Coding Assignment 3

**Posting ID: 5582-527**

# 1. Reflection

The objective of the assignment is to find least energy path of pixels (using dual gradient energy) in a given image so that the image can be reduced in width or height without losing significant information from the image. This could be achieved by using either a greedy approach or dynamic programming.

The approach taken in this submission is dynamic programming. The 4 functions implemented to achieve Seam Carving are as follows:

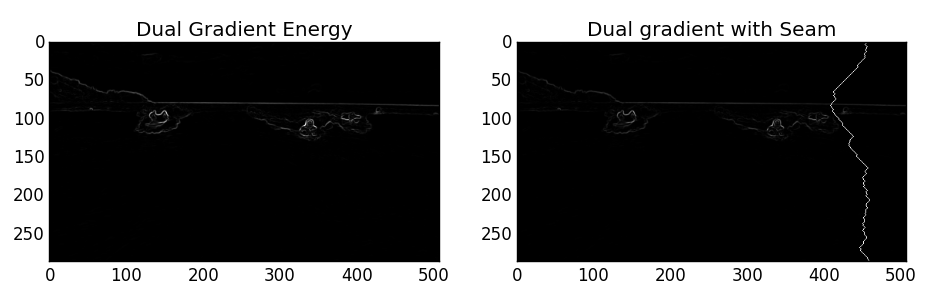
* The “dual\_gradient\_energy” function calculates and returns a 2-D array containing the energy at each pixel.
* The “find\_seam” function adopts a bottom-up approach and finds the least energy pixel path at the last row. It then returns the column index of each row that lies on the identified path.
* The “plot\_seam” function displays the image given as input, the energy function of the image and the seam identified by “find\_seam” function.
* The “remove\_seam” function returns the image by removing the pixel path in-place that was identified by “find\_seam” function.

# 2. Testing Output

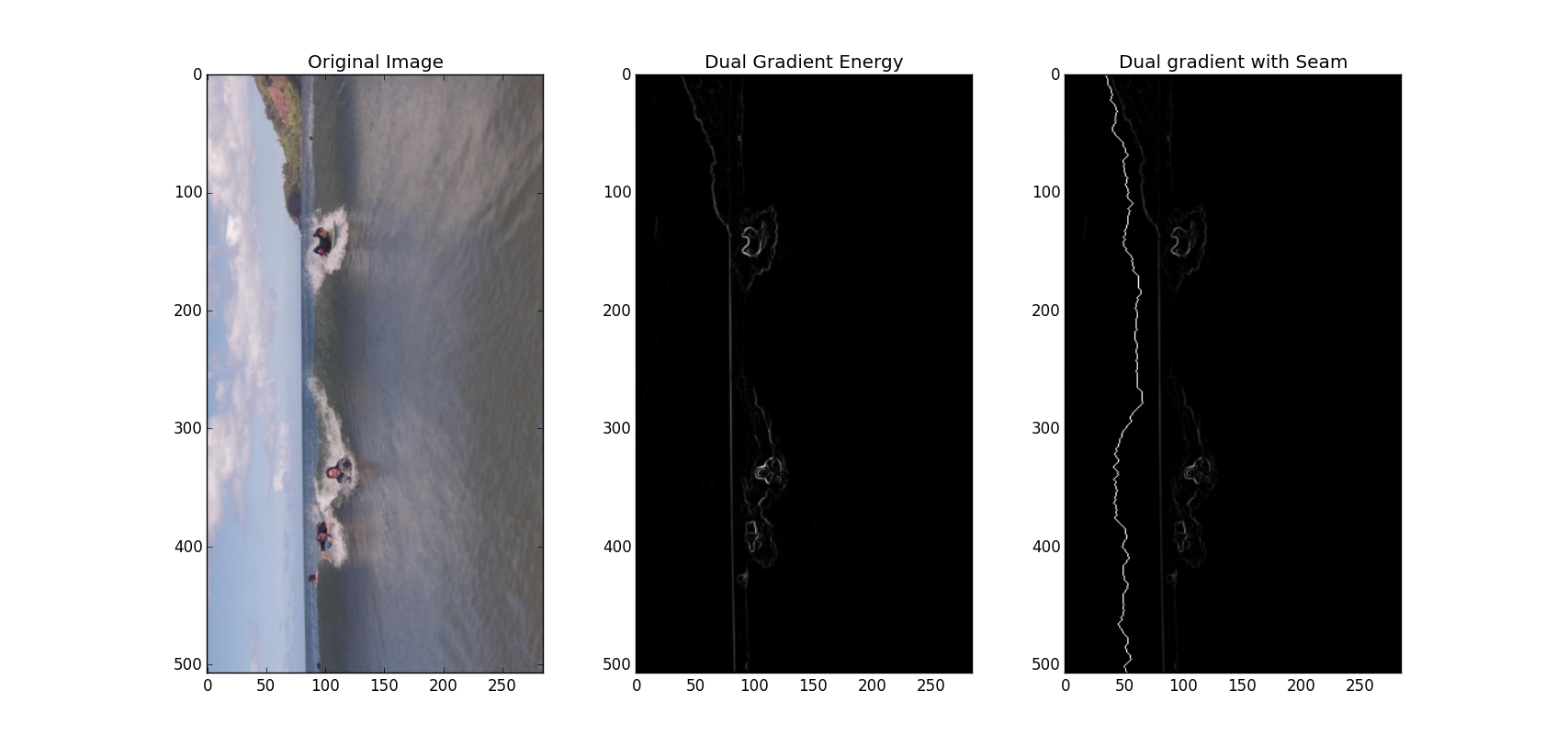
Below is the original image used as input.



The below screenshot shows the dual gradient energy of the input image (left) and the seam identified by the “find\_seam” function (right). This seam is obtained as we start evaluating the least energy path from (0, 0) to (m, n) of the 2-D energy array.



The screenshot below shows the dual gradient energy of the transpose of the input image and the seam identified by the “find\_seam” function:

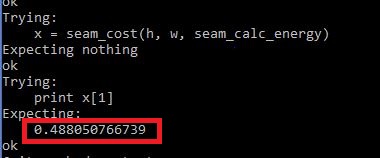


The seams highlighted in the above 2 screenshots indicate the least energy paths with total seam energy equal to

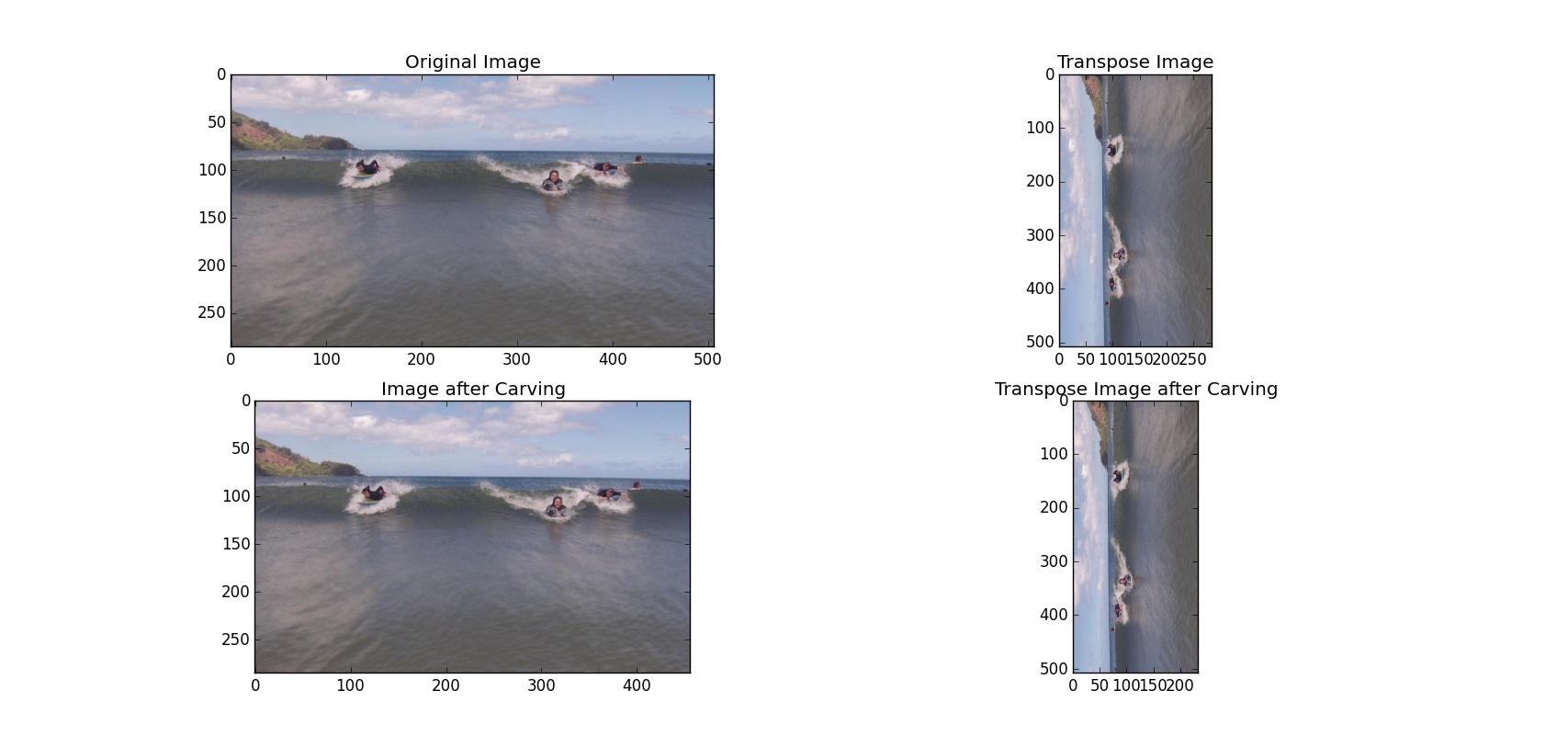
0.488050766739 🡪 for original image

0.038767777001 🡪 for Transpose image

The total seam energy of the original image is shown in the doctest as well:



After removing the least energy seam for 50 times, we obtain the images shown in the below screenshot:

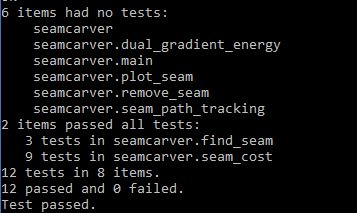


As we can see, the significant information in the image is not lost even after 50 iterations.

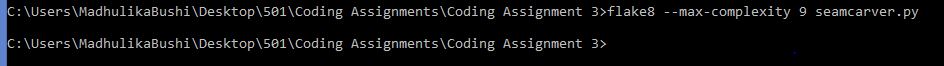
Hence, this implementation is suitable for seam carving of any image.

# 3. Static Analysis / Compilation Output

Doctest output:



Flake8 output: Complexity of the implementation is below 9.



# 4. Source Code

1. """
2. This file provides implementation of Seam Carving.
3. """
4. **import** pylab
5. **from** skimage **import** filters
6. **from** skimage **import** img\_as\_float
7. **import** numpy

10. **def** dual\_gradient\_energy(img):
11. """
12. Dual gradient energy is the sum of the square of a horizontal gradient and a vertical gradient.
13. Use skimage.filter.hsobel and vsobel to calculate the gradients of each channel independently.
14. The energy is the sum of the square the horizontal and vertical gradients over all channels.
15. :param img: input image
16. :return: dual gradient energy of input image
17. """
18. red\_channel = img[:, :, 0]      *# red channel of the image*
19. green\_channel = img[:, :, 1]    *# green channel of the image*
20. blue\_channel = img[:, :, 2]     *# blue channel of the image*
22. horizontal\_gradient\_red = filters.sobel\_h(red\_channel)  *# horizontal gradient of the red channel*
23. vertical\_gradient\_red = filters.sobel\_v(red\_channel)       *# vertical gradient of the red channel*
25. horizontal\_gradient\_green = filters.sobel\_h(green\_channel)  *# horizontal gradient of the green channel*
26. vertical\_gradient\_green = filters.sobel\_v(green\_channel)   *# vertical gradient of the green channel*
28. horizontal\_gradient\_blue = filters.sobel\_h(blue\_channel)  *# horizontal gradient of the blue channel*
29. vertical\_gradient\_blue = filters.sobel\_v(blue\_channel)    *# vertical gradient of the blue channel*
31. *# dual gradient energy at each pixel*
32. energy = (horizontal\_gradient\_red \* horizontal\_gradient\_red)\
33. + (vertical\_gradient\_red \* vertical\_gradient\_red)\
34. + (horizontal\_gradient\_green \* horizontal\_gradient\_green)\
35. + (vertical\_gradient\_green \* vertical\_gradient\_green)\
36. + (horizontal\_gradient\_blue \* horizontal\_gradient\_blue)\
37. + (vertical\_gradient\_blue \* vertical\_gradient\_blue)
39. **return** energy

42. **def** find\_seam(img):
43. """
44. An array of H (number of rows in the image) integers, for each row return the column of the seam.
45. :param img: input image
46. :return: least energy seam to be removed
47. >>> img = pylab.imread('someimage.png')
48. >>> img = img\_as\_float(img)
49. >>> print find\_seam(img)
50. [ 456.  453.  454.  453.  452.  453.  454.  455.  454.  454.  453.  453.
51. 453.  454.  453.  452.  451.  451.  452.  452.  451.  452.  451.  450.
52. 449.  448.  447.  446.  445.  444.  443.  442.  442.  443.  442.  441.
53. 440.  439.  438.  437.  436.  435.  434.  433.  432.  431.  430.  429.
54. 428.  427.  426.  425.  424.  423.  422.  421.  420.  419.  418.  417.
55. 416.  415.  414.  413.  412.  411.  411.  412.  413.  412.  411.  412.
56. 412.  413.  414.  414.  413.  412.  411.  410.  409.  408.  408.  407.
57. 408.  408.  409.  410.  411.  410.  410.  410.  411.  412.  412.  413.
58. 414.  415.  416.  417.  418.  418.  419.  420.  421.  422.  423.  423.
59. 423.  423.  424.  425.  426.  427.  428.  429.  430.  431.  432.  433.
60. 434.  435.  436.  437.  436.  435.  434.  433.  432.  433.  432.  432.
61. 431.  431.  432.  431.  432.  433.  433.  434.  435.  436.  437.  438.
62. 439.  440.  441.  442.  441.  441.  442.  443.  444.  445.  446.  447.
63. 448.  449.  450.  451.  452.  453.  454.  455.  456.  455.  454.  453.
64. 452.  452.  453.  454.  453.  452.  451.  450.  449.  449.  450.  450.
65. 451.  452.  451.  450.  450.  450.  451.  451.  450.  451.  452.  453.
66. 454.  455.  454.  453.  454.  454.  455.  454.  453.  454.  455.  456.
67. 457.  458.  458.  457.  456.  455.  454.  454.  453.  454.  454.  455.
68. 455.  454.  453.  453.  453.  453.  453.  452.  453.  453.  452.  451.
69. 450.  450.  451.  451.  451.  451.  451.  450.  449.  450.  449.  450.
70. 451.  452.  451.  450.  450.  449.  450.  451.  451.  451.  451.  452.
71. 453.  454.  454.  454.  453.  452.  451.  450.  450.  450.  450.  449.
72. 448.  447.  446.  445.  446.  447.  446.  446.  447.  448.  449.  450.
73. 451.  452.  453.  454.  455.  455.  456.  456.  457.]
74. """
75. h, w = img.shape[:2]                            *# h - rows, w - columns*
76. dg\_energy = dual\_gradient\_energy(img)           *# get the dual gradient energy of the image*
77. seam\_calc\_energy = numpy.zeros(shape=(h, w))    *# holding sum of the energies till that row for each pixel*
78. seam = numpy.zeros(shape=h)                     *# actual seam that can be removed*
79. numpy.copyto(seam\_calc\_energy, dg\_energy)       *# initializing with dual gradient energy as default*
81. seam\_path, seam\_calc\_energy = seam\_path\_tracking(h, w, seam\_calc\_energy)
82. index\_min\_energy = seam\_cost(h, w, seam\_calc\_energy)
84. index = index\_min\_energy[0]
85. seam[0] = index
87. **for** i **in** range(h - 1, 0, -1):                   *# gathering the least energy pixel path*
88. index = seam\_path[i][index]
89. seam[i] = index
91. **return** seam

94. **def** seam\_path\_tracking(h, w, seam\_calc\_energy):
95. """
96. Getting the path chosen by each pixel from the first row
97. :param h: Height (number of rows)
98. :param w: Width (number of columns)
99. :param seam\_calc\_energy: sum of the energies till the selected row for each pixel
100. :return: seam\_path for each pixel
101. """
102. seam\_path = numpy.zeros(shape=(h, w))           *# tracking the choice*
103. **for** j **in** range(0, w):                           *# initializing the seam path*
104. seam\_path[0][j] = j
106. **for** i **in** range(1, h):                           *# computing the least energy path*
107. **for** j **in** range(1, w - 1):
108. center\_seam = seam\_calc\_energy[i - 1][j]
109. **if** j == 1:                                  *# boundary case*
110. left\_seam = float('inf')
111. right\_seam = seam\_calc\_energy[i - 1][j + 1]
112. **elif** j == (w - 2):                          *# boundary case*
113. left\_seam = seam\_calc\_energy[i - 1][j - 1]
114. right\_seam = float('inf')
115. **else**:                                       *# all other cases*
116. left\_seam = seam\_calc\_energy[i - 1][j - 1]
117. right\_seam = seam\_calc\_energy[i - 1][j + 1]
119. *# tracking the pixel position to identify the choice from the previous row*
120. **if** left\_seam <= right\_seam **and** left\_seam <= center\_seam:
121. seam\_calc\_energy[i][j] = seam\_calc\_energy[i][j] + left\_seam
122. seam\_path[i][j] = j - 1
123. **elif** center\_seam <= left\_seam **and** center\_seam <= right\_seam:
124. seam\_calc\_energy[i][j] = seam\_calc\_energy[i][j] + center\_seam
125. seam\_path[i][j] = j
126. **elif** right\_seam <= center\_seam **and** right\_seam <= left\_seam:
127. seam\_calc\_energy[i][j] = seam\_calc\_energy[i][j] + right\_seam
128. seam\_path[i][j] = j + 1
129. **return** seam\_path, seam\_calc\_energy

132. **def** seam\_cost(h, w, seam\_calc\_energy):
133. """
134. Getting the index at the top row for which the energy path is least
135. :param h: height of image
136. :param w: width of image
137. :param seam\_calc\_energy: sum of the energies till the selected row for each pixel
138. :return: index at the top row of the least energy path
139. >>> img = pylab.imread('someimage.png')
140. >>> img = img\_as\_float(img)
141. >>> h, w = img.shape[:2]
142. >>> seam\_calc\_energy = numpy.zeros(shape=(h, w))
143. >>> dg\_energy = dual\_gradient\_energy(img)
144. >>> numpy.copyto(seam\_calc\_energy, dg\_energy)
145. >>> seam\_path, seam\_calc\_energy = seam\_path\_tracking(h, w, seam\_calc\_energy)
146. >>> x = seam\_cost(h, w, seam\_calc\_energy)
147. >>> print x[1]
148. 0.488050766739
149. """
150. minimum\_energy = float('inf')
151. **for** i **in** range(1, w - 1):                       *# checking the last row to identify least energy pixel path*
152. **if** seam\_calc\_energy[h - 1][i] < minimum\_energy:
153. minimum\_energy = seam\_calc\_energy[h - 1][i]
154. index = i
155. **return** index, minimum\_energy

158. **def** plot\_seam(img, seam):
159. """
160. Visualization of the seam, img, and energy func.
161. :param img: input image
162. :param seam: seam identified for the image
163. :return: NA
164. """
165. h, w = img.shape[:2]                        *# h - rows, w - columns*
166. dg\_energy1 = dual\_gradient\_energy(img)      *# get the dual gradient energy of the image*
167. dg\_energy2 = numpy.zeros(shape=(h, w))
168. numpy.copyto(dg\_energy2, dg\_energy1)
169. pylab.figure()
170. pylab.gray()
171. pylab.subplot(1, 3, 1)
172. pylab.imshow(img)                           *# plot original image*
173. pylab.title("Original Image")
174. pylab.subplot(1, 3, 2)
175. pylab.imshow(dg\_energy1)
176. pylab.title("Dual Gradient Energy")         *# plot dual gradient energy*
178. **for** i **in** range(0, h):                       *# highlighting the seam*
179. dg\_energy2[i][seam[i]] = 2
180. pylab.subplot(1, 3, 3)
181. pylab.imshow(dg\_energy2)      *# plot dual gradient energy with the identified seam*
182. pylab.title("Dual gradient with Seam")
184. pylab.show()

187. **def** remove\_seam(img, seam):
188. """
189. Modify img in-place and return a W-1 x H x 3 slice
190. :param img: input image
191. :param seam: seam identified for the image
192. :return: image after removing the seam
193. """
194. h, w = img.shape[:2]                   *# h - rows, w - columns*
196. **for** i **in** range(0, h):                  *# moving all the columns to the right*
197. width\_position = seam[i]
198. img[i, 1:width\_position + 1, :] = img[i, 0:width\_position, :]
200. **return** numpy.delete(img, 0, 1)         *# deleting the empty column after shifting*

203. **def** main():
204. img = pylab.imread('someimage.png')         *# getting the image*
205. transpose\_img = img.transpose(1, 0, 2)      *# getting the transpose image*
206. img = img\_as\_float(img)
207. transpose\_img = img\_as\_float(transpose\_img)
209. seam = find\_seam(img)                       *# find seam*
210. transpose\_seam = find\_seam(transpose\_img)   *# find transpose seam*
212. plot\_seam(img, seam)                        *# plot seam*
213. plot\_seam(transpose\_img, transpose\_seam)    *# plot transpose seam*
215. pylab.figure()
217. pylab.subplot(2, 2, 1)
218. pylab.imshow(img)                      *# plot original image*
219. pylab.title("Original Image")
221. h, w = img.shape[:2]
222. **print** 'original image dimensions: W = ' + str(w) + ' H = ' + str(h)
224. pylab.subplot(2, 2, 2)
225. pylab.imshow(transpose\_img)            *# plot transpose of original image*
226. pylab.title("Transpose Image")
228. h, w = transpose\_img.shape[:2]
229. **print** 'Transpose image dimensions: W = ' + str(w) + ' H = ' + str(h)
231. removed\_img = remove\_seam(img, seam)
232. **for** i **in** range(0, 49):                 *# image after removing 50 seams*
233. seam = find\_seam(removed\_img)
234. removed\_img = remove\_seam(removed\_img, seam)
235. pylab.subplot(2, 2, 3)
236. pylab.imshow(removed\_img)              *# plot original image after carving 50 times*
237. pylab.title("Image after Carving")
239. h, w = removed\_img.shape[:2]
240. **print** 'After carving image dimensions: W = ' + str(w) + ' H = ' + str(h)
242. removed\_transpose\_img = remove\_seam(transpose\_img, transpose\_seam)
243. **for** i **in** range(0, 49):                 *# transpose image after removing 50 seams*
244. transpose\_seam = find\_seam(removed\_transpose\_img)
245. removed\_transpose\_img = remove\_seam(removed\_transpose\_img, transpose\_seam)
246. pylab.subplot(2, 2, 4)
247. pylab.imshow(removed\_transpose\_img)    *# plot original image after carving 50 times*
248. pylab.title("Transpose Image after Carving")
250. h, w = removed\_transpose\_img.shape[:2]
251. **print** 'After carving transpose image dimensions: W = ' + str(w) + ' H = ' + str(h)
253. pylab.show()

256. **if** \_\_name\_\_ == '\_\_main\_\_':
257. main()

Revised rubric for coding assignments.

This is a 5-point rubric for coding projects. Graders should refrain from using fractional points (they are a pain to defend), choose the one one number that best reflects the assignment. For assignments with multiple parts, choose the lowest scoring part.

This rubric is based on the idea that students submit PDF write-ups with their coding assignment. Write-ups *must* be PDF’s with the source code so that graders can quickly view them annotate them using blackboard. The rubric does not address specific learning objectives — the assumption is that by completing the assignment the student has implicitly demonstrated some set objectives in addition to coding.

0 points — Student does not submit **all** parts of the assignment, meaning *both* a **PDF** writeup (all sections) that includes source code and output of testing, as well as a .**zip** file with source code.

2 points — The code does not run or does not *appear* to be able to run. The code it much longer than it should be, or does not appear to follow the scaffolding provided. The grader can but **does not have to verify that it does not run**, it is the student’s responsibility to provide a writeup that is sufficiently convincing. Student may not appeal by coming after the fact and showing that code runs on their machine.   
When grading, the grader should indicate portions of the code by annotating the writeup that are suspicious.

3 points — The code runs or looks like it would run, but the student has not shown via their writeup that it produces the correct result on reasonable inputs. Or, the student has implemented algorithms using approaches other than the ones indicated in the assignment, or the implementation has the wrong asymptotic complexity or that demonstrates a lack of understanding of the assignments objectives. The grader can, but **does not have to run the code to verify correctness** — it is the student’s responsibility to make a convincing case that the output and the algorithm is correct.   
When grading, the grader should indicate by annotating the write-up where results

4 points — The code runs or appears to run correctly, but has readability or style issues. The student has not demonstrated that their code has passed style guidelines, or the student's implementation appears to be unnecessarily complex (even though it looks like it works).   
When grading, the grader should indicate the style problems.

5 points — No issues that we can spot.