TRAVIC: A Visualization Client for Public Transit Data

Hannah Bast
University of Freiburg
79110 Freiburg, Germany
bast@informatik.uni-freiburg.de

Patrick Brosi
geOps
Kaiser-Joseph-Str. 263
79098 Freiburg, Germany
patrick.brosi@geops.de

Sabine Storandt
University of Freiburg
79110 Freiburg, Germany
storandt@informatik.uni-freiburg.de

ABSTRACT

We present TRAVIC, a thin browser-based client that is able to display smooth vehicle movements on a map. The focus is on visualizing world-wide public transit vehicle movements in an interactive way. But we also investigate other use cases, for example, traffic simulation. We describe in detail which server requests are fired and how the received data is handled. We also provide a performance evaluation conducted on several browsers. We show that, in combination with an efficient back-end, TRAVIC is able to display many thousands of vehicle movements in real-time. Our prototype implementation can be accessed under `http://tracker.geops.ch` .

Categories and Subject Descriptors

H.3.5 [**Information Systems Applications**]: Web-based service

Keywords

Real-time Visualization, Spatio-temporal Queries, Public Transit Networks

1. INTRODUCTION

Live maps for various kind of vehicles are available nowadays in form of web applications. For example, the flightradar for planes[1], the vessel tracker[2], and live train movement visualizations for Switzerland[3] and Germany[4]. There are also live maps for local traffic, for example, subways in Munich[5], or tubes[6] and buses[7] in London. Several transit agencies provide position visualizations of their vehicles. However, all current live maps are either restricted to a certain transportation mode (as train) or to a very small area. The main reason for that (besides data availability issues) is that the data associated with the complete public transit movement of a whole country (or even of the whole world) is very large. For a smooth visualization, this data has to be adequately processed and displayed in real-time.

For example, the German train network alone consists of only about 6,650 stations and about 600,000 times per day a train departs from a station. Including local traffic, there are about 250,000 stations and the number of departure events approaches 15 millions per day. In a metropolitan area like New York, thousands of vehicles move around at any point of time during the day. Visualizing this many vehicle movements on a zoomable and draggable map presents a challenge to both the back-end and the client.

We present a client implementation that in combination with a suitable client/server architecture can display many thousands of smooth vehicle movements projected onto a map. As our main application is the visualization of public transit data, we call our client TRAVIC (TRAnsit VIsualization Client). We will discuss further use cases of our client towards the end of the paper. TRAVIC makes use of Leaflet, an OpenSource JavaScript library for interactive web maps. Leaflet can handle most of the available map tile formats, but is mostly used with Google Maps or OpenStreetMap (OSM) tiles. Figure 1 provides a screenshot of TRAVIC visualizing vehicles in the Netherlands, using tiles from the OSM Transport Map[8] layer.

In this paper, we describe how TRAVIC handles the data received from the back-end, explain how the vector layer is built and how smooth vehicle movements are realized.

2. CLIENT-SERVER COMMUNICATION

We assume the availability of a server which holds static timetable data. Our system is built on freely available GTFS feeds[9], but of course other data sources could be used as well. The advantage of GTFS feeds is the already high coverage, as many agencies world-wide provide data in this format. Moreover there is an extension called GTFS-realtime, which allows to report delays and route changes. For example, public transit in the Netherlands is completely covered by GTFS and GTFS-realtime. From the GTFS feeds the server extracts vehicle trajectories. A trajectory is a sequence of spatio-temporal waypoints. Each waypoint is

[1] `http://www.flightradar24.com/`
[2] `http://www.marinetraffic.com/de/`
[3] `http://swisstrains.ch`
[4] `http://bahn.de/zugradar`
[5] `http://s-bahn-muenchen.hafas.de`
[6] `http://traintimes.org.uk/map/tube/`
[7] `http://busestraintimes.org.uk/map/london-buses/`
[8] `http://wiki.openstreetmap.org/wiki/Featured_tiles`
[9] `https://developers.google.com/transit/gtfs/`

SIGSPATIAL '14, Nov 04-07 2014, Dallas/Fort Worth, TX, USA
ACM 978-1-4503-3131-9/14/11.
http://dx.doi.org/10.1145/2666310.2666369

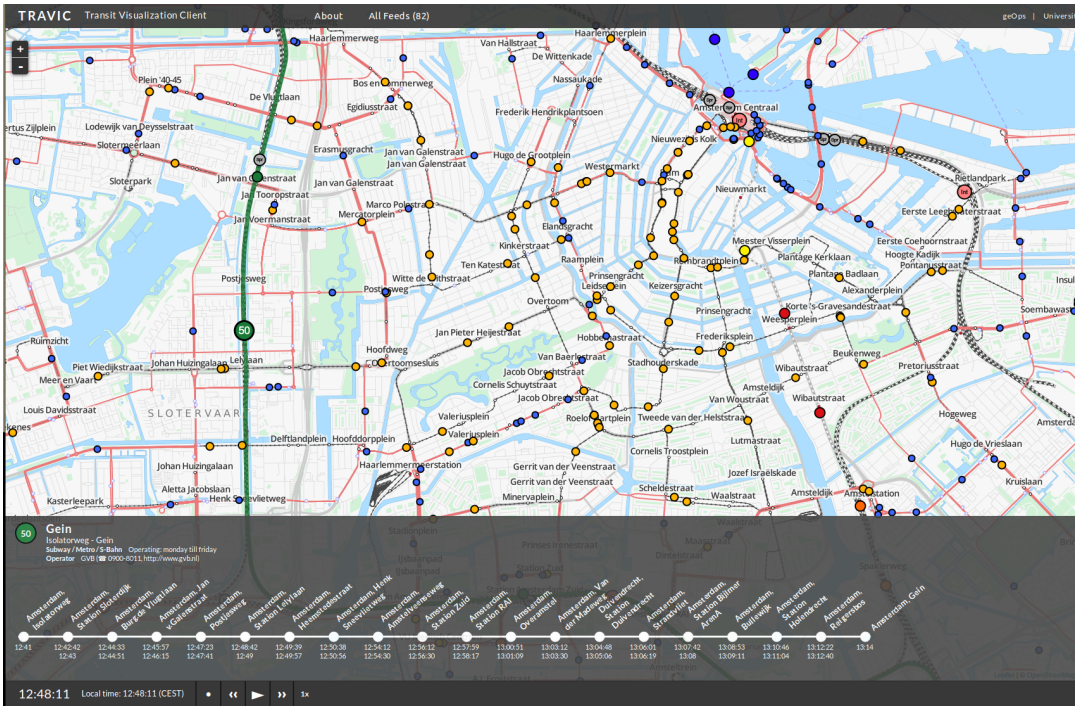

Figure 1: Visualization of long-distance and local traffic with TRAVIC. If the user clicks on a specific vehicle, the complete route with arrival and departure times is displayed (and also occurring delays).

specified by coordinates x, y in the plane and a timestamp t. So a trajectory describes how a single vehicle moves through space and time. The timestamps correspond to vehicle arrival/departure times according to the schedule, but can be updated when real-time delay information is available.

One possible client-server interface would be to let the client fire periodical position requests for all vehicles inside a spatial bounding box (the actual map view). This interface is used for most live maps based on GPS and sensor data, see e.g [1], [2] or [3]. But for temporal coverage, the frequency of such requests has to be high (especially if vehicle movements should be smooth). This results in a huge amount of client-server communication. Moreover, if the connection is interrupted there is no fall-back and vehicle movement stops on client side. To avoid these drawbacks, we use another interface based on look-ahead queries. Requests then have the form of a spatio-temporal bounding box. This means that the client asks for all vehicle trajectories which intersect the map view in a certain time interval. The server then has to identify the relevant trajectories and crop them to the requested bounding box. These partial trajectories are then send back to the client. The client then iterates over the partial trajectories, computes new vehicle positions, and moves the marker accordingly. In many GTFS feeds, spatio-temporal waypoints are only provided at stations but not in between. Therefore, temporal and spatial interpolation is necessary for smooth movement visualization. Because trajectories are basically piecewise linear curves, these interpolations can be performed on the fly.

With this interface, a new server request has only to be fired by the client after the previously requested interval expired (minus some buffer to deal with network latency etc.). In our implementation, TRAVIC fires a spatio-temporal request every 60 seconds or after the view box exceeds the bounding box of the current set of (partial) trajectories (due to dragging or zooming). As TRAVIC can send arbitrary spatio-temporal requests to the server, it is also possible to "fast-forward" vehicle movement by a factor of up to 60, and thus visualize vehicle trajectories of entire days.

Because the actual drawing of the vehicle takes up most of the computation time, TRAVIC cannot simply redraw the whole map at each interval. We describe an effective method to update the canvas in the next section.

3. THE TRANSIT LAYER

The common way to display locations on web maps is to draw a marker through the map's own API. Many transit maps handle vehicles as map markers and draw them by using some map method that usually takes latitude and longitude values as coordinates. The marker is then drawn with one of two methods. Either as a single HTML element, usually an `<IMG>` wrapped inside a `<DIV>`. This approach is used, for example, by Google Maps or Bing. Or the marker is an actual vector object on an SVG layer. This approach is used, for example, by OpenLayers or Leaflet. While vector layers are, in general, more efficient when it comes to displaying thousands of markers, the basic method to place markers on them is still a method accepting latitude (ϕ) and longitude (λ) coordinates as parameters. As mentioned above, this requires the map to do a projection of ϕ and λ onto the map plane, which, in this case, is the screen itself. For a few hundred markers that are positioned *once*, this computation is a negligible one-time cost. But consider public transit in New York during the morning rush-hour, with up to 4,000 vehicles moving around at each point in time. For a smooth visualization, each vehicle marker should be updated at least every 50 ms. If we positioned markers using latitude/longitude coordinates, there would be $4{,}000 \times 20 = 80{,}000$ projections *per second*. In a JavaScript environment or on a mobile device, this is too much for the client to handle.

Therefore, we let the server project trajectory waypoints onto the map plane and sent projected pixel coordinates to

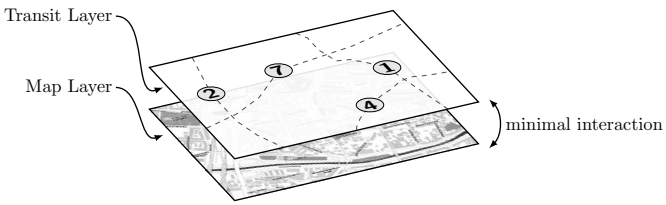

Figure 2: General architecture of TRAVIC and the Transit Layer.

the client. TRAVIC leverages this by bypassing the map service almost completely. It primarily builds on Transit Layer, a vector layer we developed for Leaflet. It is especially designed to display vehicles of any kind moving on trajectories. Interaction between the map API (Leaflet) and the Transit Layer only consists of a few callbacks responding to map dragging or zooming. The Transit Layer is based on Raphaël, a vector library for JavaScript for the sake of improved browser compatibility (there are still browsers that do not support SVG). Figure 2 shows the general concept of the Transit Layer. A canvas or a vector layer is laid over the actual map layer, pans and zooms along with it and passes through DOM events like mouse clicks. Vehicles on the Transit Layer are drawn as vector objects and are positioned via pixel coordinates that have been computed by the server and that need not be transformed further by the client in any way.

There are still subtle difficulties in redrawing the map. For example, deleting and creating a new marker is much more expensive than re-positioning a marker that already exists on the canvas. On the other hand, searching 4,000 markers to find the marker belonging to a single trajectory is equally expensive. TRAVIC is heavily optimized for time over space and holds a simple JavaScript array containing each marker that is currently visible, indexed by its trajectory ID. Because trajectory IDs are always output as integers by the server, most browsers implement the array as a map optimized for fast key access. Additionally, during the first update of a newly requested set of partial trajectories, the marker of each trajectory is saved as a reference field inside the trajectory object.

4. DEMONSTRATION

The demonstration will highlight the visualization capability of our implemented client and its compatibility with different browsers (see also our experimental evaluation in Section 6). Users are invited to access our live public transit tracker (`http://tracker.geops.ch`) on their own device and check for data coverage in their home area.

The user can zoom in and out, and drag the map to arbitrary locations. Single vehicles can be tracked easily by clicking on the respective marker. Then the marker gets enlarged and the complete route of the vehicle is drawn on the map. Additional information about the vehicle (line number, operation days, agency) are displayed in an info-box along with the sequence of stops to come and respective arrival and departure times (see Figure 1). If delay information is available it is displayed along with the schedule.

5. FURTHER USE CASES

The applicability of TRAVIC goes beyond the scope of public transit. The Transit Layer can, in theory, be used to display vehicles or moving objects of any kind. Outside the domain of public transportation, a vehicle could be a plane, a satellite or even an individual object like a car, a bike or a person. Tracking data is also available for certain animals.[10] One possible application of TRAVIC could be as a client for a server that outputs trajectories of cars travelling on certain roads to visualize the traffic volume at certain times. Figure 3 shows a visualization of the motorized private traffic

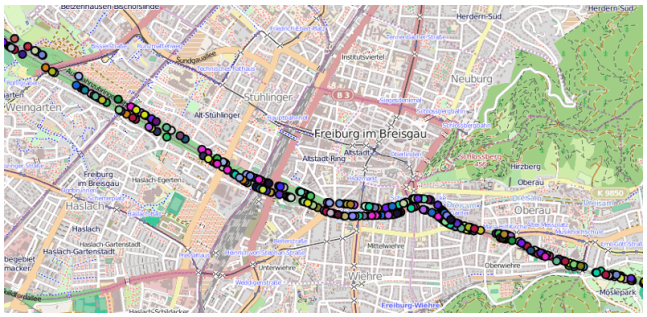

Figure 3: TRAVIC used for traffic simulation.

on Freiburg's main east-west-corridor. The vehicle numbers follow a bimodal distribution with peaks at the morning and evening rush hours. About 20,000 vehicles pass the central bridge in a single direction per day[11].

6. EXPERIMENTAL RESULTS

Measuring JavaScript performance is challenging. Code examples that run efficiently on one browser type can completely lock up another one. To evaluate the performance of TRAVIC, we chose to run tests on the current versions of five different web browsers: Firefox 27.0, Internet Explorer 11.0.2, Chromium 32.0.1700, Safari 5.1.7 and Opera 12.16. We loaded a server installation with a combined feed of 22 single GTFS feeds from Europe, North America, Australia and New Zealand and centred TRAVIC in Amsterdam at the highest zoom level possible and gradually zoomed out. We consider 20 zoom levels (with $z = 20$ zoomed in and $z = 0$ zoomed out), and a hierarchy of transportation modes (subways and buses are only displayed if the user zooms in far enough, trains and ferries are also visible on lower zoom levels). For each zoom level, we measured the number of displayed vehicles $\#v$ and the time it took to do a single screen refresh t_r, using SVG rendering. We did the same tests starting at the dome of New York City Hall, but invoking canvas rendering instead. TRAVIC uses lower refresh rates $(1/f)$ at lower zoom levels. Hence we multiplied this time with the number of refreshes per second at each level. This gives the amount of time t_{tot}/s that TRAVIC was busy with refreshing the screen during a single second. A value $t_{tot}/\text{s} > 1000$ ms would indicate that the intended number of refreshes per second is not feasible.

Tests for Firefox, Google Chrome and Opera were done on a machine with an Intel Core i5-3320M Processor, 8 GB RAM and NVIDIA Quadro NVS 5400M graphics, running Ubuntu 13.10. Tests for Internet Explorer and Safari were done on the same machine, running Microsoft Windows 8

[10] `https://www.movebank.org/`

[11] `http://www.svz-bw.de/fileadmin/verkehrszaehlung/dz/2013/rpt-95-vz-2013-06.pdf`

			Chrome		Firefox		Opera		Safari		IE	
z	$1/f$	$\#v$	t_r	t_{tot}/s	t_r	t_{tot}/s	t_r	t_{tot}/s	t_r	t_{tot}/s	t_r	t_{tot}/s
19	60	0	< 0.1	0.5	< 0.1	1.2	< 0.1	1.3	< 0.1	0.8	< 0.1	0.7
18	85	3	4.7	55.8	7.2	84.8	6.0	70.1	6.6	77.5	7.8	91.9
17	110	33	16.0	139.4	21.5	195.8	17.7	160.6	10.5	95.7	33.2	301.4
16	120	74	13.6	113.2	22.6	188.7	15.9	132.6	11.7	97.7	28.9	240.5
15	140	145	24.0	171.3	35.0	250.1	27.0	193.1	19.3	137.5	50.6	361.6
14	170	317	35.0	205.8	56.4	331.9	35.1	206.5	29.6	174.3	77.4	455.2
13	250	200	31.5	126.1	48.3	193.0	37.8	151.2	25.7	102.7	94.7	378.9
12	400	95	38.0	94.9	57.7	144.3	50.5	126.3	34.1	85.3	216.8	542.1
11	500	130	15.4	30.7	17.4	34.8	15.2	30.5	8.7	17.4	18.6	37.1
10	1 k	221	20.0	20.0	27.2	27.2	19.8	19.8	14.9	14.9	2.2	24.2
9	1 k	363	27.6	27.6	38.3	38.8	26.2	26.2	19.4	19.4	34.6	34.6
8	1 k	423	33.9	33.9	45.5	45.5	31.6	31.6	23.3	23.3	38.9	38.9
7	1 k	455	32.6	32.6	47.2	47.2	32.0	32.0	24.9	24.9	40.8	40.8
6	1 k	1 k	69.2	69.2	109.4	109.4	33.1	33.1	47.2	47.2	88.0	88.0
5	1 k	1.2 k	85.7	85.7	124.4	124.4	40.3	40.3	64.3	64.3	104.2	104.2

Table 1: TRAVIC performance on different browser types and zoom levels, using SVG rendering. Map was centered at Amsterdam

			Chrome		Firefox		Opera		Safari		IE	
z	$1/f$	$\#v$	t_r	t_{tot}/s	t_r	t_{tot}/s	t_r	t_{tot}/s	t_r	t_{tot}/s	t_r	t_{tot}/s
19	60	3	0.03	0.50	0.16	2.67	0.02	0.33	0.02	0.33	0.05	0.83
18	85	12	0.04	0.47	0.76	8.94	0.04	0.47	0.04	0.47	0.08	0.95
17	110	50	0.11	1.00	1.62	14.72	0.02	0.18	0.11	1.00	0.12	1.09
16	120	131	0.11	0.91	2.89	24.08	0.03	0.25	0.15	1.25	0.47	3.91
15	140	320	0.91	6.50	10.73	76.64	0.22	1.57	0.60	4.28	1.12	8.86
14	158	1045	2.91	18.35	14.98	96.14	1.11	6.53	1.55	9.11	4.69	27.5
13	250	412	1.52	6.08	3.18	12.72	0.59	2.36	0.87	3.48	1.71	6.84
12	400	567	1.42	3.55	3.79	9.46	1.36	3.40	1.37	3.42	4.44	11.10
11	500	623	1.79	3.58	4.27	8.54	2.52	5.04	1.89	3.78	3.74	7.48
10-5	1 k	50	0.23	0.23	13.00	13.00	0.15	0.15	0.29	0.29	1.77	1.77

Table 2: TRAVIC performance on different browser types and zoom levels, using canvas rendering. Map was centered at New York.

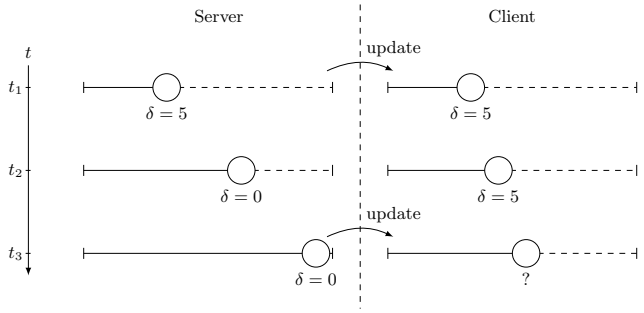

Figure 4: Asynchronous delay.

(SP1). All tests were run in full-screen mode at 1920×1080. Values are averaged from 100 sample runs. Results can be seen in Tables 1 and 2. We observe that for both rendering schemes and zoom levels, the performance of TRAVIC is sufficient to allow for real-time movement visualization, even if there are more than 1,000 vehicles present in the actual view. Using canvas rendering, the values for t_{tot}/s are far from the performance critical value of 1000 for all tested browsers. Our current implementation of TRAVIC, serving more than 80 GTFS feeds around the world, can be accessed via `http://tracker.geops.ch` .

7. FUTURE WORK

Visualizing all near-by vehicles is especially interesting for mobile users. So a natural direction of future work is to implement TRAVIC for mobile clients. The client/server architecture we presented is already suitable for this, but the GUI has to be adapted.

For large update intervals, the problem of asynchronous delay information poses a problem that has yet to be addressed. Consider the scenario depicted in Figure 4: a vehicle travels on a certain route between two waypoints. At time t_1, the client sends a spatio-temporal request to the server, which currently holds a delay $\delta = 5$ for the trajectory. A delay of 5 is communicated to the client. Then the server does an update and fetches the newest version of the real-time feed. Somehow, the vehicle has regained the lost time and δ is now 0 (at $t = t_2$). The client, however, still operates within the bounds of the spatio-temporal request answer received at t_1. Now, at $t = t_3$, the client finally fires a new spatio-temporal requests and learns that the actual position of the vehicle is far behind the position it currently displays. There is, of course, also the possibility of the vehicle being far ahead of the current client position. At the moment, we resolve such a situation by letting the vehicle 'jump' to its correct position. An alternative would be to calculate a new travel speed for the vehicle that allows to sync it with its real position, but using smooth movements.

8. REFERENCES

[1] Michela Bertolotto, Ailish Brophy, Alan Martin, O Gregory, Robin Strahan, Eoin McLoughlin, et al. Bus catcher: A context sensitive prototype system for public transportation users. In *Web Information Systems Engineering Workshops, International Conference on*, page 64. IEEE Computer Society, 2002.

[2] Frédéric Bertrand, Alain Bouju, Christophe Claramunt, Thomas Devogele, and Cyril Ray. Web architecture for monitoring and visualizing mobile objects in maritime contexts. In *Web and Wireless Geographical Information Systems*, pages 94–105. Springer, 2007.

[3] Siyuan Liu, Ce Liu, Qiong Luo, Lionel M. Ni, and Huamin Qu. A visual analytics system for metropolitan transportation. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 477–480, New York, NY, USA, 2011. ACM.