6 Smooth zooming

Chapter 2 showed that current maps on the Internet are composed of the discrete set of LODs/scale pyramids with big changes of map content and representations. This can lead to confusion for the users when they navigate in the map. Therefore, a conceptual model (SSC) was proposed, see Chapter 3. We believe that, by capturing the whole generalization process in small smooth incremental changes, it is possible to achieve a better user experience *e. g.* when the user zooms in and out. To verify this hypothesis it is necessary: (1) generate a SSC dataset (2) develop a software prototype where a dataset can be used in its full potential and (3) perform usability test. Hence this chapter will describe the benefits of such as smooth representation in more detail. Section 6.1 gives an introduction to the problem and it suggests our solution. Section 6.2 covers the theoretical background, principle and example of smooth zooming. Section 6.3 presents possible conversion strategies to smooth representation. Section 6.4 explains the software prototype for possible usability testing. This is followed by our initial usability test, which was carry out addition to our plans. More specific, Section 6.5 defines all elements of the testing. Section 6.6 presents preliminary results, and Section 6.7 describes gained experiences. Then, Section 6.8 summarizes possible improvements for the future.

Own publications

This chapter is based on the following own publications:

- Šuba, R., Meijers, M., Huang, L., and van Oosterom, P. (2014a). An area merge operation for smooth zooming. In Huerta, J., Schade, S., and Granell, C., editors, *Connecting a Digital Europe Through Location and Place*, Springer Lecture Notes in Geoinformation and Cartography, pages 275–293. Springer International Publishing. ISBN: 978-3-319-03611-3.
- Šuba, R., Driel, M., Meijers, M., van Oosterom, P., and Eisemann, E. (2016a). Usability test plan for truly vario-scale maps. In *Proceedings of the 19th ICA Workshop on Generalisation and Multiple Representation, Helsinki, Finland*.
- Huang, L., Meijers, M., Šuba, R., and van Oosterom, P. (2016). Engineering web maps with gradual content zoom based on streaming vector data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 114:274 – 293.

§ 6.1 Benefit of smooth representation

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 stored. However,

for some applications the visual process of continuously changing the level of detail is important. Instead of discretely switching from one scale to another, a continuous transformation from source to target scale is preferable because it provides a better impression to the user.

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 with different map content is stored separately. Then the zooming is effectively 'switching' from one map to another with sudden map content change. Attempts are made to relieve 'map shocks' 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).

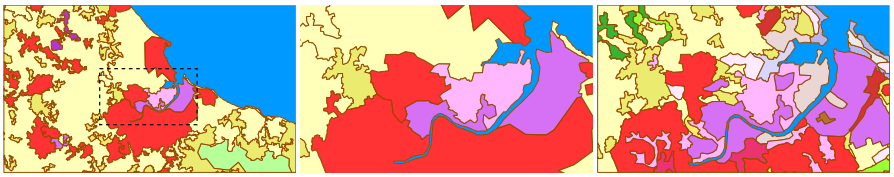
Despite some useful efforts presented so far, see Section 2.3, there is no optimal solution yet. This brings new challenges but also new demands which should be reflected in map generalization and its requirements. Based on the current solutions, we can identify two following directions to introduce more gradual change of the scale:

- Generating smooth content – The purpose of the map is considered in the data generation. The map generalization is perceived as a production of a sequence of successively more generalized maps. The sequence can then be shown to the user. The characteristic of data should be also consider for data transfer; *e. g.* (Sester and Brenner, 2005; Chimani et al., 2014).
- Using graphic techniques – The original data and data in transfer are not modified. Additional graphic techniques are used on top of the map solution. Most of the time well-known techniques from the field of computer graphics are used such as morphing, blending between the layers, transparency and initialising, *e. g.* (Reilly and Inkpen, 2007; Nöllenburg et al., 2008; Danciger et al., 2009).

Our vario-scale solution with representation of the generated sequence of the generalization steps fits into the first category, see description of the concept in Chapter 3, which provides solid base for smooth user experience. However, for optimal smooth user experience both smooth content and graphical techniques are important. Additionally, the zooming operation can be performed and implemented in a variety of different ways. Therefore, the following explanation describes the zooming functionality used in this chapter.

§ 6.2 Smooth zoom aspects

Let m_0 and m_1 be the two maps that will be shown while performing a zoom operation. The initial map m_0 shows coarse data at a specific scale (small scale). Map m_1 will contain more detailed data (larger scale), since it covers a smaller geographical area. Note that both the area and the content of the maps m_0 and m_1 are different.



(a) Initial view m_0. Dashed rectangle shows the final extend of the map m_1.

(b) Original map m_0 where only graphical zoom has been applied.

(c) Map m_1 where the content zoom has been fully applied.

FIGURE 6.1 User experience of a zoom in operation. Figure (a) is original map. The rectangle shows the area zoomed in upon by the user. Figure (b) where the content of the initial coarse map is first graphically enlarged and then replaced by the more detailed map (in one go, thus not in small incremental steps). Figure (c) Map after the *rescaling* and the *content zoom* operations have been applied.

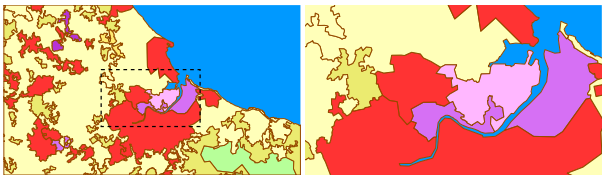
The transition from m_0 to m_1 is what we call a 'zoom in' operation. During this transition two operations are performed: For the first aspect of the transition, called *rescaling*, the objects shown on map m_0 are graphically enlarged (scaled, translated). Only the region corresponding to m_1 is shown (clipped) after this enlargement (no new content is yet retrieved or shown). The second aspect, called *content zoom*, changes the objects of the map (the content). The already graphically enlarged region of map m_0 changes its contents to become map m_1, see Figure 6.1. A zoom out operation applies the same steps as the zoom in operation, but in reverse order: instead of enlarging the map it shrinks it. Starting with map m_1, it first shrinks the objects graphically to the extent of m_0. Then the content is changed to become map m_0. Additional *graphic techniques* (*e. g.* blurring, morphing or making objects more transparent) can be applied for both (*rescaling and content zoom*) aspects in the transition. Note that a zoom operation consists of one *rescaling*, one step of *content zoom* and none or more *graphic techniques* all together integrated in one final map (*e. g.* m_1).

Figure 6.2 illustrates an alternative that, with a smaller difference in the content between map m_0 and m_1 the *content zoom* step is perceived to be changed more gradually (less of a shock). Therefore, we can apply the *content zoom* operation multiple times, changing the map content only bit by bit (leading to more temporary maps, showing the transitioning of the map content in small steps from m_0 to m_1 , *e. g.* m_0, m_0', m_0'', m_0''',... , m_1), and thus progressively refining the map. This leads to a very smooth transition from one scale to another -- hence the term *smooth zoom*.

To conclude, we can define *smooth zoom* operation as following: After one user request (*e. g.* zoom-in) a series of frames is generated with small differences per frame w.r.t. both *graphic* and *content zoom* aspects. Then, one *smooth zoom* operation consists of one *rescaling*, *content zoom* of n steps (where $n > 1$, *e. g.* m_0', m_0'', m_0''', m_1) and none or more *graphic techniques*. This is integrated in n final maps in order to achieve a very smooth transition from one scale to another.

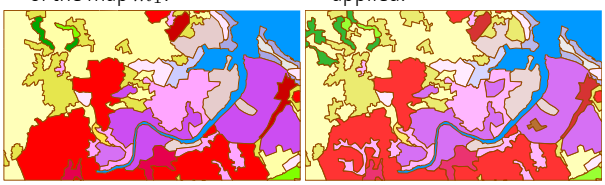
§ 6.3 Generating 3D Space-Scale Cube

In the previous section we gave detailed description of the method which could provided better user map impression. It is based on combination of *graphic techniques*,



(a) Initial view m_0. Dashed rectangle shows the final extend of the map m_1.

(b) Original map m_0 where only graphical zoom has been applied.



(c) Map m_1 where one incremental step of the *content zoom* has been applied.

(d) Map m_1 where the *content zoom* has been fully applied.

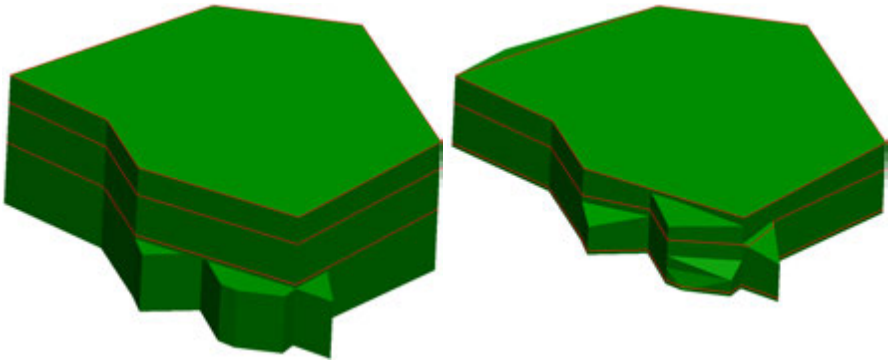
FIGURE 6.2 User experience of a smooth zoom in operation. The content of the initial coarse map is first graphically enlarged (shown in Figure (a) and (b)) and then replaced in incremental steps, leading to the refined map (shown in Figure (c) and (d)).

rescaling and transition over the sequence of maps with small changes in content. However, an issue still remains; how can such a map sequence for *content zoom* be generated?

It has been investigated in (van Oosterom and Meijers, 2011b), (see Section 3.4 on page 28) where the tGAP structure delivers vario-scale data and can be used for gradual content zoom in the form of SSC representation. The structure has very significant advantages over existing multi-scale/multi-representation solutions (in addition to being truly smooth vario-scale): (a) due to tight integration of space and scale, there is guaranteed consistency between scales (it is one integrated space-scale partition), (b) since map features are represented as volumetric data, which are 'sliced' to produce maps, it is relatively easy to implement because this is well-known method from 3D computer graphics, (c) it provides a compact (good for storage, transfer and CPU use) and object-oriented encoding (one higher dimensional representation for a single object over the complete scale range).

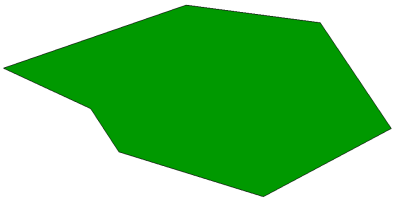
We focus on transition of vector data in a representation convenient for content zoom captured in tGAP structure to be stored in a SSC, investigating generalization operations such as merge, split, line simplification to do so. Figure 6.3 illustrates attempt at implementing a line simplification algorithm that generates smooth output for one object. In this section will focus on the merge operation (aggregation) as the most dominant operation in tGAP structure creation and we will present three algorithms for such a transition. It is important remind that tGAP/SSC exists in two versions: classic and smooth, see Section 3.4.

We aim at smooth representation in the smooth version of tGAP/SSC, where 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 Figure 3.8b on page 30. Figure 6.4

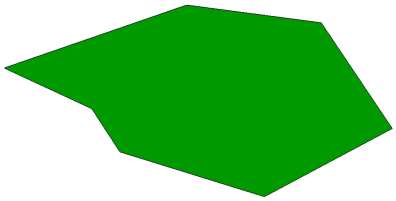


(a) 3D perspective view (*with* sudden boundary simplification). The lower part of the object is result of previous step. It is used as input for simplification.

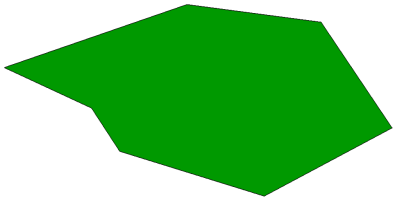
(b) 3D perspective view (*with* smooth boundary simplification)



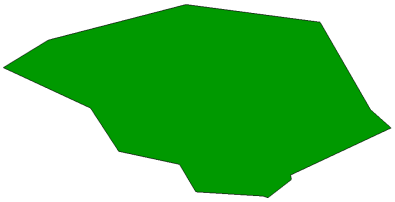
(c) (less detailed) – without smooth simplification



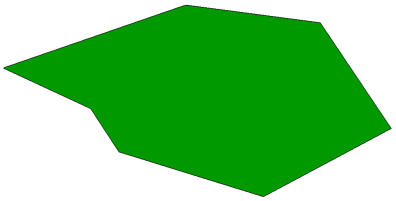
(d) (less detailed) – with smooth simplification



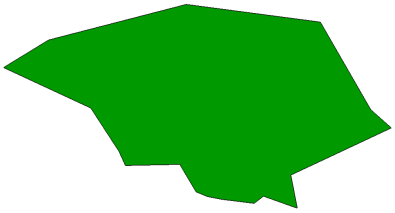
(e) without smooth simplification



(f) with smooth simplification



(g) 2D horizontal slice (detailed) – without smooth simplification



(h) 2D horizontal slice (detailed) – with smooth simplification

FIGURE 6.3 Two Space-Scale volumes with (right column) and without (left column) smooth line simplification and some derived slices. The red lines indicate the position of the slices. The boundaries of the object are simplified with the Visvalingham Whyatt algorithm (Visvalingam and Whyatt, 1993), leading to a coarser object. The order in which the vertices are removed from the line by the algorithm are stored in a binary tree structure (the so-called Binary Line Generalization tree, BLG tree). This tree structure provides enough information to construct a 3D volume for the object.

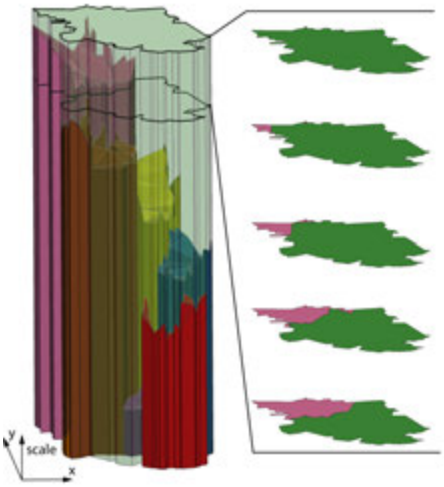


FIGURE 6.4 A small subset of CORINE Land Cover dataset in smooth tGAP with arbitrary slices. The colours are randomly assigned. Shows 7 area objects (at the most detailed level). The horizontal slices illustrate the last step of the tGAP structure where the pink object is merging with green one.

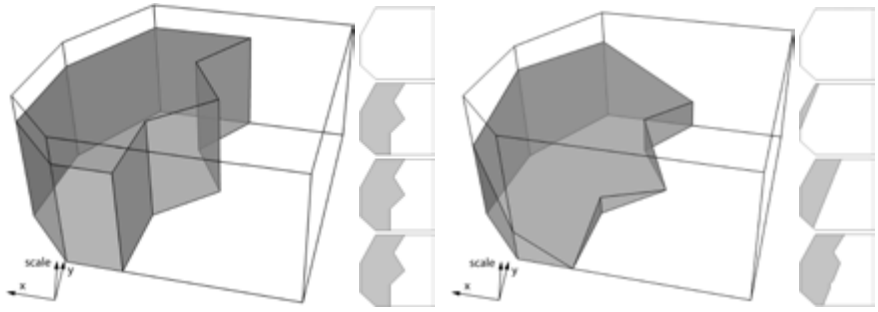
provides an example of smooth representation for small subset of CORINE Land Cover dataset. The arbitrary slices in Figure 6.4 demonstrate the final impression from a vertical shift of the slice plane.

The remainder of this section is organised as follows: First, the basic principle of smooth-merge operation is described in §6.3.1. Second, the requirement for such a conversion are covered in §6.3.2. Third, designed algorithms for the operation are demonstrated in §6.3.3 followed by §6.3.4 describing the storage efficiency. Finally, §6.3.5 presents a final comparison of the algorithms.

§ 6.3.1 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 the $loser$, with its most compatible neighbour, called the $winner$. Figure 6.5a presents such an operation in a classic tGAP structure, where the white winner merges with the grey loser and creates the growing white object. Figure 6.5b presents the same process in the smooth tGAP, which will be termed the smooth-merge operation. Figure 6.5b 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 could be deformed or misrepresented and the resulting map (slice of the smooth tGAP representation) can be confusing. For instance, a single object representation could break into multiple parts, see the white object in Figure 6.6. Such spurious multiple parts cause a transitory increase in the number of objects in the map and result in a degradation of the



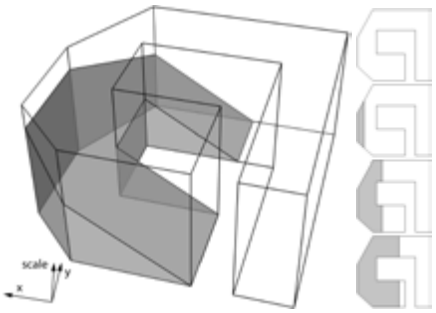
Space scale view Slices Space scale view Slices

(a)The merge operation in classic tGAP

(b)The merge operation with arbitrary slides in smooth tGAP

FIGURE 6.5 The merge operation in classic tGAP in (a) and smooth tGAP in (b). The white winner takes space from the grey loser. Slices are shown at four levels, from top to bottom in the image: from high, to low.

quality of the map. Therefore, our intention is to ensure that area features do not break into discontinuous parts.



(a)Space scale view. Only shared boundary between objects is shaded.

(b)Slices

FIGURE 6.6 More complicated shapes with arbitrary slices in smooth tGAP. The middle and low slices in (b) show two parts of the same object (white winner).

§ 6.3.2 Requirements

An important challenge 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 identified the following requirements for good smooth merge operations (each one supported by its own motivation):

I. 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

II. No new points, that is no new x, y coordinates – The construction of tilted faces make use of the already existing geometry. It follows from 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.

III. No horizontal faces – This follows from the definition of the smooth tGAP structure mentioned above as horizontal faces cause sudden changes.

IV. 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.

V. No multi-parts – Spurious multi-part objects (which should be single part), add confusion to the map.

VI. Constant steepness of tilted face/faces – The shared tilted boundary composed from multiple faces with different steepness would 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.

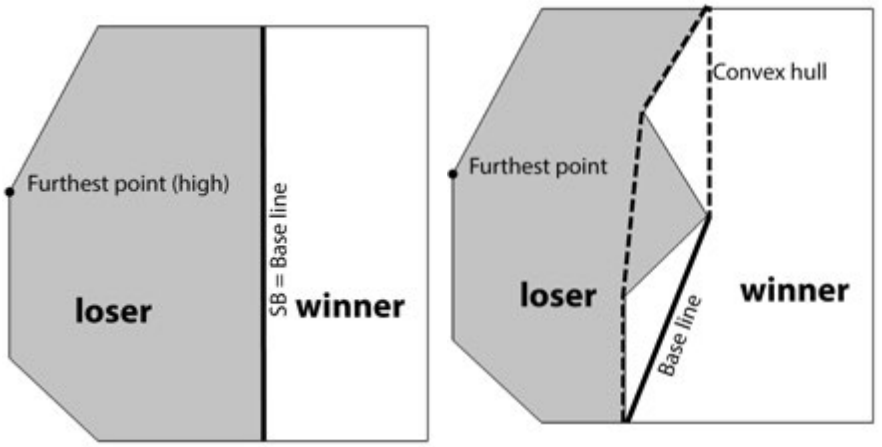
VII. Optimal shape of shared boundary – The shared 3D surface boundary can consist of multiple faces. There often exist more configurations of faces, and each defines a 3D surface of the shared boundary. The configuration that gives the best merge impression to user should be selected. That is, a natural looking boundary should remain 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 contradictory 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 for a single plane (equal steepness), but may result in multi-parts; see Figure 6.6 . The hard requirements guarantee functionality and they are crucial to finding the solution. The soft requirements are aesthetic requirements which are more like recommendations for a good map recognition (readability). We would like to fulfil them. However, they are not to be strictly satisfied, *e. g.* In some cases a loser can be composed of multi-parts.

§ 6.3.3 Three 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.

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



(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 edge of the convex hull (dashed line) of the shared boundary located in the winner.

FIGURE 6.7 Definition of the Base line for the 'Single flat plane'.

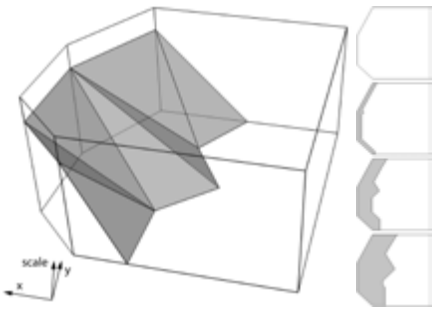
constant steepness as illustrated in Figure 6.5b. In principle, the loser (the face that has to be removed) can always be removed with a single flat plane. The plane is defined in 2D by the shared boundary, the edge(s) between the winner and the loser, and the furthest point of the loser from the shared boundary; see Figure 6.7a. 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 6.5b 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 Figure 6.7b, 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 edge of convex hull of the shared boundary located in (or on the boundary of) the winner; see Figure 6.7b.
- 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, p. 61). 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 on 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 mostly because it gives the best results in initial testing.

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 faceted 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. In addition to this, users may find objects modified by 'Single flat plane' disturbing while they use smooth zoom. This is because the boundary between the loser and winner may become a straight line during transition which can be perceive as artificial sweep line going from one side to another.

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.



(a)Space scale view. (b)Slices

FIGURE 6.8 The 'zipper' algorithm. The white winner taking over space from the gray loser. (b) shows slices at four levels from top to bottom.

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 6.8 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. We call it 'Delaunay triangulation and flipping'. It 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 more satisfying the requirements. 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 generated. Note that horizontal faces appear when a triangle connects vertices on SB only or on OB only. 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;
- 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 a good 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.

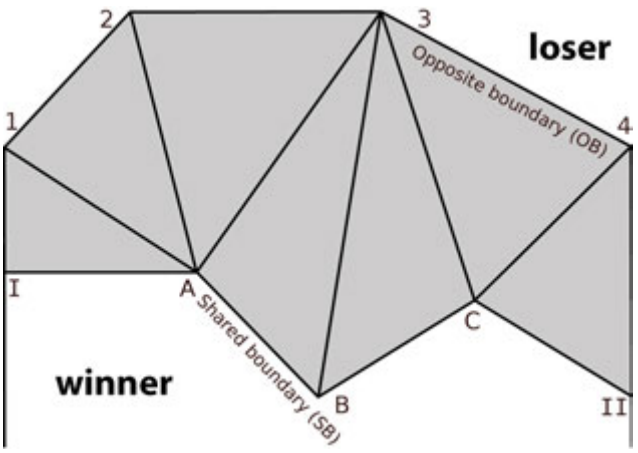


FIGURE 6.9 Principle of the 'Zipper'. The optimal solution, only the final connections are present.

Figure 6.9 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 Figure 6.9) and ends in another junction (vertex II in Figure 6.9). 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 6.10 presents an alternative visualization of the process as for a step there can be two options; 1) forward SB, 2) forward OB. 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 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.

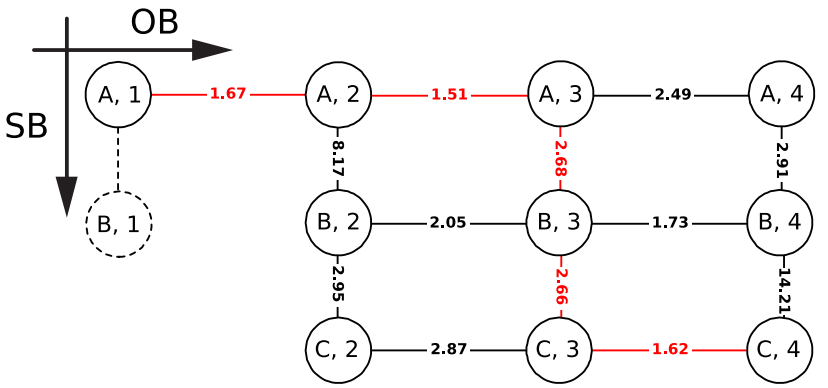
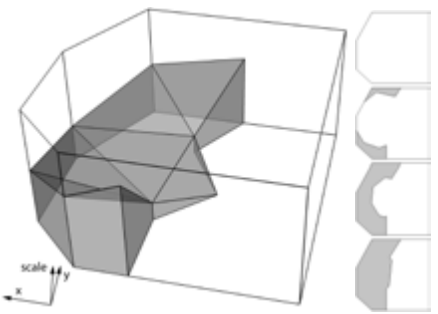


FIGURE 6.10 Graph representation of the 'Zipper' approach for configuration in Figure 6.9. The 'zipping' starts at triangle (I, A, 1) towards (II, 4, C). The process can also run in the opposite direction. The best solution is presented in red. Note that graph note (B, 1) (dashed) is not possible as triangle (B, A, 1) would be outside the loser.

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 Figure 6.10. 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.



(a)Space scale view. (b)Slices

FIGURE 6.11 Tilted faces of loser processed by the 'Eater'. (b) slices at four levels, from top to bottom in the image: from high to low.

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 triangles into a gradual change from low to high. Figure 6.11 shows the result. The Delaunay triangulation is the 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 all 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 Algorithm Eater depicts the process steps to defined the triangle in 3D space, fitting into the SSC topology. Table 6.1 presents the run of the algorithm for the case captured in Figure 6.12 results in Figure 6.11.

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 chance of multiparts. The comparison and conclusion will follow in Section 6.3.5.

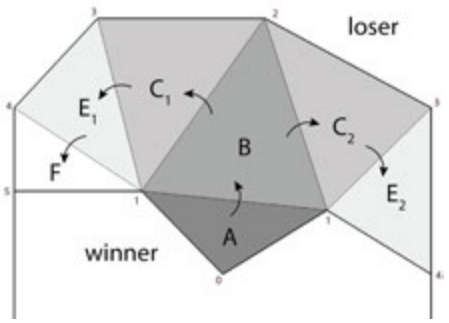


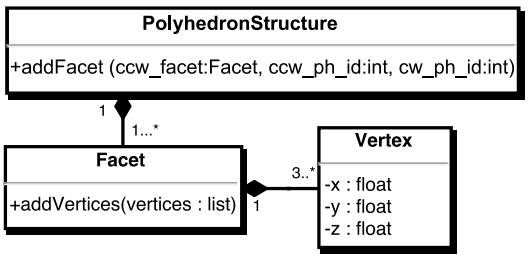
FIGURE 6.12 Principle of the 'Eater'. The triangles are walked from dark to light grey. The numbers present relative z value.

Algorithm Eater

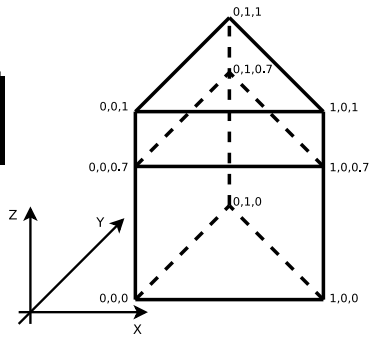
Require: All triangles of tessellated polygon
1: All triangles set to $not - visited$
2: Define starting triangle and put it to *active set*
3: Set *relative height* to 1
4: *Next active set* $= \emptyset$
5: **while** *active set* not empty **do**
6: Pop the first triangle from the *active set*.
7: **for all** vertices of triangle **do**
8: **if** vertex does not have height set **then**
9: Set vertex height to *relative height*
10: Add all $not - visited$ neighbours of triangle into the *next active set*.
11: Set the triangle to $visited$
12: **if** *active set* is empty **then**
13: *Relative height* $+1$
14: *Active set* $=$ *next active set* and *next active set* $= \emptyset$
15: **return** A set of triangles with heights for all vertices

Step in algorithm	1. Iteration		2. Iteration		3. Iteration		4. Iteration		5. Iteration		6. Iteration		7. Iteration	
6	A		B		C_1		C_2		E_1		E_2		F	
10	[]	$\{B\}$	[]	$\{C_1, C_2\}$	$[C_2]$	$\{E_1\}$	[]	$\{E_1, E_2\}$	$[E_2]$	$\{F\}$	[]	$\{F\}$	[]	{}
14	$[B]$	{}	$[C_1, C_2]$	{}	$[C_2]$	$\{E_1\}$	$[E_1, E_2]$	{}	$[E_2]$	$\{F\}$	$[F]$	{}	[]	{}

TABLE 6.1 demonstrates the principle of Algorithm Eater and shows the sequence of the traversal of the triangles. The 'Eater' ends with the triangles ordered, and a relative height assigned to every vertex. The elements in square brackets refer to the *active set*, the elements in curly brackets refer to the *next active set*.



(a)UML diagram for the structure.



(b)An example of polyhedrons.

```
ssc = PolyhedronStructure()
#bottom
ssc.add_facet(((0,0,0),(0,1,0),(1,0,0)),-1,1)
#side 1.1
ssc.add_facet(((0,0,0),(0,0,0.7),(0,1,0.7),(0,1,0)),-1,1)
#side 1.2
ssc.add_facet(((0,0,0),(1,0,0),(1,0,0.7),(0,0,0.7)),-1,1)
#side 1.3
ssc.add_facet(((1,0,0),(0,1,0),(0,1,0.7),(1,0,0.7)),-1,1)
#middle
ssc.add_facet(((0,0,0.7),(0,1,0.7),(1,0,0.7)),1,2)
#side 2.1
ssc.add_facet(((0,0,0.7),(0,0,1),(0,1,1),(0,1,0.7)),-1,2)
#side 2.2
ssc.add_facet(((0,0,0.7),(1,0,0.7),(1,0,1),(0,0,1)),-1,2)
#side 2.3
ssc.add_facet(((1,0,0.7),(0,1,0.7),(0,1,1),(1,0,1)),-1,2)
#top
ssc.add_facet(((0,0,1),(1,0,1),(0,1,1)),-1,2)
```

(c)The snippet of Python code creates two simple objects from (b) in the structure.

```
v 0 0 0
v 0 1 0
v 1 0 0
v 0 0 0.7
v 0 1 0.7
v 1 0 0.7
v 0 0 1
v 0 1 1
v 1 0 1
g 1
f 1 2 3
f 1 4 5 2
f 1 3 6 4
f 3 2 5 6
f 6 5 4
g 2
f 4 5 6
f 4 7 8 5
f 4 6 9 7
f 6 5 8 9
f 7 9 8
```

(d)An example from (b) outputted as *.obj file.

FIGURE 6.13 An UML diagram and corresponding example of the usage for 3D topological structure.

§ 6.3.4 3D storage of small dataset

The whole dataset in smooth tGAP has been stored explicitly as polyhedral volumes in a 3D topological structure, see Figure6.13. The structure records topology for a set of 2-manifold faces in 3D. The core of the structure is a list of vertices, where scale attributes are used as z values in the 3D representation. Then all polyhedrons with their facets refer to the vertex indices. Note that shared vertices among the objects are always represented just once, not duplicated.

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 Figure 6.4. It contains 676 vertices and 15 records of faces in classic tGAP structure. Because of its general applicability the 'Eater' algorithm was used for converting data stored in classic tGAP into the smooth representation.

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, 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.

When all objects of the dataset are converted to polyhedral volumes stored in SSC, a map can be obtained by horizontal slice. The continuously changing map can one imagine as gradually moving the slice plain from the top of the cube downwards. All changes result in a smooth changing 2D map: a small change in scale leads to a small change in map content.

One of the challenges here is the explicit boundary calculation of such a slice. This is computational intensive and can be very expensive due to the polyhedras' complexity.

As an alternative to this, we converted the smooth tGAP dataset to tetrahedrons (a tetrahedron is a geometric object composed of four vertices and four triangular faces) for visualization during development (Šuba et al., 2013). 1734 tetrahedrons have been stored for example above, see Figure 6.4. The ideal behind this was simple; the slicing can be done more efficiently because to make a slice of set of the tetrahedrons is much easier than making a slice of arbitrary 3D polyhedral objects.

However, the conversion to tetrahedrons adds extra computational step into the process workflow and it did not get significant speed improvement for the slicing. Therefore, we used different approach where full potential of graphic hardware can be used. Section 6.4 will cover more of this in detail.

§ 6.3.5 Comparison of three algorithms

Table 6.2 summarizes the possibility of smooth-merge operation for three algorithms, the 'Single flat plane', the 'Zipper' and the 'Eater'. Some criteria on the left side of the table come from soft requirements. Each of these criteria can be quantified and based on this the different algorithms are compared, however not all criteria are of equal importance when choosing the best solution for specific situation.

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 multi-parts and the final impression of merging may look artificial – a single straight 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 the building in Figure 6.14.

Second, the 'Zipper' offers a more generic solution. It works for all convex and even for some concave polygons, this depending 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. Higher computational complexity can

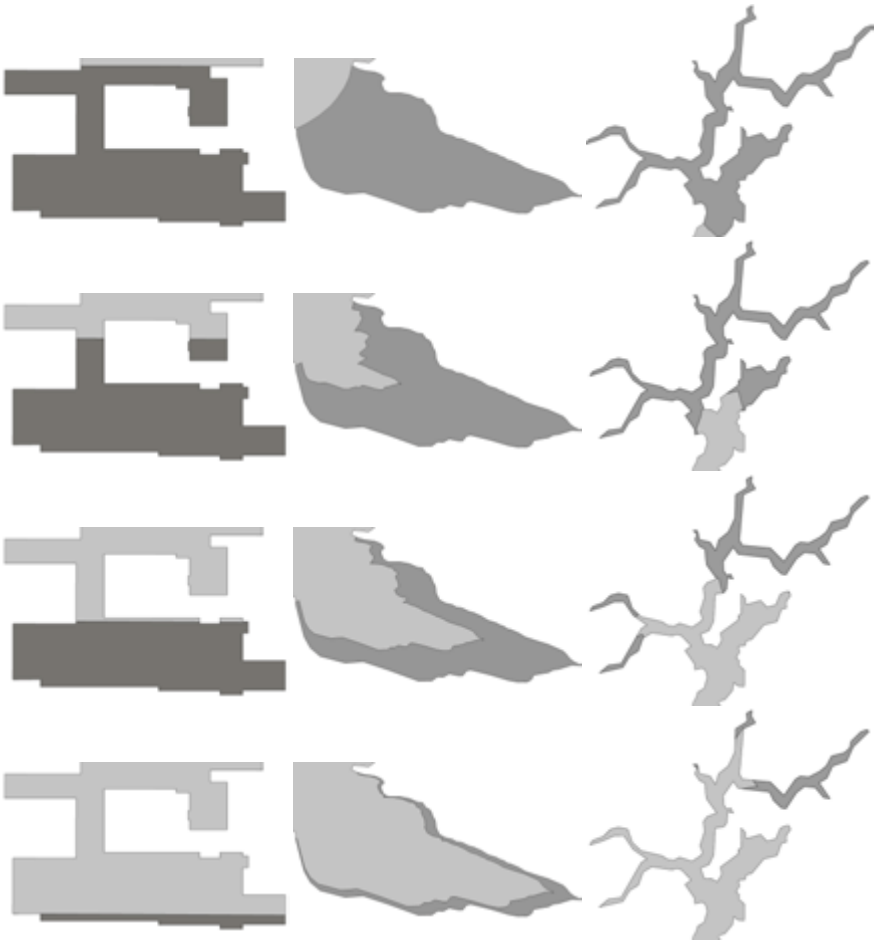
	Simple flat plane	Zipper	Eater
Always has solution	+	++	+++
Type of polygon	- (One SB only, convex, rectangular)	+ (Convex, some concave with visible SB-OB)	++ (All)
Computational efficiency	+	-	+
Multi-parts	-	+	+
Optimal shape of faces	-	++	+
Number of extra elements	+	+	-
Constant steepness	++	+	-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

TABLE 6.2 Summary table, where - = bad, + = neutral and + + = good. The first three rows evaluate the algorithms in general. The remaining rows (in *italic*) give an overview of fulfilment of the mainly aesthetic soft requirements.

be minimized by an alternative graph visualization, where optimal solution lies closer to the diagonal, and therefore some options can be omitted. 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 $\mathcal{O}(n \log n)$ 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. The negative effect of multi-parts can be reduced by including more information in algorithm; *e. g.* knowledge of a triangle configuration in a previous step would lead to a better configuration for the current step. On top of that, steepness can be improved by global information about configuration of triangles. This could also improve specific cases where the loser has many holes and 'branches', and where some 'branches' are eaten faster than others.

To be able to use smooth SSC dataset generated the way mentioned above, specific types of slicing/rendering software is needed. Only then we could perform its full potential and observe the influence on the user if there any. Therefore, the next section will describe our developed prototype written as an OpenGL application supporting smooth interaction with proper user interface.



The 'Single flat plane' – example of complex building

The 'Zipper' – example of a dam

The 'Eater' – example of a river.

FIGURE 6.14 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 (top) to more merged (bottom)). The light grey face is the winner, the dark grey face is the loser.

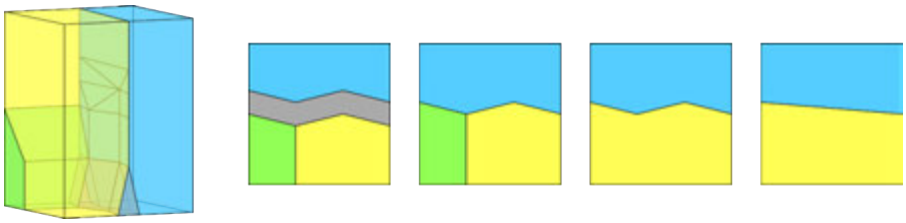


FIGURE 6.15 Example SSC data with horizontal intersections. In this example, the colours blue, grey, yellow and green respectively represent water, road, farmland and forest terrains.

§ 6.4 GPU based vario-scale viewer

As we mentioned before, the concept of map generalization research of vario-scale has shifted towards a truly smooth vario-scale structure for geographic information, see 3.4. It has the following property: A small step in the scale change leads to a small change in representation of geographic features that are represented on the map. In the implementation, the tGAP representation of 2D geo-information can be converted to a full 3D representation (smooth SSC), see Figure 3.9b on page 31.

The original SSC proposal included an explicit boundary calculation to perform the slicing with a plane. The intersection of the plane with the SSC's polyhedra yielded the set of polygons constituting the vectorial map representation. This is geometric computation intensive and not well suited for parallel execution, due to the very different workloads, depending on the polyhedras' complexity.

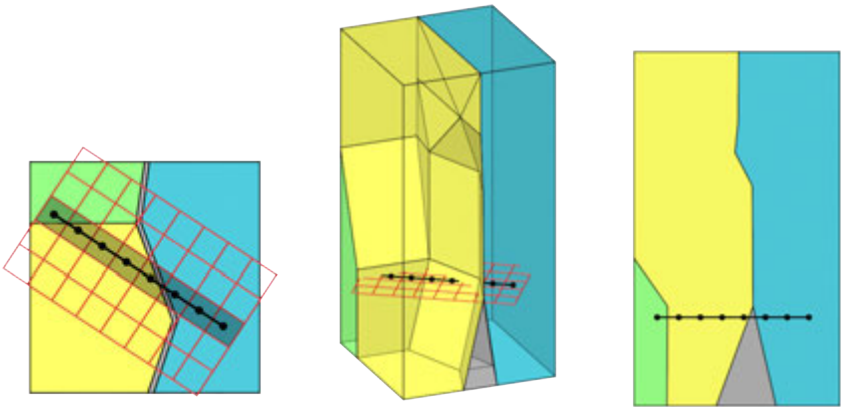
In a spirit similar to our work, some approaches (Guha et al., 2003; Hable and Rossignac, 2005) avoid calculating explicit intersections. Whenever the intersection is needed they derive only a pixel-precise location, which is sufficient. They do this by taking advantage of graphics hardware to do calculations at the level of pixels instead of primitives.

The following section describes the concept of the SSC intersection pixel rendering approach and how it is implemented on a GPU. We will use the term 'Intersector' for the implementation of the method later in this section. It describes the principle for better understanding of the environment in which the usability test would take place.

It is important to mention that this section is based on MSc thesis of Mattijs Driel, where implementation of the prototype is one of his contributions to the vario-scale project. The section covers principle of the Intersector in § 6.4.1 together with more technical details in § 6.4.2 and followed by addition rendering techniques in § 6.4.3. Even more detailed information about Intersector can be found in (Driel et al., 2016).

§ 6.4.1 Concept

Space Scale Cube representation assumes that every terrain feature represented in the data is a polyhedron (or 3D mesh) with a unique ID, and the terrain features together



(a) Raster placement viewed where the plane is intersecting.

(b) Raster placement in 3D.

(c) Side view on one row of pixels, represented as their centers.

FIGURE 6.16 The intersection method's goal is to determine what 'colour' to give each pixel in a raster. To do this, the center points of the pixels are tested for what colour polyhedron it is inside. This example highlights a single row of pixels in the raster, and determines that in this row, 1 pixel shows forest, 4 show farmland and 3 show water. Because of the low resolution and placement of the raster, the road is not visible.

form a partition of space, so each terrain feature fits tightly to its neighbours. To get a specific intersection out of the SSC, imagine a 2D pixel raster is placed somewhere inside the SSC where we want to know for each pixel what terrain feature it is intersecting. In essence, this gives us a displayable result as it is already a 2D raster. An example of this is presented in Figure 6.16.

We first explain an alternative naive intersection method, as a way to explain by contrasting it with our own. The naive method will go over all pixels and for each pixel, do a point-in-polyhedron test for all polyhedra in the data. With the degenerate case of a pixel lying on a polyhedron boundary, simply pick one of the polyhedra, the difference will hardly be visible in a full image. This method, though inefficient, does guarantee that each pixel in the raster finds the right terrain feature.

Our intersection method builds on this naive method, as it also starts with a pixel raster. We can notice how projecting each pixel along any direction will first encounter the inside boundary of the correct terrain feature: the one intersecting the pixel.

Building on this observation, we view the SSC orthogonally from the top downwards. We then render the insides of the polyhedra below the raster, while simultaneously clipping off everything above the raster.

What we essentially do here is to pick the z-axis as the projection axis, and by setting the view as orthogonal, project all pixels along the same axis. Then, rendering only below the raster will prevent any geometry above from interfering with the result. An example of this is shown in Figure 6.17.

There is one caveat in doing this method. We need to be able to assume that each pixel on the raster is in a closed polyhedron. However, because the SSC model represents a partition of space, we assume this to be the case.

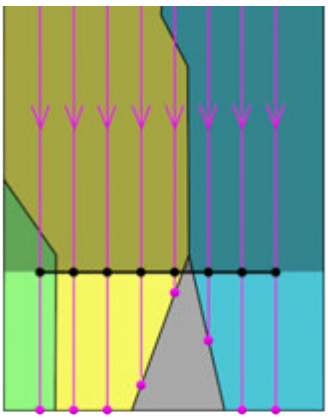


FIGURE 6.17 By projecting the points downwards we find the correct colours for the pixels. The upper shaded area must be explicitly ignored, otherwise some invalid colours will appear. In this example, if the shaded region was not clipped the most left (the first) and the fifth pixels would incorrectly get assigned yellow and blue respectively.

§ 6.4.2 Implementation

The main argument for using this raster based intersection method over one that does explicit plane-polyhedron intersections is that we only need an image as an end result. This is ideal for the GPU, specifically because it fits with the capabilities of the graphics pipeline.

Views in graphics are often a product of transforming the input vertex data into a box shaped clipping region. The clipping region represents the visible area on the x-y range, and the z-axis gives the range in which depth sorting can occur. Everything outside of the clipping region is clipped off. For the orthogonal transformation required by our intersection method, we need an orthogonal transformation matrix that specifies where in space our view is positioned.

The rendering process is initiated by feeding the GPU terrain features where the polyhedra are represented as 3D triangle meshes. Keeping with the method, we only render inside-facing triangles (method called front-face culling). Triangles are transformed from world to screen coordinates. Then pixels are computed based on orthogonal downwards viewing resulting in so-called 'fragments'. These are points that correspond to a pixel in the raster, but also maintain their post-transformation z-position used for depth sorting. We discard all fragments that appear above the z-position of the raster. We also use depth sorting on all fragments below the raster, so only the fragments closest to the raster are not discarded.

The remaining fragments store the integer ID of their corresponding terrain features into a pixel buffer. The buffer can be used to sample single IDs directly when a user wants to view information on a terrain feature under a mouse or touch cursor. Other than that, the buffer can be used as input for a full screen colour mapping technique that maps IDs to colours, which gives us our final displayable image.

§ 6.4.3 Additional rendering techniques

Our approach opens up some other possibilities, in addition to vario-scale and smooth zooming functionality, that are impossible with the traditional 2D map rendering approach: supersampling antialiasing (for sharper images), non-horizontal intersections (for mixed scale representations), near-intersection blending (for dynamic movement), and texture mapping (for enhanced map readability).

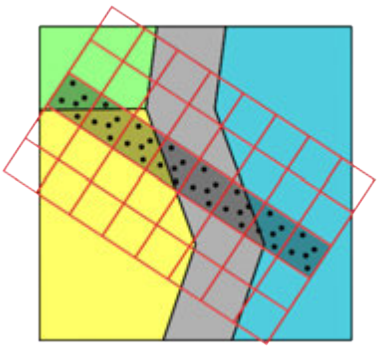


FIGURE 6.18 Using a form of SSAA, multiple samples are taken at every pixel, and their colors blended.

ANTIALIASING 2D approaches using vector based images would apply a form of antialiasing specific to vector image drawing, which is not an option for our approach. We instead use a form of supersampling antialiasing (SSAA) by re-rendering the image a few times with sub-pixel offsets and merging the result in a blending operation (Fuchs et al., 1985). Because of the static nature of maps, we can spread the re-renders out over consecutive frames to increase the image quality over time. The principle is illustrated in Figure 6.18.

NON-HORIZONTAL INTERSECTIONS As we have mentioned, the intersection method clips off geometry above the intersection plane by testing each fragment's z-position against that of the raster's plane. This implies a constant z value over the entire raster, making the intersection horizontal. We can make this more flexible by instead testing each fragment with a pre-calculated z-value that is stored in a raster sized buffer. This buffer can be prepared using any sort of function $z = f(x, y)$, implying any intersection surface that doesn't fold over in the z-axis is a valid input. Figure 6.19 shows an example using a curve shaped surface, which results in high detail near centre and lower detail near sides.

A non-horizontal intersection surface would display low and high abstraction in the same image. This could serve a function in highlighting specific areas, some examples of which are implemented in the prototype.

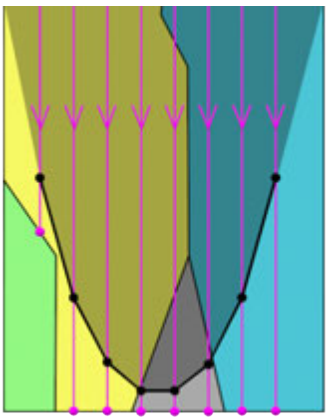


FIGURE 6.19 In this example, the center of the raster requires less abstraction than the raster edges. As a result, the forest becomes invisible and the road visible.

NEAR-INTERSECTION BLENDING When quickly zooming through much information with our approach, some changes might still feel abrupt; *e.g.* one area removed and aggregated with neighbour. Near-intersection blending provides a way to further smoothen the image by blending two terrain features that are adjacent on the z-axis. This can be conceptualized by imagining a second intersection surface placed c units below the primary intersection surface (see Figure 6.20). Any polyhedron boundaries between the two surfaces can be said to be near to the actual intersection with a distance of $d = \{0, c\}$. Below the boundary, a different terrain (and thus a different colour) is defined. If we blend the colours of both the intersecting and near-intersecting terrains with a factor of $f_{blend} = \frac{d}{c}$, we smooth the result.

In the intersection method, remember that we render the insides of terrain feature polyhedra to get the intersecting terrain. If we were to render the outsides instead, we would encounter the outside of the terrain feature directly below the intersecting terrain. Also, note that we know from the fragment what the z-position is of the boundary between the intersecting and near-intersecting terrains. We can utilize this as follows: we could do both inside and outside renders and store them. Then during colour mapping we can blend the two together with the blending factor described above.

The result is a smoother transition between terrains while actively zooming, which is visually more comfortable to look at. An example of this is shown in Figure 6.21. It does not influence anything other than the rendered colour, terrain IDs are kept intact.

TEXTURE MAPPING In the final color mapping step of the intersection method, we have used simple flat colours corresponding to the terrain feature to display for the final image. When including near-intersection blending, at most two colours are blended. Instead of, or in addition to flat colors, we can use a texture. As texture mapping characteristic we can simply use all data known to a fragment: the xyz-coordinate and the terrain feature ID.

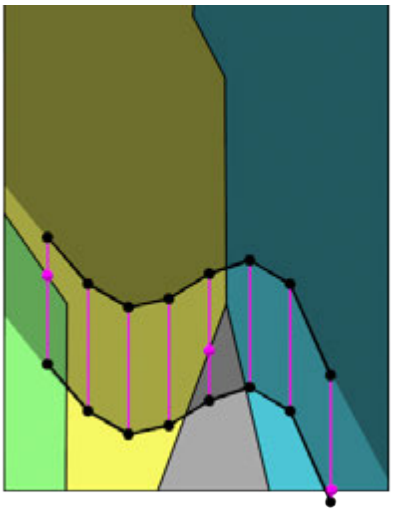


FIGURE 6.20 The three marked points represent an intersection within the given distance. The left and middle points will blend yellow-green and yellow-grey respectively. The right point intersects on the SSC boundary, so blending will not occur there.

A possible use of this is to give some types of terrain an extra visual hint to their properties (such as a tree symbol texture to a forest, or wave symbol texture in water). A simple number texture example is shown in Figure 6.22. Because the z-coordinate is also usable in the mapping, the texture can be rendered at specific ranges of abstraction levels, giving a user direct visual feedback on the level of abstraction currently visible.

§ 6.4.4 Implementation details

The Intersector prototype was written as an OpenGL ES 3.0 application in Java. Because of the use of the limited OpenGL ES feature set, the application should be possible to use on mobile phone hardware with limited efforts.

The set of intersections available for demonstration show the main classes of shapes. Shapes include a horizontal surface, a non-horizontal planar surface, a sine curve surface, a 3D mesh object based surface, and a control point surface. These shapes are shown in Figure 6.24, along with some examples of them intersecting the rural region dataset. The control point surface is based on inverse distance weighting (Shepard, 1968). This method is an example of direct real-time manipulation of the intersection shape itself.

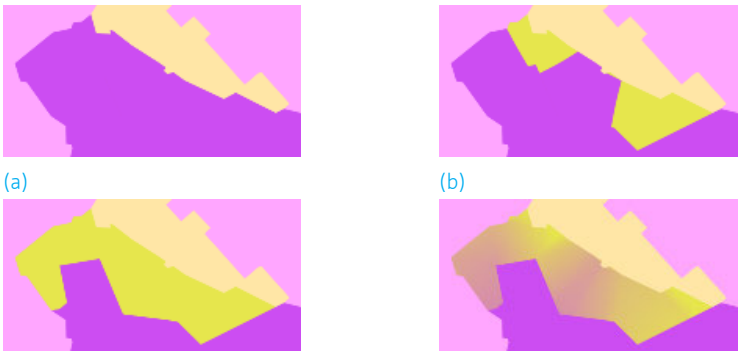
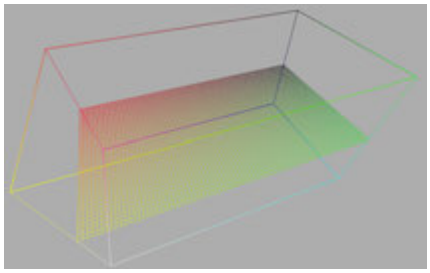


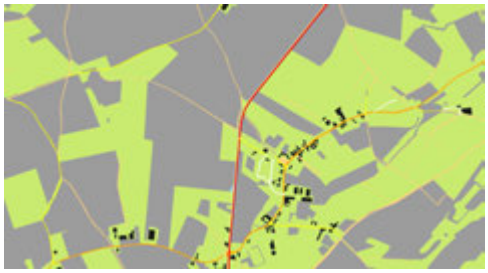
FIGURE 6.21 This example shows how a transition from one colour (a) to the next (c) can be abrupt (here shown as purple to yellow), even when the transition is non-horizontal (smoothed, or non-instant) (b). This is most noticeable while in motion, scrolling through abstractions. Blending the colours (d) can make the transition seem less abrupt.

FIGURE 6.22 Texturing is shown as the numbers rendered in the regions. They fade on the sides because of the non-planar intersection used to generate the image, showing that abstraction variations influence the result.

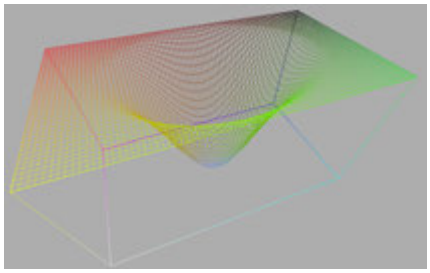
FIGURE 6.23 Subset of the testing data (source: TOP10NL, near Maastricht, NL), shown at the lowest level of abstraction.



Horizontal raster.

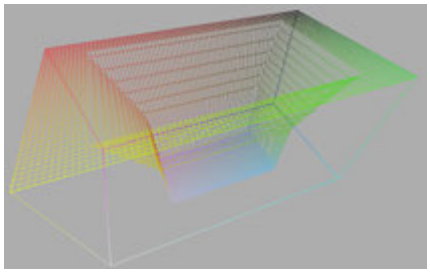


Abstraction equal everywhere.



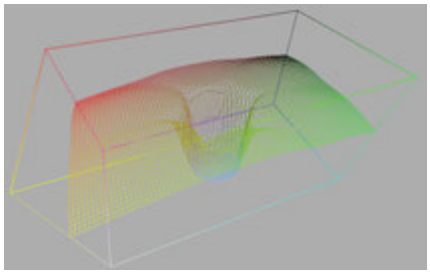
Sine curve raster.

Abstraction lower in the middle.

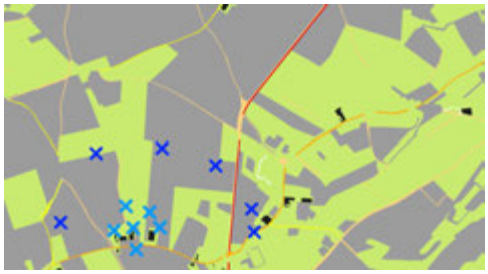


3D Mesh input raster.

Abstraction lower in the middle.



Inverse distance weighting derived raster.



Abstraction lower around the bottom left buildings, where control points (colored crosses) are placed. Dark blue and light blue crosses respectively indicate high and low abstraction. Note that the centre of the raster was shifted to the bottom left corner to generate this map.

FIGURE 6.24 Rasters on the left produce corresponding intersections shown right. With lower abstraction, more buildings and roads become visible.

§ 6.5 User's experiences

With the Intersector prototype implemented the next step was to perform a usability test with users even though it was not in our original research plan. After short round of experiments we discovered that the prototype is too complex (with more functionalities than needed) and with a non-intuitive interface. Therefore, we developed more optimized 'light' version of the Intersector and performed initial basic usability test. Here we describe initial test with preliminary results. It was carried out with a small group of users using the 'light' Intersector prototype. Only the core of our vario-scale concept was tested mainly for time reasons. For additional testing more participants, additional functionality (non-horizontal intersections, colour blending, etc.) and different SSC content with a (range of generalization techniques and options to create the structure) can be introduced. Our usability test plan included the following elements: user profiling, scope, apparatus, procedure & task and usability evaluation methods.

USER PROFILING An important initial analysis was user profiling to limit the research to a well-defined group of users. The age range of the potential user group could vary from 18 to 70 years, with no restrictions on their nationality, gender and education level. Their previous experience of map use were not important. The experiment was drawn upon the scenario of a map user observing unknown map location.

SCOPE This usability test should indicate if vario-scale maps can provide better understanding and more intuitive perception of the data compared to a fixed and limited number of maps for discrete map scales (the more traditional multi-scale approach).

The main scope was to test two types of content. The 'light' Intersector was the tool which helped us demonstrate our vario-scale approach. To eliminate the influence of the different working environment the experiment was carried out with the same 'light' Intersector prototype, that is same GUI, but for two datasets. A classical discrete multi-scale dataset (SSC content where at limited number of fixed scales the objects change: all prisms ending and succeeded by new set of prisms) and a vario-scale dataset (classic SSC based on tGAP content, see Section 3.4 on page 28), both covering the same geographic extent. We assumed that users would perform better using vario-scale maps, especially for specific tasks concerning dynamic changes of the map content such as a zooming operation. This leads us to a series of questions that the usability testing should answer:

- Can users get better understanding of data with our new vario-scale representation?
- Can users navigate faster and more effective with vario-scale maps?
- Could users also be confused by vario-scale maps and if so: how often, when and why?
- Do users benefit (more pleasant map reading experience and better quality execution of tasks) from vario-scale concept?

APPARATUS All techniques described in the previous sections were implemented as a prototype to demonstrate their potential, but the implementation was not optimized to provide the most user friendly experience, which has had a negative effect on the

results of the experiment. For example, the user performs badly because he is irritated or confused by the set of mouse interactions and key strokes needed. Therefore, we implemented and used the new 'light' Intersector prototype. We reduced controls to four keys for movement and four keys for zoom in and out operation in two speeds (no mouse). Specifically, *arrow keys* for movement, *dash* (-) and *equal* (=) keys for slow, *detail* and *open bracket* ([) and *closed bracket* (]) for fast, *coarse* zooming in and out.

Note that the 'light' Intersector is implemented such a way that it can generate high rate of frames per second (FPS). Our working assumption is that 30 FPS (Read and Meyer, 2000) (in our solution can be even higher) is sufficient to provide good user impression. Smooth experience is provided by fast interaction; one key press gives immediately one new map (direct jump to target frame, throwing away some of the possibilities of our data structure). More in detail based on smooth zoom aspects, see Section 6.2 on page 88, one zoom operation is composed of one *rescaling*, one *content zoom* and one *graphic technique* (antialiasing) integrated in one new map/frame. Additionally, we offer zooming in two speeds *detail* and *coarse*, where only different is a bigger step in the scale.

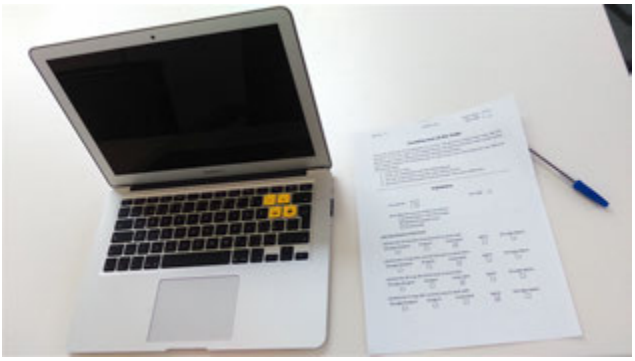


FIGURE 6.25 The experiment set-up.

For executing the experiment one notebook, powerful enough[1] to run the 'light' Intersector was used by one test person (TP) in a session of maximum 15 minutes. The 'light' Intersector ran as a window fixed in size, only showing a map view (no additional controls on screen). The equipment were set up in a quiet room to create a comfortable experiment environment where the TP accomplished the task, see Figure 6.25. The instructor was present throughout the experiment for giving an introduction and in case any unexpected situations occurs.

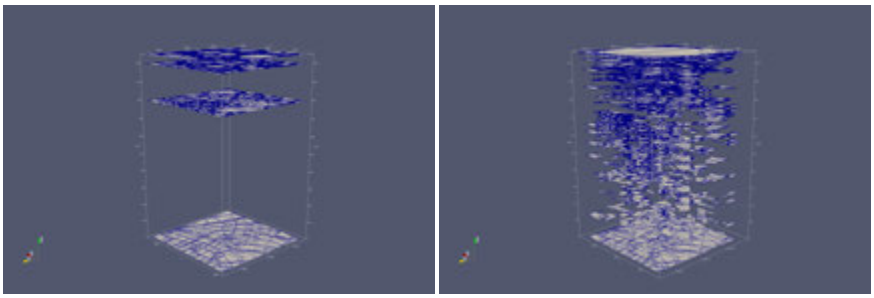
The test methodology consisted of a questionnaire, audio-video observation and synchronized screen recording. In addition, the screen geographic coordinates and scale denominator were recorded, with a time stamp. When the user was performing unusual actions, *e.g.* navigating very fast or extremely slow the TP was asked think aloud. All this allowed further detailed investigation of the TP's behaviour, thinking and potential hurdles during the test session.

1 MacBook Air, Early 2015

dataset	source	tGAP	Intersector	
	no. faces	no. faces	no. prisms	no. triangles
multi-scale	13.200	37.000	27.000	2.114.000
vario-scale			129.000	9.624.000

TABLE 6.3 Two datasets used in experiment. The source is topological planar partition. TGAP represents generated structure based on the source. Intersector represents generated testing datasets from tGAP where number of prisms indicates tGAP faces converted to 3D prisms.

For this experiment, we generated two main datasets; First, discrete multi-scale, represents a classic approach, where 7 discrete scales are generated based on the data factor 2 (in each dimension), *i. e.* level 0 corresponding scale denominator 1:10,000 with 13,200 features, level 1 corresponding scale denominator 1:20,000 with 3,300, level 2 corresponding 1:40,000 with 825, etc. Second, a vario-scale dataset, which captures our approach. Both were generated from the same tGAP structure. This structure has been created based on a subset of Dutch topographic map data (TOP10NL) intended for use at a 1:10,000, in regions of 9×9 km, see Table 6.3. Figure 6.23 shows map design used in experiment on a subset of the data. The structure has been generated with merge and split operations. Figure 6.26 shows both dataset used in the experiment in 3D view.



(a)Multi-scale (b)Vario-scale

FIGURE 6.26 The test datasets in 3D view.

PROCEDURE & TASK Before the experiment the TP was welcomed and the experiment was briefly introduced. Then the TP spent a few minutes randomly using the 'light' Intersector to 'get a grip' on the working environment. It was practised on similar dataset at different location of area 1×1 km at first. Later, we used a bigger practice dataset of area 7×7 km. This was done mainly to eliminate the factor of a new unfamiliar environment.

Several types of tasks based on map reading were specified: orientation, searching, analysis, routing, and planning tasks. Since we wanted perform the experiment within 15 minutes per person, it would not have been realistic to have performed all the tasks. Therefore, we chose only the following task:

I. Orientation-task: zoom out completely to see whole extent of the dataset and then try to get back as accurately as possible to the same location and zoom level as started. The time and the precision (the position of the screen) is captured.

We believed that this task had the highest chance to test our hypothesis and it can be performed multiple times in limited time to provide statistically sufficient data.

In more detail, when the experiment started, the environment was reset. The TP started with the practise dataset where controls and the task were demonstrated. When the TP felt ready we changed to the vario-scale or multi-scale dataset (half of the TPs started with the multi-scale data and other half started with vario-scale data to eliminate learning effect). Then, the TP performed the following task, as fast as the TP could three times for two different datasets (6 $\times$ total):

- TP pressed the key (s) to start. Intersector showed randomly generated zoomed in location.
- TP zoomed out completely to see the whole dataset.
- TP used the *arrows* keys to move the map 'off the screen'.
- TP got back to the initial location and zoom level. Then TP pressed *space* key (final position was tagged)

At the end of the session the TP had been asked filled the questionnaire to indicate their preferences. The experiment was concluded by explaining what the difference between the datasets are (the TPs did not have any understanding before) and answering additional questions.

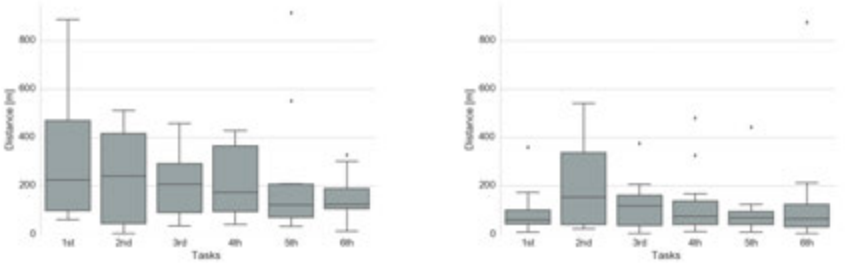
USABILITY EVALUATION METHOD The experiment was captured in questionnaires, see Appendix 7.3 on page 143. The screen geographic coordinates and scale denominator were recorded, with a time stamp. On top of that audio-video recording of both the screen and the TP took place. Furthermore, when the user was performing unusual activity, we asked the TP to talk out-loud.

The aim of the questionnaire was to get a more extended profile of the TP together with his/her opinion. It consisted of questions about the TP his/her gender, age and how often they use maps. Another part of questionnaire concerned the validation of the session, where self-reported participant ratings for satisfaction, *i. e.* ease of use of the two types of data. The TP rated the measure on a 5 point scales and it was used for quantitative measures. In last part of the questionnaire they indicated likes, dislikes, suggestions and recommendations to provide subjective measures and their preferences. This data can be then used for planning additional usability testing in the near future.

Quantitative metrics were covered by the duration of the task, the amount of time it takes the participant to complete the tasks, and accuracy of the user's performance, *i. e.* the difference in distance between real world position of starting and ending point for a given task. On top of that, the recording was intended to reveal the TP's thinking and the reasoning behind the TP's actions to give an indication where possible confusion arises.

§ 6.6 Measures

We collected data from 10 TPs in the first group and 13 TPs from the second, see more in Appendix 7.3 on page 145. We captured all tasks in the order in which they have



(a) Results after training with a small practice dataset (1 × 1 km) for the first group of TPs (10 persons).

(b) Results after training with a bigger practice dataset (7 × 7 km) for the second group of TPs (13 persons).

FIGURE 6.27 precision of the tasks in the same order as they have been presented (regardless of type of the dataset).

been performed in Figure 6.27a. A box plot graphs[2] have been used. It shows that the results are affected by the process of learning *i. e.* the users are less accurate and spend more time on the first task and they perform better later in the experiment. We assumed that this is influenced by the practice dataset, where users did not experience the environment enough. To improve this we introduced bigger practice dataset in a rural region (more similar to the testing location) of the larger area 7 × 7 km. Then the same experiment was run for the second group of users (13 TPs), see Figure 6.27b.

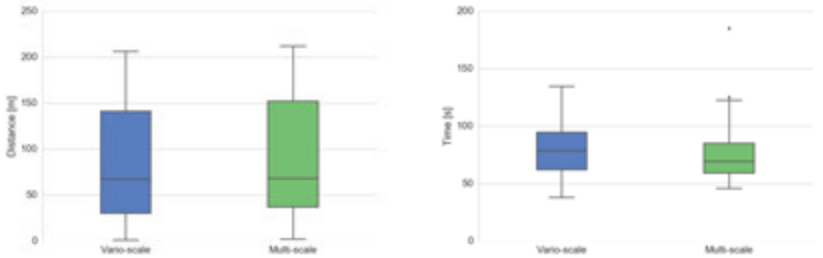
Figure 6.27b demonstrates that overall precision with the bigger practice dataset drops under 180 m in average and indicates overall more precise measurements (with significantly smaller Interquartile Ranges). It means that practise dataset had significant influence on the results. Therefore, only more precise results of the second group with 13 TPs will be considered, see Figure 6.28.

Moreover, every task is composed of three phases; (1) Zoom out, where user started at initial location and zoomed out to see the whole dataset at once. (2) Move (panning), where user used the arrow keys to shift the dataset 'out of the window' and back. (3) Zoom in, where user zoomed in to the initial location again. Figure 6.29 gives a better notion how much time is spent in which phase for the second group.

§ 6.7 Lesson learnt – smooth zoom

For initial user testing the 'light' intersector prototype has been designed in such a way that a smooth experience is provided by fast interaction; one key press gives immediately one new map. Assumption behind this was that fast response would provide the best smooth impression where the user gets new map as fast as requested. Additionally, the user's confusion is reduced because zooming in and out is less time consum-

2 The box extends from the lower to upper quartile values of the data, with a centre line at the median. Lines extending vertically from the boxes (whiskers) indicate variability outside the upper and lower quartiles. Outliers are plotted as individual points.



(a) Distance between starting and ending location

(b) Time for task

FIGURE 6.28 The results for the second group of TPs (13 persons).

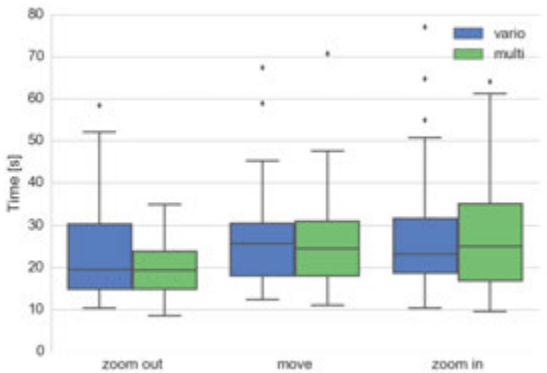


FIGURE 6.29 Three phases (zoom out, move and zoom in) of the task for vario-scale and multi-scale datasets.

ing. Operation in two speeds was provided; fast, *coarse* to quickly browse throughout the scales and slow, *detail* to perceive all small map content changes. Note that one zoom action resulting in one new map (frame) includes following smooth aspects: one *rescaling* (scale, translation), one *content zoom* (change of content) and one *graphic technique* (antialiasing).

However, the results of user testing indicates something different based on the screen recording and responses filled in the questionnaires, see Appendix on page 145. The users used only *coarse* type of interaction most of the time. Some characterised the *detail* zoom as "annoying slow zoom". This meant they could never get a smooth impression (small changes in content). The same is reflected in the responses because users could not distinguish between vario-scale (small changes in content) and multi-scale dataset. From the experiences gained here we may now define *smooth zoom* operation giving the best smooth user impression. The elements which should be included are following;

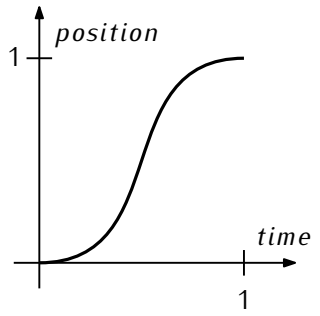
MORE STEPS FOR CONTENT ZOOM The configuration when one pressed key gives immediately one new map is not optimal. More precisely, a *content zoom* ratio of $1:1$ between user interaction and number of generated steps is not favourable. From initial testing it seems that every independent zoom action should return map content in more steps. This can be done by causing a *content zoom* multiple times, changing the map content only bit by bit (leading to more temporary maps, showing the transitioning of the map content in small steps from m_0 to m_1 , *e. g.* m_0, m_0', m_0'', m_0''',... , m_1), and thus progressively refining the map.

The optimal number of small steps in *content zoom* is not know. However, if we consider 30 FPS as optimal (see Apparatus on page 113), and assume one zoom action should take approximately $1/3$ of a second, which makes 10 frames for one zoom operation. Then relation $1:n$ for *content zoom* where $n \approx 10$ is favourable.

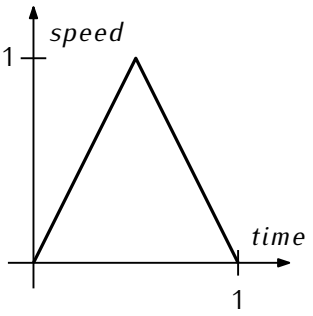
SPEED PERCEPTION The perception of the speed is another aspect that needs to be considered to provide smooth experience to the users. The illusion of speed can be simulated by the frames (temporal maps) and the timing in which they are presented, *e. g.* timing of a *content zoom* sequence m_0', m_0'', m_0''', m_1. This is similar to the process of generating intermediate frames between two images to give the appearance that the first image evolves smoothly into the second image known from animation as *inbetweening* or *tweening* (Penner, 2002).

However,if the frames are presented with long pauses between them, then the transition between consecutive maps (frames) feels stiff, artificial and mechanical. Penner (2002) claims that the notion of acceleration, a change in speed, is needed. The term *easing* is used as the transition between the states of moving and not-moving. When an object is moving, it has a velocity of some magnitude. When the object is not moving, its velocity is zero. There are many options for the shape of the acceleration. Extensive list of examples for *easing* can be found, see (Penner, 2002).

Figure 6.30 shows example of the most common easing for animation. The curve shown produces the most natural looking motion where the object speeds up from an initial still position, then slows down and stops at the end. Lifts, for example, use the same *Ease-In-Out easing* curve (Penner, 2002). We believe that similar kind of tweening together with techniques mentioned above can significantly improve the user's



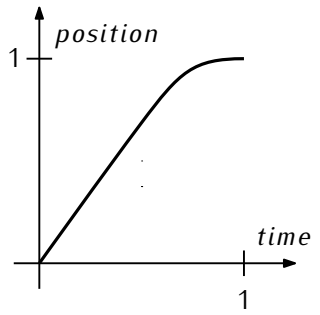
(a)Ease-In-Out easing, an example of the lift. The graph captures the situation when the object accelerates first, then slows down and comes to a stop at its destination. Source: (Penner, 2002).



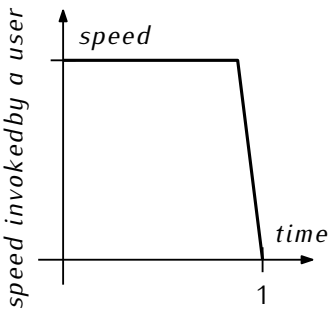
(b)The same situation as in (a). Here, the focus is on speed over the time during the Ease-In-Out easing.

FIGURE 6.30 Perception of speed by using *Ease-In-Out easing*.

overall interaction in the map. Based on our experiences so far we propose an optimal tweening for panning in Figure 6.31. Similar design can be used for zooming in and out.



(a)Our proposed tweening for map interaction. Then user's action invokes linear acceleration first slowly goes to a stop.



(b)The same situation as in (a). Here, the focus is on speed over the time during the interaction. Note that initial interaction speed depends on user's action, *i. e.* speed invokes by a user's action is equal to initial interaction speed.

FIGURE 6.31 Proposed tweening for panning which improves the user's interaction in the map.

DEPENDENT ON UI Zoom operations were provided in two speeds; fast, *coarse* and slow, *detail*. However, both operations were implemented the same way; a key press makes the map change by constant change of scale . One can imagine this as a moving the slicing plane (and scaling) in the SSC by ϵ scale where ϵ is constant, see Figure 3.9 on page 31.

The results of the testing shows that users have different preferences in zooming speed which are not really reflected in user interface (UI) *i. e.* users can only browse throughout scales by using *coarse* or *detail* zoom.

We propose to use other available UI elements supporting more levels of freedom (*e. g.* duration of action), for instance, mouse wheel or touch screen. This will makes change of scale more variable and dynamic. It will lead to the movement of the slicing plane in the SSC by ϵ where ϵ would be variable based on the duration of the action, how far it scrolled (for mouse wheel) or the size of the finger gesture (on touch screen). Then the steps for *content zoom* should be chosen and presented accordingly.

The discussion above indicates elements of smooth zoom action. An optimal use case can be described as follows: When the user performs a action such as a roll of mouse wheel or makes finger gesture a new series of content zoom steps of size ϵ of SSC is created. This series is *rescale* and is combined with *graphic techniques* to create a series of pictures/frames. Then, this sequence of frames is visualized to provide a very smooth transition from one scale to another, similar to animation – hence the term smooth zoom. Note that all this happens in a short amount of time (*e. g.* 300 ms).

§ 6.8 Reflection

In this chapter, we have tried to find indicators that the vario-scale approach can provide significant improvement in map use for users. We presented our concept of zooming together with three options how the smooth map content can be generated. Moreover, new SSC-based visualization prototype was presented. On top of that we carried out initial, preliminary usability test to indicate or verify our design assumption that vario-scale approach could provide a better user experience. This helps understand zoom action and its effect on user experience. We reflected this in Section 6.7, where also smooth zoom operation is designed.

From initial test can draw following conclusions; Figure 6.28 shows that difference between vario-scale and multi-scale in user performance. Figure 6.28a shows the vario-scale method to be more accurate ($8m$ in average). Figure 6.28b, on the other hand, presents the vario-scale as being slower ($12s$ in average). Overall, with the current number of test persons and small differences between methods we cannot reach any general conclusion. The consequence is that the initial experiment in this particular configuration is inconclusive.

We believe that the influence of other factors such as the environment, generated datasets, styling, GUI, etc. was more significant and have stronger impact on test (more than multi-scale versus simple vario-scale content). These factors cover most of the measures and, therefore, the differences between the two datasets are not obvious. Therefore, the next usability testing should included all aspects in their further developed stage. Only then further usability testing should take a place. Here the list of our hypotheses (together with our *SUGGESTIONS I* (*I* a sequence number)) why the results are not really conclusive follows:

- The majority of the TPs used only *coarse* zoom in big steps. This means that users perceived the generalized data in big steps *i. e.* They did not really experience smooth map content, but made scale jumps.
  SUGGESTION 1: Improve the zoom functionality. Users should use more convenient UI to control smooth zoom operation which includes more steps of content zoom properly integrated with rescaling and additional graphic techniques creating series of frames. These frames properly displayed would provide perception

SUGGESTION

of motion leading to better user interaction as we described in Section 6.7. It is important to keep in mind that users tend to prefer tools that they normally use, therefore, improved zoom functionality should feel "familiar".

- Since the part of the user's task is move the map 'off the screen' the users cannot keep track of their initial location. This means that they have to memorize the original position and keep it in mind ('create mental map'). Later when they see the map again they have to recall that mental map again. This makes the task more a memory exercise than a testing of the map content.

SUGGESTION *SUGGESTION 2:* The testing dataset should be significantly bigger.

SUGGESTION *SUGGESTION 3:* Other types of task (not memory based) should also be considered, see next bullet.

- Besides orientation task used in experiment there are other following tasks which can be consider in future:

  II. Searching-task: Locate a specific object (*e. g.* landmark/house) by zooming and panning: e.g. find church on central market square in town X.

  The time and the correct identification of the church is captured.

  III. Analysis-task 1: What is the size of central market square in square meters?

  The time and the closeness to real size of square is captured.

  IV. Routing-task 1: Find the largest lake within 500 m route via the road, starting from the church (a more nearby lake via 'the air' may be possible, but it should be checked if this can be reached via road and zoom/out in needed).

  The time and the correct identification of the lake is captured.

  V. Routing-task 2: Go to a specific house/location in a different town.

  The time and the correct identification of the house/location is captured.

  VI. Analysis-task 2: Estimate distance between two churches that are somewhat further apart.

  The time and the closeness to the real distance is captured.

  VII. Planning-task: Design a running track of about 10 km, which includes among others about 20% urban area and 30% forest tracks.

  The time and the correct quality (length and composition) of the track is assessed.

- Some faces in the structure changed their classification multiple times and then later in process changed back again. This creates impression that faces are flickering which is distracting and users have a harder time to create their mental maps, similarly to previous bullet.

SUGGESTION SUGGESTION 4: The generalized dataset should be improved, methods such as grouping may help, see Section 4.6. In addition to this, the next vario-scale dataset should be a smooth-merged SSC see Section 6.3. This way, the classification change would feel more gradual.

- The map generalization is applied in such a way that buildings disappear, but no build-up area is generated. However, the buildings are one of the most significant features in the map, because they have distinct colour (black) and specific

shape. This generated an unusual situation for users, when they lost a significant orientation point.
Our recommendation will be same as *Suggestion 4*

- The datasets for the prototype contain road features. All road segments were selected from the structure, converted to triangles and stored in the prototype dataset. This method is not optimal because it increases the amount of the data drastically.

  Suggestion 5: The road features should be handled as line geometry in the prototype (used directly by GPU) instead of set of triangles as it is now. Suggestion

- We used a classic SSC model generated by the non-smooth version of merge/remove and split/collapse operations. However, some users have had difficulty to distinguish between datasets.

  Suggestion 6: The next version of a vario-scale dataset should be smooth SSC using smooth version of generalization operations namely; 1) merged/remove operation, see Section 6.3, 2) split/collapse and 3) line simplification. This way, changes of geometry would feel more gradual and the difference between vario-scale and mutli-scale datasets will be more significant. Suggestion

- So far we have developed our solution on a desktop. The next logical step should be mobile usability tests. Is should be conducted after developing a mobile implementation of the Intersector and creating other tasks for a mobile user in a real world setting. This also might include finger gestures for zoom, pan and rotate in a natural manner. Additionally it suggests a challenging solution for the 3D viewer with proper streaming on the Internet.