Practice of Software Developement

End of Term Assignment

Visualising Tweets in Google Earth

Simon Donike
Anna Porti Suárez
Copernicus Master in Digital Earth
19.02.2021

Introduction

Within the end of term assignment for the *Practice of Software Development* course, a *KML* file is created visualizing both the result of an *OGC Web Map Service*[1] request as well as the parsed information from a *CSV* file containing tweets and their locations. Additionally, the tweets are analysed regarding possible profanity and visualized accordingly.

The following report will detail the structure of the code used to implement such visualization and the design decisions carried out.

Implementation and Design

The general structure of the project submission is comprised of four Java classes:

- *GoogleEarthTweetMapper.java*
- *WMSconnector.java*
- *CSVtoKML.java*
- *CSVtoKML_polygon.java*

All java files, including the *KML* outputs, as well as the file with the tweets (*tweets.csv*) and other external used sources, can be found in a public GitHub Repository.

GoogleEarthTweetMapper

The *GoogleEarthTweetMapper.java* is the main class, which calls functions from the other classes and saves the resulting *WMS* image.

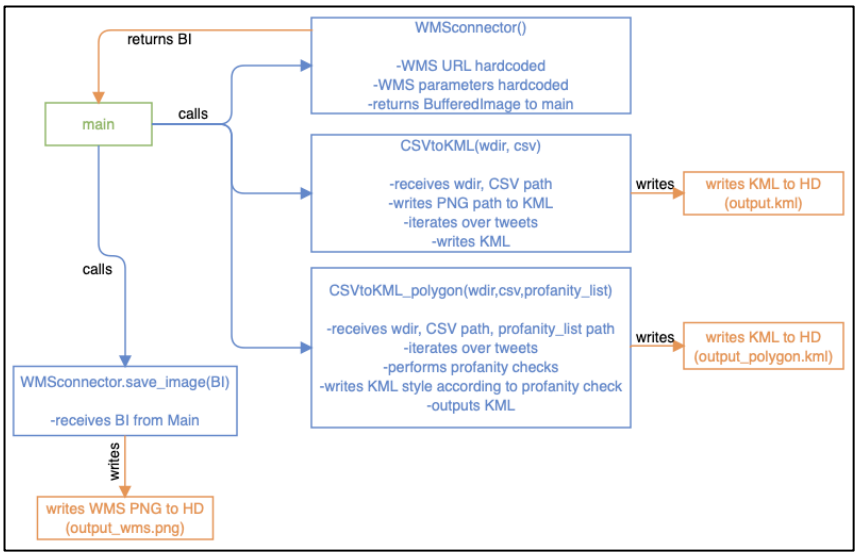

Figure 1. GoogleEarthTweetMapper diagram.

[1] Open Geospatial Consortium (n.d.). *Web Map Service*. Retrieved February 12th 2020 from https://www.ogc.org/standards/wms

As shown in *Figure 1* the *WMSconnector* is called first by instantiating a *WMSconnector* object. Then, on that object, the *getWMSimage(…)* method is called, which returns a Buffered Image (from now on *BI*). This is then saved and written to the Hard Drive via a file writer, passing the *BI* to the *WMSconnector.save_image(…)* function.

Next, the *CSVtoKML* class is called, passing the directory and file name of the *twitter.csv* file, which saves the created *KML* into the working directory. The same is done with the *CSVtoKML_polygon* class, which receives the same arguments. The created *KML* is saved to the Hard Drive from within the class. It would also be possible to write the finished *HTTP WMS* request into the *KML* so that it is dynamically loaded when opening the file, but it was explicitly stated in the task to store the *WMS PNG* on the hard drive.

WMSconnector Class

The *WMSconnector.java* sends the *WMS* request and returns a *BI*, which is then saved by the main class.

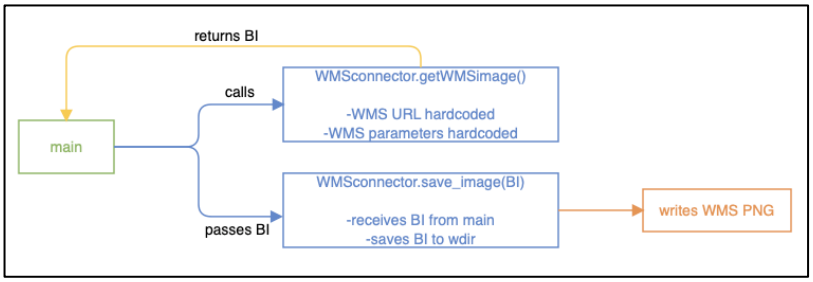

Figure 2. WMSconnector diagram.

The *WMSconnector* class contains two functions: *getWMSimage()* and *save_image(BI)* (see *Figure 2*). The first function is called from main without any arguments, since all relevant parameters are hard-coded into *GetMapRequest.request.* Before that, a try-catch block validates that the input *WMS URL* is valid. After sending the request via *.issueRequest(wms)*, the returning image is read via *getInputStream()* and saved as a *BI* variable and returned to main. From main, the *BI* is then sent to the second function, which is passed the *BI* and then saves it as *PNG* to the working directory.

CSVtoKML class

This java class takes the working directory and the *CSV* file name as arguments when it is called from main. As displayed in *Figure 3,* it only contains one function, which handles all operations. This was done because the writing of the *KML* is quite straight forward, the iterating over the *CSV* file can easily be done with a while loop and therefore it seems overly complicated to split the steps up in different functions.

First, a ground overlay element is created via pre-defined strings and attached to the *KML* string. The transparency is set via the *color* tag, which is explicitly not intended for use with raster data, but it works. The *WMS* image is inserted by pointing to the correct storage location of the *PNG* (see *Figure 4*).

Then, the *CSV* file is read via *Buffered Reader* and an empty String Array created. Also, the string which will later on contain the whole *KML* string is created and filled with the *KML* header information. Iterating over the tweets list (excluding the header) via a while loop, the information for each tweet is read and stored in the array. Using string concatenation, the relevant *KML* tags are opened and the info from the *CSV* line inserted.

The extended data tags are used because they are shown as a nicely formatted table when clicking on the icons in *Google Earth Pro,* as visible in *Figure 5*. Additionally, the timestamp from the tweet does not conform to the specifications as given by the *KML* documentation, so the space is replaced with a *T* and the hour added to the end of the string. The timestamps are saved within *"TimeStamp"*[2] *KML* tags so *that Google Earth* can correctly identify the time series of the tweets. After each iteration, the string holding this line's tweet information is appended to the *KML* string. All statements within that *KML* string are given with formatting statements such as tabs and new lines. At the end, the *KML* string s saved as a *KML* file and can be opened with *Google Earth Pro.*

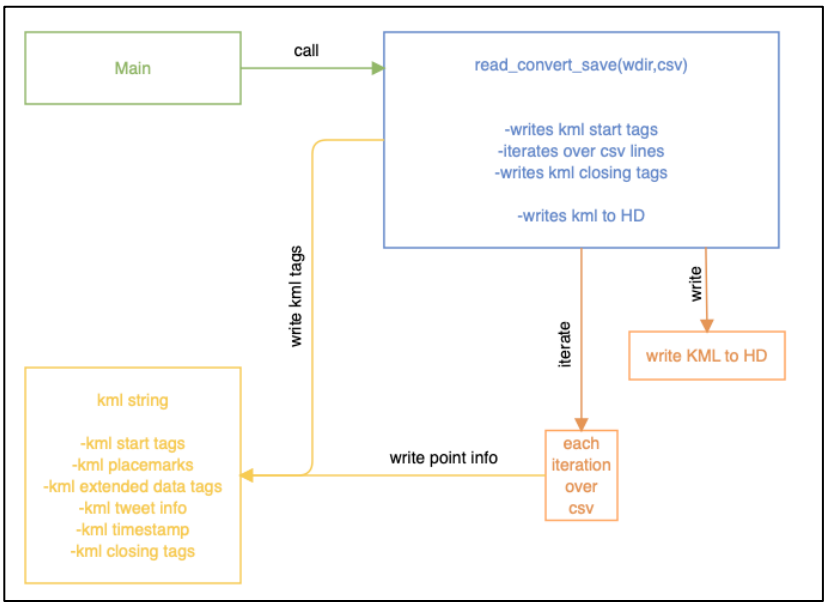

Figure 3. CSVtoKML diagram.

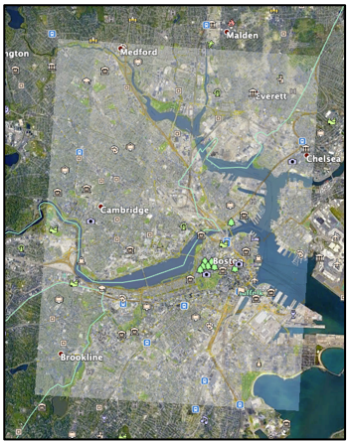

Figure 4. Ground Overlay.

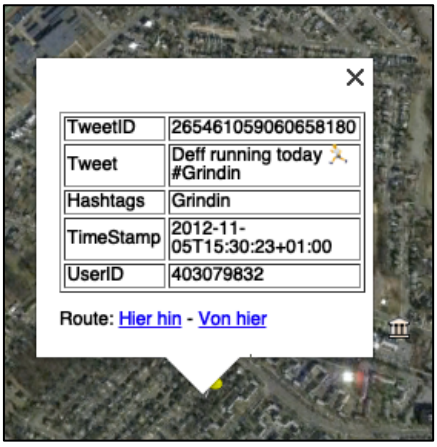

Figure 5. Opened KML point location, showing "Extended Data" tag contents.

CSVtoKML_polygon Class

CSVtoKML_polygon.java performs the same iteration and *KML* creation as the *CSVtoKML* class, but additionally creates an extruded polygon (instead of points shown on the map) and checks the tweets for profanity, color-coding the polygons to visualize whether or not the tweets contain profanity. Therefore the inner workings of the *KML* creation will not be detailed again, but are visualized in a diagram in *Figure 6.*

The list of offensive words is taken from *Full List of Bad Words and Top Swear Words Banned by Google* (Gabriel R.J., last updated 19.09.2020)[3], which is the list of words considered offensive by the *Google Search Engine.*

[2] Keyhole Markup Language (n.d). *Time and Animation.* Retrieved February 12th 2020 from https://developers.google.com/kml/documentation/time#gps
[3] Gabriel R.J. (Last updated 19.09.2020). Full List of Bad Words and Top Swear Words Banned by Google. Retrieved from GitHub, February 12th 2020, from https://github.com/RobertJGabriel/Google-profanity-words

Since the creation and extrusion of polygons, as well as the saving of *KML* style templates and the opening, checking and visualization of the tweets according to their profanity content contains quite a lot more complexity, many of those tasks were outsourced to functions. The functions are all called from within the *read_convert_save_polygon(...)* function, therefore only one call from main is necessary to start the whole process.

First, after writing the opening *KML* tags to the string, the *create_styles(...)* function is called. This returns style templates for the polygons as a string which is written to the *KML* string and can later be referenced by their IDs from within the polygon tags.

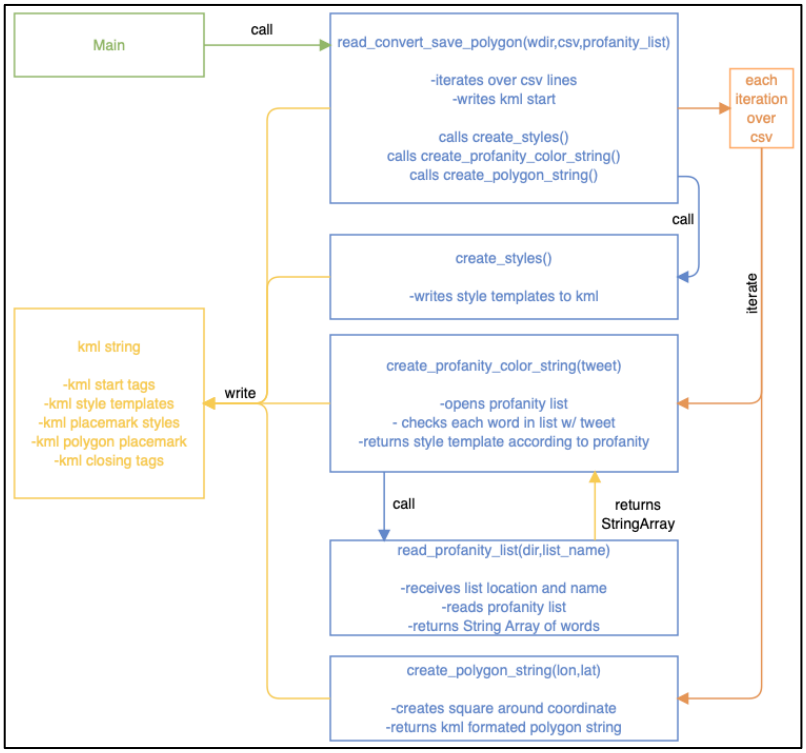

Figure 6. CSVtoKML_polygon diagram.

Then, within every loop over a new line of the *tweets.csv* file, the *create_profanity_color_string(...)* function is called. This function is passed the tweet itself as well as the location and name of the profanity list file. For each tweet, the *read_profanity_list(...)* function is called, which receives the location and name of the profanity list and opens it, returning it as a String Array. Back in the *create_profanity_color_string(...)* function, this String Array[4] is iterated over and checked for matches with the tweet. If a match is found, the *KML* color style tag for *"red"* is received and written into the *KML* structure of the polygon of said tweet. The detection is not perfect, since checking for the words itself would return many false positives ("*I passed my exam*"). Therefore, a space in front of each offensive word is added to exclude such false positives. Adding a space behind the word would have excluded many common expressions where a punctuation mark is added at the end, such as *"you're a bitch!"*.

After the style information is added according to the profanity content, the polygon around the point is created. The *create_polygon_string(...)* function is called, which receives the coordinates of the tweet. Adhering to the *KML* polygon specifications[5], the polygon tags are created as strings. The coordinates are transformed, adding and subtracting a certain value to and from the coordinate to create a square around the coordinate, as explained in the schema in *Figure 7*. Also, the parameters for extrusion are enabled and set as relative to ground, meaning the polygons are extruded from the surface by 100m (as specified with the *z* coordinate in the coordinate section). The *KML* string containing the polygon is then returned to the loop.

Since for each tweet, the information of the tweet itself, the polygon and the check for profanity and the definition of the according style is now done (see *Figure 8*), the temporary tweet *KML* string can be added to the ever-growing *KML* string. Finally, the *KML* string is stored as *"output_polygon.kml"* in the working directory.

[4] W3Schools (n.d.). Java Arrays. Retrieved February 12th 2020, from https://www.w3schools.com/java/java_arrays.asp
[5] Keyhole Markup Language (n.d). KML Reference. Retrieved February 12th 2020 from https://developers.google.com/kml/documentation/kmlreference#polygon

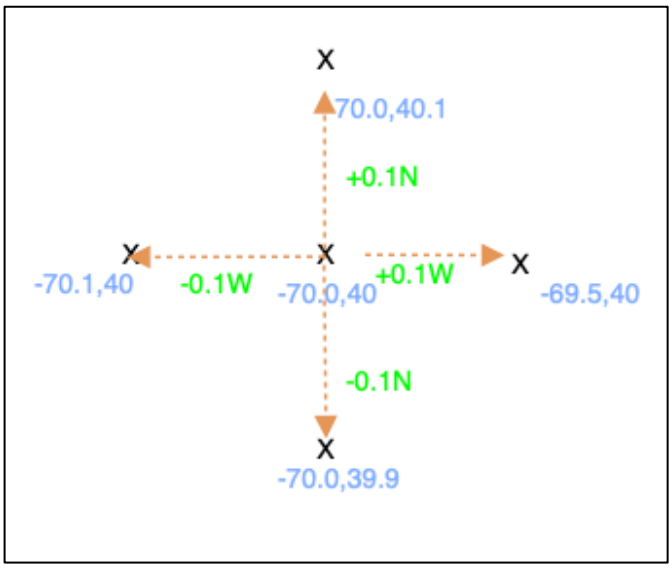

Figure 7. Coordinate parameters schema for polygon extrusion (actual values differ).

Figure 8. Extruded polygons coloured according to profanity (red)/non-profanity (green).

References

- Gabriel R.J. (Last updated 19.09.2020). *Full List of Bad Words and Top Swear Words Banned by Google.* Retrieved from *GitHub,* February 12th 2020, from https://github.com/RobertJGabriel/Google-profanity-words
- Keyhole Markup Language (n.d*). KML Reference.* Retrieved February 12th 2020 from https://developers.google.com/kml/documentation/kmlreference#polygon
- Keyhole Markup Language (n.d*). Time and Animation.* Retrieved February 12th 2020 from https://developers.google.com/kml/documentation/time#gps
- Open Geospatial Consortium (n.d.). *Web Map Service*. Retrieved February 12th 2020, from https://www.ogc.org/standards/wms
- W3Schools (n.d.). *Java Arrays.* Retrieved February 12th 2020, from https://www.w3schools.com/java/java_arrays.asp