



Generalizing Simultaneously to Support Smooth Zooming: Case Study of Merging Area Objects

Dongliang Peng¹ · Martijn Meijers¹ · Peter van Oosterom¹

Accepted: 3 May 2022 / Published online: 11 May 2023
© The Author(s) 2023

Abstract

When users zoom in or out on a digital map, the map should change correspondingly to present geographical information at proper levels. A way to help map users better keep track of their interested objects is to change the map smoothly instead of discretely switching between several levels of detail. This paper focuses on the problem of providing smooth merging of area objects. We propose to merge multiple areas simultaneously to share their animation durations. In this way, each merging operation can be prolonged, and it is visually smoother. We present a greedy algorithm to decide which areas should be merged at each step. The merging process is pre-computed and is recorded into a space-scale cube (SSC). When a user accesses our web map, the SSC is sent to the client side so that the map can be generated by slicing the SSC in the graphics processing unit (GPU). We also explain how to snap the zooming to valid states so that the zooming will not stop halfway of the merging operations. Our case study shows that it is visually smoother to merge simultaneously than to sequentially merge each pair of areas.

Keywords Space-scale cube · Vario-scale map · Continuous map generalization · Web map · Simultaneous generalization

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, and aggregation. Often, a multi-representation database (MRDB) is utilized to store map data of different scales, and the proper data is sent to clients on request (e.g., Hampe et al. 2004). As a result, the scale transition of a map is realized by switching between different LoDs. However, that strategy often brings large and discrete changes, which confuse users. In order to provide users with better experience of zooming, we propose to realize the scale transition with smooth changes. In other words, each object on the map should be changed smoothly when the scale changes. For example, a smooth way to simplify a polyline is to move some of the vertices to a straight line, and a smooth way to make a polygon disappear is to fade it out. Because all the objects are changed smoothly, users can keep track of their interested objects more easily. The technology to realize the smooth scale transition is known as *continuous map generalization* (CMG). 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. 2017; Peng and Touya 2017; Touya and Dumont 2017), to transform road networks

This article belongs to the Topical Collection: *Automated map generalization: emerging techniques and new trends*
Guest Editors: Xiang Zhang, Guillaume Touya, Martijn Meijers.

✉ Martijn Meijers
b.m.meijers@tudelft.nl

Dongliang Peng
D.L.Peng@tudelft.nl

Peter van Oosterom
P.J.M.vanOosterom@tudelft.nl

¹ GIS Technology, Faculty of Architecture and the Built Environment, Delft University of Technology, Julianalaan 134, Delft, The Netherlands

or river networks (e.g., Šuba et al. 2016; Chimani et al. 2014; Huang et al. 2017; Peng et al. 2012), and to morph administrative boundaries (e.g., Peng et al. 2016). Recently, Shen et al. (2020) proposed a method to progressively collapse rivers based on superpixels. Furthermore, CMG can be also used to simulate some temporal evolution of phenomena, for example, how a flooding submerges the land.

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., Haurert and Wolff 2010), amalgamating (e.g., Ware et al. 1995), aggregating (e.g., Peng and Touya 2017), splitting (e.g., Meijers et al. 2016), and collapsing (e.g., Haurert 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 the *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, each generalization operation has more time to take place and thus expands in more animation frames, which makes the generalization operation visually smoother. The reasoning is as follows, taking merging as an example. For zooming out, we wish to merge the relatively unimportant areas into their neighbors to form larger areas. In order to provide small and smooth changes, this paper merges a pair of areas by gradually expanding one (winner) over the other (loser). At the same time, the loser gradually adapts its color to the winner. However, if all the merging operations happen sequentially, then each operation has to be processed very fast because the map user wants to see the map at the target scale rather soon after applying a zooming. That is to say, each merging expansion may take place in only one frame, and users see only discrete changes. In contrast, if some merging operations happen simultaneously, then they can share their time duration, and each merging expansion can take place in many frames. As a result, users really see the smooth merging.

This paper is organized as follows. The “[Related Work](#)” section reviews some related work. Our methodology is presented in the “[Methodology](#)” section. We show a case study in the “[Case Study](#)” section. Finally, the “[Concluding Remarks](#)” section draws the conclusion and presents the future work.

Related Work

Merging, amalgamation, and aggregation are three popular operators of combining area objects. According to Shea and McMaster (1989), merging combines neighbor objects, which (visually) share their boundaries, into a single one, and the result has the same dimension as the merged objects. Amalgamation is different from merging in that it combines nearby objects into a single one. In contrast, aggregation often involves the change of dimension. For example, points are aggregated to become an area.

Both Su et al. (1997) and Sester (2005) used morphological operators (e.g., a dilation followed by an erosion) to amalgamate area objects, where the former article worked on raster data and the latter worked on vector data. Regnaud (2003) amalgamated area objects by merging, bridging, flooding, or sampling. Shen et al. (2019) amalgamated area objects based on the superpixel method of Achanta et al. (2012). Ware et al. (1995) amalgamated some pairs of objects based on the constrained Delaunay triangulation (CDT), 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 the CDT. Peng and Touya (2017) continuously aggregated buildings to built-up areas by bridging and growing the buildings. Touya and Dumont (2017) aggregated buildings by progressively covering them with blocks.

We have just briefly reviewed the related work of amalgamation and aggregation. In the remainder of this section, we will focus on merging because our research belongs to this topic.

Merging of Area Objects

Cheng and Li (2006), for a target area, proposed three choices of selecting a neighboring area to merge, i.e., the neighbor has the largest size, shares the longest boundary with the target area, or has the closest class to the target 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, merging, reclassifying, and simplifying. Both Haurert and Wolff (2010) and Oehrlein and Haurert (2017) employed integer linear programs to merge area objects in order to find some optimal solutions when generating a map at a certain scale.

Gradual Merging of Area Objects

To provide the scale transition with small changes, van Oosterom (1995) proposed the generalized area partitioning

(GAP) tree. In the tree, each leaf node represents a simple area on the map, and each of the other nodes represents a compound area, where a compound area contains some other areas. To build the tree, the least important area is found and merged into its most compatible neighbor. Correspondingly, the former's node becomes a child of the latter's node. This process repeats until the root node is found. Each level of the GAP tree corresponds to a scale of the map. As the changes between two neighboring levels of the tree are small, it is possible to realize the scale transition with small changes. Because that method works only on areas without gaps, Ai and van Oosterom (2002) made an extension to the GAP tree to allow gaps between areas. They identified gaps based on the CDT and filled the gaps to connect nearby areas. The topological GAP (tGAP) structure consists of a face tree and an edge tree (van Oosterom 2005). The aim of proposing the tGAP structure is to minimize the redundancy, where each vertex is recorded only once while it may be presented in many map representations of different scales. Peng et al. (2020b) tried to find an optimal sequence to merge area objects based on the A* algorithm or an integer linear program. Their comparison, also 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 its local condition (e.g., no compatible neighbor at the same side of the road), they may take different decisions for the least important area object: increasing its importance, collapsing it, or merging it into an adjacent area.

To provide real smooth changes of zooming, van Oosterom and Meijers (2014) 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 becomes a vertical wall. Whenever a generalization operation happens, the extrusions of the involved areas stop; then, the newly generated areas take over 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 (e.g., Fig. 1). 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 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. To build such an SSC, Šuba et al. (2014) proposed three methods to merge a pair of areas in a gradual manner, namely the *Single flat plane*, the *Zipper*, and the *Eater*. Basically, the *winner* area gradually expands over the *loser* area. We will

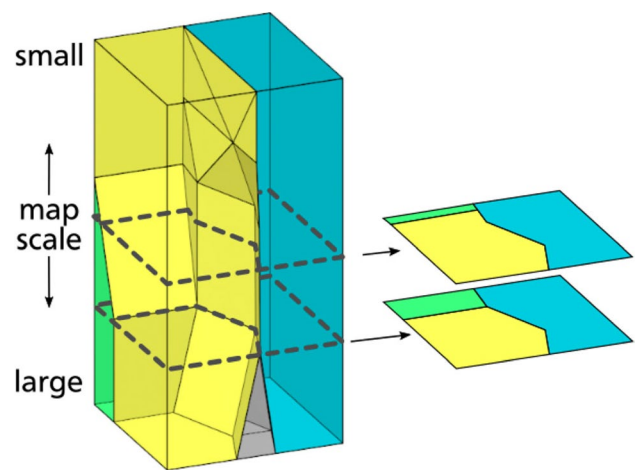


Fig. 1 Maps can be obtained by slicing the SSC; taken from Meijers et al. (2020)

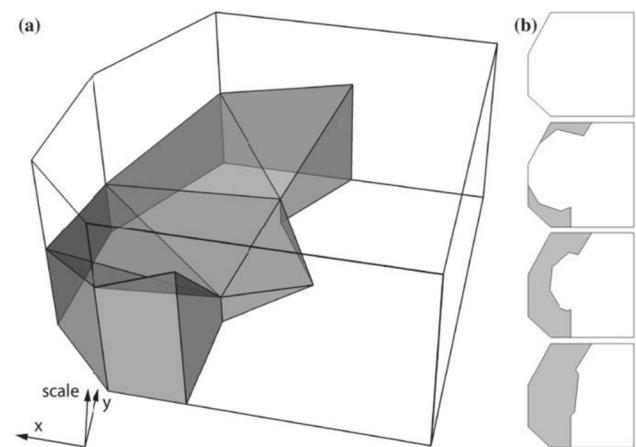


Fig. 2 The principle of the Eater; taken from Šuba et al. (2014). (a) The space scale cube of a loser area. (b) The eating process

use the *Eater* because it works for all kinds of polygons, while the other two methods have their limitations for some cases. For example, the two other methods do not work for certain concave polygons. The principle of the Eater is as follows. First, the interior of the loser is triangulated with a CDT (see Fig. 2a). Second, the triangles are visited starting from the boundary between the winner and the loser: If there are triangles with two shared edges, then the visiting starts from the shared vertex of the two edges; otherwise, it starts from the shared edges. During visiting, the vertices of the triangles are assigned with increasing z -values, and the tilted triangles are generated, which become the boundaries of polyhedra in the SSC. When slicing the SSC with a horizontal plane, the eating process is presented as shown in Fig. 2b.

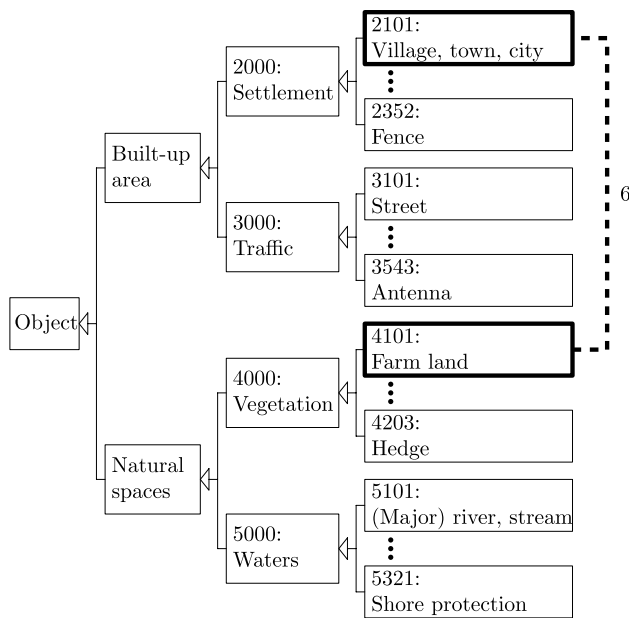


Fig. 3 A way of defining distances between the classes; taken from Peng et al. (2020b)

Merging Considering Semantic Properties

As mentioned in the “Introduction” section, 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’s (1995) 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. (2020b) defined the semantic distance based on a tree of classes, which guarantees that the distance is a metric (see Fig. 3).

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.

Simultaneous Generalization Operators

Many methods of CMG naturally apply multiple generalization operators simultaneously. In morphing polylines, the points of the polylines are moved at the same time (e.g., Nöhlenburg et al. 2008; Li et al. 2017a). Li et al. (2017) simultaneously generalized individual buildings. 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.

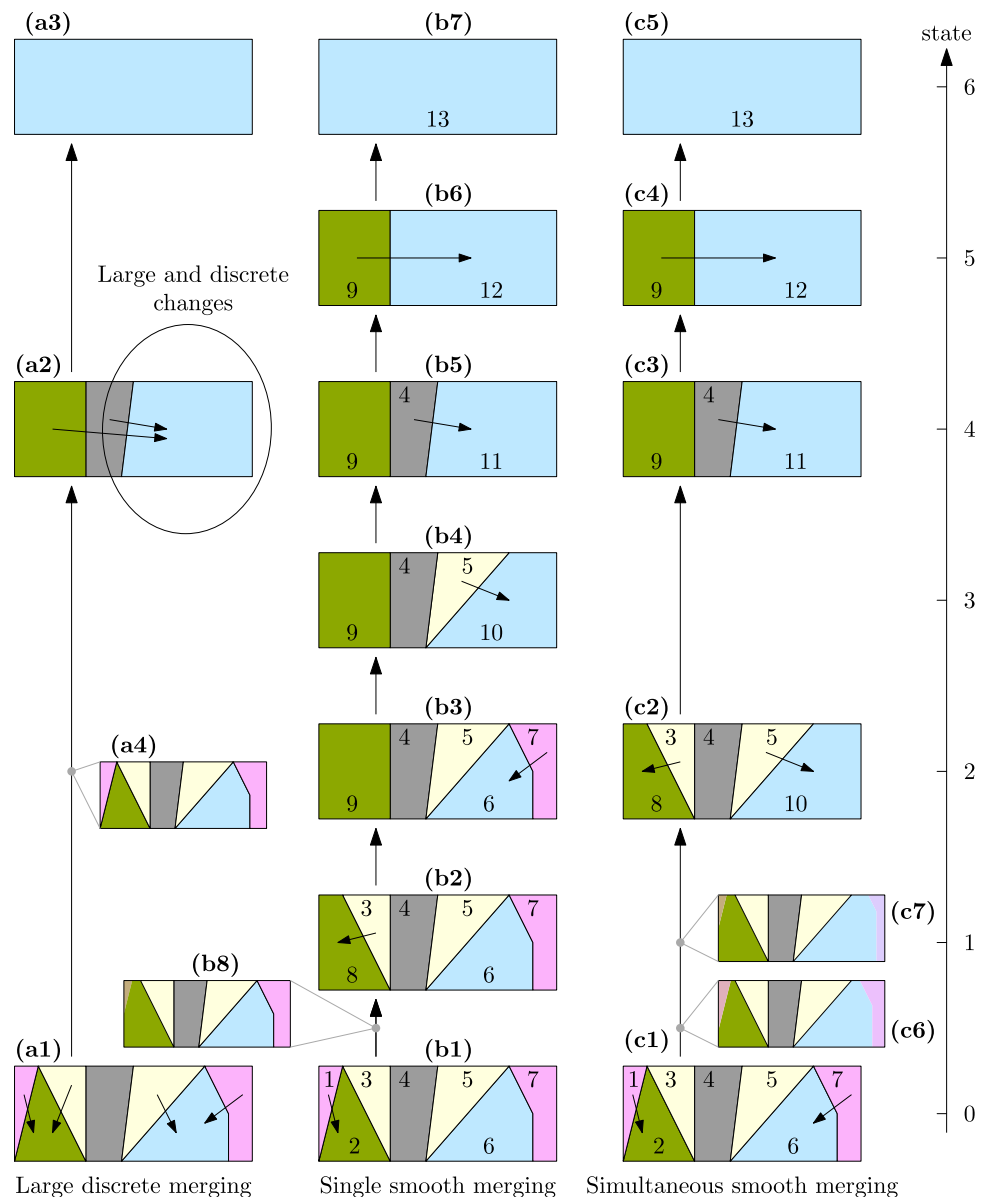
Gradual Transformation in Web Environment

Based on the SSC, 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. In addition to slicing the SSC with a horizontal plane, they also sliced the SSC with a curly surface to have a locally more detailed map or with a tilted surface to have a perspective view. Huang et al. (2016) pointed out that the effort of implementing online maps had been spent mainly on preparing data on the server side. They studied the communication of map data between the server side and the client side. They proposed different strategies of assigning the work of processing map data according to the machine abilities of the clients (i.e., thin, medium, or thick client). Their implementation or *option C* supports gradual transformation of objects. For a zooming operation, that implementation continuously requests data from the server side and present them on the map until the map of the target scale is complete. Peng et al. (2020a) presented a tool to compare two web maps side by side. In order to allow users to easily access other map information, the tool presents a multi-scale raster layer as the background. Their example respectively used a vario-scale and a multi-scale vector layer as the foregrounds and compared between them.

Methodology

Figure 4 shows three different merging strategies. In Fig. 4a1–a3, all the changes from a level to the next level are processed in one go, at a specific point. In Fig. 4b1–b7, there is only one merging operation from a level to the next level, and the change is realized by an expanding animation (see Fig. 4b8). In Fig. 4c1–c5, there can be many merging

Fig. 4 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. Subfigures (a1–a3), (b1–b7), and (c1–c5) represent the states at which the zooming may stop. Subfigures (a4), (b8), (c6), and (c7) are the map representations during the scale transition, where their corresponding states are indicated by the gray dots. The numbers are the IDs of the areas. Note that the colors of the smaller areas adapt to the colors of the larger areas during merging



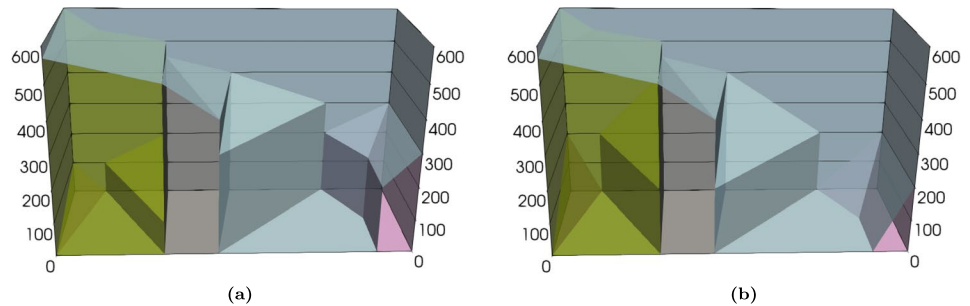
operations from a level to the next level, and each change is realized by an expanding animation (see Fig. 4c6). The first strategy often brings large and discrete changes (see for example from a1 to a2), which should be avoided. Both the second one and the third one have the ability to provide smooth changes. Comparing to the second strategy, the third one results in longer animation durations for some merging operations because the changes can share their animation durations.¹ The smooth changes are realized by slicing the

space-scale cubes (SSCs) of Fig. 5 with a moving horizontal plane. For example, smooth animations of zooming out are obtained by slicing an SSC from bottom to top. In detail, Fig. 4b8 is obtained by slicing Fig. 5a at $z = 50$. The details of slicing an SSC are illustrated in Meijers et al. (2020). The SSCs of Fig. 5 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 Fig. 5). In Fig. 5, the z -coordinates are 100 times of the state values in Fig. 4. We performed this multiplication for illustrative purpose only so that the contents of the SSC can be better observed.

The SSCs of Fig. 5a and b respectively serve for the single smooth merging (Fig. 4b1–b7) and the simultaneous smooth merging (Fig. 4c1–c5). The differences of the

¹ A comparison of the single merging and the simultaneous merging can be found at <https://pengdlzn.github.io/webmaps/2021/10/merge/eg-7-comparer-overlay-single-simultaneous.html>, where the swiper can be moved to see the differences of the two maps, for example, at scale 1:11, 832.

Fig. 5 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. (a) The SSC of the single merging of Fig. 4b1–b7. (b) The SSC of the simultaneous merging of Fig. 4c1–c7



two SSCs result in the two different merging processes. For example, at $z = 50$ of Fig. 5a, there is only one polyhedron with the tilted face, hence there is only a pair of polygons merging in Fig. 4b8. In comparison, there are two polyhedra with tilted faces at $z = 50$ of Fig. 5b, and there are two pairs of polygons merging in Fig. 4c6. Note that the choices of selecting areas to merge are made in a pre-processing step and are stored in a database, before users start zooming on the map. Therefore, the choices are independent of users' area objects of interest.

We define an *event* as a single generalization operation, such as merging an area with a neighbor. For example, Fig. 4b2 is obtained from Fig. 4b1 by processing one merging event. Similarly, Fig. 4c2 is obtained from Fig. 4c1 by processing two merging events. Note that 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). We define a *step* as a set of events happening at the same animation duration, for example, from Fig. 4b1 to b2 or from Fig. 4c1 to c2. 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 Fig. 4b1–b7 and five states in the merging sequence of Fig. 4c1–c5 (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, generalization operations that are processed all sequentially 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. When showing an animation zooming, we set 16 as the default value of the 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 time interval between two frames is larger than that between two states. Then, there is no animation of

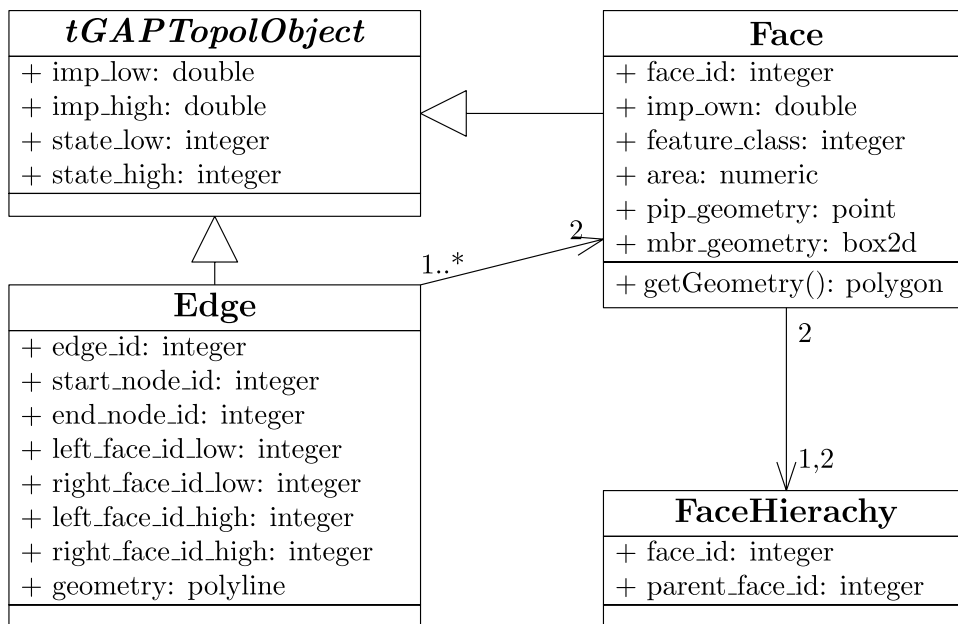
smooth merging shown at all. For example, if the consecutive frames are Fig. 4b1, b2, and b3, then users can only see discrete merging. In contrast, if the consecutive frames are Fig. 4c1, c7, and c2, then users can see one frame of ongoing expansion. As a result, the merging expansions are visually smoother when there are more merging operations processed simultaneously.

When merging simultaneously, we require that the area objects involved in different merging events of the same step must not be neighbors. This requirement 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 pair of areas of a merging event, as well as their neighbors. These areas become *blocked areas*. The areas are *free* if they are not blocked yet. We develop a greedy algorithm to find the simultaneous merging events for each step in the “A greedy Algorithm” section. Then, we integrate the events into the tGAP database tables (section “Integrating the Simultaneous Events into the tGAP Database Tables”), followed by integrating the events into the SSC (section “Integrating the Simultaneous Events into the SSC”). In the “Snapping to a Valid State” section, we show how to snap the zooming to some valid states to avoid that the merging animation stops halfway. In the “Animation Duration of a Step” section, we define the animation duration of zooming from one state to another state. Note that the steps of the “A greedy Algorithm,” “Integrating the Simultaneous Events into the tGAP Database Tables,” and “Integrating the Simultaneous Events into the SSC” sections are done in a preprocess, and the final results are saved in some files. Then, the files are sent to the client on request when a user is browsing the map, where the steps of the “Snapping to a Valid State” and “Animation Duration of a Step” sections are done in real time.

A Greedy Algorithm

We use a greedy algorithm to find the simultaneous merging events for all steps. The merging events will be stored as records in the tGAP database tables (see Fig. 6). Some instances of the tables are shown in Table 1 of the “Integrating the Simultaneous Events into the

Fig. 6 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()*. The face geometry is not stored because we want to avoid redundancy, as the edges already stored the sequences of the points



tGAP Database Tables” section. To compose a merging event, we wish to merge the least important area into its most compatible neighbor. 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 1 shows our implementation of computing the weight values and the class similarities.

Figure 7 shows the flowchart of our greedy algorithm. The process starts with a detailed map of area objects. The map is denoted by M_s , where state s is 0 at this point. Parameter r_{simul} specifies the proportion (i.e., percentage, when multiplied by 100) of area objects that are expected to be merged simultaneously. As a value of percentage, r_{simul} is in the range from 0 to 100%, which means $r_{simul} \in [0, 1]$. We denote by $|M_s|$ the number of M_s ’s area objects. If there is more than one area ($|M_s| > 1$), then the algorithm finds merging events for a new step. In other words, in each iteration when we have $|M_s| > 1$, a set of merging event for a step will be defined. We first compute the number of areas that we expect to merge by

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

where expression $\lceil x \rceil$ returns the ceiling of x , which is the smallest integer greater than or equal to x . The ceiling function guarantees $n_{target} \geq 1$. That is to say, the greedy algorithm finds at least one merging event for each step.

When $n_{target} > 1$, however, the greedy algorithm cannot always find n_{target} merging events because some areas may be blocked (also see Fig. 8). Therefore, we use variable n_{event} to represent the number of events that are actually found for a step.

In Fig. 7, the dashed rectangle marks the process of finding merging events for a single step. If the process has not found n_{target} events yet ($n_{event} < n_{target}$) and there are still free areas, then the process continues looking for merging events. In detail, the greedy algorithm selects the least important area, a_{least} , from the free areas. Then, the algorithm finds a_{least} ’s most compatible neighbor a_{nbr} .

- If area a_{nbr} is also free, a merging event has been found, consisting of areas a_{least} and a_{nbr} . Consequently, the number of events, n_{event} , increases by 1. Then, a_{least} , a_{nbr} , and their neighbors are blocked (see Fig. 8a). Note that if an area shares only one vertex with a_{least} and/or a_{nbr} , that area will not be blocked.
- If area a_{nbr} is not free, then it must have been 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.

Now, let us move back to the start of finding merging events for a single step, that is, the condition “ $n_{event} < n_{target}$ and free areas exist.” If we have found n_{target} events or there is no free area anymore, then finding merging events of the step finishes. The greedy algorithm merges all the pairs of areas of the found events to generate new areas, frees all the blocked areas, increases state s by value n_{event} , and creates map M_s based on the new areas and the freed areas. Then, finding merging

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

f_{id}	s_{low}	s_{merge}	s_{high}
(a) The face table of the single merging shown in Fig. 4b1–b7			
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 Fig. 4c1–c7			
1	0	0	2
2	0	0	2
3	0	<u>2</u>	4
4	0	4	5
5	0	2	4
6	0	<u>0</u>	2
7	0	<u>0</u>	2
8	2	2	4
9	4	5	6
10	2	2	4
11	4	4	5
12	5	5	6
13	6	—	—

events for the next step starts. This loop of finding completes when there is only one area left on the map ($|M_s| = 1$). Figure 4c1–c5 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).

The ideal situation to apply our method is that small areas distribute evenly and the areas do not have holes. The reason is as follows. We wish to use a rather large simultaneous parameter r_{simul} so that many events can share their merging durations. However, if the small areas do not distribute evenly, then some small areas will be blocked and kept while some larger areas will be merged, which is unreasonable. If an area has many holes, where each hole is filled with an area, then each step a hole merging into its surrounding area will forbid other holes to merge into it. This situation results in that some holes are merged until the scale is very

small. A typical example is that a built-up area contains many buildings as holes.

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 (see Fig. 6 the UML diagram of the tables). Note that both class Face and class Edge inherit the attributes from superclass tGAPTopoObject. His face table contains columns *face_id*, *imp_low*, *imp_high*, *imp_own*, *feature_class*, *area*, and *mbr_geometry*. 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 (i.e., an area object) should appear or disappear (the same is done also for the edge table). Values s_{low} and s_{high} of relevant areas in a step are assigned when all the pairs of areas are merged (see the step in Fig. 7). In detail, all those pairs of areas that are merged have $s_{high} = s + n_{event}$, and the generated areas will have $s_{low} = s + n_{event}$. Tables 1a and 1b shows the two new columns with column *face_id*. A face appears as a result of merging two faces during zooming out when the slicing arrives at the face’s low state. When the slicing arrives at its high state, the face should have been merged with another area. Comparing between the tables of single merging (Fig. 4b1–b7) and simultaneous merging (Fig. 4c1–c5), one can 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). Correspondingly, the s_{low} value of face 8 is changed from 1 to 2 (see Table 1). Note that the face IDs are defined in Fig. 4. Similarly to the face tables, the columns and records of both the edge table and the face-hierarchy table will be changed accordingly.

Integrating the Simultaneous Events into the SSC

Recall that we merge a pair of areas by expanding the winner over the loser. The Eater of Šuba et al. (2014) is used to triangulate the loser and to tilt the triangles. Then, the tilted triangles are integrated into the SSC (see Fig. 5) 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 $s_{merge} = s_{high} - 1$ (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 over the combined place of the pair of areas. Take Fig. 4 as an example, area 1 is merged into area 2 (Fig. 4b1), and area 8 is generated to take over the combined place (Fig. 4b2). The tilted triangle is the one that spans from $z = 0$ to $z = 100$ in

Fig. 7 The flowchart of our greedy algorithm. This algorithm finds the merging event for all the steps. The dashed rectangle marks the process of finding merging events for a single step

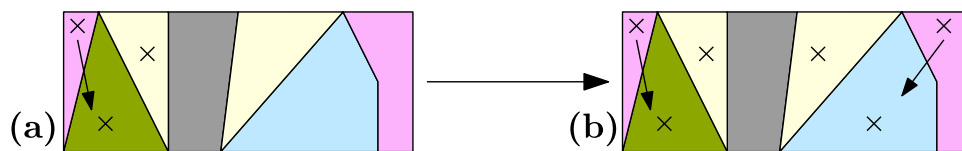
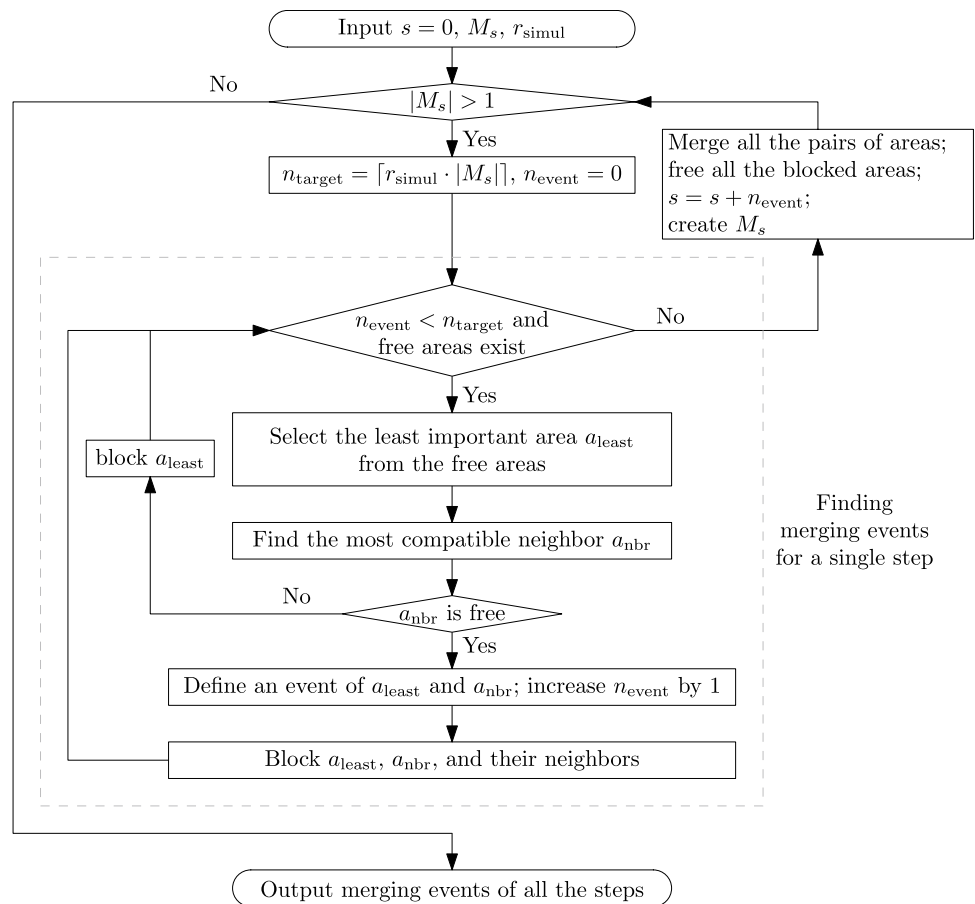


Fig. 8 The process of finding simultaneous merging events for a single step. (a) From all the free areas, the least important one is selected to merge into its most compatible neighbor. Then, the two areas and the surrounding areas are blocked (marked by the crosses).

(b) Next, the least important area from the remaining free areas is selected to merge with its most compatible neighbor, and the relevant areas are also blocked.

Fig. 5a. In Table 1a, the s_{low} value of area 8 is 1, which is the s_{high} value 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 $s_{merge} = s_{high} - n_{event}$. The reason is that if the n_{event} events would happen sequentially (i.e., single merging), then the first of the n_{event} events would start at state $s_{high} - n_{event}$, the second would start at state $s_{high} - n_{event} + 1$, and so on. Now that all the n_{event} events share their merging durations, all of them can start at state $s_{high} - n_{event}$. 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.

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 loser’s ceiling 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:

$$n_{\text{event}}(s_{\text{high}}, \mathbf{s}_{\text{high}}) = \frac{\sum_{s \in \mathbf{s}_{\text{high}}} \{s = s_{\text{high}}\}}{2} \tag{2}$$

where vector \mathbf{s}_{high} denotes the 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:

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

Take the case of Table 1b for example, we have $\mathbf{s}_{\text{high}} = [2, 2, 4, 5, 4, 2, 2, 4, 6, 4, 5, 6]$, $n_{\text{event}}(4, \mathbf{s}_{\text{high}}) = 2$, and $s_{\text{merge}}(4, \mathbf{s}_{\text{high}}) = 2$. Therefore, there are two merging events finishing at state 4, i.e., the event of merging area 3 into area 8 and the event of merging area 5 into area 10 (also see Fig. 4c2 and c3). The merging animation takes place from state 2 to state 4. This merging can also be observed from the two tilted triangles spanning from $z = 200$ to $z = 400$ in Fig. 5b. In merging sequence of Fig. 4b1–b7, 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 Fig. 5a), 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 Fig. 5a). As a result, the animation duration of merging area 3 into area 8 of sequence Fig. 4c1–c5 is almost twice as that of sequence Fig. 4b1–b7. We say *almost* because the animation duration is also dependent on the state value of the map (see the “Animation Duration of a Step” section).

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 stopping halfway. Take the sequence of Fig. 4c1–c5 for example, the merging animation is allowed to stop at Fig. 4c1 or c2, but not at Fig. 4c6 or c7. In this example, state 1 is invalid because some merging operations have not completed. Here, the list of valid states is $\mathbf{s}_{\text{valid}} = [0, 2, 4, 5, 6]$. In order to snap to one of the valid states, we have to communicate them to the client side. There are multiple options. The simplest one assumes that, the greedy algorithm can always find the n_{target} number of events in all steps. In that case, we just need to communicate the number of areas and the ratio r_{simul} . However, this assumption may be incorrect in case of high value ratios (e.g., $r_{\text{simul}} > 0.01$). We then have to communicate the valid states by sending them explicitly. Because this list may get rather large, we only send exceptions (see Appendix 2 for more details). As a result, the list of valid states $\mathbf{s}_{\text{valid}}$ is generated on the client side.

According to how much a user has zoomed, the target scale, say, $1 : S_t$ can be computed. Huang et al. (2016) suggested that the average density of the base 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 $1 : S_t$ is A_{real}/S_t^2 . In order to keep the density, we require

$$\frac{N_b}{A_{\text{real}}/S_b^2} = \frac{N_b - E_t}{A_{\text{real}}/S_t^2} \tag{4}$$

where parameter $N_b = |M_0|$ 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 processed from the base map to the map at scale $1 : S_t$ (in this case, an event is that an area is merged into another one). Eq. 4 yields:

$$E_t = N_b \left(1 - \frac{S_b^2}{S_t^2} \right) \tag{5}$$

In our example of Fig. 4c1–c5, if event number $E_t \leq 0$, the base map should be presented; if $E_t \geq 6$, the map with the final single area should be presented. Otherwise, if $0 < E_t < 6$, we snap event number E_t to a value (measured in events) of list $\mathbf{s}_{\text{valid}}$, which is denoted by $E_{t,\text{snap}}$. The snapping also depends on if the map user is zooming in or out. For zooming in, $E_{t,\text{snap}}$ is the closest value in $\mathbf{s}_{\text{valid}}$ that is smaller than or equal to E_t . For zooming out, $E_{t,\text{snap}}$ is the closest value in $\mathbf{s}_{\text{valid}}$ that is larger than or equal to E_t . This way of snapping prevents $E_{t,\text{snap}}$ from being the same value before zooming; otherwise, the map will stand still if the map user zooms only a little bit. The scale denominator corresponding to event number $E_{t,\text{snap}}$ can be computed by

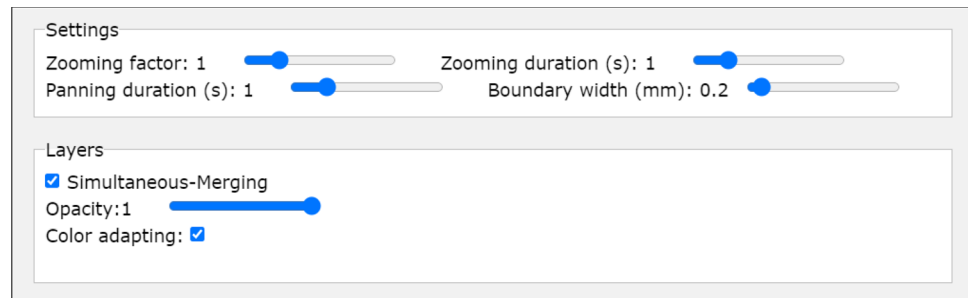
$$S_{t,\text{snap}} = S_b \sqrt{\frac{N_b}{N_b - E_{t,\text{snap}}}} \tag{6}$$

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

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

Fig. 9 Our panel of settings. Among others, one can set how much to zoom when scrolling the mouse wheel and set the zooming duration



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 Fig. 9). 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 (Suba (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

$$t_{\text{simul}} = t_{\text{single}} \frac{N_{\text{event}}}{n_{\text{step}}} \quad (7)$$

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 Eq. 7 is shown in Appendix 3.

Case Study

We have stored the result of the tGAP as a set of tables (see the “Integrating the Simultaneous Events into the tGAP Database Tables” section) 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, 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 the interactive map with smooth zooming can be output by slicing the SSC.

Figure 10 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 Eq. 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 arrives at 1,150,565 according to Eq. 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 $r_{\text{simul}} = 0.001, 0.01, \text{ and } 0.1$ (see the “A greedy Algorithm” section), which are independent of the size of the map dataset. Fig. 11 shows two examples of our web map when simultaneous parameter $r_{\text{simul}} = 0.01$.⁵

Some statistics of the results when simultaneous parameter $r_{\text{simul}} = 0.001, 0.01, \text{ or } 0.1$ 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 the “A greedy Algorithm” section, for

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

³ Figure 10a is obtained from article *12 Most Beautiful Regions in the Netherlands*; see <https://www.touropia.com/regions-in-the-netherlands-map/>, accessed on 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 on January 15, 2020.

⁵ The three versions of the map can be browsed online at <https://pengdlzn.github.io/webmaps/2021/10/merge/>.

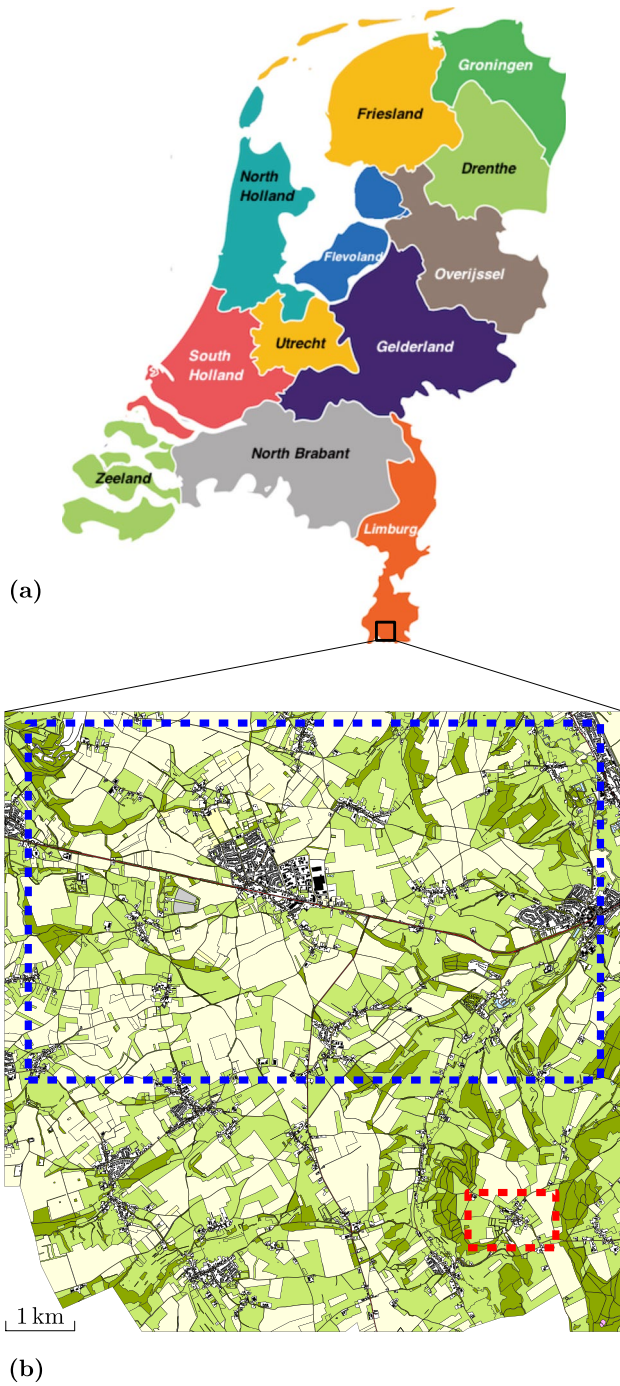
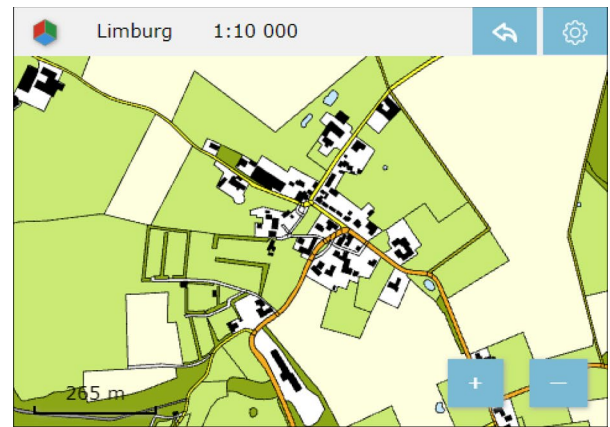
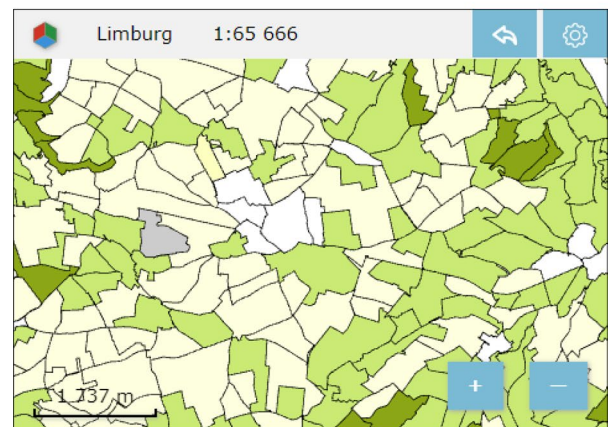


Fig. 10 The data used in our case study. (a) A map of the Netherlands. (b) The topographic map used in our case study. There are 13,238 area objects. The map is for scale 1: 10,000

each merging step we iteratively select the least important area and its most compatible neighbor to define a merging event; then, we block the two areas and their neighbors. Sometimes, a least important area is already blocked because of the previously found events. This situation happens 2, 714



(a)



(b)

Fig. 11 Two examples of our web map with different scales. (a) A part of the base map. The place is marked by the red dashed rectangle in Figure 10b. (b) An overview map. The place is marked by the blue dashed rectangle in Figure 10b. The overview map is generated from the base map by simultaneous merging with parameter $r_{simul} = 0.01$

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

Table 2 Some statistics when different simultaneous parameters area used. 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

the previously found events. This case happens 1, 383 times in total for all the steps when $r_{\text{simul}} = 0.01$ (see column $n_{\text{nbr_blocked}}$ 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 $r_{\text{simul}} = 0.01$ or $r_{\text{simul}} = 0.001$. Only when pushing beyond the limit (e.g., $r_{\text{simul}} = 0.1$), we cannot reach the target number of events in a step (and we need correction information to compute the actual number of found events). When the target number of events cannot be met, one could also question the cartographic quality because there is hardly any free choice when generalizing.

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

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

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 only a few valid states or scales), whereas we would like to have representations at nearly arbitrary scales.

Comparing to the map based on the single merging, the map based on the simultaneous merging indeed provides smoother zooming. We set zooming factor $f_{\text{zoom}} = 1$ and zooming duration $t_{\text{zoom}} = 1s$ (see the “Animation Duration of a Step” section). The map based on the single merging gives the impression of discrete scale transition, where it is difficult to see a winner expands over a loser.⁶ The reason is that the merging happens too fast, so the time for animation available is too short. This is also the case when we use simultaneous parameter $r_{\text{simul}} = 0.001$.⁷ We get the feeling of smooth merging when $r_{\text{simul}} = 0.01$.⁸ When $r_{\text{simul}} = 0.1$, the smooth merging is already obvious.⁹

Figure 12 shows a problem when we use simultaneous parameter $r_{\text{simul}} = 0.1$. That is, some tiny and relatively unimportant areas stay until the scale is quite small, where they should be merged when the scale is larger. This problem has been mentioned in the “A greedy Algorithm” section. The reason of the problem is that there are many buildings in

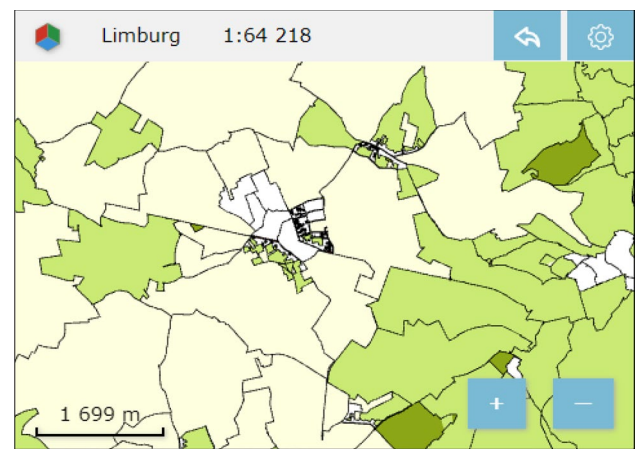


Fig. 12 Some tiny areas should be merged when the scale is larger, where the simultaneous parameter is 0.1

the middle of the figure. When the buildings share the same surrounding area, they become its holes. In each step, only one of the buildings can be merged into the surrounding area because of the blocking. In the meantime, the areas at other places of the map merge relatively fast because we expect to merge 10% of the areas in each step. Fortunately, we would not need to use such a big simultaneous parameter in reality.

Concluding Remarks

Conclusion

This paper has examined the simultaneous processing of generalization operations, using the merging operation as a case study. The purpose of having simultaneous generalization operations is to provide smoother zooming experience later on (compared to the pure sequenced individual generalization events) so that users can better keep track of their interested objects. This paper developed a greedy algorithm to find simultaneous events of merging area objects. The simultaneous events were integrated into the tGAP and the SSC to nicely visualize the merging animations. To guarantee that the merging animations are completely shown while zooming, we managed to snap zooming operations to valid states. This paper also presented a recipe to define the animation duration of an event. According to our case study, the simultaneous merging indeed provides smoother zooming than the single merging.

Future Work

Many topics related to this research need to be studied further. Our case study with 13, 238 area objects demonstrated the

⁶ See the web map at <https://pengdlzn.github.io/webmaps/2021/10/merge/limburg-single-merging>.

⁷ See the web map at <https://pengdlzn.github.io/webmaps/2021/10/merge/limburg-0.001.html>.

⁸ See the web map at <https://pengdlzn.github.io/webmaps/2021/10/merge/limburg-0.01.html>.

⁹ See the web map at <https://pengdlzn.github.io/webmaps/2021/10/merge/limburg-0.1.html>.

efficiency of our prototype. 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. This strategy also allows progressive transfer of data (van Oosterom and Meijers 2014). Furthermore, 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 is able to handle a map with arbitrary number of area objects.

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. (2020b)).

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 neighboring 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 the events is consistent 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 greedy algorithm and the algorithm considering even distribution.

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 Haurert 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 of Haurert and Sester (2008) or the skeleton obtained from a CDT (Ai and van Oosterom 2002); Meijers et al. (2016). In addition to area features, we also need to support line features (e.g., roads, river, rail). In order to apply appropriate generalization operators

for a certain scale, we need to extend and implement the framework to guide the 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. The 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.

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 can map users benefit from our technique. We will conduct some usability tests based on the experience of Section 6.7, Šuba (2017), and Midtbø and Nordvik (2007). Another future work is to find optimal simultaneous parameters for different kinds of datasets.

Appendix 1. 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 the "A greedy Algorithm" section). 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. In table *class_weights*, field code stores the codes of the classes, and field weight stores the class weight. Table *class_comp_matrix* stores the distances and the compatibility values between the classes. The distance is defined based on a tree similar to Fig. 3. The compatibility value is between 0 and 1. If two areas are with the same class, then the compatibility value is 1.

```

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();
```

Appendix 2. Communicate valid states

The “Snapping to a Valid State” section shows how to snap to only valid states to avoid halfway 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 Fig. 4c1–c5 (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 Eq. 1, our target is to process three events simultaneously ($n_{target,1} = 3$). However, only two events can be processed in step 1 because some areas are blocked (see Fig. 8b). Therefore, we have $n_{event,1} = 2$. We require that the low state is $s_{low,1} = 0$ for the first step. Then, the s_{high} value can be computed by

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

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

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

Table 3 Some information of the merging sequence shown in Table 1b. Column n_{area} shows the number of areas at the beginning of a step

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

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

where variable $n_{area,i}$ denotes the number of the areas at the low state of step i . Furthermore, the state-low value of step $i + 1$ (i.e., $s_{low,i+1}$) is the same as the state-high value of step i (i.e., $s_{high,i}$). Again, the target number of simultaneous events (i.e., $n_{target,i+1}$) is computed by Eq. 1, the number of actual simultaneous events is obtained from the greedy algorithm, and the state-high value (i.e., $s_{high,i+1}$) is computed by Eq. 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 areas, and the simultaneous parameter will be sent to the client side.

B.2 On the client side

When a user accesses our web map, the client side receives the exception list, the number of areas, and the simultaneous parameter from the server side. Starting from step $i = 1$, the client side checks if the step is in the exception list. If so, the number of events associated with step i , from the list, is assigned to $n_{event,i}$; if not, value $n_{target,i}$ is computed by Eq. 1 and assigned to $n_{event,i}$. As a result, the client side has the n_{event} values (see column n_{event} in Table 3). By accumulating the n_{event} values, the client side obtains the list of valid states $s_{valid} = [0, 2, 4, 5, 6]$.

Appendix 3. Animation duration of an event

The general idea is as follows. The amount of scale change is based on the zooming factor. The scale change influences the number of events. According to the number of events, we

can compute the number of steps. Finally, all the steps in a zooming duration have the same merging time.

Let f_{zoom} be the zooming factor, and let t_{zoom} be the zooming duration. Let $1 : S_{t,snap}$ be the snapped scale before the zooming operation, and let $E_{t,snap}$ be the number of events processed from the base map. Let $1 : S_o$ be the zoomed out scale (not snapped yet). For zooming out, we define the relationship between the two scale denominators as

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

For scale $1 : S_o$, the number of events that should be processed from the base map is

$$E_o = N_b \left(1 - \frac{S_b^2}{S_o^2} \right).$$

This equation is derived from Eq. 5. As there may be no valid state corresponding to E_o , we snap the map to a valid state (see the “[Snapping to a Valid State](#)” section), and we have a snapped value $E_{o,snap}$. Then, the scale denominator $S_{o,snap}$ is computed by Eq. 6. According to Eq. 5, we have merged $E_{t,snap}$ areas when arriving at scale $1 : S_{t,snap}$. The event number of zooming out from scale $1 : S_{t,snap}$ to scale $1 : S_{o,snap}$ is

$$N_{event} = E_{o,snap} - E_{t,snap}. \tag{10}$$

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

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

That is to say, the $E_{o,snap} - E_{t,snap}$ 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

$$t_{single} = \frac{t_{snap}}{N_{event}} = \frac{t_{zoom}}{E_o - E_{t,snap}}. \tag{11}$$

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 $r_{simul} \cdot |M_s|$ of Eq. 1 always returns an integer, then we do not need the ceiling function of Eq. 1 (if we are not that lucky, the value of n_{step} will be slightly different). We have

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

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

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

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

$$t_{simul} = \frac{t_{snap}}{n_{step}}, \tag{12}$$

which is also the animation duration of each of the simultaneous events. Putting Eqs. 11 and 12 together, we have

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

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 $S_{o,snap}$ to scale S_t , we have

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

which is the inverse function of Eq. 9. We will be able to snap to scale $1 : S_{t,snap}$. 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.

Acknowledgements We thank Radan Šuba for partly creating the data used in our case study.

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

Declarations This research work is carried out in compliance with transparency, moral values, honesty, and hard work. No human participation or animals are involved in this research work.

Conflict of Interest The authors declare no competing interests.

Ethics Approval As per the literature review, this is neither a repetition of any work nor copied key data from other’s work. The methodology, findings, and conclusions made here belong to original research work as per our knowledge and belief.

Informed Consent Every step of processing for publication informed to all co-authors of this paper at the earliest, and everything is carried out with collective decision and consent.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated

otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

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. <https://doi.org/10.1109/TPAMI.2012.120>
- 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. https://doi.org/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. https://doi.org/10.1007/978-3-540-72385-1_9
- Cheng T, Li Z (2006) Toward quantitative measures for the semantic quality of polygon generalization. *Cartographica* 41(2):487–499. <https://doi.org/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. <https://doi.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. <https://doi.org/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. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.184.3303>. Accessed 22 Apr 2023
- Haunert JH, Sester M (2008) Area collapse and road centerlines based on straight skeletons. *GeoInformatica* 12(2):169–191. <https://doi.org/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. <https://doi.org/10/c8v8s2>
- 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. <https://doi.org/10.1016/j.isprsjprs.2015.11.011>
- 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). <https://doi.org/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. <https://doi.org/10.1016/B978-0-444-70467-2.50011-4>
- 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). <https://doi.org/10.3390/ijgi6080242>
- Li J, Li X, Xie T (2017b) Morphing of building footprints using a turning angle function. *ISPRS International Journal of Geo-Information* 6(6). <https://doi.org/10.3390/ijgi6060173>
- Li J, Liu P, Yu W, Cheng X (2018) The morphing of geographical features by fourier transformation. *PLOS ONE* 13(1):1–13. <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. <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. https://doi.org/10.1007/978-3-642-19789-5_17
- Meijers M (2011b) Variable-scale geo-information. phdthesis, Delft University of Technology. http://www.gdmc.nl/publications/2011/Variable-scale_Geo-information.pdf. Accessed 22 Apr 2023
- 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. <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*. <https://doi.org/10.5194/isprs-archives-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. <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. <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. <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. <https://doi.org/10.5311/JOSIS.2017.15.379>
- 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. https://icaci.org/files/documents/ICC_proceedings/ICC2009/html/refer/19_5.pdf. Accessed 22 Apr 2023
- 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. <https://doi.org/10.1201/9780203869352.ch18>
- Peng D (2019) An optimization-based approach for continuous map generalization. PhD thesis, University of Würzburg. <https://doi.org/10.25972/WUP-978-3-95826-105-1>

- 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., <https://doi.org/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. 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). https://kartographie.geo.tu-dresden.de/downloads/ica-gen/workshop2013/genemappro2013_submission_6.pdf. Accessed 22 Apr 2023
- 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, 10/c5kh
- Peng D, Meijers M, van Oosterom P (2020a) Multi-layer vario-scale web map comparer with dynamic transitions and visual analytical tool. In: Proc. 23rd ICA workshop on generalisation and multiple representation (ICAGM), Delft, The Netherlands, pp 1–8
- Peng D, Wolff A, Haunert JH (2020b) Finding optimal sequences for area aggregation—A* vs. integer linear programming. *ACM Transactions on Spatial Algorithms and Systems* 7(1):1–40. <https://doi.org/10.1145/3409290>
- Read P, Meyer MP (2000) Restoration of motion picture film. Elsevier. <https://www.elsevier.com/books/restoration-of-motion-picture-film/read/978-0-08-051619-6>. Accessed 22 Apr 2023
- Regnauld N (2003) Algorithms for the amalgamation of topographic data. In: Proc. 21st International cartographic conference, Durban, South Africa, pp 222–234
- Sester M (2005) Optimization approaches for generalization and data abstraction. *International Journal of Geographical Information Science* 19(8–9):871–897. <https://doi.org/10.1080/1365881050161179>
- Shea KS, McMaster RB (1989) Cartographic generalization in digital environment: when and how to generalize. In: Proc. Auto-Carto 9, American Congress on Surveying and Mapping, Fall Church, pp 56–67
- 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. <https://doi.org/10.1016/j.compenvurbsys.2019.01.009>
- Shen Y, Ai T, Li J, Huang L, Li W (2020) A progressive method for the collapse of river representation considering geographical characteristics. *International Journal of Digital Earth* 13(12):1366–1390. <https://doi.org/10.1080/17538947.2020.1715495>
- 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. <https://doi.org/10.1080/136588197242374>
- Šuba R (2017) Design and development of a system for vario-scale maps. phdthesis, Delft University of Technology, <https://doi.org/10.7480/abe.2017.18.1877>
- 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, https://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), <https://doi.org/10.3390/ijgi5080145>
- 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. 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). https://kartographie.geo.tu-dresden.de/downloads/ica-gen/workshop2017/genemr2017_paper_1.pdf. Accessed 22 Apr 2023
- van Kreveland M (2001) Smooth generalization for continuous zooming. In: Proc. 5th ICA workshop on generalisation and multiple representation (ICAGM). Beijing, China. <http://www.staff.science.uu.nl/~kreve101/papers/smooth.pdf>. Accessed 22 Apr 2023
- 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, <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. <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. <https://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. https://doi.org/10.1007/978-3-319-00203-3_4
- 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. www.gdmc.nl/oosterom/sdh98.pdf. Accessed 22 Apr 2023
- van Smaalen J (2003) Automated aggregation of geographic objects. phdthesis, Wageningen University, The Netherlands
- 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 Kreveland 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. https://doi.org/10.1007/3-540-63818-0_5