Continuous Generalization of Administrative Boundaries Based on Compatible Triangulations

Dongliang Peng, Alexander Wolff and Jan-Henrik Haunert

Abstract Continuous generalization aims to produce maps at arbitrary scales without abrupt changes. This helps users keep their foci when working with digital maps interactively, e.g., zooming in and out. Topological consistency is a key issue in cartographic generalization. Our aim is to ensure topological consistency during continuous generalization. In this paper, we present a five-step method for continuously generalizing between two maps of administrative boundaries at different scales, where the larger-scale map has not only more details but also an additional level of administrative regions. Our main contribution is the proposal of a workflow for generalizing hierarchical administrative boundaries in a continuous and topologically consistent way. First, we identify corresponding boundaries between the two maps. We call the remaining boundary pieces (on the larger-scale map) *unmatched* boundaries. Second, we use a method based on so-called *compatible triangulations* to generate additional boundaries for the smaller-scale map that correspond to the unmatched boundaries. Third, we simplify the resulting additional boundaries. Fourth, we determine corresponding points for each pair of corresponding boundaries using a variant of an existing dynamic programming algorithm. Fifth, we interpolate between the corresponding points to generate the boundaries at intermediate scales. We do a thorough case study based on the provinces and counties of Mainland China. Although topologically consistent algorithms for the third step and the fifth step exist, we have implemented simpler algorithms for our case study.

Keywords Map · Scale · Topological consistency · Dynamic programming · Morphing · Rubber-sheeting · Mainland China

D. Peng (✉) · A. Wolff
Chair of Computer Science I, University of Würzburg, Würzburg, Germany
e-mail: dongliang.peng@uni-wuerzburg.de

A. Wolff
URL: http://www1.informatik.uni-wuerzburg.de/en/staff/wolff_alexander/

J.-H. Haunert
Institute for Geoinformatics and Remote Sensing (IGF),
University of Osnabrück, Osnabrück, Germany
e-mail: janhhaunert@uni-osnabrueck.de

© Springer International Publishing Switzerland 2016

T. Sarjakoski et al. (eds.), *Geospatial Data in a Changing World*,
Lecture Notes in Geoinformation and Cartography,
DOI 10.1007/978-3-319-33783-8_23

1 Introduction

Nowadays people often browse through digital maps on computers or small displays to get geographic information. To understand maps better, users interactively zoom in and out to read maps from different levels. A typical strategy to support zooming is based on a multiple representation database (MRDB). Such a database stores a discrete set of levels of detail (LODs) from which a user can query the LOD for a particular scale (Hampe et al. 2004). A small set of LODs leads, however, to complex and sudden changes during zooming. Since this distracts users, hierarchical schemes have been proposed that generalize a more-detailed representation to obtain a less-detailed representation based on small incremental changes, e.g., the BLG-tree (van Oosterom 2005) for line simplification or the GAP-tree (van Oosterom 1995) for area aggregation. Such incremental generalization processes are represented in data structures that allow users to retrieve a map of any scale. Still, the generalization process consists of discrete steps and includes abrupt changes. Discrete steps can easily cause users to lose their "mental map" during interaction, which is annoying. To support continuous zooming, van Kreveld proposed five ways of gradual changes, which are *moving*, *rotating*, *morphing*, *fading*, and *appearing* (van Kreveld 2001). This is actually a step towards continuous generalization. Comparing to traditional generalization, continuous generalization has the advantage of generating maps without abrupt changes, which is ideal to improve zooming experience. To achieve continuous generalization, Sester and Brenner (2004) suggested simplifying building footprints based on small incremental steps and animating each step smoothly. With the same objective, Danciger et al. (2009) investigated the growing of regions, meanwhile preserving their topology, area ratios, and relative position. The strategy of using two maps at different scales to generate intermediate-scale maps has been studied in multiple representations, e.g., with respect to the selection of roads or rivers (Girres and Touya 2014). Actually, this strategy is the key idea of morphing-based methods for continuous generalization. For instances, several authors have developed methods for morphing between polylines (Cecconi 2003; Nöllenburg et al. 2008; Peng et al. 2013; Schneider and Hormann 2015) and methods for morphing between raster maps (Pantazis et al. 2009; Reilly and Inkpen 2004).

Topological consistency is a property that must be guaranteed in continuous generalization. In this paper, we investigate the problem of generalizing continuously a two-level hierarchical subdivision—from a larger-scale map of administrative boundaries to a smaller-scale map; see Fig. 1 for an example. Our aim is to generate maps at any intermediate scale without topological conflicts by generalizing continuously (going from Fig. 1a–d). Our method consists of the following five steps.

In Step ①, we find corresponding boundaries; see the red boundaries in Fig. 1a and d. We call the remaining boundaries on the larger-scale map *unmatched* boundaries; see the blue boundaries in Fig. 1a. In order to achieve continuous generalization, we *morph* (that is, deform continuously) between the corresponding boundaries (see, e.g., Nöllenburg et al. 2008). The unmatched boundaries must be morphed in

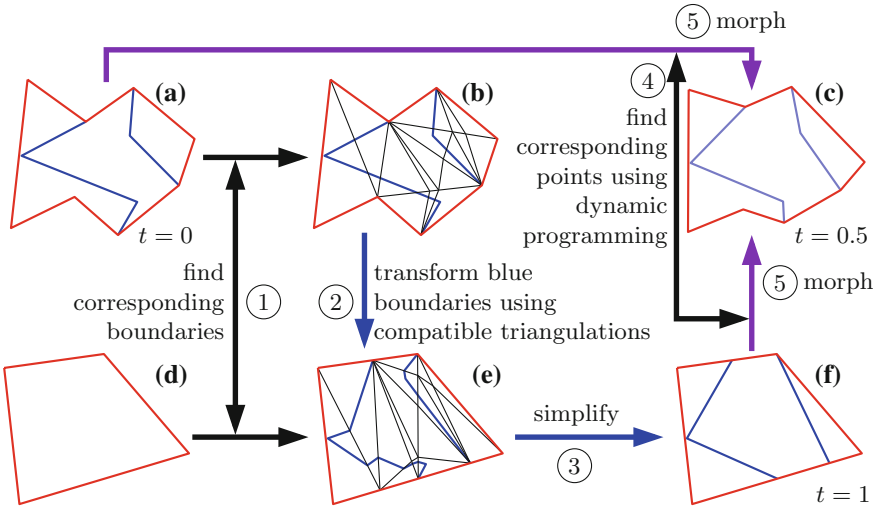

Fig. 1 The framework of our method. **a** The larger-scale administrative boundaries of a region. **d** The smaller-scale administrative boundaries of the same region as in (**a**). **b**, **e** Compatible triangulations constructed for *red* polygons in (**a**) and (**d**), where the *blue* polylines in (**e**) are transformed from the *blue* boundaries in (**b**) based on the compatible triangulations. **f** The *blue* boundaries are simplified from the transformed ones in (**e**). **c** The result of continuous generalization when $t = 0.5$. Continuous generalization is achieved by morphing between (**a**) and (**f**), without the *blue* boundaries displayed in (**f**). The numbers in the *circles* indicate the step orders

a way that is consistent with what we do to the corresponding boundaries. As there is no correspondence for the unmatched boundaries, we generate the corresponding boundaries in Step ② and Step ③.

In Step ②, we transform the blue boundaries based on *compatible triangulations*; see Fig. 1b and e. Two triangulations are *compatible* if there is a correspondence between their vertex sets and the two triangulations are topologically equivalent (Surazhsky and Gotsman 2001). With compatible triangulations, we can transform a blue boundary in one triangulation (see Fig. 1b) to a boundary in the other triangulation by traversing the triangles correspondingly (see Fig. 1e). Therefore, if there is no conflict in one triangulation, then there is no conflict in the other triangulation.

In Step ③, we simplify the transformed (blue) boundaries, using the Douglas–Peucker algorithm (Douglas and Peucker 1973), so that the blue boundaries have the same complexities as the red ones in Fig. 1d; see the blue boundaries in Fig. 1f. We use the simplified boundaries as the correspondences for the blue boundaries in Fig. 1a. On this basis, we are able to generalize continuously by interpolating between each pair (both red pairs and blue pairs) of corresponding boundaries. This process of interpolation is also known as morphing. Since the blue boundaries do not exist on the smaller-scale map, we fade them out during morphing to make them disappear continuously.

In Step ④, we determine corresponding points for each pair of corresponding boundaries using a variant of an existing dynamic programming algorithm (Nöllenburg et al. 2008). Our variant minimizes the distance between corresponding points, which is favorable for Step ⑤.

In Step ⑤, we interpolate (or "morph") between the corresponding points with straight-line trajectories to generate intermediate-scale administrative boundaries.

In order to achieve a topologically consistent workflow, we need to make sure that three of the above steps are topologically consistent, namely Step ②: transform (b–e), Step ③: simplify (e–f), and Step ⑤: morph (a–c–f); see Fig. 1. In this paper, we concentrate on the transformation step (Step ②; Fig. 1b–e) and show how to accomplish this step without introducing topological conflicts. We do not guarantee topological consistency of the other two steps *in our implementation*, but this can be done by integrating topologically consistent methods as proposed by Saalfeld (1999) for Step ③ and Gotsman and Surazhsky (2001) for Step ⑤.

Initially, we tested the rubber-sheeting method of Doytsher et al. (2001) (making all vertices "influential"). We soon noticed, however, that transformed boundaries often cross boundaries of the smaller-scale map. An example is shown in Fig. 2 (which corresponds to Fig. 1e), where the rubber-sheeting method leads to two crossings. Similar problems occured when we applied other variants of rubber-sheeting (such as the one by Haunert 2005).

This is why we decided to search for a more robust method. It turned out that compatible triangulations (Aronov et al. 1993) can, by definition, realize the transformation without introducing topological conflicts. The (quite old) idea is as follows. Suppose that there is a point a_i inside a triangle $\Delta p_1 p_2 p_3$. Then, this point can be expressed as a *unique* convex combination of *simplicial coordinates* $\lambda_{i,1}$, $\lambda_{i,2}$, $\lambda_{i,3}$ (Saalfeld 1985):

$$a_i = \lambda_{i,1} p_1 + \lambda_{i,2} p_2 + \lambda_{i,3} p_3,$$

where $\lambda_{i,1}, \lambda_{i,2}, \lambda_{i,3} > 0$, and $\lambda_{i,1} + \lambda_{i,2} + \lambda_{i,3} = 1$. We can *uniquely* locate the corresponding point b_i in a different triangle $\Delta q_1 q_2 q_3$ by using a_i's simplicial coordinates in $\Delta p_1 p_2 p_3$:

$$b_i = \lambda_{i,1} q_1 + \lambda_{i,2} q_2 + \lambda_{i,3} q_3.$$

This implies that if two distinct points a_i and a_j are in the same triangle, then we are able to locate their corresponding points b_i and b_j in another triangle such that b_i and b_j do not coincide. Moreover, if points a_i and a_j are in two different triangles

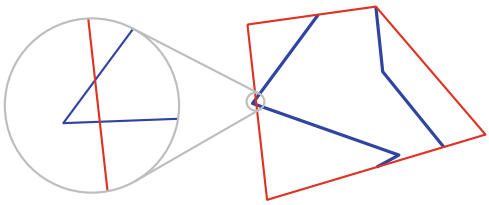

Fig. 2 Crossings caused by the rubber-sheeting method of Doytsher et al. (2001)

of a triangulation, then b_i and b_j can be located in two different corresponding triangles of the compatible triangulation. As a result, once we manage to construct compatible triangulations of two maps, we can transform polylines consistently, which means the transformed polylines will not conflict with each other and will not conflict the boundaries already on the smaller-scale map. Indeed, compatible triangulations have been applied to compare maps from different time periods by hand (Fuse and Shimizu 2004). Instead, we construct compatible triangulations automatically, using an algorithm of Aronov et al. (1993).

Our contributions are as follows. We propose a workflow based on compatible triangulations for generalizing administrative boundaries in a continuous and topologically consistent way; see Sect. 2. We do a thorough case study with the boundaries of the counties and the provinces of Mainland China to test the effectiveness and the efficiency of our method; see Sect. 3. We conclude in Sect. 4.

2 Overall Algorithm

Suppose that we have two maps M_+ and M_- of administrative boundaries of the same area, respectively at a larger scale and a smaller scale. We use a parameter $t \in [0, 1]$ to define the continuous generalization process. We require that the generalization yields exactly M_+ when $t = 0$, M_- when $t = 1$, and that M_+ is being generalized continuously into M_- when t increases from 0 to 1.

The map M_+ is more detailed than M_-, and a region of M_- consists of several regions of M_+. Consequently, a boundary on M_- has a corresponding boundary on M_+, but a boundary on M_+ may not have a smaller-scale correspondence. Thus we need to determine the corresponding boundaries between the two maps, and the leftovers on M_+ are the unmatched boundaries. For a pair of corresponding boundaries, we use a dynamic programming algorithm similar to the algorithm OPTCOR (Nöllenburg et al. 2008) to determine corresponding points. Then, we morph between corresponding points on straight-line trajectories. For an unmatched boundary on M_+, we generate its corresponding boundary by first transforming the boundary using a compatible-triangulation-based method, and then simplifying the transformed boundary with the Douglas–Peucker algorithm (Douglas and Peucker 1973). As a result, we are able to morph between the unmatched boundaries and the generated boundaries (i.e., the simplified ones). We fade the morphing results of the unmatched boundaries out, such that they will disappear when $t = 1$.

The administrative regions are polygons. An administrative region usually shares different parts of its boundary with other administrative regions. These shared parts should be always shared during the continuous generalization process. When generalizing administrative boundaries, we should avoid processing a shared boundary twice. Therefore, we prefer to treat administrative boundaries as a set of consecutive polylines instead of polygons when it is possible. For the input of two maps, we first preprocess them to obtain the correspondences between the polylines.

2.1 Preprocessing

Given polygons on map M_+ and map M_-, the preprocessing consists of three steps. Note that if the inputs are boundaries of the polygons, i.e., polylines, we can easily generate the polygons based on a technique that is known as doubly-connected edge list (Berg et al. 2008). First, we make a copy of the polygons on M_+ and merge the copied polygons according to the polygons on M_-. For each copied polygon, we try overlapping it with every polygon on M_-, and record the one that has the largest overlap area. Then we merge all the copied polygons that record the same polygon on M_-. The merging step can be done more correctly with the help of semantic attributes if possible, but one should be careful about that there may be some separate parts (e.g., enclaves) with ambiguous semantic values.

Second, we obtain matched polylines respectively from the boundaries of the merged polygons and the polygons on M_-. We define that a vertex is an intersection if the vertex has degree at least 3. As the merged polygons and the polygons on M_- have corresponding intersections, we utilize these intersections to find matched polylines. We split the boundaries of the polygons at every intersection, respectively for the merged polygons and the polygons on M_-. Note that two polygons on the same map may share some parts of their boundaries, it is sufficient to take only one copy of the shared parts. Then we match the split boundaries of the two sources to get matched polylines. As we render these matched polylines using red color, we call them *the red polylines on* M_+ and *the red polylines on* M_-. Although there is a data-matching system (Mustière and Devogele 2008) available, we use a simple method to attain the matching. We match the split boundaries according to the overlap areas of their buffers. The buffer-based method works well in our case study as corresponding boundaries have relatively close positions.

Third, we obtain unmatched polylines on M_+. We split the boundaries of the polygons on M_+ at every intersection, then we exclude all the split boundaries that overlay with the matched polylines on M_+. The remained polylines are *the blue polylines on* M_+ as we render them using blue color.

After the preprocessing, we have three types of polylines. The first type consists of the red (matched) polylines on M_+, namely each of these polylines has a corresponding polyline on M_-; see the red polylines in Fig. 1a. The second type consists of the blue (unmatched) polylines on M_+, each of which does not have a corresponding polyline on M_-; see the blue polylines in Fig. 1b. The third type consists of the red (matched) polylines on M_-; see the red polylines in Fig. 1d.

2.2 Morphing a Polyline to Its Corresponding Polyline

For a pair of corresponding polylines, one being a red polyline on M_+ and the other being a red polyline on M_-, we use a variant of the dynamic programming algorithm

OPTCOR of Nöllenburg et al. (2008) to determine corresponding points (possibly injecting additional vertices).

The algorithm OPTCOR models the problem of finding corresponding points as an optimum correspondence problem, with respect to a cost function, between characteristic parts of each pair of corresponding polylines. The characteristic part is either a characteristic point or a subpolyline between a pair of neighbouring characteristic points. OPTCOR considers three cases of a correspondence for a subpolyline, namely, a subpolyline is mapped to a characteristic point, mapped to a subpolyline, or mapped to a merged sequence of subpolylines. We call all the three cases corresponding subpolylines as a point is a degenerate subpolyline and a merged sequence of subpolylines is still a subpolyline. Nöllenburg et al. (2008) describe a way to find characteristic points in order to speed up the dynamic program that computes the correspondences. However, they also mention that simply all vertices can be used as characteristic points, which is what we decided to do in order to obtain high-quality correspondences.

For a pair of corresponding subpolylines, Nöllenburg et al. (2008) define the cost as a combination of three values: (i) the distance between the corresponding points, (ii) the length difference of the pair of subpolylines, and (iii) the changes of the vectors of the corresponding points. Then OPTCOR determines the corresponding points by dynamically "looking back" to combine the at most k last subpolylines into a sequence of subpolylines, while minimizing the cost over all corresponding polylines. Here, k is the user specified look-back parameter, which gives a trade-off between quality and efficiency.

To make the problem simple, our variant considers only the first value as the cost, that is, the distance between corresponding points. Nöllenburg et al. denote this distance by δ_1, but we will simply use δ. In order to compute the cost function, we linearly interpolate between each pair of corresponding subpolylines so that each vertex on one subpolyline has a, possibly injected, corresponding vertex on the other one. The pairs of corresponding vertices subdivide the (sub)polylines into corresponding line segments. Note that a line segment is (part of) an edge of a polyline. The cost of a pair of (whole) polylines is the sum of the costs for each pair of corresponding line segments. The cost for a pair of corresponding line segments can be computed as follows (for a few examples, see Fig. 3).

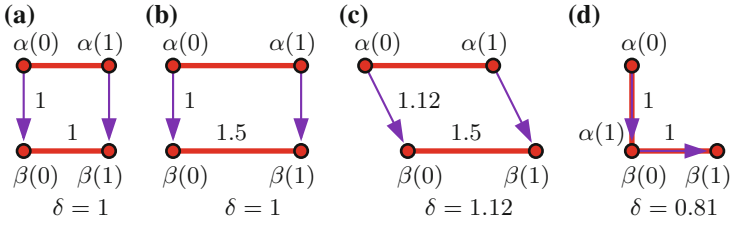

Fig. 3 Examples of computing $\delta(f, g)$. The values in the subfigures represent the lengths of the edges

Let polyline F on M_+ and polyline G on M_- be a pair of corresponding polylines. Let $f = \overline{\alpha(0)\alpha(1)}$ be a line segment on F, and let $g = \overline{\beta(0)\beta(1)}$ be a line segment on G that corresponds to f. Let $\alpha(0) = (x_1, y_1)$, $\alpha(1) = (x_2, y_2)$, $\beta(0) = (x_3, y_3)$, and $\beta(1) = (x_4, y_4)$, which are already known. We define a pair of corresponding points $\alpha(u) \in f$ and $\beta(u) \in g$ as

$$\alpha(u) = ((1-u)x_1 + ux_2, (1-u)y_1 + uy_2),$$
$$\beta(u) = ((1-u)x_3 + ux_4, (1-u)y_3 + uy_4).$$

We define the cost for the correspondence between f and g as the integral over the distances between the pairs of corresponding points (suppose that we move $\alpha(u)$ to $\beta(u)$), that is,

$$\delta(f, g) = \int_0^1 |\beta(u) - \alpha(u)| du,$$

where $|\beta(u) - \alpha(u)|$ is the Euclidian distance between $\alpha(u)$ and $\beta(u)$, and can be represented as $\sqrt{au^2 + bu + c}$. The coefficients a, b, and c depend on the coordinates of $\alpha(0)$, $\alpha(1)$, $\beta(0)$, and $\beta(1)$, as follows.

$$\begin{aligned} a = {} & (x_1 - x_2 - x_3 + x_4)^2 + (y_1 - y_2 - y_3 + y_4)^2, \\ b = {} & -2((x_1 - x_3)(x_1 - x_2 - x_3 + x_4) \\ & + (y_1 - y_3)(y_1 - y_2 - y_3 + y_4)), \\ c = {} & (x_1 - x_3)^2 + (y_1 - y_3)^2. \end{aligned}$$

Let $X = au^2 + bu + c$. We have $a \geq 0$, and since $X \geq 0$ (X is the square of the Euclidian distance), $\Delta = 4ac - b^2 \geq 0$. Note that, if $a = 0$, then $b = 0$. Let

$$\delta(f, g) = \int_0^1 |\beta(u) - \alpha(u)| du = \int_0^1 \sqrt{X} du.$$

Then $\delta(f, g)$ can be computed, according to Bronstein et al. (p. 1064, integrals 241 and 245) (2001), as follows:

$$\delta(f, g) = \begin{cases} \sqrt{c}u|_0^1 & \text{if } a = 0, \\ \frac{(2au+b)\sqrt{X}}{4a}|_0^1 & \text{if } a > 0, \Delta = 0, \\ \frac{(2au+b)\sqrt{X}}{4a}|_0^1 + & \\ \frac{\Delta}{8a\sqrt{a}} \ln(2\sqrt{aX} + 2au + b)|_0^1 & \text{if } a > 0, \Delta > 0. \end{cases}$$

Figure 3 shows a few examples where we compute δ. We obtain the optimum correspondence by minimizing the sum of the distances between corresponding points weighted by the lengths of the line segments, that is,

$$\delta(F,G) = \min_{\substack{\pi\text{: correspondence} \\ \text{between } F \text{ and } G}} \sum_{\substack{f \in F,\, g \in G, \\ \text{where } f \text{ corresponds to } g \text{ in } \pi}} \frac{|f| + |g|}{2} \delta(f,g).$$

Recall that OPTCOR considers three cases of a correspondence for a subpolyline. We find that the first case, i.e., a subpolyline is mapped to a characteristic point, may result in different numbers of vertices on the two polylines. Our major change to OPTCOR is that we remove this first case from the algorithm. This change ensures that the resulting pairs of corresponding polylines have the same numbers of vertices. This property is essential for the construction of compatible triangulations, which is a later step in our workflow. We name our modified version OPTCOR-S, where the letter *S* stands for *Simplified*.

Suppose that there are n_F vertices on F and n_G vertices on G, then OPTCOR-S requires that the look-back parameter k is bounded from below by n_F/n_G and n_G/n_F. Otherwise, there will be at least one segment that corresponds to a vertex. In our experiments, we always use a value of k that produces results with high quality (in the sense of the dynamic-programming algorithm). We morph by interpolating between corresponding points using straight-line trajectories. Figure 4 shows an example with 0.25, 0.5, and 0.75 for t.

Some other algorithms for determining corresponding points can be used (e.g., linear interpolation). We observed that an algorithm that determines corresponding points more carefully yields better results, meaning that the interpolated polylines are more similar to the two sources and it is less likely that crossings are introduced. This is why we decided to modify the algorithm OPTCOR. Some sophisticated algorithms can be considered to define the interpolation trajectories, such as geodesic shortest paths (Bereg 2005), b-morphs (Whited and Rossignac 2011), or a method based on least squares adjustment (Peng et al. 2013). Specifically, it is possible to use compatible triangulations not only for the transformation step (as in our method) but also to ensure the topological consistency in the morphing step (Surazhsky and Gotsman 2001).

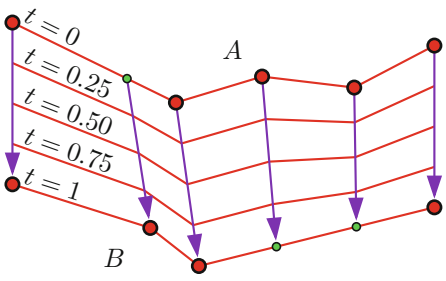

Fig. 4 Morphing a polyline A to its corresponding polyline B. The *arrows* show the moving trajectories of the vertices

2.3 Morphing a Polyline to Its Generated Corresponding Polyline During Fade-Out

For the blue polylines on M_+, morphing them must be consistent with what we do to the red corresponding polylines. To achieve this, we generate their corresponding polylines, that is, blue polylines on M_-. We first transform the blue polylines on M_+ based on compatible triangulations, and then simplify the transformed polylines using the Douglas–Peucker algorithm.

For each pair of polygons correspondingly bounded by the red polylines on M_+ and the red polylines on M_-, we construct a pair of compatible triangulations; see Fig. 1b and e. We call them the *triangulation on* M_+ and the *triangulation on* M_-. The construction of compatible triangulations requires that the two polygons have the same number of vertices. We have ensured this property using OPTCOR-S to determine corresponding points between the red polylines on M_+ and M_-. We use the algorithm of Aronov et al. (1993) to construct compatible triangulations. For the two polygons both with m vertices, we first triangulate them independently. Then we create a regular m-gon, and map the chords of the two triangulations into the regular m-gon and overlay the mapped chords. The mapped chords may cross with each other, which results in some convex faces. We use the crossings as dummy vertices and split the mapped chords. We triangulate each convex face with more than three vertices. To triangulate, we select one vertex and add an edge between this vertex to each of the other vertices, except the two immediate-neighbouring ones. As a result, we have a *combined triangulation*. We map the combined triangulation (including dummy vertices and new edges) back to modify the two original triangulations. After the modification, we have a pair of compatible triangulations of the two original polygons.

With the compatible triangulations, we can transform a blue polyline in the triangulation on M_+ to a polyline on M_-, using the same simplicial coordinates. To guarantee that the transformed polyline traverses exactly the "same" triangles as a blue polyline on M_+, we need to compute the crossings between the blue polyline and the triangulation edges, and also transform these crossings into the triangulation on M_-. Because of the additional crossings, the transformed polylines have higher complexity than the red polylines on M_+. While our aim is to generate polylines that have the same complexity as the red polylines on M_-. Therefore, we simplify the transformed polylines to the target complexity; see Fig. 1f. For a *blue hole* (polygon), we keep at least three vertices to avoid degenerating it into a straight line or a point. We call the simplified polylines the *blue polylines on* M_-.

We use the algorithm OPTCOR-S again to determine corresponding points for each pair of corresponding blue polylines, which are respectively on M_+ and M_-. We use straight-line trajectories to interpolate between corresponding points. As the blue polylines do not exist when $t = 1$, we fade them out during the morphing process. An example is shown in Fig. 1c, where $t = 0.5$.

2.4 Running Time Analysis

We analyze the running time for a pair of polygons correspondingly bounded by the red polylines on M_+ and the red polylines on M_-. We use N to denote the vertex number of the polygon on M_+, n the vertex number of the polygon on M_-, and N' the vertex number of all the blue polylines inside the polygon on M_+. For simplicity, we assume that $O(N') \in O(N)$.

Constructing the compatible triangulations takes $O(N \log N + l)$ time, where $O(l) \in O(N^2)$ is the number of dummy vertices inserted during the construction. Simplifying the transformed polylines, using the Douglas–Peucker algorithm, costs time $O(N(N + l) \log N)$ (Hershberger and Snoeyink 1992). OPTCOR-S takes time $O(k^2Nn)$ to determine corresponding points, where k is the look-back parameter. Note that, outputting a representation at a target zoom level only takes $O(N)$ time (and, hence, is feasible in real time). In fact, we store, for each (possibly injected) vertex v on F, a representation such as $v(t) = (1 - t)v + tw$, where w is the vertex on G that corresponds to v. In our implementation, determining the corresponding points is the by far most time-consuming step.

3 Case Study

We implemented our method based on C# (Microsoft Visual Studio 2010) and ArcGIS Engine 10.1. We ran our case study under Windows 7 on a 3.3 GHz quad core CPU with 8 GB RAM. We measured time consumption by the built-in C# method System.Environment.TickCount.

We tested our method on the administrative boundaries of Mainland China, which are from the National Fundamental Geographic Information System, and based on the projected coordinate system *Krasovsky 1940 Lambert Conformal Conic*; see the provincial map in Fig. 5b. The tested data sets are obtained from the complete data sets of China by removing the only enclave in Gansu province and all the islands. We use a data set of county boundaries at scale 1 : 5,000 k with 493,625 vertices (5, 909 polylines after preprocessing), and a data set of provincial boundaries at scale 1 : 30,000 k with 7,527 vertices (90 polylines after preprocessing). Since we can hardly see the details if we present the whole data set, we only show the results of the Tianjin province in Fig. 5.[1]

Our aim is to generalize continuously from map of counties M_+ to map of provinces M_-. Using the provincial boundaries in Fig. 5e, we are able to distinguish the hierarchies of county boundaries in Fig. 5a, and then find the matched polylines (the red polylines in Fig. 5b and e) and the unmatched polylines (the blue polylines in Fig. 5b); see Step ①.

[1]Interactive animations of some provinces are available at http://www1.pub.informatik.uni-wuerzburg.de/pub/data/agile2016/. (We recommend opening the link with Google Chrome).

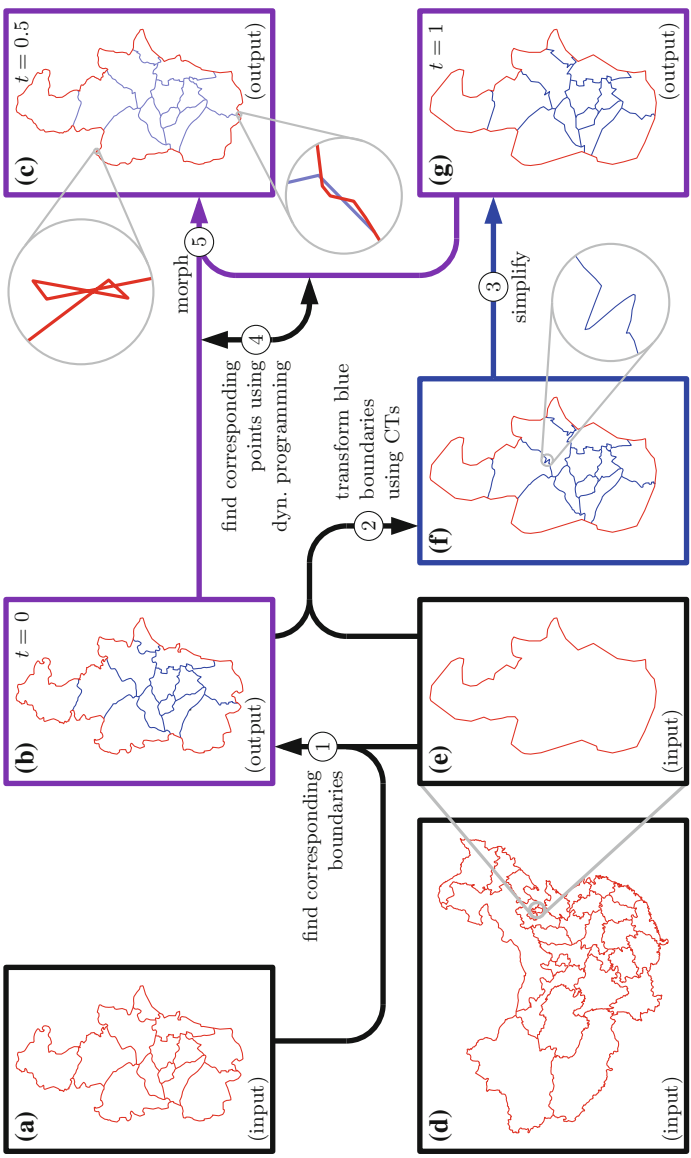

Fig. 5 Case study on administrative boundaries of Mainland China. The numbers in the *circles* indicate the step orders, analogously to Fig. 1. For the sake of legibility, however, we do not display the compatible triangulations. Continuous generalization is achieved by morphing between (**b**) and (**g**). The *blue* boundaries in (**g**) are faded out in the transition from $t = 0$ (in **b**) and $t = 1$ (in **g**)

In Step ② we import the red polylines on M_- into Fig. 5f. Then we use the algorithm OPTCOR-S to determine corresponding points between the red polylines on M_+ and M_-. We construct compatible triangulations for the polygons bounded by the red polylines of Fig. 5b and f so that we can transform the blue polylines on M_+; the transformed polylines are the blue ones in Fig. 5f. During the determination of corresponding points, we used a fact that the 90 red polylines (with 55,533 vertices) on M_+ and 90 red polylines (with 7,527 vertices) on M_- share many vertices (M_- may be generalized from M_+). We split the red polylines of M_+ and M_- into many subpolylines at the shared vertices so that we can compute corresponding points for each pair of subpolylines, which makes the computation much faster. Using a look-back parameter 145, the determination takes 266 s with cost $\sum \delta(F, G) = 125{,}050\,\mathrm{km}^2$, where the look-back parameter 145 is the smallest value that yields the optimum result in the sense of the dynamic programming algorithm. The construction of the compatible triangulations takes 168 s. There is no conflict for the transformed polylines in the whole data set. However, a flaw is that there are zigzags caused by the compatible-triangulation-based transformation, e.g., the one shown in the enlarged figure next to Fig. 5f. We also tested the rubber-sheeting method of Doytsher et al. (2001), which, unfortunately, introduced 39 crossings.

In Step ③, we simplify the polylines, transformed by the compatible triangulations, to the blue polylines on M_- using the Douglas–Peucker algorithm, which takes 29 s. This simplification caused 8 crossings and 2 overlaps. We corrected the 10 conflicts manually. Note that we can avoid these conflicts by implementing a topologically consistent line simplification method, for example, the algorithm of Saalfeld (1999). In Step ④, we use OPTCOR-S again to determine corresponding points between the 5,819 blue polylines (with 438,092 vertices) on M_+ and 5,819 blue polylines (with 58,105 vertices) on M_-. This time there are no shared vertices, and the computation takes about 16.5 h with look-back parameter 203, where this parameter is required by a certain pair of corresponding polylines to guarantee $k \geq n_F/n_G$ (see Sect. 2.2). The cost of the resulting correspondence is $\sum \delta(F, G) = 477{,}185\,\mathrm{km}^2$.

We morph from counties to provinces using straight line trajectories; see Step ⑤. We show the results of our continuous generalization of Tianjin Province in Fig. 6. Exporting the data set of Mainland China with 5, 909 polylines (496,106 vertices) at any intermediate scale takes 45 s, mainly due to the slow creation of features in ArcGIS Engine. Unfortunately, this morphing causes conflicts in the intermediate-scale maps; two examples are shown in the enlarged figures next to Fig. 5c. For an instance, there are in total 41 crossings on the intermediate-scale map of Mainland China when $t = 0.5$. To avoid these crossings, we can use an algorithm which guarantees topological consistency during morphing, e.g., the algorithm of Surazhsky and Gotsman (2001).

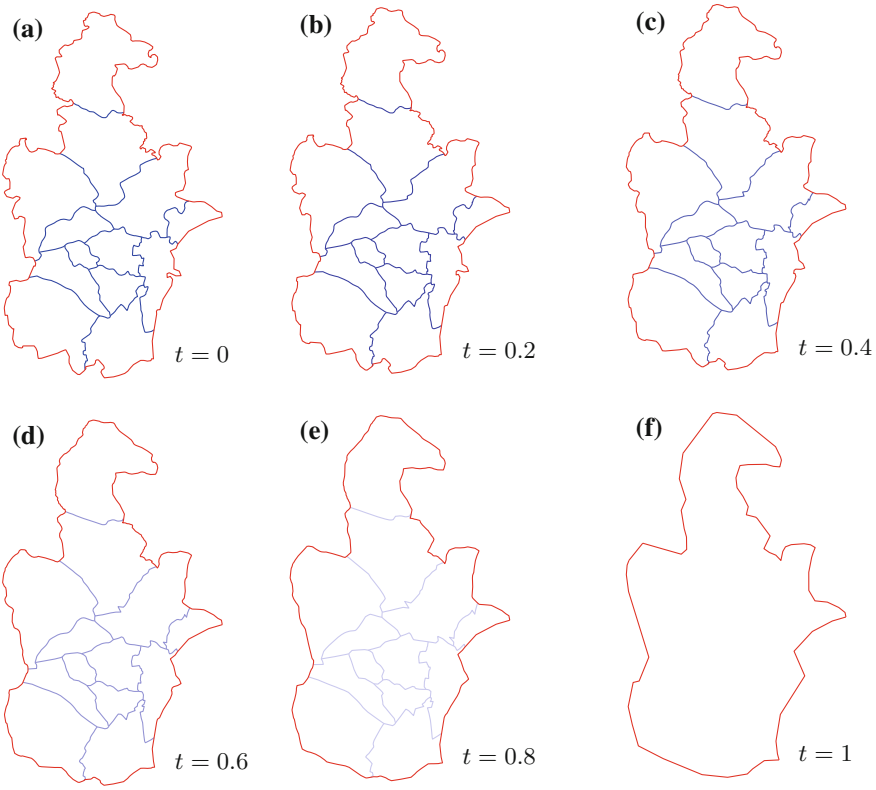

Fig. 6 The continuous generalization of Tianjin province

4 Conclusions

In this paper, we have shown that rubber-sheeting, a popular method for data matching, can yield topologically inconsistent results. Therefore, we proposed a method to generalize continuously administrative boundaries based on compatible triangulations, which apparently have not been used in GIScience before (except by hand Fuse and Shimizu 2004). We have used compatible triangulations to transform unmatched polylines and managed to achieve topologically consistent. Although computing correspondences is slow, the computed result supports real-time interactions (e.g. zooming). By comparing our compatible-triangulation-based method to the rubber-sheeting method, we found that our method tends to yield results with larger distortions. An extreme instance is shown in Fig. 7. To decrease the amount of distortion, one could try constructing compatible triangulations that use the maximum number

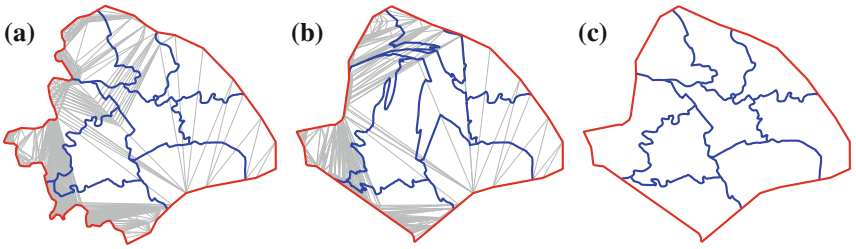

Fig. 7 A comparison of the compatible-triangulation-based method and the rubber-sheeting method for transforming the *blue* polylines on M_+, using the data sets of Shanghai as an instance. **a** M_+ and the compatible triangulations of *red* polylines on M_+. **b** M_-, the compatible triangulations of *red* polylines on M_-, and the transformed polylines (*blue*) by the compatible-triangulation-based method. **c** M_- and the transformed polylines (*blue*) by the rubber-sheeting method of Doytsher et al. (2001)

of chords common to two triangulations. To that end, we could extend the dynamic programming algorithm mentioned by Aronov et al. (1993), [p. 34, bottom]. Whether this actually yields transformation results of higher quality is a question that requires further research.

Moreover, our current implementation of constructing compatible triangulations cannot deal with holes on a smaller-scale map M_-. Babikov et al. (1997) suggested a solution for this.

We used the Douglas–Peucker algorithm to simplify the transformed polylines. As expected, this led to some topological conflicts. To solve this problem, we need a topologically consistent algorithm, e.g., Saalfeld's variant of the Douglas–Peucker algorithm (Saalfeld 1999). In the morphing process, we have used straight-line trajectories to interpolate between corresponding points. Again, these have introduced crossings. In order to guarantee topological consistency in the morphing process, we can use a compatible-triangulation-based algorithm to define the interpolation trajectories, e.g., the algorithm of Surazhsky and Gotsman (2001). With these two replacements, our workflow can generalize two-level hierarchical subdivisions (such as administrative boundaries) in a continuous and topologically consistent way.

Acknowledgments The authors thank Thomas C. van Dijk for proofreading an earlier version of this paper and the China Scholarship Council (CSC) for (partly) supporting this work.

References

Aronov B, Seidel R, Souvaine DL (1993) On compatible triangulations of simple polygons. Comput Geom 3:27–35

Babikov M, Souvaine DL, Wenger R (1997) Constructing piecewise linear homeomorphisms of polygons with holes. In: Proceedings of 9th Canadian Conference Computing Geometry (CCCG'97), pp 6–10

Bereg S (2005) An approximate morphing between polylines. Int J Comput Geom Appl 15(2):193–208

Bronstein IN, Semendjajew KA, Musiol G, Mühlig H (2001) Taschenbuch der Mathematik, 5th edn. Wissenschaftlicher Verlag Harri Deutsch

Cecconi A (2003) Integration of cartographic generalization and multi-scale databases for enhanced web mapping. Ph.D. thesis, Universität Zürich

Danciger J, Devadoss SL, Mugno J, Sheehy D, Ward R (2009) Shape deformation in continuous map generalization. Geoinformatica 13:203–221

de Berg M, Cheong O, Kreveld MV, Overmars M (2008) Computational geometry: algorithms and applications, 3rd edn. Springer-Verlag TELOS

Douglas DH, Peucker TK (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica 10(2):112–122

Doytsher Y, Filin S, Ezra E (2001) Transformation of datasets in a linear-based map conflation framework. Surv Land Inf Syst 61(3):159–169

Fuse T, Shimizu E (2004) Visualizing the landscape of old-time Tokyo (Edo city). In: Gruen A, Murai S, Fuse T, Remondino F (eds) The international archives of the photogrammetry, remote sensing and spatial information sciences

Girres JF, Touya G (2014) Cartographic generalisation aware of multiple representations. In: Duckham M, Stewart K, Pebesma E (eds) Proceedings of 8th international Conference on Geographic Information Science—poster session

Gotsman C, Surazhsky V (2001) Guaranteed intersection-free polygon morphing. Comput Graph 25(1):67–75

Hampe M, Sester M, Harrie L (2004) Multiple representation databases to support visualization on mobile devices. In: Proceedings of ISPRS congress, volume XXXV–B4: IV of international archives of photogrammetry, remote sensing and spatial information sciences, pp 135–140

Haunert JH (2005) Link based conflation of geographic datasets. In: Proceedings of 8th ICA workshop generalisation and multiple representation

Hershberger J, Snoeyink J (1992) Speeding up the Douglas–Peucker line-simplification algorithm. In: Proceedings of 5th international sympoisum on spatial data handling, pp 134–143

Mustière S, Devogele T (2008) Matching networks with different levels of detail. GeoInformatica 12(4):435–453

Nöllenburg M, Merrick D, Wolff A, Benkert M (2008) Morphing polylines: a step towards continuous generalization. Comput Environ Urban Syst 32(4):248–260

Pantazis D, Karathanasis B, Kassoli M, Koukofikis A, Stratakis P (2009) Morphing techniques: towards new methods for raster based cartographic generalization. In: Proceedings of 24th International Cartographic Conference (ICC'09)

Peng D, Haunert JH, Wolff A (2013) Morphing polylines based on least squares adjustment. In: Proceedings of 16th ICA generalisation workshop (ICAGW'13), 10 pages

Reilly DF, Inkpen K (2004) Map morphing: making sense of incongruent maps. In: Heidrich W, Balakrishnan R (eds) Proceedings of graphics interface, volume 62 of ACM international conference proceedings series. Canadian Human-Computer Communications Society, pp 231–238

Saalfeld A (1985) A fast rubber-sheeting transformation using simplicial coordinates. Am Cartogr 12(2):169–173

Saalfeld A (1999) Topologically consistent line simplification with the Douglas-Peucker algorithm. Cartogr Geogr Inf Sci 26(1):7–18

Schneider T, Hormann K (2015) Smooth bijective maps between arbitrary planar polygons. Comput Aided Geom Des 35:243–354

Sester M, Brenner C (2004) Continuous generalization for fast and smooth visualization on small displays. In: Altan O (ed) In: Proceedings of 20th ISPRS congress, volume XXXV (Part B4) of international archives of the photogrammetry, remote sensing and spatial information sciences, pp 1293–1298

Surazhsky V, Gotsman C (2001) Controllable morphing of compatible planar triangulations. ACM Trans Graph 20(4):203–231

van Kreveld M (2001) Smooth generalization for continuous zooming. In: Proceedings of 20th International Cartographic Conference (ICC'01)
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 datastructures suitable for progressive data transfer: the GAP-face tree and GAP-edge forest. Cartogr Geogr Inf Sci 32(4):331–346
Whited B, Rossignac J (2011) Ball-Morph: definition, implementation, and comparative evaluation. IEEE Trans Vis Comput Graph 17(6):757–769