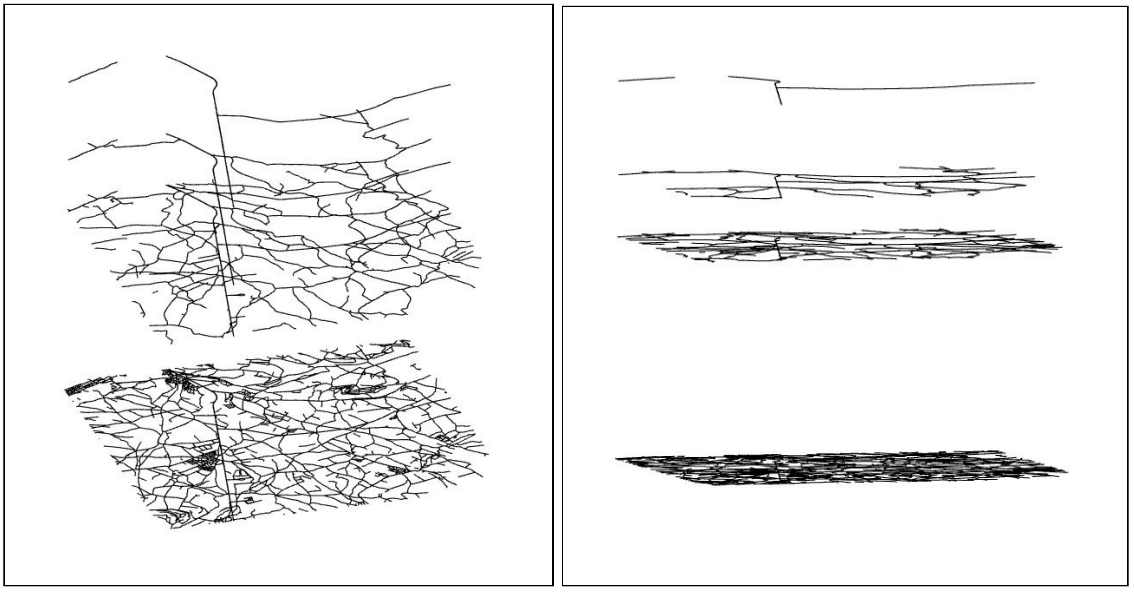
Improving vario-scale implementation based on needs of Kadaster topographic data users

Technical report
GEO1101 - Synthesis Project

Freek Boersma, Shenglan Du, Pim Klaassen & Pablo Ruben
MSc Geomatics, Faculty of Architecture and the Built Environment, TU Delft



TU Delft

Abstract

Vario-scale is a new mapping technique which automatically generalizes maps from a base layer of faces. Applications of vario-scale are continuous, smooth zoom in web maps, multi-scale representation in one map and being able to generate maps at arbitrary scale. Also, this would only require having to maintain the dataset at the highest scale level, since all other scales are derived from it.

Potentially, vario-scale could be an alternative for current web maps and generalization algorithms. The Dutch national mapping agency, Kadaster, currently employs its own generalization process. However, they would like to know whether the users of their topographic datasets are interested in vario-scale. At this moment, there is a working implementation of vario scale (made by dr. ir. Martijn Meijers). This implementation, however, is still lacking in, for example, cartographic quality. Therefore the research question in this project is: *how can the implementation of vario-scale be improved to better meet the needs for end users of Kadaster topographic data?*

This question is answered by questioning surveying users of Kadaster data on what they would like to see improved about the existing implementation. Combining this with an exploration of the current software leads to an attempt at improving the current implementation. The project goal is set as enabling the road network visualization and mobile map adaptation. Road network visualization is achieved by building the roads space scale cube and overlay with the background area at the front-end. Mobile map adaptation is realized by creating the touch screen interaction between the device and the user. Finally, a validation survey is conducted to examine the difference between the original vario scale implementation and the adapted one.

Acknowledgements

In this section we would like to express our thanks to a number of individuals who helped us during the research project.

First of all, we would like to thank Martijn Meijers for his constant guidance and support. He has walked us through all stages of this project. He was always willing to share his knowledge and insights with us concerning this topic. Without his tutoring and instruction we wouldn't overcome those technical obstacles and this project couldn't have reached it present form.

We would like to thank Daniël te Winkel, Alex de Jonge and Anouk Huisman from Kadaster, for their continuous commitment towards our project. The visit to Kadaster and being able to attend the user meeting were broadening experiences. The gatherings and feedback sessions were very helpful contributions to our research.

Furthermore, we would like to thank Edward Verbree for his tutoring during the project. His advice was inspiring and always motivated us to look for new challenges.

Last but not least, we also want to express our gratitude to the members of the BRT-user group. Being allowed to assist one of the meetings was a great insight into practitioners' world - more importantly, the participation to the surveys was essential for this project.

Table of contents

1. Introduction

Over the last decades, geographic information has become increasingly digital (Dilo et al., 2009). One of the most used ways to visualize geographic information is with topographic maps. Topographic maps were always printed on paper. The digitization of GI has led to the topographic data being processed with and maintained on computers. Along with this came the option to visualize the digital topographic maps with online viewers. These viewers allow users to access to all kinds of different maps.

Often, national mapping agencies (NMA's) are tasked with the responsibility of making topographic maps for the corresponding country. These NMA's produce maps on different scales. Up until recently, these maps on different scales were created apart from each other. Over the last two decades, research in map generalization has taken a flight (Van Oosterom & Meijers, 2011). Kadaster, the Dutch NMA, has created its own generalization algorithms in order to create their lower scale maps. They maintain their base data on their highest scale level (1:10 000, called TOP10NL) and create the lower scale maps (TOP50NL, TOP100NL, etc.) with generalization algorithms. All created topographic maps are uploaded to, and can be viewed on, the Dutch geoportal, PDOK (https://www.pdok.nl/).

Web viewers most often utilize a multi-scale representation, which means that they "represent objects at different resolution levels and support modification across resolution levels" (NCGIA, 1993). However, this solution often requires storing separate map objects for the separate map scales with no reference between two adjacent scales, which will produce abrupt changes during map zooming process (Meijers, 2011). Moreover, current implementations of automatic generalization processes sometimes cause unwanted results. This, combined with the abrupt scale changes during zooming, can lead to lack of orientation when using web viewers.

In order to tackle these problems, the vario-scale technique has been developed. Vario-scale is a method which aims to generate all wanted scales in a smooth way. Instead of storing separate datasets at different scales, vario-scale structures datasets with level of details for a whole scale range (Van Oosterom & Meijers, 2014). It avoids abruptions as much as possible, enables smooth generalization, and, reduces the dataset redundancy as well. "Advantages of the variable-scale approach are that only one dataset needs to be managed and that data can be displayed at any scale" (Meijers, van Oosterom & Quak, 2009). Moreover, implemented properly together with web cache technology, vario-scale is able to solve the problems of losing track or losing connection, thus providing even more cartographically pleasant experience for users.

Vario-scale visualization still stays on a research level (Van Oosterom & Meijers, 2014). It has not been applied widely among companies or citizens. Therefore its benefits to suppliers and users still remains unclear. The Kadaster would like to know whether the users of their topographic datasets are interested in the vario-scale technique. This project aims to bridge the vario-scale research with the end users. Instead of studying the vario-scale from a pure technical view, this project will put focus on the end users side. The central question is therefore *how can the implementation of vario-scale be improved to better meet the needs for end users of Kadaster topographic data?*

In order to answer this question, we will look at this problem from the technical side and from the user's side. These two sides correspond with two sub-questions that need to be answered in order for us to answer the research question. These questions are: how does the current implementation of vario-scale work, and what are use cases for current Kadaster data users and what can be changed in order to make vario-scale suitable for these use cases. First, we will study the concept of map generalization and the theory behind vario-scale. Then the current implementation of vario-scale will be explored. The users of the topographic data of the Kadaster will be questioned on vario-scale through a survey. Based on the comprehension of the current implementation and the needs of the users, an analysis will be made of how the current implementation can be improved. Based on this strategy, a demo will be created.

This report is structured as follows: in chapter 2 the vario-scale concept is studied and the current implementation will be explored. In chapter 3 the results of the user survey are presented. In chapter 4 the results of the survey, combined with our comprehension of the current implementation, will be used to decide on a strategy to improve the current implementation. In chapter 5 the results of our efforts will be presented. Chapter 6 will provide conclusions and a discussion on the methods used.

2. Vario-scale and its implementation

In this chapter the vario-scale concept will be introduced. First the theory will be explored. It is shown that vario-scale can be an alternative to map generalization and will enable a true smooth zoom in web mapping. After that, the current implementation of vario-scale will be explained. This will lead to an insight in how the software works and into technical possibilities for improvement.

2.1 Arguments for using vario-scale

National mapping agencies generate maps in different scales. One traditional method of multi-scale map visualization is to store the datasets as separate sets of objects for different map scales. When these maps are viewed sequentially in a web viewer, this will always produce abrupt changes during map zooming (Meijers, 2011). Vario scale is a method to generate maps which have a truly smooth zoom between them. This also means that for every arbitrary scale a map can be made (if the scale is lower than that of the base layer) (TU Delft, 2018). It avoids abruptions as much as possible, enables smooth generalization, and reduces the dataset redundancy as well. "Advantages of the variable-scale approach are that only one dataset needs to be managed and that data can be displayed at any scale" (Meijers, van Oosterom, Quak, 2009).

2.2 Vario-scale concept: tGAP and SSC

Vario-scale depends on two different concepts: the topological Generalised Area Partitioning-structure (tGAP) and the Space-Scale Cube (SSC). The tGAP structure is capable of effectively running automatic generalization on the single base dataset (Meijers, van Oosterom, Quak, 2009). Rather than storing separate map objects at different scale, the generalization result in each scale is stored in the tGAP tree structure. From this structure, a 3-dimensional cube can be built, where the x,y-axes represent the geographical extent of an area and the z-axis represents the scale.

The main concept behind the tGAP structure is that objects are given an importance value. In this case, objects refer to faces. The structure starts with a planar partition at the largest scale. Importance can be computed by multiplying the area and the weight assigned to the class of the polygon object. For every generalization step, the object with lowest importance value is merged with the most compatible neighbor. This merged object is given a new identity and, if the feature class of the neighbor is different, is given the feature class of the neighbor. The importance of this object is recomputed. The hierarchy is stored in what is called the tGAP-tree. In figure 1, a small tGAP-tree is shown, alongside a visualization of a step of the generalization process. Aside from merging, objects may also be split and assigned to multiple neighbors. This will not result in a tree structure, but in a directed acyclic graph (DAG). In parallel, the boundary of the affected polygons may be simplified. This is stored in the BLG-tree (binary line generalization). The tGAP-DAG along with the BLG-tree is referred to as the tGAP structure (van Oosterom & Meijers, 2011).

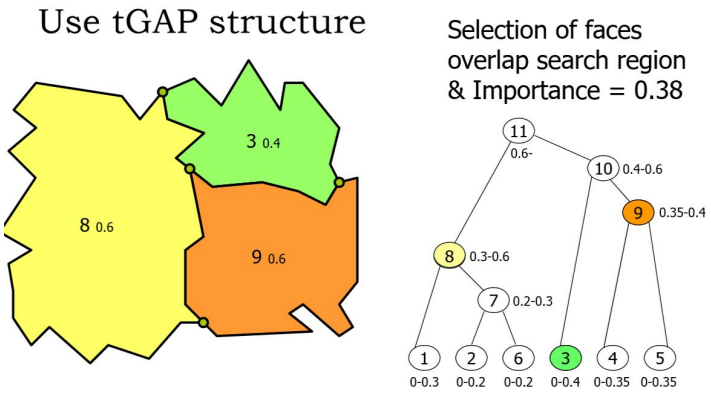


Figure 1. tGAP structure (Meijers et al., 2012)

From this tGAP structure, we can deduce that when a spatial dataset has *n* features, there are *n-1* steps in the generalization process, since in every generalization step one polygon disappears until there is one left. This already gives *n-1* representations of the same area in different scales, in effect leading to a multi-scale representation (see figure 2 left). From this, a space-scale cube (SSC) can be derived (figure 2 middle). The faces of the levels are extruded to its neighboring level, leading to a three-dimensional representation of space and scale. Faces on the map are now represented by polyhedrons. The idea of this cube is that at any level, one can take a 'slice' through the cube to obtain a map at arbitrary scale.

However, when zooming out, the changes in the map will still be abrupt. Because of the extrusion, the space between the levels is still the same and is in effect the same as an *n-1* multi-scale representation. The *smooth* SSC is obtained by making the merges or splitting operations continuous between the steps. In this case the features will disappear continuously, see figure 2 right. This means that a small delta in scale will result in a small delta in changes in the map (van Oosterom & Meijers, 2011).

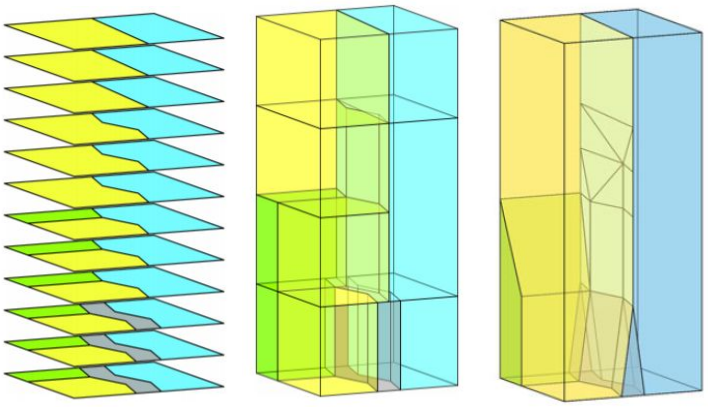


Fig. 2. Space scale cube concept: multi-scale, SSC, smooth SSC (van Oosterom et al., 2014).

The main advantages of using the SSC, is that maps can be obtained at each scale level by taking a slice. Thus it can serve as an alternative to current map generalization processes. It also can be implemented as a smooth zoom in web mapping services. A third advantage is that one can make non-horizontal, tilted slices through the cube to obtain multi-scale maps. An application of this would be if one would want to make a map which is very detailed near and less detailed far away.

The tGAP structure can be implemented and visualized, see section 2.4. However, there are some current drawbacks with using vario-scale. First, it has not yet been implemented on a very large dataset, like the TOP10NL topographic 1:10 000 dataset of the Netherlands. However, Meijers et al. (2015) suggest a creating a fieldtree organization, which divides the workload of processing. It subdivides space into different fields, and different scales. Fields on a higher level have a shifted origin such that the borders do not overlap. Only features completely in one field will be processed. All fields can be processed in parallel, which results in an efficient process. Still, this has not yet been tested with the whole TOP10NL dataset, but it shows promising results.

A big point which can be improved is the resulting cartographic representation that the vario-scale structure offers. What objects remain at a certain scale level is entirely dependent on the importance values assigned to the polygons. Different importance will thus result in different maps. In the current implementation, the only thing shown is the actual slice through the SSC. This not necessarily makes a good map. Symbols are missing, labels are missing, main road networks do not stay connected and neighboring houses do not aggregate into a built-up area. These, among other things, are topics that can be investigated further.

Networks, such as road networks, rail networks or hydrography networks, are important for orientation in maps (Suba et al., 2016; Groeneveld, 2018). Ideally, at a certain scale, if a part of the network of a certain hierarchy level is shown, all parts of the network of that hierarchy and higher should be shown. For example, if a map shows regional roads, you want it to also show all highways. Algorithms for road networks and hydrography networks have been devised (Suba et al., 2016; Groeneveld, 2018). They also generalize polygons into lines at a certain scale. Implementing these technologies in a working vario-scale environment has been done to some extent, see section 2.4.

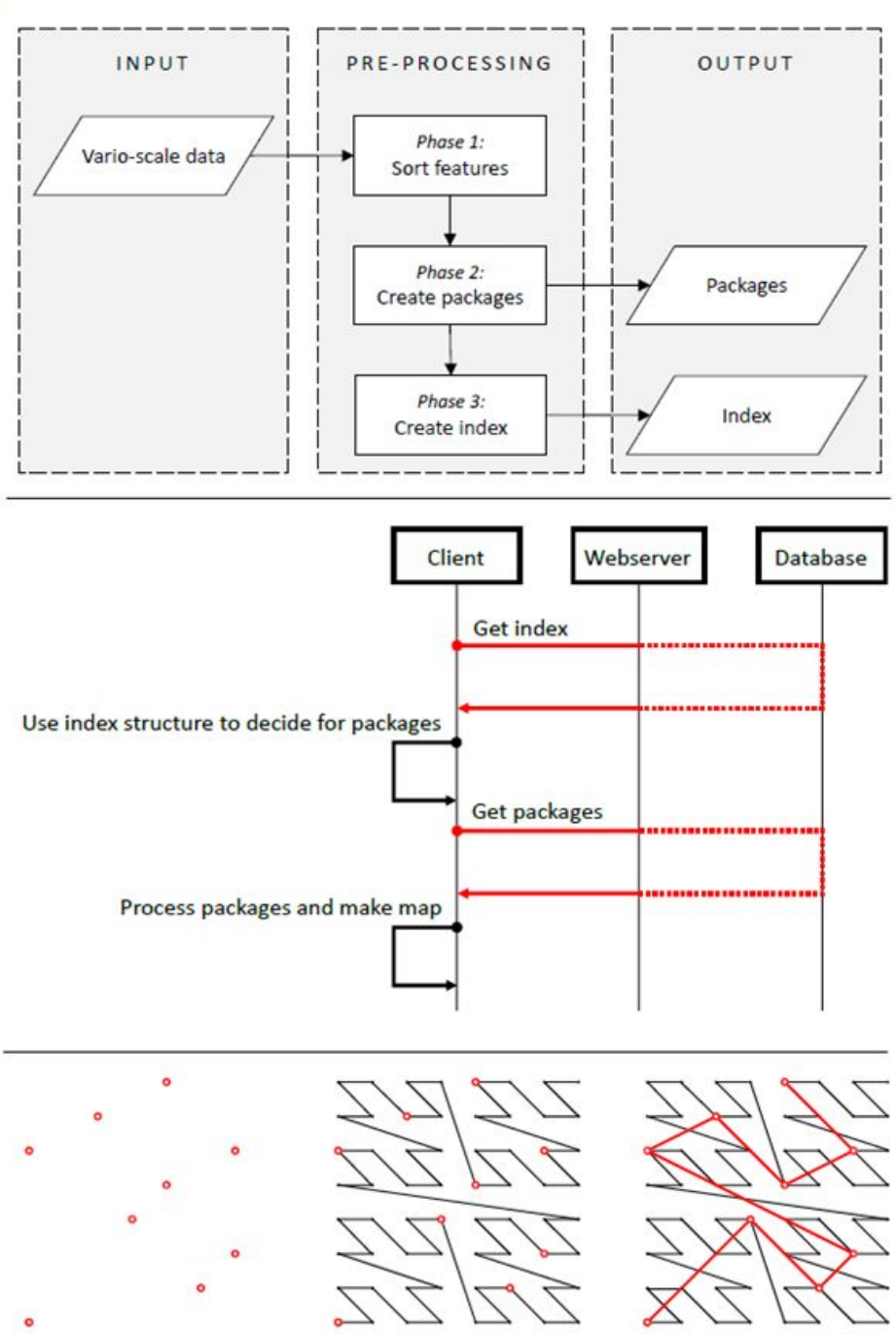


Figure 3: Methodology of storing vario-scale data and the client-server architecture in three steps (Rovers, 2016).

A better cartographic accuracy at smaller scales can also be obtained by using a constrained tGAP. This uses a reference map of smaller scale than the base data (van Oosterom et al., 2014). In this process, the smaller scale map serves as a target on which generalization decisions are based. Using these constraints, more accurate intermediate maps can be devised. However, since vario-scale can be seen as a means to replace current map generalization processes, this might not be a good strategy. On the other hand, the constraints used could lead to knowledge of certain parameters used for the generalization, which could be used for future generalization purposes.

2.3 Recent developments

Recent years have seen some new developments in vario-scale. Most of these efforts concern making the data transfer more efficient, or deal with visualization. Rovers (2016) has implemented a client-server method with the goal to reach efficient communication for vario-scale data, without too many redundant data transfers. The idea is that data that has already been cached in the browser does not have to be loaded again. Only the data that is newly requested will be transferred from the server to the client. To reach this objective, the system needs to group data likely to be used together into packages on the server. After that, the client cache is used to reuse the data already loaded. Newly requested information should be retrieved by the client from the server using a spatial index.

Packages are created by using a space-filling curve, for example a Hilbert or Morton curve. Centroids of the objects are used as a reference for the curve. From this a linear ordering in the objects can be obtained. An optimal package size should be determined on the client side, to find a balance between package size and response time. After this an index is created on the packages using an R-tree. The index can be retrieved by the client using a HTTP GET request. Using this index, the client can determine which packages (the majority of the data needed) it actually has to request. Then these packages need to be filtered on geographic location and level of detail. The process is visualized in figure 3 (Rovers, 2016).

After the wanted data from the SSC is retrieved from the server, the data needs to be rendered on the screen. Driel (2015) has devised a method where the intersection of the SSC is explicitly done and the result is rendered using an OpenGL based interactive application. This model also contains subdividing data and storing in a database, in order to send chunks of data to the client. This model, however, divides the data based on an octree-structure. Because in an SSC the data is not uniformly distributed (more detail generally at the bottom), this structure would be practical.

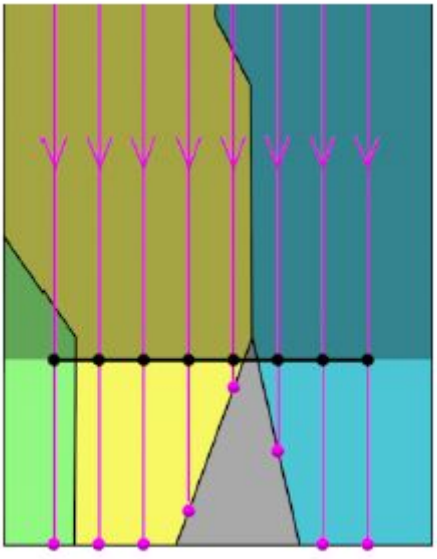


Figure 4: Pixels are projected downwards until it reaches the boundary of the polyhedron it intersects (Driel, 2015).

The intersection of the SSC is made explicit by Driel (2015). A 2D raster, a representation of the pixels, can be imagined through the cube. Instead of doing a direct point in polyhedron test and assigning the pixels the according color, the pixel centroids are projected downwards. Everything above the raster is not considered, it is "clipped off". The projection downwards (along the z-axis) then first encounters the boundary of the polyhedron the pixel is located in (see figure 4). This method is preferred over the "simple" point in polyhedron method because only an image is needed as an end product, which is suited to the GPU. The input data is clipped by a box shaped clipping region. All fragments above the x,y viewing plane are discarded. The fragments below are sorted by z value, and per pixel, the fragment closest to it in z-value in that x,y position is kept (Driel, 2015). This visualization method is used in the current implementation of vario-scale, see section 2.4.

Xu (2017) builds upon the above two studies to try to provide a "technology for the smooth and timely rendering of large SSC datasets that is also applicable for the domestic consumer". The goal is stated as the implementation of a web-based service with a preprocessor that scales well, enables fast transmission of data, also with "smart fetch" (no double requesting of data). The implementation follows the user's action on the web application using JavaScript and uses this to fetch the needed information. The SSC is first put into an obj-file, and then transformed into a bin-file. After, WebGL is used to create a rendering program that runs on the GPU of the client's computer. The process is summarized in figure 5.

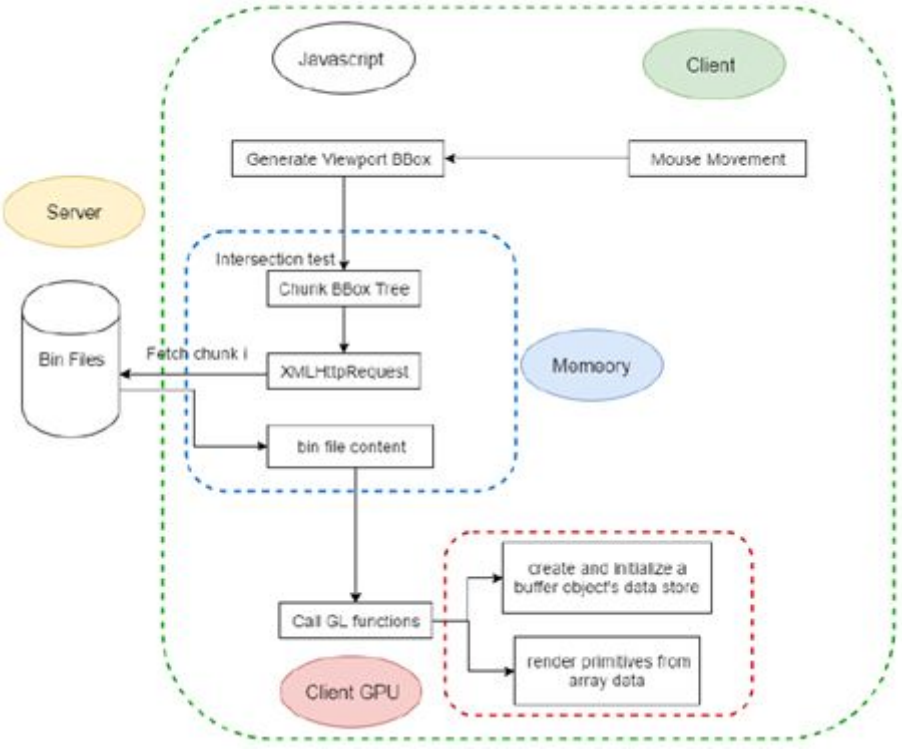


Figure 5: Interaction between client, server, memory and GPU (Xu, 2017).

2.4 Current implementation of vario-scale

This section aims at providing a technical summary of the TU Delft Vario-scale Software Development Kit (SDK). More specifically, it explains the process in which a vario-scale map

is created from dataset to web viewer. The process is described in a stepwise manner, covering the toolkit, preprocessing, topological Generalized Area Partition (tGAP) creation, space scale cube creation, binary format conversion and the implementation of a web viewer. These terms will all be explained in the following section.

2.4.1 Toolkit

The TU Delft Vario-scale SDK consists of multiple modules. These modules are software packages in a Python environment, each one responsible for a different task. They are described shortly below:

- ***tgap module*:** core of the whole SDK. From here, all other modules are coordinated.
- ***connection***: sets up the connection with a database, to send or retrieve data.
- ***sink*:** used to create table oriented output.
- ***topomap*:** represents data as a topological structure.
- ***simplegeom*:** represents data as geometrical objects.
- ***oseq***: used to create ordered sequences for Python**.**
- ***splitarea*:** splits faces in case of case of enabled network generalization.
- ***tri*:** creates a Delaunay triangulation.
- ***geompreds*:** performs orientation and incircle tests.

All of these modules are connected to each other, this is demonstrated in figure 15. Furthermore there are some software dependencies which are all open source and freely available:

- ***Python 2.7**:* language of the SDK
- ***PostgreSQL**:* database management system
- ***PostGIS**:* provides spatial objects to PostgreSQL
- ***C compiler**:* dependency for the SDK
- ***Some Python libraries**:* dependencies for the SDK

These are all necessary for the SDK used to create a vario-scale map. However, theoretically there are other ways to do this. The explanation below is tuned to the SDK used, but also provides a conceptual description of how the process works.

2.4.2 Preprocessing

The input data has to be a table of edges and a table of faces. In this case, the dataset was downloaded from PDOK as a GML file. This can be processed into the required topological structure using FME or other data conversion/transformation software.

	face_id bigint	feature_class bigint	area double precision	mbr_geometry geometry	geometry geometry(Geometry,28992)	pip_geometry geometry
1	13238	13000	112.239497499772	0103000020407100000100000005000000	0103000020407100000100000005000000	0101000020407100008E7549DAFA10074
2	13231	13000	60.3977430002224	0103000020407100000100000005000000	0103000020407100000100000005000000	0101000020407100008AC1CDADD10074
3	13230	13000	402.784947000109	0103000020407100000100000005000000	0103000020407100000100000005000000	0101000020407100007C14AEC7E010074
4	13229	13000	407.107449999428	0103000020407100000100000005000000	0103000020407100000100000005000000	0101000020407100000AAF1D2CD8A11074
5	13089	13000	67.226326000294	0103000020407100000100000005000000	0103000020407100000100000005000000	0101000020407100000FA7E6A3C7FC8064
6	12892	13000	261.302761000423	0103000020407100000100000005000000	0103000020407100000100000005000000	0101000020407100000DF4F8D17A546074

Figure 6. Face table

	edge_id bigint	left_face_id bigint	right_face_id bigint	start_node_id bigint	end_node_id bigint	geometry geometry(Geometry,28992)
1	26208	0	12994	9332	9333	0102000020407100000200000000(
2	26207	0	12987	9171	9176	0102000020407100000200000000(
3	26206	0	12937	17369	17370	0102000020407100000200000000(
4	26205	0	12928	17367	17359	0102000020407100000200000000(
5	26204	0	12928	17360	17368	0102000020407100000200000000(
6	26203	0	12777	9334	9321	0102000020407100000200000000(

Figure 7. Edge table

The tables are demonstrated in figure 6 and 7 (faces and edges). The face table has a feature class attribute, referring to the class of the object: building, road, water, etc. The area of the face is calculated from the geometry. This value will be used in the generalization process. Furthermore, the face table has two extra geometry attributes which are needed for the generalization process and debugging: a minimum bounding rectangle geometry and a point in polygon geometry. The string in the geometry columns is how PostGIS represents geometries; in the face table these are of the type 'polygon'. In the edge table, the geometry has the type 'linestring'. Additionally, it contains start and end node id's and left and right face id's (these id's refer to the face table). The face with id 0 is the universe face, which is the topological face outside of the area represented. The tables as described above are in the required format of the input data. All modern geo database management systems - open source or proprietary - should be able to transform datasets into this format (assuming the extra attributes are not already present in the database).

The SDK requires some extra databases to configure the settings of the process. These are demonstrated with a user icon in figure 9. The tGAP metadata table should be created to provide the software with some basic information and instructions. It contains the names of some tables which have to be created later on, also the id of the universe face, SRID, initial scale of the original dataset (e.g. 1:10 000) and instructions on how to generalize the dataset.

The fieldtree table needs only the field-size as a user input (the function of the fieldtree will be explained). The last table which needs user input is the feature class mapping table. The input data contains feature class information in the face table. This information is needed to filter faces during generalization. The table contains those class values and also instructions on how to merge/split faces of each class. This is demonstrated in figure 8. In this case, the ranges mean 'split this type of face from step 0 until the end'. The vario code is just a mapping. Multiple different feature classes can be mapped to one vario code. Depending on the input dataset, the user can decide how each feature type should be treated. More of this will be explained in detail in the next section.

	oid	feature_class [PK] integer	vario_code integer	name character varying	rgb character varying	description character varying	ranges character varying
1	19459	10310	5	Important road	230,0,0	Very important road (e.g. highway)	(0,-1,split)
2	19458	10311	5	Important road	230,0,0	Very important road (e.g. highway)	(0,-1,split)
3	19461	10410	4	Regional road	255,170,0	2nd most important road, e.g. regional roads of 1st class	(0,-1,split)
4	19460	10411	4	Regional road	255,170,0	2nd most important road, e.g. regional roads of 1st class	(0,-1,split)
5	19463	10510	3	Local road	255,255,0	local road	(0,-1,split)
6	19462	10600	2	Street	255,255,255	Street	(0,-1,split)

Figure 8. Feature class mapping table

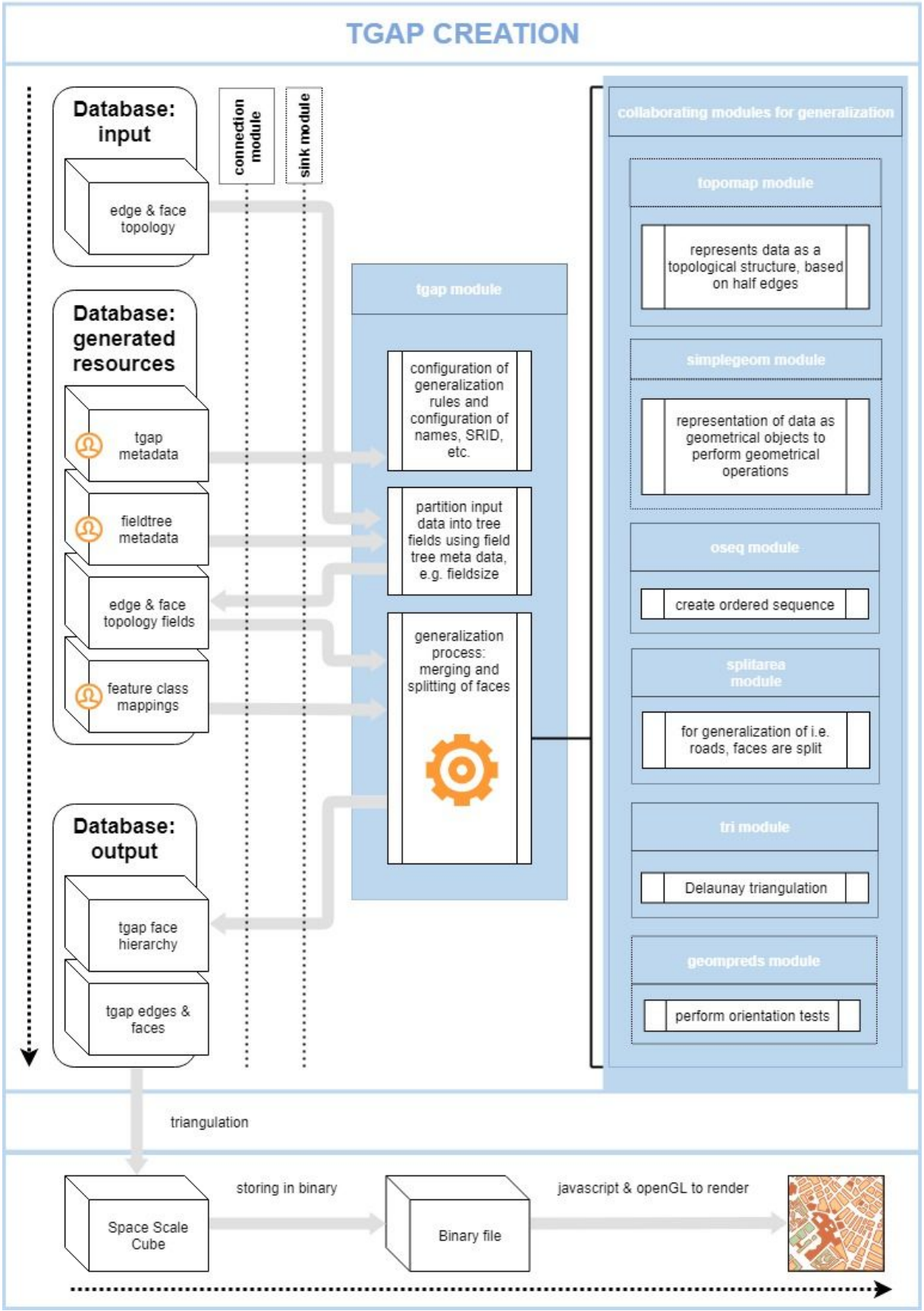


Figure 9. Vario-scale SDK toolkit

2.4.3 tGAP creation

The first thing the program does is partitioning the dataset into a field tree (see figure 10). This is implemented to process bigger datasets. The field partitions space into multiple overlaid grids, of various sizes. Objects are inserted by storing them in the smallest field in which they completely fit. Multiple objects can be stored in one field. By inserting data into a field tree it can be processed in parallel. For the test dataset, this is not necessary because it is relatively small. Therefore the minimum field size was put on 20000 km, this far exceeds the extents of the data set area (9000 km) and therefore all the objects are stored in one field. This can be processed easily by a moderate computer (the model used for this project has 8 GB of RAM).

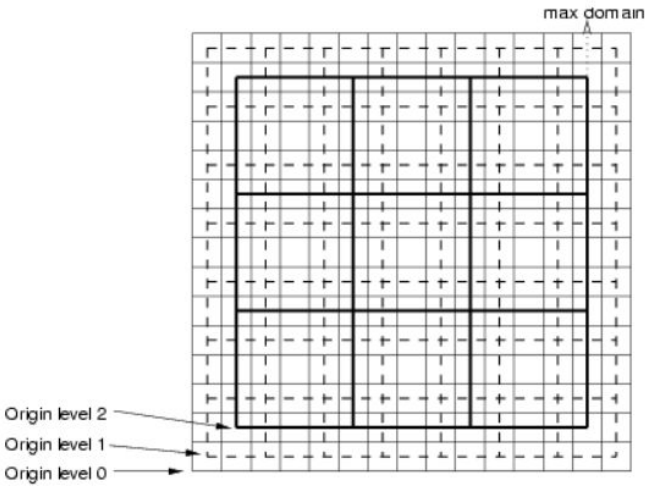


Figure 10. Field-tree.

The generalization process is based on the importance levels of the faces. The importance level of all the initial faces are set up as the value of their areas by default. However, the software provides a possibility to create weighted importance values which are defined by the feature class of a face. For instance, if two faces with equal surfaces have different feature classes, weights are added to their importance. The importance of feature classes can be defined by the user by adding additional metadata. All the faces are stored in an ordered sequence in the form of a red-black tree. This is a type of binary tree which is self-balancing during insertions and deletions. The ordered sequence can be seen as a queue of faces in which the least important face is in the front and the most important face in the back. The faces will be treated in that order. Once a face is processed, it will either be merged or split.

The merge operation will simply merge the face with one of its neighbours. The neighbouring face that is chosen to merge with is the least important face of all the neighbouring faces. The two merging faces will be stored in the tGAP table as geometries containing a ‘step low’ and ‘step high’ value. These values can be seen as the lifespan of a face. The first two merged faces of a dataset will have a ‘step low’ of 0 and a ‘step high’ of 1. Also, there will be a newly merged face with a ‘step low’ of 1. The geometry will be the combined geometries of the two former faces, the importance will be the summation of the two former faces and the feature class will be that of the most important face from which it originated. A good analogy to understand this concept is a stack of chessboards, where the top has only one big square left. On the second board, two squares are merged, on the third board two faces from the second board are merged, and so on. The boards refer to the step values, and to the scale of the map.

The bottom board is the most detailed map, the top one is the least detailed. This is, in a nutshell, how the merge operation works.

The edges are also stored in a tGAP table, but these act in a different way. The initial edges have a 'left face id' and a right face id', but when the generalization happens they get a 'low left face id', 'high left face id', 'low right face id' and a 'high right face id'. This is because edges don't necessarily have to disappear if their neighbouring faces merge with other faces (note, not with each other). The lifespans of edges can differ very much from the lifespan of their adjacent faces. If their adjacent faces merge with each other, the edge lifespan concludes at that step.

Edges do not contain feature class information, however this changes when the split operation is enabled. This operation applies to roads and water streams. By using only merge operations without weights, all faces are merged according to their area. Thus in this case, the software does not care about roads or water. This is what is visible in the current demo when zooming out to certain degrees (see figure 11).

Figure 11. Equally treated merging, note the disconnected parts of road.

The software provides the possibility to take into account the roads while merging faces, as well as defining at what scale the faces may start merging 'over' the roads. Let's demonstrate this with an example. When a road sits in front of the queue, it is ready to merge. What would normally happen is that it would merge with its most compatible neighbour. The newly merged face would later merge with other faces or - more importantly - the other neighbouring face of the former road. In the latter case, there is nothing left of the road that was there before. Other pieces of that road network might still exist, this is what is visible in some parts of figure 11 (equally treated merging). The split operation treats this situation in a different way. When the road is in the front of the queue, it recognizes by the feature class that this face is a road. It then 'splits' the road (figure 12).

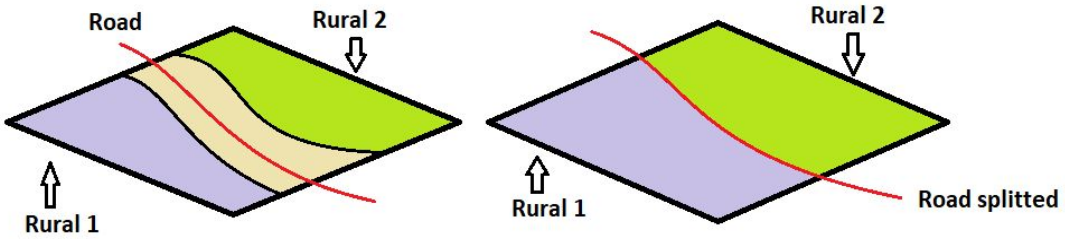


Figure 12. Road network generalization

The underlying operations for splitting the roads are triangulation and a baseline split. The road is triangulated. The baseline is the line through the midpoints of all diagonal line segments (see figure 13). The polygon split operation is done along this newly created baseline. In the same operation the split face is merged with the two neighbouring faces. The baseline is then added to the edge table. The feature class of the road is stored in the newly created baseline. This means that the information of the road is maintained, but transferred from a face to an edge. This information is used when a face adjacent to this new edge is in the front of the queue. When that face wants to merge 'over' the road edge, the face will merge with another face (the second most compatible one) to keep the edge intact. If a face is surrounded by only road edges, it will merge over the least important road, according to the decisions of the user. In this case the road edge will disappear and the road information is lost. If the face is surrounded by only road edges and the universe face, the face will merge with the universe face. This means the face will disappear. This whole functionality makes sure the road (or water) network stays intact as long as possible. Note that this functionality is part of a research and is not complete. Also note that maintaining the network in this manner is only useful when line segments (1-dimensional objects) are visualized. In the current implementation, the viewer only visualizes triangles.

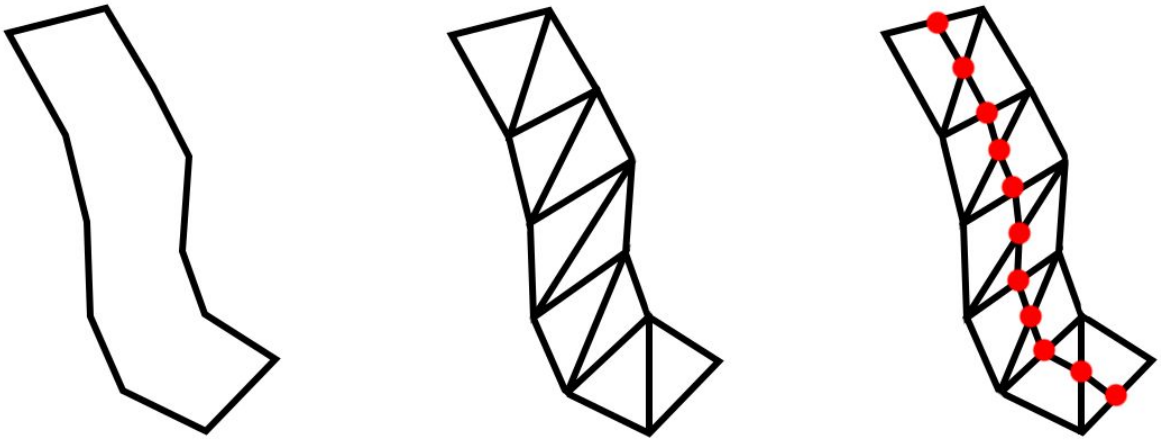


Figure 13. Baseline extraction.

Another useful implementation is the simplification function. This implements the aforementioned binary line generalization tree. The benefits from this are a smoother zoom and less data storage for points. The implementation simplifies edges using the Visvalingam-Whyatt line simplification algorithm. This algorithm is suited for natural lines and deletes the points which are the least characteristic. This way, detailed information will simplify at lower scales, which is desirable. Without this function turned on, very detailed edges will still be visible at lower scales while they still exist. The use of this function does result in a different type of 'space scale cube', this will be elaborated on in the next section.

In short, this is how the generalization works. It is notable that as the adaptability according to the needs of the user improves, the complexity of the software increases. And even with the additional functionalities which were discussed above, the current implementation is still inflexible and homogeneous in character. It applies a set of rules on which the user can apply exceptions and somewhat steer the process. To do so, a deep understanding of theory and software is necessary. In an ideal fashion, the user would have full control over the generalization process by only having to 'turn the knobs'.

2.4.4 SSC creation and visualization

After the tGAP structure is built, the final steps of creating a vario-scale map is making a space-scale cube and rendering it in the browser. This happens by converting the tGAP table into a geometry definition file (OBJ file). This is a human readable file format which is demonstrated in figure 14 (obj file example) by example of a simple cube. The first part of the structure consists of a vertex list, which are all the lines that start with a letter 'v'. All the lines that start with a letter 'f' indicate a face, the consecutive numbers are indices of the prior vertex list. The line that starts with a letter 'g' indicates that all following faces belong to that group. Note that the second number of the group is a feature class code. This way it is possible to assign the correct colors to groups of objects. In this example, the faces are made out of 4 vertices. The .OBJ file that will be created from the tGAP structure will mainly contain triangles. Also, it will be much bigger than this one. Although this depends on how big the tGAP structure is, and that in turn depends on multiple different factors like the kind of operation performed or the dataset used. File size will be elaborated on the chapter 5.

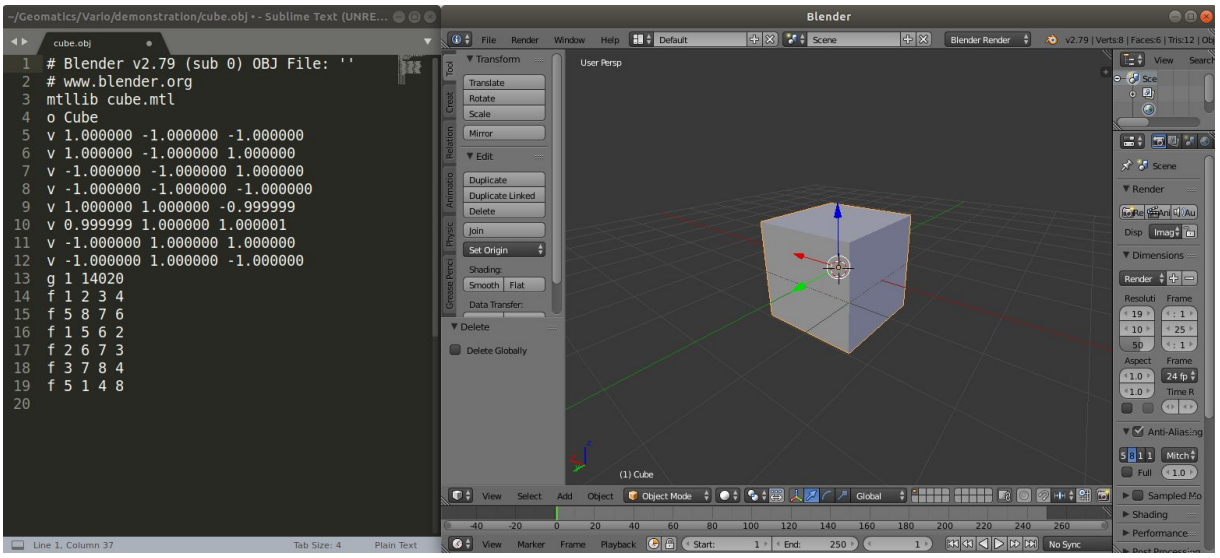


Figure 14. OBJ file format example of a cube

As mentioned before, all the faces are stored with a 'step low' and 'step high' value in the tGAP structure. As the faces need to be stored in a 3-dimensional file format, these values are used as the height component. Also, the faces will be triangulated before they are being stored. The space scale cube is demonstrated in figure 15. It is visible that at the bottom all the most detailed faces are are present (the original dataset). If you traverse the cube upwards while looking down, newly merged faces will cover up the underlying faces. This action is analogous to zooming out. Imagine sliding an orthogonal camera back and forth perpendicular to the viewport. Objects which are closer to the front of the camera will cover up the objects behind them. If you zoom out to a maximum there will be only one face left which covers up all the other faces. The lower merging operation will be executed on the smallest faces, while later on the faces become bigger and bigger. This explains the gradual increase in density towards the top of the cube as is visible in figure 15.

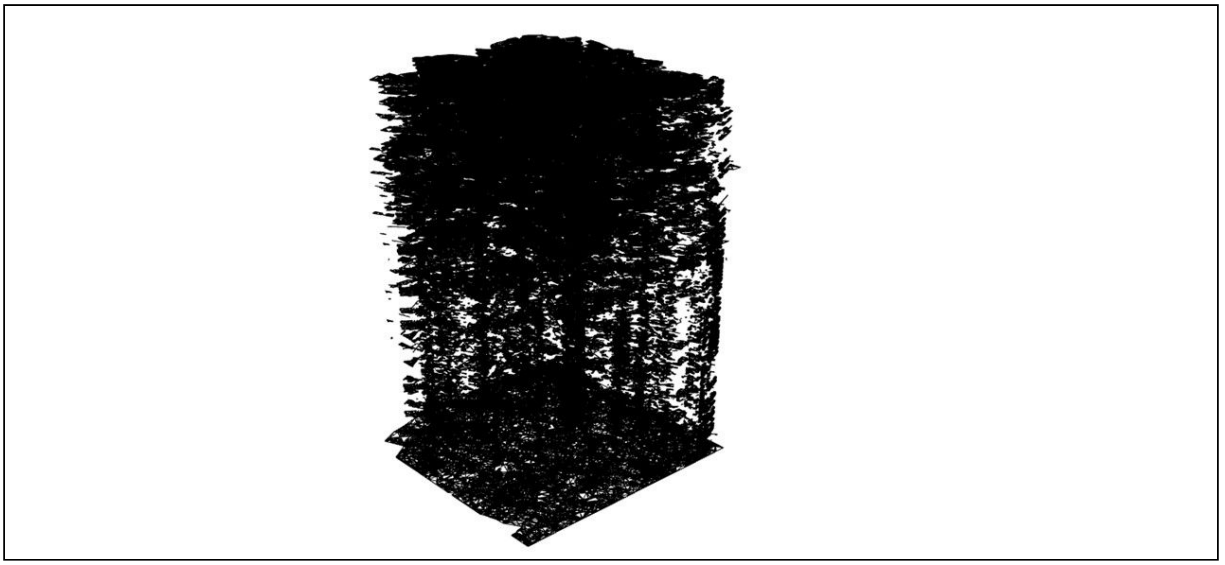


Figure 15. Space Scale Cube in .OBJ file format

By default, the faces stored in the .OBJ file are horizontal, like a stack of paper. When the line simplification generalization function is turned on, this is not the case. Figure 16 demonstrates this by showing a simplified edge (from bottom to top) as it would look in 3D. Instead of storing only horizontal faces, also connections are made between consecutive steps. This makes the simplification of the line look smoother.

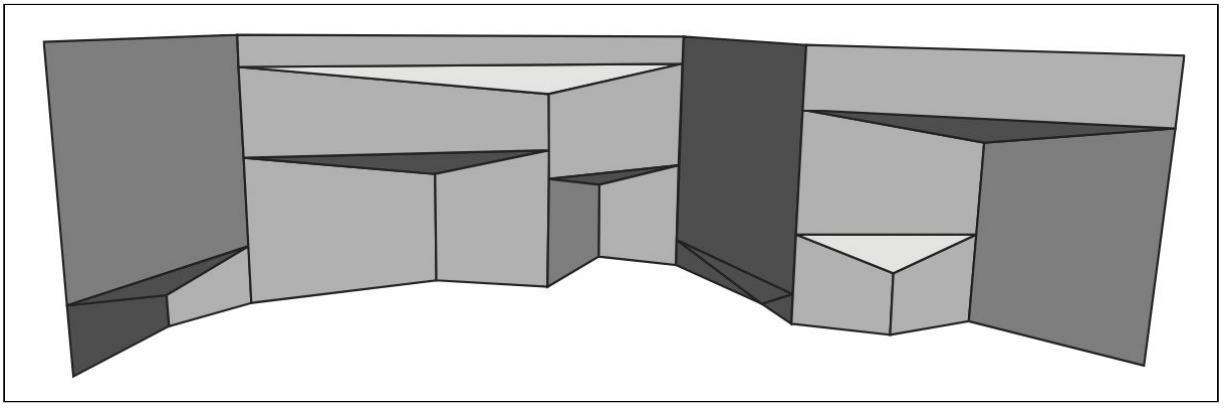


Figure 16. Line simplification in 3D (van Oosterom & Meijers, 2013)

Another possibility is to add smoothness to the .OBJ file. This would be an implementation of the *smooth* SSC. This is done by varying the heights of vertices of faces. This can be seen in the two examples of figure 17. On the right, the regular 'horizontal' space scale cube is shown. Here, the face merges with another face is an abrupt fashion. On the left, the face gradually takes over the other face. Just in the same way described above, try to imagine a section cut sliding vertically across these faces, while looking down. It is logical that the left example looks better, because is flows towards the surface it is going to 'eat'.

The simplification function and the smooth space scale cube were never tested to work together. Also, the implementation of the smooth space scale cube is still work in progress. The usage of both methods simultaneously would be very likely to cause conflicts.

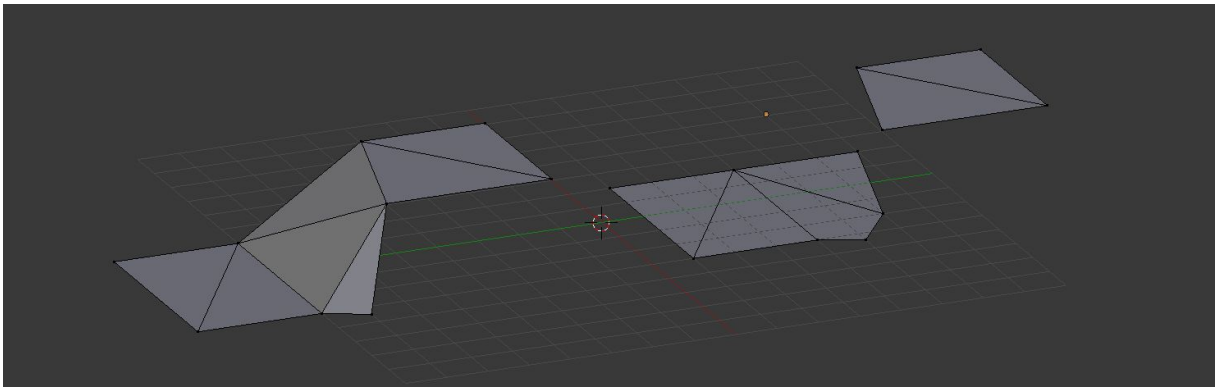


Figure 17. On the left: smooth space scale cube. On the right: normal

While the space scale cube in .OBJ format is a nice way to visualize it in 3D-modelling software, it is not the final format to render it in a web viewer. This is the binary file format. The reason to transform the .OBJ file into a binary file format (.BIN) is to reduce size and being able to process it in an easier way. The first part of figure 18 demonstrates an overview of the transformations.

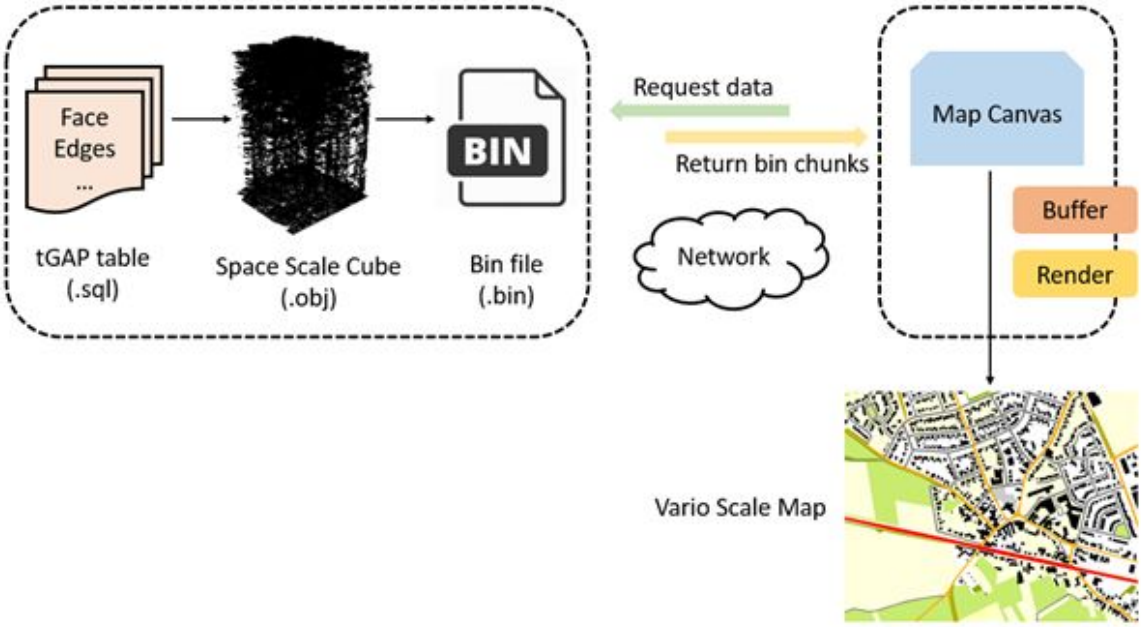


Figure 18. SSC creation and vario-scale map visualization

The binary file records only the information necessary to be rendered. The bin file is created by serializing the .OBJ file. The software does this by writing the triangles to file as a consecutive stream of vertex coordinates and their corresponding RGB values as floats of 64 bits:

X, Y, Z, R, G, B, …

Note that this format is actually redundant, as it stores every point and color explicitly. Just as the .OBJ file it could store a list of points and a list of vertices and then store the triangles as index lists. This method is currently not implemented, which results in a big bin file. From a

certain threshold the bin file does even get bigger than the .OBJ file. After the transformation, this file is stored on the server.

At the front-end, once the client sends a request and receives the bin chunk of the space-scale polyhedron, it will render the polyhedron on the map canvas according to the current view port. Screen grids will directly look downwards to the polyhedron. The first hit triangle will indicate the corresponding feature which will be rendered. This is done akin to the method by Driel (2015), explained in section 2.3. Then WebGL engine is utilized in the rendering process. WebGL is a technology which enables users to create 3D graphics within a web browser. Through the WebGL API the CPU of a computer is bound to its graphics card. No specific browser plugin is required in the WebGL rendering process (Danchilla, 2012). To make WebGL work, two types of shader programs need to be created: the vertex shader, which is used to compute vertex positions, and the fragment shader, which is used to compute the color for each output pixel. The work principle of WebGL is shown in figure 19.

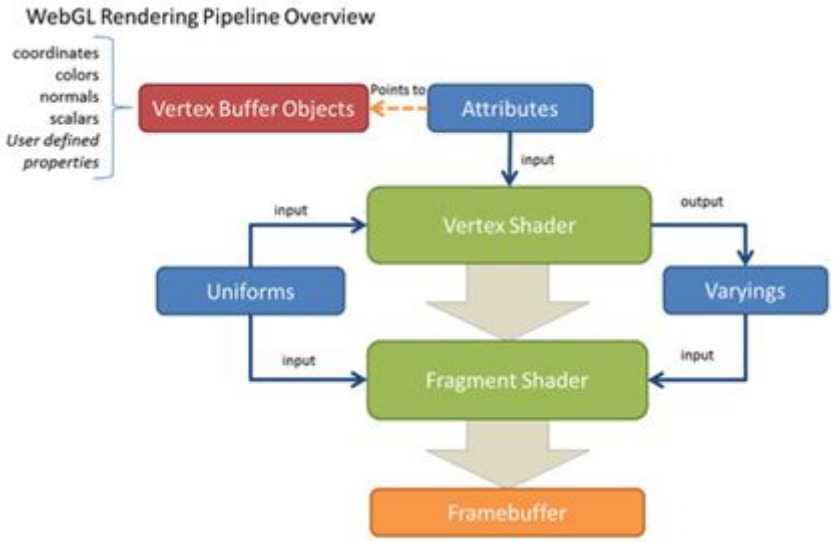


Figure 19. WebGL work pipeline (Brandon Jones, Diego Cantor, 2012)

To render the triangles that we have in the bin file, several steps are required. First, the binary data are read into the buffer zone. Then both the coordinate and color information of each vertex are retrieved into the vertex shader and the fragment shader. At the same time the transformation matrix is set so that vertices can be transformed from the world coordinate reference system(CRS) to the local screen CRS. It is determined where the slice plane is (the screen), and where the pixels are. Per pixel, all triangles below it are selected and sorted on distance from the slice plane. The closest one is then rendered in that pixel. Finally, the WebGL will draw vertices on the screen in a triangle format.

Figure 20 demonstrates the work principle of the front-end. The map module provides functions to initialize and modify the map. The work module loads bin files to the front-end. The draw module reads input binary data and renders it on the map canvas. The transform module together with matrix and rect module is responsible for setting transformation CRS. Last but not least, the mouse module enables user interactions with the map canvas such as panning and zooming.

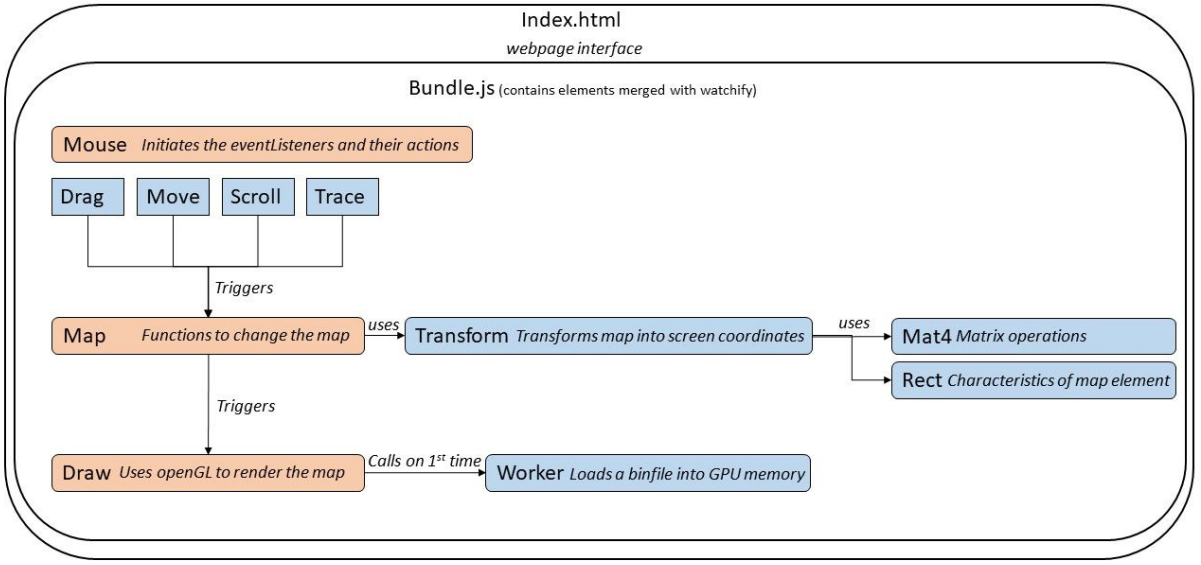


Figure 20: Front-end work principle

The above illustrates how the existing implementation of vario-scale, from input data to web map, works. After this technical exploration, current users of Kadaster topographic data will be surveyed on what they would like to see improved in the current vario-scale implementation. Based on this, and the technical knowledge of the current implementation, a decision will be made on which aspects will be improved.

3. Assessing vario-scale user needs

The objective of this paper is to improve the current vario-scale implementation based on needs of the users of the topographic data of Kadaster. We have seen what vario-scale is and how it is currently implemented. The goal of this chapter is to research the preferred use cases of vario-scale of the users of Kadaster data and how they think the current implementation could be improved to make it fit these use cases better. First we will explore some use cases for vario-scale. These use cases will be used in the survey conducted for assessing user needs. Thereafter the results of the survey will be presented.

3.1 Vario-scale use cases

The implementation of vario-scale is still in an early stage. Several potential applications of it exist. Some are rather 'traditional' as they deal with the changes induced by the arising of on-screen maps while others look at more innovative, new and still mainly unknown applications. A total of five use-cases will be presented here, among which the first three try to optimize current web-cartography and the two last give an idea of possibilities that might still arise. Use cases are derived from van Oosterom & Meijers, 2011.

Use case #1: seamless zooming. Having web viewers instead of paper maps leads to new possibilities. For example, one can zoom conveniently using the mouse scroll wheel. However, most implementations only zoom in and out of the same picture without changing

the content. When the content gets adapted, it either disappears (e.g. implementation of TOPNL maps in PDOK) or switches to another map (e.g. geoportal of IGN). Both induce an abruption which is undesirable as the user might lose track of the observed place. This is an aspect that might change with vario-scale as the content of the map would directly adapt to the specific zoom-level. By applying vario-scales, negligible abruption between maps take place. The eye does not lose track of what it is looking at. In fact, objects do not suddenly disappear, change styles or are abruptly aggregated anymore. Vario-scale acts as a transition animation between maps scales by showing the changes that are performed (figure 21).

Figure 21. Seamless zoom-in/out. (images: Meijers, n.d.)

Use case #2: automatic generalization at any scale. Maps are traditionally produced at a fixed set of scales which is often nationally specific and inherited from the past. When automatic generalization is applied, it is custom-made for these scales and new ones require new algorithms. By applying vario-scale one could be used for generating maps of any scale. The number of scales becomes potentially infinite (figure 22).

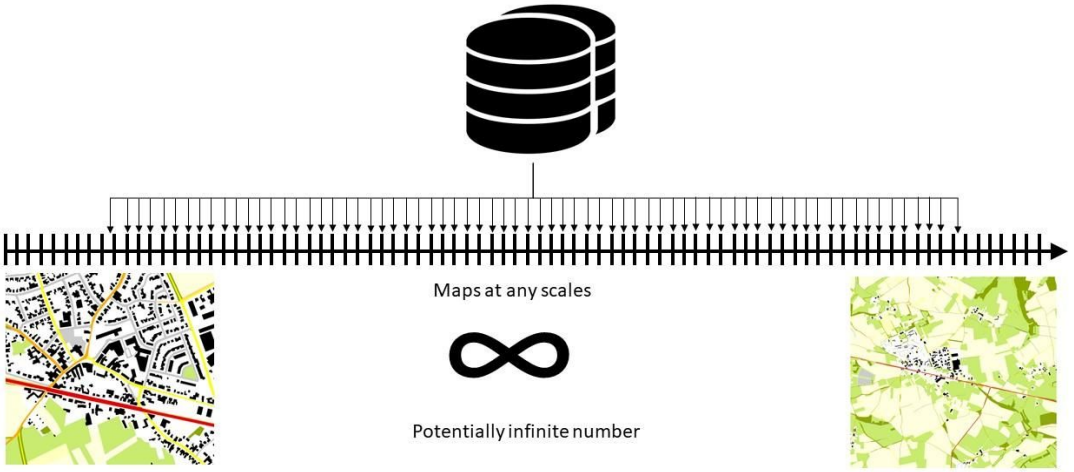


Figure 22. Automatic generalization at any scale. (images: Meijers, n.d.)

Use case #3: less network usage. Within the element rendering process, the data is usually transferred from server to client side after the rendering. This involves transferring raster data

which is rather heavy. By performing the rendering (including transition animations) on the client side, the data is transferred in a rawer stage which induces less network usage (figure 23).

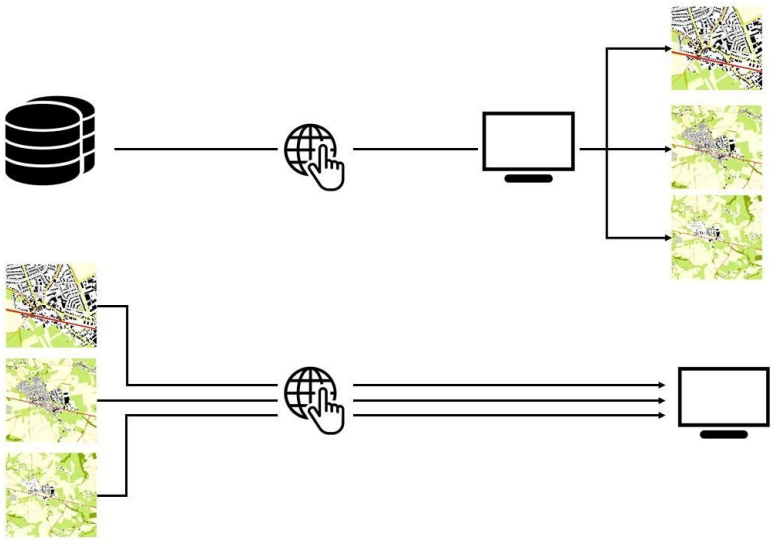


Figure 23. Reduced network usage. (images: Meijers, n.d.)

Use case #4: fisheye. Vario-scale maps are created by taking slices from the space-scale cube. These slices don't necessarily need to be horizontal. By letting the slice plane be a 2-dimensional parabola (e.g. $z = x^2 + y^2$) one can obtain a fisheye view. More details can thus be displayed in the center and less on the sides of the view (or vice versa if wanted), creating the effect of a magnifier (figure 24). This could have applications in web apps.

Figure 24. Fisheye - several zoom levels in one view. Source of the pictures: Harrie, L. (2002) & Hampe, M. (2004) as in van Oosterom, P., & Meijers, M. (2014).

Use case #5: tilted maps. First applications of tilted maps have been introduced, such as the beta version of the vector tiles of the Dutch Kadaster. An issue is that the detail level is the same in the entire view. Using tilted slice planes through the SSC, the detail level could be adapted to the distance from the viewer, avoiding too heavy maps (figure 25).

The above use cases were identified in order to assess the users' potential applications of vario-scale and what they would like to see changed in the implementation. In the next section the survey will be discussed.

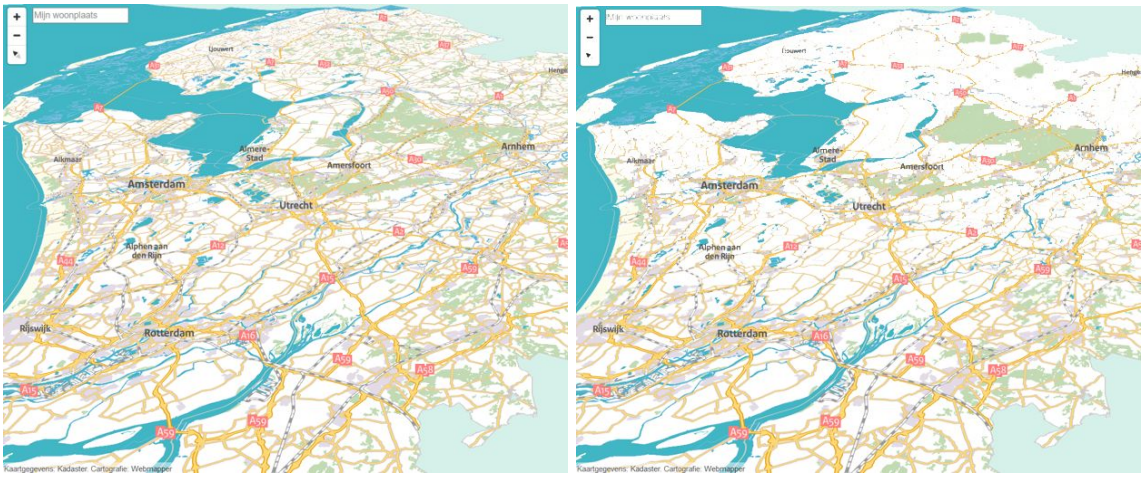


Figure 25. Perspective view - enhanced tilted maps with the existing version (left) and an artist impression of the improved one (right). (Source: Kadaster.nl)

	Mean score out of 10	**Disagreement (standard deviation)**
seamless zoom	8.3	0.7
automatic generalization at any scale	7.9	0.6
reduced network usage	7.3	0.7
fisheye	5.6	2.1
perspective view	7.1	2.4

Table 1. Survey on the use-cases

A small survey with a total of 7 participants was conducted among the BRT-user group (table 1). This survey shows that all use-cases except the fisheye view are perceived as having the highest potentials. Among these 4, the highest agreement is observed for automatic generalization at any scale, seamless zoom and reduced network usage. For the perspective view, the agreement between users is much smaller.

3.2 BRT-user survey design

On 22/05, we had the opportunity to hold a presentation at the BRT-user meeting of the Kadaster in Apeldoorn. This opportunity allowed us to give a presentation providing insight into what vario-scale is and what the potentials are. More importantly, this opportunity also allowed us to conduct a hard-copy survey to get the opinion of the group members and learn more about what they use to orientate between different scales of TOPNL maps.

Figure 26. Pictures of the presentation held at the BRT-user meeting in Apeldoorn (source: Edward Verbree) and of the survey conducted afterwards.

The short survey contained questions about their preferred use cases, what the user misses in the current implementation and a small orientation exercise. In this last exercise, the users are asked to name objects they used to orient themselves during zoom. With these questions, the user needs for vario-scale improvement can be assessed. The whole survey can be found in appendix I.

The questionnaire showed that members of the BRT user-group see continuous, animated and smooth zoom-in as a big potential of vario-scale within the traditional world of cartography. When asked about suggestions for the modifications of the existing demo, two thirds of the remarks were about the interrupted roads and buildings (which are not aggregated). Some of the remarks could not be classified, others were out of the scope of the project (more about the definition of the project's scope can be found in section 3.3).

These statements were verified with a paper map assignment: a building in the outer part (A Ge Veld 57) of a town of Mechelen (1840 inhabitants) in Limburg was shown to the user in Top10NL (figure 27). The user then had to find back this one in TOP25Raster (which still contains individual buildings) and in TOP50Raster where buildings are aggregated into built-up areas (see figures below). At the same time as identifying the place, the users were asked about which elements they think they used to identify the location on the other maps (by circling around them). The results of this study with 13 participants were then classified (figure 13) to draw conclusions.

Figure 27. Maps of TOP10NL, TOP25Raster and TOP50Raster used for the map orientation survey. The participants had to place the cross on the left on the two other maps while indicating the elements they used.

3.3 Survey results

The survey was conducted with 13 participants in the BRT-user meeting. Concerning the use cases, most users indicated that their preferred application of vario-scale would be either continuous zoom or automatic generalization. Both these use cases imply a satisfactory cartographic result. Summarized results can be found in figure 28.

The following observations can be made with respect to the orientation exercise. Road network elements (including crossings) were used both in TOP25Raster and TOP50Raster: by respectively 7 and 9 out of the 13 subjects. The polygonal elements (in this case: buildings) are often used in the TOP25Raster map: they are used by 7 out of the 13 participants. The disappearance of these elements in the TOP50NL leads to an increase of the usage of built-up areas and their borders: their usage grows from 5 to 8 times.

Although the name of the municipality was clearly printed on the map, it was only used by one subject. This could be related to the fact that especially the TOP25Raster is still rather zoomed-in and only contains this one name (the TOP50NL map contains more names). Another reason might be that the position of the label changes with the different scales. Also, no user indicated having used symbols such as the local church. Some of the indications could not be classified. For one participant it seems that the assignment was not understood (he simply circled the locations, not the elements used to find that one). All the other cases occurred in TOP50NL where some circles were not clear enough to be assigned to one of the categories.

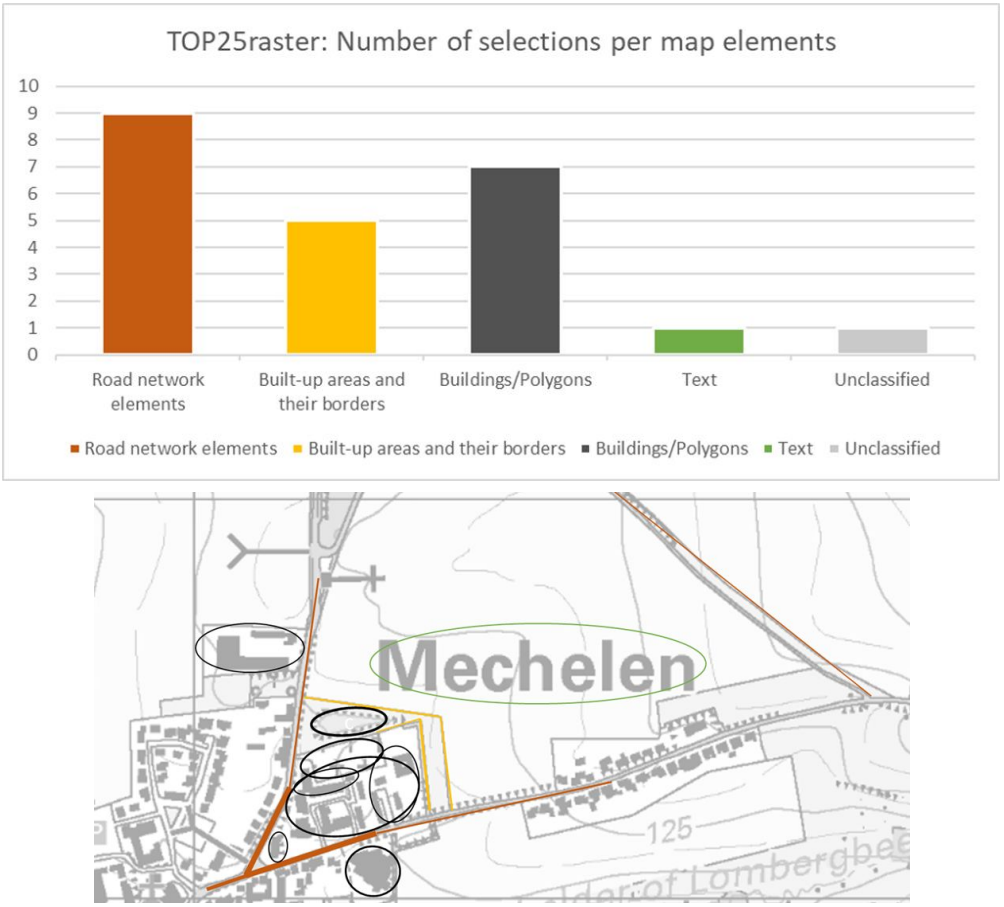


Figure 28a: results of the TOP25Raster orientation exercise.

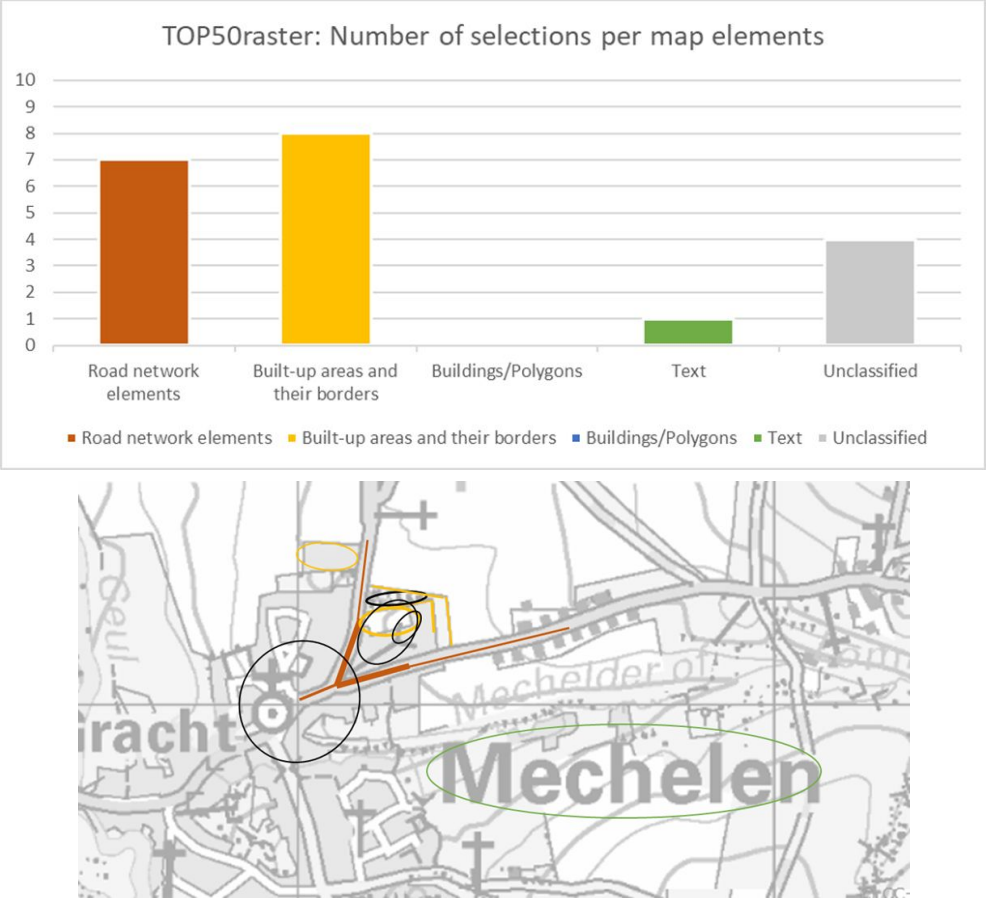


Figure 28b: results of the TOP50Raster orientation exercise.

This exercise shows that the subjects would use vario-scale mainly for continuous zoom and automatic generalization. In continuous zoom, orientation is important and the users mainly use road elements and built-up areas to orient oneself for the lower scales. If available and thus at higher scales, the shape of buildings is strongly used too. The users were also explicitly asked what they would like to see changed in the current implementation. Cartographic needs that were mentioned are:

- The roads in the map shall be visible from the bottom till the highest level of detail.
- The roads shall stay continuous and connected, not be broken up at each map level.
- The built-up area in the map shall be visible from the bottom till the highest level of detail.
- Houses at the lower map level shall be aggregated into built-up areas at higher level.
- The labels and symbols of roads and built-up areas shall be always visible.

Other technical needs also mentioned by the users were:

- The map visualization shall be made customizable.
- The web visualization framework shall establish a double-screen, which can compare between original map and vario-scale map.
- The web visualization framework shall maintain high efficiency in data transforming.
- The systems shall be easy to update and maintain.
- The vario-scale method shall be applicable to 2D/3D vector tiles.

Another potential improvement was mentioned by Martijn Meijers, the main developer of the current implementation. The demo only works on computers, not on mobile devices. Therefore an improvement would be to adapt the implementation so that it works on all devices.

Not all the functional needs listed above are viable to research in this project. Considering the existing technology and skills of the team, the budget and the schedule of the project, and also the importance of each item, we classified the functional needs into three categories (figure 29): viable, of interest (thus relevant but too broad for this project) and out of scope. Some should be considered, others could be implemented but, while others are out of scope of the objectives of this project and will not be considered.

Viable options	Options of interest	Out of scope
○ *Roads remain visible and the network remains topologically correct.* ○ *The built-up area in the map shall be visible through all scales and house will aggregate.* ○ *Implement labels and symbols.* ○ *Adapt implementation for mobile services.*	○ *The web visualization framework shall maintain high efficiency in data transforming.* ○ *The system shall be easy to update and maintain.*	○ *The map visualization shall be made customizable.* ○ *The web visualization framework shall contain a double-screen, to compare between original map and vario-scale map.* ○ *The vario-scale method shall be applicable to 2D/3D vector tiles.*

Figure 29. Classification of the needs expressed by users at the BRT-user meeting into viable, of interest (too broad but relevant) and out of scope (not relevant in this project).

Researching all four viable options is also out of scope for this project. Therefore, a choice will be made on which research options will be pursued. In the next chapter, a SWOT analysis will be performed on all four options. From this, a research plan will be developed.

4. Selecting improvements for implementation

We want to research what can be improved of the current implementation of vario-scale. To answer the question, it is assessed what users of Kadaster data want to see improved, and what we are able to provide within the scope of this research. After assessing user needs, four viable research option were identified: networks, aggregation, symbols/labels and mobile services. In this chapter, these research options will be submitted to a SWOT analysis. Based

on these analyses, the final research goals will be set. These will then be implemented in the current demo.

Networks (roads & hydrography)

- ***Strengths:*** Networks are one of the most important features used for orientation in maps during zooming. Improving the topological correctness of networks during the zoom by using certain rules should quickly improve user experience of vario-scale use.
- ***Weaknesses:*** Manually maintaining a certain network through the generalization slightly undermines the concept of vario-scale.
- ***Opportunities****:* Research has already been conducted about how to generalize roads and hydrography networks in a vario-scale setting (Groeneveld, 2018; Suba et al., 2016). Also, for roads this has already been implemented in the current version of the vario-scale tGAP creation software.
- ***Threats****:* The challenge in implementing lies in the visualization of line elements. This has to be done on the client-side, writing code that instructs the GPU. No group members have experience with this, so this will take time. Another threat is the method of visualizing both road polygons and lines at the same time. Roads of the same hierarchy level might at a certain zoom level be represented by both polygons and lines. Rules need to be set to either avoid these situations, or deal with it in a satisfactory manner.

Aggregation into built-up area

- ***Strengths:*** On smaller-scale zoom levels of the current implementation, housing disappears and only a white area remains. Aggregating housing into built-up area will be advantageous for orientation on smaller scale zoom levels.
- ***Weaknesses:*** It is unclear whether houses need to merge with each other, or that houses should merge into the now white area that represents the space around housing.
- ***Opportunities****:* The white area that surrounds buildings in the current implementation is already a representation of built-up area. So there already is information about how far this area stretches.
- ***Threats****:* Aggregation of polygons into a new class will require extensive adaption of the tGAP creation process. New rules will have to be designed about how and when houses merge and into what other polygons, and at what zoom level the new "built-up area" class should start to appear. In the current implementation, small buildings are the first polygons to disappear (merge into neighboring polygons).

Placement of symbols and labels

- ***Strengths:*** Symbols and especially labels are very important for orientation purposes if someone wants to know where they look the first time. So for inexperienced users labels are essential.
- ***Weaknesses:*** Symbology and labels were only sparsely used during orientation tests. Therefore they might not have the highest priority of implementation.
- ***Opportunities****:* Now completely lacking in the implementation. Therefore implementing symbols and labels would be a big improvement in itself.
- ***Threats****:* Just like the representation of networks, text and symbols (representation of point objects) will have to be visualized on the client side. Moreover, when using labels, the decision to show what at what moment because increasingly impossible. At increasingly smaller scale, less labels and symbols have to be represented to prevent

text filling the entire display. This would require something like a semantic mirror of the tGAP structure, which is not developed yet.

Adapting for mobile services

- ***Strengths:*** Having vario-scale available would increase its availability. The current implementation of vario-scale focuses on web services. Since mobile devices are very much used to access web maps, having a working mobile vario-scale implementation would improve user experience.
- ***Weaknesses:*** Adding mobile services will not change anything about the current cartographical implementation.
- ***Opportunities****:* The group has (basic) knowledge about web mapping services and mobile devices, which will be advantageous for this implementation.
- ***Threats****:* No one in the group has direct hands on experience with implementing web services for mobile devices, so there might be a big learning curve.

Looking at the four research options, some different things can be concluded. The first three are concerned with changing the cartography. These tasks imply big changes in how the vario-scale structure is built or in how it is visualized. The last research option, adapting the current service so it is compatible with mobile phones, seems like a smaller task. Because of this, it is decided that we will research one of the three cartographic options along with implementing mobile services.

Symbols and labels were less used in our orientation exercises. Also, visualizing point objects and text is currently not implemented in vario-scale. So while implementation will be a big addition to the current demo, it might be too much work due to time constraints to implement this from scratch. The same can be said for aggregation. The lack of aggregation now may mean that taking generalized maps at intermediate scales will give strange results, with some buildings still present and other removed but not aggregated. However, implementing our own aggregation algorithm will be too much work. Therefore, in this research we will try to implement the maintaining of road networks. There have already been efforts to implement this. Therefore there is a foundation on which can be built. Also, it was mentioned as one of the biggest factors in map orientation. Therefore this research option was chosen.

The current implementation will be adapted in order to implement both the mobile services and to maintain road networks throughout the scales. In the next chapter the process, methodology and the eventual results will be discussed.

5. Process and results of technical vario-scale improvements

After exploring the current software and assessing the user needs, two research options have been chosen: maintaining a correct road network throughout all scales and adapting the implementation for mobile devices. In this chapter, the process of attempting to achieve this result is documented. Since the exploration of the software (see section 2.4) and adapting it overlapped, the methodology was not immediately clear from the beginning. Therefore this chapter will also outline the methods used to achieve the results. The first part of this chapter outlines the results of our attempts to improve the visualization of the road network. After, the results of the implementation for mobile device is presented. A new demo that implemented these changes has been sent back to the users for a review. This is done to check if they

experience the adaptations as actual improvements. This way we want to answer our research question. In the end of this chapter their feedback on the new demo will presented.

5.1 Better visualization of the road network

In the current implementation of vario-scale, the roads are represented as faces. When roads merge with adjacent faces that are of a different class, pieces of the road network disappear. In the survey conducted with Kadaster users this came forward as one of the biggest cartographic deficiencies. Technically, this problem was labeled as solvable within the temporal scope of this project. This statement resulted from several technical meetings with Martijn Meijers. The goal of this section is to explain how we attempted to solve this problem. As stated earlier, there had been research on the maintenance of the road network in the generalization (see section 2.4). In this case, road information is stored in edges. The original method for maintaining road networks was to visualize these edges, combined with road faces still remaining. After this proved too difficult, the method changed to creating a seperate cube for just the road data. The downside of this method is that abruptions between scales return. Therefore a method was created to create a double tapered structure for the road polygons. The whole process is explained in the following subsections.

5.1.1 Input dataset

The first step of the project plan was to get comfortable with the software. In close cooperation with Martijn Meijers, we tried to create the most basic tGAP structure (only the database). The same dataset as the existing demo on the vario-scale website was used: a seven by seven kilometer patch of the TOP10NL Kadaster data in the south of the Dutch province of Limburg. Before creating tGAP the structure, we ran into some setup issues. As described in the first part of section 2.4 (current implementation of vario-scale) the software environment is extensive in its dependencies. While running the software by its most default form, some errors occurred. These were mainly the consequences of badly defined metadata or the lack thereof.

Within a couple of days, a default tGAP structure was created. The results could be visualized in QGIS, see figure 30. These figures demonstrate different slices from the tGAP. The figure on the left is a higher detail, higher scale map, the figure on the right is a lower detail, lower scale map. The red rectangles are the minimum bounding rectangles of the faces. A slice refers to all the content of one step value. A slice can be selected by the following query:

$$SELECT\ Face\ id,\ Geometry\ WHERE$$
$$Step\ high\ >=\ n\ AND\ Step\ low\ <\ n$$

Here *n* represents the step value. To browse through the slices in a comfortable fashion, a QGIS Python script was created, which is attached in the appendix III.

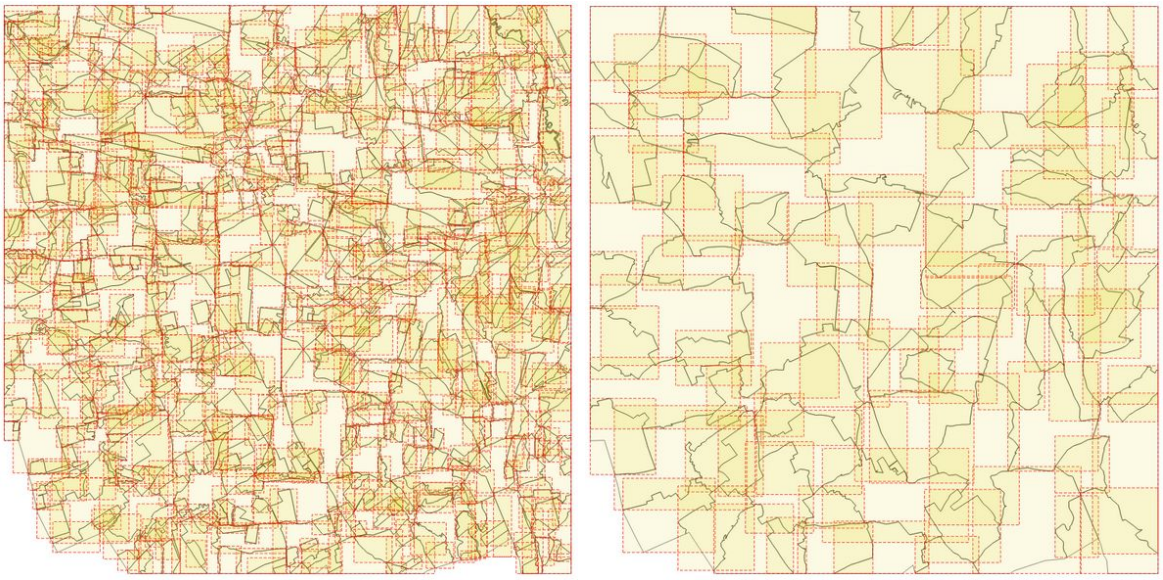


Figure 30. Slices from first default tGAP structure

5.1.2 Results of *do roads* tGAP

As mentioned, the first method was to visualize the lines generated by the split operation. To recall the implementation of the maintenance of the road network - referred to as *do roads* in technical terms - we summarize the current functionality briefly:

1. The user defines which feature classes are roads as metadata.
2. The user defines at what scale range the roads should stay intact.
3. When a road is at the front of the queue, it is split and not merged.
4. The feature class of that road is stored in the created baselines.
5. These feature classes are being taken into account during generalization.

The main problem of this functionality is that while the road network stays intact in the edges, it is not visible. Also, throughout the process the road network will consist of both faces and lines, which need to be visualized in a consistent manner. The solution to this problem would be a step closer to our main goal. Having created a tGAP with default setting, we now aimed at creating one with the *do roads* function turned on. To proceed, the feature classes of the specific dataset had to be mapped. This information was taken from the TOP10NL visualization stylesheets. This sheet is meant to assign color codes to polygons in the map. Based on an XML file which was created by Radan Suba during his research on this implementation, the table in figure 31 was created.

This file assigns the ranges to the vario codes, which in turn we assigned to the visualization codes. To give as an example: the program asks, what is feature class *10311*? This is then looked up in the visualization stylesheets and the vario code is put as *5*, because this is an important road. Then, the XML was used to look up the range(s) for important roads, which is *(0, -1, split)*. This means: if an important road is at the front of the queue, and the scale is between the initial scale and the end scale, split it. Note that other ranges might instruct the face to merge. The range could also be *(0, 20000, merge);(20000, -1, split)* which means: merge this face until a scale of 1:20000, from then on, split it.

```
psql (10.4 (Ubuntu 10.4-2.pgdg18.04+1))
Type "help" for help.

postgres=# select *
postgres-# from result_attr_map;
 feature_class | vario_code |      name      |     rgb     |                                description                                |    ranges
---------------+------------+----------------+-------------+---------------------------------------------------------------------------+--------------
         14160 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14010 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         12500 |       1000 | Water          | 160,220,255 | all water features                                                        | (0,-1,split)
         13000 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14050 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14090 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14180 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14060 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14140 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14170 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14100 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14030 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         10780 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14040 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14080 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         12400 |       1000 | Water          | 160,220,255 | all water features                                                        | (0,-1,split)
         14120 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         14130 |       3000 | Terrain        | 201,235,112 | terrain, green areas                                                      | (0,-1,merge)
         10700 |          1 | Other          | 156,156,156 | all other features                                                        | (0,-1,merge)
         10740 |          1 | Other          | 156,156,156 | all other features                                                        | (0,-1,merge)
         10741 |          1 | Other          | 156,156,156 | all other features                                                        | (0,-1,merge)
         10750 |          1 | Other          | 156,156,156 | all other features                                                        | (0,-1,merge)
         10760 |          1 | Other          | 156,156,156 | all other features                                                        | (0,-1,merge)
         14162 |       3000 | Other          | 156,156,156 | all other features                                                        | (0,-1,merge)
         10311 |          5 | Important road | 230,0,0     | Very important road (e.g. highway)                                        | (0,-1,split)
         10310 |          5 | Important road | 230,0,0     | Very important road (e.g. highway)                                        | (0,-1,split)
         10411 |          4 | Regional road  | 255,170,0   | 2nd most important road, e.g. regional roads of 1st class                 | (0,-1,split)
         10410 |          4 | Regional road  | 255,170,0   | 2nd most important road, e.g. regional roads of 1st class                 | (0,-1,split)
         10600 |          2 | Street         | 255,255,255 | Street                                                                    | (0,-1,split)
         10510 |          3 | Local road     | 255,255,0   | local road                                                                | (0,-1,split)
         10710 |          2 | Street         | 255,255,255 | Street                                                                    | (0,-1,split)
         10730 |          2 | Street         | 255,255,255 | Street                                                                    | (0,-1,split)
         10720 |          2 | Street         | 255,255,255 | Street                                                                    | (0,-1,split)
(33 rows)

postgres=#
```

Figure 31. Attribute mapping table

Figure 32. Left: do roads tGAP slice. Right: background map of area

The result of the *do roads* tGAP, shown in figure 32, is remarkable. By looking at this slice (low detail, low scale) it can be seen that the road network was maintained quite nicely, comparing it to the background map. The roads are visible as edges, not as faces. Another notable observation of this slice is that the faces at the border disappear. This happens when a face is fully surrounded by roads and the universe. When the face is at the front of the queue,

it chooses the path of least resistance to merge with. The *do roads* function will make faces only merge over road edges when anything other is not possible. Because the adjacent edge to the universe is (most likely) not a road, it will merge with the universe. In other words, the universe will eventually take over the whole structure. The problem with this is that the universe is transparent, it does not contain any geometry to add to the structure. As described before, the .OBJ file represents a stack of faces which together visualize the entire map. Creating a space scale cube of this newly created tGAP would leave 'holes' in the structure. Therefore, this is not compatible with the way the viewer works. This problem had to be solved to be able to see the tGAP structure in the viewer. This was not a functionality to turn on or off but had to be altered in the code. It took some days to find out what should be changed. Eventually the software was adjusted so that faces never merge with the universe.

5.1.3 Rendering lines

It was now key to visualize the edges in the viewer. However, there were some problems we had encountered at this stage. The first problem concerned the technical difficulty of rendering lines by means of using WebGL. This rendering engine is a whole topic in itself and its extents reach further than we were able to explore in a short amount of time. Next to the already complex software we were getting acquainted with, it would be risky to start researching this field.

The second problem was the concern about the visual quality the line rendering would achieve. We could test this by extracting the road edges from the *do roads* tGAP structure in QGIS. Figure 33 shows these lines with added thickness. QGIS does not have state of the art rendering possibilities, but the combination of sloppy edges and dangling segments do not give a good impression.

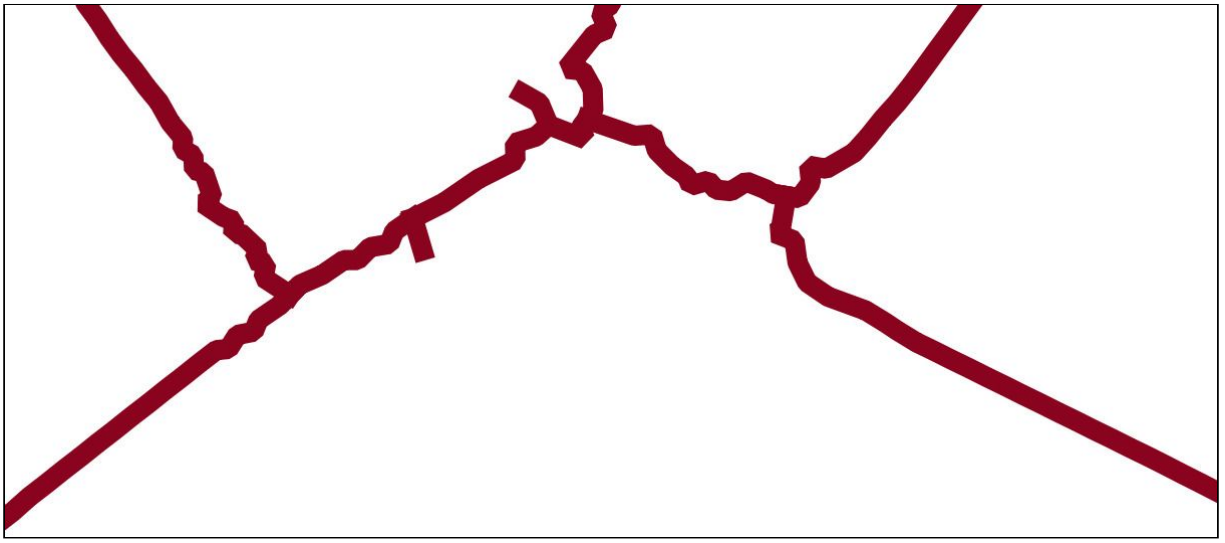


Figure 33. Lines styled in QGIS. Note the zigzags and dangling segments.

The third problem relates to the network of roads. As stated before, faces split individually. Roads consist of multiple faces. This means that at some slices of the tGAP only parts of the road were split. If edges would be visualized by lines and faces by triangles, this might not look continuous. Potential solutions were more difficult than they seemed at first sight. Also they would not fit the temporal scope of this project. Moreover, if they would succeed, the former two problems would still be unsolved. This is why we abandoned the idea of using line

rendering to visualize the road network. We came up with a more lightweight and faster solution.

5.1.4 Creating a road space scale cube

After rejecting the line rendering approach, the other option was to keep working with faces. In search of a method to better control which roads are visible at what scale level, the idea was suggested to create a seperate cube for just the road faces. With this seperate cube, no line rendering is needed, the original shapes of the roads are maintained and we have control over when we want to see which roads. A downside is that the thickness of the roads is not easily controlled, whilst it is by using lines. However, there are possibilities that solve this, which we explored as well, see subsection 5.1.7.

Creating a space scale cube for the road network was not difficult. A series of scripts were created to do this. The procedure is to extract all the roads from the most detailed map, by using the attribute mapping table. This network is then divided in different importance levels by the user. This makes sure that all the roads of a class appear at the same time, meaning the network stays intact (this seems like something not suited for vario-scale. This problem is also addressed in subsection 5.1.7). Finally, the roads are triangulated and serialized to an .OBJ file just like the normal procedure. The height values are based on the importance and scaled according to the original SSC, which we refer to as the background SSC. Figure 34 demonstrates the newly created road space scale cube. Appendix IV and V contain the scripts that were used to create the road space scale cube.

Just as if it were a normal space scale cube, this cube can be transformed in a binary file and seen in the viewer. The script that was used to do this is available in appendix VI. Because the binary file is a consecutive stream of triangles, they could be appended to the .BIN file of the background tGAP. The results were visible in the viewer, but unfortunately there was a trapdoor problem. The background tGAP still contained roads. Even while the newly created road network covered up the existing roads, some of the background roads still appeared when that class already disappeared in the road SSC. For example, all small streets disappear when zooming out, but a couple of these streets still live in the background map (they weren't generalized yet at that stage in the background SSC). The solution to this problem was to exclude the roads from the background tGAP.

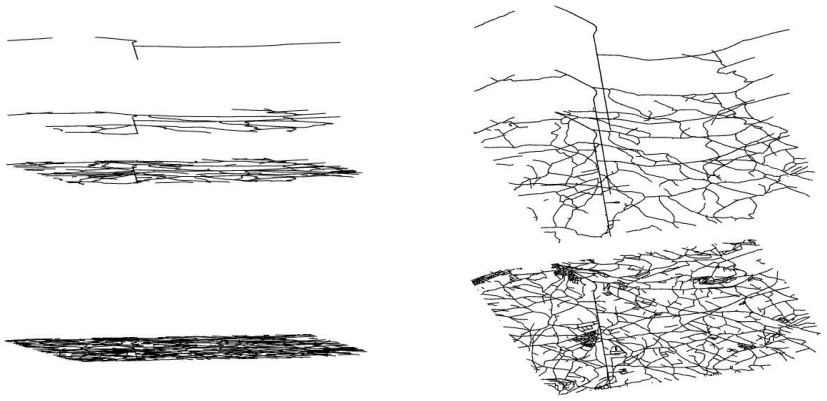


Figure 34. Road space scale cube in .OBJ format.

5.1.5 Excluding roads from tGAP

It is not possible to exclude the roads and then create a tGAP. This leaves holes in the initial map. The roads need to be split before the tGAP is created, but the splitting operation is embedded in the software. Therefore, all roads of the initial map were given a really low importance level before they were added to the queue. This resulted in a queue where all roads are in the front. When the tGAP is created, the first thing that happens is that all the roads are split until there are no more roads left. If at that level the SSC is 'cut' and everything below it discarded, the result would be a background without roads. And when the split function is turned on, the roads would still be maintained in the edges. This would fit perfectly with the road space scale cube.

In order to cut off the bottom of the background tGAP, the step value *n* of the last road had to be extracted first. This was done by using a simple query. A script was made to do the cutting. The procedure for the face table was easy: delete all faces with a 'step high' value below *n*, update the 'step low' to *n* for all faces that have a 'step low' below *n*, finally subtract *n* from every 'step low' and 'step high'. The procedure for the edge table was slightly more difficult, because the edges refer to a 'low right face id' and 'low left face id'. By only performing the same steps as with the face table, these attributes would point to faces that don't exist anymore. However, there is a face hierarchy table created by the software, which connects child faces with parent faces. An iterative program was created which traverses up the face hierarchy until it finds a face which exists in the previously created face table. This script is attached in appendix VII.

This resulted in a road free background, while maintaining the network. Some other variations were made to create a background as well. One used only the merge function, one other merge and split, another merge and simplify, and the last one with merge, split and simplify. We found that using only 'merge' worked best, mainly because of the relatively small file size. However, for every combination the result seemed to be very big (100 - 500 Mb for the .BIN file). Throughout the whole project this was a bottleneck for testing and displaying. In appendix VIII, a full size analysis of different operations is attached.

5.1.6 Building the viewer which supports the visualization of road network

After the data structures are created, they need to be rendered on the computer. The job of the front-end is to present the datasets in the server side to the end users in a cartographically pleasant fashion. As mentioned, two separate binary files need to be created and overlaid in the map canvas. In this subsection it is explained how the viewer is adapted to overlay the binary files. Also, the Map User Interface is redesigned to better meet users' needs. Figure 35 shows the modification made in the front-end modules to support the road network visualization.

The first step to make the client-side work is to create the multi-scale road network space scale cube in a form of OBJ file (figure 36). The road network is divided into four different levels based on their class (main road, regional road, local road, street road). Based on testing, we determined the life span for each road class on the scale dimension. The rendering color for roads are defined in table 2.

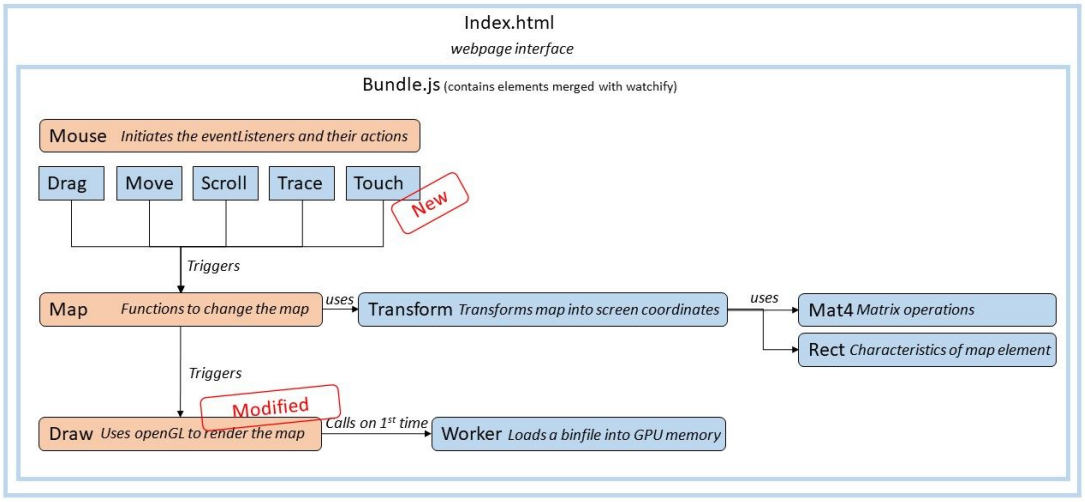


Figure 35. Modification of the front-end to support roads visualization

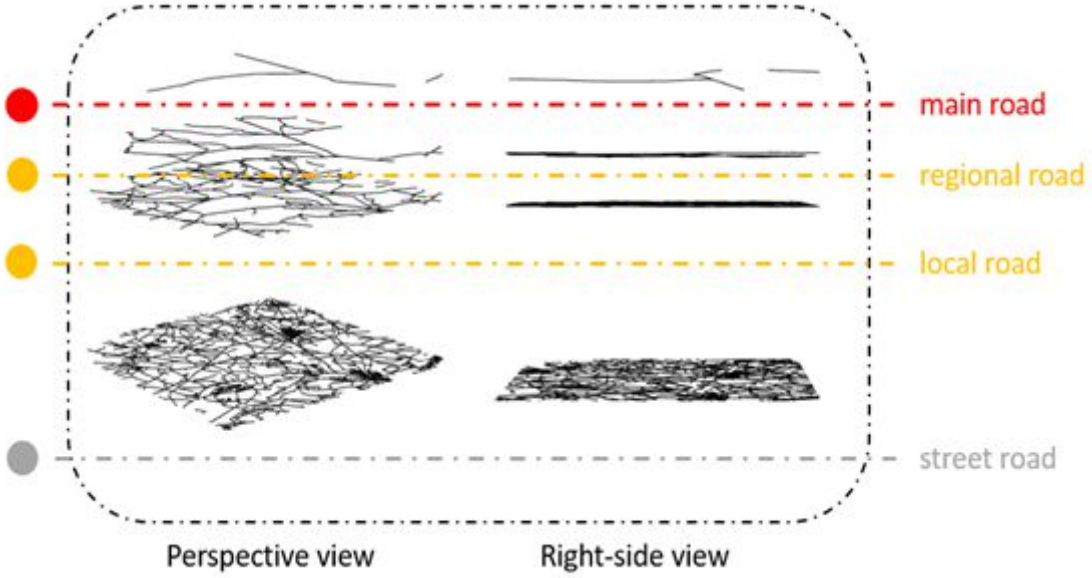


Figure 36. Multi-Scale road network SSC structure.

class	*step span*	*color*
main road	0-9595	red
regional road	0-8500	orange
local road	0-6500	orange
street road	0-5000	grey

Table 2: the step span and rendering color assigned for each road class.

Then the OBJ file is again converted into a binary file which can be easily accessed by the client-side. To increase the efficiency of network data transportation, only the information necessary to be rendered in the client canvas will be recorded in the binary file. That includes the vertex coordinate and the color information. Same as the strategy we used in creating built-up area bin files, records in the road binary file are also structured as: Vx, Vy, Vz, R, G, B…

The rendering principle for road network and built-up area are similar. Both the binary data of roads and built-up area are read into the buffer zone. Both the coordinate and color information of each vertex are retrieved into the vertex shader and the fragment shader. Also, the rendering of roads and built-up area share the same transformation matrix from the world system to the screen system. Finally, the WebGL will draw triangles for both the roads and the built-up area. The overall workflow is shown in figure 37. The script that was used to draw multiple binary files is attached in appendix IX.

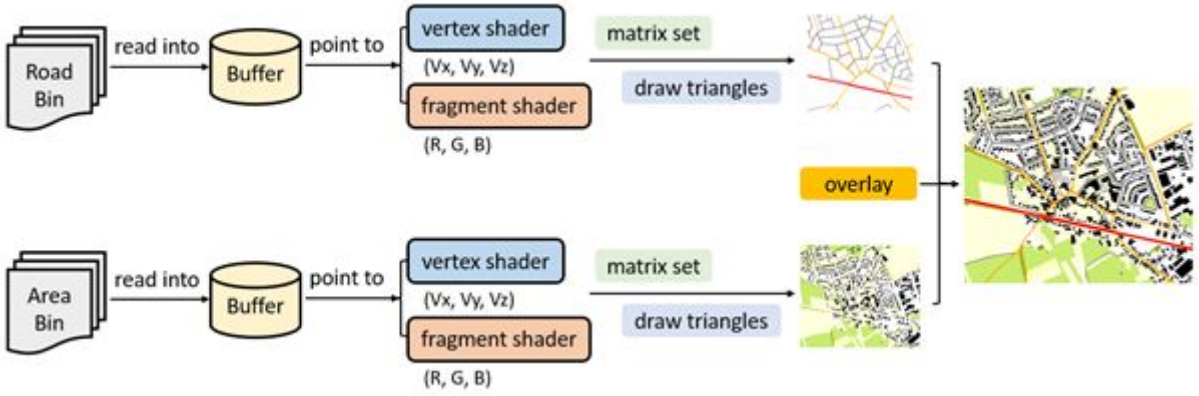


Figure 37. Workflow of rendering the roads and the built-up area.

However, some slight differences exist when rendering the roads and the area. First, the roads need to be drawn later than the area so that they won't be overdrawn by the area polygons. Second, we need to set two different sets of viewport. A viewport determines the visible part within a space scale polyhedron, and, is defined by the near and far planes.

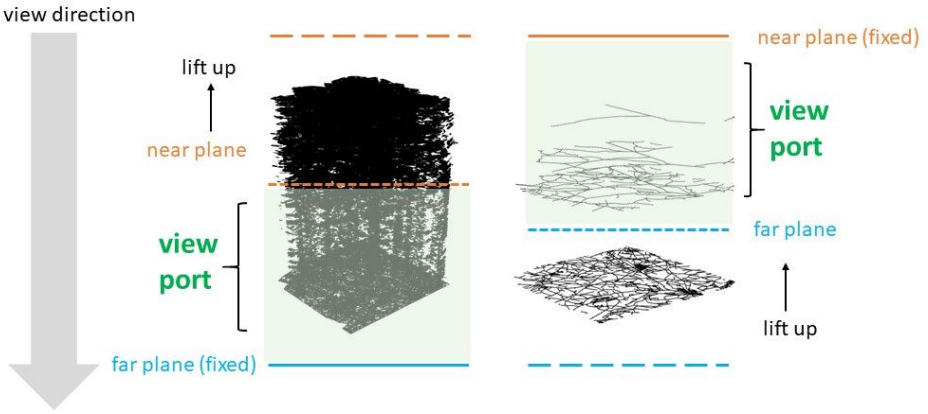


Figure 38. Viewport for built-up area and road network

For the background, the far plane is fixed and the near plane is lifted gradually with the mouse zooming. As the screen grid will directly look downwards, only the first triangle hit will be rendered. Thus only polygons which are existing in the current step level will be drawn. But for roads, the viewport acts in reverse. The near plane is fixed while the far plane is lifted, enabling that only corresponding road classes will be visible with one specific step range. Figure 38 shows how the view port is determined for the area polygons and the roads.

Aside from changing the viewer, also the map user interface was adapted. Figure 39 shows the UI of the original demo. As shown below, although this UI version achieves basic visualization of the vario-scale map, some drawbacks still exist that might affect its interaction with the end users. For example, the parameter sliders are placed at the bottom of the webpage, which are not always visible for users. Besides, there is no clear description of the demo. To make it more convenient for end users to use, the map UI is redesigned (see figure 40).

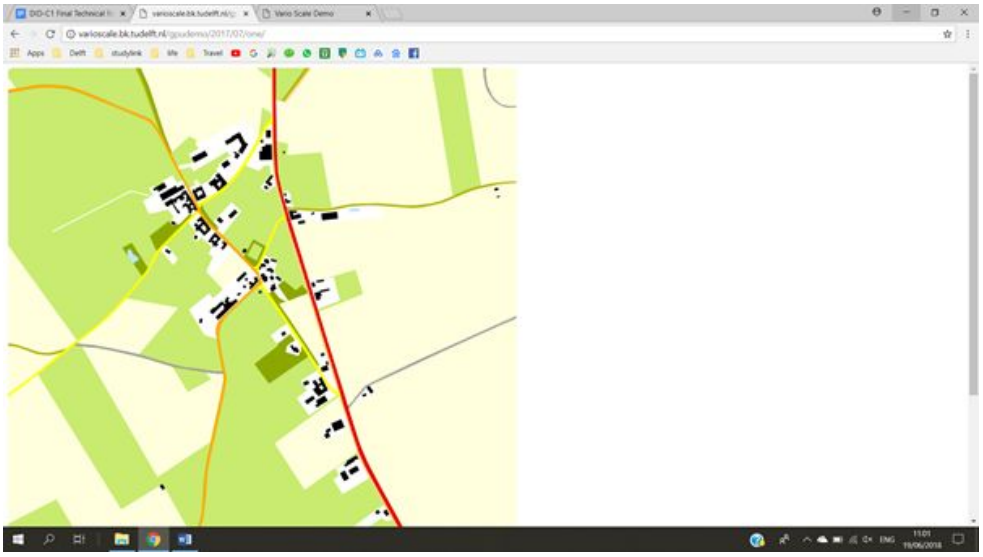


Figure 39. User interface of the original map demo

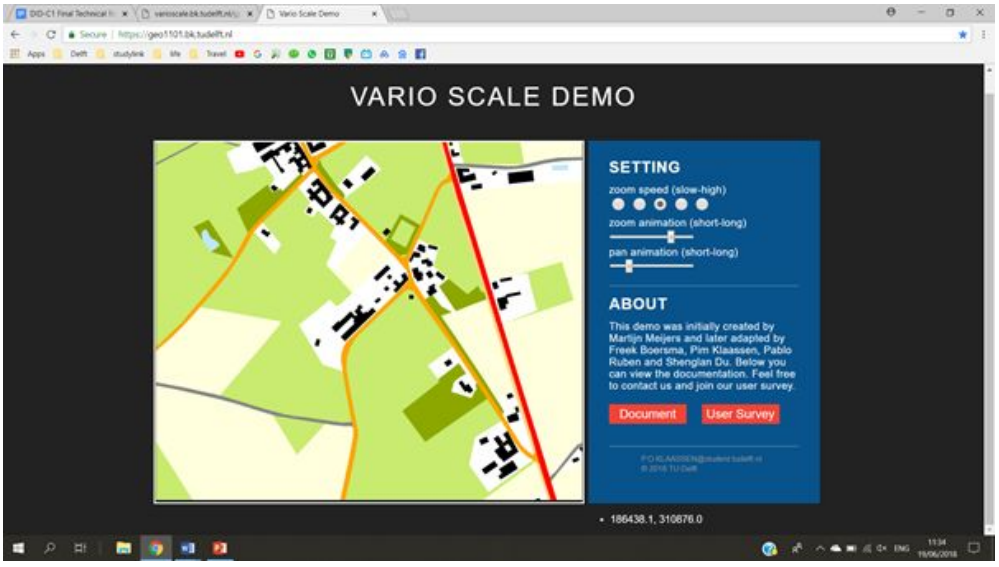


Figure 40. New user interface design.

Furthermore, to make this map demo adaptable to mobile devices, Responsive Web Design (RWD) is taken into account in the UI design. Not only the map canvas but also the setting module and the description module are resized when loaded on a mobile device. Figure 41 shows the map UI design on mobile devices.

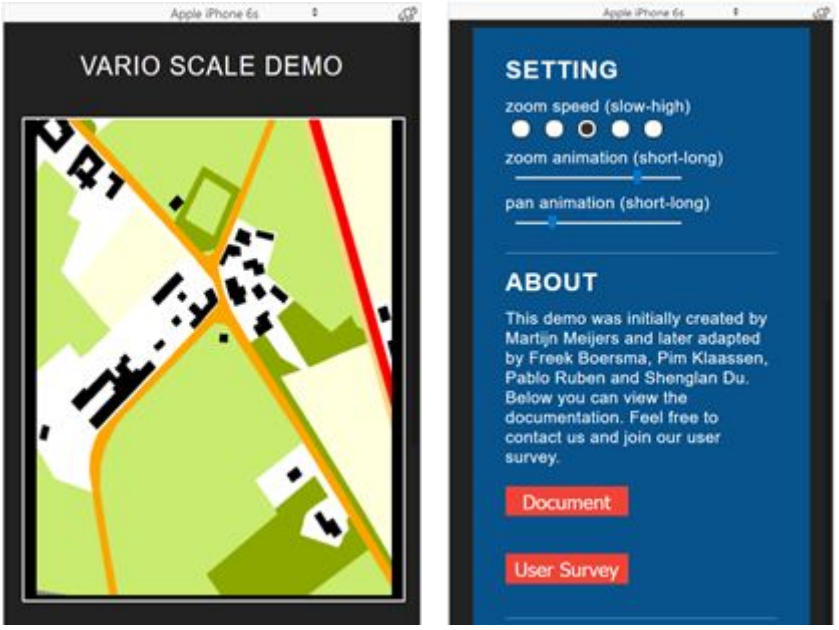


Figure 41. Mobile user interface design. Right image is seen when scrolled down from the left image.

5.1.7 Double tapered approach for vario-scale road thickness

At this stage, the main goal has been realized: throughout all scales, a topologically correct is visualized. At the highest scales, all roads are visible, and when zooming out, whole hierarchy levels disappear at the same time. This however, has its downsides. As already mentioned, there are two main problems with this approach. The main problem is that one of the main use cases of vario-scale is the smooth, continuous zoom. Having big parts of the roads disappear at once negates this purpose of vario-scale. Another is that if roads stay the same size as on their base layer, when one zooms out they will become very small and hard to see. It would be nice if the roads would gradually become relatively bigger in size when zooming out (on the screen they would then stay roughly the size absolute size).

As a solution to this problem, it is proposed here that roads in the roads SSC will not merely be represented as triangles, as is now, but as polyhedrons. More specifically, there will have a double tapered structure, where their width continuously grows up until they should disappear. After that, their width will gradually decrease to zero. The concept of this structure is detailed in figure 42. In the bottom part the thickness of the road is increased by creating displaced twins of the outer points of the original road (the plane at the bottom). In the upper part, the thickness of the road is decreased by using exactly the same approach, but this time displacing the planes toward the inside.

In this figure, the bottom taper is the color of the road, in this case red. The top taper is colored with a specific color which all top tapers get, in this case dark blue. The goal of this latter color is that it can be coded that specific colors will not be rendered. This is needed for when the slice plane is taken through the top taper. On the sides, when looking down from the

pixel grid, the first triangle hit will be these dark blue ones. What one would like to render here is what is below this part, but then in the background SSC. By telling the render program to ignore these triangles, in the end visualization the corresponding triangles from the background SSC will be rendered.

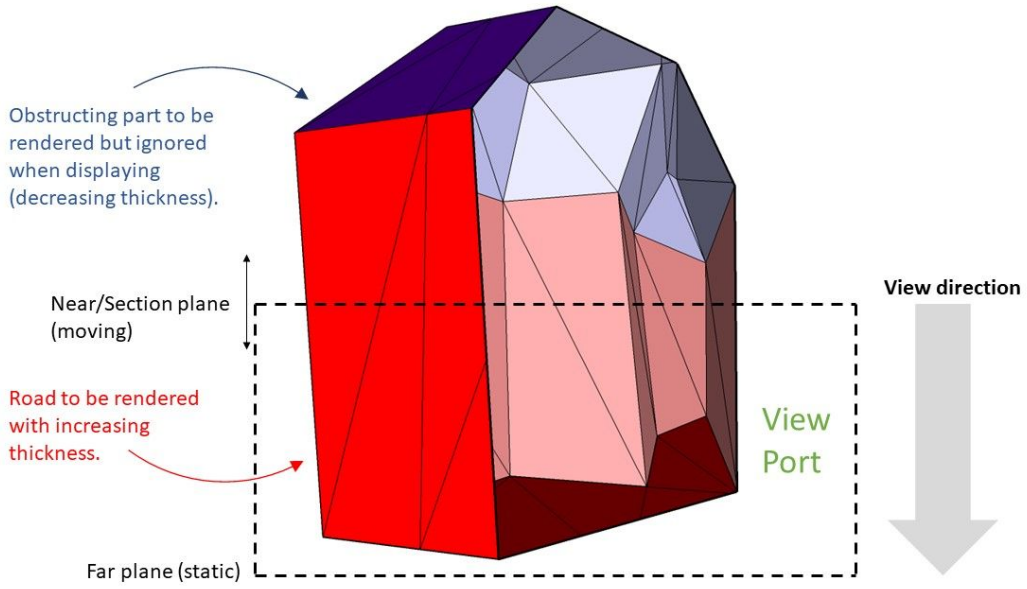


Figure 42. Double tapered approach concept

The result is that when the near/section plane is lifted, the dark blue part takes over (see figure 43). As the dark blue part is rendered but not filtered out in the final step, it would not be visible on the screen. Instead, the original background map would be visible in these pixels, which is again why we require the 'non-render'-color. The fact that the filtering takes place after the rendering but before the visualization is extremely important to allow the original pixels of the background geometry to stay visible.

Figure 43. Continuous changing of the road thickness

Optimally, the geometry in the top (dark blue) part would end up being a line. However, reliably deducing a line from polygon geometry is an extensive task, especially as the road network contains complex crossings. A check was done as the TOP50NL dataset contains roads as line geometry. However, the required quality for the double reversed cone approach is high and improvements of the TOP50NL line generation algorithm would be essential (see figure 44).

Figure 44. Comparison between TOP10NL polygon (green) and TOP50NL (vector) line geometry (red), showing the impact of the generalization process.

5.1.8 Mathematical model for double tapered approach

To implement the double tapered method, a topological analysis first needs to be performed on the input geometry of the roads. This geometry is delivered as two tables: one containing an ordered list of vertices, their coordinates and the type of road they belong to; and another table containing the list of triangles with the corresponding vertex indices (row numbers of the first table). To find the vertices which are on the edge of a road network (an 'outside edge'), a check whether these ones have two neighbour vertices which appear in respectively only one shared triangle needs to be performed. If a point has only neighbours which appear in two shared triangles, this point is not on the edge of the road network geometry, see figure 45.

Furthermore, given that all triangles are oriented in the same way (clockwise or counterclockwise), previous and next neighbor vertices can be defined with the inside of the road network always being on the right side of the polyline 'previous neighbor - edge vertex - next neighbor'. This information is then stored for further processing to create the displaced edge vertices, see also figure 45.

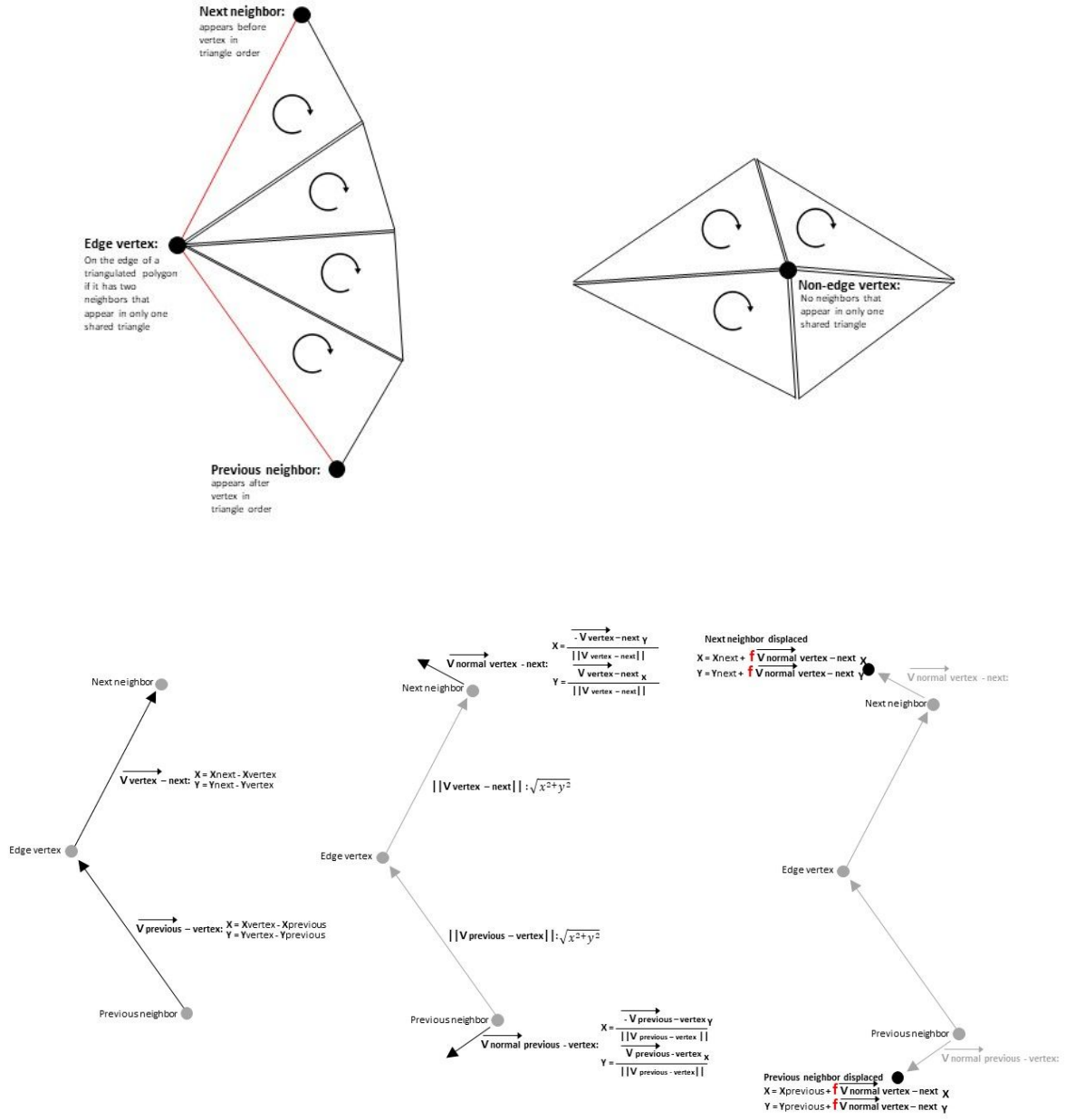


Figure 45. Topological model for the double tapered approach

With this information, the thickness of the road network can then be modified using the following steps. First, identify the vectors between the previous neighbor and the edge vertex; and the vector between the edge vertex and the next neighbor. Using the two first vectors, calculate the respective normal vectors and normalize them so that they have a length of 1. Using the desired increase/decrease of the normal vector, a displaced version of the previous and next neighbor can be calculated. This happens by multiplying the normal vector by the desired factor (a thickness increase of one unit requires a factor $f = 1$, a decrease of one unit requires $f = -1$). The resulting vector can then be used to displace the previous and the next vector. One can then calculate the two line equations of the displaced lines (which are parallel with a distance of f to the previous 'neighbor - edge vertex' and 'edge vertex - next neighbor' lines). The equations $f(x) = ax + b$ can be set up using the vectors calculated in the first step

and the displaced points of the third step. The intersection's *x* coordinate is then obtained by the equation $ax + b = a'x + b'$ and the *y* coordinate can be found by inserting *x* into the *f(x)* of any of the two lines.

The displaced points generated by this process can then be lifted up to the right height in order to generate the desired geometry. The vertical triangles can be generated by connecting each vertex to the new previous and next neighbor. Moreover, a connection to one of its previous level neighbors and to its previous level twin is needed. The same can be done to create the blue vertical triangles. To generate the 'roof', the horizontal blue triangles, not only the displaced edge but also the inner vertices need to be moved to the new height. The new plane can then be created by applying the same triangle topology as at the bottom level, which requires keeping track of the number of added vertices for updating the indexes of the triangles.

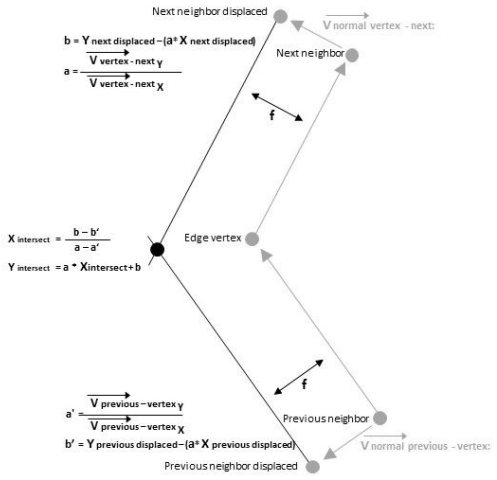


Figure 46. Modifying the thickness of roads according to the double tapered approach

Due to time constraints this model could not be implemented in the adapted demo. Still, a further exploration of this topic was still done (by coding the intersections in sql). This allowed us to identify the most important challenges and aims at preparing for further research. A first point requiring attention is the generation of the flat roof (the horizontal plane at the top of the dark blue part). When doing so one needs to make sure that a correct topology is preserved, especially when the 'roof' is reduced to a less thick road than at the initial level this can be problematic. In fact, none of the inner points (which are not displaced) should find themselves outside the edges of the roads. It might be safest not to reduce the 'roof' to a lower thickness than the original geometry.

Another important point to keep a correct topology is related to the method itself. In fact, when the interior angle (the angle 'previous neighbor - edge vertex - next neighbor') becomes smaller, the distance between the original edge vertex and the displaced edge vertex increases. This is problematic as too high displacements might create conflicts with other roads in the neighborhood. The introduction of a maximum threshold seems to be a solution but prompts the question of what the displacement would look like in that case. Simply moving the displaced point closer to the original one doesn't seem to be a solution as this might negatively influence the user experience by creating strange variations in road thickness.

A third point is related to the calculation method itself. It might be noted that this method does not work if the segments 'previous neighbor - edge vertex' and 'edge vertex - next neighbor' are perfectly parallel (or in the prolongation of each other). When this situation occurs, the displaced edge vertex can simply be found by applying the normal and normalized vector to the edge vertex.

A final observation which could be done during the coding implementation is that inaccuracies tend to appear the closer lines are to the parallel situation. However, it appears this is not a strict tendency as there seems to be a random parameter involved too. A possible explanation of this phenomenon is that small inaccuracies induced by finite number representation have bigger consequences when lines are nearly parallel. For instance, a small rounding of the a values of step 4 (the slopes of the lines) will induce bigger divergence on the intersection if this one is far away from the original point than if it close to it.

As mentioned, this approach has not been implemented due to time constraints. However, the theoretical model proposed here can be implemented in future research. Up until then, the implementation has been updated by maintaining different road hierarchy levels up until certain zoom levels by separating the roads from the background SSC and placing them in the road SSC.

5.2. Making the viewer compatible for mobile devices

Next to maintaining the roads, it was attempted to make the implementation available for mobile devices. The modifications to make this happen was exclusively performed on the Javascript at the client side. The structure of the javascript snippets which were used for developing the existing demo (and which are bundled by using node.js and watchify) were also reused. The functions that tell the map canvas how to change what is displayed (the extent/zooming and the location) were already present. Therefore, the task focus moved to catching the user's touchscreen interactions and processing them correctly.

5.2.1 Javascript touchscreen interactions

Just as for a computer's mouse, specific events triggered by the touchscreens of mobile/tablet devices exist and can be caught by the EventListener function of the Canvas (canvas.addEventListener; canvas.removeEventListener). In total, there are three events. The TouchStart event is sent as soon as a one or several fingers touch the screen. As soon as one or several of these fingers move, the TouchMove event is sent. And as soon as some (including one) or all of the fingers are removed from the screen, the TouchEnd event is sent. A main challenge compared to the current implementation is that the panning (moving a finger) and zooming (pinching with fingers) actions use the same types of events. On a traditional desktop, the wheel (mouseWheel) and mouse clicking (mouseDown, mouseMove, mouseUp) allow a clearer separation. The touchpads also send different events - the separation does therefore take place at hardware level, outside of the browser.

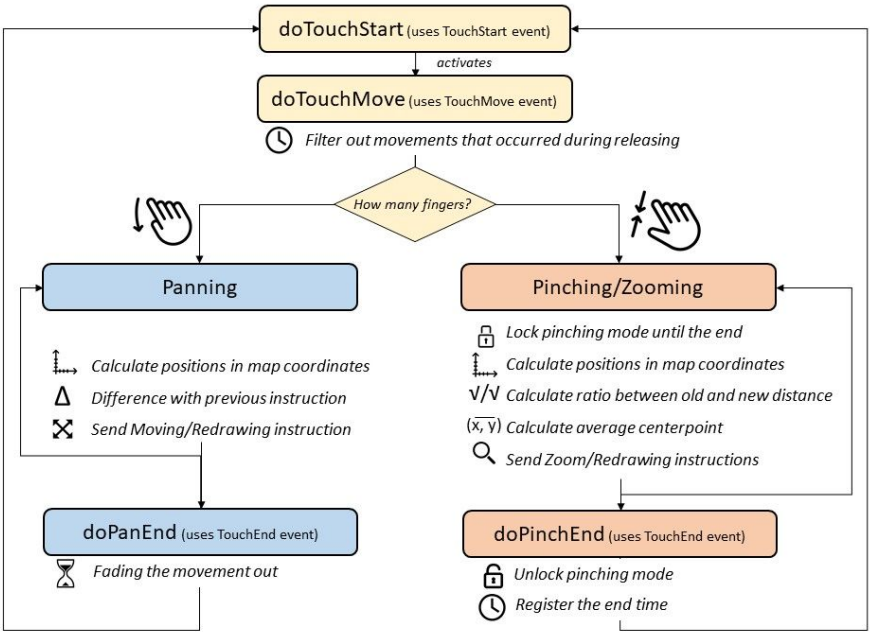


Figure 47. Work principle of JavaScript touchscreen interaction

In opposition, to be able to separate touch screen moving/panning and pinching/zooming, if-statements have to be implemented in the client-side run javascript. First, the EventListener is created by specifying which function is created when the event occurs.

```
canvas.addEventListener("touchstart", doTouchStart, false);
```

Second, the event is passed as input to the function which is triggered.

```
function doTouchStart(evt)
```

Third, the number of touches at that moment can be accessed with the evt.changedTouches. Length object. In such way, it is possible to identify the number of fingers present.

```
const touches = evt.changedTouches;
if (touches.length==1 && touchmode == undefined) {
```

Furthermore, the coordinates of the touches can be accessed with the following statements (the r.left and canvas.clientLeft are used to transform screen coordinates into canvas coordinates).

```
new_pos_X_0 = touches[0].pageX - r.left - canvas.clientLeft;
new_pos_X_1 = touches[1].pageX - r.left - canvas.clientLeft;
new_pos_Y_0 = touches[0].pageY - r.left - canvas.clientLeft;
new_pos_Y_1 = touches[1].pageY - r.left - canvas.clientLeft;
```

These coordinates can then be used to pan the map, *dx* and *dy* being expressions calculating how much the map moved since the last pan TouchMove event:

```
const x = touches[0].pageX - r.left - canvas.clientLeft;
const y = touches[0].pageY - r.top - canvas.clientTop;
let dx = x - prev[0];
let dy = y - prev[1];
map.panBy(dx, dy);
```

When two fingers are used, the coordinates can be used to send a zoom instruction as follows. The focus point for the zoom action is defined as the mean between all the four screen contacts (each finger, both start and end).

```
var zoom_pos_X = (prev_pos_X_0 + prev_pos_X_1 + new_pos_X_0 + new_pos_X_1)/4;
var zoom_pos_Y = (prev_pos_Y_0 + prev_pos_Y_1 + prev_pos_X_1 + prev_pos_Y_1)/4;
}
//calculate the change of distance for the zoom factor!
let new_dist =(new_pos_X_0 - new_pos_X_1)**2+(new_pos_Y_0 - new_pos_Y_1)**2;
let zoom_factor = new_dist/prev_dist;
_map.zoomNeutralAnimated(zoom_pos_X, zoom_pos_Y, zoom_factor);
```

Finally, once one of the fingers is released and the TouchEnd event is sent, depending on the number of fingers active, either the function doPanEnd (named doTouchEnd in the code; one finger, panning) or doPinchEnd (two fingers) will be triggered. The doTouchEnd is essentially a copy of the existing doMouseUp function (drag.js). The doPinchEnd function essentially resets a number of variables and registers the time at which the fingers left the screen.

5.2.2 Limitations of hardware

While the approach to distinguish the presence of one vs. two fingers sounds pretty simple on paper, it is a bit more complicated in practice, which is due to hardware performance limitations. Indeed, touchscreens have a limited reliability, especially in detecting two present fingers. This results in having few moments where only one out of two fingers is detected (theoretically up to 10 fingers are supported). These moments need to be filtered out. Which is done by the touchmode variable which can only be overwritten when it is undefined.

```
const touches = evt.changedTouches;
if (touches.length==1 && touchmode == undefined) {
  touchmode = 1;
→//console.log("touchmode" + touchmode);
} else if (touches.length==2) {
→touchmode =2;
→//console.log("touchmode" + touchmode);
}
```

This prevents the touchmode from switching back to one after it has been switched to two. This filters out any potential interruptions due to losing the detection of one of the finger when pinching. Furthermore, an attempt to detect static fingers and indicate that one as the center position for the zoom action was done. The logical approach here seems to detect whether one of the fingers is static. However, this is never totally the case: instead of being really static, the fingers rather move around a fixed point. A trial to identify a finger which is not moving was done using a threshold, comparing the current position with the very first when the pinch started.

```
if (new_pos_X_0-first_pos_X_0 < threshold &&
new_pos_X_0-first_pos_X_0 >-threshold &&
new_pos_Y_0-first_pos_Y_0 < threshold &&
new_pos_Y_0-first_pos_Y_0 > -threshold)
```

However, it appeared that the interval at which the movement of the fingers are sent to the javascript are not stable (this might be due to the contact between the finger and the touchscreen not being regular). Therefore, the usage of a counter is needed in order to see whether one of the fingers is 'rather' moving or stable:

```
if (pos_0_counter>mult_pos_counter && pos_0_counter>pos_1_counter)
{
var zoom_pos_X = first_pos_X_0;
var zoom_pos_Y = first_pos_Y_0;
console.log("finger 0 static");
} else if (pos_1_counter>mult_pos_counter && pos_1_counter>pos_0_counter)
{
var zoom_pos_X = first_pos_X_1;
var zoom_pos_Y = first_pos_Y_1;
console.log("finger 1 static");
} else
```

However, this attempt was not fruitful either as a static finger was perceived as moving just as many times as it was static. Several thresholds were used to perform a test. While a small threshold led to static fingers not being detected, a bigger threshold led to moving fingers being detected as static. There seems to be no real middle in between. And even if the middle would be found, no warranty that the same middle could be used for another device or for different pinching speeds. Therefore, the implementation of the detection of a static finger was dismissed.

A third aspect which was, this time successfully, explored is the appearance of interactions shortly after the release of the fingers. In fact, the two fingers are actually never released at the same time - often resulting in an unexpected panning when finishing the pinch. This issue is solved by registering the time when the pinch was ended, using a function of the existing draw.js file (in the DoPinchEnd function).

```
//store the finish time to filter out any panning at a later stage
prev_time = Drawing.now();
```

When a pan instruction might occur in the DoTouchMove function, a check was done to make sure the last end of a pinch took place sufficiently long ago.

```
if (touchmode == 1) {
→if ((now - prev_time) < 300)
  {
→ → →//console.log("avoided pan")
      return
  }
```

5.2.3 Differences between browsers

During the development of the support for touchscreens, it appeared that the browser and the hardware have a high impact on the correct functioning of these interactions. Using different phones, it appeared that the demo was only working correctly with the Firefox browser for mobile. Chrome does offer the same functionalities, with the differences that the zoom does only react after releasing the fingers. On Safari for iPhone, only the panning appears to work. This shows that on top of the different touchscreen technologies and reliabilities, the browser also needs to be taken into account.

Moreover, it was observed that depending on the hardware performance, crashes of the browser could occur (especially on mobile devices with smaller GPUs) when interactions are done at a quicker pace than the average 'browsing' user. Because, as mentioned, the binary file that is loaded to the browser is relatively large in our adapted implementation, we clipped the original dataset for the mobile service. In order to limit the size of the task to a doable one, it was decided not to do a complete debugging with several browser/hardware combinations.

5.3 Validation survey

After the adaptations to the vario-scale demo were done, the same BRT-user group as for the first survey was contacted for a validation survey. A different target sample was given for the old and new demo and users were asked to record the time needed to find a location on a different scale. To avoid bias of users having already explored the map, half of the participants received the instruction to do the exercise on the new demo first (v2 on figure 48, 5 respondents), while the other ones (v1 on figure 48, 3 respondents) had to start with the old demo.

Figure 48. Target samples of the old (left) and the new (right) demo exercise. Respectively in blue and red on the next image.

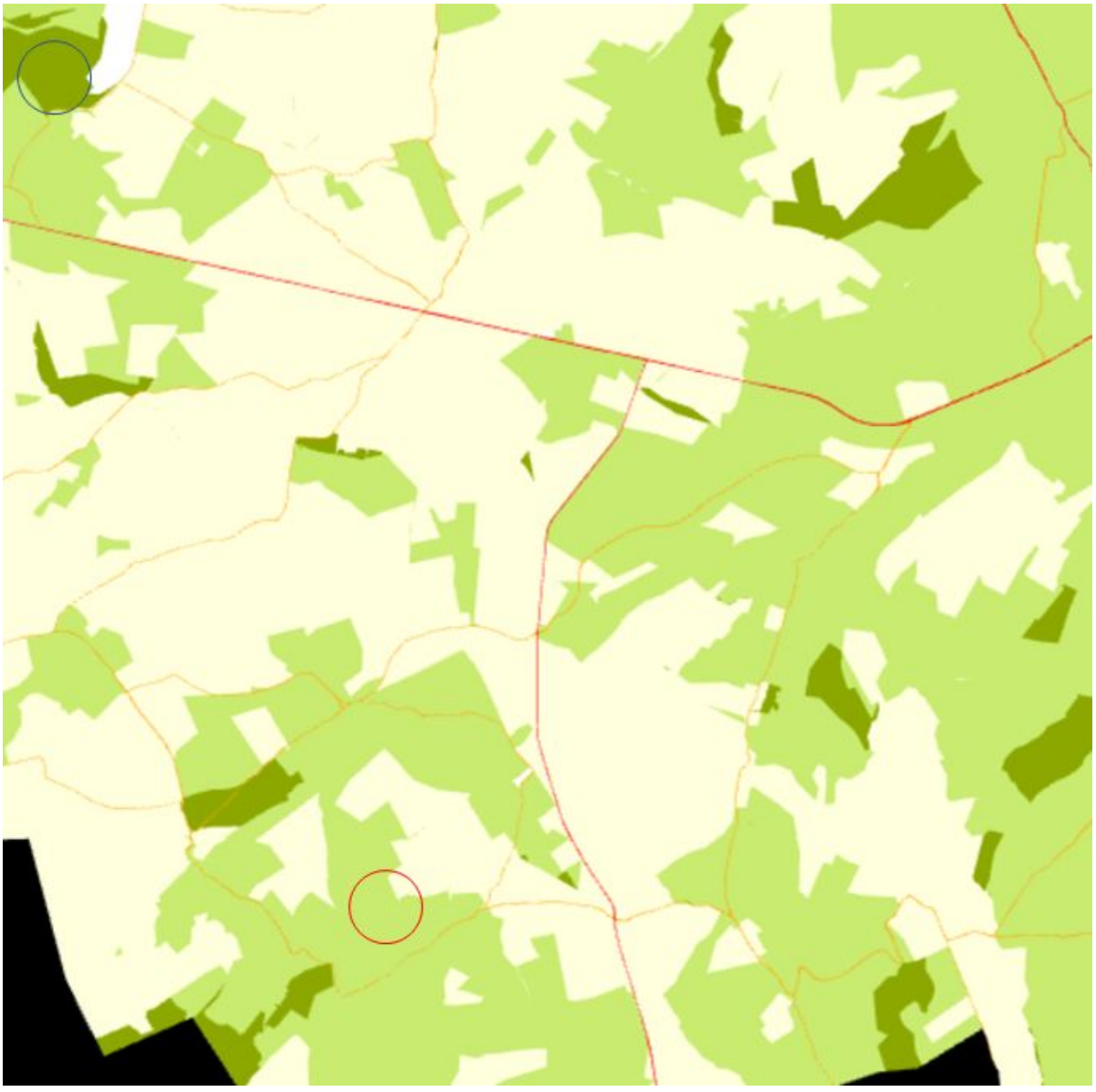


Figure 49. Location of the target samples: old (blue) and new demo (red)

The conclusions that can be drawn from the relative time needed to find the target samples are rather limited. In fact, for some users, the new demo allowed a faster finding while for other users, the old demo appears to be quicker. The standard deviation of the results is nearly twice as big for the new demo (128,5 seconds for the new demo, for the users that stopped before finding the sample) than for the old demo (63 seconds). In both cases, given the average values (101 seconds for the old demo, 190 seconds for the new one) - the standard deviations are too big to allow solid conclusions. Also, the amount of respondents is too low to come to any conclusion for this test. For the target sample in the new demo, two persons did abandon before finding the sample. Two out of nine users were quicker in the new demo. They both belonged to the v1 group. However, a third person of the v1 group did abandon before finding the target sample in the new demo, hereby nuancing these results.

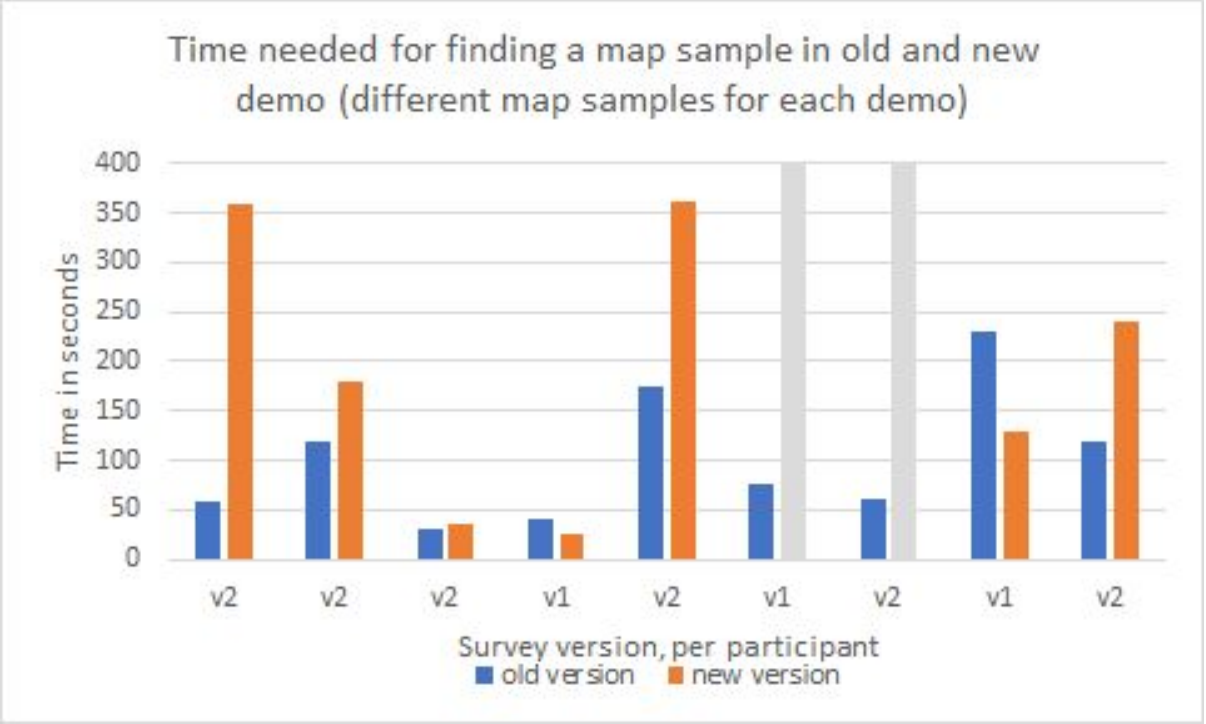


Figure 50. Statistic results of the validation survey

Potential conclusions being rather limited on the data itself, they should rather be drawn on the way to conduct such surveys. In fact, several hypotheses can be made on which factors influenced the mean results and the standard deviations. First, the fact that the survey was conducted by e-mail implied the usage of different hardware. Especially for the new demo which reaches the limits of some devices, the usage of different hardwares can be problematic as user interaction might be less smooth for some users than for others, especially regarding the size of the binary file. The influence of hardware was confirmed by some users which reported to have issues interacting with the map or lacked smoothness. Second, given that two target samples are never equal (in the ease to find them) it is rather hard to compare results between them. This would require a third survey using the same sample on two identical implementations. As about eight participants were expected (with nine respondents so this assumption was rather correct) and a sample of four considered too small, it was therefore decided to use different different target samples for the two demo versions.

Finally it appears that such a survey is hard to conduct with a limited number of participants (more than 20 participants are definitely needed). This would allow the usage of a single target sample for both old and new demo versions - hereby allowing for a more direct comparison. However, a randomised factor might still persist as a form of ‘chance’ when finding the target sample in the demo might still persist. Several factors such as zoom level differences and euclidean distance between start screen and target screen play a role here. An additional important point is that as the client side and thus hardware plays a big role, in opposition to more classical hardware surveys. Therefore, a survey in person rather than online, and using identical hardware should be preferred for these kind of exercise-surveys.

Concludingly, we can say that the two research options have been implemented. The road network is now correct throughout all scales. However, our framework for continuously resizing these roads has not yet been implemented. Mobile services have been implemented.

The feedback towards the users has been made. Their reaction on the implementation, however, was hard to measure due to the survey set up and time constraints.

6. Conclusion / Discussion

Vario-scale is a new data storage and mapping technique that attempts to shift the paradigm towards variable scale geo-information in order to automatically generalize maps and facilitate smooth zoom while minimizing data redundancy. In this research we attempted to bridge the gap between assessing Kadaster topographic data users' interest in vario-scale and trying to improve the existing implementation of vario-scale. The main research question was *how can the implementation of vario-scale be improved to better meet the needs for end users of Kadaster topographic data?*

User needs were assessed and a number of cartographic and other functional needs were determined. From both categories, one research option was chosen, respectively the road network and adapting the implementation for mobile devices. The latter has been implemented without many problems. Adapting the road network was arguably more important since the users indicated that it is very important for orientation during zooming. First a different method was tried where road faces were split and a combination of faces and lines were visualized as a method (Suba et al., 2016). While this method remains more true to the vario-scale concept, the combination of rendering lines and polygons with the same thickness through different scales proved too difficult. This caused a change of method to the current one. Because of this, there was not enough time to fully implement the double tapered approach of visualizing roads. However, the framework has now been developed and in future research this can be fully implemented in the existing demo.

A nice outcome of this research is that the viewer has been adapted so it can process different bin files, and thus different cubes, at the same time. This can lead to a greater flexibility in overlay of different datasets. One of the main user needs was to have labels and symbols represented in vario-scale. Having the possibility of rendering multiple cubes could help this in the future.

A problem we encountered was the size of the background SSC binary file. The smallest bin file we created was roughly eight times the size of the one that is loaded in the browser in the implementation of the previous demo. Much effort was put into trying to reduce the file size but the exact cause of the increase in size was not found. This also led to problems in trying to load the map on the client side. Some browsers crashed trying to render the large bin file. Research has already been conducted into how to process large datasets for vario-scale purposes (Meijers et al., 2015; Rovers, 2016; Xu, 2017). More research can perhaps be done in how to fully implement this in the web application.

Another point for future research is solving hardware differences that will affect the client-side rendering. This would enable more coherent client-side performance among different devices. Also, the demo should be adapted to not only work for Firefox but every type of browsers. Furthermore, the assessment of the adaptations with the second survey was not very satisfactory. It would be good if an improved validation survey with larger group of users would be conducted. Currently the validation survey is performed in a small group and therefore it's hard to evaluate the survey results. Random factors such as hardware differences

can have a noticeable influence on the survey process. To obtain a valid evaluation survey at least twice the number of users we had now are needed. A final future research suggestion is that the openGL javascript can be explored more in depth. This way it can be checked whether there are more efficient ways to render two bin files and overlay them.

Regarding the interest of Kadaster in vario-scale technology, the outcomes of this research provide some useful information. It was found that users of geo-information do see the benefits of vario-scale data, in different ways. Seamless zooming, map retrieval at any scale and less network usage are some of those benefits. It became clear that users have heard about vario-scale but still do not have a lot of knowledge about the subject. Implementing new technologies is a leap into the dark for companies, unless those technologies are well researched and standardized. Vario-scale is a broadly researched topic with irrefutable benefits, but the implementation of the software can cause some headaches. The software *as is*, requires a careful installation and a collaborative environment. Right now, it can not be characterized as 'plug and play software'. However, this project shows that a small team can familiarize themselves with the software in a small amount of time. Based on this, it is can be assumed that companies in the geo-information profession could reach the point of offering vario-scale data in a reasonable amount of time, if they are willing to embrace the technology.

Literature

Cantor, D., & Jones, B. (2012). WebGL beginner's guide. Packt Publishing Ltd.

Danchilla, B. (2012). Beginning WebGL for HTML5. Apress.

Dilo, A., Oosterom, P. van, Hofman, A. (2009) Constrained tGAP for generalization between scales: The case of Dutch topographic data. Computers, Environment and Urban Systems, Vol 33, pp. 388–402.

Driel, M. (2015) Real time intersections on Space Scale Cube data. MSc Thesis, Universiteit Utrecht.

Groeneveld, IJ. (2018) Generalisation of Hydrography Networks for a Vario-Scale Basemap. TU Delft.

Harrie, L., Sarjakoski, T., Lehto, L. (2002) A variable-scale map for small-display cartography. Int Arch Photogrammetry Remote Sens Spat Inf Sci. Vol 34(4), pp. 237–242.

Hampe, M., Sester, M., Harrie, L. (2004) Multiple representation databases to support visualization on mobile devices. In: Proceedings of the 20th ISPRS Congress, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Istanbul, Turkey, vol 35(6), pp 135–140.

Kadaster.nl (n.d.). Vector tiles BRT en BGT [Software]. Available from https://geodata.nationaalgeoregister.nl/beta/topotiles-viewer/#8/52.33/5.19

Meijers, M. (2011) Variable-scale Geo-information. PhD Thesis, TU Delft.

Meijers, M., Oosterom, P. van, Quak, W. (2009). A storage and transfer efficient data structure for variable scale vector data, in: M. Sester el al. (eds.), Advances in GIScience, 2009, Springer-Verlag.

Meijers, M., Stoter, J., Oosterom, P. van. (2012). Comparing the vario-scale approach with a discrete multi-representation based approach for automated generalisation of topographic data. 15th ICA Generalisation Workshop, Istanbul, Turkey, 2012

Meijers, M., Suba, R., Oosterom, P. van (2015). Parallel Creation of Vario-Scale Data Structures for Large Datasets. 4th ISPRS International Workshop on Web Mapping and Geoprocessing Services, 01-03 July 2015, Sardinia, Italy.

Meijers, M. (n.d.). Vario-scale Geoinformation webGL demo [Software]. Available from http://varioscale.bk.tudelft.nl/gpudemo/2017/07/one/

NCGIA (1993) Research Initiative: Initiative 3: multiple representations, closing report.

Oosterom, P. van, Meijers, M. (2011) Towards a true vario-scale structure supporting

smooth-zoom, 14th ICA/ISPRS Workshop on Generalization and Multiple Representation, 2011, Paris.

Oosterom, P. van, Meijers, M. (2014) Vario-scale structures supporting smooth zoom and progressive transfer of 2D and 3D data. International Journal of Geographical Information Science, Vol 28(3), pp. 255-278.

Oosterom, P., Meijers, M., Stoter, J., Suba, R. (2014) Data Structures for Continuous Generalisation: tGAP and SSC, in: D. Burghardt et al, Abstracting Geographic Information in a Data rich World, 2014, Springer.

Rovers, A. (2016) Exploring the use of a generic spatial access method for chaching and efficient retrieval of vario-scale data in a client-server architecture. MSc thesis in Geomatics for the Built Environment, TU Delft.

Suba, R., Meijers, M., Oosterom, P. van. (2016) Continuous Road Network Generalization throughout All Scales. International Journal of Geo-Information, Vol 145(5).

TU Delft (2018) Vario-scale Geo-Information. http://varioscale.bk.tudelft.nl/

Xu, Y. (2017) Construction of Responsive Web Service for Smooth Rendering of Large SSC Dataset. MSc Thesis in Geomatics for the Built Environment, TU Delft.

Appendix I

BRT user group meeting survey

Survey

1. What is the biggest potential of varioscale from your perspective?

- continuous zoom/animations
- automatic generalization
- several zoom levels in one visualization/tilted slice
- something else? ……..

2. Visually speaking, what is missing in the demo? what would you add?

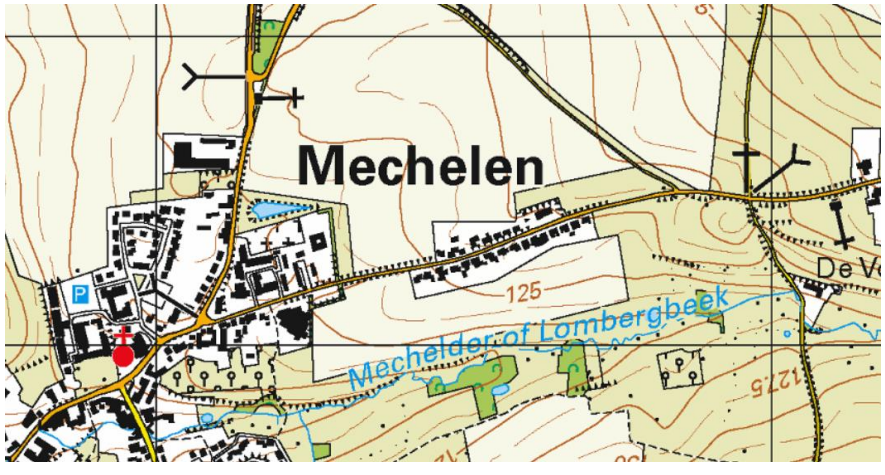
3. Please make the small map orientation exercise on the A3 page!

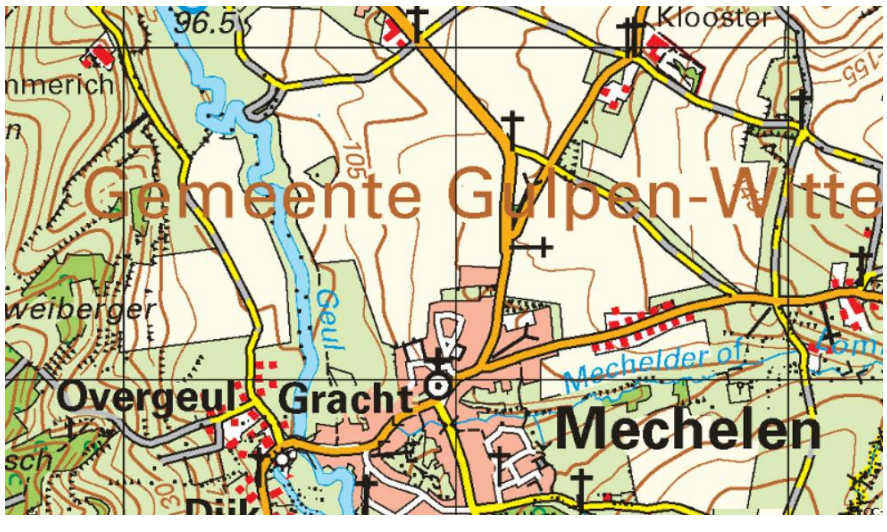
4. Are you willing to help us for the validation of the new demonstration (about 30 minutes, around 15-18th of June)? Please give us your contact details!

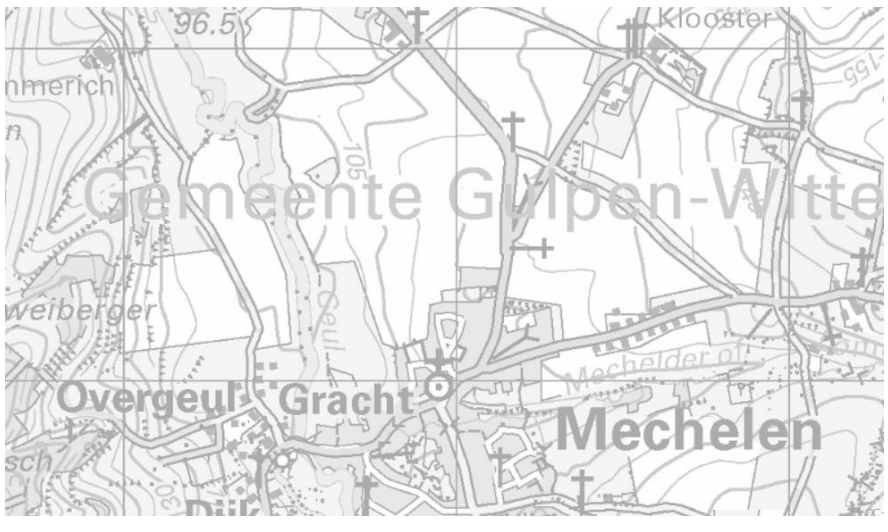
5. At your company, are there specialists in map visualization which we might contact? Or anyone else who might be interested in varioscale?

Name E-mail

The top map shows you a location (red cross) in zoomed-in view. Place the same cross on the smaller scale maps and circle the elements you used for orientation on the black/white maps below.







Appendix II

2-minute online web survey

2-minute survey: Which of the following 5 varioscale use-cases sounds the most promising to you?

This form is part of a TU-Delft MSc Geomatisc synthesis-project which is conducted in collaboration with the Dutch Kadaster. Within this one, we aim to answer the question "How can the implementation of varioscale visualization be improved to better meet the needs of the end-users?".

This form contains a total of 5 use-cases and we would be very thankful if you accept to give your opinion on them!

Thank you,
With kind regards,

Pablo, Pim, Shenglan and Freek

*Required

1) Fisheye: several zoom levels in one view

1. **Fisheye: potential** *
   The continuous transition between the zoom levels is made possible by varioscale technology. More detailed can thus be displayed in the center and less on the sides of the view, creating the effect of a magnifier. Source of the pictures: http://www.gdmc.nl/publications/2014/Vario-scale_smooth_zoom_progressive_transfer.pdf
   Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Low potential	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High potential

2) Perspective view: enhanced tilted maps

2. **Perspective view: potential** *

First applications of tilted maps have been introduced, such as the beta version of the vector tiles of the Dutch kadaster. An issue is that the detail level is the same in the entire view. Using varioscale technology, the detail level could be adapted to the distance from the viewer, avoiding too heavy maps! Source: own gif created using the kadaster vector tile demo (https://geodata.nationaalgeoregister.nl/beta/topotiles-viewer/) and image editing.
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Low potential	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High potential

(For the ones who attended the meeting on monday: the last three use-cases were already mentioned there. Thus, you can directly answer the questions)

3) Seamless zoom-in/out

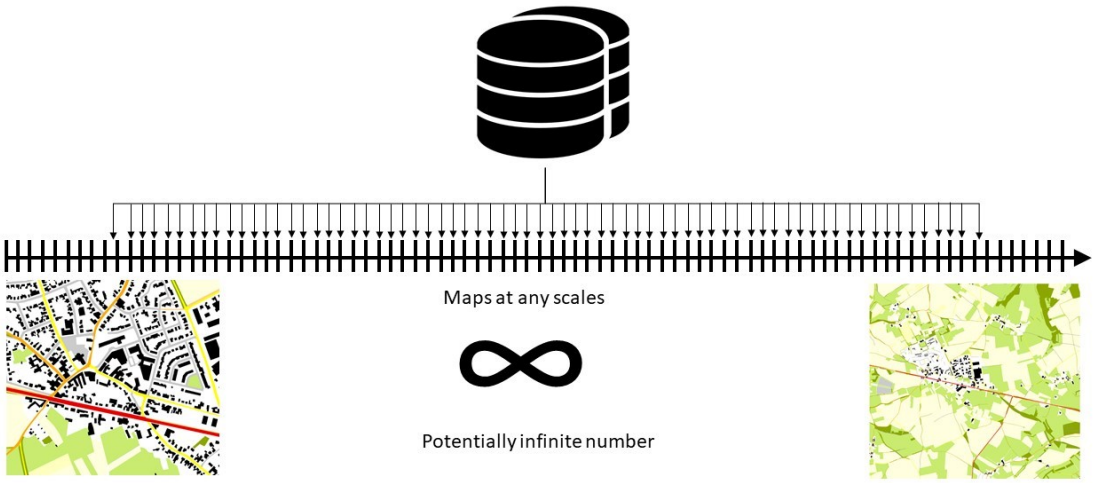
3. **Seamless zoom: potential** *

This is an aspect that might change with varioscale as the content of the map would directly adapt to the specific zoom-level. By applying vario-scales, no abruption between maps take place anymore. The eye does not lose track of what it is looking at. In fact, objects do not suddenly disappear, change styles or are abruptly aggregated anymore. Varioscale acts as a transition animation between maps scales by showing the changes that are performed. Source of the images: varioscale demo by Martijn Meijers (http://varioscale.bk.tudelft.nl/gpudemo/2017/07/one/)

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Low potential	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	High potential

4) Automatic generalization at any scale

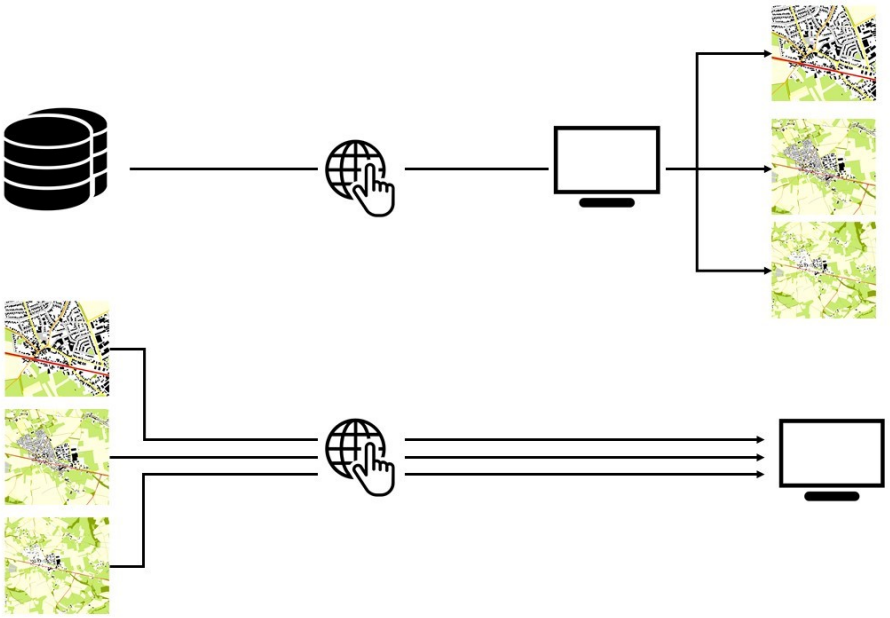


4. **Automatic generalization at any scale: potential** *

Maps are traditionally produced at a fixed set of scales which is often nationally specific and inherited from the past. When automatic generalization is applied, it is custom-made for these scales and new scales require new algorithms. By applying varioscale one algorithm could be used for generating maps of any scale. The number of scales therefore becomes potentially infinite. Source of images: varioscale demo by Martijn Meijers
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Low potential	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High potential

5) Less network usage



5. **Less network usage: potential** *

Within the element generation - rendering - visualization process, the data is usually transferred from server to client side after the rendering. This involves transferring raster data which is rather heavy. By performing the rendering (including transition animations) on the client side, the data is transferred in a rawer stage which induces less network usage. Source of images: varioscale demo by Martijn Meijers
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Low potential	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High potential

6. **Any remarks?**

Appendix III

Python script to display slice from tGAP in QGIS.

```
from qgis.core import *
from qgis.utils import iface

canvas = iface.mapCanvas()
layers = canvas.layers()

MAIN = [10300, 10301, 10302, 10310, 10311, 10312]
REGIO = [10400, 10401, 10402, 10410, 10411, 10412]
LOCAL = [10500, 10501, 10502, 10510, 10511, 10512]
STREET = [10600, 10601, 10602, 10720, 10710, 10730]

ROADS = True

main = True if ROADS else False
regio = True if ROADS else False
local = True if ROADS else False
street = True if ROADS else False

i = 1

query = "step_high >= {0} AND step_low < {1} AND ".format(str(i), str(i))

class_query = "("

if main:
    for n in MAIN:
        class_query += "feature_class = " + str(n) + " OR "
if regio:
    for n in REGIO:
        class_query += "feature_class = " + str(n) + " OR "
if local:
    for n in LOCAL:
        class_query += "feature_class = " + str(n) + " OR "
if street:
    for n in STREET:
        class_query += "feature_class = " + str(n) + " OR "

if len(class_query) > 1:
    query += class_query[:-4] + ")"
else:
    query = query[:-5]

print query

for layer in layers:
    layer.setSubsetString(query)
canvas.refresh()
```

Appendix IV

Python script to create the databases for only roads

```
# encoding: utf-8
from connection import connection

def main(tgap_name):
        """Configure feature data
        """

        # Create query for road codes
        SQL = """
        SELECT DISTINCT
                feature_class, vario_code
        FROM
                {0}_attr_map
        WHERE
                vario_code = 2
        OR
                vario_code = 3
        OR
                vario_code = 4
        OR
                vario_code = 5;
        """.format(tgap_name)

        # Set configuration roads
        db = connection()
        roads = [int(record[0]) for record in db.recordset(SQL)]
        codes = [int(record[1]) for record in db.recordset(SQL)]
        attr_map = {}
        inv_map = {}
        for feature, code in zip(roads, codes):
                attr_map[feature] = code
                inv_map[code] = feature

        rds = ['10411', '10410', '10310', '10311', '10720', '10730', '10600',
'10710', '10510']
        print inv_map
        return

        # Create query for number of steps
        SQL = """
        SELECT
                max(step_high)
        FROM
                {0}_tgap_face;
        """.format(tgap_name)

        # Set configuration steps
        db = connection()
        steps = [int(record[0]) for record in db.recordset(SQL)][0]

        # Create topology name query
        SQL = """
        SELECT
                topology_nm
        FROM
                tgap_metadata
        WHERE
                tgap_nm = '{0}'
        """.format(tgap_name)

        # Set configuration topology name
        db = connection()
        topology_name = [record[0] for record in db.recordset(SQL)][0]

        # Create road table
        SQL = """
        SET CLIENT_ENCODING TO UTF8;
        SET STANDARD_CONFORMING_STRINGS TO ON;
        BEGIN;
```

```
69          DROP TABLE IF EXISTS "{0}_roads_faces";
70          CREATE TABLE "{0}_roads_faces" (
71                  "face_id" numeric NOT NULL,
72                  "feature_class" numeric,
73                  "area" numeric
74          );
75          """.format(topology_name)
76
77          db = connection(True)
78          db.execute(SQL)
79
80          # Record road network
81          SQL = """
82          SELECT
83                  face_id, feature_class, area, mbr_geometry, geometry,
    pip_geometry
84          FROM
85                  {0}_face
86          WHERE
87                  feature_class = {1}
88          """.format(topology_name, roads[0])
89
90          SQL += ''.join([
91          """
92          OR
93                  feature_class = {}
94          """.format(code) for code in roads[1:]])
95
96          # Save road data
97          road_data = []
98          db = connection(True)
99          for record in db.recordset(SQL):
100                 road_data.append(record)
101
102         # Insert road data
103         db = connection(True)
104         db.execute("SELECT AddGeometryColumn('','{0}
    _roads_faces','geometry','28992','POLYGON',2);".format(topology_name))
105         db.execute("SELECT AddGeometryColumn('','{0}
    _roads_faces','pip_geometry','28992','POINT',2);".format(topology_name))
106         db.execute("SELECT AddGeometryColumn('','{0}
    _roads_faces','mbr_geometry','28992','POLYGON',2);".format(topology_name))
107         for face_id, feature_class, area, mbr_geometry, geometry,
    pip_geometry in road_data:
108                 SQL = """INSERT INTO "{0}
    _roads_faces" ("face_id","feature_class","area","mbr_geometry","geometry","pip_geometry")
    VALUES ('{1}','{2}','{3}','SRID=28992;{4}','SRID=28992;{5}','SRID=28992;{6}');
109                 """.format(topology_name, str(face_id), str(feature_class),
    str(area), str(mbr_geometry), str(geometry), str(pip_geometry))
110                 db.execute(SQL)
111         db.execute('COMMIT;')
112
113         # Record all road faces
114         SQL = """
115         SELECT
116                 face_id, feature_class, area, mbr_geometry, geometry,
    pip_geometry
117         FROM {0}_roads_faces
118         """.format(topology_name)
119
120         # Save road faces
121         road_data = []
122         db = connection(True)
123         for record in db.recordset(SQL):
124                 road_data.append(record)
125
126         # Create tGAP roads
127         SQL = """
128         SET CLIENT_ENCODING TO UTF8;
```

```
129         SET STANDARD_CONFORMING_STRINGS TO ON;
130         BEGIN;
131         DROP TABLE IF EXISTS "{0}_tgap_roads_faces";
132         CREATE TABLE "{0}_tgap_roads_faces" (
133                 "face_id" numeric NOT NULL,
134                 "feature_class" numeric,
135                 "area" numeric,
136                 "step_low" numeric,
137                 "step_high" numeric
138         );
139         """.format(topology_name)
140 
141         db = connection(True)
142         db.execute(SQL)
143 
144         # Insert tGAP data
145         db = connection(True)
146         db.execute("SELECT AddGeometryColumn('','{0}
    _tgap_roads_faces','geometry','28992','POLYGON',2);".format(topology_name))
147         db.execute("SELECT AddGeometryColumn('','{0}
    _tgap_roads_faces','pip_geometry','28992','POINT',2);".format(topology_name))
148         db.execute("SELECT AddGeometryColumn('','{0}
    _tgap_roads_faces','mbr_geometry','28992','POLYGON',
    2);".format(topology_name))
149         step_low = 0
150         for face_id, feature_class, area, mbr_geometry, geometry,
    pip_geometry in road_data:
151                 step_high = int(feature_class) - 1
152                 SQL = """INSERT INTO "{0}
    _tgap_roads_faces" ("face_id","feature_class","area","mbr_geometry","geometry","pip_geometry
    "step_low", "step_high") VALUES ('{1}','{2}','{3}','SRID=28992;
    {4}','SRID=28992;{5}','SRID=28992;{6}','{7}','{8}');
153                 """.format(topology_name, str(face_id), str(feature_class),
    str(area), str(mbr_geometry), str(geometry), str(pip_geometry),
    str(step_low), str(step_high))
154                 db.execute(SQL)
155         db.execute('COMMIT;')
156 
157         # Create edge table
158         SQL = """
159         SET CLIENT_ENCODING TO UTF8;
160         SET STANDARD_CONFORMING_STRINGS TO ON;
161         BEGIN;
162         DROP TABLE IF EXISTS "{0}_tgap_roads_edges";
163         CREATE TABLE "{0}_tgap_roads_edges" (
164                 "edge_id" numeric NOT NULL,
165                 "left_face_id" numeric,
166                 "right_face_id" numeric,
167                 "end_node_id" numeric,
168                 "step_low" numeric,
169                 "step_high" numeric
170         );
171         """.format(topology_name)
172 
173         db = connection(True)
174         db.execute(SQL)
175 
176         # Record road edges
177         SQL = """
178         SELECT DISTINCT
179                 e.edge_id, e.left_face_id, e.right_face_id, e.end_node_id,
    e.geometry, r.step_low, r.step_high
180         FROM
181                 vario_edge e
182     RIGHT JOIN vario_tgap_roads_faces r ON
183                 e.left_face_id = r.face_id
184         OR
185                 e.right_face_id = r.face_id
186         """.format(topology_name)
```

```
187
188         # Save road edges data
189         edge_data = []
190         db = connection(True)
191         for record in db.recordset(SQL):
192                 edge_data.append(record)
193
194         # Insert edge data
195         db = connection(True)
196         db.execute("SELECT AddGeometryColumn('','{0}
    _tgap_roads_edges','geometry','28992','LINESTRING',2);".format(topology_name))
197         for edge_id, left_face_id, right_face_id, end_node_id, geometry,
    step_low, step_high in edge_data:
198                 SQL = """INSERT INTO "{0}
    _tgap_roads_edges" ("edge_id","left_face_id","right_face_id","end_node_id","geometry","step_
    VALUES ('{1}','{2}','{3}','{4}','SRID=28992;{5}','{6}','{7}');
199                 """.format(topology_name, str(edge_id), str(left_face_id),
    str(right_face_id) ,str(end_node_id), str(geometry), str(step_low),
    str(step_high))
200                 db.execute(SQL)
201         db.execute('COMMIT;')
202
203 main('result')
```

Appendix V

Python script to serialize the roads to a .OBJ file.

```
from connection import connection
from polyhedron_structure import PolyhedronStructure, serialize_obj_groups
from tri import ToPointsAndSegments, triangulate
from tri.delaunay import RegionatedTriangleIterator, TopologyViolationError
from itertools import groupby

def serialize(topology_name):
        """Write road tGAP to file
        """

        # Create structure of bottom
        structure = PolyhedronStructure(universe_id = 0)

        SQL = """
        SELECT
                face_id, feature_class, step_low, step_high
        FROM
                {0}_tgap_roads_faces
        """.format(topology_name)

        # Insert faces in structure
        db = connection(True)
        for face_id, feature_class, step_low, step_high in db.recordset(SQL):
                attributes = {'feature_class': feature_class, 'step_low':
step_low, 'step_high': step_high}
                structure.add_polyhedron(face_id, attributes)

        ranges = [9595 / 3 * x for x in range(0, 4)]
        ranges[-1] = 8500
        print ranges
        return
        for epoch, height in zip(range(0, 4), ranges):

                SQL = """
                SELECT
                        f.face_id, f.step_low, e.edge_id, e.step_low,
e.step_high, e.geometry
                FROM
                        {0}_tgap_roads_faces f,
                        {0}_tgap_roads_edges e
                WHERE
                        f.step_high > {1}
                AND
                (
                        f.face_id = left_face_id
                OR
                        f.face_id = right_face_id
                )
                ORDER BY
                        f.face_id
                """.format(topology_name, str(epoch))

                # Insert facets of all other faces
                db = connection(True)
                for k, v in groupby(db.recordset(SQL), key = lambda record:
record[0]):
                        pts_segs = ToPointsAndSegments()
                        for (face_id, step_low, edge_id, edge_step_low,
edge_step_high, geometry) in v:
                                for pt in geometry:
                                        pts_segs.add_point(pt)
                                for i, j in zip(xrange(0, len(geometry)-1),
xrange(1, len(geometry))):
                                        start, end = geometry[i], geometry[j]
                                        pts_segs.add_segment(start, end)
                                assert edge_step_low <= step_low <=
edge_step_high
                        try:
                                dt = triangulate(pts_segs.points,
```

```
    pts_segs.infos, pts_segs.segments)
64                                  for group, depth, triangle in
    RegionatedTriangleIterator(dt):
65                                          if depth == 1:
66                                                  tri = [(v.x, v.y, height) for
    v in triangle.vertices]
67                                                  structure.add_facet([tri],
    None, face_id)
68                          except TopologyViolationError:
69                                  raise ValueError ('Problem with face
    {0}'.format(face_id))
70
71          return structure
72
73  with open('../binfile/roads.obj', 'w') as io:
74          serialize_obj_groups(serialize('vario'), io)
```

Appendix VI

Python script to transform a .OBJ file to a .BIN file.

```
from array import array

def lut():
    """
    Produce lookup table for rgb values, based on feature class (Top10NL).
    """
    lut = {}
    inv_map = {2: (132, 132, 132), 3: (255, 165, 0), 4: (255, 165, 0), 5:
(255, 0, 0)}
    fclassnew = [
        13100, 13300, 13400, 13000, 10000, 10001, 10002, 10100, 10101, 10102,
        10200, 10201, 10202, 10300, 10301, 10302, 10310, 10311, 10312, 10400,
        10401, 10402, 10410, 10411, 10412, 10500, 10501, 10502, 10510, 10511,
        10512, 10600, 10601, 10602, 10740, 10741, 10742, 10750, 10751, 10752,
        10760, 10761, 10762, 10780, 10781, 10782, 10700, 10701, 10702, 10710,
        10711, 10712, 10790, 10791, 10792, 10720, 10721, 10722, 10730, 10731,
        10732, 12400, 12500, 12600, 12405, 12505, 12605, 12800, 12820, 12810,
        12700, 14000, 14002, 14010, 14012, 14020, 14022, 14030, 14032, 14040,
        14042, 14050, 14052, 14060, 14062, 14070, 14072, 14080, 14082, 14090,
        14092, 14100, 14102, 14110, 14112, 14120, 14122, 14130, 14132, 14140,
        14142, 14160, 14162, 14170, 14172, 14180, 14182, 14190, 14192]
    rednew = [
        168, 156, 204,   0, 204, 204, 204, 204, 204, 204,
        153, 153, 153, 230, 230, 230, 230, 230, 230, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 179, 179, 179, 156, 156, 156, 190,
        190, 190, 190, 190, 190, 190, 190, 190, 190, 104,
        104, 255, 255, 110, 110, 156, 156, 201, 201, 255,
        255, 140, 140, 140, 140, 140, 140, 140, 140, 204,
        204, 204, 204, 255, 255, 201, 201, 252, 252, 255,
        255, 201, 201, 255, 255, 255, 255]
    greennew = [
        0, 156, 179,  0, 204, 204, 204, 204, 204, 204,
        96,  96,  96,  0,   0,   0,   0,   0,   0, 170, 170, 170,
        170, 170, 170, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 211, 211, 211, 167, 167, 167, 167, 167,
        167, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 179, 179, 179, 156, 156, 156, 232,
        232, 232, 232, 232, 232, 232, 232, 232, 232, 104,
        104, 255, 255, 110, 110, 156, 156, 235, 235, 255,
        255, 168, 168, 168, 168, 168, 168, 168, 168, 204,
        204, 204, 204, 255, 255, 235, 235, 179, 179, 255,
        255, 235, 235, 255, 255, 255, 255]
    bluenew = [
        0, 156, 102,  0, 204, 204, 204, 204, 204, 204,
        137, 137, 137,   0,   0,  0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,  0,   0,   0,   0,   0,   0,   0, 255,
        255, 255, 127, 127, 127, 127, 127, 127, 127, 127,
        127, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255,   0,   0,   0, 156, 156, 156, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 104,
        104, 222, 222, 110, 110, 156, 156, 112, 112, 190,
        190,   0,   0,   0,   0,   0,   0,   0,   0, 204,
        204, 204, 204, 222, 222, 112, 112, 251, 251, 255,
        255, 112, 112, 255, 255, 115, 115]

    colors = [rednew, greennew, bluenew]

    for i in range(2, 6):
        fclassnew.append(i)
        for col, key in zip(colors, inv_map[i]):
                        col.append(key)

    l = zip(fclassnew, rednew, greennew, bluenew)
    for item in l:
        lut[item[0]] = item[1:]
    return lut
```

```
69
70  def main(in_name, out_name):
71
72          zero_based = lambda x: x-1
73          LUT = lut()
74          color = None
75          count = 0
76          arr = array('f')
77
78          f_in = open(in_name, 'r')
79          f_out = open(out_name, 'wb')
80
81          ct = 0
82
83          vertices = []
84
85          lines = f_in.readlines()
86
87          for line in lines:
88                  if line.startswith('v'):
89                          v = map(float, line.strip().split(' ')[1:])
90                          vertices.append(v)
91
92          x = [_[0] for _ in vertices]
93          y = [_[1] for _ in vertices]
94          bounds = min(x), max(x), min(y), max(y)
95          deltas = [3000, -3000, 1000, -3000]
96          # deltas = [0, 0, 0, 0]
97          new_bounds = [a + b for a, b in zip(bounds, deltas)]
98          minx, maxx, miny, maxy = new_bounds
99
100         for line in lines:
101                 if line.startswith('g'):
102                         color = map(int, line.strip().split(' ')[1:])[1]
103
104                 elif line.startswith('f'):
105                         f = map(zero_based, map(int, line.strip().split(' ')
    [1:]))
106
107                         if len(f) == 3:
108                                 rgb = LUT[color]
109                                 check = 0
110                                 for vertex_indices in f:
111                                         if (vertices[vertex_indices][0] <
    minx) or (vertices[vertex_indices][0] > maxx) or (vertices[vertex_indices][1]
    > maxy) or (vertices[vertex_indices][1] < miny):
112                                                 check += 1
113
114                                 if check == 3:
115                                         continue
116
117                                 for vertex_indices in f:
118                                         if ct > 20000000:
119                                                 arr.tofile(f_out)
120                                                 arr = array('f')
121                                                 ct = 0
122
123                                         [arr.append(x) for x in
    vertices[vertex_indices]]
124                                         [arr.append(c/255.) for c in rgb]
125                                         ct += 6
126
127         f_in.close()
128         arr.tofile(f_out)
129         f_out.close()
130
131 if __name__ == '__main__':
132         main('road_io_new.obj', 'road_io_new.bin')
133
```

Appendix VII

Python script to cut of the bottom of the background tGAP.

```python
# encoding: utf-8
from connection import connection

def main(tgap_name, n):
        """
        tgap_name -- name of tgap
        n -- place where to cut off
        """

        # Cut out all faces that end before/on threshold
        SQL = """
        DROP TABLE IF EXISTS final_tgap_face;
        CREATE TABLE final_tgap_face AS (
        SELECT * FROM {0}_tgap_face
        WHERE step_high > {1}
        );
        """.format(tgap_name, str(n))
        db = connection(True)
        db.execute(SQL)

        # Update all step lows to threshold of faces
        SQL = """
        UPDATE final_tgap_face
        SET step_low = {0}
        WHERE step_low < {0};
        """.format(str(n))
        db = connection(True)
        db.execute(SQL)

        # Level all steps to normal scale
        SQL = """
        UPDATE final_tgap_face
        SET step_low = step_low - {0};
        UPDATE final_tgap_face
        SET step_high = step_high - {0};
        """.format(str(n))
        db = connection(True)
        db.execute(SQL)

        print 'face table made'

        # Cut out all edges that end before/on threshold
        SQL = """
        DROP TABLE IF EXISTS final_tgap_edge;
        CREATE TABLE final_tgap_edge AS (
        SELECT * FROM {0}_tgap_edge
        WHERE step_high > {1}
        );
        """.format(tgap_name, str(n))
        db = connection(True)

        print 'connected for edge'

        db.execute(SQL)

        print 'cut out edges'

        # Update all step lows to threshold of edges
        SQL = """
        UPDATE final_tgap_edge
        SET step_low = {0}
        WHERE step_low < {0};
        """.format(str(n))
        db = connection(True)
        db.execute(SQL)

        print 'update step lows'

        # Level all steps to normal scale
```

```
70          SQL = """
71          UPDATE final_tgap_edge
72          SET step_low = step_low - {0};
73          UPDATE final_tgap_edge
74          SET step_high = step_high - {0};
75          """.format(str(n))
76          db = connection(True)
77          db.execute(SQL)
78
79          print 'done'
80
81  def loop(tgap_name, n):
82
83          to_be_updated = {}
84
85          # Record face table
86          SQL = """
87          SELECT face_id
88          FROM final_tgap_face
89          """
90          db = connection(True)
91          face_table = set([x[0] for x in db.recordset(SQL)])
92          face_table.add(0)
93
94          # Record the hierarchy
95          SQL = """
96          SELECT face_id, parent_face_id
97          FROM {0}_tgap_face_hierarchy
98          """.format(tgap_name)
99          db = connection(True)
100         hier_table = {}
101         for face_id, parent_id in db.recordset(SQL):
102                 hier_table[face_id] = parent_id
103
104         SQL = """
105         SELECT edge_id, left_face_id_low, right_face_id_low
106         FROM final_tgap_edge
107         WHERE step_low = 0
108         """
109         db = connection(True)
110         edge_set = list(db.recordset(SQL))
111         db.close()
112
113         # Start of loop
114         for edge_id, left_low, right_low in edge_set:
115
116                 left_id = left_low
117
118                 while 1:
119                         if left_id in face_table:
120                                 break
121                         else:
122                                 left_id = hier_table[left_id]
123
124                 right_id = right_low
125                 while 1:
126                         if right_id in face_table:
127                                 break
128                         else:
129                                 right_id = hier_table[right_id]
130
131                 to_be_updated[edge_id] = left_id, right_id
132
133         ct = len(to_be_updated)
134
135         print 'start counting'
136
137         for edge_id in to_be_updated:
138                 SQL = """
```

```
                UPDATE final_tgap_edge
                SET left_face_id_low = {1}
                WHERE edge_id = {0};
                UPDATE final_tgap_edge
                SET right_face_id_low = {2}
                WHERE edge_id = {0};
                """.format(edge_id,
                                   to_be_updated[edge_id][0],
                                   to_be_updated[edge_id][1])

                db = connection(True)
                db.execute(SQL)
                db.close()
                print ct
                ct -= 1

        # Update face hierarchy table
        SQL = """
        DROP TABLE IF EXISTS final_tgap_face_hierarchy;
        CREATE TABLE final_tgap_face_hierarchy AS (
        SELECT * FROM {0}_tgap_face_hierarchy
        WHERE face_id IN (SELECT face_id FROM final_tgap_face))
        """.format(tgap_name)
        db = connection(True)
        db.execute(SQL)
        db.close()

        # Update face hierarchy table ranges
        SQL = """
        UPDATE final_tgap_face_hierarchy
        SET step_low = step_low - {0};
        UPDATE final_tgap_face_hierarchy
        SET step_high = step_high - {0};
        """.format(str(n))
        db = connection(True)
        db.execute(SQL)
        db.close()

        SQL = """
        UPDATE final_tgap_face_hierarchy
        SET step_low = 0
        WHERE step_low < 0;
        UPDATE final_tgap_face_hierarchy
        SET step_high = 0
        WHERE step_high < 0;
        """.format(str(n))
        db = connection(True)
        db.execute(SQL)
        db.close()

        # VACUUM ALL TABLES

        SQL = """
        VACUUM (FULL, ANALYZE) final_tgap_face_hierarchy;
        VACUUM (FULL, ANALYZE) final_tgap_face;
        VACUUM (FULL, ANALYZE) final_tgap_edge;
        """
        db = connection(True)
        db.execute(SQL)
        db.close()

if __name__ == "__main__":
        main('result', 3645)
        loop('result', 3645)
```

Appendix VIII

Demonstration of disk size and number of records of all possible tGAP structures. Sizes of .BIN files are also included.

Before operations

edge_table	5040 kB	**26208** records
face_table	7528 kB	**13238** records

Merge

before cutting bottom

tgap_edge	18 MB	**37143** records
tgap_face	49 MB	**26475** records

after cutting bottom

tgap_edge	18 MB	**23541** records
tgap_face	76 MB	**19189** records
bin file		**~170** MB

Merge + Simplify

before cutting bottom

tgap_edge	18 MB	**37143** records
tgap_face	49 MB	**26475** records

after cutting bottom

tgap_edge	18 MB	**23541** records
tgap_face	76 MB	**19189** records
bin file		**~170** MB

Merge + Split

before cutting bottom

tgap_edge	40 MB	**72085** records
tgap_face	101 MB	**40148** records

after cutting bottom

tgap_edge	34 MB	**30376** records
tgap_face	139 MB	**19295** records
bin file		**~340** MB

Merge + Simplify + Split

before cutting bottom

tgap_edge	40 MB	**72091** records
tgap_face	101 MB	**40150** records

after cutting bottom

tgap_edge	34 MB	**30376** records
tgap_face	138 MB	**19295** records
bin file		**~340** MB

Appendix IX

Javascript file that draws multiple layers on the viewer.

```
"use strict";

/* from mapbox-gl-js - BSD licensed? */
module.exports.now = (function() {
    if (window.performance &&
        window.performance.now) {
        return window.performance.now.bind(window.performance);
    } else {
        return Date.now.bind(Date);
    }
}());

const frame = window.requestAnimationFrame ||
    window.mozRequestAnimationFrame ||
    window.webkitRequestAnimationFrame ||
    window.msRequestAnimationFrame;

exports.frame = function(fn) {
    return frame(fn);
};

const cancel = window.cancelAnimationFrame ||
    window.mozCancelAnimationFrame ||
    window.webkitCancelAnimationFrame ||
    window.msCancelAnimationFrame;

exports.cancelFrame = function(id) {
    cancel(id);
};

exports.timed = function (fn, dur, ctx) {
    if (!dur) {
        fn.call(ctx, 1);
        return null;
    }

    let abort = false;
    const start = module.exports.now();

    function tick(now) {
        if (abort) return;
        now = module.exports.now();

        if (now >= start + dur) {
            fn.call(ctx, 1);
        } else {
            let k = (now - start) / dur
            fn.call(ctx, k);
            exports.frame(tick);
        }
    }

    exports.frame(tick);

    return function() { abort = true; };
};
/* END from mapbox-gl-js */

exports.draw = function (el)
//new modification
{
    var vertexShaderText =
    [
        'precision highp float;',
        '',
        'attribute vec3 vertexPosition_modelspace;',
        'attribute vec4 vertexColor;',
        'uniform mat4 M;',
        'varying vec4 fragColor;',
```

```
70          '',
71          'void main()',
72          '{',
73          '  fragColor = vertexColor;',
74          '  gl_Position = M * vec4(vertexPosition_modelspace, 1.0);',
75          '}'
76      ].join('\n');
77
78      var fragmentShaderText =
79      [
80          'precision mediump float;',
81          '',
82          'varying vec4 fragColor;',
83          '',
84          'void main()',
85          '{',
86          '  gl_FragColor = vec4(fragColor);',
87      '}'
88      ].join('\n');
89
90      // get a WebGL program
91
92      var canvas = document.getElementById('canvas');
93      var gl = canvas.getContext('webgl', { alpha: false , antialias: true});
94
95      //test if the browser has webgl
96      //if (!gl)
97      //{
98      //    //console.log('WebGL not supported, falling back on experimental-
    webgl');
99      //    gl = canvas.getContext('experimental-webgl');
100     //}
101     if (!gl)
102     {
103         alert('Your browser does not support WebGL');
104     }
105     //console.log('going !' + gl.getParameter(gl.MAX_VIEWPORT_DIMS));
106
107     // create vertex and fragment shader
108     var vertexShader = gl.createShader(gl.VERTEX_SHADER);
109     var fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
110
111     // vertex shader source and compile
112     gl.shaderSource(vertexShader, vertexShaderText);
113     gl.compileShader(vertexShader);
114     if (!gl.getShaderParameter(vertexShader, gl.COMPILE_STATUS))
115     {
116         console.error('ERROR compiling vertex shader!',
117                       gl.getShaderInfoLog(vertexShader));
118         return;
119     }
120
121     // fragment shader source and compile
122     gl.shaderSource(fragmentShader, fragmentShaderText);
123     gl.compileShader(fragmentShader);
124     if (!gl.getShaderParameter(fragmentShader, gl.COMPILE_STATUS))
125     {
126         console.error('ERROR compiling fragment shader!',
127                       gl.getShaderInfoLog(fragmentShader));
128         return;
129     }
130
131     // create program: attach, link, validate, detach, delete
132     var program = gl.createProgram();
133     gl.attachShader(program, vertexShader);
134     gl.attachShader(program, fragmentShader);
135     gl.linkProgram(program);
136     if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
137         console.error('ERROR linking program!',
```

```
                    gl.getProgramInfoLog(program));
        return;
    }
    gl.validateProgram(program);
    if (!gl.getProgramParameter(program, gl.VALIDATE_STATUS)) {
        console.error('ERROR validating program!',
                    gl.getProgramInfoLog(program));
        return;
    }
    gl.detachShader(program, vertexShader);
    gl.detachShader(program, fragmentShader);
    gl.deleteShader(vertexShader);
    gl.deleteShader(fragmentShader);


    var triangleRoadPositionBuffer = gl.createBuffer();
    var triangleAreaPositionBuffer = gl.createBuffer();

    var _loaded = false;


    if (window.Worker)  // Check if Browser supports the Worker api.
    {
        // Requires script name as input
        initBuffer(triangleRoadPositionBuffer, 'out_roads.bin');
        initBuffer(triangleAreaPositionBuffer, "out_background.bin");
        _loaded = true;
    }
    else
    {
        alert('Worker API not supported - No data retrieval :(');
    }

    function initBuffer(buffer, filenm)
    {
        var myWorker = new Worker("worker.js");
        console.log('start worker')
        myWorker.onmessage = function (evt)
        {
            console.log('Message received from worker');
            let vertices = new Float32Array(evt.data);
            // bind buffer
            gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
            gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
            // remember size
            buffer.itemSize = 3;
            buffer.numItems = (vertices.length)/6 - 1;
            // we do not keep the worker alive
            myWorker.terminate()
            // inside the worker we have implemented: close();
        };
        myWorker.postMessage(filenm);
    }


    gl.useProgram(program);

    function draw(matrix, near)
    {
        if (_loaded === true)
        {
            gl.clearColor(0., 0., 0., 1.0);
            gl.clearDepth(1.0);
            gl.clear(gl.COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT);
            singleDraw(triangleAreaPositionBuffer, matrix);

            //start computing matrix for road
            var matrix_road = matrix
            var far_road = near - 3000;
```

```
            var near_road = 9595;
            matrix_road[10] = -2.0 / (near_road - far_road)
            matrix_road[14] = (near_road + far_road) / (near_road - far_road)
            singleDraw(triangleRoadPositionBuffer, matrix_road);
        }
    }

    function singleDraw(buffer, matrix)
    {
            gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
            const positionAttrib = gl.getAttribLocation(program,
                                                'vertexPosition_modelspace');
            gl.vertexAttribPointer(
              positionAttrib,
                3,
                gl.FLOAT,
                false,
                24,
                0
            );

            const colorAttrib = gl.getAttribLocation(program, 'vertexColor');
            gl.vertexAttribPointer(
                colorAttrib,    // * Attribute location
                4,              // * Number of elements per attribute
                gl.FLOAT,       // * Type of elements
                false,          // * Is normalized?
                24,             // * Size of an individual vertex
                12              // * Offset from the beginning of
                                //     a single vertex to this attribute
            );

            gl.enableVertexAttribArray(positionAttrib);
            gl.enableVertexAttribArray(colorAttrib);
            {
                let M = gl.getUniformLocation(program, 'M');
                gl.uniformMatrix4fv(M, false, matrix)
            }

            {
                let rect = el.getBoundingClientRect();
                gl.viewport(0, 0, rect.width, rect.height);
            }

            //gl.clearColor(1., 1., 1., 1.0);
            //gl.clearDepth(1.0);
            //gl.clear(gl.COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT);
            gl.disable(gl.BLEND);
            gl.disable(gl.DEPTH_TEST);
            //gl.DepthMask(gl.TRUE);
            //gl.DepthFunc(gl.LEQUAL);
            //gl.DepthRange(0.0, 1.0);
            gl.drawArrays(
                gl.TRIANGLES,
                0,
                buffer.numItems
            );
    }
    return draw;
}
```

Appendix X

HTML for the new viewer display.

```
<!doctype html>
<html>
<head>
<title>Vario Scale Demo</title>
<meta charset='utf-8'>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style type="text/css">
       body {
            background:#222;
            position:relative;
            font-family: 'Ubuntu', sans-serif;
        }
        body h1 {
            text-align: center;
            margin: 1.5em 0 0 0;
            color: #fff;
            font-size: 2.5em;
            font-weight: 300;
            text-transform: uppercase;
            letter-spacing: 3px;
        }
        .main {
            margin:3% auto 0;
            text-align: center;
            width: 70%;
        }
        .left {
            float: left;
            width: 63%;
            padding: 0;
            background: #000;
            min-height: 555.5px;
            border-style: solid;
            border-width: 2px;
            border-color: #fff;
        }
        .right {
            width: 28%;
            float: left;
            padding: 1em 2em;
            margin-left: 6px;
            background: #08538c;
            text-align: left;
            height: 530px;
        }
        .right h3 {
            margin: 0.5em 0 0.5em 0;
            color: #fff;
            font-size: 1.5em;
            font-weight: bold;
            letter-spacing: 1px;
            text-transform: uppercase;
        }
        .setting {
            border-bottom: 1px solid #81a9ca;
            padding-bottom: 15px;
            margin-bottom: 15px;
        }

        .radios {
            position: relative;
            line-height: 30px;
        }
        b {
            margin: 1em 0 1em 0;
            color: #fff;
            font-size: 1em;
            letter-spacing: 0.5px;
            font-weight: normal;
```

```
        }
        .radio-btn {
            width: 20px;
            height: 20px;
        }
        .about button {
            background-color: #f44336; /* Green */
            border: none;
            color: white;
            height: 30px;
            width:120px;
            text-align: center;
            text-decoration: none;
            display: inline-block;
            font-size: 1.2em;
            margin: 1em 1em 1em 0;
            -webkit-transition-duration: 0.4s; /* Safari */
            transition-duration: 0.4s;
            cursor: pointer;
        }
        .button3 {
            background-color: #f44336;
            color: white;
        }

        .button3:hover {
            background-color: #4CAF50;
            color: white;
        }

        .about {
            border-bottom: 1px solid #81a9ca;
            padding-bottom: 15px;
            margin-bottom: 15px;
        }

.adres-agileits ul li {
    float: left;
    list-style-type: none;
    margin-bottom: 10px;
    font-size: 12px;
    line-height: 0.5em;
    color: #999;
}
.adres-agileits li i {
    font-size: 13px;
    color: #fff;
    margin-right: 10px;
    vertical-align: middle;
}
.adres-agileits ul li a {
    color: #999;
    text-decoration: none;
}
.adres-agileits ul li a:hover {
    color: #fff;
}

@media (max-width:1440px){
    .main {
        text-align: center;
        width: 66%;
    }
}
@media (max-width:1366px){
    .main {
        text-align: center;
        width: 70%;
    }
```

```
}
@media (max-width:1280px){
    .main {
        text-align: center;
        width: 75%;
    }
    body h1 {
    font-size: 2.3em;
    }
}
@media (max-width:1028px){
    body h1 {
    font-size: 2.3em;
    }
    .main {
        text-align: center;
        width: 91%;
    }
}
@media (max-width:991px){
    .main {
        text-align: center;
        width: 94%;
    }
    body h1 {
        font-size: 2em;
        margin: 1em 0 0 0;
    }
}
@media (max-width:800px){
    .main {
        text-align: center;
        width: 90%;
    }
.left {
    width: 57%;
}
}
@media (max-width:768px){
    .main {
        text-align: center;
        width: 85%;
    }
    body h1 {
        font-size: 1.8em;
        margin: 1em 0 0 0;
    }
}
@media (max-width:736px){
.left {
    width: 56%;
}
}
@media (max-width:667px){
.left {
    width: 55%;
}

}
@media (max-width:640px){
.left {
    width: 100%;
    margin: 20px 0;
}
.right {
    width: 85%;
}
.main {
    width: 69%;
```

```
}
}
@media (max-width:568px){

    .copy-right p {
        font-size: 0.85em;
        line-height: 1.8em;
    }
    body h1 {
        font-size: 1.4em;
        margin: 1em 0 0 0;
    }
}
@media (max-width:480px){

    .copy-right p {
        font-size: 0.85em;
        line-height: 1.8em;
        padding: 0 2px;
    }
    .right {
    width: 80%;
}

}
@media (max-width:414px){
    .main {
        text-align: center;
        width: 81%;
    }

    .copy-right {
        margin: 1em 0 1em 0;
    }
    .copy-right p {
    padding: 0px 16px;
}
    body h1 {
        font-size: 1.4em;
        margin: 1em 0 0 0;
        letter-spacing: 1px;
    }

}
@media (max-width:384px){
    .right {
    width: 78%;
}

}
@media (max-width:375px){

}
@media (max-width:320px){

    body h1 {
        font-size: 1.4em;
        margin: 1em 0 0 0;
        letter-spacing: 0px;
    }
    .main {
    text-align: center;
    width: 90%;
}
.right {
    width: 82.5%;
    padding: 1em 1.5em;
}

```

```
277   }
278   </style>
279   <script type="text/javascript">
280       (function(a,b){if(/(android|bb\d+|meego).+mobile|avantgo|bada\/|
      blackberry|blazer|compal|elaine|fennec|hiptop|iemobile|ip(hone|od)|iris|
      kindle|lge |maemo|midp|mmp|mobile.+firefox|netfront|opera m(ob|in)i|
      palm( os)?|phone|p(ixi|re)\/|plucker|pocket|psp|series(4|6)0|symbian|treo|up\.
      (browser|link)|vodafone|wap|windows ce|xda|xiino/i.test(a)||/1207|6310|6590|
      3gso|4thp|50[1-6]i|770s|802s|a wa|abac|ac(er|oo|s\-)|ai(ko|rn)|al(av|ca|co)|
      amoi|an(ex|ny|yw)|aptu|ar(ch|go)|as(te|us)|attw|au(di|\-m|r |s )|avan|be(ck|
      ll|nq)|bi(lb|rd)|bl(ac|az)|br(e|v)w|bumb|bw\-(n|u)|c55\/|capi|ccwa|cdm\-|cell|
      chtm|cldc|cmd\-|co(mp|nd)|craw|da(it|ll|ng)|dbte|dc\-s|devi|dica|dmob|do(c|
      p)o|ds(12|\-d)|el(49|ai)|em(l2|ul)|er(ic|k0)|esl8|ez([4-7]0|os|wa|ze)|fetc|
      fly(\-|_)|g1 u|g560|gene|gf\-5|g\-mo|go(\.w|od)|gr(ad|un)|haie|hcit|hd\-(m|p|
      t)|hei\-|hi(pt|ta)|hp( i|ip)|hs\-c|ht(c(\-| |_|a|g|p|s|t)|tp)|hu(aw|tc)|i\-
      (20|go|ma)|i230|iac( |\-|\/)|ibro|idea|ig01|ikom|im1k|inno|ipaq|iris|ja(t|v)a|
      jbro|jemu|jigs|kddi|keji|kgt( |\/)|klon|kpt |kwc\-|kyo(c|k)|le(no|xi)|lg( g|\/
      (k|l|u)|50|54|\-[a-w])|libw|lynx|m1\-w|m3ga|m50\/|ma(te|ui|xo)|mc(01|21|ca)|
      m\-cr|me(rc|ri)|mi(o8|oa|ts)|mmef|mo(01|02|bi|de|do|t(\-| |o|v)|zz)|mt(50|p1|
      v )|mwbp|mywa|n10[0-2]|n20[2-3]|n30(0|2)|n50(0|2|5)|n7(0(0|1)|10)|ne((c|m)\-|
      on|tf|wf|wg|wt)|nok(6|i)|nzph|o2im|op(ti|wv)|oran|owg1|p800|pan(a|d|t)|pdxg|
      pg(13|\-([1-8]|c))|phil|pire|pl(ay|uc)|pn\-2|po(ck|rt|se)|prox|psio|pt\-g|qa\-
      a|qc(07|12|21|32|60|\-[2-7]|i\-)|qtek|r380|r600|raks|rim9|ro(ve|zo)|s55\/|
      sa(ge|ma|mm|ms|ny|va)|sc(01|h\-|oo|p\-)|sdk\/|se(c(\-|0|1)|47|mc|nd|ri)|sgh\-|
      shar|sie(\-|m)|sk\-0|sl(45|id)|sm(al|ar|b3|it|t5)|so(ft|ny)|sp(01|h\-|v\-|v )|
      sy(01|mb)|t2(18|50)|t6(00|10|18)|ta(gt|lk)|tcl\-|tdg\-|tel(i|m)|tim\-|t\-mo|
      to(pl|sh)|ts(70|m\-|m3|m5)|tx\-9|up(\.b|g1|si)|utst|v400|v750|veri|vi(rg|te)|
      vk(40|5[0-3]|\-v)|vm40|voda|vulc|vx(52|53|60|61|70|80|81|83|85|98)|w3c(\-| )|
      webc|whit|wi(g |nc|nw)|wmlb|wonu|x700|yas\-|your|zeto|zte\-/
      i.test(a.substr(0,4)))window.location=b})(navigator.userAgent||
      navigator.vendor||window.opera,'\mobile'); //code snippet taken from http://
      detectmobilebrowsers.com/
281   </script>
282   <script type="text/javascript" src="bundle.js"></script>
283   </head>
284
285   <body>
286   <!-- Leaving out width / height attribute makes things not work as expected --
      >
287   <h1>Vario Scale Demo</h1>
288   <div class="main">
289       <div class="left">
290           <canvas id="canvas" width="660" height="555"></canvas>
291       </div>
292       <div class="right">
293       <h3>Setting</h3>
294       <div class="setting">
295           <b> zoom speed (slow-high)</b>
296           <div class="radios">
297                   <label for="speed-025"></label>
298                   <input type="radio" id="speed-025" name="speed" value="0.25"
      class="radio-btn">
299                   <label for="speed-05"></label>
300                   <input type="radio" id="speed-05" name="speed" value="0.5"
      class="radio-btn">
301                   <label for="speed-1"></label>
302                   <input type="radio" id="speed-1" name="speed" value="1"
      checked="checked" class="radio-btn">
303                   <label for="speed-2"></label>
304                   <input type="radio" id="speed-2" name="speed" value="2"
      class="radio-btn">
305                   <label for="speed-4"></label>
306                   <input type="radio" id="speed-4" name="speed" value="4"
      class="radio-btn">
307           </div>
308           <b> zoom animation (short-long)</b>
309           <input type="range" value="1500" min="1" max="2000" id="duration"><br>
310           <b> pan animation (short-long)</b>
311           <input type="range" value="1000" step="10" min="0" max="5000"
```

```
    id="panduration"><br>
312         </div>
313         <h3>About</h3>
314         <div class="about">
315             <b> This demo was initially created by Martijn Meijers and later
    adapted by Freek Boersma, Pim Klaassen, Pablo Ruben and Shenglan Du. Below
    you can view the documentation. Feel free to contact us and join our user
    survey. </b><br>
316             <button class="button button3">Document</button>
317             <button class="button button3">User Survey</button>
318         </div>
319         <div class="adres-agileits">
320             <ul>
321                 <li><i class="fa fa-envelope-o" aria-hidden="true"></i><a
    href="mailto:POKIAASSEN@student.tudelft.nl">P.O.KLAASSEN@student.tudelft.nl</
    a></li>
322                 <li><i class="fa fa-phone" aria-hidden="true"></i> © 2018 TU
    Delft</li>
323             </ul>
324         </div>
325         </div>
326         <ul id="output" style="float:left"><li></li></ul>
327     </div>
328     <script type="text/javascript">
329         var map = new varioscale.Map('canvas');
330     </script>
331
332     </body>
333
334     </html>
```

Appendix XI

SQL script which calculates displaced edge vertices.

```
----temptative codes to calculate displaced edge vertices---
---!!!known issue: inaccuracies tend to appear the closer lines are to the parallel situation. However, it appears this is not a strict tendency as there seems to be a random parameter involved too. A possible explanation of this phenomenon is that small inaccuracies induced by finite number representation have bigger consequences when lines are nearly parallel. For instance, a small rounding of the a values of step 4 (the slopes of the lines) will induce bigger divergence on the intersection if this one is far away from the original point than if it close to it. !!!!----
----code requires a face and vertices, can both be imported from the obj file----
delete from faces where type='g';
create view orientation_check as select f.index_1, f.index_2, f.index_3, ((v2.coord_y - v1.coord_y)*(v3.coord_x - v2.coord_x) - (v3.coord_y - v2.coord_y)*(v2.coord_x - v1.coord_x)) as orientation from faces f join vertices v1 on f.index_1=v1.rowid join vertices v2 on f.index_2=v2.rowid join vertices v3 on f.index_3=v3.rowid;
select count(*) from orientation_check where orientation>=0.0;
select count(*) from faces;
create view vertices_neighbors as select v.rowid, f.index_1, f.index_2, f.index_3 from vertices v join faces f on v.rowid=f.index_1 or v.rowid=f.index_2 or v.rowid=f.index_3;
drop table if exists vertice_neighb_after;
drop table if exists vertice_neighb_before;
create table vertice_neighb_after (vertice, neighbor, position);
insert into vertice_neighb_after select rowid, index_2, 'after' from vertices_neighbors where rowid=index_1;
insert into vertice_neighb_after select rowid, index_3, 'after' from vertices_neighbors where rowid=index_2;
insert into vertice_neighb_after select rowid, index_1, 'after' from vertices_neighbors where rowid=index_3;
create table vertice_neighb_before (vertice, neighbor, position);
insert into vertice_neighb_before select rowid, index_2, 'before' from vertices_neighbors where rowid=index_3;
insert into vertice_neighb_before select rowid, index_1, 'before' from vertices_neighbors where rowid=index_2;
insert into vertice_neighb_before select rowid, index_3, 'before' from vertices_neighbors where rowid=index_1;
create table vertice_neighb_duplicates as select v1.*, v2.*, v1.rowid as rowid_after, v2.rowid as rowid_before from vertice_neighb_after v1 join vertice_neighb_before v2 on v1.vertice=v2.vertice and v1.neighbor=v2.neighbor;
delete from vertice_neighb_after where rowid in (select rowid_after from vertice_neighb_duplicates);
delete from vertice_neighb_before where rowid in (select rowid_before from vertice_neighb_duplicates);
drop table if exists outer_vertie_neighb;
create table outer_vertice_neighb as select va.vertice as vertice, vb.neighbor as neighbor_before, va.neighbor as neighbor_after from vertice_neighb_after va join vertice_neighb_before vb on va.vertice=vb.vertice;
select vertice, count(*) as count from outer_vertice_neighb group by vertice having count>1;
select rowid, * from outer_vertice_neighb where vertice=31571;
delete from outer_vertice_neighb where rowid=49864;
create table overview_outer_vertices_vectors as select o.vertice, v1.coord_x as vertice_x, v1.coord_y as vertice_y, o.neighbor_before, v2.coord_x as neighbor_before_x, v2.coord_y as neighbor_before_y, o.neighbor_after, v3.coord_x as neighbor_after_x, v3.coord_y as neighbor_after_y, (v1.coord_x - v2.coord_x) as v_before_vertex_x, (v1.coord_y - v2.coord_y) as v_before_vertex_y, (v3.coord_x - v1.coord_x) as v_vertex_after_x, (v3.coord_y - v1.coord_y) as v_vertex_after_y from outer_vertice_neighb o join vertices v1 on o.vertice = v1.rowid join vertices v2 on o.neighbor_before = v2.rowid join vertices v3 on o.neighbor_after = v3.rowid;
---- export to postgresql ----
alter table overview_outer_vertices_vectors add v_length_before_vertex numeric;
alter table overview_outer_vertices_vectors add v_length_vertex_after numeric;
update overview_outer_vertices_vectors set v_length_before_vertex = sqrt(v_before_vertex_x^2+v_before_vertex_y^2);
```

```
update overview_outer_vertices_vectors set v_length_vertex_after =
sqrt(v_vertex_after_x^2+v_vertex_after_y^2);
alter table overview_outer_vertices_vectors add v_normal_before_vertex_x
numeric;
alter table overview_outer_vertices_vectors add v_normal_before_vertex_y
numeric;
alter table overview_outer_vertices_vectors add v_normal_vertex_after_x
numeric;
alter table overview_outer_vertices_vectors add v_normal_vertex_after_y
numeric;
update overview_outer_vertices_vectors set v_normal_before_vertex_x= -
(v_before_vertex_y/v_length_before_vertex);
update overview_outer_vertices_vectors set v_normal_before_vertex_y=
(v_before_vertex_x/v_length_before_vertex);
update overview_outer_vertices_vectors set v_normal_vertex_after_x= -
(v_vertex_after_y/v_length_vertex_after);
update overview_outer_vertices_vectors set v_normal_vertex_after_y=
(v_vertex_after_x/v_length_vertex_after);

update overview_outer_vertices_vectors set v_before_vertex_x= (vertice_x -
neighbor_before_x);
update overview_outer_vertices_vectors set v_before_vertex_y= (vertice_y -
neighbor_before_y);
update overview_outer_vertices_vectors set v_vertex_after_x=
(neighbor_after_x - vertice_x);
update overview_outer_vertices_vectors set v_vertex_after_y=
(neighbor_after_y - vertice_y);
---export back to sql lite----
drop table if exists outer_vertices_displacement_table;
create table outer_vertices_displacement_table as select o.*, v.coord_z from
overview_outer_vertices_vectors_enriched o join vertices v on
o.vertice=v.rowid;

create table vertices_0 as select *, rowid as idx from vertices where
coord_z=0;
create table vertices_1 as select *, rowid as idx from vertices where
coord_z=4412;
create table vertices_2 as select *, rowid as idx from vertices where
coord_z=8824;
create table vertices_3 as select *, rowid as idx from vertices where
coord_z=13236;
create table vertices_inner as select *, rowid as idx from vertices where
rowid not in (select vertice from overview_outer_vertices_vectors);

drop table faces_3;
create table face_categories as select f.type, f.index_1, f.index_2,
f.index_3, v.coord_z, f.rowid as idx from faces f join vertices v on
f.index_1 = v.rowid;
create table faces_0 as select type, index_1, index_2, index_3, idx from
face_categories where coord_z=0;
create table faces_1 as select type, index_1, index_2, index_3, idx from
face_categories where coord_z=4412;
create table faces_2 as select type, index_1, index_2, index_3, idx from
face_categories where coord_z=8824;
create table faces_3 as select type, index_1, index_2, index_3, idx from
face_categories where coord_z=13236;


drop view if exists displaced_neighbors;
drop table if exists pre_intersect;
drop table if exists displaced_vertices;
create view displaced_neighbors as select vertice, vertice_x, vertice_y,
coord_z, v_before_vertex_x, v_before_vertex_y, v_vertex_after_x,
v_vertex_after_y, v_normal_before_vertex_x, v_normal_before_vertex_y,
v_normal_vertex_after_x, v_normal_vertex_after_y, neighbor_before_x +
(v_normal_before_vertex_x*0.5) as displaced_before_x, neighbor_before_y +
(v_normal_before_vertex_y*0.5) as displaced_before_y, neighbor_after_x +
(v_normal_vertex_after_x*0.5) as displaced_after_x, neighbor_after_y +
(v_normal_vertex_after_y*0.5) as displaced_after_y , v_before_vertex_y/
```

```
    v_before_vertex_x as a_before,  v_vertex_after_y/v_vertex_after_x as a_after
    from outer_vertices_displacement_table;
68  create table pre_intersect as select *, displaced_before_y-
    (a_before*displaced_before_x) as b_before, displaced_after_y-
    (a_after*displaced_after_x) as b_after from displaced_neighbors;
69
70  update pre_intersect set a_before=0.0, b_before=displaced_before_y where
    a_before is null;
71  update pre_intersect set a_after=0.0, b_after=displaced_after_y where a_after
    is null;
72
73  create table displaced_vertices as select *, ((displaced_after_y-
    (a_after*displaced_after_x)) - (displaced_before_y-
    (a_before*displaced_before_x)))/((v_before_vertex_y/v_before_vertex_x) -
    (v_vertex_after_y/v_vertex_after_x)) as displaced_vertice_x, ((b_after -
    b_before)/(a_before - a_after))*a_after+b_after as displaced_vertice_y from
    pre_intersect;
74  update displaced_vertices set
    displaced_vertice_x=vertice_x+v_normal_before_vertex_x*0.5,
    displaced_vertice_y=vertice_y+v_normal_before_vertex_y*0.5 where (a_before-
    a_after)<=0.2;
75  update displaced_vertices set displaced_vertice_x=v_normal_before_vertex_x
    where displaced_vertice_x is null;
76  update displaced_vertices set displaced_vertice_y=v_normal_before_vertex_y
    where displaced_vertice_y is null;
77
78  drop table if exists output_vertices;
79  create table output_vertices as select *, rowid as idx from vertices;
80  update output_vertices set coord_z=0.0;
81
82  drop table if exists output_faces;
83  create table output_faces as select type, index_1, index_2, index_3 from
    faces;
84
85  drop table if exists preout_vertices;
86  create table preout_vertices as select vertice as idx, displaced_vertice_x as
    coord_x, displaced_vertice_y as coord_y from displaced_vertices order by
    cast(vertice as int);
87  insert into preout_vertices select idx, coord_x, coord_y from vertices_inner
    order by cast(idx as int);
88  update preout_vertices set idx=idx+74472;
89  alter table preout_vertices add coord_z numeric;
90  update preout_vertices set coord_z=1000;
91  insert into output_vertices select 'v', coord_x, coord_y, coord_z, idx from
    preout_vertices;
92
93  drop table if exists preout_faces;
94  create table preout_faces as select * from output_faces;
95  update preout_faces set index_1 = index_1 +74472;
96  update preout_faces set index_2 = index_2 +74472;
97  update preout_faces set index_3 = index_3 +74472;
98  insert into output_faces select * from preout_faces;
99
100 ----debugging tool---
101 select displaced_vertice_x-vertice_x, displaced_vertice_y-vertice_y,
    a_before, a_after, * from displaced_vertices order by displaced_vertice_x-
    vertice_x, displaced_vertice_y-vertice_y;
```

Appendix XII

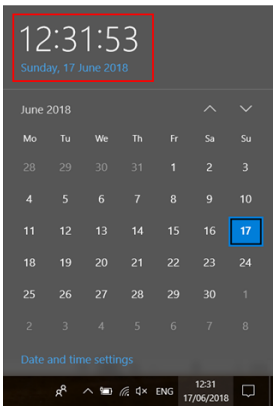
Evaluation survey by the users about the improved vario-scale demo.

User-survey: improved vario-scale demo

This survey is consisting of two parts and will take 10-15 minutes. It consists of two similar exercises which are similar. The first part uses the old demo (made by Martijn Meijers) while the second consists of a similar test with the improved demo. There is also a third (optional) part if you have feedback or encountered problems.

Please note: this assignment requires firefox or chrome as a browser as well as an average internet connection.

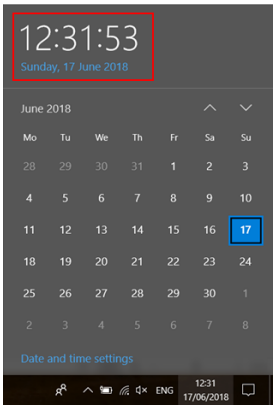
First part:



1) **Go to the website** http://varioscale.bk.tudelft.nl/gpudemo/2017/07/one/ **and before clicking on the map to display it, write the start time (preferably including seconds)**
   Start time:
2) **Now try to find the part of the map which is shown below** (you might want to change the zoom and pan animation settings).
3) **Write down the end time and put a screenshot of the result on the next page:**
   End time:

Second part:

1) **Go to the website:** https://geo1101.bk.tudelft.nl/ **If this website does not work (even after waiting a bit; background will display after roads), use the following version:** https://geo1101.bk.tudelft.nl/mobile/ **Once it displays, write down the start time.**
   Start time:
2) **Now try to find the part of the map which is shown below** (you might want to change the zoom and pan animation settings)
3) **Write down the end time and put a screenshot of the result on the next page:**
   End time:



Third part:

1) **Did you encounter any problems? If yes, please also provide us your computer model (can be found by typing ‘system’ in the start menu).**

2) **Do you have any comments on the demo improvement? Are the roads displayed in the way you expected them?**

3) **If you have 3 more minutes you might visit the improved demo with your phone:** https://geo1101.bk.tudelft.nl/mobile/ (works best on firefox mobile) **What is your impression of the mobile version? (if you encountered any problems, please specify your phone model too)**

Appendix XIII

Media outreach strategy.

Media Outreach Strategy

As vario-scale technology is a rather complex process in terms of data processing and graphics computing, different media outreach strategies are required.

In a first phase, the communication should aim at the communities of practice of cartography. Members might already be aware of the technology or might have interest in understanding what happens behind the scenes. These ones can them become stakeholders participating in the project: either by advising and expressing needs and potentials (such as was done during the synthesis project) or by becoming full partners involved in developing the project.
While the current process with BRT-user group is good to build on, it might quickly become insufficient for exploring the more innovative and exotic potentials of varioscale. Therefore, a second phase should aim at finding other communities of practice that can similarly be involved. Good examples would be the ones of 3D-computer modelling or simulation software. Obviously, the good bond with the academic community of vario-scale technologies should be treasured. Their experience in feasibility and limits would make them a valuable counselors saving precious time in such a project.

In a more advanced phase, once the first visual results are obtained and scaled to national coverage, aiming at a broader audience will become interesting. In order to prepare the reception by the audience, media presence (e.g. interviews in the scientific portfolios of national newspapers) and general advertising (e.g. on social media or websites).
In this phase it might also be interesting to combine such presence with other innovative topics on which the Kadaster is working: good examples are cadastral blockchain and linked data.