Delft University of Technology

Web-based dissemination of continuously generalized space-scale cube data for smooth user interaction

Meijers, Martijn; van Oosterom, Peter; Driel, Mattijs; Šuba, Radan

DOI
10.1080/23729333.2019.1705144

Publication date
2020

Document Version
Final published version

Published in
International Journal of Cartography

Citation (APA)
Meijers, M., van Oosterom, P., Driel, M., & Šuba, R. (2020). Web-based dissemination of continuously generalized space-scale cube data for smooth user interaction. *International Journal of Cartography*, *6*(1), 152-176. https://doi.org/10.1080/23729333.2019.1705144

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Taylor & Francis
Taylor & Francis Group

International Journal of Cartography

ISSN: 2372-9333 (Print) 2372-9341 (Online) Journal homepage: https://www.tandfonline.com/loi/tica20

Web-based dissemination of continuously generalized space-scale cube data for smooth user interaction

Martijn Meijers, Peter van Oosterom, Mattijs Driel & Radan Šuba

To cite this article: Martijn Meijers, Peter van Oosterom, Mattijs Driel & Radan Šuba (2020) Web-based dissemination of continuously generalized space-scale cube data for smooth user interaction, International Journal of Cartography, 6:1, 152-176, DOI: 10.1080/23729333.2019.1705144

To link to this article: https://doi.org/10.1080/23729333.2019.1705144

© 2020 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group

Published online: 13 Jan 2020.

Submit your article to this journal

Article views: 293

View related articles

View Crossmark data

Full Terms & Conditions of access and use can be found at
https://www.tandfonline.com/action/journalInformation?journalCode=tica20

INTERNATIONAL JOURNAL OF CARTOGRAPHY
2020, VOL. 6, NO. 1, 152–176
https://doi.org/10.1080/23729333.2019.1705144

OPEN ACCESS Check for updates

Web-based dissemination of continuously generalized space-scale cube data for smooth user interaction

Martijn Meijers[a], Peter van Oosterom[a], Mattijs Driel[b] and Radan Šuba[a]

[a]GIS Technology, Faculty of Architecture and the Built Environment, Delft University of Technology, Delft, Netherlands; [b]Twnkls, Rotterdam, Netherlands

ABSTRACT
The Space-Scale Cube (SSC) model stores the result of a generalization process, that supports smooth scale transitions for map objects. The third dimension is used to describe geometrically the smooth transitions between objects at different levels of detail. Often-used map generalization operators fit in this SSC model. The 3D SSC model to derive 2D maps can be used in a mobile web client, where these days powerful graphics hardware is available. This article shows the steps needed for producing and disseminating SSC data with smooth transitions over the web. Firstly, we explain how SSC data can be obtained and subsequently be rendered by making effective use of the GPU. Secondly, we show how we organize data in chunks and how this 'chunked' data can be used for efficient communication between client and server. In the third place, we describe which operations can be used on the client side for deriving maps. Fourthly, the SSC also allows for (a) mixed abstraction slicing surfaces useful for highlighting specific regions by showing more detail and (b) near-intersection blending, which helps to prevent abrupt transitions while the slicing surface is in motion. Finally, we show how animated pan and zoom functionalities may be realized. A set of prototypes allows us to disseminate the data with smooth transitions on the web and in practice judge the effect of continuous generalization and animating the map image.

RÉSUMÉ
Le modèle du Cube Echelle-Espace (SSC) stocke le résultat dun processus de généralisation qui permet des transitions d'échelle fluides pour les objets cartographiques. Les opérateurs standards de généralisation cartographique s'intègrent dans le modèle SSC. Le modèle 3D SSC pour dériver des cartes2D peut tre utilisé dans un client web mobile. Cet article montre les étapes nécessaires pour produire et distribuer des données SSC avec des transitions fluides sur le web. En premier, nous expliquons comment les données SSC peuvent tre obtenues et restituées par une utilisation efficace du processeur graphique (GPU). Puis nous montrons comment organiser les données en morceaux et comment ces morceaux' peuvent tre utilisés pour une

ARTICLE HISTORY
Received 23 March 2018
Accepted 11 December 2019

KEYWORDS
Map visualization; continuous generalization; real time; interaction; web mapping

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

This article has been republished with minor changes. These changes do not impact the academic content of the article.

© 2020 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group
This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

communication efficace entre le client et le serveur. En troisième lieu, nous nous décrivons les opérations qui peuvent être utilisées côté client pour dériver des cartes. En quatrième point, nous montrons que le SSC peut aussi permettre (a) d'extraire des coupes de surfaces de niveaux d'abstraction mixtes et (b) de créer des mélanges proche des intersections afin d'éviter des transitions abruptes lorsque une coupe de surface est en mouvement. Enfin, nous montrons comment les fonctionnalités de déplacements et zoom animés peuvent être réalisées. Un ensemble de prototypes nous permet de diffuser sur le web des données avec des transitions fluides et en pratique de juger de l'effet de la généralisation continue et de l'animation de la carte.

1. Introduction

In the last decade, the use of interactive maps has entered the daily life for many, through the use of smartphones or other electronic devices. Applications using maps are in general quite flexible and responsive; a map browsing application should show a new view immediately when a user is panning (moving around) or zooming in and out, resulting in a feeling of high responsiveness. However, there is a lingering issue in interactive cartographic maps, which is found in the interaction of zooming. Zooming here is defined as scrolling through a limited set of maps with increasing levels of abstraction, with more abstraction showing lower levels of detail. Hence, zooming in and out implies picking one level from the set and showing it over a larger or smaller area respectively. The issue is that this set of pre-defined maps is limited. Using this model, it is impossible to show data in between two adjacent levels, because there simply is no data present here. Current web mapping applications have implemented workarounds for this fact, by either snapping to the pre-defined levels, or by introducing graphic blending between levels. However, the fact is that geometric data is missing for how the map objects should look between the fixed levels.

To fill in this gap, people have worked on continuous generalizations (Chimani, van Dijk, & Haunert, 2014; Nöllenburg, Merrick, Wolff, & Benkert, 2008; Peng, Wolff, & Haunert, 2017; Sester & Brenner, 2009). Hence, we want to obtain a web mapping solution that allows to use the result of these continuous generalization algorithms. The solution should work well in a server–client architecture, where the load is shared between server and clients in a balanced way and also the hardware of the client is effectively used. This means that the data for the system has to be transferred effectively across the web. As Dumont, Touya, and Duchêne (2017) note: 'Moreover, usual visualization systems for multi-scale maps are regularly based on tiled grid structured as a quad tree. Adding intermediate representations may require adaptations of these systems or even new solutions'. Next to that, we also want to have interaction that feels 'smooth' for the end user, i.e. with interaction (zooming) the content of the map should gradually change.

The Space-Scale Cube (SSC) as introduced by Meijers and van Oosterom (2011); van Oosterom and Meijers (2013) is an alternate model for representing cartographic data. van Oosterom and Meijers (2013) mention the need of intersecting such an SSC with a surface to produce a presentable map and the effects of non-horizontal and curved surface intersections resulting in mixed-scale maps. Up to now, this model was used as a conceptual model, without any technical implementations. This paper presents an

approach that allows intersections of SSC based datasets to be rendered in real time on a client machine. In summary, this paper presents the following contributions, it shows:

- How we obtain SSC based data and how the theoretical model has been refined in order to be implemented.
- The general approach of deriving maps in real time using rendering techniques by intersecting the SSC conceptually with a slicing surface.
- Performing intersections with non-horizontal and curved surfaces to produce mixed-scale maps (high and low level of detail in one map).
- SSC data transfer over the web based on chunks.
- Exploration of functionalities needed for rendering interactive maps based on the SSC model to enable animation after user interaction.

The remainder of this paper is structured as follows: Section 2 describes how data for an SSC can be obtained. Section 3 explains the proposed architecture for deriving maps from an SSC in detail. Section 4 describes the prototypes, each testing/trying a different aspect of the proposed architecture. Section 5 summarizes what we learned from the prototypes and gives ideas for future work. Section 6 concludes the paper.

2. Obtaining continuously generalized space-scale cube data

The Space-Scale Cube (SSC) is a 3D data cube that uses the third dimension to represent Level of Detail (scale) instead of storing a stack of 2D multi-scale maps (van Oosterom & Meijers, 2013; van Oosterom, Meijers, Stoter, & Šuba, 2014). Figure 1 shows an example SSC. In this model, map objects change from 0D points into 1D lines, 1D lines

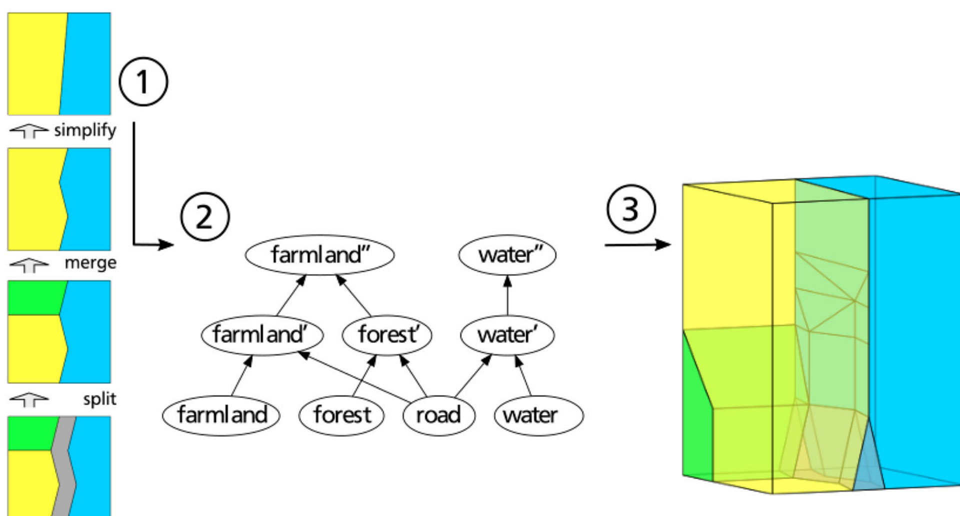

Figure 1. Example of how to obtain content for a Space-Scale Cube (SSC): ① from a sequence of generalization operations, ② a Directed Acyclic Graph (DAG) can be derived and ③ the DAG and step-wise generalization results can be converted in a 3D data cube, the SSC. In this example, the colours blue, grey, yellow and green respectively represent water, road, farmland and forest terrains.

into 2D surfaces and from 2D polygons into 3D polyhedron objects, i.e. to every map object the range of map scales the object is intended to be used for is added as extra dimension.

To produce content for a Space-Scale Cube (SSC) a step-wise generalization process is run, where generalization operations are iteratively applied starting with the largest scale base map that contains a planar partition (in which point, line and polygonal map objects are embedded). A directed acyclic graph (DAG) structure is used to store the result of and the parent-child relations between the map objects as result of generalization operations. van Oosterom (2005) introduced the tGAP data structure (topological Generalized Area Partition) to store the DAG and information on the (boundaries of the) map objects. Meijers, Šuba, and van Oosterom (2015) show how the automated generalization process using the tGAP structure can be achieved for large data sets.

Figure 1 shows a sequence of generalization operations recorded as DAG and then converted to 3D SSC. First, a road object is split over its three neighbours, then the forest area is merged into neighbouring farmland and finally the boundary between farmland and water area is simplified. The key to obtain an SSC is to convert the DAG structure into a cube with 3D geometry (where the *z*-component of every vertex indicates at what map scale the geometry is supposed to be used).

The 2D geometry of the map objects and the intended range of map scale of each map object (as *z*-component) forms the input for this conversion. The DAG structure supplies information on the generalization operations that were applied. The input is converted into faces that connect map objects at two different map scales. We call these faces the trans-scale boundaries of the embedded map objects. The trans-scale boundaries (i.e. the tilted/vertical faces in the SSC) of the objects describe how the geometry of the map objects changes over map scale, i.e. how the map objects are gradually generalized.

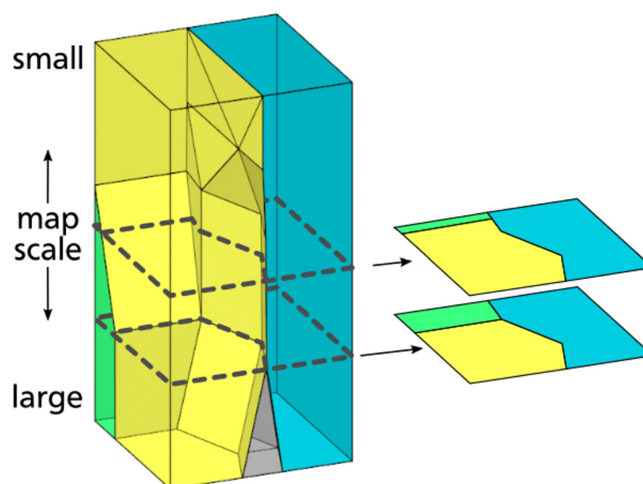

Figure 2. A 2D map from the SSC can be obtained by slicing: calculating the resulting intersection of the 3D cube with a 2D surface.

The SSC, that stores the results of the generalization steps, leads to what we call a vario-scale data structure from which maps can be derived. Figure 2 illustrates that in order to get a presentable 2D map from the SSC, one needs to calculate the intersection at the desired level of detail. Conceptually, maps are created by intersecting the cube with a finite surface (further referred to as 'slicing surface'). We intend to make the result of continuous generalization visible on the resulting maps as gradual as possible. In this way, 'a small change in map scale leads to a small delta (change) in geometry of the objects represented on the map, and when delta scale approaches zero, the delta in the map approaches zero' (van Oosterom & Meijers, 2013).

Within the resulting cube, each area object is stored as 3D polyhedron, described by its boundary ('boundary representation'). Each 3D object records the transitions that will be shown to the end user by means of its trans-scale boundaries. The trans-scale boundaries permit us to define the exact and smooth change in geometry of a terrain feature when the shape and position of the slicing surface changes. The geometry of the trans-scale boundaries in the cube has a huge influence on how the user will be able to perceive the result of a generalization operation. Figure 3 illustrates how the object on the left (grey) is merged with the object on the right (white) by applying this algorithm. The grey object disappears from the map when the slicing surface is moved up, while the white object takes over its space. Dependent on the geometry of the trans-scale boundaries (either using completely vertical or tilted trans-scale boundaries) the result can appear at once (Figure 3(a)) or in a more gradual fashion (Figure 3(c)). Šuba, Meijers, and van Oosterom (2016) have introduced some algorithms to obtain the trans-scale boundaries, for two area objects that are merged. Figure 4 shows that the outcome of a line simplification algorithm, for which the order in which vertices are removed from a polyline is stored, can be reconstructed as trans-scale boundary, showing a gradual change in geometry of the polyline. Map browser applications using an SSC are thus able to show smooth transitions when a user is zooming, all while displaying the intended data correctly at any level of detail.

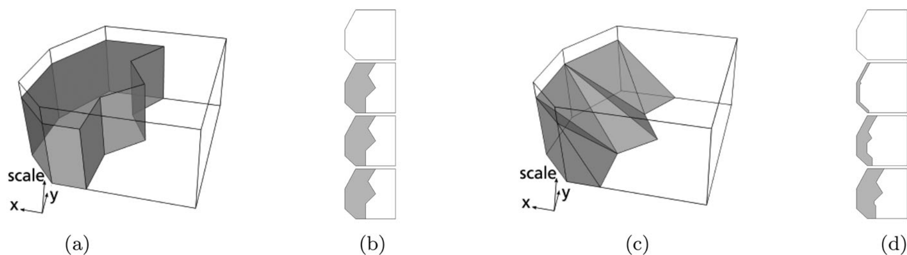

Figure 3. Comparing vertical with tilted trans-scale boundaries. (a) Vertical trans-scale boundaries between grey area object on the left and white area object on the right. Note that the grey object has a horizontal top (at which it disappears from the map). (b) Maps obtained by slicing at 4 different locations in the cube. Slices as viewed from the top. Outcome of merge appears at once. (c) Tilted trans-scale boundaries describe how the grey area object on the left is merged gradually to the white area object on the right. (d) Maps obtained by slicing at 4 different locations in the cube. Slices as viewed from the top. Outcome of merge is presented gradually to the end user.

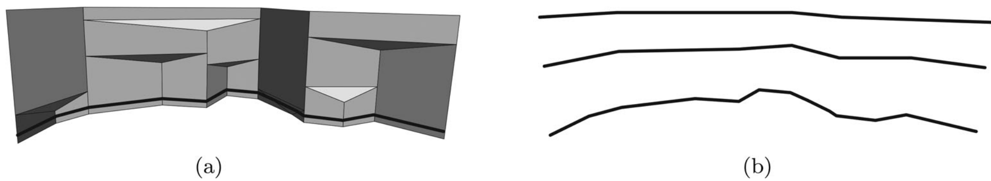

Figure 4. When a polyline is simplified, the outcome of the simplification can be stored as trans-scale boundary. (a) A trans-scale boundary for a polyline simplified with the Visvalingham-Whyatt algorithm (the original polyline is visualized with the black line at the bottom of the trans-scale boundary). Each time a vertex is removed from the polyline, this results in a triangle in the resulting transscale boundary. (b) Slicing at different locations through the resulting trans-scale boundary, leads to a polyline with less (top) or more (bottom) vertices. Result of slicing operation shown in perspective view.

3. Visualizing and interacting with SSC data on the web

This section will explain the design of a possible architecture that makes it possible to visualize and interact with SSC data in a web environment. In a web environment, the SSC data is stored on a server and a client is used where a user can retrieve and interact with the retrieved SSC data. Dissemination via the web with a fat client has our preference, for the possibilities this offers. A fat web client (with powerful graphics hardware) has the following advantages for rendering vector data:

- The graphics hardware on the client side can be effectively used by using WebGL. WebGL is available within all mainstream browser implementations and there is no requirement for installing specific desktop applications. Also on mobile devices (telephone/tablet) these browsers are available.
- It is possible to style the map interactively (e.g. change colours).
- Rotating the map interactively by means of a cursor or finger gestures becomes possible, as well as making perspective renderings of the map.
- Interactivity with objects is possible, for example highlighting an object, after a user has clicked on it.

Section 3.1 explains how with the graphics hardware of the client map images can be produced (rendered). Moreover, Section 3.2 explains three additional rendering techniques (non-horizontal intersections, near-intersection blending and texture mapping). Section 3.3 gives an overview of how SSC data can be requested in parts (chunks of data) from a server so that retrieval also works for datasets of nationwide extent. Finally, Section 3.4 gives a description of how panning and zooming of the map image can be animated improving the gradual appearance of the map.

3.1. Intersecting an SSC in real time

Zooming and panning a 2D map in the SSC approach is conceptually realized by intersecting the 3D data cube with a surface patch of which the user can influence the shape and its location. As an end user will expect to see the results of any modification of the slicing surface in real time, it is of utmost importance that the 2D image of the resulting intersection can be obtained fast.

One possible route to obtain the resulting intersection (2D map) is to calculate explicit boundary intersections. A dataset based on the SSC model, represented as closed polyhedra, is algebraically intersected with a surface resulting in a collection of polygons. The polygons can then be fed to a vector rendering application to produce the final image. The complexity of this algorithm is determined by the amount of polyhedra the surface is intersecting with. However, the polyhedra are likely to have differing amounts of faces. Moreover, because the input data is non-uniform, it is difficult to take advantage of efficient parallelization. The algorithm does not fit the Single Instruction Multiple Data (SIMD) paradigm well (Flynn, 1972), which is the currently preferred paradigm in graphics hardware. One idea to make a SIMD approach more feasible was to represent the polyhedra as tetrahedra. This would make the data uniform and thus fit better with graphics hardware. Even though this could work, it will not use the GPU rendering pipeline which requires triangle primitives. Taking a broader perspective on visualizing intersections, some domains have faced very similar problems. Constructive Solid Geometry (CSG) uses solids to represent manufactured products and virtual environments. Solids are defined as a series of boolean operations on geometric primitives (e.g. spheres, cylinders). Computing the result of the boolean operations is necessary to visualize the solid. This computation is quite costly. Therefore, a variety of works (Guha, Krishnan, Munagala, & Venkatasubramanian, 2003; Hable & Rossignac, 2005) avoid calculating the explicit result of the boolean operations when pixel precision is sufficient, which is whenever the result needs to be displayed on a screen. They use a rendering approach that takes advantage of graphics hardware and avoids the need of computing explicitly the exact outcome of the boolean operations. The rendering approach proposed here uses similar ideas from this domain: For our purpose, we can avoid the need of computing the intersection result explicitly.

Every terrain feature represented in the SSC data is a polyhedron (or 3D boundary representation, with triangles) with a unique identifier (ID), and the terrain features together form a partition of space, so each terrain feature fits tightly to its neighbours. To get a specific 2D map out of the SSC imagine a set of grid-aligned points that is placed on the surface patch inside the SSC where we want to produce a 2D map. Figure 5 presents an example. For the 2D map, we will need to know for each grid point what terrain feature it is intersecting to determine its colour. This will give us a displayable result as the grid is equivalent with a 2D raster. A naive rendering approach could go over all grid-aligned points and for each point, do a point-in-polyhedron test for all polyhedra in the data. With the degenerate case of a grid point lying on a polyhedron boundary, we simply pick one of the polyhedra, the difference will hardly be noticeable in a full image. Although inefficient, this brute force approach does guarantee that each grid point finds the right terrain feature (and the associated pixel can be coloured accordingly).

Figure 6 shows that our intersection approach builds on this naive approach, as it also starts with a set of grid-aligned points on the slicing surface. Notice that when you shoot a ray from the top of the SSC downwards, the ray can intersect multiple polyhedron boundaries. One should take the first intersection point of the ray below the slicing surface (with the polyhedron at the side closest to the slicing surface determining the colour) to determine the colour for a particular grid point in the resulting map image. Based on this observation, we render (produce an image of) the visible insides of the polyhedra below the slicing surface, while simultaneously clipping off (discarding) all geometry above the slicing surface.

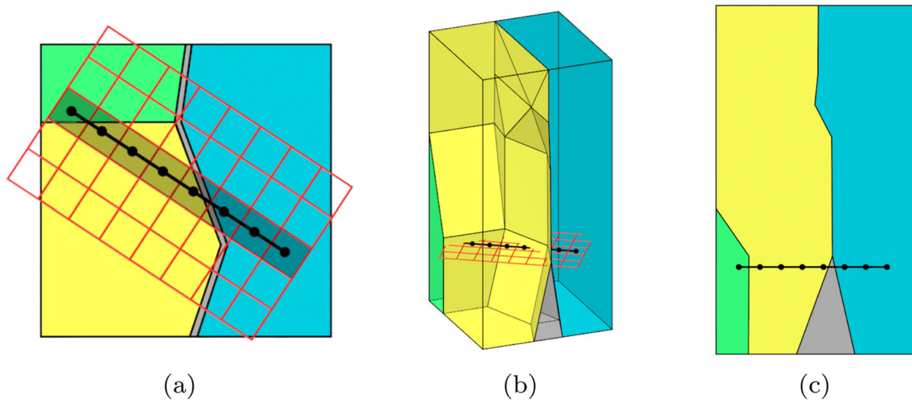

Figure 5. The rendering approach its goal is to determine what 'colour' to give each pixel in a raster. To do this, the centre points of the pixels are tested for what colour polyhedron it is inside. This example highlights a single row of grid points in the raster, and determines that in this row, 1 pixel shows forest, 4 shows farmland and 3 shows water. Because of the placement of the points, the road is not visible. (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.

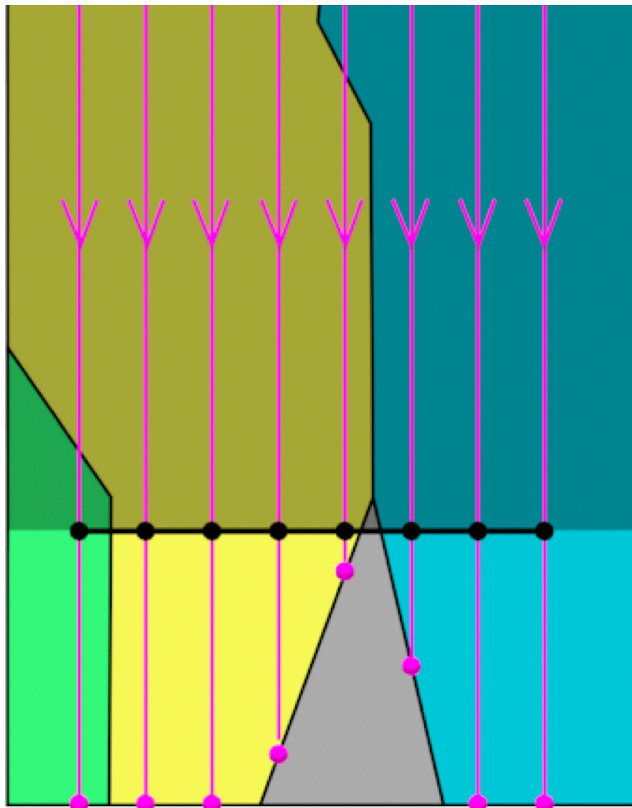

Figure 6. By shooting a ray through the grid points downwards we find the correct colours for the pixels. The greyed area must be explicitly ignored, otherwise some invalid colours will appear. In this example, if the greyed region was not clipped the first and fifth pixels would incorrectly get assigned yellow and blue respectively.

For use on the graphics hardware, the boundary representation of the SSC polyhedrons needs to be converted to a structure that is based on triangles. Figure 7 shows that the rendering process consists of a series of steps (the so-called rendering pipeline). The process is initiated by uploading to the GPU the trans-scale boundaries of the SSC, represented as properly oriented 3D triangles. For producing the 2D raster image, we also send an orthogonal transformation matrix to the GPU. This matrix specifies for which part of the SSC we want to produce the image of the insides of the polyhedra. The triangles are transformed with the transformation matrix to a box-shaped clipping region. The triangles that overlap fully or partly with this box-shaped region and that represent the correct sides of the polyhedron boundaries (which is inferred from the triangle orientation) are converted into so-called 'fragments'. Fragments are grid-aligned points, which next to the ID of the associated polyhedron, also maintain their *z*-coordinate (from which the relation to the slicing surface can be determined). We can discard all fragments that appear above the slicing surface. Subsequently, we can also sort all fragments below the slicing surface, to keep only the fragments closest to it. The remaining fragments store the integer ID of the corresponding terrain polyhedra. We convert this set of fragments from ID to colour, to obtain our final 2D raster image. This information can also be used to sample single IDs directly when a user wants to view information on a map object under a mouse or touch cursor location.

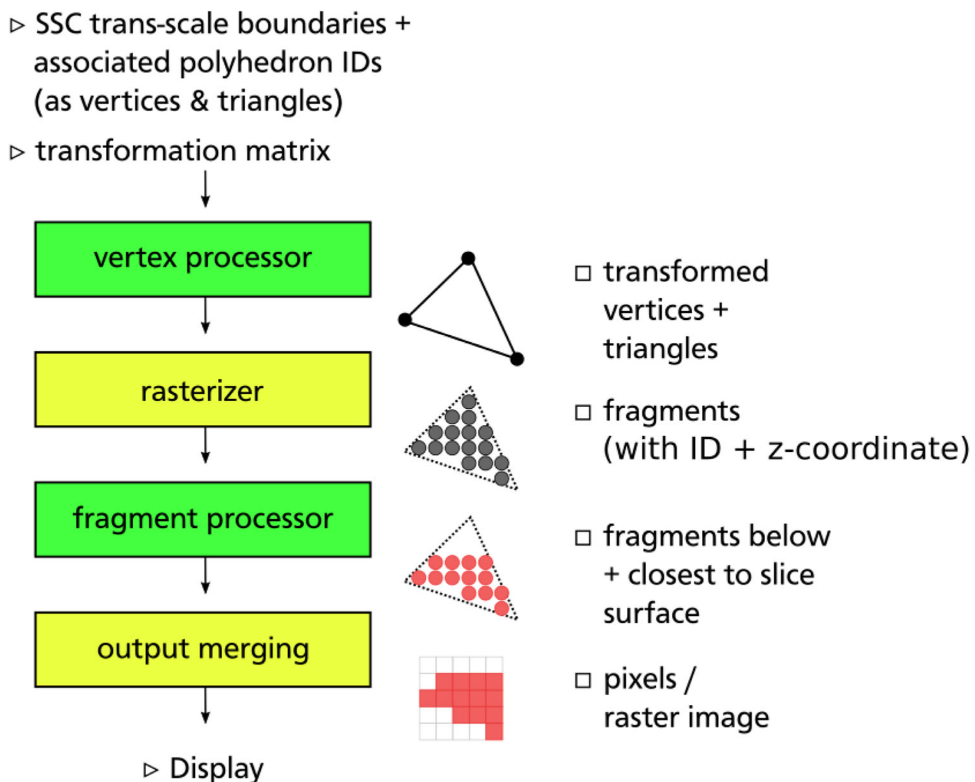

Figure 7. Intersecting an SSC in real time. SSC data runs through some steps on the GPU (the rendering pipeline) before a map is displayed to the end user. Note that the steps in green are programmable and can be modified by the programmer. Image after Hock-Chuan (2019).

3.2. Additional rendering techniques

Section 3.1 explained that by intersecting the SSC with a horizontal planar surface patch we can produce and visualize a map with one Level of Detail (map scale). Here we will explain three additions: using non-horizontal surface patches to intersect the SSC, blending colours while the slicing surface is in motion and mapping textures to the interiors of displayed polygons.

3.2.1. Non-horizontal intersections

As we have mentioned, we discard the SSC geometry above the slicing surface by testing each fragment its *z*-position against that of the *z*-location of the slicing surface. This implies a constant *z*-value over the entire slicing surface, making the intersection horizontal. We can make this more flexible by instead testing each fragment with a pre-calculated *z*-value that is stored for every grid point. This grid can be prepared using any sort of function $z = f(x, y)$. This implies that any slicing surface that does not fold over in the *z*-axis (scale) is a valid input. The grid of the slicing surface is uploaded to the GPU, to make it available for the fragment processor to determine which fragments to consider. The end user can be given tools in the user interface that allows to modify the shape of the surface (e.g. the geometry of the slicing surface can be obtained by means of control points and interpolation). Figure 8 shows an example of a side-view using a curve-shaped slicing surface. A non-horizontal or curved slicing surface will display low and high level of detail in the resulting map image. This serves a function in highlighting specific areas.

3.2.2. Near-intersection blending

When quickly zooming over a large set of scales with our approach, some changes could appear abruptly. Near-intersection blending provides a way to further smooth a temporary map image by blending the colours for two terrain features that are adjacent on the scale-axis. Imagine a second slicing surface placed *c* units below the primary slicing surface (see Figure 9). 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 $f_{blend} = d/c$, we make the appearance of the soon to be displayed terrain feature less abrupt.

Remember that we render the insides of polyhedra to get the intersection result. If we render the outsides instead, we encounter 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 polyhedrons. We can utilize this as follows: We perform both inside and outside renders and store them. Then at the last stage where we provide a colour for every ID we can blend the two colours together with the blending factor described above. Note that it is possible that outside rendering will look through the SSC (cf. Figure 9), but this simply means that the intersecting polyhedron is already the lowest one in the cube at that grid position and no colours will need to blend.

Note that near-intersection blending is applied only when the slicing surface is in motion, i.e. during the zooming interaction. The result is a smoother looking transition

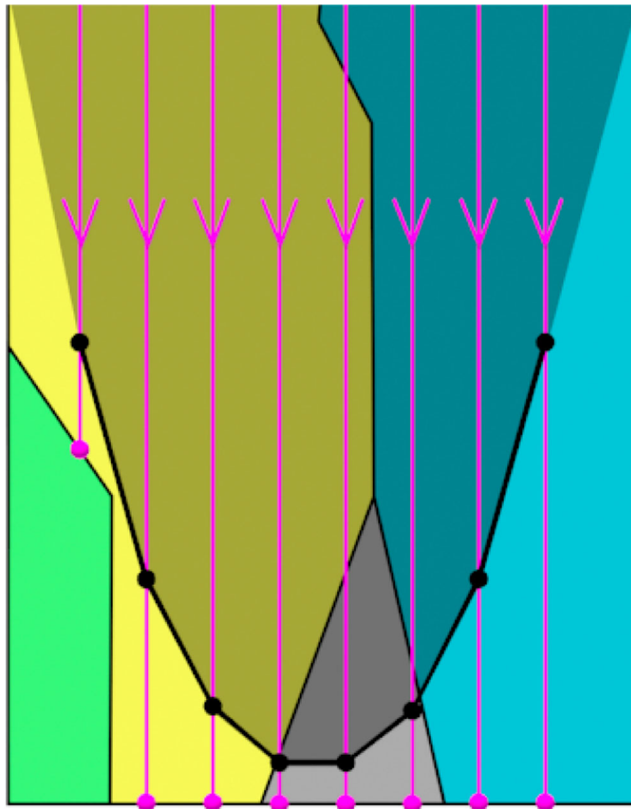

Figure 8. In this example, the centre of the raster requires less abstraction than the raster edges. As a result, the forest becomes invisible and the road visible.

between map objects while actively zooming, which is visually more comfortable to look at. Figure 10 shows an example of this.

3.2.3. Texture mapping

In the final stage to provide colours, we have used simple flat colours corresponding to the polyhedron IDs to produce the final image for display. When including near-intersection blending, at most two colours are blended. Instead of, or in addition to flat colours, we can easily add texture in this step (by using an additional small raster image containing the texture). A possible use of this is to give some types of terrain an extra visual hint to their properties (such as a tree texture to a forest, or waves in water). Figure 11 shows a simple number of texture example. Because the *z*-coordinate (map scale) is also known for every fragment, the texture can be used for specific map scales only, giving a user direct visual feedback on the map scale currently visible.

3.3. Chunking data

If we want to realize the visualization of an SSC over the web for nationwide datasets, we will need to have a way to request parts of the SSC dataset (as it would take too long to

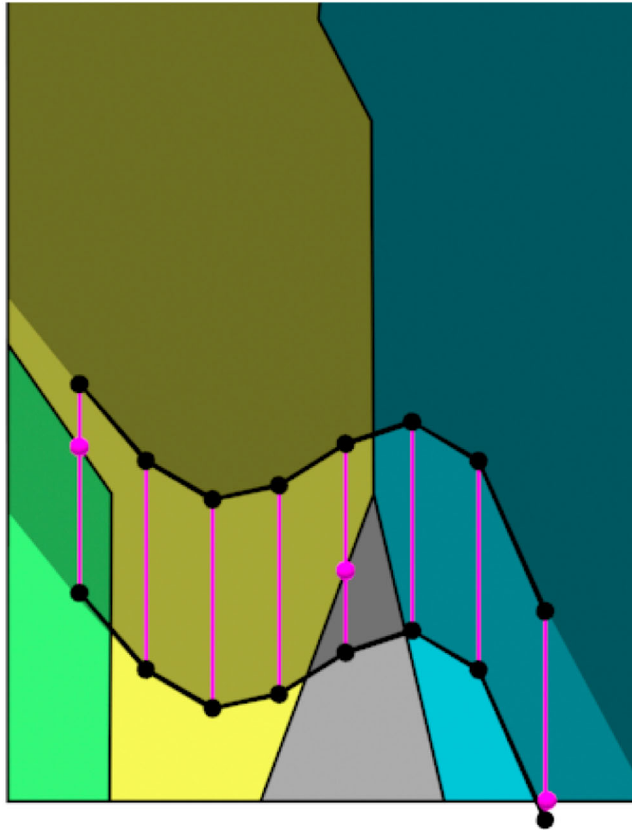

Figure 9. The three marked points represent an intersection within the given distance. The left and middle points will blend yellow-green and yellow-grey respectively. The right point resides on the cube boundary, so blending will not occur there.

transfer the complete dataset at first use). With the rendering approach explained, we will explain how access to large SSC data sets can be accomplished. Figure 12 shows an overview of the architecture:

(1) The vario-scale server, on which chunks of SSC data and an index structure are stored.
(2) The web client which has main memory, a CPU, a GPU, memory on the graphics card and possibly permanent storage memory (such as a flash memory card in a mobile telephone).

Huang, Meijers, Šuba, and van Oosterom (2016) introduced a protocol for retrieving tGAP data, based on the tGAP structure. The client sends a bounding box to the server and then the server determines which part of the tGAP data structure the client *exactly* needs to draw the map (note, the full tGAP structure is available at server side). With the SSC approach, the protocol is adjusted: chunks of data are prepared in advance (before use, with a pre-processing step) and these chunks together with an index structure are placed on the server. The index structure, which is retrieved in an initial step, makes it

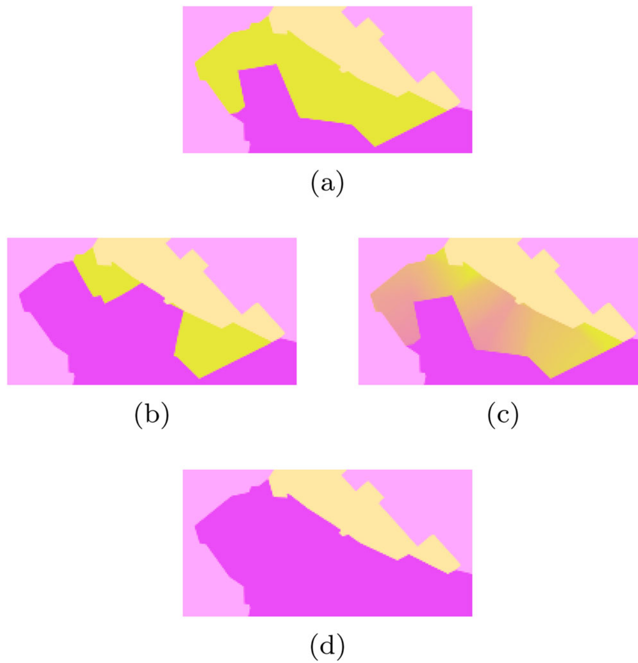

Figure 10. This example shows how a transition from one object (a) to the next (d) can be abrupt while zooming out (the purple object at the bottom takes over the region of the yellow object in the middle). This is noticeable, while the slicing surface is in motion, zooming in or out. Compared to (b) when near-blending is disabled (c) blending the colours makes the transition appear less abrupt.

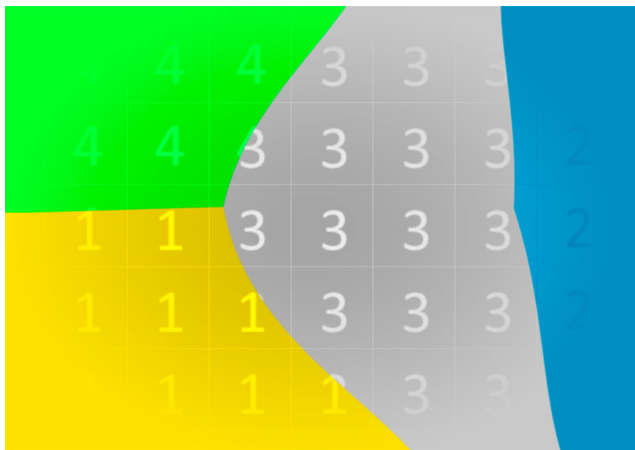

Figure 11. Texturing is shown as the numbers rendered in the regions. They fade on the sides because of the non-planar intersection used to generate the image, showing that abstraction variations influence the result.

possible that the client can retrieve individual chunks by requesting these from the server. Figure 13 shows the two protocols.

Once the client has retrieved the index structure, the index can be used by the client to determine which chunks are needed for making a map for the current viewport (while the

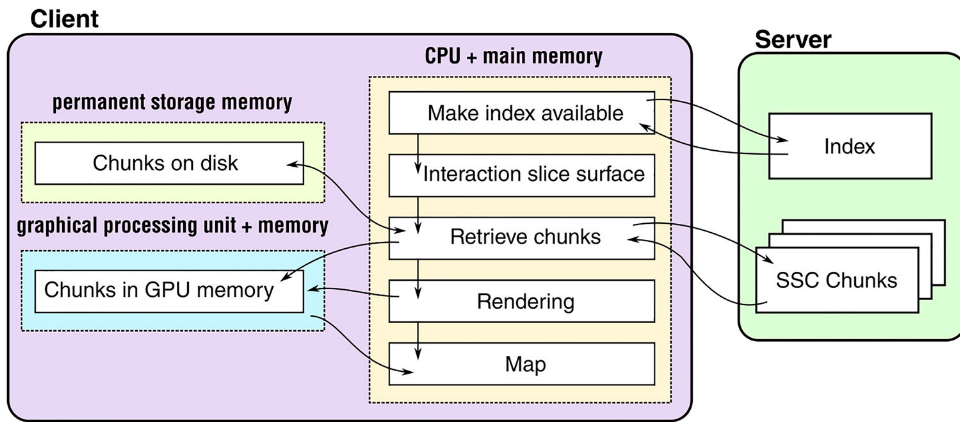

Figure 12. Architecture overview. How data flows through the architecture for each map image to be rendered is shown.

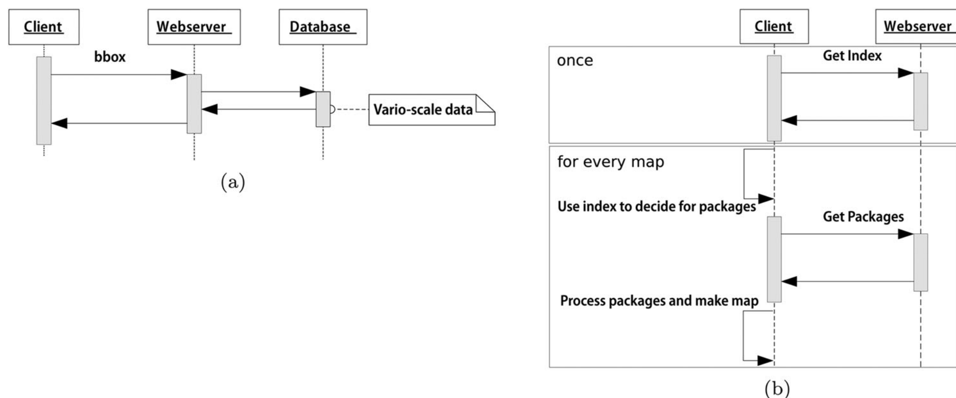

Figure 13. Protocol for retrieving (a) tGAP data compared to (b) SSC data.

user is zooming and panning). The task of the server becomes easier (more scalable with additional users), as it only has to serve the chunks, without additional work. Furthermore, (a part of) the chunks can be retrieved before using the data, such that later without network traffic the data can be also used. The relevant chunks (those intersecting the slicing surface) have to be sent to the GPU memory, such that the data can be used for rendering.

How should the chunks be made, which requirements do we have for them and how should the index structure look? In theory, there are multiple ways to make chunks of data:

(1) the 3D objects can be cut into multiple pieces, once they overlap with a piece of space (space division approach, e.g. by using an Octree);
(2) the 3D objects can be left as one piece, without having to introduce additional geometry (object division approach, for example by using an R-tree).

Furthermore, for the chunks we have some desiderata:

- Chunks should not introduce (or at least in a very limited way) additional geometry. Therefore, we prefer not to cut the geometry into multiple pieces, as this leads to additional maintenance load (heavier pre-processing and more work while interacting with the data on the client, e.g. combine all pieces to highlight one object).
- Chunks need to have approximately the same amount of object geometry (this would lead to chunks having the same data size, once the data is serialized for network transmission). Once a chunk is requested, it is clear how much network traffic this would cause. Evenly sized chunks are also useful for using the GPU memory without too much fragmentation (as the active chunks that reside in the GPU memory will change over time, when the user is zooming and panning).
- Chunks need to have a compact bounding box. For example, in case a chunk has one very long side (either for the space or for the scale dimension) the result is that this chunk subsequently is needed for many visualizations, but only limited amount of data from the chunk is possibly used for the visualization.

For the index structure it is also good when:

(1) Requests for chunks are possible in real time (so traversing the index structure should be possible to be executed in the order of a few milliseconds).
(2) It should be possible to request the index structure in parts (when chunks are available for a very large dataset, e.g. with the extent of the world, the index structure will be large itself).

All in all, we need to find a balance between all given (and sometimes contradictory) requirements. We elaborate more on this in Sections 4.2 and 5.4.

3.4. Smooth interaction techniques by applying animation

Once we have retrieved SSC data to the client side, we can have an in-depth look to the interaction possibilities. The way the end user can control the movement of the slicing surface through the SSC data cube, should emphasize a smooth and continuous impression. In a web browser, interaction can be realized by making use of the event loop that the JavaScript programming language offers. By means of actions that the end user performs events can be generated. Concretely this means that once the user instructs with a pointer device, e.g. by clicking the mouse or performing a click with the mouse wheel, this can move the position and shape of the slicing surface (i.e. setting up a different transformation matrix for the rendering) and other actions, such as data retrieval, can be initiated.

Although a user action (changing the slicing surface) can ensure that a rendering action takes place on the GPU, such rendering actions (producing a map image with the help of the GPU) do not have to be linked one to one to user actions. For example, we can instruct the GPU to perform every 1/60 of a second a rendering action during a certain period of time after the user has executed an action. This then leads to the system showing an animation, i.e. a rapid succession of several map images (as if a video is played). A description of the animation is given in the form of the current starting point of the map in world

coordinates and its scale (x_s, y_s, $scale_s$), the desired end point and its scale (x_e, y_e, $scale_e$), the start time and the desired duration of the animation.

For producing an animation it is important to keep track of the path that the mouse cursor travels during an action of the user. From this path the speed and direction can be derived when the mouse button is released. Hence, the map can continue to slide in the direction in which the user is already moving the map (i.e. the path to display during the animation is extrapolated using the path of the cursor). Furthermore, when producing the animations, a deceleration factor can also be used, so that the speed of the moving map gradually decreases and the map image slowly comes to a standstill. A point of attention when using animated zooming and panning is that the system should be able to abort running animations. For example, a new zoom-in action has to stop the currently running animation which was running as a result of a pan action, to schedule a new animation from the point that the user has broken into the original animation.

Note that a user action not only triggers an animation, but also leads to requesting chunks. In order to minimize interference of handling of the requests with the rendering with the GPU, so-called web workers can be used in Javascript. A web worker starts a second process within the internet browser, which can be made responsible for a separate task, e.g. for the data transfer. The advantage is that this makes the client more responsive (because receipt and processing of the received data can be handled in parallel with the rendering instructions).

4. Prototypes – experiences with SSC data on the web

We have implemented different prototypes, ranging from a stand-alone application to web clients, to experiment with the features of the described architecture. Each prototype demonstrates a part of the above sketched architecture: Using the GPU for rendering (Prototype I), packing and sending chunked data over the web (Prototype II-a and II-b) and allowing user interaction with animation to take place (Prototype III).

4.1. Prototype I: rendering; using the GPU allows real-time interaction

Driel (2015) has implemented a stand-alone desktop application (using the Java programming language), which did load SSC data completely at start-up time of the program, without network access. Figure 14 illustrates the datasets that were used. Advanced rendering features were implemented, making it possible to get a feel for the different options a vario-scale map browser could bring, while effectively using the GPU. Figure 15 demonstrates different types of slicing surfaces that were implemented. Several screen casts were recorded for illustrating the resulting effects.[1] Prototype I was run on a laptop computer with modest graphics hardware (MSI GE620DX). When rendering the maps at a resolution of 1280 × 720 frame rate remained at a satisfying 40–100 frames per second while the slicing surface was in motion.

4.2. Prototype II-a and II-b: chunking SSC data

We produced two web clients for experimenting with how to obtain chunks of SSC data and to investigate what effect chunking the original SSC into smaller parts has on the total size of the dataset.

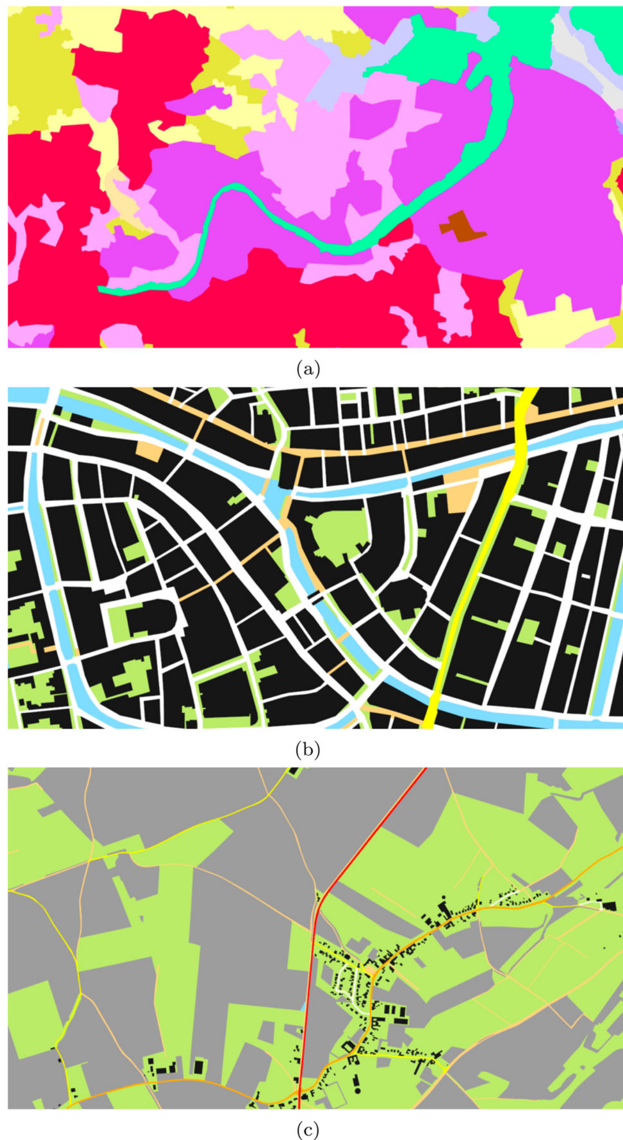

Figure 14. Testing data based on the SSC model, shown at the highest level of detail. (a) Land cover dataset (source: CORINE Land Cover dataset, Middlesbrough, UK). (b) City centre dataset (source: TOP10NL, Leiden city centre, NL). (c) Rural region dataset (source: TOP10NL, near Maastricht, NL).

4.2.1. Prototype II-a: Octree chunks are verbose!

Xu (2017) has produced a web client (using Javascript and WebGL) and preprocessor for making use of the Octree as an index structure. This prototype showed that the index structure also has its role to prevent transmission of chunks multiple times. By administrating in the index structure which status each block has (e.g. 'not yet available', 'being requested', 'retrieved', 'retrieved and made available on the GPU'), multiple transfers can be prevented. However, it was also shown that this Octree structure leads to an explosion of data: Tests with an area of 9 × 9 km show that 40 MB of storage space is needed for 1

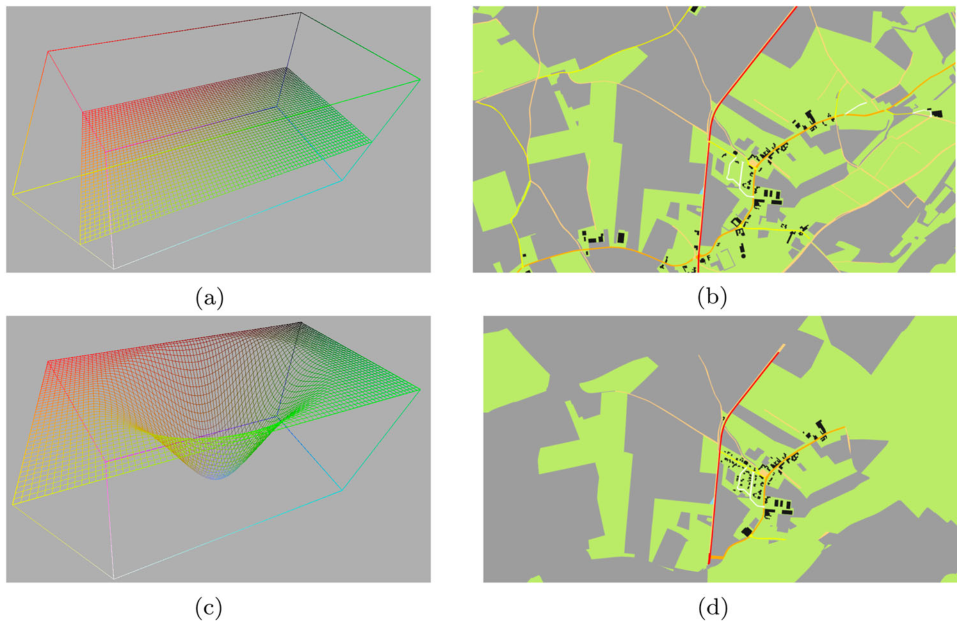

Figure 15. The slice surface on the left produces the corresponding map image shown on the right. Note that with higher level of detail more buildings and roads become visible. (a) Horizontal slice plane. (b) Map scale equal everywhere. (c) Sine curve slice plane. (d) Larger map scale in the middle.

chunk (not splitting up the SSC) and if the SSC is split up over 1135 chunks these chunks together have a total size of 239 MB: 6 times more data, only because the objects have to be split up over the Octree nodes! This problem originates from the fact that the chosen Octree data structure very much resembles a multi-layer approach as is currently common in current web map services (Web Map Tile Service, WMTS). As explained in Section 3.1, we render the insides of the polyhedra currently intersecting the slicing surface. To produce a complete map, a set of triangles at the bottom of each Octree leaf node is needed, otherwise you will see through the bottom of the chunks, without being able to get an image of the inside of the polyhedra. For this, the triangles which have to be added to the bottom of each Octree chunk lead to the explosion of data.

4.2.2. Prototype II-b: chunks with non-cut objects

Rovers (2016) elaborates on another attempt to create chunks of vario-scale data (trans-scale boundaries). Starting point is that the 3D objects must remain complete (geometry is not cut). Each object is completely contained in only one chunk and this prevents the redundancy of the Octree approach (the need of having to add extra triangles at the bottom of every chunk). A Space-Filling Curve (SFC) can be used to create groups of the objects. An SFC is a curve that traverses linearly through an *n*-dimensional (*n*D) grid (hyper cube) with a given resolution. Each cell in the grid is visited exactly once by the curve. An important property is that locality from the *n*D space is preserved in the location on the curve (Dai & Su, 2003). Each object is approximated by a point in the 3D Space-Scale Cube (*x*, *y*, *scale*) and then the location of this point is mapped to the chosen Space-Filling Curve. Objects that are nearby each other in the SSC will

also get a nearby location on the SFC and will subsequently be grouped in the same chunk. This work has shown that tuning this approach requires quite a lot of experimentation and an important question to deal with is how to treat the scale dimension in relation to the space dimensions (we need to express what is the relation of 1 m in the space dimension to the values in the scale dimension), as this determines how objects will be grouped.

4.3. Prototype III: enabling smooth interaction

The different user interactions followed by the client producing an animation were implemented in Prototype III.[2] This client has been implemented using Javascript and WebGL. The zoom action that has been implemented zooms the map image in (or out) around the location where the mouse is positioned. This allows the user to zoom in easily around a point on the map. It does not (yet) retrieve data in chunks, but all data is retrieved when the client program is loaded in the web browser, using one chunk.

The following parameters associated with the animated zoom and pan interactions can be adjusted by the end user in our implementation (Figure 16 illustrates the user interface controls below the displayed map image):

- Size of the step for zooming in or out (how big is the scale factor, which corresponds to a click of the mouse wheel, by default a factor of 2).
- The duration that an animation takes after the user has finished interacting with the map, either for zooming or for panning (e.g. 2 s). By setting the duration parameters to 0 s the system does not show the animation of the map image. This makes it possible to make comparisons between smooth/gradual interaction with added animation and interaction that feels more step-by-step.

5. Discussion and future work

Based on the described principles (Section 3) and experiences (Section 4) we will now reflect on the results covering the following aspects: Rendering lines, points and texts, the place of different parts in the architecture, an alternative way of making chunks and making smooth transitions better visible. Based on this reflection we will give some pointers for future work: carrying out usability tests, prediction of user movement, and dealing with 4D data (3D geometry, 1D Level of Detail).

5.1. Rendering linear and point features

The rendering approach as implemented in the prototypes assumes 3D volumetric features that are reduced to 2D polygons, but maps in practice include line and point objects. Roads will not appear on the resulting maps, although these are kept in the original tGAP structure by our generalization process (by applying the approach from Šuba, Meijers, & van Oosterom, 2016). By making an additional rendering pass for every frame that is produced, rendering lines should be possible. However, it is an open question whether lines need to be sent to

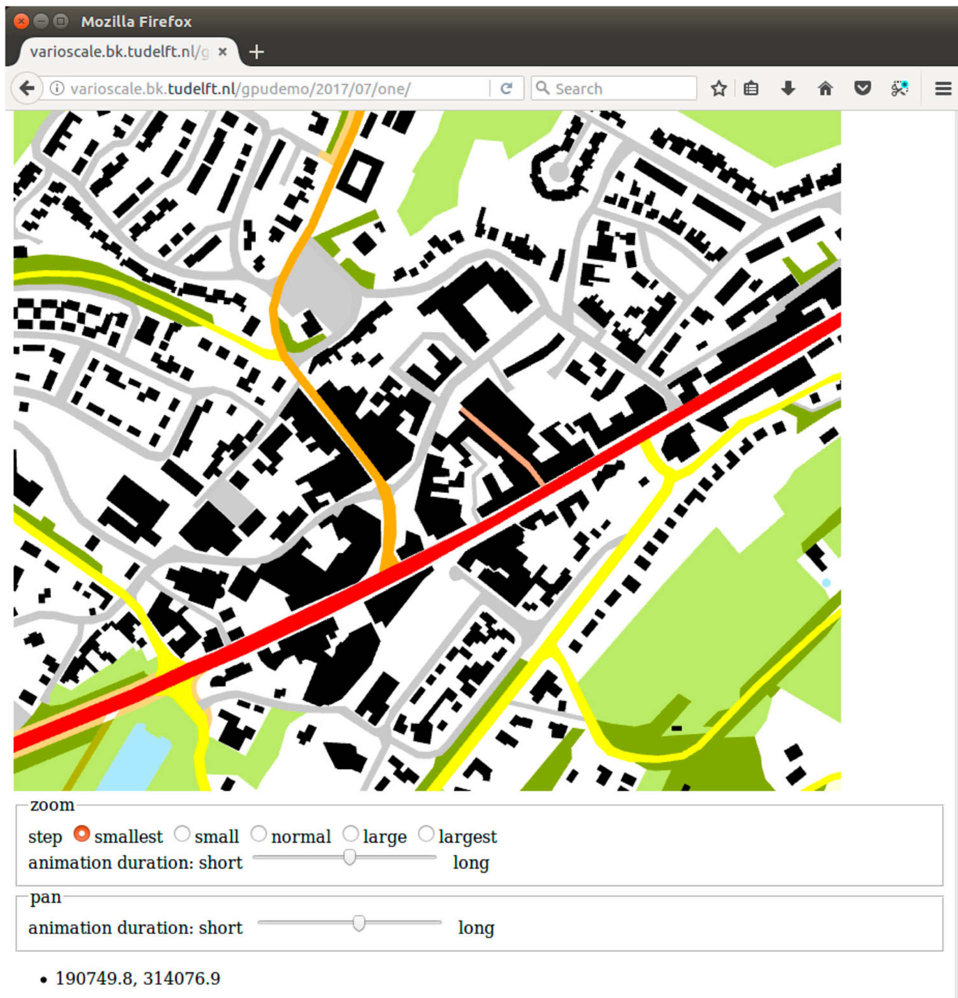

Figure 16. Parameters for controlling smooth transitions as implemented in the user interface.

the GPU next to the trans-scale boundary geometries in a different way or that the same set of triangles can be used for making the linear objects visible.

5.2. Rendering text/labels

Text labels or other types of objects in maps are also commonly placed along the direction of roads or centred in specific areas. Been, Daiches, and Yap (2006) discuss placing map labels for interactive maps. Placement of labels could be optimized in correspondence with the SSC polyhedrons (i.e. minimize overlap of labels with background polygonal objects). Flickering and jumping effects of the labels should be avoided (following the vario-scale definition small delta in scale should lead to a small delta in map content). Next to placing the labels, it is also an open question how to embed the location of the placed labels in the SSC or that a separate data structure is needed.

5.3. Which functionality needs to go where in the architecture

A number of the steps to arrive at maps from the 3D SSC data are now performed as a pre-processing step. Chunks with triangle geometry are finally prepared and offered on the server side. However, this is not the only option. We can also perform some parts in real time instead of as a pre-processing step. Or even on a different part of the architecture (server or client), i.e. we should reconsider which component does what when: Can the triangular data be derived in real time from blocks of data from the tGAP structure on the client? For example, this would be advantageous if tGAP data can be send more compactly than the triangular SSC data (more efficient transfer), and processing on the client side can be executed fast.

5.4. Alternative index structures/chunking approach

Prototype II-a showed that the Octree structure yields a huge increase in the data volume. However, the experiments of Protoype II-b with using a Space-Filling Curve for packing data in chunks also illustrated that getting good chunks is not easy. To obtain a good set of chunks an approach, that balances size and compactness of chunks and that is efficient to use at run time (not too much overhead per frame) is necessary. This is to be tested in a representative benchmark of chunk retrieval.

Furthermore other ways of obtaining chunks should be investigated. Next to using an Octree where polyhedra are stored non-split and not only in the leaf nodes of the Octree, another idea under study is to make compact chunks based on the following approach: The geometry of each object is represented by a 3D box and all geometries are sorted for each dimension. For each sorting step, we split the sorted object set into two subsets and for both sets we look at a number of heuristics for the resulting enveloping 3D box (we can apply various measures, such as a ratio between the box area versus circumscribed tight-fitting sphere area). The algorithm then chooses the best dimension with the most favourable overall measure for sorting and splitting. This is repeated on the two subsets separately. Sorting and splitting continues until a set of objects is sufficiently small (the amount of data is smaller than a pre-defined tolerance value). This guarantees that each chunk has a maximum size. Furthermore, if the resulting binary tree structure is too deep, then the number of levels in the binary tree can be limited by grouping several nodes together (by re-using the same algorithm, but now on the node geometry). The advantage of this is that the depth of the tree is limited, which can lead to faster tree traversal (more pruning takes place).

5.5. Making smooth transitions better visible

Prototype III showed that the outcome of the continuous generalization operations (the smooth merge operation from Figure 3) is only visible when putting the right parameters: Making the smallest steps for zooming and having long animation duration. Even when the continuous transitions are visible, these are difficult to notice. This is due to the fact that the slicing surface does intersect only once or twice per trans-scale boundary, but this should happen more frequently during a zoom

operation to be able to observe the gradual transition that is modelled by the trans-scale boundary. Hence, we should model the continuous transitions differently (e.g. give a longer scale span for each trans-scale boundary). That polyhedra their scale span is too short is caused by the step-wise generalization actions to obtain the tGAP structure. A possible solution is to perform generalization actions on multiple map objects in parallel, giving the whole group the same start and end map scale. The transition of the map objects in the group goes slower, instead of just one object after another having a transition.

5.6. Usability tests

We would like to customize Prototype III further, so that it also works on mobile devices, such as phone or tablet, with interaction using a touch screen and using gestures as input. After this has happened, we want to subject the vario-scale approach to further user research, using for example a questionnaire (to get a qualitative evaluation of the user experience) and modern techniques such as eye-tracking. Such an experiment could find user preferences, regarding settings such as what zoom steps and animation speeds users prefer. In order to test well, we want to make a larger test dataset available (for example, starting with the 1:10, 000 base map of The Netherlands). The tool for building a large tGAP is available, but converting it to an SSC divided into chunks still requires the necessary ‘engineering’.

5.7. Anticipation of user actions

A logical next step during interaction is to predict which chunks will be needed in the near future, as rendering and determining which chunks are needed for making a map are distinct in our approach. This can be a form of user interaction anticipation by requesting chunks ahead of a smooth pan or zoom movement by extrapolating recent/past movements (Weber, 2010). However, this could also lead to data being retrieved that turns out to not be needed, if the prediction is wrong. Animated jumping from one place to another on the map (for example by entering a place name) is a special case of this. Given that the path through space and scale can already be determined, because the start and end point are known at the beginning (e.g. when a user has just finished typing in the place name, something that is not the case with free movement of users), the chunks can be ordered in sequence when they are needed. In this case, data can already be transferred in advance (as the path where the user will look within x seconds is fully known beforehand). van Wijk and Nuij (2003) describe how such a path can be determined.

5.8. 4D data

Another option noted by van Oosterom and Meijers (2013) is that an additional axis could be added to the SSC producing a 4D hypercube which can be ‘intersected’ to produce a 3D intersection. Instead of starting with 2D maps (x, y), we now start with 3D models (x, y, z) and add scale (Level of Detail) as the fourth dimension. Some parts of the proposed rendering approach could work if we assume that the 4D data would

still need to be projected to a 2D screen, which is where the advantage of using fragment operations on existing GPU hardware is still present. This idea is nevertheless difficult to conceptualize, so we can not predict whether some parts might prove impractically difficult.

6. Conclusion

In this article, we have shown the first steps we have made with an online web platform for displaying vario-scale data with more gradual transitions, based on the SSC model. For the first time, we have a complete implementation of the SSC model available and we are able to assess the results of the continuous generalization operations of the vario-scale approach and show it animated and continuously across the web. This will make it possible to carry out usability research (how end users perceive continuous generalization). Moreover, we have identified several challenges remaining and we also have a 'wish list' for further implementation.

Notes

1. Screen casts:

   - Intersecting an SSC in real time – https://vimeo.com/193857079 – user zooming in.
   - Near-intersection blending – https://vimeo.com/193857089 – same zoom in action, but now with near-intersection blending activated.
   - Non-planar slicing surface – https://vimeo.com/193857103 – user drags the blue dot over the map, a slice surface is used as shown in Figure 15(c). The result is that around the blue dot the map scale is larger and map objects of larger scale are connected seamlessly to surrounding objects of smaller scale.

2. Prototype III is available at https://varioscale.bk.tudelft.nl/ ▹ 'WebGL demo'.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes on contributors

Martijn Meijers (1981) started his studies in Geodesy and Cartography at Utrecht University of Professional Education (Hogeschool van Utrecht), where he specialised in Geographic Information Systems. He continued his studies at Delft University of Technology, where he obtained a Master of Science degree in Geomatics in 2006. In 2011, he successfully defended his PhD thesis at this university on the topic of Variable-scale Geo-information. Currently, he is employed as Assistant Professor at Delft University of Technology, Department of GIS technology. Martijns research interests include map generalisation, geo-database management systems, cartography and geo-visualisation, (applied) computational geometry for GIS, handling large datasets and topological consistency.

Peter van Oosterom (1963) obtained an MSc in Technical Computer Science in 1985 from Delft University of Technology, The Netherlands. In 1990 he received a PhD from Leiden University on the topic 'Reactive Data Structures for GIS'. From 1985 until 1995 he worked at the TNO Physics and Electronics Laboratory in The Hague where he was one of the developers of GEO++, an open GIS based on the Postgres DBMS. From 1995 until 2000 he was senior information manager at the Dutch

Cadastre, where he was involved in the renewal of the Cadastral (Geographic) database. Since 2000, he is professor at the Delft University of Technology and head of the section 'GIS Technology'. His research topics include: Spatial databases (point clouds), GIS architectures, Map Generalization, Vario-Scale, 5D Modeling, Querying and presentation, Internet/interoperable GIS and (3D) Cadastral applications. He is the current chair of the FIG working group on '3D-Cadastres'.

Mattijs Driel (1987) completed an MSc at Utrecht University in 2015 working on the visualisation of the Vario-Scale data structure for real-time applications, under the supervision of Prof. dr. ir. P.J.M. (Peter) van Oosterom, dr.ir. B.M. (Martijn) Meijers, and Prof. dr. E Eisemann. Following the completion, he diverted away from research to focus more on improving his software engineering, architecture and development skills. In the latter half of 2015 he started at Double Dutch Games helping to refactor their game's architecture to better support multi platform deployment. In 2016 switching to work at Twnkls (now a branch of PTC) to help and lead construction of, amongst others, the Ikea Place app. Currently, he is still employed at PTC.

Radan Šuba (1986) finished his studies in 2012 at the University of West Bohemia, in Pilsen, the Czech Republic, and obtained his Master degree in Geomatics. During his studies, he gained practical experience when he worked as a land surveyor. In 2012 Radan was appointed as a PhD candidate at the Delft University of Technology, the Netherlands, under the supervision of Prof.dr.ir. P.J.M. (Peter) van Oosterom and dr.ir. B.M. (Martijn) Meijers funded by the Dutch Technology Foundation STW (projectnumber 11185). He researched vario-scale maps: Alternative solution for obtaining and maintaining geographical data sets on different map scales. He focused on R&D to expand and to enrich current prototypes for automated map generalisation.

References

Been, K., Daiches, E., & Yap, C. (2006). Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics*, *12*(5), 773–780.

Chimani, M., van Dijk, T. C., & Haunert, J.-H. (2014). How to eat a graph: Computing selection sequences for the continuous generalization of road networks. In *Proceedings of the 22nd ACM SIGSPATIAL international conference on advances in geographic information systems* (pp. 243–252). ACM.

Dai, H. K., & Su, H. C. (2003). On the locality properties of space-filling curves. In T. Ibaraki, N. Katoh, & H. Ono (Eds.), *Algorithms and computation. ISAAC 2003. Lecture notes in computer science* (Vol. 2906, pp. 385–394). Berlin: Springer.

Driel, M. (2015). *Real time intersections on space scale cube data* (Master's thesis). Utrecht University.

Dumont, M., Touya, G., & Duchêne, C. (2017). Alternative transitions between existing representations in multi-scale maps. In *Proceedings of 2017 international cartographic conference*.

Flynn, M. (1972). Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, *C-21*(9), 948–960.

Guha, S., Krishnan, S., Munagala, K., & Venkatasubramanian, S. (2003). Application of the two-sided depth test to CSG rendering. In *Proceedings of the 2003 symposium on interactive 3D graphics* (pp. 177–180). ACM.

Hable, J., & Rossignac, J. (2005). Blister: GPU-based rendering of boolean combinations of free-form triangulated shapes. In *ACM transactions on graphics (TOG)* (Vol. 24, pp. 1024–1031). New York, NY: ACM.

Hock-Chuan, C. (2019). Yet another insignificant… programming notes. 3D graphics with OpenGL. Retrieved from: http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

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

Meijers, M., Šuba, R., & van Oosterom, P. (2015). Parallel creation of vario-scale data structures for large datasets. In *ISPRS archives volume XL-4/W7, 4th ISPRS international workshop on web mapping and geoprocessing services*, Sardinia (pp. 1–9).

Meijers, M., & van Oosterom, P. (2011). The space-scale cube: An integrated model for 2D polygonal areas and scale. In E. Fendel, H. Ledoux, M. Rumor, & S. Zlatanova (Eds.), *ISPRS archives volume XXXVIII-4/C21, 28th urban data management symposium*, Delft (pp. 95–101).
Nöllenburg, M., Merrick, D., Wolff, A., & Benkert, M. (2008). Morphing polylines: A step towards continuous generalization. *Computers, Environment and Urban Systems*, *32*(4), 248–260.
Peng, D., Wolff, A., & Haunert, J.-H. (2017). Using the A★ algorithm to find optimal sequences for area aggregation. In M. P. Peterson (Ed.), *Advances in cartography and GIScience* (pp. 389–404). Cham: Springer.
Rovers, A. (2016). *Exploring the use of a generic spatial access method for caching and efficient retrieval of vario-scale data in a client-server architecture* (Master's thesis). Delft University of Technology.
Sester, M., & Brenner, C. (2009). A vocabulary for a multiscale process description for fast transmission and continuous visualization of spatial data. *Computers & Geosciences*, *35*(11), 2177–2184.
Šuba, R., Meijers, M., Huang, L., & van Oosterom, P. (2014). An area merge operation for smooth zooming. In J. Huerta, S. Schade, & C. Granell (Eds.), *Connecting a digital Europe through location and place* (pp. 275–293). Cham: Springer.
Šuba, R., Meijers, M., & van Oosterom, P. (2016). Continuous road network generalization throughout all scales. *ISPRS International Journal of Geo-Information*, *5*(8), 21.
van Oosterom, P. (2005). Variable-scale topological data structures suitable for progressive data transfer: The GAP-face tree and GAP-edge forest. *Cartography and Geographic Information Science*, *32*(4), 331–346.
van Oosterom, P., & Meijers, M. (2013). Vario-scale data structures supporting smooth zoom and progressive transfer of 2D and 3D data. *International Journal of Geographical Information Science*, *28*(3), 455–478.
van Oosterom, P., Meijers, M., Stoter, J., & Šuba, R. (2014). Data structures for continuous generalisation: tGAP and SSC. In *Abstracting geographic information in a data rich world* (pp. 83–117). Cham: Springer.
van Wijk, J. J., & Nuij, W. A. A. (2003). Smooth and efficient zooming and panning. In *Proceedings of the ninth annual IEEE conference on information visualization*, INFOVIS'03 (pp. 15–22). Washington, DC: IEEE Computer Society.
Weber, B. T. (2010). *Mobile map browsers: Anticipated user interaction for data pre-fetching* (Master's thesis). The University of Maine.
Xu, Y. (2017). *Construction of a responsive web service for smooth rendering of large SSC dataset and the corresponding preprocessor for source data* (Master's thesis). Delft University of Technology.