BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2025
http://excel.fit.vutbr.cz

Tool for rendering maps with traveled routes

Lukáš Kotoun

Abstract
The thesis deals with the creation of a tool that allows the user to render a high-quality and visually appealing map with custom routes in a simple way. The rendering tool is implemented using Python. The configuration interface of the tool is created using the SvelteKit framework. The basemap is rendered in vector form using data from the OpenStreetMap project. The resulting tool provides the ability to set the map area, the size of the resulting page for possible printing, and detailed customization of the appearance of the map base along with the appearance of the traveled routes. The data for these routes is extracted from uploaded GPX files. The resulting tool is easy to run thanks to containerization via Docker.

*xkotou08@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Most hikers, cyclists, runners and other athletes use some sort of smart device to record their sporting performance. They then often share these performances with others and also view them themselves, especially if they are challenging or record-breaking. One of the most important indicators of such a performance is its route. If an athlete wanted to exhibit their challenging routes, they would most likely have to get a detailed map of the area where they completed these routes and then mark them manually on the map. However, it may not be entirely easy to find a map that suits the level of detail and size, and then accurately draw the routes in question.

The aim of this work is to create a tool that would allow the user to create a customized map for easy printing by using simple but detailed settings. The tool allows the selection of a map area, or a combination of several areas, setting the paper size with the possibility of calculating it according to the selected area, uploading custom routes in GPX format and selecting map elements. It also offers detailed options for customizing the appearance of both these elements and routes. It also provides the ability to prototype the map appearance by rendering a smaller area on smaller paper with a scale corresponding to the main area, for faster and easier customization of the appearance.

The resulting solution is implemented as a web client and server. The client, created using the SvelteKit framework and Tailwind CSS library, serves as a configurator. The server, created using the Python FastAPI framework, then performs the actual map generation.

2. Area and paper selection

When creating a custom map, the first thing user is asked to do is select the area they would like to display on the map. There are two input options to choose from. The first option is by using the name of the area, for example for a more complex area such as the borders of the Czech Republic. The second option is more for entering a simpler area, such as a square area, directly using coordinates. For areas, it is also possible to set the width of the borders, joining the areas, which will result in removing the borders where the areas touch. It is also possible to set the area to fit the paper, which will fill in missing parts so that the area perfectly copies the paper.

After the setting area there is a paper setting where the user can select from standardized sizes or choose a custom size. If only one dimension of custom size is filled in, the second dimension will be automatically calculated based on the dimensions of the selected area and the specified limiting paper size. This approach is particularly useful when using industrial printers that print on a virtually infinite roll of paper of a given width.

The configured area can then be easily viewed on the created preview. Figure 2 shows an example of this preview for 2 connected areas.

3. Traveled routes upload and styling

After setting the area and paper, the user has the option to upload their own recorded routes in GPX format that they would like to display on the map. The user can then categorize the uploaded routes into their own groups and set the appearance of these groups.

The appearance settings are very detailed. Routes can be set to color, size, transparency, style, or position relative to the text for the basic appearance and line borders. The settings also allow detailed customization to mark the beginning and end of routes, including the option to turn off marking. A sample of the default appearance of GPX routes with the added start and end marking can be seen in the Figure 3 image.

4. Map elements and design

After setting the previous elements, the map basemap appearance is specified. The colors and detail appearance are fixed based on the selected style. The user can select 1 of 10 levels of detail to adjust this fixed appearance or leave an automatic value calculated based on the ratios of the selected paper and map area.

The elements that will be displayed on the map can be freely changed by the user, based on one of the zoom levels and automatic determination of elements for this level, or completely customized. For all elements where this setting makes sense, the user can then change the width and font of the element description texts. Appearance and elements are set separately for points, paths and areas. Figure 4 shows a sample map with the Mapycz style, without texts and with a GPX route.

5. Rendering

In the last step, the user simply chooses whether to create the entire map or a quick preview on a smaller paper, which uses ratio calculations to determine a smaller map area to be rendered on the smaller paper for easy and quick customization. This smaller area will then be scaled to match the larger area, so when you overlay the preview area on the corresponding portion of the final area, the elements will overlap perfectly. Any settings made can also be saved for further editing at a later date.

The data from the OpenStreetMap project [1] is used to create the map base, which is pre-filtered to ensure that it does not contain any non-valid elements. All the above mentioned settings are done in the user interface created using the SvelteKit framework and Tailwind CSS library. These settings are sent to the Python server, where a parallel process is created to process the request and perform the rendering.

This Python parallel process uses the osmium command line tool to select the specified region from the data file or merge multiple files and create a new osmium file with the required data. This file is then loaded by pyosmium into a GeoDataFrame construct from the GeoPandas library designed for working with spatial data. The loaded data is first preprocessed by, for example, filtering the elevation points using a position and elevation-based algorithm to determine their significance (based on Topographic Prominence algorithm [2]), or a path-joining algorithm to avoid rendering aretafacts. The preprocessed data are further assigned styles and sorted for correct layering before the actual plotting. Rendering based on the assigned styles is then handled by the Matplotlib library, which is passed areas, paths, routes and as last points in turn.

The resulting map is saved in PDF format and ready to be sent back to the user. A sample map showing Brno and its surroundings can be seen in the Figure 5.

References

[1] Jonathan Bennett. *OpenStreetMap: Be your own cartographer*. Packt Pub, 2010.

[2] Andrew Kirmse. Topographic prominence, Jan 2017. https://www.andrewkirmse.com/prominence.