Multi-layer vario-scale web map comparer with dynamic transitions and visual analytical tool

Peng, Dongliang; Meijers, Martijn; Oosterom, Peter van

International Cartographic Association
Association Cartographique Internationale

Multi-layer vario-scale web map comparer with dynamic transitions and visual analytical tool

Dongliang Peng *, Martijn Meijers, Peter van Oosterom

Section GIS Technology, Faculty of Architecture and the Built Environment, Delft University of Technology, Delft, The Netherlands, {D.L.Peng, B.M.Meijers, P.J.M.vanOosterom}@tudelft.nl

* Corresponding Author

Abstract: This paper presents a web map juxtaposition comparer. Using the comparer, we can place two maps side by side for comparison. The maps of the comparer can contain multiple layers, and each of them can be a multi-scale layer or a vario-scale layer. We enhance the comparer's visual analytical ability by developing more functionalities, including toggling on/off the layers, tuning the opacities, and swiping to change the maps' widths. The aim of developing the web map juxtaposition comparer is to carry out usability studies to see if a vario-scale map is better then a multi-scale map in helping map users to keep their context during zooming. This paper presents two versions of the comparer. The first one uses respectively a multi-scale and a vario-scale vector layer of area objects in the left map and in the right map. Both maps use the same multi-scale raster layer as the background. The second comparer also uses the raster layer as the background, and it uses three thematic layers as the foreground. The change between the thematic layers is realized by switching on/off according to scales. In future, we will implement continuous changes between the three thematic layers for the right map.

Keywords: Space-scale cube, continuous map generalization, database, WebGL, framebuffer

1. Introduction

A vario-scale map has the property that a change of scale results in a change of the map (van Oosterom and Meijers, 2014), where the map change is, for example, a merge of two area objects. This is different from a multi-scale map, where there is no change until the scale moves into another scale interval (in the same scale interval, the multi-scale map will simply enlarges or shrinks on the screen). It is believed that vario-scale maps allow users to keep their context more easily during zooming (van Kreveld, 2001; Nöllenburg et al., 2008). Much research has been devoted into providing vario-scale maps (e.g., Chimani et al., 2014; Li et al., 2017; Peng et al., 2020). Now, it is necessary to develop a web map comparer so that a usability test is possible to prove if vario-scale map is really better than a multi-scale map.

The development of web maps started not long after the World Wide Web was created, and the state-of-the-art web maps are interactive and multi-sourced (Veenendaal et al., 2017). Van Oosterom and Meijers (2014) developed the concept of the space-scale cube (SSC). Based on the SSC, Meijers et al. (2020) implemented a vario-scale web map of area objects. They made chunks of the SSC data, where each chunk stores a part of the SSC. On this basis, they were able to send only the chunks relevant to users' interested places, from the server side to the client side. They showed how to efficiently slice the SSC to output a web map at a given scale based on WebGL using the GPU at the client side. In addition to slicing the SSC with a horizontal plane, they also sliced the SSC with a curly surface to have a locally more detailed map or with a tilted surface to have a perspective view. Lobo et al. (2015) evaluated several map comparison techniques, including *juxtapose*, *translucent overlay*, *swipe*, *blending lens*, and *offset lens*.

This paper develops a juxtaposition comparer of web maps, where the comparer is implemented in JavaScript with WebGL. The juxtaposition comparer places two maps side by side, and each map has multiple layers. Using this comparer, we display a multi-scale layer in the left map and a vario-scale layer in the right map (see Section 2). Section 3 shows the progress of preparing for a usability test on some thematic layers with attribute dryness values. Section 4 concludes the paper and present some future work.

2. Juxtaposition comparer

Our juxtaposition comparer places two maps side by side. The left map presents a multi-scale vector layer of area objects, and the right map shows a vario-scale vector layer of area objects. To allow map users to access more geographic information, both maps use the same multi-scale raster layer as the background. In order to make the background visible, the foreground vector layers are set to be transparent. In the following, Section 2.1 introduces the raster layer, which covers The Netherlands and the surrounding areas. Then, Section 2.2 illustrates the vario-scale vector layer that represents province Drenthe of The Netherlands. In Section 2.3, we generate the multi-scale vector layer from selecting some points in scale of the vario-scale vector layer, where the multiple scales are aligned with the multi-scale raster layer. Section 2.4 shows how to avoid visual flaws using a framebuffer.

(a) Level 2

(b) Level 3. The transparent polygon in the upper-right corner represents province Drenthe, The Netherlands.

Figure 1. Two screenshots of the multi-scale raster layer.

2.1 Multi-scale raster layer

The raster layer used in this paper is provided as a public service by Kadaster, The Netherlands.[1] The raster layer contains a rich set of information, including streets, waters, labels, etc.[2] It consists of 15 levels, numbered from 0 to 14, where level 0 is the most general and level 14 is the most detailed (see Figure 1 for example).[3] Each of the levels covers the same extent of the earth, i.e., The Netherlands and the surrounding areas. The levels are represented by a set of images of 256×256 pixels. We term an image a *tile*. Level i consists of 4^i tiles according to the organization of the raster layer. As the numbers of the tiles increase exponentially with the increasing level, the data size of all the levels is huge. To make our comparer feasible, we only use levels 0–10 in our prototype. For these 11 levels, there are 1,398,101 tiles with size 2.8 GB. As level 10 is the most detailed, it is used as the base level. Currently, we use scale 1 : 12,000 as the scale for level 10. According to the idea of Huang et al. (2016), we have

$$S_i = S_\mathrm{b} \sqrt{\frac{N_\mathrm{b}}{N_i}}, \tag{1}$$

where $S_\mathrm{b} = 1 : 12{,}000$ and S_i are the scale denominators of the base level and level i. Variables $N_\mathrm{b} = 1{,}048{,}576$ and N_i are the tile numbers of the base level and level i. As $N_b = 4^{10}$ and $N_i = 4^i$, we have

$$S_i = 12000 \cdot 2^{10-i}.$$

According to scale denominator S_i, we define the scale range for each level so that a level is presented when the scale is in the corresponding scale range. Table 1 presents the details of the levels.

[1] The website of Public Services On the Map (PDOK) is https://www.pdok.nl/. Accessed: October 30, 2020.

[2] The metadata of the raster layer is available at http://www.nationaalgeoregister.nl/geonetwork/srv/dut/catalog.search#/metadata/ee543323-0fe4-4353-9161-eda61ff26c07. Accessed: October 30, 2020.

[3] More information about the levels is available at https://geodata.nationaalgeoregister.nl/tiles/service/tms/1.0.0/brtachtergrondkaartgrijs/EPSG:28992. Accessed: October 30, 2020.

Table 1. The levels of the raster layer. Column S_{range} represents the range of scale denominators for each level. Column n_{tiles} represents the number of tiles for each level.

level	S_{range}	n_{tiles}
0	$[12288000, \infty)$	1
1	$[6144000, 12288000)$	4
2	$[3072000, 6144000)$	16
3	$[1536000, 3072000)$	64
4	$[768000, 1536000)$	256
5	$[384000, 768000)$	1024
6	$[192000, 384000)$	4096
7	$[96000, 192000)$	16384
8	$[48000, 96000)$	65536
9	$[24000, 48000)$	262144
10	$(0, 24000)$	1048576

2.2 Vario-scale vector layer

The vector layer covers province Drenthe of The Netherlands. The position of the province is shown in Figure 1b. The area objects of this layer is extracted from the TOP10NL data set.[4] We set the base scale of the layer to be $1 : 10{,}000$. At the base scale, the layer has $288{,}726$ area objects. In order to make the layer variable in scale, for zooming out, we repeatedly find the smallest area and merge it into its most compatible neighbor. The most compatible neighbor is defined as the one having the longest common boundary with the smallest area. For zooming in, we will observe a sequence of splitting an area into two. Similar to Equation 1, the relationship of the scale denominator and the number of remaining area objects is established by formula

$$N_t = \left\lfloor N_{\text{b}} \frac{S_{\text{b}}^2}{S_t^2} \right\rfloor, \tag{2}$$

where we have number $N_{\text{b}} = 288{,}726$ and $S_{\text{b}} = 10{,}000$.

2.3 Multi-scale vector layer

The multi-scale vector layer is a selection from some points in scale of the vario-scale vector layer. We align the scales with that of the multi-scale raster layer. When a scale is selected, the number of objects remained on the map is computed by Equation 2. Table 2 presents the selected scales. There are two exceptions despite that we try to align the scales with that of the raster layer. First, scale $1 : 10{,}000$ is selected because this is the base scale of the vector layer. Second, the smallest scale selected is $1 : 384{,}000$ because there is no change anymore for an even smaller scale; although there are still 469 areas, they are separate and cannot be merged.

Figure 2 shows our juxtaposition comparer, where the left map and the right map respectively use the multi-scale and the vario-scale vector layer.[5] Both maps use the multi-scale raster layer as the background. In order to make the background layer visible, the foreground layers are set to be semi-opaque, where the layer opacities can be tuned by map users on the setting panel. Figure 3 shows the setting panel, which can be invoked by clicking the three dots in the upper right corner of the comparer.

Table 2. The levels of the multi-scale vector layer. Column S_t represents the selected scale from the vario-scale vector layer for each level. Column S_{range} represents the range of scale denominators for each level. Column n_{areas} represents the number of areas for each level.

S_t	S_{range}	n_{areas}
384000	$[384000, \infty)$	469
192000	$[192000, 384000)$	783
96000	$[96000, 192000)$	3132
48000	$[48000, 96000)$	12531
24000	$[24000, 48000)$	50126
12000	$[12000, 24000)$	200504
10000	$(0, 12000)$	288726

Figure 2a shows the comparison at scale $1 : 192{,}000$. At this scale, the left map and the right map are the same because this level of multi-scale vector layer is selected from the state of the vario-scale vector layer at scale $1 : 192{,}000$. After zooming out to scale $1 : 235{,}254$ (see Figure 2b), the area objects of the left map only shrink because the level spans over range $[192000, 384000]$ of scale denominators, while the area objects of the right map merge continuously.

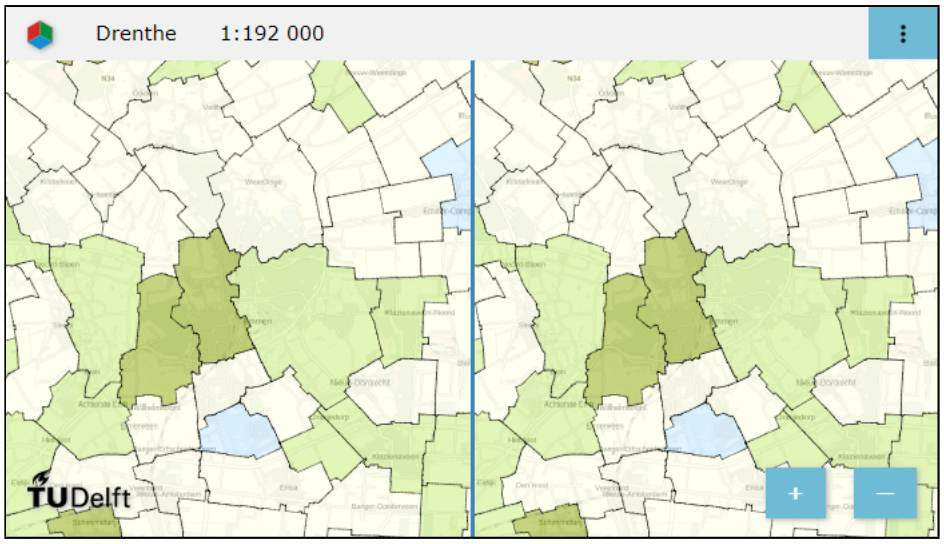
2.4 Dynamic transition using framebuffer

Section 2.3 made the foreground layers semi-opaque so that the background layer is also visible. However, this functionality is flawed during dynamic transition without using a framebuffer. In the following, we explain the flaw using the case of Figure 4 as an example. Then, we show how to avoid the flaw by using a framebuffer.

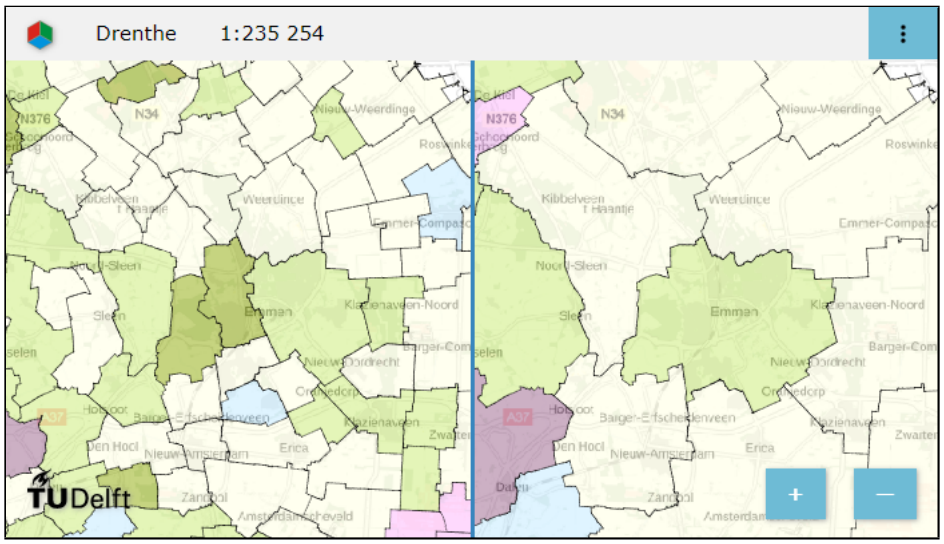
According to Figure 4, the base level of the layer, an SSC can be built and be sliced to generate a representation at any scale (see Figure 5). This generated representation is fine when the opacity is 1 (see Figure 5a). The generation can be also understood as a ray shooting a problem, where the faces stop the rays will be drawn to the screen Meijers et al. (2020); see Figure 6a. However, when generating a semi-opaque representation, we will have Figure 7b, while we expect the result of Figure 6b. The principle is as following. In our implementation, we use a slicing plane to intersect with the SSC and to find the intersected polyhedrons (see Figure 5b). Then, those polyhedron's faces that are below the slicing plane and are facing up will be drawn to the screen. If the faces of the forest, the farmland, and the water are drawn first, and the the face of the road is drawn afterwards, then we get the expected result (see Figure 6b). The reason is that, when a new face is being drawn, the GPU does a depth test against the faces already drawn to the screen. A pixel of the new face will be drawn only when the pixel's z-coordinate is larger than that of the pixel on the screen. For the same reason, if the road's face is drawn beforehand, the enlarged neighboring faces will be drawn on top of it (see Figure 7b). An intuitive solution

[4] More information of TOPNL dataset is available at https://www.pdok.nl/introductie/-/article/basisregistratie-topografie-brt-topnl. Accessed: October 30, 2020.

[5] The juxtaposition comparer is available at https://pengdlzn.github.io/webmaps/2020/10/merge/drenthe-comparer-juxtaposition.html. Accessed: October 30, 2020.



(a) The maps at scales 1 : 192,000



(b) The maps at scale 1 : 235,254.

Figure 2. The juxtaposition comparer with a multi-scale layer (left) and a vario-scale layer (right) as the foregrounds.

to avoid the case of Figure 7b is to sort all the polyhedrons according to their maximum z-coordinates. Then, we draw the faces of each of the polyhedrons according to the order. However, this solution requires fetching and sorting all the relevant polyhedrons before drawing, which is inefficient.

In order to guarantee the visualization of Figure 6b and draw efficiently, we use a framebuffer so that we can fetch the polyhedrons and draw the faces at the same time. As explained in Section 1, the data of the polyhedrons are stored in chunks on the server. We fetch the chunks of the intersected polyhedrons from the server (see Meijers et al., 2020), we opaquely draw the relevant faces into the framebuffer once a chunk is ready on the client side. After all the relevant faces are drawn into the framebuffer, the framebuffer actually stores the image of Figure 5a. Then, we draw the image of the framebuffer to the screen semi-opaquely (see Matsuda and Lea, 2013, pp. 392–403). In this way, our web map gives the feeling of smooth transition during zooming without the flaw mentioned before.

3. Using swipe juxtaposition comparer to show the dryness of area objects

Based on the juxtaposition comparer proposed in Section 2, this section develops a swipe juxtaposition comparer to compare two maps of area objects with dryness attribute (see Figure 8). Using the swiper, users can tune the widths of the two maps. A legend is also added to relate the colors to the dryness values. The goal of using swipe juxtaposition comparer is to compare a map with discrete zooming (thematic layers are simply switched on/off) and a map with smooth zooming (continuous changes are applied to change between the thematic layers). The thematic layers are farmlands (Figure 9), water supply areas (Figure 10), and water authorities (Figure 11). Again, the multi-scale raster layer, presented in Section 2.1 is used as the background to provide more geographic information. The reaming of this section shows our progress so far.

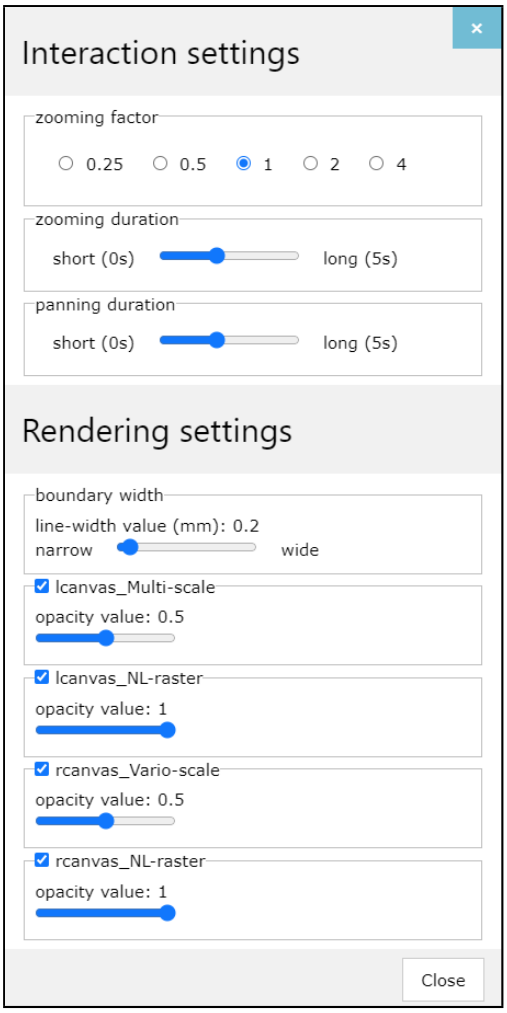


Figure 3. The setting panel of our web map tool. *Zooming factor* specifies how much should be zoomed when a user scrolls the mouse wheel. *Zooming duration* specifies the amount of time for a scrolling. *Panning duration* specifies the amount of time for the panning after the user releases the mouse button. Furthermore, users are allowed to set the boundary width of the area objects, to toggle on/off the layers, and to tune the opacities.

The thematic layers with dryness values are stored in some shapefiles, while the SSC should be stored as text in the format of the Wavefront .obj file.[6] Therefore, we need to convert the shapefile data to the SSC data. We did the conversion by software Feature Manipulation Engine (FME) Desktop[7]. FME provides many functionalities and allows us to make a workflow to convert data. For example, Figure 12 shows our workflow of converting the shapefile data to the data stored in PostgreSQL database. Shapefile *Aanvoergebieden2_drenthe* contains 36 water supply areas of Drenthe. Based on these water supply areas, the TopologyBuilder builds 1,632 nodes, 2,767 edges, and 95 faces, where the edges are extracted from the boundaries of the

[6]An illustration of Wavefront .obj file is available at https://en.wikipedia.org/wiki/Wavefront_.obj_file. Accessed: October 30, 2020.

[7]Software FME Desktop can be found at https://www.safe.com/fme/trial/. Accessed: October 30, 2020.

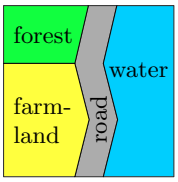


Figure 4. The base level of a vector layer.

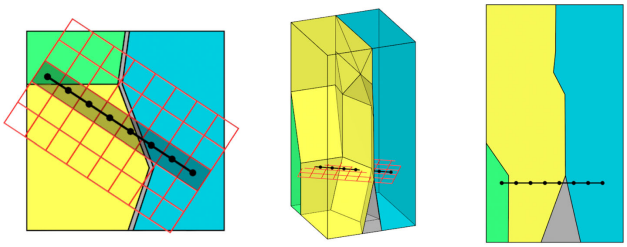


Figure 5. A representation generated by slicing an SSC. (a) The generated representation and the slicing plane, where the slicing plane is represented by the grid. (b) The position of the slicing plane in the SSC. (c) A side view of the slicing. This figure is taken from Meijers et al. (2020).

faces. The AttributeManagers manage the attributes from the shapefile. In our case, we need to maintain the dryness values. Then, the Generalizer_edge simplifies the edges, and the Generalizer_face simplifies the boundaries of the faces. Indeed, it is problematic to simplify the edges and the faces independently because the simplified edges and the simplified boundaries of the faces are not consistent anymore. The simplification should be improved by implementing our own prototype. Finally, the edges and the faces are stored in the PostgreSQL tables ab2_edge and ab2_face.

Then, we wrote a Python script to read the edges and the faces from the PostgreSQL database. The script triangulates the faces and output the triangles with attribute *dryness* into an obj file. The faces are triangulated because GPU takes triangles as input to draw images to the screens. Although the triangles of the faces will be drawn by GPU, the boundaries of the faces are invisible because their widths are 0. Therefore, we also save the edges into the obj file. At the client side, some rectangles with a small width will be generated based on the edges. Then, the rectangles will be triangulated so that the GPU can display the edges.

We have prepared the SSC data for the three thematic layers.[8] However, we have not got the dryness values for all the layers, so we currently used some fake values. Furthermore, we use GitHub as a server to host our SSC data so that users can access the web map at anytime.[9]

[8]The swipe juxtaposition comparer with the three thematic layers is available at https://pengdlzn.github.io/webmaps/2020/10/merge/drenthe-comparer-juxtaposition-swipe.html. Accessed: October 30, 2020.

[9]A guidance of using GitHub as a server is available at https://pages.github.com/. Accessed: October 30, 2020.

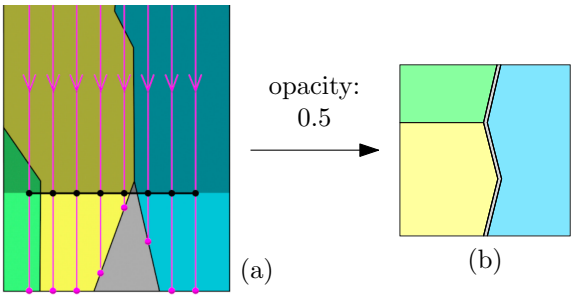


Figure 6. The expected representation when drawing semi-opaquely. (a) is taken from Meijers et al. (2020).

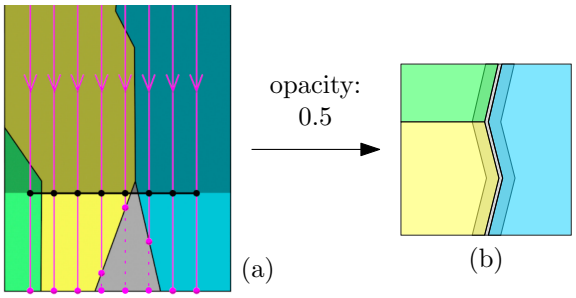


Figure 7. The real representation when drawing semi-opaquely without using a framebuffer.

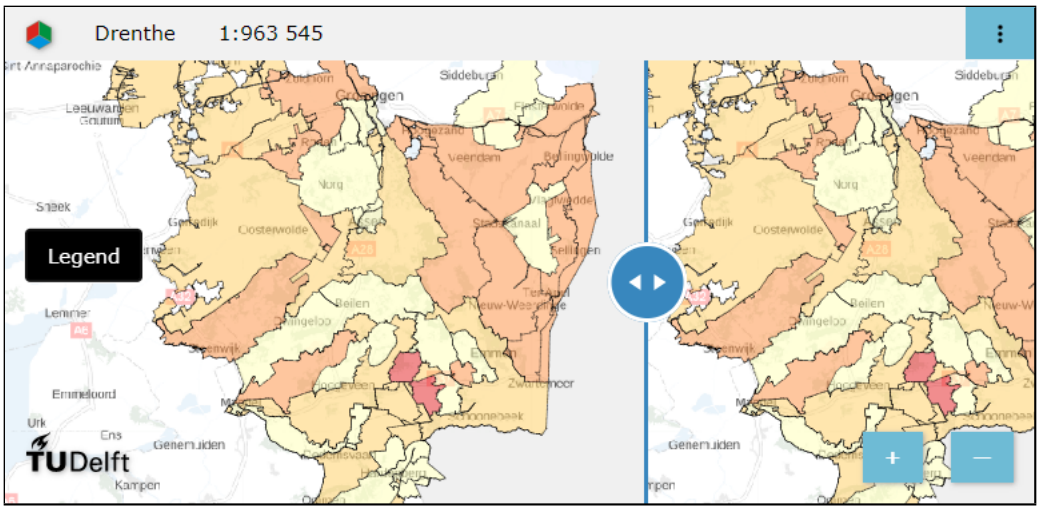


Figure 8. The swipe juxtaposition comparer. The thematic layer presented is for water supply areas.

4. Conclusion and future work

The paper presents our web map juxtaposition comparer, which will be used for a usability study to see if a multi-layer web map with vario-scale layers are better than a multi-layer web map with multi-scale layers in helping users to keep their context during zooming. There are many topics to be explored.

First, we want to simply the faces and the edges with a larger tolerance so that there is less data to load when the scale is small. Our current simplification is attained by FME, and we have to use a small tolerance to avoid topological problems. As we cannot change the simplification implementation of FME, we must do the simplification by implementing our own prototype. Second, we currently simplify the faces and the edges independently, which brings inconsistency of the data. For this reason, we also need to develop our own method to simplify the data. Third, we wish to smoothly change between two thematic layers instead of simply switching between them. This work is difficult when the two layers are quite different, which is our case. For example, for one layer, there are gaps between area objects, and, for another layer, the area objects are aggregates of the former. To solve this problem, we could use a buffer-based method (Peng and Touya, 2017) or a Voronoi-diagram-based method (Ai and van Oosterom, 2002) to generate a region for each area object to grow so that continuous changes between thematic layers can be achieved. For the growing, the morphing can applied, where the corresponding points between the boundaries of the area object and the region can be built by a dynamic algorithm (Peng et al., 2016; Nöllenburg et al., 2008).

Acknowledgements

This work is part of the research programme Maps4Society with project number 17644, which is (partly) financed by the Dutch Research Council (NWO).

References

Ai, T. and van Oosterom, P., 2002. GAP-tree extensions based on skeletons. In: D. E. Richardson and P. van Oosterom (eds), *Proc. 10th International Symposium on Spatial Data Handling (SDH)*, Springer Berlin Heidelberg, Ottawa, Canada, pp. 501–513.

Chimani, M., van Dijk, T. C. and Haunert, J.-H., 2014. How to eat a graph: Computing selection sequences for the continuous generalization of road networks. In: *Proc. 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACMGIS)*, Dallas, TX, USA, pp. 243–252.

Huang, L., Meijers, M., Šuba, R. and van Oosterom, P., 2016. Engineering web maps with gradual content zoom based on streaming vector data. *ISPRS Journal of Photogrammetry and Remote Sensing* 114, pp. 274–293.

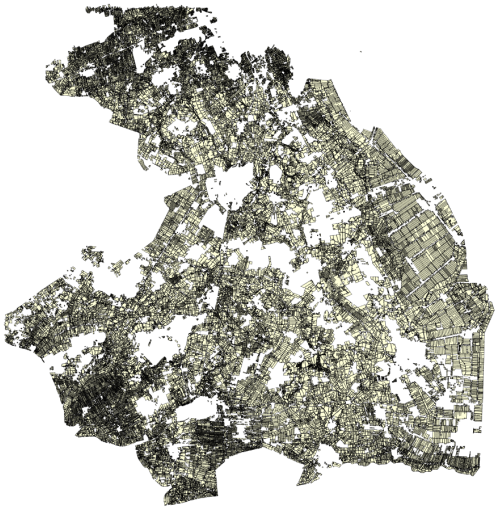


Figure 9. The thematic layer of farmlands of province Drenthe. A fake dryness value is used for all the area objects. The scale denominator of this scale spans over range $(0, 500000]$.

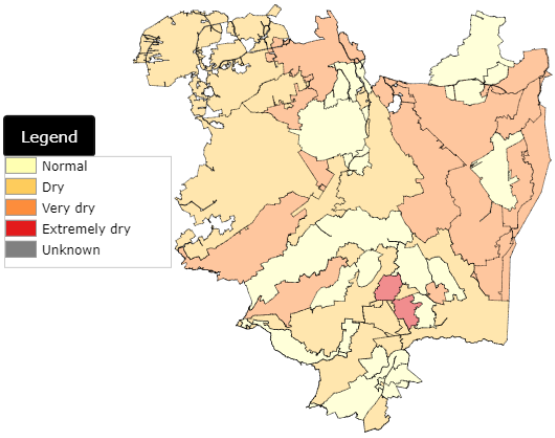


Figure 10. The thematic layer of water supply areas in or around Drenthe. Some random dryness values are used for the area objects. The scale denominator of this scale spans over range $(500000, 1000000]$.

Li, J., Ai, T., Liu, P. and Yang, M., 2017. Continuous scale transformations of linear features using simulated annealing-based morphing. *ISPRS International Journal of Geo-Information* 6(8), pp. 1–15.

Lobo, M.-J., Pietriga, E. and Appert, C., 2015. An evaluation of interactive map comparison techniques. In: *Proc. 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI)*, Association for Computing Machinery, New York, NY, USA, pp. 3573–3582.

Matsuda, K. and Lea, R., 2013. *WebGL programming guide: Interactive 3D graphics programming with WebGL*. OpenGL series, Addison Wesley Publishing Company.

Meijers, M., van Oosterom, P., Driel, M. and Šuba, R., 2020. Web-based dissemination of continuously generalized space-scale cube data for smooth user interaction. *International Journal of Cartography* 6(1), pp. 152–176.

Figure 11. The thematic layer of water authorities of The Netherlands. A fake dryness value is used for all the area objects. The scale denominator of this scale spans over range $(1000000, \infty]$.

Nöllenburg, M., Merrick, D., Wolff, A. and Benkert, M., 2008. Morphing polylines: A step towards continuous generalization. *Computers, Environment and Urban Systems* 32(4), pp. 248–260.

Peng, D. and Touya, G., 2017. Continuously generalizing buildings to built-up areas by aggregating and growing. In: *Proc. 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics (UrbanGIS)*, ACM, Redondo Beach, CA, USA., pp. 1–8.

Peng, D., Wolff, A. and Haunert, J.-H., 2016. Continuous generalization of administrative boundaries based on compatible triangulations. In: T. Sarjakoski, Y. M. Santos and T. L. Sarjakoski (eds), *Proc. 19th AGILE Conference on Geographic Information Science, Geospatial Data in a Changing World*, Lecture Notes in Geoinformation and Cartography, Springer, Helsinki, Finland, pp. 399–415.

Peng, D., Wolff, A. and Haunert, J.-H., 2020. Finding optimal sequences for area aggregation—A* vs. integer linear programming. *ACM Transactions on Spatial Algorithms and Systems* 7(1), pp. 1–40.

van Kreveld, M., 2001. Smooth generalization for continuous zooming. In: *Proc. 5th ICA Workshop on Generalisation and Multiple Representation (ICAGM)*, Beijing, China, pp. 1–8.

van Oosterom, P. and Meijers, M., 2014. Vario-scale data structures supporting smooth zoom and progressive transfer of 2D and 3D data. *International Journal of Geographical Information Science* 28(3), pp. 455–478.

Veenendaal, B., Brovelli, M. A. and Li, S., 2017. Review of web mapping: Eras, trends and directions. *ISPRS International Journal of Geo-Information* 6(10), pp. 1–31.

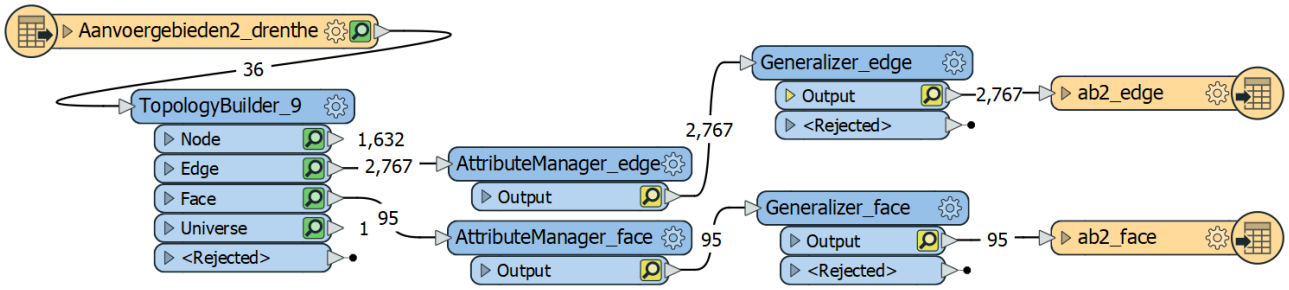


Figure 12. A workflow of FME to convert the data from a shapefile to the data stored in a PostgreSQL database.