

Seam Carving for image resizing

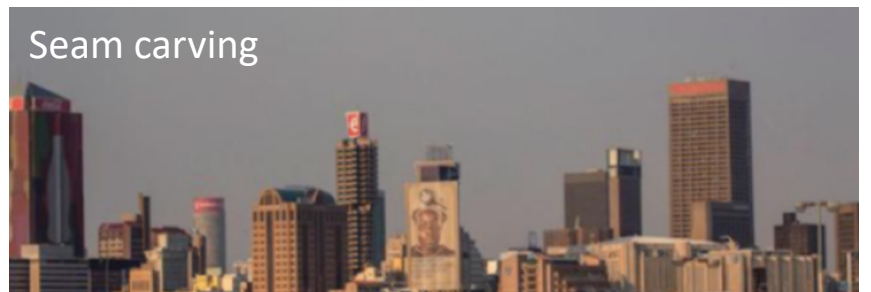
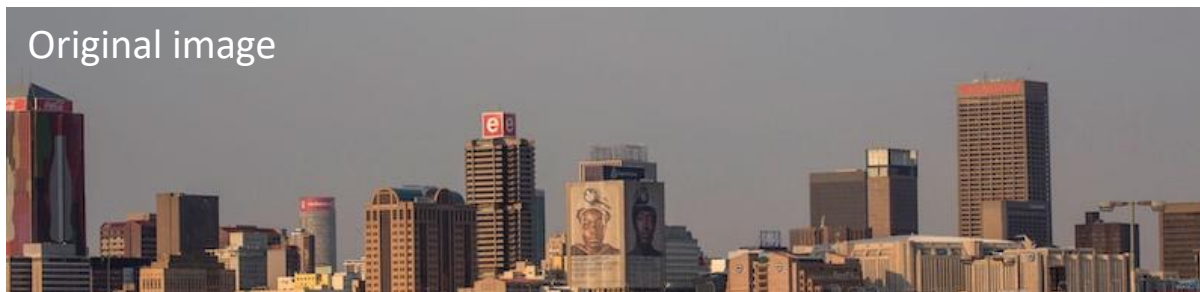
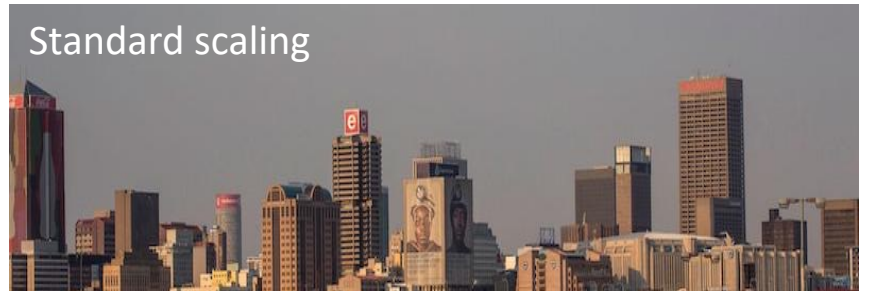
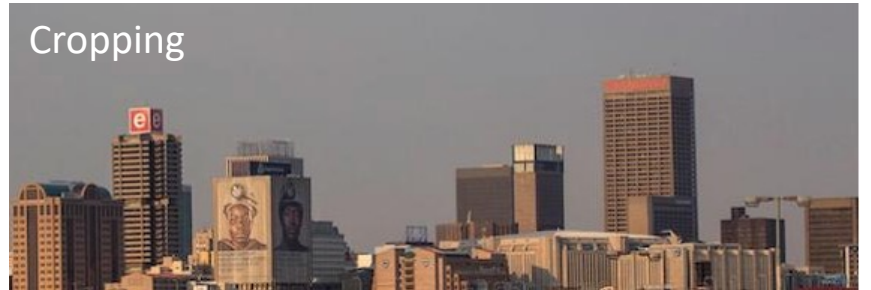
Course project of 20602

Kun Chen 3144546

What is seam carving?

In image resizing, traditional methods often encounter a trade-off between reducing the image contents and distorting the aspect ratio.

Seam carving provides a generally balanced result by firstly defining an importance measure for each pixel, and then protecting the shape and size of more important items, while resizing less important ones.

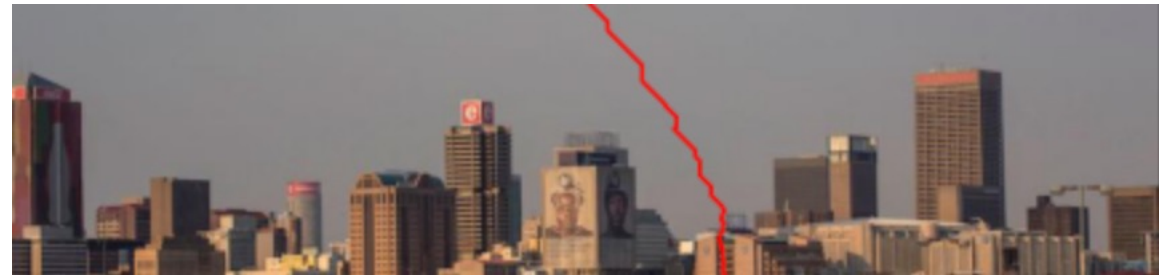


What is seam carving?

Taking width reducing as an example, seam carving iteratively identifies a one-pixel wide seam from top to bottom, which contains the unimportant pixels to be deleted. More specifically, it

- computes the pixel-importance, energy,
- aggregates energies for each possible seam,
- finds and deletes the seam with minimum energy.

Dynamic programming



Why dynamic programming here?

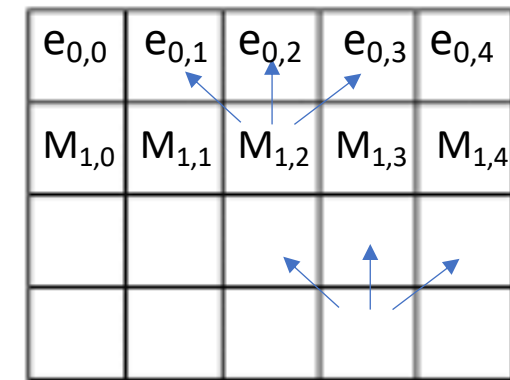
Dynamic programming is a technique for efficiently solving problems that can be broken down into highly-repeated subproblems.

- Optimal substructure: an optimal solution to the problem contains within its optimal solutions to subproblems.
- Overlapping subproblems: the space of subproblems must be “small” in the sense that a recursive algorithm for the problem solves the same subproblems over and over.

$$M(x, y) = e(x, y) + \min\{M(x - 1, y - 1), M(x - 1, y), M(x - 1, y + 1)\}$$

Cumulative Energy matrix

$e_{0,0}$	$e_{0,1}$	$e_{0,2}$	$e_{0,3}$	$e_{0,4}$
$M_{1,0}$	$M_{1,1}$	$M_{1,2}$	$M_{1,3}$	$M_{1,4}$



Bottom-up
implementation

Implementation

```
def energy(img, method='grad'):
    if method == 'grad':
        x_grad = np.hstack([img[:,1:], img[:, -1:]]) - np.hstack([img[:, :1], img[:, -1]])
        y_grad = np.vstack([img[1:, :], img[-1:, :]]) - np.vstack([img[:1, :], img[-1:, :]])
    if method == 'sobel':
        x_grad = cv.Sobel(img, cv.CV_64F, 1, 0, ksize=3)
        y_grad = cv.Sobel(img, cv.CV_64F, 0, 1, ksize=3)
    energy = np.abs(x_grad/2) + np.abs(y_grad/2)
    return energy
```

The measures of image importance

- Simple gradient magnitude
- Sobel filter

			-1		1
-1		1	-2		2
			-1		1

Horizontal changes

```

def get_one_seam(energy):
    h, w = energy.shape
    pos = np.zeros([h,w])
    agg_energy = energy.copy() # store the sum of all possible energy paths
    energy = np.hstack([energy[:,1:], energy[:,0:1]])

    for i in range(1,h):          # from the second row of agg_energy
        for j in range(w):        # from the first item of the row
            neighbor = energy[i-1, j:j+3]
            min_val, min_id = min((val, idx) for (idx, val) in enumerate(neighbor))
            agg_energy[i,j] = agg_energy[i,j] + min_val
            if (min_id==0) & (j==0):
                pos[i,j] = 0
            elif (min_id==2) & (j==w-1):
                pos[i,j] = 0
            else:
                pos[i,j] = min_id - 1

    # extract the path of the min seam
    min_val, min_id = min((val, idx) for (idx, val) in enumerate(agg_energy[-1,:]))
    path = np.ones(h) * min_id
    for i in range(1,h):
        path[i] = path[i-1] + pos[h-i, int(path[i-1])]
    path = path[::-1]
    return {'min':min_val, 'path':path}

```

Given a $W \times H$ image to be resized...

Runtime

- without DP: $(3W - 2)^{H-1}$
- with DP: $O(W \times H)$

Seam path reconstruction

The $(H - 1) \times W$ matrix 'pos' keeps track of the position of selected pixel from 3 upper neighbours. By adding its value iteratively to the smallest value entry, the lowest cost path is reconstructed.

Multiple-seam removing

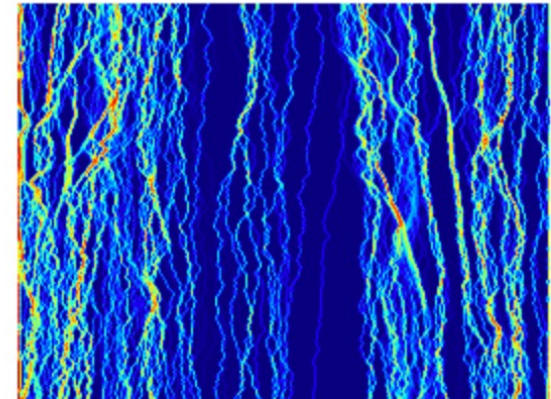
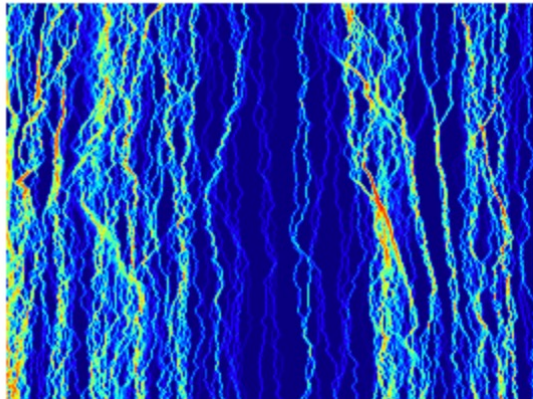
- recompute the energy matrix,
- apply function 'get_one_seam',
- delete each pixel from the image.

For better visualizations

Create a matrix 'traj' for restoring the original positions of past seams. An integer is assigned to the pixel if it is on the n^{th} low-cost path.

```
def get_final_image(imgcolor, target_width, energy_method='grad'):  
    n_rgb = imgcolor.shape[2]  
    n_seam = imgcolor.shape[1] - target_width  
    imggray = cv.cvtColor(imgcolor, cv.COLOR_RGB2GRAY)  
    h = imggray.shape[0]  
  
    traj = np.zeros_like(imggray)  
    for i in range(n_seam):  
        e = energy(imggray, method=energy_method)  
        p = get_one_seam(e)['path']  
        mask = np.ones_like(e, bool)  
  
        for j in range(h):  
            loc = int(p[j])  
            mask[j, loc] = False  
            l = np.where(traj[j, :] == False)  
            traj[j, int(l[0][loc])] = int(n_seam - i)  
        imggray = imggray[mask].reshape((h, -1))  
        mask_col = np.repeat(mask[:, :, None], n_rgb, axis=2)  
        imgcolor = imgcolor[mask_col].reshape((h, -1, n_rgb))  
    return imgcolor, traj
```

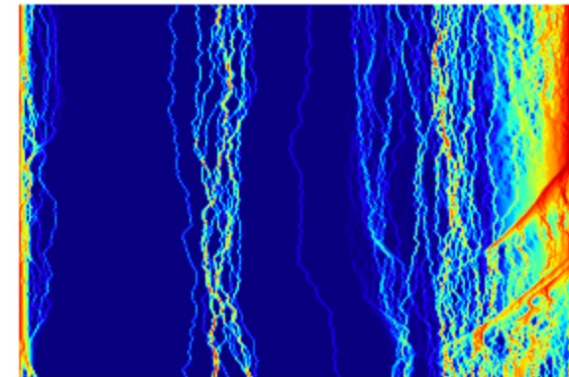
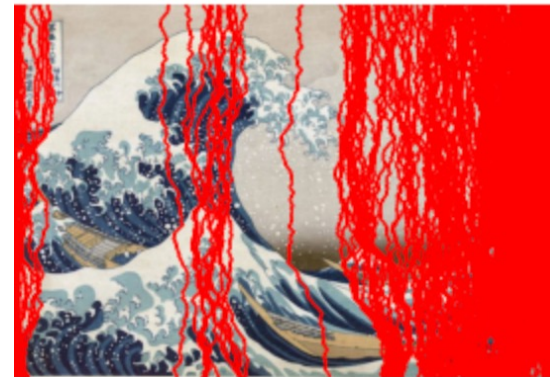
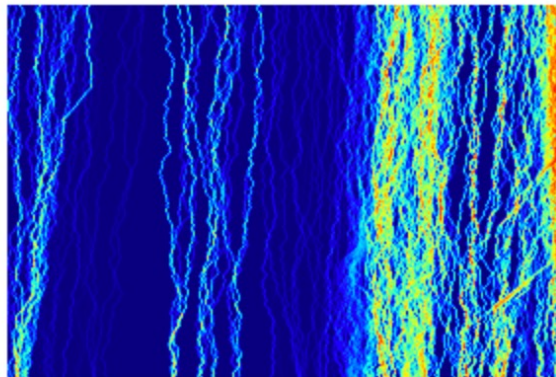
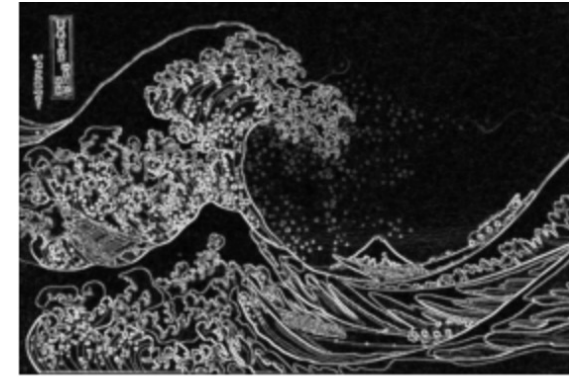
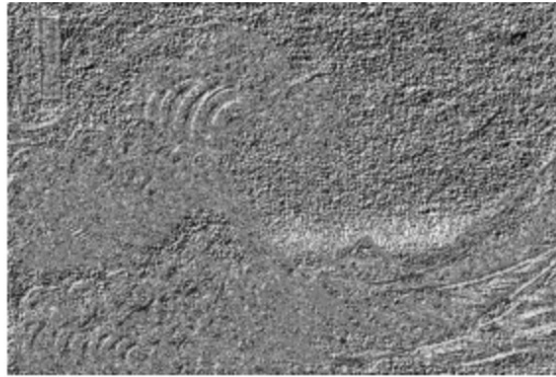

Results with different energy functions



Simple gradient magnitude

Sobel filter

Results with different energy functions



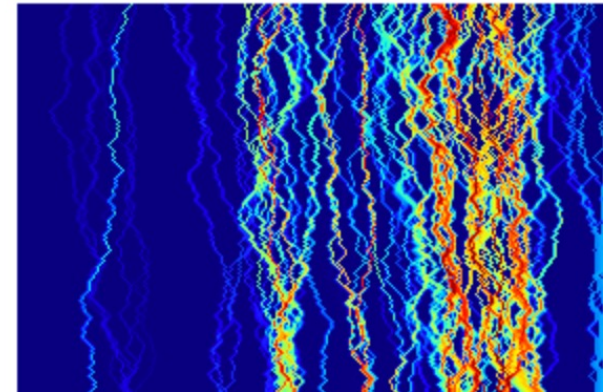
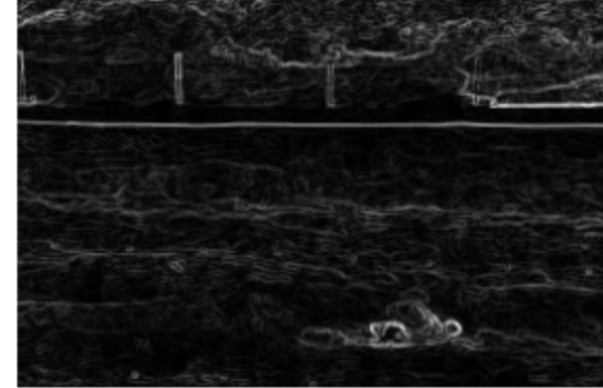
Simple gradient magnitude

Sobel filter

Results with different energy functions



Results with different energy functions



Other use cases

Image Enlarging

The opposite way of removing seams, which involves finding the k optimal seams and duplicate them by averaging them with neighbour pixels.



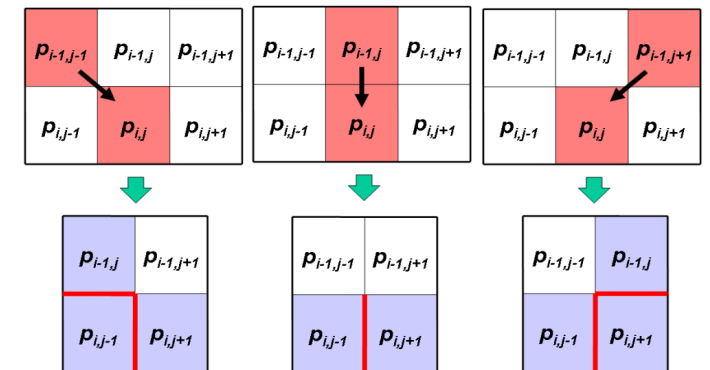
Object protection and removal

A manually created mask can be used to select an object to "protect." The pixels selected are incorporated into the gradient value matrix, making the corresponding values "high-cost."



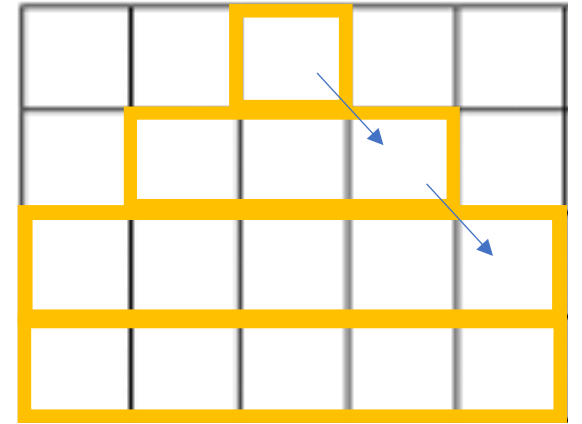
Video resizing

Taking into account what happened after removing one seam by identifying three types of extra cost when new neighbors are connected from different directions.



Optimization

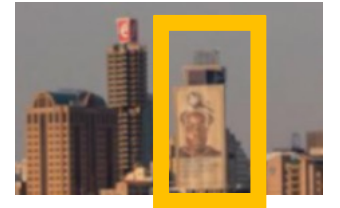
- Not all pixels of the cumulative energy matrix should be recalculated after each removal of the seam.
- Only the ones within a large triangular area containing the seam are affected. Its performance depends highly on the aspect ratio of the given image.
- It is possible to remove multiple seams using one energy matrix.
- This method is developed under the consideration of time and accuracy trade-off. Having around 4 seams being deleted per round for 500-pixel width, the impact on image quality can be negligible.



Limitations

Seam carving does not perform ideally on all images.

- Sometimes it can be improved by adding extra information.
 - manual constraints,
 - object detector.
- Other times even with higher level information, it still does not work well.
 - full of objects,
 - specific layout, irregular shape with approximately straight outlines.



Thank you