UNIVERSITY OF LEEDS

This is a repository copy of *Large-Scale Pixel-Precise Deferred Vector Maps*.

White Rose Research Online URL for this paper:
http://eprints.whiterose.ac.uk/169264/

Version: Accepted Version

Article:
Thöny, M, Billeter, M orcid.org/0000-0003-1806-2587 and Pajarola, R (2018) Large-Scale Pixel-Precise Deferred Vector Maps. Computer Graphics Forum, 37 (1). pp. 338-349. ISSN 0167-7055

https://doi.org/10.1111/cgf.13294

© 2017 The Authors Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd. This is the peer reviewed version of the following article: Thöny, M., Billeter, M. and Pajarola, R. (2018), Large-Scale Pixel-Precise Deferred Vector Maps. Computer Graphics Forum, 37: 338-349. , which has been published in final form at https://doi.org/10.1111/cgf.13294. This article may be used for non-commercial purposes in accordance with Wiley Terms and Conditions for Use of Self-Archived Versions

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Volume 0 (1981), Number 0 pp. 1–11 COMPUTER GRAPHICS *forum*

Large-Scale Pixel-Precise Deferred Vector Maps

Matthias Thöny[†1] Markus Billeter[1,2] Renato Pajarola[1]

[1]Departement of Informatics, University of Zürich
[2]Chalmers University of Technology

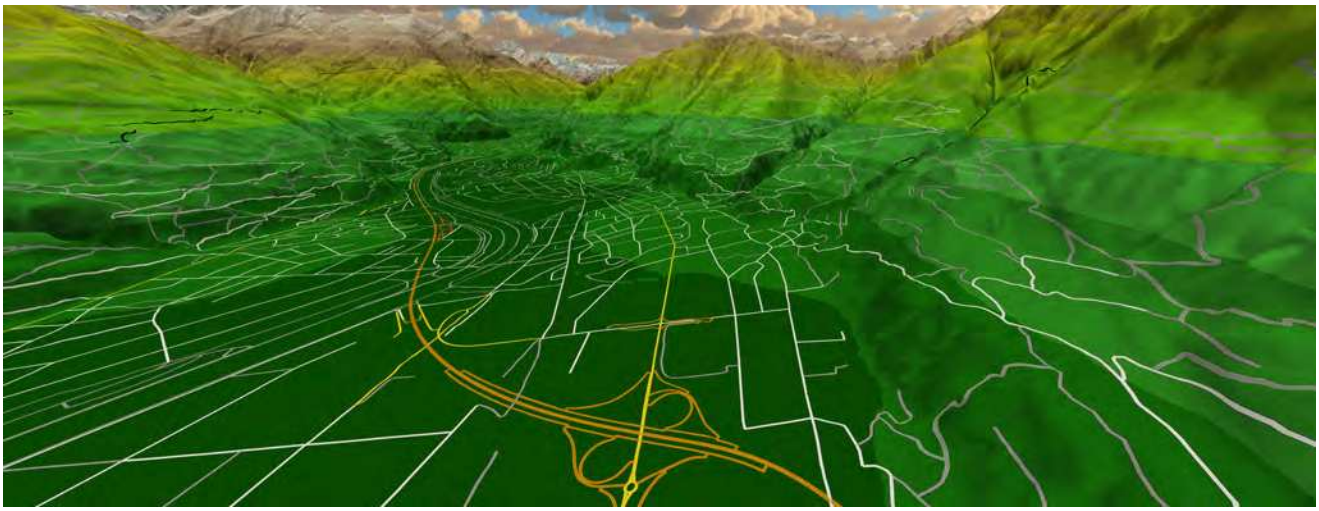


Figure 1: *Example of a vector map data set showing a part of a street network consisting of 16 million line segments.*

Abstract

Rendering vector maps is a key challenge for high-quality geographic visualization systems. In this paper we present a novel approach to visualize vector maps over detailed terrain models in a pixel-precise way. Our method proposes a deferred line rendering technique to display vector maps directly in a screen-space shading stage over the 3D terrain visualization. Due to the absence of traditional geometric polygonal rendering, our algorithm is able to outperform conventional vector map rendering algorithms for geographic information systems, and supports advanced line antialiasing as well as slope distortion correction. Furthermore, our deferred line rendering enables interactively customizable advanced vector styling methods as well as a tool for interactive pixel-based editing operations.

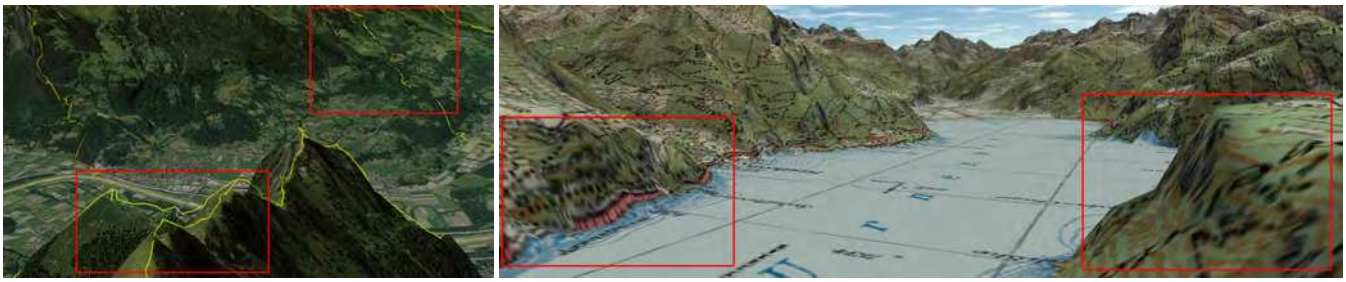
Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Displaying vector maps is an important component in geographical visualization applications such as virtual globe software, mapping and navigation systems, as well as other interactive geo-spatial 3D environments. Vector maps are used to represent geographic features such as streets, rivers, contour lines or land use information. An example of such data can be seen in Fig. 1. Displaying and visualizing these data sets interactively in real-time 3D applications, such as Google Earth, Caesium Virtual Globe and Map Engine, or NASA Worldwind is a challenging task. Improving the performance and accuracy of interactive visualization of large-scale vector maps is thus an important goal. In this paper we address the problem of rendering large-scale vector maps with many millions of line segments within a 3D real-time geo-visualization application. This massive amount of vector map data is challenging be-

† e-mail:{mthoeny|billeter|pajarola@ifi.uzh.ch}

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.



(a) (b)

Figure 2: *Example artifacts when rendering vector maps. (a) Vector lines floating or intersecting the underlying 3D terrain, and (b) texture mapping based projection and resolution artifacts.*

cause of limited memory capacities and because of the rendering cost per frame which must be minimized for real-time purposes.

In geographic visualization systems, previous vector map rendering methods may cause visual artifacts that negatively affect the interactive data exploration quality and corresponding geo-spatial analysis tasks. Examples of such artifacts are shown in Fig. 2. In particular, when combining multiple 3D vector maps and a continuous multiresolution level-of-detail (LOD) terrain mesh the mismatching resolutions may cause significant artifacts in the visualization. The problems shown in Fig. 2(a) occur because line segments of vector maps (in yellow) with differing resolution float above or intersect the terrain. This unpredictable scene configuration makes the problem very complicated for geometric line rendering methods. Texture based approaches do not suffer from this intersection problem, but from other artifacts such as aliasing and projective distortions as shown in Fig. 2(b).

Compared to other vector map rendering methods, our approach performs a deferred shading pass for displaying the vector map data on top of the traditionally rendered 3D terrain surface such as to avoid dependence on modified (preprocessed) vector map geometry. Consequently this avoids the coupling between different vector maps or between vector maps and the terrain height-field during a preprocessing stage, and all data set combinations are changeable at runtime. Furthermore, on the one hand, we can avoid artifacts as shown in Fig. 2, and on the other hand we also achieve pixel-precise line display results even with multiple layers of vector maps with different resolutions being visualized on top of the 3D terrain. Moreover, modern map visualization systems should allow the user to interactively change their map visualization with advanced vector styling methods. This requires a flexible line rendering method capable of extending the geometric line rendering to include line styling capabilities.

In this paper, we present an extended description of our approach [TBP16], including newly added functionality on coverage-based line antialiasing as well as slope distortion-corrected line rendering. In particular, we describe our deferred rendering approach which exploits a clustered line buffer structure to interactively visualize large-scale vector maps with many millions of line features in a pixel-precise manner, and we report new GPU-optimized rendering performance results.

2. Related Work

In the following section we review state-of-the-art techniques for vector map rendering, as well as relevant work related to clustered deferred shading. The terrain rendering in this work is based on RASTeR [BGP09, GMBP10] but could be replaced by other state-of-the-art systems such as [LH04, DKW09, LKES09, RRPCC12, KJCH15].

2.1. Vector Map Visualization

Vector maps as shown in Figs. 1 and 2 are usually line or polygon based data describing geometric objects with specific attributes. This geometric information can be used directly for a 3D visualization as shown in [BN08]. However, the most popular method is the combination of vector maps with image based information, like aerial photographic data, projected onto a 3D terrain surface model. In this context, the closest related prior approaches for vector map visualizations can be divided into three categories: (1) texture based, (2) applying geometric subdivision and (3) using shadow volumes. These methods are described in more detail in the survey of interactive visualization of vector data [KD02] and the survey of a digital earth [MAAS15]. A possible system description for these methods can be found in [CR11]. In Tab. 1 we summarize the main advantages and limitations of these three approaches in comparison to our new deferred vector map visualization method.

The most common vector map rendering method used in geo-visualization systems is the texture based approach. Different descriptions of this method as well as comparisons to other methods can be found in [KD02, WKW*03, SLL08, WLB09]. The basic idea is that vector maps are orthographically projected and rasterized to images and used as textures mapped on the terrain, e.g. as in Fig. 2(b). In general, any rendering system can easily apply textures to (terrain) surfaces, therefore, it is convenient and simple to implement such a texture based vector map visualization. In addition, texture based methods are often used if the development targets have limited hardware capabilities such as embedded devices, mobile platforms or browser based applications.

Texture based vector maps, however, may suffer from artifacts as shown in Fig. 2(b). The highlighted artifacts are caused by the 2D texture projection as well as limited texture resolution. If the texture

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

Criteria	Geometric Approach	Texture Mapping Approach	Shadow Volumes	Deferred Vector Maps
Rendering artifacts (Fig. 2)	Intersections, z-Buffer artifacts	texture aliasing, distortions	geometric aliasing	geometric aliasing
Output accuracy	tesselation resolution	texture resolution	pixel-accurate	pixel-accurate
Dynamic changes	recompute tesselation	redraw textures	yes*	yes
Interactive styling & editing	recompute tesselation	evaluate texture content	missing geometry at shading	update line buffer
Memory consumption	geometry only	hi-res textures	geometry only	geometry only
Additional geometry needed	recompute tesselation	none	geometry extrusion	only line information
Preprocessing requirements	complete terrain necessary	texture hierachy	no preprocessing	line buffer generation
GPU requirements	none	none	geometry shader*	random memory reads
Applicable to large data	intensive preprocessing	expensive redraw/preprocessed	pixel overdraw too expensive	demonstrated with $> 10^6$ lines

Table 1: *The table summarizes the differences between previous methods and our new technique (last column). *GPU requirements may be traded for a preprocessing step, making dynamic changes more costly.*

resolution is increased more memory is needed. Many systems thus work with preprocessed texture pyramids, but adding higher resolutions increases on one hand the memory consumption and on the other hand also the amount of texture files in these systems exponentially. Additionally, the texture pyramid may introduce border artifacts of the vector map information during rendering when used in a multiresolution LOD system. Moreover, most of the systems using this type of visualization do not allow immediate modification and styling of vector maps within an interactive 3D display session.

To achieve a precise and suitable visualization for interactive vector map modification, it is necessary to manage a dynamic texture pyramid and to implement an on-the-fly rasterization step for updating vector maps. The amount of re-rasterization grows with the complexity of modification possibilities such as selection of vector map layers or highlighting of selected elements. Artifacts often appear when moving the view frustum close to the surface and the camera points to a far distance. In these cases, the texture based approach can get overly complex and costly in terms of memory, rendering time and system flexibility.

In geometric subdivision approaches for vector maps [KD02, SGK05, XSWJ10, DXZS13] lines are subdivided according to the terrain mesh structure as illustrated in Fig. 3. Along the line segment at every change in slope of the underlying terrain, corresponding to crossing triangle edges, the line segment is subdivided. An application of the geometric line subdivision in combination with advanced map styling features can be found in [VTW11] and [WSL12]. The methods show possible ways to do precise line renderings. However, this geometric approach for vector maps has the drawback that its line subdivision requires a predetermined and fixed combination of terrain triangulation and vector map subdivision. Dynamically changing terrain information, as is the case in continuous LOD terrain rendering, as well as unforeseeable combinations or editing of vector map data sets are hard to manage in this approach, and therefore, interaction possibilities are limited. This problem becomes even harder when the terrain information contains multiple layers. This is the case when height maps of different resolutions are combined and transitions between them are generated dynamically.

It cannot be assumed that a single point has a unique fixed height value, and often terrain blending is done in the shader stage such that the final mesh cannot be retrieved. In such cases graphical artifacts would appear. Another issue is the precise overlay of planar geometric objects, which often leads to z-buffer artifacts (*z-fighting*) due to the limited precision of the depth buffer.

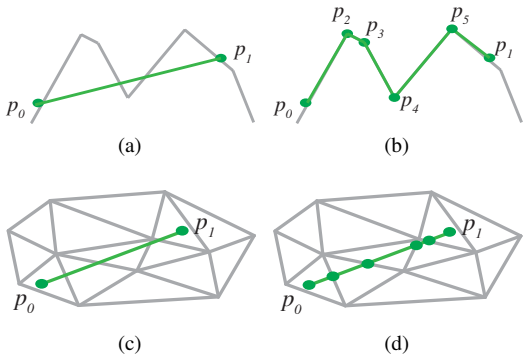


Figure 3: *Example for subdivision of a line according to the terrain height field. (a) Shows the elevation profile and (b) the subdivision. (c) Shows the line on top of the terrain and (d) the subdivision of the line matching the terrain mesh.*

The shadow volume approach for vector map visualizations produces high quality solutions because the algorithm works independent from the terrain resolution and always produces a pixel-precise result on the screen [DZY08, WLB09, YZM*11]. The idea is that the geometry of a vector map is orthographically projected on the terrain by extruding the vector map's line segments into 3D polyhedral objects. The vertically extruded polyhedrons are then rendered in two steps, first front faces then back faces. Analog to shadow volumes, every screen pixel counts the difference between front and back faces. The result per pixel then contains the information if this pixel is a part of a projected vector map or not.

The main drawback of shadow volume approach is that multiple geometry rendering passes are required for the vector map, in addition to the terrain, and that the vector map geometry has to be extruded, e.g. in a geometry shader. On one hand there is about four times more geometry than in the original vector map, and on the other hand every extruded line segment has to be rendered twice. Furthermore, in case of large vector maps the vertically extruded geometry covers large portions of the screen and may produce a

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

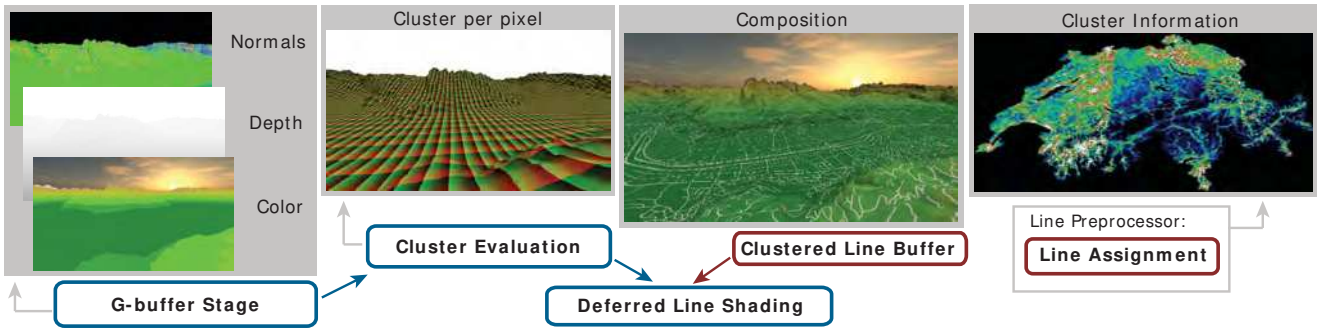


Figure 4: *Our deferred line rendering pipeline for large-scale vector map visualization. After the generation of the G-buffer, the cluster evaluation is performed and used as input in combination with the clustered line buffer for the deferred vector map line shading. The clustered line buffer is prepared in the line assignment preprocess.*

massive overdraw. Overall, this severely limits the technique to rendering very moderately sized vector maps of maybe a few thousand line segments. A special case is shown in [OC11] where the method is optimized for vector maps only containing lines. Furthermore, it is hard to preserve the original vector map information so that advanced vector styling or procedural texturing can be achieved.

2.2. Deferred Shading

Our deferred line rendering approach is inspired by the way lighting calculations are done in deferred shading pipelines [ST90, LD12]. Deferred shading is applied to reduce the amount of shading operations by introducing a two-pass rendering pipeline. The first pass renders the geometry and produces a set of textures containing geometric scene information such as color, normal, depth or light information. The collection of output textures of the first pass is called a G-buffer. All shading operations are done as image based effects using the information from the G-buffer in as few shading passes as possible. These render passes implement the effective shading and lighting for every pixel as well as other screen-space post processing operations such as e.g. antialiasing [CML11] or ambient occlusion [BS08].

Building up a deferred shading pipeline raises on one hand the GPU memory consumption for additional texture layers, and on the other hand the pixel fill rate because many more images are produced than effectively used as final output frames. Eventually the overall rendering effort per frame can nevertheless be reduced drastically and allows for more image-space effects within one single frame. Clustered Deferred Shading described in [OBA12] subdivides the view frustum into clusters to improve the rendering speed for scenes with many lights. In our concept we took over the idea of clustering scene objects by creating line clusters in world-space as a preprocess.

3. Deferred Vector Map Rendering

All approaches discussed in Sec. 2.1 are using some kind of extracted geometry or geometry rasterized on textures for vector map visualizations. However, modern programmable GPUs allow us to develop much more flexible systems. The basic idea of our line projection and rendering approach is to directly project and display vector maps on top of the terrain surface using an adaptation of the deferred shading principle without the need to generate intermediate geometric objects or rasterize vector lines into textures. In contrast to common rendering methods where the color of a pixel is derived from the main (geometry) rendering pass, our approach inverts this principle for vector map visualization.

In Fig. 4 we outline the main steps of our deferred line rendering method as further detailed below. In a deferred shading stage, for every pixel corresponding to a point on the terrain it is determined if it contributes to the visualization of a vector map feature. Using the G-buffer data obtained from the main terrain rendering pass, we back-project each pixel into the 3D world and determine its location within the vector map, see also Fig. 6. To identify candidate line features, we use a spatial data structure storing all vector map line elements, which we call a *clustered line buffer*. An optimized search within the clustered line buffer allows us to find the closest line features quickly and shade pixels accordingly.

3.1. Clustered Line Buffer

In our approach, during the deferred shading stage, for each pixel the closest intersecting line feature, if any, must be determined very quickly. For this we use our *clustered line buffer* which *clusters* the individual line segments of the vector map features into a large 2D structure of cells. Every line segment is assigned to each cell it intersects. This can be a regular grid as currently implemented, but it could also be a multi-level nested grid or other space-partitioning hierarchy to better adapt to variations in the feature density in very large vector maps. Important is the ability to perform point-to-cell look-ups very efficiently, to allow fast identification of the cluster containing the lines potentially intersecting a pixel. Moreover, within each cluster the line segments are organized in an effective spatial search index structure to accelerate line search and pruning as well as minimize distance calculations after the coarse cluster identification.

The clustered line buffer is thus a data structure enabling fast point-to-line search queries and is designed to be used efficiently on the GPU during the deferred shading pass, as illustrated in Fig. 5(a).

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

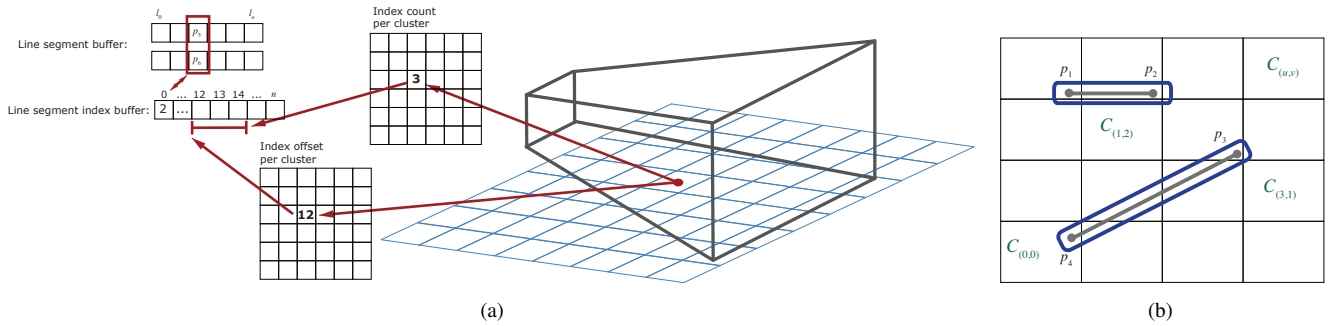


Figure 5: *(a) The* clustered line buffer *GPU data structures consist of multiple array buffers and textures supporting efficient point-to-cluster location and point-to-line search queries from a given geographical location. (b) Assignment of feature lines to clusters by rasterization.*

The clusters of this line buffer are formed during a preprocessing pass which assigns all vector map line segments to their corresponding clusters as further described below.

Two *line segment buffer* arrays store the start- and end-point coordinates $\{p_s, p_e\}$ of the individual line segments l_i on the GPU. A single *line segment index buffer* stores the indices to line segments concatenated for all cluster. The cluster grid is represented by two 2D integer textures, $o_{u,v}$ representing the cluster's *index offset* into the line segment index buffer and $c_{u,v}$ for the *index count* denoting the number of lines in the cluster. In a line assignment preprocess, for each cluster $C_{u,v}$ the vector map line features intersecting it are recorded, counted, and then concatenated to form the line segment index buffer.

The texture sizes for $o_{u,v}$ and $c_{u,v}$ equal the size of the cluster grid $C_{u,v}$. On one hand the grid should not be too coarse, because that would include too many line segments within each cluster. On the other hand the amount of clusters is limited to the texture memory that can feasibly be afforded. In our implementation we typically divide the map space into 256×256 clusters to express the cluster indices u, v as one byte each. Other multi-level nested grids or space partitioning structures could be used as well, given a memory efficient implementation and fast cluster identification on the GPU.

The line assignment to a specific cluster follows a Bresenham-like line rasterization as illustrated in Fig. 5(b), with certain line segments being assigned to multiple clusters. As the actual line drawing style is not known beforehand, we currently assume a predetermined conservative maximal line width which is incorporated into the line assignment. As indicated in Fig. 5(b), the line segment $l_1 = \{p_1, p_2\}$ lies in at least two clusters, but since the line has a certain line width it must be assigned to all clusters it overlaps, e.g. $C_{1,2}$. Similarly, for $l_2 = \{p_3, p_4\}$ all overlapping clusters along the line are included, e.g. also including $C_{3,1}$.

To optimize the point-to-line query after the coarse point-to-cluster location, we use a hierarchical spatial index structure to organize all line segments within one cluster $C_{u,v}$. For simplicity combined with efficiency we generate a fully balanced binary bounding volume hierarchy (BVH) over the lines within each cluster which are sorted along a space filling curve according to their midpoint. Other optimized BVHs could further improve upon this solution.

3.2. Deferred Line Shading

The final fragment color for a screen pixel is determined in the *deferred line shading* stage, see also Fig. 4. In this stage our approach decides whether a pixel contributes to the visualization of a vector map element or not. Using the clustered line buffer, where all line segments are grouped into clusters, the search space of all line segments influencing a single pixel can be restricted to the line segments belonging to a certain cluster into which the pixel projects. Hence the line identification consists of two main steps: in the first step it is necessary to determine the cluster in the clustered line buffer affecting a certain pixel. The second step determines if and which line effectively intersects the given pixel considering the applied line-style width.

The cluster of a certain pixel $s_{(x,y)}$ is determined by a backwards projection of the pixel position from screen space to world space coordinates. For every screen position $s_{(x,y)}$ a back projection can be applied using the corresponding depth $d_{(x,y)}$ from the G-buffer information, illustrated in Fig. 6 by the red line. The back projection of $s_{(x,y,d_{(x,y)})}$ then results in the point I_s which is the pixel's location in world coordinates. The back projection can be expressed as multiplication with the inverse view-projection matrix M_{VP} as $I_s = M_{\text{VP}}^{-1} \cdot s_{(x,y,d_{(x,y)})}$.

With the information about the clustered line buffer's subdivision of the vector map and the point I_s in world space, it is easy to calculate the cluster $C_{u,v}$ containing the point I_s. In other words, this step maps the screen-space pixel locations (x, y) to the cluster-index space (u, v). In Fig. 4 we visualize this cluster evaluation by coloring each pixel in (red,green) based on its relative position within the corresponding cluster.

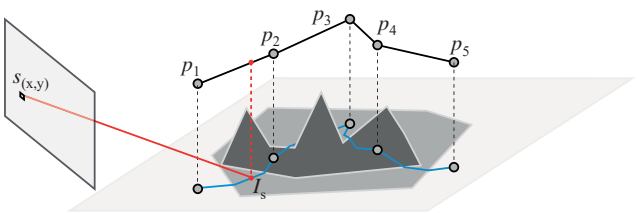


Figure 6: *Pixel back-projection and vector map location.*

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

Using the back-projected 3D point location I_s of a pixel $s_{(x,y)}$ with depth $d_{(x,y)}$ from the G-buffer we thus have determined the cluster $C_{u,v}$ containing any potential line candidates. We now have to determine if the point I_s lies within a certain distance of any line segment $l_i \in C_{u,v}$ in the 2D vector map plane. As illustrated in Fig. 6, point I_s lies inside a certain distance to the line segment p_1, p_2 of the vector map. Thus it is considered to be part of that line segment and the pixel $s_{(x,y)}$ can be colored accordingly using the current style and visualization parameters.

For large and complex vector maps as shown in Figs. 1 and 13, however, per-pixel line identification and point-to-line distance tests can become costly in our deferred line shading approach and some further optimizations are called for as described below.

The point-to-line search and distance calculation within a cluster $C_{u,v}$ should be optimized to keep the per-pixel computation cost low. Given the varying number of line segments in different clusters, for large vector maps we organize the line segments within each cluster in a BVH as already mentioned in Sec. 3.1 to limit the number of distance test computations.

Given a pixel's position I_s in world coordinates and in vector-map space, the BVH can be traversed effectively to identify the leaf nodes containing any line segments $l'_i \subseteq C_{u,v}$ which are potentially closer than a certain given distance from I_s. Only for this subset of lines l'_i the point-to-line distance has eventually to be computed. Therefore, even for very large vector maps with many millions of lines, the amount of line distance tests required per cluster is eventually reduced to a small number and can thus be performed in a fragment shader for each pixel efficiently.

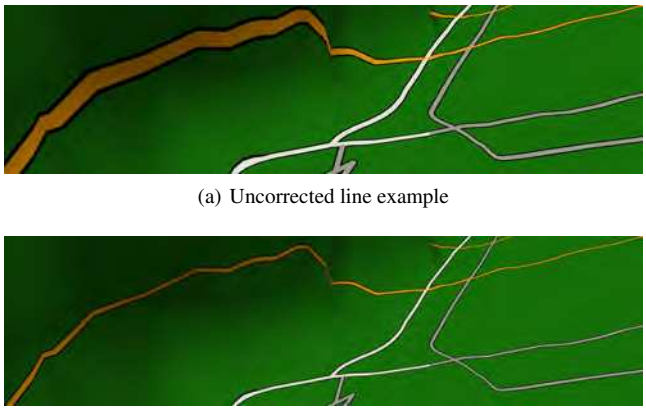
3.3. Projective Line Adjustment

The orthographic projection of line segments to the terrain stretches the line segments and does not take the terrain slope into account. This apparent artifact appears especially for high elevation difference. An example is shown in Fig. 7. The problem can be solved by adjusting the line distance considering the terrain slope value and the view information according to the following formula:

$$d' = \frac{(n \cdot E_y)}{(1 - v) \cdot E_y} \cdot d$$

To adjust a line segment to the terrain slope we have to adjust it's line width and therefore the distance for which a pixel is considered to be on the line. The illustration in Fig. 8 shows the correlation of projection, terrain slope and the camera view.

As an adjustment factor we use the dot product between the normal n and up vector E_y of our local coordinate system E. A value close to 0.0 describes a steep slope, whereas values close to 1.0 describe almost horizontal areas. Therefore, lines lying on extreme slope values are shaded narrower than before. From a top view, those lines appear now tighter than others. Thus, it is necessary to consider the view angle v as well as using the dot product between the inverse vector of the camera direction v and the coordinate system's up vector E_y. A horizontal view point will now rely on a strong gradient adjustment while a view point from top will limit the line adjustment. Results for the projective line adjustment can be seen in Fig. 7.



(a) Uncorrected line example

(b) Slope distortion-corrected lines

Figure 7: *Slope distortion-corrected line rendering results. (a) Uncorrected line with a street pattern partly on steep terrain. (b) Adjusted lines with respect to terrain normal. Values in plain areas are adjusted less.*

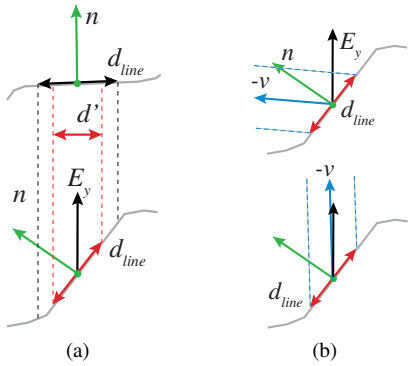


Figure 8: *Illustration to projective line adjustment. (a) Higher slope vectors narrow the line distance. So that d_{line} has to be correct to d'. (b) Due to the normal correction the camera view angle has to be adjusted as well as to prevent different line sizes for the same view angle.*

3.4. Line Styles and Interaction

The deferred line rendering method outlined above can further support visualization features beyond bare line rendering. In particular, the screen-space per-pixel shading allows the implementation of advanced colorization decisions like applying different line styles and line patterns on-the-fly during interactive rendering. Cross-sectional color patterns can easily be incorporated into a generic line shader taking the width of the line feature and the distance of the pixel to the line into account (see also Fig. 10 and results in Sec. 4). Longitudinal procedural patterns can be applied using pattern buffers and more complex line shaders as well. Furthermore, given the pixel-to-line distance to the closest or all pixel-intersecting line features, blended compositing of multiple features or over background can be achieved. Dynamically varying or view-dependent visualization parameters can also be incorporated into

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

the deferred line shading process, as demonstrated e.g. in Fig. 12 with an interactive lens function.

3.5. Coverage-based Anti-aliasing

Proper anti-aliasing is a key concern when rendering lines. In particular, the presented system needs to deal with lines at all scales. A line close-up may end up covering many pixels, while a view from a far distance needs to deal with lines much thinner than the footprint of a pixel (or of a pixel's subsample).

We draw inspiration from Persson's method for rendering distant phone wires [Per12]. In summary, when finding lines that may affect a pixel (i.e., during traversal), the lines' widths are adjusted to be at least one pixel wide. During shading, we find approximatively the line's coverage of the pixel to be shaded. The coverage is used to blend the line with the sample's background color.

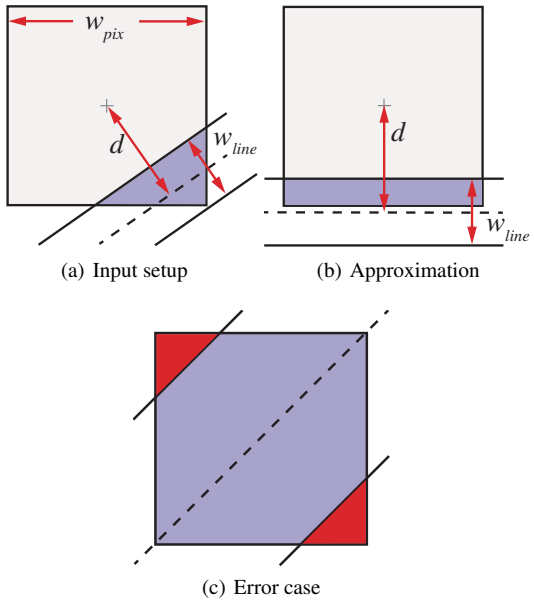
The coverage is a value between zero and one and represents the fraction of the pixel's area that a line covers. In order to efficiently compute the coverage, we introduce an approximation where we consider lines to always be aligned with one of the pixel grid's axes and that the line segment's start and end lies far outside of the pixel. With these assumptions, the coverage, c_{line}, can be expressed as:

$$A_{\text{pix}} = w_{\text{pix}}^2$$
$$A_{\text{line}} = w_{\text{pix}} \cdot \left(\min\left(\frac{w_{\text{pix}}}{2} - d, \frac{w_{\text{line}}}{2}\right) + \min\left(\frac{w_{\text{pix}}}{2} + d, \frac{w_{\text{line}}}{2}\right)\right)$$
$$c_{\text{line}} = \text{clamp}\left(A_{\text{line}}/A_{\text{pix}}, 0, 1\right).$$

With this approximation, the coverage only depends on the line's width (w_{line}), the pixel's width (w_{pix}) and d, the distance from the center of the pixel to the line (Figures 9(a) and 9(b)). Note that a factor of w_{pix} appears on both sides of the division in the final step, and an implementation can thus avoid multiplications with w_{pix} in the first place.

In our experiments, the error from the approximation is always less than 0.1 of coverage. The worst case occurs for thin lines (approximately one pixel wide) centered on a pixel and at a 45° angle (see Figure 9(c)). Our approximation overestimates the coverage to 1.0 (full coverage) instead of about 0.914. Figure 13(b) and 13(c) show a comparison in practice for our coverage-based anti-aliasing.

The cheap approximation enables its use for the procedural styling. We overlay borders and other details onto the main line by evaluating the approximate coverage of the details and blending them onto the main line. This causes the different elements to automatically blend into a single average color as the line gets thinner and details become smaller than a single pixel. In particular, symmetric details (such as borders on each side of a line) may be produced by (procedurally) placing two lines with different widths on top of each other.



(a) Input setup (b) Approximation

(c) Error case

Figure 9: *Coverage estimate. (a) The goal is to estimate the coverage, that is, the highlighted area (purple) as a fraction of the whole area. (b) Our method approximates coverage as if the line were aligned with the pixel setup. (c) The estimated coverage deviates from the real coverage in some cases. A worst case occurs when the line is centered on the pixel at a* 45° *angle; the worst observed error is nevertheless less than* 0.1 *of coverage per pixel.*

4. Results

Our test results were made on an Intel Core i7 3.5 GHz, 16 GB RAM, Nvidia GTX1080 8GB machine using C++ and OpenGL. The G-Buffer stage as well as the Deferred Line Shading pass are performed on a 3840×2160 framebuffer, which is later downsampled to 1920×1080 for viewing. Our implementation allows us to load and interactively visualize large-scale terrain and vector map datasets. In particular, the system supports exploration of large-scale vector maps interactively in a full 3D environment. Experimental results illustrated in Fig. 14 show that our approach can be used for large-scale interactive vector map visualization and achieves a good performance for all views and flyovers. Resulting images can be seen in Fig. 1, 13 and 14.

Tab. 2 lists the vector map datasets in detail and provides timings for the line assignment stage as well. The use of hierarchical spatial line indexing within the grid-based clustered line buffer makes the interactive exploration of large vector maps possible. Vector maps with several millions of line segments can be visualized at interactive frame rates with our methods using the maximal cluster sizes indicated in Tab. 2. The rendering performance is depending on the per-pixel line search effort which relies on the distance calculation costs, the number of lines and their organization within each cluster in the line buffer. Fig. 4 shows an example for the content of the cluster information.

In comparison to our first implementation in [TBP16] we could achieve a significant performance gain thanks to a reimplementation of the deferred line shading stage. Significant increases in performance (approx. $6-9\times$ speedup, w.r.t. Tab. 2 and [TBP16]) could be demonstrated by carefully identifying parts where double precision is needed and where single precision is sufficient to display the data without any precision artifacts. Double precision is only required for the distance calculation and the view transfor-

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

Data set	Vector map Lines	Cluster max. size	Line Assignment	Render time
Vorarlberg ski slopes	64,724	835	3.37 s	8 ms
Vorarlberg streets	728,661	631	9.8 s	8 ms
New York PlutoMap 14v1	5,578,677	2,230	38.06 s	15 ms
Switzerland (TLM streets)	16,556,412	3,429	101.7 s	10 ms
Carinthia (isolines)	31,297,095	4,650	120.7 s	20 ms

Table 2: *Dataset overview. For each vector map we used a fixed cluster grid of* 256×256*. Average render times measured for viewpoints (3840 × 2160 for 2 × supersampling) shown in Fig. 13(d), 13(a), 13(c), including ∼5-10 ms for rendering terrain.*

mation. The traversal of the binary hierarchy can be done in float precision. In addition, we optimized the shader according to standard procedures such as reduced memory accesses.

In contrast to texture-based visualizations, it can be guaranteed that the visual result is a pixel-precise rendering as demonstrated in Fig. 10. Our system maintains full pixel precision at any zoom-in factor even close to the terrain. In these situations, texture mapping falls short, because it will always suffer from resolution artifacts at some point, see also Fig. 10(f). In contrast to geometric line rendering approaches, intersecting or floating line artifacts as shown in Fig. 2(a), as well as z-fighting problems as shown in Fig. 10(a) can be avoided. Furthermore our method offers the possibility to introduce a coverage based anti aliasing method as described in Sec. 3.5. The visual improvement is shown in Fig. 13(b) and 13(c).

Our deferred line rendering is able to support interactively changing visualization parameters such as line size or styling properties as well as view-dependent data selection without reloading or re-rasterization of any textures. Examples for such advanced styles can be seen in Fig. 10, 11 and 13. Other styles properties are possible as well as style ordering shown in Fig. 11 and line stipples shown in Fig. 13(e). Furthermore, a screenshot of an interactive pixel-precise vector map data-lens example is shown in Fig. 12.

5. Conclusion

In this paper we present a novel approach for handling vector data sets within interactive 3D environments. Our approach represents an efficient and flexible solution for different purposes. It can be used for interactive exploring and editing vector maps and vector map styles as well as for large-scale visualizations. In particular, it is also suitable for dynamic level-of-detail and out-of-core geographic visualization systems. Our next steps are the extension of the system to work with polygonal objects and incorporating a vector map out-of-core level-of-detail system to make larger data sets accessible.

Acknowledgements

The authors want to thank the Swiss Federal Office of Topography Swisstopo for providing the Swiss VECTOR25 and SwissTLM data sets as well as the Landesvermessungsamt Feldkirch, Austria, for providing the data sets of Vorarlberg. This project was partially supported by a Forschungskredit of the University of Zürich (grant no. FK-16-015) and a Swiss National Science Foundation (SNSF) research grant (project no. 200021_169628).

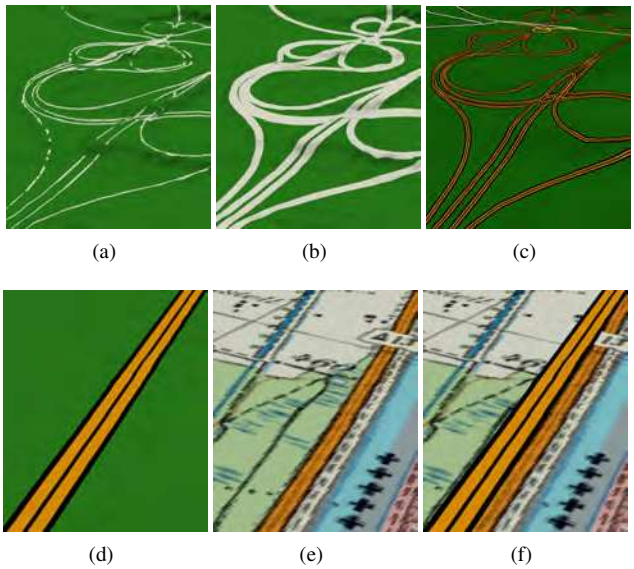


Figure 10: *Comparison of our approach (b,c,d,f) with normal 3D geometry (a) and texture based rendering (e): (a) Street rendered as a geometric object. (b) Street rendered with our method. (c) & (d) Streets rendered with our method using an advanced vector style. (e) A street rendered with texture mapping at highest resolution available. (f) Overlay of (d) and (e) for direct comparison.*

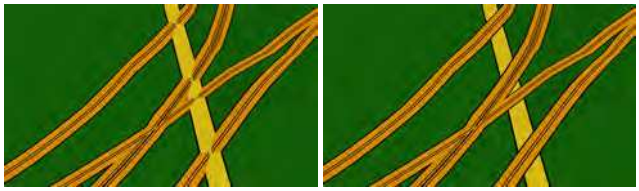
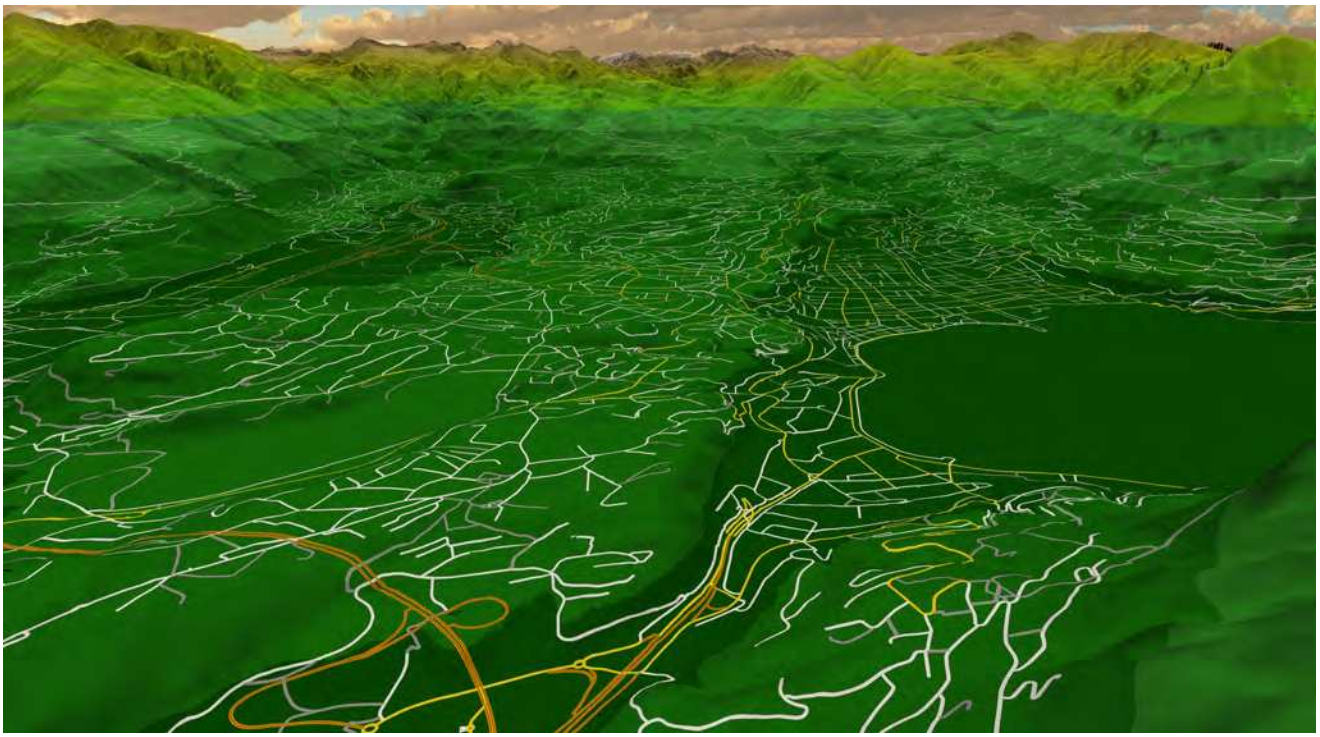


Figure 11: *Overlapping line segments. (a) Without ordering the method will introduce an exact distance-based split. (b) Styles can be selectively prioritized over other styles so that highways (orange) can be drawn over regular streets (yellow).*

Figure 12: *Example for interactive editing functionality. It is possbile to interactivly fade out specific categories of streets from the vector map on a pixel-precise basis.*

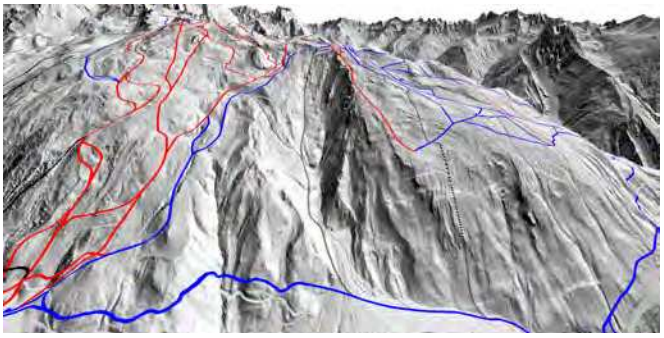
© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.



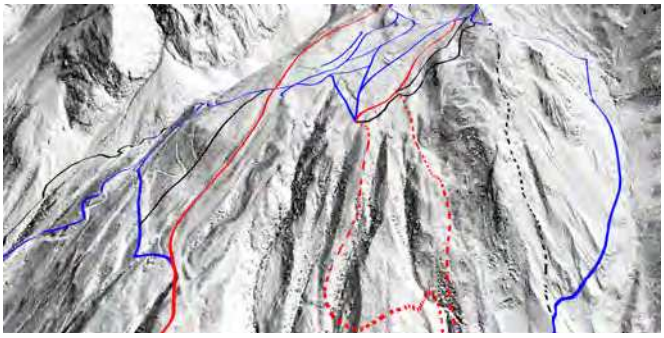
(a)

(b)

(c)



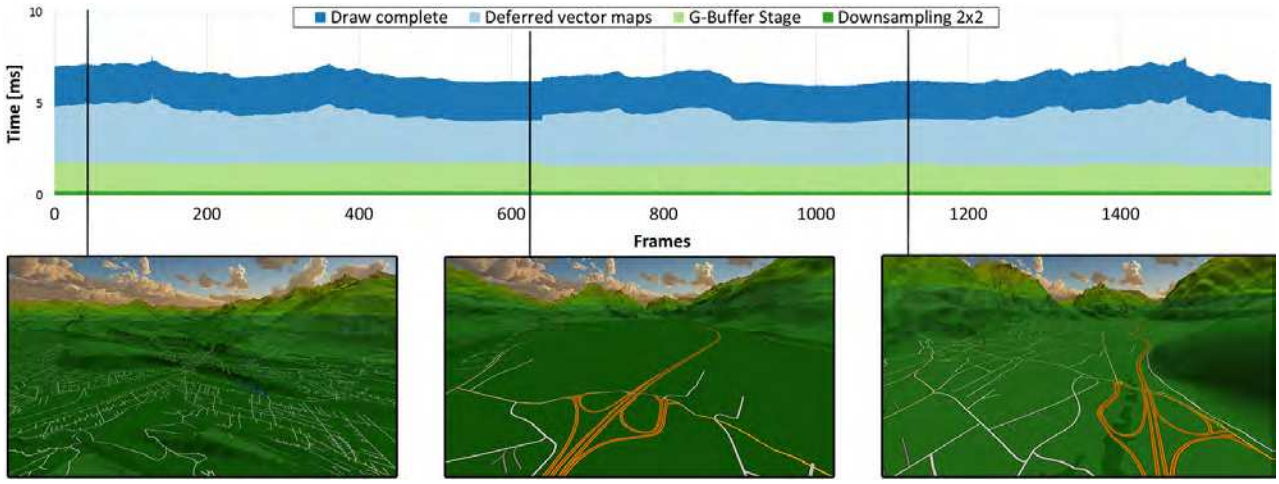
(d)



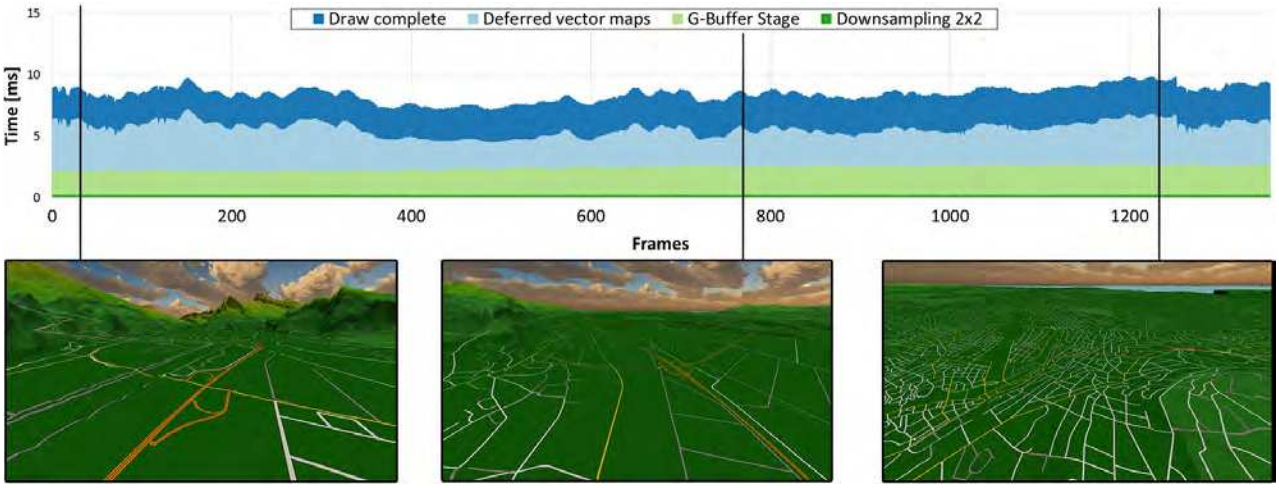
(e)

Figure 13: *(a) An example for line rendering of street data with styling pattern. (b) & (c) Carinthia isolines with black contour lines. (b) Distant contour lines suffer from aliasing artifacts. (c) Distant contour lines rendered with our coverage based antialiasing. (d) Visualization of ski slopes over a hill-shade textured terrain with blue and red colored slopes for beginner and advanced levels respectively. (e) Zoom in to ski slopes showing off-piste tracks with a stippled style pattern.*

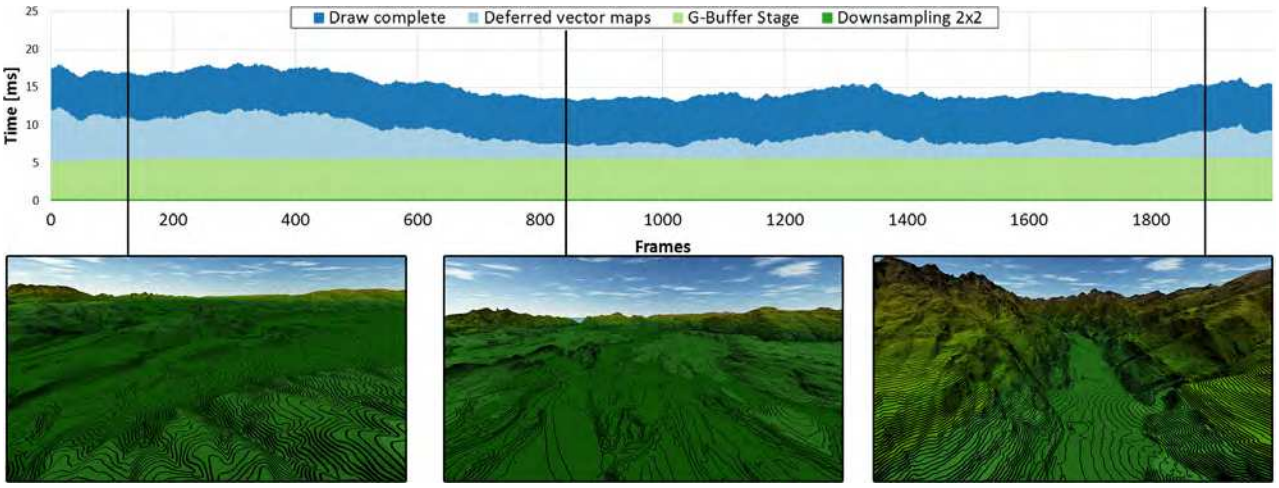
© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.



(a)



(b)



(c)

Figure 14: *Rendering times for every frame, including the G-buffer LOD terrain rendering pass, the deferred vector map pass and a downsampling pass. Framebuffers have size* 3840×2160 *using line coverage anti-aliasing and 2x2 downsampling for a* 1920×1080 *output resolution: (a) Vorarlberg flyover (b) Switzerland flyover (c) Carinthia flyover. Screenshots below measurements show views at particular frame times.*

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

References

[BGP09] BÖSCH J., GOSWAMI P., PAJAROLA R.: RASTeR: Simple and efficient terrain rendering on the GPU. In *Proceedings Eurographics - Area Papers* (2009), pp. 35–42. doi:10.5167/uzh-29729. 2

[BN08] BRUNETON E., NEYRET F.: Real-time rendering and editing of vector-based terrains. *Computer Graphics Forum 27*, 2 (April 2008), 311–320. doi:10.1111/j.1467-8659.2008.01128.x. 2

[BS08] BAVOIL L., SAINZ M.: Screen space ambient occlusion. In *ShaderX 7* (2008), NVIDIA Corporation. 4

[CML11] CHAJDAS M. G., MCGUIRE M., LUEBKE D.: Subpixel reconstruction antialiasing for deferred shading. *Proceedings Interactive 3D Graphics and Games* (2011), 15–22. doi:10.1145/1944745.1944748. 4

[CR11] COZZI P., RING K.: *3D Engine Design for Virtual Globes*. A. K. Peters, Ltd., 2011. 2

[DKW09] DICK C., KRÜGER J., WESTERMANN R.: GPU ray-casting for scalable terrain rendering. In *Proceedings Eurographics - Area Papers* (2009), pp. 43–50. doi:10.2312/ega.20091007. 2

[DXZS13] DENG B., XU D., ZHANG J., SONG C.: Visualization of vector data on global scale terrain. In *Proceedings International Conference on Computer Science and Electronics Engineering* (2013), pp. 85–88. doi:doi:10.2991/iccsee.2013.22. 3

[DZY08] DAI C., ZHANG Y., YANG J.: Rendering 3D vector data using the theory of stencil shadow volumes. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 37* (2008), 643–648. 3

[GMBP10] GOSWAMI P., MAKHINYA M., BÖSCH J., PAJAROLA R.: Scalable parallel out-of-core terrain rendering. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization* (2010), pp. 63–71. doi:10.2312/EGPGV/EGPGV10/063-071. 2

[KD02] KERSTING O., DÖLLNER J.: Interactive 3D visualization of vector data in GIS. In *Proceedings ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2002), pp. 107–112. doi:10.1145/585147.585170. 2, 3

[KJCH15] KANG H., JANG H., CHO C.-S., HAN J.: Multi-resolution terrain rendering with GPU tessellation. *The Visual Computer 31*, 4 (April 2015), 455–469. doi:10.1007/s00371-014-0941-6. 2

[LD12] LIKTOR G., DACHSBACHER C.: Decoupled deferred shading for hardware rasterization. In *Proceedings ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2012), pp. 143–150. doi:10.1145/2159616.2159640. 4

[LH04] LOSASSO F., HOPPE H.: Geometry clipmaps: Terrain rendering using nested regular grids. *ACM Transactions on Graphics 23*, 3 (August 2004), 769–776. doi:10.1145/1015706.1015799. 2

[LKES09] LIVNY Y., KOGAN Z., EL-SANA J.: Seamless patches for GPU-based terrain rendering. *The Visual Computer 25*, 3 (February 2009), 97–208. doi:10.1007/s00371-008-0214-3. 2

[MAAS15] MAHDAVI-AMIRI A., ALDERSON T., SAMAVATI F.: A survey of digital earth. *Computers and Graphics 53* (December 2015), 95–117. doi:10.1016/j.cag.2015.08.005. 2

[OBA12] OLSSON O., BILLETER M., ASSARSSON U.: Clustered deferred and forward shading. In *Proceedings ACM SIGGRAPH/Eurographics Symposium on High-Performance Graphics* (2012), pp. 87–96. doi:10.2312/EGGH/HPG12/087-096. 4

[OC11] OHLARIK D., COZZI P.: A screen-space approach to rendering polylines on terrain. In *ACM SIGGRAPH Posters* (2011), pp. 68:1–1. doi:10.1145/2037715.2037792. 4

[Per12] PERSSON E.: Graphics gems for games: Findings from avalanche studios. ACM SIGGRAPH Advances in Real-Time Rendering in Games - Course Material, August 2012. URL: http://advances.realtimerendering.com/s2012/index.html. 7

[RRPCC12] RIPOLLES O., RAMOS F., PUIG-CENTELLES A., CHOVER M.: Real-time tessellation of terrain on graphics hardware. *Computers & Geosciences 41* (April 2012), 147–155. doi:10.1016/j.cageo.2011.08.025. 2

[SGK05] SCHNEIDER M., GUTHE M., KLEIN R.: Real-time rendering of complex vector data on 3D terrain models. In *Proceedings International Conference on Virtual Systems and Multimedia* (2005), pp. 573–582. 3

[SLL08] SUN M., LV G. L., LEI C.: Large-scale vector data displaying for interactive manipulation in 3D landscape map. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 37* (2008), 507–512. 2

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-D shapes. In *Proceedings ACM SIGGRAPH* (1990), pp. 197–206. doi:10.1145/97879.97901. 4

[TBP16] THÖNY M., BILLETER M., PAJAROLA R.: Deferred vector map visualization. In *Proceedings ACM SIGGRAPH Asia Symposium on Visualization* (2016), pp. 16:1–16:8. doi:10.1145/3002151.3002157. 2, 7

[VTW11] VAARANIEMI M., TREIB M., WESTERMANN R.: High-quality cartographic roads on high-resolution DEMS. *Journal of Winter School of Computer Graphics 19* (2011), 41–48. 3

[WKW*03] WARTELL Z., KANG E., WASILEWSKI T., RIBARSKY W., FAUST N.: Rendering vector data over global, multi-resolution 3D terrain. In *Proceedings Eurographics Symposium on Data Visualization* (2003), pp. 213–222. doi:10.2312/VisSym/VisSym03/213-222. 2

[WLB09] WANG X., LIU J., BI J.: Rendering of vector data on 3D virtual landscapes. In *Proceedings IEEE International Conference on Information Science and Engineering* (2009), pp. 2125–2128. doi:10.1109/ICISE.2009.880. 2, 3

[WSL12] WILKIE D., SEWALL J., LIN M. C.: Transforming gis data into functional road models for large-scale traffic simulation. *IEEE Transactions on Visualization and Computer Graphics 18*, 6 (2012), 890–901. doi:10.1109/TVCG.2011.116. 3

[XSWJ10] XU Y., SUI Z., WENG J., JI X.: Visualization methods of vector data on a Digital Earth System. In *Proceedings International Conference on Geoinformatics* (2010), pp. 1–5. doi:10.1109/GEOINFORMATICS.2010.5567902. 3

[YZM*11] YANG L., ZHANG L., MA J., KANG Z., ZHANG L., LI J.: Efficient simplification of large vector maps rendered onto 3D landscapes. *IEEE Computer Graphics and Applications 31*, 2 (March/April 2011), 14–23. doi:10.1109/MCG.2010.63. 3

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.