Aalborg University
Geoinformatics

Visualizing and comparing population projection rasters

Author:
Andreas Gram Riisgaard

Supervisor:
Carsten Kessler

January 2019

Titel:

Visualizing and comparing population projection rasters

Abstract:

This is edited in Formalia/Abstract

Projekt:

Thesis project

Project Period:

February 2020 - June 2020

Author:

Andreas Gram Riisgaard

Supervisor:

Carsten Kessler

Number of pages: 78
Number of annexes: 9
Afsluttet: 4-6-2020

Preface

This is edited in the file Formalia/Preface

Contents

Introduction 1

In 2019 the population in the world reached 7.7 billion people, which is an increase of one billion over the past twelve years. According to The United Nations Department of Economic and Social Affairs' (UN DESA) median scenario the growth is expected to continue reaching 9.7 billon in 2050. [**?**]

To be able to adapt infrastructures to this population growth it is necessary to predict where these people will settle. While UN DESA provides this information on a national level [**?**], it is more ideal with a more nuanced picture, since most planning are based on local or regional scale spatial projections. [**?**]

Other researchers (SEDAC, CISC) have used simulations to distribute the population within each country as raster layers. However due to the high resolution and/or small scales, visually comparing these raster datasets is a time-consuming task. The purpose of this project is to create a tool allowing fast and easy comparison of such raster datasets, focusing on the use case of population projections.

1.1 Problem statement

To explore the possibilities for creating such a comparison tool the following research question have been defined:

How can population rasters be visualized and compared efficiently and effectively?

This broad main question will be answered by answering the following three subquestions:

Which conventions exist for visualization of population projections?

Which functionalities are relevant for comparing different rasters?

How can a responsive user experience be ensured, when loading and visualizing large raster dataset?

1

[1] FiXme Note: Maybe write something about the limitation - eg. that user testing wasn't a possibility due to time

1.2 Limitations

1.3 Report structure

[2] The report have been divided into three parts. The first part is the literature review, which will address the first two subquestions and also present the two projections visualised in this project. Chapter x explores which conventions there exist for population projections, while the relevant functionalities for raster comparison are detailed in chapter x. Lastly chapter x will give an overview of the population projection SEDAC and CICS, which will be used as case for comparison.

The second part is addressing the last subquestion. First there is a definition of how a "responsive user experience" has been defined. Then different methods of visualising raster datasets are being tested in chapter x. Based on these initial tests a method will chosen, which will be evaluated in the next section.

The last part starts with a discussion in chapter x of the results of the previous part. This is then followed by the last two chapters x and x, which are the conclusion and future work.

[2] FiXme Note: ADD: quick overview of what the solution is going to be, what is population projections, SSP

Part I

Literature review

Case data The population projection visualized in this project have been created by a geosimulation, which geographically distribute the predicted population. This geosimulation have been run every tenth year from 2010 to 2100. It has also been run for different scenarios for the feature.

1.4 The data behind the simulation

This projection is a geographical distribution of the global prospects created by The United Nations Department of Economic and Social Affairs. These prospects estimate the population numbers in each country living in rural and urban areas. The spatial distribution has been based on raster data from the Global Rural Urban Mapping Project (GRUMP). In this dataset the world has been divided into 1x1 km cell, each of which contains the number of estimated people. GRUMP also have a dataset for the extent of urban areas. This urban extent has been estimated based on the light emitted after dark, which as a result means that it overestimates the extent of urban areas. To compensate for this the dataset was combined with the Global Land Cover Map from the European Space Agency (GlobCover). This dataset more accurately defines the extent of urban areas. Only the areas which qualified as urban in GRUMP dataset and were classified as "artificial surfaces and associated urban areas" in the GlobCover data were treated as urban areas.

1.5 The simulation

Using this dataset to define the extent of urban, suburban, and rural areas, the simulation was run as illustrated in figure 1.1

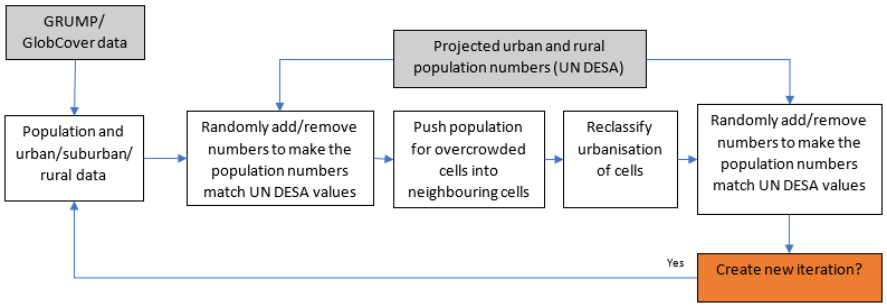

Figure 1.1: A flowdiagram of simulation creating the population projections

The population in rural and urban areas were adjusted to match the projected values from UN DESA. This was done by randomly adding or removing people from the area types.

The cells would have a country-specific limit on how many people could live in them. Any cells with a population above this limit would push the excess population to neighboring cells.

After this, the cells would be reclassified as more urbanized if their population had increased above a country-determined threshold. This way a rural cell might become a suburban one or a suburban become urban if its population increased. The data is then adjusted to match projected values again with the same approach as before. This is required because the reclassification would mean that the number for the different urbanization classes would no longer match the numbers from UN DESA.

These steps are then repeated for every iteration.

[**?**]

1.6 Shared Socioeconomic Pathways

[3]

The geosimulation have been run for five different future scenarios called the Shared Socioeconomic Pathways (SSPs). The different scenarios are based on different ways, climate change could affect society. As illustrated in figure x the five scenarios are based on two axes: socio-economic challenges for mitigation and for adaptation of climate change impacts. "Socio-economic" is referring to a large array of societal and socioecological system. Covering aspects of a political, social, demographic, cultural, lifestyle, institutional, economic, and technological nature and condition of ecosystems. The horizontal axis is environmental or societal conditions, which would make adapting to climate change more difficult. This include climate hazards (temperature and precipitation changes, sea level rise and extreme weather phenoms); what and whom these hazards will affect. [**?**] The vertical axis is factors, which in absence of climate policy would increase that amount of emissions of greenhouse gasses and factors reducing the society's capacity to lessen those emissions. Examples of factors reducing society's capacity is inadequate technologies and insufficient resources to support mitigation policies. The increase in greenhouse gasses could for example be a result of population and economic growth. [**?**]

How the population growth and the urbanization levels will be in the different scenarios can be seen in table 1.1. This table have been created by **?**. The population growth values are based on current income and fertility condition. The urbanization predictions are based on the current income.

[4]

[3] FiXme Note: Create own figure

[4] FiXme Error: Need more context to the table - what does it mean? Do I need it? - use it to select what to show in results

	SSP1 Sustainability	SSP2 Middle of the road	SSP3 Regional rivalry	SSP4 Inequality	SSP5 Fossil-fueled Development
Population Growth					
High fertility	Low	Medium	High	High	Low
Other low fertility	Low	Medium	High	Medium low	Low
Rich low fertility	Medium	Medium	Low	Medium low	High
Urbanization level					
High income	Fast	Central	Slow	Central	Fast
Medium income	Fast	Central	Slow	Fast	Fast
Low income	Fast	Central	Slow	Fast	Fast
Spatial pattern	Concentrated	Historical patterns	Mixed	Mixed	Sprawl

Table 1.1: Population growth and urbanization for each of the SSPs

1.7 Technical details about the case data

5

[5] FiXme Note: Write this - leads to the next section

1.8 Raster formatting conventions

Aside from the standard connected to the visualization of raster data there are also standard for the formatting of the raster files. Common raster formats include jpeg, png and tiff. One of the key differences between these formats are the bit depth. This value determines how many different colors that can be assigned each pixel in the image. Each bit can only be assigned one of two values, 0 or 1. This number of available colors for a pixel can therefore be calculated as $2^{numberofbits}$. As an example, a bit depth of 8 bits would then result in $2^8 = 256$ different potential colors, since is the number of combinations of ones and zeroes, that are possible. For most maps this amount of colors is enough. The human eye can comprehend around 10 million different colors, but often that many colors are not needed. When displaying aerial photographs, the depth of 24 bits is often used, since it can display 16.7 million different colors and therefore can appear in true-colors to humans. The limitations of the human eye does not mean that having more than 10 million bit combination is pointless. The bit combination can also be used for other thing than colors. It can be used to create transparency values or to store metadata about the image file. For instance, the geotiff format can use the additional bit to store georeferencing information. These advantages of higher bit count come at a cost of a larger file size. Having a larger color depth will result in a slower loading and larger requirements for storage space. [**?**]

1.8.1 Tiled raster

This section describes standards for dividing large raster files into smaller tiles. A way of only visualising the necessary parts of a raster is to divide it into smaller raster tiles and then only load the relevant tiles. When loaded into the map these tiles then gets places next to eachother, so they appear as a single large map image. These tiles can also be created with different resolutions, so that zooming in on the map with return tiles more details. As illustrated in figure 1.2 each of these zoom layers have more tiles. [**?**] When zoomed all the way out the entire world is rendered as a single tile. Whenever the zoom level gets increased by one each tile in the previous layer get replaced with four smaller tiles. The number of tiles on a zoom level z can therefore be calculated as 2^2z. [**?**]

Figure 1.2: Illustration of the increase in tiles for each zoom level

Openstreetmaps is an example of a service, which use tiled rasters for their map. When this

service is used, requests for tiles are send to http://tile.openstreetmap.org/zoom/x/y.png In the request the values zoom, x and y are replaced with the current zoom level, tile column and tile row. [?]

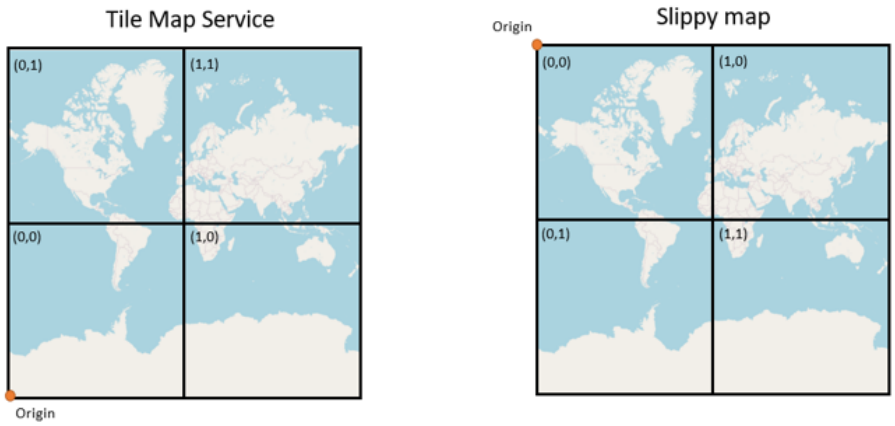

Figure 1.3: Illustration of the difference between the TMS and XYZ formats. The individual maps are inspired similar maps from ? and ?

The naming of these tiles are done differently for different standards. In this section only the Tile Map Service (TMS) and Slippy map (XYZ) standard will be addressed. Both of these have the same approach to naming zoom level and columns, but different approach to naming the rows. This difference has been illustrated in figure 1.3. The reason for the difference is that TMS tiles number their rows from south northwards, whereas XYZ are numbering rows the reverse way. Due to this difference in numbering rows loading tiles from the wrong standard results in a map as shown in figure 1.4. [?]

Figure 1.4: How the map looks, when the TSM and XYZ get switched

Visual conventions 2

This chapter will address the visual convention connected to population projection. Within the field of spatial convention there are multiple convention, however not all of these are relevant in this particular context. To understand which conventions are relevant for visualizing data one must understand the properties of the data to be visualized. Spatial data can be presented as either discrete-objects or continuous-fields also known as vector and raster data. [**?**] These two types of data have different ways of being visualised. As mentioned in chapter **??** the data is in the raster format and is sequential. Visual conventions connected to discrete-objects will therefore not be covered.

Visual conventions connected to raster maps are largely covered by convention connected to color.

2.0.1 Color

A color can be defined by three parameters: hue, saturation and lightness. These different concept have all been illustrated in figure 2.1.

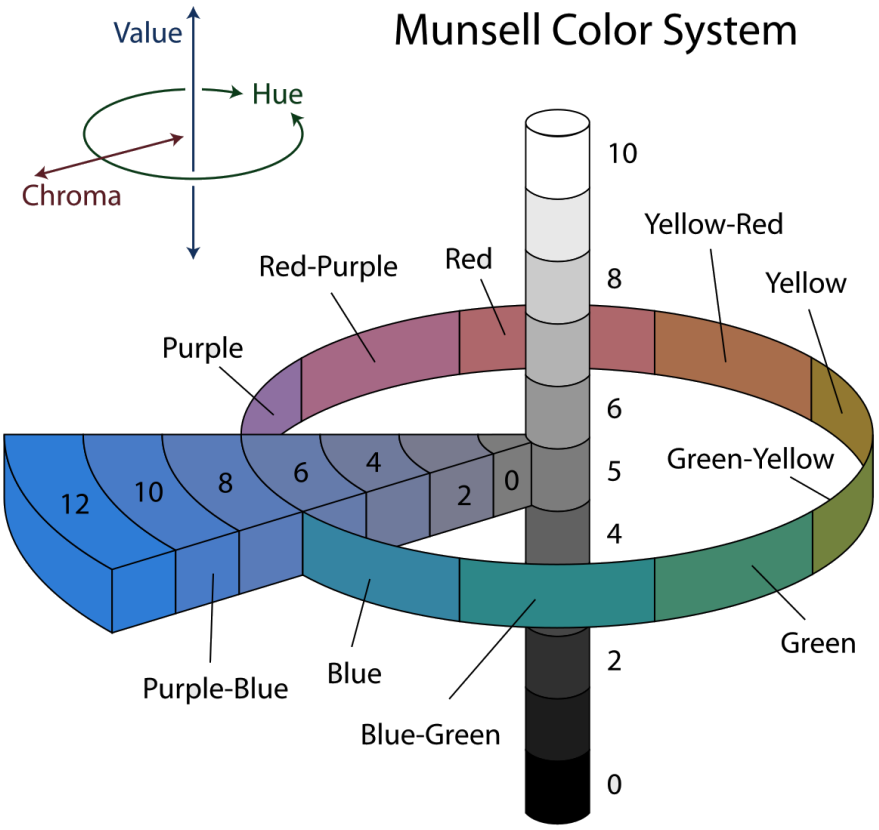

Figure 2.1: Illustration of the concepts: hue, saturation and lightness. Source: **?**

Hue

The hue is what would traditionally be referred to as colors (red, green, yellow). In the figure the hue is illustrated as the circle around the column.

Saturation

The saturation is also referred to by some authors as chroma, intensity or purity. It is a measurement for how vivid the color is. A color with a low saturation would be close to the color grey. If the saturation increases more of the color pigment is added to the color, until there is no trace of grey left. The saturation has a value from 100% (fully colored) to 0% (grey). It is illustrated in the figure as the distance from the center.

Lightness

A measurement of the color's lightness or darkness. In the literature this is often referred to as the colors value, but Brewer remarks that this use of terminology is not ideal in data science since "value" also could refer to the data values. It is illustrated as the vertical axis in the figure. [**?**]

Selecting a color scheme

There are multiple elements, that should be considered, when choosing a color scheme. Naturally the different colors should be easily distinguishable, but one should also consider the user of the map and the medium used for presenting the map. The right choice of color pattern allows the user to see patterns in complex data, which otherwise would be obscured.

Cindy Brewer divides color schemes into the four categories; binary, qualitative, diverging and sequential. These categories and combinations between them have been visualized in figure 2.2. [?]

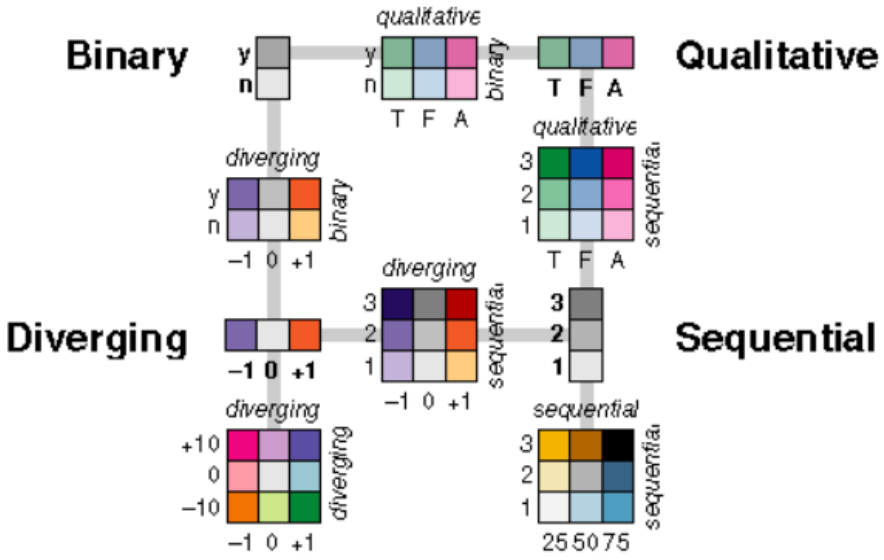

Figure 2.2: Illustration of the different categories of color schemes. Source: **?**

Each of these categories are good for visualizing different kinds of data. The qualitative is well suited for illustrating nominal data. This color scheme is for instance commonly used for land use maps. The binary scheme is a qualitative scheme, but with only two categories. A example of a use case for this scheme could be visualizing whether countries are members of the European Union or not. The sequential scheme is useful for illustrating the ordered data going from low values to higher ones. An example of this could be height of vegetation. The differentiation between the values are illustrated with differences in lightness. The diverging scheme is similar to having two sequential color schemes. This enables highlighting a critical value in the middle of the data. It is for example being used to illustrate temperatures, where the neutral middle value would be 0^oC. [?]

Based on these categories of color schemes Brewer have developed an online tool, colorbrewer2.org, which can aid mapmakers in picking a color scheme. This tool also takes into colorblindness into consideration and inform the user if a colorscheme is suitable for printing or viewing on small screens. [?]

Visual conventions for colors

The conventions can be divided based on whether the visualized data is qualitative or quantitative. The qualitative datasets have many historical convention – like using green colors for vegetation or using blue for water. It is not the same case for quantitative data, where “No conventions exist for color choice on quantitative maps. (for example, population density maps are always blue, income maps are always green, and so on)” [**?**]. There are however convention for the choice of lightness, where "light is less - dark is more"

[**?**]

Related work 3

1 2

This is not the first project attempting to visualising large raster datasets in an interactive way. It is also not the first attempt at visualizing this particular dataset.

3

3.1 Tilebased raster visualisation

This is not the first project to follow many of these core concepts. In Bernhard Baumrocks master thesis, he created a webmap, where raster tiles were being colored locally. [?]

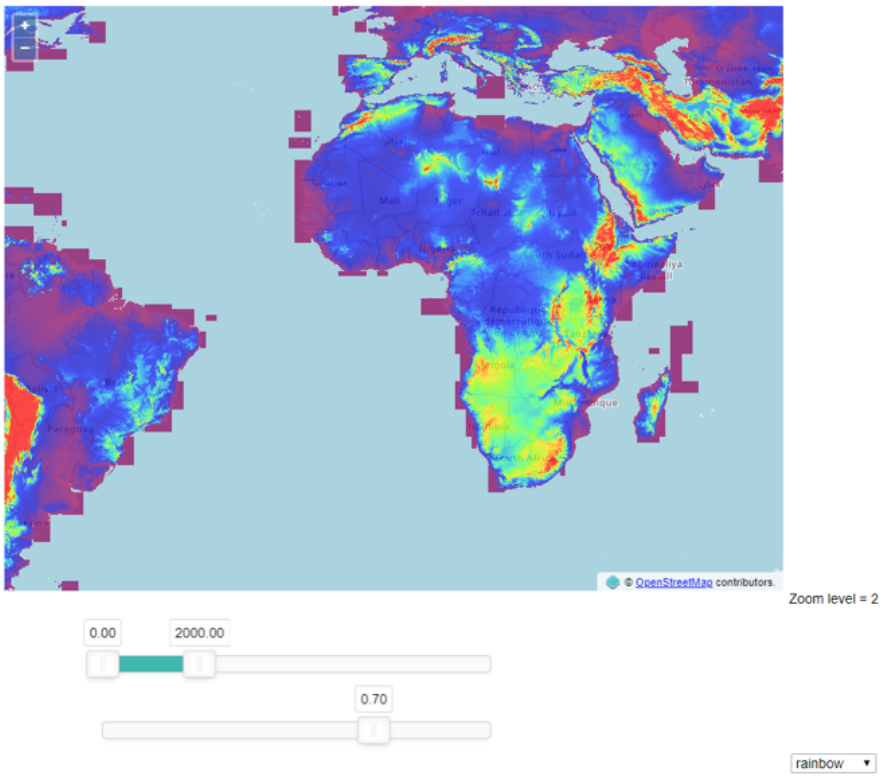

Figure 3.1: The map created by [?]

[1] FiXme Note: Write about Cloud optimized Geotiff somewhere
[2] FiXme Note: Write about gdal2tiles parallelly
[3] FiXme Note: Write about Sarahs stuff here

A picture of one of his maps can be seen in figure 3.1. This particular map is not included in his thesis, but it is the most similar to this project. The map shows an unspecified dataset colored in rainbow colors. Below the map are two sliders and a dropdown list with the label “rainbow”. Changing the value in this dropdown list allows the user to switch to another color scheme. The bottom slider is controlling the opacity for the raster layer. The upper slider controls which maximum and minimum values the coloring should be based on. How the map change, when changing the values in the upper slider can be seen in figure 3.2. When zooming in another more detailed layer gets loaded and rendered.

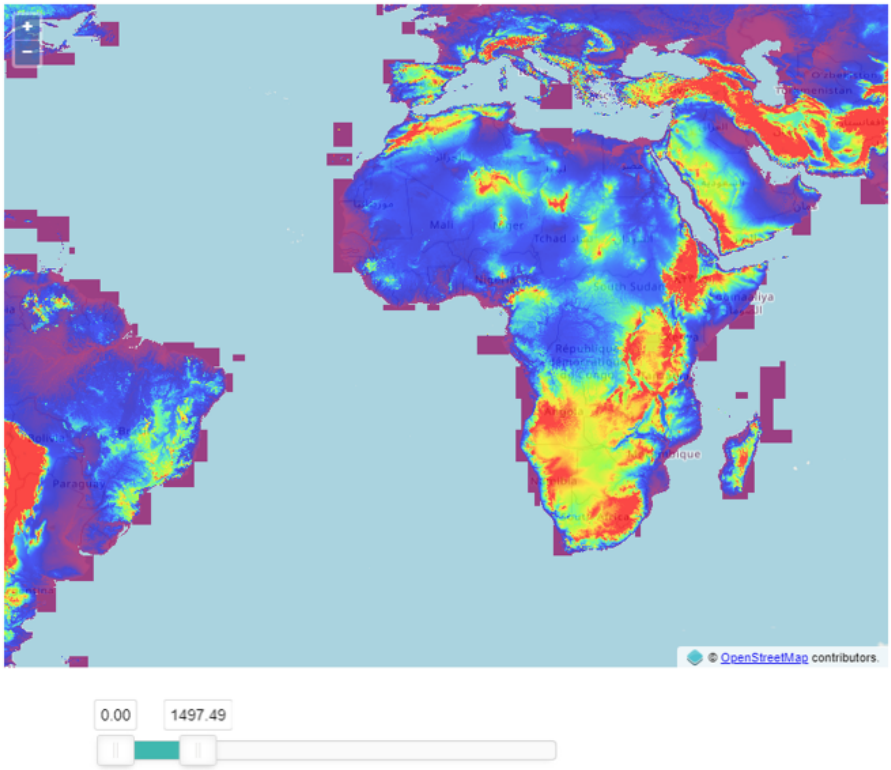

Figure 3.2: How the map change, when using the slider

4

When taking a closer look at the source code it can be seen that the raster is being loaded as tiff tiles with a bit depth of 16 bit. [**?**] Only tiles the currently are visible are being requested unless the tile already have been requested. [**?**]

When comparing with the core concepts for this project, Baumrocks map fulfil most of the criteria. It is locally visualizing only the necessary data. The core concept related to bit depth is relevant in the creation of the tiles, so it does not really apply to this. It is also to some extent allowing the user to color the layer based on the current extent. The user can adjust the maximum and minimum values but does not know what the maximum values are in the current extent. How the tool functions from a technical perspective is further detailed in section x.

[4] FiXme Note: Update figures to be created by Baumrock

3.2 Sections to be written here

Other ways of trying to visualizing large rasters:

- qgis
- makeCitywebsite?

[5]

[5] FiXme Note: Write about other methods

Part II

Methodology

3.3 Core concepts

Based on the initial research and tests a four core concepts were developed:

- Load only the needed data
- Coloring should be based on the current extent
- Tiles should not be precolored
- Coloring should be done locally
- File format should have same bit depth as input format

3.3.1 Load only the needed data

The case data presented in chapter x holds information for the entire world. However, loading data for the entire world would be unnecessarily time consuming. Initial experiments of rendering this quantity of data also revealed that the browser did not have enough memory to load that amount of data. Instead only the relevant data should be downloaded. The relevant data in this case would refer to the data, which the user is able to see. Getting a subset of a raster dataset can be accomplished by dividing it into smaller tiles. Section x will present two different ways this can be achieved.

3.3.2 Coloring should be based on the current extent

The initial exploration of the data showed that the coloring of the tiles should be based of the current extent. This is due to the how much the values are varying across the world. Figure 3.3 shows an example of the difference between a map where the coloring is based on all values or if it is limited to the current extent. Both maps illustrate the population in Denmark. The left map has its coloring based on values from the entire world, while the right has its colors based on the current extent. The left map appears empty since the population density in Denmark is negligible compared with the densest areas in the world. In the right map it is possible to see the location of the most populated cities. Since the left map provide the user with no relevant information and the right one does, the coloring should be based on the current extent.

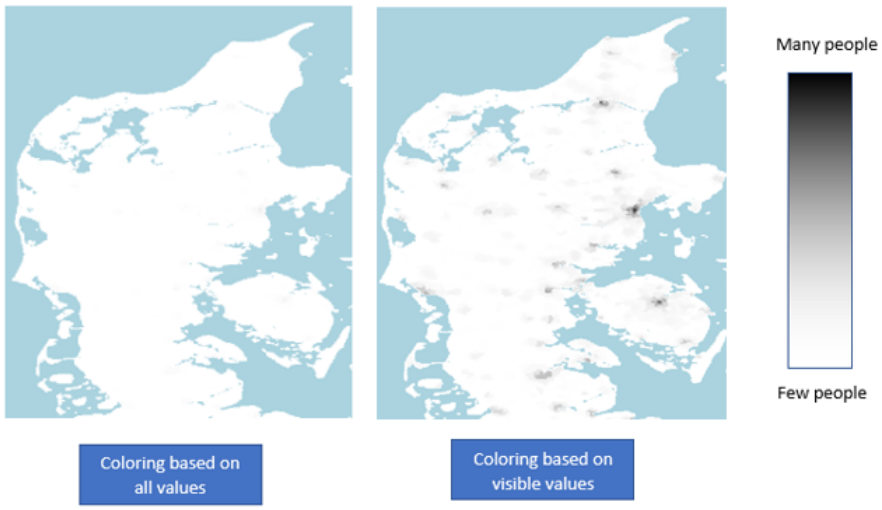

Figure 3.3: Population density in Jutland

3.3.3 Tiles should not be precolored

If the coloring should be based on the current extent and the map is interactive the coloring of the tiles should be done on the fly instead of once and for all. The reason for this is best explained with the example in figure 3.4. This figure is illustrating the population in a small subset of India. The three boxes are illustrating examples of possible map extents, which the user could get by zooming in on different parts of the map. Each of these three have a different max value in their cell with the highest population. The blue square is covering the city center of Indore a therefore have a high value than the green, which is covering the outskirts of the city. The last square does not include any part of a bigger city and therefore have a lower value. Since their max values are different, they would each need their own coloring. If the tiles were to be precolored it would be necessary, create three coloring of the tiles – one for each the extents. However, since the map is interactive, the extent is not limited to those three. If the user instead were to zoom in on an area between the red and blue square a new max value would maybe be found. When scaling this example up to the entire world and to multiple zoom extents there would be thousands of combinations. If all of these needed to be prerender it would both require lots of initial processing time and storage space for an immense amount of data.

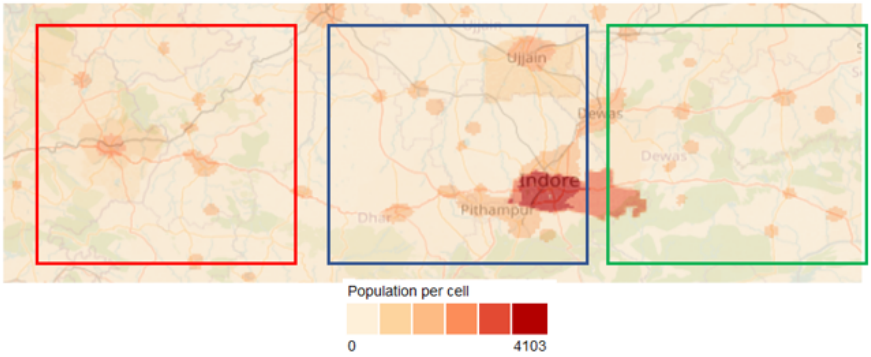

Figure 3.4: Population density. The three boxes are examples of possible extents an interacting user could get by zooming.

The alternative is to color the tiles on the fly. When the user would zoom to a new area the map, a script could register the highest currently visible value. This information could then be sent to recoloring script, which would color the tiles based on this. This way only tiles with the needed colors would be rendered.

3.3.4 Coloring should be done locally

The coloring can be done in two ways as illustrated in figure 3.5. It can be done by sending the information to a tileserver, which then would color the tiles and send them to the client. Alternatively, the coloring could be done on the client.

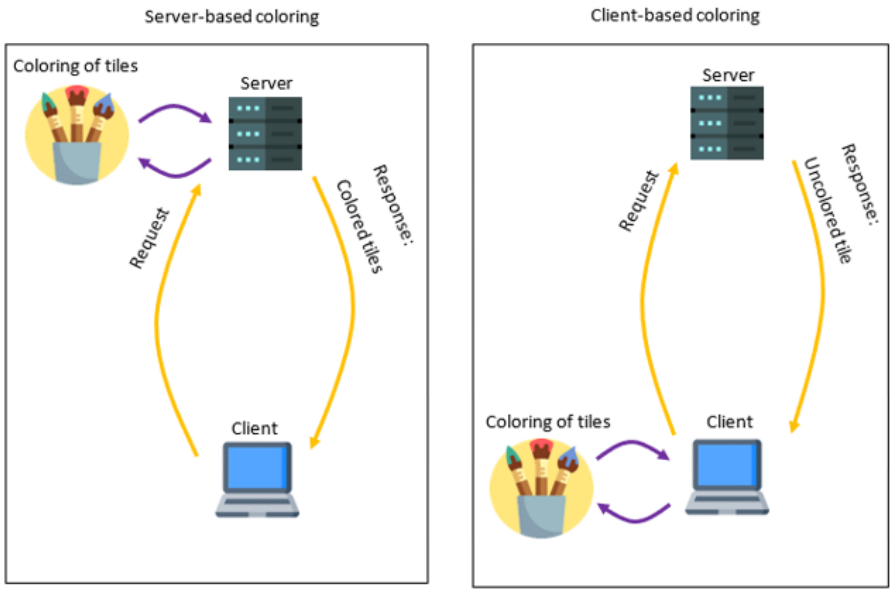

Figure 3.5: Difference between coloring the tile on the server and locally

Comparing the two options is best done with an example, which has been illustrated in figure 3.7. Here the blue and green box are two different possible views both containing 16 raster tiles. The two boxes have different max values, since blue one contains the city

center. In the example a user will pan from the blue view to the green one. If the coloring is done on the server side the initial 16 tiles in the blue box will be requested on load.Then when the user pan to the green view the 16 tiles in that box will be requested and colored. The four tiles that are shared between the blue and the green boxes must be requested again, since they need a new color due to the change in max value. This is not the case if the coloring is done locally. In this case only 12 tiles would have to be requested, when changing from the blue to the green view. Since the coloring is happening locally the coloring script can recolor the four tiles it already has downloaded from the initial load. In the right part of figure 3.5 the yellow loop would be run 12 times, whereas the purple one would be run 16 times. This should result in a faster experience for the user.

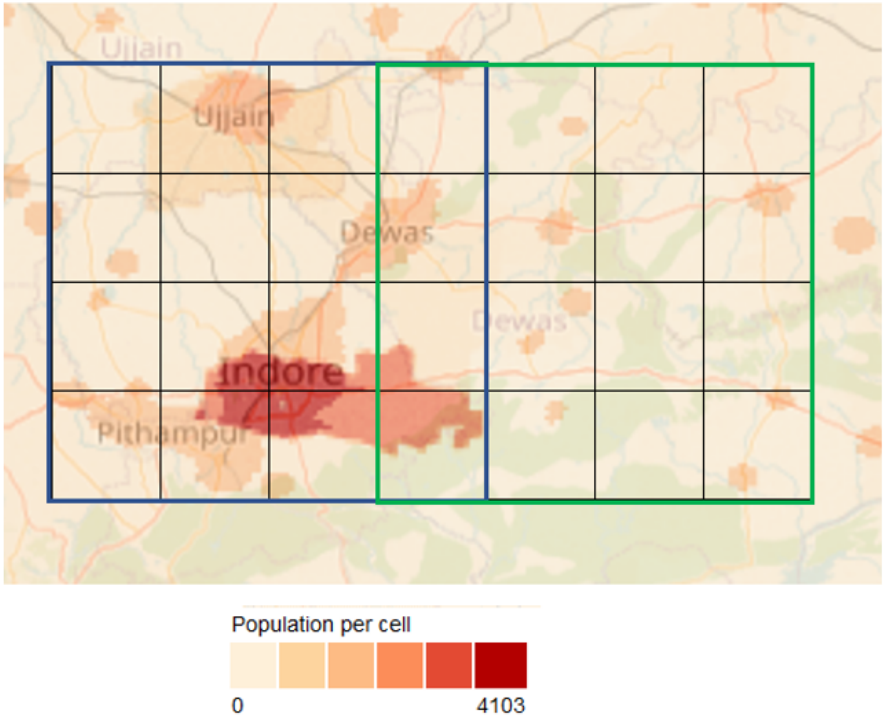

Figure 3.6: Example of why local coloring is needed. The squares are different extents used for the example

3.3.5 File format should have same bit depth as input format

As mention in section x the data from the case has a bit depth of 32 bit. To visualize the data the file format must not be changed to a format with less than 32 bits. Figure 3.7 is an example of how a subset of the case data look originally and when converted to an 8-bit format. As mentioned in section x the pixels in an 8-bit raster can have values between 0-255. This means that this format is unable to correctly display the original data range, where the data range is covering thousands of values. The original data get clamped into the 8-bit format producing wrong result https://gdal.org/programs/gdal2tiles.html .

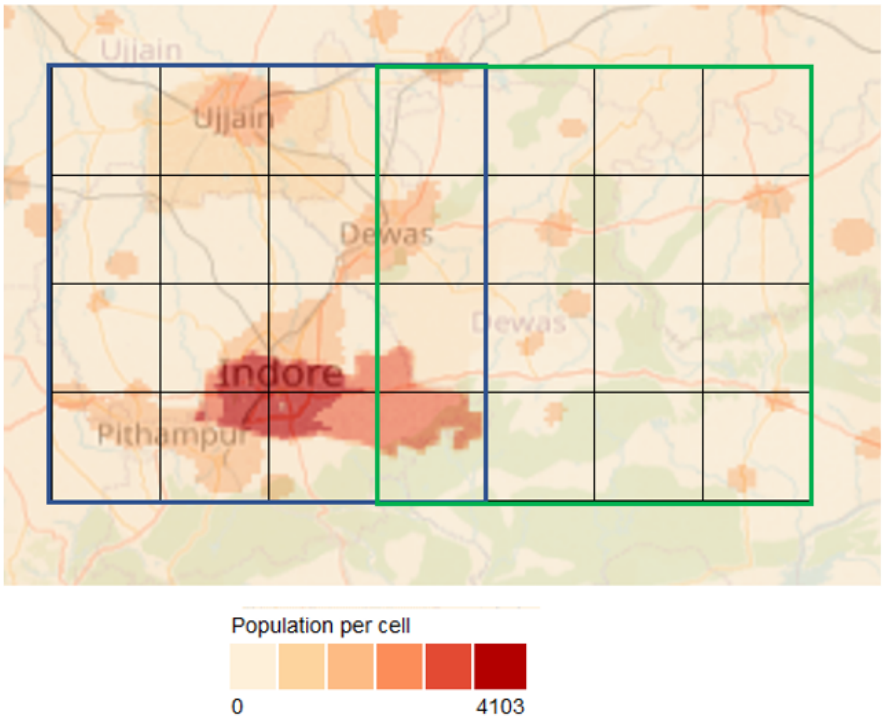

Figure 3.7: 32 bits cramped into 8 bits

Normally this issue could be worked around by rescaling the input data to the new format [?] . This rescaling would be a lossy compression resulting in loss of information since the original data cannot be expressed with values between 0-255. [?] For the normal use of tiles this would not be a problem since tiles would only be used for visualization. If the tiles are being recolored locally the tiles are being used for more than just visualization. It is necessary to access the data within the tile. This means that solutions resulting in a lossy compression are not an option. To ensure that the correct data reach the client the file format used for tiles must have a bit depth of 32.

6 7

[6] FiXme Note: Add interactive somewhere in the research question
[7] FiXme Note: Who is the target audience?

Creating an interactive map 4

When creating an interactive map one must consider which features to add to enhance the user experience. To give an overview of some of the basic features used for navigating and get information from a map an example of an interactive map has been used. www.openstreetmap.org has been used as an example of an interactive map. A picture of this map can be seen in figure x. Its User Interface (UI) have been numbered with boxes, which have been colored green if the feature would make sense for exploring and comparing large population dataset. [**?**]

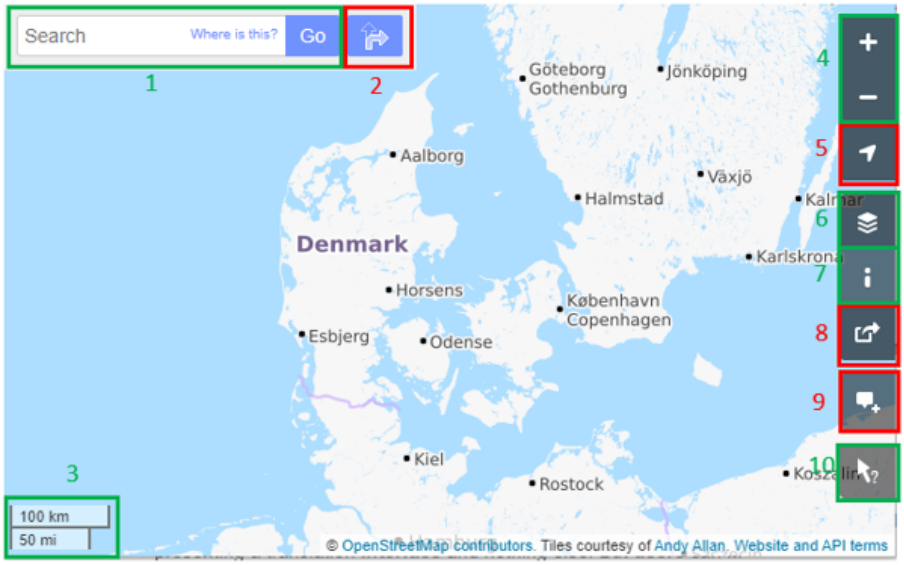

Figure 4.1: An example of an interactive map

What the different parts of the UI does can be seen in table x. The functions of the UI can be classified in five different categories. Using the map navigation UI the user can control which part of the map is being displayed. The real-life navigation tool is used for navigating in the real world. The map explanation category gives the user information about the map. This can be general information about the scale or symbols on the map or specific information about a clicked point. The tool for controlling the displayed data is used for controlling what is being visualized on the map. The last category, communication with others, is for sharing information with other users of the map. [1] [2]

[1] FiXme Note: Write that my own search bar can be improved
[2] FiXme Note: Did I write about toogling dual legends?

Number	Name	Category	Function
1	Search bar	Map navigation	Pans to the searched location
2	Route calculation	Real-life navigation	Calculate a route between two given points
3	Scale	Map explanation	Shows the scale of the map
4	Zoom	Map navigation	Allow the user to zoom in or out
5	Show my location	Map navigation	Show the location of the user and navigates to it
6	Layers	Control data display	Control which layer is being displayed
7	Map key	Map explanation	Displays the map legend explaining the meaning of the map keys
8	Share	Communication with others	Generate a link for what currently is displayed on the map or download an image for the map
9	Add note to the map	Communication with others	Add notes about mistakes, so that the map can be corrected
10	Query features	Map explanation	Give more information about a clicked map object

Table 4.1: Overview of the UI elements highlighted in figure 4.1

Evaluation tool 5

[1] Due to the limitations mentioned in section x user testing was not a possibility. This meant that it was necessary to find another way to evaluate the user experience. This was done using Google Lighthouse version 5.7. This chapter starts with an overview of the tool followed by an evaluation of which criteria are relevant for this project. The relevant criteria and their metrics are then expanded upon.

5.1 Overview

Google Lighthouse is an automated tool in Chrome DevTools, which is the developer tool included in Chrome. It can be used to check the quality of websites within the five categories: Best practice, performance, accessibility, search engine optimization and progressive web app. [**?**] [2]

Performance The performance audit is measuring the site performance and load speed. This is done by measuring the time needed for different stages of loading and when the user is able to interact with the site. [**?**]

Accessibility The accessibility check is for ensuring all users can effectively access and navigate the webpage. The check is mainly focused on Accessible Rich Internet Applications, which is used by assistive technologies such as screen readers.[**?**] It is pointed out that accessibility is difficult to automatically test, so further manual testing is recommended.

[**?**]

Best practice The best practices cover different types of website enhancements. There checks to ensure that the website is fast, secure, and not using deprecated technologies, among others. [**?**]

Search engine optimization The checks within the Search Engine Optimization (SEO) category evaluate how well the page is optimized for ranking by search engine. This optimization is achieved by ensuring that the page is readable search engines and that search engines can access the page. It also includes some measurements for being mobile friendly. [**?**]

Progressive web app The Progressive Web App (PWA) audits are target towards a mobile audience. These includes having a fast and reliable experience on mobile networks

[1] FiXme Note: Write limitations
[2] FiXme Note: Inset Image here

and to which degree the website act in a similar fashion to a native mobile application. [**?**]

5.1.1 Relevant evaluation criteria

Not all the audit categories in Google Lighthouse are relevant for this particular project. As mentioned in section **??** the tool is developed for local use on a computer. This means that it is not relevant how optimized a site is for the phone. Since it is used locally it can not be found by a search engine, which makes the SEO category irrelevant. The best practices for enhancing the websites speed are relevant, while the security measures are not, since the tool is not accessible to others on the internet. While accessibility is important it has not been prioritised for the development of this prototype. The main argument is that the tool is being developed for data scientist and not for anybody. It is presumed that the majority of people, who daily work with computers have the sensory capacity to easily do so. A high performance is vital to ensuring a responsive user experience, so the performance audit will also be included.

[3]

5.2 Metrics for performance

[4]

[5]

First Contentful Paint The First Contentful Paint (FCP) is the time in seconds it takes before the browser to render the first parts of the website.

https://web.dev/first-contentful-paint/ **Speed Index** The speed index is a time measurement of content appearing visually during load. To calculate it the visual progression between each frame of the loading process is measured. This is then being processing is done using the Speedline module. The unit is seconds. https://web.dev/speed-index/?utm$_source$ = $lighthouseutm_medium = devtoolshttps://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index$**FiXme**$Note : Knowaboutthisfortheexam, butanextendedexplanationseem$

It should be noted that FMP is being replaced with Largest Contentful Paint (LCP) in the next release of Chrome Lighthouse. The reason for this is that FMP is did not give consistent result and was difficult to standardize in all web browsers. https://web.dev/first-meaningful-paint/ LCP is the seconds needed to render the single largest content element, which is visible without scrolling.

https://web.dev/lcp/ **Time to Interactive** The Time to Interactive (TTI) is the seconds before a loaded website is completely interactive. This time is defined as after FCP when most of visible elements can be interacted with and respond within 50 milliseconds. https://web.dev/interactive/ **First CPU Idle** First CPU Idle is the seconds before a

[3] FiXme Note: Write target audience
[4] FiXme Note: Write about timing the python processing
[5] FiXme Note: Source: Return to this one later

page becomes minimally interactive. Minimally interactive is defined as when the majority of the User Interface elements are interactive and giving responds within a reasonable time. The difference between this and TTI is the degree of interaction. When the user can begin interacting with a page the First CPU Idle can be measured. TTI is measured when all interactions can be performed. This feature is being replaced with Total Blocking Time (TBT) in the next Lighthouse release. The reason for this is that this and TTI are too similar to maintain both. https://web.dev/first-cpu-idle/ TBT is the total amount of milliseconds where the website is not responding to user input. It is measured during the time between FCP and TTI.

Part III

Development of the tool

In this chapter it is explained how the tool has been developed. The first section is describing the coding languages used and the plugins used to create the tool. The following sections are describing the necessary steps to build the tool. These steps can be seen in figure 5.1.

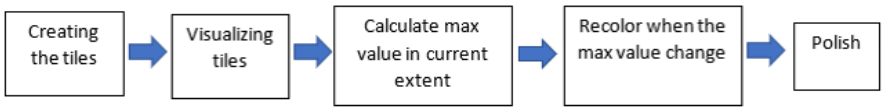

Figure 5.1: An overview of the code

First it is necessary to divide the raster into tiles to be able to limit the loaded data to the current extent. How this is done is explained in section x. Section x describes how the tiles have been visualized. To be able to color the tiles based on the current values it is required to know what the current values are. In section x it is explained how this information is calculated. The raster layer is then rerendered with these values as described in section x. Lastly some measurements were taken to ensure a more user-friendly experience. These additions are described in section x.

Coding languages and plugins 6

For this project, the coding language Python was used to create the tiles, while visualization of the tiles was done in a webgis build with the languages HTML, CSS and Javascript. This webgis have been tested on a local caddy server for reasons explained in section x

6.1 Languages

Python

Python is programming language with a simple syntax, which functions across multiple different platforms. This simple syntax means that Python can achieve the same as some other coding languages in fewer lines.[**?**] Python was used in this project because of the gdal library expanded further upon in section x This project has been using version 3.6 of Python.

HTML

HTML is short for Hyper Text Markup Language. It is the language used for defining and structuring a web page's content.

CSS

CSS is an abbreviation for Cascading Style Sheets. This language defines how the HTML will be displayed.

Javascript

How a webpage behaves is defined by the language Javascript. This is what makes the web page interactive. [**?**]

6.2 Libraries

To transform the raster data into smaller tiles the python library Gdal is used.

Gdal

GDAL is library for translating between multiple different geospatial data formats. [**?**]

Included in this library is the gdal2tiles program, which can divide raster files into smaller tiles. At the time of using this program it was only able to generate tiles structured after the TMS standard. The function to follow the XYZ structure was added the 3th of May 2020 [**?**] [**?**] The script rendering the tiles in the map was based on the XYZ structure. This meant that the rows of tiles were ordered incorrectly when the generated tiles were imported. Therefore, the official version of gdal2tiles was replaced by a version made by a github user named commenthol. This version is modified to allow the creation of tiles following the XYZ structure. [**?**] Both the official version and the modified version have their output format as mbtiles with a bit depth of 8 bit. To be usable for this project a bit depth of at least 32 bit is needed, as mentioned in section x. The workaround for this issue will be explained in section x.

Openlayers

The map in which the tiles are being showed are created in Openlayers, which is an open source JavaScript library for creating dynamic maps for web pages. [**?**] Openlayers was chosen because the tool presented in related work was built in Openlayers. Therefore, using Openlayers would enable expanding upon this existing tool instead of starting from scratch.

olGeoTiff

olGeoTiff is a Javascript class for visualising geotiff tiles in Openlayers, utilising the libraries geotiff.js and Plotty. The visualized tiles are being processing in the client instead of on a server. It was used in the map presented in section x. The class is a modified version of Openlayers WMTS layer, where the internal tile loading function has been changed. The regular function would request precolored tiles and then add them to the map. The tiles requested by the modified version are not precolored and need to be processed before getting added to the map. A simplified illustration of this processing is illustrated in figure x. This simplified figure is enough to explain the mechanics of the class but does not detail the callback function structure. Aside from being used for error handling callbacks are also necessary to ensure that Openlayers do not try to add the tiles to the map, before they have been processed. More detailed figures can be found in [**?**] thesis.

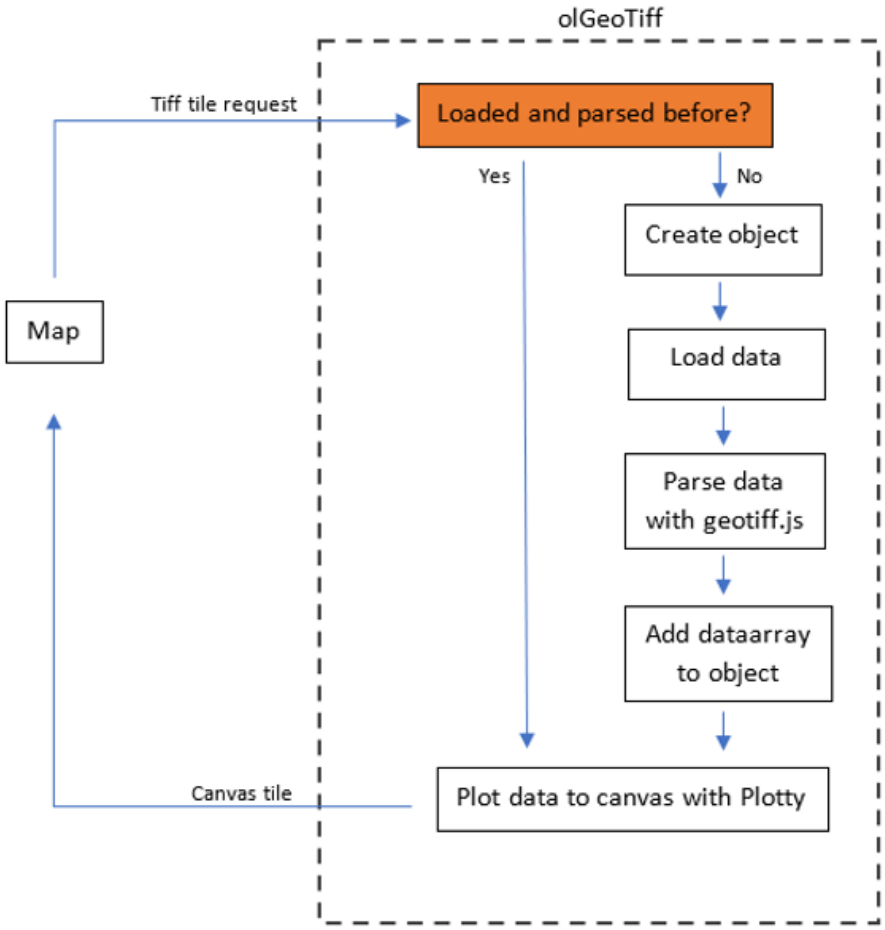

Figure 6.1: A simplified illustration of how olGEoTiff functions

To ensure that tiles are only being downloaded once an object keeps track of all the downloaded data. The object is organized by the tiles' url. Whenever tiles are being requested the object will always be checked to see if it already contains the url. If it does not the object will be updated to include the requested tile. The tile will then be loaded before being processed. [**?**] The processing is done with the TIFF parser geotiff.js [**?**] and Plotty, which is a library for creating images from data arrays [**?**].

The loaded tiles first get parsed with geotiff.js and added to the object before Plotty get used to render tiles in the designated colors. Then the tiles get added to the map. [**?**] The olGeoTiff also have a redraw function, which when triggered will redraw the tile layer based on the current designated colors. This was for instance used in Bernnard Baumrocks map, whenever the color sliders were changed.

Developing the tool 7

This chapter details how the visualization tool was built.

7.1 Creating the tiles

The tiles got created using a modified version of gdal2tiles, which was changed to create tiles following the XYZ format. However, since the default output bit depth is too low, it would have to be modified further. The file format would also have to be changed to tiff in order to be visualized with the geotiff.js library.

7.1.1 Changing the file type

Changing the file format can be done by changing two lines in the gdal2tiles script:

```
#Original code
#self.tiledriver = 'PNG'
#self.tileext = 'png'
#New code
self.tiledriver = 'GTiff'
self.tileext = 'tiff'
```

Listing 7.1: Changing the file format

This changes the raster driver from png to the geotiff format and the file extension from png to tiff. [**?**] The geospatial information, which is the difference between a geotiff and a regular tiff, gets lost in process. This information is not important for this project since the tiles are being loaded based on their name and folder placement, not based on the internal metadata.

Running gdal2tiles with these changes will produce a tiff file, which still would be limited to 8 bits.

7.1.2 Increasing the bit depth

The reason for the bit limit is that gdal2tiles uses the memory dataset driver, which have 8 bits as default. This default can overwritten to 32 bits by adding "gdal.GDT_Int32" to every instance where the driver is being used as demonstrated in code x. [**?**]

```
self.mem_drv = gdal.GetDriverByName('MEM')
 ...
```

```
#Old code
#dstile = mem_drv.Create('', tile_size, tile_size, tilebands)
#New code
dstile = self.mem_drv.Create('', self.tilesize, self.tilesize, tilebands, gdal.GDT_Int32)
```

Listing 7.2: Increasing the bit depth

The memory driver is being used four times, which all have been changed in a similar fashion.

The script is now generating tiff tiles correctly. However it also generates KML files for visualization in Google Earth, which is undesired since it would increase the processing time and required storage space.

7.1.3 Prevent generation of Google Earth files

gdal2tiles is automatically generating KML files if the projection is EPSG:4326. According to the documentation for the official gdal2tiles this can be disabled with the command "–no.kml". [**?**]

This command does not prevent the creation of the files in the modified version. Therefore the automatic generation has been removed from the code. This was done by commenting out the lines shown in x.

```
if self.out_srs and srs4326.ExportToProj4() \
        == self.out_srs.ExportToProj4():
    self.kml = True
    self.isepsg4326 = True
    if self.options.verbose:
        print('KML autotest OK!')
```

Listing 7.3: Increasing the bit depth

1

7.1.4 Command for tile creation

After all the modification are in place the script can be run by running the command illustrated below:

python <path to modified gdal2tiles script> –leaflet –zoom=<desired zoom levels> –profile=raster –webviewer=none <input file> <output directory>

Most of these inputs are from the documentation for the original gdal2tiles, though the leaflet command is from the modified version. This command ensures that tiles are being created following the XYZ formatting instead of the TMS. The zoom command defines which zoom levels should be generated. An example of an input could be 0-2, which would generate tiles for the zoom levels 0, 1 and 2.

[1] FiXme Note: Write about the addition to time the whole thing

Setting the profile to raster was done because the two other options (mercator and geodetic) resulted in the error message "list index out of range", while the raster profile gave a useable product. The selection of this options and its consequences is expanded upon in section x.

The webviewer option prevent the generation of webviewers to visualize the tiles. These default webviewers is created to visualize tiles of the original format, so they would be unable to visualize the modified tiles. The input file is the name of the processed file and output directory is the folder, where the tiles get created.

7.1.5 Result

While creating the tiles the error message "ERROR 1: Buffer too small" gets displayed. The tiles are still being generated with the correct formatting and bit depth. This is an error message from numpy, which gdal2tiles are using. [**?**] The generated tiles appears as they should, so this error message is potentially referring to the geospatial information, which gets lost.

The tiles at the rightmost and bottom edges have an edge as shown in figure x. This edge is not being shown, when the tile is being loaded into the map. A comparison between the input file and the tile shows that all data is being displayed. The bottom part of the input file is the last part before the edge. This edge seems to be generated because the input file was not perfectly dividable with the tile size as illustrated in figure x. https://github.com/ARiisgaard/Thesis/issues/22 This edge could be another explaination for the error message.

2

3

The script also produces an xml file with metadata. An example of the content of said metadata file can be seen in code x.

```
<?xml version="1.0" encoding="UTF-8"?>
<TileMap tilemapservice="http://tms.osgeo.org/1.0.0" version="1.0.0"><Abstract/><SRS>GEOGCS["WGS
    84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],
AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,
AUTHORITY["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS["Longitude",EAST],AUTHORITY["EPSG","4326"]]
</SRS><BoundingBox maxy="25.00000000000814" maxx="80.99999999999989" miny="19.00000000000815"
    minx="72.99999999999989"/><Origin y="19.00000000000815" x="72.99999999999989"/><TileFormat
    extension="tiff" mime-type="image/tiff" height="256" width="256"/><TileSets
    profile="raster"><TileSet order="2" units-per-pixel="0.00833333333333" href="2"/><TileSet
    order="3" units-per-pixel="0.00416666666667" href="3"/><TileSet order="4"
    units-per-pixel="0.00208333333333" href="4"/><TileSet order="5"
    units-per-pixel="0.00104166666667" href="5"/><TileSet order="6"
    units-per-pixel="0.00052083333333" href="6"/><TileSet order="7"
    units-per-pixel="0.00026041666667" href="7"/><TileSet order="8"
    units-per-pixel="0.00013020833333" href="8"/><TileSet order="9"
    units-per-pixel="0.00006510416667" href="9"/></TileSets></TileMap>
```

Listing 7.4: The metadata from the xml file generated by the modified gdal2tiles

[2] FiXme Note: Make figures - remove link
[3] FiXme Note: Test if normal gdal2tiles tiles have edges - if false, then point it out here

4

Running it parallelly it parallelly

7.2 Visualizing tiles

After the tiles got created, they are stored in a folder, which gets uploaded to the testserver along the index file. The metadata from the xml file must be loaded into the map to be able to visualize the tiles. Normally this could be done using the WMTSCapabilities() function in Openlayers. [**?**] However, the formatting of the xml file produced by gdal is different from the format, which this function can read. Therefore, a small script has been created to parse the xml file and store the information in a metadata object. This object, tileMetadata, stores the bounding box, origin, center coordinates as well as tilesize. The initial version of the object also stored the resolution data, which is called units-per-pixel in code x. This led to some inconsistency when loading tiles, which had been generated without some of the lower zoom level. This will be further expanded upon in section x. Therefore, the resolution was instead generated using the script x, where 0.0333 is the value for units-per-pixel at zoom level 0.

```
1 for (var z = 0; z < 14; ++z) {
2 // generate resolutions and matrixIds arrays for this WMTS
3 //The number in the resolution calculation is the units-per-pixel value at zoomlayer 0 in the xml
      file generated by gdal2tiles
4 resolutions[z] = 0.03333333333514 / Math.pow(2, z);
5 matrixIds[z] = z;
6 }
```

Listing 7.5: The JavaScript in the project

Using this metadata, the tiles could be visualized using the olGeoTiff class.

7.2.1 Custom colors scheme

Using colorbrewer a custom sequential colorscheme was generated. This scheme was added to Plotty and selected as a color palette. *This code might change a bit, so wait with writing

7.3 Loading data at a wrong resolution

Figure x shows how the map looked, when loaded with a manually defined max value.

[4] FiXme Note: How long time did it take to run?

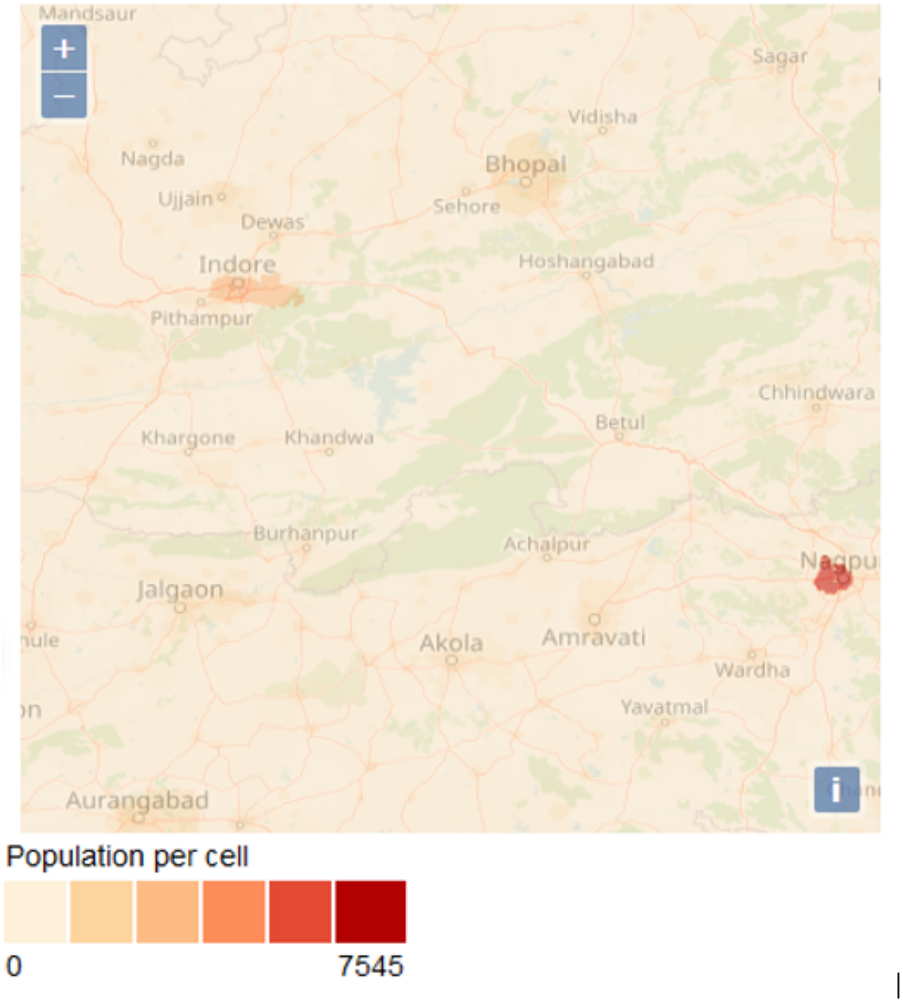

Figure 7.1: The map looking right, but with tiles from the wrong zoom level

While the map appeared to look alright, it was loading tiles from the wrong zoom level. The tiles always got loaded from a zoom level 3 lower than intended. So the map in figure x is visualizing the map at zoom level 7 but loading and displaying tiles from zoom level 4.

Some experiments with loading the tiles from the correct zoom level Setting up Openlayers to download tile from the current view and correct zoom level chrashed the map with the error message “Insufficient Resources”. The amount of loaded tiles seems unnecessarily high and size of the tiles too small. This seems to indicate, that the tiles, which the modified version of gdal2tiles associates with zoom level 7 in fact belongs to a higher zoom level.

The reason for this bug was never discovered and the bug never got fixed. When the resolution data was gathered from the metadata file, the difference between the loaded and the actual zoom level would between different sets of tiles. This variance would depend on the which zoom levels were not being generated. So, if only the zoom levels 2-7 were generated, the difference would increase by one for each missing layer. In this case the loaded tiles would instead be wrong by 5, calculated as the default wrongness of 3 plus 2 for missing zoom layer 0 and 1. This bug will be defining for the rest of the code.

7.4 Calculate max value in current extent

Calculating the highest value in the current extent can be divided into two smaller tasks. Figuring out which tiles currently are being displayed and processing these tiles.

7.4.1 Current displayed tiles

The tiles, which currently is within the view, can be found using the Openlayers tileGrid method forEachTileCoord. This method can trigger a function for each tile within a given zoom level and extent. [**?**] The method is going through the tiles based on their coordinates, but this can be translated to the tile urls using the getTileUrlFunction.

```
var tileUrlFunction = wmslayer.getSource().getTileUrlFunction()
var zoomlevelAdjustment = 3
wmslayer.getSource().getTileGrid().forEachTileCoord(loadExtent, mapZoom - zoomlevelAdjustment,
    function(tileCoord) {
       tileName = tileUrlFunction(tileCoord, ol.proj.get('EPSG:4326'))
       %Find max value
}
```

Listing 7.6: The JavaScript in the project

forEachTileCoord is triggering for each tile in the given zoom extent. This means that it on zoom level 7 it would load the tiles, that should be rendered on zoom level 7. Due to the bug mentioned section 7.3 it is not the tiles from that zoom level, which are being displayed. Instead the tiles from zoom level 2 are being displayed. This will complicate some of the next steps. The adjustment to the zoom level is in order to load the tiles, which should have displayed.

7.4.2 Max value for each tile

If the tiles added to the map had been from the correct zoom level calculating max value of a tile could have been done as olGeoTiff were running. olGeoTiff holds the values for the tiles in a dataarray already, so finding the maximum value could be done with a single line adding a tiles max value to the dictionary for that tile. This line is shown in code x, where urlToTiff is the name of the object, which holds the data. maxValueTileData[url].maxValue = Math.max(...urlToTiff[url][0]) However, since olGeoTiff holds data from an incorrect zoom layer this would not function. A possible solution would be to trigger forEachTileCoord for the same wrong zoom layer as the displayed tiles are from. This solution was not implemented since it would result in a map with could potentially so misleading information. The explanation for this has been illustrated in figure x.

Todo: Figure with showing how large the loaded tiles are compared with how large they should be.

The tiles from the wrong zoom level are so large, that they would show data outside the view of the map. This means that the coloring could be based on the information that was outside of the current view.

The alternative solution is to run another function going through the tiles, which should have been displayed and find the highest value among these. This would result in a map where the wrong tiles were being colored based on information from the smaller correct tiles. This solution is by no means ideal. It means requesting tiles from two layers, one for displaying on the map a one for calculating the relevant max values. Loading more data than necessary will make the script slower, but since no better solution was not found this have been implemented. The calculation of the maximum value for each tile was done in a very similar fashion to how olGeoTiff operates as illustrated in figure 7.2. An object holds information about all tiles, which have been processed and the max value is known. When the function is run for a tile it first checks if the tile already has been processed. If that is not the case, then it will create an object for the data, which it then will load and parse with geotiff.js. The data array with parsed data will then be run through as descripted in code x to calculate the maximum value. This maximum value will then be returned to the map.

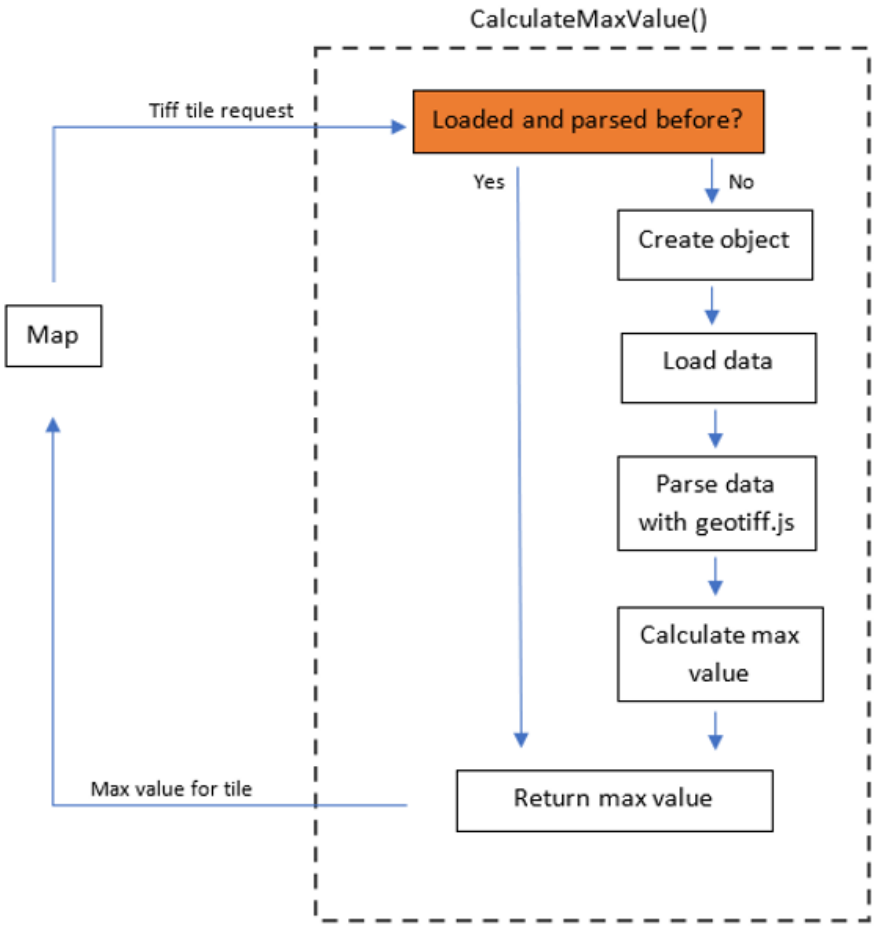

Figure 7.2: Flow diagram of the calculation over max value in a tile

7.4.3 Highest tile value the current displayed

To find the largest tile value among all the currently displayed tiles forEachTileCoord as mentioned earlier is used. forEachTileCoord does not have a trigger for when it has run through all the tiles. This functionality is necessary for ensuring that the produced maximum value is actually the maximum value. Without a precise end trigger the coloring applied to the map could be based on the biggest value found before the coloring script stated running instead of the absolute highest value in the current display. Since this trigger is necessary it has been created by running forEachTileCoord another time to count the number of tiles. This part of the code can be seen in code x.

```
1 var tileNumber = 0;
2 wmslayerMap1.getSource().getTileGrid().forEachTileCoord(loadExtent, mapZoom - zoomlevelAdjustment,
      function(tileCoord) {
3       tileNumber++;
4 })
```

Listing 7.7: The JavaScript in the project

The total number of tiles in the current extent can then be used as a trigger. The recoloring of the map can then be delayed until the function for calculating the maximum value have been running for the same amount of times as there are tiles on the map. In practical terms this can be accomplished by adding a counter and an if statement to the maximum-value-function. The counter would check how many times the function has run. The if statement would check if the amount of time the function had been run was equal to the number of tiles. If this is the case and the registered max value is different from the previous one the recolor function would trigger. The calculation of the maximum value is the function presented in figure 7.3. It is running asynchronously because the array otherwise would be filled with "undefined" values instead of actual values. Running it asynchronously ensures that the script awaits the calculation of the value.

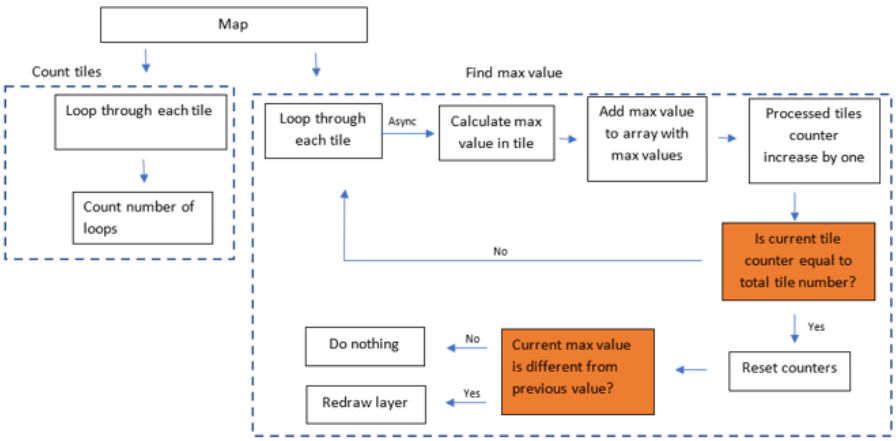

Figure 7.3: Finding the largest value in all currently displayed tiles

7.5 Recolor when the max value change

The recoloring function was already part of olGeoTiff. However, in Bernhard Baumrocks' thesis this recoloring was triggered manually by the user when changing the color sliders. In this project the changing of colors will trigger automatically. The function for recoloring the map have therefore been set up to trigger on the user stopping this changing the view. By not triggering before the map movement is finished a smoother user experience is ensured, since new tiles does not have to processed before the user is finished with interacting with the map. The recolor function is running through all of the code mentioned in the previous sections.

```
1 map.on("moveend", function() {
2 recolorMap()
3 });
4 });
```

Listing 7.8: The JavaScript in the project

7.6 Polish

In addition to the rendering of the raster in the map, some features were also added to improve the user experience.

7.6.1 Two maps

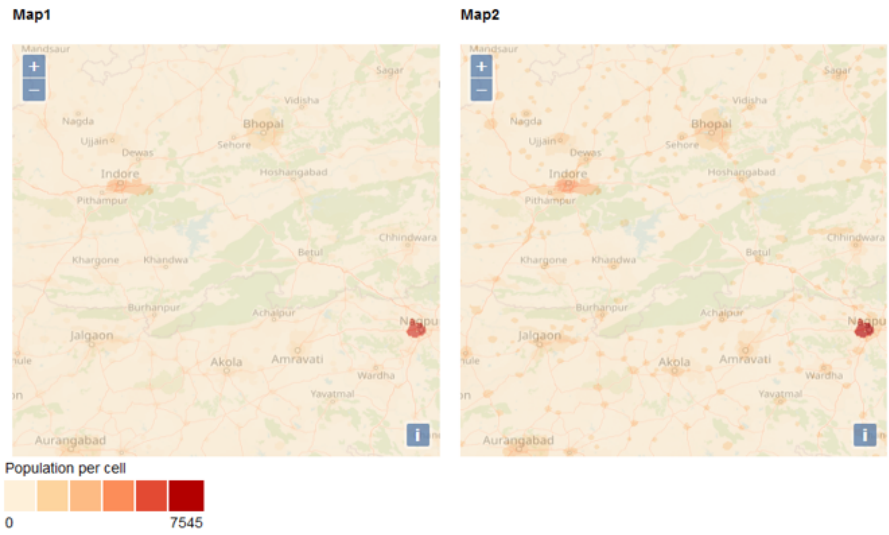

Figure 7.4: A second map

To be able to compare different rasters with eachother a second map was added as illustrated in figure 7.4. This second map is having its own raster dataset but sharing the view with the first map. This means that the two maps would always show the same area. Panning

or zooming in one map would do the same action in the other map. The two maps are sharing the same legend.

TODO: Write about evaluating max value based on two maps

7.6.2 Search function

In order to be able to faster navigate the map a search function was created as shown in figure 7.5.

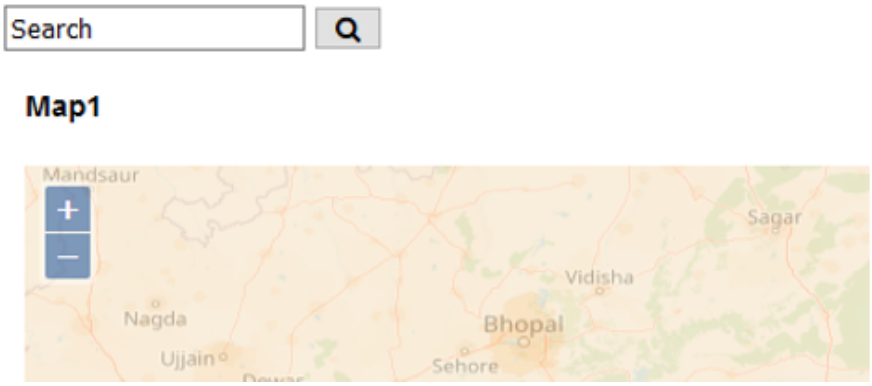

Figure 7.5: A searchbar

The user can use this to change the current view of the maps to a given location. This is accomplished through the help of Nominatim. This tool can search through Openstreetmap data by location names. It then returns data about the searched location.

Among this data is the latitude and longitude for the central point of the place. [**?**] This coordinate is then used as the coordinate for the center of the map.

Result 8

here is a map of the end result and writting about what it can do

[**?**]

Part IV

Evaluation and discussion

Evaluation 9

This chapter is the evaluation of the script based on the Google Lighthouse audit. This chapter will highlight some elements of the audit. The full audit can be seen in appendix x. The given score in the relevant categories can be seen in figure x. There was a bug in the measurement of the performance score, which will be further expanded upon in section x. This means that the tool. It does however still provide some useful suggestions as to how the site could be improved.

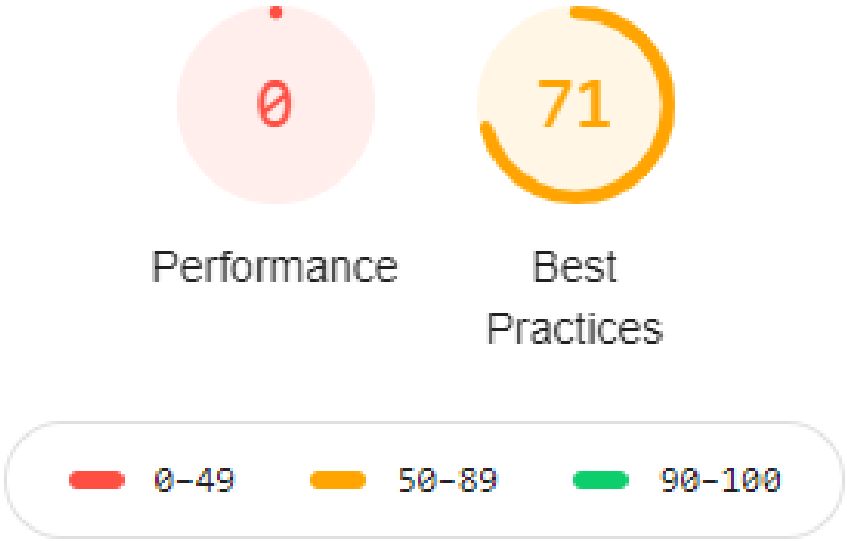

Figure 9.1: The score for performance and best pratices from Google Lighthouse

9.0.1 Performance

As shown the performance scored nothing out of a hundred. The performance in each of the metrics can be seen in figure x.

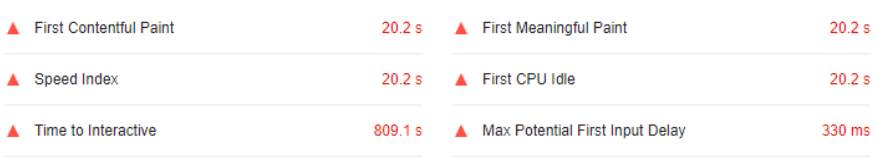

Figure 9.2: The result of the performance audit

[1]

9.0.2 Opportunities

In addition to providing the metrics for the performance Lighthouse also comes with suggestion to how the loading time can be reduced. **Eliminate render-blocking resources** The opportunity for the largest estimated time saving is to eliminate render-blocking resources. These are the URLs, which must be loaded before the first paint can be applied to the page. [**?**]

Table x shows the different libraries, which have to be loaded before the site is rendered. It can be seen that Openlayers is 82 % of the loaded data. [2]

Plugin name	Size	Unused bytes	
Openlayers	2784 KB	1188 KB	42.7 %
Geotiff.js	172 KB	100 KB	58.1 %
jQuery	87 KB	61 KB	70.7 %
Plotty	26 KB	1 KB	5.3 %
olGeoTiff	7 KB	0.1 KB	1.5 %

Table 9.1: Size of loaded modules and the much they are used

When analysed further with the Coverage tool it can be seen that a large part of the Openlayers library is not being used as show in table 9.1. [3]

[4]

Minification and compressing Files sizes and script parsing time can be reduced by minifying and compressing the files.

A minified file is a file, where all the unnecessary parts have been cut away. Whitespaces and unused code are removed leaving a smaller file, which still functions perfectly.

Compression is using an algorithm to modify the data, so that it takes up less space. [**?**] [5]
Avoid enormous network payloads Long loading times are highly correlated with the amount of loaded data. [**?**]

Loading the raster tiles for the map does require loading multiple files with a large bit depth, which requires a lengthy loading time.

[1] FiXme Note: Write that it is nonsense
[2] FiXme Note: Have I written about Jquery??
[3] FiXme Note: TODO: Make a table from the information in the figure below – tabtext – resourse use of the 3 larges libraries
[4] FiXme Note: Write about Coverage earlier – maybe in evaluation tools
[5] FiXme Note: For exam: Try using Terser to minify and compress – Write about Terser in future work

Does not use HTTP/2 for all of its resources This suggested improvement appears if some of the page's resources are being served with a version of HTTP/1. According to Google Audit all loaded resources are being delivered using HTTP 1.1.

6

[**?**]

Uses deprecated APIs When loading the metadata about the tiles a synchronous XMLRequest is used. This is a deprecated API, which will be removed from Chrome in a feature edition. [**?**]

[6] FiXme Note: Write about HTTP/2 earlier?

9.1 Discussion

Discussion 10

10.1 Further development

Not all the intended functionality got added to the map. This section will list some of the features, which there was not time to implement.

10.1.1 Data at point of click

A function could be added to get the population counts from a clicked point for both maps. An example of this have been shown in figure x. When the user clicks the map an infobox informs the user about the value at the clicked point for both maps. TODO: Figure showing a popup on click - Popup text: Data in this map and data in the other

10.1.2 Parallel generation of tiles

As mentioned in section x the generation of tiles were done without using multiple processers, which meant that it became a time-consuming process. With the official gdal2tiles being able to generate tiles following the XYZ standard it is possible that using this would allow for parallel generation of tiles. This is probably the most important missing feature, since the processing time otherwise would be so high, that it would not be a faster alternative to the currently available options.

10.1.3 Changing layers

Another functionality, which could have been added was the option to change the datasets of the maps. As mentioned in section x the case data contains population projection for ten different years. It does therefore make sense to have the option display more than the two datasets. As illustrated in figure x data from other years could be added as separate layers. This would allow the user to compare different datasets without having to reload the webgis with different datasets.

This way the map could also be used to visualize the different in the same projection from one year and another.

10.1.4 Option for multiple colorschemes

If there is a vast difference between the values in the two shown projection coloring based on the same maximum values does not necessarily make sense as was shown in the figure in core concept 2. It would therefore make sense to have the option to enable separate coloring as shown in figure x. TODO: make a figure showing the map with multiple legends and color schemes.

10.2 Performance enhancements

Based on the performance and best practice audits by Google Lighthouse the performance could be enhanced in multiple ways.

Eliminate render-blocking resources As mentioned in section x the largest render-blocking resource was Openlayers. The same section also highlighted that 43.2 % of the library never get used. The performance could therefore be improved by only loading the necessary parts of the library. [1] [**?**]

Avoid enormous network payloads The size of data loads will be reduced by only having to load one set of tiles with the solution mentioned in section x.

[2] - Tenser

[3] [4]

Does not use HTTP/2 for all of its resources Caddy was used to serve the resources with HTTP/2. However even after setting up Caddy the error message was still present. It is uncertain if this means that the connection still is HTTP/1 based or if it is HTTP/2 but labelled incorrectly. Figure x is two screenshots from the network tab of chromes developer tool when the page is served with a python server (top) and caddy server (bottom). The grey bars are when a loaded file is being stalled. The blue bars are when they are loaded. This figure shows that the serving is being optimized, where the stalling largely is gone. This could indicate that the server no longer has the HTTP/1 limit of only having 6 TCP connections. [5]

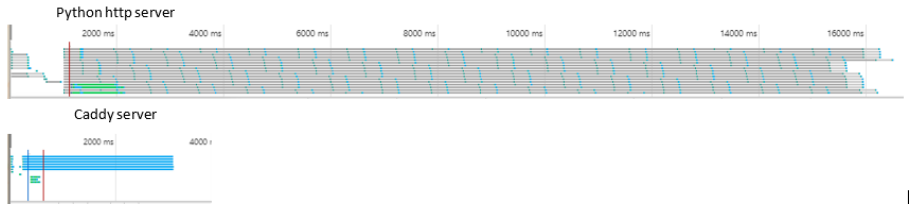

Figure 10.1: Screenshot of the network connection for a python and a caddy server

[1] FiXme Note: Check with Carsten – can this only be done with node?
[2] FiXme Note: Write about asynchronous XMLREquest and minifying/compressing
[3] FiXme Note: Write about zoom level bug
[4] FiXme Note: Add user testing to future work
[5] FiXme Note: Source!!!

6

10.3 Is this even a good measurement for performance?

[6] FiXme Note: Mention caddy – 6 connection issue earlier

11 Conclusion

Bibliography

This is at test A

A.1 Or is it?

List of Corrections