An Area Merge Operation for Smooth Zooming

Radan Šuba, Martijn Meijers, Lina Huang and Peter van Oosterom

Abstract When zooming a digital map it is often necessary that two or more area features must be merged. If this is done abruptly, it leads to big changes in geometry, perceived by the user as a "jump" on the screen. To obtain a gradual merge of two input objects to one output object this chapter presents three algorithms to construct a corresponding 3D geometry that may be used for the user's smooth zooming operations. This is based on the assumption that every feature in the map is represented in 3D, where the 2D coordinates are the original representation, and 1D represents the scale as a Z value. Smooth zooming in or out is thus equivalent to the vertical movement of a horizontal slice plane (downwards or upwards).

1 Introduction

Cartographic generalization is the process of transforming a map from a detailed scale to a less detailed scale. Only the end result of such a process is usually considered. However, for some applications the visual process of continuously changing the Level of Detail (LoD) is important. For example what is visible when zooming the map.

R. Šuba (✉) · M. Meijers · L. Huang · P. van Oosterom
Section GIS Technology, OTB—Research for the Built Environment, Delft University of Technology, Delft, The Netherlands
e-mail: R.Suba@TUDelft.nl

M. Meijers
e-mail: B.M.Meijers@TUDelft.nl

P. van Oosterom
e-mail: P.J.M.vanOosterom@TUDelft.nl

L. Huang
School of Resource and Environmental Science, Wuhan University, Wuhan 430079, China
e-mail: L.Huang-1@tudelft.nl

J. Huerta et al. (eds.), *Connecting a Digital Europe Through Location and Place*,
Lecture Notes in Geoinformation and Cartography, DOI: 10.1007/978-3-319-03611-3_16,

Instead of discretely switching from one scale to another, a continuous transformation from source to target scale is preferable.

An essential factor in map usage is how the user perceives a transition between map scales at different levels of detail in the course of zooming in or out. From the experiment carried out by Midtbø and Nordvik (2007) it is evident that sudden changes during zooming are distracting to the user, and may also result in losing track of the objects the user is interested in.

The classical solutions for zooming are based on a multi-scale approach, where every scale is stored separately and the zooming is effectively "switching" from one map to another. This drawback is offset by tricks such as graphic enlargement of the map layer before "switching", blurring the map at the time of switching or map morphing i.e. an animated translation from one map to another (Reilly and Inkpen 2004, 2007).

The visualization of the continuous scale change has been investigated. van Kreveld (2001) focuses on analysing the different ways of visually continuously changing a map, defining a number of operators that can be used. His work is based on *transitional* maps (maps that connect different predefined scales) and techniques of *cartographic animation* (Robinson et al. 1995; Maceachren and Kraak 1997), such as morphing and fading.

Sester and Brenner (2005) demonstrate the gradual change of objects as a decomposition into a sequence of elementary steps. They call this *continuous generalization* and so far it has been applied only for buildings.

Danciger et al. (2009) introduce deformation of the shapes of regions in a map during a continuous scale change. They define mathematical functions for objects. However, the geometry forming a complete subdivision of space (a planar partition), which is important for vector map data, is not considered in this work.

Alternatively, van Oosterom and Meijers (2011a) introduced an approach termed *smooth zooming* of vector data where the geometry gradually changes, but it has not been implemented. This approach for smooth zooming is based on a 3D space, two dimensions representing the map at a level of detail and one the scale representation, allowing smooth merging of features as will be demonstrated in this chapter. Map features are represented as volumetric data, which are "sliced" to produce a representation. It is based on the vario-scale concept and it offers a gradual change of map objects in the process of zooming, without switching between discrete map layers.

We focus on the transformation of vector data in a representation convenient for supporting smooth zooming and aim to find the requirements for such an operation.

The remainder of this chapter is organised as follows: the part entitled 'Related work' describes the principles and applications of the vario-scale approach. The implementation details are explained in the 'Methods' part. '3D Storage of small dataset' describes the storage efficiency and this is followed by 'Analysis'. Finally, the 'Conclusion and future work' are presented.

2 Related Work

There are two ways of managing a dataset on different levels of detail: the multi-scale approach and the variable scale (vario-scale) approach. Each of these approaches deals differently with scale transition.

The classical multi-scale approach makes use of several versions of the map, explicitly stored at different scales in the multiple representation databases (MRDB). Every time a map of specific scale is required, the most appropriate scale from pre-defined scales is selected and displayed. The zooming only switches from one pre-defined scale version to another pre-defined scale of the map, which is closer to the target scale.

The second method is the vario-scale approach, which creates an index structure on the base map that enables one to extract a map on exactly the right scale whenever one requires. If you want a map on a specific scale it is constructed for you on-the-fly. Vario-scale needs only one dataset to be managed, while data can be displayed on any scale. These aspects strongly motivated our research and in our implementation we only make use of the vario-scale approach.

2.1 Smooth Topological Generalized Area Partition

One of the structures used for the vario-scale approach is known as tGAP (topological Generalized Area Partition) (van Oosterom 2005; van Oosterom and Meijers 2011b). The principle of the tGAP structure can lead to smoother user interaction, for which the concept has been introduced by Meijers and van Oosterom (2011); van Oosterom and Meijers (2011a). It is presented as a space-scale partition. The 3D structure is termed the Space-Scale Cube (SSC). It allows a single real world feature to have a single database representation, by contrast with the discrete scales approach which not only has different representations, but that these are often separately maintained. Example: The same river is named at one scale, and unnamed at a larger scale. Two versions exist: classic and smooth (tGAP/SSC).

The classic SSC stores prism based representations for objects; see Fig. 1a. This gives an idea of how map generalization can be seen as an extrusion of the original data into an additional dimension connecting the discrete scales (without topology errors). Map scale is seen as an additional geometric dimension. The resulting vario-scale representation for 2D map objects is a 3D structure. The 2D area object is then presented as a 3D volume. However, a merge operation still causes a sudden local map change (“shock”). A small change in the map scale does not automatically result in a small geometrical change. This situation will not, however, occur in the smooth SSC representation of tGAP using polyhedra without horizontal boundary faces; see Fig. 1b. All generalization actions must lead to a smoothly changing map. One can imagine this as making a gradual shift of a slice plane (where a horizontal slice is taken) from the top of the cube downwards, there will not be any object suddenly

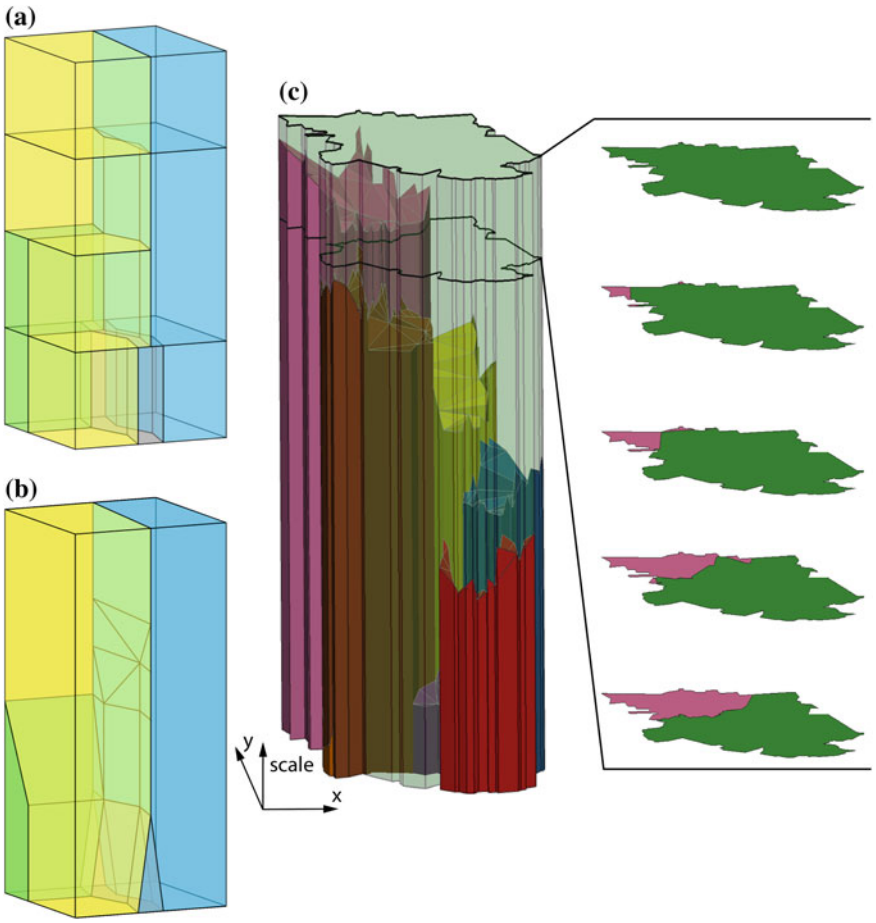

Fig. 1 The space–scale cube of the tGAP structure: **a** SSC for classic tGAP, **b** SSC for smooth tGAP, **c** A small subset of real data dataset with arbitrary slices. The *colors* are randomly assigned. Figures (**a**) and (**b**) *source* (van Oosterom and Meijers 2011a). (**c**) Shows seven area objects (at the most detailed level). The horizontal *slices* illustrate the last step of the tGAP structure where the *pink* object is merging with *green* one

appearing or vanishing. All changes result in a smoothly changing 2D map: a small change in the map scale means a small change in the geometry of the resulting map, see Fig. 1b. Figure 1c provides an example of smooth representation for small subset of CORINE Land Cover dataset. The arbitrary slices in Fig. 1c demostrate the final impresion from a vertical shift of the slice plane.

This chapter investigates requirements of merge operation (aggregation) for smooth zooming on the map and it furthermore presents three algorithms which create 3D object representations for this transition. In other words, the merge operation is converted in a real volumetric representation of the vario-scale data structure,

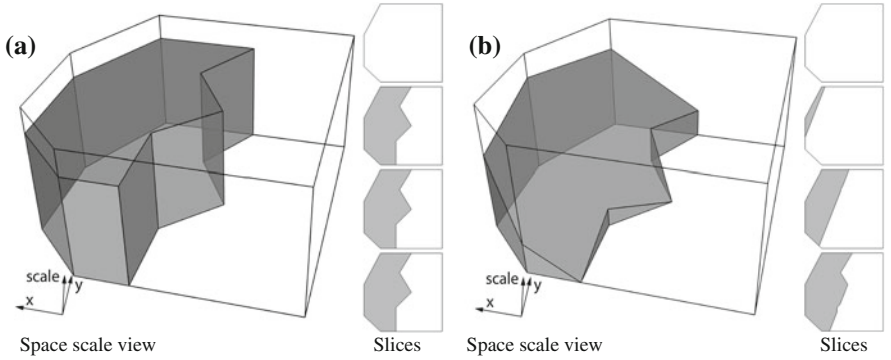

Fig. 2 The merge operation in classic tGAP (**a**) and smooth tGAP (**b**). The *white* winner takes space from the *gray* loser. *Slices* are shown at four levels, from *top* to *bottom* in the image: from *high*, to *low*

where vario-scale 2D objects are represented as 3D geometric objects (2D geometry + 1D scale).

It is important to underline that this chapter is focused on development of merge operation for smooth zooming with the tGAP structure only and thus inherits the advantages and drawbacks of this structure.

2.2 The Principle of Smooth-Merge Operation

The creation of the tGAP structure is based on the merge operation of the least important object which is called call the loser, with its most compatible neighbour, called the winner. Figure 2a presents such an operation in a classic tGAP structure, where the white winner merges with the grey loser and creates the white object. Figure 2b presents the same process in the smooth tGAP, which will be termed the smooth-merge operation. Figure 2b shows that any arbitrary horizontal slice leads to a new 2D map. If the slice plane is moved up, then the white winner grows and the grey loser shrinks. All the geometry changes gradually.

During the transition to the smooth tGAP structure, some objects can be deformed or misrepresented and the resulting map (slice of the smooth tGAP representation) can be confusing. For instance, a single object representation can break into multiple parts—see the white object in Fig. 3. Such spurious multiple parts cause a transitory increase in the number of objects in the map and result in a degradation of the quality of the map. Therefore our intention is to ensure that area features do not break into discontinuous parts.

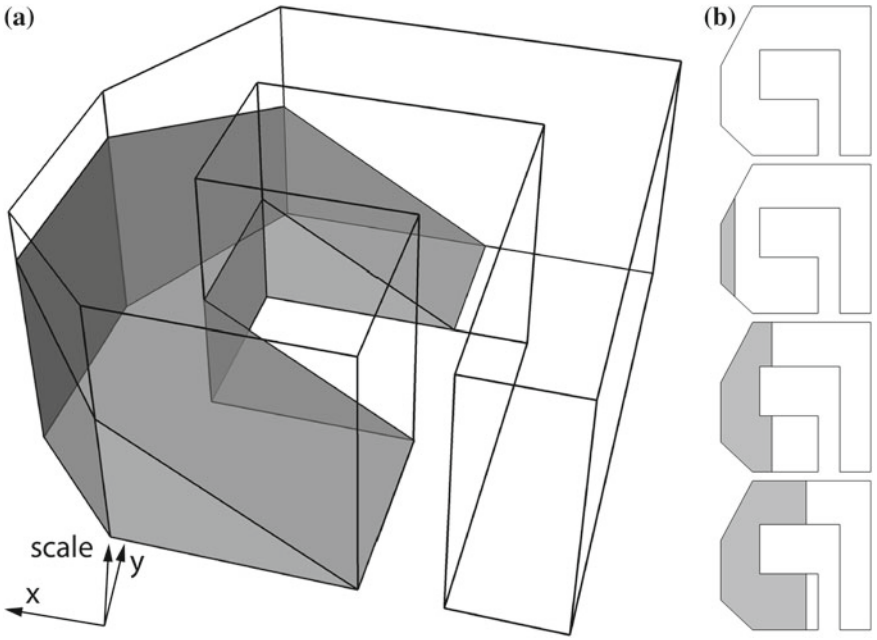

Fig. 3 More complicated shapes with arbitrary slices in smooth tGAP. The *middle* and *low slices* in (**b**) shows two parts of the same object (*white* winner). (**a**) Space scale view. Only share boundary between objects is colored, (**b**) Slices

2.3 Requirements

The main goal of our research was to transfer objects from classic tGAP to the smooth tGAP represented in SSC where horizontal faces, causing abrupt changes, do not exist. A smooth-merge operation was developed. This should compute in 3D a set of tilted faces representing the boundary surface between the loser and the winner. We defined the following requirements for good smooth-merge operations (each one supported by its own motivation):

1. Topologically correct in 3D—The input 2D map is topologically correct and forms a 2D partition of space, the resulting 3D representation should also be topologically correct and not have gaps, overlaps or intersections.
2. No new points, that is no new x, y coordinates—The construction of tilted faces make use of the already existing geometry. It follows one of the main aspects of the vario-scale approach, which is to minimize the redundancy of data. Only the new edges and faces are created using existing geometry.
3. No horizontal faces—This follows from the definition of the smooth tGAP structure mentioned above as horizontal faces cause sudden changes.
4. No vertical faces between winner and loser—The winner should gradually take over the area of loser, but vertical face means nothing happens here for a while.

5. No multi-parts—Spurious multi-part objects (which should be single part), confusion to the map.
6. Constant steepness of tilted face/faces—The shared tilted boundary composed from multiple faces with different steepness will result in the effect that some parts of the loser merge much faster than others during smooth zoom. The steepness of the faces should be as constant as possible.
7. Optimal shape of shared boundary—The shared 3D surface boundary can consist of multiple faces. There often exist more configurations of faces, and each defines a 3D surface of the shared boundary. The configuration that gives the best merge impression to user should be selected. That is, a natural looking boundary during the smooth zoom (slicing operator).

The list above contains three "hard" and four "soft" requirements. The first three requirements will always be guaranteed by the algorithms. It is not possible to guarantee the all other requirements, as they are sometime contradicting and they are therefore classified as soft requirements. It will be tried to optimized these in a well balanced manner. Note that the different "soft" constraints may be competing; e.g. constraint 6 has preference to single flat plane (equal steepness), but may result in multi-parts; see Fig. 3. The hard requirements guarantee functionality and they are crucial to finding the solution. The soft requirements are more aesthetic requirements which are more like recommendations. We would like to fulfill them. However, they are not to be strictly satisfied. Example In some cases a loser composed from multi-parts can result.

3 Methods

Three different algorithms for smooth-merge operation have been created and implemented to satisfy the hard requirements (and optimize the soft requirements): the "Single flat plane", the "Zipper" and the "Eater". The difficulty of implementation with these algorithms ranges from trivial, with the "Single flat plane"; to more complex, with the "Zipper"; to rather complex, with the "Eater". Based on the various types of map objects—the thematic classification and shape, which may result in different emphasis on the soft requirements, it should be possible to select the most suitable algorithm.

3.1 Single Flat Plane

The first implemented algorithm was "single flat plane". It originated from idea that smooth-merge can be represented by simple single flat plane of constant steepness as illustrated in Fig. 2b. In principle, the loser (the face that has to be removed) can always be removed with a single flat plane. The plane is defined in 2D by the shared

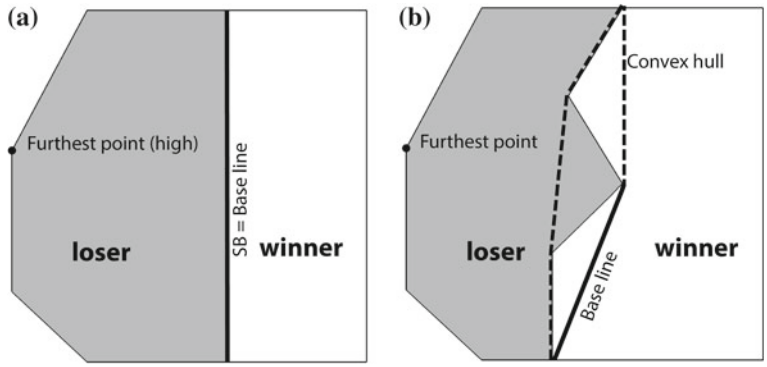

Fig. 4 Definition of the base line for the "Single flat plane". (**a**) A case where the base line is same as the shared boundary, (**b**) A case where the shared boundary is not a *straight line*. The base line is the longest site of the convex hull (*dashed line*) of the shared boundary located in the winner

boundary, the edge(s) between the winner and the loser, and the furthest point of the loser from the shared boundary; see Fig. 4a. Then in the 3D the plane is lowest at the shared boundary and highest at the furthest point. As it is a single flat plane, linear interpolation can be applied for calculating the height at any other point on the share boundary surface between the winner and the loser. Figure 2b shows the final 3D representation of smooth merge using the "Single flat plane" algorithm.

In context of the smooth-merge operation the shared boundary is used, where the merging starts and the distance from the shared boundary to every point of the loser is calculated to find the point most far away. If the shared boundary is not a straight line; see Fig. 4b, then the concept of a base line is introduced. The base line is an "approximation" of the shared boundary and it is used for measuring distances and finding the furthest point. Three ways of finding the base line are investigated:

- The base line is the longest site convex hull of the shared boundary located in (or on the boundary of) the winner; see Fig. 4b.
- The base line is the edge of the smallest rectangle around the loser which has biggest overlap with the face of winner (and has loser completely at other side).
- The base line is result of a best-fit line technique known as Eigenvector line fitting to obtain the orientation of the base line (van Oosterom 1990). There are two options: use all points of the loser or just the points of the shared boundary. Then this Eigenvector line is translated to make sure the loser is completely at one side (and touching the line) with the winner on both sides, but preferably with the largest part opposite the loser.

The first approach, the longest edge of convex hull of the shared boundary has been used for our implementation.

The most serious limitation of the "Single flat plane" approach is orientation. There always exists only one direction where the plane can be oriented. However, in some cases the winner and the loser can have more shared boundaries, e.g. the

boundary between the winner and the loser is broken by another polygon. In such a case, the decision where the tilted plane should be placed must be made. The situation is even worse when the loser lies inside the winner, where no good quality solution exists. Also in case of a very long and facetted boundary, the base line will not be able to represent this well. Some vertical faces are needed to make the single flat plane fit into the 3D SSC. Note this is also the case when base line fits rather well, only then the vertical faces will not need to be very high. The vertical faces result in the effect that at their locations for a while, nothing changes while other parts the winner is already taking space from the loser.

Despite the above mentioned limitations of the "Single flat plane" algorithm, it can be very effective for simple convex polygons where a simple shared boundary exists. The computation of the base line, together with a definition of the tilted face, is trivial. From the smooth zooming point of view, the "Single flat plane" has the advantage of the winner face growing with constant speed. These aspects make the "Single flat plane" algorithm a good candidate for processing simple convex faces, such as rectangular buildings.

3.2 Zipper

The second approach is based on the decomposition of the loser polygon into triangles. These triangles will then in 3D become the tilted surface of the shared boundary. The segments of the boundary of the loser are classified into two types: 1. The Shared Boundary (SB), which is the boundary between the winner and the loser and 2. The Opposite Boundary (OB), which is the remaining part of the boundary of the loser, holes included. Figure 5 shows the final 3D representation. It is assumed that the tilted faces should be tilted from SB (low) to OB (high). Before we designed the "Zipper" algorithm, we first investigated another method for finding a good solution satisfying as many requirements as possible:

- "Delaunay triangulation and flipping"—which starts with a Delaunay triangulation of the loser, giving the fattest triangles possible (satisfying the soft requirements). Then it traverses the triangulation and flips edges which might provide a better configuration of the triangles. This method is computationally expensive and does not guarantee the best solution, because a local flip of the triangles may not lead to the global optimal solution.

Therefore, another method, the "Zipper", was implemented. The hard requirements remain valid during the whole process: topology is correct, no new vertices (x, y coordinates) are created and no horizontal faces are created. Besides, the other optimization rules have been met:

- Every triangle connects SB and OB, which means that at least one vertex lies on the SB and at least one other vertex lies on the OB;

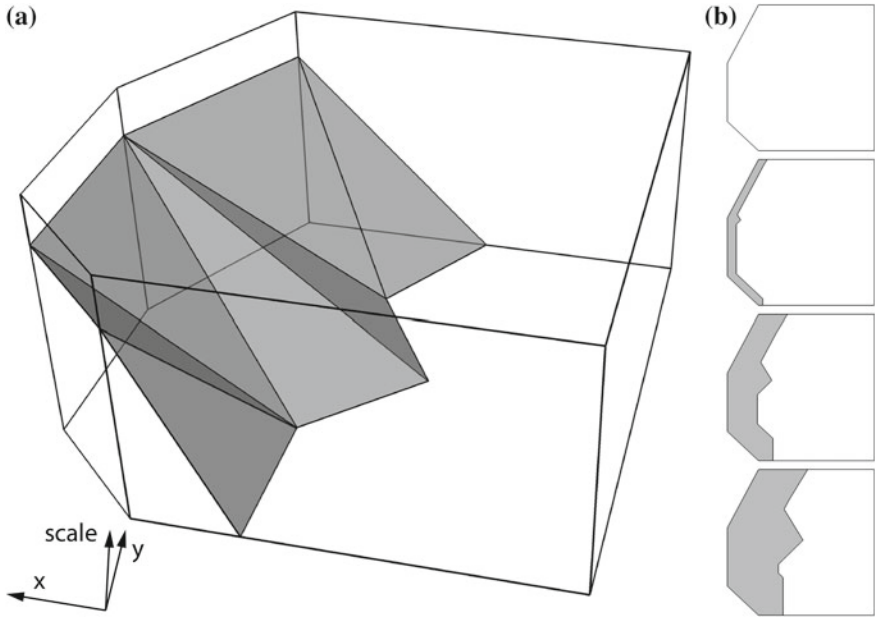

Fig. 5 The zipper algorithm. The *white* winner taking over space from the *gray* loser. (**a**) Space scale view, (**b**) *Slices* at four levels, from *top* to *bottom* in the image: from *high* to *low*.

- The triangles should have a low aspect ratio—the aspect ratio is the longest side divided by the shortest altitude of triangle (Shewchuk 1996). It guarantees that the triangles have no sharp angles;

These additional rules guarantee that we get the best solution according to the hard and soft requirements. Another optimization rule, not further explored could be: the two edges that run from SB to OB should be as much as possible of equal length, making sure that the speed of transition is as nearly as possible equal.

Figure 6 describes the principle of the "Zipper" algorithm. One can imagine the process as a walk along the SB and OB simultaneously, creating of a connection from one vertex at SB to another vertex at OB, if possible. When the connection is created, a new triangle is defined. The process starts at the junction of SB and OB (vertex I in Fig. 6) and ends in another junction (vertex II in Fig. 6). For every edge, connecting SB and OB, the aspect ratio is computed. Then the process can start from the other side of the SB. The optimal solution is defined as one where the whole area of the loser is processed and where the sum of aspect ratios is minimal.

Figure 7 presents an alternative visualization of the process. The process can be represented as a graph where every connecting edge is a node in the graph. The vertical axis corresponds to the index of SB. The horizontal axis corresponds to the index of OB. Only moving to the right or moving downwards in the graph is possible as the algorithm to create triangles either takes a step on SO or OB, but

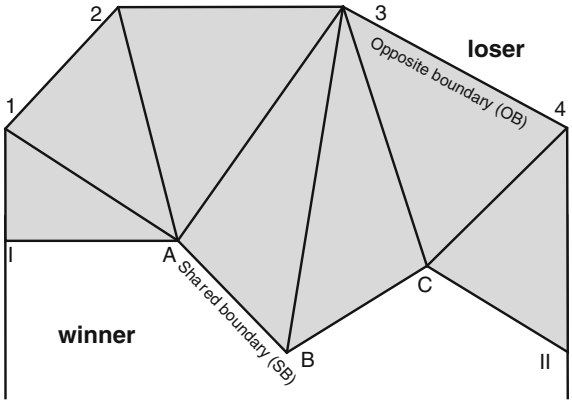

Fig. 6 Principle of the "Zipper". The optimal solution, only the final connections are present

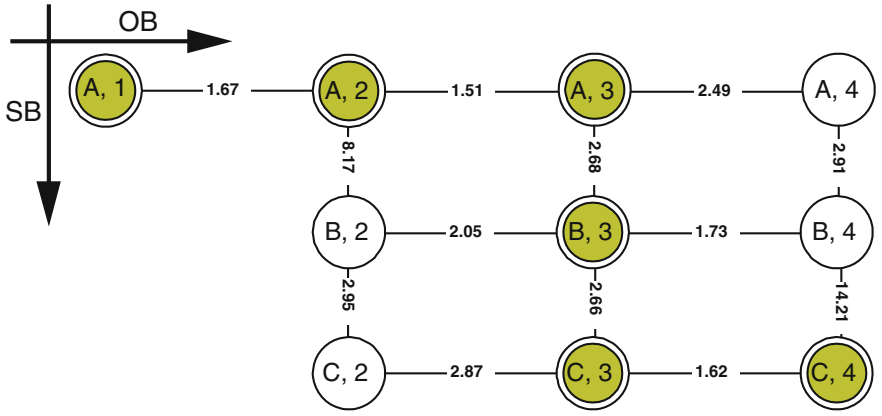

Fig. 7 *Graph* representation of the "Zipper" approach for configuration in Fig. 6. The "zipping" starts at *triangle* (I, A, 1) towards (II, 4, C). *Note* that the process can run in the opposite direction. The *yellow* presents

not both as this would result in a quadrangle. The weight of a graph edge is defined as the aspect ratio of the triangle associated with the graph edge connects.

The process starts in the upper left corner and finishes (if a solution exists) in the bottom right corner. If a solution exists, then a connection between the upper left and the bottom right corner exists.

The graph representation can be used for optimization. The best solution is the solution with the lowest total sum of aspect ratios of triangles (corresponding to graph edges). Basically, it is the cheapest path through the graph. As it can be observed, the edges with the lowest ratio tend to lie mostly on the diagonal of the graph, see Fig. 7. This means that for an optimal solution it is sufficient compute the diagonal connection from the upper left to the bottom right corner and then if such a connection does

not exist one can start to compute other nodes further from diagonal. If there is no connection between the upper left and the bottom right corner this means that it is not possible to process the loser in a single step, one of the faces of the loser must be split into more pieces. Note that in such a situation the other algorithm "Delaunay triangulation and flipping" also cannot find a solution as there is no solution without splitting loser in multiple parts. Only certain types of polygons (all nodes on OB are visible from nodes of SB) can be processed directly. This always the case with convex polygons, but also for certain type of concave polygons; e.g. relatively long polygons with visibility between the two parts of the boundary SB and OB (e.g. road or river polygons without side branches, which should be treated as separate objects).

Where it is not possible to process the polygon in this way, the graph can suggest where a split can take place. The associated vertex of the graph nodes, where the connection fails in most cases, can be a candidate for splitting. Unfortunately, with splitting a number of possible solutions arise and for the overall optimal solution many or all of them should be checked. Therefore another algorithm, which is called the "Eater" was developed. It will be presented in the next section offering a general solution for arbitrary polygons without the need to split either feature.

3.3 Eater

The third approach, which is called the "Eater" provides a solution to any arbitrary polygon (with holes, concave, or multiple shared boundary parts). It is based on a triangular tessellation of the polygon and ordering the triangle into a gradual change from low to high. Figure 8 shows the result. The Delaunay triangulation takes initial step, where the face of the loser is tessellated. Then finding the starting triangles for the walk takes place. The triangles which have two edges of shared boundary are selected as starting triangles if any exist, and added to the so-called *active set*. These triangles are processed first which makes the shared boundary less "curvy". Note that when processing such a triangle in 3D there are two options how to set relative heights: 1. keep both edges of SB completely at the lowest start height, or 2. keep only the shared node of these two edges at the lowest start and other two nodes at the first height step. The first option will result in a horizontal triangle and therefore violates a hard requirement. Therefore the second option is selected, which has only the drawback of introducing some vertical triangles, but this is 'just' violating a soft constraint and not a hard one. If there are no triangles with two edges in SB, then the triangles which have one edge in SB are selected as starting triangles and these add into the *active set* (and both nodes of involved edge get fixed height at start level). The process starts with all triangles set on *not-visited*, some triangles in the *active set*, the *relative height* set to 1 and empty *next active set*. The elements in square brackets refer to the *active set*, the elements in curly brackets refer to the *next active set* for initial iteration. The steps to process and sort the triangles are as follows:

1. start with starting triangles as the *active set*, [A] { };
2. until the *active set* becomes empty do:

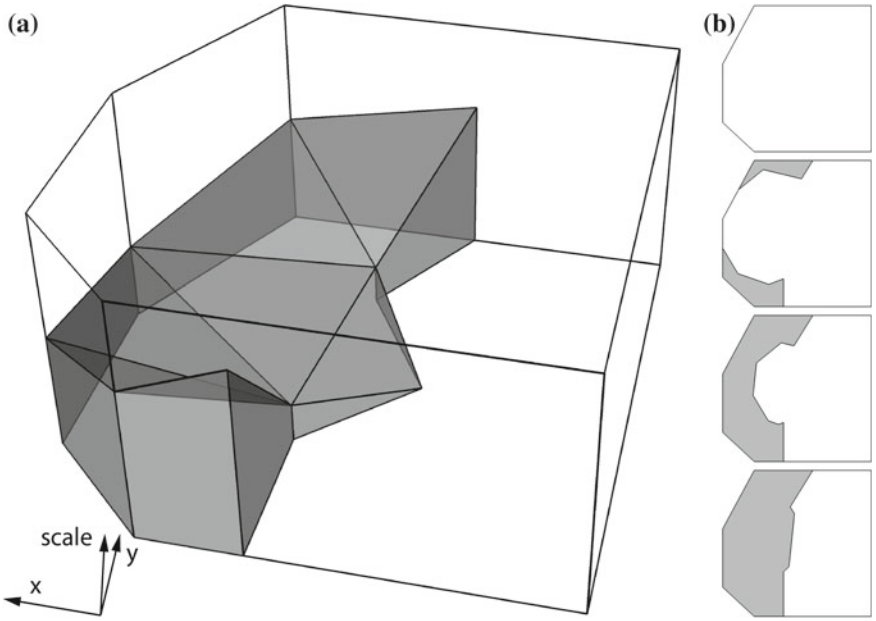

Fig. 8 *Tilted* faces of loser processed by the "Eater". (**a**) Space scale view, (**b**) *Slices* at four levels, from *top* to *bottom* in the image: from *high* to *low*

(1) select a triangle from the *active set*, A;
(2) if a node has fixed height use it, otherwise use the *relative height* for a node (defining the triangle in 3D space, fitting into the SSC topology);
(3) add all non-visited neighbours of triangle into the *next active set,* [A] {B};
(4) remove the triangle from the *active set* and set to *visited,* [ ] {B};
(5) if there are no more triangles in the *active set*, then increase the *relative height* and move the triangles from the *next active set* into the *active set,* [B] { };

For next iterations we start with [B] { } and other steps are as follows:

	2. Iteration	3. Iteration	4. Iteration	5. Iteration	6. Iteration
(1)	B	C_1	C_2	E_1	E_2
(2)					
(3)	[B] {C_1,C_2}	[C_1,C_2] {E_1}	[C_2] {E_1,E_2}	[E_1, E_2] {F}	[E_2] {F}
(4)	[] {C_1,C_2}	[C_2] {E_1}	[] {E_1,E_2}	[E_2] {F}	[] {F}
(5)	[C_1,C_2] { }	[C_2] {E_1}	[E_1,E_2] { }	[E_2] {F}	[F] { }

Figure 9 demonstrates the principle and shows the sequence of the traversal of the triangles. The "Eater" ends with the triangles ordered, and a relative height assigned to every node.

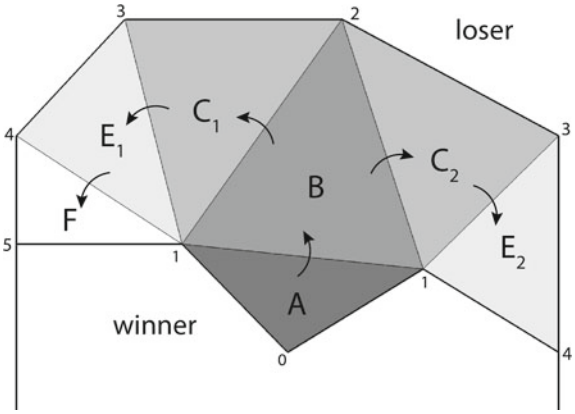

Fig. 9 Principle of the "Eater". The *triangles* are walked from *dark* to *light grey*. The numbers present relative Z value

In general, this algorithm guarantees a solution for every object, the face with multiple shared boundaries included. With this algorithm it is always possible to convert a dataset in classic tGAP representation into a smooth tGAP representation, where no horizontal faces exist. The price for that is some vertical faces and a significant chance of multi-parts. The cooperation with other approaches and conclusion will follow in next sections.

4 3D Storage of Small Dataset

The number of resulting elements can give us some idea how efficient the storage is. The small subset of CORINE Land Cover dataset (7 faces) is used as an input; see Fig. 1c. It contains 676 vertices and 15 records of faces in classic tGAP structure. Scale attributes will be used as z values in the conversion to the 3D representation.

Because of its general applicability the "Eater" algorithm was used for converting data stored in classic tGAP into the smooth representation. The whole dataset in smooth tGAP has been stored explicitly as polyhedral volumes in a 3D topological structure containing:

- A list of vertices where x, y, z coordinates are stored for every vertex.
- Faces, stored as a list of indices pointing to the vertex list. Shared faces between objects are always represented just once, not duplicated. On the other hand, the edges between two faces are stored twice.

This small example is represented by 989 vertices (which is more than the original 676 vertices as some of these have multiple counterparts at different height levels; however, there are no new x, y coordinates introduced) and 684 faces, consisting of: 289 triangles—the tilted faces of shared 3D boundaries between winners and losers,

Table 1 Summary table, where − = bad, + = neutral and + + = good

	Single flat plane	Zipper	Eater
Always has solution	−	+	++
Type of polygon	− (One SB only, convex, rectangular)	+ (Convex, some concave with visible SB-OB)	++ (All)
Computational efficiency	+	−	+
Multi-parts	−	+	−
Optimal shape of faces	−	++	+
Number of extra elements	+	+	−
Constant steepness	++	+	−
No vertical faces at SB	−	++	−

The first three rows evaluate the algorithms in general. The remaining rows give an overview of fulfilment of the mainly aesthetic soft requirements

and 395 vertical rectangular faces (normal boundaries, which are not the result of a smooth merge). Finally there are 7 horizontal bottom faces and 1 horizontal top face.

Alternatively, the smooth tGAP dataset has been converted to tetrahedrons (a tetrahedron is a geometric object composed of four vertices and four triangular faces) for visualization and slicing purposes (Šuba et al. 2013). 1734 tetrahedrons have been stored explicitly in a 3D topological structure comprised of:

- A list of vertices;
- Tetrahedrons, stored as a list of indices pointing to the vertex list.

The slicing can be done very efficiently because to make a slice of set of the tetrahedrons is much easier than making a slice of arbitrary 3D polyhedral objects. On the other hand the number of elements is in the thousands for even a small example.

From the simple example presented above we can see that making the 3D representation explicit can be space consuming, but more important is when and where the conversion is performed. Once a *subset* of the tGAP structure is transferred from a server to a client, the conversion can be performed very efficiently. Then the 3D reconstruction and visualization can use the full potential of graphic hardware such as computing slices of tetrahedrons or rendering.

5 Analysis

This research investigated the possibility of smooth-merge operation and implemented three algorithms, the “Single flat plane”, the “Zipper” and the “Eater”. These approaches are summarized in Table 1. Some of the criteria on the left side of the table come up from soft requirements. Each of these criteria can be quantified and based on this the different algorithms are compared. In addition not all criteria are of equal weight when choosing best solution for specific situation.

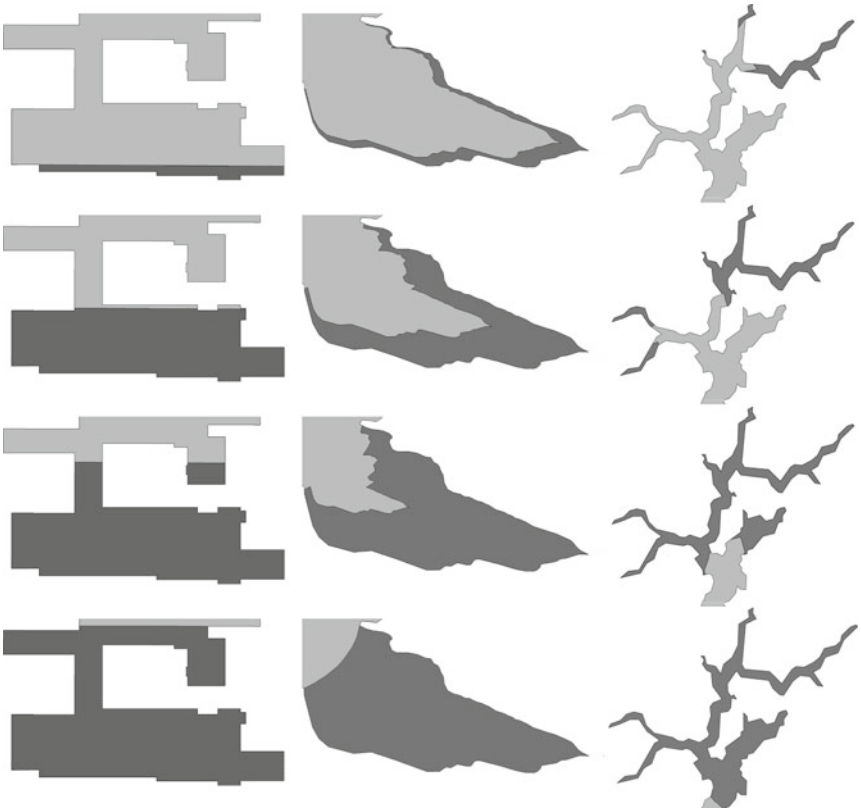

Fig. 10 Example of three approaches with real data, the "single flat plane" on the *left*, the "Zipper" in the *middle* and the "Eater" on the *right*. The arbitrary slides simulate gradual merge (from less merged (*bottom*) to more merged (*top*)). The *light grey face* is the winner, the *dark grey face* is the loser

First, the "Single flat plane" offers easy computation, constant speed of merging and it always creates just one tilted plane. The main disadvantages are defining the orientation of the tilted plane where the shared boundary is rather complex or more shared boundaries exist. The algorithm also does not try to avoid a multi-parts and the final impression of merging looks artificial—a single line "sweeping on the screen". A low score has been given to "Always has solution", because it in some cases the solution is not meaningful, e.g. where the winner lies inside the loser. Despite those limits the "Single flat plane" algorithm can be very effective for simple convex rectangular polygons where only one shared boundary exists, such as building see Fig. 10.

Second, the "Zipper" offers more generic solution. It works for all convex and even for some concave polygons, this depends on visibility between SB and OB. It fulfils soft requirements and finds the optimal solution, if it exists; which results in good impression of merging and minimal risk of multi-parts. On the other hand, the quadratic computational complexity can be overcome by graph representation together with finding optimal solution. If there is no single part solution, the graph also gives an indication of how to split the polygon into multiple parts.

Third, the "Eater" offers a solution for any arbitrary polygon. It always processes one triangle at the time and the whole polygon is processed in one operation; which makes the algorithm more effective and robust. The Delaunay triangulation with $O = (n \log n)$ is the most expensive part of the algorithm. When using the 3D structure and creating slices, the risk of multi-parts is high and it cannot guarantee constant steepness. However, steepness can be improved by global information about configuration of triangles.

Figure 10 shows process of merging in few steps (slices of the 3D representation). It gives an impression of smooth merge operation. The reader can see that delta zoom means delta change in geometry. As can be seen, the "Single flat plane" approach simulate merging by sweeping straight line trough area of looser which (from our experience) gives an artificial impression while the zipper and the eater looks more "natural".

6 Conclusion and Future Work

This research investigated various algorithms for the smooth-merge operation, of which three were implemented (the "Single flat plane", the "Zipper" and the "Eater") and tested with real world data. Each method has its own strong and weak points. However, various improvements are possible in future work.

First of all, to minimize the chance of multi-parts. The "Eater" as a promising algorithm can be further improved in the future. Using other information involved, such as predecessor of processing triangle, the negative effect of multi-parts can be reduced. However, some cases need to be treated more globally. Example: Where the loser has many holes and "branches", and where some "branches" go faster than others.

Additionally, more user testing can be involved. Right now, there are no fixed rules how the process of merging should be done and how the final impression should look. In this research the hard and soft requirements have been identified, based on analysing the properties of the smooth merge operation. However, it is not completely clear which process of merging is capable of delivering the best quality; i.e. balancing the various soft requirements.

Another concern is the storage aspect of the 3D data structure. Minimum redundancy is one of the main principles of vario-scale, but our current encoding of Space-Scale Cube takes significant storage (and especially the number of tetrahedrons is quite big). How can redundancy be minimized also for explicit 3D storage

and is explicit 3D storage really needed? Storing the data in a 3D topology data structure could be one option. Another option could be to derive what is needed from the tGAP data structures that store more or less separately the 2D geometry and 1D scale range, creating the 3D representation when needed, e.g. at client side, because the current tGAP data structure implicitly contains 3D data, but does not store it as such.

Currently, all smooth merge operations are sequenced. First one operation is finished and then the next merge takes place. So, only at one location at the time something in the maps is changing. It will look more natural if several, non-interfering, smooth merge operations take in parallel.

Furthermore, investigating the effects of other generalization operators in the smooth tGAP structure. Besides merging, also simplification and collapsing/splitting are supported in current tGAP structure. Apart from making a smooth version for these operation an open question is how to store the result in the tGAP structure. For example, after the collapse of an area object to a line (or point) object, the line (or point) object lives on. In the SSC this object is therefore represented by a polyhedral volume to which a vertical surface (or vertical line) is connected at the top. All attributes are attached to the same object, which is represented in the SSC by connected multiple parts of respectively dimension 3 and 2 in case of collapse to line and 1 in case of collapse to point.

Acknowledgments This research is supported by the Dutch Technology Foundation STW (project number 11185), which is part of the Netherlands Organisation for Scientific Research (NWO), and which is partly funded by the Ministry of Economic Affairs. Furthermore, we would like to thank the anonymous AGILE reviewers and Rod Thompson for their constructive comments, suggestions and questions. Extra thanks to Rod for correcting our English.

References

Danciger J, Devadoss SL, Mugno J, Sheehy D, Ward R (2009) Shape deformation in continuous map generalization. GeoInformatica 13(2):203–221

Maceachren AM, Kraak M-J (1997) Exploratory cartographic visualization: advancing the agenda. Comput Geosci 23(4):335–343. doi:http://dx.doi.org/10.1016/S0098-3004(97)00018-6

Meijers BM, van Oosterom PJM (2011) The space-scale cube: an integrated model for 2D polygonal areas and scale. Int Arch Photogram Remote Sens Spatial Inf Sci XXXVIII-4/C21:95–102. doi:10.5194/isprsarchives-XXXVIII-4-C21-95-2011

Midtbø T, Nordvik T, (2007) Effects of animations in zooming and panning pperations on web maps: a web-based experiment. Cartogr J 44(4):292–303. doi:10.1179/000870407X241845

Reilly DF, Inkpen KM (2004) Map morphing: making sense of incongruent maps. Paper presented at the proceedings of graphics interface 2004, London, Ontario, Canada

Reilly DF, Inkpen KM (2007) White rooms and morphing don't mix: setting and the evaluation of visualization techniques. Paper presented at the proceedings of the SIGCHI conference on human factors in computing systems, San Jose, California, USA

Robinson AH, Morrison JL, Muehrcke PC, Kimerling AJ, Guptill SC (1995) Dynamic/interactive mapping, Chap. 29. In: Robinson AH, Morrison JL, Muehrcke PC, Kimerling AJ, Guptill SC (eds) Elements of cartography, 6th edn. Wiley, New York

Sester M, Brenner C (2005) Continuous generalization for visualization on small mobile devices. In: Peter Fisher (ed) Developments in spatial data handling. Springer, Berlin, pp 355–368. doi:10.1007/3-540-26772-7_27

Shewchuk JR (1996) Triangle: engineering a 2D quality mesh generator and delaunay triangulator. Paper presented at the selected papers from the workshop on applied computational geormetry, Towards Geometric Engineering

Šuba R, Meijers M, van Oosterom P (2013) 2D vario-scale representations based on real 3D structure. In: Proceedings of the 16th ICA generalization workshop, pp 1–11

van Kreveld M (2001) Smooth generalization for continuous zooming. In: Proceedings of the 20th international cartographic conference, pp 2180–2185

van Oosterom P (1990) Reactive data structures for geographic information systems. Ph.D. Theses, Department of Computer Science, Leiden University, The Netherlands

van Oosterom P (2005) Variable-scale topological data structures suitable for progressive data transfer: the gap-face tree and gap-edge forest. Cartography and geographic information science 32(4):331–346. doi:10.1559/152304005775194782

van Oosterom P, Meijers M (2011a) Towards a true vario-scale structure supporting smooth-zoom. In: Proceedings of 14th ICA/ISPRS workshop on generalisation and multiple representation, pp 1–19

van Oosterom P, Meijers M (2011b) Vario-scale data structures supporting smooth zoom and progressive transfer of 2D and 3D data. NCG Jaarverslag, pp 21–42