1. Introduction

Seam carving and seam insertion techniques were originally proposed by Avidan and Shamir in their report "Content Aware Seam Carving" (this report will be referred to as the source paper). In the subsequent sections of this paper, the results of seam carving, seam insertion and retargeting with optimal seams-order processes proposed by Avidan and Shamir (Figures 5, 7 and 8 from the source paper) are compared against a replicated output. All replication was done in Python with the use of numpy, OpenCV, and plotting libraries. Additionally, video was created to accompany the explanations provided in this paper. The link to the video can be found here.

2. Seam Carving

Avidan and Shamir outlined several steps to accomplish seam carving. These include, calculating/creating an energy image, calculating the minimum cumulative energy given the energy image, finding the seam which has the lowest energy and removing it from the source image, and finally, repeating this process until an image of the desired size is achieved. Implementation details when replicating each step in the seam carving algorithm described above will be outlined in the sections below.

2.1 Image Energy and Cumulative Minimum Energy

To calculate e_1 energy of the image given by Equation 1 in the source paper, Sobel filters with a kernel size of 3 were used in the vertical and horizontal directions of a gray scale version of the original image. The magnitude of the output of these filters were taken and added together to get the energy image. Taking the energy image, the minimum cumulative energy matrix was then calculated pixel by pixel, top down. Values were calculated by adding the minimum of the energies between the 3 closest pixels above the current pixel to the current pixel value. At the end of this process, the minimum value of the last row will represent the end of the minimal vertical seam. These processes can be seen in the functions image_energy and v_cum_min_energy of my code.

2.2 Finding a Seam and Removing One Seam

The process of finding a seam began by taking the minimum value of the last row in the cumulative energy matrix and storing its row and column indices in a seam vector, which represented the index of every pixel to be removed from a seam. Starting from this point, the minimum energy of the 3 closest pixels above the current pixel was found, and its row and column index was stored in the seam vector. Subsequently, the index of the current pixel was updated to the last indices stored in the seam vector. This process was then repeated for each row in the input image. The result was the complete seam where each pixel was connected to one of its 3 closest neighbors above it. Next, a new matrix was initialized to be the size of the input image reduced by 1 column. This matrix served as the output image after seam removal. The pixels from the input image, with exception to the ones included in the seam, were copied over to the output image. This entire process represents a utility for removing one seam from an input image and can be found in functions find_seam and remove_seam in my code.

2.3 Seam Carving Process, Output, and Comparison

The processes outlined in sections 2.1 and 2.2 for finding and removing a seam was repeated until an image of the desired size was achieved. To show the replication performance, the shore image from Figure 5 in the source paper was scaled down horizontally by 50%. The input and output images from the source paper (Figures 1 and 2 below), along with the output from my replication process (Figure 3) are shown below.

Figure 1. Input Image

Figure 2. Source Paper Output

Figure 3. Replicated Output

Comparing the output images visually, no differences can be found between the photos. Seeing that they both use the same energy functions and cumulative energy summation processes, it can be assumed that replication for the seam carving process was successful.

3. Seam Insertion

Avidan and Shamir explained the seam insertion process by inverting the seam removal process. Artificial seams with the least energy from the seam carving process were found, duplicated, averaged against their left and right neighbors from the input image, and added to the input image to create an expanded output image. Implementation details of the seam insertion process during replication will be discussed below.

3.1 Adding One Seam

Like the seam removal process, the seam adding process represents a utility function used in the seam insertion process where one seam is added to the input image. To accomplish this, the utility takes a seam and an input image and creates an output image 1 column larger than the input image. Having an input seam and its associated indices for insertion, the seam is averaged with its left and right neighbors from the input image and then inserted left of the input seam in the output image. This utility can be found in the add_seam function in my code.

3.2 Seam Insertion Process, Output, and Comparison

In order to do seam insertion, the seams removed from the seam carving process had to be recorded first. Each seam was then added to the image using the add_seam function mentioned above. Concurrently, after each added seam, the column indices of the remaining seams were compared against the added seam. If any of the seams overlapped with the newly added seam, the remaining seams were increased by 2. This was done to accommodate for adding 1 column in the image and to increase the seam index relative to the newly added seam. During the seam addition process, an optional flag of averaging was implemented. If averaging was turned on, then the seam to be added to the output image would first be averaged with its left and right neighbors in the input image before being added. Below are the outputs of a 2-fold 50% expansion from the original image given by the source paper and my code (seam_carving).

Original Image

Figures 4 & 5:
Source Paper Output

Figures 6 & 7:
Replication Output
No Averaging

Figures 8 & 9:
Replication Output
With Averaging

The images above show that no-averaging replicated the source paper results better than with averaging. When averaging was added, additonal ghostly seam artifacts from the averaged seams were included in the output images. Overall, the results were good, but clearly were not the same as the source paper. It can be seen that the replicated images were stretched under the water splash. This is likely a result of imporoper index tracking. We can also see that the dolphin is of slightly different size and the water splashes created by the dolphin are different.

4. Retargeting with Optimal Seams-Order

Optimal Seams-Order involves created a transport map (T-map) which evaluates the cost of removing a desired number of rows and columns in optimal order. A replication of the optimal seams-order path can be found below and in functions tmap and traverse_tmap of my code.

4.1 Transport Map (T-map) Generation, Optimal Seams-Path, and Output Comparison

The T-map was created row by row iterating over each cell in the row. A cache size was used to store the equivalent of one row of energies. Each cell in the represented the summation of removing all other rows and columns up to

that point in optimal order. So, when calculating the value of the current cell, vertical and horizontal seam removal had to be done from the left and upper neighbor and the minimum energy from that output was then added to the energy of the previous cell. The optimal seams-path was created by traversing the T-map from the last row and column in the desired output image size to cell (0, 0) on the map. The optimal path was found by taking the minimum of the current cell's left and top neighbor and storing a 1 in a bitmap for the direction chosen. As the map traversal was taking place; a bit map was concurrently being filled that showed the optimal removal path.

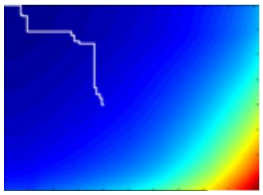

Figure 10. Source Paper T-map

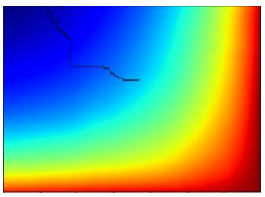

Figure 11. Replication T-map

When comparing the contours of the outputted T-maps in Figures 10 and 11, it was clear that the optimal seam removal paths and processes are different. This could be because e_1 energy in replication was used instead of e_{hog}. The replicated T-map had much higher energy density thought the map and particularly towards the edges of the map where most of the rows and/or columns would be removed. The optimal path taken for removal was also different, however we can generally see that the path taken was the optimal path in that it required the least energy moving to the desired output area by moving diagonally through the T-map, since the energies were generally uniform in the horizontal and vertical directions. The output images of optimal seams removal can be seen in Figures 13 and 14 below.

Figure 12. Original Image (scaled)

Figure 13. Source Paper Output

Figure 13. Replication Output

Comparing Figure 13 and 14, we can see that the outputs weren't alike as expected from the T-map output. Seams associated with the butterfly in the replication output were given a higher value, thus the size of the original buttefly was better preserved when compared to the original image in Figure 12. In the source paper output, it was can be inferred that the low energy seams in the butterfly were removed, creating a scaling effect on the butterfly. Overall, the method used in the source paper yielded a better output as it represented the important aspects of the original image (the butterfly) more clearly. However, the cause of the difference between the outputs is not certain.

5. Implementation Challenges

Many challenges were faced when replicating the techniques used in Avidan and Shamir's paper. Firstly, it was not mentioned that indices had to be updated relative to one another in the seam insertion process. Creating a method to do so was difficult and not intuitive. Second, it was unclear to me when to average seams in the insertion process. Intuitively, I thought averaging the inserted seam once it was in the expanded image would result in a smoother looking output image. However, the paper briefly implied that averaging of the added seam must be done on the input image prior to insertion. Third, challenges were faced when finding the optimal seam path. The source paper stated that they store an n x m one bit map of which choices were made during T-map generation, but then they subsequently stated that they backtracked from T(r,c) and applied removal operations. I failed to see how a bit map of each choice in the T-map creation process (1 or vertical removal and 1 for horizontal removal) created an optimal seams path. After much experimentation, I found that the bit map was the optimal seams path and was created by backtracking the T-map from the desired image size (T(r,c)).

6. References

Avidan, S., & Shamir, A. (n.d.). Seam Carving for Content-Aware Image Resizing. Retrieved February 18, 2019, from http://www.faculty.idc.ac.il/arik/SCWeb/imret/index.html