. On-line discovery of hot motion paths. In *EDBT*, Vol. 261. ACM, 392–403. https://doi.org/10.1145/1353343.1353392
[77] Günther Sagl, Martin Loidl, and Euro Beinat. 2012. A visual analytics approach for extracting spatio-temporal urban mobility information from mobile network traffic. *ISPRS International Journal of Geo-Information* 1, 3 (2012), 256–271.
[78] H. Samet. 2006. *Foundations of Multidimensional and Metric Data Structures*.
[79] Hanan Samet. 2013. Sorting in Space: Multidimensional, spatial, and metric data structures for applications in spatial databases, geographic information systems (GIS), and location-based services. In *ICDE*. IEEE, 1254–1257. https://doi.org/10.1109/ICDE.2013.6544917
[80] D. W. Scott. 1992. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley. https://books.google.com.hk/books?id=7crCUS_F2ocC
[81] Shuo Shang, Lisi Chen, Zhewei Wei, Christian S. Jensen, Kai Zheng, and Panos Kalnis. 2017. Trajectory Similarity Join in Spatial Networks. *Proc. VLDB Endow.* 10, 11 (2017), 1178–1189. https://doi.org/10.14778/3137628.3137630
[82] Shuo Shang, Lisi Chen, Kai Zheng, Christian S. Jensen, Zhewei Wei, and Panos Kalnis. 2019. Parallel Trajectory-to-Location Join. *IEEE Trans. Knowl. Data Eng.* 31, 6 (2019), 1194–1207. https://doi.org/10.1109/TKDE.2018.2854705
[83] Seyedeh Zeinab Shogrkhodaei, Seyed Vahid Razavi-Termeh, and Amanollah Fathnia. 2021. Spatio-temporal modeling of PM2.5 risk mapping using three machine learning algorithms. *Environmental Pollution* 289 (2021), 117859.
[84] Bernard W Silverman. 2018. *Density estimation for statistics and data analysis*. Routledge.
[85] Justin Song, Richard Frank, Patricia L. Brantingham, and Jim LeBeau. 2012. Visualizing the spatial movement patterns of offenders. In *SIGSPATIAL*. ACM, 554–557. https://doi.org/10.1145/2424321.2424413
[86] Lisa Tompson, Henry Partridge, and Naomi Shepherd. 2009. Hot routes: Developing a new technique for the spatial analysis of crime. *Crime Mapping: A Journal of Research and Practice* 1, 1 (2009), 77–96.
[87] Reaz Uddin, Chinya V. Ravishankar, and Vassilis J. Tsotras. 2018. Indexing moving object trajectories with hilbert curves. In *SIGSPATIAL*. ACM, 416–419. https://doi.org/10.1145/3274895.3274912
[88] Haojun Wang and Roger Zimmermann. 2010. A Novel Dual-Index Design to Efficiently Support Snapshot Location-Based Query Processing in Mobile Environments. *IEEE Trans. Mob. Comput.* 9, 9 (2010), 1280–1292. https://doi.org/10.1109/TMC.2010.63
[89] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB*. 194–205.
[90] Jessica L Weir, Kirsten Vacura, Jay Bagga, Adam Berland, Kieran Hyder, Christian Skov, Johan Attby, and Paul A Venturelli. 2022. Big data from a popular app reveals that fishing creates superhighways for aquatic invaders. *PNAS nexus* 1, 3 (2022), pgac075.
[91] Xike Xie, Hua Lu, and Torben Bach Pedersen. 2013. Efficient distance-aware query evaluation on indoor moving objects. In *ICDE*. IEEE, 434–445. https://doi.org/10.1109/ICDE.2013.6544845
[92] Bo Yang, Lin Liu, Minxuan Lan, Zengli Wang, Hanlin Zhou, and Hongjie Yu. 2020. A spatio-temporal method for crime prediction using historical crime data and transitional zones identified from nightlight imagery. *International Journal of Geographical Information Science* 34, 9 (2020), 1740–1764.
[93] Wenyue Yang, Bi Yu Chen, Xiaoshu Cao, Tao Li, and Peng Li. 2017. The spatial characteristics and influencing factors of modal accessibility gaps: A case study for Guangzhou, China. *Journal of Transport Geography* 60 (2017), 21–32. https://doi.org/10.1016/j.jtrangeo.2017.02.005
[94] Wei Yang, Jie Hu, Yong Liu, and Wenbo Guo. 2023. Examining the influence of neighborhood and street-level built environment on fitness jogging in Chengdu, China: a massive GPS trajectory data analysis. *Journal of transport geography* 108 (2023), 103575.
[95] Xue Yang, Xuejiao Zheng, Yanjia Cao, Hao Chen, Luliang Tang, and Honghai Yang. 2023. Connectivity analysis in pedestrian networks: A case study in Wuhan, China. *Applied geography* 151 (2023), 102843.
[96] Jia Yu and Mohamed Sarwat. 2021. GeoSparkViz: a cluster computing system for visualizing massive-scale geospatial data. *VLDB J.* 30, 2 (2021), 237–258. https://doi.org/10.1007/s00778-020-00645-2
[97] P. Zezula, G. Amato, V. Dohnal, and M. Batko. 2006. *Similarity Search: The Metric Space Approach*. Springer US. https://books.google.com.hk/books?id=KTkWXsiPXR4C
[98] Baihua Zheng, Jianliang Xu, Wang-Chien Lee, and Dik Lun Lee. 2006. Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services. *VLDB J.* 15, 1 (2006), 21–39. https://doi.org/10.1007/s00778-004-0146-0
[99] 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.
[100] Yan Zheng, Yi Ou, Alexander Lex, and Jeff M. Phillips. 2021. Visualization of Big Spatial Data Using Coresets for Kernel Density Estimates. *IEEE Trans. Big Data* 7, 3 (2021), 524–534. https://doi.org/10.1109/TBDATA.2019.2913655
[101] Yan Zheng and Jeff M. Phillips. 2015. L∞ Error and Bandwidth Selection for Kernel Density Estimates of Large Data. In *SIGKDD*. 1533–1542. https://doi.org/10.1145/2783258.2783357