Linköpings universitet

Technical Documentation

Visualization of Climate Data

Marcus Ekström
marek898@student.liu.se

Kerstin Söderqvist
kerso255@student.liu.se

Carl Dehlin
carde650@student.liu.se

Erik Dahlström
erida516@student.liu.se

Supervisor:
Ingemar Ragnemalm

December 15, 2017

Contents

1 Introduction

This project is aimed towards climate research, performed as a part of the CDIO-course *TSBB11* at Linköping University. This report aims to describe the system developed during the course of this project, with its' functionality and performance. It is written for the customer - *SMHI* (Swedish Meteorological Institute), the examiner of the course and also other people with previous knowledge of and interest in visualization and climate research.

1.1 Problem Description

Climate data is large in size and complex to the point that even climate researchers have to spend significant amounts of time to interpret it. The data is received as output from simulations using various different climate models, scenarios and resolutions. Visualization has been and will be an important tool for climate research in developing a deeper understanding for the measured and simulated data. One purpose of climate research in general is to learn about how the climate is changing and predict how it will change in the years to come, but also to help the public gain an understanding for this complex problem. Visualization tools can communicate to the more average user on a whole different level than just the data itself.

1.2 Project Objectives

With the background described in 1.1 in mind, this project was designed with focus on developing an application to visualize climate data that has the following features

- Interactive
- Pedagogic and accessible to interest groups apart from climate researchers
- As close to real-time as possible

1.3 Data

The data that was used in this project was gathered from the WCRP - *World Climate Research Programme* and their database containing all the data from the project CORDEX - *Coordinated Regional Climate Downscaling Experiment*. This data uses the commonly used data format *netcdf* (Network Common Data Format) and contains a grid of data points distributed over Europe given in longitude and latitude coordinates and sampled over time. The primary parameters that was investigated in this project was daily precipitation, maximum temperature and how common the extreme cases of these are. The system can work with other parameters as well, but the visual appearance is optimized for these parameters in terms of the chosen color mappings.

The data used has a spatial resolution of 0.11 degrees and a temporal resolution of days or months. Data on day resolution is only used for temporary calculations and is then

aggregated to months and years for visualization. All the data used is simulated for the climate scenario RCP 8.5, simulating comparatively high greenhouse gas emissions [1].

In the project a reference period is used to give context to the data visualized. Whenever reference period is mentioned in this report it refers to the data for that parameter over the period 1981 - 2010. All other data is over the period 2011 - 2100.

1.4 Limitations to the Project

The project is focused on visualizing climate data over Europe for simplicity and to reduce the amount of data management needed. 960 hours were available and had to be distributed equally between the four group members. Climate data can require a lot of disk space and time to pre-process for visualization which limits the number of parameters this project can present to the end user.

2 Related Work

Visualization tools in climate research is not something new and there exists quite a few applications online that has already been developed. On the Swedish Meteorological Institute (SMHI) website they present images of climate data for a set of different parameters, with appropriate color scales. These images are static, but the application is still useful.

An obvious difference between most of the visualization tools for climate research that is available at the moment and the visualization tool implemented in this project is that this application does not contain static images, but will choose what to show depending on the options chosen. The user can also choose data from a certain time and can easily see differences in the data between years, if using the time scrollbar available in the tool.

There already exist a few visualization tools which do not contain static images. In those cases the main difference between those tools and this product is the combination that this tool is both interactive and also close to real-time. Several other climate visualization tools were tried, to get an opinion on what already was on the market, and they could show data in a similar way as this tool, although they were much slower when interacting with the user. Real-time capabilities are crucial when developing interactive tools to prevent extensive waiting time for users.

Another difference from most other climate visualization tools seen is that the tool developed can show statistics for the chosen data. The statistics are shown in a graph, where there is a line representing the value during the reference period, making it easy to compare future data to the reference period.

3 System Overview

The system developed in this project consists of two major parts. One client application and one server application. As the system is a web application data is handled by the server and requested from the client. Programming languages used is *Python* for server related implementations and *Javascript*, *HTML* and *CSS* to handle the client application. *WebGL* was used to visualize a map over Europe with climate data.

Data is pre-processed and packaged into a suitable format on the server side, and the client side is then responsible for displaying the data and all interaction with the user. The user gives input to the system through the web browser using the mouse, which in turn generates a series of events on the server side. A graphical overview is presented in figure 1.

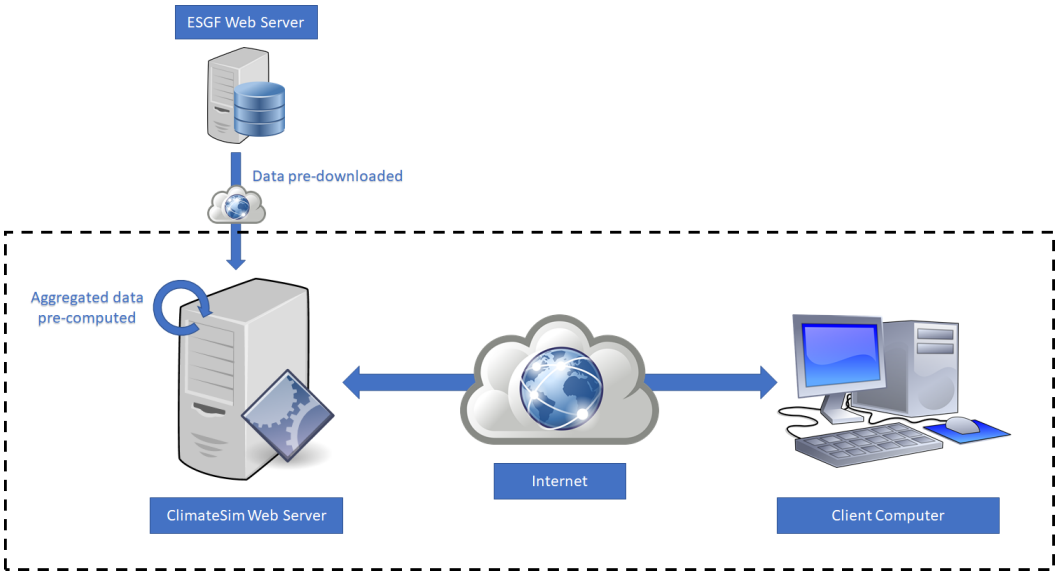


Figure 1: An overview of the system as whole. Data is downloaded from ESGF servers to the ClimateSim server. The user can access the data through the ClimateSim application over the internet. Images are taken from [2][3][4][5]

Important sub modules in the system are

- Coordinate Systems
- Server
  - Data management
  - Heavy Precipitation, Extra Warm Temperatures and Longest Heatwave in a Year
  - Aggregation of Data
  - Map
  - Statistics
- Client
  - Rendering the Map
  - Texturing
  - Color Scales and Legend
  - Mouse Interaction with the Map
  - Camera Movement

In this section an overview of each module will be presented. For a more in depth description, see section 4.

Before the server and client modules are introduced system functionality that is shared between the two is presented.

3.1 Coordinate Systems

There are several coordinate systems to handle when working with climate data. The most common coordinate system is latitude-longitude-coordinates with a specified north pole. The north pole may be placed differently than the geographical north pole. The reason for this is to increase grid resolution for climate simulations. If the north pole is not aligned with the geographical north pole a transformation may be needed before computing statistics or performing texturing operations. One could decide to interpolate and transform all the data to align with the geographical north pole and the accompanying coordinate system. Another option that was chosen in this project is to transform the required data into this new coordinate system when needed. The strategy that was used was to transform longitude and latitude coordinates to spherical coordinates, and then to a Cartesian coordinate system. After this the Cartesian coordinates was rotated before finally transformed back to longitude-latitude coordinates. This transformation is described in detail in section 4.1 and is visualized in figure 2.

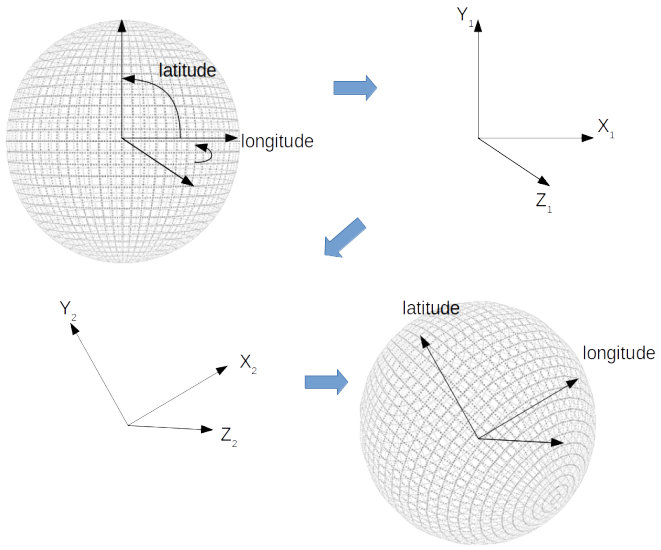


Figure 2: Climate data can be given in a rotated coordinate system (relative to the geographical). The figure shows the transformation chain for taking spherical coordinates from one system to another.

3.2 Server

The server handles all the data and is responsible for performing all the time consuming calculations. All the data that should be displayed in the application is stored on the server, and therefore also maintained by the server. The server uses the *flask* library to create an interface for applications who wants to talk to the server. In the following subsections a brief overview of what is involved in every subsystem and the solved challenges are presented. For more details, see section 4.2.

3.2.1 Data Management

Climate data is large in size and needs to be managed in a good way. The server reads the data files on start and maintain handles to all the data files while running. The data is stored in the highly flexible *netcdf* format. The server stores meta data about the files in memory, but the data itself is only unpacked when requested, otherwise the data would quickly overflow the system's memory limit. In section 4.2.1, more details regarding data management is presented.

3.2.2 Heavy Precipitation, Extra Warm Days and Longest Heatwave in a Year

As mentioned in section 1.3, the parameters precipitation, max temperature, heavy precipitation and extra warm days was studied. The needed data for heavy precipitation and extra warm days was aggregated from the *CORDEX* data. In each geographical coordinate (longitude, latitude) for every measurement in time the value for precipitation or max temperature was compared to an appropriate threshold. For heavy precipitation this was selected as more than 20 mm/day precipitation based on similar datasets developed by SMHI and from feedback given from SMHI.

The climate in different parts of Europe changes from the northern to the southern parts, and similarly from the east to the west. Therefore the definition for what is considered an extra warm day needs to be different for different parts of Europe. This was achieved by calculating a percentile for every country in Europe from an evaluation dataset for the reference period. This percentile is then used as a threshold for the max temperature data to find extra warm days.

Another dataset that can be computed from data of maximal temperature is the longest heatwave in a year. This dataset has not been stored on the web server, but the functions for computing the dataset is implemented.

For more details on how these datasets were formed, see section 4.2.2.

3.2.3 Aggregation of Data

The raw data downloaded from the CORDEX Database has a certain resolution. In our case monthly resolution was mostly used. In general when studying climate changes large time intervals are important to reduce noise and gain information about trends. The data is the result of climate model simulations, which gives a prediction of how the climate will change. These predictions are more valuable when studied over a longer period of time. A common unit to use is thirty year averages of a parameter. To attain these averages the libraries *cdo* and *nco* were used [6][7]. These libraries provide an API to aggregate climate data packaged in the *netcdf* data format. The details of how the aggregation is done is presented in section 4.2.3.

3.2.4 Map

The map of Europe is represented as a set of points for the country borders. These points are given in longitude and latitude coordinates. The data was found in the *shapefiles* format, which was convenient because *python* provides a module for reading such files. When rendering the map, a surface is desired and not just a list of points. To create a surface the map was triangulated using the *Delanuay Triangulation* algorithm. This surface is used when the map is to be rendered without any data. More details regarding how this was implemented in our system is presented in section 4.2.4.

3.2.5 Statistics

When analyzing climate data statistics are a useful tool. The visualization implemented will only be able to show the data at one given time instance at a time on the screen. Statistics gives more information and opens up the opportunity for comparison between different time periods. Statistics are calculated as a mean value for every region for a given dataset. Since the dataset is already averaged over a set time period, as described in sections 3.2.3 and 4.2.3, the simple mean of all data points within a region is a good choice of statistics. The details on how this mean value is found is described in section 4.2.5.

3.3 Client

The client is the module that the user runs to use the application. It is contained entirely in the web browser and communicates with the server module over the http protocol. This module is responsible for managing user input, displaying the user interface and rendering all the data.

3.3.1 Rendering the Map

The map and the data was rendered using hardware acceleration with WebGL. All the data for the map was uploaded into buffers in the GPU and rendered using the standard OpenGL pipeline. Since the map is in 2D the matrix that transforms the map onto the screen is only a scaling and translation. The data was rendered onto the map by use of textures. The data was uploaded to a texture and texture coordinates was used in a fragment shader to colorize each pixel on the map. The color was determined by the value in the data. How the textures are used and how the color is applied is described more in their specific sections. More details on how the map is rendered is available in section 4.3.1.

3.3.2 Texturing

When creating a texture from a data slice the data is scaled according to the minimum and maximum value over the whole dataset.

Texture coordinates are calculated online in the shaders using a set of transformations. These are used to look up the data from the data texture. The color is stored in another texture, and the value in the data is used as a texture coordinate to look up the color in the color texture. Some of the parameters contain data with a heavy-tailed distribution. A few extremes in the data set makes the subtle variations in the rest of the data set difficult to recognize. This was solved by linearizing the dataset using the "*Head/tail Breaks* algorithm" [8]. The linearization is applied to the texture coordinates to produce a new set of coordinates that gives the final color.

Read more about the texturing implementation in section 4.3.2.

3.3.3 Color Scales and Legend

The linearized data samples an array containing a color scale. Each parameter has an unique color scale. In section 4.3.3 there is more information about how this is implemented.

3.3.4 Mouse Interaction with the Map

The user can interact with the map using the mouse. Specifically, a country can be selected by clicking within its borders with the mouse. For this to work, the position of the mouse needs to be determined and used as input. The map is expressed in longitude and latitude coordinates, or in other words in world coordinates, as described in section 3.2.4. Interaction with the map through the screen occurs in screen space. To complicate things further, the mouse click is registered with coordinates expressed as the position in the HTML element containing the *WebGL* rendering surface. The mouse coordinates are transformed from this form to world coordinates. Then an algorithm is used to check if the mouse is inside any of the countries. A more detailed description of all the transformations involved and the final algorithm is found in section 4.3.4.

3.3.5 Camera Movement

To navigate what is visible on the screen, the user can move the camera object. The camera refers a viewpoint, a direction and a look at point in general in the field of computer graphics. As mentioned in section 3.2.4, the transformation from world coordinates to camera or screen coordinates is here a scaling and a translation. To move the camera means to change these two parts of the transformation in a suitable way. Translation to move the camera sideways and scaling to zoom in or out. The mouse wheel or trackpad is used to alter the zoom, while the user can alter the translation by click and drag functionality. The details on how the translation and scaling is updated is described in section 4.3.5.

4 Implementation

In this section the implementation of all the modules presented in section 3 is presented in more detail.

4.1 Coordinate Systems

If the data was expressed in a different coordinate system, where the north pole has been moved the data contained this information. In that case, a transformation from the original longitude-latitude coordinate system to the new coordinate system was created. This transformation was done in several stages. The first step was to convert the coordinates (lon_1, lat_1) to Cartesian coordinates (x, y, z) according to equation 1.

Since the transformation uses trigonometric functions and the coordinates are expressed in degrees they are first converted to radians.

$$\begin{aligned} x_1 &= \cos(lon_1)\cos(lat_1) \\ y_1 &= \sin(lon_1)\cos(lat_1) \\ z_1 &= \sin(lat_1) \end{aligned} \tag{1}$$

This is not the standard mapping from spherical to cartesian coordinates, but that is because the definition of longitude and latitude is different compared to how spherical coordinates are usually defined. Latitude goes from zero at the equator and increasing towards the north pole, while the angle in spherical coordinates is defined with origo in the north pole and increasing towards the equator. Next step is to rotate this cartesian coordinate system around the y and z-axis. First, the two angles to rotate with has to be found. The new coordinate system is described by the new location of the north pole X_{NP}, expressed in longitude and latitude. This is used to find the position of the south pole X_{SP} as

$$X_{SP} = \begin{pmatrix} lon_{SP} \\ lat_{SP} \end{pmatrix} = \begin{pmatrix} lon_{NP} - 180 \\ -lat_{NP} \end{pmatrix} \tag{2}$$

From this information the two angles θ and ϕ are found as

$$\begin{aligned} \theta &= 90 + lat_{SP} \\ \phi &= lon_{SP} \end{aligned} \tag{3}$$

These angles are then converted to radians. Next step is to form the transformation that will rotate the coordinate system (x_1, y_1, z_1) to a new coordinate system (x_2, y_2, z_2), by rotating θ around the y-axis and ϕ around the z-axis. The final equation that describes these rotations is presented in (4).

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \begin{pmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \tag{4}$$

After this transformation the new cartesian coordinates are transformed back to longitude and latitude by

$$\begin{aligned} lon_2 &= \arctan(\frac{y_2}{x_2}) \\ lat_2 &= \arcsin(z_2) \end{aligned} \tag{5}$$

Finally, the new longitude and latitude coordinates are converted back to degrees.

4.2 Server

In this section an in depth description of how the server module is implemented will be presented.

4.2.1 Data Management

The different datasets are stored on disk in a file structure that represents and groups the data in a reasonable way. At startup, the server traverses through this file structure, opens the *netCDF* file for each dataset and reading additional info about the file and stores the data in an internal data structure. When possible, JSON files are used for storage of pre-calculated data such as statistics to give a speedup and improvement of the user experience.

4.2.2 Heavy Precipitation and Extra Warm Days

Heavy Precipitation A function has been implemented to calculate days with heavy precipitation from data of precipitation. This is done independent on the web application, but needs to have the precipitation data in day resolution.

The implementation is done by comparing data from every data point in each time step with a threshold specified as 20.0 mm per day. This will result in data for each point and in each time step where the value is 1 when the precipitation is heavier than the threshold and 0 if it is not. This data is saved to NetCDF files which will be possible to read and show on the map just as other datasets. However, this dataset is in day resolution and hard to grasp if visualizing only one day at once on the map. Therefore the data is aggregated to according to what is written i section 4.2.3.

Extra Warm Days In a similar way as heavy precipitation is calculated from data of precipitation, extra warm days are calculated from data of maximal temperature of a day. The result after aggregating the data calculated will be how many days in the time period on average there are with a maximal temperature over a threshold, where the threshold is different for each country.

When computing unusually high temperatures there is a need to know the regular temperature to be able to define what an unusual temperature is. In this case an usual temperature is different for different locations and has to be determined in another way than by a fixed threshold for the whole Europe. To determine these thresholds a percentile of all the data in the country for the reference period was calculated. To choose what percentile that was reasonable, a percentile that resulted in a threshold of 25°C in Sweden was chosen. The same percentile was used for every country, but calculated on the data in their own country, which therefore resulted in different thresholds for every country.

To only get the data from one country to calculate it's threshold, an index image was created from data of the country's limits and longitude- and latitude coordinates from the data. All points inside of the country got the value 1 and all points outside the country got the value 0.

After creating an index image over the country, the threshold was calculated. This had to be done in two steps because of the large amount of data. At first the percentile was calculated over all the data points in the country for the same time step. This would

result in one threshold for the country in every time step. The next step was to calculate the percentile over these thresholds, which would result in one threshold for the country over the whole reference period.

When thresholds were calculated for each country, an image over Europe with the countries' thresholds were created. This image had the threshold for each country in all the points of the country. The values in this image could then easily be compared element-wise with an image with the data in each point.

Longest heatwave in a year A function has also been implemented that will take in a dataset with the parameter "Daily maximum Near-Surface Air Temperature" with day resolution (the same as when calculating Extra Warm Days in paragraph *Extra Warm Days*) and create a dataset of data for how many days it is in the longest heatwave in the year. The number calculated is for one year, but is saved once in every month so that when selecting a month it will always be the same value for all the year.

The definition of a heatwave used by SMHI in a fact sheet about heatwaves in Sweden says that a common way to define a heatwave in Sweden is to say that it is a coherent period of at least five days, where the maximal temperature for each day is above 25°C [9]. Therefore the thresholds calculated in paragraph *Extra Warm Days* were also used here, which would make the threshold used for the data in Sweden to agree with the threshold in this definition. The thresholds for other countries are still calculated in the same way and are different for each country depending on their reference data.

To keep track of the number of days in a row of days with higher temperature than the thresholds a counter matrix is created. In the counter matrix the current days in a row with higher temperature than the threshold is saved for each data point. Another matrix is also created in the same size with the maximum values of days in a row for each data point.

To calculate the maximum number of days in a row for each data point, it loops through all the data points with values and compare with the threshold map. If the point is above the threshold, one is added to that value in the counter. If it is not above the threshold it means that a heatwave may have ended, thus the counter is compared to the value in the matrix of the maximum values of days in a row this far. If the value is higher it will overwrite the smaller one. This process will continue for every day of the year, and when the year is over a new layer in the matrix of maximum values of days in a row are added for the next year. Then the same thing will continue until the end of the dataset.

When maximum days in a row of temperatures over the thresholds are calculated for every year in the dataset, all the values less than five days is set to 0, because less than five days in a row of warm weather is said not to be a heatwave, according to the definition stated earlier. The result will be saved to a NetCDF file and can hereafter be used in the web application as a dataset.

4.2.3 Aggregation of Data

The process of aggregating the datasets involved modifications of the time axis in the data. From the user manual for *cdo*[10], important functions are

- *selmonth* - Select all the datapoints in time for a certain month
- *selseason* - Select all the datapoints in time for a certain season, where seasons are defined as
  - Winter - December, January, February
  - Spring - March, April, May
  - Summer - June, July, August
  - Autumn - September, October, November
- *yearsum* - Aggregating the data to the total for each year, all points in the same year are added together.
- *seldate* - Select all datapoints between two given dates. Used to extract thirty years at a time from the data.
- *timmean* - Calculates the mean over all time steps in a data file, in each spatial coordinate (longitude and latitude).

These functions were put together into a bash script that aggregates the original data into the final set of datasets. These include for each parameter ten and thirty year averages over each month and season, but also an average over the whole year. This gives the mean precipitation and max temperature for a month, season or per year in a thirty year period in each geographical coordinate.

Important to note that the Heavy Percipitation and Extra warm Temperature datasets go through the same process, but with a slight alteration. Since this data represents if a day was below or above a threshold, giving binary 0/1 data, a mean value over all the timesteps makes no sense. Instead the total sum of all data points for the month or season in each year, or the whole year is calculated, resulting in datasets that contain the mean total number of days below or above the threshold during the time period, averaged over a number of years.

4.2.4 Map

The Delaunay Triangulation algorithm uses a closed polygon as input. These are found as a list of points for each individual region of land in Europe. The triangulation is then performed separately for each such region. The algorithm for Delaunay Triangulation that is used forms triangles inside the convex hull of a set of points, so therefore an additional step was needed to remove triangles outside of the initial set of points. This was done by checking if the center of a triangle was inside the polygon formed by all the points. To test if a point is inside a polygon an algorithm similar to the odd-even rule [11, pp. 221-222] was used, where each line segment of the polygon was tested. An

algorithm for ray-casting was used to test each line segment. The actual implementation was available online at GitHub, see [12]. If the center point of a triangle is outside the full polygon formed by all the points, the triangle is discarded. This test is not a perfect one, but good enough for our purposes. Some small artifacts are present along the edges of a region, but they do not impact the visual appearance to much unless the user is very close to the map with the camera. The data that represents the map is then packaged into a suitable format, grouping all regions belonging to a country together, and stored internally in the server.

To improve performance and reduce redundancy, a JSON file is written as a final output from this whole process. Next time the server receives a request to retrieve the map data, and especially after it has first been restarted, this file can be read instead of having to perform the time consuming Delaunay Triangulation and the post processing. This whole process is repeated for every type of map data available. This includes at time of writing all the countries of Europe and the regions of Sweden.

4.2.5 Statistics

Statistics are calculated country by country, giving the mean value within a country. To extract all the data points within a country the raw map data is here used to render an image with the same size as the data array. In the image each region is given its own integer pixel value to segment it from all the other regions. This image acts as a mask to extract all the data points within a certain region.

The map data is given in longitude and latitude coordinates as discussed in section 3.2.4. Therefore a transformation is constructed transforming these coordinates into the interval given by the data. In order to extract all the data points within a country the coordinates describing the country needs to be expressed in the same coordinate system as the data. This is solved by transforming all the map data according to section 4.1. When the map data is in the same coordinate system as the data, another transformation can be constructed. This second transformation is constructed from properties in the data, and enables rendering of the map data into an image. The transformation consists of three stages, a translation, followed by a scaling and projection. First, translation according to

$$T = \begin{pmatrix} 1 & 0 & -lon_{min} \\ 0 & 1 & -lat_{min} \\ 0 & 0 & 1 \end{pmatrix} \tag{6}$$

which translates the coordinates so the origin in the new coordinate system is in (lon_{min}, lat_{min}), where (lon_{min}, lat_{min}) is found in the data. Next, the scaling according to

$$S = \begin{pmatrix} \frac{width}{lon_{max} - lon_{min}} & 0 & 0 \\ 0 & \frac{height}{lat_{max} - lat_{min}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{7}$$

where width is the number of coordinates along the longitude axis in the data, and height is the number of coordinates along the latitude axis in the data. This part scales the

coordinates to fit the size of the data, and scales the data accordingly. Finally, the last step project the data onto the xy-plane as

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{8}$$

The total transformation is given by

$$Transform = PST \tag{9}$$

When working with a uniform grid sampled in longitude and latitude coordinates over the surface of the earth as done here, there will always be some degree of distortion. The result from a uniform grid is that each grid-cell represents different simulation volumes. This is compensated by calculating a scale factor for each pixel which is used to reduce the amount of distortion when calculating the statistics. This is applied as a last step before computing the mean of all the data points for a specific country.

4.3 Client Software Modules

In this section the implementation and its details for each sub module of the client is presented.

4.3.1 Rendering the map

The map is rendered with longitude and latitude directly mapped against screen coordinates. This results in a distorted representation of the map but it suffices for visualization purposes. The scaling and translation matrix mentioned in section 3.3.1 determines what area of the map is visible in the GL viewport. Also, a frustum transform is used to handle changes in the browser window size. The translation matrix moves the camera and the scaling matrix handles the zoom functionality. The movement of the camera is described in more detail in section 4.3.5. The initial values for these transformations are

$$T = \begin{pmatrix} 1 & 0 & -14 \\ 0 & 1 & -53 \\ 0 & 0 & 1 \end{pmatrix}$$

$$S = \begin{pmatrix} \frac{9}{360} & 0 & 0 \\ 0 & \frac{9}{180} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P = \begin{pmatrix} \frac{400}{canvasWidth} & 0 & 0 \\ 0 & \frac{400}{canvasHeight} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The numbers may seem random, which is true for the translation. For the scaling 360 corresponds to the span of the longitude axis and 180 to the span of the latitude axis. The user can move around the camera by updating these matrices. This is further described in section 4.3.5.

4.3.2 Texturing

The climate data is on a uniform grid format in longitude and latitude coordinate system. This is also the reason why the map is rendered as described in 4.2.4. The data is stored in a texture and to access the data texture coordinates are needed. These are calculated online in the shader based on the data sent to the shader. As mentioned in section 4.1, it may be that the map is not in the same coordinate system as the data. This is solved by implementing the procedure outlined in section 4.1 in the fragment shader. Once the map and the data is expressed in the same coordinate system a transform is created that for each point in the map calculates the texture coordinate for that point. This transformation consists of a translation and a scaling. This transformation matrix is called *texFromWorld* and is derived as in (13). Here lat and lon stands for the coordinate vectors in the data.

$$\begin{aligned} lon_{range} &= lon_{max} - lon_{min} \\ lat_{range} &= lat_{max} - lat_{min} \end{aligned} \tag{10}$$

$$S = \begin{pmatrix} 1/lon_{range} & 0 & 0 \\ 0 & 1/lat_{range} & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{11}$$

$$T = \begin{pmatrix} 0 & 0 & -lon_{min} \\ 0 & 0 & -lat_{min} \\ 0 & 0 & 1 \end{pmatrix} \tag{12}$$

$$texFromWorld = S * T \tag{13}$$

The data in the grid is scaled with the minimum and maximum values for the whole set and then sampled in a shader program. Some of the parameters have a so called Heavy-tail distribution. If the data was to be sampled linearly we would get a result as shown in Figure 4a and 4b if the color texture would be sampled right away. A few extreme values make the rest of the data very difficult to differ from each other. The data is not evenly distributed in its values, which leads to problems with the resolution in color when rendering. Therefore a "Head/tail breaks algorithm" [8] is used. The algorithm divides the data set into a head part and a tail part as seen in Figure 3. The head is used as the data set for the next iteration. A new mean is derived and the data set is divided into a head and tail part once more. This is done til a desired length of the head set is reached. The mean for every iteration is stored in an array. This array is used together with a function for mapping the means from "Head/tail breaks algorithm" to create a linearizing texture, see algorithm 1 to be sampled in the shader program. Figure 4c shows the data colorized after sampling the linearizing texture.

Input: Array of Head/tail breaks means for chosen parameter $HBT[mean_1, mean_2, ..., mean_i]$**, a mapping function** Y_i **and an array with values** $array[0, 1, ..., N]$**.**

for *every element* i **in** HBT **do**

  $pixel_i = HBT[i] * array.length$

  $pixel_{i+1} = HBT[i+1] * array.length$

  $Y_i = f(mean[i])$

  $Y_{i+1} = f(mean[i+1])$

  for *every element* k **in** $(pixel_i, ..., pixel_{i+1})$ **do**

    tex[k] = LinearInterpolation(Y_i, Y_{i+1})

  end

  LinearizedTexture.push(tex)

end

Algorithm 1: Algorithm for linearization texture.

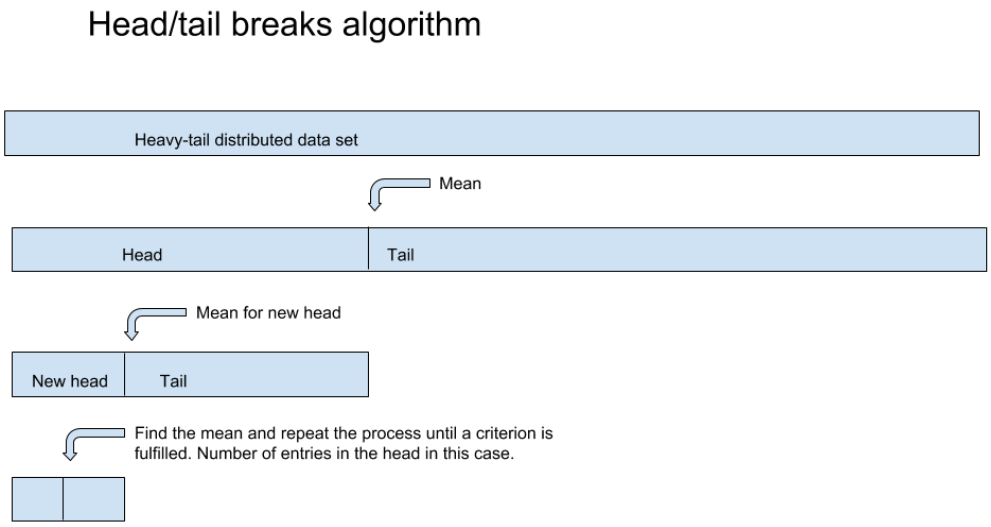
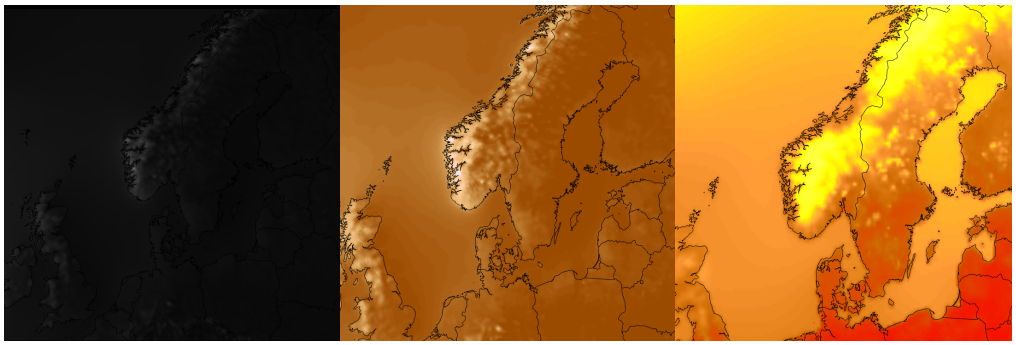


Figure 3: The "Head/Tail breaks algorithm"

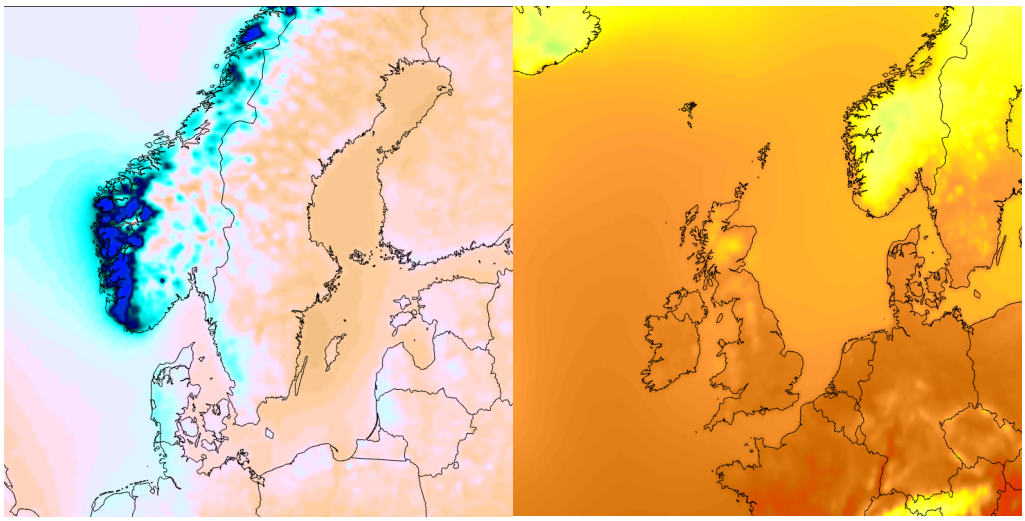


(a) Sampled data. (b) Sampled color scale. (c) Linearized texture sampled.

Figure 4: Figures showing the map after the color scales have been sampled.

4.3.3 Color Scales

Each parameter have a suitable color scale. The color scale is as rgb-values in an array. For example, the color scale for precipitation-based parameter ranges from brown representing areas with low precipitation to a rich blue color representing areas with high precipitation as seen in 5a. While the temperature-based parameters ranges from a green color for low temperatures to a strong red color for hot temperatures as seen in 5b. The linearized texture derived in 4.3.2 samples the array with the color scale for the chosen parameter in a shader program.



(a) pr. (b) tasmax

Figure 5: Figures showing the map after the color scales have been sampled.

4.3.4 Mouse Interaction with the Map

The *WebGL* rendering surface is a set of pixels on the screen. A mouse click registers the mouse position as a pixel (x, y) in the HTML element, which is one of these pixels in the rendering surface. The HTML element has a certain size, typically around 600 x 800 pixels. The data for the different countries in the map data is all expressed in world coordinates, as longitude and latitude coordinates. In order to be able to compare the mouse position with the countries, the mouse coordinates X_{mouse} are transformed into world coordinates X_{world}. This is done by a combination of two different transforms. The first one is a mapping from X_{mouse}, the pixel grid in the HTML element, to the *WebGL* rendering surface and screen coordinates X_{Screen}. The rendering surface in *WebGL* is a square around origo in the interval $[-1, 1]$. The transformation is according to (14).

$$X_{Screen} = 2 * \begin{pmatrix} \frac{1}{canvasWidth} \\ \frac{1}{canvasHeight} \end{pmatrix} X_{mouse} - \begin{pmatrix} 1 & 1 \end{pmatrix} \tag{14}$$

The second transformation is from screen coordinates X_{Screen} to world coordinates X_{World}. It is simply the inverse transformation of what usually is called the camera matrix, or camera transformation. This is the transformation that is used to transform the world coordinates into screen coordinates in the rendering pipeline. In our case, as described in section 4.2.4, this is a scaling and translation moving the camera to the center of Europe in world coordinates and scaling the data to fit the screen. Applying this transformation to X_{Screen} gives us the mouse position in world coordinates X_{World}.

The last step to finding which region the user clicked on if any, is to check if the mouse was inside any region. To do this, algorithm 2 was used. The algorithm was originally described and developed by others, and available to download at *GitHub*, see [12].

Input: A set of points **P** describing a polygon, the border of a region. A point (x,y) representing the mouse.
for *Line Segment* **l in** *Polygon* **do**
  The Line Segment is described by two points $X_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, $X_j = \begin{pmatrix} x_j \\ y_j \end{pmatrix}$.
  Calculate the boolean expression
  $((y_i > y) != (y_j > y))$ **AND** $(x < (x_j - x_i) * \frac{y - y_i}{y_j - y_i} + x_i)$
end
If the expression evaluates to true an odd number of times the point is inside the polygon, inside the region, otherwise outside.

Algorithm 2: Algorithm for selecting a region with the mouse.

4.3.5 Camera Movement

Each time the user moves the mouse the state of the mouse is updated. In this way, the program knows at all times where the mouse is and its velocity. The velocity is described

as $\Delta\mathbf{x} = \begin{pmatrix} dx \\ dy \\ dz \end{pmatrix}$, and the position as $\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

Altering the translation is done when the user click and drag the mouse. The vector describing the movement from start to end position is easy to find from the state, and is defined as $\mathbf{m} = \begin{pmatrix} dx \\ dy \end{pmatrix}$. A translation matrix representing this movement is formed and applied to the original translation matrix. This moves the camera from its former position by the movement vector to its new position.

The zoom functionality is operated by the mouse wheel or track-pad. When the scroll event is received, the state of the mouse is once again updated. This time changing only the z-component, and the velocity in the z-direction. To avoid a too large scroll speed and to compensate for differences between computers and web browsers the speed in the z-direction is capped to a max limit.

A new scaling to apply along the x and y-axis is found as

$$ds = 1 - \frac{dz}{100} \tag{15}$$

This new scaling is then applied similarly to how the translation was updated, by forming a new scale matrix that performs this scaling and multiplying the new scale matrix with the old scale matrix.

5 Results

ClimateSim web-application can properly visualize climate data for four different parameters at near real-time. In this context, real-time means that the user is not able to notice the time delay when changing a parameter until that the data arrives and is displayed. The parameters are precipitation, heavy downfall, daily maximum near-surface air temperatures and extra warm days. Precipitation and daily maximum near-surface air temperatures are data from CORDEX climate database while extra warm days and heavy downfall are parameters derived from precipitation and daily maximum near-surface air temperatures. Statistics are derived for each parameter and can be visualized on demand.

A web server is positioned at the university with a domain where the web application can be reached from a fairly new computer with drivers that are sufficient and where the computer has access to internet and an updated browser.

In figure 6 the web application can be seen before any parameter has been chosen. Basically it shows a map over Europe. In the top left corner there are buttons to modify which parameter to visualize and if the statistics for a chosen country will be visible.

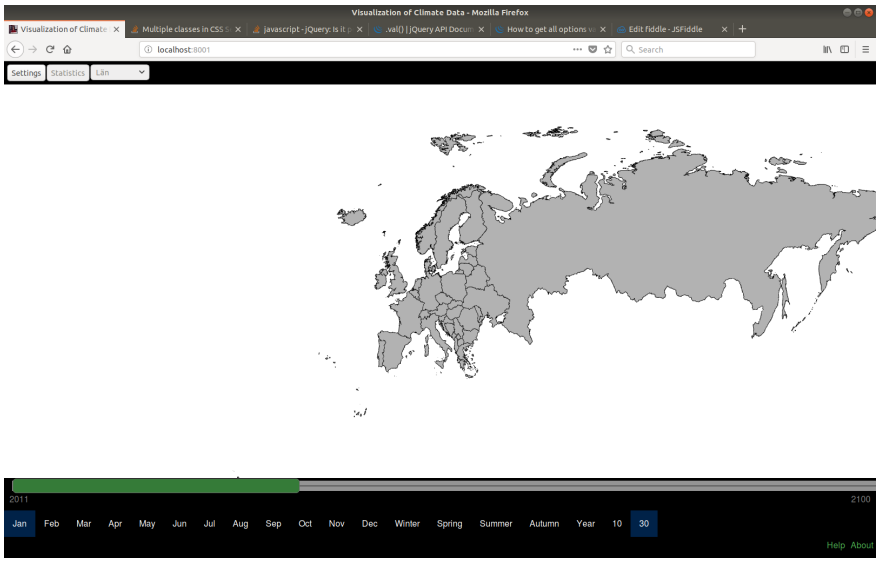


Figure 6: The web application before any parameter is chosen.

If the user clicks the button *Settings* in the top left corner, a pop-up box will be visible where the user can choose which parameter to look at. This is shown in figure 7.

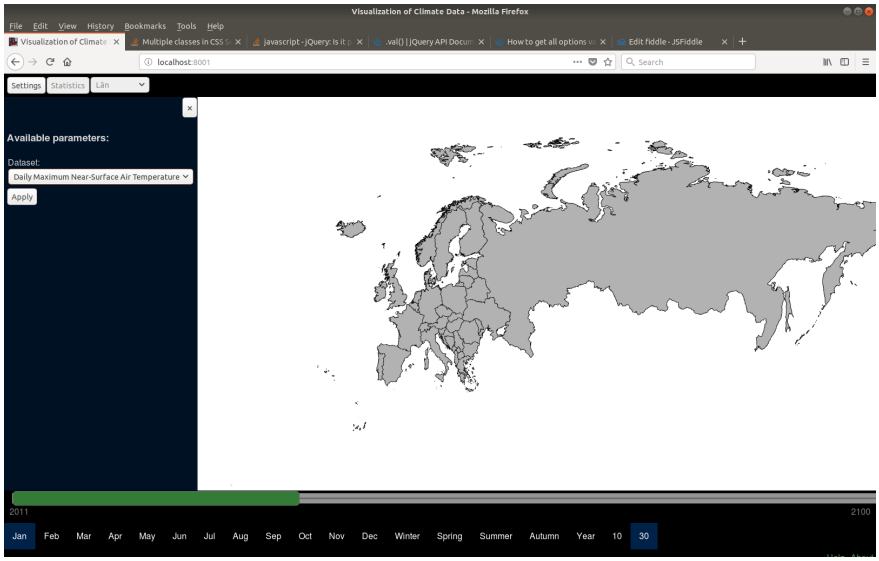


Figure 7: The web application when choosing a parameter.

When a parameter is chosen, the data will be painted over the map and a color bar will tell what is pictured and what the limits between the colors are. An example of data

for a parameter is shown in figure 8 where the map is zoomed out to see the all data available in the dataset.

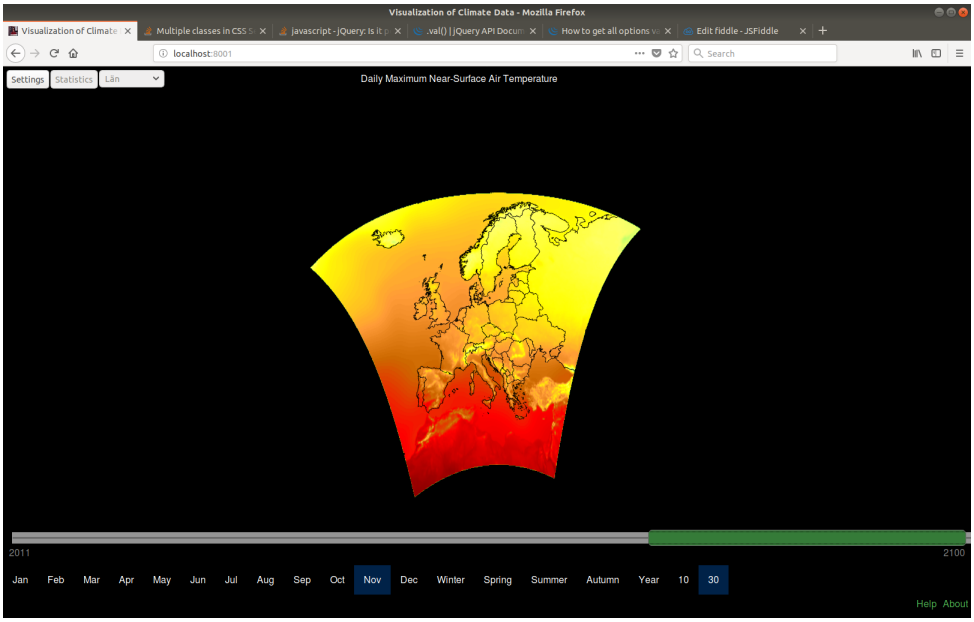


Figure 8: The web application after choosing a parameter and zoomed out the map.

The map can also be zoomed in to clearly see a specific place on the map. By clicking on a country on the map, a country will be chosen. If the user choose to look at the Statistics, it will then be calculated for the chosen country. An image of how that can look like is shown in figure 9. After selecting Sweden as a country, it is also possible to further choose a county or municipality. This is only available for Sweden for now, but can of course be further expanded.

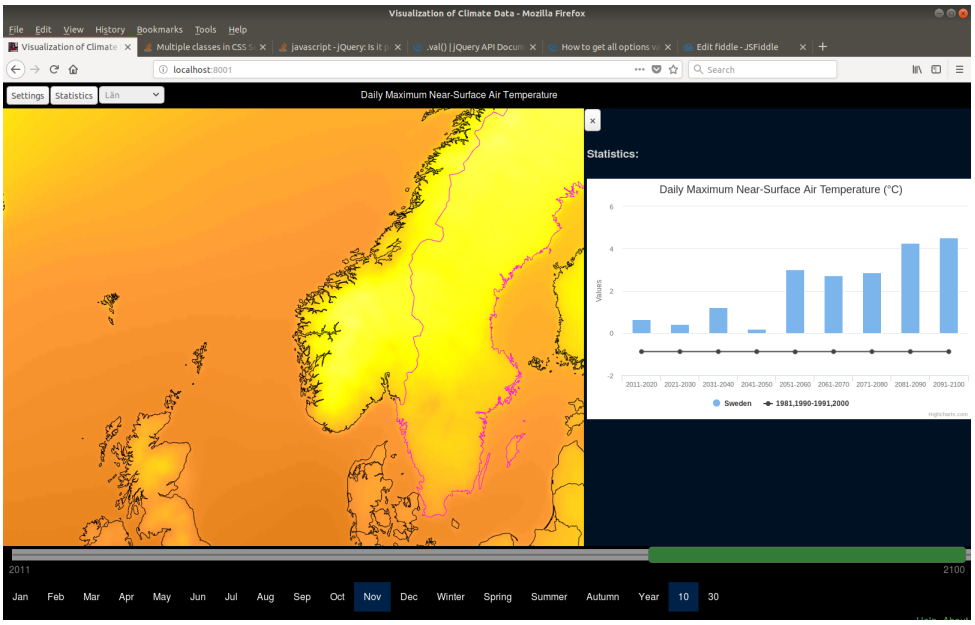


Figure 9: The web application when zoomed in and showing statistics.

As can be seen in the figures 6-9, there is options in the bottom of the page. A month has to be chosen, and that will visualize a mean over that month. The user also have to select if the data painted on the map and the statistics will be computed as a mean over 1, 10 or 30 years. This mean will still only be for the chosen month over a span of years.

Furthermore, as can be seen in the figures in this section, there is a time scrollbar right above the options in the bottom. When moving the scrollbar, the same parameter and options can be seen for different time periods, which will enable the user to see the climate changes between the year written to the left of the scrollbar and the year written to the right.

6 Discussion

How reading and storing the climate data is done in a good way or if it could have been done in a better way can be further discussed. To develop an interactive visualization tool it has to be responsive. When the user clicks on a button or chooses a parameter, the next step has to be performed fast before the user gets tired of waiting and stops using the tool. Because of a heavy amount of data stored in NetCDF files, which we found could be time-consuming to read, we found a way to store the data in a temporary data structure while using the tool.

The data files are on the format NetCDF Classic, which means the data is stored in a binary format. When reading it we store it in JSON (JavaScript Object Notation)

format, which is easy for computers to both parse and generate. It is also easy to read and write for developers, and is therefore a motivated choice. A disadvantage is that the data files are, as already mentioned, in binary format, and to read and convert to JSON format takes time and effort for the computer as this is done every time the web application opens. If there was a library with functions to make the reading from the data files easier and faster, this problem could be less of a problem.

Another way to make it faster to read the data could have been to read and convert the data to another format and save new data files with the processed data. Then the time to read the data could be reduced. It could also be discussed where to store the data because the files are very big, even though that goes a bit hand in hand on what has already been mentioned. If it should be stored on the server, it has to have memory enough to do so. If it has to be stored somewhere else, it may be even more time consuming when reading.

A possible way to reduce the time waiting when using the tool is to read and calculate as much as possible in beforehand or by threading and thus be able to do things parallel to each other. For at least some parameters it may be possible to reduce the size of the data by decrease the number of decimals without reducing the accuracy of the data. When displaying the data, there is for most cases no reason to show a span of six decimals or more. That will not give a clearer picture of the climate. Especially not since what is shown is climate data from scenarios that has not yet happened.

It's performance over the web is a bit unstable from time to time, which probably is due to that the server is not great and also it may be disturbance in the network or it may be due to a low connection speed (Mbit/s). When running the application on a Python server on the client computer, the performance is better and it is close to real time. A faster server would of course increase the speed of the web application. Although if expanding the web application to handle more parameters/data, it could slow down the tool depending on how and when the data is being read. That will lead us to the next question, what is the better: developing the tool as a web application or as program? Both has it's advantages and disadvantages. A web application can be made more accessible to more people, but has also a risk to be more unstable because of different browsers and the fact that they have different standards and support different functions. What may work in one web browser may not work at all in another or look different.

In section 4.3.1 it is mentioned that because the map is rendered with longitude and latitude directly against the screen coordinates, there will be a distortion in the map. That is, it will not be exactly how we may be used to see it, even though it is not too bad. When looking at the figures of the web application in section 5, it can easily be seen that it is a map over Europe. On the other hand it could be further discussed what a correct way to display the map actually is. A simple answer to that question may be to say that the most common way to map it to a two dimensional plane is the correct one, and this will probably be the most pedagogic way to map it. However, instead of a two dimensional map it could have been done as a three dimensional globe. The distortion we got when mapping it to a two dimensional map would not have been existed in the same way if it had been a three dimensional mapping.

Even though there is a lot of data (0.11° resolution) it may be a bit too little for smaller regions when calculating statistics. In Sweden where counties and municipalities can be selected, it can be as low as one or a few data points in each region. In theory, if there is a data point on the line of two small municipalities both of them can get the same data point. If they are that small that this is the only data they get, it may be misleading to show statistics and say that both municipalities have the same statistics when they are in fact sharing the same data point.

Moreover it can also be discussed if it is a proper way to calculate the thresholds for a heatwave with a percentile of the temperatures in the country, as is described in section 4.2.2. If there, for example, are mountains with quite low temperatures in some part of the country and beaches in another with high temperatures. That may result in a threshold that almost always will be exceeded by the warm part of the country. Although what is easy to say is that it is far better than to have the same threshold for every country, because of a small difference in the climate normals in the countries. It would be hard to find the country's own definition of a heatwave, so a median value, which is calculated with a percentile over the reference values, is at least better than several other ways.

To be able to calculate a threshold with a percentile it was necessary to calculate the percentile over the data for each country separately and also calculate the percentile for each date first. That resulted in a list with thresholds for a country in each time step. The final threshold was then calculated after calculating the percentile of the thresholds from each time step. Why this was necessary was because of the large amount of data. It was too much data in order to calculate the percentile on all the data in the country at once. As a result it may give a slight difference in the threshold by doing it in two steps. On the other hand the reason for doing so was because it was too much data for the computer to handle, so it is very likely that it did not affect the final threshold noticeable.

6.1 Further development

Additional features that could be added after the course is finished or if another project group would continue this project next year could be the following features.

- Additional parameters. There is a vast amount of climate data available.
- Add different scenarios and make it possible to compare two different scenarios in real-time.
- Simulating the scenarios in real-time.
- Dividing all countries into smaller regions. Making the application more attractive for climate researchers and policy makers in regional offices.

7 Conclusion

Climate data is large in size, complex and hard to get a grasp of, but it doesn't have to be. By scaling down the data to the information it carries, and then make it graspable, it can be both easy to understand and interesting. Not only for the most informed climate researcher but for an average person. We stand before great challenges in the changes of the climate. The attempt of this project has been to make it easy and interesting to learn about the climate changes and for the user to see how the climate may change in the future.

The visualization tool developed paints the climate data for a chosen parameter over a two dimensional map of Europe and has possibilities to be further developed. This project has shown one possible way to create a climate visualization tool. Even though the tool is far from the optimal climate visualization tool to be created and needs more time invested in it, it has some good and interesting features. It is close to real-time on a good day and with a stable internet and is then accessible not only to climate researchers but to other interest groups as well. It is also interactive with the user, since the user can choose parameter, change options and select which period of time to look at. Because the map is painted with, what is hoped to be, intuitive colors depending on parameter and because it is easy to see changes over a period of time by moving the time scrollbar, the tool is also somewhat pedagogic.

If you had an interest in the climate and what it would be like in the future, what would you say if you could find out in an interactive and pedagogic way what the climate may look like in 80 years from now?

References

[1] Keywan Riahi et al. "RCP 8.5—A scenario of comparatively high greenhouse gas emissions". In: *Climatic Change* 109.1 (Aug. 2011), p. 33. ISSN: 1573-1480. DOI: 10.1007/s10584-011-0149-y. URL: https://doi.org/10.1007/s10584-011-0149-y.

[2] *Server Png Image*. http://www.freepngimg.com/download/server/5-2-server-png-image.png. (Access date 2017-12-06) [image]. 2017.

[3] *Application Server*. https://openclipart.org/download/182736/application-server.svg. (Access date 2017-12-06) [image]. 2017.

[4] *Cloud Service Cliparts 2655074*. http://clipart-library.com/img/1768721.png. (Access date 2017-12-06) [image]. 2017.

[5] Akshay Sharma. *Workstation Computer Hardware Office Desktop*. http://maxpixel.freegreatpicture.com/static/photo/1x/Workstation-Computer-Hardware-Office-Desktop-147953.png. (Access date 2017-12-06) [image]. 2017.

[6] *Climate Data Operators*. https://code.mpimet.mpg.de/projects/cdo/.

[7] Open Source. *netCDF Operator*. http://nco.sourceforge.net/.

[8] Bin Jiang. “Head/tail Breaks: A New Classification Scheme for Data with a Heavy-tailed Distribution”. In: *The Professional Geographer* 65.3 (2012), pp. 482–494.

[9] SMHI. `https://www.smhi.se/kunskapsbanken/klimat/varmebolja-1.22372`. (Access date 2017-12-04). 2017.

[10] *CDO User Guide*. `https://code.mpimet.mpg.de/projects/cdo/embedded/cdo.pdf`.

[11] Ingemar Ragnemalm. *Polygons Feel no pain*. CreateSpace Independent Publishing Platform, 2017.

[12] *point-in-polygon*. `https://github.com/substack/point-in-polygon`. GitHub repository.