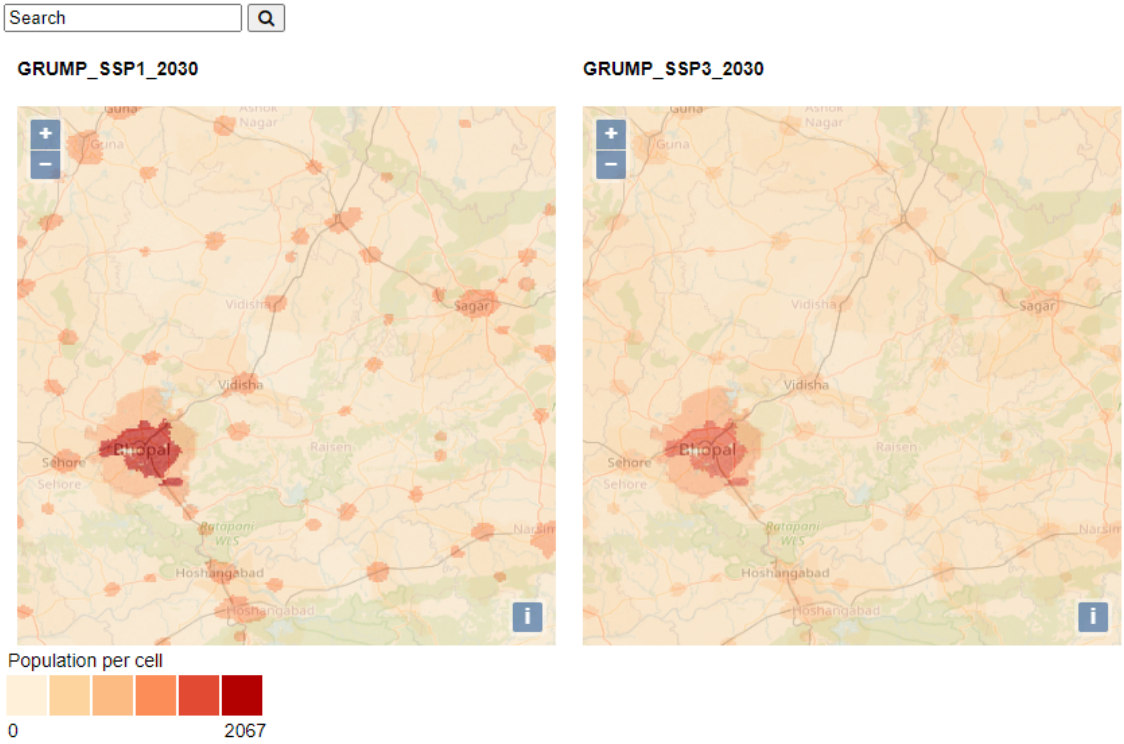
Aalborg University
Geoinformatics



Visualizing and comparing population projection rasters

Author:
Andreas Gram Riisgaard

Supervisor:
Carsten Keßler

June 2019

Titel:

Visualizing and comparing population projection rasters

Projekt:

Thesis project

Project Period:

February 2020 - June 2020

Author:

Andreas Gram Riisgaard

Supervisor:

Carsten Kessler

Number of pages: 69
Number of annexes: 2
Afsluttet: 4-6-2020

Abstract:

In this thesis project an interactive tool for visual comparison of raster datasets have been developed using population projections as a case. To develop a tool able to enable such comparisons it is important to know how population projections should be visualized and which functionalities are important for the tool. There is a technical challenge in visualizing large raster datasets, while still maintaining a responsive user experience.
The conventions for visualization of population projections was explored through a literature review. A quantitative sequential dataset like a population projection should be colored, so that the areas with least population is colored in a lighter color, than the more densely populated areas.
The functionalities for the tool was determined by comparing with another interactive map. It was decided to have two maps showing different population projections. The maps can be navigated either by panning and zooming or using a search bar.
To ensure a responsive user experience the raster was not loaded into the tool in its entirety. Instead it was divided into smaller tiles, which got loaded based on the extent of the map. These tiles were then colored on the client.
The tool was created as an Openlayers map displaying tiles, which was created with a modified version of the python program gdal2tiles. While the user experience is responsive while using the map the creation of tiles is time consuming. The tool could therefore be improved in the future by using cloud optimized geotiffs instead of tiles.

Preface

Resume

I dette kandidatprojekt er et interaktivt værktøj til sammenligning af raster dataset blevet udviklet med befolkningsfremskrivelser som case.

Udvikling af sådan et værktøj kræver indgående viden om standarder for visualisering af befolkningsfremskrivninger. Det er også centralt at vide hvilke funktioner, der er relevante for sådan et sammenligningsværktøj. Der vil også være nogle tekniske udfordringer ved at visualisere store dataset uden lange loade tider for brugeren.

Visualiseringsstandarder for befolkningsfremskrivninger er blevet undersøgt gennem et litteraturstudie. Et kvantitativt sekventiel dataset, som befolkningsfremskrivninger, bør farves, så de tyndest befolkede områder er farvelagt med lysere farver end de tætbefolkede områder.

Værktøjets funktioner blev udvalgt ved at sammenligne med et nuværende interaktivt kort. Det blev bestemt at vise forskellige befolkningsfremskrivelser i to kort ved siden af hinanden. Der kan navigeres i disse to kort enten ved at panorere og zoome eller ved hjælp af en søg funktion.

Den tidligere omtalte tekniske udfordring blev omgået ved ikke at loade hele datasættet. I stedet blev det opdelt i mindre tiles, hvor kun de tiles, der kunne ses på kortet blev hentet ind i værktøjet. Disse tiles blev løbende farvelagt ved klienten.

Visualiseringsværktøjet blev bygget i Openlayers, mens tiles blev lavet af en modificeret version af python programmet gdal2tiles. Selve brugeroplevelsen var hurtig, men produktionen af tiles var tidskrævende. Dette ville måske i fremtiden kunne undgås ved at anvende Cloud Optimized Geotiffs i stedet for tiles.

Acknowledgements

This Geoinformatics thesis project have been created between February 2020 and June 2020 by Andreas Gram Riisgaard. A couple of people helped making this thesis a reality and they deserve recognition. First of among them is my supervisor, Carsten Keßler, who had the initial idea for the thesis topic. He also provided guidance during the research, code development and thesis writing. Secondly my parents, who continuously have been encouraging and supporting, deserve a heartfelt thank you. Lastly, I want to thank my dormmates Anna Clara Bay Lundqvist and Miriam Bressaglia, who helped with creating a productive local study environment in a time, where Covid-19 disrupted my regular study environment.

Contents

1 Introduction

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. [United Nations Department of Economic and Social Affairs, 2019b]

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 [United Nations Department of Economic and Social Affairs, 2019a], it is more ideal with a more nuanced picture, since most planning is based on local or regional scale spatial projections. [Jones and O'Neill, 2016]

Other researchers have used simulations to distribute the population within each country as raster layers [Keßler and Marcotullio, 2017]. 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 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?

1.2 Limitations

Determining relevant functionalities and the responsiveness would ideally have been done with user testing. However it was determined that both the creation of the tool and a scientific approach to user testing would require too much time. Therefore the tool creation got prioritised and other evaluation methods not involving users were chosen. This is expanded upon in section 12.

1.3 Target audience

The target audience for this project is academic researchers. It is meant as a tool for these researchers to be able to quickly compare different population projections. This prototype of the tool has only been created for internal use by single individuals. It will therefore not be created with multiple users in mind and there will be no considerations for security. The tool is also created with only computers in mind, so it will not be optimized for smartphone users.

1.4 Report structure

Figure 1.1: Overview of how the research question gets answered in the first part of the report.

The report has been divided into three parts; design, development and evaluation.

In the design part the thought process behind the design of the solution is explained. The first part is focused on answering the subquestions as illustrated in figure 1.1.

It starts with some background information about the case data in chapter 2, raster formatting in chapter 3 and chapter 4 about how the raster data currently is being compared.

The first subquestion is answered through a literature review in chapter 5. After the visual conventions have been explained an example of an interactive map gets analysed in chapter 6. This is done to understand which functionalities are important for such a map. This is followed by a review of related work in chapter 7. Based on the related work and an initial exploration of the data five technical concepts for the tool are created. These are described in chapter 8. All of the considerations presented in this part then get collected into the design in chapter 9.

The second part is the development of the tool in chapter 10. In the beginning of that chapter there is a further description of the structure of this part.

The last part starts with a presentation of evaluation methods in chapter 11 followed by an evaluation of the developed tool in chapter 12. This is followed by chapter 13 with a discussion of the tool and how it could be developed further. The last chapter is the conclusion in chapter 14.

Part I

Design

Case data: Population projection 2

The population projection visualized in this project have been created by Keßler and Marcotullio [2017] using a geosimulation, which geographically distribute a predicted population. This geosimulation has been run every tenth year from 2010 to 2100 and for different scenarios for the future.

2.1 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 cells, each of which contains the number of estimated people. [Keßler and Marcotullio, 2017]

The geosimulation has been run with two different definitions for the urban extent. The first one is also created by GRUMP. This urban extent has been estimated based on the light emitted after dark. It is known to overestimates the extent of urban areas. [Keßler and Marcotullio, 2017] The second dataset was provided by Global Land Cover Map from the European Space Agency (GlobCover), where areas classified as "artificial surfaces and associated urban areas" in the GlobCover data were treated as urban areas. [Keßler and Marcotullio, 2017]

2.2 The simulation

Using this dataset to define the extent of urban and rural areas, the simulation was run as illustrated in figure 2.1.

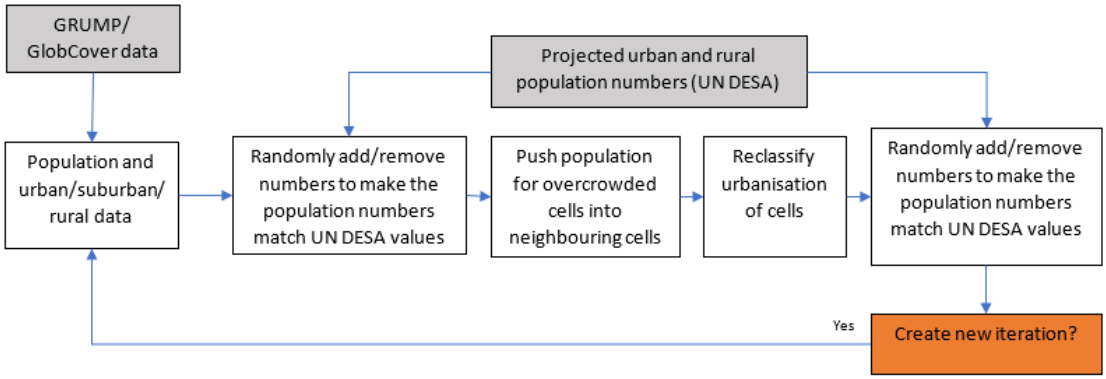


Figure 2.1: A flowdiagram of simulation creating the population projections

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. [Keßler and Marcotullio, 2017]

2.3 Shared Socioeconomic Pathways

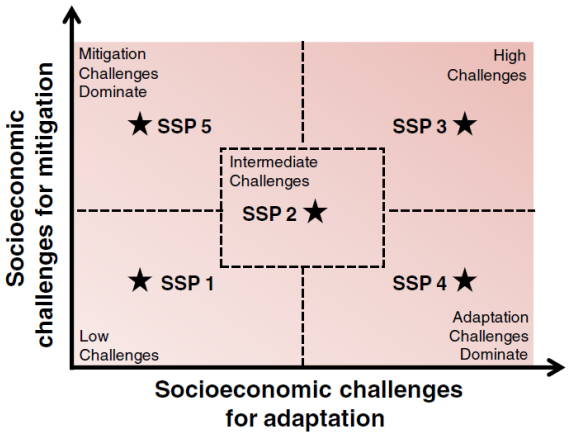


Figure 2.2: Five different future scenarios based on how clima change could affect society. Source: O'Neill et al. [2013]

The geosimulation has 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 2.2 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 systems. 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 includes climate hazards (temperature and precipitation changes, sea level rise and extreme weather phenomena) and what and whom these hazards will affect. [O'Neill et al., 2013] The vertical axis is factors, which in absence of climate policy would increase the amount of emissions of greenhouse gasses and factors reducing the society's capacity to reduce 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. [Ebi et al., 2014]

	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

Table 2.1: Population growth and urbanization for each of the Shared Socioeconomic Pathways (SSP). Source: Jones and O'Neill [2016]

How the population growth and the urbanization levels will be in the different scenarios can be seen in table 2.1. The population growth values are based on current income and fertility condition. The urbanization predictions are based on the current income. The Rich low fertility group has been defined as members of the Organisation for Economic Co-operation and Development. [Jones and O'Neill, 2016]

2.4 Technical details about the case data

Table 2.2 is highlighting some technical specifications of the dataset, which will be important for the development. The technical details have been acquired using GDAL's raster information program, gdalinfo. GDAL is further described in section 10.2.

Some of these concepts will be explained in upcoming sections, but the NoData value will be explained here. A raster file has values for every pixel. If no data is available for a pixel,

Technical information		
Attribute	Value	Further explanation
NoData value	-2147483648	
Bit depth	32	Section 3.1
Projection	EPSG: 4326	Section 5.2
Projection unit	Degree	

Table 2.2: Technical attributes of the case data

then the pixel gets assigned the NoData value. [ESRI, No date]

If the value was set to 0 it would be impossible to determine if an area had nobody living in it or if no data was available. That is the reason for the NoData value is set to a negative value, since the population density never can be negative.

Raster formatting 3

Visualizing rasters requires an understanding of how they work from a technical perspective.

3.1 Bit depth

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 to 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 $2^{number\ of\ bits}$. As an example, a bit depth of 8 bits would then result in $2^8 = 256$ different potential colors, since this 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 perceive 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. Since there are more than 10 million colors, images with this bit depth can therefore 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. [Dent et al., 2009]

3.2 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 get places next to each other, 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 with a higher spatial resolution. As illustrated in figure 3.1 each of these zoom layers have more tiles. [Liedman, No date] 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}. [OpenStreetMap Wiki, 2020]

Figure 3.1: Illustration of the increase in tiles for each zoom level

Openstreetmaps is an example of a service, which use tiled rasters for their maps. 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. [OpenStreetMap Wiki, 2020]

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 approaches to naming the rows. This difference has been illustrated in figure 3.2. The reason for the difference is that TMS tiles number their rows from south northwards, whereas XYZ are numbering rows the reverse way. [OpenStreetMap Wiki, 2019]

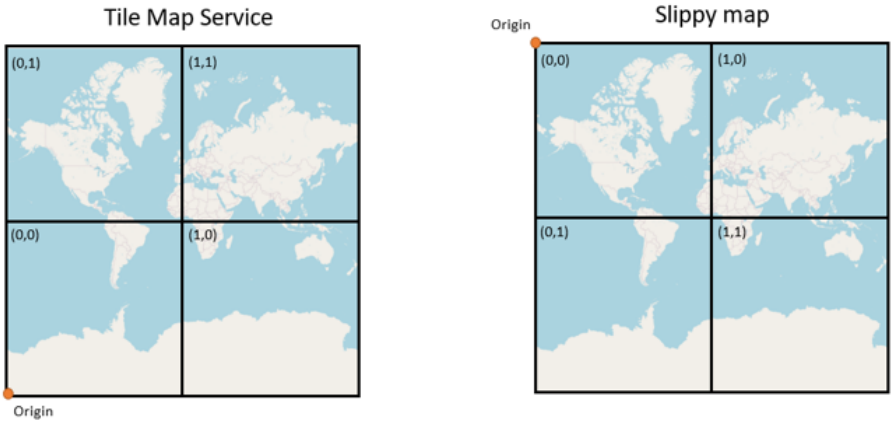


Figure 3.2: Illustration of the difference between the TMS and XYZ formats. The individual maps are inspired by similar maps from Planet [No date] and OSGeo Wiki [2012]

Current methods of comparing rasters 4

When trying to solve an issue it is important to understand what the current methods for achieving the same is. This chapter will present three current methods for comparing rasters.

4.1 Desktop Gis program

One way of visualising the population data would be to load the data into a desktop GIS programs like QGIS or ArcGIS. Both of these can visualise TIFF files and it is possible to visually compare different raster layers by changing back and fourth between the different layers. However one would have to manually style the layer to make spatial patterns become visible. Without styling the layer the tiff file will be colorized based on the most and least populated areas in the world. When coloring based on these extremes most of the world will appear in the same color as illustrated in figure 4.1. Only the most populated areas in India and China will appear in a different color. This is further expanded upon in section 8.2.

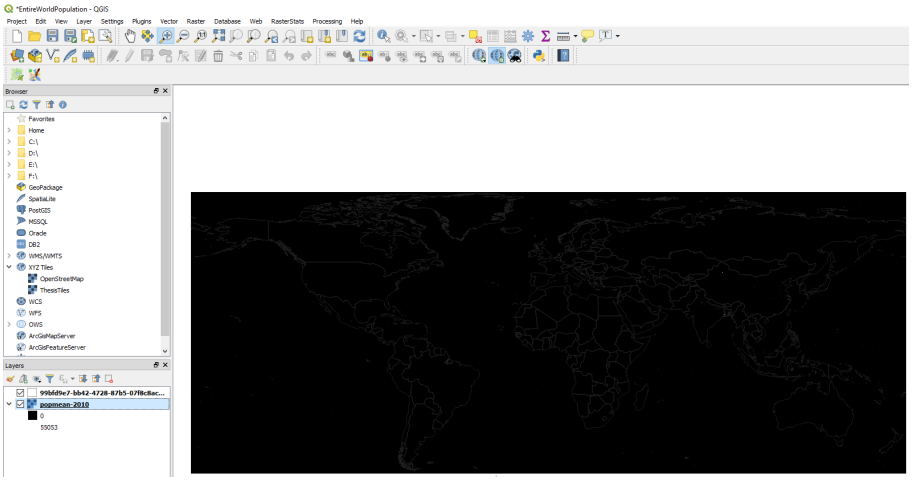


Figure 4.1: The population distribution visualised in QGIS and colored based on data from the entire world. The map appears to be colored in a single color due to the vast difference between the most densely populated areas and the rest of the world.

4.2 Interactive visualization tool for population simulations

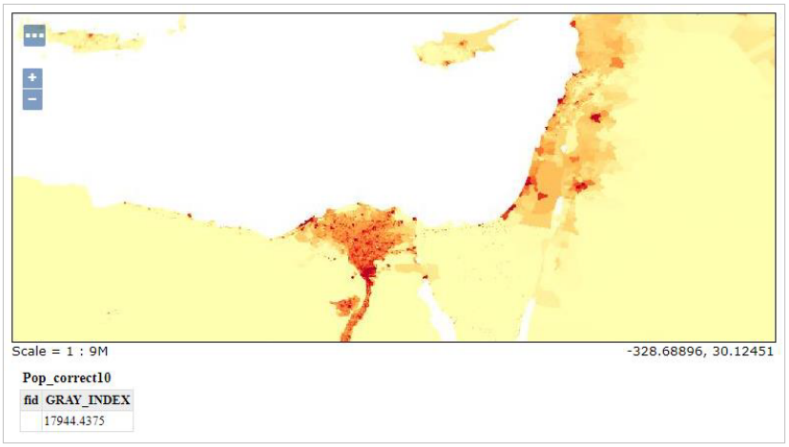


Figure 4.2: An interactive visualization tool made in GeoServer showing the population near Cairo. Source: Lafaire [2018]

Lafaire [2018] wrote a her master thesis about interactive visualisation of the same population projections, which is the case data for this project. In her thesis she tests different methods for interactive visualisation. The first tests are the python libraries Bokeh and Folium. Both of these are creating interactive webbased maps, but were unable to visualize the population projects due to the file size of the rasters. Therefore other solutions were created using servers for geographic imagery. The servers GeoServer and MapServer were used in her thesis, which both uses Openlayers for their interactive interface. Figure 4.2 shows the Geoserver visualisation of the raster data. Figure 4.3 shows the Mapserver solution. Both tools required a manual styling.

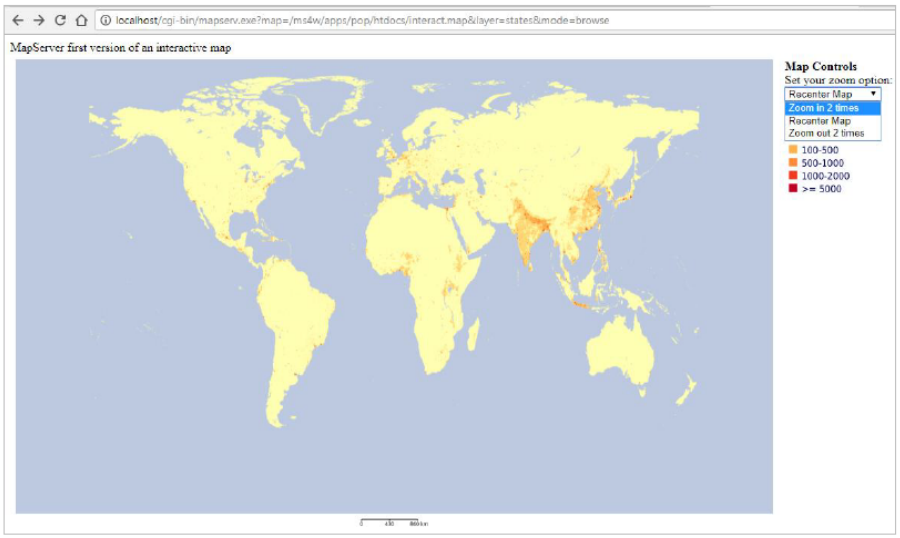


Figure 4.3: An interactive visualization tool made in MapServer. Source: Lafaire [2018]

The conclusion in her thesis was that MapServer was the faster option, whereas GeoServer was easier to learn. Using either server were slower than viewing in QGIS. Both had limited

customizability. As shown in the two figures the tool was only showing a single map and not having an interface for changing the raster layers. The tools were therefore more of a visualization tool, than a comparison tool. [Lafaire, 2018]

4.3 MakeCityWebsite

One of the creators of the case data created a python script, called MakeCityWebsite, for comparing the different population projections for a chosen city. The final product of the tool can be seen in figure 4.4 and 4.5.

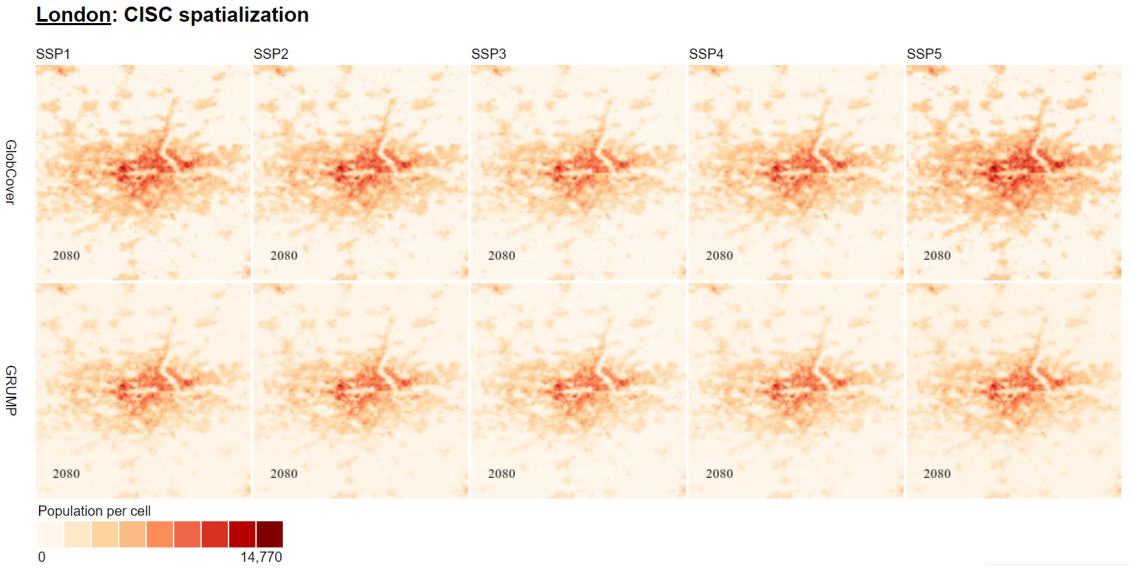


Figure 4.4: The output of running the MakeCityWebsite script for London. Each population density map is a gif based on different combinations of urbanization model and scenario. The other pictures in the gif are other years. Produced with a script by Keßler [2019a].

The input for the tool is the rasters from each of the two urbanization models, the five SSPs and every tenth year from 2010-2100, so a total of hundred different rasters. The tool first create new raster files, which have been cut to the extent of the chosen city. Then the minimum and maximum value in those file are calculated. The maximum and minimum values among all the rasters are then used to determine a color scale. This color scale is then used to create a new set of colored rasters.

So at this point the tool have generated one hundred rasters, which all have been colored using the same color scale. For the user to be able to comprehend this amount of data two additional steps are taken. First the individual combinations of urbanization models and SSPs are collected in gifs. These gifs display how the population evolve in their scenario by chronology changing between different years. These different gifs are then placed next to each other in a website as illustrated in figure 4.4.

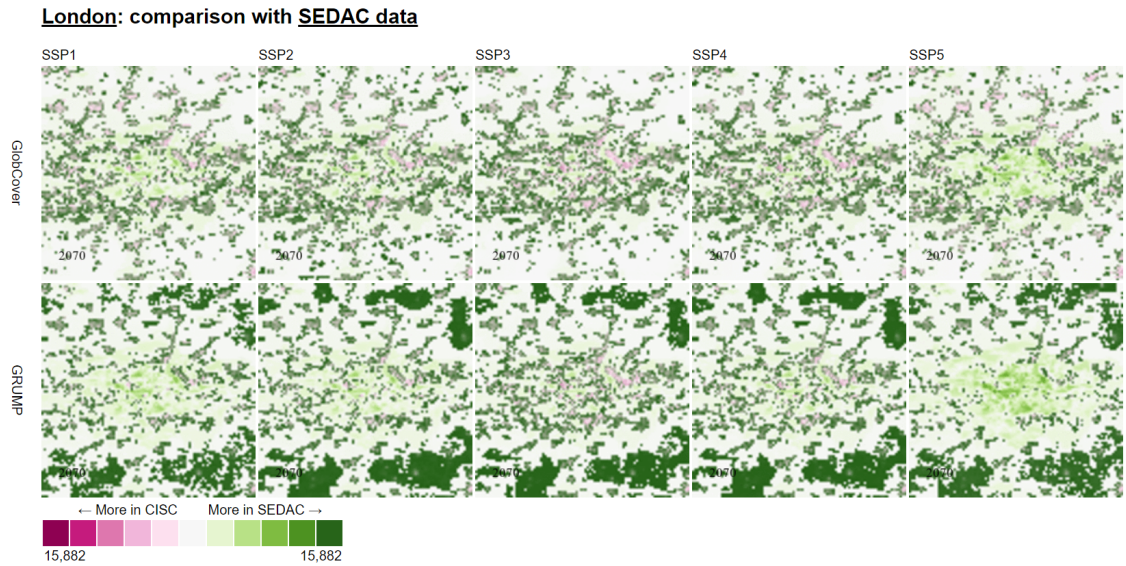


Figure 4.5: MakeCityWebsite also creates a comparison with another population projection called SEDAC. Each map shows the difference in values between the two population projection for each combination of urbanization model and scenario. Produced with a script by Keßler [2019a]

The tool also creates a comparison with another population projection called SEDAC. This is done by going though the same steps above as above, but with a custom raster. This raster visualises the difference between the two projection. This comparison part of the website can be seen in figure 4.5

The creation of these comparison rasters takes 20 minutes for a single SSP [Keßler, 2019b].

Visual conventions 5

This chapter will address the visual conventions connected to population projection.

5.1 Color

A color can be defined by three parameters: hue, saturation and lightness. These different concept have all been illustrated in figure 5.1.

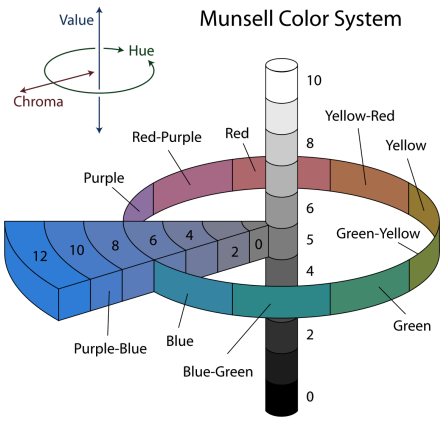


Figure 5.1: Illustration of the concepts: hue, saturation and lightness. Source: Rus [2007]

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 color's 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.

[Dent et al., 2009]

5.1.1 Selecting a color scheme

There are multiple elements, that should be considered, when choosing a color scheme. The right choice of color pattern allows the user to see patterns in complex data, which otherwise would be obscured.

Brewer divides color schemes into the four categories; binary, qualitative, diverging and sequential. These categories and combinations between them have been visualized in figure 5.2. [Brewer, 1994]

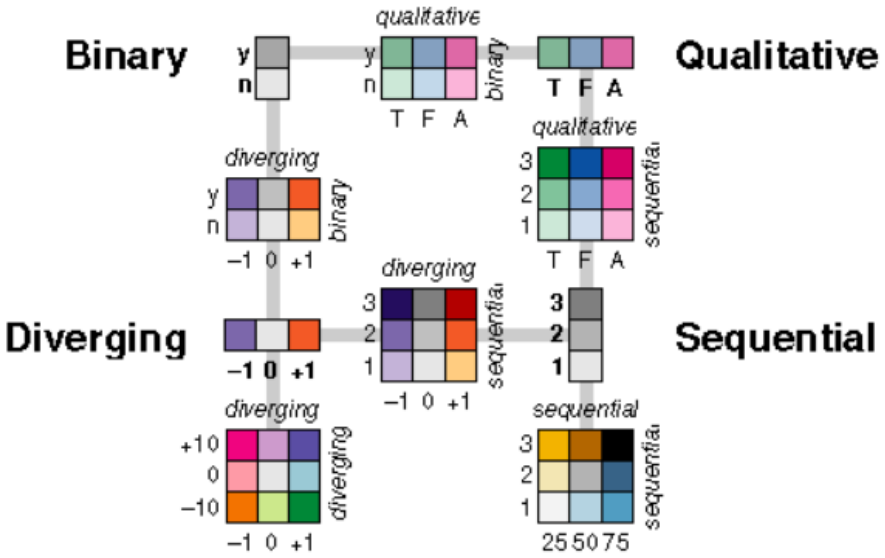


Figure 5.2: Illustration of the different categories of color schemes. Source: Brewer [No date]

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, such as population density. 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. [Brewer, 1994]

Based on these categories of color schemes Brewer has developed an online tool, colorbrewer2.org, which can aid mapmakers in picking a color scheme. This tool also

takes colorblindness into consideration and informs the user if a colorscheme is suitable for printing or viewing on small screens. [Brewer et al., 2013]

5.1.2 Number of data classes

After selecting a color scale, it is important to select how many different colors should be displayed on the map. Having a high number of data classes produces a detailed map, since with more classes there is less data generalization. There can also be too many different data classes to the point where it becomes difficult to tell them apart. How many classes to use depends on the spatial distribution of the data. Maps with irregular spatial patterns should have five to seven different classes, whereas regularly patterned maps can have more. The reason for this is that when similar colors are placed side by side it is easier to distinguish between them.[Brewer et al., 2013]

5.1.3 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 conventions – 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)” [Dent et al., 2009]. There are however convention for the choice of lightness, where "light is less - dark is more" [Garlandini and Fabrikant, 2009].

5.2 Map projection

When creating a map, the three-dimensional earth gets transformed into a two-dimensional map, which leads to distortion. The severity of these distortions depends on the scale. For large areas such as subcontinental or continental, the distortion has bigger consequences compared to the smaller scale. When mapping at large scale, it is impossible to prevent distortion to some of the projection’s properties. This means that area, angles, distance, or direction on the map will be distorted. Which properties get affected depends on the choice of projection. [Dent et al., 2009]

Components of an interactive map 6

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 interactive map www.openstreetmap.org has been used as an example. [OpenStreetMap, 2020] A picture of this map can be seen in figure 6.1. Its User Interfaces (UI) have been numbered with boxes. These boxes have been colored based on how useful the function would be for exploring and comparing large population dataset. The green are essential, the yellow are useful and the red are not relevant. The reasoning behind the coloring is explained in section 6.1.

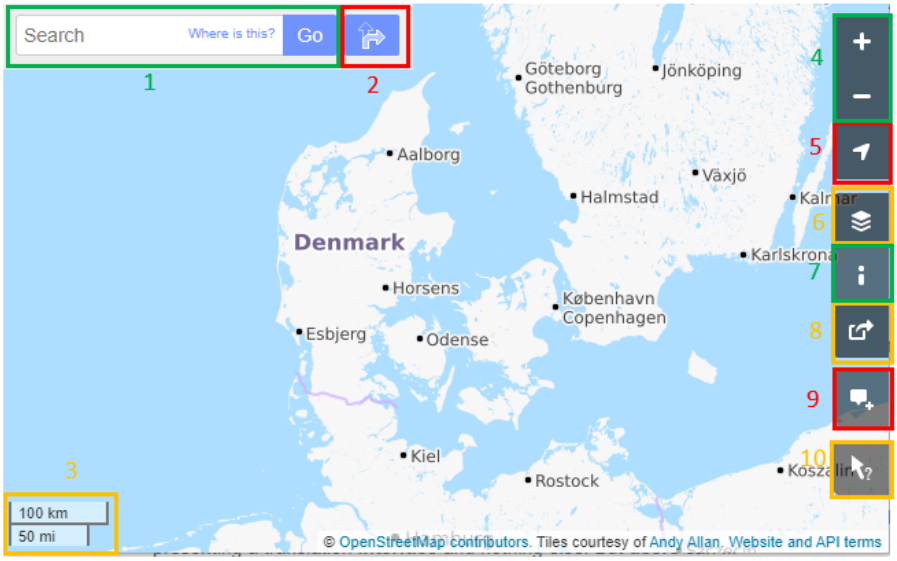


Figure 6.1: An example of an interactive map, Openstreetmap. The user interface components have been colored green if it is deemed useful for the developed tool or otherwise red. Source: [OpenStreetMap, 2020]

What the different parts of the UI do can be seen in table 6.1. 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.

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 6.1: Overview of the UI elements highlighted in figure 6.1

6.1 Relevant functionalities

The essential functionalities are the searchbar, zooming and Map key. Having a search bar allows the user to quickly pan to an area of interest and the zoom function allows the user to control the extent. Having a legend is also central to explain to the user which values the colors of the raster would correspond to.

The useful functions are the scale, the layer control, parts of the share function and the Query features. A scale can help the user get a sense of scale. However, this is only useful if the scale is presented with a unit, which the user easily can comprehend. As mentioned in section 2.4 the unit of the projection is in degrees, which would result in the scale also being in degrees. The layer control is important, since this would allow the map to be able to display multiple different population projections. The tool have been developed for single individuals, so the communicating with other is outside of the scope of this project. However it could still be relevant for a single user to be able to save an extent or export an image of the map. Getting more information about a clicked point could also be a beneficial feature. This information could for instance be the exact value in the clicked point.

The functions which are irrelevant are the route calculation, "Show my location" and "Add note to the map". The first is only useful for route planning, which is not the purpose of this map. The "Show my location" function is not relevant for this target audience. It would be relevant if the target audience was citizens, who just wanted to explore a population projection out of curiosity. These would potentially be interested in knowing what the expected population would be in their neighbourhood. The last function rely on multiple users, which is outside the scope of this project.

6.2 Comparing layers

The functions presented so far does not enable an easy visual comparison of raster data. It is possible currently by changing back and forth between different layers with different population projections. This however is a bit inconvenient. An alternative way would be to use comparison layers in a similar fashion to how it currently is being done with the makeCityWebsite tool mentioned in section 4.3. This would require the creation of comparison layers for each combination of SSPs and years, which should be compared. A third option would be to display multiple maps at the same time as illustrated in figure 6.2. These two maps are linked together by a shared view. The means the panning or zooming in one map would do the same operation on the other map.

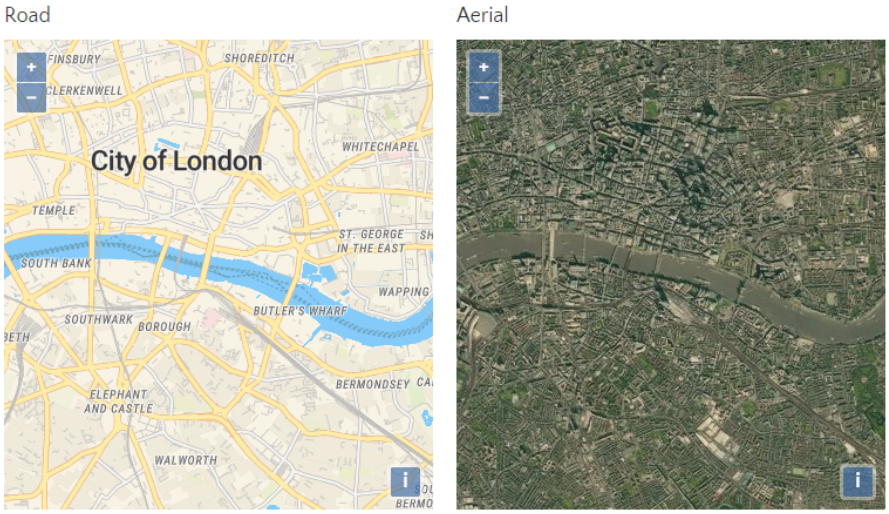


Figure 6.2: An example of two maps sharing the same view. Source: OpenLayers [No date]

This third option was chosen.

Related work 7

This is not the first project attempting to visualising large raster datasets in an interactive way.

7.1 Tilebased raster visualisation

In Baumrock [2018a] master thesis, he created a webmap, where raster tiles were being colored locally.

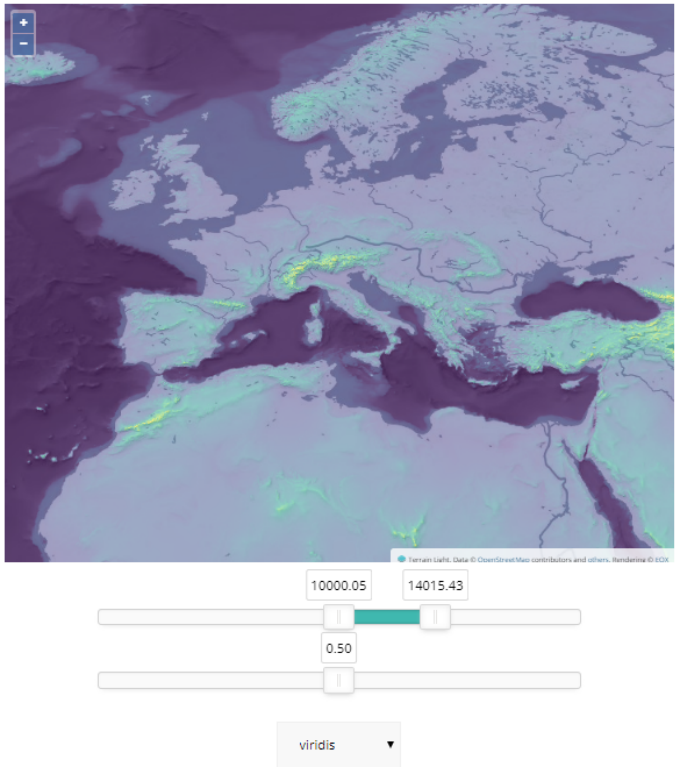


Figure 7.1: An elevation map, where the user can control how the raster is colored using the controls below the map. Created by [Baumrock, 2018b]

A picture of one of his maps can be seen in figure 7.1. This particular map is not included in his thesis, but is from an article he wrote [Baumrock, 2018b]. It is highlighted because it is the most similar to this project. The map shows an elevation data. Below the map are two sliders and a dropdown list with the label “viridis”. 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 7.2. When zooming in another more detailed layer gets loaded and rendered.

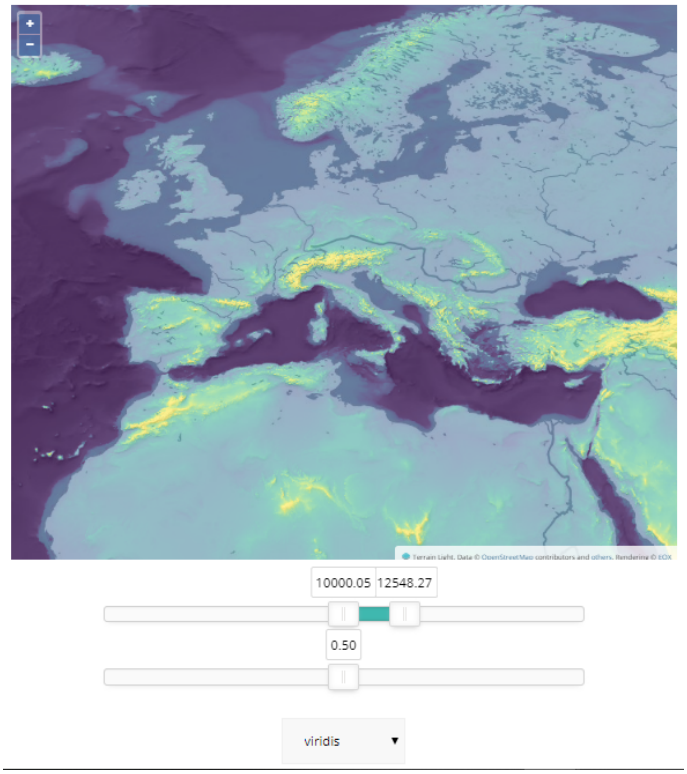


Figure 7.2: How the map change, when using the slider. Created by [Baumrock, 2018b]

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. [Baumrock, 2018b] Only the tiles that currently are visible are being requested unless the tile already have been requested. [Baumrock, 2018a]

How the tool functions from a technical perspective is further detailed in section 10.2.

Technical concepts 8

Five technical concepts have been developed based on the current solutions, related work and initial tests with the case data:

- 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

The purpose of these concepts is to ensure that a large dataset can be loaded and visualized correctly.

8.1 Load only the needed data

The case data presented in chapter 2 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.

8.2 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 how much the values are varying across the world. Figure 8.1 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 to have no population since the population density in Denmark is negligible compared with the most densely populated areas in the world. In the right map it is possible to see the location of the most populated cities. Since the left map provides the user with no relevant information and the right one does, the coloring should be based on the current extent.

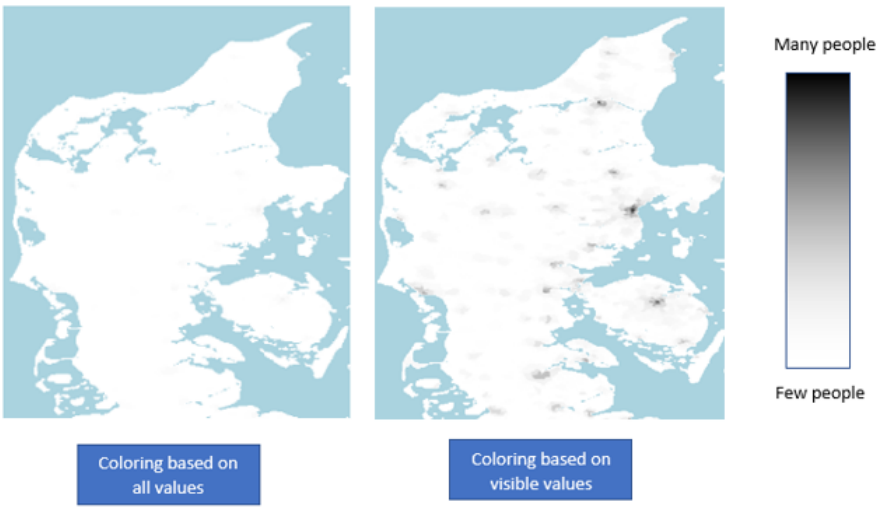


Figure 8.1: Population density in Jutland and Funen

8.3 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 8.2. 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 therefore has a higher 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 to 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 prerendered it would both require lots of initial processing time and storage space for an immense amount of data.

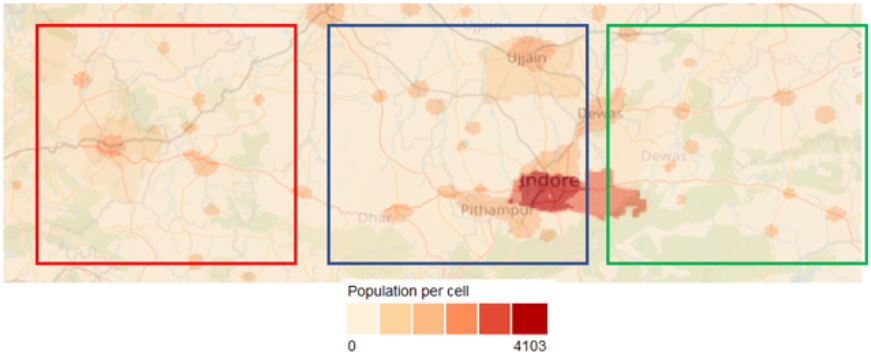


Figure 8.2: Population density. The three boxes are examples of possible extents an interacting user could get by zooming.

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

8.4 Coloring should be done on the client

The coloring can be done in two ways as illustrated in figure 8.3. 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.

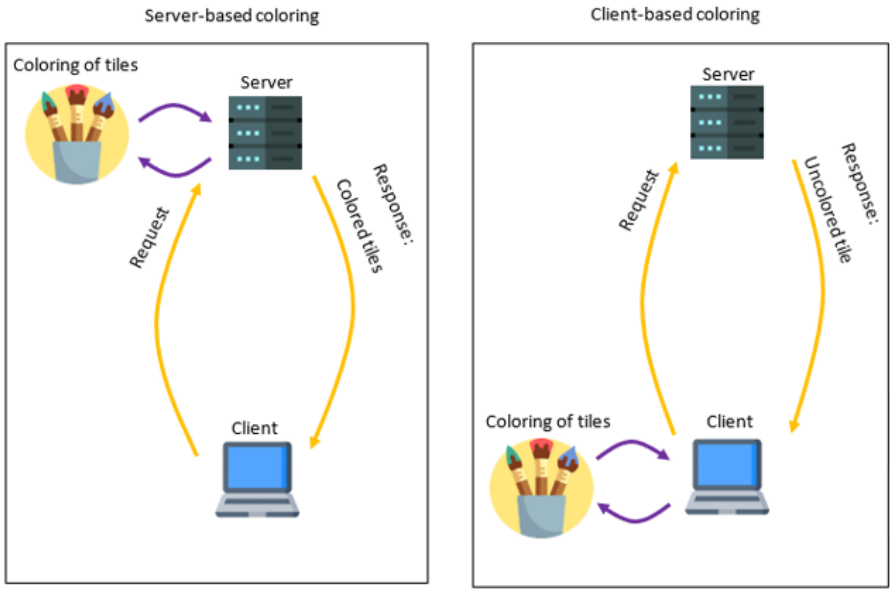


Figure 8.3: Difference between coloring the tile on the server and client. Inspired by a similar looking model by Baumrock [2018a]. Icon source: [Freepik, 2020]

Comparing the two options is best done with an example, which has been illustrated in figure 8.4. Here the blue and green box are two different possible views both containing

16 raster tiles. The two boxes have different max values, since the 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 pans 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 8.3 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.

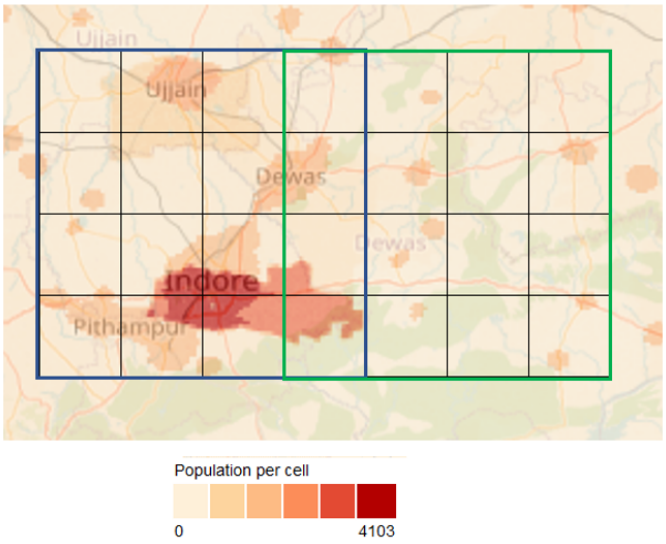


Figure 8.4: Example of why local coloring is needed. The squares are different extents used for the example

8.5 File format should have same bit depth as input format

As mentioned in section 2.4 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 8.5 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 3.1 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 gets clamped into the 8-bit format producing wrong result [GDAL, 2020].

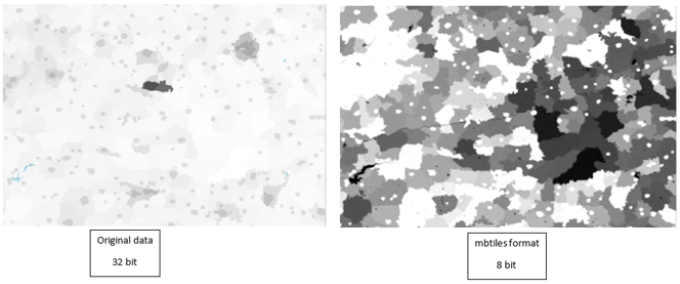


Figure 8.5: 32 bits cramped into 8 bits

Normally this issue could be worked around by rescaling the input data to the new format [GDAL, 2020] . This rescaling would be a lossy compression resulting in loss of information since the original data cannot be expressed with values between 0-255. [Dent et al., 2009] 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 reaches the client the file format used for tiles must have a bit depth of 32.

Final design 9

In this chapter the content of the previous chapter is summarised into design guidelines, which the tool will be build upon.

Most of the technical concepts are fulfilled by the map created by Baumrock [2018a] as mentioned in section 7.

It is loading and visualizing only the necessary data and 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.

It has therefore been decided to build the projection upon the foundation created by Baumrock. One major change will be the automatization of the coloring. Instead of the user defining values to color the map based on, it is instead done automatically. The map detects the maximum, which are within the current map extent and adjust the color scale dynamically to fit these. Minimum values are not being calculated, since the calculation resulted in a less responsive webpage without any visual changes. This is elaborated upon in section 13.2.

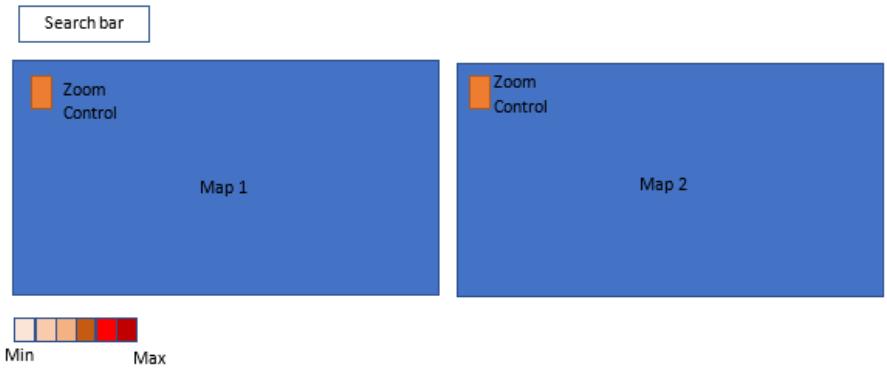


Figure 9.1: An illustration of the final design

The concept for a solution has been illustrated in figure 9.1. The tool will use two maps with a shared view, so that both maps show the same area. Each map will show a different scenario, so that different population projections can be compared. Only the essential functionalities will be implemented in this prototype. The other useful functions would be excellent options for future work.

9.0.1 Visualisation

The sequential color scale chosen for the visualized rasters has been created by colorbrewer.org. It was decided to use six different colors on the maps. This is despite the fact that Brewer et al. [2013] advise that more colors safely can be used, when similar colors are neighboring each other. The reason for this is that even though similar colors are neighboring each other within the individual map, the user still has to compare none-neighboring colors, since comparison is done between the two maps. The breakpoints between the different classes are determined as equal sized percentage steps of 20 % of the current maximum value.

The projection EPSG:4326 was both used in Baumrock's original map and for the case data and will also be used for this project. The map will not be used to visualise data at a subcontinental scale, so the distortion from this projection, will not have serious consequences. The reason for this limit in scale is the time and diskspace required to create the tiles, which will be expanded upon in section 10.5.

Part II

Development

Developing the tool 10

In this chapter it is explained how the tool has been developed. The two sections 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 10.1.

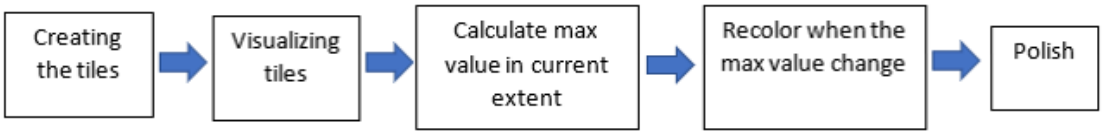


Figure 10.1: An overview of steps for building the code

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 10.4. Section 10.6 describes how the tiles have been visualized. To be able to color the tiles based on the current values one must know what the current values are. In section 10.8 it is explained how this information is calculated. The raster layer is then recolored based on these values as described in section 10.9. Lastly some measurements were taken to ensure a more user-friendly experience. These additions are described in section 10.10.

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 map has been tested on a local caddy server for reasons explained in section 10.3.

10.1 Languages

Python

Python is a 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.[W3Schools, No dateb] Python was used in this project because of the gdal library expanded further upon in section 10.2. This project has been using version 3.8.3 of Python, which at the time of writing is the latest version. [Python.org, 2020]

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.

[W3Schools, No datea]

10.2 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. [GDAL, 2020a]

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 [GDAL, 2020] [GDAL, 2020b] 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. [Commenthol, 2020] 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 8.5. The workaround for this issue will be explained in section 10.4.2.

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, No datea] Openlayers was chosen because the tool presented in related work was built in Openlayers.

Therefore, using Openlayers enables expanding upon this existing tool instead of starting from scratch.

The existing tool was created with the debug version of Openlayers, which this tool also have build on. The debug version is a version of the Openlayers, which have not been minified. Because of this it is a larger file than the regular one. [MappaGnosis, 2015]

Attempts at replacing it with the minified version resulted in errors from olGeoTiff, which have been build based on the debug version. Optimizing other parts of the code was given higher priority, so the debug version never got replaced.

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 related work section 7. 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 10.2. 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 [Baumrock, 2018a] thesis.

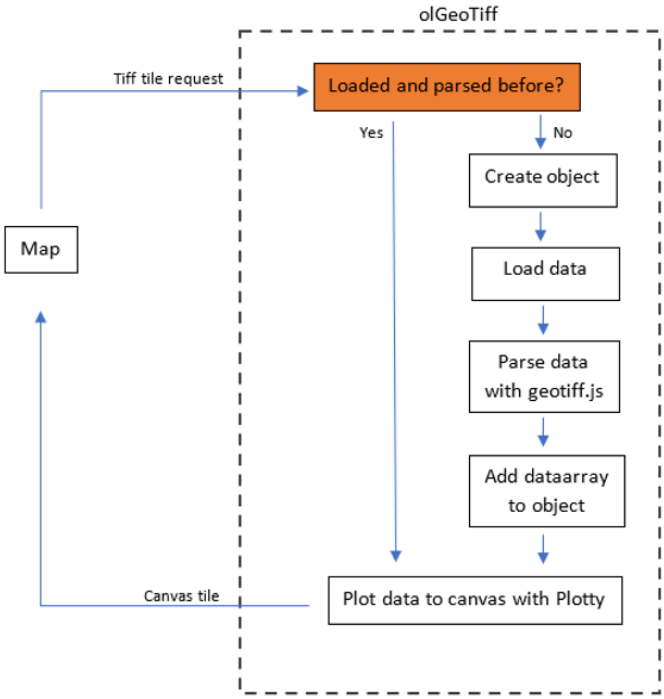


Figure 10.2: A simplified illustration of how olGeoTiff functions

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. [Baumrock, 2018a] The processing is done with the TIFF parser geotiff.js [Schindler, 2018] and Plotty, which is a library for creating images from data arrays [Santillan, 2020].

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. [Baumrock, 2018a] 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 Baumrock's map, whenever the color sliders were changed.

10.3 Testing server

To test the tool a local test server was set up. This was initially a Python-based http server, but was later replaced with a Go-based Caddy server to improve performance. The performance difference between the two is elaborated upon in section 12.0.2.

Caddy was chosen because it supports the second version of Hyper Text Transfer Protocol (HTTP) [Surma, 2016], which is used for the communication between the client and server. [w3schools, No date] HTTP/2 is allowing a faster communication between them. [Google Developers, 2019d] This is partially done by supporting an unlimited number of concurrent requests [MDN web docs, 2019], where HTTP/1 in Chrome only supports 6 [Google Developers, No datef].

10.4 Creating the tiles

The tiles were 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.

10.4.1 Changing the file type

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

```
1 #Original code
2 #self.tiledriver = 'PNG'
3 #self.tileext = 'png'
4 #New code
5 self.tiledriver = 'GTiff'
6 self.tileext = 'tiff'
```

Listing 10.1: Changing the file format

This changes the raster driver from png to the geotiff format and the file extension from png to tiff. [GDAL, No date] 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.

10.4.2 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 10.2. [Tosovsky and Strip, 2014]

```
1 self.mem_drv = gdal.GetDriverByName('MEM')
2  ...
3 #Old code
4 #dstile = mem_drv.Create('', tile_size, tile_size, tilebands)
5 #New code
6 dstile = self.mem_drv.Create('', self.tilesize, self.tilesize, tilebands, gdal.GDT_Int32)
```

Listing 10.2: 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.

10.4.3 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". [GDAL, 2020]

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 10.3.

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

Listing 10.3: Preventing KML files from being created

10.4.4 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>
```

Listing 10.4: Command for creating tiles

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.

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 option and its possible consequences is expanded upon in section 13.1.

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.

10.4.5 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. [Tosovsky and Strip, 2014] The generated tiles appears as they should, so this error message is potentially referring to the geospatial information, which gets lost.

The script also produces an xml file with metadata. An example of the content of said metadata file can be seen in code 10.5.

```
<?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 10.5: The metadata from the xml file generated by the modified gdal2tiles

10.5 Processing time

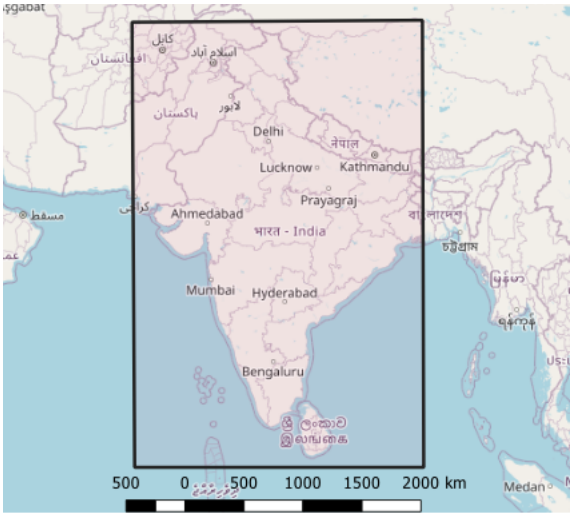


Figure 10.3: Illustration of the area used for timing the tile generation

The processing time is tested using the area of India shown in figure 10.3. This polygon with a 9 291 842 km^2 area was processed into tiles for each zoom level between 0 to 9. This is the equivalent of 349525 tiles as calculated with the formula presented in section 3.2.

$$2^{2n}2^{2*9} + 2^{2*8}...2^{2*0} = 349525 tiles \tag{10.1}$$

The creation of these tiles took 6 hours and 4 minutes and requires 6,52 GB of diskspace The amount of time mainly depends on the highest zoom level, since the number of tiles in a layer get quadrupled whenever the zoom level increases. In comparison the same area at the zoom levels 0-7 only took 22 minutes.

This processing time is not ideal, but it can theoretically be shortened with multiprocessing.

10.5.1 Parallel processing of tiles

Multiprocessing is the usage of multiple of the computer's processors. Python is by default only able to use a single processor, even if multiple are available. This can be circumvented with the multiprocessing module. [Data@Urban, 2018] The modified version by Commenthol [2020] also has a multiprocessing version allowing each processer of the computer to work separately on generating tiles. However, the task is not properly distributed between the processes. This means that not all of the tiles are being generated. [Herjar, 2019] Therefore, the slower single process version has been used for this project.

10.6 Visualizing tiles

After the tiles are created, they are stored in a folder, which is 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. [Openlayers, No dateb] 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 10.5. This led to some inconsistency when loading tiles, that had been generated without some of the lower zoom level. This will be further expanded upon in section 10.7. Therefore, the resolution was instead generated using the script 10.6, 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 10.6: The generation of a resolution matrix

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

10.6.1 Custom colors scheme

Using colorbrewer a custom sequential colorscheme was generated. This scheme was added to Plotty and selected as a color palette.

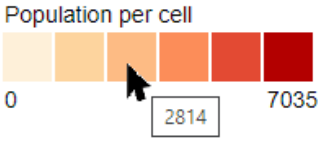


Figure 10.4: The color scale added to the map. When mousing over a color, the assigned value is displayed

The value assigned to each color can be found by mousing over the color as shown in figure 10.4

10.7 Loading data at a wrong resolution

Figure 10.5 shows how the map looked, when the tiles are visualized.

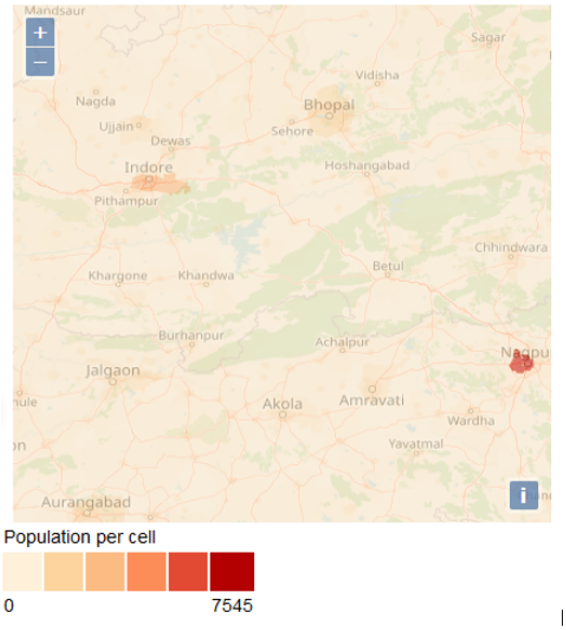


Figure 10.5: The map is looking correct, but with tiles from the wrong zoom level

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 three levels higher than intended. So the map in figure 10.5 is visualizing the map at zoom level 7, but loading and displaying tiles from zoom level 2. It was not possible to force the map to load tiles from a different zoom level.

Some experiments with loading tiles from the current view and zoom level 7 crashed 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.

In figure 10.6 these different zoom levels have been illustrated. The Displayed zoom level (DZL) is the tiles, that are being loaded and visualised in Openlayers. The expected zoom level (EZL) is the zoom level, which Openlayers is displaying the map in. However as mentioned earlier there are too many tiles to load, when using this zoom level and map extent. The coloring should therefore ideally be done based on a zoom level between these two zoom levels. This zoom level have been defined as the intended zoom level (IZL).

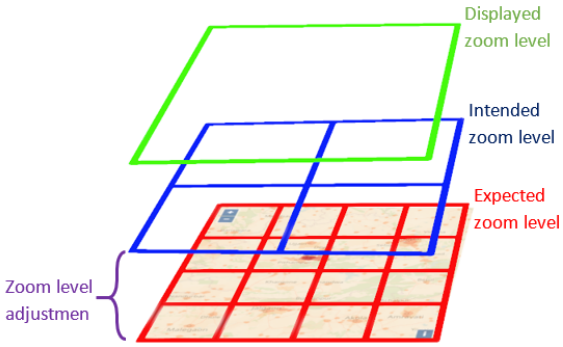


Figure 10.6: An overview of the differences in zoom levels, which the map is displayed at (red), the displayed tiles get loaded in (green) and the zoom level, which should have been displayed (blue). The zoom level adjustment is the difference between the Expected and Intended zoom level

This issue was not fixed, but theories behind the origin of it is discussed in section 13.1. This bug will be defining for the rest of the code.

10.8 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.

10.8.1 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 coordinate within a given zoom level and extent. [Openlayers, No datec] These tile coordinates can then be translated to the tile name and folder location using getTileUrlFunction.

```
1 var tileUrlFunction = wmslayer.getSource().getTileUrlFunction()
2 var zoomlevelAdjustment = 3
3 wmslayer.getSource().getTileGrid().forEachTileCoord(loadExtent, mapZoom - zoomlevelAdjustment,
      function(tileCoord) {
4       tileName = tileUrlFunction(tileCoord, ol.proj.get('EPSG:4326'))
5       //Find max Value In Tile function
6 }
```

Listing 10.7: Getting the tile name using getTileUrlFunction

If the function is just given the zoom level of the map it will trigger for the tiles in the Expected Zoom Level. This would not work, since there are too many tiles in this level with the current extent. Instead the function is used with the Intended Zoom Level, which is calculated as the Expected Zoom Level minus a zoom level adjustment. Using an adjustment value of 3 was found to give a responsive user experience.

10.8.2 Max value for each tile

If the tiles added to the map had been from the Intended Zoom Level, then the max value of a tile could have been calculated as olGeoTiff was running. olGeoTiff already holds the values for the tiles in a dataarray, so finding the maximum value could be done with a single line adding a tile's max value to the object for that tile.

However, since olGeoTiff holds data from an incorrect zoom layer this does not function. A possible solution would be to trigger forEachTileCoord for the Displayed Zoom Level. This solution was not implemented since the tiles from the Displayed 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 results in a map where the tiles from the Displayed zoom level were being colored based on information from the tiles at the Intended zoom level. 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 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 10.7. 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. This maximum value will then be returned to the map.

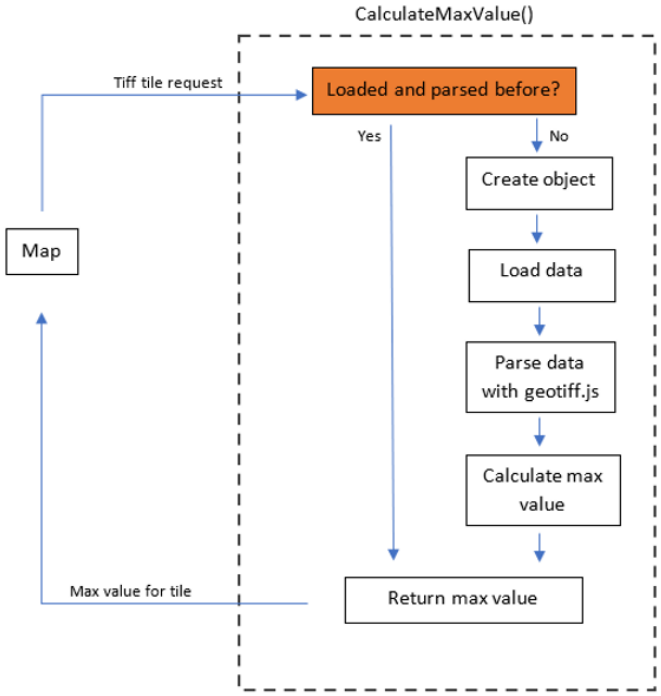


Figure 10.7: Flow diagram of the calculation over max value in a tile

10.8.3 Highest tile value currently displayed

To find the largest tile value among all the currently displayed tiles forEachTileCoord 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 started 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 10.8.

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

Listing 10.8: Counting the amount of tiles within the current extent

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 times 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 10.8. 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.

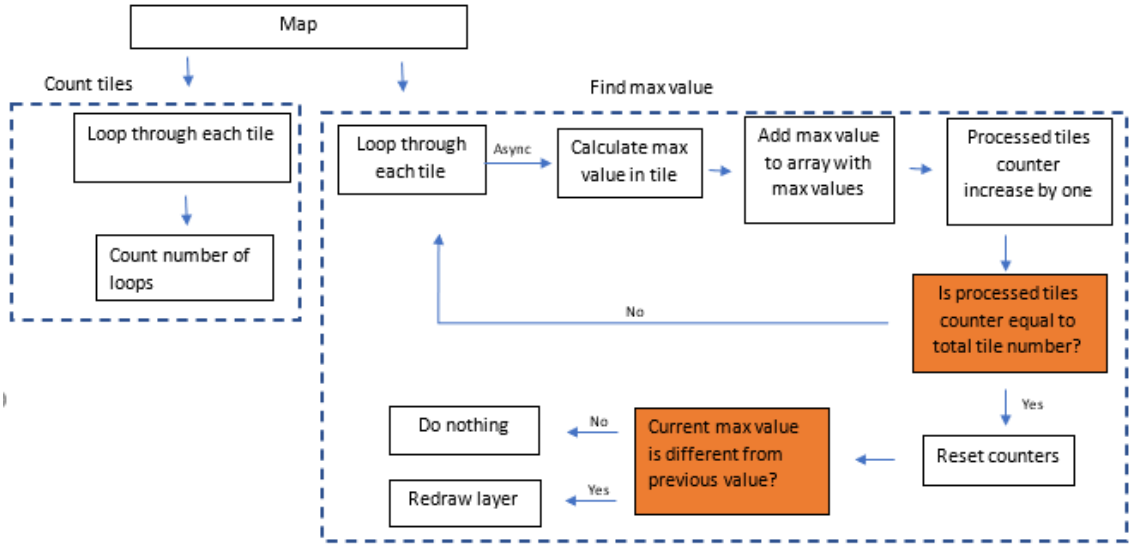


Figure 10.8: Finding the largest value in all currently displayed tiles

10.9 Recolor when the max value change

The recoloring function was already part of olGeoTiff. However, in Baumrock’s 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, when the user stops 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 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 10.9: The JavaScript in the project

10.10 Polish

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

10.10.1 Two maps

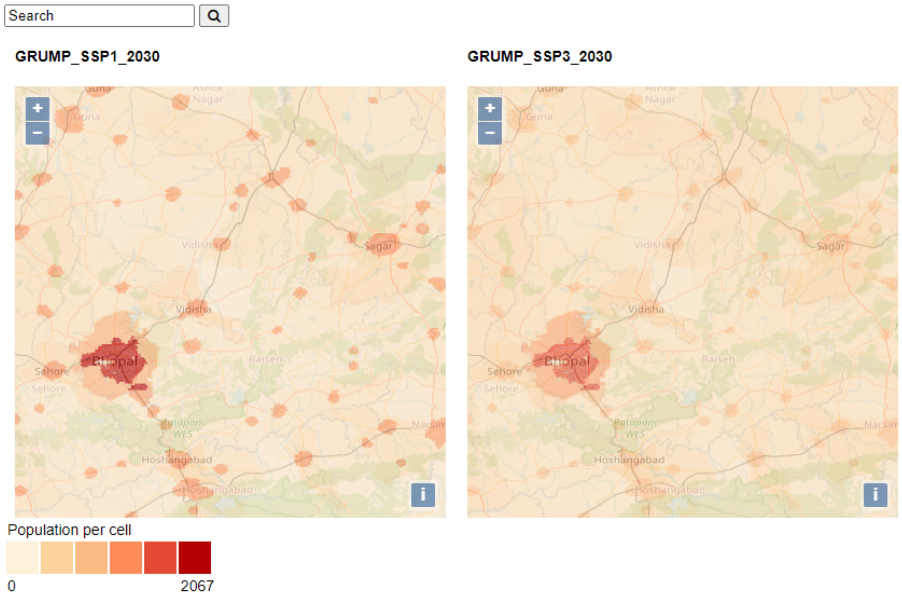


Figure 10.9: A second map beside the first one. Each map is showing a different dataset as shown above the maps

To be able to compare different rasters with each other a second map was added as illustrated in figure 10.9. 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. Above each map is the name of the dataset, which is shown in the map.

The two projection are also sharing the same color scale. This means that the coloring is done based on the maximum value in the current extent of either of the maps. To accomplish this changes were made to the function, which were finding the maximum value among the tiles as illustrated in figure 10.10.

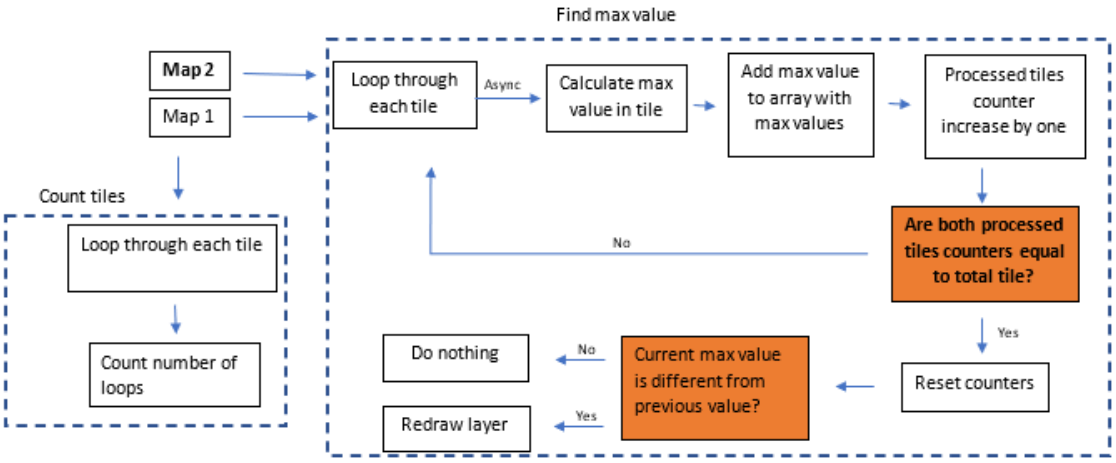


Figure 10.10: Calculating the maximum value for two maps. Changes from the original have been written in bold

This function is now being used twice, once for each layer. If the same function as presented in section 10.8 was used, then the redrawing would potentially trigger twice.

To avoid this each of the maps are given their own processed tiles counter. The if statement, which is resetting the counters and triggering the recoloring, has been changed, so that it now is checking if both of these processed tiles counters is equal to the total number of tiles. Thereby the recoloring can only happen, when the tile in both maps have been processed.

The two maps would always display the same amount of tiles, since they are sharing the same view. It is therefore only necessary to calculate the amount of tiles for one of the layers.

The size of the maps were set to 400x400 pixels, which did not take up the full width of the page. This was done for both a performance and a scientific reason. From a performance perspective a small map requires less tiles and is therefore faster to loaded. Decreasing the width from using the full extent of the screen (700 pixels per map) improved the initial loading time from 5.31 seconds to 3.61 seconds. The measurement of performance is further elaborated in chapter 11. The scientific reason is consistency in performance measurements. By having a fixed size, there would be no difference between measurement due to differences in screen size. How the empty space next to the maps could be utilised is described in section 13.3.1.

10.10.2 Search function

In order to be able to faster navigate the map a search function was created as shown in figure 10.11.

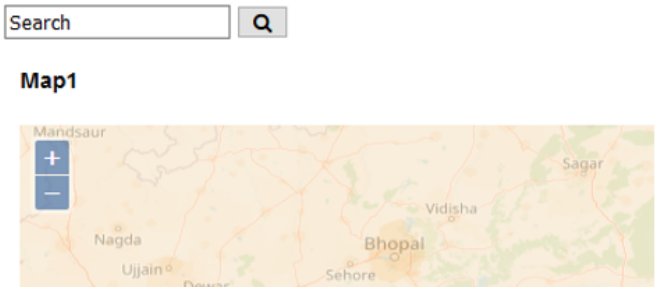


Figure 10.11: The map can pan to a specific area, by searching for that area in the searchbar

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. [Nominatim, No date] This coordinate is then used as the coordinate for the center of the map.

10.11 Final product

This section is an overview of the final product, which can be seen in figure 10.12. The datasets highlighted in the figure is elaborated upon in section 10.12.

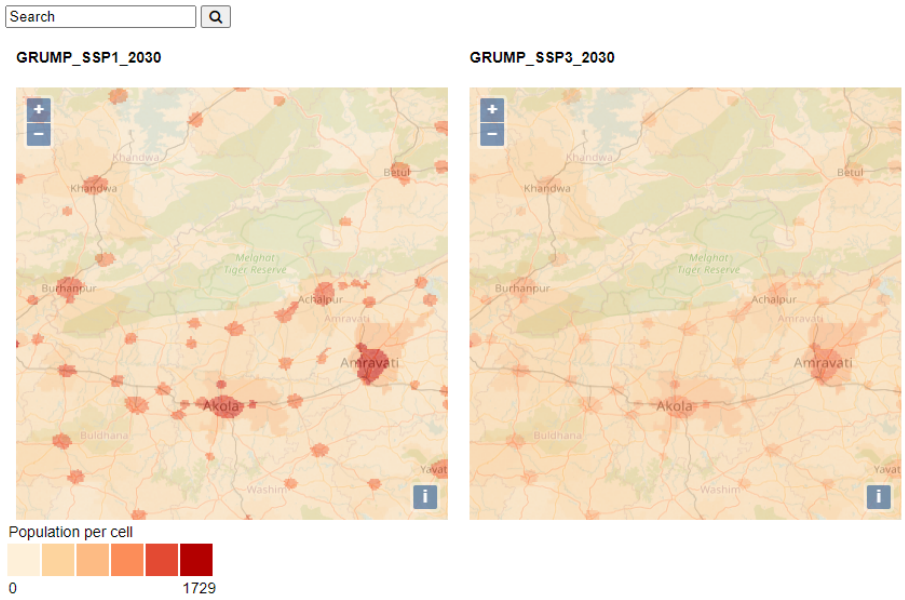


Figure 10.12: The final product

The main part of the final product is two maps, which are showing two different scenarios for the future. These maps have the same color scale and are always showing the same extent. The values in this color scale are automaticity updated to reflect the highest value

within the current extent. The user can see which values the colors are assigned by mousing over the colors.

Above the two maps are the names of the datasets, which currently are being displayed. Above these names is a search bar, which can be used to pan to a searched location.

10.12 Comparisons

This section highlights comparisons between different scenarios.

The datasets, which were displayed in figure 10.12 are SSP1 and SSP3 with GRUMP as urbanisation definition and year 2030. These two scenarios have been chosen, because of their differences as mentioned in section 2.3. The first scenario have a low population growth and fast urbanisation rate, whereas the other have a high population growth and slow urbanisation rate. This leads to areas with higher density in the SSP1 scenario, even though there is a bigger population in the SP3 scenario. The faster urbanisation means that this larger population gets spread over a larger area, which lowers the density.

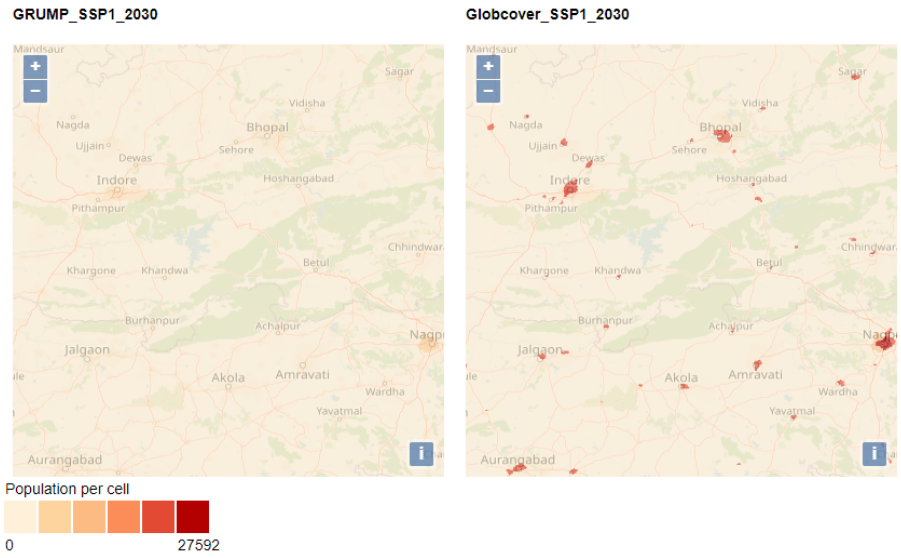


Figure 10.13: A comparison between different urbanisation definitions with the same scenario and year

Figure 10.13 shows the difference between the urbanisation definitions. The urban density in much higher, when using the GlobCover definition (right) compared with the GRUMP definition (left). This is due to the fact that both models have the same amount of people living in urban environments, but GRUMP overestimates the urban areas as mentioned in section 2.1.

This mean that a geosimulation with GlobCover as urban definition has smaller areas to squeeze the same population into, which leads to higher densities in the cities.

Part III

Evaluation

Evaluation Methods 11

In this chapter the methods for evaluating the product will be presented.

Due to the limitations mentioned in section 1.2 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.

11.1 Setup

The tests have been performed in Google Chrome version 81. To ensure that tests were unaffected by outside influences all Chrome extensions were disabled during testing. The cache was also disabled, so that every that tests would not be affected by previous tests.

11.2 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. [Google Developers, 2020c]

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. [Google Developers, No dated]

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.[MDN Web Docs, 2020] It is pointed out that accessibility is difficult to automatically test, so further manual testing is recommended. [Google Developers, No datea]

Best practice
The best practices cover different types of website enhancements. There are checks to

ensure that the website is fast, secure, and not using deprecated technologies, among others. [Google Developers, No dateb]

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. [Google Developers, No datee]

Progressive web app
The Progressive Web App (PWA) audits are targeted towards a mobile audience. These includes having a fast and reliable experience on mobile networks and to which degree the website act in a similar fashion to a native mobile application. [Google Developers, No datec]

11.2.1 Relevant evaluation criteria

Not all the audit categories in Google Lighthouse are relevant for this particular project. As mentioned in section 1.3 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 scientists and not for everybody. It is presumed that the majority of people, who work daily 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.

11.3 Metrics for performance

First Contentful Paint

The First Contentful Paint (FCP) is the time in seconds it takes for the browser to render the first parts of the website. [Google Developers, 2019a]

Speed Index

The Speed Index (SI) 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 processed using the Speedline module. The unit is seconds. [Google Developers, 2019g]

The Speedline module calculates the SI by calculating an Interval Score between each frame and then summaries all the Interval Scores. The Interval Score can be calculated using the formula 11.3, where the Interval is the elapsed time since last frame and Completeness is

the percentage of visual completeness. [WebpageTest.org, 2013]

$$IntervalScore = Interval * \left(1.0 - \frac{Completeness}{100}\right) \tag{11.1}$$

First Meaningful Paint

The First Meaningful Paint (FMP) is the seconds from the initial loading of the page to the primary content is visible. Where primary content is referring to the largest layout change, which is visible without scrolling.

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 did not give consistent result and was difficult to standardize in all web browsers. [Google Developers, 2019c] LCP is the seconds needed to render the single largest content element, which is visible without scrolling.[Google Developers, 2020b]

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. [Google Developers, 2019j]

First CPU Idle

First CPU Idle is the seconds before a page becomes minimally interactive. Minimally interactive is defined as when the majority of the visible 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. [Google Developers, 2019b] TBT is the total amount of milliseconds between FCP and TTI where the website is not responding to user input. [Google Developers, 2019i]

11.4 Calculating score

Each of the metrics in the section above are then converted into a score between 0 and 100. This score is calculated by comparing each metric with performance data from real websites in the HTTP Archive. [Google Developers, 2020a] The scores from each metrics are then weighted to reflect that each factor does not have the same importance for the perceived performance. The weight of each metric can be seen in table 11.1.

FCP	SI	FMP	TTI	FCI
20%	26.7%	6.7%	33.3%	13.3%

Table 11.1: Percentage weighting of the different metrics. Data source: Google Developers [No dateg]

11.5 Additional measurements

The measurement mentioned so far only take the initial load into consideration, not the postload performance. Therefore, two additional measurements were conducted. Both of these were measured using the Performance tool in the Google Chrome Developer tool. Since there are slight variation in the time needed to perform the requests, each operation was tested multiple times and the average was calculated.

11.5.1 Time to color new layer

This is the time in seconds required to load and color completely new sets of tiles for both maps. The scenario is created by zooming to a new zoom extent. This is done to ensure that none of the tiles on the map have been loaded before. The measured time is the time from the map is clicked, initiating the zooming, till the layer have been colored.

11.5.2 Time to recolor a loaded layer

This measurement is the time to recolor a layer, which already have been loaded. This scenario is created by panning to a previously loaded part of the map, so that it is recolored. It is measured from the moment, the panning stops since the recoloring script is initiated on ended movement. The timer is stopped, when the layer is recolored.

Evaluation 12

This chapter is the evaluation of the script based on the Lighthouse audit. This chapter will highlight some elements of the audit. The full audit can be seen in appendix A, but the score for Best Practices was 71 and for Performance 0. The low performance score was due to a bug. The actual score have been manually calculated in the end of section 12.0.1.

12.0.1 Performance

The performance in each of the metrics can be seen in table 12.1.

FCP	SI	FMP	TTI	FCI
20.2	20.2	20.2	809.1	20.2

Table 12.1: The result of the performance audit. The unit is seconds.

It does seem like some bug in the tool must have occurred because all of the values are too high. This can be confirmed by comparing with the measurements from the developer tool's performance tool. These show that the page is fully loaded in 4.6 second, which does not fit with Lighthouse's 20.2 seconds for First Contentful and First Meaningful Paint. Another sign that something is amiss is the 13.5 minutes before the Time to Interactive, which the tool determined within 10 seconds of analyzing the site. In the Lighthouse Scoring Guide the developers also point out the "A 0 score usually indicates an error in Lighthouse"[Google Developers, 2020d].

Even if the automatic calculation of the performance metrics failed the metrics are still relevant. Using the data from the performance tool these values can be manually estimated.

Figure 12.1: After 0.3 seconds the search bar and legend appears.

Figure 12.1, 12.2 and 12.3 are all screenshots of the website taken by the performance tool, while the page was loading. The maps are not utilising the full screen extent as mentioned in section 10.10.1, which is the reason for the empty white space next to the maps. Figure 12.1 is taken 0.3 seconds after the page started loading. This time is both the First Meaningful Paint and First CPU Idle since the user can interact with the searchbar.

Figure 12.2: After 1 second the map appears without the raster layer

After 1 second the maps appear on the website as shown in figure 12.2. The user is able to pan and zoom in the map, so this is the Time to Interactive.

Figure 12.3: After 3.6 seconds the map is fully loaded

The last screenshot at figure 12.3 shows the map fully loaded after 3.6 seconds. With the tiff tiles being loaded and visualized this is the time for the First Meaningful Paint. This can be used as a conservative estimate for the Speed Index. This would be the value for the Speed Index if everything got visualized at this time and nothing had been visualized on the website prior.

The real Speed Index would be lower, since the legend, map titles and search bar have been visualized at an earlier stage.

These metrics can then manually be calculated into a score using the Lighthouse Scoring Calculator as shown in figure 12.4. The calculator in unable to take inputs below 1 second. Based on the given value the performance should have scored 94. This score will be discussed upon in section 13.3

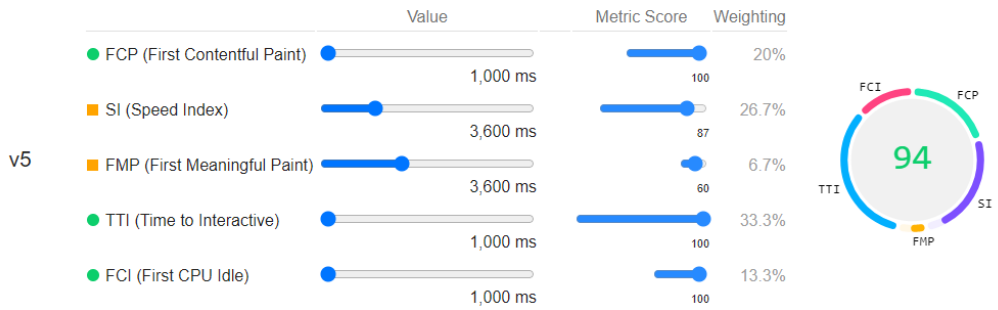


Figure 12.4: The website for manually calculating the scores, which Lighthouse grants using the Lighthouse Scoring Calculator. Source: Google Developers [No dateg]

12.0.2 Opportunities

In addition to providing the metrics for the performance Lighthouse also comes with suggestion to how the loading time can be reduced. In this subsection only some of these will be preset. The rest can be seen in appendix A.

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. [Google Developers, 2019h]

The largest render-blocking resource is Openlayers, which at 2784 kB makes up 82 % of the blocking resources. When analysed further with the Coverage tool in the dev tools it can be seen that 42.7 % of the Openlayers library is not being used. The performance could therefore be improved by only loading the necessary parts of the library. The size of it could also be reduced by using the minified version of it instead of the debug-version.

Avoid enormous network payloads
Long loading times are highly correlated with the amount of loaded data. [Google Developers, 2019e]

Loading the raster tiles for the map does require loading multiple files with a large bit depth, which requires a lengthy loading time. The amount of loaded tiles could be reduced, if the issue coursing tiles from the wrong layer to be loaded got fixed. A potential course to this issue is presented in section 13.1.

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. [Google Developers, 2019d]

This was the suggestion, which lead to replacing initial testing server with a Caddy test server, which vastly reduced the load time as illustrated in figure 12.5.

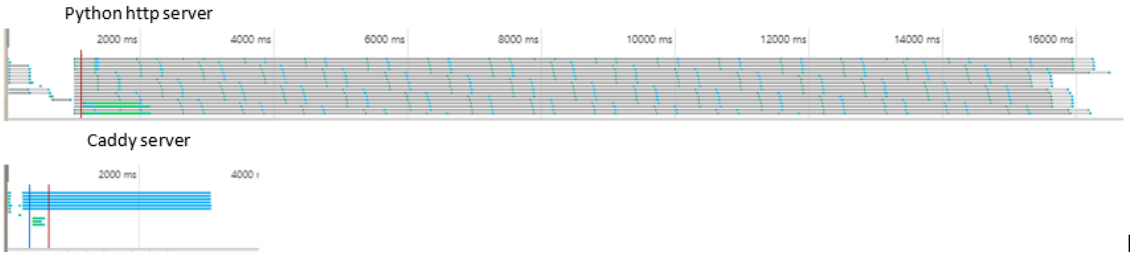


Figure 12.5: Screenshot of the network connection for a python and a caddy server

It 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. [Google Developers, No datef] This figure shows that the serving has being optimized, since the stalling largely is gone.

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.

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. This should be rewritten as an asynchronous request instead. [Google Developers, 2019f]

12.1 Postload performance

In addition to the measurements of the performance for loading the site the performance after loading was also measured. In table 12.2 is measurements of performance of using the tool. It can be seen that loading and coloring a new layer takes 3.89 seconds, whereas coloring a layer, which already have been loaded takes on average 0.56 seconds.

Time to color new layer	Time to recolor loaded layer
3.92	0.6
3.85	0.4
3.98	0.5
3.82	0.8
3.88	0.5
Average	
3.89	0.56

Table 12.2: Seconds to load a new layer and color it and to recolor a loaded layer

Discussion 13

This chapter is a discussion about the developed tool, how it was evaluated and how it could be improved in the future.

13.1 Loading tiles from the wrong zoom level

As mentioned in section 10.7 the tiles are being loaded from an incorrect zoom layer. A potential explanation for this could be the difference between what gdal2tiles and Openlayers define as zoom level 0. Openlayers defines zoom level 0 as the whole world, whereas gdal2tiles defines it as the whole raster.

It was discovered that the tile from zoom level 0 would be rendered in Openlayers, when the entirety of the raster was visible. This means that if tiles were generated for the entire world, then the zoom levels would be the same.

The tiles could possibly be ordered and named correctly by running gdal2tiles with a different profile. It was originally run with the raster profile, because the other profiles resulted in an error message as mentioned in section 10.4.4.

13.2 Not calculating minimum values in tiles

In the original version of the map both the minimum and maximum values were being calculated. However the program consistently defined the lowest value in currently visible tiles as 0. This does not mean that all the tiles had minimum values of 0, just that there always were a tile in the map extent with a value of 0. In the vast majority of case the value would be 0. Few would be below 100. In the most densely populated cities the minimum value for some tiles was higher than 1000. It is possible that by zoom close enough to a major city it none of the tiles within view would have a value of 0 in which case the minimum value should be adjusted. This is the main argument for calculating a minimum value. The downside of calculating it is that it has an significant effect on the processing time of the raster layer. Table 13.1 shows the time in seconds it takes for the layer to be rendered with and without calculating the minimum value. On average the rendering time is a second faster, when not calculating the minimum value.

The reason that the calculation of minimum value has such an effect on the processing time is presumably that the minimum value cannot be calculated in the same way as the

Calculating minimum value	Not calculating minimum value
4.77	3.66
4.78	3.56
4.7	3.77
4.79	3.79
4.61	3.62
Average	
4.73	3.68

Table 13.1: Seconds to fully render the website with and without calculating the minimum value

maximum value. The maximum value is calculated by finding the maximum value in the array with all values.

If the same operation is done to find the minimum value, the found minimum value often is -2147483648. As mentioned in section 2.4 that value means that no data is available for that pixel. It is therefore necessary to find the lowest positive value since no data is not the same as 0. This can be done by first filtering the raster, so only the positive values are left. The smallest of these values can then be calculated in the same way as the maximum value get calculated. It was decided that calculating a minimum value, which mostly would be 0 was not worth an additional second of loading.

13.3 Evaluating with Lighthouse

Having a performance score of 94/100 would indicate that the created tool is very responsive. This is surprising considering that there is a delay between 3-4 seconds before the raster is loaded, which does not make it "feel" very responsive. There are multiple reasons as to why such a high score have been achieved.

As mentioned in section 11.3 both the First Meaningful Paint and First CPU Idle will be replaced in the next release. Using the new measurements, the score might change.

It should also be pointed out that the score is calculated by comparing with real website data as mentioned in sectionScoreCal. The created tool is with its total of 4 components maybe more simplistic than most websites, which could explain why it is performing relatively well. A slow simple website might comparatively do well, when compared with a fast, but more complex website.

Ultimately it is difficult to determine how responsive the website is in comparison to other sites. Despite this Lighthouse have been a useful improvement tool, even though it was not the best evaluation tool. The most beneficial part about the tool has been the advice on how a website can be improved. Its advice on upgrading to a HTTP/2 connection meant that the website became responsive enough to be useful.

13.3.1 Additional information

The comparison between the scenarios SSP1 and SSP3, which was done in section 10.12, showed that the user could benefit from more information about the datasets. In the comparison it was not possible to see that the overall population was higher for the SSP3 scenario, since the urban areas had a higher density in the other scenario. Multiple features could be implemented to give the user more information about the data.

The results of simple statistical calculations for each map could be displayed next to the maps. This could be the average density for the current extent, highest density in the map and the standard deviation.

Statistical data could also be visualised through histograms. In the empty space beside the maps there could be a histogram for each map, showing the distribution of the data.

A function could also be added to get the population counts from a clicked point for both maps. When the user clicks the map an infobox informs the user about the value at the clicked point for both maps.

13.4 Optimizing the tile generation

Currently the creation of the tiles is rather time consuming. This issue could be eliminated by either using multiple processers or a different technology.

13.4.1 Parallel generation of tiles

As mentioned in section 10.5.1 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.

13.4.2 Cloud Optimized Geotiff

An alternative to separating the rasterfile into smaller tiles would be to have the entire raster as one file, but then only send part of the file. This is possible using Cloud Optimized GeoTIFF (COG). These are GeoTiff files, which are organized in internal “tiles”. The files also contain multiple versions of the same image, where each is a downsampled version of the original. Each of these image versions can match a different zoom level.

This way of organizing the file is then combined with Range requests, which allows the client to request parts of a file instead of the whole file. [Cogeo.org, No date]

By using these two technologies tiles of different spatial resolution can be sent to the client, which is similar to what the current solution is doing. The main difference is that with COG there would be one file, while the current solution has thousands of files. The

creation of this file takes significantly less time compared to creating the individual tiles. India was as mentioned in section 10.5 processed into tiles in 6 hours and four minutes. Converting the same file into a COG takes 3 seconds.

There is currently no support for COG in Openlayers and no extensions enabling it. Leaflet, another mapping program, does have extensions, which support visualisation of the filetype. [Mohr, 2020]

Conclusion 14

In this chapter the research questions listed below will be answered.

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

The intent with this project was to create a tool for easy visual comparison of raster datasets using population projections as a case. This required understanding how raster data should be visualized and which functionalities a comparison tool should have. How population raster should be visualized have been determined through a literature review of visualization conventions.

Population projections are based on quantitative data, which is ordered sequentially. For these it is important that lighter colors are assigned the least dense areas The functionalities needed for an interactive tool was selected by investigating a state-of-the-art interactive map. Based on this it was decided to add a search bar to enable easy navigation.

To be able to compare different population projections two maps were displayed side by side. These were set up to always show the same area. The color of the projections inside was set up to automatically adjust to the population within the visible extent.

To ensure a responsive user experience when visualizing large raster dataset technical requirements were set up for the solution. The data loaded into the map should be limited to a minimum, since loading unnecessary data would slowdown or crash the tool. To limit the loaded data the population projection was divided into raster tiles. Only the tiles within the map's extent was loaded. These tiles were then colored based on the maximum values in the current extent. This meant that the same tile was colored in different ways dependent on the current extent. This coloring was done on the client instead of on the server. This limited the number of requests to the server, since already loaded tiles could be reused. To be able to color the tiles at the client the tiles had to contain information about population within each of their pixels instead of the pixels just having a color.

The responsiveness of the tool also got evaluated using the Lighthouse tool. Using this tool, it was discovered that the main factor for the responsiveness was the type of test server. Changing to Caddy as a test server reduced the tile loading time from 16 to below 4 seconds.

Bibliography

Baumrock, **2018a**. Bernhard Baumrock. *Client-side Visualisation of Scientific RasterData Using WebGL and Open-Source Web Mapping Technologies*. `https://pdfs.semanticscholar.org/16e2/ee741969d14ba6effb9eb9686415edbb32c9.pdf`, 2018. Page: 10, 27, 33-36, Last Accessed: 24-05-2020.

Baumrock, **2018b**. Bernhard Baumrock. *Visualizing GeoTIFF Tiles with OpenLayers*. `https://web.archive.org/web/20191031034339/https://eox.at/2018/01/visualizing-geotiff-tiles-with-openlayers/`, 2018. Last Accessed: 24-05-2020.

Baumrock, **2020**. Bernhard Baumrock. *olGeoTiff class*. `http://webportals.ipsl.jussieu.fr/ScientificApps/dev/forge_patrick/eox/lib/olGeoTiff.js`, 2020. Last Accessed: 02-06-2020.

Brewer, **1994**. Cynthia Brewer. *Guidelines for use of the perceptual dimensions of color for mapping and visualization*. `https://www.spiedigitallibrary.org/conference-proceedings-of-spie/2171/0000/Guidelines-for-use-of-the-perceptual-dimensions-of-color-for/10.1117/12.175328.short`, 1994. Last Accessed: 05-05-2020.

Brewer, **No date**. Cynthia Brewer. *Color Use Guidelines for Mapping and Visualization*. `http://www.personal.psu.edu/cab38/ColorSch/Schemes.html`, No date. Last Accessed: 05-05-2020.

Brewer et al., **2013**. Cynthia Brewer, Mark Harrower, Ben Sheesley, Andy Woodruff og David Heyman. *ColorBrewer 2.0*. `https://colorbrewer2.org/#`, 2013. Last Accessed: 24-05-2020.

Cogeo.org, **No date**. Cogeo.org. *Cloud Optimized GeoTIFF in depth*. `https://www.cogeo.org/in-depth.html`, No date. Last Accessed: 02-06-2020.

Commenthol, **2020**. Commenthol. *gdal2tiles-leaflet*. `https://github.com/commenthol/gdal2tiles-leaflet`, 2020. Last Accessed: 24-05-2020.

Data@Urban, **2018**. Data@Urban. *Using Multiprocessing to Make Python Code Faster*. `https://medium.com/@urban_institute/using-multiprocessing-to-make-python-code-faster-23ea5ef996ba`, 2018. Last Accessed: 27-05-2020.

Dent et al., **2009**. Borden D. Dent, Jeffrey S. Torguson og ThomasW. Hodler. *Cartography - Thematic Map Design*. ISBN 978–0072943825, 2009. Page: 61, 251-252, 261-262, 283-286, Last Accessed: 05-05-2020.

Ebi et al., 2014. Kristie L. Ebi, Tom Kram, Detlef P. van Vuuren, Brian C. O'Neill og Elmar Kriegler. *A New Toolkit for Developing Scenarios for Climate Change Research and Policy Analysis*. https://doi.org/10.1080/00139157.2014.881692, 2014. Page 10, Last Accessed: 24-05-2020.

ESRI, **No date**. ESRI. *NoData and how it affects analysis*. https://desktop.arcgis.com/en/arcmap/latest/extensions/spatial-analyst/performing-analysis/nodata-and-how-it-affects-analysis.htm, No date. Last Accessed: 04-06-2020.

Freepik, **2020**. Freepik. *Freepik*. https://www.flaticon.com/authors/freepik, 2020. Last Accessed: 04-06-2020.

Garlandini og Fabrikant, **2009**. Simone Garlandini og Sara Irina Fabrikant. *Evaluating the Effectiveness and Efficiency of Visual Variables for Geographic Information Visualization*. http://www.geo.uzh.ch/~sara/pubs/garlandini_fabs09.pdf, 2009. Page: 1, Last Accessed: 24-05-2020.

GDAL, **2020a**. GDAL. *GDAL*. https://gdal.org/, 2020. Last Accessed: 24-05-2020.

GDAL, **2020b**. GDAL. *Download*. https://gdal.org/download.html#current-releases, 2020. Last Accessed: 24-05-2020.

GDAL, **No date**. GDAL. *Raster drivers*. https://gdal.org/drivers/raster/index.html, No date. Last Accessed: 24-05-2020.

GDAL, **2020**. GDAL. *gdal2tiles*. https://gdal.org/programs/gdal2tiles.html, 2020. Last Accessed: 24-05-2020.

Google Developers, **2019a**. Google Developers. *First Contentful Paint*. https://web.dev/first-contentful-paint/, 2019. Last Accessed: 27-05-2020.

Google Developers, **2019b**. Google Developers. *First CPU Idle*. https://web.dev/first-cpu-idle/, 2019. Last Accessed: 27-05-2020.

Google Developers, **2019c**. Google Developers. *First Meaningful Paint*. https://web.dev/first-meaningful-paint/, 2019. Last Accessed: 27-05-2020.

Google Developers, **2019d**. Google Developers. *Does not use HTTP/2 for all of its resources*. https://web.dev/uses-http2/?utm_source=lighthouse&utm_medium=devtools, 2019. Last Accessed: 25-05-2020.

Google Developers, **2020a**. Google Developers. *Lighthouse performance scoring*. https://web.dev/performance-scoring/, 2020. Last Accessed: 04-06-2020.

Google Developers, **2020b**. Google Developers. *Largest Contentful Paint*. https://web.dev/lcp/, 2020. Last Accessed: 27-05-2020.

Google Developers, **No datea**. Google Developers. *Accessibility audits*. https://web.dev/lighthouse-accessibility/, No date. Last Accessed: 24-05-2020.

Google Developers, **No dateb**. Google Developers. *Best Practices audits*. https://web.dev/lighthouse-best-practices/, No date. Last Accessed: 24-05-2020.

Google Developers, **No datec**. Google Developers. *PWA audits*. https://web.dev/lighthouse-pwa/, No date. Last Accessed: 24-05-2020.

Google Developers, **No dated**. Google Developers. *Performance audits*. https://web.dev/lighthouse-performance/, No date. Last Accessed: 24-05-2020.

Google Developers, **No datee**. Google Developers. *SEO audits*. https://web.dev/lighthouse-seo/, No date. Last Accessed: 24-05-2020.

Google Developers, **2020c**. Google Developers. *Lighthouse*. https://developers.google.com/web/tools/lighthouse/, 2020. Last Accessed: 24-05-2020.

Google Developers, **2019e**. Google Developers. *Avoid enormous network payloads*. https://web.dev/total-byte-weight/?utm_source=lighthouse&utm_medium=devtools, 2019. Last Accessed: 25-05-2020.

Google Developers, **2019f**. Google Developers. *Uses deprecated APIs*. https://web.dev/deprecations/?utm_source=lighthouse&utm_medium=devtools, 2019. Last Accessed: 25-05-2020.

Google Developers, **No datef**. Google Developers. *Timing breakdown phases explained*. https://developers.google.com/web/tools/chrome-devtools/network/reference?utm_source=devtools#timing-explanation, No date. Last Accessed: 04-06-2020.

Google Developers, **2019g**. Google Developers. *Speed Index*. https://web.dev/speed-index/?utm_source=lighthouse&utm_medium=devtools, 2019. Last Accessed: 27-05-2020.

Google Developers, **2019h**. Google Developers. *Eliminate render-blocking resources*. https://web.dev/render-blocking-resources/?utm_source=lighthouse&utm_medium=devtools, 2019. Last Accessed: 25-05-2020.

Google Developers, **No dateg**. Google Developers. *Lighthouse Scoring Calculator*. https://googlechrome.github.io/lighthouse/scorecalc/, No date. Last Accessed: 04-06-2020.

Google Developers, **2020d**. Google Developers. *Lighthouse Scoring Guide*. https://developers.google.com/web/tools/lighthouse/v3/scoring, 2020. Last Accessed: 04-06-2020.

Google Developers, **2019i**. Google Developers. *Total Blocking Time*. https://web.dev/lighthouse-total-blocking-time/, 2019. Last Accessed: 27-05-2020.

Google Developers, **2019j**. Google Developers. *Time to Interactive*. https://web.dev/interactive/, 2019. Last Accessed: 27-05-2020.

Herjar, **2019**. Herjar. *gdal2tiles.py multiprocessor option not working for overview tiles*. https://github.com/OSGeo/gdal/issues/1188, 2019. Last Accessed: 27-05-2020.

Jones og O'Neill, **2016**. Bryan Jones og Brian C. O'Neill. *Spatially explicit global population scenarios consistent with the Shared Socioeconomic Pathways*. https://iopscience.iop.org/article/10.1088/1748-9326/11/8/084003/meta#references, 2016. Last Accessed: 24-05-2020.

Keßler, **2019a**. Carsten Keßler. *MakeCityWebsite.py*. https://github.com/crstn/CISC/blob/master/MakeCityWebsite.py, 2019. Last Accessed: 04-06-2020.

Keßler, **2019b**. Carsten Keßler. *CompareWithSEDAC.py*. https://github.com/crstn/CISC/blob/master/CompareWithSEDAC.py, 2019. Last Accessed: 04-06-2020.

Keßler og Marcotullio, **2017**. Carsten Keßler og Peter J. Marcotullio. *A Geosimulation for the Future Spatial Distribution*. http://carsten.io/agile-2017-short-Kessler-Marcotullio.pdf, 2017. Last Accessed: 24-05-2020.

Lafaire, **2018**. Sarah Sophie Lafaire. *An interactive visualization tool for population simulations*. https://projekter.aau.dk/projekter/files/281131238/MasterThesis_Sarah_Lafaire.pdf, 2018. Page: 42- 44, 50-51, Last Accessed: 04-06-2020.

Liedman, **No date**. Per Liedman. *The Hitchhacker's Guide To Tiled Maps*. http://www.liedman.net/tiled-maps/, No date. Last Accessed: 24-05-2020.

MappaGnosis, **2015**. MappaGnosis. *What's difference between 'ol.js' and 'ol-debug.js' files in OpenLayers-3?* https://gis.stackexchange.com/a/155532, 2015. Last Accessed: 04-06-2020.

MDN Web Docs, **2020**. MDN Web Docs. *ARIA*. https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA, 2020. Last Accessed: 24-05-2020.

MDN web docs, **2019**. MDN web docs. *Domain sharding*. https://developer.mozilla.org/en-US/docs/Glossary/Domain_sharding, 2019. Last Accessed: 04-06-2020.

Mohr, **2020**. Matthias Mohr. *Add tutorials for webmapping support (Openlayers, Leaflet, ...)*. https://github.com/cogeotiff/www.cogeo.org/issues/36#issuecomment-604331729, 2020. Last Accessed: 02-06-2020.

Nominatim, **No date**. Nominatim. *Search queries.* https://nominatim.org/release-docs/develop/api/Search/, No date. Last Accessed: 25-05-2020.

OpenLayers, **No date**. OpenLayers. *Shared Views.* https://openlayers.org/en/latest/examples/side-by-side.html, No date. Last Accessed: 04-06-2020.

Openlayers, **No datea**. Openlayers. *A high-performance, feature-packed library for all your mapping needs.* https://openlayers.org/, No date. Last Accessed: 24-05-2020.

Openlayers, **No dateb**. Openlayers. *WMTS Layer from Capabilities.* https://openlayers.org/en/latest/examples/wmts-layer-from-capabilities.html, No date. Last Accessed: 25-05-2020.

Openlayers, **No datec**. Openlayers. *ol/tilegrid/TileGrid TileGrid.* https://openlayers.org/en/latest/apidoc/module-ol_tilegrid_TileGrid-TileGrid.html#forEachTileCoord, No date. Last Accessed: 25-05-2020.

OpenStreetMap, **2020**. OpenStreetMap. *OpenStreetMap.* www.openstreetmap.org, 2020. Last Accessed: 24-05-2020.

OpenStreetMap Wiki, **2020**. OpenStreetMap Wiki. *Slippy map tilenames.* https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames, 2020. Last Accessed: 24-05-2020.

OpenStreetMap Wiki, **2019**. OpenStreetMap Wiki. *TMS.* https://wiki.openstreetmap.org/wiki/TMS, 2019. Last Accessed: 24-05-2020.

OSGeo Wiki, **2012**. OSGeo Wiki. *Tile Map Service Specification - TileMap Diagram.* https://wiki.osgeo.org/wiki/Tile_Map_Service_Specification#TileMap_Diagram, 2012. Last Accessed: 24-05-2020.

O'Neill et al., **2013**. Brian C. O'Neill, Elmar Kriegler, Keywan Riahi, Kristie L. Ebi, Stephane Hallegatte, Timothy R. Carter, Ritu Mathur og Detlef P. van Vuuren. *A new scenario framework for climate change research: the concept of shared socioeconomic pathways.* https://link.springer.com/article/10.1007/s10584-013-0905-2, 2013. Page: 391-392, Last Accessed: 24-05-2020.

Planet, **No date**. Planet. *Slippy maps 101.* https://developers.planet.com/tutorials/slippy-maps-101/, No date. Last Accessed: 24-05-2020.

Python.org, **2020**. Python.org. *Python - Download the latest version for Windows.* https://www.python.org/downloads/, 2020. Last Accessed: 02-06-2020.

Rus, **2007**. Jacob Rus. *File:Munsell-system.svg.* https://commons.wikimedia.org/wiki/File:Munsell-system.svg, 2007. Last Accessed: 24-05-2020.

Santillan, **2020**. Daniel Santillan. *plotty*. https://github.com/santilland/plotty, 2020. Last Accessed: 24-05-2020.

Schindler, **2018**. Fabian Schindler. *geotiff.js*. https://geotiffjs.github.io/, 2018. Last Accessed: 24-05-2020.

Surma, **2016**. Surma. *Setting up HTTP/2*. https://dassur.ma/things/h2setup/, 2016. Last Accessed: 02-06-2020.

Tosovsky og Strip, **2014**. Jan Tosovsky og David Strip. *gdal2tiles for 16bit data*. http://osgeo-org.1560.x6.nabble.com/gdal-dev-gdal2tiles-for-16bit-data-td5163094.html#a5163098, 2014. Last Accessed: 24-05-2020.

United Nations Department of Economic and Social Affairs, **2019a**. United Nations Department of Economic and Social Affairs. *Total Population - Both Sexes*. https://population.un.org/wpp/Download/Standard/Population/, 2019. Last Accessed: 24-05-2020.

United Nations Department of Economic and Social Affairs, **2019b**. United Nations Department of Economic and Social Affairs. *World Population Prospects 2019 Highlights*. https://population.un.org/wpp/Publications/Files/WPP2019_Highlights.pdf, 2019. Last Accessed: 24-05-2020.

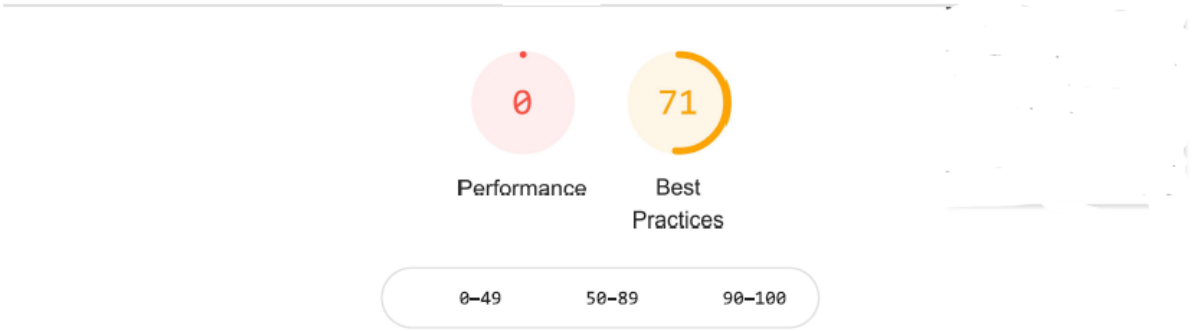
w3schools, **No date**. w3schools. *What is HTTP?* https://www.w3schools.com/whatis/whatis_http.asp, No date. Last Accessed: 02-06-2020.

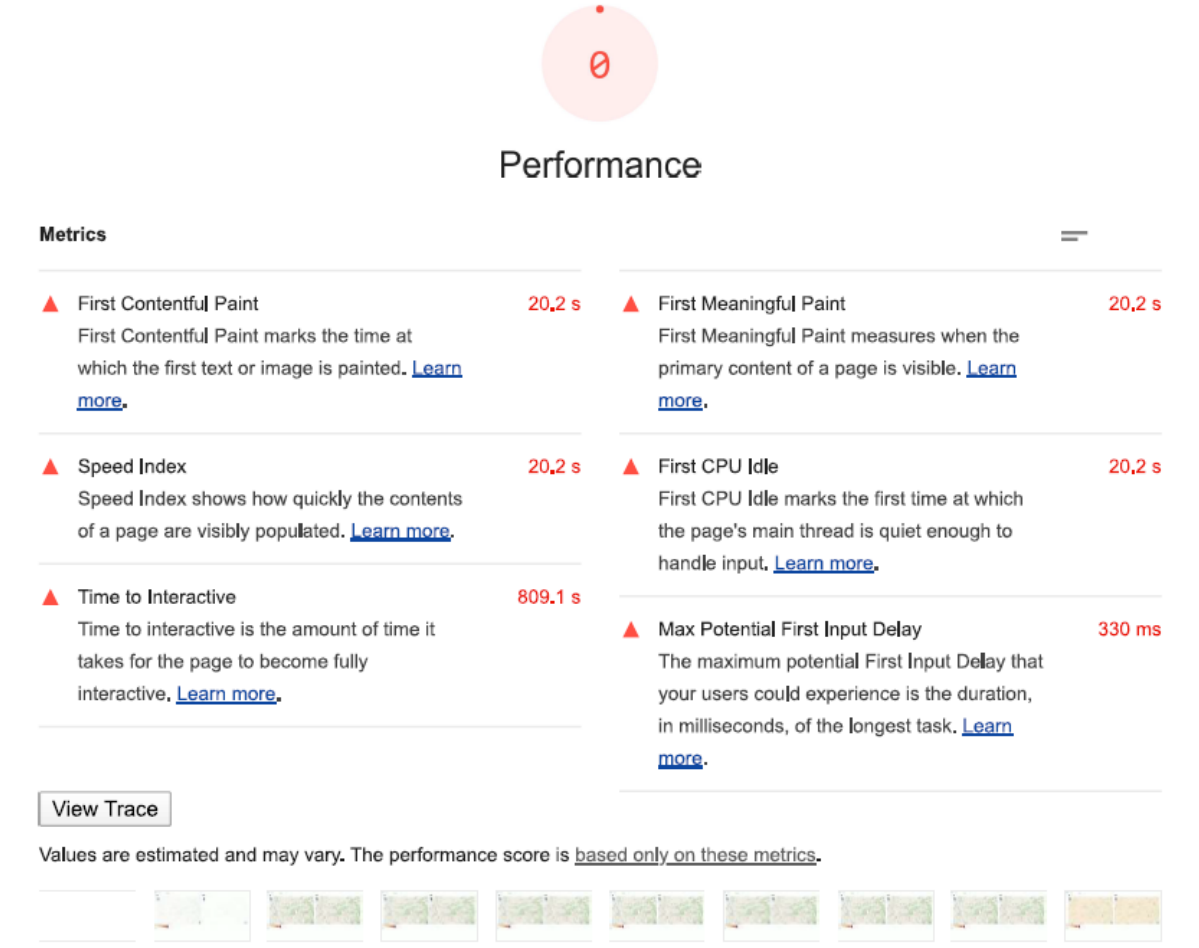
W3Schools, **No datea**. W3Schools. *JavaScript Tutorial*. https://www.w3schools.com/js/default.asp, No date. Last Accessed: 24-05-2020.

W3Schools, **No dateb**. W3Schools. *Python Introduction*. https://www.w3schools.com/python/python_intro.asp, No date. Last Accessed: 24-05-2020.

WebpageTest.org, **2013**. WebpageTest.org. *Speed Index*. https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index, 2013. Last Accessed: 27-05-2020.

Lighthouse audit A





Opportunity	Estimated Savings
▲ Eliminate render-blocking resources	17.71 s
▲ Enable text compression	16.8 s
▲ Minify JavaScript	6.75 s
Serve images in next-gen formats	0.6 s
Preconnect to required origins	0.3 s

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- ▲ Avoid enormous network payloads — Total size was 154,429 KB
- ▲ Serve static assets with an efficient cache policy — 17 resources found
- ▲ Minimize main-thread work — 17.5 s
- ▲ Ensure text remains visible during webfont load
- ▲ Reduce JavaScript execution time — 11.2 s
- Avoid chaining critical requests — 10 chains found
- Keep request counts low and transfer sizes small — 318 requests • 154,429 KB

Passed audits (12)

Best Practices

- ▲ Does not use HTTP/2 for all of its resources — 305 requests not served via HTTP/2
- ▲ Does not use passive listeners to improve scrolling performance
- ▲ Includes front-end JavaScript libraries with known security vulnerabilities — 1 vulnerability detected
- ▲ Uses deprecated APIs — 2 warnings found

Passed audits (11)

Fetch time	May 21, 2020, 11:38 AM GMT+2
Device	Emulated Desktop
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
User agent (network)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1483

Generated by **Lighthouse** 5.7.0 | File an issue

The code in the project B

B.1 HTML and CSS

```
1
2  <!DOCTYPE html>
3  <html>
4
5  <head>
6  <meta charset="utf-8" />
7  <link rel="shortcut icon" href="#" />
8  <title>maps from geotiff</title>
9
10 <script src="./lib/plotty.min.js"></script>
11 <script src="./lib/geotiff.browserify.js"></script>
12
13 <script src="./lib/olGeoTiff.js"></script>
14 <script src="./lib/ol-debug.js"></script>
15 <link rel="stylesheet" href="./lib/ol-debug.css">
16 <link rel="stylesheet"
       href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
17
18 <style>
19 body {
20 font-family: Arial, sans-serif;
21 font-size: 14px;
22 }
23
24 .map {
25 width: 400px;
26 height: 400px;
27 margin-top: 20px;
28 }
29
30 .mapcontainer {
31 padding: 0 100px;
32 text-align: center;
33 }
34
35 td {
36 border: 0;
37 margin: 0
38 }
39
40 td.l {
41 color: white;
42 text-align: center;
43 width: 30px;
44 height: 30px;
45 padding: 0
46 }
47
48 @media (min-width: 900px) {
49 .wrapper {
```

```
display: flex;
}

.half {
padding: 0 10px;
width: 400px;
float: left;
}
}
</style>
</head>

<body>
<input type="text" id="requestedCity" value="Search">
<button onclick="SearchCity()"><i class="fa fa-search"></i></button>
<div class="wrapper">
<div class="half">
<h4 id="firstMapName">Map1</h4>
<div id="firstmap" class="map"></div>
</div>
<div class="half">
<h4 id="secondMapName">Map2</h4>
<div id="secondMap" class="map"></div>
</div>
</div>
<!-- Add legend -->
<table>
<tr>
<td colspan="6">Population per cell</td>
</tr>
<tr>
<td class='l' id='d0' title='0'></td>
<td class='l' id='d1' title='0'></td>
<td class='l' id='d2' title='0'></td>
<td class='l' id='d3' title='0'></td>
<td class='l' id='d4' title='0'></td>
<td class='l' id='d5' title='0'></td>
</tr>
<tr>
<td colspan="4" style='text-align: left'>0</td>
<td></td>
<td id="MaxValue" colspan="4" style='text-align: right'>0</td>
</tr>
</table>

<script src="tileMetadata.js"></script>
<script src="map.js"></script>

</body>

</html>
```

Listing B.1: The HTML and CSS in the project

B.2 Javascript

```

////////////
// Getting Metadata
////////////


```

```
7
8  //The folder, where the tiles are extracted from.
9  var tileFolders = ['g2tTiles', 'g2tSecondMap']
10
11 //Add layer names to map
12 document.getElementById('firstMapName').innerHTML = tileFolders[0]
13 document.getElementById('secondMapName').innerHTML = tileFolders[1]
14 //Get tile-metadata produced by gdal2tiles32 and store it in the tileMetadata object.
15 var tileMetadata = {};
16 getTileMetadata(tileMetadata, tileFolders);
17
18 //Resolutions from the metadata xmlfile creates issues with loading the correct file if any
       zoomlayers are excluded
19 //Therefore resolutions tables are created manually.
20 var resolutions = new Array(14);
21 var matrixIds = new Array(14);
22 for (var z = 0; z < 14; ++z) {
23 // generate resolutions and matrixIds arrays for this WMTS
24 //The number in the resolution calculation is the units-per-pixel value at zoomlayer 0 in the xml
       file generated by gdal2tiles
25 resolutions[z] = 0.03333333333514 / Math.pow(2, z);
26 matrixIds[z] = z;
27 }
28
29 ////////////
30 // Creation of colorscale
31 ////////////
32
33 //Here a color scale is created and added to the available color scales
34 //The colors are selected in colorbrewer - https://colorbrewer2.org/#type=sequential&scheme=OrRd&n=6
35 var colorScale = {};
36 colorScale.color_steps = [
37 '#fef0d9',
38 '#fdd49e',
39 '#fdbb84',
40 '#fc8d59',
41 '#e34a33',
42 '#b30000'
43 ]
44 colorScale.percentage_steps = [
45 0,
46 0.2,
47 0.4,
48 0.6,
49 0.8,
50 1
51 ]
52 plotty.addColorScale("sequentialMultiHue6Colors", colorScale.color_steps, colorScale.percentage_steps);
53
54 ////////////
55 // Creating the map
56 ////////////
57
58 // Setting map projection
59 const projection = new ol.proj.get('EPSG:4326');
60
61 //Create the layers with the raster tiles. Some metadata is collected from the metadta object
62 var wmslayerMap1 = new ol.layer.Tile({
63 source: new ol.source.WMTS({
64 url: tileFolders[0] + '/{TileMatrix}/{TileCol}/{TileRow}.tiff',
65 projection: projection,
66 tileGrid: new ol.tilegrid.WMTS({
67 origin: tileMetadata[tileFolders[0] + "origin"],
68 resolutions: resolutions,
69 matrixIds: matrixIds,
70 tileSize: 256
```

```
}),
requestEncoding: 'REST',
transition: 0
}),
//The extent has been limited, since there I didn't test with the raster for the entire world
extent: tileMetadata[tileFolders[0] + "boundingBox"],
opacity: 0.65
});

var wmslayerMap2 = new ol.layer.Tile({
source: new ol.source.WMTS({
url: tileFolders[1] + '/{TileMatrix}/{TileCol}/{TileRow}.tiff',
projection: projection,
tileGrid: new ol.tilegrid.WMTS({
origin: tileMetadata[tileFolders[1] + "origin"],
resolutions: resolutions,
matrixIds: matrixIds,
tileSize: 256
}),
requestEncoding: 'REST',
transition: 0
}),
extent: tileMetadata[tileFolders[1] + "boundingBox"],
opacity: 0.65
});

// define the base layer
var osmSource = new ol.source.OSM();
var osm = new ol.layer.Tile({
source: osmSource
});

// This view is shared between both maps, so they always show the same
var sharedView = new ol.View({
projection,
center: tileMetadata[tileFolders[0] + "center"],
zoom: 7,
maxZoom: 11,
minZoom: 2
})

// define the left map
var map = new ol.Map({
target: 'firstmap',
layers: [
osm, wmslayerMap1
],
wrapDateLine: true,
view: sharedView
});

// define the right map
var map2 = new ol.Map({
target: 'secondMap',
layers: [
osm, wmslayerMap2
],
wrapDateLine: true,
view: sharedView
});



// olGeoTiff setup
var olgt_map1 = new olGeoTiff(wmslayerMap1);
var olgt_map2 = new olGeoTiff(wmslayerMap2);
```

```
olgt_map1.plotOptions.palette = 'sequentialMultiHue6Colors';
olgt_map2.plotOptions.palette = 'sequentialMultiHue6Colors';

// Color the maps based on their values
recolorMap()


//Add the colors from the color palette to the legend
for (i = 0; i < colorScale.percentage_steps.length; i++) {
document.getElementById('d' + String(i)).style.background = colorScale.color_steps[i]
}

//Recolor map on movement or zoom
map.on("moveend", function() {
recolorMap()
});


////////////
// Recoloring the map
////////////
// Find the highest value currently displayed and recolor based on this

//variable for holding the max value and checking if the max value changes
var currentMax = 0;
var oldMax = currentMax;

function recolorMap() {

//Array holding max values for all currently displayed tiles
var maxValuesAllTiles = [];

//Getting map extent and zoom
var mapExtent = map.getView().calculateExtent(map.getSize())
var mapZoom = map.getView().getZoom();

//This variable is adjusting for the fact that the wrong zoom level is being loaded
var zoomlevelAdjustment = 3

//The loadExtent is the same as the mapextent, unless the mapextent shows an area outside the data area
// In this case the loadExtent gets reduced to the bounding box -
// this is to avoid attempt at loading data, which doesn't exist
var loadExtent = new Array(4);
loadExtent[0] = Math.max(mapExtent[0], tileMetadata[tileFolders[0] + "boundingBox"][0]);
loadExtent[1] = Math.max(mapExtent[1], tileMetadata[tileFolders[0] + "boundingBox"][1])
loadExtent[2] = Math.min(mapExtent[2], tileMetadata[tileFolders[0] + "boundingBox"][2])
loadExtent[3] = Math.min(mapExtent[3], tileMetadata[tileFolders[0] + "boundingBox"][3])

//Get the total number of tiles - both maps have the same number of tiles, so no need to run this twice
var tileNumber = 0;
wmslayerMap1.getSource().getTileGrid().forEachTileCoord(loadExtent, mapZoom - zoomlevelAdjustment, function(tileCoord) {
tileNumber++;
})

//Counting variables keeping track of how many tiles have been processed in each map
var currentTile = {};
currentTile[tileFolders[0]] = 0;
currentTile[tileFolders[1]] = 0;

//Calculate the highest value in each map. The counting variables have been included to know when to draw the layer
findHighestValue(wmslayerMap2, tileFolders[1], tileFolders[0])
findHighestValue(wmslayerMap1, tileFolders[0], tileFolders[1])
```

```
199
200 function findHighestValue(wmslayer, selfCounter, otherCounter) {
201
202 //Getting the url (name) of the tile based on its coordinates
203 var tileUrlFunction = wmslayer.getSource().getTileUrlFunction()
204
205 //Checks which tiles that currently are being displayed
206 //This is done at a lower resolution than the current zoomlevel, since loading otherwise would be
      too slow
207 wmslayer.getSource().getTileGrid().forEachTileCoord(loadExtent, mapZoom - zoomlevelAdjustment,
      function(tileCoord) {
208
209 //Gets the name of each currently displayed tile
210 tileName = tileUrlFunction(tileCoord, ol.proj.get('EPSG:4326'))
211 asyncCall()
212 async function asyncCall() {
213
214 //Get the maximum value in the tile and add it to the array
215 tileMaxValue = await calculateMaxValue(tileName);
216 maxValuesAllTiles.push(tileMaxValue)
217 //Update the counter - one more tile have been processed
218 currentTile[selfCounter]++;
219
220 //Checks if the function is finished with finding max value for tiles in both layers
221 if (currentTile[selfCounter] == tileNumber && currentTile[otherCounter] == tileNumber && tileNumber
      != 0) {
222
223 //Resets all counters and set current max to the highest value
224 currentTile[selfCounter] = 0;
225 currentTile[otherCounter] = 0;
226 tileNumber = 0;
227 currentMax = Math.max(...maxValuesAllTiles)
228 //Recolor the map, if max value have changed
229 if (Number.isInteger(currentMax) && currentMax != oldMax) {
230 oldMax = currentMax
231 olgt_map1.redraw(olgt_map1, currentMax, colorScale);
232 olgt_map2.redraw(olgt_map2, currentMax, colorScale);
233 }
234 }
235
236 }
237 })
238
239 }
240
241 }
242
243
244
245 //This function pans to a searched city.
246 function SearchCity() {
247 //Get the name from the search bar and request the coordinates for it from Nominatim
248 var cityName = document.getElementById("requestedCity").value;
249 var request = "https://nominatim.openstreetmap.org/search?q=" + cityName + "&format=geojson"
250
251 var xhttp = new XMLHttpRequest();
252
253 xhttp.onreadystatechange = function() {
254 if (this.readyState == 4 && this.status == 200) {
255 var cityData = JSON.parse(this.responseText)
256 var cityCoordinates = cityData.features[0].geometry.coordinates
257
258 map.getView().setCenter(cityCoordinates)
259 map.getView().setZoom(9)
260
261 }
```

```
}
xhttp.open("GET", request, true);
xhttp.send();

}
```

Listing B.2: The JavaScript for the map

```
//Making a request for the tileData - see function getTileMetadata
function getTileMetadata(objectWithMetadata, folderArray) {
for (i = 0; i < folderArray.length; i++)
{

var xhttp = new XMLHttpRequest();
xhttp.open("GET", folderArray[i] + "/tilemapresource.xml", false);
xhttp.send();
processTileMetadata(xhttp, objectWithMetadata, folderArray[i]);

}
}

//When gdal2tiles32.py is run it creates a xml file with metadata. This function extracts the
    relevant data
function processTileMetadata(xml, objectWithMetadata, folderName) {
var xmlDoc = xml.responseXML;
var parser = new DOMParser();
var xmlDoc = parser.parseFromString(xml.responseText, "application/xml");

//Getting the coordinates for bounding box, origin and center
var minx = parseFloat(xmlDoc.getElementsByTagName("BoundingBox")[0].attributes.minx.value);
var maxx = parseFloat(xmlDoc.getElementsByTagName("BoundingBox")[0].attributes.maxx.value);
var miny = parseFloat(xmlDoc.getElementsByTagName("BoundingBox")[0].attributes.miny.value);
var maxy = parseFloat(xmlDoc.getElementsByTagName("BoundingBox")[0].attributes.maxy.value);
objectWithMetadata[folderName + "boundingBox"] = [minx, miny, maxx, maxy]
objectWithMetadata[folderName + "origin"] = [minx, maxy]
objectWithMetadata[folderName + "center"] = [
(minx + maxx) / 2,
(miny + maxy) / 2
]

}
```

Listing B.3: The JavaScript for getting Metadata about tiles

```

//Everything above is the same as the original olGeoTiff file

olGeoTiff.prototype.redraw = function(map, currentMax, legendValues) {

map.plotOptions.domain = [0, currentMax];
this.layer.getSource().refresh();
updateLegend(legendValues, currentMax);
}


// Updates the legend
function updateLegend(colorValues, maxValue) {
//Updates the number displaying the maximum value
document.getElementById('MaxValue').innerHTML = maxValue;
//Updates the values shown, when holding the mouse over the individual color classes
for (i = 0; i < colorValues.percentage_steps.length; i++) {
document.getElementById('d' + String(i)).title = Math.round(colorValues.percentage_steps[i] * maxValue)
}
}

```

```
// This object contains the maximum value for the loaded tiles data
var maxValueTileData = {};

//Returns the maximum value for a given tile. In the value does not exist it will be calculated and
    added to the object
function calculateMaxValue(url) {
//The value gets returned as a asynchronous function, since it otherwise would return undefined
return new Promise(resolve => {

//If the tiles have previously been loaded and stored in the object, then it is returned
if (maxValueTileData[url]) {
resolve(maxValueTileData[url].maxValue)
}
//If the tile is not in the object, then it gets calculated, added to the object and then returned
else {
maxValueTileData[url] = {
maxValue: 0 //Edited
};

// send new request
var xhr = new XMLHttpRequest();
xhr.open('GET', url, true);
xhr.responseType = 'arraybuffer';

// setup the async function that is executed AFTER the tile was loaded
xhr.onloadend = function(e) {
if (xhr.status == 200) {
// save rasters of parsed tiff
var parsed = GeoTIFF.parse(this.response);
var raster = parsed.getImage().readRasters();
// calculate the maximum value in the tile
var maxPop = Math.max(...raster[0])

//Add to the object and return the value
maxValueTileData[url].maxValue = maxPop
resolve(maxValueTileData[url].maxValue)
}
//
//      // send ajax request
}
xhr.send();

}

})
}
```

Listing B.4: Additions to olGeoTiff.js. The original code can be found at Baumrock [2020]