Tramzone: An Interactive GeoData Visualization of Zürich

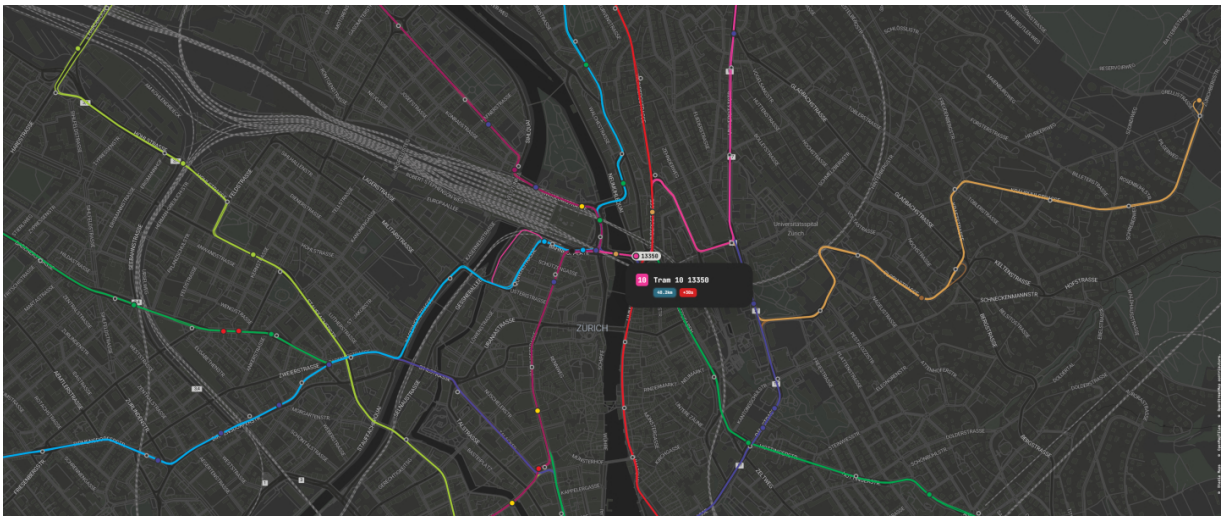
Alec Franco
francoa@ethz.ch

Jonas Tham
jotham@ethz.ch

Alexis Elisseeff
aelisseeff@ethz.ch

Martin Shen
mashen@ethz.ch

ETH Zürich
Zürich, Switzerland



Abstract

As more and more people have taken public transport in recent years, being able to get reliable information about the transport network situation has become a need for many. There already exist multiple solutions to plan ones route efficiently. However, a proper mainstream visualization of public transport still doesn't exist. Therefore, we introduce Tramzone, a web application that visualizes the live situation of public transport. The application entails the basic functionality of displaying the trams' live locations and updating itself based on service interruptions. By focusing solely on the trams in the city of Zürich, we focus on visualizing their locations in the best way possible.

1 Introduction

Many people experience the problem of not knowing where their tram is when they are on a tight schedule and depend on its punctual arrival. While some applications inform users of the delay of public transport, none of them visualize the location of the delayed vehicle. This leads to the users having to infer if it is worth waiting for the tram or using different means of transport.

The existing solutions in the Swiss market are the SBB app, Google Maps, and Geops[1]. What differentiates the first two from each other are two points. First, the SBB app has more accurate information about the vehicle and displays the proper delays compared to Google Maps. However, Google Maps visualizes the lines of public transport and the trams' live location, which the SBB app does not. It is important to note that the visualization of the trams made by Google is very unreliable, as they are often not displayed on the map.
Geops, on the other hand, visualizes the live location of trains and buses in Switzerland. It provides the users with the routes of each vehicle and the expected arrival time to each station. However, Geops doesn't support the trams in Switzerland.

Therefore this report introduces Tramzone, a web application that displays the live location of all trams in the city of Zürich. By constantly getting live data of the trams' delays and which stations they visited, it interpolates their accurate location and visualizes it on a map. The Web app attempts to inform the user about the nearest trams they could use, and about any other service interference e.g. a snowstorm.

2 Design

As visualization is the main functionality of our web app, having a clean design was the highest priority. The approach we took was to first start with a coarse design, and then iterate over it to end up with a final, clean result. The goal was to have the static tram map provided by ZVV as an interactive live map. The design of the tram and station icon, as well as the colors of the tram, are all the same design made by ZVV. This enables new users to intuitively know what each icon and color represents.

Hence our first decision was to find a proper map to use as our basis. We iterated over three different types of maps: the first one was the basic OpenStreetMap (Figure 1), the second one was the grayscale version of the first one, and the last one was the StadiaMap (Figure 2). The reason why we moved from a colored map to a grayscale one was the amount of color clutter we had. Since we have hundreds of trams at a time moving on a map and

[1] Link to the Geops Tool

each one of them is colored, we had big issues with contrast. The second iteration was made due to the impact on the performance of the grayscale OpenStreetMap.
[h]

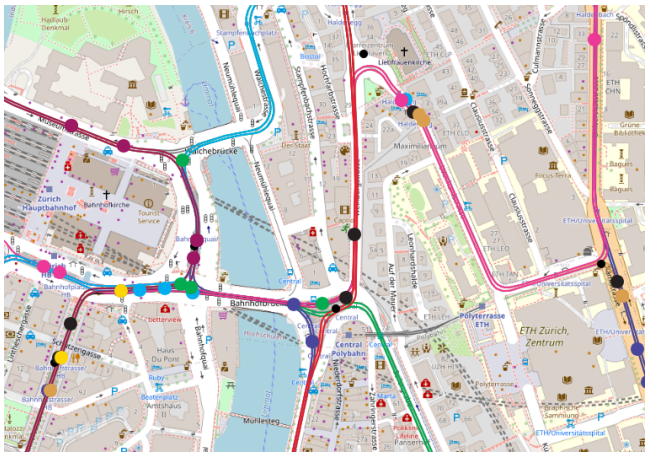


Figure 1: The old design using the colored OpenStreeMap

The StadiaMap, next to fulfilling our criteria which the other two maps didn't, also has favorable features. One of those is the dark mode, which we use to make a distinction between daytime and nighttime service.

To keep on reducing the amount of clutter caused by colors, another design direction we went for was to only color the important features of the application. Those are the trams and the corresponding routes they take, the user's live location, and the delay of the tram. This leads the user to easily find the information they want: the live location of the tram.

Lastly, the biggest issue we had with our visualization was the amount of clutter generated by the features alone. Having the lines, the trams, the stations, and the user's location all displayed at once impacted the visibility of the trams' live locations greatly. Therefore, the starting view of the app doesn't show the lines anymore. Those are only shown in two cases:

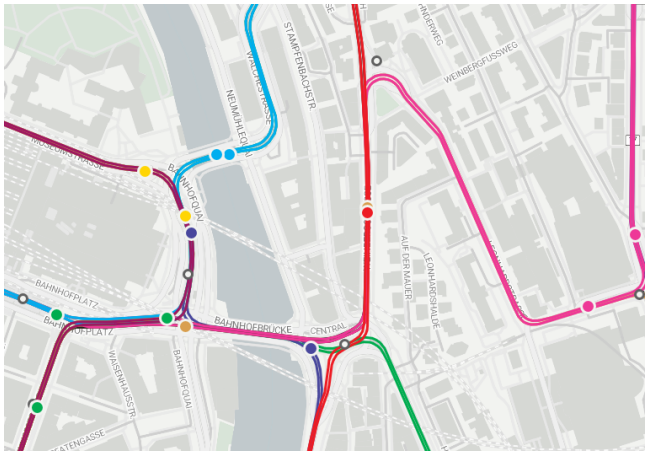


Figure 2: The new design using the grayscale StadiaMap

(1) The user clicks on a tram. The line of the tram will then be displayed on the map to guide the user to their desired destination.

(2) The user toggles the line layers on.

Consequently, a filter has been added to the application to allow the user to freely decide which information he wants to see.

3 Implementation

To be able to immediately work on the project and solve the problem as quickly as possible, we decided to use NextJS as our framework. The main reason was that we all had experience with it and it fit our needs.

After the quick decision-making on the framework, we tackled the parsing of the data. The data came from 3 different sources: the static datasets, which consist of the stations and lines, including names, coordinates and geometry; the static GTFS timetable, containing the planned public transport schedule for all of Switzerland in the next 3-4 days; and the GTFS realtime and service alert APIs, which provide live information about current trip changes and delays. The datasets and map were provided by Stadt Zürich[2] and Stadia Maps[3]. The GTFS static and realtime data was fetched from Open Data-Plattform Mobilität Schweiz[4].

Figure 3 presents an overview of the backend structure. We will briefly go into more detail on the parsing process. The station and line datasets can be directly converted into our internal Station and Line types and saved as JSON files. A Line contains a list of Segments, each with DIVA id references to its start and end station, the current direction and sequence number along its line, and a GeoJSON LineString describing the geometry of the rails between two stations. All coordinates are also converted from Swiss LV95 to standard GPS at this point. They can then directly be drawn on the map.
Then we determine the most recent Monday or Thursday after 15:00, and fetch the corresponding GTFS timetable data. This is provided as a zip file containing a number of large csv files. We first filter routes.txt by vehicle type and agency id, and save the resulting 16 routes as JSON. These do not contain much information, we mainly care about route_id and name. Next, trips.txt contains all scheduled instances of trips along routes. We filter by route_id and convert to a list of type Trip. This provides us with the important trip_id and service_id, but does not yet contain any time information. calendar.txt indicates between what dates and on which weekdays each service_id is active. calendar_dates.txt additionally specifies dates on which a certain service_id is explicitly added or removed. stop_times.txt is a list of 10 million stops containing arrival and departure as time of day, stop sequence number and stop_id, with each stop belonging to a trip_id. We can now combine all of this into a list of TramTrip objects. Each TramTrip has a unique trip_id, an associated route_id and service_id, as well as a list of stops sorted by sequence, each with stop_id and arrival and departure as timestamp offsets from midnight. They are then split into 7 separate JSON files, based on service activity per weekday. This concludes parsing, which only has to be run twice per week.
When the /api/trams route receives a request, it reads the current weekday's TramTrip file and makes a (potentially cached) request to the GTFS realtime api. This returns a list of updates, containing delays and skipped stops associated with a trip_id. We convert each

[2] https://data.stadt-zuerich.ch/dataset/
[3] https://stadiamaps.com/
[4] https://opentransportdata.swiss/en/group/timetables-gtfs

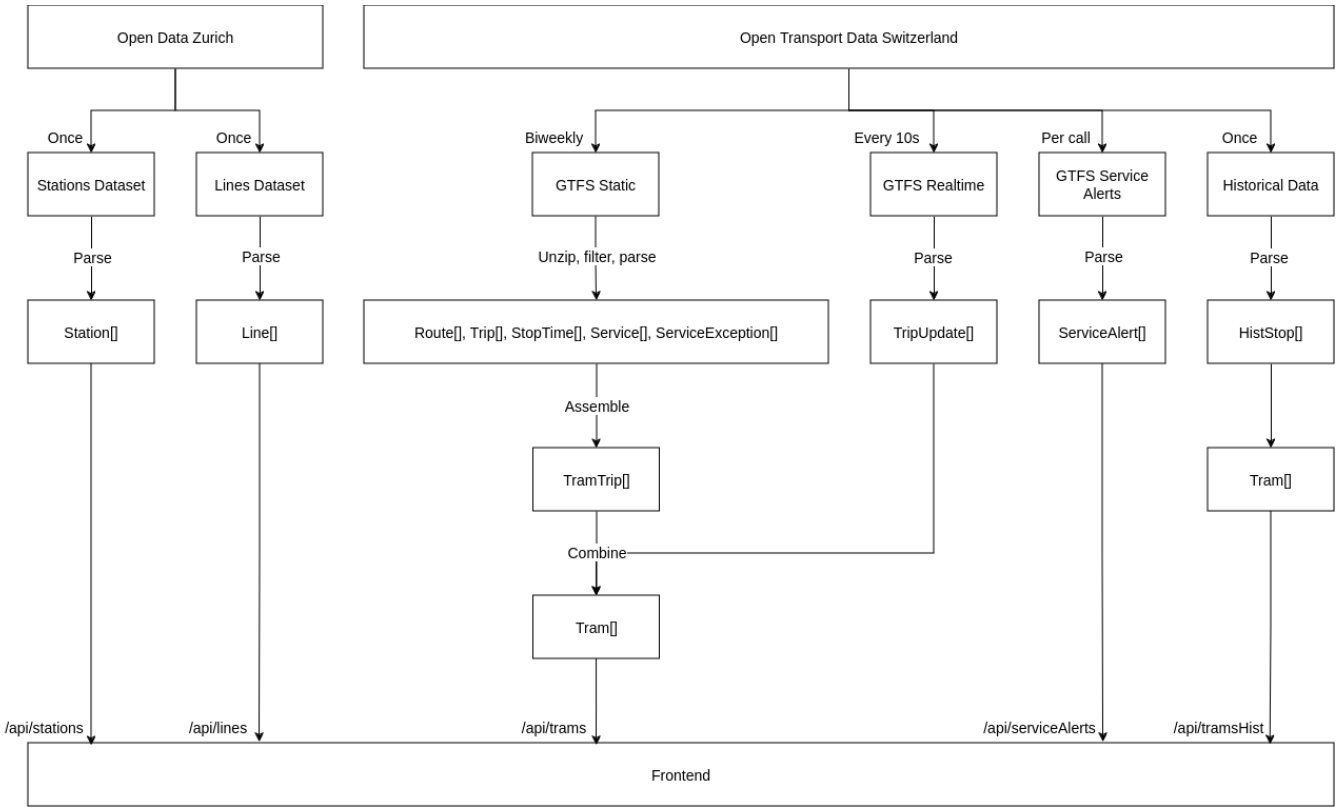


Figure 3: Backend structure

TramTrip into a Tram object, with stop times adjusted by delays and converted to a real timestamp on the current day, as opposed to the GTFS offset. We then check for service exceptions, malformed stop time data, filter by currently active trams, calculate route progress, and return the list of active trams to the frontend.
Service alerts are simply another GTFS api, again filtered by region and converted to an internal type.
Historical data is provided as a csv file containing a list of recorded stops, which requires different parsing but again results in a frontend-compatible Tram list.

While many countries in Europe support the geolocation of their public transport vehicles, Switzerland doesn't. This forces us to infer the live location of the trams based on their departure time from a station, their delays, and their expected arrival time to the next station. We define a progress number that represents the tram's current position on its route. The frontend updates this once per frame and uses it to display the tram's position on the map by linearly mapping its progress onto the geometry of the corresponding line segment. To represent inactive trams accurately, we also set each trips first stop arrival time to the departure of the previous trips final stop. This way, when a tram reaches its final stop and disappears from the map, it is seamlessly replaced by its successor and waits there until departure.

As the amount of data that we need to parse is quite large, we encountered slow-downs in performance and crashes due to memory limits. Since the data couldn't just be read into memory directly, we extensively used streams and pipes to process it line by line. As the GTFS data contains information for all of Switzerland, filtering by region and route before parsing was repeatedly necessary. To make cross referencing of information at this scale possible, we frequently used sets and maps for id indexable values. Furthermore, we partitioned the data into 7 separate files whenever possible, one for each service day, to ensure our api routes only had to load and process the currently relevant trips. Lastly, to ensure we operate within the memory constraints, many parsed files had their results written to disk. Thus each parsing stage streams only the necessary data and writes the results to a file immediately, before freeing memory and moving on to the next stage.

In addition to raw size, the data contents caused some issues. GTFS stop times are defined as HH:MM:SS strings representing offsets from midnight. Trips after midnight are considered part of the previous service day, so hour values above 24 are possible. This means night trams need to be written to the next days TramTrip file as well, with adjusted offsets, while referring to a different service day schedule. This, in addition to countless timezone edgecases, complicated the system considerably. Night trams are also treated like regular trams on their usual routes in the schedule, but are in reality buses, often with different stop sequences. Some of the timing data provided by the schedule also simply didn't make sense. Departures before arrivals and instant travel between stations required some intervention on our end. Some line segments were also missing from the dataset and had to be added manually. For unknown reasons, the dataset for line 11 contained a detour to the airport on the segment connecting Glattpark and Oerlikerhus. This caused tram 11 to suddenly accelerate to mach 5 and race to the airport, overtaking all other trams in its way, before looping back around and continuing to the next stop as normal.

4 SHOWCASE

Tramzone's main use case is to give users information about the current situation of the trams' network. We now present a hypothetical scenario to showcase the interaction of an average user with the web application.

The scenario consists of some service interruptions and delays that occurred in the network. The user opens Tramzone to get an overview of the network's situation. Once opened, the user is presented with a map and all the tram's live location. They will notice that for one, fewer trams are moving than usual, and a popup that can be interacted with. By clicking on this popup, the user will be provided with the recent anomalies in the tram's network, e.g. a collision happened at the "Zürich, Central" station.

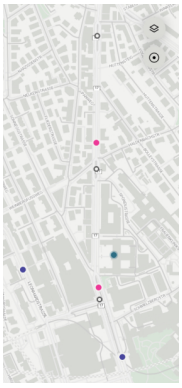


Figure 4: First view that the user is confronted with

In the case that the user's tram being canceled due to service interruptions, the user can infer from the map which trams still run and which other tram they should take to reach their destination. On the other hand, if their tram is not canceled, the user can click on it to receive more information about the vehicle. This interaction will lead to the user seeing an overlay that informs the user about the delay of the tram and its distance from the user's location.

Lastly, for a more personalized view, the user has the option to click on two buttons. To center the map around their current location, the user clicks on the lower button of the two. Using this new view, the user can more easily infer which trams they could reach and take. The upper button opens up a filter window, with which the user can toggle some layers on or off. With this, the user can decide how much information they want at once on the screen.

5 DISCUSSION

Overall, the resulting tram locations are surprisingly quite accurate. The main limiting factors are the accuracy of GTFS realtime updates and our interpolation method between stations. We currently simply scale current time and predicted arrival times linearly along the length of each segment, which implies trams move at a constant speed. This could be improved by weighting each point along a segment with a speed value, either set manually or based on local curvature, since trams are more likely to speed up on long straight sections and slow down in corners. Another factor to consider are intersections, since trams spend much of their travel time waiting there. A very impactful influence that we cannot meaningfully predict however is traffic. As such, it is not practically feasible to reflect tram locations with perfect accuracy without GPS information. The current system still works quite well, since arrivals and departures at stations are as accurate as possible within the constraints of GTFS, and the user can easily intuit some of the above factors and account for them.

We also observed some interesting tram behaviors. Since each tram's first trip of the day starts at a depot, you can see them appear at the closest station and then make their way to their usual routes. Likewise, at the end of the day, trams slowly converge on the five VBZ depots. This leads to cases where trams travel between stations that are not part of their route, which requires additional considerations for the segment mapping. There are also many occurrences of physical rail layout limitations affecting tram trips. A particular example is line 14, which leaves the depot at Sternen Oerlikon, moves up to Bahnhof Oerlikon, and ends its trip there. It then uses a turning loop to switch tracks, begins a new trip, and heads back down to Sternen Oerlikon before proceeding to its actual route.

A related issue occurs due to trams often having different precise stop locations at each station. A relevant example is ETH/Universitätsspital. There are 3 distinct locations for trams to stop, which are quite far apart, yet the GTFS stop id does not distinguish between them. In the current implementation, trams always prefer their own route whenever possible and stop at the corresponding segment endpoints, which generally do not line up with the station icons. A future improvement would be displaying each stations precise stop locations explicitly. In addition, the current limitations mean that a tram 6 coming up from HB will always turn right at ETH, following its route to Zoo. But trams returning to the depot at Langmauerstrasse should actually take a left turn. On our map it would arrive at the stop in front of HG, suddenly jump to the stop in front of CAB and continue towards Haldenbach. A robust approach to solve this could utilize graph algorithms to determine the correct connecting segments on nonstandard routes. However solving this issue fully to support arbitrary reroutes would require manually defining segments for all possible paths across each intersection.

Some more improvements can be made in the frontend visualization. Currently, movement is very jittery due to OpenLayers drawing mechanisms and our naive per frame update implementation. Instead keeping track of previous positions and animating tram movement to the current position could lead to a smoother user experience. Some of the initially planned features are also not implemented, such as the search bar and station/tram detail sidepanels. The required infrastructure and data are already available, so these features could be implemented fairly easily in the future.

6 CONCLUSION

We have shown a basic implementation to solve the problem of visualizing the public transport situation. Tramzone is focused solely on the core features of solving such a problem, i.e. the visualization of the trams' live locations. Therefore our web application should be taken as a foundation to build on top of.

The paper also demonstrates that one can show the precise location of a tram without having its geolocation data. The progress of each tram is calculated and used to map the vehicle to the right location. Due to some inaccuracies for uncommon behaviors of the trams like using unknown routes to reach the proper station, future work can be done to cover those edge cases.

Lastly, we have shown that handling the huge amount of

data for GTFS with limitations in memory and computational power due to the nature of the project is possible. With proper read and write management, as well as fully utilizing streams to read and process files, memory usage is kept to a minimum. Once the limitations are removed, we can more easily serve the data quicker. Additionally, there is still room for improvement in optimizing the parser and backend.