Aaron Lo
atl2374
11 September 2023

CS 376 Assignment 1

## I) Short Answer Problems:

1) What are three applications of image filtering
    a) One application of image filtering is noise reduction. Filters like Gaussian filters or Median filters can help smooth out random noise on the image to clean it up. This is useful in processing raw thermal camera footage or photography in general.
    b) Another application of image filtering is edge detection. Filters like the Canny edge detector can help identify edges in an image, which can be useful in object detection and image segmentation.
    c) Another application of image filtering is sharpening. Filters like the Laplacian filter help sharpen and enhance details, allowing for crisper images. This is useful in image editing contexts.
2) What is the difference between the mean filtering and the median filtering?
    a) Mean filtering takes the mean of the pixel values around a point while median filtering takes the mean of the pixel value around a point. This leads to mean filtering producing a blurring effect while being sensitive to extreme outliers of noise while median filtering generally manages to preserve edges and features of an edge.
3) In class, we talked about image smoothing followed by computing image gradients. Is it identical to computing image gradients first and then performing image smoothing on the resulting image gradients?
    a) The order in which these processes are carried out can make a big difference on the final product. If image smoothing is performed before computing image gradients, while it will have a great impact in removing noise, it will also reduce the detail granularity that the gradient can capture. On the other hand, calculating gradient before smoothing will lead to finer details being captured but noise can have a big impact.
4) How to take the advantage of the separability of a filter for fast image filtering calculation?
    a) As the filter is separable, it is decomposable to two smaller filters. As the number of calculations required for the convolution of a kernel on the image is proportional to the size of the kernel, these two smaller filters (generally 2D to 2 1D kernels) are much faster in terms of calculation time and efficiency.
5) In non-maximum suppression, we detect the maximum pixel along the image gradient direction. Provide examples where this approach is sub-optimal. You can draw

illustrations or provide results on real examples. Please provide a short justification (2-3 sentences) on why this is the case.

a) This approach can be sub-optimal in several cases. If there are edges that have low contrast with the background, the gradient might not be significant enough to determine from the surrounding pixels, making it hard to select the maximum pixel. Another issue is if there are multiple edges close to each other. It may be hard to select the maximum pixel along the image gradient direction as it might select one that doesn't represent any of the edges in this local neighborhood.

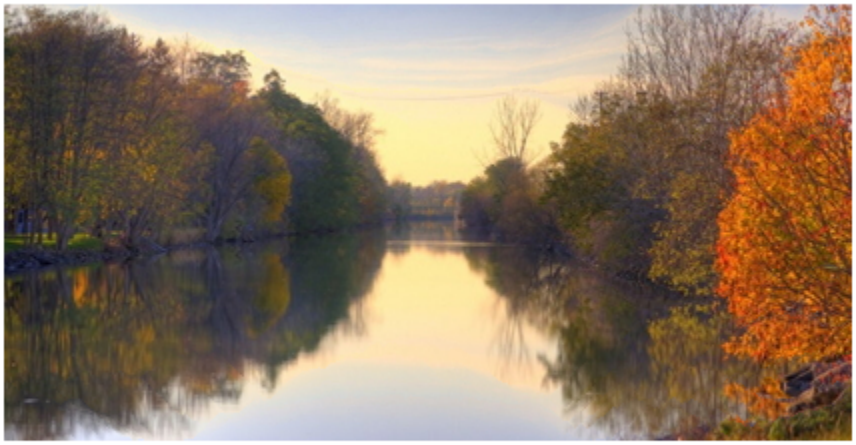
6) Bonus: So far we have covered filtering and edge detection for images. Please mention how to extend the idea to videos. Please discuss how to de-noise in both the spatial and/or temporal domain,how to compute gradients in the spatial and/or temporal domain, and how to detect ”edges” in the spatial and/or temporal domain.

a) In the spatial domain for de-noising, computing gradients, and detecting edges, the same techniques we used for images can be applied to videos as well, just for each and every single frame. On the temporal domain, we can denoise over frames, using perhaps frame averages or differences to reduce noise. For computing gradients in the domain, we have algorithms like Lucas-kanade to estimate temporal gradients.

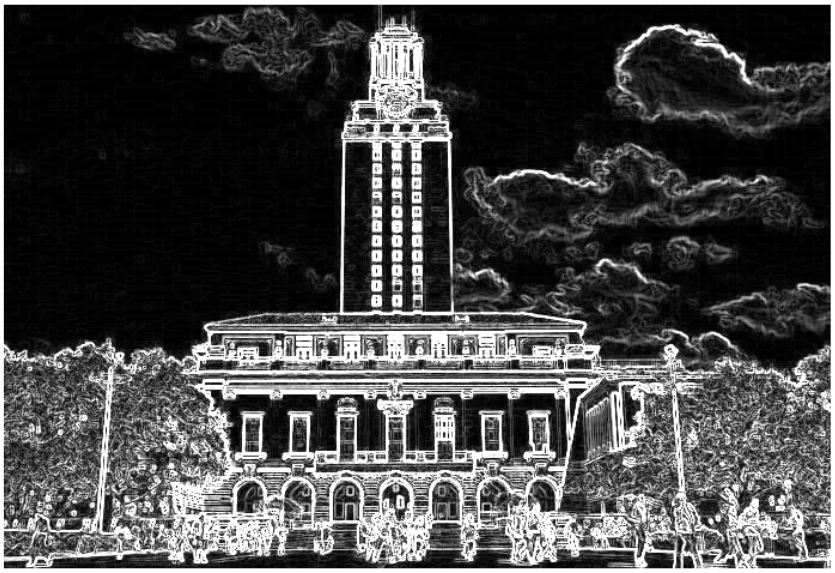
II) Programming Problem:

Removing Function

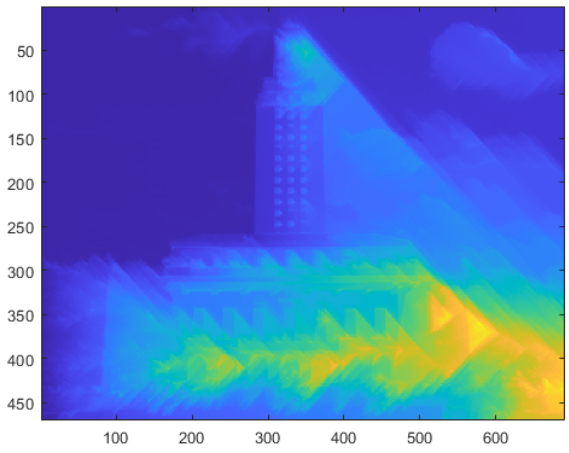
removeVertical of ut.jpg with numPixels = 100



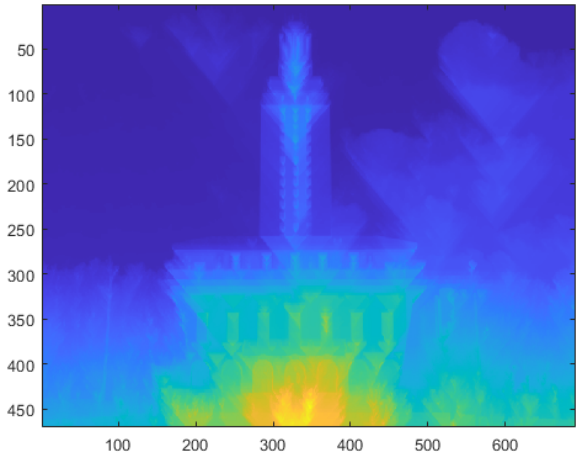
removeHorizontal of river.jpg with numPixels = 100



Total Gradient Magnitudes




Total Horizontal Cumulative Minimum Energy Maps




Total Vertical Cumulative Minimum Energy Maps

The energy function output for ut.jpg shows high values at locations of edges and changes. This makes sense as that iss what the total gradient magnitude is calculating for, the places of difference from its neighbors. As for the horizontal minimum energy map, we can see that at the bottom half contains high values as the sum of gradients there are much bigger at the bottom half. For the vertical cumulative minimum energy map, the middle contains the highest cumulative energy as the sharp color/shadow changes of the UT tower increases the values of the gradients in that area.

First selected horizontal seam. This seam was selected as it is the minimum cumulative energy across the horizontal space. As you look at it, the seam navigates around the clouds and the tower to stay in constant blue of the sky as much as possible.

First selected vertical seam. Again, the vertical seam in the top half of the image finds a route to avoid the clouds (which represent massive gradient changes). In the bottom half, the seam routes through the trees as their colors have mostly blended together at the distance the photo was taken, reducing total impact.

Figure 1: Normal -50 Horizontal

Figure 2: Gauss -50 Horizontal

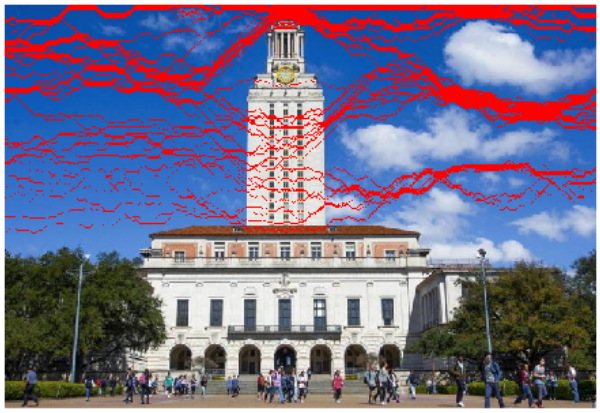


Figure 3: Removals of Figure 1

Figure 4: Removals of Figure 2

Figure 1 and Figure 3 show the normal removeHorizontal code at work for remove 50 pixels from the height. However, for Figure 2 and Figure 4, I applied a gaussian filter with a sigma of 8 before running the gradient magnitude filter. The results seems to have grouped up all the horizontal seams in 3 major paths compared to the many paths seen in Figure 3. This arises from the image more smoothed out, so the values of the gradients of the pixels next to each other are more similar, leading to if one of the “seams” being removed from an area, it leads to all the other ones being next up to be removed too, causing this river-like removal seams.

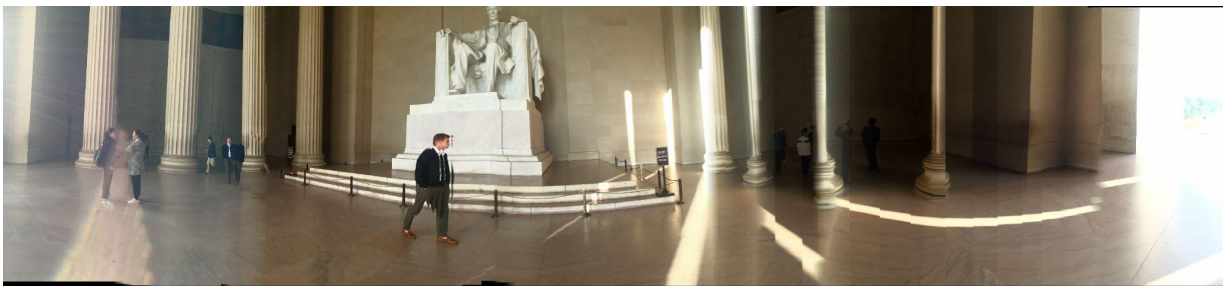
Default Image: 630 by 959

Sequence of Removal: -50 to Width

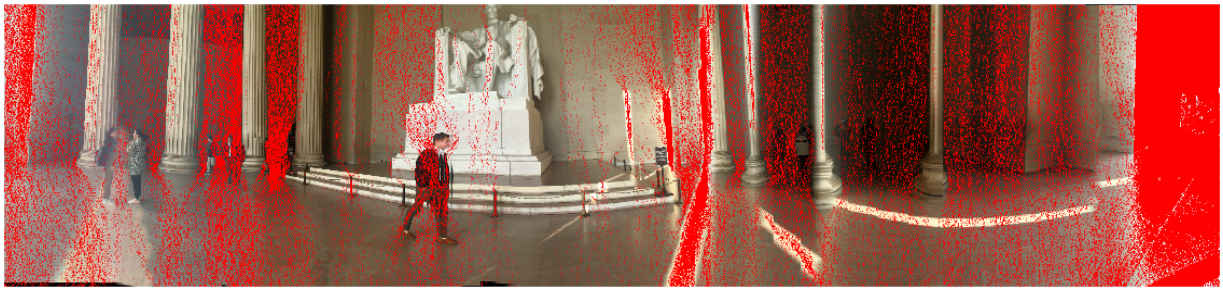
Imresize to 580 by 959

System resize to 580 by 959

In this output, the system attempts to remove 50 pixels from the image. However, I chose an image of the Washington Monument where the monument itself is very uniform in color. This leads it to have a very low gradient compared with even the sky, leading it to be removed. This is where seam carving falls short, removing our main subject even though we want it there. On the other hand, a simple resize kept all of our important elements.



Default Image: 12746 by 2936



Sequence of Removals: -3000 to Width

System resize to 9746 by 2936

Imresize to 9746 by 2936

I gave a photo of a badly taken panorama of the Lincoln memorial that is too big! Running the system resize, it first removes the over exposed parts of the image. Then, it removes the very bright and very dark parts of the image while keeping the pillars and people mostly intact. This results in the panorama being smaller but keeping most of the important details (ie the pillars, people, and the statue of Lincoln).

Default Image: 697 by 520

Sequence of Removal: -200 to Width

Imresize to 497 by 520

System resize to 497 by 520

In this output, I gave it a close up shot of flowers. I slightly cheated by using a photo I took in portrait mode, which blurred the background while leaving the flower unblurred. Because of that (and the separation of foreground and background), even by removing nearly a third of the image's width, the flower was mostly untouched by the system resize method. Meanwhile, the imresize function just squashed the image and made the flower look bad.