http://excel.fit.vutbr.cz

Visualization of lidar, camera, and vector data from a railway mobile mapping system

Zuzana Miškaňová*

Abstract
The aim of this paper is to present a web application focused on visualizing data from a mobile mapping system. The application aims to visualize data in a practical, clear and customizable way. The application is written in Python using framework Dash. For a smoothly running animation of the movement of the train even with a large point cloud, the rendering of the final image is done on the client's side. For the visualization itself, framework deck.gl is used. At present, the application offers sufficient functionality and performance. However, there is still room for further improvement, mainly in customizability. This new web application has a potential to be exploited within the field of mobile mapping system data visualizations. The knowledge acquired throughout the work could also useful to other developers creating visualization systems.

*xmiska03@stud.fit.vut.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

When working with large datasets, transparent and handy visualization systems play a key role in the process. The application described in this paper tries to satisfy the demand for a specialized visualization system targeted on data from a railway mobile mapping system.

The main challenge of the work was to find an effective way of rendering animations of large point clouds in a web application, because point clouds, along with frames from the camera, constitute the largest part of the visualized data. The created application, on top of good functionality, also needs to be intuitive and user-friendly to be useful.

Examples of existing solutions for mobile mapping systems data visualizations include program CloudCompare and the Autonomous Visualization System (AVS). However, CloudCompare cannot visualize the point cloud from the point of view of the train operator and cannot display camera data simultaneously and AVS is a framework for developing advanced systems, not a ready-made application [1].

The application developed in this work satisfies specific demands, and therefore is unique. It allows the user to upload various data files. The data is then displayed from the point of view of the train operator and the animation of the movement of the train can be played.

2. The visualized data

The application can display multiple kinds of data:

- point cloud data, divided or united – Figure 2,
- video from the camera – Figure 3,
- predicted positions of the train profile – Figure 4,
- other vector data.

In order to display the point cloud and vector data from the perspective of the train driver, the data must include camera positions, with each position composed of a translation, a rotation and a timestamp. For the point cloud to fit the video, all of the camera parameters need to be taken into cosideration. This includes the camera matrix and distortion coefficients – Figure 5.

All of this data is put together and used to create the final layered picture – Figure 1.

3. Camera movement

In the dataset provided for this work, the point cloud did not fit the video well when it was displayed using only the provided camera positions and parameters. It was necessary to add a certain offset to each camera position – one more translation T_2 to the provided translation T_1 and rotation R. This situation is illustrated in Figure 6, reduced to 2D space for simplification.

Translation T_2 is relative to rotation R. For example, on the illustration it is always "to the right".

This creates a small mathematical problem, because in deck.gl, the camera position is set by using one translation and one rotation, T_V and R_V. Therefore, we need to find T_V and R_V such that

$$T_1 R T_2 = T_V R_V.$$

The solution is quite simple, it suffices to compute $T_2 R T_1$ and then "divide" the matrix into T_V and R_V.

4. Deck.gl layers

In deck.gl, the visualization is composed of layers. There are many kinds of layers, two of which are useful for this work - `PointCloudLayer` and `PathLayer` [2]. The application typically displays 10 point cloud layers and 3 path layers, which is shown in Figure 7.

The displayed point cloud data consists of small chunks of data, which were scanned step by step as the train moved forward. The application displays one current and nine previous chunks.

That means that the layers data needs to be changed whenever the position of the train changes. When the animation is running, at most one layer's data needs to be changed in one step – the oldest chunk needs to be replaced by a new one. This is done in an effective circular way, which is shown in Figure 8.

The application also includes a possibility to display only one (unchanging) point cloud layer.

5. Implementation

For the user's convenience, the described system has been implemented as a web application. The whole structure of the implementation is illustrated in Figure 9. The application is written in Python using framework Dash and library Dash Bootstrap Components for styling.

5.1 Visualization rendering

To avoid delays caused by communication between the client and the server, the rendering of the visualization itself takes place on the client's side, using framework deck.gl. This framework uses WebGL for acceleration and can visualize large datasets effectively [2]. Although it would be possible to render and control the visualization in this way using only Python code running on the server, the result would not be desirable because of a very poor performance.

So instead, the visualization is controlled directly by JavaScript functions put in a module which is attached to the application. *Clientside callbacks* defined in Dash then call the functions in the module. This optimization allows the application to really use the full potential of deck.gl.

5.2 Animation of train movement

The animation begins by playing the video, which is inserted as an ordinary HTML `video` element. The synchronization between the video and the deck.gl visualization is provided by registering a JavaScript callback, which runs when a new video frame is sent to the compositor. The callback updates the visualization according to timestamps data.

5.3 The project file

As it was described above, the data that needs to be visualized is composed of many files, sometimes hundreds, in various formats, which means that it would be inconvenient for the users to upload them manually. This problem was solved by accepting a project file which specifies paths to all data files, under the assumption that the user has access to the server that the application server is running on. This assumption can be made, because the application is intended for specific users, not for the general public.

6. Conclusions

The aim of this work was to create a practical visualization system. The application was made and is, with the exception of the distortion calculation, quite well-performing, rendering an animation which includes a million and a half points at 46-52 FPS.

However, there are many features that could be added to make the visualization even more customizable and the application even more user-friendly, for example point cloud filtering and more interactivity.

Acknowledgements

I would like to thank my supervisor Ing. Ondřej Klíma, Ph.D. for his help and useful advice.

References

[1] Uber ATG. Autonomous visualization system. online. https://avs.auto/.

[2] OpenJS Foundation. Deck.gl. online, 2025. https://deck.gl/.