Automatically identifying surface prediction errors

William Stevens

27th December 2025

Automatically identifying sheet switching in scroll surface predictions

This report describes a method for detecting scroll surface prediction errors that cause sheet switching. The method has been applied to a large fraction of scroll 4 and the results are currently being evaluated so that the algorithm can be refined. The goal of this work is to be able to take as input a surface prediction zarr that contains errors that lead to sheet switching, and output a zarr that contains no such errors.

Introduction

Current methods for virtually unwrapping scans of the Herculaneum scrolls begin by inferring scroll surfae positions using machine learning models. The current Kaggle competition aims to make these models as good as possible. Currently, the outputs from such models usually include errors that cause problems when trying to obtain a flat scroll surface from a surfce predictions. The types of errors that can occur include:

1. Merging of two surfaces.
2. Splitting and rejoining of a single surface.
3. Incorrect joining of neighbouring windings.
4. Surface dropout

The first 3 of these are illustrated in figure 1.

The least problematic of these is surface dropout : so long as it isn't extensive it simply results in holes in otherwise fully connected surfaces.

The other three types of error are more difficult to handle. If it's possible to detect these types of error, then deleting areas that cause these problems (i.e. turning them into 'surface dropout' errors) might be easier than trying to workaround them or correct them. Ultimately it might be possible to turn a surface prediction zarr containing errors into a zarr with all of the error regions removed, at the expense of an increased number of surface dropouts.

Figure 1: Three kinds of surface prediction error

Detecting surface prediction errors

The method described here is implemented specifically in the pipeline described in previous reports [1]. I hope that the principles are general enough that they will translate into other pipelines. Even if not transferrable, then the goal of this work (producing a surface pre-

diction zarr in which the only errors are surface dropouts) should be usable by others.

The pipeline involves growing small flat octagonal patches (ranging in size from $1.6mm^2$ to $40mm^2$) and working out how to align them so as to form a 2D surface. Alignment of a pair of patches is based on finding pairs of 3D volumes coordinates common to each patch, and using these to calculate a 2D affine transformation that will rotate, translate, and possibly flip one patch to match the other. The readme.md file in the repo linked to in the Source Code section of this report describes how to run this pipeline.

Figure 2 illustrates a problematic surface prediction from scroll 4, and figure 3 shows two of the patches that the pipeline has grown in this region. You can see that the two patches have an area in common, but diverge in other places. When these patches are aligned and rendered in flat 2D space, the result is that some 2D points map to more than one 3D volume point, separated by quite a large distance. This is illustrated graphically in figure 4 (produced by `find_mismatch.cpp`). The black octagons represent patches, and the grey to white shading shows the distance between one patch and another.

By randomly selecting pairs of overlapping patches, running the analysis described in the previous paragraph, and detecting when the 3D distance between points that occupy the same point in 2D space exceeds a distance threshold, it is possible to build up a count of how often each patch occurs in a pair that causes sheet switching. This has been done for 94142 patches, covering a large fraction of scroll 4, and resulted in 1153 problem patches when run with a distance threshold of 50.

Figure 5 shows some of the patches that were identified as problem patches in a slice at z=4096 in scroll 4.

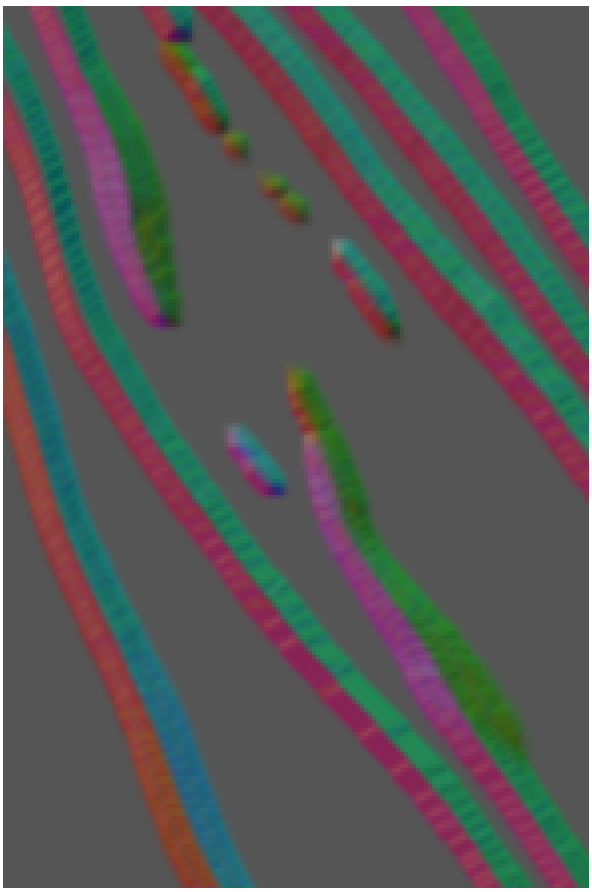

Figure 2: A problematic surface prediction

False positives and false negatives

False positive 'problem patches' are possible, because patches in the same sheet can potentially be misaligned in-plane to a large enough extent that two 3D volume points reside on the same 2D point. In future this type of false positive can be detected algorithmically and rejected as a 'problem patch'. One such false positive has been identified so far.

False negatives (identified manually) were also found among the 94142 patches produced. Figure 6 shows an area at z=4096 where an error remains even after removing the 1153 problem patches.

By inspecting cases such as this, it was found

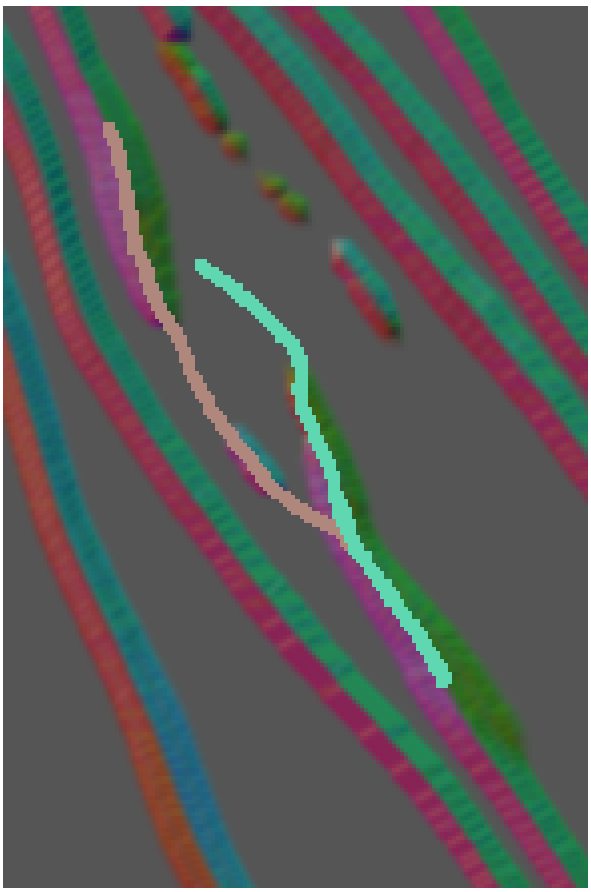

Figure 3: Patches grown in an area of problematic surface prediction

that the reason they weren't found is that the distance threshold was set too high - e.g. the blue and lilac patches in figure 6 don't diverge by more than 41 units, so the distance threshold of 50 is not exceeded. Rerunning the algorithm again and reducing the distance threshold will likely detect most of the remaining problem patches. Reducing the distance threshold is likely to result in more false positives too, so false positive identification must be implemented before doing this.

Future work

Rather then selecting overlapping patches at random, it will be possible to iterate through all pairs of overlapping patches. After doing

Figure 4: Grey to white shading shows increasing 3D volume distances between points in two patches that partially overlap and share some points in the 2D plane

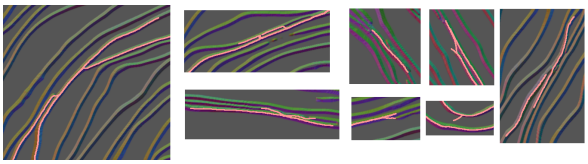

Figure 5: Examples of some of the problem patches found in scroll 4 at z=4096

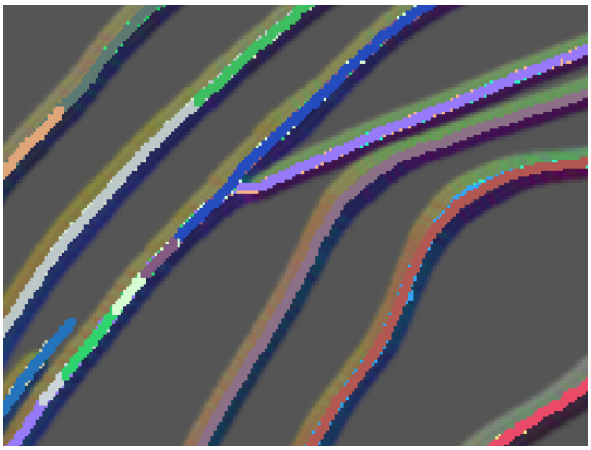

Figure 6: An error that was not detected by the first run of this algorthm on scroll 4

this, I want to use the results of this algorithm to output a zarr containing the remaining 'good' patches ANDed with the original surface prediction (or perhaps the original surface prediction ANDed with all patches, then XORed with 'bad' patches). I hope that this will result in a surface prediction with fewer sheet switches, at the expense of a larger number of surface dropout errors.

I believe that most problems can be detected using pairs of overlapping patches, but it is possible that not all problems can be detected using pairs, and it might be necessary to use longer sequences of patches to detect some problems. The code already allows for this. I don't beleive it will be practical to iterate over all sequences once the patch sequence length exceeds 2, so a compromise of iterating over all individual patches, and starting a few random patch sequences from each patch will probably be used.

There is value in having a pipeline that uses small patches with a large amount of overlap - errors can be localised more easily if patches are small, and removing a single patch is fairly harmless, because overlapping neighbours will often cover some of the same area. I have found through experiments that I can remove at least 10 percent of patches and overlaps among the remaining patches will still result in a more-or-less fully connected graph.

Source code

The source code for this pipeline has been put into a single standalone folder here: `https://github.com/WillStevens/scrollreading/pipeline6`. This is a self-contained folder that contains all of the code needed to run the version of the pipeline used for this report (including code to produce the vector field zarr). This folder will be tagged at the time this report is produced because work is likely to continue on this version of the pipeline.

The only C/C++ depedencies are blosc2 and libtiff-dev.

The readme.md file in this folder contains a description of the programs that implement the algorithm described in this report.

References

[1] `https://github.com/WillStevens/scrollreading/blob/main/report.pdf` to `https://github.com/WillStevens/scrollreading/blob/main/report9.pdf`