Efficient and Accurate Rendering of Vector Data on Virtual Landscapes

Martin Schneider Reinhard Klein

University of Bonn
Institute of Computer Science II
Computer Graphics
Römerstraße 164
53117 Bonn, Germany
{ms,rk}@cs.uni-bonn.de

ABSTRACT

In geographical information systems (GIS) vector data has important applications in the analysis and management of virtual landscapes. Therefore, methods that allow combined visualization of terrain and geo-spatial vector data are required. Such methods have to adapt the vector data to the terrain surface and to ensure a precise and efficient mapping. In this paper, we present a method that is based on the stencil shadow volume algorithm and allows high-quality real-time overlay of vector data on virtual landscapes. Since the method is a screen-space algorithm it is per-pixel exact and does not suffer from aliasing artifacts like texture-based techniques. In addition, since the method is independent of the underlying terrain geometry, its performance does not depend on the complexity of the data set but only on the complexity of the vector data.

Keywords

vector data, terrain rendering, GIS, shadow volumes

1 Introduction

Vector data is one of the fundamental information representation stored and managed in current GIS. It usually consists of points, lines, polygons, etc., encoding geographic entities, e.g. road networks, buildings, vegetation and soil types. Typically, vector data is either derived (semi-)automatically from measurements (e.g. satellite imagery or GPS) or is created manually through user input. Once generated, vector data can be examined and modified by the user, serving as a valuable resource for various kinds of further investigations. Methods for the visualization of vector data on a virtual landscape can broadly be divided into two different classes: texture-based and geometry-based techniques. The first group of methods rasterizes the vector data into a texture and projects it onto the terrain geometry by applying texture mapping techniques. The main drawback of this kind of methods is that the accuracy of the mapping is limited by texture resolution resulting in aliasing artifacts. An especially critical scenario for these methods are large environments with vector data of considerable spatial extent. Increasing texture resolution alleviates the aliasing problems but at the expense of occupying equally increasingly large amounts of valuable texture memory.

Figure 1: Visualization of roads in the Wettersteingebirge (Germany).

Methods belonging to the second class create geometry from the vector data by adapting it to the terrain surface. Most terrain representations are based on a level-of-detail (LOD) terrain model whose geometry is refined according to the current viewpoint. Therefore, geometry from the vector data has to be created and adapted to each LOD. This results in a drastically increased primitive count, if precomputed and stored, or enormous additional computational costs, if performed at run-time. Furthermore, a suitable offset has to be determined and added to the generated coplanar primitives in order to avoid z-buffer stitching artifacts.
The basic idea of our approach is to extrude the vector data to polyhedra and use them to create a mask in the stencil buffer. The generated mask corresponds to the projection of the vector data onto the terrain surface. It is applied to the scene by rasterizing geometry covering at least the entire mask with the appropriate stencil test enabled. An advantage of our method is that it works in screen-space and can therefore be performed per-pixel exact. At the same time, it is independent of the underlying terrain geometry and utilized rendering engine offering high performance even for very high resolution data sets and a wide applicability.
The remaining part of the paper is structured as follows. First, we briefly review related work. Then, in Section 3 we formulate the problem of projecting vector data onto the terrain surface as a point-in-polyhedra problem and establish a connection to the shadow determination problem. We describe our approach in Section 4, discuss it and show results in Section 5 and draw conclusions in Section 6.

2 Previous Work

Real-time terrain rendering techniques have been extensively studied for a long time but methods for projecting additional vector data on a virtual landscape have gained less attention. The few methods existing so far can basically be divided into texture-based and geometry-based approaches.
A texture-based approach to visualize vector data was proposed by Kersting et al. in [KD02]. Textures containing the vector data are generated on-the-fly using p-buffers. An on-demand texture pyramid that associates equally sized textures with each quadtree node is used to improve visual quality when zooming in. However, many expensive p-buffer switches have to be performed, which leads to decreased rendering performance. Even with more recent and efficient extensions (e.g. framebuffer objects) each switch still requires a complete pipeline flush. In [SGK05] a texture-based approach is presented that also creates textures on-the-fly in an offscreen buffer. A perspective reparameterization adopted from perspective shadow mapping is applied taking into account the current point-of-view. The reparameterization allows a drastically improved utilization of the available texture resolution, which results in reduced aliasing artifacts.
Wartell et al. [WKW^{+}03] presented an algorithm and an associated data structure that allows rendering of polylines on multiresolution terrain geometry. Since their system is based upon a continuous level-of-detail terrain rendering engine, an adaption of the polyline to the current state of geometry is required at run-time resulting in additional computational costs. In addition to the previously mentioned texture-based approach, Schneider et al. also presented in [SGK05] a geometry-based approach for rendering engines based on static level-of-details. The vector data geometry is mapped to each LOD in a preprocessing step and integrated in the used quadtree ensuring rendering of corresponding terrain and vector data LODs. Since the number of geometric primitives that have to be created grows with the terrain complexity, this method is not suited for very high resolution data sets, especially for vector data covering large areas.

3 Problem Formulation

Rendering vector data on virtual landscapes requires the determination of its projection along the nadir onto the terrain geometry. This area is equivalent to the parts of the terrain that are inside the infinite projection pyramid defined by the vector data extruded towards the geocenter and in the opposite direction. Thus, by restricting the projection volume appropriately at both ends, i.e. above and below the terrain surface, the problem of determining the projection area can be interpreted as a point-in-polyhedra problem.

3.1 Point-In-Polyhedra Algorithm

A general algorithm for performing a point-in-polyhedra test can be formulated as follows: Assume a point O that is outside all polyhedra is given. For a point P in question we consider the line segment $\overline{PO}$. The objective is then to find all intersections of this line segment and the polyhedra. At each intersection a counter is incremented if the line enters and decremented if the line exits the polyhedron. After all intersection tests have been performed the counter corresponds to the number of polyhedra containing P. For our purposes, it is sufficient to note that the counter is zero when P is outside all polyhedra which corresponds to the absence of vector data. If the counter is non-zero, P is inside at least one polyhedron denoting the presence of vector data.
Note the relation to the problem of shadow determination that can also be expressed as a point-in-polyhedra problem [Cro77]. Crow defined a shadow volume as a region of space that is in the shadow of a particular

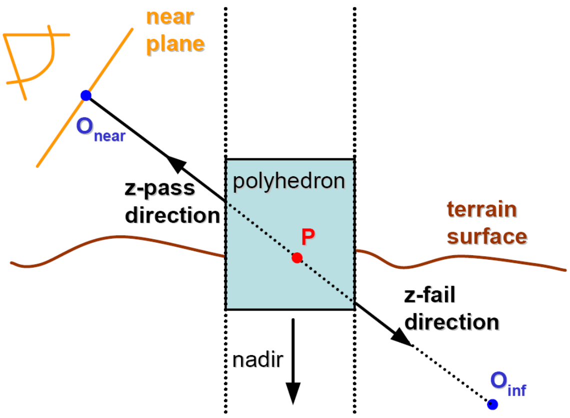


Figure 2: A 2d diagram of the point-in-polyhedra problem.

occluder given a particular ideal light source. The shadow test determines if a given point is inside the shadow volume of any occluder. Heidmann [Hei91] adapted Crow's algorithm to hardware acceleration by exploiting the stencil buffer to evaluate the per-pixel count for the point-in-polyhedra test.

By rendering the polyhedra's front- and back-faces to the stencil buffer the test can be performed simultaneously for all visible points of a scene. Each pixel is interpreted as a point P and the ray from the viewpoint through the pixel is considered. There are two possible choices for a point O along the ray outside any polyhedron (see Figure 2). The first choice is the intersection O_{near} of the ray and the near clipping plane. This point is known to be outside all polyhedra if the near clipping plane does not intersect any polyhedra. The other choice is the point O_{inf} at infinity at the far end of the ray. This point is always outside all polyhedra because it is infinitely far away from the scene.

Note that entering intersections must correspond to polyhedra front-faces and exiting intersections must correspond to polyhedra back-faces. Thus, counting intersections can be performed by rasterizing the polyhedra faces in the stencil buffer. The stencil operation must be configured to increment the stencil value when a front-face is rasterized and to decrement the count when a back-face is rasterized. Intersection counting is actually performed only along $\overline{PO_{near}}$ or $\overline{PO_{inf}}$ respectively and not along the entire ray. Since P is a visible point, these two kinds of intersections can be discriminated by a depth test. If O_{near} at the near clipping plane is used, only polyhedra faces passing the depth test are counted, if O_{inf} at infinity is chosen, only the polyhedra faces failing the depth test are considered. Counting towards the near clipping plane is thus called the z-pass method whereas counting towards infinity the z-fail method [EK02, MFT$^+$03]. After rendering, a stencil value of zero indicates that the same number of front- and back-faces were rendered and thus the corresponding pixel is outside all polyhedra otherwise the pixel is inside at least one polyhedron.

In the shadow volume algorithm the z-pass method fails when the shadow volume intersects the near clipping plane. This near clipping problem was the reason for the development of the z-fail technique which processes shadow volume fragments that fail (instead of pass) the depth test. This approach moves the problems from the near to the far clipping plane which can be handled robustly by moving the far plane to infinity. However, this robustness comes at the expense of performance since in the z-fail case the shadow volumes must be closed at both ends.

4 Our Approach

In our method we take up the idea to utilize the stencil buffer to perform an efficient point-in-polyhedra test originally used for shadow determination. Our technique consists of three parts: constructing the polyhedra from the vector data, rendering the polyhedra to the stencil buffer to create a mask and applying the mask to the scene.

4.1 Vector Data Extrusion

In the first step we need to extrude the vector data geometry into polyhedra that are afterwards rendered into the stencil buffer to generate an appropriate mask. Construction is started by duplicating each vertex of the vector data. One vertex of each of the created pairs is translated towards the geocenter, the remaining vertices are moved into the opposite direction. The group of upper and lower vertices constitute the polyhedron's top and bottom cap. The amount of translation has to be chosen such that the top and bottom cap are located

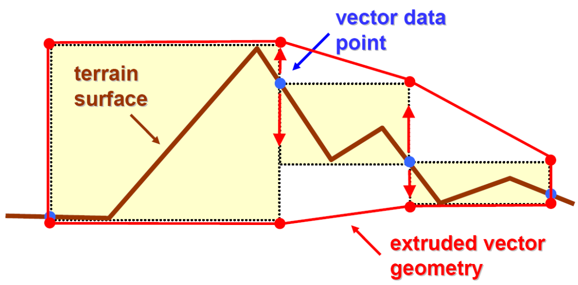


Figure 3: A 2d diagram of the extrusion of a linestrip. The original vector data points (blue) are duplicated and moved to the upper and lower bounds of the line segments bounding box constituting the top and bottom caps (red).

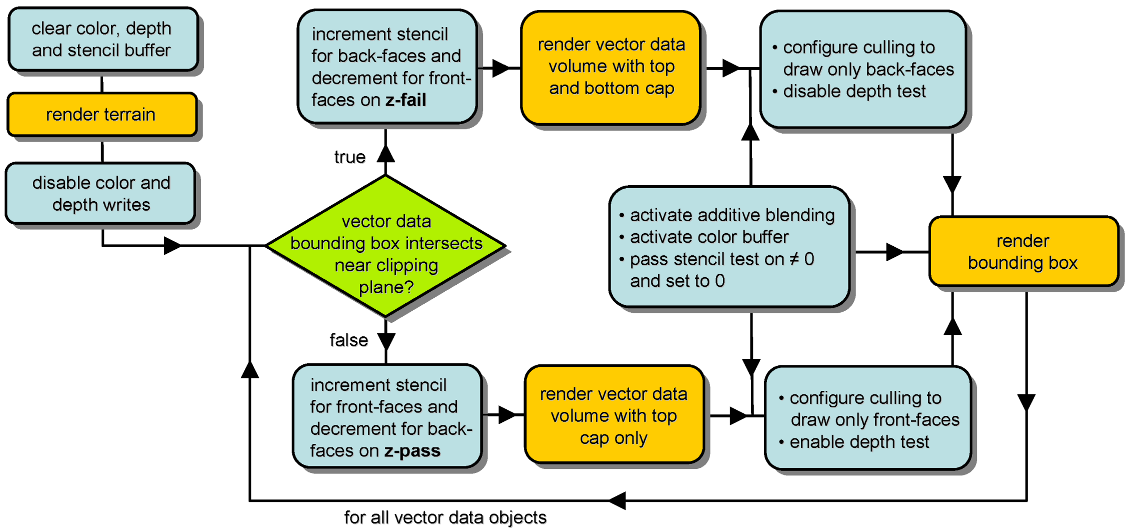


Figure 4: Flow chart of the rendering process.

completely above and below the terrain surface respectively. Applying the described construction the resulting polyhedron encloses the part of the terrain surface that is supposed to contain the vector data.

In order to minimize the high rasterization workload potentially caused by large polyhedra (usually the bottleneck when using shadow volumes) we reduce the size of the polyhedra. To accomplish this, we move the top and bottom caps towards the terrain surface from both sides as far as possible but without intersecting it. In our implementation we utilize the bounding boxes of the quadtree cells inherent in the terrain rendering engine. In particular, the bounding boxes encode an upper and lower bound of the enclosed terrain and therefore provide conservative but reasonable upper and lower bounds for the polyhedra caps as well.

In the case of linestrips as vector data primitives we consider each line segment separately. The height values of the corresponding vertices of the top and bottom cap are the minimum and maximum height values of the bounding boxes containing the projection of the line segment (see Figure 3). In the case of polygons we use the minimum and the maximum height value of the bounding boxes enclosing the projection of the whole polygon.

The constructed polyhedra are tesselated ensuring a consistent winding order with all face normals pointing outwards. The resulting geometry of each object is stored in its own vertex buffer object remaining valid as long as the vector data is not modified.

4.2 Vector Volume Rendering

The creation of the mask and its application to the scene has to be performed for each object separately. This is necessary because each object is allowed to have a different color. Therefore, first rendering all objects to the stencil buffer and applying the generated mask afterwards at once by rendering a screen sized quad would prevent us from distinguishing the separate objects. If there are only objects with few different colors in the scene, sorting by color and then rendering each color group at once can help to reduce the required fill rate and state changes.

4.2.1 Generate Mask in the Stencil Buffer

Now that we have created the polyhedra from the vector data they can be rendered into the stencil buffer. It is common practice when using shadow volumes to decide on a per frame and volume basis if the z-pass or the z-fail technique is used. The z-pass method is preferred because it does not need capping, i.e. top and bottom caps need not to be rendered, and is therefore generally faster than z-fail. However, since the z-pass technique does not produce correct results when the near plane intersects a shadow volume, the robust z-fail technique is applied in these cases. We follow this approach and decide conservatively which method to use by checking if the current viewpoint is inside the bounding box of the considered polyhedron. Note that in contrast to shadow volumes we need a top cap in the z-pass method because in our case there is no occluder that covers the top end making a cap unecessary.

First, color, depth and stencil buffer are cleared and the terrain is rendered initializing the depth buffer with the required depth values. Next, depth buffer writing is disabled, but the depth test still remains active. Rendering is then restricted to the stencil buffer only. The polyhedron's faces are rendered using different stencil operations depending on whether they face towards or away from the camera. To this end, face culling is enabled and the polyhedron is rendered twice, one time with back-face culling enabled, the other time with front-face culling enabled. If the z-pass method is used, because the polyhedron does not intersect the near clipping plane, the values in the stencil buffer are modified when the depth test passes. The stencil value is incremented for fragments belonging to front-facing polygons and decremented for fragments belonging to back-facing polygons. If the z-fail technique is applied, values in the stencil buffer are modified when the depth test fails. The stencil value is incremented for fragments belonging to back-facing polygons and decremented for fragments belonging to front-facing polygons.

We make use of the OpenGL extensions EXT_stencil_wrap and EXT_stencil_two_side, if supported, that aim at simplifying the mask creation in the stencil buffer. The EXT_stencil_wrap extension specifies two additional stencil operations. These new operations are similiar to the existing increment and decrement operations, but wrap their result instead of saturating it, which leads to a reduction of the likelihood of incorrect shadow results due to limited stencil buffer resolution. The EXT_stencil_two_side extension provides two-sided stencil testing where the stencil-related state can be configured differently for front- and back-facing polygons. With two-sided stencil testing front- and back-faces can be rendered in a single pass instead of two separate passes which may improve performance.

A simple triangle fan can be used to draw the top and bottom caps, without needing to triangulate it in advance, even if they are non-convex or contain holes. The fan itself may be convex but the pattern of front- and back-faces will produce the correct non-convex shape in the stencil buffer. The process is identical to that of computing the signed area of a polygon by constructing a fan. An example is shown in Figure 5. The concave polygon on the left is tessellated into a fan. In the middle the decrements caused by front-facing polygons (top) and the increments caused by back-facing polygons (bottom) are shown. Combining them results in the original concave polygon (right). This technique has previously been used for rendering filled silhouettes in the stencil buffer from possible silhouette edges for Silhouette Clipping [SGG+00].

Another issue we have to deal with in the z-fail case is to avoid far plane clipping. To accomplish this, we move the far plane to infinity by using the following projection matrix to transform from eye-space to

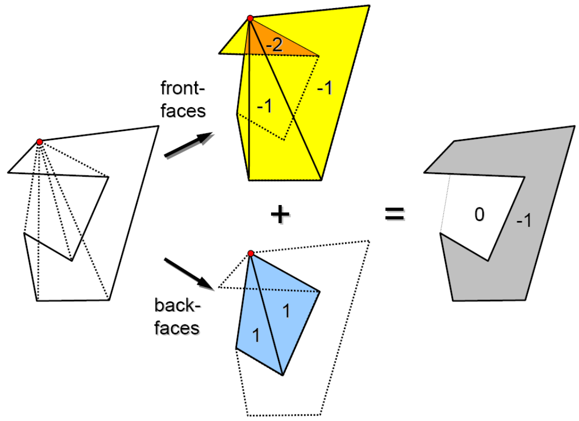


Figure 5: Tesselation of a concave polygon into a convex fan creating a concave mask in the stencil buffer.

clip-space:

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

where n and f are the respective distances from the viewer to the near and far clipping plane. $(l, b, -n)$ and $(r, t, -n)$ specify the (x, y, z) coordinates of the lower-left and upper-right corners of the near clipping plane. Positioning the far plane at infinity typically reduces the depth buffer precision only slightly. However, if the OpenGL NV_depth_clamp extension is supported and enabled during rendering of the polyhedra, the conventional projection matrix can be kept.

4.2.2 Apply Mask to the Scene

Now that we have created the mask in the stencil buffer we apply it to the scene. Therefore, we reactivate writing to the color buffer and activate additive blending. The stencil test is configured to pass only when the value in the stencil buffer does not equal zero. Instead of drawing a screen-sized quad to apply the mask to the scene, we rasterize the bounding box of the respective polyhedron in order to save rasterization bandwith. This is performed with depth test enabled and drawing only front-faces in the z-pass case and with depth test disabled and drawing only back-faces in the z-fail case. In order to avoid a complete stencil clear per object we configure the stencil function to set the value in the stencil buffer to zero for each fragment that passes the stencil test. As a consequence, the entire stencil buffer is zero again when rendering is finished and does not need to be cleared.

5 Results and Discussion

The presented algorithm allows high-quality vector data visualizaton as provided by other geometry-based methods. However, it does not suffer from their shortcomings, namely the expensive adaption process and the increased primitive count coupled with the terrain complexity. In our method we actually render a multiple of the amount of primitives present in the original vector data but it is a small constant factor independent of the underlying terrain. Considering the fast evolving acquisition devices resulting in ever higher sampled terrain data sets this fact will become even more important in the future.

In comparison to texture-based techniques that immediately render the vector data into a texture our method demands slightly more primitives to be rendered but provides superior quality. Interactive editing and manipulation of the vector data is also possible with our method. It only requires updating the polyhedra of the modified vector data object allowing interactive response.

In Figure 6 some results obtained with our method are shown. It is capable of visualizing thin features, like roads, as well as large and complex polygons (that may be concave and contain holes) accurately and efficiently. Note that the small deviations visible in some places of the vector data, e.g. the roads, from their counterparts in the terrain data are due to inaccuracies in the definition of the given vector data (which originate from a different source than the terrain data and were created at a diferent time) and are by no means due to deficiencies of our method.

We currently implement vector data extrusion on the CPU. There also exist techniques for purely hardware accelerated stencil shadow volumes [BS03] that perform silhouette detection and shadow volume extrusion on the GPU. However, there is no need for silhoutte detection in our case because silhouettes correspond to the vector data and are therefore explicitly given. Moreover, available vector data is static and consequently the advantage of a GPU implementation, to be able to easily extrude geometry that is animated by a vertex program, is currently of no use. This may become an issue in the future when time-varying vector data, e.g. for modelling processes, will be available.

Another issue are the problems appearing in steep slopes which are not a special problem of our method but a general one. When rendering objects that should retain a constant width on steep slopes, e.g. roads or contour lines, their projection is distorted.

6 Conclusions

We have presented an algorithm that is capable of visualizing vector data on virtual landscapes offering high-quality at real-time. The algorithm is robust, straightforward and requires no special hardware extensions that are not ubiquitous today. The fact that it is independent of the underlying terrain rendering engine allows an easy integration in any terrain visualization system. Since our method is not affected by changes in the terrain geometry it is especially suited to work with view-dependent LOD representations.

The presented technique has been implemented as a part of the Scarped [WMD+04] visualization engine.

References

[BS03] Stefan Brabec and Hans-Peter Seidel. Shadow volumes on programmable graphics hardware. *Comput. Graph. Forum*, 22(3):433–440, 2003.

[Cro77] F. Crow. Shadow algorithms for computer graphics. In *Proceedings of SIGGRAPH*, pages 242–248, 1977.

[EK02] C. Everitt and M. Kilgard. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. *Published on-line at developer.nvidia.com*, 2002.

[Hei91] T. Heidmann. Real shadows real time. *IRIS Universe*, 18:28–31, 1991.

[KD02] O. Kersting and J. Döllner. Visualization of vector data in gis. In *Proceedings of the 10th ACM International Symposium on Advances in GIS*, 2002.

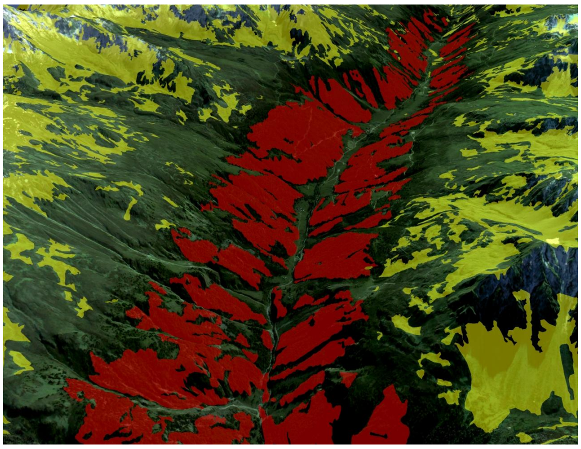
[MFT+03] McGuire M., Hugues J. F., Egan K. T., Kilgard M., and Everitt C. Fast, practical and robust shadows. *Technical Report CS03-19*, 2003.

[SGG+00] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 327–334. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

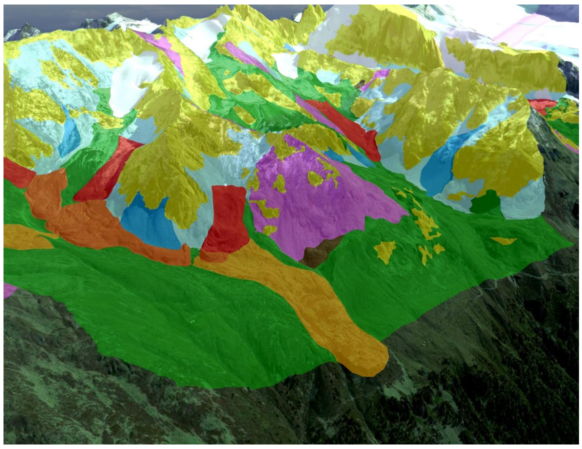
[SGK05] M. Schneider, M. Guthe, and R. Klein. Real-time rendering of complex vector data on 3d terrain models. In *Proceedings of The 11th International Conference on Virtual Systems and Multimedia*, pages 573–582, 2005.

[WKW+03] Z. Wartell, E. Kang, T. Wasilewski, W. Ribarsky, and N. Faust. Rendering vector data over global, multiresolution 3d terrain. In *Proceedings on the Symposium on Data Visualization*, volume 40, pages 213–222, 2003.

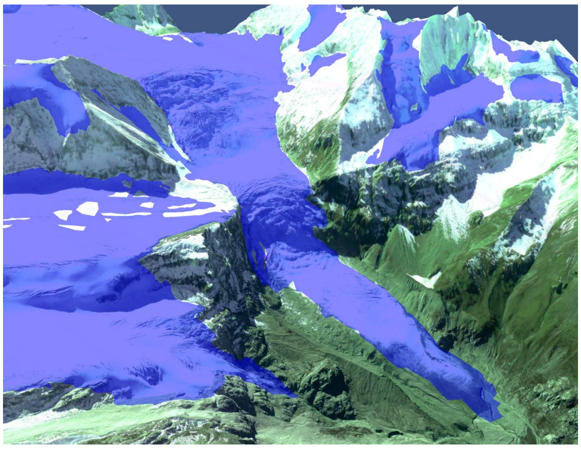
[WMD+04] R. Wahl, M. Massing, P. Degener, M. Guthe, and R. Klein. Scalable compression of textured terrain data. *Journal of WSCG*, 12(3):521–528, 2004.



(a) Forest (red) and debris (yellow)

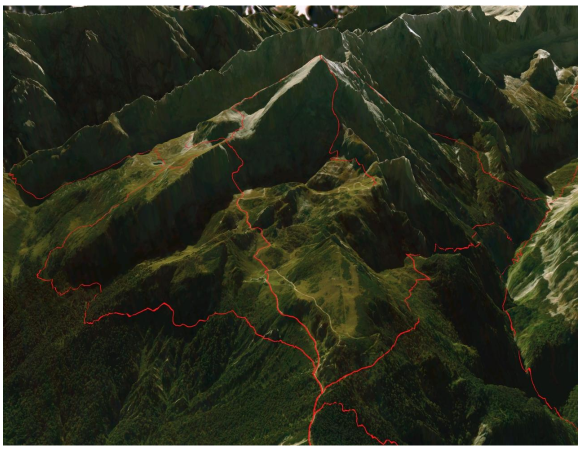


(b) A geomorphological map

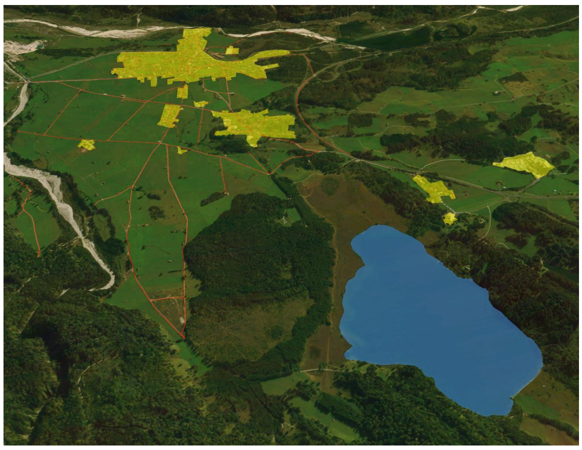


(c) The Turtmann glacier

(d) Roads and trails



(e) Trails



(f) Roads, villages and a nearby lake

Figure 6: Images (a)-(c) are located in the Turtmann valley (Switzerland) showing complex polygonal vector data rendered semi-transparently on the landscape. In (d)-(e) polygonal and polyline vector data in the Wettersteingebirge (Germany) representing roads, trails, villages and a lake are overlayed on the terrain.