Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Mapping the Past

Semester Thesis

Marc Müller

marcmue@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Barbara Keller, Christian Decker,
Prof. Dr. Roger Wattenhofer

February 16, 2015

Abstract

We created an aerial image based web map, allowing the user to experience a travel through time from 1926 to 2007, consisting of the images from swisstopo's image collection, which were digitised and made publicly available. Swisstopo's image collection consists of more than 500,000 images, of which approximately 200,000 are aerial images used to update the Swiss National Map. We crawled for a subset of the images and their respective metadata through swisstopo's API, processed the raw images, and stitched them to a single image per year. These final images are then presented in an interactive web based map, such that the user can inspect the images year by year. As a Proof-of-Concept, we showed how one can combine the available images to create a fluid experience and provide the tools with which to expand this to all of Switzerland.

Contents

Chapter 1

Introduction

Over the last decades, the Swiss Federal Office of Topography, swisstopo, captured thousands of analogue aerial images of Switzerland; they used those images to create the renowned Swiss National Map. For years these images were securely stored by the federal Office of Topography in a collection consisting of approximately half a million raw images. In an effort to preserve this unique cultural treasure swisstopo decided in 2007 [1] to catalogue and digitize the entire inventory of images. So as a first step in 2014, 88 years after the first images were taken in 1926, they started to release digital copies of these images and make them publicly available. As of now, approximately 200,000 aerial photos listed in Table 1.1, captured as reference for updating the Swiss National Map, are digitally available.

Period	**Film Type**	**Aprx. no. of neg.**
1926 - 1954	Glass negatives	45,000
1946 - 1972	18 x 18 cm b/w negatives	44,000
1967 - 2003	23 x 23 cm b/w negatives	62,000
1998 - 2008	23 x 23 cm colour negatives	70,000

Table 1.1: A list of all the analogue aerial images from 1926 - 2008, the period in which a certain type of equipment was used and the approximate number of images that are available in the collection [2].

These images grant the rare opportunity to travel back in time and to observe a city or place change over a long period of time. Seeing this opportunity, we create a web based map to display these images in an interesting way to enable the user to experience a journey back in time and observe how Switzerland changed over all those years. Living in an ever-changing world, we often do not realize how fast everything evolves - how cities grow, buildings are removed,

whole airports are built, and rivers diverted. Combining these historic images allows us to get a glimpse of how Switzerland came to be what it is today.

In this project we build a web map containing the small subset of the photos released by swisstopo of the Zurich Airport, so we are able to see how the airport was built in the late 1940s and extended over the years with more terminals and runways. We start with creating our database containing the images' metadata and download the needed images. Then we process these images to make it possible to display them in a map-like way. This comprises cropping, editing, and stitching, the process of combining two partly overlapping images into one, so we finally acquire one single seamless image per year. In the end, we create a user friendly web map and position our finished images to enable a fluid experience watching the years fly by.

1.1 Related Work

In our work we compress a time period of nearly one hundred years into an effortlessly viewable container, and as such comparable to time-lapse video clips. Time-lapses exist for all different kinds of evolving objects: like plants [3], buildings [4], clouds [5], a moving crowd, and so on, these videos make it possible to see things evolve that are very hard to observe due to their inherently slow pace of change. Here we are not just creating a mere time-lapse of a part of Switzerland, rather we are creating an interactive map, where one can zoom to inspect certain areas, choose one year of interest, or even compare changes that happened between two not necessarily consecutive years. There is also an interesting map by swisstopo called Journey through time [6], it allows the user to view the old maps of Switzerland dating back to the year 1844. Unfortunately, cartographic data does not allow for the in depth experience we provide using actual images of the past. Plenty of work was done on image stitching and how it can be optimized for certain scenarios, see [7], [8], or [9].

With our work we provide the unique experience of time travel through Switzerland's 19th century.

1.2 Overview

Our goal for this project is to create a beautiful web based map, so one is able to select a certain year and see how Switzerland looked at that time. To show the reader how we were able to achieve our goal, we divided the problem into three parts. The data fetching part, where we will discuss how the data was acquired, the image processing part, and finally, the presentation. To conclude this report, we will have a brief discussion of what we achieved, and a short outlook of the next steps.

First, we have to find out what information we need, search for what information is available, and think about how to compensate for the difference. So in the first part we will discuss what information is needed, how we acquired them, what problems we came across and how we were able to solve or work around them. We will also have a look at the database and show some statistically interesting statistical data, and finally bring up some concerns about the amount of data with respect to storage capacity. We will also see here how we managed to download the images.

In the third chapter, we will discuss the problem of image processing; there we will come across different topics all related to image stitching. We will show that we were able to overcome the issues of our stitcher, talk about some possible issues with the way the images where taken, and also see how the downloading process influenced our stitcher. In this chapter, we will also come across the topic of the scaling properties of our stitching algorithm, and how we minimized the calculation time with help of pre known metadata. We will not go into detail on how the stitching process works, or how exactly the images should be pre-processed, as we will keep a logical approach to things.

The fourth chapter will then discuss how we presented our data. In this chapter, we will have a look into the system architecture, what software we used, and how we implemented the website itself.

Chapter 2

Data Fetching

The first step in the creation of the historical map is the acquisition of the necessary imagery. As all the images are geo referenced against the Swiss map projection called Swiss Grid [10], we have to first understand its difference from the global coordinate system. The Swiss Grid is a uniform map projection which was introduced in 1903; its fundamental point is the old observatory in Bern (compare with Figure 2.1). This grid allows us to reduce the complexity of properly positioning rectangular images on the map as we are able to think of earth's surface as a plane not a square. As such the Swiss Grid is a very helpful tool, since it introduces a rectangular coordinate system while the global coordinate system relies on angular longitude and latitude coordinates. The use of a rectangular system is only feasible due to Switzerland's small size compared with the surface of the entire earth.

2.1 The Images' Metadata

The previously described coordinate system reduces our problem to the point where only a pair of coordinates [y,x] per image is needed to properly position the image (compare with Figure 2.2). And since we would like to see the changes over time, some temporal information, a date or at least the year of captures, is required too. Swisstopo provides a public API through which, in addition to the needed information like the exact size of the image in pixels, the corner coordinates, and the height from the camera above sea level is available.

With all these constraints in mind, we are ready to fetch our needed data. Their API allowed us to efficiently gather all the metadata of each image. So we are able to create our database containing all the useful information of the images in question. Our database includes the data of 200,000 pictures, the center position in Swiss coordinates and GeoJSON [13], a format for encoding all kinds of geographic data structures using longitude and latitude values, the bounding box, the size in pixels, the height of the camera, the film type, the year of capture, and of course, the name of the image.

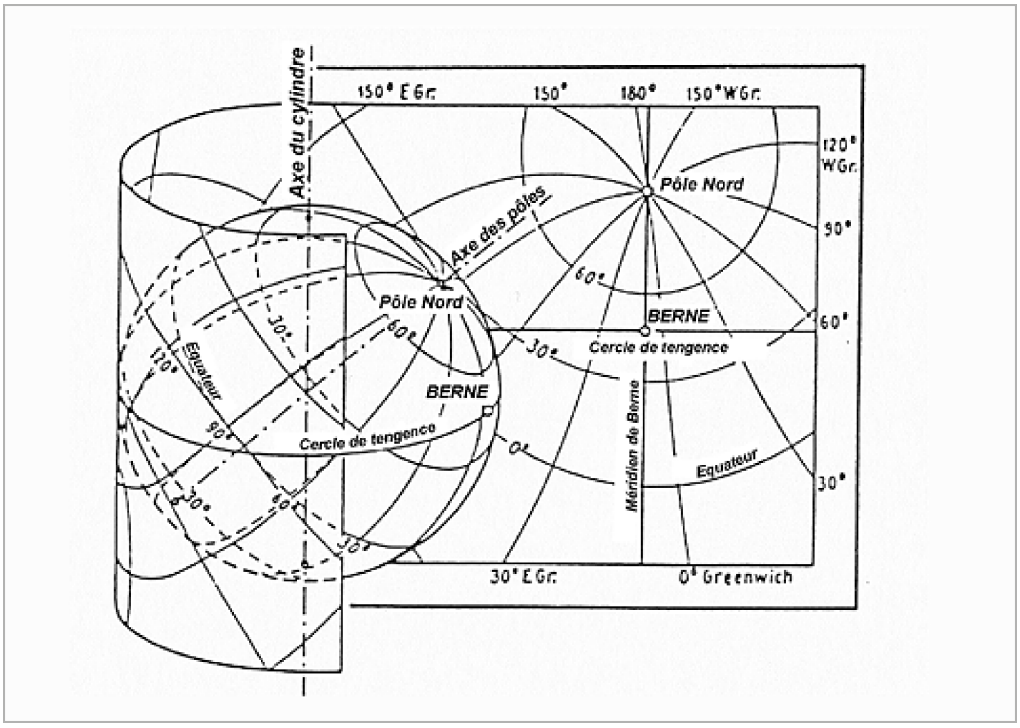


Figure 2.1: The Swiss Grid, an oblique, conformal cylinder projection, with the old observatory in Bern as the fundamental point [10].

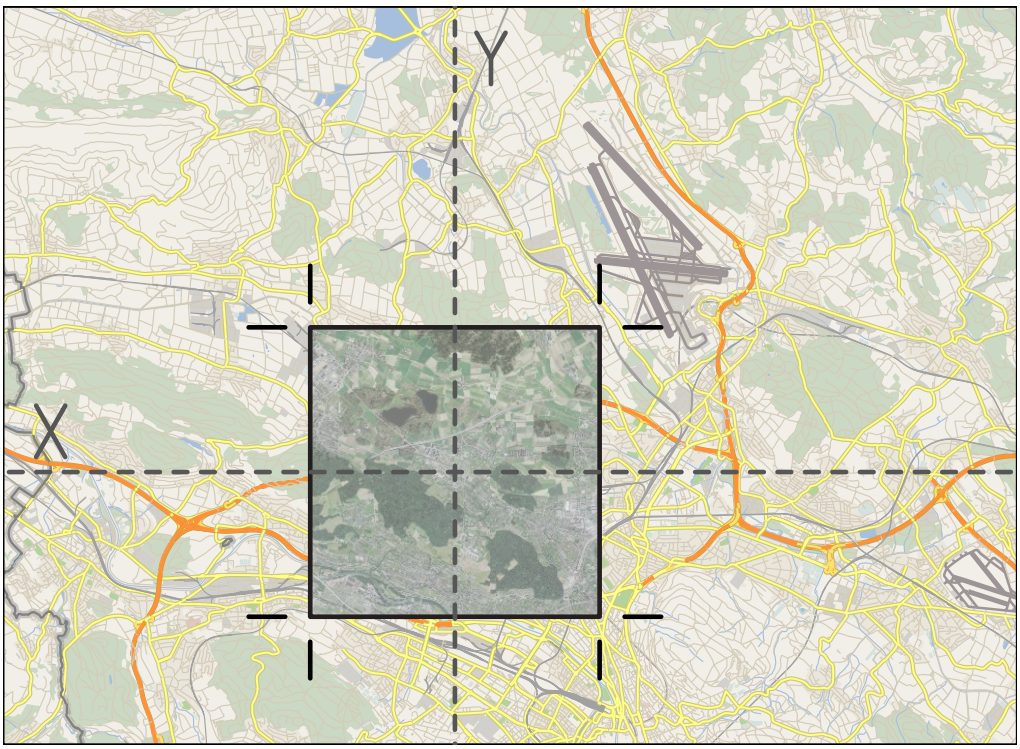


Figure 2.2: Rectangular images can be placed easily with a pair of rectangular coordinates [y,x]. Cartographic material from [11], image from [12].

In order to aggregate and collate all 200,000 datasets we had to crawl the raw data made available by the API and extract for us relevant data. Listing 2.1 shows one dataset from our database.


```
{
 "_id" : 19460200000035,
 "year" : 1946,
 "bbox" : [467964.439, 114032.85, 478738.111, 124691.866],
 "location" : [473591.7900390625, 119730.96992187575],
 "loc" : {
  "type" : "Point",
  "coordinates" : [6.406475616951898, 46.21715157190165]
 },
 "scale" : 50000,
 "size" : [16863, 16647],
 "originalSize" : [23, 23],
 "filmtype" : "bw",
 "hasImage" : true,
}
```


Listing 2.1: The dataset of an image as saved within our database.

2.1.1 Some Statistical Data

The total number of datasets is 235,077; these are all catalogued images, of which 173,623 have their respective photos actually digitised and publicly available. All these images are described by $3.12845 \cdot 10^{13}$ pixels. To save such a huge amount of data, one would need approximately 85 TB space when saved as JPEG with 24 bits/pixel, but to prevent quality loss these images are saved on the swisstopo's servers as TIFF, which results in more than 18 times bigger file sizes, totalling to about 1.5 PB.

To properly visualize the data distribution across all those years, we show in figure 2.3 the number of black and white images, the number of colour images and the mean resolution of the images per year.

It is interesting to see how the resolution increased over time, by comparing with Table 1.1 we see that with the adoption of new equipment between 1946 - 1954 and 1967 - 1972, and as such with the actual size of the original negative, corresponds with increased resolution. The cause of the peak in 1944 is anyhow for us incomprehensible.

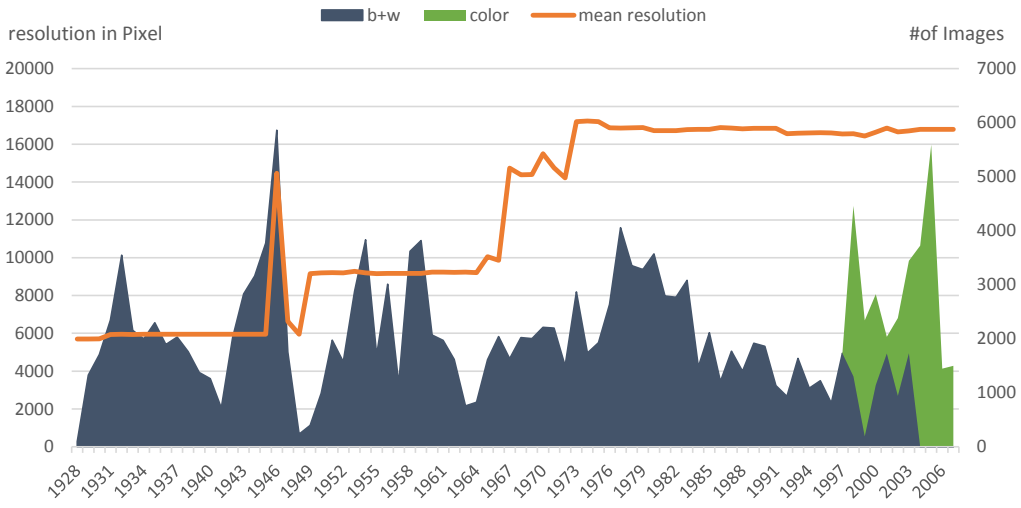


Figure 2.3: Visualizing the amount of available data: The mean resolution in pixel per year and the corresponding picture count.

2.2 The Images

The images themselves are displayed in the LUBIS viewer [14]. There the images are not just displayed as single images but through a so-called WMTS, a Web Map Tile Service [15]. Thus the images are split up into multiple smaller images with a fixed size of 256 x 256 pixels. For each image, multiple resolutions are available, and as the size of each sub images is constant, for a higher resolution more sub images become necessary (compare with figure 2.4. We call these sub images tiles, hence the name.

Figure 2.4: A Tile Server serves the images in multiple zoom levels, while keeping the resolution of all subimages constant, the number of images increases with the total resolution. Cartographic material from [11].

As for the task of downloading the full images, the following complications arise: we first need to determine how many tiles we need, then have to download all of them to finally combine them back together so we get a nice displayable image back. As trivial as it sounds the problem is slightly more complicated. The number of tiles used for one specific zoom level is determined based on the maximal available resolution by dividing the full resolution with the tile size:

$$\begin{bmatrix}\text{tileWidth}\\ \text{tileHeight}\end{bmatrix} = \left\lceil \begin{bmatrix}\text{pictureWidth}\\ \text{pictureHeight}\end{bmatrix} \cdot \frac{1}{\text{tileSize} \cdot 2^{\left(\left\lceil \log_2\left(\frac{\max[\text{pictureWidth},\text{pictureHeight}]}{\text{tileSize}}\right)\right\rceil - \text{zoom}\right)}} \right\rceil$$

To create the proper HTTP-requests we need to know the exact resolution of the image (pictureWidth and pictureHeight), to calculate the max resolution. Then we calculate which tiles would be empty, to minimize the overhead created and to not generate a multitude of error messages since these tiles are not available. And as such will be able reproduce the needed links to the given images properly.

For our web map we decided to download all images with a constant resolution; we chose the fourth zoom level, resulting in approximately 9×9 tiles or 2304×2304 pixels. This image resolution presents us with an additional trade-off: large images contain more details, but come at a larger transfer, storage and processing cost.

Chapter 3

Image Processing

As our goal is to display the images as smoothly as possible, we need to combine all images of a certain year into one big, seamless image. The process of doing so is called stitching and is common in digital photography. Image stitching is used for example to create panoramas, for high dynamic range (HDR) imagery or to generate high resolution images of the night sky.

For our project we face a slightly different problem than the common image stitching tools can handle. Most of the time, these tools implement predefined assumptions, for example a common point of view, on how the images where taken, and as our problem does not fit these standards we need to search for a more sophisticated solution. OpenCV, a Computer Vision library, is a toolkit comprising a multitude of tools for C++, C, Python and Java specialized for specific tasks, from advanced robotics over interactive art to image stitching. In addition it comes with a multitude of working example scripts that are free to use. OpenCV matches the requirements of this project, since it includes tools for image stitching and can be run in non-interactive batch jobs.

The exemplary stitching code included in OpenCV is sufficient for a Proof-of-Concept. Further optimization may provide faster stitching and improve the quality of the resulting image by being tailored to our scenario. We implemented a multitude of external optimizations to improve the whole processing.

3.1 Image Quality and Pre-Processing

A multitude of external factors can affect the performance of our stitcher. We can divide those factors into three groups: the way the images were initially captured, the downloading process, and the scaling properties of the problem at hand. In the following sections we will inspect these factors more closely.

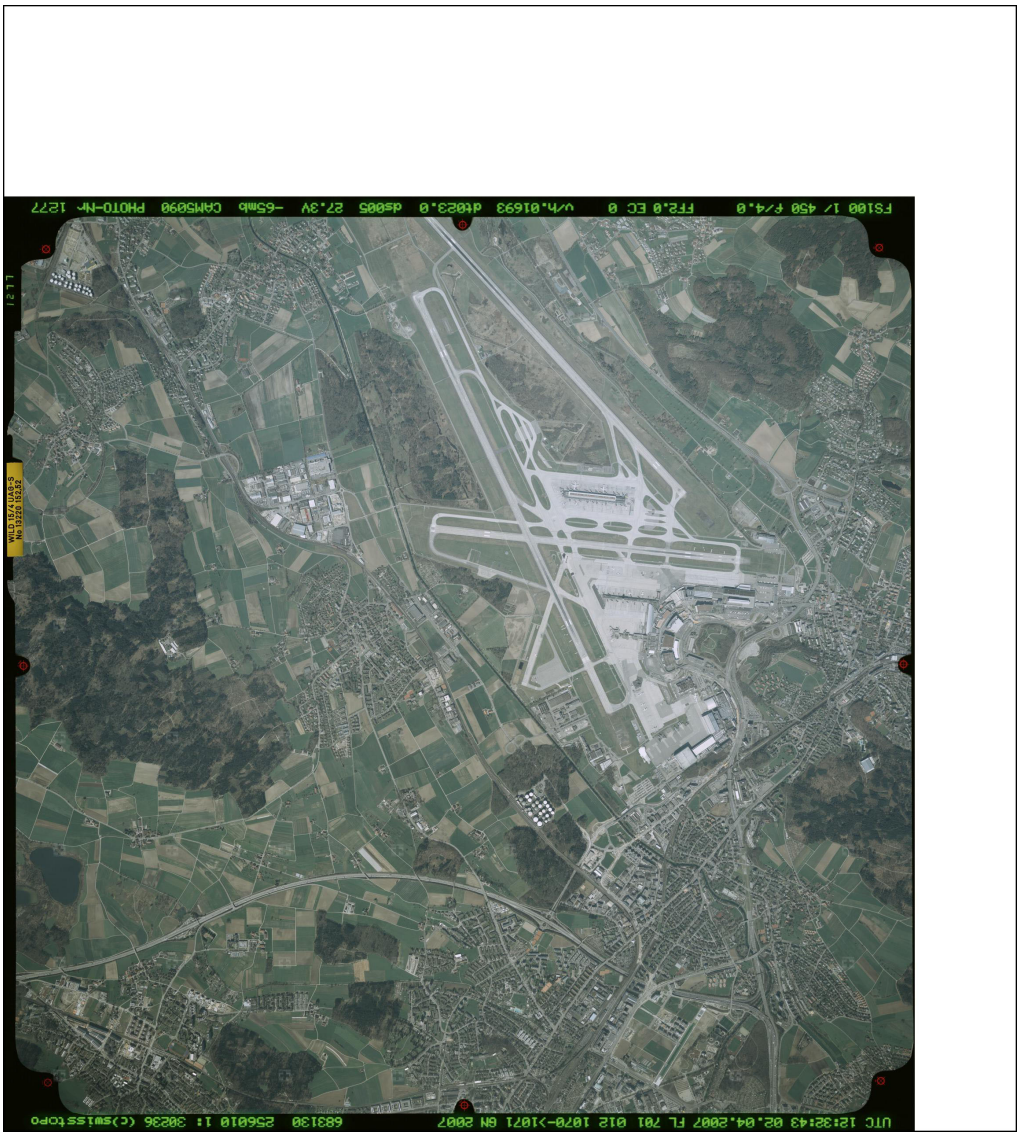


Figure 3.1: An image of the Zurich Airport in 2007 as downloaded; the white border is used to increase the size of the image to a multitude of 256 pixels. The black borders contain some useful metadata, but interfere with the stitching process. Image from [12].

3.1.1 Image Capturing

The images were captured from planes flying approximately 6,000 meters above sea level. This influences the images in three distinct ways: first there is some distortion created by the camera lens itself, secondly the occasionally improper alignment of the camera with respect to the Earth's surface and lastly there is the influence of the surfaces' relief.

Lens The distortion created by the camera lens itself is a well-known effect among photographers; it is also a good indicator for the quality of a lens. For long focal length lenses, over 50 mm, we see mostly pincushion distortion, for shorter lenses the opposite, barrel distortion. Pincushion distortion creates a ballooning effect of the image, in other words straight lines in an image tend to bend to the middle of the frame, and barrel distortion would create bending towards the edges of the image.

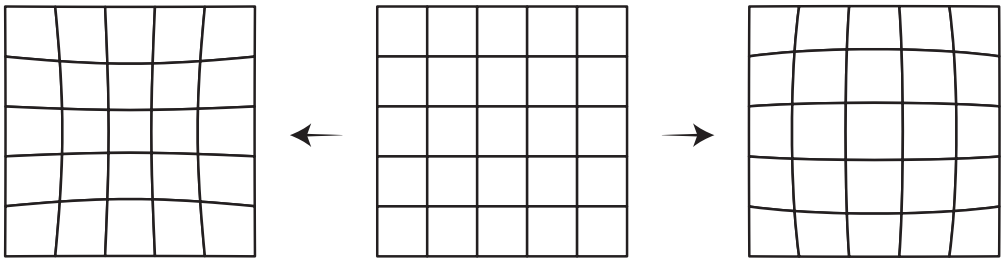


Figure 3.2: Pincushion vs. barrel distortion: while pincushion bends lines towards the middle of the frame, barrel distortion does bend them towards the edges.

Alignment As the images were taken from flying planes it can be assumed that the images were not always taken straight down but also with some slight angling. The area captured by photographing a flat surface at an angle is a trapezoid and not as rectangle.

Relief If one were to imagine a plane flying over a mountain range, the horizontal distance one could see would be further into a valley than at a mountain. This leads to a big distortion of the images, as we can see in Figure 3.4, the rectangle that the camera sees is actually not a rectangle on the relief seen in Figure 3.5.

To correct for these issues, a multitude of information is needed. To compensate for the lens distortion we need to know the distortion function specific to the lens used, but as we can expect that the equipment used here is of very high quality, we should be able to ignore it. Compensating for the alignment is impossible as the needed information is not available, so we have to assume

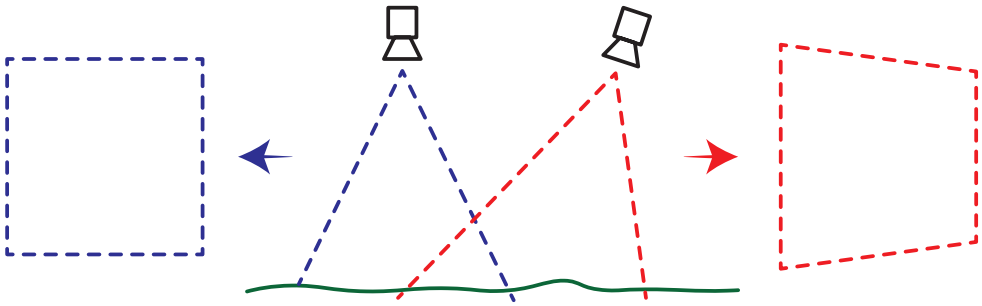
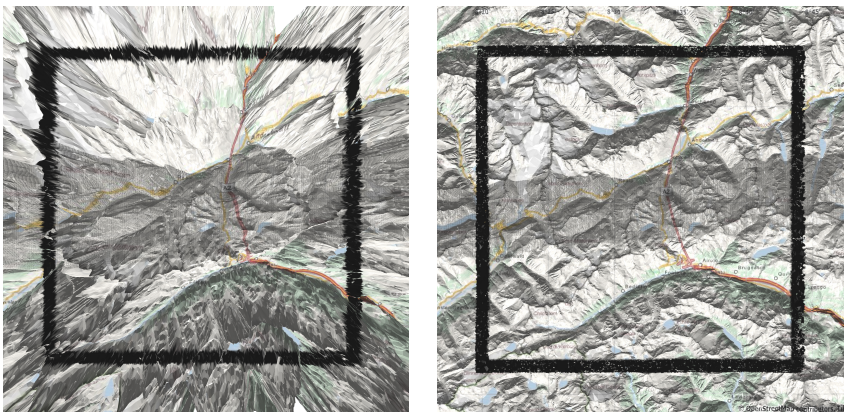
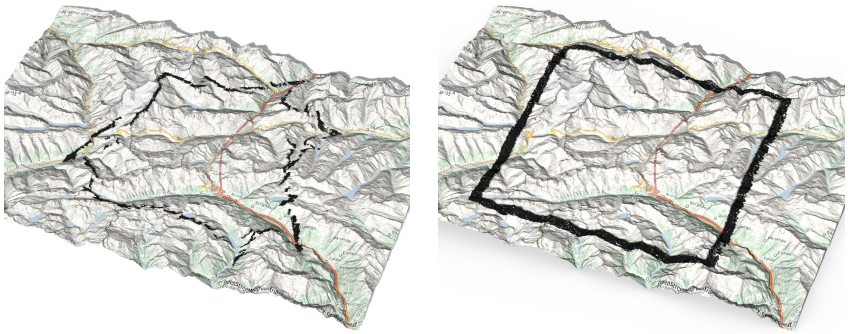


Figure 3.3: Trapezoid distortion created by photographing a flat surface from an angle. The blue lines would be a plane flying perfectly parallel while the red lines are not.



(a) With a wide-angle lens. (b) With an ultra-zoom lens.

Figure 3.4: Through the cameras lens the captured area looks like a square independently of focal length. Relief information and cartographic material from [11].



(a) With a wide-angle Lens. (b) With an ultra-zoom Lens.

Figure 3.5: Seeing the captured area as a third person one realises how the wide-angle lens distorts the image. Relief information and cartographic material from [11].

that the deviations are not big enough to cause a major problem. To remove the stretching introduced by the relief, one needs to know the relief in question to project the image back onto the relief and generate a virtual image taken with an infinite focal length. As the Zurich Airport is located in the flat areas of Switzerland the influence of the relief is negligible to the point where we were able to ignore it completely.

3.1.2 Image Downloading

As we see in Figure 3.1, the images also contain white and some black borders with very distinct patterns. The white border results from the tiled images, as these increase the pixel count to exactly 2304, a number divisible by 256, and thus can be cropped. The OpenCV stitching process searches for distinct areas in images, then compares these areas with the ones found in the other images to determine how the images should be stretched and placed on the final images. It can ignore single-color borders but not the one present in our images, as these pose perfect matching areas. As a result, the stitching process fails by returning a single black pixel. The solution for this problem is to crop the images so that only useful information is left (compare with Figure 3.6).

3.1.3 Scaling Properties

The stitching process itself has some very interesting scaling properties. We will discuss two ways through which we were able to reduce the calculation time significantly. First, the size of the source images, and second the algorithm to stitch the images in a recursive fashion.

(a) The stitching algorithm will not be able to process these images properly.

(b) The copped images can be stitched properly.

Figure 3.6: Comparison of the uncropped images and their cropped counterpart. While with the uncropped images the stitching process will fail, stitching the cropped images works perfectly fine. Images from [12].

Image Size The image size has quite an influence on the performance. Large images provide more data which leads to a more precise stitching result but increases the amount of pixels to process which does increase the calculation time. Reducing the image resolution decreases the time to calculate the final image but also makes it harder to find matching patterns as the overlapping number of pixels are reduced. As mentioned in Chapter 2, we download our images with a middle resolution of 2304^2 pixels which proves to be a good compromise between speed and quality.

Recursive Stitching As the function does not scale linear but super quadratic we implemented a recursive stitching algorithm that does start in a first step by combining small subsets of about nine images to one, and in further steps combines these sub images recursively so that we get one final image. With such a recursive implementation we were able to reduce the complexity of our algorithm close to a linear problem. For example, stitching 16 images takes $16^2 = 256$ time units. When stitching 4 times 4 images and then in a second step combining the four resulting images together we find that it takes only $4 \cdot 4^2 + 4^2 = 80$ time units.

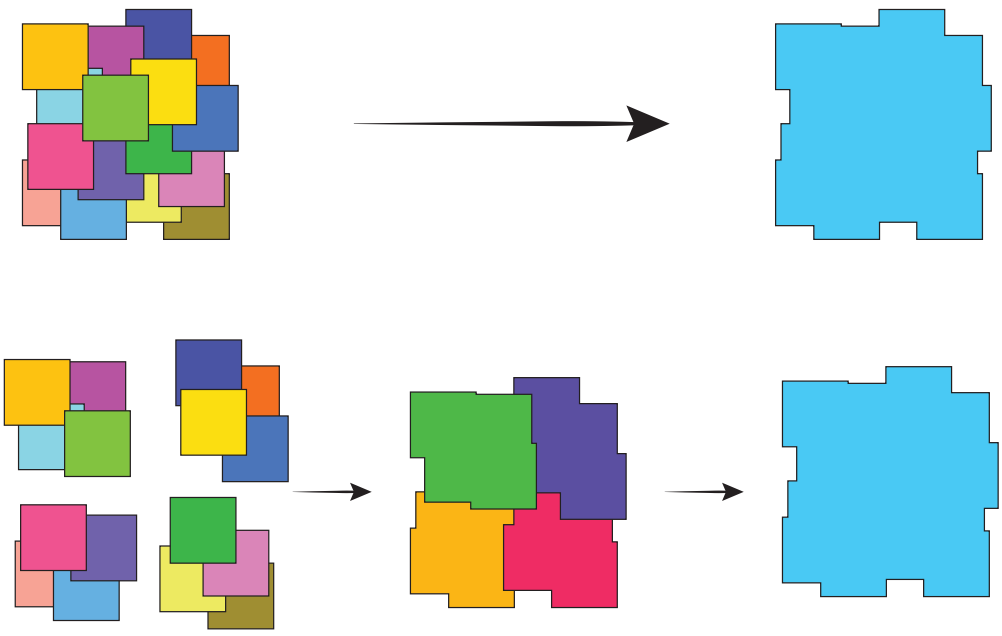


Figure 3.7: The recursive stitcher does indeed reduce the calculation time needed to create the final image by reducing the complexity of the problem.

To implement this, we use the available metadata to find overlapping images and were additionally able to ensure that the images stitch in one batch do actually overlap and as a result the stitcher spends less time searching for matching images. To find overlapping images, we calculate

Figure 3.8: As not all images overlap, it is not intelligent to start the stitcher with images of which we know in advance do not overlap. Images from [12].

the dimensions of the area mapped by a centre image according to its bonding box. Then we query the database for all images with their respective centre point within an area double the size in each direction that one image would occupy. This works since all the images of one year have the same extent. By finding overlapping bounding boxes, we could increase the accuracy of the results as we take the images rotation into account. One could even go further by using the rings information, the actual outer border of the images in Swiss coordinates, as this would also compensate for the relief distortion discussed in Section 3.1.1. Our algorithm then saves the names of these found images into a file, the name of which is determined by where the final image should be positioned on a grid. For the next step, we use these distinct names to find which images are adjacent and again save the names of the existing files into a next file until only one file remains, compare with 3.9. We then save these files according to their level into specific folders, so we could run the stitcher with the help of a batch script for all the files in one folder, reusing the filename as the name for the resulting image. Running the batch script for one level at a time, we generate our resulting final image.

3.2 Post-Processing

After downloading, cropping, and stitching the images, we now have one final image per year. Now we need to determine some metadata for each of these images, metadata such as the exact position, the area visible to properly scale the images, and the rotation of the images. Furthermore, it may be needed to calculate the final distortion to be able to display the images on top of a reference map.

Most of these metadata can be acquired by comparing the final images with a reference, which, in our case, is the map by swisstopo with the aerial imagery

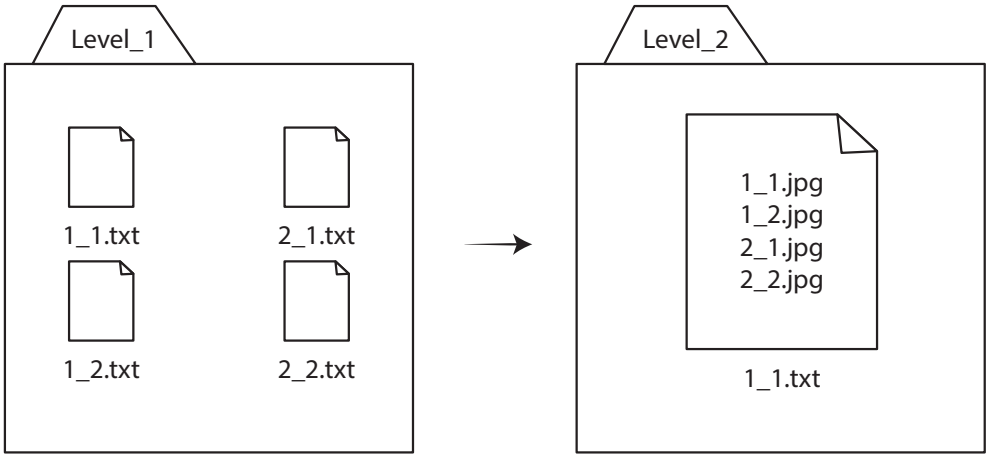


Figure 3.9: Finding overlapping images, saving their names in files, which then are saved in respective folders, enables us to stitch the images efficiently in a cascading fashion.

of 2014 [16]. First, we need to find the rotation of the image, and place the rotated image according to it onto a new canvas so that all images are aligned northward. Secondly, we can find the actual positioning data for each of these images. The easiest way to do so is to find the bottom-left and the top-right corner on the reference and save the found coordinates into our database. With these two points we can not only calculate the centre position of the image, but the respective scaling parameter as well. With these two steps we can at least position the images properly and are able to display the images nicely for a reasonably small area as the Zurich Airport. For larger areas, the cumulated distortion would result in a unpleasant experience, as the images would from year to year not overlay close enough, and further improvement will be inevitable.

To further improve the viewing experience, and to be able to display larger parts of Switzerland, it is also a good idea to serve the images in the way a WMTS does it. Improving the images means compensating for the stretching error resulting from the stitching process. As one can see in Figure 3.10, the images are not perfectly straight.

To compensate for the skewing present in the images, we need to select multiple points on the images and compare those with the reference map. Comparing the coordinates of each of these points from where they are on the image, and where they should be, will result in a transformation matrix. After applying this matrix onto the stitched image, the resulting image should match the reference. Logically, as more points per image are selected, the results become more accurate, but as always, calculation time will increase too.

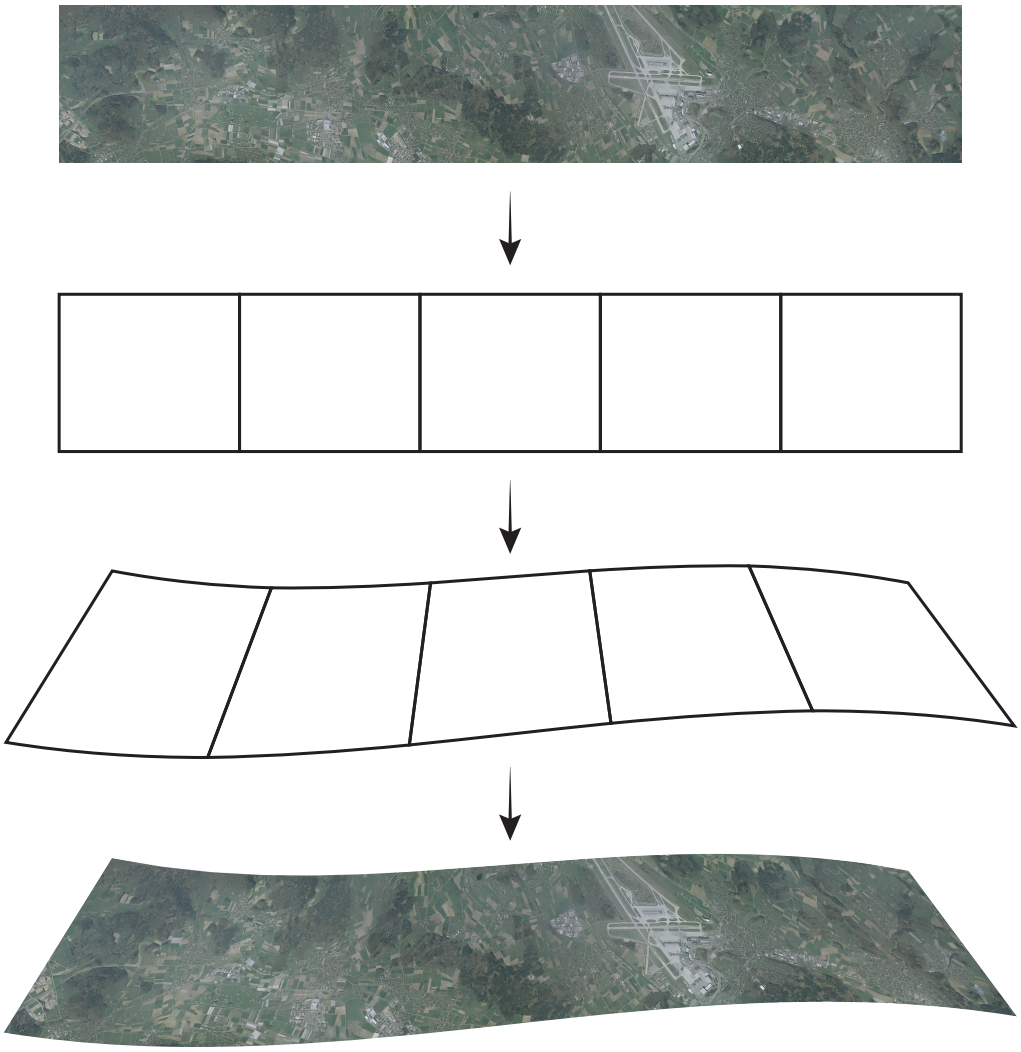


Figure 3.10: As the stitching algorithm has no information about the real position of the image, it will distort the image slightly. The more images are combined to a single image, the bigger the final distortion will be. Source images from [12].

Chapter 4

Presentation

As the final goal for this project is to display the images of the Zurich Airport as a web map, we have to set up a web server that serves our images to a website containing the web map itself. We will have a look at some technical details of the system.

4.1 Server

We choose to use Node.js as our backbone for the whole project [17]. Node.js is based on Google Chromes V8 JavaScript engine and allows the creation of all sorts of synchronous server side applications. One big plus for Node.js is the lightness with which it handles I/O heavy situations, and of course, it is convenient to use the same programing language for the front- and the backend. Furthermore, Node.js is, for the moment at least, broadly used in today's web due to its modular structure and expandability.

For our server we used Express on top of Node.js; Express, in the words of the creators, is "a fast, unopinionated, minimalist web framework for Node.js" [18]. As for the database, we chose MongoDB, a well-known NoSQL database [19], which interacts natively with JavaScript, and as such, nicely with Node.js. It uses documents and collections instead of rows in a table, and the connection is JSON based instead of SQL queries in a typical SQL database. We use the Node.js module Mongoose [20] to connect to the database. Mongoose is a persistence layer for JavaScript objects.

4.2 Website

The final component of the project is the website displaying the data. For the web map we make use of some of the newest web technologies. The images are placed in a window filling HTML5 Canvas (Compare with Figure 4.1), and automated with JavaScript. The server first looks up the years which have images to display

and then creates the site accordingly. Once loaded, the JavaScript dynamically sends asynchronous JavaScript and XML, AJAX, requests for the needed images and stores them in the browsers cache. The JavaScript then handles the position of the images according to user input. The user can slide and zoom the images, or select the desired years. Panning works by clicking and dragging, zooming either with a double click or with the mouse wheel, a single year is selected by clicking corresponding text, and multiple years with ctrl+click.

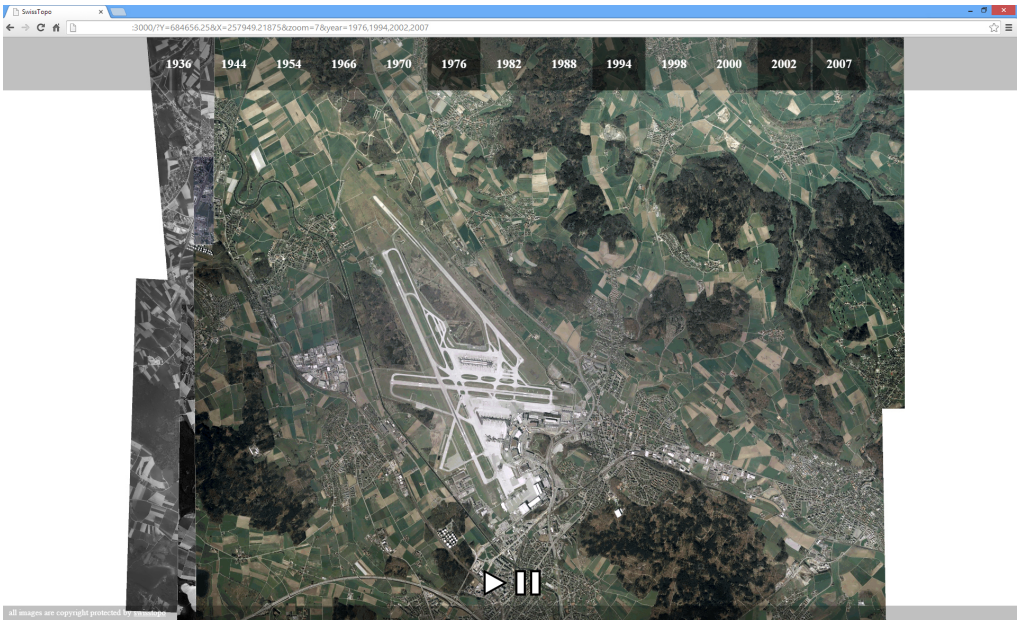


Figure 4.1: A screenshot of the final web map. Multiple years: 1976, 1994, 2002, and 2007, are currently selected.

Chapter 5

Conclusion and Outlook

We had this cool idea of letting the users experience Switzerland in a new way. After some research, we implemented our idea and were able to extract some very interesting facts about these images. We have seen how many images are available, and as such now understand the cultural value of swisstopo's image collection better. Despite all the problems we had to overcome, we can present a good looking web map combined of about two hundred raw images.

Creating such a historical map for larger areas of Switzerland is possible, but not trivial, as we now have the knowledge of how one could scale the project. We know how to download the images efficiently and have tackled the scaling properties of the stitching algorithm. We also know how to post-process the resulting images to undo the distortion created by the stitcher. So with enough time and computer resources, it would be possible to create this experience for all of Switzerland despite of 85 GB of raw image data.

Bibliography

[1] : swisstopo: Preservation and accessibility measures. http://www.swisstopo.admin.ch/internet/swisstopo/en/home/topics/geodata/historic_geodata/im_coll/pre_me.html Accessed: 12.02.2015.

[2] : swisstopo: Key data and feuatures. http://www.swisstopo.admin.ch/internet/swisstopo/en/home/topics/geodata/historic_geodata/im_coll/key_dat.html Accessed: 12.02.2015.

[3] Csobot, D.: Macro timelapse. http://vimeo.com/69225705 Accessed: 12.02.2015.

[4] Cury, B.: Official 9/11 memorial museum tribute in time-lapse 2004-2014. https://www.youtube.com/watch?v=V5teyjPeVco#t=57 Accessed: 12.02.2015.

[5] Black, S.: Adventure is calling. http://vimeo.com/76820114 Accessed: 12.02.2015.

[6] : swisstopo: Journey through time. http://map.geo.admin.ch/?layers=ch.swisstopo.zeitreihen Accessed: 12.02.2015.

[7] Brown, M., Lowe, D.: Automatic panoramic image stitching using invariant features. International Journal of Computer Vision **74**(1) (2007) 59–73

[8] Liu, Q., Liu, W., Zou, L., Wang, J., Liu, Y.: a New Approach to Fast Mosaic Uav Images. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences **3822** (September 2011) C271

[9] Xiang, R., Sun, M., Jiang, C., Liu, L., Zheng, H., Li, X.: A method of fast mosaic for massive uav images. Proc. SPIE **9260** (2014) 92603W–92603W–9

[10] : swisstopo: Swiss map projections. http://www.swisstopo.admin.ch/internet/swisstopo/en/home/topics/survey/sys/refsys/projections.html Accessed: 12.02.2015.

[11] : Openstreetmap. http://www.openstreetmap.org/copyright Accessed: 12.02.2015.

[12] http://www.swisstopo.admin.ch

[13] : Geojson. http://geojson.org/ Accessed: 12.02.2015.

[14] : Lubis-viewer. http://www.swisstopo.admin.ch/internet/swisstopo/en/home/apps/geodata_portal/lubis.html Accessed: 12.02.2015.

[15] : Opengis web map tile service implementation standard. http://www.opengeospatial.org/standards/wmts Accessed: 12.02.2015.

[16] : swisstopo: Aerial imagery. http://map.geo.admin.ch/?bgLayer=ch.swisstopo.swissimage Accessed: 12.02.2015.

[17] : Node.js. http://www.nodejs.org Accessed: 12.02.2015.

[18] : Express. http://www.expressjs.com Accessed: 12.02.2015.

[19] : mongodb. http://www.mongodb.org Accessed: 12.02.2015.

[20] : mongoose. http://www.mongoosejs.com Accessed: 12.02.2015.