# CSC420 Assignment 2

Mengning Yang
Student Number: 1002437552

October 18, 2019

1. Implement seam carving

   (a) Compute magnitude of gradients of an image.
   The optimal seam can be found using dynamic programming.
   1. The first step is to traverse the image from the second row to the last row and compute the cumulative minimum energy M for all possible connected seams for each entry (i, j): $M(i,j) = e(i,j) + min(M(i1,j1), M(i1,j), M(i1,j+1))$
   2. At the end of this process, the minimum value of the last row in M will indicate the end of the minimal connected vertical seam. Hence, backtrack from this minimum entry on M to find the path of the optimal seam

```
import scipy
import numpy as np
from pylab import *
from scipy import ndimage
import scipy.misc as sm
from PIL import Image
import cv2

#Q1 Seam Carving
#Step 1 calculate image gradient (energy) of given image
def get_energy_function(img):
    R = img[:, :, 0]
    G = img[:, :, 1]
    B = img[:, :, 2]

    Gr = magnitude_of_gradients(R)
    Gg = magnitude_of_gradients(G)
    Gb = magnitude_of_gradients(B)

    #TA said simply add them together
    img_gradient = Gr + Gg + Gb

    #sm.imshow(img_gradient)

    return img_gradient

def magnitude_of_gradients(img):

    # Define kernel for x and y differences
    kx = np.array([[-1,0,1],[-2,0,2],[-1,0,1]],np.float32)
    ky = np.array([[1,2,1],[0,0,0],[-1,-2,-1]],np.float32)

    x = ndimage.convolve(img,ky, mode='constant', cval=0.0)
    y = ndimage.convolve(img,kx, mode='constant', cval=0.0)

    #result = math.sqrt(x*x + y*y)
    result = np.hypot(x, y)
    result = result / result.max() * 255

    #convert type int32 to type unit8
    result2 = np.array(result,dtype=np.uint8)

    #cv2.imshow('result',result2)
```

```
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    return result2
```

(b) Find the connected path of pixels that has the smallest sum of gradients. A path is valid if it is connected (the neighboring points in the path are also neighboring pixels in the image), it starts in the first row of the image and in each step continues one row down. It finishes in the last row of the image.

```
#Step 2: build a map of the minimum cumulative energies
def cumulative_energy(energy):
    """
    energy: 2D numpy.array, image gradient
    Returns:
        tuple of 2 2D numpy.array
        paths: x-offset of the previous seam element for each pixel.
        cumulative_energies: cumulative energy at each pixel.
        seam_tail: the x-coordinate of the pixel in the last row
        of cumulative_energies with least energy.
    """
    height, width = energy.shape

    #pad with zeros first
    paths = np.zeros((height, width), dtype=np.int64)
    cumulative_energies = np.zeros((height, width), dtype=np.int64)
    paths[0] = np.arange(width) * np.nan

    for i in range(1, height):
        for j in range(width):
            #M(i,j) = e(i,j) + min(M(i-1, j-1),M(i-1,j),M(i-1,j+1))
            prev_energies = cumulative_energies[i-1, max(j-1, 0):j+2]
            least_energy = prev_energies.min()
            cumulative_energies[i][j] = energy[i][j] + least_energy
            paths[i][j] =
            np.where(prev_energies == least_energy)[0][0] - (1*(j != 0))

    #get the x-coordinate of the pixel with least energy
    seam_tail
    = list(cumulative_energies[-1]).index(min(cumulative_energies[-1]))

    return paths, cumulative_energies, seam_tail


#Step 3: do a backtrace to get the path (seam) with the lowest energy
def find_seam(paths, end_x):
    """
    paths: 2D numpy.array. Each element of the matrix is the offset
    of the index to the previous pixel in the seam
    end_x: integer. The x-coordinate of the pixel with the min
    energy in the last row
    Returns a 1D numpy.array with length == height of the image
```

```
    Each element is the x-coordinate of the pixel to be
    removed at that y-coordinate. e.g.
    [4,4,3,2] means "remove pixels (0,4), (1,4), (2,3), and (3,2)"
    """
    height, width = paths.shape[:2]
    seam = [end_x]
    for i in range(height-1, 0, -1):
        cur_x = seam[-1]
        offset_of_prev_x = paths[i][cur_x]
        seam.append(cur_x + offset_of_prev_x)
    seam.reverse()
    return seam
```

(c) Remove the pixels in the path from the image. This gives you a new image with one column less.

```
#Step 4: delete the pixels which belong to the seam
def remove_seam(img, seam):
    """
    img: 3D numpy.array RGB image
    seam: 1D numpy.array seam to remove
    Returns 3D numpy array of the image with the seam removed
    """
    height, width, _ = img.shape
    return np.array([np.delete(img[row], seam[row], axis=0) for
            row in range(height)])
```

(d) Remove a few paths with the lowest sum of gradients. Create a few examples and include in your document.

```
if __name__ == "__main__":
    img = "/Users/CYANG/Desktop/fall.jpg"
    image = np.array(Image.open(img))

    for i in range (100):
        #Step 1: calculate the energy for each pixel of the image
        energy_map = get_energy_function(image)
        paths, cumulative_energies, seam_tail = cumulative_energy(energy_map)

        #Step 2: build a map of the minimum cumulative energies
        paths, cumulative_energies, seam_tail = cumulative_energy(energy_map)

        #Step 3: do a backtrace on this data to get the path (seam)
        #with the lowest energy
        seam = find_seam(paths, seam_tail)

        #Step 4: delete the pixels which belong to the seam
        image = remove_seam(image, seam)


    image = image[:,:,::-1]

    cv2.imshow('result',image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Figure 1: before seam carving

Figure 2: after seam carving

Figure 3: before seam carving

Figure 4: after seam carving

Figure 5: before seam carving

Figure 6: after seam carving

2. Image upscaling

(a) Write down the mathematical form of the convolution filter that performs up- scaling of a 1D signal by a factor d. You do not need to write code. Please plot this filter (you can plot it by hand).

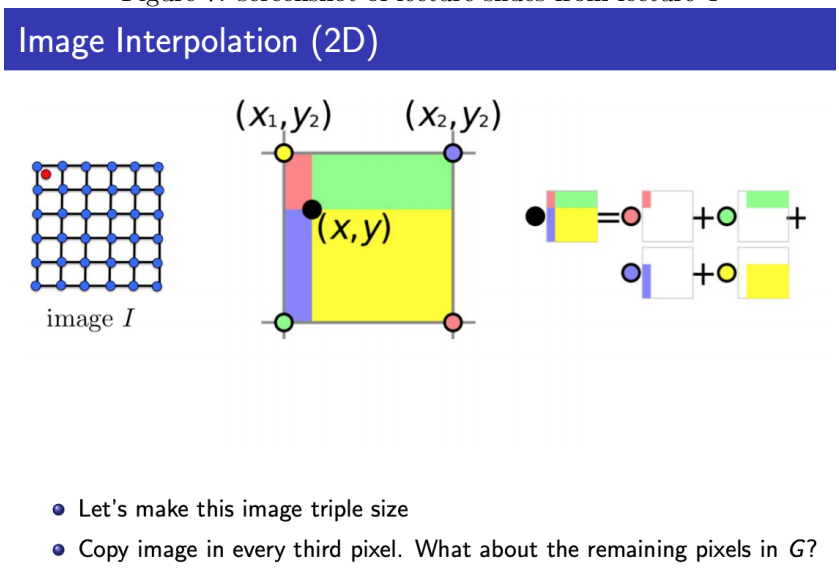
2(1) Write down the mathematical form of the convolution filter that performs upscaling of a 1D signal by a factor of $d$.

Convolution filter: $[0, \frac{1}{d}, \frac{2}{d}, \frac{3}{d}, \ldots\ldots \frac{d-1}{d}, 1, \frac{d-1}{d}, \frac{d-2}{d} \ldots\ldots \frac{1}{d}, 0]$
where $d$ is the upscaling factor.

mathematical form:
$$h(x) = \begin{cases} \frac{x}{d} & 0 \le |x| \le d \\ 1 - \frac{x}{d} & |x| > d \end{cases}$$
where $d$ is the upscaling factor

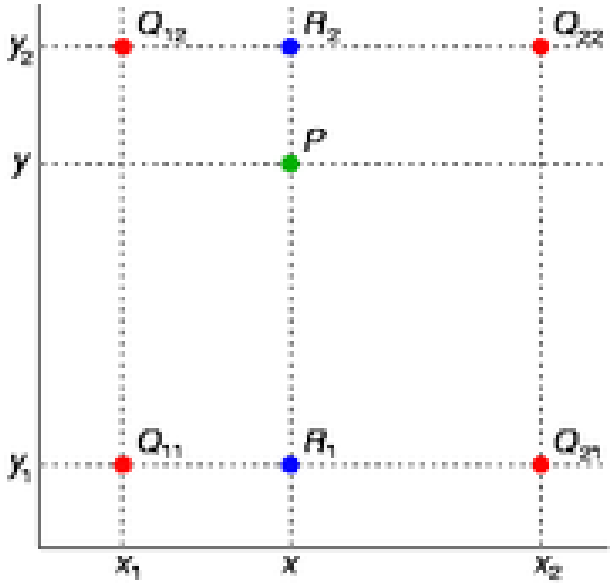
(b) Implement a function that upscales an image to a 3x resolution. Please explain your implementation. Do not use built-in functions for upscaling.
Explaination of my implementation:
My implementation basically follows the algorithm in the following attached lecture slide.
As you can see from the following figure, we have four points that are known, Q11, Q12, Q21, Q22, and we wish to interpolate the points between these four points, for instance, point P(x,y) in the following figure. We calculate its value by using the proportion of how much of a mix the output consists of between the four known points.
We can do a linear interpolation along the X axis and we have
$f(R1) = \frac{x2-x}{x2-x1} f(Q11) + \frac{x-x1}{x2-x1} f(Q21)$ *where* $R1 = (x, y1)$ and
$f(R2) = \frac{x2-x}{x2-x1} f(Q12) + \frac{x-x1}{x2-x1} f(Q21)$ *where* $R2 = (x, y2)$.

Figure 7: screenshot of lecture slides from lecture 4




Then, we do a linear interpolation along the Y axis and we have
$f(P) = \frac{y2-y}{y2-y1} f(R1) + \frac{y-y1}{y2-y1} f(R2)$.

Finally, combine these result and we have our formula for bilinear interpolation.
$f(x,y) = \frac{f(Q11)}{(x2-x1)(y2-y1)}(x2-x)(y2-y) + \frac{f(Q21)}{(x2-x1)(y2-y1)}(x-x1)(y2-y) + \frac{f(Q12)}{(x2-x1)(y2-y1)}(x2-x)(y-y1) + \frac{f(Q22)}{(x2-x1)(y2-y1)}(x-x1)(y-y1)$.



```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import math

def BiLinear_interpolation(img):
    img = np.array(Image.open(img))
    scrH,scrW,_ = img.shape
    #upscales an image to a 3x resolution
    dstH = scrH*3
    dstW = scrW*3
    img=np.pad(img,((0,1),(0,1),(0,0)),'constant')
    retimg=np.zeros((dstH,dstW,3),dtype=np.uint8)

    for i in range(dstH):
        for j in range(dstW):
            scrx=(i+1)*(scrH/dstH)-1
            scry=(j+1)*(scrW/dstW)-1
            x=math.floor(scrx)
            y=math.floor(scry)
            u=scrx-x
            v=scry-y

            retimg[i,j] = (1-u)*(1-v)*img[x,y] +
            u*(1-v)*img[x+1,y] +
            (1-u)*v*img[x,y+1] +
            u*v*img[x+1,y+1]

    result = Image.fromarray(retimg.astype('uint8')).convert('RGB')
    result.save("/Users/CYANG/Desktop/test_result.png")
    return result
```

3. Interest point detection

   (a) Implement a function to perform Harris corner detection. The function should take as input an image, and return corners. You can make the image grayscale.

```
function out = harriscorner(filename)
% read image and grayscale
im = imread(filename);

% convert to grayscale
img = rgb2gray(im);

%get dimensions
[img_height, img_width] = size(img);

% get image gradients Ix and Iy
[Ix, Iy] = imgradientxy(img);

% compute 3 matrixes Ix^2, Iy^2, Ix * Iy
```

```
Ix2 = Ix.^2;
Iy2 = Iy.^2;
Ixy = Ix.*Iy;

% compute matrix M = [Ix2g, Ixyg; Ixyg, Iy2g];
% convolving each one with a filter, e.g. a box or Gaussian filter
Ix2g = conv2(Ix2, fspecial('gaussian'), 'same');
Iy2g = conv2(Iy2, fspecial('gaussian'), 'same');
Ixyg = conv2(Ixy, fspecial('gaussian'), 'same');

% compute R = det(M) - alpha*trace(M)^2
alpha = 0.04; % a constance

Rmax = 0;
R = zeros(img_height, img_width);

det_M = Ix2g.*Iy2g - Ixyg.^2;
trace_M = Ix2g + Iy2g;

R = det_M - alpha*((trace_M).^2);
Rmax = max(max(R));

% Non-maximum suppression
Res = zeros(img_height, img_width);
thresh = Rmax*0.028;

for row = 1:img_height
    for col = 1:img_width
        % check for local max
        if R(row,col) > thresh
            local_max = true;
            for n = row-1:row+1
                for m = col-1:col+1
                    if R(n, m) > R(row, col)
                        local_max = false;
                        break
                    end
                end
                if local_max == false
                    break
                end
            end
            if local_max == true
                Res(row,col) = 1;
            end
        end
    end
end

[cols, rows] = find(Res == 1);
imshow(img);
hold on;
plot(rows,cols,'r.');
```

**end**

(b) Plot your result for the attached image building.jpg, and add it to your pdf/doc file.

Figure 8: building after harris corner detection