VU VRIJE UNIVERSITEIT AMSTERDAM

Large Scale Data Engineering

Group 9 - Project L1

Sam Ansmink, Thijmen Kurk, Stan Swanborn

***Abstract*—*Context*: The Landsat program is the longest-running enterprise for acquisition of satellite imagery of Earth [18]. Over the years, the satellites have gathered millions of images of Earth. These images are open to the public for download to facilitate research. This download is available at an AWS S3 bucket[1].**

***Goal*: The assignment, as part of the course Large Scale Data Engineering at the Vrije Universiteit Amsterdam, was given to analyse, process and visualise this data in a viewer which would allow the user to select a specific geo-location and see an animation of the evolution of that piece of Earth over time. This paper elaborates on each of the steps taken to reach this goal and is considered the final report for this course.**

***Method*: Use available tools like Rio-Tiler [16] and existing industry standards like the Web Mercator projection [8] [4] [17] to process the Landsat data into usable Web Mercator tiles. These tiles, compressed to reduce them to a manageable size, can then be used in a responsive web application; the LSDE Earth Viewer.**

***Results*: The results of this paper and its research are threefold and entail: (1) a step-by-step guide on how to approach Landsat data and to prepare and process it against the industry-standard mapping (Web Mercator Tiles), (2) the complete data product with a replication package for researchers to recreate our work and (3) a modern and responsive web visualisation of processed data allowing for a fast and easy interpretation of the satellite data.**

I. Introduction

Millions of images of Earth have been gathered in the Landsat program as it is the longest-running enterprise for acquisition of satellite imagery of Earth. Many tools and techniques on how to process these images have thus been developed. As an assignment for the course Large Scale Data Engineering, at the Vrije Universiteit Amsterdam during the year 2019-2020 of the Master Computer Science, the task was given to process these images and display them in an application. The application needs to enable the user to select a specific location on Earth and show the evolution of that location over time.

A. Related work

Numerous applications in the Landsat community have already been developed, most of these applications offer similar, or a part of the, functionality as required for the assignment. The core functionality of this assignment; showing the evolution of Earth as an animation of showing multiple years of Landsat data consecutively, is not a unique contribution to the Landsat community. This aspect and other related work is further elaborated on in section II.

[1] https://registry.opendata.aws/landsat-8/

B. Research questions

While investigating the problem and setting up the roadmap for successfully completing the assignment, the research questions that needed to be answered were identified and described in section III. These answers were found during the project setup, section IV, and summarised in the conclusion; section VI.

C. Project setup

The Landsat-8 data set is used for this project. This data set consists of all the images captured by the Landsat-8 satellite which range between 2013 and 2019, totalling 1.7 million scenes as of writing this paper (October 2019) and continues to grow to this day. The focus for this project, in-line with the goal of the Large Scale Data Engineering course, has been on processing the vast amounts of data contained in the Landsat-8 data set. The project was completed in multiple steps, each step being described in detail in section IV.

D. Results

The result of this paper's efforts is twofold; firstly the data product, and secondly the visualisation. The data product consists of processed Landsat-8 data for the entire world at multiple granularities for the available years. The visualisation is called the LSDE Earth Viewer and will visualise the data product and allow the user to animate the land changes. This is further explained in section V.

II. Related work

Tools and theories for visualising, analysing and processing Landsat images have been developed by numerous individuals and organisations. These visualisations are used as inspiration for the visualisation resulting from this paper.

A. USGS Earth Explorer

The USGS Earth Explorer [5] is used by researchers to be able to view and download with Landsat data. The viewer itself is aged and very static, it can thus be concluded that the use of old technologies and synchronous loading should be avoided as it makes the visualisation feel dated and slow to use. It compromises usability and general user experience.

B. USGS LandsatLook

The USGS LandsatLook [6] visualisation is exactly what the assignment describes and performs the same functionality as that the LSDE Earth Viewer will need to be able to perform. The visualisation can show any Landsat satellite's images, from Landsat 1 to 8 (excluding 6 as it could not

reach orbit), perfectly mapped on an ESRI [1] map. Even though the visualisation would be a perfect match, in terms of functionality, as required for this paper's assignment, it is still considered an example of how it should not be done. This visualisation also uses synchronous loading, presenting the user with pop-ups stating it is loading or drawing the Landsat images on the map, causing the visualisation to be inoperable. These old, static technologies make for a generally horrid user experience, causing evermore emphasis on modern responsive web-technologies.

C. Google Earth Engine

The Google Earth Engine has timelapse [2] functionality where Landsat data is projected on Earth and the user is able to view the evolution over time from 1984 to 2018, with intervals of one year. The visualisation is exactly what this paper's assignment describes and the fact that it is provided by Google means that there will surely be valuable insights to gain. Even though this visualisation implements the correct functionality, this paper still seeks to improve on it. For one, the animation controls are too simplistic, only giving the user access to 0.25x, 0.5x and 1x, a more explicit control (for example an explicit interval in seconds) should be implemented in the LSDE Earth Viewer. This visualisation is the only one, from the ones mentioned in this section, that allows the user to view the animation of Earth smoothly and also while zooming in or out, up to the highest zoom level (close to Earth) and the lowest zoom levels (high above Earth). This aspect of this viewer will most definitely need to be in the LSDE Earth Viewer as it greatly improves the general user experience and the value of the data and the viewer.

D. Remote pixel viewer

The remote pixel viewer [13] was of great help during the exploratory phase. The viewer is a basic visualisation that allows the user to map any Landsat-8 image onto a selected tile. The viewer uses the Mapbox framework [12], which seems very suited for this job especially considering that the Mapbox website features a piece of documentation on processing satellite imagery[2].

Using the unfiltered data, as provided in this viewer, shows that cloud coverage needs to be at a minimum in the data product as it is concluded that this quickly reduces the value of the shown image. It is also concluded that showing raw Landsat data is not suitable for showing large chunks of Earth simultaneously as they would overlap and do not perfectly border each other. The Web Mercator projection [4] [8] [17] is therefore most definitely needed, this is further elaborated on in section IV-A.

III. Research Questions

The main goal of this project is to construct movies for every segment of Earth (at multiple granularities), and show how it evolves over time. This task also includes stitching together imagery of different "scenes" as well as efforts to clean imagery from clouds. In order to achieve this goal, four research questions have been drafted:

1) What is the most efficient way to access satellite imagery?
2) How can the satellite imagery be combined so that they form a coherent map (taking into account factors such as cloud coverage)?
3) Can the quality assurance layer provided with the satellite imagery be used to remove clouds?
4) What is a feasible way of storing the obtained imagery in a way suitable for visualisation?

The answers to these questions are investigated throughout the project setup (section IV). Finally the questions are answered in section VI.

IV. Project Setup

This section describes the different steps taken while realising the project.

A. Preliminary analysis

There are multiple sources from which satellite data can be obtained, however in the interest of time only the satellite imagery obtained from the Landsat-8 satellite is considered for this project. In figure 2 all the scenes captured by Landsat-8 (capturing scenes from 2013 to 2019) are displayed. A scene captured by the satellite is depicted as a green rectangle and consists of metadata, multiple bands and a quality assessment (QA) layer [3].

The metadata contains information such as the exact coordinates of where the scene was captured (latitude and longitude) and the percentage of cloud coverage inside of the scene [3].

A band is an image representing light from a certain wavelength. For this paper only the red, green and blue bands (RGB) are used, since these are the bands required to create colored images. An example of a colored image composed from these bands is shown in figure 1.

The QA layer can be used to identify the pixels that exhibit adverse instrument, atmospheric, or surficial conditions such as snow or clouds [3].

The Landsat-8 data set contains about 1.7 million scenes, the total size of the combined RGB bands equals to approximately 243TB of data (one band is estimated to be around to 50 MB). A CSV file is used to index these scenes. The index file contains the location of where the scenes are captured in the form of two coordinates, the minimum and maximum point of the scene. Additionally, the index file contains the percentage of cloud coverage per scene, an unique identifier and the date of acquisition.

Some scenes had a cloud coverage percentage of -1, this means that the cloud detection algorithm, ran during the processing of the scene by NASA, failed. Scenes with no cloud coverage percentage were removed, since this metric is required for the scene selection (as described in section IV-B). This resulted in 0.8% $(= 14139)$ of the total scenes being dropped.

In order to create a two dimensional map, a projection is required which projects the spherical Earth onto a two dimensional plane (the computer screen). Choosing a map projection will have consequences on the properties of your map. There exists a wide variety of projection techniques all with their own advantages and disadvantages. For example there are projections that preserve area, projections that preserve distance, or more exotically, projections preserving shortest route. Projections typically have only one or two of these properties as they are often mutually exclusive.

Choosing a projection for the project turned out to be an easy choice since only one projection is to be considered a standard for displaying maps in an interactive way in a browser, the Web Mercator projection [8] [4] [7].

[2]https://docs.mapbox.com/help/tutorials/processing-satellite-imagery/

[3]https://docs.opendata.aws/landsat-pds/readme.html

Fig. 1. Example scene ($LC08\_L1TP\_201023\_20130717\_20170503\_01\_T1$)

The Web Mercator projection is often combined with a tiling system to minimise the amount of data required to render a map view. The tiling system works by dividing a map into tiles at different zoom levels. Zoom level 0 contains the whole world in a single tile, zoom level 1 divides zoom level 0 tile into four tiles, etc. see figure 3 for a visualisation. Each tile is generally 256 by 256 pixels in size served by a web server to a client that requests tiles for a specific view of the map.

To generalise; each Mercator tile is identified with x, y, z coordinates, has four children with $z + 1$, and given that $z > 0$ has one parent with $z - 1$. In order to generate a single coherent map from the satellite imagery each Mercator tile is composed out of 0 or more scenes in a process named mosaicing. While for some Mercator tiles there are no scenes available, for example the ocean tiles, for others there are 1900+ scenes available, averaging at around 165 scenes per tile.

To kick off the mosaicing process, a base zoom level was chosen. This zoom level is the zoom level on which the scenes are mapped to the Mercator tiles, this is further elaborated in section IV-B3. Zoom level 7 was found to be the ideal base zoom level since this level is the closest to the average size of the scenes. A higher zoom level results in the same scene being mapped to a lot of Mercator tiles causing a lot of duplication, while a lower zoom level results in a lot of scenes mapped onto a single Mercator tile causing a very high memory usage when mosaicing the scenes together.

After determining the base zoom level a max zoom level was chosen, which is the highest zoom level that will be generated. The Landsat-8 imagery has a resolution of 30 meters per pixel which falls right in between zoom level 12 and 13 with a respective resolution of 38.2185 and 19.1093 meters per pixel. However to save total storage costs and because subjective differences between the zoom levels 11, 12, and 13, were very small, max zoom level 11 was chosen.

To perform the actual mosaicing of scenes the tool rio-tiler-mosaic [15] is used. A naive approach would be to simply run this tool on every tile of the world with all the scenes available for this tile. However, this approach is infeasible since (1) it does not take the quality of the scenes into account (a scene can be fully covered in clouds), (2) the mosaicing process is a very I/O intensive process due to the need to download each scene and store it in memory and (3) the tool becomes unstable possibly due to reason (2). How the selection of scenes is done is described in section IV-B.

Since the goal of the project is to create animations of how the land changes during time, it was necessary to define a $time\_unit$ which is used to group the scenes. The $time\_unit$ was assigned to be in years. Any smaller $time\_unit$ such as months would result in too much data, while a larger $time\_unit$ such as 5 years would result in too little.

B. Preparing the scenes

It was determined during the preliminary analysis that the Web Mercator projection is the standard when it comes to the grid system of digital world maps. This step covers the ingestion of all the available satellite scenes from the Landsat-8 satellite and processing them in such a way that every Mercator tile of the world is fully covered with the least number of scenes necessary. This process has to take into account cloud coverage (scenes with the least amount of cloud coverage should be used first) and scene coverage (which is the percentage of overlap of the scenes on the Mercator tile, the highest scene coverage should be used first).

1) Downloading the scene list and aggregating it with additional metadata: The index file contains the minimum and maximum coordinates of a scene and the percentage of cloud coverage, both being vital information for the proper execution of the scene selection process. However, during the execution of this step it was uncovered that these two coordinates do not take into account the rotation of a scene, which is caused by the satellite not traversing Earth over the prime meridian line. This would make the scene selection algorithm, described in section IV-B4, less accurate when nearing the pools.

To resolve this, additional metadata was downloaded from the Landsat-8 Bulk Metadata Service and merged with the index file. The Bulk Metadata Service allows users to download all the metadata available for the satellite scenes inside of a CSV file (1.3 GB in size). The exact coordinates for each corner of the scenes could be extracted from the metadata which greatly improved the accuracy of the scene selection algorithm. Unfortunately, not all the scenes inside of the index file are covered by the metadata, this resulted in 0.9% $(= 15817)$ of the total scenes being dropped.

Additionally, the min and max coordinates had to be recalculated, since some edge cases did not conform to the newly obtained corner coordinates. The min and max coordinates are required to easily determine the overlap between scenes and Mercator tiles, which is discussed in section IV-B3.

2) Resolving the anti meridian line problem: During the analysis of the initial data, large scenes of size 180+ in terms of longitude were discovered, initially nothing was done with these scenes. In hindsight, this caused quite some confusion during the preparation of the scenes on a world scale. Since as apparent from figure 4, one of these large scenes, marked in blue, ran straight through the Netherlands, the place on which the scene selection algorithm was tested the most.

The large scenes were a product of scenes crossing the anti meridian line, this line marks where the two dimensional plane wraps around Earth. To resolve this problem an algorithm was created that splits all the tiles, crossing the anti meridian line, into two. This resulted in a tile on the left and on the right of the two dimensional plane. The results of this algorithm are shown in figure 4 and marked in yellow. The algorithm had to take into account the possible rotations of each scene, making it less than trivial to implement.

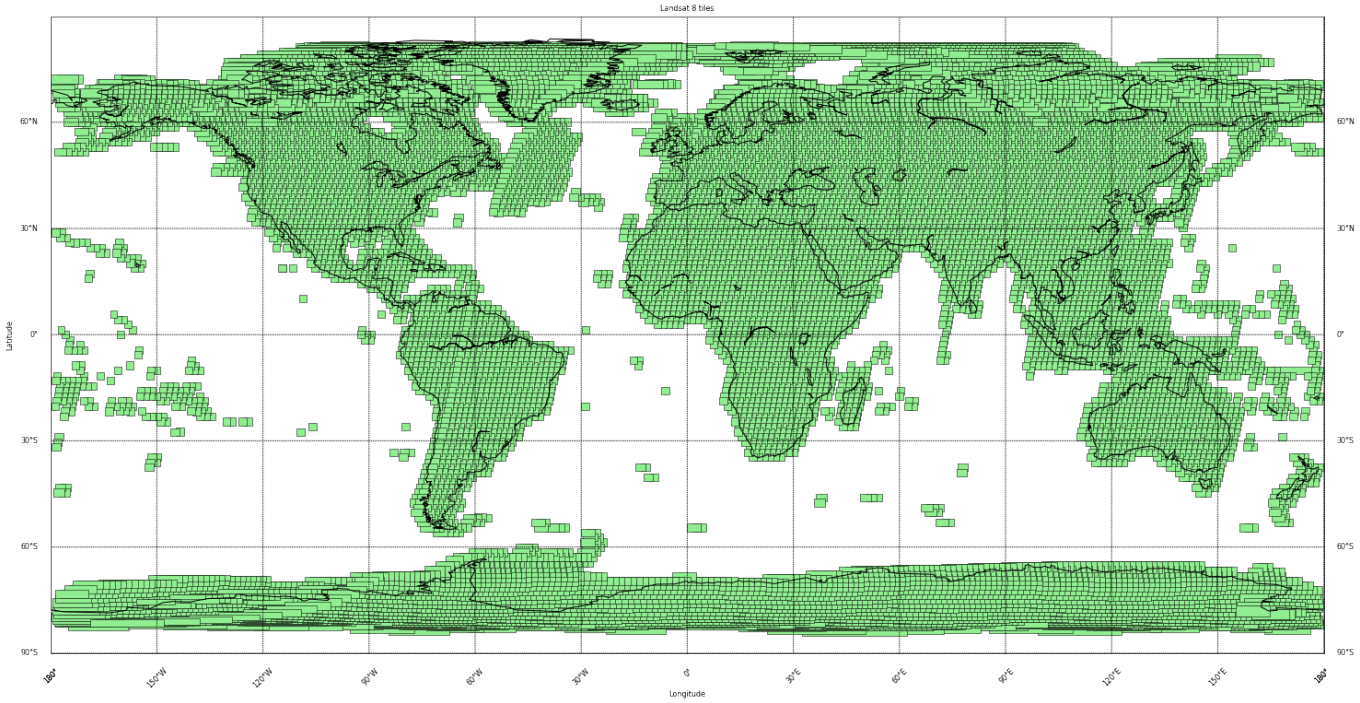

Fig. 2. Landsat-8 - all available scenes (2013 to 2019).

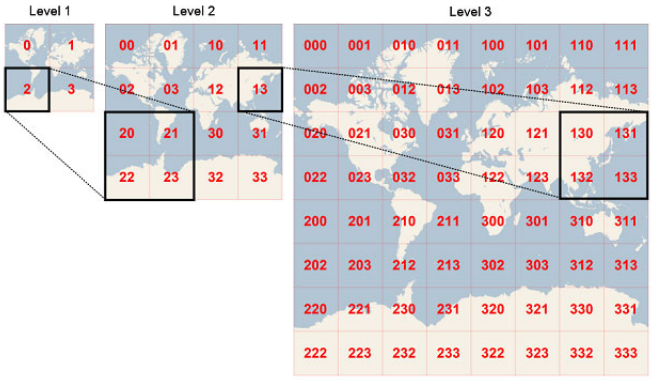

Fig. 3. The Web Mercator projection method visualised [17].

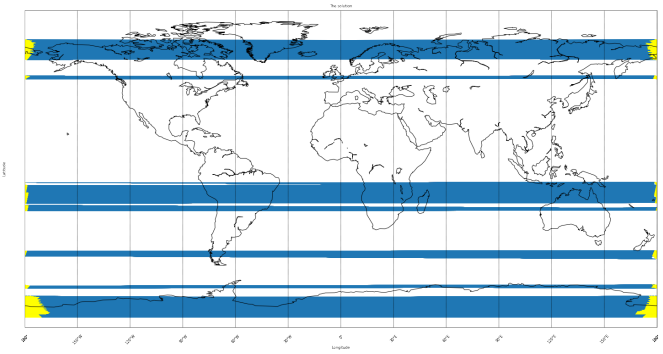

Fig. 4. Large tiles visualised (the problem tiles are marked blue and the same but corrected tiles are marked yellow)

The algorithm utilizes the Shapely[4] library, which is a library that is used for set-theoretic analysis and manipulation of planar features using functions from the well known and widely deployed GEOS library.

Now every step of the algorithm is described, it is recognized that this algorithm utilizes some 'hacky' properties of rounding:

1) The scenes with $max\_lontigude - min\_longitude > 180$ are selected for processing.
2) For each scene s that is to be processed:
   a) $180 \cdot 2$ is added to the negative longitude coordinates of the corners of the scene, causing the scene to fold to the right.
   b) The scene is cut by two lines: the $left\_line = [(180, -90), (180, 90)]$ and the $right\_line = [(180.000000001, -90), (180.000000001, 90)]$, resulting in a set of polygons called p. The scene is skipped when $p.length < 3$.
   c) The polygon with the smallest area is removed from p (which is the small rectangle in between the $left\_line$ and $right\_line$). This results in the scene being split into two.
   d) $180 \cdot 2$ is subtracted from the coordinates of corners with $longitude > 180$, and a rounding of 5 decimals is applied (which conforms to the number of decimals provided for the coordinates of scenes by the metadata).
   e) The minimum and maximum coordinates of the two parts of the scene in p are recalculated.
   f) Both parts of the scene are saved as actual scenes.

This algorithm resulted in 5749 scenes being added to the scene list.

3) Linking the scenes to Mercator tiles: Before the scene selection it is necessary to determine what scene overlaps with what Mercator tile and by what percentage.

To implement this rather trivial algorithm, the libraries Mercantile[5] and Shapely are used. First it is determined what Mercator tiles lay within each extreme point of a scene, the min and max coordinates. Then using linear extrapolation, the inner tiles are determined. Furthermore, the scene coverage

[4]https://github.com/Toblerity/Shapely

[5]https://github.com/mapbox/mercantile

percentage is calculated per covered Mercator tile by the scene.

This algorithm inflated the index file from 1.7 million scenes to a file containing 8.8 million overlaps of scenes on different Mercator tiles.

4) Selecting scenes to mosaic inside of a Mercator tile: Selecting scenes in such a way that they fill one Mercator tile lays at the core of this project, since this algorithm basically determines what scenes are shown inside of the final data product and in which order. Additionally it is our primary and only defense against cloud covered tiles.

The following properties of scenes are taken into account during selection:

- The cloud coverage percentage of the scene (this should be as low as possible);
- The scene coverage percentage (this should be as high as possible).

The goal of the algorithm is to find a balance between the quality of scenes in terms of cloud and scene coverage and the number of scenes required to fill a Mercator tile. This balance is important since the mosaicing time per tile (greatly) increases when it is created out of more scenes. Additionally, the tool used to download the scenes (rio-tiler) creates small artifacts inside of scenes due to resampling, unfortunately no configuration of rio-tiler was found that resolve these artifacts. However, these artifacts do become less apparent at our max zoom level of 11.

Now every step of the algorithm is described.

1) Splitting the cloud coverage percentage into groups with an interval of 5%.
2) The scenes are sorted by cloud group (ascending), scene coverage (descending) and cloud coverage (ascending). Sorting the scenes in such a way creates a balance between cloud coverage and scene coverage. This balance can be demonstrated per example: consider two scenes, the first with 3% cloud coverage and 85% scene coverage and the second with 1% cloud coverage but only has 10% scene coverage. After this sort, the first scene is selected, even though the second scene has a lower cloud coverage, since both tiles are inside of the same cloud group.
3) The scenes are grouped per $time\_unit$ and Mercator tile, resulting in a scene list per Mercator tile per $time\_unit$ containing candidate scenes for selection.
4) For every $time\_unit$ and Mercator tile m with $candidate\_scene\_list$:
    a) Polygon x is defined to track how much area is being consumed by the selected scenes, starting at 0.
    b) The list $selected\_scene\_list$ is defined to track the selected scenes for m.
    c) For every scene s in $candidate\_scene\_list$:
        i) Polygon s is constructed from the intersection between m and s, note that for s the corner coordinates obtained from the metadata are used.
        ii) The scene s is added to $selected\_scene\_list$ when $unary\_union(x, s).area > x.area$. Additionally the tile coverage added by selecting s is calculated and stored, which is done using the following formula: $(unary\_union(x, s).area - x.area)/m.area$.
        iii) Break when $m.area == x.area$.

The total number of unique scenes selected by this algorithm equals to 335483 scenes, which is 20% from the total available scenes.

While the algorithm is quite successful in selecting the proper scenes, and thus hiding most of the clouds, it is not perfect. Further options of cloud removal are explored in section IV-C4.

5) Aggregate selected scenes per tile: The final step is the aggregation of the selected scenes per Mercator tile which results in the following format, where $selected\_scenes$ is an array of strings containing the unique identifier of each selected scene in the order in which they are to be mosaiced:

$$time\_unit, x, y, z, selected\_scenes \quad (1)$$

The data is written to a CSV file assets.csv with the format above and stored in AWS S3.

To further reduce the number of scenes, a last filter is applied onto the tile coverage calculated by the selection algorithm. A scene is to be included when it covers $> 0.5\%$ of a Mercator tile (excluding scenes contributing only a few pixels to a tile). This leaves only 32% (= 167470) of the scenes selected by the scene selection algorithm.

From the results it is determined that the total number of world tiles that can be covered by the Landsat-8 data set equal to 52940 for all the time units (2013 to 2019), averaging around 7563 tiles per year. This average is deceptive since it was found out that starting from 2017, a lot of new scene capture locations have been added. This is clearly depicted in figure 5, where dark green represents scenes present in both 2016 and 2017, light green marks scenes only present in 2017 and red marks scenes only present in 2016.

To conclude, the selection algorithm reduced the average scenes per Mercator tile from 165 to 10, relative to the naive approach mentioned in section IV-A. Additionally it also improved the scene quality by mimicking the mosaicing process, taking into account the different quality metrics. This means that only around 5% (= 93164) of the total scenes are used in this project, which is approximately 13 GB of data (only considering the RGB bands at 50 MB per band).

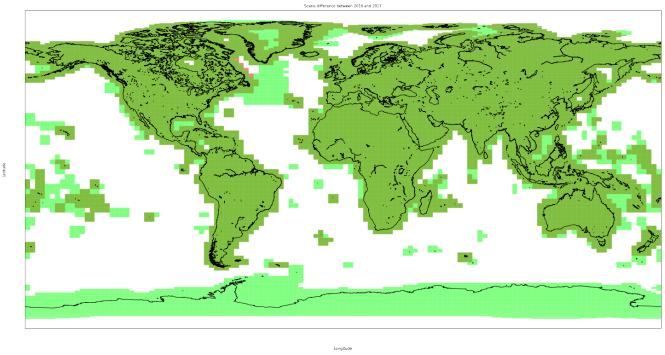

Fig. 5. The difference between scenes available in 2016 compared to 2017.

C. Render the tiles

1) Mosaic the tiles found by the preparing scenes step: With the assets file complete, the rendering step has all the required resources to generate the tiles. For each row, the render worker will use rio-tiler-mosaic [15] to combine the list of scenes into a single Mercator tile at the predetermined zoom level. The rio-tiler-mosaic uses GDAL [9] to stitch the different scenes together.

An important thing to note is that the mosaicer does the tiling process in an incremental way: it adds scenes to the resulting tile one by one until the mosaiced tile is completely filled or the mosaicer runs out of scenes. This means that mosaicing a tile with 100 scenes of which the first completely covers up the tile will be relatively fast since not all tiles need to be downloaded. However, mosaicing a tile with 100 scenes that, together, do not cover the tile completely will take significantly longer since the mosaicer will have to download and add every scene. Another note worth taking is that since every row can be processed completely independently by a worker, the workload is perfectly parallel.

2) Mosaicing performance & parallelisation: Initially early benchmarks and time estimations indicated that running the mosaicing worker multi-threaded on a single, or at most a few, large AWS instances would suffice for our workload. However after running into serious stability issues both with the external tiling libraries and the DBFS layer of Databricks, a switch to an Apache Spark based implementation was made. To parallel the work, the assets.csv was parsed and transformed to a Spark Resilient Distributed Dataset (RDD) and repartitioned to allow Spark to distribute the work across 24 AWS i3.xlarge instances.

3) Color correction: Simply combining the satellite imagery into an RGB image is not sufficient to generate a photorealistic image. To achieve such an image, a color correction profile can be provided to GDAL by the rio-tiler-mosaic. For the data product a default color correction profile, provided by rio-tiler-mosaic was used.

However to achieve a uniform map such as the Sentinel-2 Cloudless map[6], another technique called histogram matching [10] should be applied. In this process adjacent tiles are color graded to create a smooth color transition between the lines separating the scenes. This can be achieved through tools such as OSSIM [7]. Unfortunately no color matching was done due to time constraints.

4) Cloud removal: As mentioned in section IV-B4, simply relying on ordering by cloud coverage percentage is not enough to achieve a completely cloud-free. .

One way of achieving a cloud free map is using the QA layer, which contains the exact pixels which are believed to be clouds. Using this information, the mosaicer can make the parts containing clouds transparent, these parts are then filled by the next scene. It was attempted to perform this type of cloud removal, however it introduced holes inside of the tile due to the low number of scenes (which can be attributed to the selection algorithm). It was therefore decided to drop this method of cloud removal to prevent changes in the selection algorithm since the data product resulting from this algorithm is already quite good.

Depending on other map requirements another approach to get cloudless maps is to actually remove clouds with a technique called Cloud-GAN [19].

5) Storing finished base images: Each row in assets.csv corresponds to a Mercator tile (x, y, z) where z is the base zoom level. The images at this zoom level are referred to as base images. The resolution of the base images can be calculated with:

$$base\_resolution = tile\_size \cdot (2^{max\_zoom - base\_zoom})) \quad (2)$$

[6]https://s2maps.eu/

[7]https://trac.osgeo.org/ossim/wiki/histogram$_o perations$

For our settings of $base\_zoom$ 7 and max_zoom 11 this results in base images of 4096x4096 pixels.

Now to be able to display the base images as a tiled web map, other zoom levels need to be generated from the base images. For tiles at max_zoom this can be achieved by simply cutting out the correct part from the base images since the base images are already at the right resolution for these tiles. For the zoom levels, between $base\_zoom$ and max_zoom, this is achieved by cutting out the correct part of the base image and then downsampling it to the standard tile resolution of 256x256 pixels. For the tiles at $base\_zoom$ this is done by downsampling the base images to 256x256 pixels. For the remaining zoom levels below $base\_zoom$ each tile is downsampled from its four children at $zoom\_level + 1$ and then downsampled to 256x256 pixels.

This process is split into two steps:

1) Create tiles at zoom levels $base\_zoom$ up to and including max_zoom.
2) Create the remaining zoom levels 0 up to $base\_zoom$

In step 1 the same assets.csv is used as a task list. Since each row in assets.csv corresponds to one base image, each of these tasks consists of creating 341 images from the base image. With z_m as max_zoom and z_b as $base\_zoom$ the following formula gives us the number of images generated per task in this step:

$$\sum_{i=0}^{z_m - z_b} 4^i = \sum_{i=0}^{4} 4^i = 341 \text{ images} \quad (3)$$

For each task only one base image needs to be downloaded and can be kept in memory for all 341 tiles. For step 2 a task list is generated and iterated over using the following algorithm:

1) Create an empty dictionary to map tiles to their parent.
2) Map all $base\_zoom$ tiles in assets.csv to their parent tile in a dictionary.
3) Add mappings from the parent tiles in the dictionary to their respective parents.
4) Repeat step 3 until zoom level 0 is reached.

The resulting task list is of format:

$$time\_unit, x, y, z, parent_x, parent_y, parent_z \quad (4)$$

First the tasks are grouped by parent tile, this will result in groups of between one and four mappings from tiles to the same parent. These groups are grouped in descending order by z. These groups cannot be executed in parallel since the result of the lower zoom level groups depend on the results of the higher zoom level groups. The tasks within each group can be paralleled since they all operate in the same zoom level.

6) Storage format & Compression: Initially the plan was to use the Cloud Optimised GeoTiff Format [14] to store the finished product. This format is made for the sole purpose of efficiently storing map imagery in the cloud. Its main addition to the regular GeoTiff format is its internal tiling and embedded overviews. This allows for a client to request only a part of the GeoTiff through the HTTP1.1 Accept-Ranges: bytes header. This results in less duplicated data while maintaining relatively fast access for a frontend consuming the data. However, due to the assignment requiring a static website visualising the data, the switch to storing PNG tiles was made.

D. Visualizing the data product

To visualize the data a simple, Mapbox based visualization was made. It is a statically hosted frontend based on Mapbox JS [12] and fetches the tiles as rendered PNG's from an AWS S3 bucket.

The features include:

- A selector box to select the currently displayed year.
- A search box to search and display a specific city.
- A backward, play/pause and forward button to control the animation.
- A selector to select a specific interval (display time).
- A toggle to enable the compare mode.
- The compare mode
  - Two selector boxes on both the left and right side of the screen where the user can select two years to compare.
  - A bar to drag across the screen to view either the left or right selected year.

See figures 9, 10 and 11 for screenshots of the frontend visualising changes in various locations. The compare functionality can be seen in figure 6.

V. Results

The result from the steps mentioned in section IV are presented inside of the frontend. The initial state of the frontend is shown in figure 7, demonstrating the result of the downsampling from the base zoom level to zoom level 0. The figure summarizes the entire project. Four problems become immediately apparent from this overview:

1) Areas of the world where snow is common in the winter do not look proper due to the selection algorithm not being aware of snow inside of these scenes.
   A solution for this lays inside of the QA layer provided with each scene. This layer includes the location and amount of snow in a scene. The next iteration of the selection algorithm could take this information into account.
   Another solution would be to set the $time\_unit$ to a month. This is only viable if there is a bigger budget, since this multiplies the total size of the data product and the required processing time by 12.
2) Some years contain missing tiles. This is due to; (1) some scenes not being accessible and (2) the tool (rio-tiler-mosaic) used for mosaicing crashing for certain tiles.
   A solution for this would be forking rio-tiler-mosaic, and investigating what exactly is going wrong. An additional solution would be prechecking if the scenes provided inside of the metadata actually exist.
3) Some tiles contain artifacts. These artifacts seem to be present in other similar projects hinting at the usage of similar tools (such as GDAL, which is used by rio-tiler-mosaic).
   Take for example a project from the European Space Agency (ESA) in which they rendered a cloudless Africa using imagery from Sentinel-2[8]. Artifacts found in our project look quite similar to the artifacts found in the imagery created by the ESA (shown in figure 8). This seems to be an issue with the GDAL tool itself, and thus out of scope for this project.

[8]http://www.esa.int/ESA_Multimedia/Images/2016/05/African_mosaic

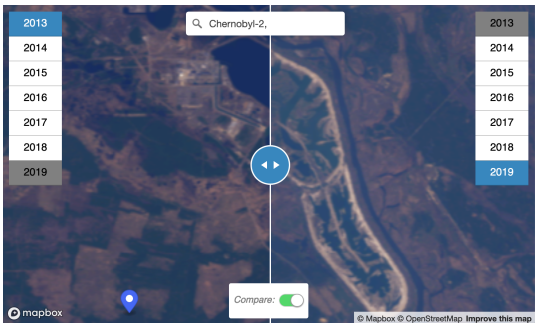

Fig. 6. The compare view of the frontend

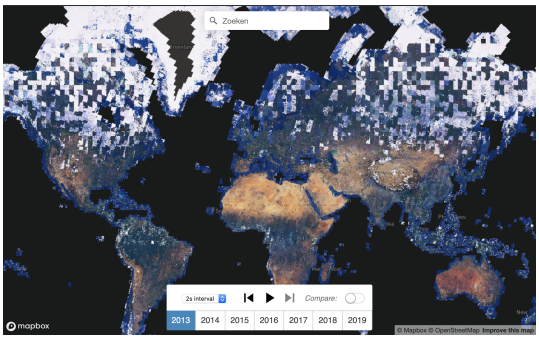

Fig. 7. The initial state of the frontend

4) Some tiles are still covered with clouds. Further exploration of cloud removal using the QA layer, as described in section IV-C4, is required to resolve this problem.

The final size of our data product is around 18 million PNG images, displaying Earth over the spawn of 7 years in multiple granularities (from zoom level 0 to 11). All these images are accessible straight from S3, providing a fast load time.

Some interesting examples demonstrating differences on Earth between various years are:

- Marker Wadden being build in the Netherlands, shown in figure 9.
- The construction of the Benban Solar Park in Egypt shown in figure 10.
- The construction of one of the world's longest bridges created near Kuwait City shown in figure 11.

VI. Conclusions

From this project the following conclusions regarding the research questions, as defined in section III, can be drawn.

A. The most efficient way to access satellite imagery

When using satellite data the biggest challenge is the large amount of data. For a project with satellite data with the ambition to render the whole world, minimising the amount of data used is crucial. In this project this was primarily achieved by an extensive filtering step minimising the amount of scenes while still maintaining sufficient coverage. By using the GeoTiff format, combined with industry-standard libraries such as GDAL, the total data transfer is greatly reduced by allowing partial downloading of specific parts and/or bands of images.

B. Combining satellite imagery into a single coherent map

Combining separate satellite scenes into a single coherent map is one of the fundamental problems in the field of satellite imaging. Papers on creating mosaiced Mercator projections go as far back as the 1960s [11]. While technology has come

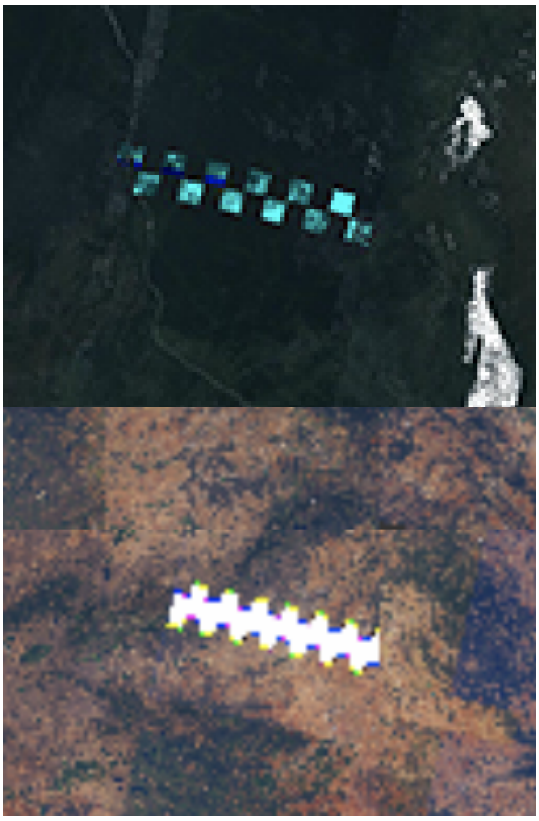

Fig. 8. The top image is an artifact found inside of the cloudless image of Africa created by ESA. The bottom image is an artifact inside of Spain in the year of 2017 found in our project.

a long way since then, the problem is still relevant to this day as new technologies in this field are still being actively developed [19]. The main problems with mosaicing can be divided into two categories:

1) Selecting cloud and snow free pixels: As is clear from the results of this project, snow and cloud can have a significant impact on the quality of a mosaic. To counter this problem there are three categories of solutions available:

1) Scene selection.
2) Selecting only cloud/snow free pixels with tools like GDAL [9].
3) Cloud removal algorithms such as Cloud-GAN [19].

For this project only the first technique was used. This project clearly shows that when grouping Landsat-8 data by year this technique is very effective at removing most of the clouds from the mosaic. However to achieve an even lower cloud coverage in the mosaic, or in the case that less data is available, other techniques are most likely necessary to get sufficient quality mosaics.

2) Color correcting individual scenes: Even though implementing a solution was considered out of scope for this project due to time constraints, this project and earlier work show that to achieve a sufficient mosaic quality both color correcting individual tiles and color matching adjacent tiles is required. Depending on map requirements, selecting scenes from the time period, and especially from the same season, will strongly influence mosaic quality.

C. Feasible ways of storing satellite imagery for visualisation

With the current price of storage, storing the large amount of data is relatively cheap and simple. For tiled web maps the easiest approach is to store every Mercator tile as a PNG on a

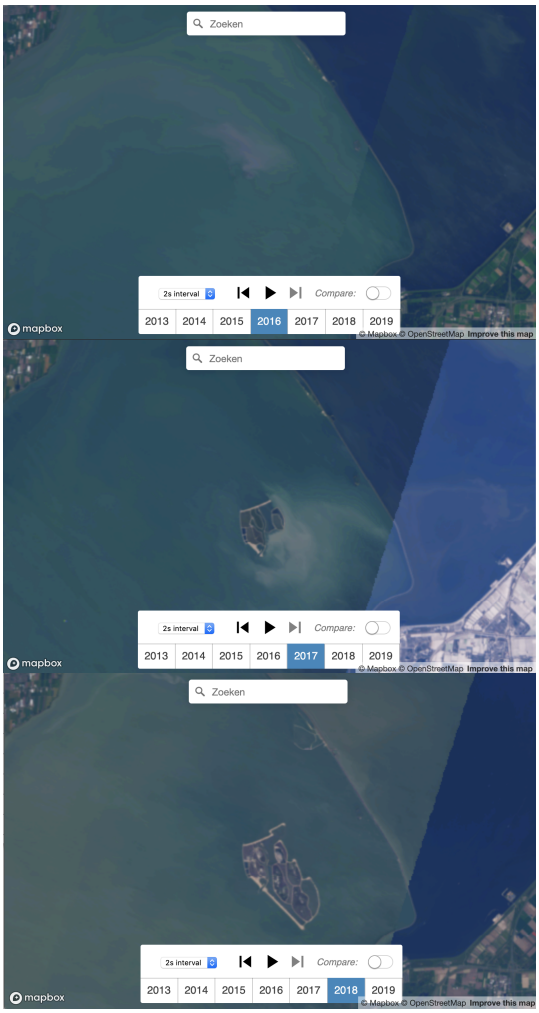

Fig. 9. The development of the Marker Wadden in the Netherlands visualised using our project.

cloud storage server. This, combined with a Content Distribution Network (CDN) makes creating a scalable satellite map service almost trivial. Another, more flexible, approach would be to store the tiles as Cloud Optmised GeoTiffs and use a tiling server to serve PNG tiles to a frontend. Combined with a CDN this approach would have very similar performance and scalability, although it does require a server where the first approach would simply be serving static files.

References

[1] "Esri map framework," https://www.esri.com/en-us/home.
[2] "Google earth engine," https://earthengine.google.com/timelapse/.
[3] "Landsat science." [Online]. Available: https://landsat.gsfc.nasa.gov/
[4] *Our map data*, https://docs.mapbox.com/help/how-mapbox-works/mapbox-data/.
[5] "Usgs earth explorer application," https://earthexplorer.usgs.gov/.
[6] "Usgs landsatlook," https://landsatlook.usgs.gov/viewer.html.
[7] S. E. Battersby, M. P. Finn, E. L. Usery, and K. H. Yamamoto, "Implications of web mercator and its use in online mapping," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 49, no. 2, pp. 85–101, 2014.
[8] M. P. Finn, E. L. Usery, K. H. Yamamoto, and S. E. Battersby, "Implications of web mercator and its use in online mapping," 2014.
[9] e. a. Frank Warmerdam, Evan Roualt, "Gdal," https://gdal.org/.
[10] E. Helmer and B. Ruefenacht, "Cloud-free satellite image mosaics with regression trees and histogram matching," *Photogrammetric Engineering & Remote Sensing*, vol. 71, no. 9, pp. 1079–1089, 2005.
[11] R. Mach, "Research on the processing of satellite photography by digital techniques," AIR FORCE CAMBRIDGE RESEARCH LABS LG HANSCOM FIELD MASS, Tech. Rep., 1962.
[12] Mapbox, "Mapbox js," https://github.com/mapbox/mapbox.js/.
[13] V. Sarago, "Remotepixel-tiler repository," https://github.com/remotepixel/remotepixel-tiler.

Fig. 10. The development of the Benban Solar Park in Egypt visualised using our project.

[14] ——, "Rio cogeo repository," https://github.com/cogeotiff/rio-cogeo.
[15] ——, "Rio tiler mosaic repository," https://github.com/cogeotiff/rio-tiler-mosaic.
[16] ——, "Rio tiler repository," https://github.com/cogeotiff/rio-tiler.
[17] J. Schwartz, *Bing Maps Tile System - Bing Maps*, Feb 2018. [Online]. Available: https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system
[18] N. Short, *The LANDSAT Tutorial Workbook: Basics of Satellite Remote Sensing*, 1982.
[19] P. Singh and N. Komodakis, "Cloud-gan: Cloud removal for sentinel-2 imagery using a cyclic consistent generative adversarial networks," in *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2018, pp. 1772–1775.

APPENDIX

The table showing the distribution of work is shown below.

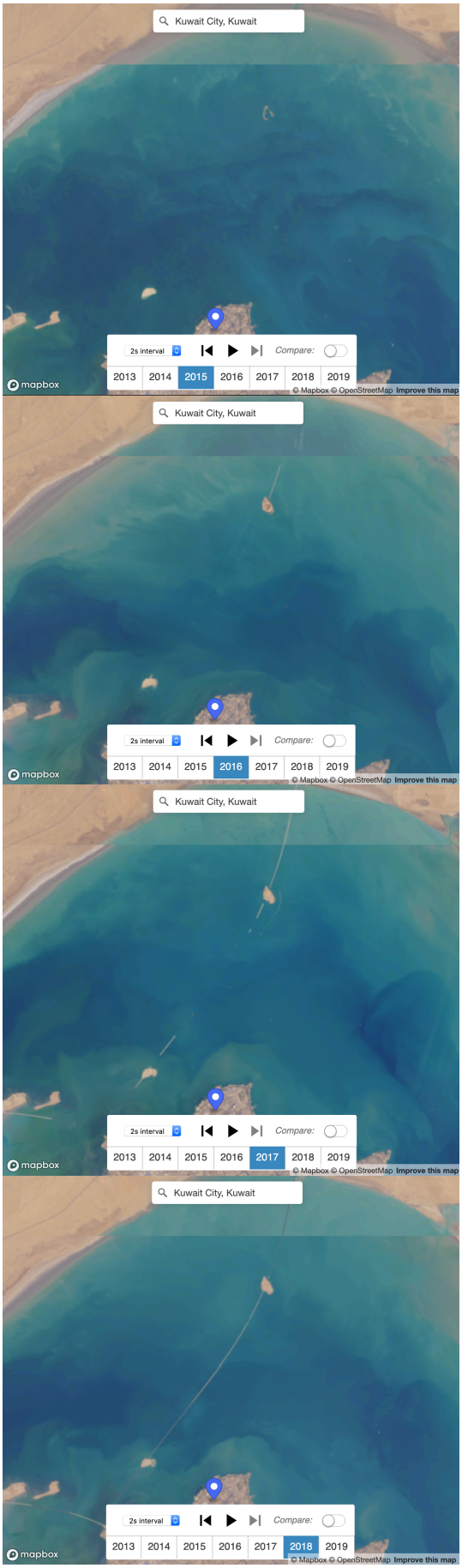

Fig. 11. The construction of one of the world's longest bridges created near Kuwait City visualised using our project.

Task	**Who**	**Time in man hours**
Writing Paper: Abstract, Introduction & Related work	Stan	6 hours
Writing Paper: Research Questions	Thijmen	2 hour
Writing Paper: Project Setup: Preliminary Analysis	All	6 hours
Writing Paper: Project Setup: Preparing scenes	Mostly Thijmen	10 hours
Writing Paper: Project Setup: Render the tiles	Mostly Sam	6 hours
Writing Paper: Project Setup: Results	Mostly Thijmen	5 hours
Writing Paper: Conclusion	Mostly Sam	2 hour
Finalizing/rewriting all paper sections	All	16 hours
Creating the visualisation	All	20 hours
Preparing scenes	All	∞ (estimated to be around 150 hours)
Rendering tiles	All	∞ (estimated to be around 100 hours)
Splitting tiles	All	∞ (estimated to be around 80 hours)
Down sampling images	All	∞ (estimated to be around 25 hours)
Running & debugging infrastructure	All	∞ (estimated to be around 72 hours, including night shifts)