Project Report

Avalanche Risk visualization Layer for Alpine Maps

Jörg Christian Reiher

Matriculation Number: 12213117
e12213117@student.tuwien.ac.at

Supervisor: Adam Celarek

TU Wien
Faculty of Informatics
Institute of Visual Computing & Human-Centered Technology
Research Unit of Computer Graphics

15.10.2025

Abstract

An avalanche risk visualization layer has been implemented along the lines of an existing implementation [1] of a thesis [2] on the topic. Four layers can be accessed through a dedicated UI button: risk level, avalanche report, slope angle, and stop-or-go. These overlays are based on the EAWS report for the date selected in the UI (use swipe interface for date selection). The EAWS report is preprocessed and provided by a dedicated server Johannes Eschner was kind enough to configure. This server uses EAWS regions as of July 1, 2025. This reference date must be provided to the class `UIntIdManager` in the current implementation to ensure compatibility.

Contents

1 Class Structure

1.1 Shader and uniform buffer

`gl_engine.glsl`
`gl_engine.frag`
`nucleus/avalanche/eaws.h`
The avalanche layer is rendered by a custom fragment shader running after the vertex shader `tile.vert` which had to be modified to pass the correct altitude. The fragment shader relies upon an uniform buffer object (ubo) that stores the currently selected EAWS report data as an array of `uvec4`. Every EAWS region has an internal ID (`uint`) that corresponds to the index of its current report in the ubo. The internal ID of the EAWS region to which the fragment belongs is read from a texture array. The `vec4`-report that belongs to the fragment is organized as follows:

Index	Content
x	Exposure (negative value means no report available)
y	Bound (altitude of bound between low and high region)
z	Danger rating for high region
w	Danger rating for low region

See aforementioned thesis [2] for details. The shading uses lookup tables based on the colors suggested therein with two differences:

1. For avalanche warning level 0 the fragment shader applies a strong gray overlay for warning level and stop-or-go vizualization, while the reference implementation[1] renders no color layer (transparent) over the terrain for warning level visualization. See Fig.2.

2. At warning level 0 the fragement shader applies the same grey overlay for risk level vizualization while for this case the reference implementation [1] colors areas with more than 45 45-degree slope in black, very flat areas in white (very low risk) and in between in yellow (low-medium risk). See Fig.1.

The color according to user selected avalanche layer type (warning level, risk level, stop-or-go, slope) is mixed with the underlying terrain texture provided to `AvalancheWarningLayer` (see Sec.1.5).

1.2 EAWS Report Load Service

`nucleus/avalanche/ReportLoadService.h`
`nucleus/avalanche/ReportLoadService.cpp`
Upon date change in the UI the class `ReportLoadService` is responsible for loading the corresponding avalanche report from the report server [3], parsing it and updating the ubo for the fragment shader. `ReportLoadService` needs the class `UIntIdManager` that converts EAWS regions to the corresponding internal ID that determines the index of the

report data of an EAWS region in the ubo (see Sec.1.4). It obtains the `UIntIdManager` instance from `nucleus::avalanche::Scheduler` since both must use the same `UIntIdManager`. If an EAWS report contains an EAWS region that has no internal ID yet, `UIntIdManager` creates a new one on the fly. This is no problem since `ReportLoadService` and `nucleus::avalanche::Scheduler` both run in the same scheduler thread.

Note:
The report server[3] has been set up by Johannes Eschner and uses the geojson files [4] (region code AT02-AT08) as of July 1, 2025. This reference date must match the reference date the class `UIntIdManager` obtains (see Sec. 1.4) as well as the date of the pmtiles of the martin tile server (see Sec.1.3) to ensure compatibility. The avalanche reports are published for each EAWS region. EAWS regions are updated regularly (usually by subdividing existing regions). An avalanche report for a certain day is only valid for the set of EAWS regions that was the latest at that day. Older reports refer to outdated regions. The report server[3] consolidates reports at any date to match the set of EAWS regions as of July 1, 2025. This ensures that old reports (that refer to outdated EAWS regions) can be displayed.

1.3 Tile Scheduler

`nucleus/avalanche/eaws.h`, `nucleus/avalanche/eaws.cpp`
`nucleus/avalanche/scheduler.h`, `nucleus/avalanche/scheduler.cpp`
`nucleus/avalanche/setup.h`, `nucleus/avalanche/setup.cpp`
The class `nucleus::avalanche::Scheduler` generates rasterized tiles with internal IDs of EAWS Regions. It therefore has an `UIntIdManager` (which it shares with `ReportLoadService`) that transforms EAWS regions to internal IDs. A martin[5] tile server[6] for EAWS Regions has been set up by Adam Celarek and serves pmtiles as of July 1, 2025 (obtained from EAWS regions website[7]). See Sec.1.2 for compatibility requirements of the reference date.
The class `nucleus::avalanche::Scheduler` uses the function `nucleus::avalanche::draw_regions` to rasterize applicable EAWS regions of a tile. The result is an 8-bit RGB `QImage` that stores in every pixel the internal ID of the EAWS region occupying this pixel.
`nucleus::avalanche::Scheduler` then converts this `QImage` to a 16bit `uint` raster. This raster is written to the corresponding texture array by the class `gl_engine::AvalancheWarningLayer` (see Sec.1.5). The function uses `UIntIdManager` not only to convert EAWS Regions to internal IDs and their corresponding RGB colors but also
`UIntIdManager::m_reference_date` to determine which regions to draw and which ones not. The pmtiles file contains all EAWS regions, i.e. current as well as outdated regions. With the reference date, only those

regions are drawn that were current at that date. This ensures that every pixel can belong to at most one region.
Please see `doc/readme_eaws_vector_tile.md` for detailed information on EAWS region tiles. The struct `EawsTextureSchedulerHolder` is a wrapper for the setup of the scheduler and the tile loading.

Note:
The aforementioned conversion from an 8-bit RGB image to a 16-bit `uint` raster is neccessary since no solution has been found that lets the `QPainter` write 16-bit grayscale or RGB images to store the 16-bit internal ID.

1.4 UIntIdManager

`nucleus/avalanche/UIntIdManager.h`
`nucleus/avalanche/UIntIdManager.cpp`
The class `UIntIdManager` converts EAWS regions which have an EAWS Region ID (`QString`) defined by the EAWS to an internal ID (`uint`). It also converts internal IDs to 8-bit RGB color. The internal ID 0 is reserved for the case that no region is assigned. It also stores the reference date, which ensures that only regions valid at that date are processed. This is why in the current implementation
`nucleus::avalanche::Scheduler` constructs and owns an
`UIntIdManager` instance that it passes to `ReportLoadService`.

1.5 Avalanche Warning Layer

`gl_engine/AvalancheWarningLayer.cpp`,
`gl_engine/AvalancheWarningLayer.h`
The Alpine Maps framework provides the class `TextureLayer` for managing layers that can be drawn on the alpine geometry (e.g., an orthographic layer or a height map layer). It manages textures, ubos etc. The class `AvalancheWarningLayer` basically is a `TextureLayer` with some customisations. It is a friend class of `TextureLayer` and has a pointer to a `TextureLayer` which allows
`AvalancheWarningLayer` to bind textures from `TextureLayer` onto which the transparent avalanche warning colors are drawn in the fragment shader.

1.6 Terrain Renderer Rendering Context

The class `TerrainRenderer` holds a `RenderingContext` instance to which a `EawsTextureSchedulerHolder` and a
`nucleus::avalanche::ReportService` have been added. It also holds a `gl_engine::Context` instance to which the `AvalancheWarningLayer` has been added.

2 Results

The renderings in Fig.1 show similar results. Only the stop-or-go differs on a smaller scale which can be attributed to slightly different tile geometries. The result for the other layers are all in line and suggest correct implementation.

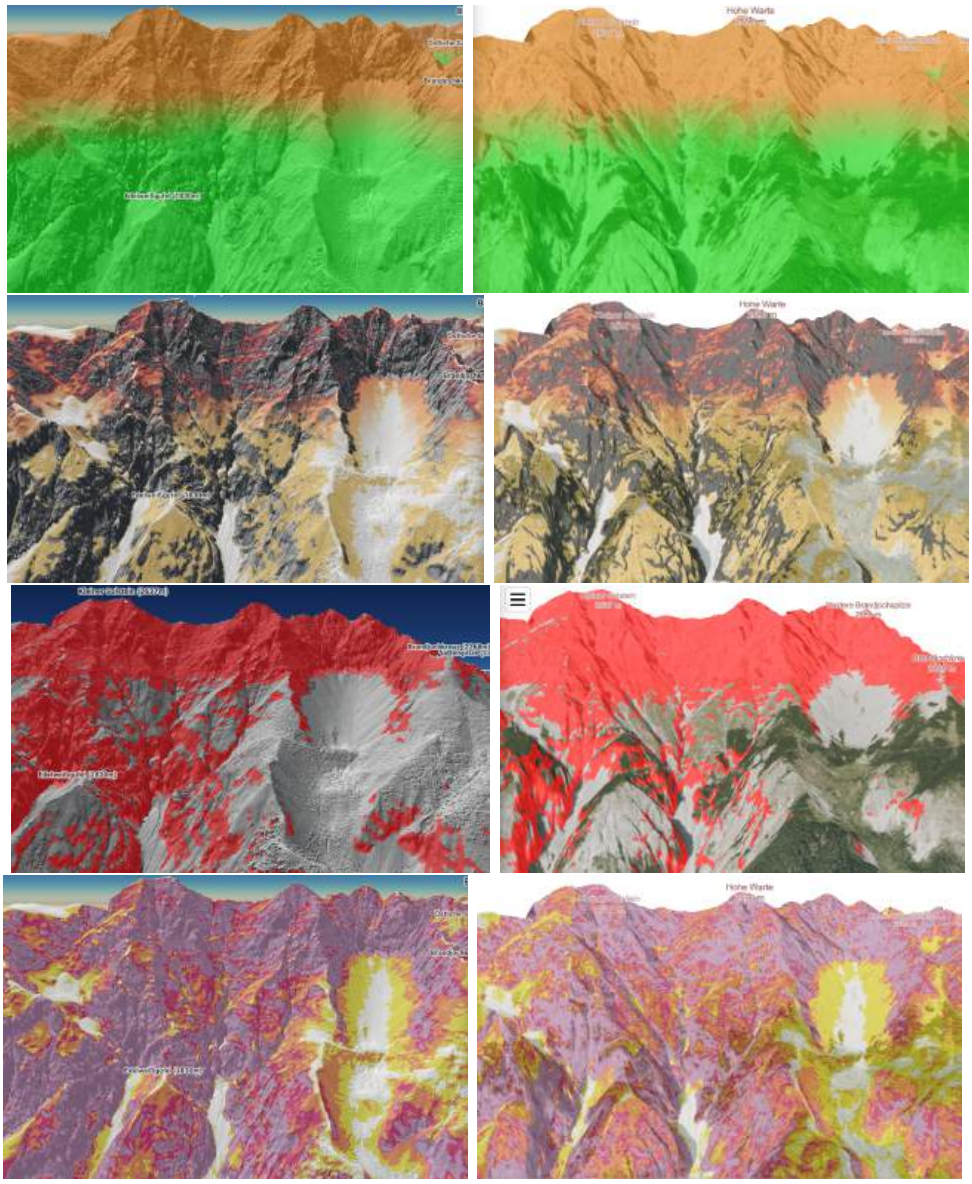

Figure 1: Top to bottom: Comparison of warning level, risk, stop-or-go and slope overlay in this implementation (left) and in the reference implementation[1] (right).

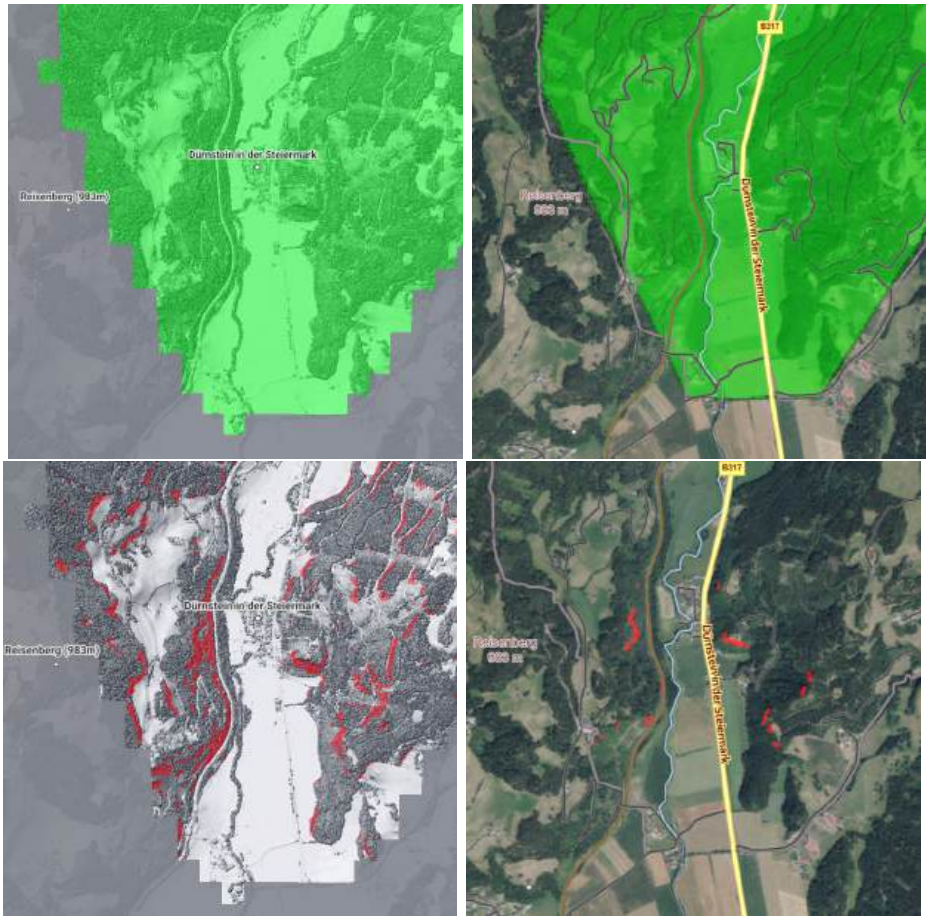

Figure 2: Comparison of how areas with no report are handled. Top: Warning level visualization, bottom: Stop-or-Go visualization. This implementation (left) uses a grey overlay where no report is available, while the reference implementation[1] (right) uses no overlay.

3 Future Work

- **Rasterization**: The rasterization with `QPainter` should be substituted by a dedicated rasterizer.
- **UI**: Professional icons could be used, and the menu should be responsive to varying screen sizes.
- **Slope Angle**: The slope angle should be averaged over a certain area, depending on the level of detail, to smooth out the effects of trees, houses, or small rocks.
- **Details on Demand**: Clicking on a pixel should open a small label with the report at that fragment, as in the reference implementation[1].

References

[1] J. Eschner, "Alpinemaps." https://alpinemaps.cg.tuwien.ac.at/?day=2024-12-12, 2023. Accessed: 2024-12-12.

[2] J. Eschner, "Real-time avalanche risk visualization on a large-scale geospatial dataset," master's thesis, TU Wien, 2023.

[3] J. Eschner, "Eaws report server." https://alpinemaps.cg.tuwien.ac.at/avalanche-reports-v2/get-current-report?date=2025-07-01, 2025. Accessed: 2025-07-01.

[4] EAWS, "Eaws regions." https://regions.avalanches.org/micro-regions/AT-02_micro-regions.geojson.json, 2025. Accessed: 2025-07-01.

[5] MapLibre, "Martin server tile server." https://martin.maplibre.org/, 2025. Accessed: 2025-07-01.

[6] A. Celarek, "Martin server with eaws tiles." https://osm.cg.tuwien.ac.at/vector_tiles/eaws-regions, 2025. Accessed: 2025-07-01.

[7] EAWS, "Pmtiles with eaws regions." https://regions.avalanches.org/, 2025. Accessed: 2025-07-01.