EECS 4422 Computer Vision, Winter 2022
Assignment 1: Image Filtering
Due: 9am Februrary 28th 2022

All programming questions are to be completed in MATLAB. For each individual programming question, submit a MATLAB script. Include a file, call it `a1_script.m`, that will step through the answers to the entire assignment using `pause` between each question. Unless otherwise specified, all image processing steps are to be performed on greyscale images (if images are colour, use `rgb2gray()`). Finally, **do not share code or use code from the web or any other source**. We will be checking for copying using the MOSS system. Offenders (both copier and source) will receive a zero on this assignment and we may pursue academic sanctions. **This assignment is to be done individually.**

1 Tilt Photography (Total 20)

1. **[20 points]** Tilt-shift photography is the use of camera movements that change the orientation or position of the lens with respect to the camera's image capture sensor. Tilting the lens with respect to the image capture sensor creates a miniaturization effect. In this question, you will simulate this miniaturization effect via filtering a given image, call this function `myTiltImage`.

   (a) Define a horizontal focus line by selecting an image row (use `ginput()`). The horizontal line is meant to define points that are at the same distance out in the world and in perfect focus. Of course without depth information, this horizontal line definition is just a heuristic.

   (b) Define your depth of field with respect to the focus line. This image region is defined as several pixels above and below the focus line (do not blur this region).

   (c) Gaussian blur the region outside of the depth of field. The amount of blur should increase as you move away in the perpendicular direction from the focus line; points in the output image on the same row should be the result of the same blur kernel. Here, you have a choice on how much to increase the filter blur progressively. You can apply a different blur amount across the rows or split the image region outside of the depth of field into horizontal bands where the filter blur is the same within each band. The other choice is how fast should the blur increase as you move away from the depth of field. These implementation choices are your's, evaluate different strategies and choose the one that yields the best qualitative result. For efficiency, your program should utilize the fact that convolving two Gaussians results in a Gaussian with a larger standard deviation, i.e., more blur.

   (d) Toys tend to be colourful. To increase the toy-like result of your tilt image, boost the image saturation. To boost the saturation, convert the image to the hue-saturation-value colour space (use `rgb2hsv()`), increase the saturation channel by a percentage, and convert the resulting image back to the RGB colour space (use `hsv2rgb()`).

   (e) Include a demo of your tilt photo code that displays the output from the `following image` and an image of your choice (be creative with your selection).

   (f) Implement a second version of your code where the focus line can be in general position, i.e., not limited to a row.

2 Canny edge detection (Total 45)

1. **[30 points]** Implement the Canny edge detection algorithm as a MATLAB function, call it `MyCanny`, as described in class and the handout available on the course webpage. Your function should take as input a greyscale image and the edge detection parameters and return the Canny edge binary image. For this question, there are two edge detection parameters, the standard deviation, σ, of the Gaussian smoothing filter and the gradient magnitude threshold, τ. Note, if you implement the peak/ridge detection step correctly the output of your program should NOT have "thick" edges!

   In addition to turning in your source code for the Canny edge detector, submit a MATLAB script that runs your edge detector on the test image provided at this link and on an image of your own choosing. Your Canny output images should use the best set of parameters you find for each input image.

   For obvious reasons, you are excluded from using MATLAB's `edge()` function in your own code but are encouraged to run the `edge()` function to get a sense of what the correct output should look like.

2. **[5 points]** Implement the Gaussian convolution as a sequence of horizontal and vertical convolutions, i.e., a separable filter.

3. **[10 points]** Add the hysteresis mechanism to the function you wrote for Q2.1. For hysteresis thresholding, a high and low threshold is specified by the user beforehand. The process begins by marking all pixels with gradient magnitudes above the high threshold as a "discovered" definite edge. These pixels are placed into a queue and become the starting points for a breadth first search (BFS) algorithm. Run the BFS by iterating through the queue of pixels. The hysteresis process terminates when the queue is empty. All adjacent pixels (one of the eight neighbours) are treated as connected nodes to the current pixel removed from the queue. The criteria for adding a new pixel to the queue is given by: an adjacent pixel that has not been previously discovered and has a gradient magnitude greater than the low threshold. Adjacent pixels that meet the criteria are subsequently added to the BFS queue. Every adjacent pixel is also marked as discovered once it is checked against the criteria.

3 Seam carving (Total 20 points)

1. **[20 points]** Seam carving is a procedure to resize images in a manner that preserves "important" image content. A video demo is available on YouTube. The general steps for seam carving are as follows:

   (a) Compute the energy image, E, for the input image, e.g., the sum of the gradient magnitude images computed for each of the three colour channels of the input image.

   (b) Create a scoring matrix, M, with spatial image dimensions matching those of the input image.

   (c) Set the values of the first row of the scoring matrix, M, to match those of the energy image, E.

   (d) Set the values of every entry in the scoring matrix to the energy value at that position and the minimum value in any of the neighbouring cells above it in the seam, i.e.,

   $$M(x,y) = E(x,y) + \min\Big(M(x-1,y-1), M(x,y-1), M(x+1,y-1)\Big), \qquad (1)$$

   where $M(x,y)$ is the cost of the lowest cost seam crossing through that point. This minimization procedure is an instance of *dynamic programming*.

   (e) Find the minimum value in the bottom row of the scoring matrix. The corresponding position of the minimal value is the bottom of the optimal seam.

   (f) Using $M(x,y)$, trace back up the seam by following the smallest value in any of the neighbouring positions above.

   (g) Remove the seam from the image.

   (h) To reach a desired resized image, you will have to repeat the above procedure. Note that you will have to recompute the energy matrix (and scoring matrix) each time to take into account changes in the resized image.

Your task is to write a MATLAB function, call it `MySeamCarving`, that takes in an image and the desired new resolution by removing the necessary horizontal and vertical seams and returns the resized image. Implement the seam carving routine with a *single* helper function, call it `CarvingHelper`, that removes all the necessary vertical (horizontal) seams. You first call the helper function to remove the vertical (horizontal) seams, transpose the result of this function and then call the seam removal function again with the transposed image as the input to remove the horizontal (vertical) seams. In addition to submitting your code, submit resizing outputs for the 1728×1151 `York` image to 1200×1151 and 1728×720. Also submit an image of your own and its resized output.

(a) In addition, implement image expansion by inserting seams, see the original paper for details.