MatterhornFinder

Corentin Dumery

July 1, 2026

Problem statement.

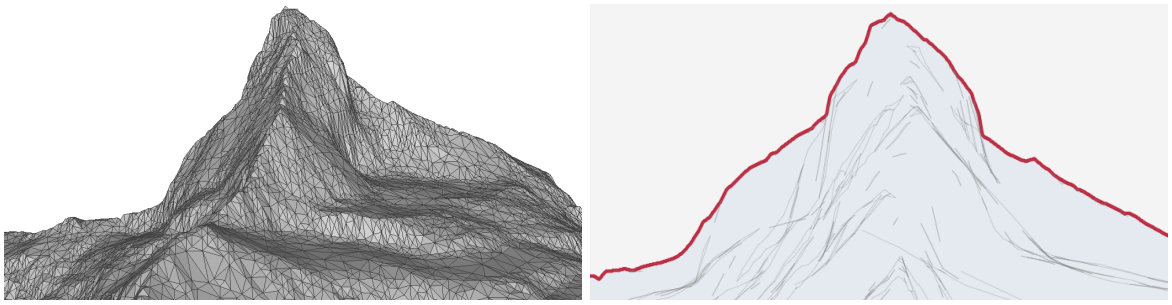
In this problem, we will design a computer-vision solution to a uniquely Swiss problem: identifying whether a given mountain on the horizon is the Matterhorn or not. To this end, the user will hold their phone in landscape mode and approximately level with the horizon. If the Matterhorn is found, its outline will be highlighted and a message will appear. If not, nothing will happen. In particular, we do not attempt to identify other mountains.

To achieve this, we will generate two outlines: the mesh outline on one end, computed from the known geometry of the mountain and the known viewpoint, and the image outline on the other end. Intuitively, if these two outlines are similar, then we have found Matterhorn.

Deriving an outline from the 3D mesh.

We will use a **3D surface mesh** of the mountain, similar to the surfaces seen in class during this semester. Specifically, a triangle mesh is denoted (V, F), where $V \in \mathbb{R}^{N_v \times 3}$ is a matrix of vertex positions and the i-th row is the 3D coordinates of the i-th vertex. $F \in \mathbb{N}^{N_f \times 3}$ is a list of triangles, such that the i-th row contains the indices of 3 vertices in V that are connected as a triangle. In the following, it may also be useful to consider $E \in \mathbb{N}^{N_e \times 2}$, the set of edges in the mesh which can be inferred from F.

Our goal is now to compute a **mesh outline**, which is a 2D line in image space representing the horizon as seen from (x, y, z). This mesh outline is an estimation of what the mesh (V, F) would look like as seen by a camera with projection matrix P. An example outline generated from the 3D mesh is visualized below.



To obtain such a mesh, a helicopter circles the Matterhorn and takes 50 photographs from known positions and orientations. We initially considered applying **shape from stereo** to these images, but the lack of texture of the snow-covered peak made this task challenging. Instead, we manually segment a binary foreground mask of the visible Matterhorn in each image and would like to use these segmentations to perform 3D reconstruction.

Question 1: *(2 points)* Which algorithm from the course can use this **segmented data** to reconstruct a 3D shape (1pt)? Explain it in two sentences (1pt).

Answer: We use shape from silhouette/contour (1pt), in which each silhouette back-projects to a cone in 3D from the corresponding camera (1pt). The mesh is recovered as the intersection of all such cones.

Question 2: *(2 points)* What is the usual limitation of this method (1pt)? Does this impact the horizon outline (1pt)? The horizon outline is the border between the sky and the mountains in the image.

Answer: Shape from silhouette reconstructs the visual hull, the largest volume consistent with all silhouettes. Adding more views can improve the visual hull, but it cannot recover concavities that are not visible in the silhouette (1pt). Thankfully, this does not matter in our problem setting since the outline does not depend on interior concavities but only on the silhouette (1pt).

Let us now assume that we have a satisfactory mesh (V, F). Using GPS localization and user altitude, we get the Matterhorn position (X, Y, Z) and user position (x_u, y_u, z_u). From these coordinates we derive a projection matrix P of the picture taken by the user looking directly towards the mountain. P is computed from the camera intrinsics K and the look-at direction $(X - x_u, Y - y_u, Z - z_u)$. For a mesh vertex $\tilde{v} = (v_x, v_y, v_z, 1)^T$, the camera projection gives $P\tilde{v} = (u, v, w)^T$, and the pixel coordinates are $(i, j) = (u/w, v/w)$. In other words, P can be used to project 3D positions to 2D pixel coordinates.

Question 3: *(2 points)* Using the mesh edges E, design an algorithm that computes the outline of Matterhorn as could theoretically be seen by the user (2pts). Use expressions from the previous paragraph.

Answer: Project all the edges in E from 3D to 2D with P (1pt). Mark the corresponding pixels red. Keep only the edges at the top in image space (1pt): for each column in the image, keep only the top red pixel and unmark all pixels under it.
Alternative answer: project only edges where one adjacent triangle is front-facing and the other is back-facing (i.e. sign changes in the dot product between the face normal and the view direction).

Image outline

In this part, we compute an outline from the user's photo, representing the horizon in the image. Some examples are displayed below, with the output outline in blue.



To detect edges, we use a simplified version of the Canny edge detector. As seen in the lectures and labs, the pipeline has 4 steps which we will implement as follows:

```
def image_to_outline(img, sigma, ksize, tau):
    smoothed = gaussian_smoothing(img, sigma, ksize)
    M, theta = sobel_gradients(smoothed)
    M_thin   = non_max_suppression(M, theta)
    e        = M_thin > tau
    return edges_to_outline(e)
```

Let us assume an array `arr` representing an image of shape $H \times W$. In your implementation, you can freely use any of the following functions:

Helper functions from the lab	
`applyImageFilter(arr, F)`	Convolves image `arr` with filter `F`, returning an array of the same shape as `arr`.
`gaussian_filter(fSize, fSigma)`	Returns a two-dimensional Gaussian kernel of size `fSize`×`fSize` with standard deviation `fSigma`, normalised to sum to 1.
`numpy` functions	
`np.sqrt(arr)`	Element-wise square root.
`np.any(arr, axis=0)`	Reduces along rows, returning a one-dimensional boolean array of length W. The value at column c is `True` when that column contains at least one `True`.
`np.argmax(arr, axis=0)`	Reduces along rows, returning, for each column, the row index of the first maximum. Applied to a boolean array, this gives the row index of the first `True`.
`np.where(cond, a, b)`	Element-wise conditional: returns `a` where `cond` is `True`, and `b` otherwise.

Instructions:

- Write your answer **in the dedicated space**, not in the question itself, and do not include the lines which do not change (e.g. `def ...`).
- Exact syntax is not required and deviations won't be penalized as long as the operation is correct.
- For full credit, do not use explicit Python loops, which are poorly parallelized.

Question 4: *(2 points)* Fill the following function to implement Step 1:

```
def gaussian_smoothing(img, sigma, ksize):
    """
    img   : numpy array of shape (H, W)
    sigma : standard deviation of the Gaussian
    ksize : kernel side length
    Returns the smoothed image.
    """
    F = ...
    return ...
```

Answer:

```
F = gaussian_filter(ksize, sigma) (1pt)
return applyImageFilter(img, F) (1pt)
```

Question 5: *(5 points)* Implement Step 2. The Sobel filters can be obtained by combining a one-dimensional derivative filter and a one-dimensional smoothing filter: $d = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$, $s = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$. The x-derivative kernel differentiates horizontally and smooths vertically, while the y-derivative kernel differentiates vertically and smooths horizontally.

```
def sobel_gradients(img):
    """
    img : np.ndarray of shape (H, W).

    Returns the gradient magnitude M of shape (H, W)
```

```
    and gradient angle theta in [0, pi) of shape (H, W).
    """
    Sx = ...
    Sy = ...
    gx = ...
    gy = ...
    M  = ...
    theta = np.arctan2(gy, gx) % np.pi    # given: angle in [0, pi)
    return M, theta
```

Answer:

```
Sx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) (1pt)
Sy = np.array([[-1, -2, -1], [ 0,  0,  0], [ 1,  2,  1]]) (1pt)
gx = applyImageFilter(img, Sx) (1pt)
gy = applyImageFilter(img, Sy) (1pt)
M  = np.sqrt(gx**2 + gy**2) (1pt)
```

Normalizing S_x and S_y is acceptable but not necessary here.

We assume that the function `non_max_suppression(M, theta)` is provided. After non-maximum suppression, the main pipeline thresholds the thinned magnitude using $e = M_{thin} > \tau$.

Question 6: *(6 points)* Implement the helper function `edges_to_outline`. The outline is computed by identifying every pixel column that has at least one edge pixel, then keeping only the topmost edge in each such column. Rows are indexed from top to bottom, so the topmost edge corresponds to the smallest row index.

```
def edges_to_outline(e):
    """
    e : np.ndarray of shape (H, W), boolean (can only take values in {0, 1}).

    Returns
    -------
    outline : np.ndarray of shape (W,), dtype int.
        outline[c] = row of the topmost edge pixel in column c,
        or -1 if column c contains no edge pixel.
    """
    has_edge = ...
    top_edge = ...
    return ...
```

Answer:

```
has_edge = np.any(e, axis=0) (2pt)
top_edge = np.argmax(e, axis=0) (2pt)
return np.where(has_edge, top_edge, -1) (2pt)
```

Outline matching

We now have two one-dimensional outlines:

- the **mesh outline**, computed by projecting the 3D Matterhorn mesh,
- the **image outline**, computed from the user's photo.

Let $m \in \mathbb{Z}^W$ be the mesh outline and $o \in \mathbb{Z}^W$ be the image outline. For both outlines, the i-th value is the row index of the horizon at the i-th column in the corresponding image, or -1 if no outline

point is available in that column.

Question 7: *(4 points)* Assume first that the two outlines are already perfectly aligned in the image. Propose a simple distance D between m and o (2pts), and a criterion that determines whether the picture is in fact of Matterhorn using a threshold distance D_t (2pts).

Answer: Be mindful that we compare only columns where both outlines are defined $\Omega = \{c \mid m_c \neq -1 \text{ and } o_c \neq -1\}$, and that if Ω is empty the distance is considered infinite. Then the mean absolute vertical error is (2pts):

$$D(m, o) = \frac{1}{|\Omega|} \sum_{c \in \Omega} |m_c - o_c|.$$

Our criteria will therefore be that the distance is below a threshold $D(m, o) < D_t$ and there are enough columns that are comparable $|\Omega| > \frac{W}{10}$ (2pts).
Note: An L2 distance could work too, but L1 is preferred here because the image outlines tend to have a lot of outliers.

Question 8: *(2 points)* In practice, how would you set the value of D_t? Assume you have access to a set of images that do feature Matterhorn, and a set of images that doesn't but has other mountains instead.

Answer: We set D_t to maximize the accuracy over this set of images. This can be done in a few different ways which do not involve repetitive trial and error or attempting to train a neural network. For example, computing the mean distance between Matterhorn images and the outline D_+, and D_- for non-Matterhorn images, we can simply set $D = \frac{D_+ + D_-}{2}$. Alternatively, we can plot the histogram of distances and set a threshold that separates the two sets.

Question 9: *(4 points)* To make the matching more robust to small errors in camera orientation, we would like to expand our search to also consider small horizontal and vertical shifts of the mesh outline. We consider potential shifts of the outline in image space, where a shift is represented as $(\Delta h, \Delta v)$. How can we use these candidate shifts in our previous method (2pts)? Which candidate shift should be kept (2pts)?

Answer: For each candidate shift $(\Delta h, \Delta v)$, we compare the image outline o with the shifted mesh outline $S(m, \Delta h, \Delta v)$ and compute a matching score (2pts). We then keep the shift with the smallest distance (2pts):

$$D_{\min} = \min_{\Delta h, \Delta v} D(o, S(m, \Delta h, \Delta v))$$

We can do the same with small variations in scale and rotation, too. This is reminiscent of the Hough transform seen in class.

Question 10: *(2 points)* Give one practical reason why the matching may produce *false negatives* (Matterhorn is on the picture but not detected, 1pt) and one for *false positives* (Matterhorn is not on the picture but detected regardless, 1pt).

Answer: The image outline may be incorrect because of clouds, increasing the outline distance and leading to a false negative(1pt). Another mountain or cloud may have a similar outline to the Matterhorn from a particular viewpoint and produce a false positive, but this is very unlikely (1pt).

Question 11: *(4 points)* Now, we drop the assumption that we have access to the user's GPS coordinates. What key input from our previous method do we lose (2pts)? What can we do instead (2pts)?

Answer: We lose the assumption that the **projection matrix P** is known (2pts). A possible solution is to generate a set of candidate mesh outlines, instead of a single one, then match the image outline against all these candidate mesh outlines. If any of them has a low distance, we have found Matterhorn, and also an estimate of the projection matrix P (2pts).