Yiran Wu 914704719 Zuang Yu 915062968

I. Short Answer Problem

1.

Given separable filter f * g =

1	1	2
1	1	2
2	2	4

And the image h =

1	1	1
1	2	1
1	1	1

but if we separate the filter into filter g =

1	1	2

and filter f =

1
1
2

We only need 3 + 3 multiply operations for each pixel.

Where g * h = , and thus f * (g * h) = 4 + 5 + 8 = 17;

4
5
4

Also, (f * g) * h = (1 + 1 + 2) + (1 + 2 + 2) + (2 + 2 + 4) = 17

Thus, the cost would be 3*3 = 9 multiply operations for each pixel.
(f * g) * h = f * (g* h).
Thus, the associative property of convolution holds.
In other word, with (f * g) * h, we need k^ 2 operations, where (f * g) is a k x k filter.
With f *(g * h), we only need 2k operations.
By the associative property of convolution filter, (f * g) * h = f *(g * h). So we can separate the filter to make it more efficient.

2. The image would be converted to [1 1 1 1 1 1 1 1].

3. Because we assume the noise is sampled from a gaussian distribution and we are adding the noise per pixel independently. So if we assume an image with salt and pepper noise has gaussian noise and apply a smoothing filter, it won't do any help.

4. We assume the camera and the light condition is fixed, and each time the part would stop at the same location with the same size in the photos, and the noises of the images would not vary significantly, and there is no pepper and salt noise. Then we would also have an image in excellent condition. We would apply the following steps to each of the photos taken and also to the perfect photo to compare the differences.

A. We assume there is no pepper and salt noise but the gaussian noises. We would perform the gaussian filter to suppress the gaussian noises.
B. We would find the magnitude and orientation of the gradients.
C. We would then apply the Non-minimum suppression
D. Then we use a high threshold to find the important edges and then use low thresholds to connect them.
E. Then we find the chamfer distances of each photo comparing to the perfect photo and store them in a database.
F. We would update this "perfect product" with the median of the database as the time goes.
G. For the chamber distances we would set a threshold that the product can be divided into qualified products and unqualified products that need further examination.

II. Programming Problem

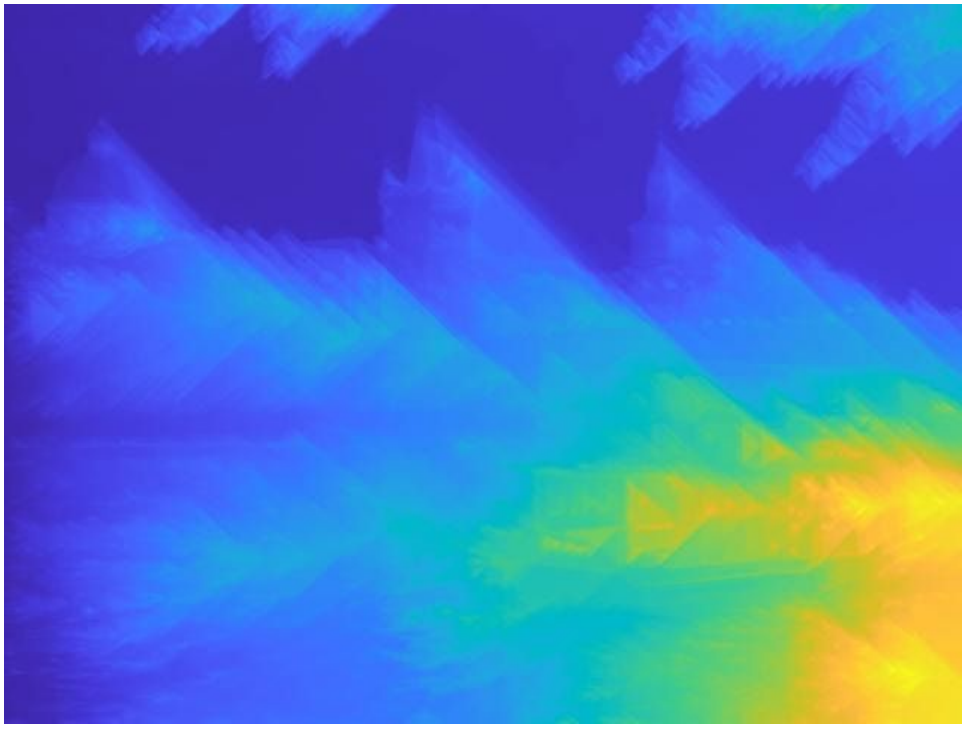
1.

2.

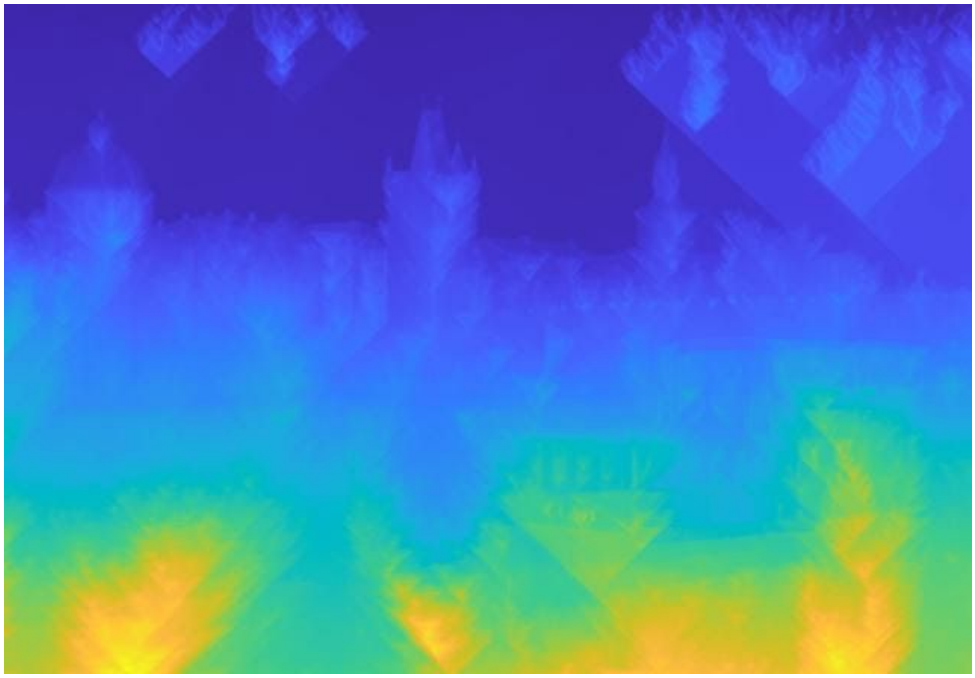
3.

The energy function calculates the energy of the picture, where we can see that the places with lower energy (such as the sky) are more blue.

Cumulative minimum energy maps for the seams in Horizontal Direction



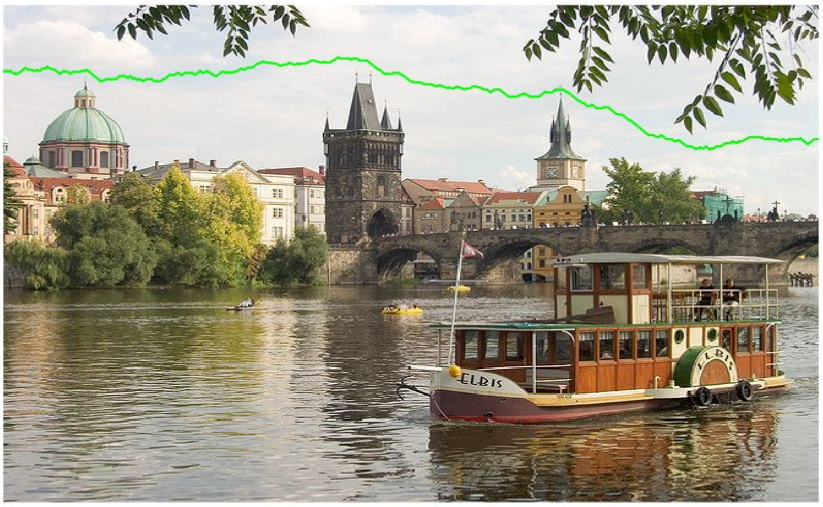
Cumulative minimum energy maps for the seams in Vertical Direction



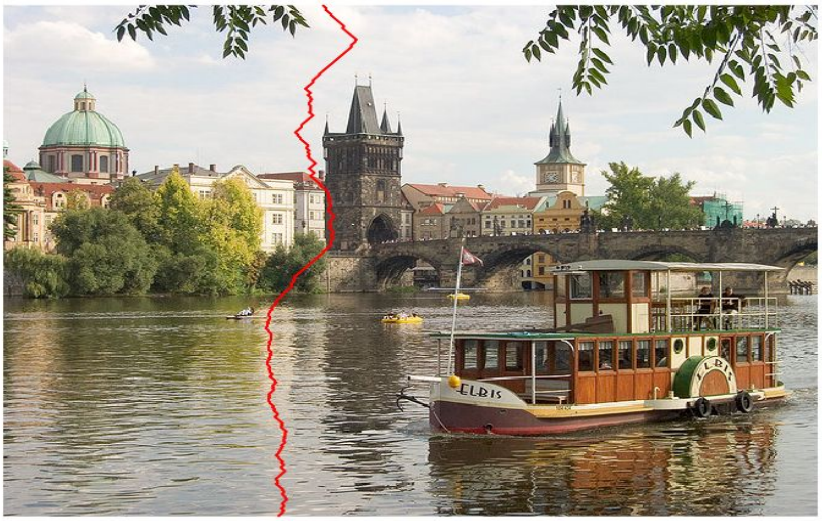
The horizontal and vertical cumulative energy map sums up the lowest energy use of the 8-connection method from each direction. The blue indicates that the energy is low, and yellow

indicates the energy is high. we can see from left to right for the cumulative energy map the color becomes more yellow because the energy cumulates up. Same for the vertical cumulative energy map from top to bottom. Also from the energy map we can see the upper half of image is darker, thus it would be more blue for the cumulative maps.

4. a. Horizontal seam



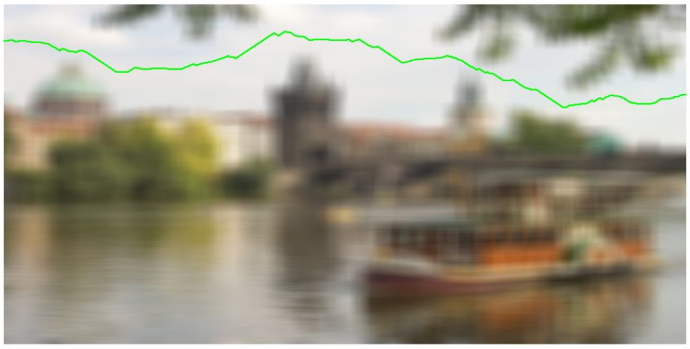
B. vertical seam

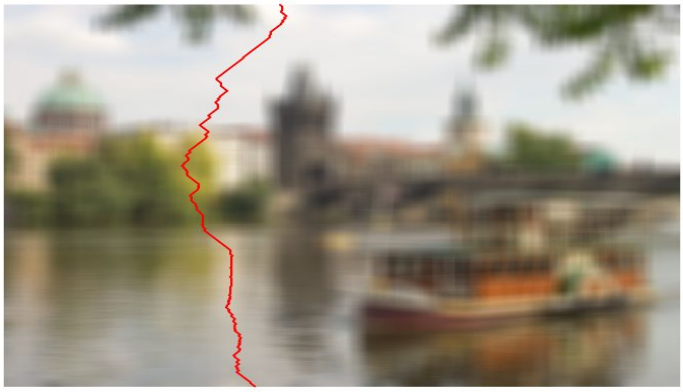


For the vertical seam, we first found the pixel with least energy in the last row of a cumulative min energy map. Then, find the pixel with least energy of above, above left, and above right pixels of the previous pixel. Then, iterating every row of pixels from bottom to the top in the cumulative min energy map to get an optimal vertical seam.

For the horizontal seam, we first found the pixel with least energy in the rightmost column of a cumulative min energy map. Then, find the pixel with least energy of left, left above, and left below pixels of the previous pixel. Then, iterating every column of pixels from rightmost to the rightmost in the cumulative min energy map to get an optimal horizontal seam.

5.





We used the average filter to blur the image and then find the seams. We can see the vertical seam moves obviously. Our reason is that the seam we found using the original seam might have the minimum energy, but the pixels around it might have big energy, and with the average filter, the original seam's energy increases significantly due to the pixels around it. So we would choose a different seam that has the minimum cumulative energy.

For the horizontal seam we can see that the sky is an area with low energy, so each with the average filter that area's energy still remains low, so the horizontal seam didn't change much.

6.

Image #1

(a) original image

(b) system resized image

(c) simple resampling image

(d) input dimension: 1080 X 1440 X 3

Output dimension: 780 X 1440 X 3

(e) For this image, we only remove height by 300 pixels.

(f) our system's resizing removes the clouds at the top of the image, which is not visually important. While the simple resizing image just simply shrinks the overall height of the original image, which makes the bridge narrower.

Image #2 ("**bad outcome**")

(a) original image

(b) system resized image

(c) simple resampling image

(d) input dimension: 1080 X 1080 X 3

Output dimension: 780 X 1080 X 3

(e) For this image, we only remove height by 300 pixels.

(f) We can see that the system resized image is cropped off and the dog is contorted, where the simple resizing would be better.

Image #3

(d) input dimension: 1080 X 1440 X 3
Output dimension: 280 X 740 X 3

(e) we removed the width by 500 pixels, and then we removed the height by 400px. Then, we removed the width again by 100 pixels, and removed height by 100 pixels. Then, we removed width by 200 pixels, and finally removed height by 200 pixels.

(f) PS: the image is resized due to subplot but in general the differences are shown clearly.
We want to figure out if the algorithm can be used to remove the margins when it comes to photo drawing. And it works out fine, where the simple resizing image looks bad.

Extra credit 3:
We had two functions enlarge_img_width(im, s) and enlarge _img_height(im, s) that enlarges the image, in “enlarge_img_width.m” and “enlarge _img_height.m”. The input “im” is the matrix representing the input image, while the input “s” is an integer that represents the number of pixels to be added to the input image.

we find the s number of different seams in the original image. Then, we duplicate the seam and insert them next to the original seams.

Below are results of our functions compared to the original image.

original image

enlarged width by 200 pixels

original image

enlarged height by 200 pixels

original image

enlarged width and height by 100 pixels