A Declarative Grammar for Interactive Trajectory Visualization: Interaction as First-Class Component

Shifu Chen*
School of Software Technology,
Zhejiang University

Xiaodan Miao†
School of Software Technology,
Zhejiang University

Dazhen Deng‡
School of Software Technology
Zhejiang University

Zikun Deng§
School of Software Engineering,
South China University of Technology

Di Weng¶
School of Software Technology,
Zhejiang University

Yingcai Wu‖
State Key Lab of CAD&CG,
Zhejiang University

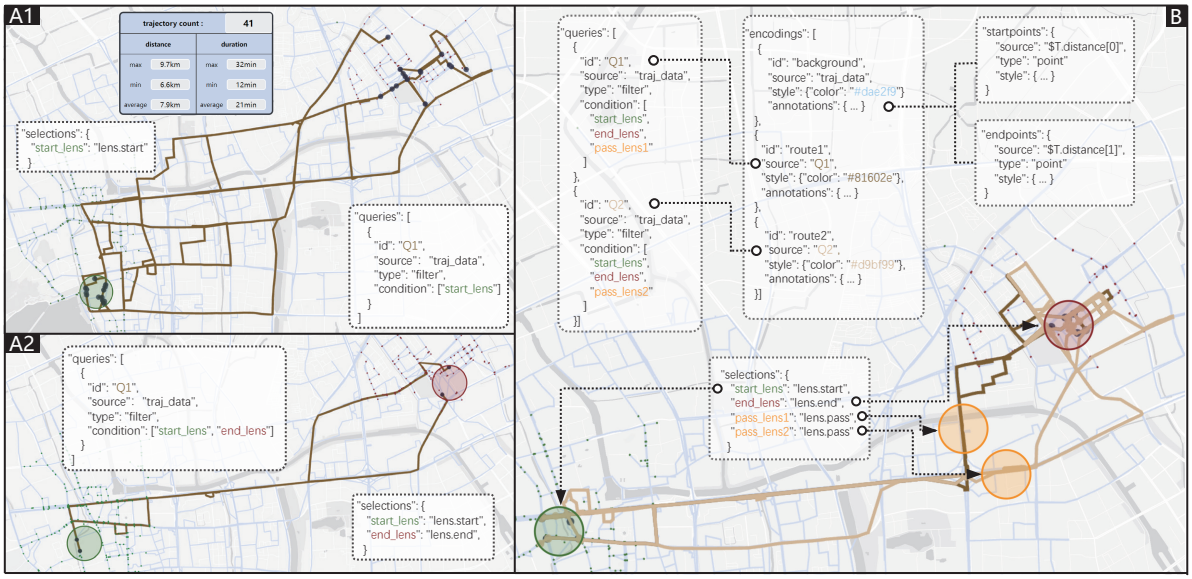

Figure 1: Using TrajGram to create trajectory visualization with lens interactions. First, display the trajectories and add a lens (*start_lens*) for filtering by starting points (A1), then add another lens (*end_lens*) for intersection filtering by ending points (A2). Further, two sets of filter lenses share the *start_lens* and *end_lens*, each having a unique lens (*pass_lens1*, *pass_lens2*) for filtering trajectories based on their path choices (A3). The selection-query-encoding chains link each component's declaration.

Abstract

Trajectory visualizations play a crucial role in urban computing, supporting tasks from analyzing road networks and vehicle movements to informing transport policy. However, creating such visualizations faces usability and flexibility limitations in trajectory visualization scenarios: limited usability due to insufficient trajectory-specific specialization, and limited flexibility due to tightly-coupled interactions that are not treated as first-class components. To address these challenges, we first conduct a systematic review of prior work and define a comprehensive design space for interactive trajectory visualization across three key dimensions: transformation, display, and interaction. We then present TrajGram, a declarative grammar built upon this design space that streamlines the creation of diverse interactive trajectory visualizations. TrajGram enables rapid prototyping with rich interactions grounded in real-world traffic analysis needs, while remaining extensible for complex scenarios. We demonstrate the effectiveness, usability, and learning curve of TrajGram through two usage scenarios and a user study.

Index Terms: Trajectory Visualization, Urban Visual Analytics, Declarative Visualization

*e-mail: shifu.chen@outlook.com
†e-mail: xiaodanmiao@zju.edu.cn
‡e-mail: dengdazhen@zju.edu.cn (co-corresponding author)
§e-mail: zkdeng@scut.edu.cn
¶e-mail: dweng@zju.edu.cn (co-corresponding author)
‖e-mail: ycwu@zju.edu.cn

1 Introduction

In the rapidly evolving realm of urban computing, trajectory visualization has emerged as a pivotal technique, serving a broad spectrum of applications, including movement analysis [7] and the design of transportation networks and public transit systems [19, 63]. The effective visualization of trajectories allows for the distillation of complex mobility data into actionable insights, which influences the decision-making processes of diverse urban planning and optimization scenarios that involve the analysis of trajectories.

However, building interactive trajectory visualizations remains a challenging task, primarily due to the interconnected nature of data transformation, visual representation, and user interaction. Trajectory visualizations can take diverse forms—from simple polylines to complex aggregated representations like heatmaps—each requiring different data processing approaches. Meanwhile, interactive operations often trigger display changes that necessitate underlying data transformations: selecting trajectories in one view may require

aggregating data for another view, filtering by time may need recomputing spatial distributions, and changing visual encodings may demand restructuring the data pipeline. This intricate relationship between interaction, transformation, and display creates implementation complexity where interactions cascade through the entire system, making it hard to develop cohesive interactive experiences.

While modern geospatial libraries like deck.gl [1] and Mapbox.gl.js [2] excel at rendering polylines on map as trajectories, they provide limited support for incorporating interactions. As an alternative, libraries like D3.js [13] offer flexible interaction capabilities, but they require developers to manually orchestrate the relationships between transformation, display, and interaction components. This manual coordination demands significant coding effort and meticulous adjustment, which impedes swift and efficient creation and poses challenges to rapid deployment and iterative design.

In this study, we propose a declarative grammar tailored for interactive trajectory visualization to address this inconvenience. Our approach is inspired by the success of declarative grammars in the visualization community, such as Vega-Lite [47] for chart rendering and Rigel [15] for data transformation, which have proven effective for rapidly prototyping interactive interfaces. By using such grammars, developers can specify complex visualizations and interactions through concise configurations, avoiding low-level event-handling code and lowering the barrier to entry.

However, existing declarative grammars do not treat interaction as first-class components, which restricts their ability to capture complex interactive behaviors in trajectory visualization [64]. Additionally, trajectory data has unique characteristics and visualization needs that existing grammars do not fully support. This motivates the design of a domain-specific grammar that treats interaction as first-class components and is optimized for trajectory visualization.

To systematically understand how interaction relates to the other components, we conducted a comprehensive review of existing work and identified a design space comprising three key dimensions: 1) Transformation – manipulating trajectory data to support visualization; 2) Display – transforming trajectory data into visual representations that convey movement and multidimensional attributes; 3) Interaction – enabling user participation in modifying transformation and display components. Based on this, we present TrajGram, a domain-specific grammar for web-based interactive trajectory visualizations. TrajGram abstracts away low-level implementation details, allowing both novice and experienced users to focus on crafting interactive visualizations. Our contributions are as follows:

- ◇ A structured design space for interactive trajectory visualization, grounded in an extensive literature review.
- ◇ TrajGram, a declarative grammar that treats interaction as first-class components and supports rich interactions through coordinated encoding, interaction, and transformation components.
- ◇ Use cases that recreate prior visualization examples using TrajGram, demonstrating its usability and extensibility.

2 Related Work

We review prior work related to trajectory visualization creation.

2.1 Geospatial Visualization

Geospatial visualization enables the analysis of complex spatial relationships, patterns, and trends by illustrating them within their geographic context. Within the realm of geospatial visualization [16, 20], point-based visualizations [23, 32, 40, 61] represent individual data events as distinct markers, line-based visualizations are usually used to depict trajectories [26, 33], road networks [25, 35], or traffic flows [10], and region-based visualizations [53, 56, 62] aggregate data across areas to highlight spatial phenomena at a larger scale.

Trajectory visualization employs diverse representation strategies. Individual trajectories are commonly represented as polylines with visual attributes such as color, width, or texture encoding trajectory-specific data [24,41]. Graph-based representations [25] model trajectories as networks of connected nodes and edges, enabling structural analysis of movement patterns. Density-based visualizations [5, 35] aggregate trajectories into heatmaps that reveal traffic density patterns and hotspots. Point-based annotations are frequently used to indicate key locations [6, 33] or movement directions [37, 54], providing contextual markers that enhance trajectory interpretation.

To support higher-level analysis, many techniques incorporate transformations. Systems aggregate trajectories by underlying road networks to produce road heatmap reflecting traffic patterns [5, 35], while tools like TrajRank [41] allow for custom route segmentation and aggregate trajectories within them to facilitate route ranking.

2.2 Interactive Analysis for Trajectory Data

Interactivity plays a key role in trajectory visualization, enabling users to selectively explore and analyze subsets of trajectory data through various tools and techniques.

Many systems adopt fixed-form selection tools for ease of use. For example, TrajectoryLenses [33] provides lens-based filters targeting trajectory start points, endpoints, and paths, while TrajRank [41] offers paired lenses for filtering and cropping trajectories between locations. Rectangular marquees are also common: Liu et al. [36] use them to filter trajectories intersecting a selected area, and Shamal et al. [5] extend this interaction for more targeted filtering tasks. Beyond fixed-form tools, free-form approaches have been explored. FromDaDy [27] supports brushing-based selection, and TripVista [24] enables sketch-based direction filtering at intersections.

While these systems offer a range of interaction designs, most are tightly coupled with specific implementations, limiting code reuse and making them difficult to adapt for new contexts. Hence,developers often need to build interactive trajectory visualizations from the ground up, resulting in high development costs. To address this, TrajGram introduces a declarative approach for building interactive trajectory visualizations. By allowing users to specify visualization and interaction logic through a high-level grammar, TrajGram streamlines prototyping and promotes reuse, enabling more efficient and iterative development.

2.3 Libraries and Grammars for Trajectory Visualization

Visualization libraries [1–3, 12, 13, 39, 46, 59] and declarative grammars [14, 29, 31, 39, 42–44, 47–49, 51, 52, 64, 68] have gained traction in the visualization community for providing structured frameworks to define visual representations and interactions, significantly accelerating the design and implementation process.

Geospatial libraries [1–3] primarily focus on at trajectory rendering rather than the integration of interaction, transformation, and representation. General-purpose libraries such as D3.js [13] can accomplish complex interactive logic and support sophisticated coordination, but their interactions are difficult to reuse and require extensive custom code to handle the cascading effects. Libra [64] enhances reusability, learnability, and usability for general visualization as a framework by treating interactions as first-class citizens, but still requires substantial implementation effort when dealing with trajectory visualization. Declarative grammars, rooted in declarative programming, focus on defining what a visualization should express rather than how it is rendered [30]. In visualization, this paradigm allows authors to specify visual encodings and interactions at a higher abstraction level, decoupling design intent from implementation logic. Vega and Vega-Lite [47, 48] exemplify this approach, supporting interactive visualizations with concise specifications.

However, general grammars are not always well-suited for trajectory visualizations. Their broad scope often limits their ability to handle the complex, domain-specific requirements of trajectory data, resulting in verbose and intricate specifications that require significant expertise to use effectively. In contrast, domain-specific

Table 1: Design space for interactive trajectory visualization, summarizing 21 surveyed papers across three dimensions: transformation (filtering, segmentation, aggregation), display (path-based, network-based, area-based), and interaction (spatial, temporal, attribute-based).

Papers	Transformation									Display			Interaction				
	Filter			Segment			Aggregate										
	🌐	🕒	</>	🌐	🕒	</>	🌐	🕒	</>	⇝	⑂	◎	📍	～	▨	🕒	</>
AL-Dohuki et. al. [5]	●	●	●	●	●	●	●	●	●	●		●			●	●	●
Liu et. al. [37]	●	●	●	●	●	●	●	●	●	●			●		●	●	●
FromDaDy [28]	●	●								●					●	●	
RoseTrajVis [4]	●	●	●						●	●						●	●
Krueger et. al. [32]	●	●	●		●			●		●					●		●
SemanticTraj [6]	●	●	●	●				●		●			●		●	●	●
Tominski et. al. [54]	●	●	●	●			●	●		●					●	●	●
TrajectoryLenses [33]	●	●	●					●		●					●		
TrajRank [41]	●	●	●	●			●			●					●	●	●
TripVista [24]	●	●	●				●	●		●				●		●	●
Liu et. al. [36]	●	●	●							●	●				●	●	●
SmartAdP [35]				●			●					●					
Wang et. al. [58]	●	●	●				●	●		●						●	●
Huang et. al. [26]	●	●	●	●						●		●	●		●	●	●
VATT [17]	●	●	●	●			●					●				●	●
Wang et. al. [55]	●	●	●	●			●					●		●			
Zhou et. al. [67]				●			●				●						
Daae et. al. [18]				●				●			●						
VAIT [38]	●	●	●	●			●			●		●				●	
Roeland et. al. [50]	●	●	●	●			●	●				●			●	●	●
Andrienko et. al. [9]				●			●				●					●	

🌐 Spatial 🕒 Temporal </> Attribute ⇝ Path-based ⑂ Network-based ◎ Area-based 📍 Point ～ Line ▨ Area

declarative grammars [29, 31, 42, 43, 51] trade generality for simplicity, enabling more accessible authoring of complex visualizations within targeted domains. Inspired by this trend and building upon Libra [64]'s efforts at the library level, we propose a declarative grammar specifically designed for trajectory visualization that elevates interactions as first-class components, which simplifies the creation of complex trajectory visualization with interactions.

3 Design Space

To ground our declarative grammar, we first establish a design space for trajectory visualization. Although trajectory visualization has been studied in urban analytics [20, 66] and traffic visualization [8, 16], existing literature predominantly focuses on visual forms (e.g., line-based, network-based or area-based visualizations) while the underlying transformation processes and interaction mechanisms are frequently underexplored. To address this gap, we conducted a comprehensive survey of prior work to synthesize a focused design space that captures these underexamined aspects.

3.1 Methodology

To ensure consistent screening and coding of prior work, we first define the scope of interactive trajectory visualizations analyzed in this section: a trajectory describes the movement of an object through geographic space, typically represented as a sequence of time-ordered points [65], each with spatial (location), temporal (timestamp), and contextual (e.g., road ID) information. Given their multi-point structure and geospatial nature, trajectories can be visualized on maps as polylines showing individual paths, networks abstracting common routes and flows, or area-based visualizations revealing spatial patterns through statistical summaries. When such visualizations incorporate interactive features, we refer to them as **interactive trajectory visualizations**.

Guided by the above definition, we surveyed published papers that use map-based approaches to visualize trajectory data. We focused on innovative techniques, when encountering papers with similar visual forms or interactions, we retained only one representative paper to ensure distinct innovations. In total, we collected 21 papers, primarily from 3 journals (IEEE TVCG, JOV, IEEE TITS) and 3 conferences (IEEE VIS, EuroVis, PacificVis), from 2009 to 2025.

Based on these seed papers, we established a preliminary design space. Following the standard pipeline for interactive visualization [21]—data transformation, visual representation, and interaction—we derived three key components: transformation, display, and interaction.

We then expanded our analysis by incorporating additional examples cited in the seed papers, as well as visualizations from online libraries, open-source projects, and GIS platforms [22]. For each visualization, we analyzed its operations across 3 components. When finding new functions that did not fit existing categories, we revised the component definitions accordingly. Through multiple iterations, we refined and finalized the design space.

3.2 Taxonomy

Following the methodology described in subsection 3.1, we organized our findings into three key dimensions: **transformation**, **display**, and **interaction**, as summarized in Table 1.

3.2.1 Transformation

Transformation encompasses operations that process trajectory collections to extract meaningful information. We identify three main types: *filtering*, *segmentation*, and *aggregation*.

These transformations operate on three fundamental basis types: *spatial basis*, using geographic or coordinate-based criteria (directions, regions, boundaries, proximity); *temporal basis*, employing time-related criteria (intervals, durations, temporal patterns); and *attribute-based basis*, leveraging trajectory properties or metadata (speed, direction, semantic labels, user characteristics).

Filtering retains or discards entire trajectories based on specified criteria. Spatial filtering can select trajectories by origins, destinations, or routes through specific regions [33]. Temporal filtering retains trajectories within certain time periods or exceeding duration thresholds [32,33]. Attribute-based filtering constrains by properties such as average speed [4] or direction [24].

Segmentation divides trajectories into sub-trajectories according to spatial, temporal, or attribute-based boundaries. Spatial segmentation extracts portions within designated regions [5] or segments based on traversed roads [35,37]. Temporal segmentation isolates segments within specific time windows [32]. Attribute-based segmentation identifies portions with particular characteristics, such as exceeding speed limits [6].

Aggregation combines multiple trajectories or segments into composite representations through binning. Spatial aggregation bins data into geographic elements, such as road-based heatmaps [5,35] or grid-based density visualizations [5,18]. Temporal aggregation bins trajectories into time intervals [4,32] to reveal temporal patterns. Attribute-based aggregation groups trajectories by shared characteristics [4,5,37] such as speed or transportation mode.

3.2.2 Display

Display refers to fundamental approaches for visualizing trajectory data in map-based contexts. We focus on map-based visualizations since non-map-based representations fall outside our scope. We examine three complementary perspectives: representation types, visual channels encoding trajectory attributes, and layout algorithms optimizing spatial arrangement:

Path-based display visualizes trajectories as polylines, typically used for analyzing specific trajectories and comparing routes. It leverages polyline visual channels to encode trajectory attributes.

Network-based display transforms trajectory data into node-edge structures, including origin-destination (OD) matrices and road network visualizations. It utilizes node and edge visual channels, with representative layout algorithms such as *edge bundling* to reduce visual clutter.

Area-based display employs spatial aggregation to reveal movement intensity and distribution. Heatmaps [5] show density as color gradients across geographic grids. Representative algorithms include *kernel density estimation* for generating continuous density surfaces.

3.2.3 Interaction

Interaction techniques in trajectory visualization exhibit tremendous diversity in form, from sketching [24] to lenses [33] to rectangular marquee tools [5], plus general interface elements like sliders, switchers, or chat panels, with potential for multimodal or collaborative interactions.

This diversity makes comprehensive categorization infeasible, and such categorization is counterproductive for grammar design since users naturally classify interaction forms differently. Hence, rather than categorizing by form, we focus on what parameters an interaction actually gives for subsequent computation. These parameters, crucial for grammar construction, are relatively stable across three primary types:

Spatial interaction inputs geometric parameters such as points, lines, or areas for subsequent spatial computations. Point-based interaction uses the mouse position (a **point**) for clicking or hovering on trajectories. Line-based interaction leverages user-drawn **lines** to filter or highlight trajectories based on path similarity. Area-based interaction enables users to define a geographic **area** to trigger region-based transformations. **Temporal interaction** inputs temporal parameters such as time ranges, duration values, or temporal patterns for time-based computations. **Attribute-based interaction** inputs attribute values or ranges such as speed thresholds, direction values, or categorical attributes for attribute-specific computations.

4 Design Framework and Scope

In this section, we establish the theoretical foundation for our interactive trajectory visualization grammar based on systematic analysis of the design space. This framework bridges the gap between the conceptual design space presented in section 3 and the concrete grammar specification detailed in section 5.

4.1 Interactions as First-Class Components

Interactions as first-class citizens is proposed in Libra [64], where interactions may be named by variables, passed as arguments to procedures, returned as results of procedures, and included in data structures. In our declarative grammar, treating interactions as first-class components means that interactions are declared independently as reusable components within the data pipeline, at the same level as reusable visualizations in the declarative specification. Following this, we treat interactions as independent components in our framework, establishing an **interaction → transformation → display** pipeline. This approach establishes a clear interaction paradigm where user interactions trigger data transformations that are immediately reflected through updated visual encodings. While interaction can be extended to encompass broader event types at the library—such as background data updates or even events that modify visualization configurations—our grammar design focuses on mouse-based events. By maintaining this structured flow, we ensure consistent behavior across different visualization types while reducing system complexity.

4.2 Grammar Scope

Our framework establishes a foundational architecture for coordinating transformation, interaction, and display components with interaction as first-class citizens. Rather than pursuing comprehensive coverage, we implement representative operations within each component to demonstrate the framework's effectiveness, with the design allowing for future extensibility to additional use cases.

Interaction. Given the breadth of possible interactions in trajectory visualization, our interaction design focuses primarily on spatial interactions, as temporal and attribute-based interactions are already well-established and easily integrated through existing visualization libraries such as ECharts [11]. Therefore, our interaction component emphasizes map-based interactions that are anchored to interactive geographic elements, which serve as the foundation for user engagement with trajectory visualizations. This focus allows us to simplify the interaction component into a focused **selection** that concentrates on filter through direct manipulation of geographic elements.

Transformation. We primarily focus on trajectory filtering operations, which represent the most fundamental data manipulation in trajectory visualization. Additionally, we support basic segmentation and aggregation operations commonly required for trajectory analysis. These operations provide sufficient coverage to demonstrate the coordination mechanisms.

Display. We focus on two representative visualization types: common polyline representation and specialized road heatmap visualization. Polyline representation serves as the fundamental form where trajectory point coordinates directly determine path positions, while road heatmap demonstrates more complex scenarios requiring data transformation during value assignment. These two types effectively illustrate different complexity levels in the display pipeline while covering the most essential trajectory visualization needs.

4.3 Data Format

Trajectory data used in visualization include two types of attributes: trajectory-level attributes, such as average speed or travel distance, and point-level attributes, such as instantaneous speed. We define a **standard input format** where each trajectory is a list item containing overall attributes and a sequence of points. Each point records its location, time, and related attributes. For faster positioning, we pre-compute each point's spatial and temporal ratios relative to the start, along with trajectoires' full travel distance and duration.

Additionally, basic map information can be imported as **container data** to define geographic elements that serve as containers: road network data can be input as road segments, each with a unique identifier and geometric path, plus optional attributes like speed limits or road names; area division data can be input as GeoJSON files containing polygon geometries that define area boundaries and centroids, each with a unique identifier. When such container data is imported, trajectory points should be pre-processed with map matching to accelerate subsequent visualization operations, each point should contain container IDs in its attributes.

5 Grammar of TrajGram

Building on our systematic analysis of the design space and the identified principles for display and interaction, we formalize a grammar structured into three parts: the general specification, common grammar elements, and the core components of the specification. Lastly, we introduce a library extending grammar's capabilities.

5.1 General Specification

For our abstract grammar notation, we follow the convention of Ren et al. [45], where := denotes assignment, ? indicates optional elements, + represents one or more occurrences, and ∗ denotes zero or more. Throughout this section, each component defined in our formal notation corresponds directly to JSON structures in the practical specification, implemented as either objects or arrays based on their semantic role and cardinality constraints.

At the highest level, a **general specification** defines the overall structure for trajectory visualizations, implementing our design principles through a systematic decomposition of functionality. Formally, the general specification is defined as a tuple:

$$\textbf{general} := (map, data^{+}, selection^{*}, query^{*}, container^{*}, encoding^{+})$$

This specification comprises six core components that realize our design framework: **Map** defines the visual and interactive foundation for displaying trajectories, including settings such as the initial zoom level and center point of the map. Standard interactions—zooming, panning, and dragging—are supported by default. **Data** specifies the trajectory datasets to be imported, following the standard format described in subsection 4.3. **Container**, **encoding**, **selection**, and **query** realize the design established in section 4.

Following our established interaction paradigm, selections trigger queries that filter and transform trajectory data, while encodings translate both raw and processed data into visual presentations. This creates the selection → query → encoding pipeline that enables dynamic, interactive trajectory visualizations where user interactions with geographic elements directly influence the displayed results.

5.2 Common Grammar Element

Our grammar includes several elements that are universally applicable across components and maintain consistent function globally.

Iteration Characters. Trajectory data visualization requires distinct iteration mechanisms for different data types: trajectory data iteration operates at two levels—iterating through individual trajectories in a collection, and sequentially through points within each trajectory. Aggregated data iteration traverses container-based results,

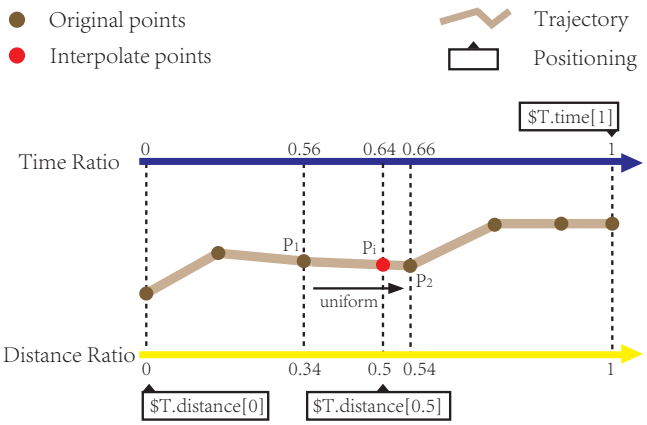

Figure 2: The expression $\$T.distance[0.5]$ specifies the midpoint P_i along a trajectory. Assuming uniform motion between two original points P_1 and P_2, the position and time of P_i are interpolated.

where containers can be imported road networks, area divisions, dynamically generated grid cells, or virtual networks abstracting spatial relationships into nodes and edges. For example, *$T* represents each trajectory within a collection, enabling trajectory-level operations; *$P* denotes each point within a trajectory, supporting point-level operations.

Built-in Functions. Common tasks such as calculating movement metrics, performing statistical analyses, and manipulating spatial data are fundamental to most analysis workflows. To reduce code complexity, we integrate commonly used built-in functions that apply operations across trajectory collections. For example, $distance(\$T)$ calculates the distance for each trajectory by summing distances between consecutive points.

Positioning determines the precise location of a conditional point along a trajectory. Since trajectories consist of discrete sampling points rather than continuous paths, locating points not explicitly stored in the data (e.g., a trajectory's midpoint) requires interpolation. We connect discrete points into a continuous representation across spatial, temporal, and attribute dimensions. For spatial and temporal estimation, we assume uniform motion between consecutive sampling points, as illustrated in Figure 2. For attributes, continuous values (e.g., numerical) are estimated via *linear interpolation* based on the distance between neighboring sampling points, while discrete values (e.g., categories) adopt the *nearest neighbor*'s value. This continuous representation is accessible through $\$T.distance$ and $\$T.time$ (Figure 2).

Predicates. Trajectory analysis requires selecting data subsets based on complex spatial, temporal, and attribute relationships beyond simple value comparisons. To enable users to express such conditions—involving geographic containment, temporal sequences, and multi-dimensional constraints—without extensive geometric and temporal calculation code, we incorporate predicates for geographic relationships (e.g., point-area containment, line-area intersection), temporal constraints (e.g., temporal order), and attribute conditions (e.g., value comparison). For example, $line.pass(area)$ determines if a trajectory intersects with a given geographic area.

5.3 Query Specification

The query specification encompasses a range of data transformation methods, and it is defined by a tuple:

$$query := (id, source, type, basis, set?)$$

where **Source** declares the target data; **Type** declares the transformation to apply—filtering, segmentation, or aggregation per the design

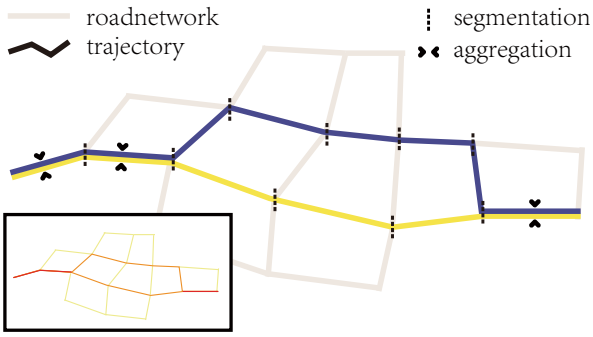

Figure 3: Segmentation and aggregation for road heatmap construction. Trajectories are first segmented by road network, then aggregated per segment (network edge). The heatmap reflects traffic flow—darker segments indicate higher trajectory density.

space; **Basis** defines detailed criteria for the transformation; and **Set** specifies additional attributes to attach to the results.

5.3.1 Filtering

For filtering, the *basis* parameter specifies constraints to select relevant trajectories. Following our design principle that interactions primarily target filtering, these constraints fall into two types:

Static constraints specify fixed criteria, including temporal (e.g., trajectories starting within a certain hour range) and attribute constraints (e.g., distance range). For example, *distance($T)>20km* retains only trajectories exceeding 20 kilometers.

Dynamic spatial constraints implement our selection-driven interaction paradigm by binding geographic elements with spatial predicates. These constraints reference geographic objects (regions, points) and apply spatial predicates (intersects, inside) to dynamically filter trajectories. For example, *$T.startpoint.inside(Area)* filters trajectories whose starting points fall within a specified area.

5.3.2 Segmentation

For segmentation transformations, the *basis* parameter defines the grouping methods that partition trajectory points within individual trajectories. The methods divide trajectory points into multiple groups, each containing a series of continuous points. A method can be a built-in function ($evenT(n)$ or $evenD(n)$) or an expression that defines grouping criteria. The expression processes each trajectory point according to the specified criteria, and continuous points satisfying the same criteria are grouped together. We also provide specialized methods, such as segmenting trajectories according to road networks, administrative area divisions, or regular grid cells.

The **set** parameter for segmentation adds new attributes to each trajectory segment produced from the grouping, such as transportation mode, speed statistics, or temporal characteristics. For example, to create a flow field visualization, trajectories are first segmented using a grid-based *basis* that groups trajectory points within spatial cells. *Set* then adds vector attributes calculated from each segment's start and end points, including the direction and magnitude.

5.3.3 Aggregation

For **aggregation** transformations, trajectory data or trajectory segments are aggregated into containers that serve as spatial or conceptual units for analysis. These containers can be either physically existing entities such as grid cells, administrative area divisions, or road network segments, or abstract constructs such as network nodes and edges in flow maps. We categorize containers into two types: those that represent internal properties of the container (e.g., traffic density within a road segment), and those that represent relationships between containers (e.g., flow volumes between regions).

The *basis* parameter defines the grouping methods that combine trajectory segments across multiple trajectories to transform them into different output types such as road segments, network edges and nodes, or grid cells. The grouping strategy can be based on spatial proximity, temporal overlap, or shared attributes of the trajectories.

The **set** parameter for aggregation determines what aggregated attributes are generated for the combined trajectory segments, such as summary statistics, flow measurements, or density calculations. For example, to construct a road network heatmap as shown in Figure 3, trajectories are first segmented by road segments using the **basis** parameter to define the road-based grouping criteria. Then, sub-trajectories corresponding to the same road segment are aggregated using **basis** to specify how segments should be grouped spatially. Finally, **set** adds aggregated attributes (e.g., traffic volume or average speed) to the results, which are then encoded with color.

5.4 Encoding

Encodings define the display method for all visual elements on the map. By default, elements on the map will move or change size as the user interacts with the map, such as zoom or pan. Each encoding scheme would retrieve data from its source and display it with a certain style on the map, which is defined by a tuple:

$$encoding := (id, type, source, style, annotation^*)$$

Type declares the input data type and the corresponding rendering type as an integrated specification, where the data structure and visualization method are inherently coupled. **Source** declares the data to visualize, which can be input data or the query results.

5.4.1 Encoding Style

Style declares the encodings of visual variables. Each visual variable is declared with an expression , linking the data field and the visual variable with mapping methods. In our encoding grammar, these variables are either static or gradient:

A **static expression** assigns the visual variable a constant value and can be applied universally across all representation types. For example, *linear(speed($T), [0, 10], [20, 60])* maps the average speed of each trajectory to a number using linear scaling between domain values *[0, 10]* and visual range *[20, 60]*. A **gradient expression**, however, is specifically designed for polyline representations, allowing for dynamic styling along the trajectory where visual attributes such as color or opacity change continuously in response to the properties of individual trajectory points. This gradient encoding capability is unique to path-based representations as they preserve the sequential structure of movement data, enabling the visualization of temporal or spatial variations that unfold along the trajectory path. For example, *gradient($P.speed, [0, 10], [#b35a00, #d6d632])* creates a color gradient that varies continuously along the trajectory based on point-level speed values, mapping speeds from 0 to 10 onto a color range from orange to yellow-green.

5.4.2 Additional Annotation

Annotation defines the additional shapes attached to visual elements. Similarly, annotation is defined by a tuple:

$$annotation := (id, type, source, style?)$$

where **type** declares the annotation's shape; **source** determines the specific location where the shape will be positioned; **style**, an optional attribute, adds visual variables to enhance the appearance of

the shape. We accommodate common design elements within our grammar, such as text, marker, and fundamental geometric shapes.

Furthermore, we provide specialized positioning capabilities for polyline representations that leverage their sequential trajectory structure: annotations' source can utilize the positioning methods (Figure 2) to determine their precise location along the trajectory path. Annotations can be oriented in two modes: default setting arranges annotations horizontally, with center point located at the position specified, while the **follow** mode, specifically designed for polylines, aligns the annotation's orientation along the polyline.

5.5 Selection

It is important to note that our selection grammar primarily focuses on spatial selection (map-based) within the design space. While temporal and attribute-based selections are common in multi-view coordinated trajectory visualizations, these other non-map visualizations often take various forms. Users can achieve temporal and attribute filtering by creating their own time-based or attribute-based views that pass filter conditions to **queries**' *basis*.

5.5.1 Definition

In our grammar, a selection comprises three fundamental elements that provide flexibility for defining custom selection behaviors:

$$selection := (type, predicate?, trigger?)$$

The **type** refers to the shape editor type, which refers to the mechanism for creating and modifying shapes on the map. It encapsulates both the geometric form and the interaction methods for shape manipulation. We integrated a set of general tools for drawing and editing points, lines, and areas. The spatial **predicate** defines the spatial relationship between the shape created by the shape editor and the trajectory data. In our grammar, we consider trajectories as lines and trajectory points as points in predicates. The **trigger** condition defines when the selection will occur. Each time a triggering event happens, the resulting shape will update. Then, all downstream components of this selection will be updated accordingly.

5.5.2 Simplified Usage

To simplify the grammar usage, we have made several design decisions. First, we simplify commonly used predicates for intuitive usage. For example, within area shapes, using *start* as a predicate represents *$T.distance[0].inside(Area)*, while *pass* represents *$T.pass(Area)*. Second, we offer intuitive naming conventions that reflect the interaction patterns, such as *mouse.click* and *mouse.hover* for different mouse events, while *lens.start* combines a draggable circular shape (lens) with a simplified spatial predicate (start). Third, we make spatial predicates and trigger conditions optional in the grammar, as many shape editors come with sensible defaults that cover the most common use cases.

5.5.3 Customed Usage

Developers have the flexibility to craft their own selection components through additional coding to create and edit shapes and equip them with spatial predicates to cater to unique scenarios.

For example, when creating a lasso selection tool, a developer needs to implement the drawing logic that captures mouse gestures and constructs a polygon shape, and override the geometric output method in the inherited class, then he can optionally set built-in default spatial predicates and trigger conditions for common use cases. Once the developer completes the implementation, the defined component becomes available for use within our framework by simply including its name in the "selections" field of the JSON specification, which activates the tool's functionality. For interactions that involve continuous drawing and modification, we recommend implementing toggle buttons to enable and disable edit mode, preventing conflicts with other direct manipulation operations. This approach preserves user creativity and ensures diverse interaction possibilities while eliminating the burden of maintaining complex downstream processing logic.

5.6 Library Support & Performance

To enhance user experience and expand capabilities for external visualization tools, TrajGram includes a library with a grammar parser and programming interfaces. The parser converts JSON specifications into component instances, and programming interfaces are used to retrieve and modify instance properties, while components emit signals when updated. This bidirectional communication enables synchronization with external systems. For additional customization, we provide standard classes to extend for new component types, which can then be declared within JSON configurations.

We conducted performance evaluations to quantify rendering efficiency across data scales. The experimental environment consisted of Google Chrome with Intel i7-12700K and RTX 4060Ti at 2560×1440 resolution. We used Hangzhou taxi trajectory data (average 288 points per trajectory, 1 million total / 6,563MB), sampled at five scales: 10, 100, 1K, 10K, and 100K trajectories. Each trajectory is rendered as a styled polyline with circles at its endpoints. The largest dataset caused browser memory overflow. We measure two types of rendering time: 1) initial rendering when data is first transferred to the WebGL buffer, and 2) refresh time after map interactions. For initial rendering (10 trials per size), mean times were 16.00 ms (SD=2.10), 102.66 ms (SD=15.66), 1,284.27 ms (SD=172.27), and 11,174.82 ms (SD=880.21) for the first four sizes. For refresh time (30 trials per size), mean times were consistently low at 0.13–0.14 ms across all sizes.

6 Qualitative Evaluation

In this section, we evaluate TrajGram's effectiveness, learning curve, and usability through two usage scenarios and a user study. We did not run a controlled comparative study because the implementation effort differs substantially: reproducing Figure 1.B in D3.js would require roughly 550 lines, versus about 90 lines in our grammar.

6.1 Usage Scenarios

We introduce two usage scenarios for constructing interactive trajectory visualizations to demonstrate the effectiveness of TrajGram. The first scenario is derived from actual requirements in an enterprise project, while the second scenario is adapted from a paper [33].

Trajectory Filtering through Lens Interactions. An urban analyst needs to explore trajectory patterns in a city transportation dataset. Using TrajGram, he begins by seeking an *overview* of trajectory distributions and key locations: he first configures a background encoding to render all trajectories with appropriate styling, along with annotations highlighting starting and ending clusters. This establishes the foundational view for subsequent interactive exploration. To enable focused analysis, the analyst then incorporates interactive lens components for spatial filtering: he instantiate a selection component of type *lens.start* that manifests as a repositionable circular lens on the map interface. A filter query component is then linked to this lens selection, creating a *drag-filter* interaction chain where lens repositioning triggers dynamic query execution. The filtered results are visualized through two parallel pathways: an encoding component that highlights matching trajectories on the map, and a statistics component that computes and displays relevant metrics. This establishes *drag-filter-highlight* and *drag-filter-statistics* interaction chains (Figure 1.A1) that provide immediate visual and quantitative feedback during lens manipulation. The modular architecture facilitates seamless extension through additional

Figure 4: A road network heatmap system with TrajGram embedded. (A1) A combined area-line chart controls the color style of $E0$, currently highlighting high-volume roads. (A2) Three bar charts with a slider control $Q1$ conditions, filtering taxi trajectories by distance (< 10km), weekdays, and morning hours (6–10 am). The top-right inset shows an alternative color encoding based on roads' average speed. (B) The map view displays real-time heatmap results from three linked queries: filtering ($Q1$), segmentation ($Q2$), and aggregation ($Q3$), with $Q3$ rendered via encoding $E0$.

lens components (*lens.end* and *lens.pass*) that connect to the same query infrastructure for intersection filtering. This enables sophisticated multi-criteria filtering based on trajectory origins, destinations, and intermediate waypoints. To support comparative analysis of trajectory subsets, the analyst can instantiate *parallel chains* that share common start/end lens (Figure 1.A2) selections while maintaining independent pass-through lens components. Each chain connects to distinct encoding components with differentiated visual styling, enabling simultaneous analysis of divergent trajectory patterns within the same interface (Figure 1.B).

Dynamic Road-Network Heatmap. A visualization researcher needs to develop a road network heatmap system for studying urban traffic dynamics. He decide to use TrajGram for the heatmap visualization and data processing and develop a custom control interface that integrates with the framework (Figure 4). After importing the trajectory data, he establishes a *transformation pipeline* (Figure 4) consisting of three sequential queries: a filter query for trajectory selection, a segmentation query that groups trajectories by road segments, and an aggregation query that merges segments and calculates traffic statistics. An encoding component renders the final heatmap visualization from the aggregated results. For the interactive control interface, the researcher develops a custom control bar with interactive charts to modify filter conditions and encoding parameters. The expert then uses the library to establish communication between these charts and framework components. For instance, changes in Figure 4.A1 update the color settings in the encoding component *E0*, while charts in Figure 4.A2 monitors the results from the query component *Q3* to trigger corresponding updates.

6.2 User Study

We conducted a user study on TrajGram's usability and learnability.

6.2.1 Study Setting

Since directly writing complete JSON configurations can be cumbersome for users, we prepare an authoring tool based on our grammar that participants can use in reproduction process. The tool consists of a grammar editor and a visualization preview: the grammar editor offers two switchable modes—a card-based editor (Figure 5) and a JSON editor, while the visualization preview renders the results.

Participants. We recruited 12 participants who majored in computer science, with 5 females (P1-5) and 7 males (P6-12). P1-11 had data visualization backgrounds, understand trajectory visualization, and were familiar with D3.js and declarative visualization tools (Vega-Lite or ECharts); while P12 had no visualization experience.

Procedure. After an initial tutorial on the grammar and authoring tool, participants independently reproduced the target visualization with assistance available upon request. Upon completion of the task, participants completed post-study questionnaires using 7-point Likert scales to assess four key dimensions of usability: 1) learnability of grammar within each component, 2) learnability of inter-component relationships, 3) iterability of creation, and 4) ease of use. The study concluded with a semi-structured interview to gather participants' feedback on their experience, including their perceptions of the grammar structure and details, the authoring tool, and any challenges encountered during the implementation process.

6.2.2 Results and Findings

The reproduction task involved 2 queries, 3 encodings, and 4 selections, and all 12 participants completed it within 15 minutes. We analyze the learning curve and usability by triangulating Likert-scale ratings (Figure 6) with participants' interview feedback.

Learning Curve. Participants reported a low learning burden: 10 out of 12 grasped the pipeline and core concepts within 15–20 minutes and started writing valid specifications. Among compo-

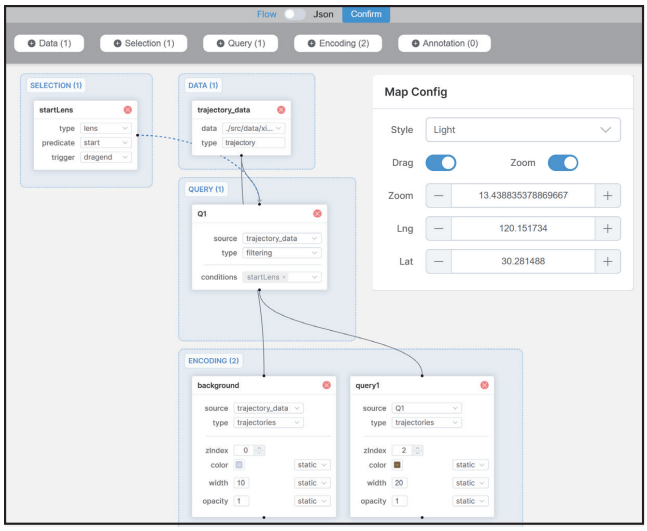

Figure 5: Card-based mode of authoring tool. Users can add new cards by clicking buttons, where each card represents a component. Users can configure internal content to modify component settings, and connect cards with lines to represent their relationships.

nents, *Selection* received relatively lower learnability ratings, which participants (P9) attributed to our new abstraction over interaction, while *Query* and *Encoding* were perceived as SQL-like and similar to Vega-Lite, respectively. For inter-component relationships, participants (P1–11) generally found the pipeline consistent with common visualization workflows, which may reduce the learning burden. Participants also spoke positively of the authoring tool: most participants primarily used the card-based editor and found it helpful for understanding the JSON-based grammar. P9 and P12 noted that "It makes relationships clear, while the JSON format is less intuitive." Importantly, all participants confirmed the consistency between the two representations. This also suggests that the authoring tool can serve as a useful bridge before users become fluent in the grammar, further supporting the learnability of our approach.

Usability. Overall, participants gave high ratings for *ease of use* and *iteration support*. P1 and P6 noted that, compared to Vega-Lite, the clearer separation of pipeline stages can improve extensibility and iterability, albeit at the cost of some conciseness. Our D3.js reproduction attempt suggests substantial overhead for implementing comparable behaviors in a general-purpose toolchain. Consistently, some participants (P1, P2, P6) commented that similar trajectory-specific behaviors may be non-trivial to achieve in Vega-Lite or ECharts without advanced features. Together, these remarks suggest that TrajGram's trajectory-oriented abstractions can reduce authoring overhead and provide distinct usability benefits.

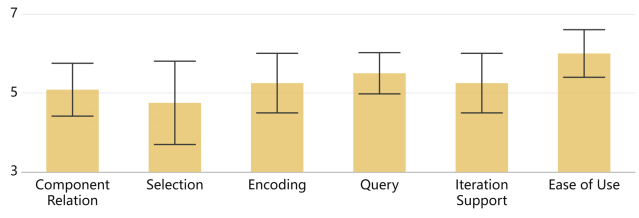

Figure 6: 7-point Likert ratings (1–7) of inter-component and 3 inner-component learnability, iterativity, and uasbility (mean ± SD).

7 Discussion and Limitations

This section discusses the implications and limitations of TrajGram.

While the design space primarily focuses on 2D trajectory visualization, the underlying categorization can be extended to accommodate 3D trajectory visualization to a certain degree.

A Pioneering Declarative Grammar for Trajectory Visualization. To the best of our knowledge, TrajGram is the first declarative grammar designed for trajectory visualization. Declarative grammar has been widely adopted in charting [47] and tree visualization [34] but not yet been explored in trajectory visualization. TrajGram introduces this paradigm to trajectory data and offers a practical and extensible solution for both academic and industrial use.

Toward Artificial Intelligence for Spatiotemporal Visualization. This work serves as a first step toward artificial intelligence for spatiotemporal visualization. While AI-driven visualization showed promising results in the context of tabular data [57, 60], its extension to spatiotemporal data remains underexplored due to the data's inherent complexity [20]. By articulating a structured design space, our study provides a foundation for future research that integrates machine learning with trajectory and spatiotemporal visualizations.

Limitations. While TrajGram offers a flexible and extensive approach to trajectory visualization, it also has several limitations. First, the system faces scalability pressure in both data processing and rendering when handling large-scale trajectory datasets. The data-processing bottleneck can be mitigated by deploying queries on a Node.js backend and streaming reduced results to the client, helping preserve interactive latency. The rendering bottleneck, however, remains due to browser memory limits when the dataset exceeds available memory; future work could address this with progressive rendering. Second, as a proof-of-concept system, the current implementation supports a relatively limited set of data transformation, visual encoding, and interaction functions. As more are introduced in future development, the detailed grammar specifications may need to be iteratively updated and extended.

8 Conclusion and Future Work

In this study, we introduce a declarative grammar TrajGram to address the challenge of developing interactive trajectory visualizations. Our approach begins with an extensive review of existing trajectory visualization techniques and visual analytics approaches. We classify and summarize them and condensed a design space containing three dimensions (interaction, transformation, and display). Leveraging this, we propose a declarative grammar that allows users to easily and intuitively design and create interactive trajectory visualizations. The usability, learning curve, and effectiveness of TrajGram are validated through usage scenarios, user study and performance analysis. In the future, we aim to enrich the specific functionality within our framework by expanding the repertoire of transformation operations, display techniques, and interactions.

Acknowledgments

The work was supported by NSFC (62421003, U22A2032, 62402428, 62402184, 62402421), Guangdong Basic and Applied Basic Research Foundation (2025A1515010162), Ningbo Yongjiang Talent Programme (2023A-396-G, 2024A-399-G). The author gratefully acknowledges the support of Zhejiang University Education Foundation Qizhen Scholar Foundation.

References

[1] deck.gl. `https://deck.gl/`.

[2] mapbox.gl.js. `https://docs.mapbox.com/`.

[3] leaflet.js. `https://leafletjs.com/`.

[4] A. P. Afonso, A. Ferreira, L. Ferreira, and R. Vaz. Rosetrajvis: Visual analytics of trajectories with rose diagrams. In *24th International Conference on Information Visualisation, IV 2020, Melbourne, Australia, September 7-11, 2020*, pp. 378–384. IEEE, 2020.

[5] S. Al-Dohuki, F. Kamw, Y. Zhao, X. Ye, J. Yang, and S. Jamonnak. An open source trajanalytics software for modeling, transformation and visualization of urban trajectory data. In *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019*, pp. 150–155. IEEE, 2019.

[6] S. Al-Dohuki, Y. Wu, F. Kamw, J. Yang, X. Li, Y. Zhao, X. Ye, W. Chen, C. Ma, and F. Wang. Semantictraj: A new approach to interacting with massive taxi trajectories. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):11–20, 2017. doi: 10.1109/TVCG.2016.2598416

[7] G. Andrienko, N. Andrienko, W. Chen, R. Maciejewski, and Y. Zhao. Visual analytics of mobility and transportation: State of the art and further research directions. *IEEE Transactions on Intelligent Transportation Systems*, 18(8):2232–2249, 2017. doi: 10.1109/TITS.2017.2683539

[8] 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. *IEEE Transactions on Intelligent Transportation Systems*, 18(8):2232–2249, 2017. doi: 10.1109/TITS.2017.2683539

[9] N. Andrienko, G. Andrienko, and S. Rinzivillo. Exploiting spatial abstraction in predictive analytics of vehicle traffic. *ISPRS International Journal of Geo-Information*, 4(2):591–606, 2015.

[10] N. V. Andrienko and G. L. Andrienko. Spatial generalization and aggregation of massive movement data. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):205–219, 2011. doi: 10.1109/TVCG.2010.44

[11] Apache Software Foundation. Apache echarts, 2021. Accessed: 2026-2-01.

[12] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, 2009. doi: 10.1109/TVCG.2009.174

[13] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185

[14] R. Chen, X. Shu, J. Chen, D. Weng, J. Tang, S. Fu, and Y. Wu. Nebula: A coordinating grammar of graphics. *IEEE Transactions on Visualization and Computer Graphics*, 28(12):4127–4140, 2022. doi: 10.1109/TVCG.2021.3076222

[15] R. Chen, D. Weng, Y. Huang, X. Shu, J. Zhou, G. Sun, and Y. Wu. Rigel: Transforming tabular data by declarative mapping. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):128–138, 2023. doi: 10.1109/TVCG.2022.3209385

[16] W. Chen, F. Guo, and F. Wang. A survey of traffic data visualization. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):2970–2984, 2015. doi: 10.1109/TITS.2015.2436897

[17] D. Chu, D. A. Sheets, Y. Zhao, Y. Wu, J. Yang, M. Zheng, and G. Chen. Visualizing hidden themes of taxi movement with semantic transformation. In *IEEE Pacific Visualization Symposium, PacificVis 2014, Yokohama, Japan, March 4-7, 2014*, pp. 137–144. IEEE Computer Society, 2014.

[18] O. Daae Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *2011 IEEE Pacific Visualization Symposium*, pp. 171–178, 2011. doi: 10.1109/PACIFICVIS.2011.5742387

[19] Z. Deng, H. Chen, Q. Lu, Z. Su, T. Schreck, J. Bao, and Y. Cai. Visual comparative analytics of multimodal transportation. *Visual Informatics*, 9(1):18–30, 2025. doi: 10.1016/J.VISINF.2025.01.001

[20] Z. Deng, D. Weng, S. Liu, Y. Tian, M. Xu, and Y. Wu. A survey of urban visual analytics: Advances and future directions. *Computational Visual Media*, 9(1):3–39, 2023. doi: 10.1007/S41095-022-0275-7

[21] A. Endert, W. Ribarsky, C. Turkay, B. Wong, I. Nabney, I. Díaz Blanco, and F. Rossi. The state of the art in integrating machine learning into visual analytics: Integrating machine learning into visual analytics. *Computer Graphics Forum*, 36, 03 2017. doi: 10.1111/cgf.13092

[22] L. Ferreira, G. Moreira, M. Hosseini, M. Lage, N. Ferreira, and F. Miranda. Assessing the landscape of toolkits, frameworks, and authoring tools for urban visual analytics systems. *Computers & Graphics*, p. 104013, 2024. doi: 10.1016/j.cag.2024.104013

[23] 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. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2149–2158, 2013. doi: 10.1109/TVCG.2013.226

[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, PacificVis 2011, Hong Kong, China, 1-4 March, 2011*, pp. 163–170. IEEE Computer Society, 2011.

[25] X. Huang, Y. Zhao, C. Ma, J. Yang, X. Ye, and C. Zhang. Trajgraph: A graph-based visual analytics approach to studying urban network centralities using taxi trajectory data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):160–169, 2016. doi: 10.1109/TVCG.2015.2467771

[26] Z. Huang, Y. Zhao, W. Chen, S. Gao, K. Yu, W. Xu, M. Tang, M. Zhu, and M. Xu. A natural-language-based visual query approach of uncertain human trajectories. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1256–1266, 2020. doi: 10.1109/TVCG.2019.2934671

[27] C. Hurter, B. Tissoires, and S. Conversy. Fromdady: Spreading aircraft trajectories across views to support iterative queries. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1017–1024, 2009. doi: 10.1109/TVCG.2009.145

[28] C. Hurter, B. Tissoires, and S. Conversy. Fromdady: Spreading aircraft trajectories across views to support iterative queries. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1017–1024, 2009. doi: 10.1109/TVCG.2009.145

[29] J. Jo, F. Vernier, P. Dragicevic, and J. Fekete. A declarative rendering model for multiclass density maps. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):470–480, 2019. doi: 10.1109/TVCG.2018.2865141

[30] H. Kim, Y. Kim, and J. Hullman. Erie: A declarative grammar for data sonification. In F. F. Mueller, P. Kyburz, J. R. Williamson, C. Sas, M. L. Wilson, P. O. T. Dugas, and I. Shklovski, eds., *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*, pp. 986:1–986:19. ACM, 2024. doi: 10.1145/3613904.3642442

[31] H. Kim, R. A. Rossi, F. Du, E. Koh, S. Guo, J. Hullman, and J. Hoffswell. Cicero: A declarative grammar for responsive visualization. In S. D. J. Barbosa, C. Lampe, C. Appert, D. A. Shamma, S. M. Drucker, J. R. Williamson, and K. Yatani, eds., *CHI '22: CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA, 29 April 2022 - 5 May 2022*, pp. 600:1–600:15. ACM, 2022.

[32] R. Krüger, D. Thom, and T. Ertl. Semantic enrichment of movement behavior with foursquare-a visual analytics approach. *IEEE Transactions on Visualization and Computer Graphics*, 21(8):903–915, 2015. doi: 10.1109/TVCG.2014.2371856

[33] R. Krüger, D. Thom, M. Wörner, H. Bosch, and T. Ertl. Trajectorylenses - A set-based filtering and exploration technique for long-term trajectory data. *Computer Graphics Forum*, 32(3):451–460, 2013. doi: 10.1111/CGF.12132

[34] G. Li, M. Tian, Q. Xu, M. J. McGuffin, and X. Yuan. Gotree: A grammar of tree visualizations. In R. Bernhaupt, F. F. Mueller, D. Verweij, J. Andres, J. McGrenere, A. Cockburn, I. Avellino, A. Goguey, P. Bjøn, S. Zhao, B. P. Samson, and R. Kocielnik, eds., *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, pp. 1–13. ACM, 2020. doi: 10.1145/3313831.3376297

[35] 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. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):1–10, 2017. doi: 10.1109/TVCG.2016.2598432

[36] H. Liu, Y. Gao, L. Lu, S. Liu, H. Qu, and L. M. Ni. Visual analysis of route diversity. In *6th IEEE Conference on Visual Analytics Science and Technology, IEEE VAST 2011, Providence, RI, USA, October 23-28, 2011*, pp. 171–180. IEEE Computer Society, 2011.

[37] H. Liu, S. Jin, Y. Yan, Y. Tao, and H. Lin. Visual analytics of taxi trajectory data via topical sub-trajectories. *Visual Informatics*, 3(3):140–149, 2019. doi: 10.1016/J.VISINF.2019.10.002

[38] S. Liu, J. Pu, Q. Luo, H. Qu, L. M. Ni, and R. Krishnan. VAIT: A visual analytics system for metropolitan transportation. *IEEE Transactions*

on Intelligent Transportation Systems, 14(4):1586–1596, 2013. doi: 10.1109/TITS.2013.2263225

[39] Z. Liu, C. Chen, F. Morales, and Y. Zhao. Atlas: Grammar-based procedural generation of data visualizations. In *2021 IEEE Visualization Conference, IEEE VIS 2021 - Short Papers, New Orleans, LA, USA, October 24-29, 2021*, pp. 171–175. IEEE, 2021.

[40] L. Lu, N. Cao, S. Liu, L. M. Ni, X. Yuan, and H. Qu. Visual analysis of uncertainty in trajectories. In *Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13-16, 2014. Proceedings, Part I*, vol. 8443, pp. 509–520. Springer, 2014.

[41] M. Lu, Z. Wang, and X. Yuan. Trajrank: Exploring travel behaviour on a route by trajectory ranking. In *2015 IEEE Pacific Visualization Symposium, PacificVis 2015, Hangzhou, China, April 14-17, 2015*, pp. 311–318. IEEE Computer Society, 2015.

[42] S. L'Yi, Q. Wang, F. Lekschas, and N. Gehlenborg. Gosling: A grammar-based toolkit for scalable and interactive genomics data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):140–150, 2022. doi: 10.1109/TVCG.2021.3114876

[43] G. Moreira, M. Hosseini, M. N. A. Nipu, M. Lage, N. Ferreira, and F. Miranda. The urban toolkit: A grammar-based framework for urban visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):1402–1412, 2024. doi: 10.1109/TVCG.2023.3326598

[44] D. Park, S. M. Drucker, R. Fernandez, and N. Elmqvist. Atom: A grammar for unit visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 24(12):3032–3043, 2018. doi: 10.1109/TVCG.2017.2785807

[45] D. Ren, B. Lee, and M. Brehmer. Charticulator: Interactive construction of bespoke chart layouts. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):789–799, 2019. doi: 10.1109/TVCG.2018.2865158

[46] M. Rodrigues, J. Dapello, P. Vaithilingam, C. Nobre, and J. Beyer. Protograph: A non-expert toolkit for creating animated graphs. In *2023 IEEE Visualization and Visual Analytics (VIS), Melbourne, Australia, October 21-27, 2023*, pp. 176–180. IEEE, 2023.

[47] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017. doi: 10.1109/TVCG.2016.2599030

[48] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, 2016. doi: 10.1109/TVCG.2015.2467091

[49] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In H. Benko, M. Dontcheva, and D. Wigdor, eds., *The 27th Annual ACM Symposium on User Interface Software and Technology, UIST '14, Honolulu, HI, USA, October 5-8, 2014*, pp. 669–678. ACM, 2014. doi: 10.1145/2642918.2647360

[50] R. Scheepens, N. Willems, H. van de Wetering, and J. J. Van Wijk. Interactive visualization of multivariate trajectory data with density maps. In *2011 IEEE pacific visualization symposium*, pp. 147–154. IEEE, 2011.

[51] M. Shih, C. Rozhon, and K. Ma. A declarative grammar of flexible volume visualization pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1050–1059, 2019. doi: 10.1109/TVCG.2018.2864841

[52] S. Srabanti, G. E. Marai, and F. Miranda. Streetweave: A declarative grammar for street-overlaid visualization of multivariate data. *Computing Research Repository*, abs/2508.07496, 2025. doi: 10.48550/ARXIV.2508.07496

[53] G. Sun, B. Chang, L. Zhu, H. Wu, K. Zheng, and R. Liang. Tzvis: visual analysis of bicycle data for traffic zone division. *Journal of Visualization*, 22(6):1193–1208, 2019. doi: 10.1007/S12650-019-00600-6

[54] C. Tominski, H. Schumann, G. L. Andrienko, and N. V. Andrienko. Stacking-based visualization of trajectory attribute data. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2565–2574, 2012. doi: 10.1109/TVCG.2012.265

[55] F. Wang, W. Chen, F. Wu, Y. Zhao, H. Hong, T. Gu, L. Wang, R. Liang, and H. Bao. A visual reasoning approach for data-driven transport assessment on urban roads. In *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 103–112, 2014. doi: 10.1109/VAST.2014.7042486

[56] F. Wang, W. Chen, Y. Zhao, T. Gu, S. Gao, and H. Bao. Adaptively exploring population mobility patterns in flow visualization. *IEEE Transactions on Intelligent Transportation Systems*, 18(8):2250–2259, 2017. doi: 10.1109/TITS.2017.2711644

[57] Q. Wang, Z. Chen, Y. Wang, and H. Qu. A survey on ML4VIS: applying machine learning advances to data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 28(12):5134–5153, 2022. doi: 10.1109/TVCG.2021.3106142

[58] Z. Wang, M. Lu, X. Yuan, J. Zhang, and H. van de Wetering. Visual traffic jam analysis based on trajectory data. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2159–2168, 2013. doi: 10.1109/TVCG.2013.228

[59] K. Wongsuphasawat. Encodable: Configurable grammar for visualization components. In *31st IEEE Visualization Conference, IEEE VIS 2020 - Short Papers, Virtual Event, USA, October 25-30, 2020*, pp. 131–135. IEEE, 2020.

[60] A. Wu, Y. Wang, X. Shu, D. Moritz, W. Cui, H. Zhang, D. Zhang, and H. Qu. AI4VIS: survey on artificial intelligence approaches for data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 28(12):5049–5070, 2022. doi: 10.1109/TVCG.2021.3099002

[61] W. Wu, J. Xu, H. Zeng, Y. Zheng, H. Qu, B. Ni, M. Yuan, and L. M. Ni. Telcovis: Visual exploration of co-occurrence in urban human mobility based on telco data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):935–944, 2016. doi: 10.1109/TVCG.2015.2467194

[62] W. Zeng, C. Fu, S. M. Arisona, and H. Qu. Visualizing interchange patterns in massive movement data. *Computer Graphics Forum*, 32(3):271–280, 2013. doi: 10.1111/CGF.12114

[63] W. Zeng, C.-W. Fu, S. M. Arisona, A. Erath, and H. Qu. Visualizing mobility of public transportation system. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1833–1842, 2014. doi: 10.1109/TVCG.2014.2346893

[64] Y. Zhao, Y. Wang, X. Luo, Y. Wang, and J.-D. Fekete. Libra: An interaction model for data visualization. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25. Association for Computing Machinery, New York, NY, USA, 2025. doi: 10.1145/3706598.3713769

[65] Y. Zheng. Trajectory data mining: An overview. *ACM Transactions on Intelligent Systems and Technology*, 6(3):29:1–29:41, 2015. doi: 10.1145/2743025

[66] Y. Zheng, W. Wu, Y. Chen, H. Qu, and L. M. Ni. Visual analytics in urban computing: An overview. *IEEE Transactions on Big Data*, 2(3):276–296, 2016. doi: 10.1109/TBDATA.2016.2586447

[67] H. Zhou, P. Xu, X. Yuan, and H. Qu. Edge bundling in information visualization. *Tsinghua Science and Technology*, 18(2):145–156, 2013.

[68] J. Zong, J. Pollock, D. Wootton, and A. Satyanarayan. Animated vega-lite: Unifying animation with a grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):149–159, 2023. doi: 10.1109/TVCG.2022.3209369