# MOVE: Interactive Visual Exploration of Moving Objects

Maxime Schoemans[1], Mahmoud Sakr[1,2] and Esteban Zimányi[1]

[1]Université Libre de Bruxelles, Brussels, Belgium
[2]Ain Shams University, Cairo, Egypt

**Abstract**

Visualization is a powerful tool in understanding moving object data. There is, however, a lack of common open-source tools that can support users in this task. The main challenges are to provide rich transformations and visualizations through an interactive interface, to help users in exploring, understanding and presenting their moving object data. In this demo, we present MOVE (Moving Objects Visual Exploration), an open-source tool that integrates MobilityDB, a moving object database in PostgreSQL, and QGIS to visualize moving objects. MOVE is capable of querying and displaying moving object data through a simple interface and visualizing both static and animated spatial data in QGIS. We use Danish AIS data to demonstrate the capabilities of MOVE by presenting a set of example queries and visualizations.

**Keywords**

Spatio-Temporal Data, Moving Objects, Visualization, Open-source

## 1. Introduction

Mobility is omnipresent in our everyday life, and understanding it is becoming increasingly important for numerous practical and financial reasons. Mobility data is best understood using visualizations, which explains the interest for common open-source tools that can handle such data and create rich visualizations of it. In this demo, we present MOVE, a PostgreSQL- and QGIS-based visualization tool to interactively visualize and explore moving objects data.

The stack of PostgreSQL, PostGIS, and QGIS[1] is popular for spatial data visualization and exploration. With this work, we aim at empowering this community with a tool for mobility data visualization. This new tool uses MobilityDB[2] as a solution to mobility data management. MobilityDB is a temporal extension to PostGIS that can handle large mobility data sets (continuous moving object trajectories) and offers a wide range of transformation and aggregations operations in SQL.

The work presented in this demo thus contributes to proposing a new tool to visualize and explore MobilityDB data in QGIS. This is done through a QGIS plugin providing a simple and interactive interface for the user. The MOVE plugin is available as open-source[3] and works with the latest versions of MobilityDB and QGIS.

As related work, perhaps the most comprehensive work in the visualization of mobility data is the foundation in [1]. This work provides concepts and application examples of exploratory analysis of movement data. There is, however, a lack of open-source visualization systems that allow such exploratory analysis through an interactive and powerful interface. Niche system implementations include V-Analytics [2] and GTX [3]. The common open-source visualization tools, mainly geospatial, have little support for movement data. Users would need to integrate a complex stack of tools to obtain both rich and scalable visualizations, as assessed in [4].

In QGIS, one could use the Time-Manager plugin [5] to create animated visualization of moving objects. It provides a user-friendly GUI for visualizing timestamped geometries (i.e., discrete point-based trajectories). Recently, it was integrated as a built-in QGIS feature, called Temporal Controller. Using client-side code, the Temporal Controller can also be used to create smoothly animated visualizations of continuous trajectories. This is, however, both complex and inefficient. Complex, because it requires users to write client-side code in QGIS for doing interpolation. In addition, for the integration with MobilityDB, the spatiotemporal types are not understood by QGIS, and the users thus have to transform and handle these types manually. Inefficient, because this solution only scales to 10's of concurrent moving objects on screen. Our work leverages the capabilities of the existing Temporal Controller while hiding this complexity from the end-user and improving the scalability.

The rest of the paper is structured as follows. Section 2 presents the architecture of the MOVE plugin used to connect QGIS to MobilityDB. Secondly, Section 3 presents multiple visualization examples that can be created using simple queries through the MOVE interface. Lastly, Section 4 concludes this paper.

*Published in the Workshop Proceedings of the EDBT/ICDT 2022 Joint Conference (March 29-April 1, 2022), Edinburgh, UK*
maxime.schoemans@ulb.be (M. Schoemans); mahmoud.sakr@ulb.be (M. Sakr); esteban.zimanyi@ulb.be (E. Zimányi)


[1]www.qgis.org
[2]www.github.com/MobilityDB/MobilityDB
[3]www.github.com/mschoema/move

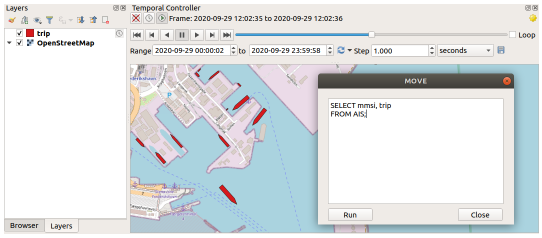



**Figure 1:** QGIS User Interface

## 2. The MOVE Plugin

The architecture of the visualization tool is composed of three parts: MobilityDB, QGIS, and the MOVE plugin linking these two systems.

MobilityDB [6] is an open-source moving object database built on top of PostgreSQL and PostGIS. It adds new temporal types to PostgreSQL that can represent moving points (`tgeompoint`) and moving rigid geometries (`tgeometry`), as well as temporal floats, integers, booleans and texts. The temporal types `tgeompoint` and `tgeometry` use the PostGIS `Point` and `Polygon` types respectively to represent the individual instants of the moving objects.

QGIS is a geospatial visualization tool. It can connect to multiple spatial data stores, including PostGIS, and display static geometry objects, such as points, linestrings and polygons. Using the Temporal Controller, it has the capability of displaying animated maps. As such, QGIS has the capability of displaying both static and animated representations of moving object data.

MOVE allows users to write SQL spatio-temporal queries and to visualize the results in QGIS. These queries are executed by MobilityDB, and the user has thus access to the complete set of operations offered by PostgreSQL, PostGIS and MobilityDB. These include for instance projection, distance, speed, azimuth, temporal aggregations, and temporal topological predicates [6]. With this rich API, users can be creative in expressing both exploratory as well as analytical queries and visualize their results. On the QGIS side, users are additionally offered a big variety of map display and symbology options.

The MOVE plugin, displayed in Figure 1, presents a simple but powerful interface to the user. When executing a SELECT query, the plugin inspects the types of the resulting columns and creates the appropriate layers in QGIS to visualize them. Columns storing PostGIS geometry types are displayed as they would have been by default. If the query returns MobilityDB spatiotemporal types, that is, `tgeompoint` or `tgeometry`, MOVE creates temporary database tables to approximate these spatiotemporal objects into a representation that uses native QGIS types, such as `LinestringM`. These tables are then added as layers in QGIS, marked as temporal, and can be animated using the QGIS built-in *temporal controller*. Additionally, indexes are built on the temporal and spatial columns of these tables. This allows users to scale up their database, without compromising the responsiveness of the display in QGIS.

It is worth mentioning that MOVE chooses by design that all the layers it creates for visualization use native QGIS representations. As such, the user has access to the whole range of styling, annotation, etc. that is available in QGIS to enrich the visualization as needed. The symbology of the created layers is always set to default and can be modified by the user. Each layer also contains all the non-geometry columns returned by the initial query, and can thus also be used by the user as usual. Specifically, columns with scalar temporal properties are also handled by the plugin and can be displayed with the use of the DataPlotly[4] plugin. These columns are displayed as 2D line plots, with the time displayed on the x-axis and the scalar values on the y-axis.

## 3. Examples of Possible Visualizations

The following subsections present examples of possible visualizations, along with the queries that we used to generate them. These queries and visualizations use a real dataset of AIS ship trajectories, that is published by the Danish Maritime Authority[5]. The data covers one day, September 29, 2020. The file size is 1.9GB and contains 8M AIS points of 1,788 different ships.

Data is loaded into MobilityDB in the table `AIS(mmsi integer, trip tgeometry, centroid tgeompoint)`. The `mmsi` attribute is a unique ship identifier. The `trip` is a temporal geometry, i.e., moving region, representing the ship movement. It represents the shape, orientation and movement of the ship. The `centroid` attribute represents the ship as a temporal point (`tgeompoint\verb`) in the case where the shape and orientation information is of no interest to the user query. It has been computed using the operation `trajectory(trip)`. Since this operation is used in multiple queries of the following subsections, we precompute the result of this operation in column `centroid` to simplify the queries. Next, we illustrate examples of possible visualizations on the AIS dataset.

### 3.1. Animated Points and Geometries

Queries 1 and 2 can be used to display columns of temporal geometries and points, respectively. Running these

[4] www.github.com/ghtmtt/DataPlotly
[5] www.dma.dk/SikkerhedTilSoes/Sejladsinformation/AIS/Sider/default.aspx

```
Query 1 (4s for 100 ships, 40s for 1000 ships):
SELECT mmsi, trip FROM AIS;
```

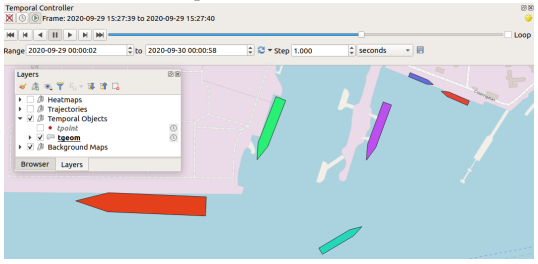



**Figure 1: Visualization of temporal geometries**

```
Query 2 (4s for 100 ships, 45s for 1000 ships):
SELECT mmsi, centroid FROM AIS;
```

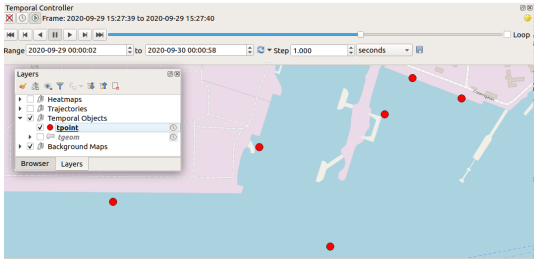
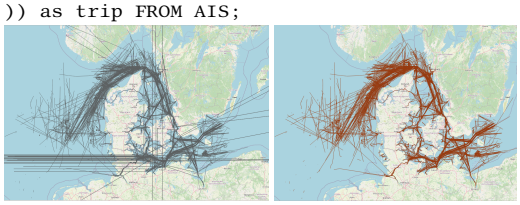



**Figure 2: Visualization of temporal points**

```
Query 3 (4s):
SELECT mmsi, trajectory(centroid) FROM AIS;
Query 4 (8s):
SELECT mmsi, trajectory(atPeriodSet(
    centroid, getTime(atValue(
        speed(centroid) #< 30, True))
)) as trip FROM AIS;
```



**(a)** **(b)**

**Figure 3: Ship trajectories, before and after cleaning**

queries in the plugin interface will automatically add a layer in QGIS with the temporal property activated. This displays the ships as animated polygons and points respectively, and it is possible to interact with the animation using QGIS Temporal Controller.

Figures 1 and 2 display a snapshot of the layers resulting from Queries 1 and 2, respectively. Notice that the added layer is already marked as temporal, as indicated with the clock symbol on the right of the layer names. Using the Temporal Controller visible at the top of the image, this data can be explored and animated.

## 3.2. Interactive Data Cleaning

When working with mobility data, errors can occur at multiple stages of the pipeline, and the data received by the database is thus not free of errors. Visualizing this data is essential to not only determine the nature of the errors but also verify that the cleaning process was successful. With the use of the plugin, this cleaning and verification process can be done interactively.

After loading the data in table `AIS`, we can display the trajectories of the ships using Query 3. Figure 3a displays the result of running this query in the plugin interface. In this figure, we can see long straight lines passing through solid terrain, which are errors in the data. These lines are present because some data points are not placed on the actual trajectory of the ship, but rather on an incorrect location far from the actual position of the ship.

A possible way to remove these errors is to compute the speed of the ship, and remove the segments of the trajectory having a speed higher than a certain threshold, such as 30 m/s. This can be done through the use of multiple MobilityDB functions on temporal points as displayed in Query 4. Figure 3b shows the result of Query 4, and the erroneous lines visible in Figure 3a have indeed been removed. If this were not the case, a new query could have been run to continue this interactive cleaning process.

## 3.3. Heat Map of Trajectories

Query 5 constructs a heat map using a second table `Grid`, storing the geometries of a regular square grid enclosing the vessel trajectories. Using the QGIS layer styling, we can then color the grid cells with a high weight. This creates the visualization shown in Figure 4.

The lines shown in Figure 4 display grid cells that were traversed by multiple ships on the same day, which could indicate navigation routes or zones with high traffic.

## 3.4. Traversed Area

MobilityDB offers a wide range of operations to process temporal points and geometries, such as distance, intersection or restriction functions. An example of a complex operation on temporal geometries is `traversedArea`. This function computes the area traversed by a temporal geometry as a PostGIS polygon and can be used, for example, to compute the closest point of approach of a ship to a fixed point on land. Query 6 computes the traversed area of a ship, and the result of this query is displayed in Figure 6.

## 3.5. Temporal Attributes

DataPlotly is a powerful plugin allowing QGIS to display attributes of a table using the Python Plotly API. This

```
Query 5 (> 1min): SELECT cell, count(*) as weight
FROM Grid, AIS WHERE intersects(centroid, cell)
GROUP BY cell;
```

**Figure 4: Heat map of the ship trajectories**

```
SELECT mmsi, trajectory(centroid) FROM AIS;
```

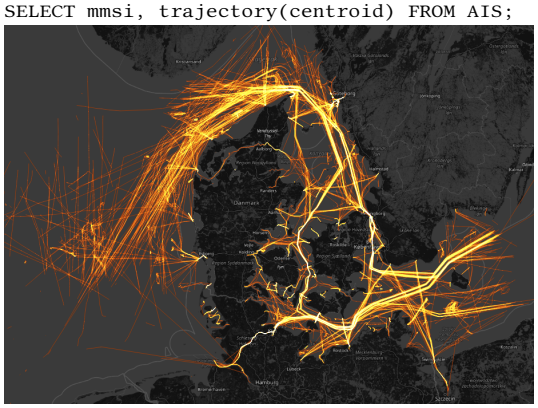


**Figure 5: Density of ship trajectories**

```
Query 6 (2s):
SELECT mmsi, trip, traversedArea(trip)
FROM AIS WHERE mmsi = 538090508;
```

**Figure 6: Traversed area of a ship arriving at the docks**

```
Query 7 (5s): SELECT port, geom,
    tcount(atValue(
      tintersects(centroid, geom), True)),
FROM Ports, AIS
WHERE port = 'Skagen'
   AND intersects(centroid, geom)
GROUP BY port, geom;
```

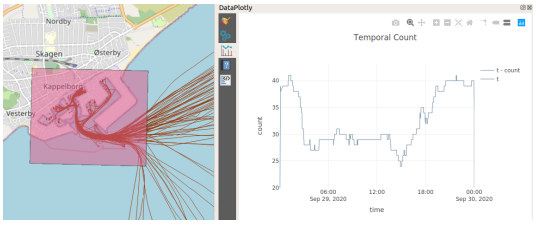



**Figure 7: Temporal count of ships at the port of Skage**

```
Query 8 (> 1min):
SELECT a.mmsi AS mmsi1, b.mmsi AS mmsi2,
    trajectory(atPeriodset(a.trip, getTime(
        atValue(tdwithin(a.trip, b.trip, 100),
            True)))) AS traj1,
    trajectory(atPeriodset(b.trip, getTime(
        atValue(tdwithin(a.trip, b.trip, 100),
            True)))) AS traj2,
    getTimestamp(NearestApproachInstant(
        a.trip, b.trip)) AS t,
    nearestApproachDistance(a.trip, b.trip) AS dist
FROM AIS AS a, AIS AS b WHERE a.mmsi < b.mmsi
```

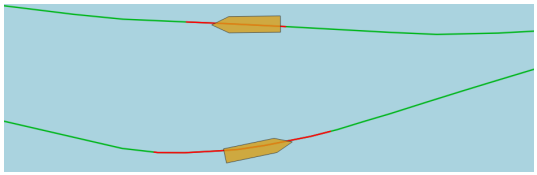


**Figure 8: Close encounter of two ships (< 100 m)**

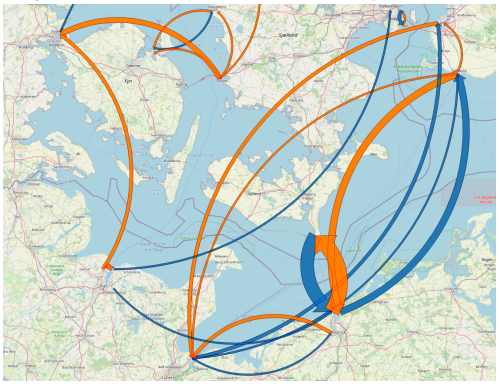


**Figure 9: Flow map between several ports**

tool allows the creation of different plot types, such as scatter plots, histograms, bar plots and more.

Visualizing temporal attributes, such as temporal floats, integers or Booleans can also be of interest. Our plugin thus allows DataPlotly to display these attributes as lines in a scatterplot, where the $x$ and $y$-axes correspond to the time and value dimensions respectively. Figure 7 displays the result of Query 7, which counts the number of ships in the port of Skagen as a temporal integer. The figure shows the extent of the port on the

map and the count of the ships as a stepwise function in a scatter plot.

### 3.6. Close Encounter

As the last example, the MobilityDB `tdwithin` operation can be used to find out when two temporal points are within a given distance of each other. Query 8 uses this operation to restrict the trajectories of ships to the times when they are within 100 meters of each other. This can for example be used to display close encounters of two ships at sea. Figure 8 shows a zoom of one close encounter returned by Query 8.

### 3.7. More Visualizations

The described examples only form a subset of the visualizations possible using the plugin. More complex queries can be also executed to create more advanced visualizations, such as the flow map displayed in Figure 9. This can be achieved by combining both the wide range of operations in MobilityDB as well as the wide range of styling features in QGIS. Indeed, since the plugin adds QGIS layers to visualize the results of the queries, the users are then free to adapt the symbology of the data as they see fit. For example, Figure 1 displays every ship in a random color, and Figure 5 corresponds to Figure 3b with different styling, giving a better impression of the density of the trajectories.

## 4. Conclusion

This paper presented MOVE, a visualization tool that integrates with MobilityDB and QGIS. MOVE presents a simple interface to interactively query and visualize spatial and spatiotemporal data. Using the extensive MobilityDB API and the large number of visualization options available in QGIS, the users can easily build rich visualizations. Especially, MOVE creates transformations of the MobilityDB temporal types to be able to build static and animated visualizations of moving objects in QGIS. MOVE delegates the data processing to MobilityDB, which can handle large data sets and offers a wide range of transformation and aggregation operations in SQL. This solution builds on the open-source stack of PostgreSQL, PostGIS, MobilityDB and QGIS, and is thus available and ready-to-use for its large community of users and developers.

## References


[1] G. Andrienko, N. Andrienko, P. Bak, D. A. Keim, S. Wrobel, Visual Analytics of Movement, Springer, 2013.

[2] N. Andrienko, G. Andrienko, V-analytics (a.k.a. commongis), 2010. URL: http://geoanalytics.net/V-Analytics/.

[3] H.-P.-I. Computer Graphics Systems, Gtx - geo temporal explorer, 2021. URL: https://www.gtx-vis.org/.

[4] A. Graser, M. Dragaschnig, Open geospatial tools for movement data exploration, KN-Journal of Cartography and Geographic Information 70 (2020) 1–8.

[5] A. Graser, Visualisierung raum-zeitlicher daten in geoinformationssystemen am beispiel von quantum gis mit "time-manager"-plug-in, in: Proceedings of FOSSGIS2011, 2011, pp. 73–75.

[6] E. Zimányi, M. Sakr, A. Lesuisse, MobilityDB: A Mobility Database based on PostgreSQL and PostGIS, ACM Transactions on Database Systems 45 (2020) 42.