Implementation of tiled vector services: a case study

Jens Ingensand[1], Marion Nappez[1], Cédric Moullet[2], Loïc Gasser[2], Olivier Ertz[1], and Sarah Composto[1]

[1]University of Applied Sciences Western Switzerland
Route de Cheseaux 1, 1401 Yverdon-les-Bains, Switzerland
[2]Swiss Federal Office of Topography Swisstopo
Seftigenstrasse 264, 3084 Wabern, Switzerland
{jens.ingensand,marion.nappez,olivier.ertz,sarah.composto}@heig-vd.ch
{cedric.moullet,loic.gasser}@swisstopo.ch

Abstract. Vector tiling aims at cutting vector data into smaller entities. It offers several opportunities, especially for the development of web-mapping systems, such as the possibilities to apply different styles, to access attributes or to render 3D data. Today no open and widely adopted standard exists for the implementation of web services involving vector tiles. In this paper we investigate several important parameters that need to be considered for the implementation of vector tile services. We then present a case-study where several tiled vector services are implemented. The results of this case study are useful for further implementations of tiled vector services and discussions regarding standardization.

Keywords: Vector tiling, web services, generalization, standardization

1 Introduction

The idea to tile vector data is very similar to raster data tiling where large raster data sets are tiled into smaller pieces and stored in hierarchical structures, either in databases or in file systems. Tiling in general allows for an efficient consumption of data through a network connection, for instance using the standardized OGC protocol WMTS (Web Map Tile Service, www.opengeospatial.org) implementation standard.

Vector data, as compared to raster data, has several advantages: it allows for more flexibility for rendering maps, such as the possibility to apply different styles or to render data in 3D. Moreover it is possible to store and transfer not only geometries, but also an entity's attributes. A third advantage is that, depending on the data layer, the features' attributes, the feature density and the level of detail, data can be compressed using different methods [3]. A fourth advantage is that data can be re-utilized and transformed on the client side; for instance using coordinate transformation, spatial analysis, and so forth. [7] On the other hand, vector tiling also has disadvantages e.g. the fact that the features must be reassembled on the client side (e.g. a polygon that has been cut into several pieces) or the problem that data can be illegally downloaded and reused for other purposes that were not intended by the administrators of a server infrastructure. The idea of a tiled vector data service is to combine both the advantages of vector data with the advantages of tiled raster data services. Today several commercial web map providers, for instance Google (http://maps.google.com) or MapBox (https://www.mapbox.com) have started using this concept.

In the following section we will discuss the key issues that need to be addressed for the generation of vector tiles. Thereafter we will present a case study where vector tile services have been created within the infrastructure of the Swiss Federal Geoportal map.geo.admin.ch. Finally we will discuss the results of this case study and suggest perspectives for further investigations.

2 Tiled vector services - key issues

Tiled vector services are more complex than tiled raster services due to several facts - for instance raster data uses a regular grid for storing information, while vector data does not, there are fewer (spatially exploited) raster data formats than vector data formats and vector data allows for storing attribute information. We have therefore chosen to list key issues that need to be addressed for the creation of tiled vector services.

Formats and standards Compared to common raster data formats such as jpeg, gif, png or tiff, far more different vector formats exist. Some vector formats are open standards (e.g. OGC's GML), some are proprietary (e.g. ESRI's File Geodatabase) and some formats are already used within a web-mapping context (e.g. the GeoJSON format). The choice of format is therefore more difficult since data needs to be compact and easy to create and to consume.

Tiling scheme Creating vector tiles implies cutting a vector layer into smaller pieces. One possibility is to set a fixed spatial extent (e.g. all generated tiles for one level of detail include features within a square of 500*500 meters). Depending on the vector layer to be tiled and the extent of a tile this might result in large quantities of empty tiles. This method has been used by Antoniou et al. [1] for instance. The other possibility is to create tiles depending on their weight (e.g. in terms of vertices per tile: a tile should for instance contain between 3 and 100 vertices) and thereby to use varying spatial extents for each tile. One drawback of this method is that it becomes more difficult to recalculate tiles if the original data layer changes frequently. Another drawback is the implementation on the client side (e.g. using a Javascript API) - the irregular organization of tiles needs to be communicated to the client and thereby the client needs to be able to request the right tiles for each level of detail at a given spatial extent. An implementation of this method has been created by Dufilie and Grinstein [5]

Levels of detail and generalization Vector tiles can be organized in different levels of detail (LOD). This allows clients to request tiles that fit a specific level of zoom. Creating vector tiles at different LOD's implies decreasing the complexity of a feature according to a certain level of detail. This can be achieved using generalization algorithms such as the Visvalingam algorithm [9], the Douglas-Peucker algorithm.[4] or Zhao-Sallfelds sleeve-fitting polyline simplification algorithm. [10]. Another common approach to generalization is to utilize semantics. For example if several levels of detail need to be created for an original road-layer which contains a classification, the classification can be used to include (or not) several classes of roads in different levels of detail.

Basic geometry types The basic 2D and 3D geometry type (point - line, etc) has an impact on the way data can be generalized and cut into pieces. For line and polygon features generalization algorithms can create tiles at different levels of detail. For point features, things get more complicated since reducing the level of detail implies reducing the number of vertices and thereby deleting features. In this case clustering algorithms at different zoom levels can help reducing the number of features. The drawback is the loss of features and/or attributes.

Grouping layers in one tile A vector tile on a web server equals a piece of data (e.g. a data file). If different layers are requested by a client, this can result in several queries to a web-server. One possibility to decrease the number of requests is to group layers (e.g. point - polygon and line-based layers) in one tile and thereby in one data file. This approach could potentially decrease overall data size by minimizing the replication of data headers, standard tags and so forth.

Update frequency If a data layer is updated frequently, vector tiles need to be recalculated for the updated regions. This has an impact on the management of vector tiles - in order to recalculate tiles a management system needs to keep track of changes.

Attributes Vector data generally consist of both vector features and associated attributes. If a feature (except point features) is split in two parts (see Figure 1), the question arises where to store the attributes. The following three options can be considered:

– A feature's attributes are simply copied in each of the parts. The advantage is that all attributes are directly available for all parts; even if all the parts have not been downloaded on a client all attributes are available. The drawback is the fact that information is duplicated.
– Only one part contains the attributes. The advantage is that no information is duplicated. On the other hand if a feature (e.g. a motorway ranging over thousands of kilometers) is split into several parts it becomes difficult to find the exact part containing the attributes. This problem however could be addressed if the exact location of the tile containing the attributes is

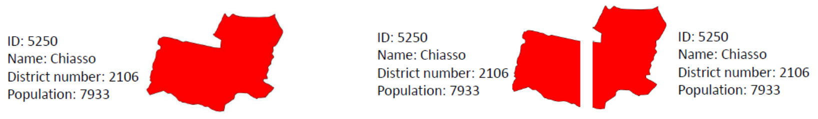

Fig. 1. The problem of attribute handling - a polygon feature is split in two parts

defined in all tiles. Nordan [8], for instance suggests a manner of storing this information in vector tiles so that a client can reassemble information.
– Attributes are stored in separate files or made available through a separate web-service. The advantage is that the attributes are not stored in the vector tiles anymore (and thereby vector tiles are lighter). The disadvantage is the fact that another web-service (or another data file) needs to be created - this can result in more queries and are more complex web-services and system architecture.

3 Case study: tiled vector services for the Swiss Federal Geoportal

3.1 Context

Map.geo.admin.ch is the official geoportal of the Swiss state, serving almost 400 data layers. The open-source framework MapFish (mapfish.org) which itself consists of several Python and Javascript libraries are the main components of the system. The geoportal uses the Amazon EC2 and S3 (aws.amazon.com) cloud computing infrastructure. A majority of the available data layers are generated using WMTS web services; some layers are available as WMS web services. About 2'500'000'000 WMTS unique raster tiles are stored in the cloud infrastructure. In 2015 a 3D web-interface based on CesiumJS (cesiumjs.org) was created.

3.2 Goals

The goal of this case study was to build and investigate tiled vector services within the infrastructure of the Swiss Federal Geoportal map.geo.admin.ch and to build a working prototype in order to consume these services. The main objectives of this case study were:

– tiled vector service should co-exist with existing web-services (such as WMTS)
– tiled vector service should use the existing server and client infrastructure
– tiled vector service should use existing standards as far as possible

Another important goal was to compare vector tiles with raster tiles considering their weight in terms of bytes and kilobytes. The tile weight is an important indicator for the efficiency of features' storage and transfer and thereby an important factor for a web-mapping system's performance.

```
{
    "type": "Topology",
    "transform": {
        "scale": [1,1],
        "translate": [0,0]
    },
    "objects": {
        "nom": {
            "type": "GeometryCollection",
            "geometries": []
        }
    },
    "arcs"; [[]]
}
```

Fig. 2. The structure of a TopoJSON-tile

3.3 Case study settings

Constraints The infrastructure of the Swiss Federal Geoportal mainly uses PostgreSQL and Python-scripts for storing and manipulating vector data. A constraint of the project was therefore the utilization of these technologies for the production of vector tiles.

Test data For our tests we utilized the following data-sets. Each data-set covers the whole of the country:

– Polygons: the zip-code-area dataset (4'166 objects, 1'874'382 vertices)
– Lines: the road network "vector 25" (1'342'108 objects, 12'800'516 vertices)
– Points: labels "SwissNames Vector 200" (19'086 objects and vertices)

Tiling scheme We decided to use the regular WMTS tiling scheme for addressing vector tiles - mainly for the two reasons that it is less difficult to address tiles with a regular tiling scheme and that the WMTS standard is already supported in several clients (desktop and mobile/web-based clients). The implementation a client that consumes both vector and raster data using the same tiling scheme appeared to be less difficult.

Vector tile format One goal of the project was to find a format that could be easily consumed by several clients. Various open standards were discussed for storing vector data such as XML-based formats (e.g. GML) and JSON-based formats (e.g.GeoJSON). We decided to utilize TopoJSON, (see Figure 2) a JSON-based format, that stores topological information; e.g. if two polygons share the same boundary, the boundary is just stored once. We chose this format mainly for its compactness and the easiness to interpret it using Javascript. We did not include any attributes in the tiles.

Generalization and simplification In order to create vector tiles at different levels of detail, data needs to be generalized and simplified. Due to the constraints of the infrastructure (PostgreSQL/PostGIS and Python) we were able to test and utilize Visvalingam algorithm [9] and Douglas-Peucker algorithm.[4]

for line- and polygon-based features. Due to the fact that the Swiss Federal Geoportal utilizes (and will utilize) raster tiles, we used raster maps as a benchmark in order to find the best parameters for the two algorithms as well as for the comparison of the results. For point-features we tested the simplification based on attributes (e.g. selecting point-features based on an attribute that was used in order to define a point features' importance.)

3.4 Web Service Implementation

Commonly used OGC web-services such as WFS or WMS implement standard queries such as GetCapabilities (to get information about the data and formats etc) or queries such as GetMap or GetFeature to receive the information. In our case the goal was to create a webserver that simply handles vector files instead of raster files according to the WMTS file organization scheme. A common WMTS file-query such as
`https://wmts.geo.admin.ch/1.0.0/roads/default/21781/17/7/3.jpeg`
would be replaced by:
`https://wmts.geo.admin.ch/1.0.0/roads/default/21781/17/7/3.json`
The generation of vector tiles was implemented in the PostgreSQL/PostGIS database using SQL-queries that were executed with Python-scripts:

- All different levels of detail were computed as new tables in the database using the aforementioned algorithms.
- Each level of detail was cut into tiles according to the same WMTS tiling scheme that has been used for the generation of all raster tiles.
- Each tile was exported in the TopoJSON format and written in a web-server directory using the WMTS hierarchy and nomenclature.

3.5 Client Implementation

The native OpenLayers library already had implemented support for TopoJSON-files. The implementation of a prototype using OpenLayers was therefore more of a configuration issue. We were able to implement a prototype that renders vector tiles according to specific attributes, however the implementation of a client that also aggregates features from several tiles (e.g. a polygon that had been cut in two pieces) into the original feature remained to be implemented at the time of writing.

3.6 Results

Generalization and simplification In order to identify the best generalization algorithm and parameters, we visually compared the output of one algorithm with the corresponding raster-tiles that had been pre-produced by Swisstopo. For instance if a zip-code border on a specific raster tile had a certain shape we tried to identify the best algorithm and corresponding parameters in order to

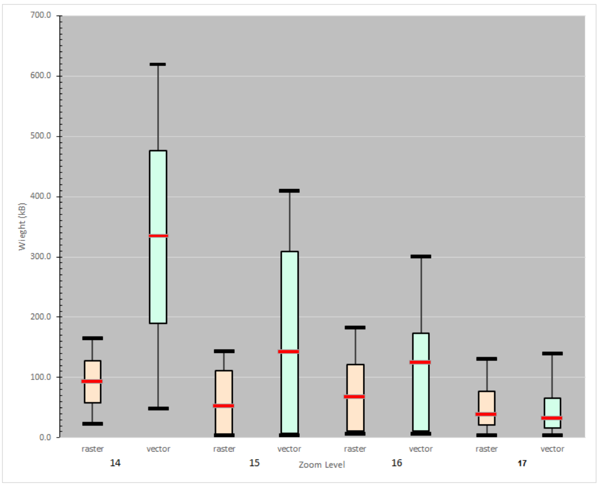

Fig. 3. Boxplot showing tile sizes (kiloBytes) at four different WMTS zoom levels for the communities-layer in raster- and vector format. Empty tiles (i.e. tiles without any features) have been omitted

make vector features (at a given zoom level / level of detail) overlap the corresponding raster features as far as possible. Using our method of raster-vector overlay we discovered that the Douglas-Peucker-algorithm is well adapted for the generalization of line-features (such as roads) since it tries to identify the most prominent vertices using distances between vertices. The Visvalingam-algorithm on the other hand appeared to be well-suited for the generalization of polygon features such as boundaries. The algorithm classifies the prominence of a vertex using the area of the triangle that is formed with its two neighboring vertices. The output of this algorithm thereby appears to be smoother.

Weight We compared the weight (in terms of kilobytes) of vector tiles with the weight of the corresponding raster tiles. Raster tiles used the png-format. Figure 3 shows a boxplot of the weight of raster tiles at four different zoom levels. We discovered that at lower zoom levels raster tiles are clearly lighter on average than vector tiles while vector tiles show a larger range of light and heavy files. At higher zoom levels vector tiles become lighter than raster tiles. These observations have been made for all data layers that were analyzed. An empty raster tile (0.18 kB) is on average twice as heavy as an empty vector tile. (0.09 kB).

4 Conclusions

This case study demonstrated the feasibility of the implementation of tiled vector services using the given existing infrastructure. The weight of tiles is a crucial point since it influences storage size, bandwidth usage and rendering speed. Compared to vector tiles, raster tiles theoretically have an upper size limit due to the fact that raster tiles are based on a regular grid and each raster cell can only store a certain amount of information, however vector tiles do not have this limit. This consideration also explains why the weight of vector tiles shows a larger range than raster tiles.

A major difficulty for the generation of vector tiles (and thus for the minimization of tile weight, storage and bandwith usage) is the generalization and simplification of features. Generalization can be automated to a certain degree, however if the data layer to be generalized is complex and if topology needs to be preserved too there is a limit to how much a data layer at a certain level of detail can be compressed. A solution is to use semantics (e.g. only certain types of features are included), but this needs to be configured manually.

Another important point is the absence of open and widely adopted standards. Within this case study we have made certain choices in order to re-utilize existing basic standards such as the WMTS tiling scheme. These choices were also influenced by the existing infrastructure. Due to the complexity of vector tiling the establishment of a standard that fits different configurations and themes (e.g. systems with a limited number of data layers) appears to be difficult.

5 Perspectives

Within the scope of this case study we did not address the handling of attributes and the dynamic aggregation of features that have been split. This subject will be part of further investigations. Another perspective for future work is the creation and utilization of vector tiles containing 3D data with different clients such as CesiumJS in order render 3D features. For each layer generalization and simplification need to be taken into account with care in order to minimize file size. If a vector data layer is very dense, the files containing vector features can get very heavy at certain zoom levels. We suggest two approaches to address this subject in future projects:

As suggested by Feixiang et al [6] vector features can be progressively transferred - a possibility is therefore to decompose the contents of a single vector tile into even more pieces and to transfer these pieces progressively. A drawback of this method is the complexity of tile generation and client implementation. A final possibility to address the problem of heavy vector tiles is to mix raster and vector tiles (e.g. raster tiles are visible until a certain zoom level and vector tiles after that). The advantages of this method would be that the size of the tiles can be kept low and that it becomes easier to automatically create tiles, the drawback is that vector features are only available at a certain zoom level and that the rendering of raster and vector tiles needs to be visually equal.

Another field of investigation are variations in spatial data; e.g. how does vector tiling react to data layers that show large variations in terms of vertex density. Furthermore other formats such as the MapBox Vector tile specification (www.mapbox.com) are worth to consider and to compare.

Standardization perspectives about styling may also be important to consider. Vector tiles are rather prepared geodata benificial to visualization since tiles can be styled when requested, allowing for many map styles. With classical webmapping using pre-drawn image tiles, the client does not have to deal with styling as the web map server does apply an internally defined style. In order to render tiled vector data, styling needs to be done on the client. We may imagine the tiled vector service offering a kind of GetStyle(s) method (e.g like the OGC WMS/SLD 1.0 profile) in order to get a default style that the client can apply. Or the symbology instructions may be shared and retrieved through a web catalog of styles. In this context, interoperability does matter and a standardized styling language is desirable to allow sharing of cartographic instructions (just like OGC Symbology Encoding). But given the large variety of client types, such a styling standard shall also consider various encodings such as XML, CSS-like or even JSON encodings [2]. Therefore a common encoding-neutral symbology model should be used.

References

1. Antoniou, V., Morley, J., Haklay, M.: Tiled vectors: A method for vector transmission over the web. In: Carswell, J., Fotheringham, A., McArdle, G. (eds.) Web and Wireless Geographical Information Systems. Lecture Notes in Computer Science, vol. 5886, pp. 56–71. Springer Berlin Heidelberg (2009)
2. Bocher, E., Ertz, O.: Towards cartographic portrayal interoperability. the revision of ogc symbology encoding standard. In: 1st ICA European Symposium on Cartography; Proceedings. pp. 116–119 (November 10-12 2015)
3. Chen, F., Ren, H.: Comparison of vector data compression algorithms in mobile gis. In: Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on. vol. 1, pp. 613–617 (July 2010)
4. Douglas, D., Peucker, T.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer 10(2), 112122 (1973)
5. Dufilie, A., Grinstein, G.: Feathered tiles with uniform payload size for progressive transmission of vector data. In: Web and Wireless Geographical Information Systems. 13th International Symposium, W2GIS 2014, Seoul, South Korea, May 29-30, 2014. Proceedings. pp. 19–35. Springer (2014)
6. Feixiang, C., Xiao, M., Haiyan, R.: Progressive transmission of vector spatial data. Journal of Networks 8(5), 1065–1072 (May 2013)
7. Gaffuri, J.: Toward Web Mapping with Vector Data, Lecture Notes in Computer Science, vol. 7478, pp. 87–101. Springer Berlin Heidelberg (2012)
8. Nordan, R.P.V.: An Investigation of Potential Methods for Topology Preservation in Interactive Vector Tile Map Applications. Master's thesis, NTNU Norwegian University of Science and Technology, Trondhein (2012)
9. Visvalingam, M., Whyatt, J.: Line generalisation by repeated elimination of the smallest area. Cartographic Journal. 30(1), 46 – 51 (1992)
10. Zhao, Z., Saalfeld, A.: Linear-time sleeve-fitting polyline. Autocarto 13, ACSM/ASPRS97 Technical Papers, Seattle, Washington (5), 214223 (April 1997)