Technische Universität Berlin

Chair of Database Systems and Information Management

Master's Thesis

Visual Analytics for Moving Objects Databases

Fabrício Ferreira da Silva
Degree Program: Computer Science
Matriculation Number: 452722

Reviewers
Prof. Dr. Volker Markl
Prof. Dr Odej Kao

Advisor(s)
Prof. Dr. Esteban Zimanyi - esteban.zimanyi@ulb.be
Prof. Dr. Mahmoud Sakr - mahmoud.sakr@ulb.ac.be
Prof. Juan Soto - juan.soto@tu-berlin.de

Submission Date
31.08.2021

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 31.08.2021

[illegible]

. . . .

Fabrício Ferreira da Silva

Zusammenfassung

Mit der steigenden Anwendung von mobilen Computersystemen und Lokalisationsverfolgungstechnologien wird fortwährend eine große Menge an raumzeitlichen Daten produziert. Solche Daten bieten für den Gesetzgeber, für Unternehmer_innen und für Wissenschaftler_innen enorme Möglichkeiten, wenn sie richtig genutzt werden. Die visuelle Analyse ist eine effektive Methode zur Datenerforschung und -analyse, um Wissen zu extrahieren, Muster zu erkennen und neue Erkenntnisse zu gewinnen. Im Kontext von Mobilität 4.0 kann es eine wertvolle Methode sein, um nachhaltige intelligente Verkehrssysteme zu gewährleisten, die Problemlösungs- und Entscheidungsaktivitäten unterstützen. Aktuelle Lösungen sind jedoch in der Regel domänenspezifisch und nicht für große Mengen von Mobilitätsdaten über den gesamten Entwicklungsverlauf geeignet. In dieser Arbeit schlagen wir eine Lösung vor, die eine Visualisierungsrahmenstruktur mit einem Datenbanksystem für bewegliche Objekte kombiniert. Wir erzielen eine bessere Leistung, indem wir jede Datenverarbeitung in den Bereich der Datenbank verschieben, weil Benutzer_innen Daten mithilfe von SQL-Abfragen effizient filtern, transformieren und zusammenführen können. Wir haben auch eine raumzeitliche Vektorkachelstrategie gewählt, die das von der Datenbank an die Visualisierungsrahmenstruktur übertragene Datenvolumen reduziert, die Ladezeit verkürzt und die Benutzerinteraktionen verbessert. Wir haben zwei Architekturlösungen vorgeschlagen: eine mit einem Kachelserver, um die Datenmenge zu maximieren, die die Lösung verarbeiten kann, und eine andere mit einem In-Memory-Kachelindex, der das schnelle herausgeben der Datenvisualisierung priorisiert. Die nun vorliegende experimentelle Studie hat gezeigt, dass unsere Lösung Trajektorien mit 3 Millionen Punkten (3 bis 7-mal mehr als der Stand der Technik) effizient verarbeitet und aufgrund ihrer Skalierbarkeit eine vielversprechende Lösung im Industriemaßstab ist.

Abstract

With the popularisation of mobile computing and location tracking technologies, a large volume of spatiotemporal data is being produced constantly. Such data can offer immense opportunities to lawmakers, business people and scientists, if properly consumed. Visual analytics is an effective way of conducting data exploration and analytics to extract knowledge, identify patterns and generate insights. In the context of Mobility 4.0, it can act as a valuable method to guarantee sustainable intelligent transportation systems supporting problem-solving and decision-making tasks. However current solutions are usually domain-specific and are not suitable for large volumes of full-trajectory mobility data. In this work, we propose a solution that combines a visualisation framework with a Moving Objects Database. We obtain better performance by pushing any data processing to the database side, since users can filter, transform and aggregate data efficiently using SQL queries. We also adopted a spatiotemporal vector tiling strategy, which reduces the volume of data transferred from the database to the visualisation framework, decreasing load time and improving user interaction. We proposed two architectural solutions: one with a tile server to maximise the amount of data the solution can handle, and another with an in-memory tile index that prioritises fast data visualisation rendering. The experimental study showed that our solution efficiently handled trajectories containing 3 million points (3 to 7 times more than the state-of-the-art) and is a promising industry-scale solution due to its scalable nature.

Acknowledgments

First and foremost, I would like to thank my parents Ivone and Joaquim, who have sacrificed a lot so my siblings and I could pursue our dreams. Without their endless support and encouragement I would not have reached this moment in my life, and this thesis would have not been possible.

I would also like to thank my advisors Esteban Zimanyi and Mahmoud Sakr for their guidance during this master thesis and throughout the degree. We all share the same vision of conducting research to make a meaningful impact in today's society and I'm very grateful for the opportunity to work on a project that is so dear to me. Additionally, I am very grateful to Clodoveu Davis, the professor who introduced me to the field of urban computing and mentored me during my studies at UFMG.

Throughout my academic life I have made so many friends that provided such a strong support system. I have to give a special thanks to my BDMA friends, with whom I have shared most of my time during the past 2 years, either studying for exams and working on projects, or partying after all assignments were done. More importantly, I am extremely grateful to my special friends in Brazil. Even though we were geographically apart, you were the closest during these mentally challenging times, either to lift me up when I most needed or to celebrate my accomplishments over our hours-long video calls. <3

Finally, I would like to thank every teacher, professor and mentor that gave me support during my academic life. Growing up in an unprivileged community was sometimes hard to picture myself succeeding in this unequal world. Working with very few resources, they were always great professionals and made me believe that I was capable of so much more than a simple boy from a public school could think of.

Contents

List of Figures

List of Tables

List of Abbreviations

AIS	Automatic Identification System
API	Application Programming Interface
CDN	content delivery network
DBMS	database management system
FPS	frames per second
GIS	Geographic Information System
GML	Geography Markup Language
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSM	Global System for Mobile Communications
MOD	moving objects database
MVT	Mapbox Vector Tiles
OGC	Open Geospatial Consortium
RFID	radio-frequency identification
SDG	Sustainable Development Goal
WebGL	Web Graphics Library
ACT	Air Traffic Control
SQL	Structured Query Language

1 Introduction

The majority of activities in modern society are centred around spatiotemporal phenomena. Mobility is intrinsic to our lives. For instance, everyday people travel from home to work/school using different means of public and private transportation. Goods are transported to supply industries and businesses. Packages are shipped to our homes. Weather phenomena such as hurricanes form and move around different areas in the world. Populations of animals engage in long distance migrations, and many more other examples. In all these, there is a challenge of extracting knowledge and producing insights by analysing the data associated with it, either to have a proper understanding of the underlying phenomena or to make decisions.

With the evolution and popularisation of location tracking technologies, such as Global Positioning System (GPS), Automatic Identification System (AIS) and mobile computing, a large volume of spatiotemporal data is being produced constantly. As a consequence, challenging problems with spatiotemporal data became common in various areas, requiring an ecosystem of tools to efficiently acquire, ingest, store, manage, retrieve, transform, clean and analyse moving object big data.

Visual analytics is an effective way of conducting data exploration and analytics to extract knowledge, discover patterns and generate insights. It combines the strengths and capabilities of humans, through their visual perception, reasoning and potential for creative thinking to generate insights, and machines, through their ability to process a high volume of data that would be too large for humans to tackle it alone. Currently, however, there is not an industry-scale solution capable of conducting visual analytics on a high volume of mobility data. Therefore, in this work, we present a solution to this problem by connecting a visualisation framework to a moving objects database (MOD) which makes it possible to push most of the processing to the database, reusing its operations and indexing strategies. This way, users can filter, transform and aggregate data efficiently on the database side, and generate data visualisations with only the relevant data. To deal with a large volume of data, we adopted a spatiotemporal vector tiling strategy, which reduces the volume of data transferred from the database to the visualisation framework and decreases its latency, making it possible to conduct interactive visual analytics efficiently.

1.1 Motivation

Mobility is ubiquitous in contemporary daily life and impacts the economy, environment and quality of life of the population. Transportation and mobility are crucial to sustainable development since it directly impacts the economy. The United Nations includes sustainable transport in its Sustainable Development Goals (SDGs) [26]. Sustainable transportation promotes economic growth, easy accessibility to public services such as culture, leisure and education, and should improve social equity respecting the environment. The importance of transport for climate action is immense since it directly contributes to global greenhouse gas emissions, which are expected to increase with the fast growing urbanisation.

Because of that, a lot of investments are happening towards intelligent transportation systems, which aims to advance transport technology to improve efficiency, accessibility, safety, comfort and more importantly sustainability. As an example, the European Commission has spent efforts towards assessing future societal challenges related to transport demand and supply, focusing on cross-modal intelligent solutions [24].

With the introduction of Mobility 4.0 [17], technologies such as autonomous driving, intelligent transportation systems and mobility-as-a-service have been adopting data science to overcome inherent mobility challenges. This movement is surrounded by the idea that we can take advantage of the digitalisation and connectivity of transportation that constantly produces an increasing volume of data to come up with improved solutions to our current inefficiency problems. Therefore, there is a need to develop methods and technologies to monitor, identify patterns, generate insights and serve as leverage for problem-solving and decision-making activities in the transportation systems planning, management and operations.

1.2 Objective

In this work, we tackle the problem of conducting visual analytics for large-scale mobility data. For that, we need to overcome the challenges of scalability and usability: produce trajectory data visualization efficiently and without requiring advanced skills from the users. We propose methods and system architectures for conducting visual analytics on top of a MOD. Such solutions are able to handle large-scale full trajectory data efficiently, generating interactive data visualisations suitable for extracting valuable knowledge and insights from mobility data.

The main goal of this work is to create a method for conducting visual analytics on a MOD. As secondary goals, the proposed solutions should:

- minimise data transfer between database and visualisation framework, so users can interact with the data with minimal latency;

- generate visualisations based on database queries, so users can filter, transform and aggregate data efficiently on the database side;
- allow visual analytics considering full trajectories, not only origin and destination points;
- handle a large volume of moving objects data.

1.3 Contributions

Previous work had solved the problem partially. To our knowledge, [31] is the only work that explored visual analytics on a MOD. Other works proposed standalone visualization tools that process an input file and use their own in-memory data structures. Yet in [31], the solution transfers data from the database to an existing visual analytics tool, which does not scale. [11] propose visual analytics by querying a database, which is part of the goal of this thesis, but they used a custom spatial database and considered only origin and destination points (not the full trajectory) in their analysis. Our solution proposes a breakthrough in visual analytics architecture. The novelty of our work is therefore a solution to efficiently conduct visual analytics of large-scale full trajectory data on a MOD. Hence, we present the following contributions:

1. We introduce a novel strategy to implement vector tiling for spatiotemporal data;
2. We propose two architectural designs of a framework to conduct visual analytics on top of a MOD using vector tiles, first with a tile server and second with an in-memory tile index;
3. We implement an industry-scale visual analytics framework for both architectural designs aforementioned, using MobilityDB and a WebGL powered visualisation framework.

1.4 Organisation

Chapter 2 gives an introduction to the relevant concepts and tools, fundamental for the understanding of the solution, which is then presented in Chapter 3. In the solution, we discuss the vector tiling strategy adapted for spatiotemporal data, and propose two architectural solutions prioritising either dealing with a high volume of data or minimal data transfer and rendering latency. In Chapter 4 we provide details on the implementation of our solutions and the results of its experimental evaluation. Chapter 5 includes a discussion about related work and how our solution differs from the state-of-the-art. Finally, we drive conclusions in Chapter 6 and present directions for future work.

2 Scientific Background

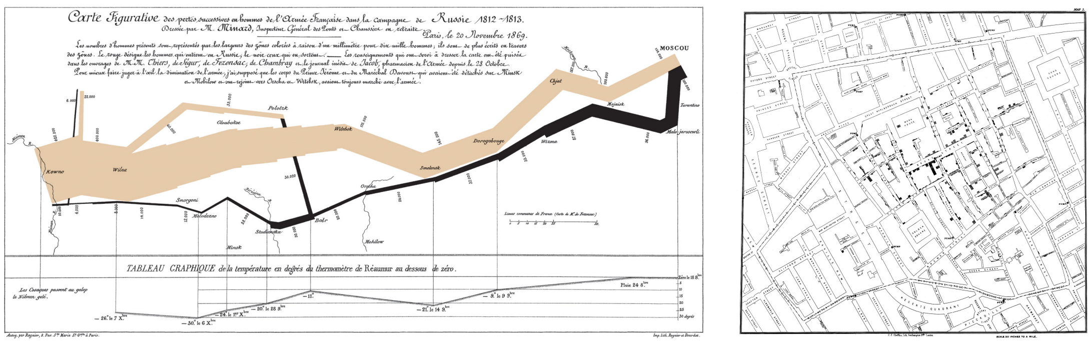
This section gives an theoretical foundation on relevant topics and technologies that will be necessary to understand the solution approach. It includes a brief introduction on visual analytics, moving object databases and vector tiling.

2.1 Visual Analytics

The democratisation of portable devices and the constant evolution of technologies to create, collect and, more importantly, store data resulted in a technological revolution we have been experiencing recently: the data revolution. We live in a world where many spheres of human activity generate and store vast amounts of data on a daily basis, from every branch of industry and business to political and personal activities. Although such a high volume of data can offer immense opportunities to lawmakers, business people and scientists, raw data has no value in itself. Alternatively, we need to be able to extract information contained in it to use it for making decisions.

When the ability to extract information is not as fast as collecting and storing data, it results in the information overload problem, which refers to the danger of getting lost in data. If we lack the ability to properly handle a high volume of data, we lose crucial opportunities to the information overload, eventually resulting in the loss of time and money. Therefore, there is a need to develop effective tools and methods to exploit large and heterogeneous data resources, extracting knowledge and identifying hidden opportunities that it might offer.

Data visualisation is an indispensable practice in the analysis of big data. It translates data into a visual context to leverage the power of the human visual system to extract information, identify patterns and pull insights from the visual representation of data, which could not be easily done by computers. Historically, data visualisation has proven to be an incredibly effective practice for centuries. In fact, two of the most famous examples of successful use of data visualisation to represent and analyse data are dated from the 19th century. Published in 1869, Minard's visualization of Napoleon's Russian campaign, illustrated by Figure 1a, uses cartographic techniques to numerically portrait Napoleon's disastrous losses suffered during the Russian campaign between 1812 to 1813. It successfully implements visual representation techniques, such as shape, colour and size, to encode six different variables pertinent to the campaign, including changes in the size of troops through time and geographic location. The second example is a map produced by John Snow in 1854 about a serious outbreak of cholera in London's Soho, shown in Figure 1b. By representing each cholera death as a bar on the map of Soho, Snow was able to ascertain clusters and eventually identified the source of the outbreak as one of the public water pumps. Snow's work is an example of a data visualisation that was effectively used for data analysis and decision-making.


MOSCOU
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.

(a) (b)

Figure 1: Famous data visualisations dated from the 19th century [36]. **(a)** Minard's visualization of Napoleon's Russian campaign. **(b)** Snow's map of the 1854 outbreak of cholera in London.

Throughout time, visual exploration has improved with the democratisation of more powerful technology and computer graphics. If we compare with the aforementioned examples, nowadays visualisations can be easily produced and highly interactive. In general, such systems provide interaction mechanisms such as semantic zoom (automatically adapt what is visible and how is displayed according to levels of detail), brushing (visually select a subset of data with a device), and linking (display a subset of data among multiple visualisations, such as a dashboard) [20]. Such interactivity enables quick hypothesis making and testing, fundamental steps for the data exploration process. The term visual analytics originates from the idea of providing better and more effective ways to understand and analyse data through interactivity.

2.1.1 Definition

Visual analytics was early defined as *"the science of analytical reasoning facilitated by interactive visual interfaces"* [35, p. 4]. It goes beyond information visualisation, focusing primarily on achieving effective understanding, reasoning and decision making by combining automated analysis techniques with interactive visualisations. In order to be successful, it needs to take into consideration not only data exploration and visualisation methods but also human factors such as semiotics, cognition, perception and interaction to create simple and yet effective visual analytics systems. The main goal is to turn the information overload into an opportunity by helping users: derive insights from massive, dynamic and complex data; discover non-obvious or unexpected information; provide timely, reasoned, and clear assessments; and use these assessments for decision-making [19].

On a large scale, visual analytics is designed for combining the strengths and capabilities of humans and machines for the most effective results. Specifically, humans apply their visual perception, reasoning and potential for creative thinking to generate insights, while computers process and mine data that would be too large for humans to tackle it alone. Visual representations are often the most straightforward method to convey information to our brains, and by adding interactivity to it, users have the chance to explore and analyse data from different perspectives and levels of abstraction, making associations to develop valuable insights. Figure 2 illustrates the idea that visual analytics can be described by the interplay of data analysis, visualisation and interaction. It needs to take into consideration human cognition and perception to build effective interactive visualisations, and it should employ methods developed in the field of information retrieval and data mining to facilitate data analysis.

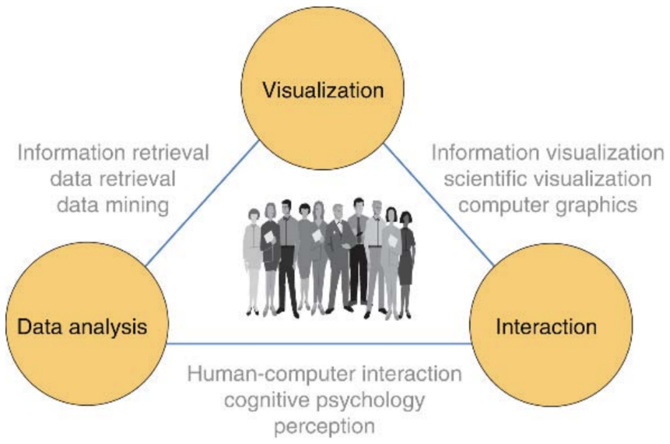


Figure 2: Visual analytics as the interplay between data analysis, visualization, and interaction methods [22].

2.1.2 Visual Analytics Process

To better understand the interaction between data, visualisations and models to achieve knowledge discovery, [19] proposed an abstract overview of the stages and transitions in the visual analytics process, illustrated by Figure 3. The first step refers to data preprocessing tasks, such as data cleaning, normalisation, aggregation and integration of heterogeneous data sources, represented by the arrow *Transformation*. After the preprocessing, one can choose between conducting automated data analysis or visual data exploration.

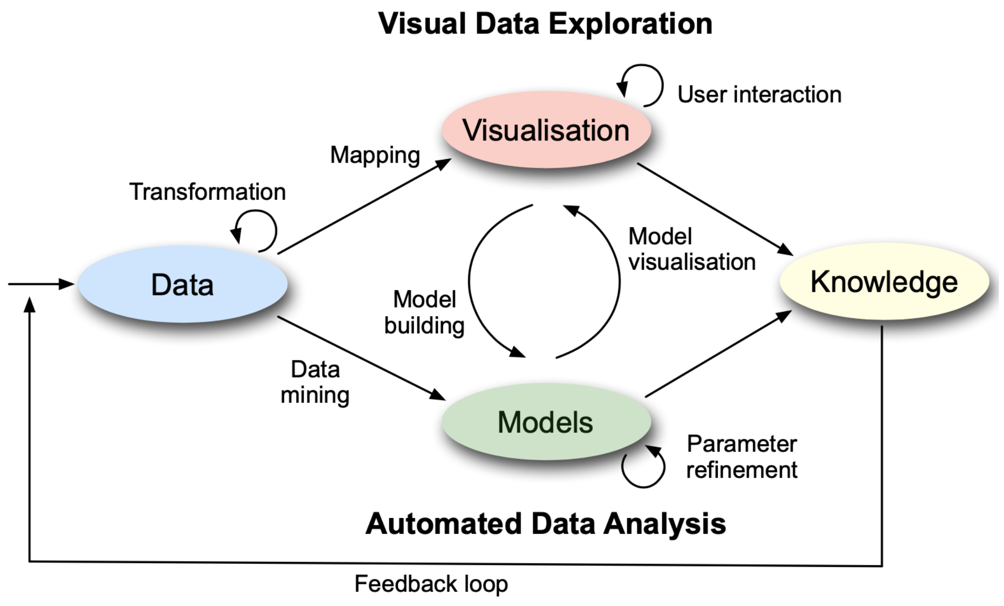


Figure 3: The visual analytics process [19].

If automated data analysis is performed first, a model of the original data is generated through data mining methods, which can later be refined by interacting with the data. Analysts can evaluate the models using visualisation to modify parameters or select other analysis algorithms. The exchange between visual and automated analysis promotes a continuous improvement and validation of the results in order to gain valuable and reliable knowledge. On the other hand, if visual data exploration is used first, the analyst maps the data to visualisations and interacts with it to steer model building in the automatic analysis. In summary, in the visual analytics process, knowledge can be obtained either from visual exploration, automated analysis or the continuous interaction between both.

However, there are many challenges in designing visual analytics systems. One concerns generalisation: it is common to find researches and tools that are too application-oriented and domain-specific, meaning that they offer solutions to specific fields and are not generic enough to apply to other problem domains. Providing real-time interactivity is another challenge when handling large datasets. We need to be able to process large amounts of data with minimal latency since real-time interactivity plays a crucial role in the data analysis process. Hence, visual analytics may adopt caching strategies, distributed computing, hardware acceleration such as Graphics Processing Unit (GPU) and scalable architectures to guarantee high speed and low latency interactive visualisations. A third challenge concerns how effective a visual representation can be when considering large and high-dimensional data. It is important to avoid visual clutter and confusing representation which can be easily achieved by filtering and aggregation strategies without compromising the discovery of patterns and the generation of insights.

Visual analytics research is a highly interdisciplinary field, which combines different related areas such as statistics, data management, cognitive science, visualisation, among others, as illustrated by Figure 4. The integration of all these diverse areas makes visual analytics a scientific discipline on its own, resulting in the potential to solve problems that were not effectively tackled by domain experts individually and to overcome the challenges aforementioned.

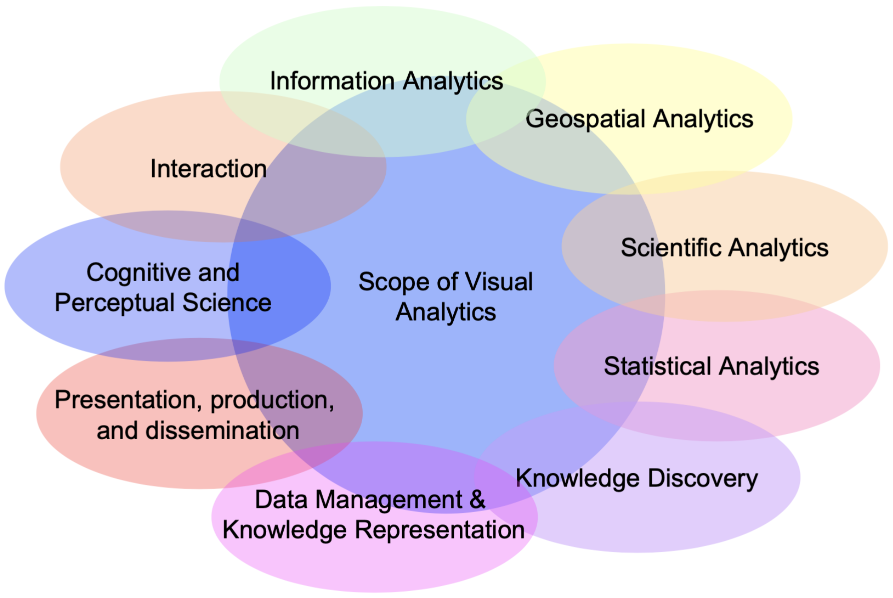

Information Analytics
Geospatial Analytics
Interaction
Scientific Analytics
Cognitive and Perceptual Science
Scope of Visual Analytics
Statistical Analytics
Presentation, production, and dissemination
Knowledge Discovery
Data Management & Knowledge Representation

Figure 4: Scope of visual analytics [21].

2.2 Moving Object Databases

When working with databases, a lot needs to be considered when deciding what technology to use. There is a variety of database management systems (DBMSs) with extensions to deal with data from different natures. Therefore we need to understand the phenomenon the data is representing and the dimensions associated with it to make a smart decision. Take for example a scenario where we want to store and analyse hurricanes. We could represent each of them as an event that happened to a location at a specific time. A spatial database would be enough to answer queries such as how many hurricanes happened in each country in the past year, or what month had the most hurricanes in the state of Florida, USA. The hurricane could even be represented as a polygon so we could access its impact based on its area or height. Now consider that with a satellite imagery data source, we identified that a single hurricane moves, and its shape changes with time. How would we structure and index a database to store this kind of data? What kind of query language would be necessary to extract information such as what direction they move, at what speed, and how the area varies with time. A spatial database would probably not be as straightforward and efficient in this situation, since now we are dealing with spatiotemporal data.

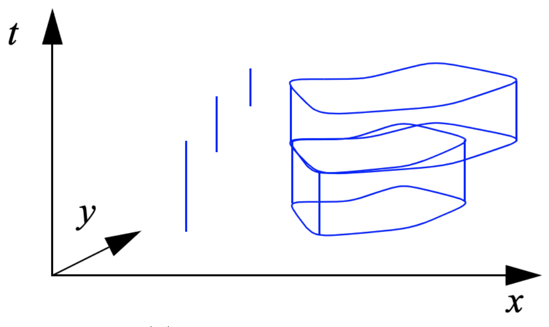
A large volume of spatiotemporal data is produced with the advances and popularisation of wireless and mobile computing, and the evolution of location tracking technologies, such as GPS, AIS, radio-frequency identification (RFID), sensor networks, Global System for Mobile Communications (GSM) localisation and WiFi positioning. As a consequence, challenging problems with spatiotemporal data became common in various areas, such as the manufacturing industry, transportation, ecology, social computing, meteorology, epidemiology, telecommunications, among many others. Therefore, there is a pressing demand for an ecosystem of software tools to efficiently acquire, ingest, store, manage, retrieve, transform, clean and analyse moving object big data.

A MOD is a piece of software to manage moving object data. It allows one to represent and query any database entities that are time dependent and have continuously changing geometries such as vehicles or animals, *"either online for current movement, or offline storing large sets of trajectories, or histories of movement"* [14]. Research in MODs has been active since the early 2000s, developing tools and methods regarding various aspects such as data modelling, operations and indexing. As illustrated by Figure 5, [15] point out the difference between MODs and spatiotemporal databases: the former emphasises that geometries may change continuously, whereas the latter supports only discrete changes.

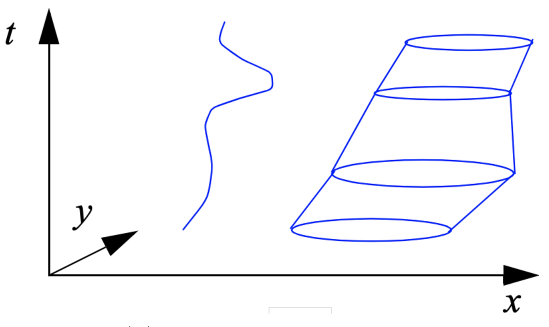
In the literature, there are multiple research prototypes of MODs and systems to manage and analyse big trajectory data. SECONDO [16], HERMES [27], Hadoop-Trajectory [4] and UlTraMan[8] are a few examples. Additionally, MobilityDB [37] is proposed as a mainstream system targeted for industrial use.

2.2.1 MobilityDB

As mentioned, MobilityDB was proposed to fill the gap of an industrial-scale MOD that has not been achieved by any previous research prototypes. It is an extension of PostgreSQL and PostGIS, both largely adopted in the industry, that supports



(a) discretely changing



(b) continuously changing

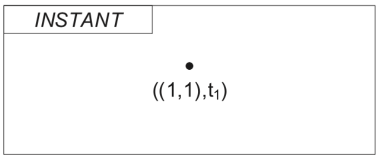
Figure 5: Changing points and regions through time (vertical axis) [15].

data type of moving objects, including a rich set of temporal and spatiotemporal query operations. MobilityDB is open-source, complies with Open Geospatial Consortium (OGC) standards on moving features and *"is by far the only existing moving object database that supports full SQL"* [37].

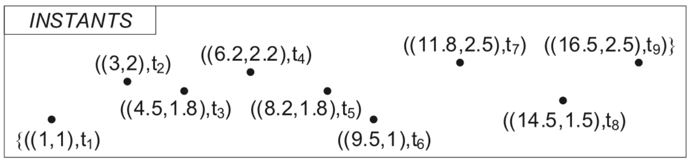
Let's consider a vehicle trajectory as an example. In a spatial database, it will be represented as a collection of discrete observations (list of spatial points, each with a timestamp) constrained by the technology that generated the data. If we want to query the position of the vehicle at a specific timestamp that is actually between two observations, we would have to interpolate the data we have to compute the location, which can be complex to do over a Structured Query Language (SQL) query. MobilityDB, on the other hand, offers the possibility to query the position of an object at a particular timestamp, or query the particular timestamp an object was at a given location. In MobilityDB, the individual observations are connected and represented as a continuous phenomenon to answer queries like those aforementioned.

Now let's imagine that during the trajectory the vehicle remained stopped at a location for a certain period. This will result in multiple consecutive observations in the same location. However, the earliest and the oldest observations on the same location are enough to represent the information that the vehicle is stationary, and the observations in between are redundant. MobilityDB simplifies the trajectory removing such redundant observations, which results in a more compact format, less space consumed and faster queries, without losing any information. It only stores the appropriate observations to describe the same trajectory. The same happens if the vehicle is moving in a straight line with constant speed: we do not need to store the intermediate points along the segment, only the extremities, since the data is stored as a continuous event and a linear interpolation can be used.

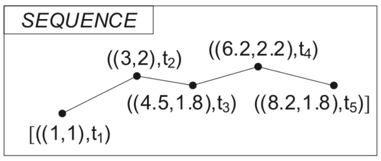
MobilityDB implements the concept of type constructors, to define extensible temporal types, which is a combination of a time type and a base type. Time types consist of timestamp (single instant), timestamp set (collection of discrete instants), period (continuous interval) and period set (disconnected collection of continuous intervals), whereas the base type can be PostgreSQL or PostGIS types, such as bool, int, float, text, geometry(Point) and geography(Point). For example, if we combine the period time type with the geometry(Point) base type, it would generate a temporal point that could be used to represent the continuous trajectory of a vehicle. Figure 6 illustrates all possible temporal types originated from the geometry(Point) base type.



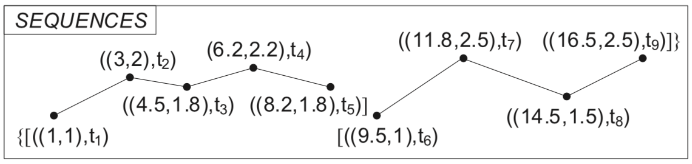
(a) INSTANT(Point)



(b) INSTANTS(Point)



(c) SEQUENCE(Point)



(d) SEQUENCES(Point)

Figure 6: MobilityDB's temporal types originated from the geometry(Point) [37].

The advantage of using native types from PostgreSQL and PostGIS is that any enhancement done to these tools will also be applied indirectly to MobilityDB. Additionally, MobilityDB extends PostGIS Application Programming Interface (API), taking into account the temporal feature. For instance, in PostGIS, one can measure the spatial distance between two static objects. In MobilityDB it is possible to compute the spatiotemporal proximity between two objects, which is the distance throughout time, given that either or both objects are moving. This is useful in the field of epidemiology, for instance, to conduct contact tracing analysis to identify when or if two moving objects (individuals) happened to be close to each other.

2.3 Vector Tiles

Throughout time, cartographers have always worked towards making maps relevant, easily accessible and constantly updated. The democratisation of the Web has changed drastically the way we address spatial information, and consequently, the way maps are designed, produced, delivered and more importantly how we procure and utilise it [7].

With the arrival of Web 2.0, mapping applications became ubiquitous on the Web. New platforms adopting interactivity and collective intelligence emerged offering wider and more democratic access and ownership of geographic information for not only professionals but also dwellers. As a result, novel technological solutions were proposed to attend to the demand for fast rendering in Web mapping, for example, tiled maps.

Map tiling is the most popular strategy to render and navigate maps. Instead of fetching a single large image for the whole viewport, it works by dividing the original space into rectangles (tiles) and only fetching the ones that are visible to the user, which is then stitched together to be displayed. Consequently, the data transfer is reduced since only the data within the current viewport at a certain zoom level needs to be obtained, resulting in a faster rendering. Additionally, since the tile boundaries are pre-defined, tiling operations can be performed in advance and cached, to enhance performance. This map tiling strategy can be used for raster or vector tiles.

Raster tiles denote that the spatial data for each tile is encoded into an individual bitmap image file such as PNG or JPG. It is usually adopted for storing continuous geographic information such as satellite and aerial imagery. The majority of maps services available on the Web today, such as Google Maps, Bing Maps and OpenStreetMap, use raster tiles as visual representations since it was the first tiling strategy to emerge and its transmission methods are easily implemented.

However, rendering raster imagery can be CPU and memory consuming, and it is inadequate in specific cases where the application requires user interactivity with the data presented on the map [6]. Such interactivity is essential in spatial data analysis applications. For this reason, vector tile maps were proposed to make possible the interaction between users and map objects [3].

Tiled vectors are a way of representing real-world geographical features through points, lines and polygons in the form of tiles. Different from raster tiled maps, which are composed of pre-rendered map images, it returns vector data clipped to the boundaries of each tile to be rendered in map symbolisation on the client-side. Figure 7 illustrates the principles of vector tiling, where the original geometries are clipped to the tile boundaries.

Vector tiled maps have several advantages compared to raster tiled maps [12]. Since vector data is usually smaller than bitmap images, data transfer is reduced if compared to a tiled raster map, resulting in less bandwidth needed and faster rendering. Moreover, it allows better flexibility in how the data is presented (e.g., graphic generalisation and label placement) since the styling can be performed during rendering time on the client-side, and may even be updated dynamically

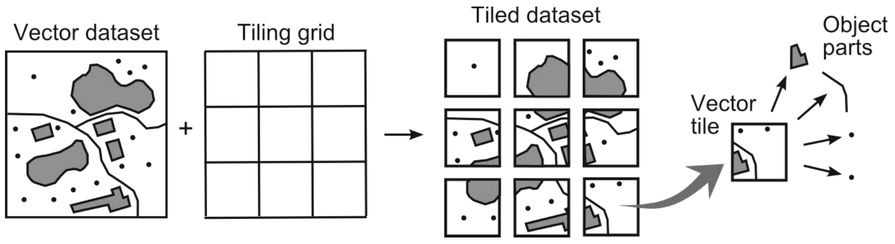


Figure 7: Principle of vector tiling [12].

given user interaction. Due to the on the fly rendering by the client, vector tiles also make possible the implementation of innovative data visualisation techniques, such as animated visualisations with moving and changing features. Another advantage over raster tiling is that vector tiles have more control over the level of details given a map scale. For different zoom levels, geometries can differ. For instance, a feature that is originally a polygon can be simplified to a point given a scale that does not require such a high level of details, avoiding redundant vector data that will not make a difference when rendering and, therefore, resulting in smaller tiles and faster data transfer.

2.3.1 Tiling Strategy

The strategy behind tile-based mapping services relies on tessellating the whole world into non-overlapping squares, i.e., tiles, that will reconstruct the original map when combined. It typically has multiple discrete zoom levels and for each, the tiles are defined with different sizes so as to guarantee the corresponding level of details of the map [32]. For instance, in higher zoom levels the map displayed includes more details and therefore each tile covers a small area of the original map. In contrast, maps in lower zoom levels require fewer details, hence each tile represents a bigger area of the original map. Therefore, multiple zoom levels represent multiple map views.

This strategy of sectioning the map area depending on the zoom level follows a pyramid type structure, as shown by Figure 8. The lowest zoom level, the pyramid's top, consists of one tile with the size of the whole map and the least level of detail. Each subsequent zoom level has 4 times the number of tiles as its predecessor.

As a means to better manage the multiple tiles of different zoom levels, tile-based maps use a simple coordinate system presented by OGC (Open Geospatial Consortium) when they released the WMTS (Web Map Tile Service) as the standard protocol in 2010 [32]. Each tile has z/x/y coordinates, where z is the zoom level and x and y the position of the tile. The tile on top of the pyramid has coordinates

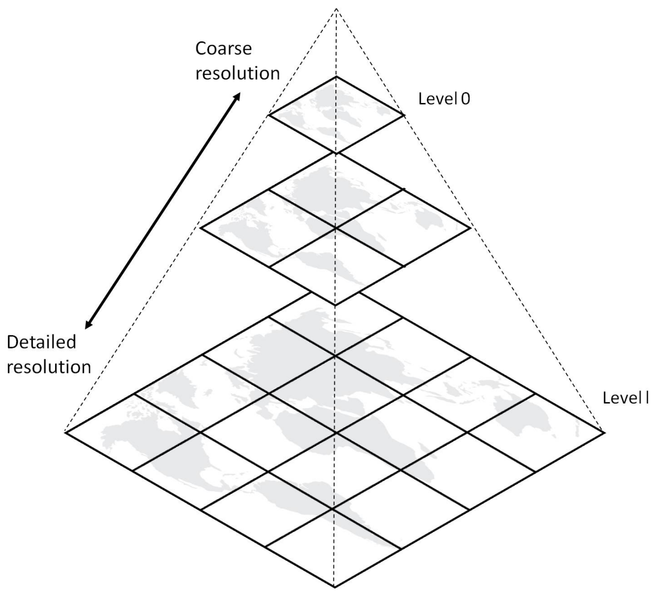


Figure 8: Pyramid of the tiling strategy based on zoom level [29].

0/0/0. Then, in the following zoom level, the map is divided into four equal tiles: 1/0/0 represents the top-left tile, 1/1/0 the top-right, 1/0/1 the bottom-left, and 1/1/1 the bottom-right tile, as shown by Figure 9, and so on. Most tile-based maps use a Web Mercator (WGS 84) projection, which is considered the *de facto* standard in tile-based mapping services.

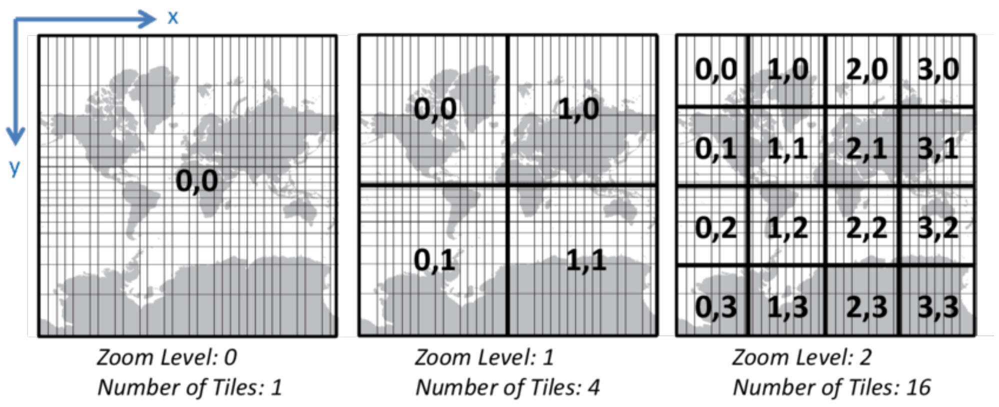


Figure 9: Coordinate system used by common tiling schemes [33].

Depending on the style some features are rendered, a map can be rendered with distortion when individual tiles are stitched together to form the complete map. Figure 10 presents an example of a road that crosses multiple tiles and is rendered with distortion around the tile borders.

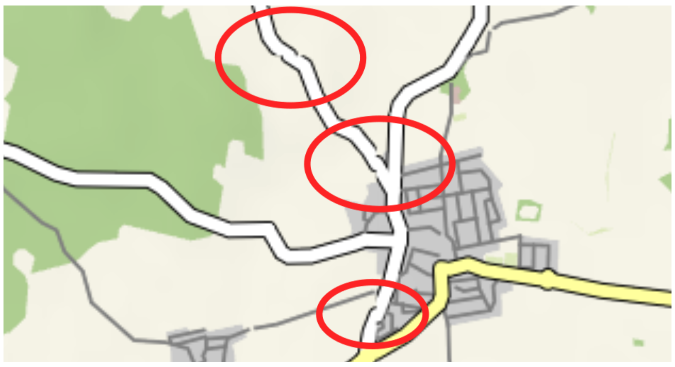
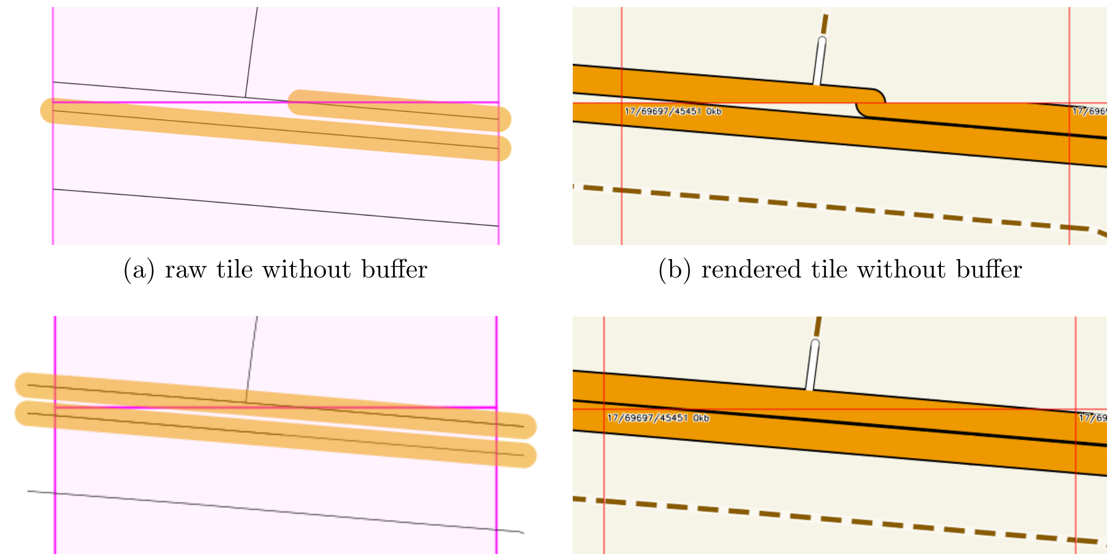


Figure 10: Example of a vector tile map with distortions [34].

To overcome this issue, vector tiles are usually created with an expanded size given a buffer. For example, if a tile of size 256 is requested with a buffer of 32, it will return a tile of size 288. That means the vector tile will contain data that is outside the visible area since it will be overlapped by the neighbour tiles.

By introducing a buffer to the vector tile during creation, features will be rendered beyond the tile boundaries promoting a visual continuity since the features of one tile will overlap their continued features on the neighbour tiles. Figure 11 presents the comparison of rendering a road that crosses multiple two tiles with and without buffer. (a) illustrates how one of the tiles is being rendered before the clipping phase without buffer and (b) includes the resulting image after the different tiles are rendered and stitched together. (c) and (d) represent the equivalent but with a buffer.

Given a certain zoom level, a tiled map only renders the appropriate collection of tiles that are visible on the viewport with the corresponding details that are visible within the scale. Even though this sectioning strategy can generate a lot of tiles for a single dataset, there are various efficient techniques and systems responsible for generating, encoding, storing, indexing and transmitting tiles. Mapbox Vector Tiles (MVT), Geography Markup Language (GML), GeoJSON and TopoJSON are some of the most popularly used vector formats for encoding vector tiles. Additionally, a large number of well established open source and commercial software support vector tiles, including databases, Geographic Information System (GIS), data servers, web clients, among others. Some databases even make available functions to fetch spatial data in a form of encoded tiles.



(a) raw tile without buffer

(b) rendered tile without buffer

(c) raw tile with buffer

(d) rendered tile with buffer

Figure 11: Use of tile buffers to avoid rendering distorted features [34].

2.3.2 Mapbox Vector Tiles

Mapbox has defined an open standard for vector tiled maps, including specifications around how data is stored and encoded. Its first version was released in 2014 and nowadays is one of the most famous and widely used standards in distributing vector tiles. The specifications [25] include directions about the projection of reference, bounds, feature and attributes encoding, and file format. MVT uses Web Mercator (WGS 84) as the projection of reference and Google Protocol Buffers as an encoding format for lightweight data serialisation, generally achieving good performance.

Various applications, ranging from parsers and GIS to databases are compatible with MVT[1]. For instance, PostGIS has two functions used to generate map tiles: *ST_AsMVTGeom* and *ST_AsMVT*. The first function is responsible for clipping the spatial data to the provided tile boundaries, simplifying features to accommodate the level of details given the zoom level, and converting data from cartesian coordinates to tile coordinate space. The second is responsible for encoding the geometry generated by the previous function and other attributes into the binary MVT representation with protobuf format.

Normally, a map tile architecture includes a web tile server responsible for converting tile coordinates into SQL queries that fetches the equivalent vector

[1]https://github.com/mapbox/awesome-vector-tiles

tile. The map client, for example, an application for rendering the map in a web browser, is responsible for sending requests to the tile server with the tiles that are visible on the viewport, which are converted to queries to the database and then executed and returned to the client to be displayed. Figure 12 illustrates an architecture with a web client sending an HTTP request to a tile server implemented in Python and a PostGIS database.

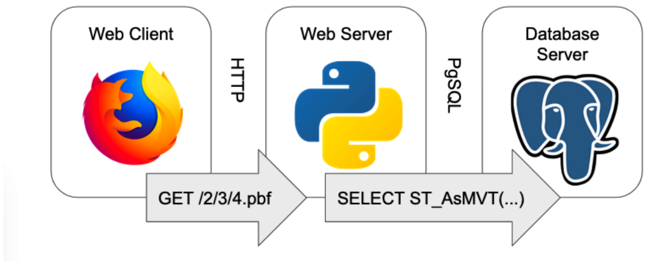


Figure 12: Example of a vector tile architecture with a tile server consuming tiles from PostGIS [29].

3 Solution

In this work, the main goal is to make it possible to conduct visual analytics on big trajectory data from a MOD. To achieve it, the solution needs to meet the following requirements:

- **R1:** it should be able to handle a large volume of data (millions of points) that might not fit into memory;
- **R2:** it should allow querying the MOD to select the data since the user might be interested in filtering the original dataset or using aggregation strategies;
- **R3:** it should load data into the visualization framework with minimal latency;
- **R4:** it should provide interaction mechanisms for effective visual analytics, such as semantic zoom, brushing and linking;
- **R5:** it should render data visualisations efficiently to make real-time interactivity possible.

We propose two solutions that guarantee most or all requirements above using vector tiling strategy. The first solution takes advantage of a vector tile server that serves only the data that is visible to the user, focusing on handling a large volume of data (R1). The second solution, on the other hand, sacrifices the ability to handle a large volume of data (R1) to guarantees minimal latency when fetching vector tiles (R3) by implementing an in-memory tile index on the client-side.

3.1 Spatiotemporal Vector Tiling

In order to be able to generate vector tiles for moving object data, we propose a tiling strategy to deal with spatiotemporal data. This approach is fairly similar to generating vector tiles for spatial data, including the same operations but with minor modifications to handle the temporal dimension. The steps (and their order) to compute vector tiles might vary according to the adopted vector map tile standard and characteristics of the system that implements it.

As explained in Section 2.3, a vector tiling strategy consists primarily of clipping the vector data to the boundaries of the tile, and might include other steps such as simplification, projection and encoding. Below, we describe some of the steps that are commonly found in popular vector tiling approaches, such as the ones implemented by popular spatial databases. We also highlight the strategy necessary to handle spatiotemporal data, more specifically for the simplification, clipping and encoding steps.

3.1.1 Filtering Small Features

Depending on the zoom level, there are features that might be too small to be visible if we try to render it and thus could be filtered out. This step is responsible for dropping geometries that are smaller than a provided resolution. Typically, the resolution is based on the tile's boundary width and height (which are proportional to the tile's zoom level) and the tile's extent (which is the tile size in the tile coordinate space).

Figure 13 exemplifies the filtering stage with a feature that is smaller than the resolution and therefore would be dropped out, and a second feature that is bigger than the resolution and will be included in the vector tile. In this example, the tile extent is 8, the horizontal resolution is tile width divided by extent (width of a pixel) and vertical resolution is tile height divided by extent (height of a pixel). Feature A is excluded because its width is smaller than the horizontal resolution and its height is smaller than the vertical resolution. Feature B is not excluded because its height is bigger than the vertical resolution, even though the same is not true for the horizontal resolution.

3.1.2 Simplifying Geometries

Similar to the previous step, the zoom level dictates the level of details of a tile. Even for complex geometry, we do not see many details of it when we zoom out. Hence, we can implement a step to simplify the original geometry, decreasing its number of vertices and consequently its size. More lightweight geometries result in a faster response.

The Ramer–Douglas–Peucker algorithm [28, 9] is a suitable alternative to simplify spatiotemporal geometries. Given a curve composed of points that define line segments and a distance dimension ϵ, it iterates over the inner points removing those that the resulting curve with its absence does not deviate more than ϵ from the original curve. Figure 14 illustrates an example of simplification using this algorithm, where the three points in red were removed from the original curve given the ϵ informed in the figure.

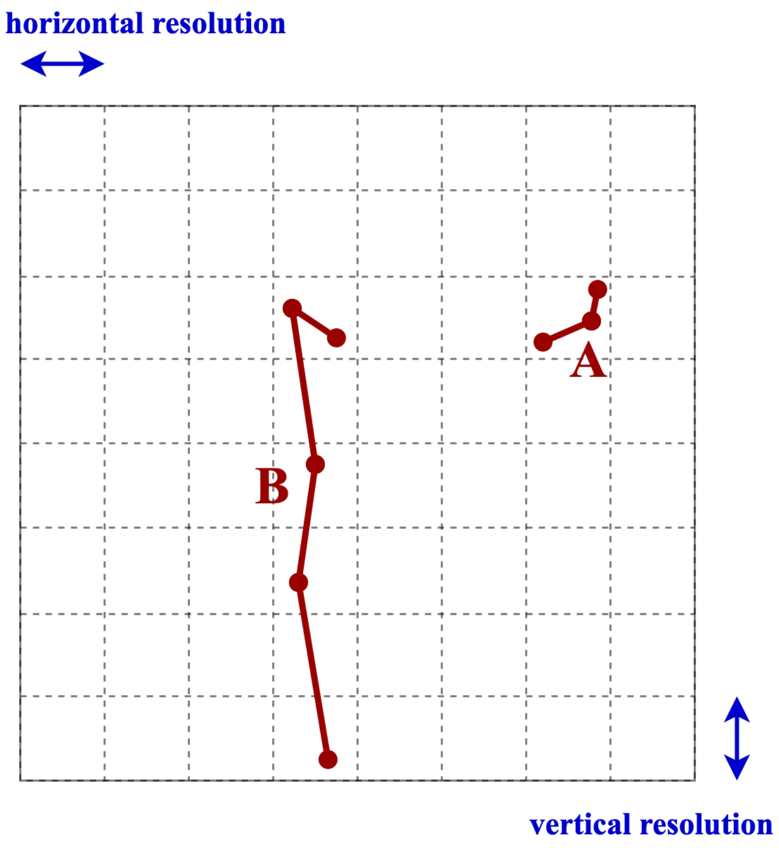


Figure 13: Filtering out features smaller than tile resolution: feature A is removed since its width and height are smaller than the resolution.

This algorithm also works for spatiotemporal data such as trajectories. Since the timestamp is associated with each vertex of the curve, when a point is removed from the original geometry, its timestamp is also removed. However, there is a downside to it. Imagine the first three vertices from left to right in Figure 14 has timestamps 0, 10 and 100, respectively. If we generate an animation with the original curve, we see that such object moves faster between the first two points and slower from the second to the third. But with the simplification, this behaviour will be replaced by a slow movement between the first and third points. In other words, the removal of intermediate vertices affects the original speed behaviour when interpolation is performed. Nonetheless, it's possible to implement a modified algorithm that considers not only the spatial deviation but also the temporal one, with a second parameter for a temporal distance dimension.

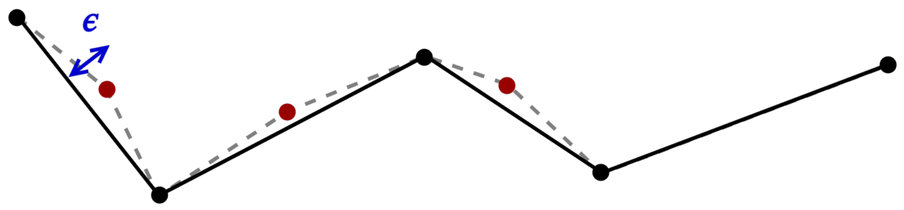


Figure 14: Ramer–Douglas–Peucker curve simplification algorithm

3.1.3 Clipping to Tile Boundaries

For each tile, we select only the data that are within its boundaries. Nevertheless, there will be cases where the geometry intersects the tile, meaning that it can have one or more segments inside the tile boundaries and one or more segments outside the tile boundaries. The clipping process consists of clipping the original geometry to remove all its content that is outside the tile boundaries, as illustrated by Figure 7.

Take as an example the trajectory and tile boundaries represented in Figure 15a. We can clip the geometry to the tile boundaries using the intersection overlay function commonly implemented in any geoprocessing tool or spatial database. The resulting geometry will include all the vertices that were inside the tile boundaries in addition to the points where there was an intersection between the trajectory and the tile boundaries, highlighted in Figure 15b. Such additional points were not part of the original trajectory and therefore has no vertex attributes associated with, such as timestamps. Hence, for clipping spatiotemporal objects, we need an extra step to compute the timestamp of these additional points. A straightforward option is to compute it using spatiotemporal interpolation [30], considering the predecessor and successor points. A bilinear interpolation, for example, computes the timestamp preserving the original speed of the moving object.

3.1.4 Projecting to Screen Coordinates

Depending on the specifications of the vector tile format, the tiles have a specific coordinate space, as explained in Section 2.3.1. Because of that, the data needs to be projected into this coordinate system. Such projection only modifies the object's geometry coordinate values, which means that only the spatial component of a spatiotemporal object will be modified.

3.1.5 Encoding Output

Depending on the vector tile standard, the vector tile needs to be encoded before being outputted. MVT specification, for example, are encoded using Google's

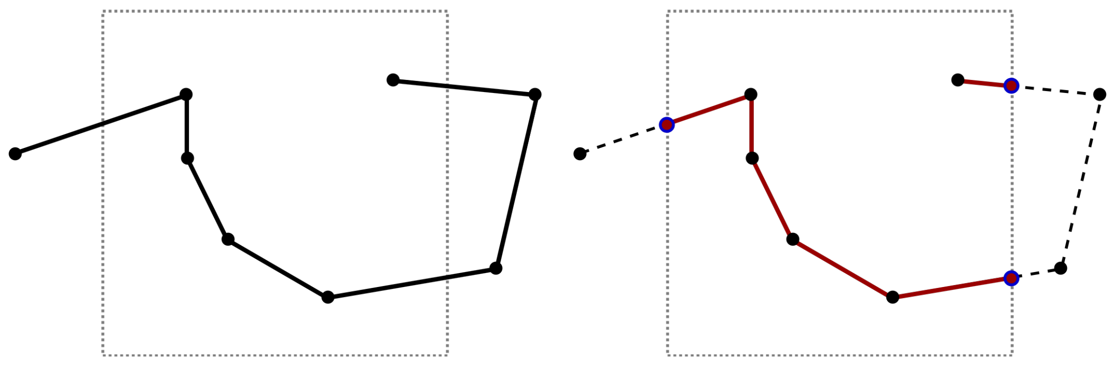

(a) original geometry (b) clipped geometry

Figure 15: Clipping geometry to tile boundaries.

protocol buffers (protobufs). It serialised the data into a *.mvt* binary format that is returned to an application that decodes it and renders the tile. The current version of MVT specification, 2.1, does not support spatiotemporal data, only geometries and attributes associated with it, which can be of type string or numerical. An alternative is to disassemble a spatiotemporal data type into the spatial and the temporal components. The spatial component is encoded normally as geometry, as already supported by the vector tile specification, while the temporal component is encoded as an attribute of the geometry in a form of a string representing the ordered list of timestamps comma separated. On the client-side, after decoding the tile and before rendering, it is necessary to extract the timestamps from the list of attributes and reassemble with the geometry to form the original spatiotemporal object.

3.2 Tile Server Architecture

Using a tile server to visualise data is already a widespread solution in spatial context. Well known GIS tools implement a data connector to consume vector tiles on the fly by fetching data when it is about to be visible in the client's viewport. This strategy, as already explained in Section 2.3.1 results in better performance for large volumes of data, since it promotes faster data transfer. The first solution proposed follows the same strategy, but now applied to spatiotemporal data.

Figure 16 illustrates the overview of solution 1, which consists of a server side and a client-side. The server side consists of the MOD, responsible for storing the data and executing queries to filter, transform and aggregate the data, and the tiler server, responsible for sending queries to the database to fetch the tiles. In the client-side, we have a visualisation framework, which is the interface where the

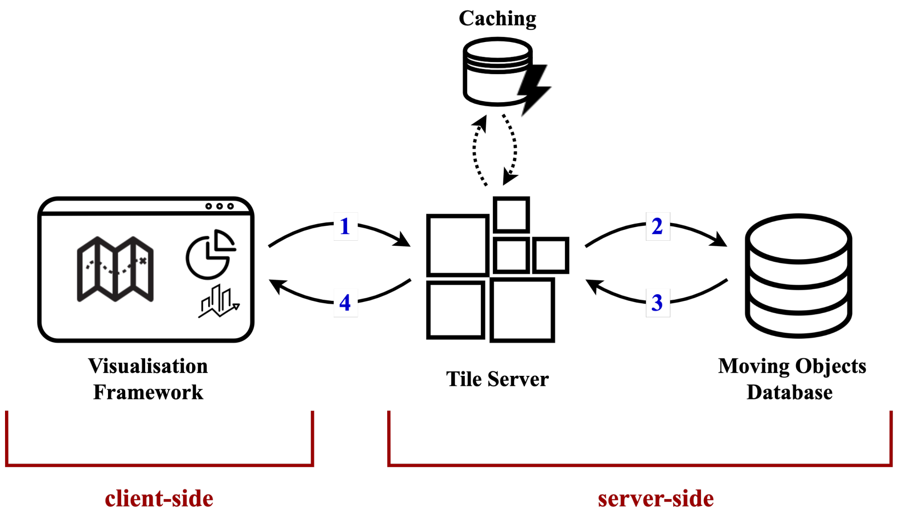


Figure 16: Solution architecture with tile server

user can conduct visual analytics. The data flow can be summarised as:

1. Given the current viewport and zoom level, visualisation framework sends a request to tile server, informing the coordinates of the tile in $z/x/y$ and data source the tile will be computed from;

2. Tile server converts this request into a SQL query that is sent to the MOD, if not cached already;

3. MOD returns the vector tile to the tile server

4. Tile server respond the visualisation framework's request with the vector tile, which will then be rendered in the client-side.

The advantage of dividing the architecture into server and client-side is that we can improve the scalability of the solution. For example, we could use a distributed database to handle high volumes of data, caching strategies to deliver faster requests, and deploy both MOD and tile server to a high performance infrastructure either on a bare-metal server or on a cloud service. This way, multiple users can simultaneously conduct visual analytics on their personal computers, consuming the same data source but fetching only the data that are valuable to them. Additionally, users will not be limited by the specifications of the hardware they are using.

3.2.1 Visualisation Framework

The visualisation framework is the interface where the user will be able to conduct the visual analytics. It is responsible for sending requests to the tile server to fetch data tiles according to the viewport and zoom level, for creating the customisable data visualisations, allowing users to interact with the data. It should also allow the user to write queries to the database to fetch the specific data they need using the operations provided by the MOD that allows filtering, transformation and aggregation. The visualisation framework should be effective but also easy to install, easy to use, and capable of rendering high volumes of data. Hence, visualisation frameworks based on Web Graphics Library (WebGL) are highly recommended, since they take advantage of GPU-accelerated rendering which happens on the web browser, meaning that no software installation is required.

3.2.2 Tile Server

Responsible for making the connection between what the user wants to visualise and the data source, the tile server translates the tile requests made by the visualisation framework into queries to SQL to be executed on the MOD. The tiles are requested using the coordinate system $z/x/y$, which results in a MVT query to the database and returned to the visualisation framework as an encoded vector tile. Along with the tile coordinates, the user also needs to inform the source of the data, which can simply be the name of the table in the database, or a selection query. If a query is provided, the tile server is responsible to query the database to create a view in the database and serve the tiles from such a view.

3.2.3 Moving Object Database

The MOD is responsible for storing the moving object data. It processes the tile queries sent by the tile server and returns the vector tiles so it can be transmitted to the visualisation framework to be visualised. The database should implement the spatiotemporal vector tiling function and also allow data filtering, transformation and aggregation operations.

3.3 In-Memory Tile Index

The previous solution architecture might meet all the requirements aforelisted. However, depending on the network connection speed, on how big the data is and therefore the size of the tiles, the latency from transferring each vector tile from the MOD to the tile server and then to the visualisation framework might be high, which would consequently fail to meet the requirement R3. In certain applications, where the interactivity with the visualisation needs to be as real time as possible, the data transfer latency needs to be minimal. The second solution proposed prioritises low latency (R3) at the expense of not being able to handle large volumes of data (R1). It does not include a tile server, as illustrated by Figure 17.

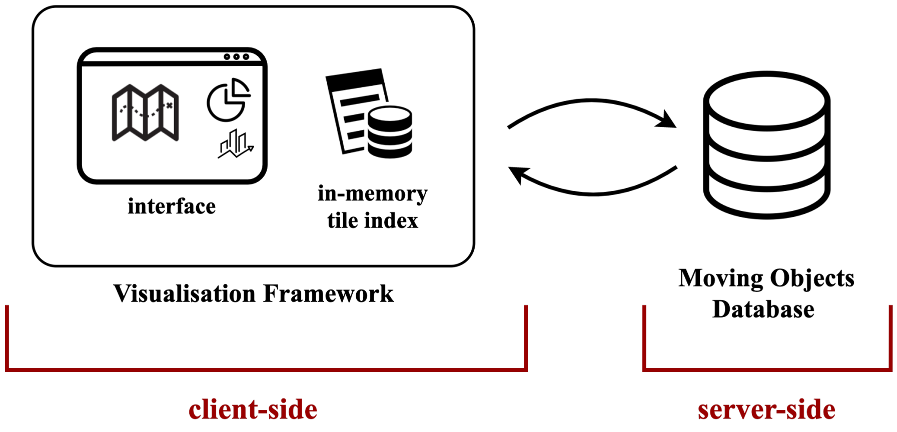


Figure 17: Solution architecture with in-memory tile index

The strategy of this solution consists of fetching all the data from the MOD, loading it to the visualisation framework and generating the tiles on the fly on itself. The tiles will be computed and saved into an in-memory tile index, meaning that the tiles will not be transferred, instead will be stored in the client-side, where the rendering step happens. Even though we achieve the minimal latency between generating the tile and rendering it, this solution has a drawback: the volume of the data is limited by the amount of memory available on the client-side. Since the full data queried by the user is loaded into the visualisation framework, this solution will not work for a large dataset that surpasses the memory capacity of the user's machine.

4 Implementation and Experimental Evaluation

In this chapter, we discuss the implementation of the solutions and the experimental study to evaluate their performance.

4.1 Implementation

In order to evaluate if the proposed architectures meet the expected requirements, we implemented open-source prototypes that could be easily used in real-world scenarios and collaboratively evolved by anyone interested in contributing to the project.

We decided to use MobilityDB as the MOD, since it is an industrial-scale solution, capable of dealing with high volumes of data with high performance, it is compatible with the PostgreSQL ecosystem, compliant with OGC standards and offers a rich collection of functions for mobility analytics.

As mentioned in Section 2.3.2, PostGIS already implements functions to generate MVT tiles but is limited to geometry types. Given a geometry and the tile boundaries, *ST_AsMVTGeom* is responsible for simplifying, clipping and projecting the geometry into tile coordinates. The output of this function is then used as an input by *ST_AsMVT* function, to encode the geometry and other relevant attributes into a binary MVT using protobuf format. We then implemented a similar strategy in MobilityDB to deal with temporal geometries, such as *tgeompoint*, which is the data type used to represent spatiotemporal trajectories.

Similarly to *ST_AsMVTGeom*, we implemented *AsMVTGeom* in MobilityDB. This function applies the operations as PostGIS' implementation, but for a *tgeompoint* data type input. Since MVT specifications do not support spatiotemporal data, *AsMVTGeom* implements a spatiotemporal vector tiling strategy that returns the vector tile geometry and list of timestamps disassembled. With this approach, one can use PostGIS' *ST_AsMVT* function to encode the output of *AsMVTGeom*, where the list of timestamps will be encoded as a string attribute of the vector tile and will need to be decoded when received.

For the visualisation framework, we chose to work with two tools: Deck.gl[1]

[1]https://deck.gl/

and Kepler.gl[2]. The first is a WebGL-powered visualisation framework for visual exploratory large-scale data analysis. It implements an approach based on layers that can be composed to build complex data visualisations. Users have access to a catalogue of standard layers, or can build their own based on the requirements of the project. Since Deck.gl supports WebGL, it provides fast processing with GPU-accelerated parallel rendering, suitable for large datasets. Likewise, Kepler.gl is a powerful open source geospatial analysis tool that uses WebGL to render large datasets quickly and efficiently in the web browser. It is actually built on top of Deck.gl, taking advantage of all its capability and high performance, but giving better support to users with drag and drop function to upload data and easy interaction to add filters, aggregations and visual customisation to each layer. Figure 18 illustrates the interface of Kepler.gl with an animated visualisation of NYC taxi trips.

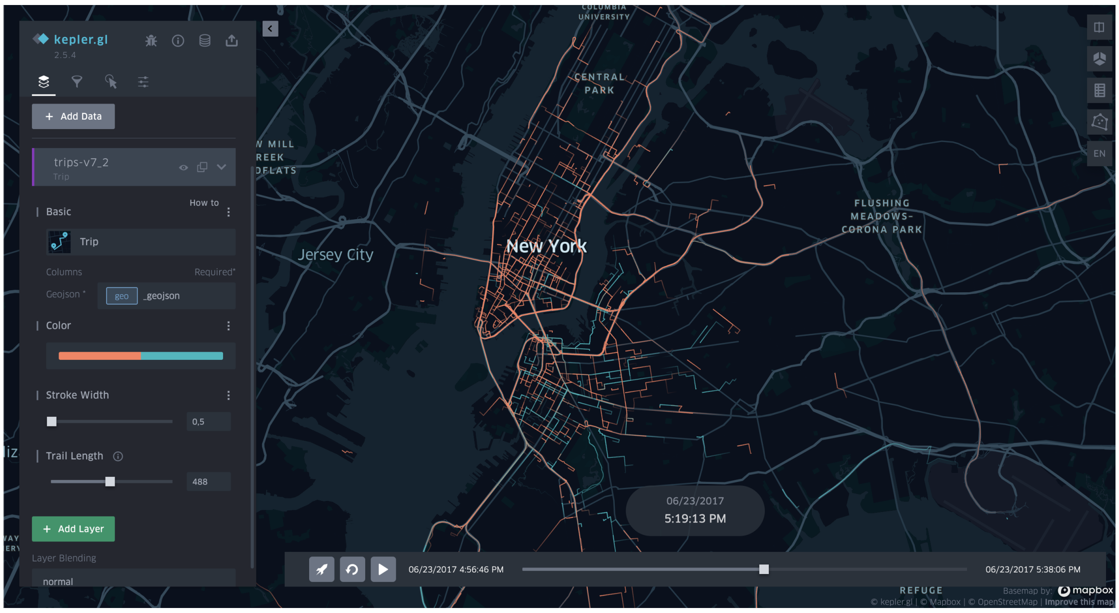

trips-v7_2
Basic: Trip
Geojson: _geojson
Stroke Width: 0.5
Trail Length: 488
Layer Blending: normal
06/23/2017 5:19:13 PM
06/23/2017 4:56:46 PM – 06/23/2017 5:38:06 PM

Figure 18: Kepler.gl interface with animated trips layer.

Ideally, Kepler.gl is a better alternative for our context than Deck.gl since it provides creation and customisation of layers with an easy interface, without any requirement of programming skills. But due to the complexity of the code, lack of support from the community and time constraints, we could not implement in Kepler.gl a layer capable of rendering spatiotemporal data (animation of trips) from vector tiles. We implemented in Kepler.gl an interface so users could write queries to the MOD to fetch the data, but so far we were only able to implement a

[2]https://kepler.gl/

MVT spatiotemporal layer in Deck.gl, which does not compromise our performance evaluation since both Kepler.gl and Deck.gl use the same rendering framework, called luma.gl[3]. We implemented a custom layer on Deck.gl based on the existing layers MVTLayer and TripsLayer that consumes spatiotemporal data from a MVT server and generates an animated visualisation of the moving objects on a map. This custom layer was used for the performance evaluation, but other layers could be implemented in the future.

For the tile server, we chose pg_tileserv[4], a lightweight PostGIS tile server implemented in Go. We modified pg_tileserv so it can also work for MobilityDB's tgeompoint data type and, in this case, use the corresponding MobilityDB's function *AsMVTGeom*. In order to implement the second architectural solution that proposes the use of an in-memory tile index, we adopted an existing tool developed by Mapbox named geojson-vt[5]. This is a JavaScript library that slices GeoJSON data files into vector tiles on the fly, designed to handle large volumes exclusively on the client-side, when there is no server. Since the tiles are computed and saved in memory, it promotes fast data rendering and interaction, without the latency of fetching tiles from a server. We modified geojson-vt so it can compute tiles for spatiotemporal data: GeoJSON geometry object with a list of timestamps in its property.

4.2 Dataset

In our experimental study, we decided to use a moving objects data generator so we can evaluate the performance of our solution over different data sizes. BerlinMOD [10] is a benchmark developed to evaluate the performance of moving object databases. It is based on a simulation scenario within the street network of Berlin where the location of vehicles is monitored for a given period of time. The benchmark contains a set of queries, but also a dedicated data generator that generates moving objects data so it can be easily executed for different data scales. In this sense, BerlinMOD simulates an ordinary person's behaviour of going from home to work and vice versa during the weekdays with a personal vehicle, and sporadic trips in the evening and weekend. The data generator has a scale factor parameter, which determines the number of simulated vehicles and the number of days, which is important for our experiment since it allows us to evaluate the impact that the size of the data has on the performance of our solution. In this experiment, we adopted 4 different sizes of data: small (S), with a scale factor of 0.005; medium (M), with a scale factor of 0.05; large (L), with a scale factor of 0.2; and extra-large

[3]https://luma.gl/
[4]https://github.com/CrunchyData/pg_tileserv
[5]https://github.com/mapbox/geojson-vt

(XL), with a scale factor of 1. Table 1 lists the characteristics of each scale factor used in this experiment.

size	scale factor	# days	# vehicles	# trips	# points
S	0.005	2	141	1,797	346,657
M	0.05	6	447	15,045	2,998,674
L	0.2	13	894	62,510	12,091,785
XL	1.0	28	2000	292,940	56,129,943

Table 1: BerlinMOD data used in the experiments.

4.3 Experimental Setup

The experimental evaluation was conducted on a MacBook Pro 2020, with a 2 GHz Quad-Core i5 processor, 16 GB of RAM and 1.5 GB GPU. MobilityDB was executed in a Docker container with 4 CPUs and memory of a maximum of 8.0 GB. We used Google Chrome's DevTools to obtain the statistics used for comparison in the result discussion:

- Memory footprint: the amount of memory consumed by the web page, in this case, Deck.gl's application;
- Data transferred: the total sum of vector tile sizes transferred from the tile server to the Deck.gl's application;
- Load time: the amount of time needed to fetch and load the vector tiles into Deck.gl's application;
- frames per second (FPS): the rate that the animation of trajectories is refreshed on the display (ideally 30 fps or more, to guarantee smooth animation).

4.4 Results

We executed the experiment using Deck.gl as the visualisation framework, where the metrics were extracted. Since the data was located in Berlin, Germany, we centred the map at longitude 13.383406 and latitude 52.515338, with a zoom level of 11.

4.4.1 Evaluation of the Architecture with a Tile Server

Impact of MobilityDB's redundant points filtering optimisation

As mentioned in Section 2.2.1, MobilityDB implements a strategy that eliminates redundant observations of trajectories without compromising its nature. In this experiment we evaluate what's the impact of this strategy, comparing the number of observations from the original file with the number of observations stored in MobilityDB. Table 2 shows that the number of observations stored by MobilityDB is between 1.2 and 1.8 times smaller than the original, which results in less space consumed and faster query responses.

size	original # points	# points in MobilityDB
S	346,657	189,417
M	2,998,674	1,640,554
L	12,091,785	10,260,475
XL	56,129,943	31,120,134

Table 2: Difference between the original and MobilityDB's number of points.

Relationship between scale factor and load time

In this experiment, we measure how long it takes to request the vector tiles and receive the response from the tile server, including the time to execute the query in MobilityDB and the data transfer to the client. Figure 19 illustrates the result of the experiment. For the scale factors of 0.005, the result took 7.64 seconds to load completely, while for 0.05 it took 44.76 seconds. For scale factor 0.2, the vector tiles took almost 8.8 minutes to be loaded into Deck.gl which is a very long time if we consider that users will execute various queries during their exploratory analytics process. For scale factor 1.0 we could not obtain results, since the hardware used to execute the experiment did not have enough memory to compute the tiles.

Relationship between scale factor and volume of transferred data

In this experiment, we evaluate what is the volume of data transferred from the tile server to Deck.gl's application. This value gives us enough information to understand the latency of transferring vector tiles in different scenarios, where there is a limitation in the communication speed between the components of the architecture. As shown in Figure 20, for scale factor 0.005 the total data transferred was 44.4 MB, while for scale factor 0.05 it was 382 MB. As expected, for a large number of points, represented by scale factor 0.2, the data transferred from the tile server to Deck.gl's application was around 2.2 GB, which explains the high

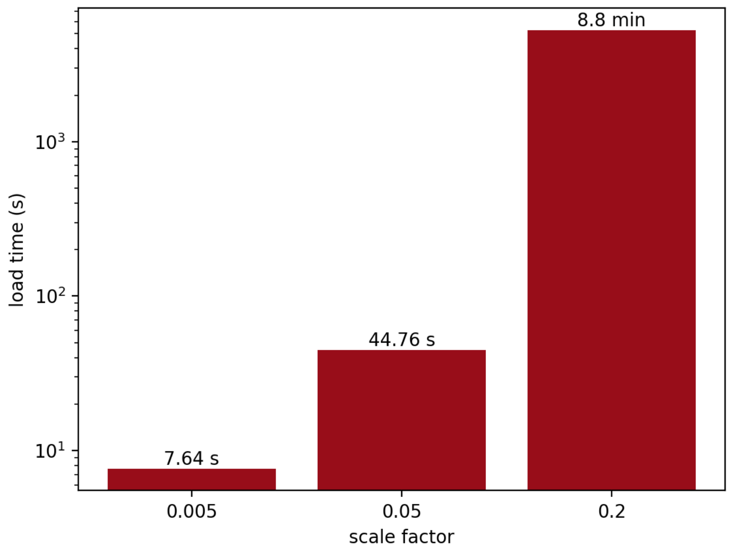


Figure 19: Experimental results regarding load time.

loading time previously shown. We believe that this amount of data might not be supported in real-world scenarios, where the client-side and server-side might be in different machines, and therefore there will be a speed-limited connection between both sides. Again, for scale factor 1.0 we could not obtain results, since the hardware used to execute the experiment did not have enough memory to compute the tiles. We can also observe that there is almost a linear correlation between the number of points and the amount of data transferred, if we evaluate the ratio number of points to data transferred. On average, 1 MB holds information of 4408 points.

Relationship between scale factor and memory consumed

In this experiment, we measure the memory footprint of Deck.gl's application for each of the data sizes. Figure 21 shows the amount of memory consumed for the scale factors 0.005, 0.05 and 0.2. For the latter, the memory consumed was around 4.4 GB, which can possibly be considered a limit, since average personal computers (client-side) will not have much more memory available than this. The hardware used to execute the experiments was not able to handle the scale factor of 1.0, since there was not enough memory to compute the tiles. Alternatively, we could execute the experiment with more robust hardware, to push the limits of the solution.

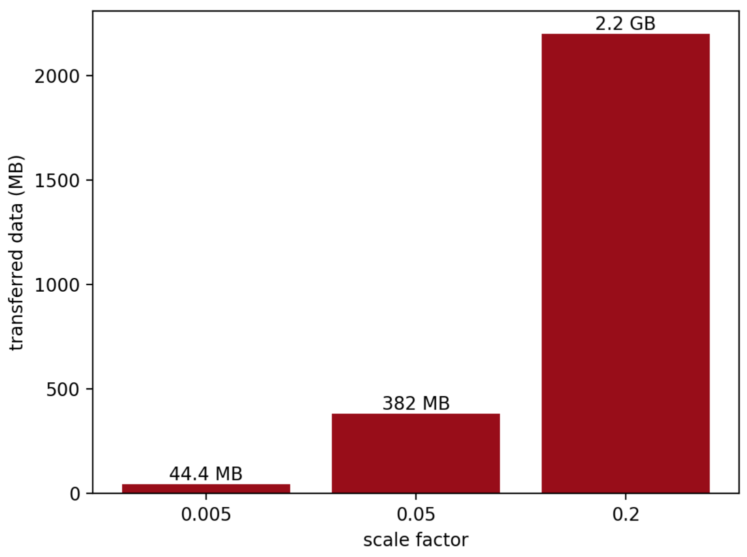


Figure 20: Experimental results regarding volume of transferred data.

Impact of the level of details

In this experiment, we evaluate what is the impact of the level of details in the load time, amount of transferred data and memory footprint. For that, we executed the same scenarios as before but for zoom level 15, which requests smaller tiles but with more level of details than zoom level 11. As shown by Figure 22, the value of the three metrics decreases. This behaviour is expected since the viewport does not include all trajectories included in a lower zoom level and therefore fewer trajectories are fetched. However, the decrease is not proportional, which means that the level of details is greater for a high zoom level, meaning that even though not all trajectories are fetched, the ones that are have more points (they are less simplified than the ones from low zoom levels), which is expected behaviour.

4.4.2 Evaluation of the Architecture with an In-Memory Tile Index

We proposed a second architecture that instead of having a tile server to send the tiles over to the client as the user interacts with the visualisation framework, we create and store the tiles in-memory at the client-side. This way we avoid the server-client data transfer and, therefore, eliminates its latency. The goal of this solution is to provide a smoother interaction with the visualisation framework, since the latency is visually noticeable when the user changes the viewport via zoom

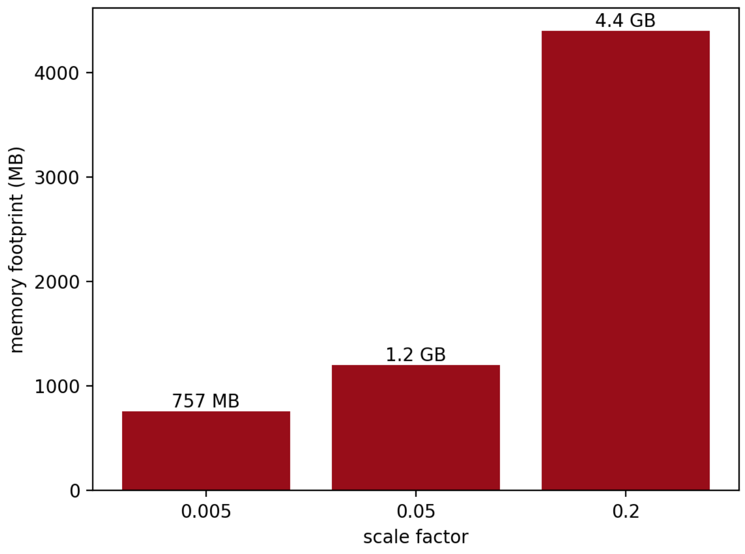
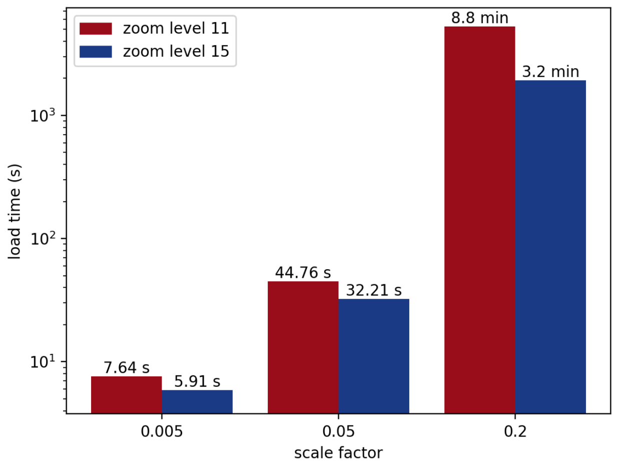


Figure 21: Experimental results regarding memory consumption.

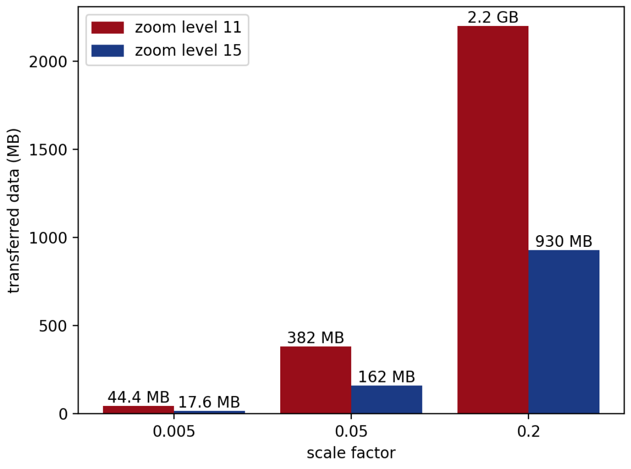
or pan. Since there is no data transferred and the load time is inconsiderable, we evaluate the FPS of the animation of trajectories.

Evaluation of frames per second

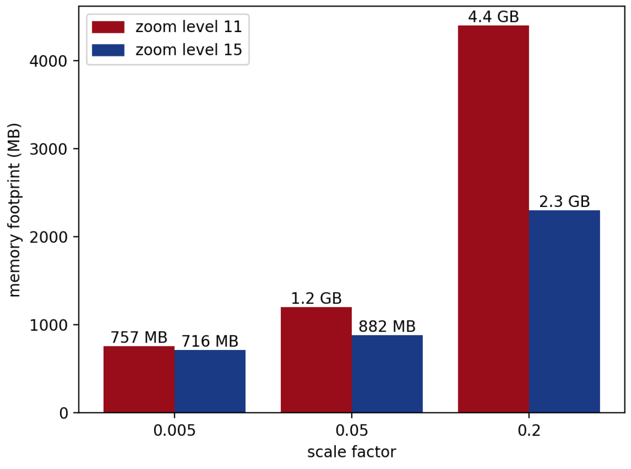
In this experiment created an animation of the trajectories with Deck.gl's TripLayer, as we did for the previous experiments. But instead of computing and sending each vector tile to Deck.gl's application, we load a file GeoJSON containing all trajectories in memory and use geojson-vt to compute the vector tiles on demand and save them in memory. Figure 23 shows that for the same viewport and dataset, the application's FPS was around 30 fps for all scale factors, higher than the solution with a tile server. We believe this happens for two reasons: (1) with tile server, each tile is requested individually and can be received apart in time, which then updates Deck.gl's layer data and triggers a rendering update. With an in-memory tile index, even though the requests are individual, they are shared by the same data structure and therefore computed and rendered as a group, triggering only one rendering update; (2) the MVT standard used by the tile server encodes the timestamps as a comma-separated string, which needs to be decoded in Deck.gl, while with geojson-vt the vector tiles are not encoded and therefore needs no further processing to render the trajectories.



(a) load time for different zoom levels



(b) transferred data for different zoom levels



(c) memory footprint for different zoom levels

Figure 22: Experimental results regarding the impact of level of details.

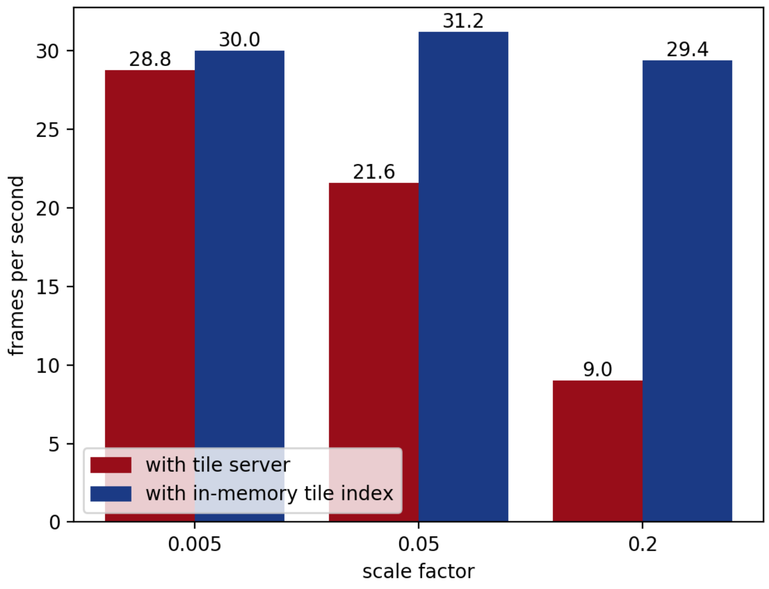


Figure 23: Experimental results regarding frames per second of animation.

5 Related Work

Previous work had solved only partially the problem of efficiently conducting visual analytics of large-scale full trajectory data from a MOD. Most of them proposed stand-alone visualization tools that load input files and use their own in-memory data structures, which is not scalable.

To our knowledge, [31] is the only work that explored visual analytics on a MOD. They propose a solution that combines a visual analytics system named V-Analytics [2] with SECONDO, an open source research prototype of MOD [16]. Both tools are integrated so that data can be exchanged between them in any direction, in a way that SECONDO's extensive set of query operations can be used to query the data and send it over to V-Analytics which enables users to visually interact with the data, preprocess, filter, select a subset of data for further analysis and also send it back to SECONDO as a data source to execute more queries with it.

The authors demonstrated the solution in a study case that aims to identify aircraft spatiotemporal patterns. By analysing real Air Traffic Control (ACT) data, they used the proposed framework to identify two less desirable landing patterns: (1) missed approach, when the aircraft is descending for landing and the pilot decides to ascend again; (2) stepwise descent landing, when the aircraft alternates between descent and cruise until it completely lands on the ground. The motivation of the study of landing patterns is to help increase safety, decrease fuel consumption, pollution and noise, and consequently improve air traffic. The dataset used in the study comprises one-day radar recording of aircraft positions over France, consisting of 17,851 trajectories and 427,651 recorded positions.

Different from [31], our solution minimises the data transfer between the visualisation framework and MOD. The data is only stored in the MOD and only vector tiles are transferred to the visualisation framework to be rendered. The user still has a chance to interact with the data, but any complex data transformation, other than for visualisation purposes, should be executed on the database side via queries. With this approach, we avoid inconsistency between the data living in both frameworks, decrease the volume of data transferred between the components and therefore promote faster interaction and rendering. Additionally, even though [31] do not make clear what are the limitations of their solution, our implementation was able to handle a lot more simultaneous points than theirs did in the study case with ACT data. Another advantage of our solution over [31] is that V-Analytics is a software that needs to be installed on the user's computer, while ours can be accessed via web browser without any previous installation.

[11] propose TaxiVis, a framework that allows users to visually query taxi trips. It supports a wide range of spatiotemporal queries, such as filtering, aggregations and origin-destination queries, and allows users to interactively pose queries over all the dimensions of the data to explore attributes associated with taxi trips. To promote high performance, TaxiVis implements efficient data storage and the use of adaptive level-of-detail rendering strategies to support interactive response times and to be capable of rendering a large number of clutter-free graphical primitives on a map. The authors proposed a system with its own storage manager, as opposed to a traditional database, to index the taxi data, since it can answer queries faster and has a smaller memory footprint than adopting SQLite or PostgreSQL.

However, TaxiVis is not suitable for full trajectory data, only the origin and destination taxi trip points are considered, which is insufficient to conduct moving objects data analytics. On the other hand, our solution considers full trajectory data. By implementing their own storage component, TaxiVis is limiting in the sense that the user would have to transfer all taxi data to the tool and index it before conducting any analysis, which depending on the size of the data it can take hours. Contrastingly, our solution allows the use of any MOD so that all data is stored in one place, avoiding consistency issues and redundancy. TaxiVis implements a strategy of visual querying, which allows users to easily query data interacting with the system. Even though it makes it possible for people without any previous knowledge of SQL to conduct visual analytics, the number of different operations one can execute is limited by what was implemented in the tool. This means that if one operation that fit our needs is not available, we would have to implement it by editing TaxiVis' source code, which requires a lot more programming knowledge than writing SQL queries. Finally, TaxiVis implements a hard limit of 1 million in the number of points it can handle, which might not be sufficient for certain big data scenarios, and it has no strategy to scale to surpass such a hard limit, whereas the architecture of our solution can be easily scaled.

Other works focused on the visual analytics dimension, developing visual strategies to interact with the data, but not focusing on how the data is stored nor adopting MOD as a data source. For instance, [23] developed TrajectoryLenses, an interaction technique to support complex filter expressions and the analysis of spatiotemporal movement data. It adopts the concept of lenses, which is basically a way of filtering data by location and/or time, which can be combined interactively to create powerful spatiotemporal queries. Like TaxiVis, spatiotemporal data can be explored in a purely visual and interactive way, instead of writing SQL-like queries. The authors also include a study case to demonstrate the applicability of TrajectoryLenses on a real-world scenario, exploring the usage behaviour of electric scooters in Stuttgart, Germany from a 2-year long dataset. In the same sense, [5] propose TRAVIC, an interactive browser-based map visualization tool that displays worldwide public transit vehicle movements. Both TrajectoryLenses and TRAVIC focuses on the expressivity of visual analytics, and does not focus on the technology to store, process and deliver the data to the visualisation client.

Many other works propose methods to effectively analyse spatiotemporal data. More specifically, in the context of mobility and transportation, [1] discuss the state of the art in visual analytics. The authors survey a list of relevant work in the field, that try to come up with forms of processing, aggregating and rendering data to generate insights for different applications, such as taxi, public transportation, vehicle fleets, human mobility behaviours, traffic modelling, forecasting, and planning. However, none of the work included in the survey tackled the problem of conducting visual analytics by adopting a MOD as a data source. They are most focused on understanding questions, problems and opportunities from a specific context and proposing solutions to those using visual analytics, not always from the perspective of big data.

6 Conclusion

This thesis tackled the problem of conducting visual analytics for big full-trajectory data, by proposing a solution that combines a visualisation framework for large-scale datasets and a Moving Objects Database. The goal of the solution is to obtain better performance by pushing any data processing to the database side, and only transferring to the client the necessary data to render the proposed data visualisations. We achieve that by a spatiotemporal vector tiling strategy, which splits the original data into equal size zoom-dependent tiles, which are stitched together and rendered in the visualisation application. The spatiotemporal vector tiling strategy was implemented for two architectural solutions with different purposes: one with a tile server, able to deal with a large amount of data that does not fit in the client's memory, and another with an in-memory tile index, which eliminates the need to compute and transfer each tile to the visualisation framework, providing faster rendering and smoother user interaction.

In our experimental evaluation we found out that, for modest hardware, our solution can produce, with high performance, data visualisations of 15 thousand trajectories with around 3 million observations, 7 times more points than [31] and 3 times more than [11], which the latter only considers origin and destination points, while our solution considered the full trajectory. We also evaluated the trade-off of building a solution capable of dealing with a high volume of data but sacrificing the rendering speed and a solution that produces fast rendering but is limited by the amount of memory available in the client's machine.

While we present an advance in the field of visual analytics of large-scale mobility data, there are still many open problems and opportunities to provide better performance, usability and ability to handle even larger volumes of data. We highlight some important directions for further work in the following.

Pre-Computation and Indexing of Vector Tiles

One of the issues identified during the experimental evaluation is that depending on the size of the dataset and the number of points that are included in a tile, there is a latency to compute and transfer vector tiles to the visualisation framework. One interesting direction for research is to come up with a method to pre-compute, store and index vector tiles either in the database or pre-cache in the tile server, so the latency to request and receive vector tiles are minimised and, therefore, provide a better experience for users to conduct visual analytics. To do so, one can come up with an algorithm to identify what tiles are worth being pre-computed, via popularity or density of points [13, 18].

Spatiotemporal Tile Slicing

Although we adapted a tiling strategy to deal with spatiotemporal data, the tiles are sliced only by geographic location. For instance, imagine that we are visualising the behaviour of buses throughout one week. In this scenario, a vector tile will have the trajectories of all the buses that cross its boundaries during 7 days. But depending on the visualisation, the animation of trajectories, for instance, we could only fetch the first day, render it and keep fetching the other days on demand. Therefore, there is a need for more research efforts that explore the idea of creating temporal vector tiles, or vector cubes, that take into consideration the time dimension in the slicing strategy in addition to the geographic location.

Architecture Scalability

The proposed architecture has some room for scaling. Even though we have not conducted an experimental evaluation in a distributed environment, it would be interesting to understand the impact of a distributed version of MobilityDB, the adoption of caching strategies such as content delivery networks (CDNs) and the deployment of both tile server and database to more robust hardware.

Integration with Kepler.gl

At the beginning we proposed to develop our implementation using Kepler.gl as the technology for the visualisation framework. Due to time limitations, the complexity of the code and lack of documentation, we opted to use Deck.gl. Even though our solution was enough for conducting the experimental evaluation, Kepler.gl is the right tool for easily conducting visual analytics, since it provides an easy interface for uploading data, creating layers and customising rendering attributes. Deck.gl, on the other hand, provides ways to create hardcoded data visualisations, meaning that the user would need a certain experience with coding to create different visualisations other than those available, which is not ideal. However, Kepler.gl is an extension of Deck.gl and our open-source implementation can be further developed into Kepler.gl layers.

Bibliography

[1] Andrienko, G., Andrienko, N., Chen, W., Maciejewski, R., Zhao, Y.: 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)

[2] Andrienko, G., Andrienko, N., Wrobel, S.: Visual analytics tools for analysis of movement data. ACM SIGKDD Explorations Newsletter 9(2), 38–46 (2007)

[3] Antoniou, V., Morley, J., Haklay, M.M.: Tiled vectors: A method for vector transmission over the web. In: Web and Wireless Geographical Information Systems, pp. 56–71. Springer Berlin Heidelberg (2009), `https://doi.org/10.1007/978-3-642-10601-9_5`

[4] Bakli, M.S., Sakr, M.A., Soliman, T.H.A.: Hadooptrajectory: a hadoop spatiotemporal data processing extension. Journal of Geographical Systems 21(2), 211–235 (2019), `https://doi.org/10.1007/s10109-019-00292-4`

[5] Bast, H., Brosi, P., Storandt, S.: TRAVIC: A visualization client for public transit data. In: GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems (2014)

[6] Bertolotto, M.: Progressive Techniques for Efficient Vector Map Data Transmission: An Overview, pp. 65–84. Springer Berlin Heidelberg (2007), `https://doi.org/10.1007/978-3-540-69878-4_4`

[7] Cartwright, W.E.: Mapping in a digital age. In: The Handbook of Geographic Information Science, pp. 199–221. Blackwell Publishing Ltd (2007), `https://doi.org/10.1002/9780470690819.ch11`

[8] Ding, X., Chen, L., Gao, Y., Jensen, C.S., Bao, H.: Ultraman: A unified platform for big trajectory data management and analytics. Proceedings of VLDB Endowment 11(7), 787–799 (2018), `http://www.vldb.org/pvldb/vol11/p787-ding.pdf`

[9] Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: The International Journal for Geographic Information and Geovisualization 10(2), 112–122 (1973), https://doi.org/10.3138/fm57-6770-u75u-7727

[10] Düntgen, C., Behr, T., Güting, R.H.: BerlinMOD: a benchmark for moving object databases. The VLDB Journal 18(6), 1335–1368 (Apr 2009), https://doi.org/10.1007/s00778-009-0142-5

[11] Ferreira, N., Poco, J., Vo, H.T., Freire, J., Silva, C.T.: 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)

[12] Gaffuri, J.: Toward web mapping with vector data. In: Geographic Information Science, pp. 87–101. Springer Berlin Heidelberg (2012), https://doi.org/10.1007/978-3-642-33024-7_7

[13] García, R., Verdú, E., Regueras, L.M., de Castro, J.P., Verdú, M.J.: A neural network based intelligent system for tile prefetching in web map services. Expert Systems with Applications 40(10), 4096–4105 (2013), https://www.sciencedirect.com/science/article/pii/S095741741300050X

[14] Güting, R.H., Behr, T., Düntgen, C.: SECONDO: A platform for moving objects database research and for publishing and integrating research implementations. IEEE Data Eng. Bull. 33(2), 56–63 (2010)

[15] Güting, R.H., Schneider, M.: Moving Objects Databases. Morgan Kaufmann (2005), https://www.elsevier.com/books/moving-objects-databases/hartmut-guting/978-0-12-088799-6

[16] Guting, R., Almeida, V., Ansorge, D., Behr, T., Ding, Z., Hose, T., Hoffmann, F., Spiekermann, M., Telle, U.: Secondo: an extensible dbms platform for research prototyping and teaching. In: 21st International Conference on Data Engineering (ICDE'05). pp. 1115–1116 (2005)

[17] information, B., 4.0, O.: On the road to mobility 4.0 (2021), https://www.basf.com/tw/en/media/BASF-Information/Innovation/On-the-road-to-Mobility-4-0.html

[18] Kefaloukos, P.K., Vaz Salles, M., Zachariasen, M.: Tileheat: A framework for tile selection. In: Proceedings of the 20th International Conference on Advances in Geographic Information Systems. p. 349–358. SIGSPATIAL

’12, Association for Computing Machinery, New York, NY, USA (2012), https://doi.org/10.1145/2424321.2424366

[19] Keim, D., Kohlhammer, J., Ellis, G., Mansmann, F. (eds.): Mastering the information age : solving problems with visual analytics. Goslar : Eurographics Association (2010), https://diglib.eg.org/handle/10.2312/14803

[20] Keim, D.A.: Information visualization and visual data mining. IEEE Transactions on Visualization and Computer Graphics 8(1), 1–8 (2002)

[21] Keim, D.A., Mansmann, F., Schneidewind, J., Thomas, J., Ziegler, H.: Visual Analytics: Scope and Challenges, pp. 76–90. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), https://doi.org/10.1007/978-3-540-71080-6_6

[22] Keim, D.A., Mansmann, F., Stoffel, A., Ziegler, H.: Visual Analytics, pp. 3341–3346. Springer US, Boston, MA (2009), https://doi.org/10.1007/978-0-387-39940-9_1122

[23] Krüger, R., Thom, D., Wörner, M., Bosch, H., Ertl, T.: TrajectoryLenses - A set-based filtering and exploration technique for long-term trajectory data. Computer Graphics Forum 32, 451–460 (2013)

[24] L’Hostis, A., MULLER, B., meyer, G., BRUCKNER, A., FOLDESI, E., Dablanc, L., Blanquart, C., TUOMINEN, A., KOSTIAINEN, J., POU, C., URBAN, M., KESERU, I., COOSEMANS, T., DE LA CRUZ, M.T., VAL, S., Golfetti, A., Napoletano, L., SKOOGBERG, J., HOLLEY-MOORE, G., CHALKIA, E., VAN DER WERF, I., BOS, F., GROSSO, S., STANS, Y., LANGHEIM, J.: MOBILITY4EU - D2.1 - Societal needs and requirements for future transportation and mobility as well as opportunities and challenges of current solutions. Research report, IFSTTAR - Institut Français des Sciences et Technologies des Transports, de l’Aménagement et des Réseaux (2016), https://hal.archives-ouvertes.fr/hal-01486783

[25] Mapbox: Specification: Vector tiles (2021), https://docs.mapbox.com/vector-tiles/specification/

[26] Nations, U.: The Sustainable Development Goals Report 2020. UN (Jul 2020), https://doi.org/10.18356/214e6642-en

[27] Pelekis, N., Frentzos, E., Giatrakos, N., Theodoridis, Y.: HERMES: A trajectory DB engine for mobility-centric applications. Int. J. Knowl. Based Organ. 5(2), 19–41 (2015), https://doi.org/10.4018/ijkbo.2015040102

[28] Ramer, U.: An iterative procedure for the polygonal approximation of plane curves. Computer Graphics and Image Processing 1(3), 244–256 (1972), `https://doi.org/10.1016/s0146-664x(72)80017-0`

[29] Ramsey, P.: Serving dynamic vector tiles from postgis (Jul 2019), `https://blog.crunchydata.com/blog/dynamic-vector-tiles-from-postgis`

[30] Revesz, P.: Spatiotemporal Interpolation Algorithms, pp. 2736–2739. Springer US, Boston, MA (2009), `https://doi.org/10.1007/978-0-387-39940-9_803`

[31] Sakr, M., Andrienko, G., Behr, T., Andrienko, N., Güting, R.H., Hurter, C.: Exploring spatiotemporal patterns by integrating visual analytics with a moving objects database system. In: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. p. 505–508. GIS '11, Association for Computing Machinery, New York, NY, USA (2011), `https://doi.org/10.1145/2093973.2094060`

[32] Sample, J.T., Ioup, E.: Tile-Based Geospatial Information Systems. Springer US (2010), `https://doi.org/10.1007/978-1-4419-7631-4`

[33] Stefanakis, E.: Web mercator and raster tile maps: two cornerstones of online map service providers. GEOMATICA 71(2), 100–109 (2017), `https://doi.org/10.5623/cig2017-203`

[34] Thasler, H.: Buffer around vectortiles (Jan 2020), `https://blog.cyclemap.link/2020-01-25-tilebuffer/`

[35] Thomas, J.J., Cook, K.A.: Illuminating the Path: The Research and Development Agenda for Visual Analytics. IEEE Computer Society Press (2005)

[36] Tufte, E.R.: The Visual Display of Quantitative Information. Graphics Press, 2 edn. (2001)

[37] Zimányi, E., Sakr, M.A., Lesuisse, A.: Mobilitydb: A mobility database based on postgresql and postgis. ACM Transactions on Database Systems 45(4), 19:1–19:42 (2020), `https://doi.org/10.1145/3406534`