[Maksim Kazakov] [CS6475] [Spring2018]
[Video link: https://youtu.be/HylcjWdox0A]

Seam Carving Replication Project

Introduction

This project aim is to try and replicate results of Seam Carving for Content-Aware Image Resizing paper [1]. In particular, I tried to replicate image shrinking, stretching and optimal resizing using seam carving to preserve maximum details without distortions.

Seams

Seam is a connected vertical(horizontal) path from top to bottom (left to right) of the image with only one pixel per row(column). To perform efficient seam carving seam has to go along the path of the minimal details of the image. To find this path original paper used concept of energy functions that return amount of details that each pixel of the image contains. Authors presented several energy functions, but since for the experiment that I tried to replicate they used only one $e_1$function that's what I used as well:

$$e_1(I) = \left|\frac{\partial}{\partial x} I\right| + \left|\frac{\partial}{\partial y} I\right|$$

Unfortunately, authors didn't specify what method they used to calculate image derivatives, which made replication harder, since different methods can lead to slightly different energy functions. I used image convolution with 3x3 Prewitt kernels.

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, K_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

I also reflected all the edges to add two rows and two columns on the sides of the image to calculate derivatives along the edges.
After calculating energy function of the image authors used dynamic programming to calculate path with minimal energy. This can be done by traversing the image from the second row to the last row and computing the cumulative minimum energy M for all possible
connected seams for each entry (i, j):

$$M(i,j) = e(i,j) + \min\left(M(i-1,j-1), M(i-1,j), M(i-1,j+1)\right)$$

What is left is to find pixel with minimal cumulative energy at the bottom row and backtrack up the image following adjacent pixels with minimal energy. Finding horizontal seam can be done similarly or by transposing the image first.

Seam removal

First experiment that I tried to replicate was removal of seams from the image to shrunk it in size. Theoretically this should preserve maximum image details while producing smaller undistorted image (unlike normal scaling).
The process is fairly straightforward: to resize image size $n\times m$ to $n'\times m', n' < n, m' < m$ one have to remove $n - n'$ vertical seams and $m - m'$ horizontal seams. Authors didn't specify if they used any interpolation method to smooth pixels' transition after seams removal so I assumed that they just shifted pixels after removing those along the seams.
In the end I achieved relatively good result on one of the pictures authors used, though not as good as theirs. Authors also didn't specify how many seams they removed, so I approximated that their image was reduced horizontally approximately in half, so I removed exactly 350 vertical seams to arrive at the resulted image presented in Figure 1.

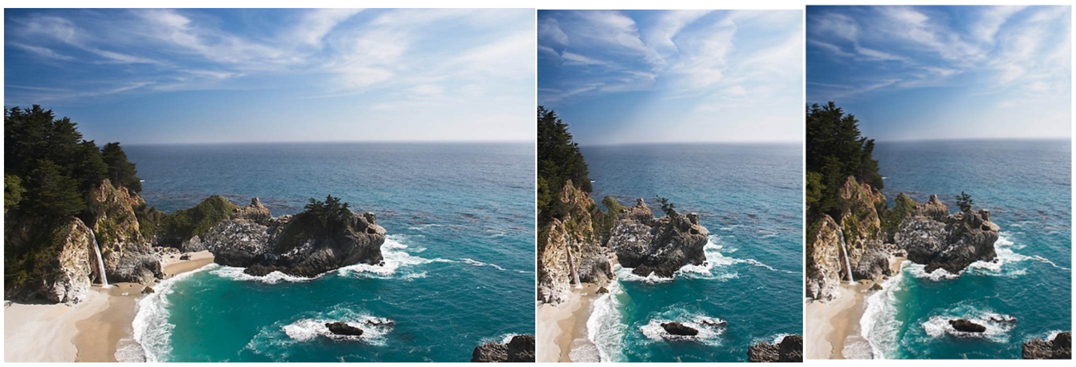


Figure 1: left - original image; middle - my version; right - authors version

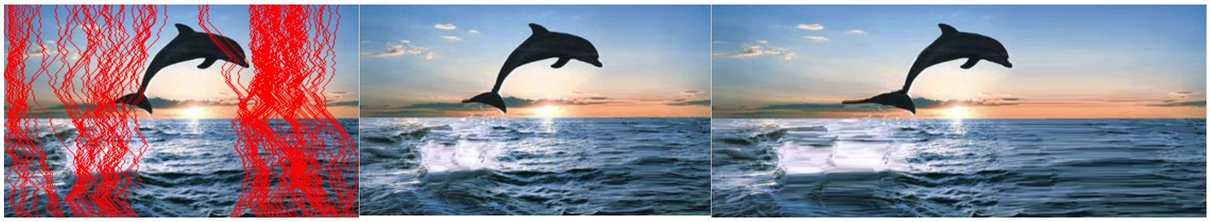
As you can see in both cases seam carving allowed to preserve main details of the image (rocks, waves, beach, clouds, waterfall) with minimal distortions to them. Authors achieved better results in this experiment, on their image objects like tree on the rock and waterfall were preserved better than on my image. These differences can be explained by original paper ambiguities mentioned above (different energy function calculation method, slightly different image size, interpolation method).

Seam insertion

Second experiment involved opposite method to seam removal – seam insertion. To enlarge the image without distorting it one can insert additional seams that bear little information (energy) and use interpolation method to calculate pixel values for the new inserted seam. But to avoid undesirable artifacts like stretching I used method similar to one described in original paper: first, find k seams for removal and then insert them to the original image. Authors didn't specify how did they find this k seams (all at once or iteratively), as well as how they inserted all those seams so that they wouldn't interfere with each other. Since my solution to solving these problems can be drastically different from what authors used, our results end up slightly different (Figure 2).
To find k seams that I wanted to later insert I've been iteratively removing seams using the same method explained in previous section and saving those seams. I also saved indices of the first seam pixel columns as they were in the original image (i_{orig}) and order in which their seams were removed. This later helped me to solve second mentioned problem of finding right offset. To calculate the offset for all images I used following formula:

$$offset = i_{orig} - i_{seam} + p,$$

where i_{orig} - column of the first seam pixel as how it would appear in original image, i_{seam} - column of the first seam pixel from the respective reduced image, p - number of seams that have been already inserted in the new image whose i_{orig} is less than i_{orig} of the current seam.
By adding this offset to all pixels of the seam I managed to find approximately correct position of the seam in the new image. But this method has a significant drawback: it assumes that all seams do not intersect, which is not always the case. Nevertheless, I was able to achieve good results with this approach.
Also to enlarge image by more that 50% it's recommended to do it by performing procedure above several times rather than in iteration, though some artifacts can still appear.



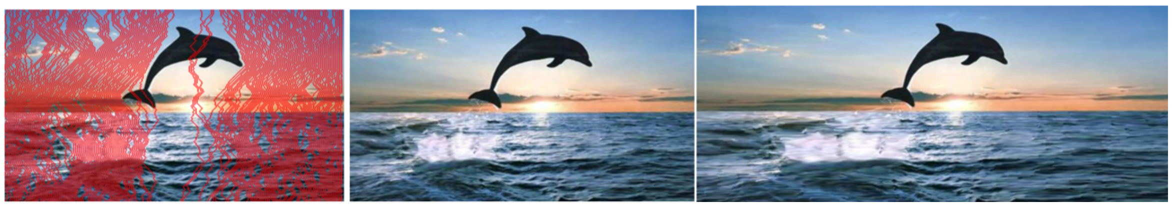


Figure 2: top – original, middle row – my version, bottom row – authors version. Left to right: seams that are being inserted, new enlarged image, image enlarged twice

My version has significantly more artifacts like stretching and object distortion, which can be due to the fact that my algorithm has found different seams and I might have used different insertion algorithm.

Optimal retargeting

Third experiment tackles the problem of finding the right order of removing horizontal and vertical seams to match new image aspect ratio. To preserve maximum amount of information authors suggested to use dynamic programming to find optimal sequence of removals. To do that transport map T have to be computed. $T(r,c)$ in this map represents amount of energy removed after removing r horizontal and c vertical seams in optimal order. It can be computed using the formula:

$$T(r,c) = \min\left\{T(r-1,c) + E\left(s_x(I_{n-r-1\times m-c})\right), T(r,c-1) + E\left(s_y(I_{n-r\times m-c-1})\right)\right\},$$

where $I_{n-r\times m-c}$ denotes an image of size n − r × m − c, $E\left(s_x(I)\right)$ is the energy of the seam from respective seam removal operation.

Iterating diagonally starting with (0,0) one can fill the whole matrix. And by remembering at each step what operation was the most optimal one can backtrack from any image dimensions to the original image dimensions and find optimal order of seam removals.

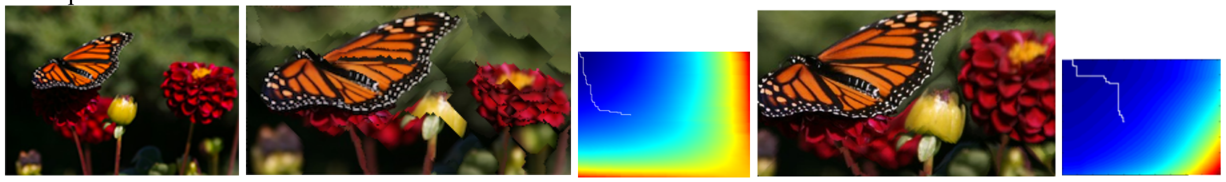


Figure 3: left to right: original image, my resized version, my transport map with optimal removal path, authors respective versions.

Figure 3 again shows that that my algorithm is slightly different from authors. Reason behind this goes back to ambiguities in seam finding and removals. This also explains why transport maps are different as well as resulted optimal paths. Authors also didn't mention what optimization techniques they used to calculate transport map. In my case its calculation took ~4hrs which makes this method hard to use in practice.

References

[1] Shai Avidan, Ariel Shamir, *Seam Carving for Content-Aware Image Resizing*