MounTune:

Musifying Mountain Photography

Final Project Report

Amos Haviv Hason & Stas Rodov

Introduction to Computational and Biological Vision

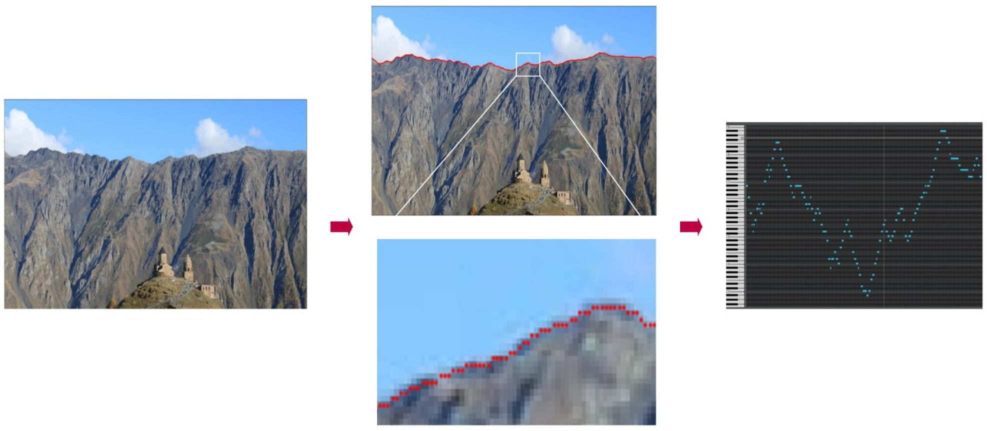
Ben-Gurion University of the Negev

March 2023

Introduction

We are mountain appreciators who like to travel around the world and take photos of mountainous landscapes. We also appreciate music. Once in Amos's travels, he thought to himself: "How would these mountains sound like as music?" Thus he came up with the idea of musifying mountain photography, and this is our attempt at implementing this idea.

In general, the system should identify the outline of the mountain's top edges, turn it into a sequence of *(x,y)* points when *y*-values are the top edge's heights (left to right), and finally turn the sequence into a coherent and non-dissonant musical tune with respect to some musical scale and some "musical boundaries", in MIDI format.



Approach

Leveraging an existing data source to enhance musicians’ composition capabilities.

Our data source: integer sequence of mountain top edges’ y-positions in the photo over the x-axis (left-to-right).

Use tools from computer vision to extract this integer sequence out of photographs.

Turn this integer sequence to a MIDI file by compressing it and fitting it to a desired musical scale and using convolution with 1D Sobel filter for generating note durations.

The output MIDI file can then be used for music production.

Method

- **From Photos:**
    - Pre-processing (“smoothed sharpening”)
    - Foreground mask extraction using GrabCut + smoothing the mask
    - Contour approximation + filtering redundant points
    - Fixing bad results with user input:
        - Input from user (currently without graphical user interface)
        - Creating new mask from GrabCut result combined with user’s input
        - Using new mask to initialize another run of GrabCut
- **To Music:**
    - Musification algorithm

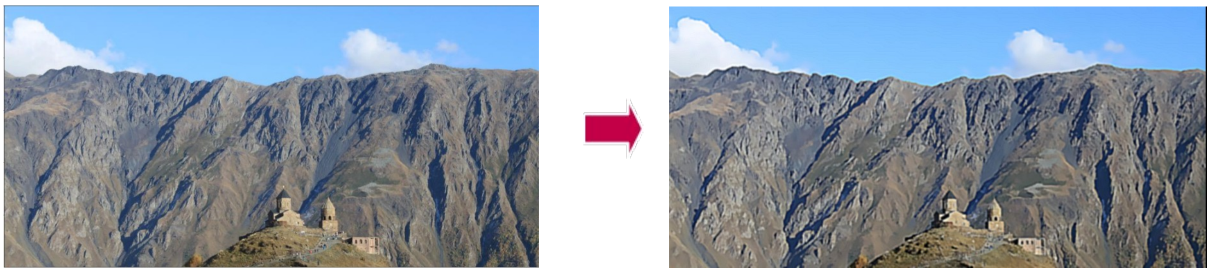
<u>**From Photos...**</u>

Pre-processing ("smoothed sharpening")

Below steps are greatly improving the <u>automatic</u> results of GrabCut tool (will be discussed further).

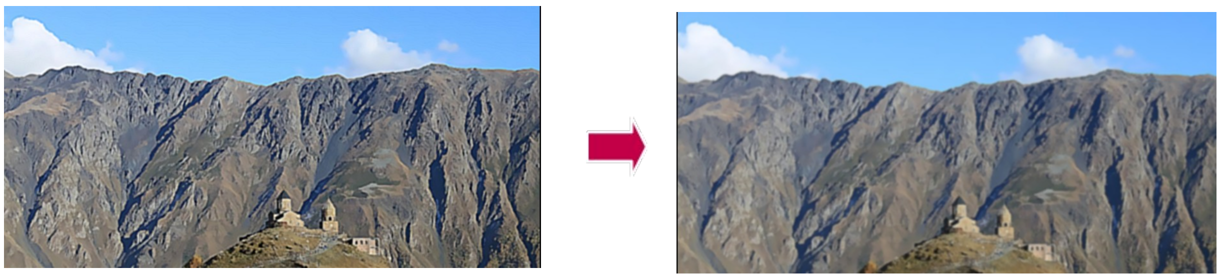
Bilateral Filter:

Basically, a Gaussian blurring that keeps edges sharp (using a second Gaussian model for pixel intensity):



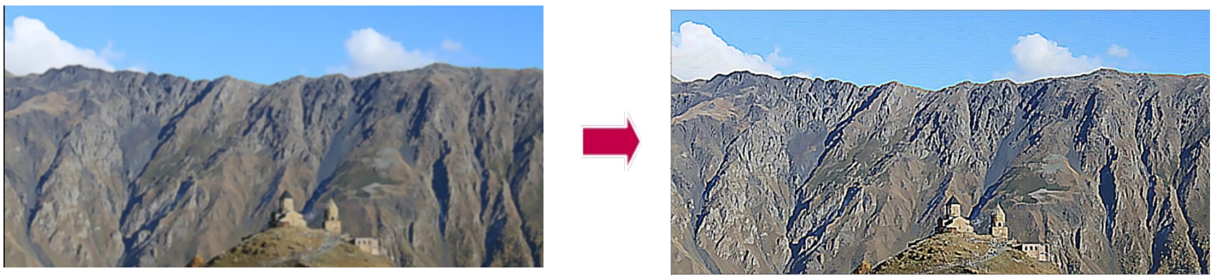
Convolutions with sharpening kernel + Gaussian blur kernel 11x11:

The result has thick and distinct mountain outline, without much noise:



Convolutions with two sharpening kernels + Gaussian blur kernel 5x5:

Final result has thick and very distinct mountain outline, while other parts are relatively not too sharp and there is relatively not too much noise:



Remarks:

We experimented a lot of combinations of different kernels and kernel sizes, on a set of about 35 photos.
We tried to achieve good automatic results of GrabCut on as wide as possible range of different settings and resolutions.

The sharpening kernel we used eventually is: *[[-1,-1,-1], [-1,9,-1], [-1,-1,-1]]*,
while this sharpening kernel: *[[0,-1,0], [-1,5,-1], [0,-1,0]]* was less useful for improving our results.

We believe that we can explore further and maybe come up with modified pre-processing steps that will improve the results even more.

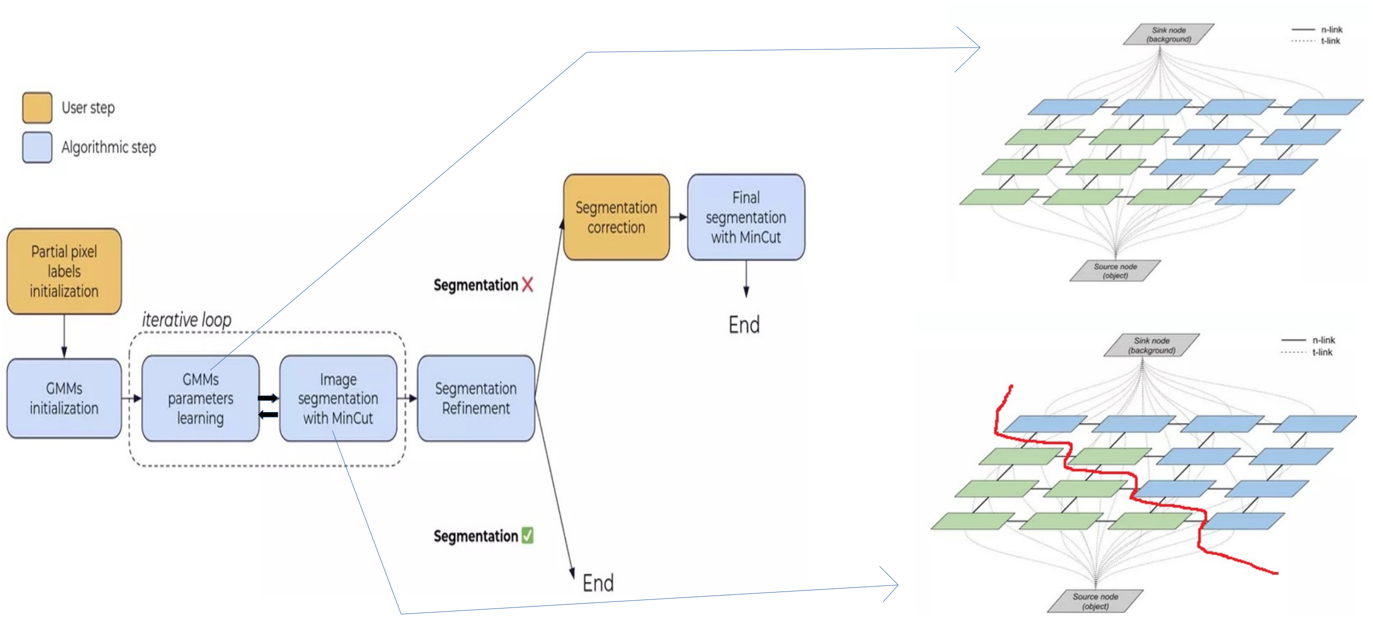
Eventually, after experiments are over, all the convolutions can be implemented using a single convolution with kernel that consists of convolving all kernels together, as we saw in class.

Foreground mask extraction using GrabCut

GrabCut is an iterative algorithm for foreground-bacground segmentation (basic idea is somewhat similar to the normalized graph cut learned in class), based on finding minimum cut in a graph and on the assumption that the colors of foreground (the mountain) and background are distributed like a mixture of Gaussians, which makes sense in our case, as things in nature are considered to have a Gaussian distribution.

The algorithm maintains two Gaussian Mixture Models (GMM) for foreground and background colors, and at each iterative step:

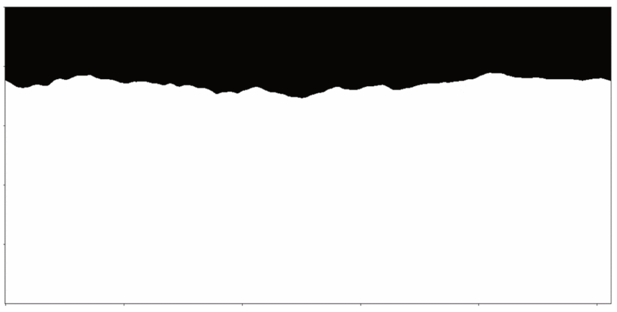
- Weights of the graph edges are set according to classification of pixels <u>as induced by current GMMs's states</u>
- *Minimum graph cut* is computed
- Parameters of the GMM models are refined by the classification of pixels <u>as induced by minimum cut result</u>



The init-rectangle for GrabCut, that represent the general area of the foreground, was chosen to be the whole image instead of receiving it from the user, so <u>user's responsibility is to use an image that captures well the area of interest.</u>

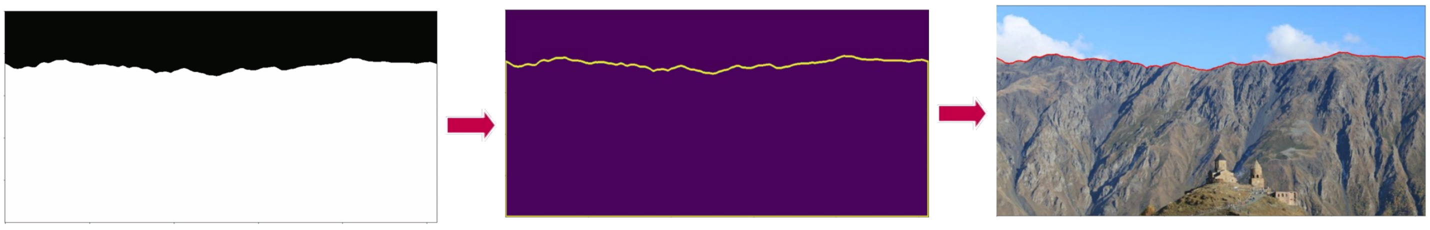
Below is GrabCut result after several iterations, before and after Gaussian smoothing.
The smoothed mask generates more nicely sound music (original mask is preserved for fixing step, see further).

The white area in below binary mask represent all pixels in the pre-processed image that are considered to be sampled from the foreground color distribution, as learned by GrabCut:

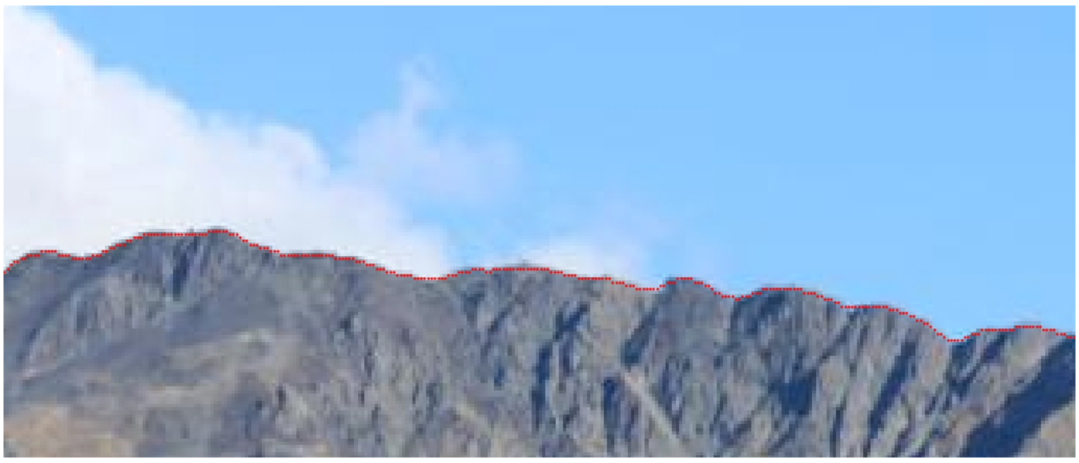


Contour approximation

Using open CV’s findContours().
After fetching the contour, we filter the contour points, leaving only those that reside on the mountain upper outline, thus leaving maximum *y* for each *x*:



The red contour consists of points that will be treated as the sequence for the “musification” process:

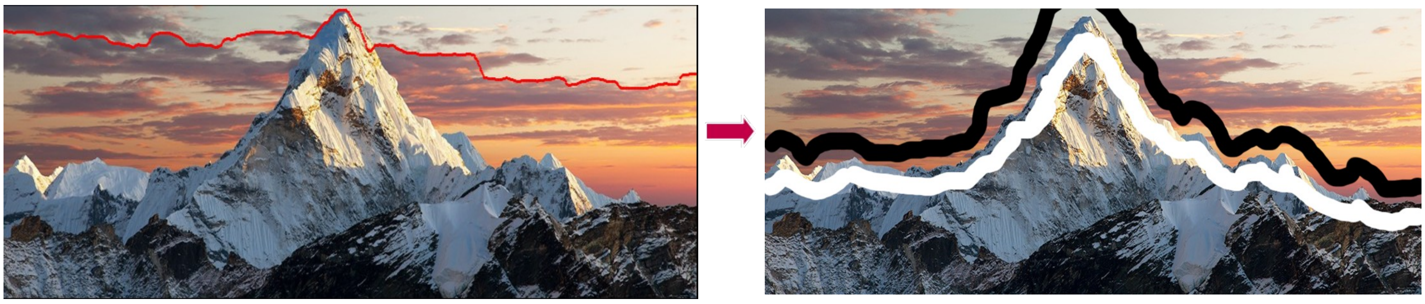


Fixing bad results with user input

GrabCut has a second way for initiating the GMM models: using a custom mask instead of rectangle.

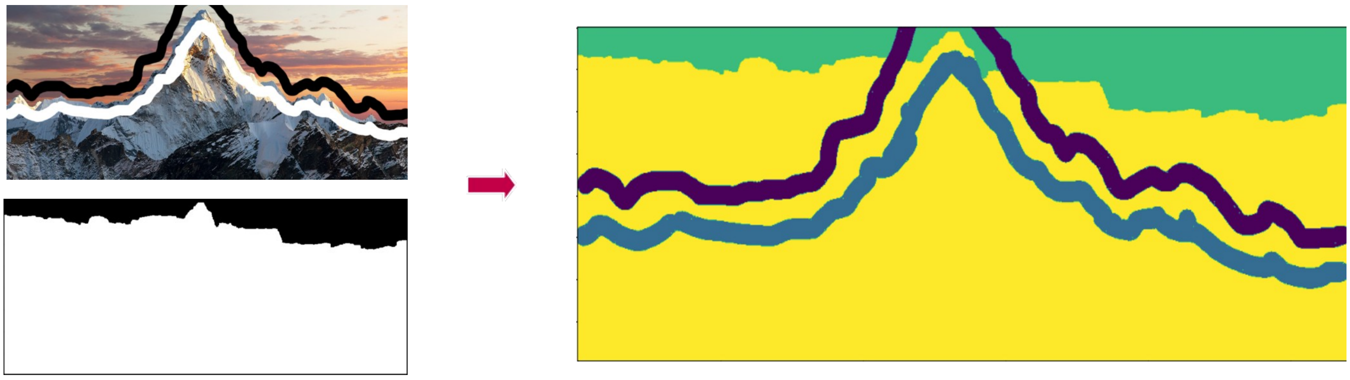
Input from user (currently without graphical user interface):

Below is a bad result, followed by user input on top of it: black curve represent ground-truth background, white curve represent ground-truth foreground.
Such user interface is not yet implemented, image was manually created using GIMP.



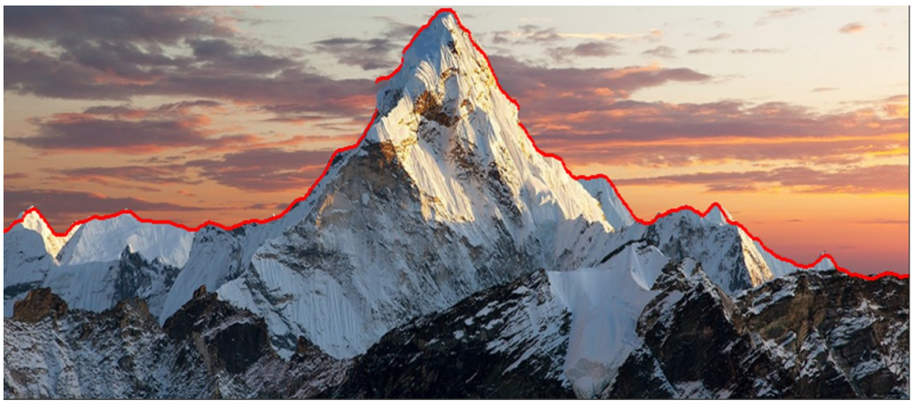
Creating new mask from GrabCut result combined with user input:

- Green: probable background (from GrabCut result)
- Yellow: probable foreground (from GrabCut result)
- Purple: ground-truth background (from user input)
- Blue: ground-truth foreground (from user input)



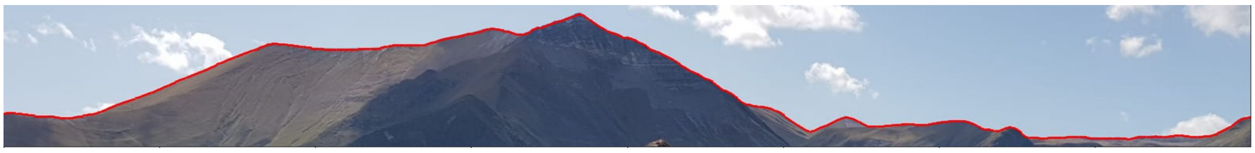
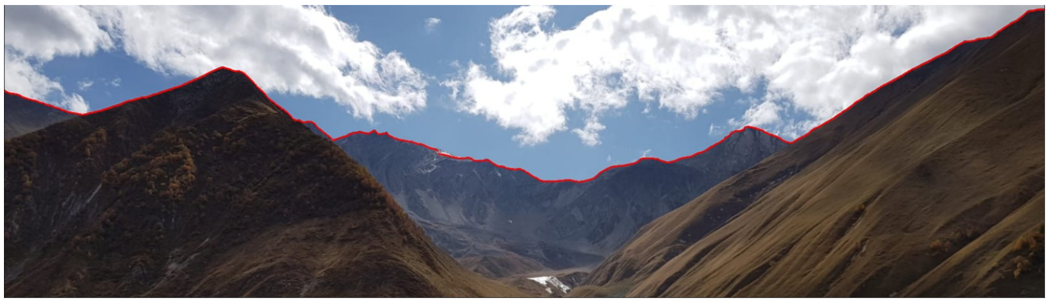
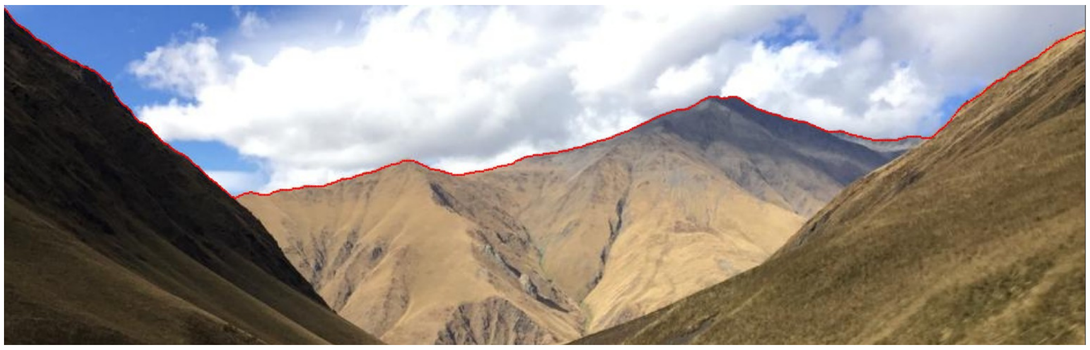
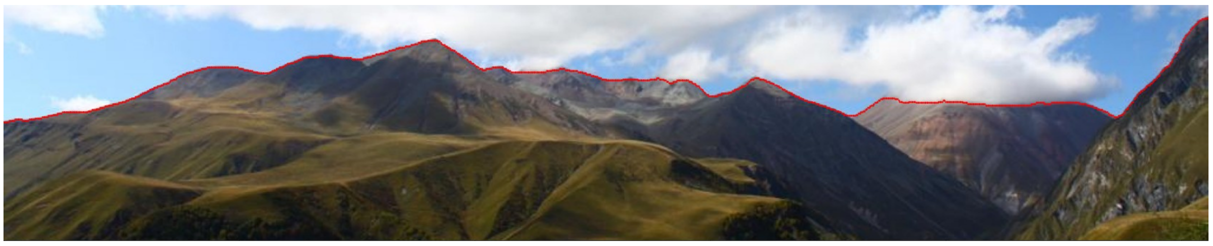
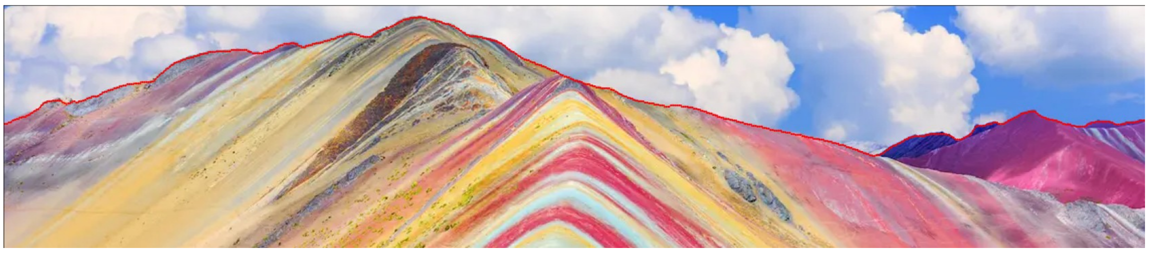
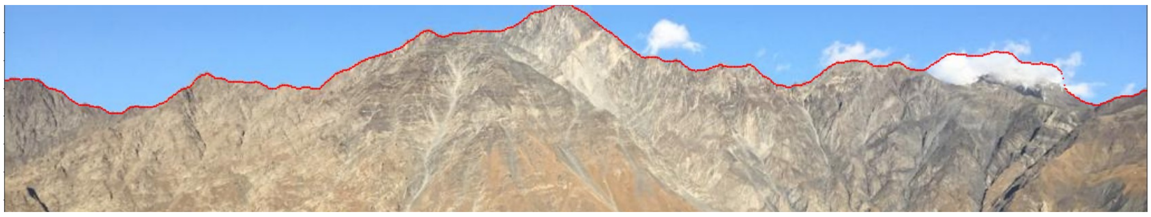
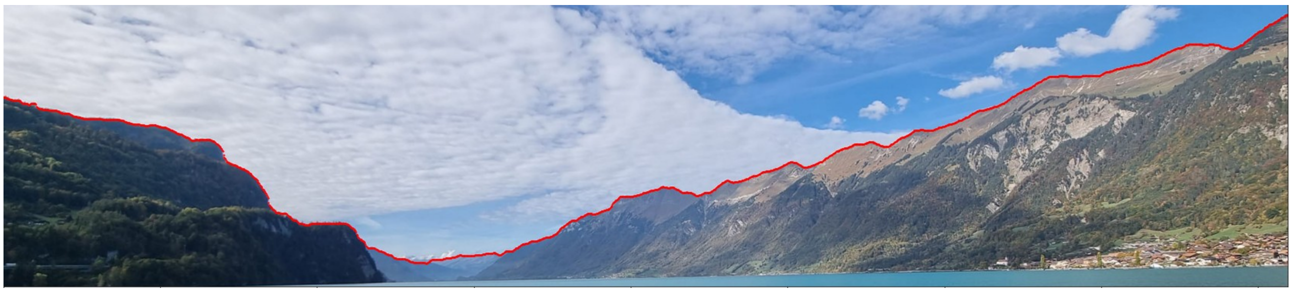
Using new mask to initialize subsequent run of GrabCut:

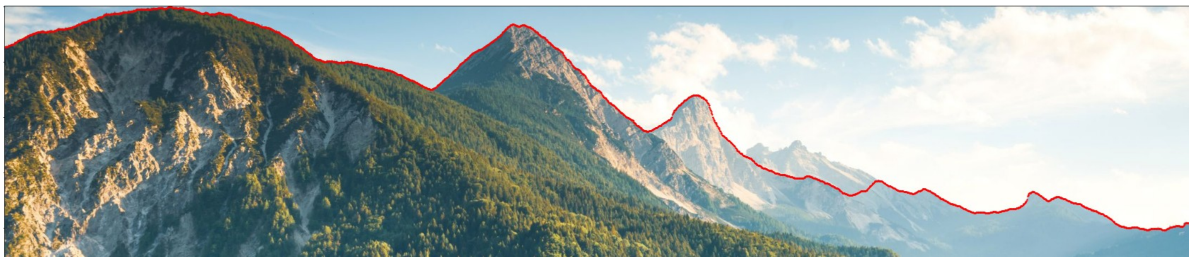
Fixed result:



Results of mountain top edge outline detection

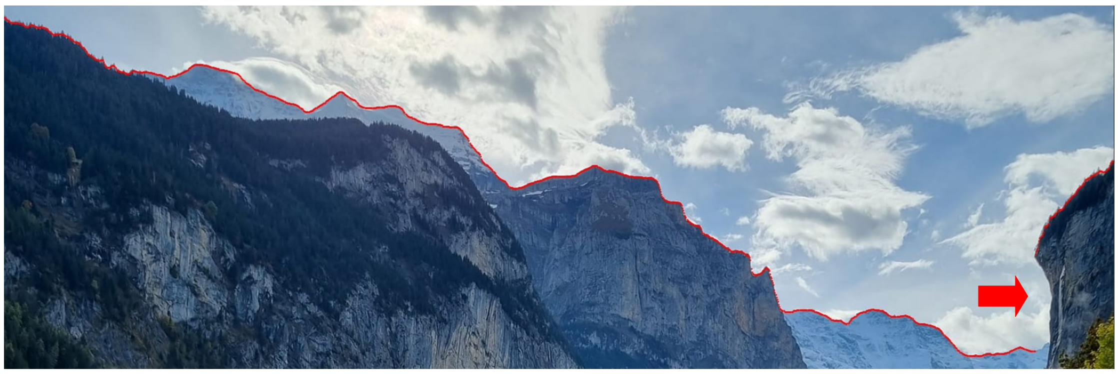
Automatic results (without fixing with user input):

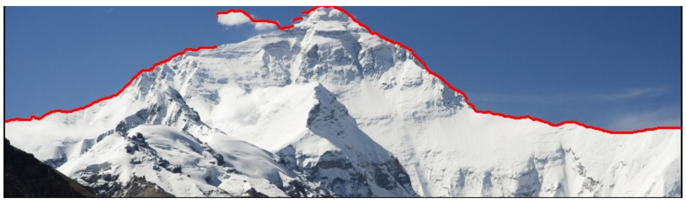


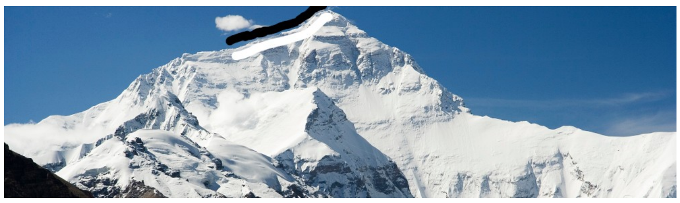
Example of very low resolution image:

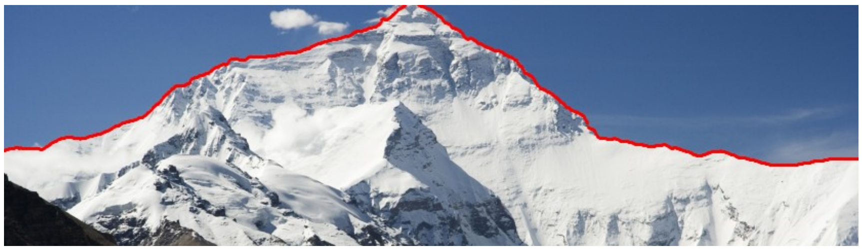
Example of mountain terrain with reversed direction of *x* values of the outline (see “Ideas for future improvements”):

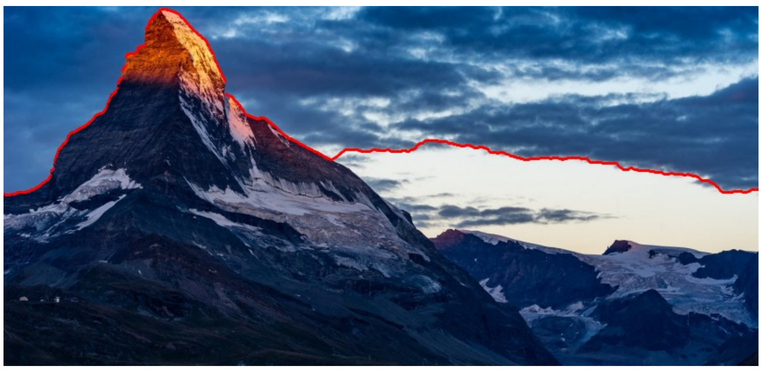


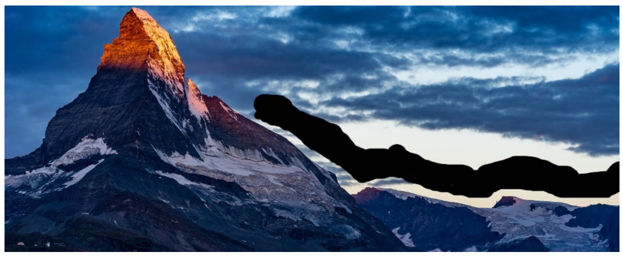
Examples of results fixed with user input:

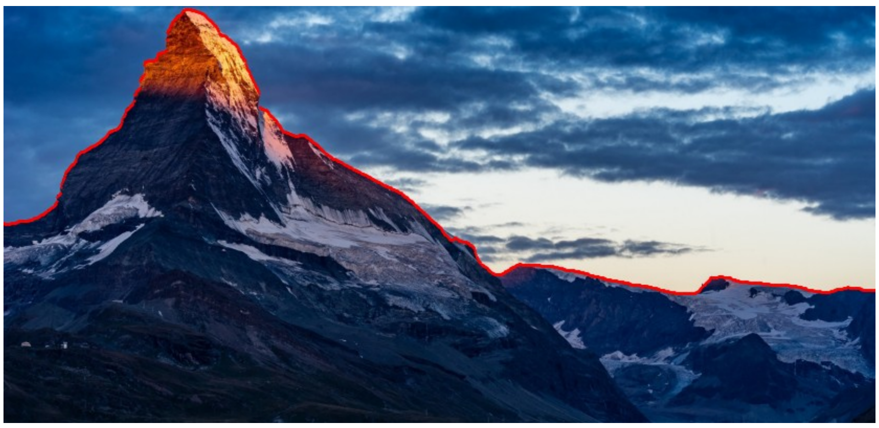












… To Music

For prior knowledge in music theory and MIDI please refer to Appendix A below.

Musification algorithm

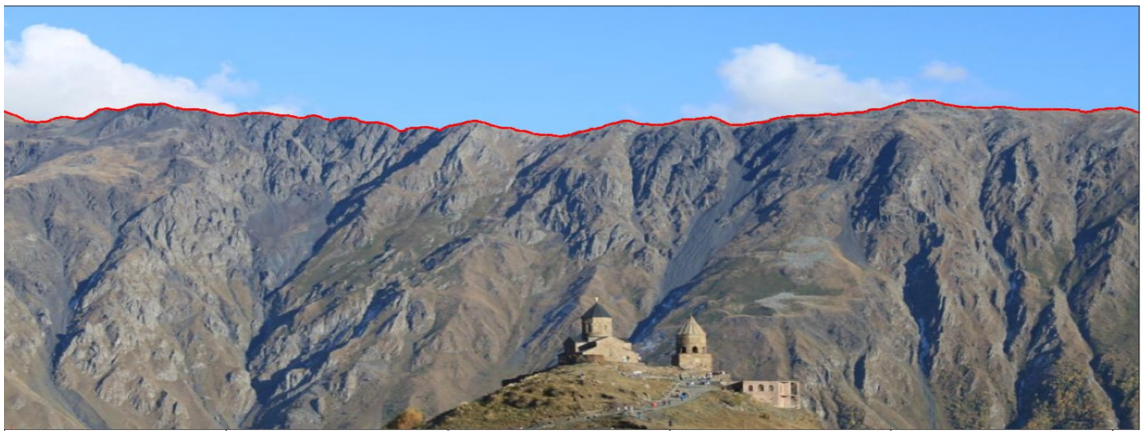
- **(Rounding to integers is used throughout this algorithm.)**
- Choose some musical scale from a list and a starting musical note.
- Create a list of relevant notes according to scale are saved, ordered from lowest pitch to highest.
- Convert mountain top edge points to list of relative heights with minimal value 0 ordered by x value.
    - **Lower height = lower value = lower pitch**
- Create note durations using convolution with 1D Sobel filter with minimal threshold 0.
    - Higher (absolute) delta = longer duration
    - **Dramatic changes in altitude are emphasized in the melody.**
- Iteratively compress note durations until maximal note duration doesn't exceed some value.
- Add 1 to all note durations.
    - Prototype design choice: enforce no moments of silence.
- Iteratively compress mountain top edge heights until maximal height doesn't exceed the index of last item in the list of relevant notes.
    - We'll reference the mountain top edge heights as "notes as integers" from now on.
- Average notes as integers and durations over windows of some size.
    - Every few consecutive notes are condensed to one note by averaging their pitches and durations.
    - Lowers the probability of consecutive note duplicates or one-step moves (for melodic variety).
    - Effectively shortens the duration of the tune.
- Optionally, remove sequential duplicate notes by averaging their durations.
    - Effectively shortens the duration of the tune.
- Convert notes as integers to (note, octave) tuples using the list of relevant notes.
    - Each integer in the sequence maps to an index of a note in the list of relevant notes, and thus to a specific note in a specific octave.
- Convert the tuples to MIDI numbers.
- Write to MIDI file using the MIDI numbers and durations.

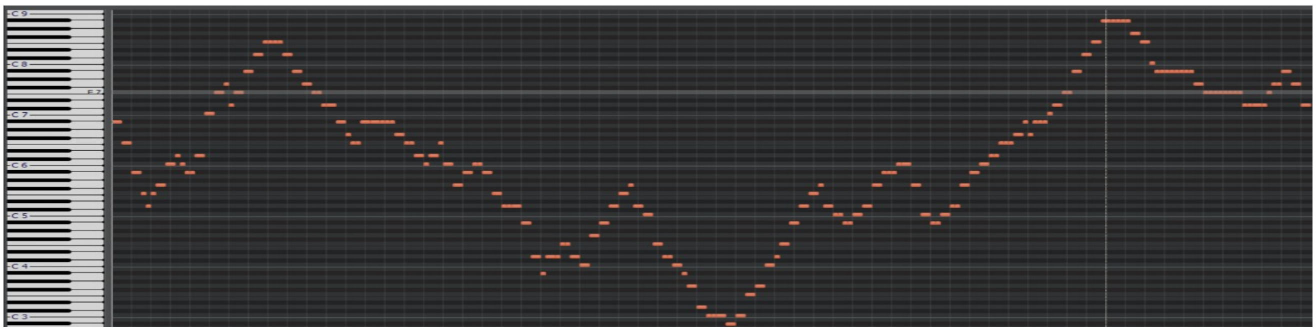
Results of generating music out of detected outline

The mountain outline is resulting in below MIDI sequence, we can see that the highs and lows of the mountain top edges correspond to the highs and lows of the MIDI sequence.

Reasons for visual differences between the mountain top outline and the MIDI sequence:

- The MIDI sequence as shown in the screenshot from music editor was visually compressed to fit the screen.
- Musification algorithm related reasons:
  - Compression of the integer sequence range.
  - Fitting to a specific musical scale and to a musical range of an instrument (a piano, in this case).





For examples of MIDI files, please refer to the project web page.

Conclusions

In conclusion, it seems that we were able to implement a basic, but sufficiently well-working prototype of our idea.

The algorithm was able to sufficiently detect the outline of mountains from photos we tested, most of them required very little if any user intervention (beside selecting the part of the photo that captures the area of interest).

The MIDI musical tunes that are being generated sound reasonably OK, they are not random or dissonant noise but rather express some “musicality”, and can serve as raw data for music production process.

Current state of the solution is not perfect, we believe that the automatic results of outline detection can be improved, as stated earlier, and also there is a huge amount of work that can be done to improve and enhance the capabilities of our music generation algorithm, which is a project of itself and outside of the scope of computer vision.

Ideas for future improvements

Graphical user interface

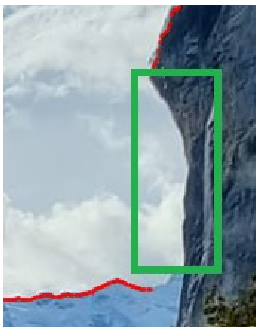
A user interface can include an option for the user to draw ground-truth information on top of the image to fix it, as discussed earlier.

The interface can also include multiple menus to control different parameters of music generation, e.g. selecting the scale, selecting whether or not to include sequential duplicates (as discussed earlier), etc.

Also, we can implement a music player with a time-slider that goes over the mountain top edges in the photo accordingly.

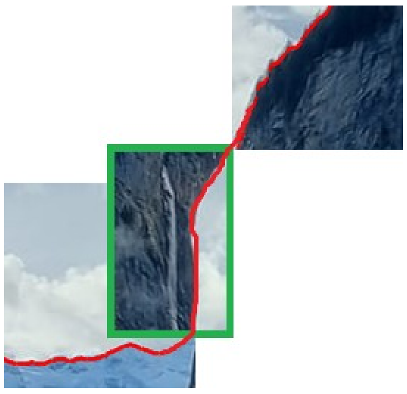
Addressing mountain top edges reversing direction in the x-axis

As shown earlier, there are examples of "backwards going" mountain outlines:



This is not a standard case of mountain photographs, but it occurs from time to time.
In such cases, the generated sequence of points doesn't include the "backwards going" terrain, as we consider only maximum *y* value for every *x*, as explained earlier.

An idea to also utilize the going-backward part of the outline (green rectangle in above image) for the sequence of points, is to convert existing outline to one such that the "backwards" part is turned "forward", see below example (the red line indicates the contour that will be converted to sequence of points):



Silent beats

Including silent beats, i.e. no notes being played, in the tune for rhythmic variety (using either deterministic or random criteria).

References

https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html (Image Filtering)
https://en.wikipedia.org/wiki/Kernel_(image_processing) (Kernel)
https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html (Smoothing Images)

https://en.wikipedia.org/wiki/GrabCut (GrabCut)
https://www.microsoft.com/en-us/research/publication/grabcut-interactive-foreground-extraction-using-iterated-graph-cuts/ (GrabCut original paper)
https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html (Interactive Foreground Extraction using GrabCut Algorithm)
https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html (Miscellaneous Image Transformations)
https://en.wikipedia.org/wiki/Mixture_model (Mixture model)
https://www.sicara.fr/blog-technique/grabcut-for-automatic-image-segmentation-opencv-tutorial (GrabCut for Automatic Image Segmentation)

https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html (Contours : Getting Started)
https://docs.opencv.org/4.x/d9/d8b/tutorial_py_contours_hierarchy.html (Contours Hierarchy)
https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html (Structural Analysis and Shape Descriptors)

https://pypi.org/project/musical-scales/ (A Python package with a list of musical scales)

Appendix A: Music Basics

Western Music Theory Basics

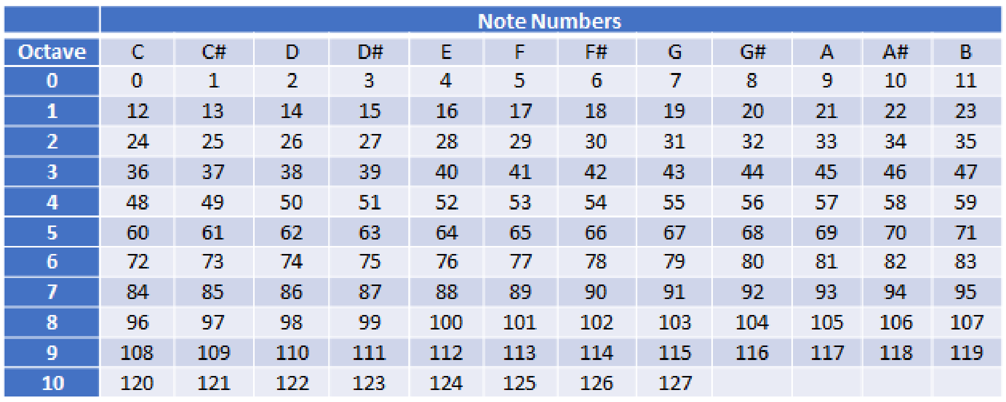
- An **octave** consists of 12 **semitones**.
- The **pitch** (sound wave frequency; physically measured in Hz) difference between each two consecutive semitones is equal on a logarithmic scale, e.g. log((k + 2)th semitone) - log((k + 1)th semitone) = log((k + 1)th semitone) - log(kth semitone).
    - Human perception of musical intervals is approximately logarithmic (sensitivity to change in pitch decreases approximately logarithmic as pitch gets higher).
- Each semitone sounds "the same" shifted up or down by an octave.
    - Same in terms of "color" of the sound.
    - Different in terms of "intensity" of the sound.
- Every musical instrument has a **range** (all pitches it can play).
- A **musical note** represents one of the 12 semitones:
- A, A#, B, C, C#, D, D#, E, F, F#, G, G#
- A **musical scale** consists of a subset of the 12 semitones, with one of those being the starting note. Examples:
    - C major: **C**, D, E, F, G, A, B
        - White keys of the piano.
    - F# minor **penta**tonic: **F#**, A, B, C#, E

MIDI Basics

MIDI is a widely used standard for representing time series of musical note events in digital format.

MIDI file time steps are according to tempo (beats per minute). Notes are represented by a number (see table below).

MIDI files serve as skeletons in music production. Professional music production software allow importing MIDI files and enable sound design on top of them.



	Note Numbers											
Octave	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119
10	120	121	122	123	124	125	126	127				

MIDI note numbers (source: MathWorks Audio Toolbox Documentation)