Visualization of Geodata

Spatio-temporal Visualization using d3.js

Study Course: Geodesy and Geoinformation M.Sc.

Supervisor: Dr. Mathias Jahnke

Date: 10.01.2019

Technical University of Munich

Chair of Cartography

1 General Information

The goal of this short and condensed tutorial is to provide a basic overview of how d3.js can be used for Geovisualization. Additionally, to the map data spatio temporal information like tweets should be displayed. To decide which tweets to display should be done using arrange slider. In the case of this tutorial the NoUiSlider[1] is used. It is a light weight slider library with no dependencies.

To achieve this goal a basic understanding of HTML, CSS and JavaScript is essential. Within this tutorial the text editor Visual Studio Code[2] (VSC or Code, as the time of writing version 1.30.1) is used. The editor (Figure 1-1) is as well installed on the lab computers in room 0712. VSC support modular expendability by different downloadable packages. An important package in the case of web development is the “Live Server” packages by Ritwick Dey which supports live reload capabilities for web sites and enables a local web server to load locally stored data into the web page. Nevertheless, any other editor like atom, brackets, sublime or notepad++ are as well sufficient for working through this tutorial.

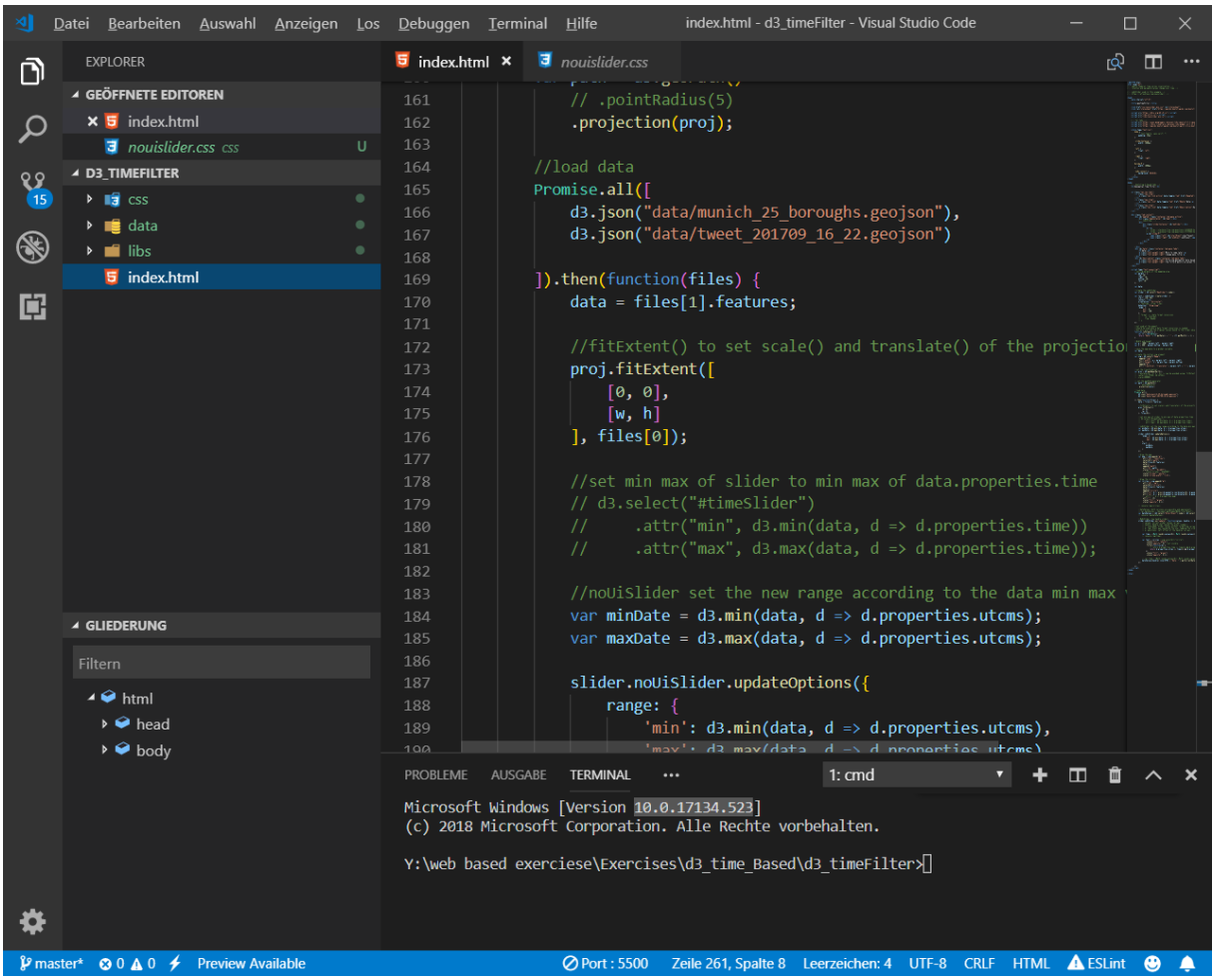

Figure 1-1: Visual Studio Code in action

Besides the editor a web browser who provides web developer tools is essential like chrome, Firefox or any other. The web developer tools in chrome or Firefox can be invoked by typing **ctrl+shift+i**.

[1] https://refreshless.com/nouislider/
[2] https://code.visualstudio.com/

In the course of this tutorial the visualization library d3[3] will be used which should be referenced via a script tag in the html document. In the below example the d3 library is retrieved from a content delivery network (CDN). The slider library must be as well referenced via a script tag.

```
<script src="https://d3js.org/d3.v5.min.js"></script>
```

D3 was developed for fast and easy data manipulation and provides as well sufficient and easy to use functionalities to develop SVG[4] visualizations. SVG is supported by all major web browsers. All features are drawn using SVG primitives like rectangles, circles or arcs. More complex figures like irregular shaped features are created using the path element.

2 What to implement

In this tutorial we are going to implement a background map in the case of this tutorial the Munich region. On top of the background map the tweets should be visualized using dots. With the range slider below the map area it should be possible to filter the tweets and to display only those who are within the time frame defined by the slider.

As mentioned before all visualization are drawn using the library d3.js which supports many different functions and methods to programmatically create and change the visualizations and the underlying SVG code.

The Geovisualization is based on a basic html structure including the head and body element as well as the reference to the used libraries.

The data files are provided via the moodle platform. The twitter data is covering one week during Oktoberfest 2017 as geojson while the base map data are Munich boroughs from OpenStreetMap.

The following source code must be seen as a whole through the different paragraphs. The paragraphs are indicating new or important information.

2.1 The Beginning

The index.html file in other words the file which host all source code needed for this tutorial starts with the typical html declaration and the head section. The head section hosts references to external libraries in this case to the d3 and the noUiSlider library. CSS styling rules are as well within the head section. In this example the font-family is set for the whole body tag and styling information for different classes are set as well.

Selectors starting with a **dot (.)** are referring to classes while selectors starting with the **hash (#)** are referring to element id's.

Beside the index.html file three different folders should exist.

The first folder named **css** holding css files of e.g. the external libraries.

The second folder should be a **data** folder, where all the data goes in in this case the data for the background map and the tweets. The background map and the tweets should be in GeoJson[5] format to support easy loading by the libraries.

The third folder is the **libs** folder which holds the external library files like the noUiSlider.js file and the utils.js file as well provided via the moddle platform.

```
<!DOCTYPE html>
<html lang="en">
```

[3] https://d3js.org/
[4] https://developer.mozilla.org/en-US/docs/Web/SVG
[5] http://geojson.org/

```
<head>
    <meta charset="utf-8">
    <title>geoTimeFilter</title>
```

The noUiSlider library must be downloaded from GitHub (the link can be found on the noUiSlider webpage). The library and the CSS file should be put in the different folder and the CSS file must be linked via the link element while the .js library must be added via the script tag. Have in mind the d3 library is served via a CDN while the slider library is stored local.

```
    <link href="css/nouislider.min.css" rel="stylesheet">

    <script src="https://d3js.org/d3.v5.js"></script>
    <script src="libs/utils.js"></script>
    <script src="libs/nouislider.min.js"></script>
```

The CSS style information sets the font-family for the body as well as the padding. .sliderContainer sets the width according to the SVG element width. .left, .right and #sliderP are setting how the date and time information should be displayed beneath the slider element.

```
    <style type="text/css">
        body {
            font-family: sans-serif;
            padding: 25px;
        }
        .sliderContainer {
            width: 1000px;
        }
        .left {
            float: left;
        }
        .right {
            float: right;
        }
        #sliderP {
            width: 1000px;
        }
        noUi-connect {
            background: #D1D1D1;
        }
    </style>
</head>

<body>
    <h1>Geospatial Time Filter</h1>
```

```
<p>Filtering geospatial data based on a time attribute in "utc ms".</p>
<p>The filterd points are green the rest is not visible.</p>
```

Div container for the map data and the slider are created. Both container have id's and class to set styling information and to select them using JavaScript functions.

```
<div class="mapContainer" id="map"></div>
<!-- noUiSlider -->
<div>
    <div class="sliderContainer" id="mySlider"></div>
    <div>
        <p></p>
        <p id="sliderP">
            <span class="left" id="startEvent">start*event</span>
            <span class="right" id="endEvent">end*event</span>
        </p>
    </div>
</div>
```

2.2 Let's start Programming

Starting with setting up some margin around the SVG canvas.

```
<script type="text/javascript">
    //setting margins of the mapping area
    var margin = {
        top: 20,
        right: 10,
        bottom: 20,
        left: 10
    };
```

2.3 Setting up the slider

The noUiSlider library is used therefore the slider node must be retrieved from the DOM and the slider is created handing over the slider node and different options: some initial start values for the slider handle, the array indicates that two handles should be created. Both handles should be connected, it's more a visualization issue. The behavior is set to drag-snap to make the area between both handles dragable and the range of the slider is initially set but have to be overridden later with the current values after loading the data.

```
    //create the noUiSlider
    var slider = d3.select("#mySlider").node();

    var test = noUiSlider.create(slider, {
        start: [20, 80],
        connect: true,
```

```
            orientation: "horizontal",
            // tooltips: [true, true],
            behaviour: "drag-snap",
            range: {
                'min': 0,
                'max': 100
            },
```

It is possible to set format options to adapt values. Here the timestamp from utc ms is converted into a human readable format. But it is not used in this tutorial.

```
            // format: { //date format conversion
            //     to: toFormat,
            //     from: Number
            // }
        });
```

The following code snippet is not used at the moment. Should be used when a date format conversion is needed but will override the original values bound to the slider object

```
        function toFormat(v) {
            var tt = new Date(v);
            return "Date:" + tt.getDate() + "." + (tt.getMonth() + 1) + "."
+ tt.getFullYear() + " Time:" + tt.getHours() + ":" + tt.getMinutes() + ":"
+ tt.getSeconds();
        }
```

2.4 Start creating the map

Width and height of the SVG canvas are set, and a global variable data to store the loaded data global is declared.

```
        //Width and height
        var w = 1000 - margin.left - margin.right;
        var h = 600 - margin.bottom - margin.top;

        //save the map data to a global variable
        var data;
```

creating the SVG canvas and adding a SVG group element in which should enclose all later added SVG elements.

```
        //create the initial svg element
        var svg = d3.select("#map")
            .append("svg")
            .attr("width", w + margin.left + margin.right)
            .attr("height", h + margin.top + margin.bottom)
            .append("g")
```

```
            .attr("transform", "translate(" + margin.left + "," + margin.top
+ ")");
```

Setting the projection

The projection for the spatial data is defined in this case the equal earth projection is used. Translate(), center() and scale() could be used to define initial values for the projection. These values should be adopted to fit the displayed data into the canvas. To avoid experimenting to find the correct values for those methods the function fitExtent() could be called after loading the data but before using the path generator

```
var proj = d3.geoEqualEarth();
// .translate([w / 2, h / 2]) <= can be avoided using .fitExtent()
// .center([11.57549, 48.13743])
// .scale(100000);
```

2.5 Defining the geoPath generator

A path generator must be defined to automatically generated the SVG path elements. In this case a geoPath() generator must be defined which needs the projection to correctly transform the spatial coordinates to screen coordinates.

```
var path = d3.geoPath()
    .projection(proj);
```

2.6 It is promised all

To handle the asynchronity of JavaScript the Promise.all() is used to load the data from disk or web space. Promise.all loads the data in the order of calling the data load function (d3.json()) and returns an array containing the data. If the data is successfully loaded the callback function is executed.

```
Promise.all([
    d3.json("data/munich_25_boroughs.geojson"),
    d3.json("data/tweet_201709_16_22.geojson"),
]).then(function(files) {
    data = files[1].features;
```

fitExtent() to set scale() and translate() of the projection automatically. See remarks above.

```
    proj.fitExtent([
        [0, 0],
        [w, h]
    ], files[0]);
```

Adjusting the parameters of the noUiSlider to the new range according to the data min and max values. Be aware of adopting the parameter name to the parameter name of your data in which the utc timestamp is saved.

```
    var minDate = d3.min(data, d => d.properties.utcms);
    var maxDate = d3.max(data, d => d.properties.utcms);
```

```
slider.noUiSlider.updateOptions({
    range: {
        'min': d3.min(data, d => d.properties.utcms),
        'max': d3.max(data, d => d.properties.utcms)
    },
    start: [
        minDate,
        maxDate
    ]
});
```

2.7 Drawing the geospatial data

The geospatial data should be drawn to SVG element or to the group element within the SVG element. The background map and the tweets are placed in different group elements to better distinguish and structure. The background map is in files[0] while the tweets are in files[1]. The Objects are geoJson feature collection therefore the array within the feature collection holding all the features have to be referenced.

To draw the background map, the path generator is used and additionally some style attributes are set.

```
var map = svg.append("g")
    .attr("id", "countries")
    .selectAll("path")
    .data(files[0].features)
    .enter()
    .append("path")
    .attr("d", path)
    .style("fill", "#ccd9ff")
    .style("stroke", "white")
    .style("stroke-width", "1.0");
```

The circles are drawn by setting the circle center using the specified projection and as well some additional style information is set.

```
var circles = svg.append("g")
    .attr("id", "eq")
    .selectAll("circle")
    .data(files[1].features)
    .enter()
    .append("circle")
    .attr("cx", d => proj([d.geometry.coordinates[0], d.geometry.coordinates[1]])[0])
    .attr("cy", d => proj([d.geometry.coordinates[0], d.geometry.coordinates[1]])[1])
    .attr("r", 7)
    .style("fill", "green")
```

```
                .style("opacity", 0.5);
```

2.8 Time based filtering

Beneath the slider on the left and right side the date and time values selected by the slider handle should be visualized and updated automatically. This is done using p- and span-elements. These two nodes where to write the information in needs to be queried. The first element (index 0) of the array is the “from” date and time while the second element (index 1) is the “to” date and time.

The both following versions are possible and equivalent.

```
            // var dateValues = [document.getElementById("startEvent"), doc-
ument.getElementById("endEvent")];

            var dateValues = [d3.select("#startEvent").node(), d3.se-
lect("#endEvent").node()];

            // console.log(dateValues);
```

2.8.1 The event handler to react on slider input

The event handler if the slider is updated. The slider updates continuously therefore the visualized tweets and the “from and to” span-elements should update as well continuously. If the slider should only update values when moving the slider handles has ended “change” must be used instead of “update” within the .on() function call.

The function that should be called on an update event is providing the following five arguments:

- values: the slider current values always as an array but in this case as a two element array (for both slider knobs).
- handle: the number of the handle which caused the event (the left or right handle) always as a number.
- c: unencoded: slider values without formatting always an array
- d: tap: event was caused by the user tapping the slider always Boolean
- e: positions: the left offset of the slider handles always as an array

```
            //NoUiSlider (two handle slider)

            slider.noUiSlider.on("update", function(values, handle, c, d, e)
{
```

converting the “values” from string to number, rounding the date and time values of the slider handles to the nearest integer and save everything as an array.

```
                var time = [Math.round(+values[0]), Math.round(+values[1])];
```

2.8.2 Setting drawing parameters according to the filter

Within the event handler on any update event all circles are set to white without any opacity. These values are filtered based on the input from the slider. The filter function creates a new selection and to this selection new style parameters for fill and opacity are applied.

```
                var test = circles //svg.selectAll("circle")
                    .style("fill", "white")
                    .style("opacity", 0) //not visible
                    .filter(function(d) {
                        return d.properties.utcms <= time[1] && d.proper-
ties.utcms >= time[0];
                    })
                    .style("fill", "green")
```

```
                    .style("opacity", 0.5);
```

Updating the information for the user to which date and time the slider handles are set/moved. The getCurrentDate() and getCurrentTime() function are from the utils.js file.

```
                dateValues[handle].innerHTML = "Date: " + getCurrentDate(new
Date(time[handle])) + " Time: " + getCurrentTime(new Date(time[handle]));
            });
        });
    </script>
</body>
</html>
```

2.9 The End

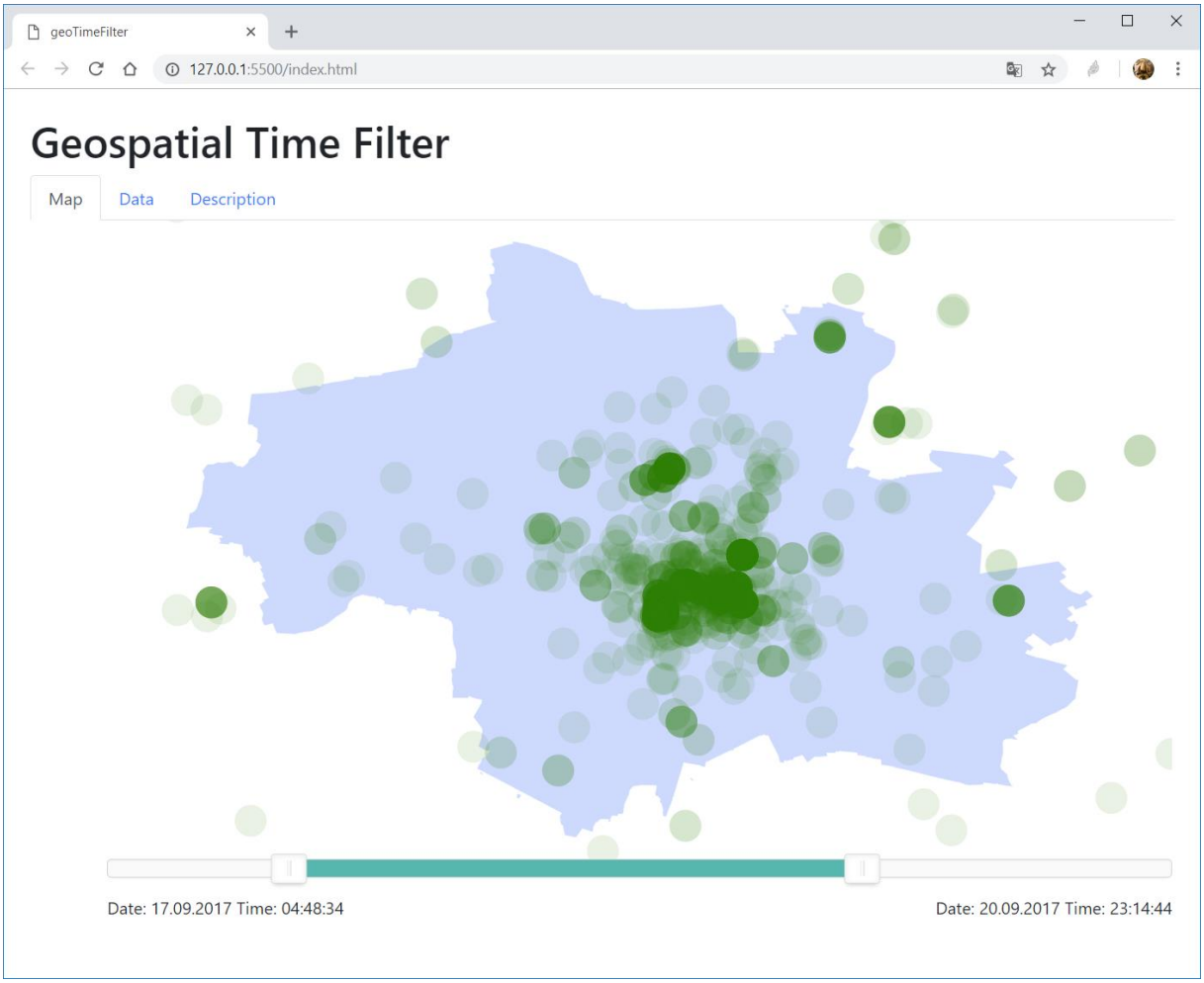

Figure 2-1: The finished web based Geovisualization

The visualization can be enhanced in many ways, graphs, charts or histograms can be included to get a deeper insight into the data respectively into the filtered data. A big issue which is not covered and implemented yet is how to interact with the map. Zoom and pan is missing as well as click or hover events to get deeper insights for every single tweet.

This visualization should be seen as a starting point how to build a web based representation of spatial data using open source and free tools.