International Cartographic Association
Association Cartographique Internationale

Paralleling generalization operations to support smooth zooming: case study of merging area objects

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

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

* Corresponding Author

Abstract: When users zoom out on a digital map, some area objects become too tiny to be seen, resulting in visual clutters. To avoid this problem, the relatively unimportant areas should be merged with their neighbors to form larger areas. In order to provide small and smooth changes so that users can easily keep their contexts, we merge a pair of areas by expanding one over the other and parallel the merging operations. We also require that the area objects involved in paralleled merging operations should not have any common neighbor so that the topology of the map can be easily maintained. The zooming of our map is realized based on the topological area partitioning tree (GAP-tree) and the space-scale cube (SSC). Our case study shows that our method can improve the zooming visualization. We consider that paralleling generalization operations is an important step towards continuous map generalization.

Keywords: Space-scale cube, vario-scale map, continuous map generalization

1. Introduction

When users are reading a digital map, they expect different levels of detail (LoDs) depending on the scales. For example, they may want to see individual buildings when zooming in and see built-up areas when zooming out. That is why 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 the vario-scale map with smooth scale transition. Algorithms of CMG have been proposed to morph raster maps (e.g., Pantazis et al., 2009), to morph polylines (e.g., Nöllenburg et al., 2008; Deng and Peng, 2015; Li et al., 2017a), to generalize buildings (e.g., Li et al., 2017b; 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), and to transform 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), amalgamating (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 parallelly generalize the area objects.

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

2. Related work

2.1 Merging of area objects

Much research has been devoted to the merging of area objects. Haunert and Wolff (2010) developed a method based on mixed-integer programming to merge area objects in order to generate a map at a certain scale. 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.

2.2 Gradual merging of area objects

To provide scale transition of small changes, van Oosterom (2005) proposed the topological Generalized Area Partitioning (tGAP) tree, where in each step the least important area is merged into its most compatible neighbor. Peng et al. (2020) tried to find an optimal sequence to merge area

objects based on the A* algorithm or an integer linear program. Šuba et al. (2016) continuously generalized a planar map of a road network.

Van Oosterom and Meijers (2014) developed the concept of the space-scale cube (SSC). The bottom of the SSC is a detailed topographic map, and 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. Furthermore, they represented the smooth tGAP in the SSC. A typical example is that an area merges with another one by gradually expanding over it. In the SSC of the smooth tGAP, the wall starts to tilt when the expansion begins. To build an SSC of the smooth tGAP, Šuba et al. (2014) proposed three methods to merge a pair of areas, 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 some concave polygons.

2.3 Paralleling generalization in CMG

Many methods of CMG naturally parallel generalization operators. In morphing polylines, the points of the polylines are moved parallelly (e.g., Nöllenburg et al., 2008; Li et al., 2017a). Li et al. (2017b) parallelly generalized individual buildings. In those methods, the polylines and the buildings were generalized parallelly 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 area objects of interest, we merge by gradually expanding an area over another area (see Figure 1q for an ongoing expanding). This expansion can be realized by slicing the space-scale cube (SSC, van Oosterom and Meijers, 2014) of Figure 2a from bottom to top. For example, Figure 1q is obtained by slicing Figure 2a at $z = 250$. The details of slicing an SSC are illustrated in Meijers et al. (2020). The SSCs of Figure 2 were built based on the *Eater* (see Section 2.2). In Figure 2, the z-coordinates are 100 times of the state values in Figure 1. We did this multiplication so that the contents can be better observed; otherwise, the two SSCs will be very short when displayed. Note that the merging is independent of users' area objects of interest; the merging operations happen outside the region of interest are also realized by expanding.

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 1e is obtained from Figure 1d by processing one merging event. Similarly, Figure 1l is obtained from Figure 1k 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 1d–j and five states in the merging sequence of Figures 1k–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 paralleling merging operations. First, parallelization avoids unnatural zooming. Without parallelization, 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 paralleling the generalization operations, but no implementation, testing, or assessment of the idea was provided. Second, parallelization brings smoother zooming. Although the SSC allows to deliver a map at any scale, only 16 frames by default will be created to display in 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 parallelly, it is more likely that the gap between two frames is larger than the gap between two states. Then there is no animation of smooth merging at all. For example, if the consecutive frames are Figures 1d, 1e, and 1f, then users can only see discrete merging. In contrast, if the consecutive frames are Figures 1k, 1r, and 1l, then users can really see an ongoing expansion. The more merging operations we parallel, the smoother the expansion process seems.

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

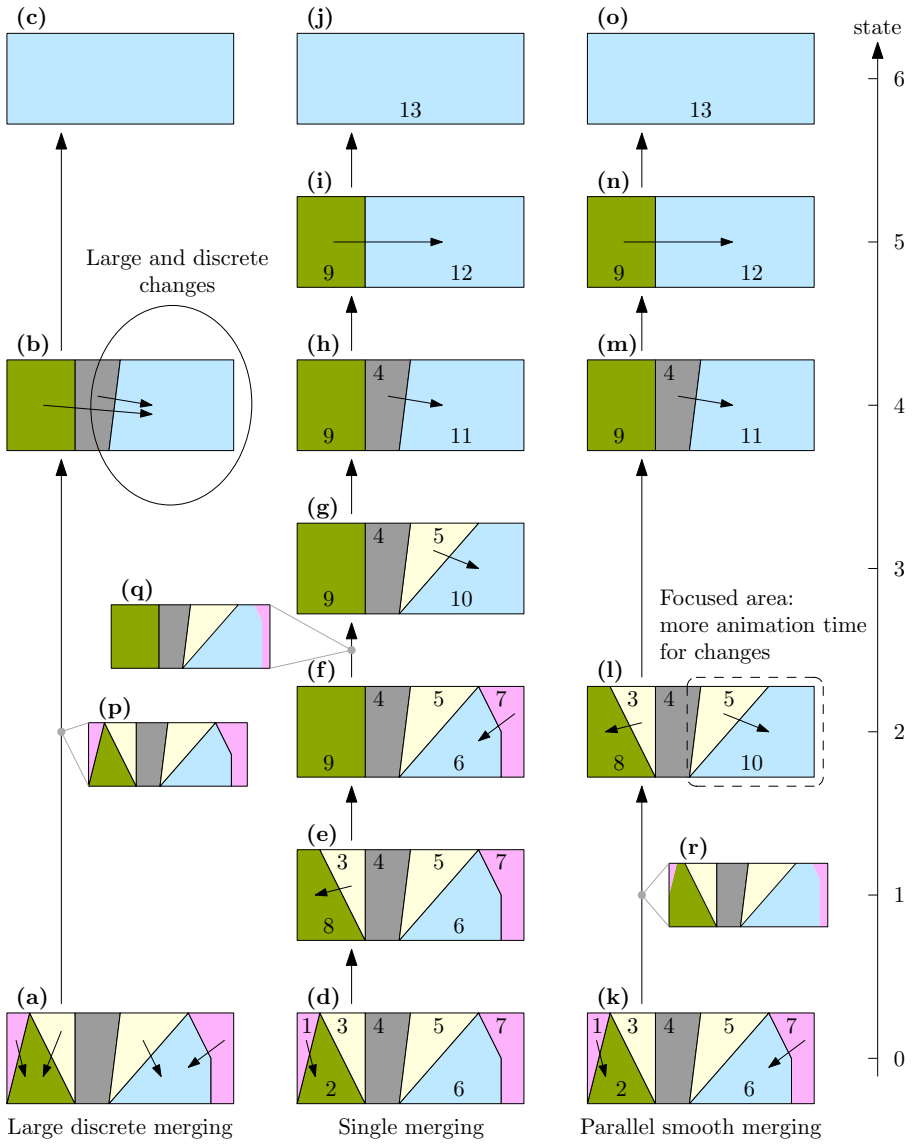
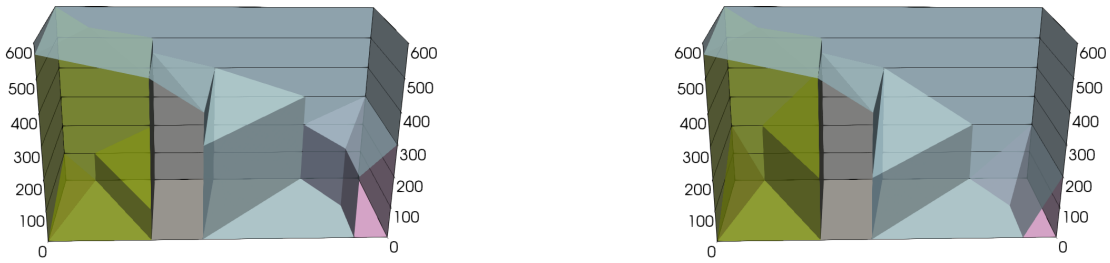


Figure 1. A comparison of different scale-transition strategies. Each arrow inside the subfigures indicates a merging operation. The arrow in the right-hand side indicates the states of zooming out. (a–c): All changes are processed in one go. (d–j): All changes are sequenced one by one rapidly. (k–o): Changes are grouped, resulting in more animation duration for every change. The numbers are the face IDs.

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.

Figure 3 shows the flowchart of our greedy algorithm. The process starts with state $s = 0$ and a detailed map of area objects, M_0. Expression $|M_s|$ denotes the number of area objects of the map at state s. Parallel parameter r_{parallel} specifies the proportion (i.e., percentage, when multiplied by 100) of the number of the area objects that we expect to merge parallelly, where $r_{\text{parallel}} \in [0, 1]$. Variable n_{target} denotes the number of area objects that we expect to merge in a step. However, we cannot always find n_{target} events because some areas may be blocked as explained before (also see Figure 4). Therefore, we use variable n_{event} to represent the number of events that actually happened within the step. In the algorithm, an area is *free* if it is not involved in an event and is not blocked. Figure 4 shows an example of blocking the surrounding neighbors of a_{least} and a_{nbr}. Note that the gray area in Figure 4a is not blocked because it is not considered a neighbor of the green area (sharing a point does not make the two areas neighbors).

Finally, the merging events will be stored as records in tGAP database tables (see Figure 5). Figures 1k–o show a sequence of four merging steps obtained by our greedy algorithm, where parallel parameter r_{parallel} is set to 0.3 (Note that this is an extremely high value, just used to explain the principle in an artificial simple example).



(a) The SSC of the single merging of Figures 1d–j.

(b) The SSC of the parallel merging of Figures 1k–o.

Figure 2. 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.

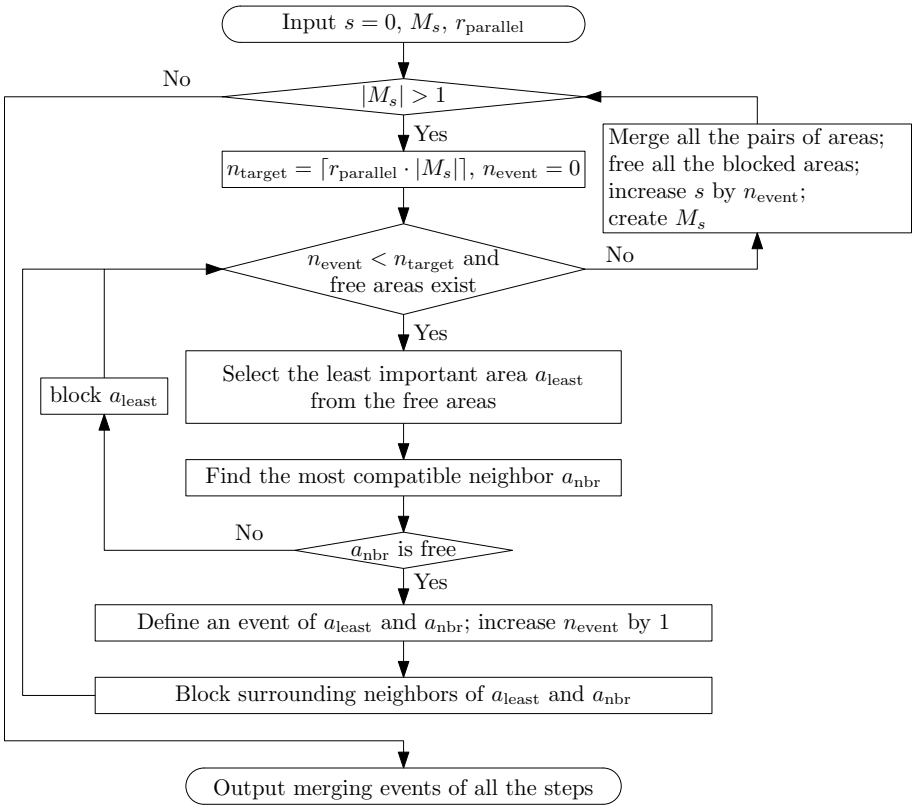


Figure 3. The flowchart of our greedy algorithm to find the merging events for all the steps.

3.2 Integrating the parallel 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 (also see Figure 5). We add columns *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, when the slicing plane arrives at the low state of a face, the face appears because of the merging of two faces. When the slicing arrives at the high state, the face should merge with another area. Comparing between the tables of single merging and parallel merging, we observed 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 1. Similarly, the columns and records of both the edge table and the face-hierarchy table will be changed accordingly.

3.3 Integrating the parallel events into the SSC

Recall that we merge a pair of areas by expanding the more important one over the less important one. 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 2) 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.

Figure 4. The process of finding parallel merging events for a step, where parallel parameter $r_{\text{parallel}} = 0.3$. (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.

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 1d–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 parallel merging shown in Figures 1k–o.

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

Take Figure 1 as an example, area 1 is merged into area 2 (Figures 1d), and area 8 is generated to take the combined place (Figures 1e). The tilted triangle is the one that spans from $z = 0$ to $z = 100$ (i.e., from state 0 to state 1) in Figure 2a. 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 parallel 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 parallelly in the same step, each of the events can share its merging duration. As a result, each of the parallel 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 parallel events.

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, we can avoid that a merging operation stops half-way, and users will not see transition artifacts (such as slivers). Take the sequence of Figure 1k–o for example, the merging animation should stop at either 1k or 1l, but not at 1r. Note that some states are not valid because of the parallel events. For example, state 1 is not valid in the sequence of Figure 1k–o, where 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 parallel 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_{parallel}. In case of high value ratios (e.g., $r_{\text{parallel}} > 0.01$), 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.

According to how much a user has zoomed, the target scale (i.e.,$1 : S_{\text{t}}$) 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 in reality of all the area objects . 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

$$\frac{N_{\text{b}}}{A_{\text{real}}/S_{\text{b}}^2} = \frac{N_{\text{b}} - E_{\text{t}}}{A_{\text{real}}/S_{\text{t}}^2}, \tag{1}$$

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

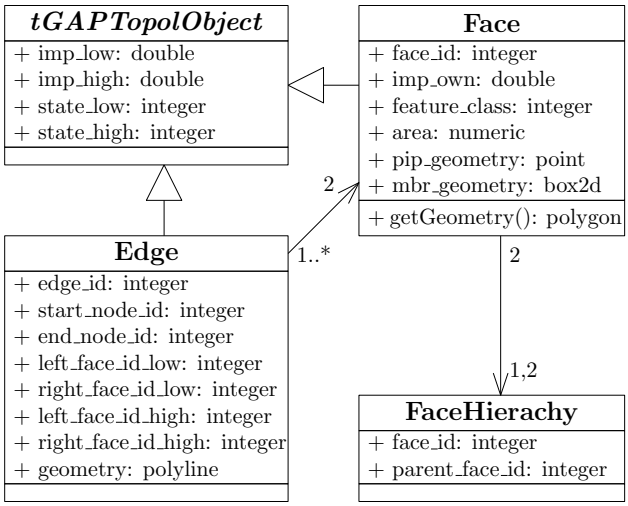


Figure 5. The UML diagram of the classes stored in tGAP database tables. This diagram is a slightly improved version of Meijers (2011b, p. 159). In the face table, property *pip_geometry* stores a point (usually the center) in the face (polygon).

one). Equation 1 yields

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

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

$$S_{\mathrm{t,snap}} = S_{\mathrm{b}} \sqrt{\frac{N_{\mathrm{b}}}{N_{\mathrm{b}} - E_{\mathrm{t,snap}}}}. \tag{3}$$

where this equation is an inverse function of Equation 2. At the end of the zooming action, the map will snap to state $s_{\mathrm{t,snap}}$ at scale $1 : S_{\mathrm{t,snap}}$. Note that state $s_{\mathrm{t,snap}}$ always has the same value as event number $E_{\mathrm{t,snap}}$.

3.5 Animation duration of a step

When users are zooming from a scale to another scale, some steps take place to change the map from a state to another one. We define the *zooming duration* as the amount of animation time that the map reacts to one scroll 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 this paper. Because of the page limit, we skip the deductions and show some conclusions as following. The animation duration of a set of events happening sequentially is

$$t_{\mathrm{single}} = \frac{t_{\mathrm{zoom}}}{N_{\mathrm{event}}},$$

where t_{zoom} is the zooming duration, and N_{event} is the number of events happening in one scroll of the mouse wheel. The animation duration of a set of events happening parallelly is

$$t_{\mathrm{parallel}} = \frac{t_{\mathrm{zoom}}}{n_{\mathrm{step}}},$$

where n_{step} is the number of steps happening in one scroll of the mouse wheel. As N_{event} is larger than or equal to n_{step}, we have $t_{\mathrm{parallel}} \geq t_{\mathrm{single}}$.

4. Case study

Figure 6 shows the topographical map of this case study. In each step, we want to parallelly merge some proportion of the areas. We tried three cases: 0.1%, 1%, and 10%. The three versions of map can be browsed online.[1] As no paper has recommended values for the zooming factor or the zooming duration, we respectively set the default values to 1 and 1 s, which performed well according to our experience. Figure 7 shows an example of our web map when parallel parameter $r_{\mathrm{parallel}} = 0.01$. When zooming on our web maps with different parallel parameters, we observed that the impressions of the maps based on single merging[2] and based on parallel merging with parameter $r_{\mathrm{parallel}} = 0.001$ 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 $r_{\mathrm{parallel}} = 0.01$. When $r_{\mathrm{parallel}} = 0.1$, the smooth merging is already obvious. In order to show a better comparison of single merging and parallel merging, we put two maps together (see Figure 8), where the parallel parameter is 0.1.[3]

5. Concluding remarks

This paper investigates on paralleling generalization operations, using the merging operation as a case study. According to our experiment, the events of parallel merging can be better observed than the events of single merging. This result shows the potential that, when zooming on a map based on parallel-event operations, users can keep their context better and can have smoother map interaction experience.

Many topics related to this research need to be investigated further. We need to test our method on a topographic map with much more objects, In that case, the client side will need to dynamically load and process the map data for the

[1] All of our web maps can be found at https://pengdlzn.github.io/webmaps/2020/10/merge/.

[2] See the web map at https://pengdlzn.github.io/webmaps/2020/10/merge/top10nl-single-merging.html.

[3] See the map of comparison at https://pengdlzn.github.io/webmaps/2020/10/merge/top10nl-0.1-comparer.html.

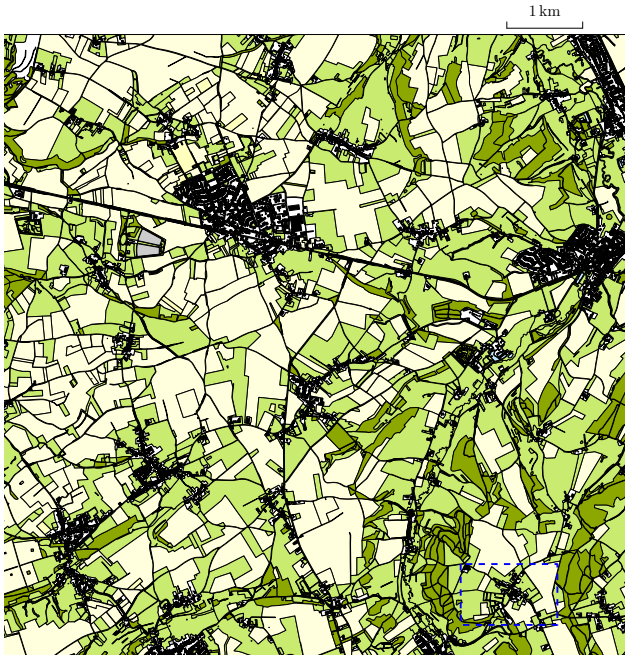


Figure 6. The topographic map represents the place in the south of Limburg, The Netherlands. There are 13,238 parcels. The map is for scale 1 : 10,000.

place and the scale being viewed. The client also needs to remove the loaded data that is not used for a while in order to release memory. With those functionalities, our prototype will be able to handle a map with arbitrary number of area objects. 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 (Haunert and Sester, 2008; Meijers et al., 2016). 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 parallel paradigm. Meijers (2011a) proposed a method to simplify the boundaries parallelly. Their results are topologically safe. Another future work is to investigate how much map users benefit from our parallel merging. We need to conduct some usability tests based on the experience of Šuba (2017,

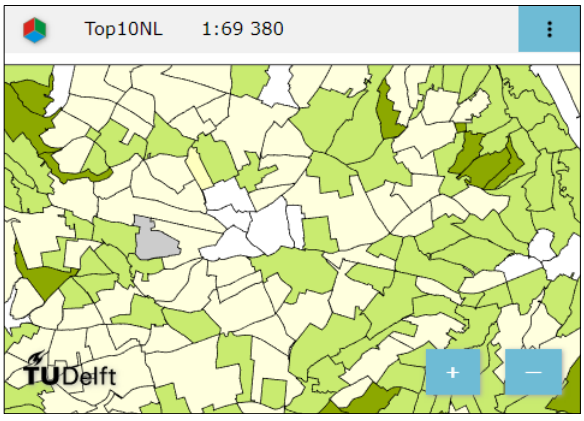


Figure 7. An overview map. The map is generated from the base map by parallel merging with parameter $r_{\text{parallel}} = 0.01$.

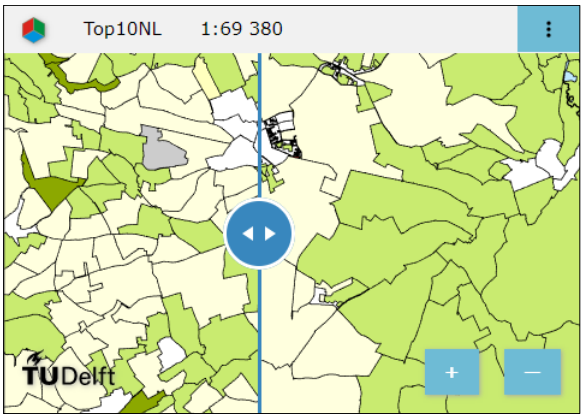


Figure 8. A comparison of single merging (left) and parallel merging (right, $r_{\text{parallel}} = 0.1$). The slider can be moved to tune the widths of the two map canvases. Some sudden changes across the slider can be observed.

Section 6.7) and Midtbø and Nordvik (2007). We currently drive the merging by the relationship between the number of areas and the scale; an other strategy is to drive by the relationship between the size of the smallest area and the scale. We also need to test which of the two strategies is better.

References

Cheng, T. and Li, Z., 2006. Toward quantitative measures for the semantic quality of polygon generalization. *Cartographica* 41(2), pp. 487–499.

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

Deng, M. and Peng, D., 2015. Morphing linear features based on their entire structures. *Transactions in GIS* 19(5), pp. 653–677.

Hampe, M., Sester, M. and 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.

Haunert, J.-H. and Sester, M., 2008. Area collapse and road centerlines based on straight skeletons. *GeoInformatica* 12(2), pp. 169–191.

Haunert, J.-H. and Wolff, A., 2010. Area aggregation in map generalisation by mixed-integer programming. *International Journal of Geographical Information Science* 24(12), pp. 1871–1897.

Huang, L., Ai, T., van Oosterom, P., Yan, X. and 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), pp. 1–20.

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

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

Li, J., Li, X. and Xie, T., 2017b. Morphing of building footprints using a turning angle function. *ISPRS International Journal of Geo-Information* 6(6), pp. 1–13.

Mackaness, W. A., Burghardt, D. and Duchêne, C., 2016. Map generalization. In: *International Encyclopedia of Geography: People, the Earth, Environment and Technology*, John Wiley & Sons, pp. 1–16.

Meijers, M., 2011a. Simultaneous & topologically-safe line simplification for a variable-scale planar partition. In: S. Geertman, W. Reinhardt and F. Toppen (eds), *Advancing Geoinformation Science for a Changing World*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 337–358.

Meijers, M., 2011b. Variable-scale Geo-information. phdthesis, Delft University of Technology.

Meijers, M., Savino, S. and 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), pp. 1522–1551.

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

Midtbø, T. and Nordvik, T., 2007. Effects of animations in zooming and panning operations on web maps: A web-based experiment. *The Cartographic Journal* 44(4), pp. 292–303.

Müller, J.-C., Weibel, R., Lagrange, J.-P. and Salgé, F., 1995. Generalization: State of the art and issues. In: J.-C. Müller, J.-P. Lagrange and R. Weibel (eds), *GIS and Generalization: Methodology and Practice*, GISDATA, Taylor & Francis, London, UK, chapter 1, pp. 3–17.

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

Pantazis, D., Koukofikis, A., Karathanasis, B. and Kassoli, M., 2009. Are the morphing techniques useful for cartographic generalization? In: *Urban and Regional Data Management*, CRC Press, pp. 195–204.

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

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

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

Read, P. and Meyer, M.-P., 2000. *Restoration of Motion Picture Film*. Elsevier.

Regnauld, N. and Revell, P., 2007. Automatic amalgamation of buildings for producing ordnance survey $1:50\,000$ scale maps. *The Cartographic Journal* 44(3), pp. 239–250.

Shen, Y., Ai, T., Li, W., Yang, M. and Feng, Y., 2019. A polygon aggregation method with global feature preservation using superpixel segmentation. *Computers, Environment and Urban Systems* 75, pp. 117–131.

Thiemann, F. and Sester, M., 2018. An automatic approach for generalization of land-cover data from topographic data. In: M. Behnisch and G. Meinel (eds), *Trends in Spatial Analysis and Modelling: Decision-Support and Planning Strategies*, Geotechnologies and the Environment, Vol. 19, Springer, chapter 10, pp. 193–207.

Touya, G. and 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)*.

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

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), pp. 331–346.

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

van Putten, J. and 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.

Šuba, R., 2017. Design and development of a system for vario-scale maps. phdthesis, Delft University of Technology.

Šuba, R., Meijers, M. and van Oosterom, P., 2016. Continuous road network generalization throughout all scales. *ISPRS International Journal of Geo-Information* 5(8), pp. 1–21.

Šuba, R., Meijers, M., Huang, L. and van Oosterom, P., 2014. An area merge operation for smooth zooming. In: J. Huerta, S. Schade and C. Granell (eds), *Proc. 17th AGILE Conference on Geographic Information Science, Connecting a Digital Europe Through Location and Place*, Lecture Notes in Geoinformation and Cartography, Springer, Cham, pp. 275–293.

Weibel, R., 1997. Generalization of spatial data: Principles and selected algorithms. In: M. van Kreveld, J. Nievergelt, T. Roos and P. Widmayer (eds), *Algorithmic Foundations of Geographic Information Systems*, Lecture Notes in Computer Science, Vol. 1340, Springer, chapter 5, pp. 99–152.