PP03: Seam Carving

CST501: Advanced Algorithms Programming Project Write-Up

Posting ID: 6421-735 .

Pledge

I Becca Little pledge, on my honor, that the work I submit is my own and that I have neither sought nor provided inappropriate help to any other. I understand that I may not share any part of my solution or design (even if I have a bug and want help!) with any person other than the grader, or instructor.

I Becca Little further understand that I must submit this completed write-up with my name inserted into the pledge, and also a .zip file containing my source-code as presented here as two separate attachments to the course website.

Highlighted Source Code:

```python
from pylab import *  # NOQA
from skimage import img_as_float
import math


def dual_gradient_energy(img):
    """Return a W x H array of floats, the energy at each pixel
      in the passed image.
    """
    h = img.shape[0]
    w = img.shape[1]
    R = img[:, :, 0]
    G = img[:, :, 1]
    B = img[:, :, 2]
    dxRed = [[[] for _ in xrange(w)] for _ in xrange(h)]
    dyRed = [[[] for _ in xrange(w)] for _ in xrange(h)]
    dxGreen = [[[] for _ in xrange(w)] for _ in xrange(h)]
    dyGreen = [[[] for _ in xrange(w)] for _ in xrange(h)]
    dxBlue = [[[] for _ in xrange(w)] for _ in xrange(h)]
    dyBlue = [[[] for _ in xrange(w)] for _ in xrange(h)]
    energy = [[[] for _ in xrange(w)] for _ in xrange(h)]
    for i in range(0, h):
        dxRed[i][0] = R[i][1] - R[i][w - 1]
        dxRed[i][0] = dxRed[i][0] * dxRed[i][0]
        dxRed[i][w - 1] = R[i][0] - R[i][w - 2]
        dxRed[i][w - 1] = dxRed[i][w - 1] * dxRed[i][w - 1]
        dxGreen[i][0] = G[i][1] - G[i][w - 1]
```

```python
        dxGreen[i][0] = dxGreen[i][0] * dxGreen[i][0]
        dxGreen[i][w - 1] = G[i][0] - G[i][w - 2]
        dxGreen[i][w - 1] = dxGreen[i][w - 1] * dxGreen[i][w - 1]
        dxBlue[i][0] = B[i][1] - B[i][w - 1]
        dxBlue[i][0] = dxBlue[i][0] * dxBlue[i][0]
        dxBlue[i][w - 1] = B[i][0] - B[i][w - 2]
        dxBlue[i][w - 1] = dxBlue[i][w - 1] * dxBlue[i][w - 1]

        energy[i][0] = dxRed[i][0] + dxGreen[i][0] + dxBlue[i][0]
        energy[i][w - 1] = dxRed[i][w - 1] + \
            dxGreen[i][w - 1] + dxBlue[i][w - 1]

        for j in range(1, w - 1):
            dxRed[i][j] = R[i][j + 1] - R[i][j - 1]
            dxRed[i][j] = dxRed[i][j] * dxRed[i][j]
            dxGreen[i][j] = G[i][j + 1] - G[i][j - 1]
            dxGreen[i][j] = dxGreen[i][j] * dxGreen[i][j]
            dxBlue[i][j] = B[i][j + 1] - B[i][j - 1]
            dxBlue[i][j] = dxBlue[i][j] * dxBlue[i][j]

            energy[i][j] = dxRed[i][j] + dxGreen[i][j] + dxBlue[i][j]

    for i in range(0, w):
        dyRed[0][i] = R[1][i] - R[h - 1][i]
        dyRed[0][i] = dyRed[0][i] * dyRed[0][i]
        dyRed[h - 1][i] = R[0][i] - R[h - 2][i]
        dyRed[h - 1][i] = dyRed[h - 1][i] * dyRed[h - 1][i]
        dyGreen[0][i] = G[1][i] - G[h - 1][i]
        dyGreen[0][i] = dyGreen[0][i] * dyGreen[0][i]
        dyGreen[h - 1][i] = G[0][i] - G[h - 2][i]
        dyGreen[h - 1][i] = dyGreen[h - 1][i] * dyGreen[h - 1][i]
        dyBlue[0][i] = B[1][i] - B[h - 1][i]
        dyBlue[0][i] = dyBlue[0][i] * dyBlue[0][i]
        dyBlue[h - 1][i] = B[0][i] - B[h - 2][i]
        dyBlue[h - 1][i] = dyBlue[h - 1][i] * dyBlue[h - 1][i]

        energy[0][i] += dyRed[0][i] + dyGreen[0][i] + dyBlue[0][i]
        energy[h - 1][i] += dyRed[h - 1][i] + \
            dyGreen[h - 1][i] + dyBlue[h - 1][i]

        for j in range(1, h - 1):
            dyRed[j][i] = R[j + 1][i] - R[j - 1][i]
            dyRed[j][i] = dyRed[j][i] * dyRed[j][i]
            dyGreen[j][i] = G[j + 1][i] - G[j - 1][i]
            dyGreen[j][i] = dyGreen[j][i] * dyGreen[j][i]
```

```
            dyBlue[j][i] = B[j + 1][i] - B[j - 1][i]
            dyBlue[j][i] = dyBlue[j][i] * dyBlue[j][i]

            energy[j][i] += dyRed[j][i] + dyGreen[j][i] + dyBlue[j][i]
    print("Energy is: ")
    print(energy)
    return energy


def find_seam(img, energy):
    """Return an array of H integers, for each row of the passed
       image return the column of the seam (lowest energy path).
    """
    h = img.shape[0]
    w = img.shape[1]
    shortestPath = [[[[] for _ in range(2)] for _ in xrange(w)]
                    for _ in xrange(h)]
    for j in range(0, w):
        shortestPath[0][j][0] = -1
        shortestPath[0][j][1] = energy[0][j]
        for i in range(1, h):
            shortestPath[i][j][0] = -1
            shortestPath[i][j][1] = float("inf")
    for j in range(0, w):
        for i in range(0, h - 1):
            if (j == 0):
                if (shortestPath[i][j][1] < shortestPath[i][j + 1][1]):
                    shortestPath[i + 1][j][0] = j
                    shortestPath[i + 1][j][1] = shortestPath[i][j][1] + \
                        energy[i + 1][j]
                else:
                    shortestPath[i + 1][j][0] = j + 1
                    shortestPath[i + 1][j][1] = shortestPath[i][j + 1][1] + \
                        energy[i + 1][j]
            if (j == w - 1):
                if (shortestPath[i][j - 1][1] < shortestPath[i][j][1]):
                    shortestPath[i + 1][j][0] = j - 1
                    shortestPath[i + 1][j][1] = shortestPath[i][j - 1][1] + \
                        energy[i + 1][j]
                else:
                    shortestPath[i + 1][j][0] = j
                    shortestPath[i + 1][j][1] = shortestPath[i][j][1] +
```

```
\
                        energy[i + 1][j]
            else:
                if (shortestPath[i][j - 1][1] < shortestPath[i][j][1]):
                    if (shortestPath[i][j - 1][1] < shortestPath[i][j + 1][1]):
                        shortestPath[i + 1][j][0] = j - 1
                        shortestPath[i + 1][j][1] = shortestPath[i][j - 1][1] + \
                            energy[i + 1][j]
                    else:
                        shortestPath[i + 1][j][0] = j + 1
                        shortestPath[i + 1][j][1] = shortestPath[i][j + 1][1] + \
                            energy[i + 1][j]
                else:
                    if (shortestPath[i][j][1] < shortestPath[i][j + 1][1]):
                        shortestPath[i + 1][j][0] = j
                        shortestPath[i + 1][j][1] = shortestPath[i][j][1] + \
                            energy[i + 1][j]
                    else:
                        shortestPath[i + 1][j][0] = j + 1
                        shortestPath[i + 1][j][1] = shortestPath[i][j + 1][1] + \
                            energy[i + 1][j]

    optimal = (-1, float("inf"))
    for j in range(0, w):
        if (shortestPath[h - 1][j][1] < optimal[1]):
            optimal = (j, shortestPath[h - 1][j][1])
    seam = [[] for _ in xrange(h)]
    seam[h - 1] = optimal[0]
    for i in range(h - 2, -1, -1):
        seam[i] = shortestPath[i + 1][seam[i + 1]][0]
    print("Seam is:")
    print(seam)
    return seam


def remove_seam(img, seam):
    """Modify passed image in-place and return a W-1 x H x 3
      slice - the image minus the seam.
    """
```

```
    h = img.shape[0]
    w = img.shape[1]

    newImage = [[[] for _ in xrange(w - 1)] for _ in xrange(h)]
    for i in range(0, h):
        currentJ = 0
        for j in range(0, w):
            if (j != seam[i]):
                newImage[i][currentJ][:] = img[i][j][:]
                currentJ = currentJ + 1
    print("The image without the seam is: ")
    print(newImage)
    return newImage


def plot_seam(img, seam, energy):
    """Plot the original image, its energy function, a visualization
      of the seam, and the new image with seam removed.
    """
    h = img.shape[0]
    w = img.shape[1]

    figure()
    gray()

    subplot(2, 2, 1)
    imshow(img)
    title('RGB')

    maxEnergy = 0
    for i in range(0, h):
        for j in range(0, w):
            if (energy[i][j] > maxEnergy):
                maxEnergy = energy[i][j]
    energyImage = [[[] for _ in xrange(w)] for _ in xrange(h)]
    for i in range(0, h):
        for j in range(0, w):
        energyImage[i][j] = math.floor(energy[i][j] / maxEnergy * 255)
    subplot(2, 2, 2)
    imshow(energyImage)
    title('energy')

    seamImage = energyImage
    for i in range(0, h):
```

```python
        seamImage[i][seam[i]] = 255.0
    subplot(2, 2, 4)
    imshow(seamImage)
    title('seam')

    newImage = remove_seam(img, seam)
    subplot(2, 2, 3)
    imshow(newImage)
    title('new')

    show()


def main():
    """Use dynamic programming to find the seam for test images
      and plot relevant visualizations.
    """
    surfing = imread("HJoceanSmall.png")
    small1 = [
     [[255.0, 101.0, 51.0], [255.0, 101.0, 153.0], [255.0, 101.0, 255.0]],
     [[255.0, 153.0, 51.0], [255.0, 153.0, 153.0],
      [255.0, 153.0, 255.0]],
     [[255.0, 203.0, 51.0], [255.0, 204.0, 153.0],
      [255.0, 205.0, 255.0]],
     [[255.0, 255.0, 51.0], [255.0, 255.0, 153.0], [255.0, 255.0, 255.0]]]
    small2 = [
     [[97.0, 82.0, 107.0], [220.0, 172.0, 141.0], [243.0, 71.0, 205.0],
         [129.0, 173.0, 222.0], [225.0, 40.0, 209.0], [66.0, 109.0, 219.0]],
     [[181.0, 78.0, 68.0], [15.0, 28.0, 216.0], [245.0, 150.0, 150.0],
      [177.0, 100.0, 167.0], [205.0, 205.0, 177.0], [147.0, 58.0, 99.0]],
     [[196.0, 224.0, 21.0], [166.0, 217.0, 190.0], [128.0, 120.0, 162.0],
      [104.0, 59.0, 110.0], [49.0, 148.0, 137.0], [192.0, 101.0, 89.0]],
     [[83.0, 143.0, 103.0], [110.0, 79.0, 247.0], [106.0, 71.0, 174.0],
      [92.0, 240.0, 205.0], [129.0, 56.0, 146.0], [121.0, 111.0, 147.0]],
     [[82.0, 157.0, 137.0], [92.0, 110.0, 129.0], [183.0, 107.0, 80.0],
```

```
    [89.0, 24.0, 217.0], [207.0, 69.0, 32.0], [156.0, 112.0, 31.0]]]

img1 = img_as_float(small1)
print("For the first small data set,")
print("The image is: ")
print(img1)
energy1 = dual_gradient_energy(img1)
seam1 = find_seam(img1, energy1)
plot_seam(img1, seam1, energy1)

img2 = img_as_float(small2)
print("For the second small data set,")
print("The image is: ")
print(img2)
energy2 = dual_gradient_energy(img2)
seam2 = find_seam(img2, energy2)
plot_seam(img2, seam2, energy2)

img3 = img_as_float(surfing)
energy3 = dual_gradient_energy(img3)
seam3 = find_seam(img3, energy3)
plot_seam(img3, seam3, energy3)


if __name__ == '__main__':
    main()
```

Example Output:

Flake8:

```
flake8 --max-complexity 0 ralittl1_seamcarving.py
ralittl1_seamcarving.py:6:1: C901 'dual_gradient_energy' is too complex (6)
ralittl1_seamcarving.py:82:1: C901 'find_seam' is too complex (16)
ralittl1_seamcarving.py:149:1: C901 'remove_seam' is too complex (5)
ralittl1_seamcarving.py:168:1: C901 'plot_seam' is too complex (8)
ralittl1_seamcarving.py:210:1: C901 'main' is too complex (2)
```

For all images, I used white as my seam color.

I used the test data from the Princeton assignment specification.

For the first small data set,

The image is:

```
[[[ 255.  101.   51.], [ 255.  101.  153.], [ 255.  101.  255.]]
 [[ 255.  153.   51.], [ 255.  153.  153.], [ 255.  153.  255.]]
 [[ 255.  203.   51.], [ 255.  204.  153.], [ 255.  205.  255.]]
 [[ 255.  255.   51.], [ 255.  255.  153.], [ 255.  255.  255.]]]
```

Energy is:

[[20808.0, 52020.0, 20808.0],
[20808.0, 52225.0, 21220.0],
[20809.0, 52024.0, 20809.0],
[20808.0, 52225.0, 21220.0]]

Seam is:

[0, 0, 0, 0]

The image without the seam is:

[[[255.0, 101.0, 153.0], [255.0, 101.0, 255.0]],
[[255.0, 153.0, 153.0], [255.0, 153.0, 255.0]],
[[255.0, 204.0, 153.0], [255.0, 205.0, 255.0]],
[[255.0, 255.0, 153.0], [255.0, 255.0, 255.0]]]

For the second small data set,

The image is:

```
[[[ 97.  82. 107.], [ 220. 172. 141.], [ 243.  71. 205.], [ 129. 173. 222.], [ 225.  40. 209.], [  66. 109. 219.]]
 [[ 181.  78.  68.], [  15.  28. 216.], [ 245. 150. 150.], [ 177. 100. 167.], [ 205. 205. 177.], [ 147.  58.  99.]]
 [[ 196. 224.  21.], [ 166. 217. 190.], [ 128. 120. 162.], [ 104.  59. 110.], [  49. 148. 137.], [ 192. 101.  89.]]
 [[  83. 143. 103.], [ 110.  79. 247.], [ 106.  71. 174.], [  92. 240. 205.], [ 129.  56. 146.], [ 121. 111. 147.]]
 [[  82. 157. 137.], [  92. 110. 129.], [ 183. 107.  80.], [  89.  24. 217.], [ 207.  69.  32.], [ 156. 112.  31.]]]
```

Energy is:

[[54572.0, 51263.0, 25436.0, 17321.0, 47599.0, 36173.0],
[69374.0, 23346.0, 51304.0, 31519.0, 55112.0, 61426.0],
[39387.0, 47908.0, 61346.0, 35919.0, 38887.0, 46630.0],
[42086.0, 31400.0, 37927.0, 14437.0, 63076.0, 16315.0],
[17637.0, 47935.0, 34879.0, 10471.0, 60270.0, 42607.0]]

Seam is:

[3, 3, 3, 3, 3]

The image without the seam is:

[[[97.0, 82.0, 107.0], [220.0, 172.0, 141.0], [243.0, 71.0, 205.0], [225.0, 40.0, 209.0], [66.0, 109.0, 219.0]],
[[181.0, 78.0, 68.0], [15.0, 28.0, 216.0], [245.0, 150.0, 150.0], [205.0, 205.0, 177.0], [147.0, 58.0, 99.0]],
[[196.0, 224.0, 21.0], [166.0, 217.0, 190.0], [128.0, 120.0, 162.0], [49.0, 148.0, 137.0], [192.0, 101.0, 89.0]],
[[83.0, 143.0, 103.0], [110.0, 79.0, 247.0], [106.0, 71.0, 174.0], [129.0, 56.0, 146.0], [121.0, 111.0, 147.0]],
[[82.0, 157.0, 137.0], [92.0, 110.0, 129.0], [183.0, 107.0, 80.0], [207.0, 69.0, 32.0], [156.0, 112.0, 31.0]]]

Finally, I used the surfing picture displayed in both our assignment specification and the Princeton assignment specification.

Reflection

I feel that I learned a lot about image processing. It is not as scary as it seems, especially when you are able to break images down into arrays of pixels. That makes sense. I also feel that I took away a lot about dynamic programming because I forced myself to think through the problem rather than looking up optimal solutions.

My biggest challenge still is with syntax for python. I do feel like I am getting better at quickly debugging as I settle into the python attitude.

Although I do not receive any flake8 warnings, my complexity levels are much higher than levels of my classmates according to discussion board postings. I imagine this is partly due to, in my dual_gradient_energy function, me not using vsobel and hsobel functions but instead precisely calculating the energy at each pixel. Also, instead of implementing a classic shortest path algorithm like Dijkstra's, I implemented my own from scratch. I felt this was more conducive to me getting a good feel for coming up with unique solutions to dynamic programming problems. I did spend a good chunk of time on this step, the find_seam function, and tried 3-4 different methods before converging on the correct solution. Overall this project took me much less time than the previous projects.

15-8a Show that the number of such possible seams grows at least exponentially in *m*, assuming that *n*>1.
At each pixel we have either 2 or 3 choices for the next pixels. Thus there are at least 2^n possible seams.
15-8b Give an algorithm to find a seam with the lowest disruption measure. How efficient is your algorithm?
For each row beyond the first row, determine which pixel in the previous row reachable from the current pixel has the lowest disruption value. Add that disruption value to the current pixels disruption value and save the column index of the choice that was made. After all rows have been treated in this way, find the pixel in the last row with the lowest summed disruption value and traverse back through the column indices. This is your seam.
My algorithm is O(n) because we must examine 2-3 points per pixel, Then we must examine every element in the last row. Then we must traverse back through the height of the image. In total, the running time is O(3n)+m+n = O(n).