Journal of Geovisualization and Spatial Analysis manuscript No.
(will be inserted by the editor)

Generalizing simultaneously to support smooth zooming: Case study of merging area objects

Received: date / Accepted: date

Abstract When users zoom out on a digital map, some area objects become too tiny to be seen, resulting in visual clutter. To avoid this problem, we merge the relatively unimportant areas into their neighbors to form larger areas. However, if two areas are merged into an area in a single step operation, then it is a large and discrete change, which disturbs map users. In order to provide small and smooth changes, this paper merges a pair of areas by expanding one (winner) over the other (loser). At the same time, the loser gradually adapts its color to the winner. However, if the changes happen sequentially, then the changes have to happen very fast because the map user wants to see the map at the target scale rather soon after applying a zooming operation. In order to ensure that every change has more time to take place, we propose to merge multiple areas simultaneously, where a greedy algorithm is used to decide which areas should be merged simultaneously. The merging process is pre-computed and then is recored into a space-scale cube (SSC). When a users browses our web map, the SSC, together with some other necessary information, is sent to the client side so that the map can be generated by slicing the SSC in GPU. We consider that generalizing simultaneously is an important step towards continuous map generalization.

Keywords Space-scale cube · vario-scale map · continuous map generalization · web map · compatibility value

Address(es) of author(s) should be given

1 Introduction

When users are reading a digital map, they expect different levels of detail (LoDs) depending on the scale. For example, they may want to see individual buildings when zooming in and see built-up areas when zooming out. That is why depicting geographical information is dependent on the scale (Müller et al. 1995, Weibel 1997). In order to prepare map data for different scales, a detailed map is generalized to generate coarser data for maps at smaller scales, which is known as map generalization. Mackaness et al. (2016) gave a taxonomy of generalization algorithms, including selection, simplification, aggregation, and so on. Often, a multi-representation database (MRDB) is utilized to store maps at different scales and to send proper data to clients on request (e.g., Hampe et al. 2004). However, large and discrete changes between different map representations may confuse users, so continuous map generalization (CMG) is needed to provide smooth scale transitions. Algorithms of CMG have been proposed to morph raster maps (e.g., Pantazis et al. 2009b,a), to morph polylines (e.g., Nöllenburg et al. 2008, Peng et al. 2013, Deng and Peng 2015, Li et al. 2017a, 2018), to generalize buildings (e.g., Li et al. 2017b, Peng and Touya 2017, Touya and Dumont 2017), to transform road networks or river networks (e.g., Šuba et al. 2016, Chimani et al. 2014, Huang et al. 2017, Peng et al. 2012), and to transit administrative boundaries (e.g., Peng et al. 2016).

Area objects are important features on maps. When users zoom out, some area objects become too tiny to be seen, which results in visual clutter. The clutter can be avoided by generalizing the area objects. The generalization operators include merging (e.g., Haunert and Wolff 2010), aggregating (e.g., Shen et al. 2019), amal-

gamating (e.g., Regnauld and Revell 2007), splitting (e.g., Meijers et al. 2016), and collapsing (e.g., Haunert and Sester 2008). However, if zooming is realized by switching between some levels of map representations, large and discrete changes usually happen. This kind of changes may cause users to lose track of their area objects of interest (van Kreveld 2001). In order to solve this problem, we smoothly and simultaneously generalize the area objects. In our setting, each of the area objects has its *semantic property*, which is also called a *class* (e.g., lake, building, and grassland).

The main contribution of this paper is to generalize simultaneously, which is the first time that the simultaneity is explicitly proposed. Because of the simultaneity, we propose to snap to the scales with completed generalization operations, and we discuss the animation time gained for each generalization operation.

This paper is organized as follows. Section 2 reviews some related work. Our methodology is presented in Section 3. We show a case study in Section 4. Finally, Section 5 draws the conclusion and present our future work.

2 Related work

2.1 Aggregation of area objects

A great deal of research has been devoted to the aggregation of area objects. Both Su et al. (1997) and Sester (2005) used morphological operators (e.g., a dilation followed by an erosion) to aggregate area objects, where the former worked on raster data and the latter worked on vector data. Regnauld (2003) amalgamated area objects by merging, bridging, flooding, or sampling. Shen et al. (2019) aggregated area objects based on the superpixel method of Achanta et al. (2012). Ware et al. (1995) merged pairs of objects based on the constrained Delaunay triangulation, where they introduced operators *append merge*, *direct merge*, and *snap merge* for rectangular objects, as well as *adopt merge* for natural objects. Ai and Zhang (2007) progressively aggregated building clusters, where they found the building clusters based on Delaunay triangulation.

2.2 Merging of area objects

Cheng and Li (2006) proposed three choices of selecting neighboring areas to merge, i.e., the neighbor has the largest size, shares the longest boundary with the least important area, or has the closest class to the least important area. Thiemann and Sester (2018) proposed a chain of operators to generalize a land-cover map. In the chain of processing area objects, they integrated cleaning, dissolving, splitting, aggregating, reclassifying, and simplifying. Haunert and Wolff (2010), Oehrlein and Haunert (2017) employed integer linear programs to merge area objects in order to generate a map at a certain scale.

2.3 Gradual merging of area objects

To provide scale transition of small changes, van Oosterom (1995) proposed the Generalized Area Partitioning (GAP) tree, where in each step the least important area is merged into its most compatible neighbor. Because that method works only on direct neighbors, Ai and van Oosterom (2002) made an extension to GAP-tree, where they connected two close neighboring areas by filling the gap identified by the constrained Delaunay triangulation. Peng et al. (2020) tried to find an optimal sequence to merge area objects based on the A* algorithm or an integer linear program. A comparison including a greedy algorithm showed that the A* algorithm outperforms the two other methods in the aspects of minimizing the class changes and maximizing the area compactnesses. Šuba et al. (2016) continuously generalized a planar map of road network. In each step, they process the least important area object. Taking into account the local condition of the area object (e.g., no compatible neighbor at the same side of the road), they may take different decisions for the area object: Putting it back with a higher importance, collapsing it, or merging it into an adjacent face, while at the same time including the boundaries (the road objects) in these decisions.

Van Oosterom and Meijers (2014) provided real smooth changes of zooming. they developed the concept of the space-scale cube (SSC). The bottom of the SSC is a detailed topographic map, then all the area objects extrude along the z-axis. In the SSC, an area on the map becomes a polyhedron, and the common boundary of two areas is a vertical wall. Whenever a generalization operation happens, the extrusions of the involved areas stop; then, the newly generated areas take the place and start to extrude. On this basis, the map at any scale can be generated by slicing the SSC with a horizontal plane at a corresponding z-coordinate. That is to say, the scale becomes the third dimension of the map in the SSC (e.g., Figure 4). Furthermore, they represented the smooth tGAP in the SSC. A typical example of the smooth generalization operation is that an area merges with another one by gradually expanding over the latter. In the SSC of the smooth tGAP, the wall starts to tilt when the expansion begins. Based on the SSC,

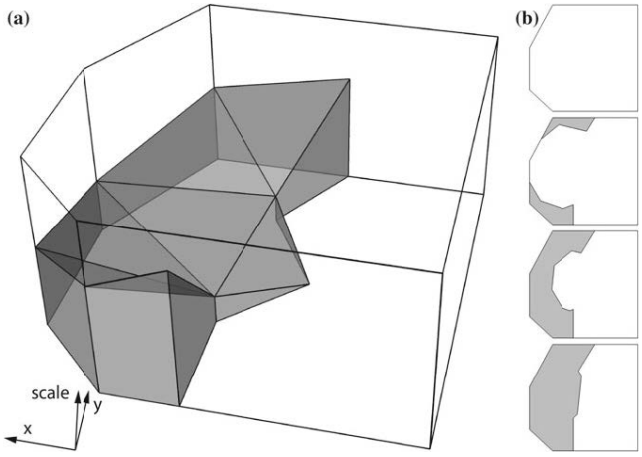


Fig. 1: The principle of the Eater; taken from Šuba et al. (2014).

Meijers et al. (2020) explained the principles of implementing a web map of area objects. They showed how to request only a part of a large dataset of a vario-scale map. They made chunks of the SSC data so that they were able to send only the chunks relevant to users' interested place. They showed how to efficiently slice the SSC to output a web map at a given scale using the GPU at the client side called by WebGL in the browser. In addition to slicing the SSC with a horizontal plane, they could also slice the SSC with a curly surface to have a locally more detailed map or with a tilted surface to have a perspective view. To build an SSC, Šuba et al. (2014) proposed three methods to merge a pair of areas in a gradual manner, which are the *Single flat plane*, the *Zipper*, and the *Eater*. Basically, the *winner* area gradually expands over the *loser* area. We will use the *Eater* because it works for all kinds of polygons, while the other two methods have limitations for some special cases. For example, the two other methods do not work for certain concave polygons. The principle of the Eater is shown in Figure 1, where the interior of the loser is contructed as a triangulation (see Figure 1a), and the eating process is realized by moving the boundary through the triangles (see Figure 1b).

2.4 Merging considering semantic properties

As mentioned in Section 1, each of the area objects has its *semantic property* (i.e., class). When we merge an area into another area, the semantic property of the former is changed to that of the latter. It is important not to cause too much change for two reasons. First, the generalized map should resemble the base map. Second, big changes cause users to lose track of their interested areas.

Van Oosterom and Schenkelaars (1995)'s greedy algorithm repeatedly merges the least important area with one of its neighbors. When choosing the neighbor, the algorithm considers the compatibility between the least important area and the neighbor, where the compatibility can be defined based on the semantic property. Haunert and Wolff (2010) defined distances between semantic properties and included those distances into the cost function of their integer linear program. Peng et al. (2020) defined the semantic distance based on a tree of classes, which guarantees that the distance is a metric (see Figure 2).

van Smaalen (2003, Section 4.4.3) mentioned the class-driven generalization, where if the two classes of two area objects are under the same super class, then the two area objects should be merged, and the new area object uses the super class. Van Smaalen (2003, Section 4.5) suggested that the merging operation should also consider classes that co-occur spatially. He proposed the *class adjacency index* to measure if two classes are often adjacent; if so, the two objects, with the two classes, should be aggregated, and a composite class should be used.

2.5 Simultaneous generalization operators in CMG

Many methods of CMG naturally apply multiple generalization operators simultaneously. In morphing polylines, the points of the polylines are moved simultaneously (e.g., Nöllenburg et al. 2008, Li et al. 2017a). Li et al. (2017b) simultaneously generalized individual buildings. In those methods, the polylines and the buildings were generalized simultaneously and independently. Peng and Touya (2017) and Touya and Dumont (2017) generalized buildings to built-up areas. However, there is no simple relationship between their intermediate-scale maps and their source maps. Therefore, all the intermediate-scale maps of buildings have to be sent from the server to the clients, which is network intensive.

3 Methodology

In order to provide smooth merging so that map users can easily keep track of their interested areas, we merge by gradually expanding an area over another area (see Figure 3q).[1] This expansion can be realized by slicing the space-scale cube (SSC) of Figure 4a. For example,

[1] See the web maps at https://congengis.github.io/webmaps/2021/10/merge/. A comparison of the single merging and the simultaneous merging can be found at https://congengis.github.io/webmaps/2021/10/merge/eg-7-comparer-

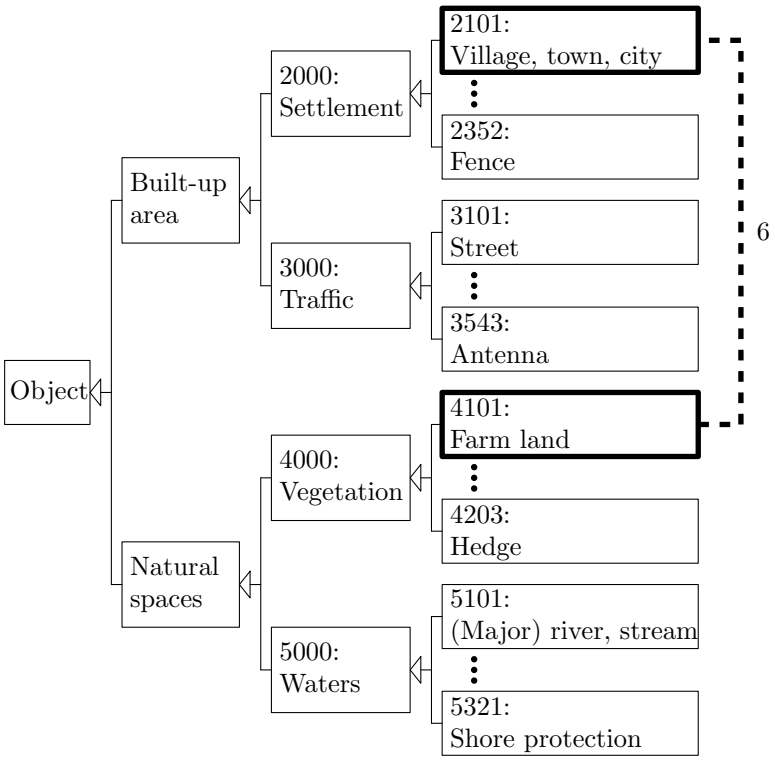


Fig. 2: A way of defining the distance of classes; taken from Peng et al. (2020).

Figure 3q is obtained by slicing Figure 4a at $z = 250$. Smooth animations of zooming out are obtained by slicing an SSC from bottom to top. The details of slicing an SSC are illustrated in Meijers et al. (2020). The SSCs of Figure 4 were built based on the *Eater* of Šuba et al. (2014). The content of an SSC is stored in an OBJ file, and the OBJ file can be visualized by software ParaView (see Figure 4). In Figure 4, the z-coordinates are 100 times of the state values in Figure 3. We performed this multiplication for illustrative purpose only, so that the contents of the SSC in the figures can be better observed. Note that the merging operation is applied in a pre-processing step, before the user zooming in and out of the map and is thus independent of users' area objects of interest.

We define an *event* as a single generalization operation, such as merging an area into a neighbor. Two areas are neighbors if they share a common boundary with length larger than 0 (sharing a point does not make the two areas neighbors). For example, Figure 3e is obtained from Figure 3d by processing one merging event. Similarly, Figure 3l is obtained from Figure 3k by processing two merging events. We define a *step* as a set of events happening at the same animation duration. In our method, a step is completely processed before the next step takes place (all sequential). We define a *state* as the point when a step starts or finishes. For example, there are seven states in the merging sequence of Figures 3d–j and five states in the merging sequence of Figures 3k–o (i.e., states 0, 2, 4, 5, and 6). Note that the value of a state is also the total number of events processed so far.

There are two benefits of merging simultaneously. First, the simultaneity avoids unnatural zooming. Without the simultaneity, sequentially processing generalization operations may result in no change at some locations in a zooming duration, which is unnatural (van Oosterom and Meijers 2014). Therefore, van Oosterom and Meijers (2014) suggested processing the generalization operations simultaneously, but no implementation, testing, or assessment of the idea was provided. Second, the simultaneity brings smoother zooming. Although the SSC allows to deliver a map at any scale, only 16 frames by default will be created to display during one second. When showing an animation zooming, we set 16 as the default value of frames per second (FPS). This value is adequate to provide the visual continuity (Read and Meyer 2000, p. 24). If the merging operations happen sequentially instead of simultaneously, it is more likely that the gap between two frames is larger than the gap between two states. Then there is no anima-

`overlay-single-simultaneous.html`, where the swiper can be moved to see the differences of the two maps at scale, for example, 1 : 11,832.

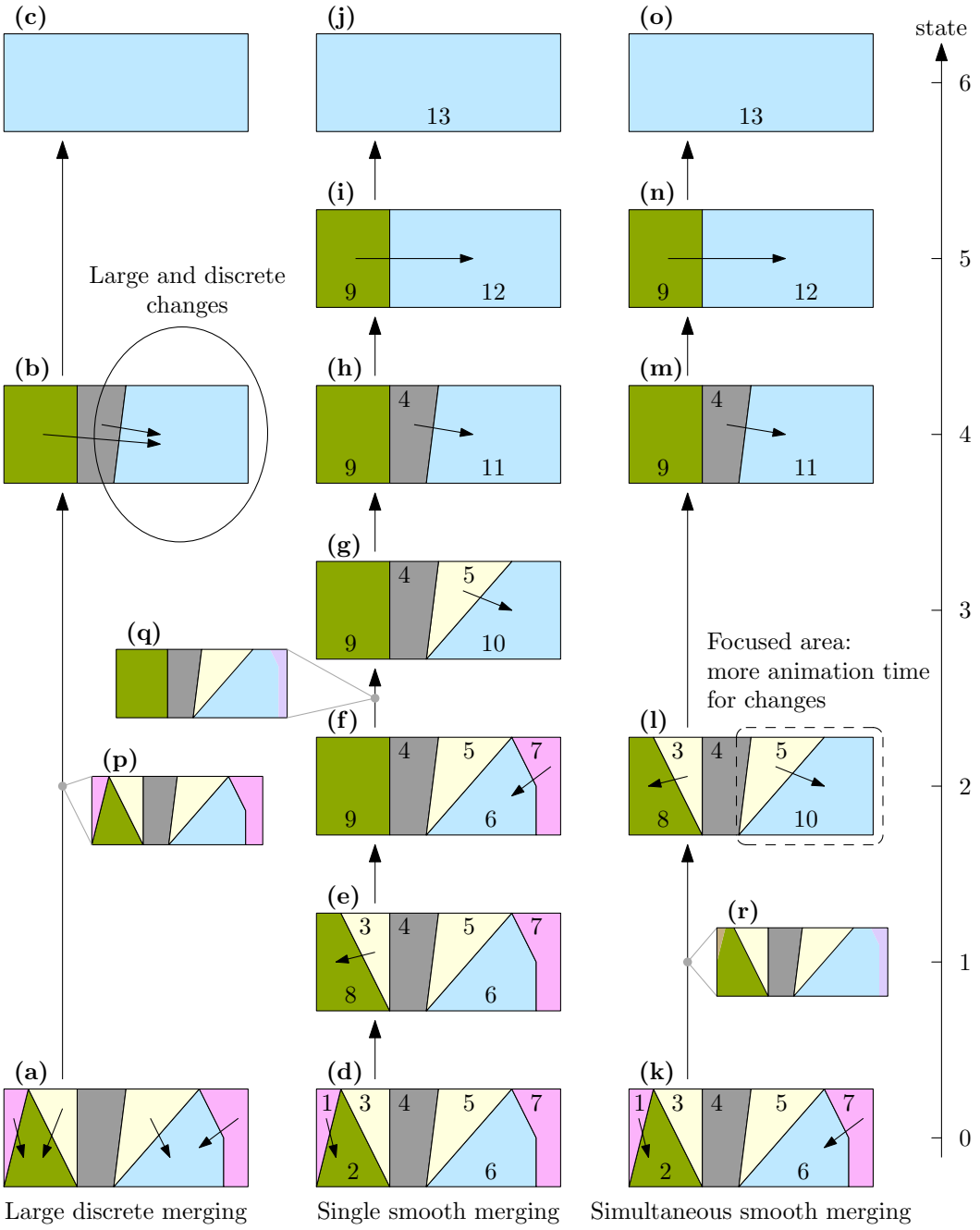
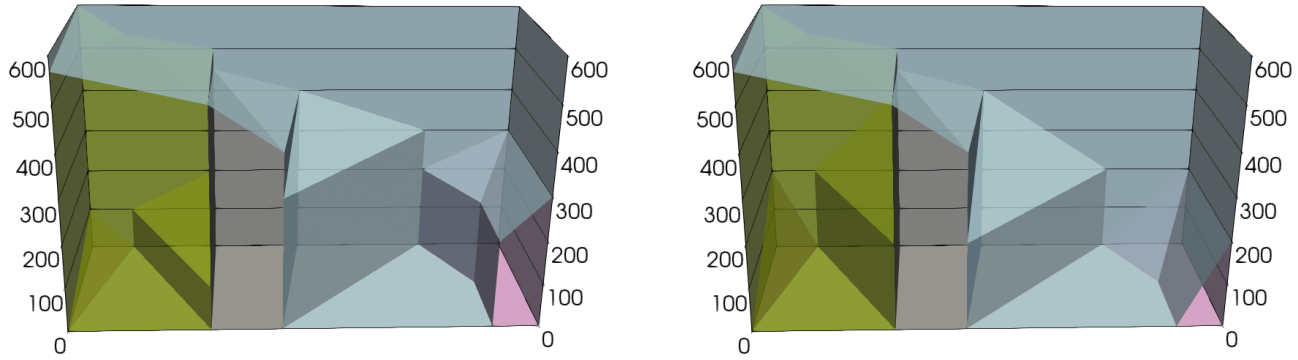


Fig. 3: A comparison of different scale-transition strategies. Each arrow inside the subfigures indicates a merging operation. The arrow in the right-hand side indicates the states of zooming out. (a–c): All changes are processed in one go. (d–j): All changes are sequenced one by one rapidly. (k–o): Changes are grouped, resulting in more animation duration for every change. The numbers are the face IDs. Note that the colors of the smaller areas adapt to the colors of the larger areas during merging.

tion of smooth merging shown at all. For example, if the consecutive frames are Figures 3d, 3e, and 3f, then users can only see discrete merging. In contrast, if the consecutive frames are Figures 3k, 3r, and 3l, then users can really see an ongoing expansion. The more merging operations we process simultaneously, the smoother the expansion process seems.

When merging simultaneously, we require that the area objects involved in different merging events of the same step must not be neighbors, which makes the



(a) The SSC of the single merging of Figures 3d–j.

(b) The SSC of the simultaneous merging of Figures 3k–o

Fig. 4: In the left SSC, only one merging event is happening at a specific state (z-dimension), while in the right SSC multiple merging events may happen at the same state.

merging events independent from each other. In this way, it is easy for us to maintain the topology of the map. In order to realize the requirement, we block the neighbors of the areas once we have found an event. We show a greedy algorithm to find the simultaneous merging events for each step in Section 3.1. Then, we integrate the events into the tGAP database tables (Section 3.2), followed by integrating the events into the SSC (Section 3.3). In Section 3.4, we show how to snap the zooming to valid states to avoid half-way merging animation as stopping halfway will result in showing slivers at a static state. In Section 3.5, we define the animation duration of zooming from one state to another state.

3.1 A greedy algorithm

For each merging event, our greedy algorithm tries to merge the least important area into its most compatible neighbor. Peng et al. (2020) proposed that the most compatible neighbor should have a close class to the least important area and the combination of the two areas should be compact; they defined the class distance based on a binary tree according to the codes of the classes. We define the importance and the compatibility according to van Putten and van Oosterom (1998). That is, the importance of an area is the multiplication of its size and its class weight. Currently, all the class weights are set to 1, which leads to that the smallest area is the least important. The compatibility value between a pair of areas is the multiplication of the common boundary's length and the class similarity of the two areas. Appendix A shows our implementation of computing the weight values and the class similarities.

Figure 5 shows the flowchart of our greedy algorithm. The process starts with state $s = 0$ and a detailed map of area objects, $|M_0|$. Parameter r_{simul} specifies the proportion (i.e., percentage, when multiplied by 100) of area objects that we expect to merge simultaneously. As a value of percentage, r_{simul} is in the range from 0% to 100%, which means $r_{\text{simul}} \in [0, 1]$. Expression $|M_s|$ denotes the number of area objects of the map at state s. If there is more than one area ($|M_s| > 1$), then we start finding merging events. We first compute the number of areas that we expect to merge by

$$n_{\text{target}} = \lceil r_{\text{simul}} \cdot |M_s| \rceil, \tag{1}$$

where the ceiling function guarantees $n_{\text{target}} \geq 1$. That is to say, we find at least one event for each step. When $n_{\text{target}} > 1$, however, we cannot always find n_{target} events because some areas may be blocked as explained before (also see Figure 6). Therefore, we use variable n_{event} to represent the number of events that actually happened within the step.

If we have not found n_{target} events ($n_{\text{event}} < n_{\text{target}}$) and there are still free areas, then we go on looking for merging events. We select the least important area a_{least} from the free areas. An area is *free* if it is not involved in an event and is not blocked. We also find a_{least}'s most compatible neighbor a_{nbr}. If area a_{nbr} is also free, we define an event of areas a_{least} and a_{nbr}. Then, we increase the number of events, n_{event}, by 1. We block the surrounding neighbors of a_{least} and a_{nbr} (see Figure 6a). Note that if the area shares only a vertex with the least important area, it is not blocked. If area a_{nbr} is not free, then it must be blocked because of the previously found events. In this case, we block a_{least} for now so that areas a_{least} and a_{nbr} may merge in the next step. Then, we continue to find more merging events and to block more areas (see Figure 6b).

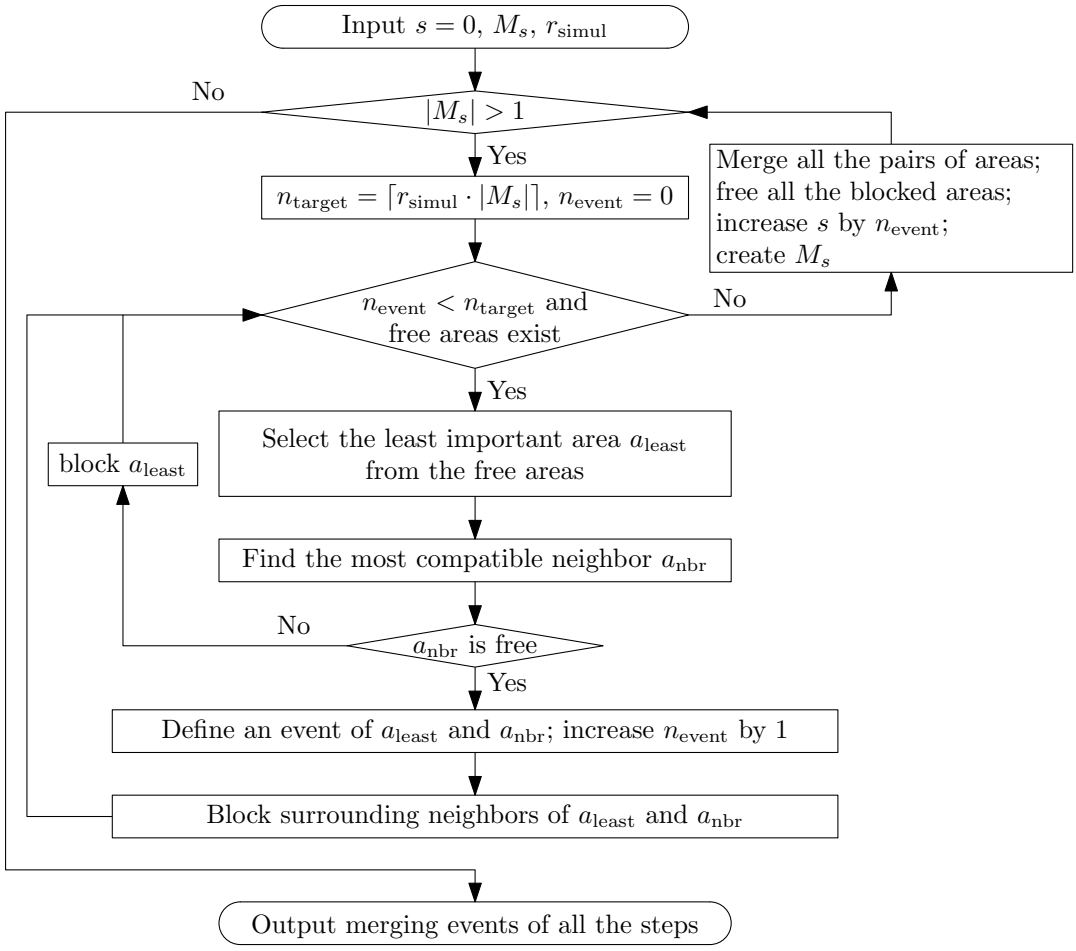


Fig. 5: The flowchart of our greedy algorithm.

Fig. 6: The process of finding simultaneous merging events for a step, where simultaneous parameter $r_{\text{simul}} = 0.3$. (a) From all the free areas, the least important one is selected to merge into its most compatible neighbor. Then the surrounding areas are blocked (marked by the crosses). (b) Next, the least important area from the remaining free areas is selected to merge with the most compatible neighbor, and the surrounding areas are also blocked.

If we have found n_{target} events or there is no free area anymore, then finding merging events of the step finishes. We simultaneously merge all the pairs of areas of the found events to generate new areas, free all the blocked areas, increase state s by value n_{event}, and create map M_s based on the new areas and the freed areas. Then, finding merging events for the next step starts. This iteration of finding completes when there is only one area left on the map ($|M_s| = 1$). The merging events will be stored as records in tGAP database tables (see Figure 7). Figures 3k–o show a sequence of four merging steps obtained by our greedy algorithm, where simultaneous parameter r_{simul} is set to 0.3 (Note that this is an extremely high value, used here to explain the principle in an artificial simple example).

3.2 Integrating the simultaneous events into the tGAP database tables

Meijers (2011b, p. 159) designed three tables to record the information of faces, edges, and face hierarchies, which together form a tGAP. For example, the face table contains columns *face_id*, *imp_low*, *imp_high*, *imp_own*, *feature_class*, *area*, and *mbr_geometry*. We add columns

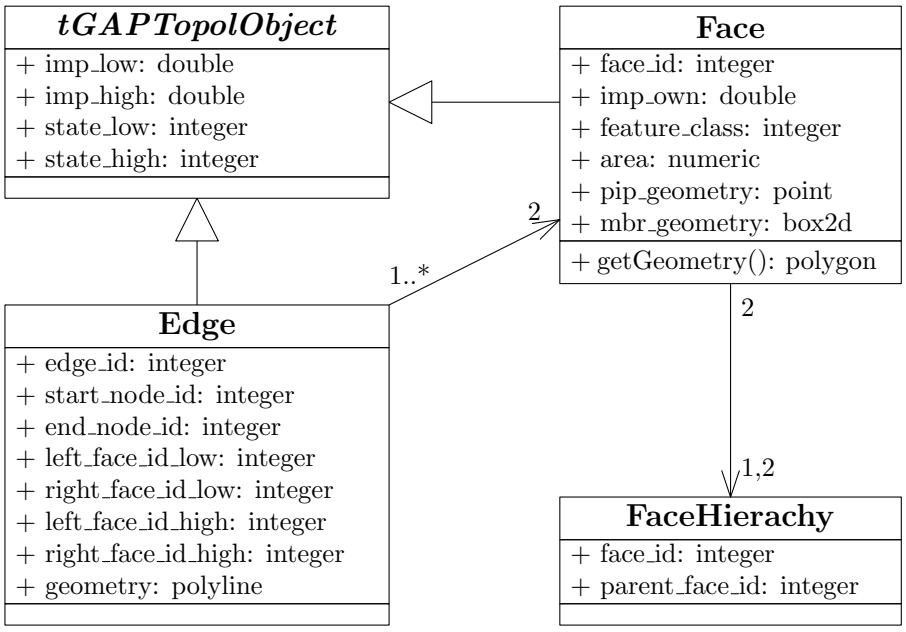


Fig. 7: The Unified Modeling Language (UML) diagram of the classes stored in tGAP database tables. This diagram is a slightly improved version of Meijers (2011b, p. 159). In the face table, property *pip_geometry* stores a point (usually the center) in the face (polygon). The geometry of a face can be obtained by calling function *getGeometry()*. We do not store the face geometry because we want to avoid redundancy, as the edges already stores the sequences of points.

state_low (s_{low}) and *state_high* (s_{high}) into the table so that it is easy to see when a face should appear or disappear (see Tables 1a and 1b, where all other columns, except face_id, are hidden). For zooming out, a face should appear, from the merging of two faces, when the slicing arrives at the low state. When the slicing arrives at the high state, the face should have been merged with another area. Comparing between the tables of single merging (Figures 3d–j) and simultaneous merging (Figures 3k–o), we observe some differences of the values. For example, the s_{high} values of faces 1 and 2 are changed from 1 to 2 (see Table 1). Also, the s_{low} value of face 8 is changed from 1 to 2 (see Table 1). Note that the face ids are defined in Figure 3. Similarly to the face tables, the columns and records of both the edge table and the face-hierarchy table will be changed accordingly.

3.3 Integrating the simultaneous events into the SSC

Recall that we merge a pair of areas by expanding over the less important area from the neighbor. The Eater of Šuba et al. (2014) is used to triangulate the less important area and to tilt the triangles. Then the tilted triangles are integrated into the SSC (see Figure 4) so that we can slice the SSC to achieve smooth merging. For the case of single merging, if a pair of areas have state-high value s_{high}, then the merging animation always starts at state $s_{\text{merge}} = s_{\text{high}} - 1$ (see Table 1a). The less important area completely disappears at state s_{high}. In the face table, a row will be added to record the new area, and its s_{low} value will be the previous s_{high} value. The new area takes the combined place of the pair of areas. Take Figure 3 as an example, area 1 is merged into area 2 (Figure 3d), and area 8 is generated to take the combined place (Figure 3e). The tilted triangle is the one spans from $z = 0$ to $z = 100$ (i.e., from state 0 to state 1) in Figure 4a. In Table 1a, the s_{low} value of area 8 is 1, which is the s_{high} values of areas 1 and 2.

For the case of simultaneous merging, if a step consists of n_{event} events and the step finishes at state s_{high}, then the step starts at state $s_{\text{merge}} = s_{\text{high}} - n_{\text{event}}$. The reason is that if the n_{event} events would happen sequentially (i.e., single merging), then they would take place from state $s_{\text{high}} - n_{\text{event}}$ to state s_{high}. When all the events take place simultaneously in the same step, each of the events can share its merging duration. As a result, each of the simultaneous events has more time to take place than the events would happen sequentially. In other words, for a merging step, each of the events has more time to take place if there are more simultaneous events.

Table 1: Some columns of the face tables. Columns s_{low}, s_{merge}, and s_{high} show the states when the faces appear, when the faces start to disappear, and when the faces completely disappear. In table (b), the different values from table (a) are underlined. Column s_{merge} is not really stored in the database. We show the column so that it is easy to see the differences between the s_{low} values and the s_{merge} values.

(a) The face table of the single merging shown in Figures 3d–j.

f_{id}	s_{low}	s_{merge}	s_{high}
1	0	0	1
2	0	0	1
3	0	1	2
4	0	4	5
5	0	3	4
6	0	2	3
7	0	2	3
8	1	1	2
9	2	5	6
10	3	3	4
11	4	4	5
12	5	5	6
13	6	—	—

(b) The face table of the simultaneous merging shown in Figures 3k–o.

f_{id}	s_{low}	s_{merge}	s_{high}
1	0	0	<u>2</u>
2	0	0	<u>2</u>
3	0	<u>2</u>	<u>4</u>
4	0	4	5
5	0	<u>2</u>	4
6	0	<u>0</u>	<u>2</u>
7	0	<u>0</u>	<u>2</u>
8	<u>2</u>	<u>2</u>	<u>4</u>
9	2	5	<u>6</u>
10	<u>4</u>	<u>2</u>	<u>4</u>
11	4	4	5
12	5	5	6
13	6	—	—

In order to build the SSC for simultaneous merging, we need the s_{merge} value for each of the merging steps so that we know from which state the triangles of the less important areas should be tilted. A simple way is to add a column, say, s_{merge} into the face table during generating the tGAP, as done in Table 1. Then, the states of starting merging can be recorded into the column. However, we would like to avoid unnecessary columns to save storage. Therefore, we compute s_{merge} values based on the s_{high} values on the fly when building the SSC. As an event involves two areas, the number of events finishing at state s_{high} can be calculated by:

$$n_{\text{event}}(s_{\text{high}}) = \frac{\sum_{s \in S_{\text{high}}} [s = s_{\text{high}}]}{2} \tag{2}$$

where notation S_{high} denotes the set of values recorded in column s_{high} of the face table (e.g., Table 1b). Expression $[s = s_{\text{high}}]$ returns 1 if the two values are equal and returns 0 otherwise. As illustrated before, the state at which the simultaneous merging starts can be computed by:

$$s_{\text{merge}}(s_{\text{high}}) = s_{\text{high}} - n_{\text{event}}(s_{\text{high}}) \tag{3}$$

Take the case of Table 1b for example, we have $S_{\text{high}} = \{2, 2, 4, 5, 4, 2, 2, 4, 6, 4, 5, 6\}$, $n_{\text{event}}(4) = 2$, and $s_{\text{merge}}(4) = 2$. Therefore, there are two merging events finishing at state 4, i.e., event of merging area 3 into area 8 and event of merging area 5 into area 10 (also see Figures 3l and 3m). The merging animation takes place from state 2 to state 4. This merging can be also observed from the two tilted triangles spanning from $z = 200$ to $z = 400$ in Figure 4b. In merging sequence of Figures 3d–j, the animation of merging area 3 into area 8 takes place from state 1 to state 2 (also see the tilted triangle spanning from $z = 100$ to $z = 200$ in Figure 4a), and the animation of merging area 5 into area 10 takes place from state 3 to state 4 (also see the tilted triangle spanning from $z = 300$ to $z = 400$ in Figure 4a). As a result, the animation duration of merging area 3 into area 8 of sequence Figures 3k–o is almost twice as that of sequence Figures 3d–j. We say *almost* because the animation duration is also dependent on the current state of the map (see Section 3.5).

3.4 Snapping to a valid state

For a zooming action based on the SSC, we always snap the map to a valid state. In this way, users will not see a merging operation stops half-way and will not see transition artifact (such as slivers). Take the sequence of Figure 3k–o for example, the merging animation should stop at either 3k or 3l, but not at 3r. Note that some states are not valid because of the simultaneous events. For example, state 1 is not valid for the sequence of Figure 3k–o. In that example, the list of valid states is $S_{\text{valid}} = [0, 2, 4, 5, 6]$. In order to snap to one of the valid states after a zooming operation, we have to communicate them to the client side. There are multiple options. The most simple one assumes that, during the creation of the simultaneous SSC, we can always perform the n_{target} number of events in all steps. In that case, we just have to communicate the number of areas and the ratio r_{simul}. In case of high value ratios

(e.g., $r_{\text{simul}} > 0.01$), this assumption may be incorrect. We then have to communicate the valid states by sending them explicitly. Because this list may get rather large, we only send exceptions. Appendix B shows the details of the technique. As a result, the list of valid states S_{valid} is sent to the client side.

According to how much a user has zoomed, the target scale, say, $1 : S_{\text{t}}$ can be computed. Huang et al. (2016) suggested that the average density of the original map should be preserved for a smaller-scale map. Their suggestion is based on the assumption that the area density of the base map is well designed, which is reasonable. We use variable A_{real} to denote the total areal size of all the area objects in reality. Then, the size on screen at scale $1 : S_{\text{t}}$ is $A_{\text{real}}/S_{\text{t}}^2$. In order to keep the density, we require

$$\frac{N_{\text{b}}}{A_{\text{real}}/S_{\text{b}}^2} = \frac{N_{\text{b}} - E_{\text{t}}}{A_{\text{real}}/S_{\text{t}}^2} \tag{4}$$

where parameter $N_{\text{b}} = |M_0|$ is the number of areas on the base map, parameter S_{b} is the scale denominator of the base map, and variable E_{t} is the total number of events happening from the base map to the map at scale $1 : S_{\text{t}}$ (in this case, an event is that an area is merged into another one). Equation 4 yields:

$$E_{\text{t}} = N_{\text{b}} \left(1 - \frac{S_{\text{b}}^2}{S_{\text{t}}^2}\right) \tag{5}$$

In our example regarding to list of valid states S_{valid}, if event number $E_{\text{t}} \leq 0$, the base map should be presented; if $E_{\text{t}} \geq 6$ (i.e., the last value of list L_{event}), the map with the final single area should be presented. Otherwise, if $0 < E_{\text{t}} < 6$, we snap event number E_{t} to the closest value (measured in events) of list S_{valid}, which is denoted by $E_{\text{t,snap}}$. The scale denominator corresponding to event number $E_{\text{t,snap}}$ can be computed by

$$S_{\text{t,snap}} = S_{\text{b}} \sqrt{\frac{N_{\text{b}}}{N_{\text{b}} - E_{\text{t,snap}}}} \tag{6}$$

where this equation is an inverse function of Equation 5. At the end of the zooming action, the map will snap to state $s_{\text{t,snap}}$ at scale $1 : S_{\text{t,snap}}$. Note that state $s_{\text{t,snap}}$ always has the same value as event number $E_{\text{t,snap}}$.

3.5 Animation duration of a step

When users are zooming from a scale to another scale, some steps take place to change the state of the map accordingly. We define the *zooming duration* as the amount of animation time that the map reacts to one "rolling click" of the mouse wheel. The zooming duration often is the sum of the *animation durations* of several merging steps. The animation duration of each event depends on the number of events between the two states, the *zooming factor* of the scale, and the *zooming duration*. On the one hand, the animation duration should not be too short as then the animation will be too fast. On the other hand, if the animation takes too long, the map will not be interactive, and users will be "frustrated". Meijers et al. (2020, Section 4.3) have introduced the zooming factor and the zooming duration. They allowed users to set the two parameters, which is also the case in our paper (see Figure 8). This section formalizes the relationship of the animation duration, the zooming duration, the zooming factor, and the number of events. In a zooming duration, there can be many merging steps, no matter single merging or simultaneous merging. The formalization is based on the setting that a zooming duration is divided equally by its merging steps (Šuba (2017, Section 6.7) showed some other possible settings). In other words, the steps happen sequentially and take the same amount of animation duration. Note that the steps from different zooming durations may have different animation durations.

Let N_{event} be the number of events happening in a zooming duration. Let n_{step} be the number of steps happening in the zooming duration. Let t_{single} be the animation duration of each of the steps, where each step consists of only one event. Let t_{simul} be the animation duration of each of the steps, where each step consists of at least one event. Then, we have

$$t_{\text{simul}} = t_{\text{single}} \frac{N_{\text{event}}}{n_{\text{step}}} \tag{7}$$

As N_{event} is larger than or equal to n_{step}, we have $t_{\text{simul}} \geq t_{\text{single}}$. That is to say, when processing the merging events simultaneously, each step has more time to take place. The derivation of Equation 7 is shown in Appendix C.

4 Case study

We have stored the result of the tGAP as a set of tables (see Section 3.2) in a PostgreSQL database. We have employed the Eater of Šuba et al. (2014), implemented in Python, to generate the elements (vertices, triangulated faces, and boundaries) of the SSC (van Oosterom et al. 2014) and saved these elements in an OBJ file.[2] When a user visits our website to access the map, some data will be sent to the client side. On the client side,

[2] Wavefront .obj file: https://en.wikipedia.org/wiki/Wavefront_.obj_file, accessed: January 14, 2020.

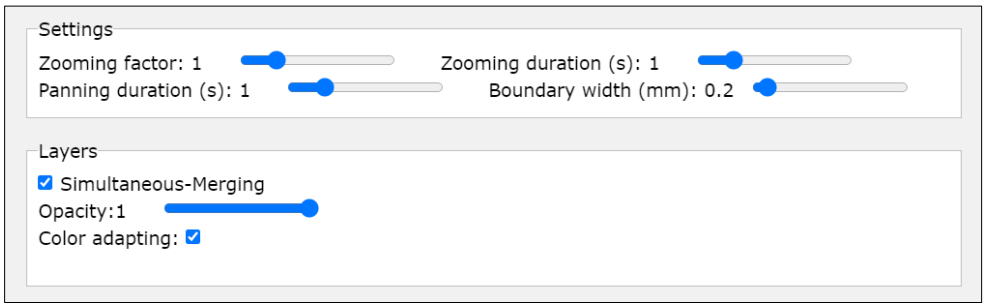


Fig. 8: Our panel of settings. Among others, one can set how much to zoom when scrolling the mouse wheel and set the zooming duration.

the received data will be processed by a map viewer implemented in JavaScript. The processed data and some code based on WebGL (Web Graphics Library) are submitted to GPU so that we can output the interactive map with smooth zooming by slicing the SSC.

Figure 9 shows the topographic map of this case study.[3] The class codes and the rendering formulas are provided by the Dutch Kadaster.[4] Because the base scale is 1 : 10,000, we have $S_b = 10{,}000$ for Equation 6. The maximum value of event number E_{snap} is 13,237 as there are in total 13,238 areas. When we zoom out far enough so that E_{snap} reaches its maximum value, the scale denominator will arrive at 1,150,565 according to Equation 6. At that moment, all the areas are merged into one single area. In each step, we want to simultaneously merge some proportion of the areas. We tried three cases: 0.1%, 1%, and 10%. That is, simultaneous parameter $r_{simul} = 0.001, 0.01,$ and 0.1 (see Section 3.1), which are independent of the size of the map dataset. The three versions of the map can be browsed online.[5] Figure 10 shows two examples of our web map when simultaneous parameter $r_{simul} = 0.01$.

Some statistics of the results when simultaneous parameter $r_{simul} = 0.001$, 0.01, or 0.1 are shown in Table 2. According to column N_{step}, the number of steps decreases when the simultaneous parameter increases. This is reasonable because more areas will be merged in each step. As explained in Section 3.1, for each merging step we iteratively select the least important area and its most compatible neighbor to define a merging event; then, we block the neighbors of the two areas. Sometimes, a least important area is already blocked because of the previously found events. This situation happens 2,714 times in total for all the steps when simultaneous parameter $r_{simul} = 0.01$ (see column $N_{blocked}$ of Table 2). Sometimes, although a least important area is free, its most compatible neighbor has been blocked because of the previously found events. This case happens 1,383 times in total for all the steps when simultaneous parameter $r_{simul} = 0.01$ (see column $n_{nbr\_blocked}$ of Table 2). According to the statistics, we encounter more cases of the areas blocked when merging a larger proportion of the area objects. However, we can still reach our target number of events perfectly for settings of $r_{simul} = 0.01$ or $r_{simul} = 0.001$. Only when we push beyond the limit (e.g., $r_{simul} = 0.1$), we can not reach the target number of events in a step (and we need correction information to compute the actual number of achieved events). When the target number of events cannot be met, one could also question the cartographic quality if there is hardly any free choice when generalizing.

Table 2: Some statistics when different simultaneous parameters area used (i.e., $r_{simul} = 0.001$, 0.01, and 0.1). Column N_{step} records the number of steps to transit from the base map to the map with a single area. Column $N_{blocked}$ records the number of times when the least important area was blocked. Column $N_{nbr\_blocked}$ records the number of times when the most compatible neighbor was blocked.

r_{simul}	N_{step}	$N_{blocked}$	$N_{nbr\_blocked}$
0.001	3,195	211	72
0.01	544	2,714	1,383
0.1	91	100,617	34,268

As we can find the target numbers of merging events for all the steps when simultaneous parameter $r_{simul} = 0.001$ or 0.01, the corresponding exceptions lists are empty. When $r_{simul} = 0.1$, the exception list is

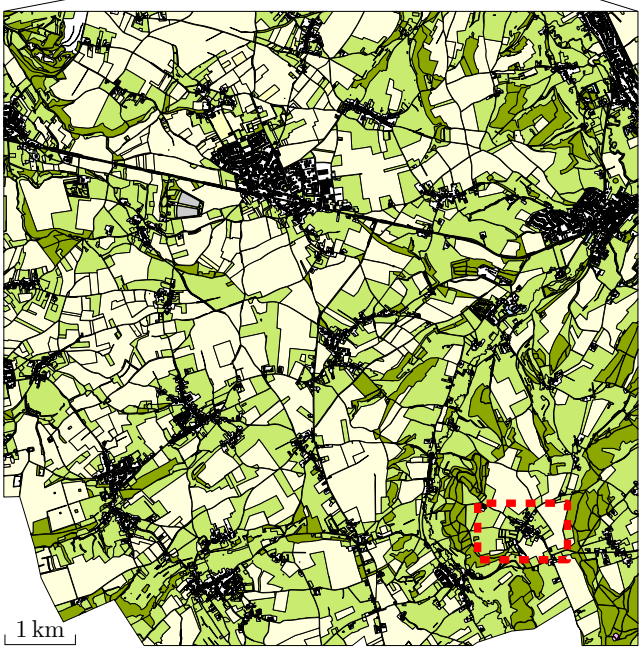
$$[[1, 1304], [2, 1070], \ldots, [77, 2]],$$

[3] Figure 9a is obtained from article *12 Most Beautiful Regions in the Netherlands*; see https://www.touropia.com/regions-in-the-netherlands-map/, accessed: October 5, 2021.

[4] See the details at http://register.geostandaarden.nl/visualisatie/top10nl/1.2.0/BRT_TOP10NL_1.2_beschrijving_visualisatie.xlsx, accessed: January 15, 2020.

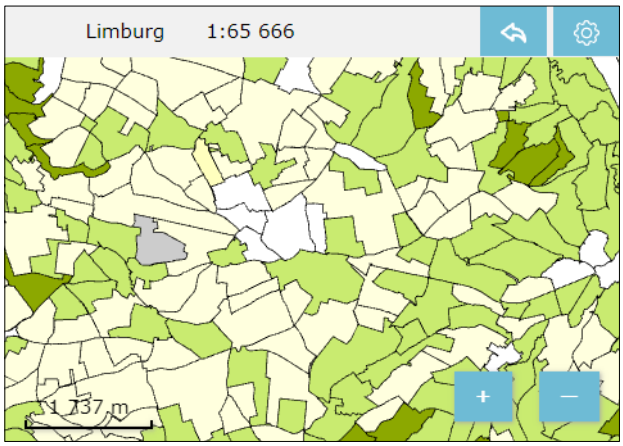
[5] All of our web maps can be found at https://congengis.github.io/webmaps/2021/10/merge/.

(a) A map of The Netherlands.

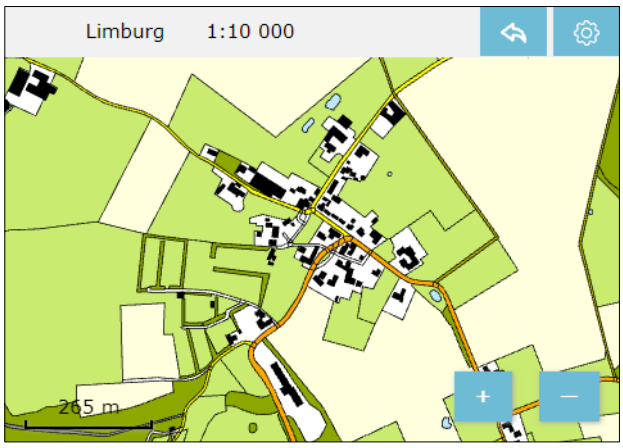


(b) The topographic map used in our case study. There are 13,238 area objects. The map is for scale 1 : 10,000.

Fig. 9: The data used in our case study.



(a) An overview map. The map is generated from the base map by simultaneous merging with parameter $r_{\text{simul}} = 0.01$.



(b) A part of the base map. The displayed place is marked by the dashed rectangle in Figure 9.

Fig. 10: Two examples of our web map with different scales.

which has 71 pairs of values. In reality, we would not use $r_{\text{simul}} = 0.1$ (merging 10% of the current areas in every step) because it is an unrealistic high value. Using such a high value results in a multi-scale representation (because we have a few valid states or scales), whereas we would like to have representations at nearly arbitrary scales.

We set zooming factor $f_{\text{zoom}} = 1$ and zooming duration $t_{\text{zoom}} = 1s$ (see Section 3.5). When zooming on our web maps with different simultaneous parameters, we observed that the impressions of the maps based on single merging[6] and based on simultaneous merging with parameter $r_{\text{simul}} = 0.001$ are almost the same. The reason is that the smooth merging happens too fast, and we cannot really see the merging animation. We get the feeling of smooth merging when $r_{\text{simul}} = 0.01$. When $r_{\text{simul}} = 0.1$, the smooth merging is already obvious. In order to show a better comparison of single

[6] See the web map at https://congengis.github.io/webmaps/2021/10/merge/limburg-single-merging.

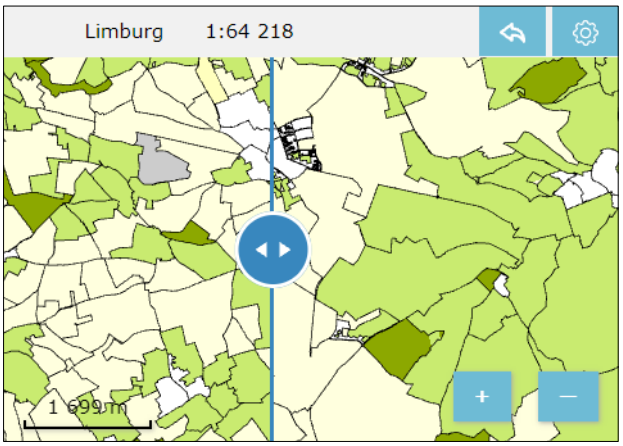


Fig. 11: The map to the left of the slider is based on single merging, and the one to the right is based on simultaneous merging with $r_{\text{simul}} = 0.1$. The slider can be moved to tune the widths of the two map canvases. The two maps are very different, and we can observe some sudden changes across the slider.

merging and simultaneous merging, we put two maps together (see Figure 11), where the simultaneous parameter is 0.1.[7]

When simultaneous parameter r_{simul} is set to 0.1, we noticed a problem in the northeastern corner of the map. That is, some tiny and relatively unimportant areas stay until the scale is very small, while they should be merged much earlier. This is due to the fact that there are many buildings in the northeastern corner of the map. When the buildings share the same surrounding area, they become holes of the polygon. In each step, only one of the buildings will be merged into the surrounding area because an area is allowed to merge with only one area in each step. In the meantime, the areas at other places of the map merge relatively fast because we allow 10% of the areas to be merged in each step. Fortunately, we would not need to use such a big simultaneous parameter in reality.

5 Concluding remarks

5.1 Conclusion

This paper examines the simultaneous processing of generalization operations, using the merging operation as a case study. The purpose of having simultaneous generalization operations is to have smoother zooming experience later on (compared to the pure sequenced individual generalization events) so that users can better keep their context during zooming. This paper develops a greedy algorithm to find simultaneous events of merging area objects. Then, the simultaneous events are integrated into the tGAP and the SSC to nicely visualize the merging animations. To guarantee that the merging animations always are completely shown while zooming, we managed to snap zooming operations to valid states. This paper also presents a recipe to define the animation duration of an event. According to our case study, the events of simultaneous merging can be better observed than the events of single merging. This result shows the potential that users can keep their context better and can have smoother map interaction experience when zooming on a map based on simultaneous-event operations.

5.2 Future work

Many topics related to this research need to be studied further. Our case study with 13,238 area objects demonstrated the efficiency of our map. In our case study, all the data of the SSC is stored in a single file because the tested map is not very big. The map will display only after the whole file is loaded. For a topographic map with much more objects, We are also developing a method that divides the SSC into many parts, and each part is stored in a file. A file will be dynamically loaded when the user is reading the relevant place and scale of the map. Further, a file at the client side will be removed to release main memory if the corresponding part of map is not browsed for a long time. With those functionalities, our prototype will be able to handle a map with arbitrary number of area objects. Those functionalities are under development and will be presented in another paper in future.

This paper used a greedy algorithm to find simultaneous merging events for each step. Alternatively, it is possible to define merging steps by selecting and combining some single-event steps of a sequence found by some existing methods (e.g., the greedy algorithm of van Oosterom (2005) or the A* algorithm of Peng et al. (2020)).

Currently, the merging events distribute randomly on a map. If we are unlucky, there may be a lot of events happening in users' focused region for a zooming duration, which may cause the users to lose track of their interested objects; for another zooming duration, there may be no event happening in the focused region at all. The strategy of blocking neighbor areas in our greedy algorithm already mitigates the problem. However, it may be even better if we explicitly distribute the merging events evenly, then the workload for a user to follow

[7] See the map of comparison at https://congengis.github.io/webmaps/2021/10/merge/limburg-comparer-overlay-single-vs-0.1.html.

the events is stable during the zooming. To this end, we could divide a map into many regions using a field-tree-like, multiple-level grid (van Putten and van Oosterom 1998) or using the road network. Then, we could find a certain number of events in each of the regions according to the regions' sizes, which should result in an even distribution of events. Finally, we could compare our current greedy algorithm and the algorithm considering even distribution.

If some small parcels distribute at a large region, then the background parcel will eat all the small parcels. This case already happens in our web maps when some small buildings are inside a settlement area. In order to solve this problem, we could generate a tessellation of the small parcels using the method of Ai et al. (2015). Then, we could grow each of the small parcels inside its own cell. By then, the small parcels will touch each other and can be merged by eating each other.

Our current event consists of only the merging operation, it is also necessary to involve split operation because sometimes a merging operation results in an unnatural area. For example, it is weird to merge a long and thin area with one of the areas that are along it (see Haunert and Sester 2008). Therefore, such kind of long and thin areas should be split into several parts first. We may integrate a split method based on the straight skeleton (Haunert and Sester 2008) or the skeleton obtained from a constrained Delaunay triangulation (Ai and van Oosterom 2002, Meijers et al. 2016). In addition to area features, we also need to support line features for these long objects (e.g., roads, river, rail) for the smaller scales. In order to apply appropriate generalization operators for a certain scale, we need to extend and implement the framework to guide our generalization choices (Meijers et al. 2018).

To avoid clutter of vertices for zooming out, it is necessary to simplify the boundaries of the areas. Many existing methods could be integrated into our simultaneous paradigm. Meijers (2011a) proposed a method to simplify the boundaries simultaneously. Their results are topologically safe. Another choice would be the method of Imai and Iri (1988), which is able to minimize the number of vertices for a given error threshold. One more choice would be to construct compatible triangulations (see Peng 2019, Chapter 3) for the two levels of topographic maps. In the SSC, we could build some tilted walls to connect the two levels of compatible triangulations. When we slice this SSC to animate a zooming action, the boundaries of the areas are morphed (moved smoothly and simultaneously) between a detailed representation and a coarse representation. Furthermore, we would like to simplify point symbols and text labels in a smooth way. For this purpose, we may need to integrate the ideas of Haunert and Wolff (2017) and Schwartges et al. (2013).

We would also like to realize smooth zooming between a foreground map and a background map, which respectively present a thematic data and a topographic data. When a user zooms in, the foreground map should be displayed, and when the user zooms out, the background map should be displayed. Furthermore, we would like to integrate time dimension into our 3-dimensional SSC to form a 4-dimensional space-scale-time structure (a tesseract). Then, we have the opportunity to store all the updates along time in the tesseract. By "slicing" it, we will be able to output a map at any scale and any time point.

This paper develops the technique for smooth zooming based on simultaneous merging, and we hope that it allows map users to follow the zooming more easily. A future work is to examine how much map users benefit from our technique. We will conduct some usability tests based on the experience of Šuba (2017, Section 6.7) and Midtbø and Nordvik (2007). Another future work is to find optimal simultaneous parameters for different kinds of datasets.

References

Achanta R, Shaji A, Smith K, Lucchi A, Fua P, Süsstrunk S (2012) SLIC superpixels compared to state-of-the-art superpixel methods. IEEE Transactions on Pattern Analysis and Machine Intelligence 34(11):2274–2282, DOI 10.1109/TPAMI.2012.120, URL https://ieeexplore.ieee.org/document/6205760

Ai T, van Oosterom P (2002) GAP-tree extensions based on skeletons. In: Richardson DE, van Oosterom P (eds) Proc. 10th International Symposium on Spatial Data Handling (SDH), Springer Berlin Heidelberg, Ottawa, Canada, pp 501–513, DOI 10.1007/978-3-642-56094-1_37, URL https://link.springer.com/chapter/10.1007/978-3-642-56094-1_37

Ai T, Zhang X (2007) The aggregation of urban building clusters based on the skeleton partitioning of gap space. In: Fabrikant SI, Wachowicz M (eds) The European Information Society: Leading the Way with Geo-information, Lecture Notes in Geoinformation and Cartography, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 153–170, DOI 10.1007/978-3-540-72385-1_9, URL https://doi.org/10.1007/978-3-540-72385-1_9

Ai T, Zhang X, Zhou Q, Yang M (2015) A vector field model to handle the displacement of multiple conflicts in building generalization. International Journal of Geographical Information Science 29(8):1310–1331, DOI 10.1080/13658816.2015.1019886

Cheng T, Li Z (2006) Toward quantitative measures for the semantic quality of polygon generalization. Cartographica 41(2):487–499, DOI 10.3138/0172-6733-227U-8155, URL https://www.utpjournals.press/doi/abs/10.3138/0172-6733-227U-8155

Chimani M, van Dijk TC, Haunert JH (2014) How to eat a graph: Computing selection sequences for the continuous generalization of road networks. In: Proc. 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACMGIS), Dallas, TX, USA, pp 243–252, DOI 10.1145/2666310.2666414, URL http://doi.acm.org/10.1145/2666310.2666414

Deng M, Peng D (2015) Morphing linear features based on their entire structures. Transactions in GIS 19(5):653–677, DOI 10.1111/tgis.12111

Hampe M, Sester M, Harrie L (2004) Multiple representation databases to support visualization on mobile devices. In: Proc. 20th ISPRS Congress, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, vol XXXV (B4: IV), pp 135–140, URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.184.3303

Haunert JH, Sester M (2008) Area collapse and road centerlines based on straight skeletons. GeoInformatica 12(2):169–191, DOI 10.1007/s10707-007-0028-x, URL https://link.springer.com/article/10.1007/s10707-007-0028-x

Haunert JH, Wolff A (2010) Area aggregation in map generalisation by mixed-integer programming. International Journal of Geographical Information Science 24(12):1871–1897, DOI 10/c8v8s2, URL http://doi.org/c8v8s2

Haunert JH, Wolff A (2017) Beyond maximum independent set: An extended integer programming formulation for point labeling. ISPRS International Journal of Geo-Information 6(11), DOI 10.3390/ijgi6110342

Huang L, Meijers M, Šuba R, van Oosterom P (2016) Engineering web maps with gradual content zoom based on streaming vector data. ISPRS Journal of Photogrammetry and Remote Sensing 114:274–293, DOI 10.1016/j.isprsjprs.2015.11.011, URL http://www.sciencedirect.com/science/article/pii/S0924271615002646

Huang L, Ai T, van Oosterom P, Yan X, Yang M (2017) A matrix-based structure for vario-scale vector representation over a wide range of map scales: The case of river network data. ISPRS International Journal of Geo-Information 6(7), DOI 10.3390/ijgi6070218

Imai H, Iri M (1988) Polygonal approximations of a curve—formulations and algorithms. In: Toussaint GT (ed) Machine Intelligence and Pattern Recognition, Computational Morphology: A Computational Geometric Approach to the Analysis of Form, vol 6, Elsevier, pp 71–86, DOI 10.1016/B978-0-444-70467-2.50011-4, URL https://www.sciencedirect.com/science/article/pii/B9780444704672500114

van Kreveld M (2001) Smooth generalization for continuous zooming. In: Proc. 5th ICA Workshop on Generalisation and Multiple Representation (ICAGM), Beijing, China, URL http://www.staff.science.uu.nl/~kreve101/papers/smooth.pdf

Li J, Ai T, Liu P, Yang M (2017a) Continuous scale transformations of linear features using simulated annealing-based morphing. ISPRS International Journal of Geo-Information 6(8), DOI 10.3390/ijgi6080242, URL http://www.mdpi.com/2220-9964/6/8/242

Li J, Li X, Xie T (2017b) Morphing of building footprints using a turning angle function. ISPRS International Journal of Geo-Information 6(6), DOI 10.3390/ijgi6060173, URL http://www.mdpi.com/2220-9964/6/6/173

Li J, Liu P, Yu W, Cheng X (2018) The morphing of geographical features by fourier transformation. PLOS ONE 13(1):1–13, DOI 10.1371/journal.pone.0191136, URL https://doi.org/10.1371/journal.pone.0191136

Mackaness WA, Burghardt D, Duchêne C (2016) Map generalization. In: International Encyclopedia of Geography: People, the Earth, Environment and Technology, John Wiley & Sons, pp 1–16, DOI 10/cx89, URL http://doi.org/cx89

Meijers M (2011a) Simultaneous & topologically-safe line simplification for a variable-scale planar partition. In: Geertman S, Reinhardt W, Toppen F (eds) Advancing Geoinformation Science for a Changing World, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 337–358, DOI 10.1007/978-3-642-19789-5_17, URL https://doi.org/10.1007/978-3-642-19789-5_17

Meijers M (2011b) Variable-scale geo-information. phdthesis, Delft University of Technology, URL http://www.gdmc.nl/publications/2011/Variable-scale_Geo-information.pdf

Meijers M, Savino S, van Oosterom P (2016) SplitArea: An algorithm for weighted splitting of faces in the context of a planar partition. International Journal of Geographical Information Science 30(8):1522–1551, DOI 10.1080/13658816.2016.1140770, URL https://doi.org/10.1080/13658816.2016.1140770, https://doi.org/10.1080/13658816.2016.1140770

Meijers M, van Oosterom P, Šuba R, Peng D (2018) Towards a scale dependent framework for creating vario-scale maps. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-4:425–432, DOI 10.5194/isprs-archives-XLII-4-425-2018, URL https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-4/425/2018/

Meijers M, van Oosterom P, Driel M, Šuba R (2020) Web-based dissemination of continuously generalized space-scale cube data for smooth user interaction. International Journal of Cartography 6(1):152–176, DOI 10.1080/23729333.2019.1705144, URL https://doi.org/10.1080/23729333.2019.1705144, https://doi.org/10.1080/23729333.2019.1705144

Midtbø T, Nordvik T (2007) Effects of animations in zooming and panning operations on web maps: A web-based experiment. The Cartographic Journal 44(4):292–303, DOI 10/dgnjmj, URL http://doi.org/dgnjmj, https://doi.org/10.1179/000870407X241845

Müller JC, Weibel R, Lagrange JP, Salgé F (1995) Generalization: State of the art and issues. In: Müller JC, Lagrange JP, Weibel R (eds) GIS and Generalization: Methodology and Practice, no. 1 in GISDATA, Taylor & Francis, London, UK, chap 1, pp 3–17

Nöllenburg M, Merrick D, Wolff A, Benkert M (2008) Morphing polylines: A step towards continuous generalization. Computers, Environment and Urban Systems 32(4):248–260, DOI 10/c7fgrw, URL https://www.sciencedirect.com/science/article/pii/S0198971508000331

Oehrlein J, Haunert JH (2017) A cutting-plane method for contiguity-constrained spatial aggregation. Journal of Spatial Information Science (15):89–120, DOI 10.5311/JOSIS.2017.15.379

van Oosterom P (1995) The GAP-tree, an approach to On-the-Fly map generalization of an area partitioning. In: Mueller JC, Lagrange JP, Weibel R (eds) GIS and Generalization: Methodology and Practice, Taylor & Francis, pp 120–132

van Oosterom P (2005) Variable-scale topological data structures suitable for progressive data transfer: The GAP-face tree and GAP-edge forest. Cartography and Geographic Information Science 32(4):331–346, DOI 10/chr7sf, URL https://www.tandfonline.com/doi/abs/10.1559/152304005775194782

van Oosterom P, Meijers M (2014) Vario-scale data structures supporting smooth zoom and progressive transfer of 2D and 3D data. International Journal of Geographical Information Science 28(3):455–478, DOI 10.1080/13658816.2013.809724, URL https://doi.org/10.1080/13658816.2013.809724, https://doi.org/10.1080/13658816.2013.809724

van Oosterom P, Schenkelaars V (1995) The development of an interactive multi-scale GIS. International Journal of Geographical Information Systems 9(5):489–507, DOI 10/fgzjvb, URL http://dx.doi.org/10.1080/02693799508902052

van Oosterom P, Meijers M, Stoter J, Šuba R (2014) Data structures for continuous generalisation: tGAP and SSC. In: Burghardt D, Duchêne C, Mackaness W (eds) Abstracting Geographic Information in a Data Rich World: Methodologies and Applications of Map Generalisation, Lecture Notes in Geoinformation and Cartography, Springer, Cham, chap 4, pp 83–117, DOI 10.1007/978-3-319-00203-3_4, URL https://link.springer.com/chapter/10.1007/978-3-319-00203-3_4

Pantazis D, Karathanasis B, Kassoli M, Koukofikis A, Stratakis P (2009a) Morphing techniques: Towards new methods for raster based cartographic generalization. In: Proc. 24th International Cartographic Conference (ICC), Santiago, Chile, URL https://icaci.org/files/documents/ICC_proceedings/ICC2009/html/refer/19_5.pdf

Pantazis D, Koukofikis A, Karathanasis B, Kassoli M (2009b) Are the morphing techniques useful for cartographic generalization? In: Urban and Regional Data Management, CRC Press, pp 195–204, DOI 10.1201/9780203869352.ch18, URL http://dx.doi.org/10.1201/9780203869352.ch18

Peng D (2019) An optimization-based approach for continuous map generalization. phdthesis, University of Würzburg, DOI 10.25972/WUP-978-3-95826-105-1, URL https://opus.bibliothek.uni-wuerzburg.de/frontdoor/index/index/docId/17442

Peng D, Touya G (2017) Continuously generalizing buildings to built-up areas by aggregating and growing. In: Proc. 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics (UrbanGIS), ACM, Redondo Beach, CA, USA., DOI 10.1145/3152178.3152188

Peng D, Deng M, Zhao B (2012) Multi-scale transformation of river networks based on morphing technology. Journal of Remote Sensing 16(5):953–968, URL http://www.jors.cn/jrs/ch/reader/view_abstract.aspx?file_no=r11272&flag=1

Peng D, Haunert JH, Wolff A, Hurter C (2013) Morphing polylines based on least squares adjustment. In: Proc. 16th ICA Workshop on Generalisation and Multiple Representation (ICAGM), URL https://kartographie.geo.tu-dresden.de/downloads/ica-gen/workshop2013/genemappro2013_submission_6.pdf

Peng D, Wolff A, Haunert JH (2016) Continuous generalization of administrative boundaries based on compatible triangulations. In: Sarjakoski T, Santos YM, Sarjakoski TL (eds) Proc. 19th AGILE Conference on

Geographic Information Science, Geospatial Data in a Changing World, Springer, Helsinki, Finland, Lecture Notes in Geoinformation and Cartography, pp 399–415, DOI 10/c5kh

Peng D, Wolff A, Haunert JH (2020) Finding optimal sequences for area aggregation—A* vs. integer linear programming. ACM Transactions on Spatial Algorithms and Systems 7(1):1–40, DOI 10.1145/3409290, URL https://doi.org/10.1145/3409290

van Putten J, van Oosterom P (1998) New results with generalized area partitionings. In: Proc. 8th International Symposium on Spatial Data Handling (SDH), Vancouver, Canada, pp 485–495, URL www.gdmc.nl/oosterom/sdh98.pdf

Read P, Meyer MP (2000) Restoration of Motion Picture Film. Elsevier, URL https://www.elsevier.com/books/restoration-of-motion-picture-film/read/978-0-08-051619-6

Regnauld N (2003) Algorithms for the amalgamation of topographic data. In: Proc. 21st International Cartographic Conference, Durban, South Africa, pp 222–234

Regnauld N, Revell P (2007) Automatic amalgamation of buildings for producing ordnance survey 1 : 50 000 scale maps. The Cartographic Journal 44(3):239–250, DOI 10.1179/000870407X241782, URL https://doi.org/10.1179/000870407X241782

Schwartges N, Allerkamp D, Haunert JH, Wolff A (2013) Optimizing active ranges for point selection in dynamic maps. In: Proc. 16th ICA Workshop on Generalisation and Multiple Representation (ICAGM), URL https://kartographie.geo.tu-dresden.de/downloads/ica-gen/workshop2013/genemappro2013_submission_5.pdf

Sester M (2005) Optimization approaches for generalization and data abstraction. International Journal of Geographical Information Science 19(8–9):871–897, DOI 10.1080/13658810500161179, URL https://www.tandfonline.com/doi/abs/10.1080/13658810500161179

Shen Y, Ai T, Li W, Yang M, Feng Y (2019) A polygon aggregation method with global feature preservation using superpixel segmentation. Computers, Environment and Urban Systems 75:117–131, DOI 10.1016/j.compenvurbsys.2019.01.009, URL http://www.sciencedirect.com/science/article/pii/S0198971518305106

van Smaalen J (2003) Automated aggregation of geographic objects. phdthesis, Wageningen University, The Netherlands

Su B, Li Z, Lodwick G, Muller JC (1997) Algebraic models for the aggregation of area features based upon morphological operators. International Journal of Geographical Information Science 11(3):233–246, DOI 10.1080/136588197242374, URL https://doi.org/10.1080/136588197242374, https://doi.org/10.1080/136588197242374

Thiemann F, Sester M (2018) An automatic approach for generalization of land-cover data from topographic data. In: Behnisch M, Meinel G (eds) Trends in Spatial Analysis and Modelling: Decision-Support and Planning Strategies, Geotechnologies and the Environment, vol 19, Springer, chap 10, pp 193–207, DOI 10/c5kj, URL https://doi.org/10.1007/978-3-319-52522-8_10

Touya G, Dumont M (2017) Progressive block graying and landmarks enhancing as intermediate representations between buildings and urban areas. In: Proc. 20th ICA Workshop on Generalisation and Multiple Representation (ICAGM), URL https://kartographie.geo.tu-dresden.de/downloads/ica-gen/workshop2017/genemr2017_paper_1.pdf

Šuba R (2017) Design and development of a system for vario-scale maps. phdthesis, Delft University of Technology, DOI 10.7480/abe.2017.18.1877, URL https://journals.open.tudelft.nl/abe/article/view/1877

Šuba R, Meijers M, Huang L, van Oosterom P (2014) An area merge operation for smooth zooming. In: Huerta J, Schade S, Granell C (eds) Proc. 17th AGILE Conference on Geographic Information Science, Connecting a Digital Europe Through Location and Place, Springer, Cham, Lecture Notes in Geoinformation and Cartography, pp 275–293, DOI 978-3-319-03611-3_16, URL http://dx.doi.org/10.1007/978-3-319-03611-3_16

Šuba R, Meijers M, van Oosterom P (2016) Continuous road network generalization throughout all scales. ISPRS International Journal of Geo-Information 5(8), DOI 10.3390/ijgi5080145, URL http://www.mdpi.com/2220-9964/5/8/145

Ware JM, Jones CB, Bundy GL (1995) A triangulated spatial model for cartographic generalisation of areal objects. In: Frank AU, Kuhn W (eds) Spatial Information Theory A Theoretical Basis for GIS, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 173–192

Weibel R (1997) Generalization of spatial data: Principles and selected algorithms. In: van Kreveld M, Nievergelt J, Roos T, Widmayer P (eds) Algorithmic Foundations of Geographic Information Systems, Lecture Notes in Computer Science, vol 1340, Springer, chap 5, pp 99–152, DOI 10.1007/3-540-63818-0_5, URL https://link.springer.com/content/pdf/10.1007/3-540-63818-0_5.pdf

A Create the table of weights and the table of compatibility values

This appendix shows how to create the table of weights and the table of compatibility values in PostgreSQL. The values of the two tables are used in our greedy algorithm (see Section 3.1). Currently, we have not examined how to define the weight for a class, so we assign value 1 to the weights of all the classes. That is to say, the least important area is the one with the smallest size. The class similarity is defined based on the class codes as the codes indeed imply a hierarchy.

```
DROP TABLE IF EXISTS public.class_weights;
CREATE TABLE public.class_weights (
  code INTEGER,
  weight FLOAT
);

DROP TABLE IF EXISTS public.class_comp_matrix;
CREATE TABLE public.class_comp_matrix (
  code_from INTEGER,
  code_to INTEGER,
  code_dis FLOAT,
  comp_value FLOAT
);


CREATE OR REPLACE FUNCTION
  populate_class_weights(class_codes INTEGER[]) RETURNS void AS
  $$
  DECLARE
    class_code INTEGER;
  BEGIN
    FOREACH class_code IN ARRAY class_codes
    LOOP
      EXECUTE format(
        'INSERT INTO %s (code, weight)
         VALUES ($1, $2);', 'class_weights')
      USING class_code, 1;
    END LOOP;
  END;
  $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION
  populate_class_comp_matrix(class_codes INTEGER[]) RETURNS void AS
  $$
  DECLARE
    code_fr INTEGER;
    code_to INTEGER;
    division_fr INTEGER;
    division_to INTEGER;
    code_dis FLOAT;
    dis_max FLOAT := 10;
    comp_value FLOAT;
    code_dis_test INTEGER[];
    code_dis_test_arr INTEGER[] := ARRAY[[1000,8], [100,6], [10,4], [1,2]];
  BEGIN
    FOREACH code_fr IN ARRAY class_codes
    LOOP
      FOREACH code_to IN ARRAY class_codes
      LOOP
        code_dis := 0;
        IF code_fr != code_to THEN
          -- code_dis_test is the inner array of code_dis_test_arr
          FOREACH code_dis_test SLICE 1 IN ARRAY code_dis_test_arr
          LOOP
            division_fr := code_fr / code_dis_test[1];
            division_to := code_to / code_dis_test[1];
            IF division_fr != division_to THEN
              code_dis := code_dis_test[2];
              EXIT;
            END IF;
          END LOOP;
        END IF;
```

```
        comp_value := (dis_max - code_dis) / dis_max;
        EXECUTE format(
          'INSERT INTO %s (code_from, code_to, code_dis, comp_value)
          VALUES ($1, $2, $3, $4);', 'class_comp_matrix')
        USING code_fr, code_to, code_dis, comp_value;
      END LOOP;
    END LOOP;
  END;
  $$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION
  populate_tables() RETURNS void AS
  $$
  DECLARE
    total_cost FLOAT := 0;
    class_codes INTEGER[] := ARRAY[10310, 10311, 10410, 10411, 10510, 10600,
        10700, 10710, 10720, 10730, 10740, 10741, 10750, 10760, 10780,
        12400, 12500, 13000, 14010, 14030, 14040, 14050, 14060, 14080,
        14090, 14100, 14120, 14130, 14140, 14160, 14162, 14170, 14180];
  BEGIN
    EXECUTE populate_class_weights(class_codes);
    EXECUTE populate_class_comp_matrix(class_codes);
  END;
  $$ LANGUAGE plpgsql;


SELECT populate_tables();
```

B Communicate valid states

Section 3.4 shows how to snap to only valid states to avoid half-way merging. This appendix illustrates how to communicate valid states from the server side to the client side. By sending only the exceptions of the event number, we try to decrease the size of the sent data.

B.1 On the server side

On the server side, we compute the values shown in Table 3. These values are for merging sequence of Figure 3k–o (also see Table 1b), where simultaneous parameter r_{simul} was set to 0.3. Note that this parameter value is extremely high, just used to explain the principle in an artificial simple example. The computation starts from step 1. At the beginning, there are 7 areas on the map, i.e., $|M_0| = 7$. According to Equation 1, our target is to process three events simultaneously ($n_{\text{target},1} = 3$). However, only two event can be processed in step 1 because some areas are blocked (see Figure 6b). Therefore, we have $n_{\text{event},1} = 2$. We require that the low state is $s_{\text{low},1} = 0$ for the first step. Then, the s_{high} value can be computed by

$$s_{\text{high},i} = s_{\text{low},i} + n_{\text{event},i}. \tag{8}$$

That is, we have $s_{\text{high},1} = 2$ (also see the s_{high} value in the first row of Table 3). At this point, the computation for step 1 completes.

Table 3: Some information of the merging sequence shown in Table 3. Column n_{area} shows the area number of the map at starting state s, that is, $|M_s|$ of Equation 1.

step	n_{area}	n_{target}	n_{event}	s_{low}	s_{high}
1	7	3	2	0	2
2	5	2	2	2	4
3	3	1	1	4	5
4	2	1	1	5	6

For the next step, the number of areas can be computed by

$$n_{\text{area},i+1} = n_{\text{area},i} - n_{\text{event},i},$$

where variable $n_{\text{area},i}$ denotes the area number at the low state of step i. Furthermore, the state-low value of step $i+1$ (i.e., $s_{\text{low},i+1}$) is the same as the state-high value of step i (i.e., $s_{\text{high},i}$). Again, the target number of simultaneous events (i.e., $n_{\text{target},i+1}$) is computed by Equation 1, the number of actual simultaneous events is obtained from the greedy algorithm, and the state-high value (i.e., $s_{\text{high},i+1}$) is computed by Equation 8. The computation of all the steps starts from step $i = 1$ and finishes until only one area left on the map. As a result, we have all the values of Table 3.

Now, we have a column of n_{event} values. Among them, we record the exceptions (i.e., when value n_{event} is different from value n_{target}) with the corresponding steps in a list. The *exception list* is $[[1, 2]]$ for the example of Table 3. For the pair of values in the inner square brackets, the first one represents the step, and the second value represents the actual number of events n_{event}. The exception list, the number of faces, and the simultaneous parameter will be sent to the client side.

B.2 On the client side

When a user opens our web map, the client side receives the exception list, the number of faces, and the simultaneous parameter from the server side. Starting from step $i = 1$, we see if the step is in the exception list. If so, the event value of step i from the list is assigned to $n_{\text{event},i}$. If not, value $n_{\text{target},i}$ (see Equation 1) is computed and assigned to $n_{\text{event},i}$. As a result, we have the n_{event} values on the client side (see column n_{event} in Table 3). By accumulating the n_{event} values, we obtain the list of valid states $S_{\text{valid}} = [0, 2, 4, 5, 6]$.

C Animation duration of an event

Let f_{zoom} be the zooming factor, and let t_{zoom} be the zooming duration. Let $1 : S_{\text{t,snap}}$ be the snapped scale before the zooming operation, and let $1 : S_{\text{o}}$ be the zoomed out scale (not snapped yet). For zooming out, we define the relationship between the two scale denominators as

$$S_{\text{o}} = S_{\text{t,snap}}(1 + f_{\text{zoom}}). \tag{9}$$

For scale $1 : S_{\text{o}}$, we compute the number of events that should be processed from the base map by

$$E_{\text{o}} = N_{\text{b}} \left(1 - \frac{S_{\text{b}}^2}{S_{\text{o}}^2}\right),$$

which is according to Equation 5. As the value of E_{o} may be not an integer, we snap it to a valid state (see Section 3.4), and we have a snapped value $E_{\text{o,snap}}$. Then, we compute scale denominator $S_{\text{o,snap}}$ by Equation 6. According to Equation 5, we have merged $E_{\text{t,snap}}$ areas when arriving at scale $1 : S_{\text{t,snap}}$. The event number of zooming out from scale $1 : S_{\text{t,snap}}$ to scale $1 : S_{\text{o,snap}}$ is

$$N_{\text{event}} = E_{\text{o,snap}} - E_{\text{t,snap}}.$$

Recall that zooming duration t_{zoom} is for zooming from scale $1 : S_{\text{t,snap}}$ to scale $1 : S_{\text{o}}$. As the map is actually zooming to $1 : S_{\text{o,snap}}$, we adjust the zooming duration to

$$t_{\text{snap}} = t_{\text{zoom}} \frac{N_{\text{event}}}{E_{\text{o}} - E_{\text{t,snap}}}.$$

That is to say, the $E_{\text{o,snap}} - E_{\text{t,snap}}$ events will happen in time duration t_{snap}. If the events happen sequentially (each step consists of a single event), then the animation duration of each event is

$$t_{\text{single}} = \frac{t_{\text{snap}}}{N_{\text{event}}} = \frac{t_{\text{zoom}}}{E_{\text{o}} - E_{\text{t,snap}}}. \tag{10}$$

If we process these events simultaneously, then we will have fewer steps and each event has more time to take place. Let n_{step} be the number of steps in a zooming duration. If we are lucky enough so that expression $r_{\text{simul}} \cdot |M_s|$ of Equation 1 always returns an integer, then we do not need the ceiling function of Equation 1 (if we are not that lucky, the value of n_{step} will be slightly different). We have

$$N_{\text{t,snap}}(1 - r_{\text{simul}})^{n_{\text{step}}} = N_{\text{o,snap}},$$

where $N_{\text{t,snap}} = N_{\text{b}} - E_{\text{t,snap}}$ is the number of areas at scale $1 : S_{\text{t,snap}}$, and $N_{\text{o,snap}} = N_{\text{b}} - E_{\text{o,snap}}$ is the number of areas at scale $1 : S_{\text{o,snap}}$. Then, the number of steps can be computed by

$$n_{\text{step}} = \log_{1-r_{\text{simul}}} \frac{N_{\text{o,snap}}}{N_{\text{t,snap}}}.$$

Because the steps happen sequentially, each of the steps in the zooming duration has animation duration

$$t_{\text{simul}} = \frac{t_{\text{snap}}}{n_{\text{step}}}, \tag{11}$$

which is also the animation duration of each of the simultaneous events. Putting Equations 10 and 11 together, we have

$$t_{\text{simul}} = t_{\text{single}} \frac{N_{\text{event}}}{n_{\text{step}}}.$$

As N_{event} is larger than or equal to n_{step}, t_{simul} is also larger than or equal to t_{single}.

When we zoom in back from scale $S_{\text{o,snap}}$ to scale S_{t}, we have

$$S_{\text{t}} = \frac{S_{\text{o,snap}}}{(1 + f_{\text{zoom}})},$$

which is the inverse function of Equation 9. We will be able to snap to scale $1 : S_{\text{t,snap}}$. We will use the same animation duration and process the same number of events and steps as we zoomed out. The difference from zooming out is that, instead of merging, areas will bubble up.