CheetahTraj: Quality and Efficiency in Large-scale Trajectory Data Visual Exploration

Qiaomu Shen, Chaozu Zhang, Xiao Yan, Chuan Yang, Dan Zeng, Wei Zeng, Bo Tang

Abstract—Visualizing large-scale trajectory data is a core subroutine for many applications, e.g., traffic management, urban planning, and route recommendation. However, naively visualizing all trajectories for a target region could result in long delay due to large data volume. Ad-hoc sampling can reduce visualization time but may harm visual quality, i.e., generating visualizations that look substantially different from the exact one. In this paper, we propose the CheetahTraj framework to provide high quality trajectory visualization for arbitrary target region with low latency. To this end, we first define a natural pixel-based *visual quality function* to measure the similarity between two visualizations and formulate a quality optimal trajectory sampling problem. Then, we design the VQGS and VQGS$^+$ algorithms to solve the trajectory sampling problem, which not only provide guaranteed visual quality but also reduce visual clutter. To generate quality guaranteed trajectory samples with high efficiency, we develop quad-tree-based index (InvQuad) that allows to use trajectory samples computed offline. Extensive experiments (i.e., case-, user-, and quantitative- studies) are conducted on 3 real-world trajectory datasets to verify visualization quality and efficiency of CheetahTraj. The results show that CheetahTraj consistently provide high quality visualizations and its visualization time is orders of magnitude shorter than visualizing all trajectories.

Index Terms—Trajectory visualization, interactive data exploration, sampling

1 Introduction

The ubiquity of location-acquisition devices leads to an explosive growth of movement data (i.e., trajectories), e.g., for vehicles, shared bikes and pedestrians. Visualizing these large-scale trajectory data is crucial for many smart city applications [1], [2], [3] and location-based services [4], [5]. Among various visualization methods, line-based trajectory visualization, which connects the locations of a moving object by polylines, is widely adopted for spatial-temporal data analytics [6], [7], [8]. To support interactive exploration on trajectory data, it is crucial to conduct line-based trajectory visualization for an arbitrary target region with high quality and low latency.

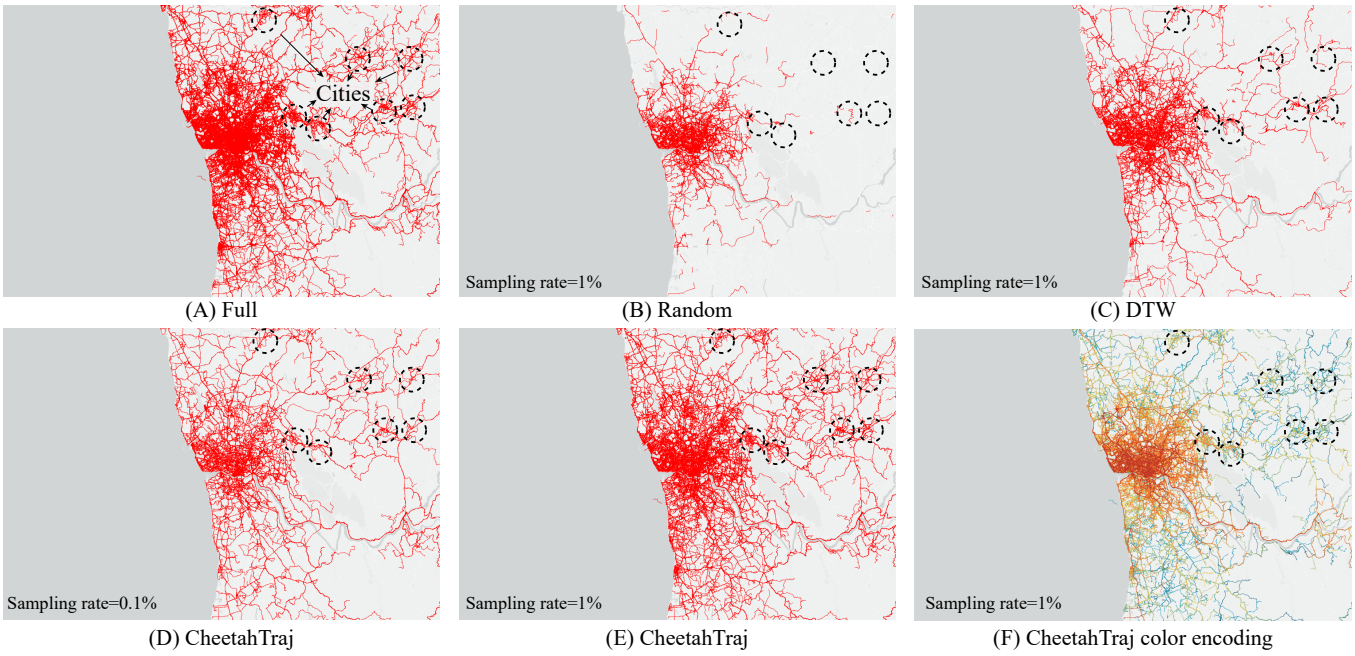
Long visualization time for large-scale datasets: To visualize the trajectories in a target region $\mathcal{Q}$, a natural solution is to find the trajectories in it and visualize all these trajectories (Full for short). However, Full may suffer from a long visualization time as the trajectory datasets can be extremely large. For example, Shenzhen has 24,237 taxis which collectively generate more than 7.72 million GPS locations each day [9]. In addition, our profiling results in Table 1[1] show that it takes 16.154 seconds to visualize the 1,000,000 taxi trajectories in the Porto dataset. As visualization time scales almost linearly with the number of trajectories (see Table 1), Full could take a long time if there are many trajectories in the target region. However, interactive visual exploration typically requires the visualization to be generated in less than 1 second [10]. Another problem of applying Full on large-scale datasets is *visual clutter* [11], where there are too many points in the visualization such that it is difficult for human users to gain insights. We provide an example Figure 1(A) for the Porto dataset, for which it is almost impossible to recognize the road networks in the dense region at the center.

TABLE 1
Visualization time on the Porto dataset (in seconds)

No. of trajectories	No. of GPS points	Mapping time	Rendering time	Total time
1,000	31,300	0.027	0.003	0.03
10,000	31,6531	0.169	0.005	0.174
100,000	316,7120	1.701	0.057	1.758
1,000,000	31,646,379	15.562	0.592	16.154

Ad-hoc sampling has poor visual quality: Sampling techniques are widely used to accelerate large-scale data analysis in both database and visualization communities [12], [13], [14], [15]. By selecting a subset of the trajectories in the target region for visualization, sampling can reduce both visualization time and visual clutter. One such example is ScalaR [16], which employs a reduction layer between the visualization layer and the data management layer. The reduction layer samples records *uniformly at random* (denoted as Random) when the query results are too large. However, Random has poor visual quality as its visualization could be significantly different from the ground-truth. We provide such an example in Figure 1(B), where Random fails to include trajectories in the sparse areas of Figure 1(A). Another natural idea is to sample trajectories with good diversity and we develop such a baseline using the famous Dynamic Time Warping (DTW) distance between trajectories [17]. As shown in Figure 1(C), DTW provides better visualization than Random but there are still obvious differences between DTW and the ground-truth in Figure 1(A). Without explicit visual quality guarantee, sampling trajectories in ad-hoc ways may produce visualizations with poor quality and

1. Mapping is the time to map the GPS points to the screen points, and rendering is the time to render the screen points. The settings of this measurement can be found in Section 6.



(A) Full (B) Random (C) DTW

(D) CheetahTraj (E) CheetahTraj (F) CheetahTraj color encoding

Fig. 1. Effectiveness of CheetahTraj at overview visualization in Porto.

mislead visual exploration.

The CheetahTraj **framework:** We explore novel algorithm and efficient index jointly in the CheetahTraj framework to provide visualizations with high quality and low latency. To conduct quality guaranteed sampling, we first propose a novel pixel-based *visual quality function* to measure how similar an approximate visualization is to the ground-truth. We also show that it is NP-hard to select an optimal set of trajectories that maximize the visual quality function. Next, we devise a *visual quality guaranteed sampling algorithm* named VQGS, which provides theoretical visual quality guarantee for the sampled trajectories. Then, we *tackle the visual clutter problem* by taking data distribution and human perception into consideration in an advance algorithm named VQGS$^+$. To avoid running the somehow complex VQGS$^+$ algorithm on-line for interactive visual exploration, we design an InvQuad-tree index based on quad-tree. InvQuad-tree allows to directly use the sampling results computed in an offline index building phase and provides quality guaranteed trajectory samples for an arbitrary target region.

We conduct extensive case study, user study and quantitative performance evaluation to validate the visualization quality and efficiency of the CheetahTraj framework. The case study shows that CheetahTraj consistently provides high quality visualizations for both large target regions and small target regions. The user study with 35 participants confirms that CheetahTraj effectively reduces visual clutter and produces visualizations that are plausible to human inspectors. The quantitative performance evaluation shows that CheetahTraj provides good visual quality by sampling only a small number of trajectories. In addition, CheetahTraj produces high quality visualizations for arbitrary target regions in less than 1 second for all 3 experiment datasets and the visualation delay is below 0.1 second in most cases.

We illustrates the merits of our CheetahTraj framework in Figure 1. Figure 1(D) and (E) are the visualizations produced by CheetahTraj on the Porto dataset with sampling rate 0.1% and 1%, respectively. Compared with uniform random sampling (i.e., Random) and diversity based sampling (i.e., DTW) in Figure 1(B) and (C), Figure 1(D) and (E) are obviously more similar to the full dataset visualization in Figure 1(A). Figure 1(F) is produced by CheetahTraj using the same parameters as Figure 1(E) but the trajectories are colored according to their algorithm-generated representativeness (warmer color means more representative). Compared with Figure 1(A), the main routes in the dense region can be identified much more easily, which shows that CheetahTraj effectively reduces visual clutter. Last but not least, it takes CheetahTraj only 0.116 seconds and 0.339 seconds to generate Figure 1(D) and (E), respectively, while the full visualization in Figure 1(A) takes 16.154 seconds.

To sum up, our technical contributions in this paper include:

- We formulate the visual quality optimal trajectory sampling problem for large-scale trajectory data visualization, and prove that it is NP-hard (Section 3).
- We devise an approximate algorithm VQGS for the visual quality guaranteed sampling problem. VQGS is further improved with VQGS$^+$ by considering data distribution and human perception (Section 4).
- We propose the CheetahTraj framework, which jointly uses the aforementioned algorithms and tailored index to achieve both quality and efficiency in large-scale trajectory data visualization (Section 5).

This rest of the paper is organized as follows. Section 2 introduces related works on trajectory visualization and interactive visualization. The quality optimal trajectory sampling problem is formulated and analyzed in Section 3. Section 4 presents our VQGS and VQGS$^+$ algorithm for quality guaranteed trajectory sampling. Section 5 elaborates the CheetahTraj framework. The experiment results are presented in Section 6. Section 7 draws the concluding remarks.

2 Background and Related Work

In this part, we survey related works on *trajectory visualization methods* in Section 2.1 and *interactive data visualization for large datasets* in Section 2.2, respectively.

2.1 Trajectory Visualization Methods

A trajectory is a sequence of spatial locations (e.g., GPS positioning results) and trajectories are the most common representations of object movements. Existing trajectory visualization methods can be classified into three categories according to the form of visualization [6], i.e., *point-based*, *region-based*, and *line-based*. We give a brief introduction to these methods and refer the interested readers to [6] for more detailed discussions.

Point-based visualization plots the locations in the trajectories independently and captures the overall spatial distribution of the moving objects. Many density-based methods [18], [19], [20], [21], e.g., kernel density estimation, are applied in point-based visualization to preserve the spatial distribution. Region-based visualization slices the entire region into sub-regions and visualizes the aggregated information in each sub-region [22], [23]. As region-based visualization focuses on aggregated statistics, it is most effective in capturing macro-patterns. In this work, we focus on line-based visualization, which uses polylines to connect the locations in each trajectory and shows the trace of object movements (see an example in Figure 3). As line-based visualization preserves the continuous movement information of objects [24], [25], it is widely used in many visual analysis applications such as traffic management, urban planning, and route recommendation. However, line-based visualization is known to suffer from severe visual clutter, especially when the dataset is large. Several techniques have been proposed to alleviate visual clutter, such as clustering-based techniques [23] and advanced interaction techniques [26].

2.2 Interactive Visualization for Large Datasets

Figure 2 illustrates a general architecture of interactive visualization systems, e.g., Spotfire[2], Tableau[3], ATLAS [27], Viate [28] and Marviq [29]. There are typically three layers: user interface in the front-end layer, optimization techniques in the middle-layer, and database management system (usually cloud-based) in the back-end layer. The visualization community usually focuses on improving the effectiveness of data visualization at the front-end, e.g., designing novel visualization methods/toolkits such as D3[4] to enable data analysts to effectively gain insights from data. The database community usually aims to improve query efficiency, e.g., devising big data systems such as Spark[5] for efficient data processing at the back-end. With the popularization of location-acquisition devices, the scale of trajectory datasets can be extremely large. For example, the taxis in Shenzhen generate ~9.3GB trajectory data per day. However, visualization generation has a long latency for large datasets due to heavy data processing/graphic rendering, which harms

2. https://www.tibco.com/products/tibco-spotfire
3. https://www.tableau.com/
4. https://d3js.org/
5. https://spark.apache.org/

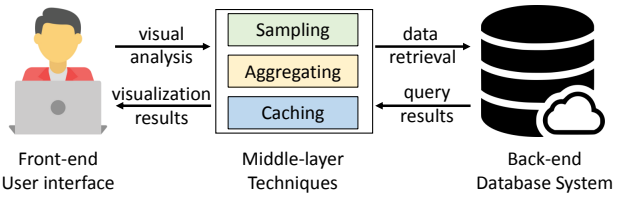


Fig. 2. System architecture for interactive visualization.

the responsiveness of interactive visualization. Therefore, both the visualization and database communities began to advance techniques in the middle-layer to reduce visualization latency for large datasets. We briefly elaborate these techniques as follows.

Aggregation-based techniques: These works divide the entire area into basic units and visualize the aggregated information of the trajectories for each unit [22], [23], [30]. For more details on aggregation-based techniques, we refer the reader to [31], [32]. Our problem and solutions are different from these works as we focus on visualizing the raw trajectories, instead of aggregated statistics.

Sampling-based techniques: Sampling is widely used in both visualization and database communities [12], [13], [14], [15], [16], [33], [34], [35]. These works try to reduce the dataset to a subset with some special characteristics: such as blue noise property [33], multi-class property [34] or maximize some user-defined quality [35]. The work most relevant to ours is [15], which is designed for scatter plots (a form of point-based visualization). It reduces the number of points in a plot while preserving the spatial distribution of the points in the original dataset. The techniques in [15] cannot be applied to our trajectory visualization problem as trajectory is more complex than individual scatter points (e.g., the order of GPS points is essential and the trajectories could have a large variance in length). Some works simplify a trajectory by sampling important points to reduce data size [36], [37] or alleviate visual clutter [17], [38]. These works are orthogonal to ours as we are sampling complete trajectories instead of points in a trajectory.

Caching-based and other techniques: Chan et al. propose ATLAS [27], which utilizes caching for efficient data communication between server and client. ATLAS also exploits a powerful multi-core server to accelerate visual analysis tasks in both the middle-layer and back-end. Piringer et al. [39] propose an architecture for interactive visual exploration, which utilizes multi-core devices and avoids the common pitfalls of multi-threading to provide quick visual feedback. Our work is orthogonal to these execution optimizations as we mainly focus on the algorithm perspective.

Novelties of our work. To the best of our knowledge, we are the first to formulate the quality optimal trajectory sampling problem to accelerate visualization on large scale datasets. We devise effective algorithms for this problem, which not only provide visual quality guarantee but also reduce the well-know visual clutter in trajectory visualization. Based on these algorithm, we design the CheetahTraj framework with a tailored InvQuad-tree index to produce high quality visualization for arbitrary target region with low latency.

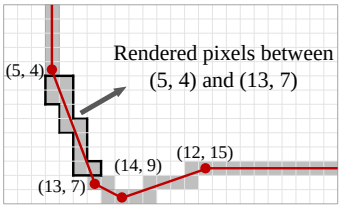


Fig. 3. Illustration of line-based trajectory visualization.

3 Problem Formulation

In this section, we first formally define the *quality optimal sampling problem* (QOSP) for large-scale trajectory data visualization in Section 3.1, and then show that it is NP-hard to solve the problem exactly in Section 3.2.

3.1 Problem Definition

We motivate our definition of the *visualization quality function* by introducing how line-based trajectory visualization works. As elaborated earlier, a trajectory contains a sequence of 2-dimensional locations. Given an empty canvas (i.e., the screen of a displaying device) with pixels indexed by horizontal and vertical coordinates (i.e., x and y), line-based trajectory visualization connects consecutive locations in each trajectory with polylines and marks the pixels passed by these polylines (with a color different from the background). As shown in Figure 3(A), the result of line-based trajectory visualization can be regarded as a 2-dimensional array of boolean variables with 1 indicting that a pixel has been marked. Alternatively, we can treat a visualization result as a set $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{n}$ that contains all marked pixels. This observation leads to the following definition of visualization quality function

$$\mathtt{Q}(\mathcal{S}, \mathcal{S}') = \frac{|\mathcal{S} \cap \mathcal{S}'|}{|\mathcal{S}|}, \tag{1}$$

in which $|.|$ measures the cardinality of a set, $\mathcal{S}$ is the visualization result of the entire trajectory dataset $\mathcal{T}$ while $\mathcal{S}'$ is the visualization result of some trajectories sampled from $\mathcal{T}$. As $\mathcal{S}' \subseteq \mathcal{S}$, $\mathtt{Q}(\mathcal{S}, \mathcal{S}')$ essentially measures the ratio of the pixels in the ground-truth visualization $\mathcal{S}$ that are marked in the approximate visualization $\mathcal{S}'$. This definition matches human visual perception and the approximate visualization $\mathcal{S}'$ will look similar to $\mathcal{S}$ if $\mathtt{Q}(\mathcal{S}, \mathcal{S}')$ is large. Sampling reduces the number of trajectories and location points to process, and thus shortens the visualization time. With the quality function, we define the quality optimal sampling problem as follows.

Problem 1 (Quality Optimal Sampling Problem, QOSP). *Let the entire trajectory dataset be $\mathcal{T}$ and a sample set that contains some trajectories from $\mathcal{T}$ be $\mathcal{R}$. Using $\mathtt{V}(\mathcal{U})$ to denote the visualization result set derived from a trajectory set $\mathcal{U}$, with a sampling rate α, the quality optimal sampling problem finds a set $\mathcal{R}$ that satisfies*

$$\max_{\mathcal{R} \subseteq \mathcal{T}, |\mathcal{R}| = \lceil \alpha |\mathcal{T}| \rceil} \mathtt{Q}(\mathtt{V}(\mathcal{T}), \mathtt{V}(\mathcal{R})) = \frac{|\mathtt{V}(\mathcal{T}) \cap \mathtt{V}(\mathcal{R})|}{|\mathtt{V}(\mathcal{T})|}. \tag{2}$$

Note that we are sampling *complete trajectories* instead of *individual locations* in QOSP such that the lines and orientations in the trajectories are persevered.

Intuitively, given a visualization quality threshold τ, the QOSP problem can be transformed to find the sampled trajectory set $\mathcal{R}$ with the smallest α, under which provides the quality requirement holds, i.e, $\mathtt{Q}(\mathtt{V}(\mathcal{T}), \mathtt{V}(\mathcal{R})) \geq \tau$.

3.2 Hardness Analysis

We use $t_i \in \mathcal{T}$ to denote a trajectory in the dataset. According to the working mechanism of line-based trajectory visualization, t_i corresponds to a set of marked pixels on the canvas in the ground-truth visualization $\mathtt{V}(\mathcal{T})$ and we also use t_i to denote this set of pixels. Thus, we have $\mathtt{V}(\mathcal{T}) = \cup_{t_i \in \mathcal{T}} t_i$ and $\mathtt{V}(\mathcal{R}) = \cup_{t_i \in \mathcal{R}} t_i$. We can transform Problem 1 as follows:

$$\begin{aligned} &\max_{\mathcal{R} \subseteq \mathcal{T}, |\mathcal{R}| = \lceil \alpha |\mathcal{T}| \rceil} \frac{|\mathtt{V}(\mathcal{T}) \cap \mathtt{V}(\mathcal{R})|}{|\mathtt{V}(\mathcal{T})|} \\ \Leftrightarrow &\max_{\mathcal{R} \subseteq \mathcal{T}, |\mathcal{R}| = \lceil \alpha |\mathcal{T}| \rceil} |\mathtt{V}(\mathcal{R})| \Leftrightarrow \max_{\mathcal{R} \subseteq \mathcal{T}, |\mathcal{R}| = \lceil \alpha |\mathcal{T}| \rceil} |\cup_{t_i \in \mathcal{R}} t_i|. \end{aligned} \tag{3}$$

The transformations use the fact that $\mathtt{V}(\mathcal{R}) \subseteq \mathtt{V}(\mathcal{T})$ as $\mathcal{R} \subseteq \mathcal{T}$, and the ground-truth marked point set $\mathtt{V}(\mathcal{T})$ has constant cardinality. The last line shows that QOSP is equivalent to the famous set cover maximization problem[6]. Specifically, given an integer k, and a collection of sets $\mathcal{T} = \{t_1, t_2, \cdots, t_n\}$, set cover maximization finds a subset $\mathcal{R} \subset \mathcal{T}$ such that $|\mathcal{R}| = k$ and the number of covered elements $|\cup_{t_i \in \mathcal{R}} t_i|$ is maximized. The set cover maximization problem is well-known to be NP-hard [40]. For sampling-based methods, the visualization quality is determined by the sample set $\mathcal{R}$, and thus we use $\mathtt{Q}(\mathcal{R})$ to denote $\mathtt{Q}(\mathtt{V}(\mathcal{T}), \mathtt{V}(\mathcal{R}))$ for conciseness.

4 QOSP Solution

In this section, we first present the VQGS algorithm as a solution to QOSP and propose techniques to optimize its efficiency in Section 4.1. Then we improve VQGS with an advanced algorithm VQGS$^+$ by considering trajectory data distribution and human perception capability in Section 4.2.

4.1 Visual Quality Guaranteed Sampling VQGS

Our visual quality guaranteed sampling method (VQGS) is presented in Algorithm 1, which takes the trajectory dataset $\mathcal{T}$ and a sampling rate α as input (i.e., $k = \lceil \alpha |\mathcal{T}| \rceil$). VQGS employs a greedy paradigm and finds the trajectory tmp in $\mathcal{T}$ that maximizes $|\text{tmp} \cup \mathtt{V}(\mathcal{R})|$ at each iteration, as shown in Line 3 of Algorithm 1. It terminates after $k = \lceil \alpha |\mathcal{T}| \rceil$ iterations and returns $\mathcal{R}$ as the result set. As the visualization quality $\mathtt{Q}(\mathcal{R})$ can be computed after each iteration in Algorithm 1 with pre-computed ground-truth $\mathtt{V}(\mathcal{T})$, alternatively, we can terminate the algorithm when $\mathtt{V}(\mathcal{R}) \geq \tau$, in which τ is the quality threshold.

Algorithm 1 provides provable visual quality guarantee for the result set $\mathcal{R}$, as stated in Theorem 1.

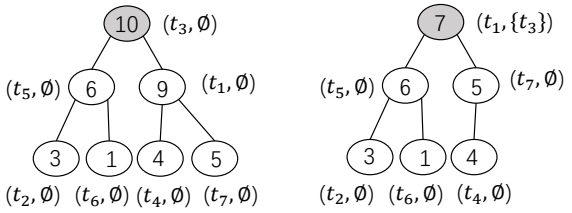
Theorem 1. *Given a sample rate α, and let the optimal solution to QOSP defined in Equation (2) be $\mathcal{R}^\star$ and the solution provided by Algorithm 1 be $\mathcal{R}$, we have $\mathtt{Q}(\mathcal{R}) \geq 0.632 * \mathtt{Q}(\mathcal{R}^\star)$.*

Theorem 1 follows directly from the submodularity of the visualization quality function $\mathtt{Q}(\mathcal{R})$, and it is well known

6. https://en.wikipedia.org/wiki/Maximum_coverage_problem

Algorithm 1 VQGS($\mathcal{T}, k = \lceil \alpha|\mathcal{T}| \rceil$)

1: Initialize the result set $\mathcal{R} \leftarrow \emptyset$
2: **while** $|\mathcal{R}| < k$ **do**
3: $\quad$ tmp $\leftarrow \arg\max_{t_i \in \mathcal{T}} |t_i \cup \mathtt{V}(\mathcal{R})|$
4: $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{\text{tmp}\}$
5: Return $\mathcal{R}$



(A) 1st iteration (B) 2nd iteration

Fig. 4. Heap-based lazy computation.

that greedy solution provides a 0.632 approximation of the optimal solution for a submodular function [41]. As $\mathtt{Q}(\mathcal{R})$ is simply a linear scaling of $|\mathtt{V}(\mathcal{R})|$ as shown in Equation (3), we prove $|\mathtt{V}(\mathcal{R})|$ is submodular as follows.

Lemma 1 (Submodularity)**.** *Define the contribution value of a trajectory t to a sample set $\mathcal{R}$ as $\Delta(\mathcal{R}, t) = |\mathtt{V}(\mathcal{R} \cup t)| - |\mathtt{V}(\mathcal{R})|$. Given a trajectory t and two sample sets $\mathcal{R}, \mathcal{R}^{'}$, if $\mathcal{R} \subset \mathcal{R}^{'}$, then $\Delta(\mathcal{R}, t) \geq \Delta(\mathcal{R}^{'}, t)$.*

Proof. The contribution value of trajectory t w.r.t. a given result set $\mathcal{R}$ (i.e., $\Delta(\mathcal{R}, t) = |\mathtt{V}(\mathcal{R} \cup t)| - |\mathtt{V}(\mathcal{R})|$) is the number of pixels covered by t but not the trajectory set $\mathcal{R}$, which can be expressed as $|\mathtt{V}(t)| - |\mathtt{V}(\mathcal{R}) \cap \mathtt{V}(t))|$. We have $\mathtt{V}(t) \cap \mathtt{V}(\mathcal{R}) \subseteq \mathtt{V}(t) \cap \mathtt{V}(\mathcal{R}^{'})$ because $\mathcal{R}^{'}$ is a superset of $\mathcal{R}$, which implies $|\mathtt{V}(t)| - |\mathtt{V}(\mathcal{R}) \cap \mathtt{V}(t))| \geq |\mathtt{V}(t)| - |\mathtt{V}(\mathcal{R}^{'}) \cap \mathtt{V}(t))|$. Thus, it holds that $\Delta(\mathcal{R}, t) = |\mathtt{V}(\mathcal{R} \cup t)| - |\mathtt{V}(\mathcal{R})| \geq |\mathtt{V}(\mathcal{R}^{'} \cup t)| - |\mathtt{V}(\mathcal{R}^{'})| = \Delta(\mathcal{R}^{'}, t)$. □

Although Algorithm 1 provides quality guarantee for the result set $\mathcal{R}$, it has a high time complexity, which we show in the following analysis.

Lemma 2 (Time Complexity)**.** *For a trajectory dataset $\mathcal{T}$ and an integer $k = \lceil \alpha|\mathcal{T}| \rceil$, the time complexity of Algorithm 1 is $O(\alpha \cdot m \cdot |\mathcal{T}|^2)$, where m is the maximum length for the trajectories in dataset $\mathcal{T}$.*

Proof. In each iteration, Algorithm 1 computes the trajectory with the largest number of uncovered pixels in dataset $\mathcal{T}$. It take $O(m)$ cost to compute the number of uncovered pixels for each trajectory in $\mathcal{T}$. Algorithm 1 runs for $k = \lceil \alpha|\mathcal{T}| \rceil$ iterations. Hence, the total cost is $O(k \cdot m \cdot |\mathcal{T}|) = O(\alpha \cdot m \cdot |\mathcal{T}|^2)$. □

The high complexity of Algorithm 1 hurts its scalability for large-scale trajectory datasets. For example, the Porto dataset contains 2.39 millions taxi trajectories, Algorithm 1 takes 413.6 seconds to obtain the result set $\mathcal{R}$ with sampling rate 0.1%.

Heap-based lazy Computation: Algorithm 1 essentially adds the trajectory that maximizes $\Delta(\mathcal{R}, t) = |\mathtt{V}(\mathcal{R} \cup t)| - |\mathtt{V}(\mathcal{R})|$ to $\mathcal{R}$ in each iteration. Lemma 1 shows that the contribution of a trajectory (i.e., $\Delta(\mathcal{R}, t)$) cannot increase when Algorithm 1 runs for more iterations because $\Delta(\mathcal{R}^{'}, t) \leq \Delta(\mathcal{R}, t)$ for $\mathcal{R} \subset \mathcal{R}^{'}$. For example, the contribution of t_1 is 9 at the first iteration (i.e., $\mathcal{R} = \emptyset$), see Figure 4(a). Its contribution turns 7 when $\mathcal{R} = \{t_3\}$ at the second iteration, as shown in Figure 4(b). Based on this property, we can use $\Delta(\mathcal{R}, t)$ calculated in the previous iterations to prune it from contribution computation. Specifically, if we have $\Delta(\mathcal{R}, t) \leq \Delta(\mathcal{R}^{'}, t')$, in which $\mathcal{R}$ and $\mathcal{R}^{'}$ are a previous and the current sample set, respectively, we know that t can not be added to the sample set in the current iteration as $\Delta(\mathcal{R}^{'}, t) \leq \Delta(\mathcal{R}^{'}, t')$ and t' is a better choice. As shown in Figure 4(b), even we do not know $\Delta(\mathcal{R}' = \{t_3\}, t_7)$ exactly, we can conclude t_7 will not be the sample set in the second iteration as $\Delta(\mathcal{R} = \emptyset, t_7) = 5 < \Delta(\mathcal{R}' = \{t_3\}, t_1) = 7$.

To implement this idea, we maintain a max-heap for the number of uncovered pixels in each trajectory and update the contribution of a trajectory only when necessary, i.e., computing in a lazy manner.

Consider a tiny example with 7 trajectories, i.e., t_1 to t_7. Figure 4(a) shows the initial max-heap and the contributions of trajectories t_1 to t_7 w.r.t result set $\mathcal{R} = \emptyset$. At the first iteration, the root node of the max-heap, t_3 in Figure 4(A), is selected. At the second iteration, the number of uncovered pixels of the new root node t_1 is updated to 7 w.r.t. result set $\mathcal{R} = \{t_3\}$ (see the gray node in Figure 4(B)). Then t_1 is selected at the second iteration without computing the contributions of other trajectories w.r.t $\mathcal{R} = \{t_3\}$. The reason is that the contributions of these trajectories are all less than 7 when $\mathcal{R} = \emptyset$, according to the submodularity in Lemma 1, their contributions must be smaller than 7 when $\mathcal{R} = \{t_3\}$. The efficiency of Algorithm 1 is significantly improved with heap-based lazy computation. Recall that Algorithm 1 takes 413.6 seconds with sampling rate 0.1% on the Porto dataset while our performance-optimized VQGS needs only 1.2 seconds.

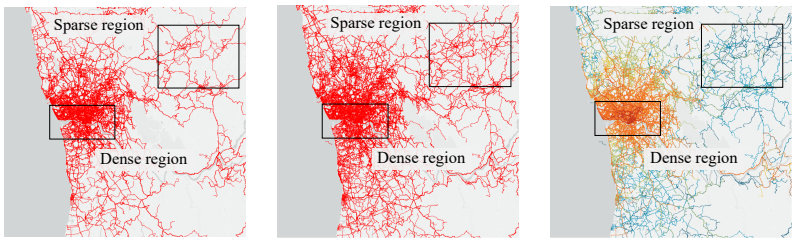
4.2 Advanced Approach VQGS$^+$

In this part, we improve VQGS by considering (i) trajectory data distribution, and (ii) human perception capability. We elaborate (i) and (ii) by the examples in Figure 5.

Trajectory data distribution: Considering the Porto trajectory dataset, Figure 5(A) is the visualization result of VQGS with sampling rate 0.5%. It is obvious that the trajectories follow a non-uniform distribution, and there are some dense regions and sparse regions as illustrated by the two rectangles in Figure 5(A). There are many points in the dense region, which creates visual clutter and makes it difficult to identify the main roads.

Human perception capability: Comparing Figures 5(A) and (B), it is easier to tell their differences in the sparse regions than in the dense regions. This is because human perception has limited capability, and hence two visualizations look indistinguishable if both of them contain a large number of points in the same area. The two dense regions look similar although Figure 5(B) contain fewer points in this region than Figure 5(A). However, for the sparse region, VQGS loses some trajectories and it is easy to tell the differences between Figures 5(A) and 5(B).

Based on the two observations above, we can improve VQGS by delivering richer information in the sparse regions and reducing visual clutter in the dense regions. VQGS$^+$



(A) VQGS (B) VQGS$^+$ (C) VQGS$^+$CE

Fig. 5. QOSP solution VQGS$^+$ on Porto ($\alpha = 0.5\%, \delta = 64$)

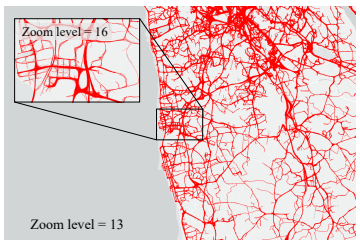


Fig. 6. An illustration of different zoom level.

in Algorithm 2 achieves both objectives using a perception tolerance parameter δ, which models the perception capability of humans. Specifically, if pixel (x, y) in the canvas is marked by the result set $\mathcal{R}$, the pixels around (x, y), i.e., from $(x - \delta, y - \delta)$ to $(x + \delta, y + \delta)$, do not need to be marked as they are close to the pixels in $\mathcal{R}$ and human perception cannot tell nearby pixels apart. We can easily modify VQGS in Algorithm 1 to incorporate the perception tolerance parameter δ as shown in Algorithm 2. VQGS$^+$ measures the contribution of each trajectory t_i w.r.t the augmented visualized point set $\mathtt{V}(\mathcal{R})^+$ in Line 4, where $\mathtt{V}(\mathcal{R})^+$ includes both pixels on the selected trajectories and their tolerance pixels (in Line 6). We also use the heap-based lazy computation to speedup VQGS$^+$.

Algorithm 2 VQGS$^+(\mathcal{T}, k = \lceil \alpha \mathcal{T} \rceil, \delta)$

1: Initialize result set $\mathcal{R} \leftarrow \emptyset$
2: Initialize augmented result set $\mathcal{R}^+ \leftarrow \emptyset$
3: **while** $|\mathcal{R}| < k$ **do**
4: $\quad \mathsf{tmp} \leftarrow \arg\max_{t_i \in \mathcal{T}} |t_i \cup \mathtt{V}(\mathcal{R})^+|$
5: $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{\mathsf{tmp}\}$
6: $\quad \mathtt{V}(\mathcal{R})^+ \leftarrow \mathtt{V}(\mathcal{R})^+ \cup \mathsf{augment}(\mathsf{tmp}, \delta)$
7: **for** each t in $\mathcal{T}$ **do** ▷ Representative encoding
8: $\quad tr \leftarrow \arg\min_{t_i \in \mathcal{R}} |t - \mathsf{augment}(t_i, \delta)|$
9: $\quad tr.\mathsf{cnt} + +$
10: Return $\mathcal{R}$

VQGS$^+$ in Algorithm 2 selects trajectories with good representativeness and some trajectories will not be included into the result set $\mathcal{R}$ even though they have more uncovered pixels w.r.t. $\mathcal{R}$. The reason is that their uncovered pixels are too close to the pixels in the selected trajectories (i.e., within the tolerance area of selected pixels). Compared with VQGS, VQGS$^+$ is more likely to sample trajectories in the sparse regions as their pixels are less likely to be covered by other trajectories as shown in Figures 5. Moreover, reducing the number of trajectories sampled from the dense regions helps to reduce visual clutter.

One subtlety is that different δ needs to used for different *zoom levels* (or regions with different sizes). For example, Google map [42] provides zoom levels from 0 to 20, with level 0 providing the largest visualization range (i.e., the whole world) but the lowest resolution, and level 20 providing the smallest visualization range (e.g., individual building, if available) but the highest resolution. We provided an illustration of zoom level in Figure 6 and users may select different zoom levels for visualization according their needs. Note that we define δ on the highest zoom level (i.e., using the raw distance of the locations) to account for different resolutions. If the zoom level is small (i.e., the visualization region is large), we can apply a large δ because locations with a large raw distance look close to each other in the visualization and we can afford to lose more details. If the zoom level is large (i.e., the visualization region is small), we need to use a small δ as users typically want to investigate some fine-grained details in this case and using a large δ will lose these details.

Color encoding scheme: The visual clutter problem for large-scale trajectory visualization can be further alleviated by encoding the representativeness of the trajectories in $\mathcal{R}$ with colors. We define the representativeness of a trajectory t_i in $\mathcal{R}$ as the size of its *reverse nearest neighbor set*, which contains the trajectories in $\mathcal{T}$ that has t_i as its nearest neighbor in $\mathcal{R}$. The distance between trajectory t and t_i is defined as the number of pixels in t that can not be covered by the augmented pixels of t_i. We compute the representativeness of each trajectory in $\mathcal{R}$ in Lines 7-9 in Algorithm 2. Figure 5(C) shows the visualization result by encoding trajectories with larger representativeness with warmer colors. Compared with Figure 5(B), the main roads in the dense region is clearer in Figure 5(C) with very warm colors.

5 THE CheetahTraj FRAMEWORK

Recall that our goal is to provide high quality trajectory visualization for any user selected region query with low latency. In this section, we first introduce the motivation behind the CheetahTraj framework, then present its two key procedures: *index building*, and *query processing*.

Motivation of CheetahTraj**:** Given a user selected region query $\mathcal{Q}$, a naive visualization procedure with our sampling algorithms works as follows: it first retrieves all trajectories (or trajectory segments) that are in this region (a.k.a, WayPoint query [43]), then it invokes VQGS$^+$ (or VQGS) to obtain a set $\mathcal{R}$ of sample trajectories, and finally the trajectories in $\mathcal{R}$ are rendered to the canvas (e.g., displaying device) as the visualization result. VQGS$^+$ has short *visualization time* as it effectively reduces the number of processed locations by sampling. However, VQGS$^+$ has a long *sampling time* (e.g., several seconds to tens of seconds) even with our performance optimization techniques. Hence, the naive procedure can not achieve low latency for large-scale trajectory visualization. To tackle this problem, we propose the CheetahTraj framework as illustrated in Figure 7. CheetahTraj consists of three modules: (i) index building, (ii) query processing, and (iii) result visualization, we elaborate them as follows.

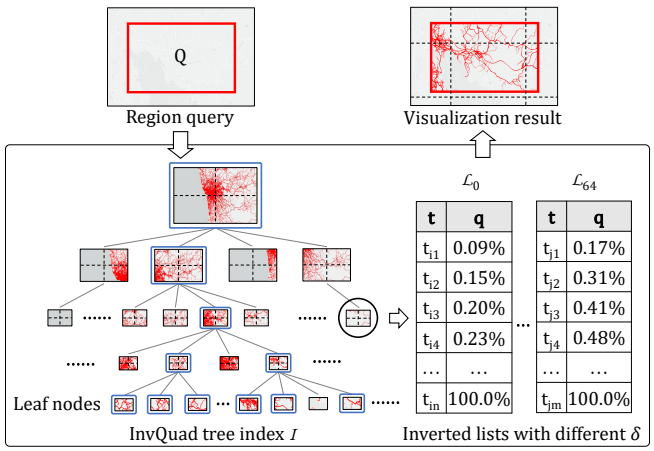


Fig. 7. The CheetahTraj framework

5.1 Index Building

The key idea of CheetahTraj is to conduct VQGS$^+$ sampling in the offline *index building* phase such that the sampling results can be used directly for online visualization. Specifically, we propose an inverted list augmented quad-tree index (InvQuad) to handle arbitrary query region.

As shown in the example InvQuad-tree index $\mathcal{I}$ at the bottom of Figure 7, we exploit a quad-tree to recursively partition the entire area (spanned by the trajectory dataset) into smaller areas and manage each area with a tree node. For each tree node, we run VQGS$^+$ using the trajectories (or trajectory segments) in its associated area as input to compute the *visualization quality inverted lists* for this area. $\mathcal{L}_0$ and $\mathcal{L}_{64}$ in Figure 7 are two example visualization quality inverted lists, in which the subscripts are the values of δ for this list. Specifically, we compute several inverted lists with different δ values[7] to support the efficient quality guaranteed result visualization at various zoom levels. For each inverted list, (i) VQGS$^+$ terminates until the quality of the sample set is 100%, i.e., the visualization result of the sample set is the same as the full dataset; (ii) the trajectory selected at each iteration of VQGS$^+$ is stored in the inverted list with its *cumulative quality* in ascending order. Take inverted list $\mathcal{L}_0$ in Figure 7 for example, t_{i4} is the trajectory selected at the 4th iteration, t_{i4}'s cumulative quality is 0.23%, which means that the quality achieved by $\{t_{i1}, t_{i2}, t_{i3}, t_{i4}\}$ as a whole is 0.23%. With the quality inverted list, searching a quality guaranteed sample set for a query region can be conducted efficiently via binary search.

5.2 Query Processing

For a region visualization query $\mathcal{Q}$ with quality threshold τ, Algorithm 3 summarizes the Query subroutine, which finds a quality guaranteed trajectory sample set $\mathcal{R}$. The algorithm starts by invoking Query$(\mathcal{Q}, \tau, \mathcal{I}.root, \mathcal{R} = \emptyset)$, i.e., from the root of InvQuad-tree index $\mathcal{I}$ with an empty result set $\mathcal{R}$. Then Algorithm 3 transverses the tree nodes recursively. If node $\mathcal{N}$ is a leaf node or its associated area is entirely contained in the query region, we retrieve a quality guaranteed trajectory set by calling subroutine findRet(), which conducts binary search on the proper inverted list in $\mathcal{N}$ (Line 2). Otherwise, we call Query() on the four children nodes of $\mathcal{N}$ (Line 3-6).

Algorithm 3 Query($\mathcal{Q}$, τ, InvQuad node $\mathcal{N}$, result $\mathcal{R}$)

1: **if** $\mathcal{N}$ is leaf node or $\mathcal{N}$ is entirely contained in $\mathcal{Q}$ **then**
2: $\quad\mathcal{R} \leftarrow \mathcal{R} \cup$ findRet$(\mathcal{N}, \tau)$
3: **else if** $\mathcal{Q} \cap \mathcal{N} \neq \emptyset$ **then**
4: $\quad$**for** i from 0 to 3 **do**
5: $\quad\quad$tmpQ $\leftarrow \mathcal{Q} \cap \mathcal{N}.child[i]$
6: $\quad\quad$Query(tmpQ, $\tau, \mathcal{N}.child[i], \mathcal{R}$)

Some trajectories in $\mathcal{R}$, the result returned by Query() for region $\mathcal{Q}$, may have segments outside $\mathcal{Q}$, we conduct a way point query WayPoint$(\mathcal{Q}, \mathcal{R})$ to filter these segments before visualization.

Correctness analysis: We first show that CheetahTraj meets the visualization quality requirement in Theorem 2 as follows.

Theorem 2. *If all selected nodes in the* InvQuad*-tree index $\mathcal{I}$ are entirely contained in the query region $\mathcal{Q}$, then the result set $\mathcal{R}$ returned by Algorithm 3 satisfies that* $\mathbb{Q}(\mathcal{R}) \geq \tau$.

Proof. Suppose query region $\mathcal{Q}$ selects areas $\mathcal{A}_1$,$\mathcal{A}_2$,$\cdots$,$\mathcal{A}_K$, these areas satisfy $\mathcal{A}_i \cap \mathcal{A}_j = \emptyset$ for $i \neq j$, and $\cup_{k=1}^{K} \mathcal{A}_k = \mathcal{Q}$. For each area $\mathcal{A}_k$, denote the number of points marked in the ground truth visualization as n_k, and the number of points marked by the trajectories in $\mathcal{R}$ as m_k, we have $\frac{m_k}{n_k} \geq \tau$ as we use the visualization quality inverted index for trajectory selection. Thus, for query region $\mathcal{Q}$ with result set $\mathcal{R}$, we have $\mathbb{Q}(\mathcal{R}) = \frac{\sum_{k=1}^{K} m_k}{\sum_{k=1}^{K} n_k} \geq \tau$. □

In more general cases, we also select some areas that only intersect with the query region $\mathcal{Q}$ and the sample set $\mathcal{R}$ may not satisfy $\frac{m_k}{n_k} \geq \tau$ for these areas. This does not significantly affect visualization quality for two reasons: (i) these areas are the leaf nodes of the InvQuad-tree index and thus reside on the border of the query region. When exploring the map, human tends to move the region of interest to the screen center, where is more "close" to eyes [44]. (ii) the areas of the border regions are small w.r.t. the query region if the InvQuad-tree has a sufficient height (i.e., the leaf nodes have a small area).

6 Experimental Evaluation

In this part, we first present a case study of the visualizations provided by CheetahTraj in Section 6.1 to demonstrate its good visualization quality. In Section 6.2, we conduct a comprehensive user study to compare the visualization quality of different methods. In Section 6.3, we quantitatively evaluate the visual quality and efficiency of CheetahTraj and compare with the baselines.

Experiment Settings: We conduct the experiments using 3 real-world trajectory datasets: Porto, Shenzhen and Chengdu. Porto [45] contains 2.39 million taxi trajectories and 75.67 million of GPS points, and the longest trajectory has 3,490 GPS points. Shenzhen [9] consists of 3.07 million taxi trajectories with 53.53 million GPS points, and the

7. We set δ as 0 (i.e., VQGS), 4, 8, 16, 32, 64. We need quality inverted lists with different δ for one area as the area may be covered by query regions of different sizes, and we use lists with larger δ for larger query region as discussed in Section 4.2.

TABLE 2
Statistics of the datasets used in the experiments

Dataset	No. of Trajectories	No. of GPS points	Maximum length
Porto	2,389,863	75,667,503	3,490
Shenzhen	3,066,861	53,527,890	2,268
Chengdu	2,400,000	80,040,361	6,468

longest trajectory has 2,268 GPS points. Chengdu [46] has 2.40 million taxi trajectories and 80.04 million GPS points, and the longest trajectory consists of 6,468 GPS points. The statistics of the datasets are summarized in Table 2. The experiments are conducted on a machine with an Intel i7-8700 CPU, 24 GB memory and an NVIDIA GeForce GTX1080 GPU with 8 GB on-chip memory, running on Windows 10. All methods are implemented using Java 1.8. UnfoldingMap 0.9.92 [47] is used to provide interactive map and GPS mapping, and the Processing 3 library [48] is used for rendering. All timing results are measured in single-thread mode. The datasets and source codes to reproduce our results are available at [49].

Competitors: We compare CheetahTraj with three competitors, i.e., Full, Random and DTW. Full visualizes all trajectories in the user selected region while Random selects trajectories in the user selected region at random for visualization. DTW is based on the DTW distance between trajectories [17] and designed by us to select trajectories with good diversity. Specifically, DTW samples the trajectory that maximizes the aggregate DTW distance to all remaining trajectories in each step. We note that it takes DTW several days to run on the experiment datasets because it needs to compute expensive DTW distance (quadratic complexity w.r.t. trajectory length) between all trajectory pairs. For fair comparison, we ensure that Random and DTW use the same number of trajectories as CheetahTraj.

6.1 Case Study

We conduct case study on the Porto dataset to demonstrate the visualization quality of CheetahTraj. Similar phenomenons are also observed on the other datasets and we omit their results for conciseness.

6.1.1 Overview visualization

We illustrate the visualization results of different methods for the entire Porto dataset in Figure 1.

Good visual quality for overview: At zoom level 11, Figure 1(A) is the visualization result of Full on the Porto dataset. With a sampling rate $\alpha = 1\%$, Figures 1(B), (C) and (E) are the visualizations produced by Random, DTW, and CheetahTraj, respectively. Comparing with Figure 1(B) and (C), it is obvious that Figure 1(E) is more similar to Figure 1(A). In particular, Figure 1(E) not only preserves the overall visual structure of the entire region but also keeps the details of cities that are far from the center (marked by the dashed cycles in the figure). However, the details of these cities are lost in Figure 1(B) as Random is more likely to select trajectories in the dense region. DTW in Figure 1(C) preserves more details than Random in the sparse regions as it considers diversity in trajectories but its visualization quality is still inferior compared with CheetahTraj in Figure 1(E).

Good visual quality under different sampling rates: Figure 1(D) and (E) are the visualizations produced by CheetahTraj with a sampling rate of 0.1% and 1%, respectively. We can make two observations: (i) the larger the sampling rate, the better the visual quality, i.e., Figures 1(E) is more similar to Figure 1(A) compared with Figure 1 (D); (ii) the visualization of CheetahTraj with a sampling rate of 0.1% (i.e., Figure 1(D)) looks more appealing than the visualizations of Random and DTW with a sampling rate of 1% (i.e., Figure 1(B) and (C)) as Figure 1(D) better preserves the visual structures of Figure 1(A).

Color encoding effectively mitigates visual clutter: At zoom level 11 and with a sampling rate of 1%, Figures 1(E) and (F) are the visualizations produced our CheetahTraj and CheetahTrajCE (i.e., CheetahTraj with color encoding), respectively. Visual clutter is severe for Full (i.e., Figure 1(A)) and CheetahTraj (i.e., Figure 1(E)) because many pixels are visualized for the dense region in the center, which makes it difficult to identify the main routes. The visualization of CheetahTrajCE in Figure 1(F) alleviates this problem by encoding more representative trajectories with warmer color, making it easier to identify some prominent routes than Figure 1(A) and (E).

6.1.2 Detail visualization

We analyze the visualizations produced by different methods for small areas with details by investigating two regions of interest in the Porto dataset in Figure 8.

Reduce visual clutter and preserve micro structures for dense region: At zoom level 15, region B in Figure 8(A) is the center of Porto and has the highest concentration of trajectories. Therefore, Full suffers from severe visual clutter and it is difficult to identify the road networks in Figure 8(B1). Random and DTW in Figure 8(B2) and (B3) reduce the visual clutter to some extent by sampling some trajectories. CheetahTraj in Figure 8(B4) is more successful in reducing the visual clutter of Full and allows to identify a much larger number of routes. In addition, CheetahTraj preserves more micro structures of the trajectories than Random and DTW, e.g., the circular route in the dashed circular region.

Preserve overall layout for sparse region: At zoom level 14, region C in Figure 8(A) contains the city of Casino Espinho and has fewer trajectories than the dense region in the center. In this case, the sampling methods need to keep the overall layout of the trajectories to provide good visualization quality. Compared with Full in Figure 8(C1), Random and DTW in Figure 8(C2) and (C3) fail to meet this requirement as they do not show any trajectory for areas far from the city, e.g., in the dashed circle. This makes their entire visualization layout very different from Full. In contrast, CheetahTraj in Figure 8(C4) preserves the trajectories in areas far from the city and has a overall layout similar to Full.

To sum up, the case study shows that CheetahTraj effectively mitigates visual clutter with sampling and color en-

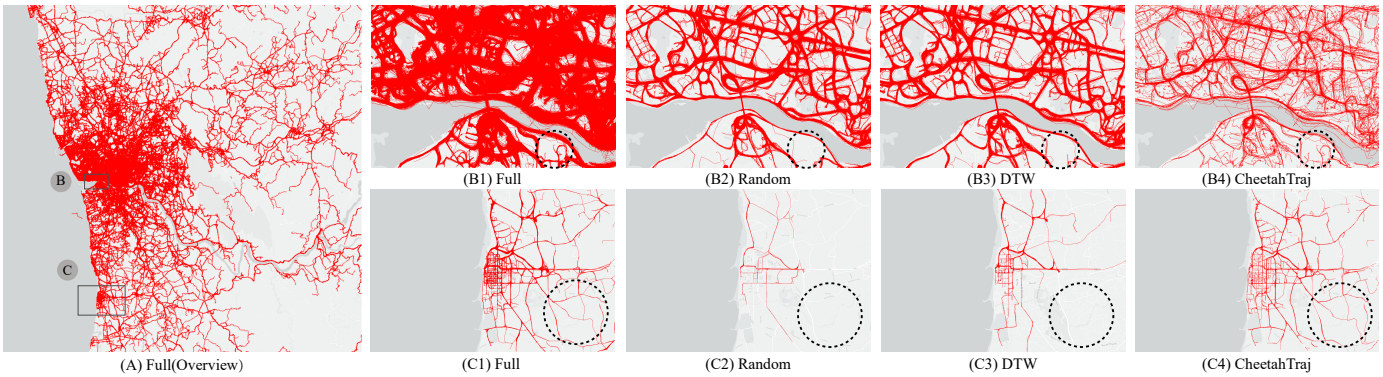


Fig. 8. Case study of the visualization quality of CheetahTraj for two detail regions.

coding. With the quality-aware VQGS$^+$ sampling algorithm, CheetahTraj also provides better visualization quality than Random and DTW by preserving the micro structures and overall layout of full visualization.

6.2 User Study

In this part, we conduct a user study to evaluate the quality of visualizations generated by different methods objectively.

Settings: We recruited 35 participants with 10 females, 25 males, aged 19 to 31 with a mean of 24.78 for the user study. The user study is conducted on the Porto and Shenzhen datasets, and four visualization methods are investigated, i.e., Full, Random, DTW and CheetahTraj. We manually select 22 *center points* in the two datasets and define 3 *visualization scales* including: large-scale region (with zoom level smaller than 13), middle-scale region (with zoom level between 13 and 15), small-scale region (with zoom level larger than 15). For each center point and at each visualization scale, we generate a *comparable visualization group*, which includes one visualization generated by each of the 4 methods. This results in 66 comparable groups (22 center points $\times$ 3 scales) and 264 visualization results (66 comparable groups $\times$ 4 visualizations).

We are interested in the visual quality and visual clutter of the visualizations, and hence designed three tasks for a comparable group:

- Task 1 (T1): rank the visualizations in a group from the highest visual quality to the lowest visual quality by 1st, 2nd, 3rd, and 4th.
- Task 2 (T2): rank the visualizations in a group from the least visual clutter to the most severe visual clutter by 1st, 2nd, 3rd, and 4th.
- Task 3 (T3): select the visualizations considered acceptable (multiple choices allowed) and choose the reason for the visualizations considered unacceptable. We provide three reasons including "severe visual clutter", "poor visual quality" and "others".

The user study system is a web-based platform, in which all visualizations are displayed with a resolution of 450*300.

User study procedure: When the participants enter the user study system, they are given a tutorial on how to conduct the tasks to get familiar with the interface and tasks. For each participant, we randomly select 16 comparable groups. Thus, we obtain $35 \times 16 = 560$ results for each task. For each comparable group, the 4 visualizations (*without specifying generated by which method*) in it are displayed on the same web-page and a participant is required to perform task T1, T2 and T3 by inspecting them.

Result analysis: Figure 9(A) reports the visual quality ranking of the 4 methods in T1. The results show that Full ranks the 1st in most cases while CheetahTraj usually ranks 1st or 2nd, i.e., the percentage of CheetahTraj ranks top-2 among 4 methods is 88.9%. In contrast, DTW and Random rank 3rd and 4th at most times. This suggests that the visualizations generated by CheetahTraj are more appealing to the participants than DTW and Random. We also observed that the participants tend to rank CheetahTraj before Full for large-scale regions with many trajectories, and the other way for smaller regions.

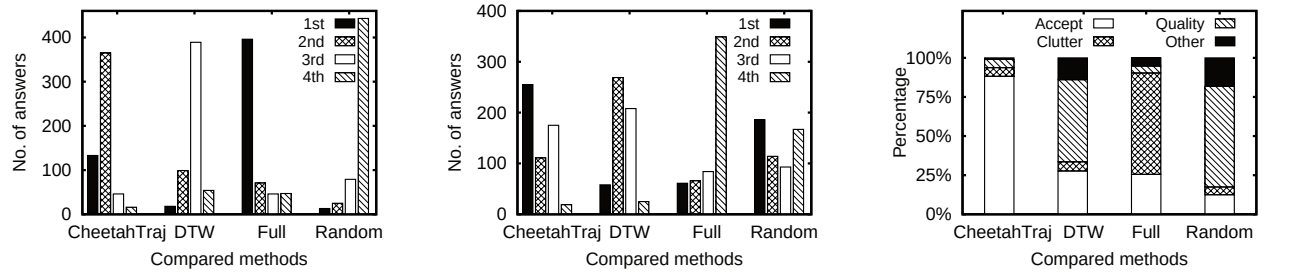
Figure 9(B) reports the anti-visual clutter ranking of the 4 methods in T2. The results show that visual clutter is most severe for Full, ranking 4th in most cases (349 over 560). With sampling, DTW usually rank 2nd and 3rd but Random ranks 4th for a considerable number of times as it tends to create clutter in the dense regions. CheetahTraj is the most successful in reducing visual clutter, ranking 1st in 255 out of the 560 cases and ranking 4th for only 19 cases.

We report the frequency each of the 4 methods is selected as acceptable and why a method is not selected in T3 using bar chart in Figure 9(C). Each column corresponds to a method, and from bottom to top, the lengths of the bars indicate the percentage of participants choosing "acceptable", "not acceptable due to visual clutter", "not acceptable due to poor visual quality" and "not acceptable for other reasons". The results show that CheetahTraj is regarded acceptable for about 88.2% of the cases, and the other methods have significantly lower acceptance rate than CheetahTraj. Specifically, DTW and Random have low acceptance rate mainly due to poor visual quality while Full suffers from severe visual clutter.

6.3 Quantitative Evaluation

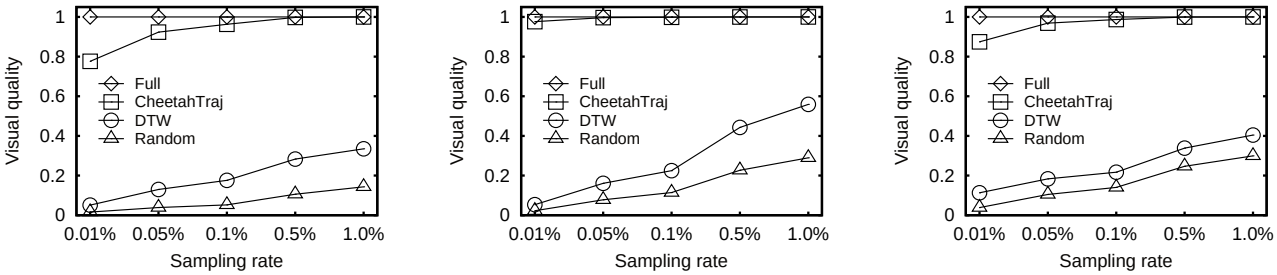
In this part, we quantitatively evaluate the visual quality and efficiency of CheetahTraj on the three real-world trajectory datasets.

Visual quality: Figure 10 reports the visualization quality (as defined in Equation (1)) of the methods under different sampling rate. We consider the entire region in each



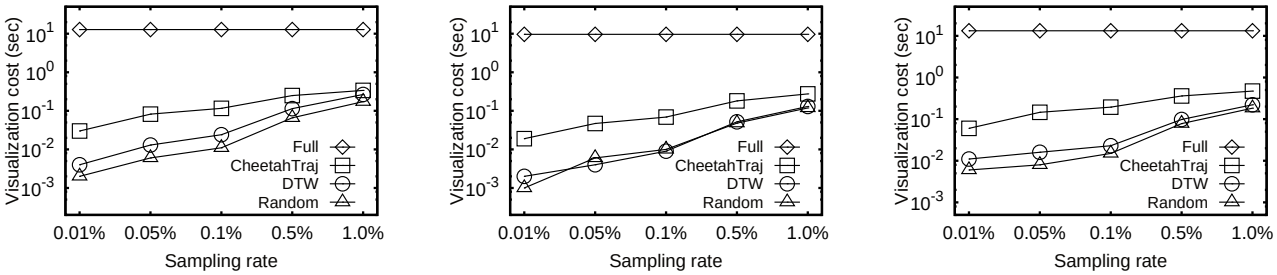
(A) T1: visual quality study (B) T2: visual clutter study (C) T3: acceptable visualization study

Fig. 9. User study results of different visualization methods



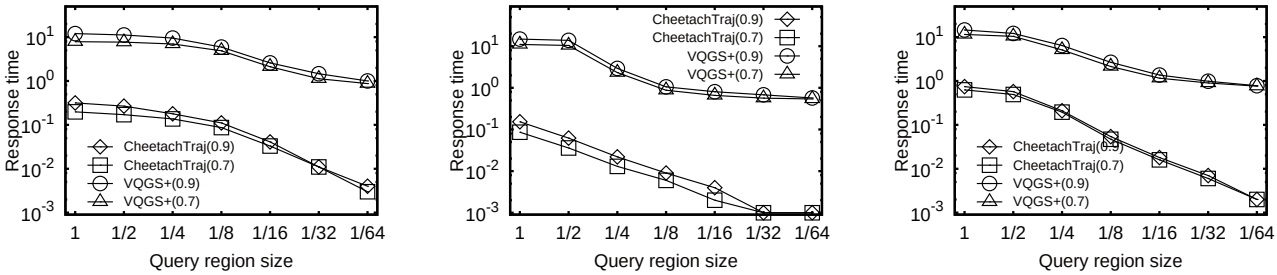
(A) Porto (B) Shenzhen (C) Chengdu

Fig. 10. Effect of varying sampling rate visual quality.



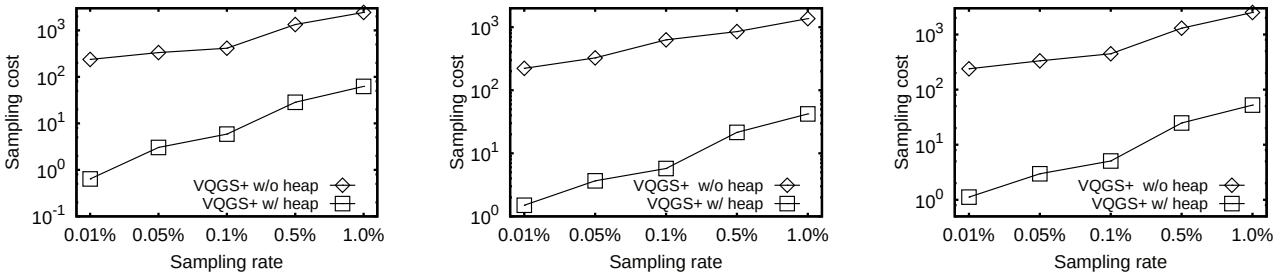
(A) Porto (B) Shenzhen (C) Chengdu

Fig. 11. Effect of sampling rate on visualization cost.



(A) Porto (B) Shenzhen (C) Chengdu

Fig. 12. Effect of region size on end-top-end response time.



(A) Porto (B) Shenzhen (C) Chengdu

Fig. 13. Effect of sampling rate on the sampling cost of VQGS$^+$ with/without optimization

dataset for this experiment. The results show that our proposal CheetahTraj achieves significantly higher quality than Random and DTW under the same sampling rate. This is because the sampling algorithm VQGS and VQGS$^+$ in the CheetahTraj framework are designed with explicit considerations for visual quality. Specifically, the quality of CheetahTraj approaches 1 when the sampling rate is still less than 1% for all 3 datasets. DTW has a higher quality than Random because it considers the diversity of trajectories.

In Figure 11, we report the visualization cost (i.e., the wall clock time to generate visualization result using the sampled trajectories) for the methods under different sampling rate. We still consider the entire region in this experiment and the visualization time of Full (which does not change with sampling rate) is included at the top of each figure for reference. The results show that all sampling methods achieve significantly shorter visualization cost than Full, and the speedup can be 1 to 4 orders of magnitude. This confirms our observation that sampling is effective in improving visualization efficiency. Under the same sampling rate, our CheetahTraj takes slightly longer visualization time than Random and DTW because CheetahTraj tends to select long trajectories for quality maximization. Combining Figure 10 and 11, we can conclude that CheetahTraj can achieve high visualization quality with short visualization latency.

Efficiency of CheetahTraj**:** We evaluate the *response time* of our CheetahTraj framework under different quality guarantees and region sizes in Figure 12. The response time of CheetahTraj is the end-to-end time for generating visualization for a selected region, which includes querying the CheetahTraj index and computing the visualization. For comparison, we also plot the response time of VQGS$^+$ (with $\delta = 8$), which uses on-line sampling instead of querying the index in CheetahTraj framework. We constrain the regions to be rectangles with a constant height/width ratio and measure the size of a region by dividing its height over the height of the entire region. For each region size, we report the average response time of three typical regions, i.e., a dense region, a sparse region and a medium region. The results show that CheetahTraj achieves a short response time (less than 1 second in all cases and 0.2 second for most cases) for different region sizes and quality guarantees. VQGS$^+$ is 1 to 2 orders of magnitude slower than CheetahTraj and takes at most 14.802 seconds in all cases. These results show that VQGS$^+$ cannot support interactive visual exploration and the InvQuad-tree index in CheetahTraj is effective in improving efficiency. In addition, the response time decreases rapidly when the region size shrinks as there are fewer trajectories in a smaller region. However, the response time required to achieve a high quality (e.g., 0.9) is not significantly longer than a low quality (e.g., 0.7) as quality improves quickly with the number of sampled trajectories as a shown in Figure 10.

Effect of heap-based lazy computation: In Figure 13, we report the running time of VQGS$^+$ with and without the heap-based lazy computation. The results show that the heap-based optimization reduces the running time of VQGS$^+$ around 2 orders of magnitude. For the sampling rates we considered, VQGS$^+$ runs efficiently and can finish within 1 second for the entire dataset.

TABLE 3
The cost of InvQuad-tree index

Dataset (size)	Height	Building time	Memory size
Porto (1.44G)	13	526.390s	3.65GB
Shenzhen (1.02G)	13	435.291s	3.12GB
Chengdu (1.49G)	13	454.151s	3.71GB

InvQuad**-tree index cost evaluation:** We report the building time and memory cost of the InvQuad-tree index in Table 3. For all three datasets, it takes less than 10 minutes to build the InvQuad index with a height of 13. The memory cost of the InvQuad index in the last column is also comparable with the size of the raw data shown in the first column.

7 Conclusions

This paper presents the CheetahTraj framework, which achieves high visual quality and low visualization latency for large-scale trajectory datasets. CheetahTraj provides guaranteed visual quality in trajectory sampling by formulating a quality optimal sampling problem and developing effective solutions including VQGS and VQGS. Low visualization latency is achieved with the InvQuad-tree index, which allows to use the sampling results computed offline. Experiment results show that CheetahTraj consistently provides high quality visualization in different cases and its visualization time is orders of magnitude shorter than full visualization.

Acknowledgments

The authors would like to thank...

References

[1] Z. Wang, T. Ye, M. Lu, X. Yuan, H. Qu, J. Yuan, and Q. Wu, "Visual exploration of sparse traffic trajectory data," *TVCG*, vol. 20, no. 12, pp. 1813–1822, 2014.
[2] B. Tang, M. L. Yiu, K. Mouratidis, and K. Wang, "Efficient motif discovery in spatial trajectories using discrete fréchet distance," in *EDBT*, 2017.
[3] Y. Zheng and X. Xie, "Learning travel recommendations from user-generated gps traces," *TIST*, vol. 2, no. 1, pp. 1–29, 2011.
[4] D. Liu, D. Weng, Y. Li, J. Bao, Y. Zheng, H. Qu, and Y. Wu, "Smartadp: Visual analytics of large-scale taxi trajectories for selecting billboard locations," *TVCG*, vol. 23, no. 1, pp. 1–10, 2016.
[5] V. W. Zheng, Y. Zheng, X. Xie, and Q. Yang, "Collaborative location and activity recommendations with gps history data," in *WWW*, 2010, pp. 1029–1038.
[6] W. Chen, F. Guo, and F.-Y. Wang, "A survey of traffic data visualization," *TITS*, vol. 16, no. 6, pp. 2970–2984, 2015.
[7] G. L. Andrienko, N. V. Andrienko, W. Chen, R. Maciejewski, and Y. Zhao, "Visual analytics of mobility and transportation: State of the art and further research directions," *TITS*, vol. 18, no. 8, pp. 2232–2249, 2017.
[8] G. L. Andrienko, N. V. Andrienko, S. M. Drucker, J. Fekete, D. Fisher, S. Idreos, T. Kraska, G. Li, K. Ma, J. D. Mackinlay, A. Oulasvirta, T. Schreck, H. Schumann, M. Stonebraker, D. Auber, N. Bikakis, P. K. Chrysanthis, G. Papastefanatos, and M. A. Sharaf, "Big data visualization and analytics: Future research challenges and emerging applications," in *EDBT/ICDT joint conference*, ser. CEUR Workshop Proceedings, vol. 2578. CEUR-WS.org, 2020.
[9] "Shenzhen dataset," http://jtys.sz.gov.cn/, 2020.
[10] B. Shneiderman, "Response time and display rate in human performance with computers," *ACM Computing Surveys*, vol. 16, no. 3, pp. 265–285, 1984.

[11] B. C. Kwon, J. Verma, P. J. Haas, and C. Demiralp, "Sampling for scalable visual analytics," *IEEE Computer Graphics and Applications*, vol. 37, no. 1, pp. 100–108, 2017.
[12] X. Qin, Y. Luo, N. Tang, and G. Li, "Making data visualization more efficient and effective: A survey," *The VLDB Journal*, vol. 29, no. 1, pp. 93–117, 2020.
[13] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang, "Sample + seek: Approximating aggregates with distribution precision guarantee," in *SIGMOD*, 2016, pp. 679–694.
[14] A. Kim, E. Blais, A. G. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld, "Rapid sampling for visualizations with ordering guarantees," *PVLDB*, vol. 8, no. 5, pp. 521–532, 2015.
[15] Y. Park, M. Cafarella, and B. Mozafari, "Visualization-aware sampling for very large databases," in *ICDE*, 2016, pp. 755–766.
[16] L. Battle, M. Stonebraker, and R. Chang, "Dynamic reduction of query result sets for interactive visualizaton," in *IEEE BigData*, 2013, pp. 1–8.
[17] O. Borcan, "Improving visualization of trajectories by dataset reduction and line simplification," Master's thesis, 2012.
[18] S. Liu, J. Pu, Q. Luo, H. Qu, L. M. Ni, and R. Krishnan, "Vait: A visual analytics system for metropolitan transportation," *TITS*, vol. 14, no. 4, pp. 1586–1596, 2013.
[19] X. Yang, Z. Zhao, and S. Lu, "Exploring spatial-temporal patterns of urban human mobility hotspots," *Sustainability*, vol. 8, no. 7, p. 674, 2016.
[20] J. Chae, D. Thom, Y. Jang, S. Kim, T. Ertl, and D. S. Ebert, "Public behavior response analysis in disaster events utilizing visual analytics of microblog data," *Computers & Graphics*, vol. 38, pp. 51–60, 2014.
[21] G. Borruso, "Network density estimation: a gis approach for analysing point patterns in a network space," *Transactions in GIS*, vol. 12, no. 3, pp. 377–402, 2008.
[22] D. Guo, "Flow mapping and multivariate visualization of large spatial interaction data," *TVCG*, vol. 15, no. 6, pp. 1041–1048, 2009.
[23] T. Von Landesberger, F. Brodkorb, P. Roskosch, N. Andrienko, G. Andrienko, and A. Kerren, "Mobilitygraphs: Visual analysis of mass mobility dynamics via spatio-temporal graphs and clustering," *TVCG*, vol. 22, no. 1, pp. 11–20, 2015.
[24] H. Guo, Z. Wang, B. Yu, H. Zhao, and X. Yuan, "Tripvista: Triple perspective visual trajectory analytics and its application on microscopic traffic data at a road intersection," in *IEEE Pacific Visualization Symposium*, 2011, pp. 163–170.
[25] C. Hurter, B. Tissoires, and S. Conversy, "Fromdady: Spreading aircraft trajectories across views to support iterative queries," *TVCG*, vol. 15, no. 6, pp. 1017–1024, 2009.
[26] N. Ferreira, J. Poco, H. T. Vo, J. Freire, and C. T. Silva, "Visual exploration of big spatio-temporal urban data: A study of new york city taxi trips," *TVCG*, vol. 19, no. 12, pp. 2149–2158, 2013.
[27] S.-M. Chan, L. Xiao, J. Gerth, and P. Hanrahan, "Maintaining interactivity while exploring massive time series," in *IEEE Symposium on Visual Analytics Science and Technology*, 2008, pp. 59–66.
[28] C. Yang, Y. Zhang, B. Tang, and M. Zhu, "Vaite: A visualization-assisted interactive big urban trajectory data exploration system," in *ICDE*, 2019, pp. 2036–2039.
[29] L. Dong, Q. Bai, T. Kim, T. Chen, W. Liu, and C. Li, "Marviq: Quality-aware geospatial visualization of range-selection queries using materialization," in *SIGMOD*, 2020, pp. 67–82.
[30] J. Wood, J. Dykes, and A. Slingsby, "Visualisation of origins, destinations and flows with od maps," *The Cartographic Journal*, vol. 47, no. 2, pp. 117–129, 2010.
[31] G. Andrienko and N. Andrienko, "Spatio-temporal aggregation for visual analysis of movements," in *IEEE symposium on visual analytics science and technology*, 2008, pp. 51–58.
[32] N. Adrienko and G. Adrienko, "Spatial generalization and aggregation of massive movement data," *TVCG*, vol. 17, no. 2, pp. 205–219, 2010.
[33] T. Rapp, C. Peters, and C. Dachsbacher, "Void-and-cluster sampling of large scattered data and trajectories," *TVCG*, vol. 26, no. 1, pp. 780–789, 2019.
[34] H. Chen, W. Chen, H. Mei, Z. Liu, K. Zhou, W. Chen, W. Gu, and K.-L. Ma, "Visual abstraction and exploration of multi-class scatterplots," *TVCG*, vol. 20, no. 12, pp. 1683–1692, 2014.
[35] J. Yu and M. Sarwat, "Turbocharging geospatial visualization dashboards via a materialized sampling cube approach," in *ICDE*, 2020, pp. 1165–1176.
[36] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. T. Shen, "Trajectory simplification: an experimental study and quality analysis," *VLDB*, vol. 11, no. 9, pp. 934–946, 2018.
[37] M. van Kreveld, M. Löffler, and L. Wiratma, "On Optimal Polyline Simplification using the Hausdorff and Fréchet Distance," *arXiv e-prints*, p. arXiv:1803.03550, Mar. 2018.
[38] K. Vrotsou, H. Janetzko, C. Navarra, G. Fuchs, D. Spretke, F. Mansmann, N. Andrienko, and G. Andrienko, "Simplifly: A methodology for simplification and thematic enhancement of trajectories," *TVCG*, vol. 21, no. 1, pp. 107–121, 2015.
[39] H. Piringer, C. Tominski, P. Muigg, and W. Berger, "A multithreading architecture to support interactive visual exploration," *TVCG*, vol. 15, no. 6, pp. 1113–1120, 2009.
[40] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.
[41] S. Fujishige, *Submodular functions and optimization*, 2005.
[42] "Google map," https://www.google.com/maps/preview, 2020.
[43] R. Krüger, D. Thom, M. Wörner, H. Bosch, and T. Ertl, "Trajectorylenses–a set-based filtering and exploration technique for long-term trajectory data," in *Computer Graphics Forum*, vol. 32, no. 3pt4. Wiley Online Library, 2013, pp. 451–460.
[44] "Fitts' law: Tracking users' clicks," https://www.interaction-design.org/literature/article/fitts-law-tracking-users-clicks, 2020.
[45] "Porto dataset," http://www.geolink.pt/ecmlpkdd2015-challenge/dataset.html, 2020.
[46] "Chengdu dataset," https://outreach.didichuxing.com/app-vue/dataList, 2020.
[47] "Unfolding maps," http://unfoldingmaps.org/, 2020.
[48] "The open-source graphical library," https://processing.org, 2020.
[49] "Source code," https://github.com/ChrisZcu/CheetahTraj, 2020.

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.