Visualization

Winter Semester 2021/2022

Prof. Tobias Günther

Friedrich-Alexander-Universität
Technische Fakultät

Programming 1 – The Basics

Am I able to solve the programming exercises without prior experience in programming?

No, most likely not. Note that the programming exercises are not needed to reach a top score in the exam.

Introduction

This is the first of the four programming exercises for this semester. On the second and the fourth exercise you will be able to score four points plus one bonus point from an optional task. The first and the third exercise will both contain tasks worth five points. In total, you will be able to score 18 (+2 bonus points). You need 10/18 points to get the bonus for the exam. Each exercise will be graded by the TAs **during** the exercise session **before/on** the due date (check the schedule for due dates). To submit your exercise you have to upload your `html` file to studon and present your visualization to one of the TAs during the exercise sessions. The TAs will also ask a couple of questions to ensure that you did the exercise on your own and fully understand how it works. You may work together as a group of up to three students but make sure that each of you understands all the elements that you've added to the visualization since the TAs will be asking questions individually.

For all the four exercises you will be working on the same visualization framework, and you will need to continuously add more elements until an interactive visualization is built up which allows the user to explore the missing migrants data set. Please note that most of the tasks (except the last exercise) are independent of each other. This allows you to skip certain tasks completely but you might end up with an incomplete visualization and fewer points in total.

It is not necessary to install any type of software on your computer except a modern web browser such as Firefox or Chrome. For editing of your html file, you can use the developer tools of the browser or use your favourite text editor (for example: Visual Studio Code) . Using the developer tools allows you to see changes immediately. It is also possible to work on the computers in the CIP pool where the exercise is held. For any questions on the programming exercise please come to one of the exercise sessions:

- Thursdays from 09:00 - 12:00 at CIP4 (00.156-113), Martensstraße 3 or
- Thursdays from 13:00 - 13:45 on Zoom

Task 1: Preparation (0 Pts)

From our StudOn page download the skeleton for the exercise. The whole exercise will be done in this single `html` file. Here you will find a lot of `TODOs` which you need to complete throughout this semester. The `TODOs` are comments in the source code sometimes starting with // or wrapped in $<!---->$. They always start with a reference to the task they belong to, for example: `TODO 4.1: Instructions go here`. When comments are indented, it usually means that something is added to or nested within something that was added before. In some cases you will also find larger comments that will mark certain section of the source code to guide you in your search for `TODOs`. We recommend to

remove the `TODOs` that you have completed for each task to keep track of the things that you still have to do.

While working, you will usually use a browser to see the current state of your visualization and a text editor (such as Visual Studio Code) which you will use to edit your visualization. Usually, an `html` file on Windows will be opened in your standard browser by default. To edit your `html` file, you can use the browser's developer tools or a text editor of your choice. It is good practice to continuously refresh the browser page after making changes, which allows you to spot errors quickly. In the case that nothing is shown, you could have a look at the browser console.

Task 2: Getting familiar with HTML and React (1.0 Pts)

In this first part you have to insert a title and a description into your visualization. The title is static and can be added as a simple `html` element. You will find a `TODO` in the source code showing where to add the title. You can use *Missing Migrants across the Globe* as the tiel or come up with your own title.

We also want to add a description to the visualization. The description should be dynamic, which means that we want to be able to modify the text from JavaScript. For this we have to create something that is called a component. If you check the end of your `html` file you will already find a number of components. The app component wires everything together and contains the final visualization. *WorldGraticule* and *Histogram* where added by us as placeholders for the components of the final visualization. Both of them will be extended later to show the different components of the visualization. The last component we added is the *Introduction* which we will now extend to show some custom text.

The new component will return two *div* elements. One will contain a subtitle for the description and the other the description text. Both of them will be placed inside a react fragment which is necessary whenever we return more than one `html` tag from a component. Both of them require a `className` attribute to be able to style the text. You will find the values of these attributes and more detailed instruction in the `TODO`. Next, we have to add the variable which will contain our introduction text. This variable will be inserted into our second div element but we have to wrap the variable name into curly braces to make sure React and JSX understand that the content of the variable should be inserted here. Again you can find additional information in the `TODOs`. To be able to see the text, you have to add the component to the `App` component in the same way as `Histogram` an `WorldGraticule` components were added. You should now be able to see the title, the subtitle and the description in your browser.

This task is now complete. Feel free to style the title and the description the way you like it by checking out styling information at the top of the `html` file inside the `style` html tag. You can try to color the title, change the font, etc. The code inside the style tag is called CSS and is quite powerful. If you don't know how to do certain things you can quickly search it on the internet. There are great resources out there which describe the capabilities of CSS.

Task 3: Data Loading (1.5 Pts)

In this task we need to load the data that we will work with throughout the upcoming exercises. We require two data sets. The first is a JSON data set which describes the topology of the world and the borders of countries. The second contains the actual data we are visualizing and is stored in CSV format. Both data sets are loaded in a separate function in your source code. The `useWorldAtlas()` function, that loads the topology data, is already given to you right above the code for loading the CSV data. Have a look at it and try to understand whats going on. The way we will load the CSV data is quite similar.

To load the CSV data, we already defined the `useData()` function for you, which returns the data. For efficient data loading, we have to use `React.useState` and `React.useEffect`. The first allows us to define a state which can be modified later with a set function. This way, the data can be downloaded in the background and set at a later point in time. The second, allows us to ensure that the data downloading is only performed once. In this case we need to use `d3.csv` because we are working with CSV data. When the download is finished, the function defined inside the `.then` function call will be invoked which we will use to call the set function of our previously defined state. Check the `TODOs` to complete the `useData()` function!

To make sure we can work with the data, we need to first apply some transformations on it. For example we have to transform the number of dead and missing migrants from a string to a number and the date string to a Date object. Furthermore, we have to split up the coordinates string such that we have two individual numbers to work with. You will find detailed instructions in the source code. The last thing we have to do is to add a special case in the `App` component whenever the data has not been loaded yet.

Task 4: Data Metrics (0.5 Pts)

As a small first step, we will write some data metrics to our introduction. You can add anything you like to the introduction such as the number of rows (`data.length`) or the number of columns (`data.columns.length`). To be able to access the data from within the `Introduction` component you have to pass it to the component. For reference you can check how width and height are passed to the `Histogram` and `WorldGraticule` component.

Task 5: World Sphere and Graticule (2.0 Pts)

`d3.geo` contains a number of useful projection methods that can transform longitude/latitude coordinates to pixel coordinates. Given a projection method, we will first define a path generator that takes a d3 object as input and transforms it to pixel coordinates. We will use this path generator for everything to be drawn on the map. As a starting point, we begin drawing a sphere and a few grid lines, called graticules. In the script, find the `TODO` that says "`TODO: define a projection`". Use `d3.geoNaturalEarth1()` to create a projection and assign the result of this function to a `const` variable called `projection`. Feel free to later try other projection methods, such as `d3.geoEquirectangular()` or `d3.geoMercator()`. We will be able to change the entire visualization just by changing this line of code!

Afterwards, feed the `projection` object into `d3.geoPath( )`, and assign the result to a `const` variable called `path`. This path object may later receive d3 geometry and then transforms it using the given projection to image space. As a first visual element on our globe, we want to visualize a lon/lat grid. Thus, define a variable `const graticule` and assign the result of `d3.geoGraticule()` to it. Note that geoGraticule does not need the projection or the path element yet. This transformation will come later down below.

Next, we want to display the first elements on the map. Find the React component `WorldGraticule` and look into the group element `<g>` that it returns. Inside this group element, you find a number of `TODO`s. This is where we will create all the visual elements to be displayed. First, we will draw a simple sphere, using

```
<path className="sphere" d=path( type:  'Sphere' ) />
```

This will create a sphere. We draw the sphere using an SVG `<path>` element. The path element will be named "sphere", for stylization in the CSS `<style>` section. The data d of the path receives the projection of a default sphere.

Next, we will draw the graticules on top. In that same group element `<g>` insert a `<path>` as child element after the sphere, namely:

```
<path className="graticule" d=path(graticule()) />.
```

With this, we are doing two things. First, we give the path object a class name, which will be used in the CSS `<style>` section above to define the color of the lines. Second, we define the path geometry, for which we pass the graticule into the path generator that we defined above. When done correctly, you should see the grid lines on the sphere.

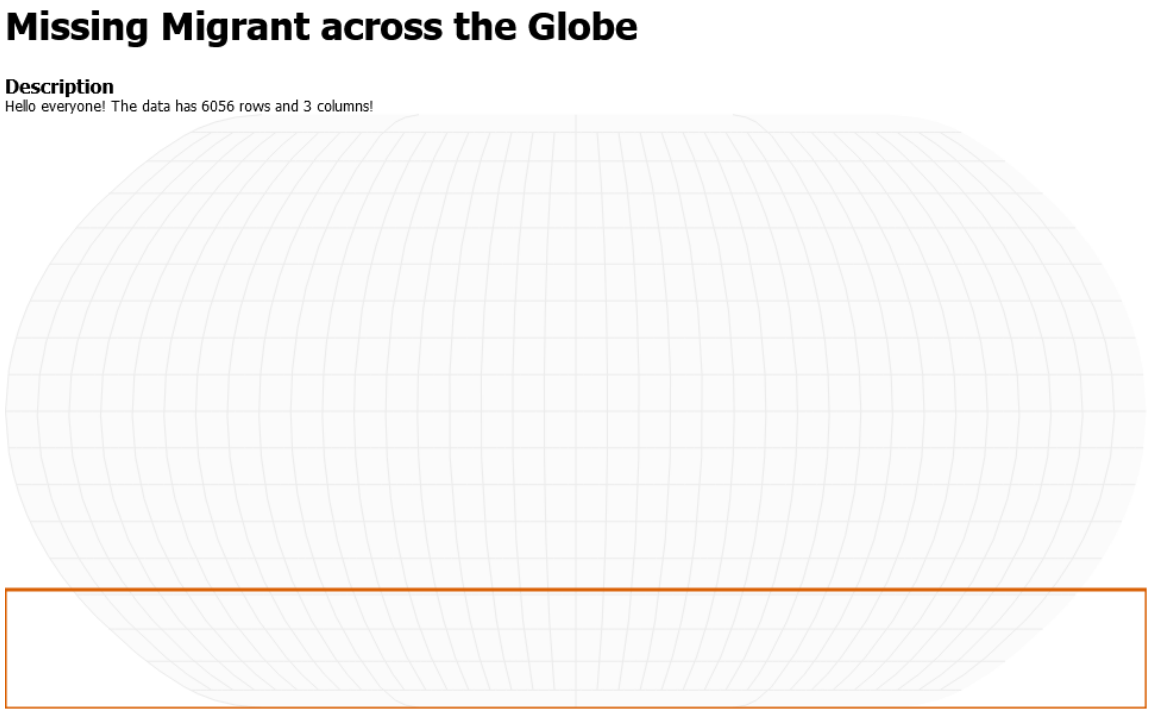

Fig. 1: Solution to this exercise: We draw the Earth as sphere and place graticule lines on it. If you want you can remove the orange box by removing the Histogram component from the App.