

# Contents

1	Basic Test Results	2
2	README	5
3	answer q1.txt	6
4	answer q2.txt	7
5	answer q3.txt	8
6	sol3.py	9
7	blend/givat2.jpg	13
8	blend/mask1.jpg	14
9	blend/mask2.jpg	15
10	blend/race2.jpg	16
11	blend/stat1.jpg	17
12	blend/trump1.jpg	18

# 1 Basic Test Results

```
1 Archive: /tmp/bodek.pPLY7U/impr/ex3/liavst2/presubmission/submission
2   inflating: current/README
3   inflating: current/answer_q1.txt
4   inflating: current/answer_q2.txt
5   inflating: current/answer_q3.txt
6   creating: current/blend/
7   inflating: current/blend/givat2.jpg
8   inflating: current/blend/mask1.jpg
9   inflating: current/blend/mask2.jpg
10  inflating: current/blend/race2.jpg
11  inflating: current/blend/stat1.jpg
12  inflating: current/blend/trump1.jpg
13  inflating: current/sol3.py
14 ex3 presubmission script
15
16     Disclaimer
17     -----
18     The purpose of this script is to make sure that your code is compliant
19     with the exercise API and some of the requirements
20     The script does not test the quality of your results.
21     Don't assume that passing this script will guarantee that you will get
22     a high grade in the exercise
23
24 === Check Submission ===
25
26 login: liavst2
27
28 submitted files:
29
30 =====
31 = README for ex3: Image Pyramids & Pyramid Blending =
32 =====
33
34
35 =====
36 = List Of Submitted Files =
37 =====
38
39 - README - this file.
40 - answer_q1.txt
41 - answer_q2.txt
42 - answer_q3.txt
43 - sol3.py - contains the implementation of all function
44 - blend - directory which contains the images for examples 1,2
45 - stat1.jpg
46 - trump1.jpg      =>    for blending_example1
47 - mask1.jpg
48
49
50 - race2.jpg
51 - givat2.jpg      =>    for blending_example2
52 - mask2. jpg
53
54 === Answers to questions ===
55
56 Answer to Q1:
57
58 Answer to Q1:
59 =====
```

```

60
61 Laplacian pyramid is like a band pass filter. So multiplying each
62 level with a different value allows us to highlight certain
63 sections in the frequency domain, meaning that we try to control
64 on certain sections in the frequency domain.
65
66 Answer to Q2:
67
68 Answer to Q2:
69 =====
70
71 The bigger the gaussian filter, the better the blending is in
72 the environment. The effect is noticed only along the edges of
73 the mask, so other parts of both images remain the same.
74 For bigger filter, we have less high frequencies remaining so
75 we get spreader combination of the pixels. Thus the edges of
76 the mask are more difficult to identify (As you'll see in
77 my blending_example1)
78
79 Answer to Q3:
80
81 Answer to Q3:
82 =====
83
84 The more levels of the pyramid, the better blending in the environment.
85 As we go downwards the pyramid we get more low frequencies participating
86 in the reconstruction procedure, which makes the blending more softer
87 and absorb the edges in the environment more efficiently.
88
89 === Section 3.1 ===
90
91 Trying to build Gaussian pyramid...
92     Passed!
93 Checking Gaussian pyramid type and structure...
94     Passed!
95 Trying to build Laplacian pyramid...
96     Passed!
97 Checking Laplacian pyramid type and structure...
98     Passed!
99
100 === Section 3.2 ===
101
102 Trying to build Laplacian pyramid...
103     Passed!
104 Trying to reconstruct image from pyramid... (we are not checking for quality!)
105     Passed!
106 Checking reconstructed image type and structure...
107     Passed!
108
109 === Section 3.3 ===
110
111 Trying to build Gaussian pyramid...
112     Passed!
113 Trying to render pyramid to image...
114     Passed!
115 Checking structure of returned image...
116     Passed!
117 Trying to display image... (if DISPLAY env var not set, assumes running w/o screen)
118     Passed!
119
120 === Section 4 ===
121
122 Trying to blend two images... (we are not checking the quality!)
123     Passed!
124 Checking size of blended image...
125     Passed!
126 Tring to call blending_example1()...
127     Passed!

```

```
128 Checking types of returned results...
129     Passed!
130 Tring to call blending_example2()...
131     Passed!
132 Checking types of returned results...
133     Passed!
134
135 === All tests have passed ===
136 === Pre-submission script done ===
137
138
139     Please go over the output and verify that there are no failures/warnings.
140     Remember that this script tested only some basic technical aspects of your implementation
141     It is your responsibility to make sure your results are actually correct and not only
142     technically valid.
```

## 2 README

```
1  liavst2
2
3  =====
4  = README for ex3: Image Pyramids & Pyramid Blending =
5  =====
6
7
8  =====
9  = List Of Submitted Files =
10 =====
11
12
13 - README - this file.
14 - answer_q1.txt
15 - answer_q2.txt
16 - answer_q3.txt
17 - sol3.py - contains the implementation of all function
18 - blend - directory which contains the images for examples 1,2
19     - stat1.jpg
20     - trump1.jpg    =>    for blending_example1
21     - mask1.jpg
22
23     - race2.jpg
24     - givat2.jpg    =>    for blending_example2
25     - mask2. jpg
```

### 3 answer q1.txt

```
1 Answer to Q1:
2 =====
3
4 Laplacian pyramid is like a band pass filter. So multiplying each
5 level with a different value allows us to highlight certain
6 sections in the frequency domain, meaning that we try to control
7 on certain sections in the frequency domain.
```

## 4 answer q2.txt

```
1 Answer to Q2:
2 =====
3
4 The bigger the gaussian filter, the better the blending is in
5 the environment. The effect is noticed only along the edges of
6 the mask, so other parts of both images remain the same.
7 For bigger filter, we have less high frequencies remaining so
8 we get spreader combination of the pixels. Thus the edges of
9 the mask are more difficult to identify (As you'll see in
10 my blending_example1)
```

## 5 answer q3.txt

```
1 Answer to Q3:
2 =====
3
4 The more levels of the pyramid, the better blending in the environment.
5 As we go downwards the pyramid we get more low frequencies participating
6 in the reconstruction procedure, which makes the blending more softer
7 and absorb the edges in the environment more efficiently.
```



## 6 sol3.py

```
1 #####
2 # FILE: sol3.py
3 # WRITER: Liav Steinberg
4 # EXERCISE : Image Processing ex3
5 #####
6
7 import os
8 import numpy as np
9 from scipy import signal as sg
10 from scipy.misc import imread
11 from skimage.color import rgb2gray
12 from scipy.ndimage import filters as flt
13 from matplotlib import pyplot as plt
14
15
16 # -----helpers-----#
17
18
19 def read_image(filename, representation):
20     #####
21     # reads an image with the given representation
22     #####
23     image = imread(relpath(filename)).astype(np.float32) / 255
24     return image if representation == 2 else rgb2gray(image)
25
26
27 def relpath(filename):
28     #####
29     # returns the relative path of the image
30     #####
31     return os.path.join(os.path.dirname(__file__), filename)
32
33
34 def create_gauss_filter(k_size):
35     #####
36     # Helper function to calculate gaussian filter_vec
37     #####
38     base = filter_vec = np.array([[1, 1]])
39     for i in range(2, k_size):
40         filter_vec = sg.convolve2d(filter_vec, base).astype(np.float32)
41     # normalize the filter_vec
42     filter_vec /= np.sum(filter_vec)
43     return filter_vec
44
45
46 def stretch_values(pyr_element):
47     #####
48     # stretching pyramid values to [0,1] before displaying
49     #####
50     minimum = np.min(pyr_element)
51     maximum = np.max(pyr_element)
52     range_ = maximum - minimum
53     return 1 - ((maximum - pyr_element) / range_)
54
55
56 def reduce(im, filt):
57     #####
58     # shrinks image by a factor of 1/2
59     #####
```

```

60     red = flt.convolve(flt.convolve(im, filt), filt.reshape(filt.size, 1))
61     return red[:, :, 2]
62
63
64 def expand(im, filt):
65     #####
66     # expand image by a factor of 2
67     #####
68     exp = np.zeros((im.shape[0] * 2, im.shape[1] * 2), dtype=np.float32)
69     exp[:, ::2, ::2] = im[:, :]
70     return flt.convolve(flt.convolve(exp, 2 * filt), 2 * filt.reshape(filt.size, 1))
71
72
73 def blend_images(im1, im2, mask, flt_size, pyr_size):
74     #####
75     # displays the blending result
76     #####
77     R1, G1, B1 = im1[:, :, 0], im1[:, :, 1], im1[:, :, 2]
78     R2, G2, B2 = im2[:, :, 0], im2[:, :, 1], im2[:, :, 2]
79     R = pyramid_blending(R2, R1, mask, pyr_size, flt_size, flt_size).astype(np.float32)
80     G = pyramid_blending(G2, G1, mask, pyr_size, flt_size, flt_size).astype(np.float32)
81     B = pyramid_blending(B2, B1, mask, pyr_size, flt_size, flt_size).astype(np.float32)
82
83     im_blend = np.zeros_like(im1)
84     im_blend[:, :, 0] += R
85     im_blend[:, :, 1] += G
86     im_blend[:, :, 2] += B
87
88     # plotting the images
89     f, ax = plt.subplots(2, 2)
90
91     ax[0, 0].imshow(im1)
92     ax[0, 0].set_title('image 1')
93     ax[0, 1].imshow(im2)
94     ax[0, 1].set_title('image 2')
95     ax[1, 0].imshow(mask, 'gray')
96     ax[1, 0].set_title('mask')
97     ax[1, 1].imshow(im_blend)
98     ax[1, 1].set_title('result')
99     plt.show()
100    return im1, im2, mask, im_blend
101
102
103 # -----3.1-----#
104
105 def build_gaussian_pyramid(im, max_levels, filter_size):
106     #####
107     # returns an array representing gaussian pyramid built
108     # by a gaussian filter
109     #####
110     gauss_pyr = [im]
111     filter_vec = create_gauss_filter(filter_size)
112     for i in range(max_levels-1):
113         im = reduce(im, filter_vec)
114         # if the image has exceeded the legal size, stop
115         if im.shape[0] < 16 or im.shape[1] < 16:
116             break
117         gauss_pyr.append(im)
118     return gauss_pyr, filter_vec
119
120
121 def build_laplacian_pyramid(im, max_levels, filter_size):
122     #####
123     # returns an array representing laplacian pyramid built
124     # by the algorithm from the tirgul
125     #####
126     gauss_pyr, filter_vec = build_gaussian_pyramid(im, max_levels, filter_size)
127     lapl_pyr = []

```

```

128     for i in range(max_levels-1):
129         curr = gauss_pyr[i]
130         exp_curr = expand(gauss_pyr[i+1], filter_vec)
131         # check images sizes before subtracting
132         if exp_curr.shape[0] > curr.shape[0]:
133             exp_curr = np.delete(exp_curr, -1, axis=0)
134         if exp_curr.shape[1] > curr.shape[1]:
135             exp_curr = np.delete(exp_curr, -1, axis=1)
136         # adding to the pyramid
137         lapl_pyr.append(curr - exp_curr)
138     # add the last gauss pyramid element
139     lapl_pyr.append(gauss_pyr[-1])
140     return lapl_pyr, filter_vec
141
142
143 # -----3.2-----#
144
145 def laplacian_to_image(lpyr, filter_vec, coeff):
146     #####
147     # constructs the original image from its laplacian
148     # pyramid
149     #####
150     img = np.zeros_like(lpyr[-1])
151     correct_shape = lpyr[0].shape
152     for mat, co in zip(reversed(lpyr), reversed(coeff)):
153         # adjustments before adding the matrices
154         if img.shape[0] > mat.shape[0]:
155             img = np.delete(img, -1, axis=0)
156         if img.shape[1] > mat.shape[1]:
157             img = np.delete(img, -1, axis=1)
158         img += mat * co
159         img = expand(img, filter_vec) if \
160             img.shape != correct_shape else img
161     return img
162
163
164 # -----3.3-----#
165
166 def render_pyramid(pyr, levels):
167     #####
168     # calculates the height and width of the image where
169     # the pyramid will be displayed
170     #####
171     # calculating height and width of the image
172     height = pyr[0].shape[0]
173     cols = float(pyr[0].shape[1])
174     width = 0
175     for i in range(levels):
176         width += cols
177         cols = float(np.ceil(cols/2))
178     res = np.zeros((height, int(width)))
179     # rendering the pyramid
180     Xbegin_pos, Xend_pos = 0, 0
181     for i in range(levels):
182         Xend_pos = pyr[i].shape[1]
183         Ypos = pyr[i].shape[0]
184         # set the image in its appropriate place in the pyramid
185         res[0:Ypos, Xbegin_pos:Xbegin_pos + Xend_pos] += stretch_values(pyr[i])
186         # advancing the starting position of the next image
187         Xbegin_pos += Xend_pos
188     return res
189
190
191 def display_pyramid(pyr, levels):
192     #####
193     # displays the pyramid of a given image, amount of
194     # levels deep
195     #####

```

```

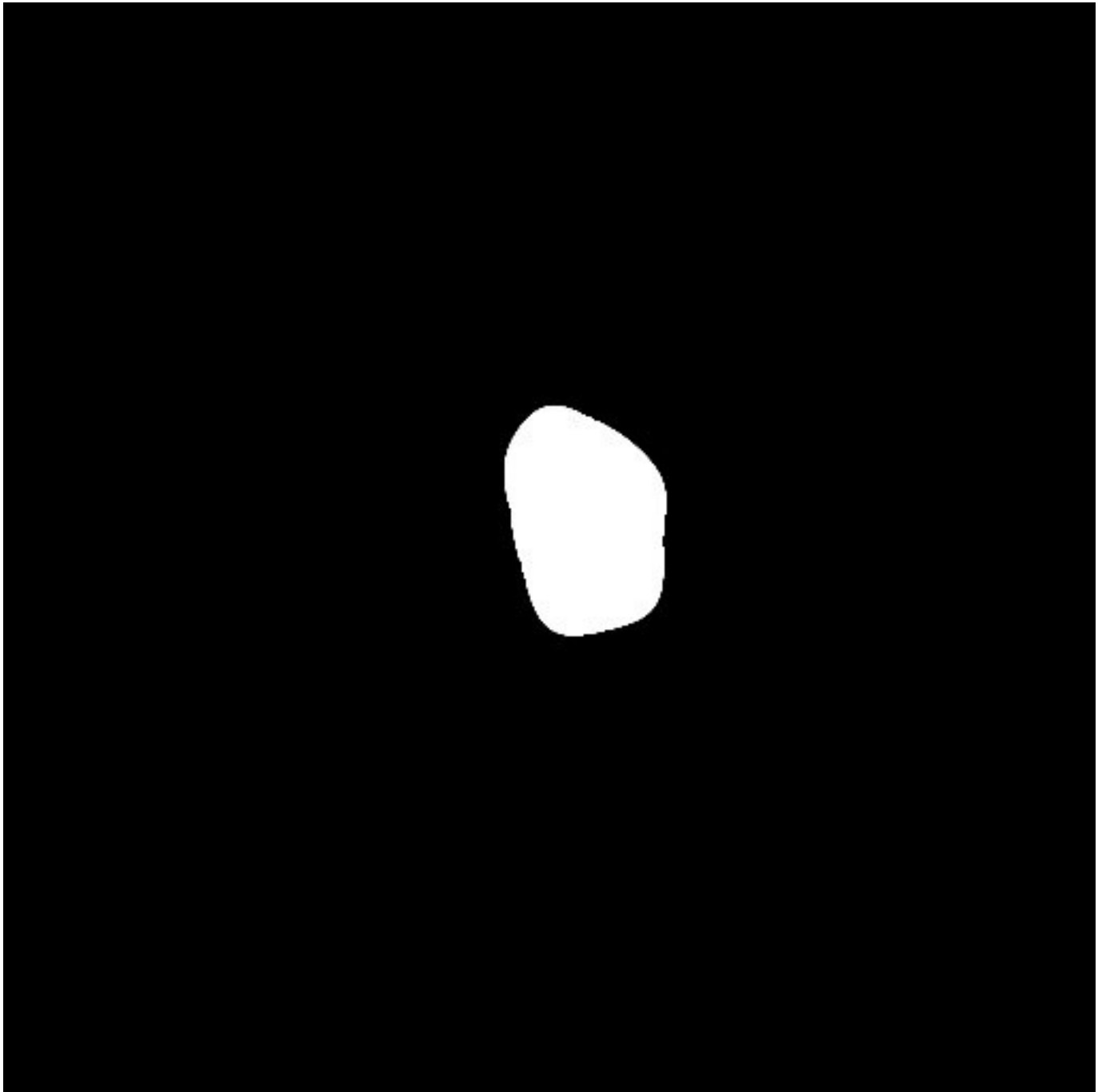
196     res = render_pyramid(pyr, levels)
197     plt.figure()
198     plt.imshow(res, 'gray')
199     plt.show()
200
201
202 # -----4-----#
203
204 def pyramid_blending(im1, im2, mask, max_levels, filter_size_im, filter_size_mask):
205     #####
206     # blends two images according to a given mask
207     #####
208     L1, filt1 = build_laplacian_pyramid(im1, max_levels, filter_size_im)
209     L2, filt1 = build_laplacian_pyramid(im2, max_levels, filter_size_im)
210     G, filt2 = build_gaussian_pyramid(mask.astype(np.float32), max_levels, filter_size_mask)
211     Lout = []
212     for i in range(max_levels):
213         curr = (G[i] * L1[i]) + ((1.0 - G[i]) * L2[i])
214         Lout.append(curr)
215     return np.clip(laplacian_to_image(Lout, filt1, np.ones(len(Lout))), 0, 1)
216
217
218 # -----4.1-----#
219
220 def blending_example1():
221     #####
222     # example 1
223     #####
224     im1 = read_image(relpath('blend/stat1.jpg'), 2).astype(np.float32)
225     im2 = read_image(relpath('blend/trump1.jpg'), 2).astype(np.float32)
226     mask = read_image(relpath('blend/mask1.jpg'), 1)
227     # some adjustments on the mask to disable dirt along the edges
228     mask[mask > 0.5] = 1
229     mask[mask <= 0.5] = 0
230     mask = mask.astype(np.bool)
231     return blend_images(im1, im2, mask, 35, 6)
232
233
234 def blending_example2():
235     #####
236     # example 2
237     #####
238     im1 = read_image(relpath('blend/givat2.jpg'), 2).astype(np.float32)
239     im2 = read_image(relpath('blend/race2.jpg'), 2).astype(np.float32)
240     mask = read_image(relpath('blend/mask2.jpg'), 1)
241     # some adjustments on the mask to disable dirt along the edges
242     mask[mask > 0.5] = 1
243     mask[mask <= 0.5] = 0
244     mask = mask.astype(np.bool)
245     return blend_images(im1, im2, mask, 35, 1)
246
247
248 # -----end-----#

```

## 7 blend/givat2.jpg



## 8 blend/mask1.jpg

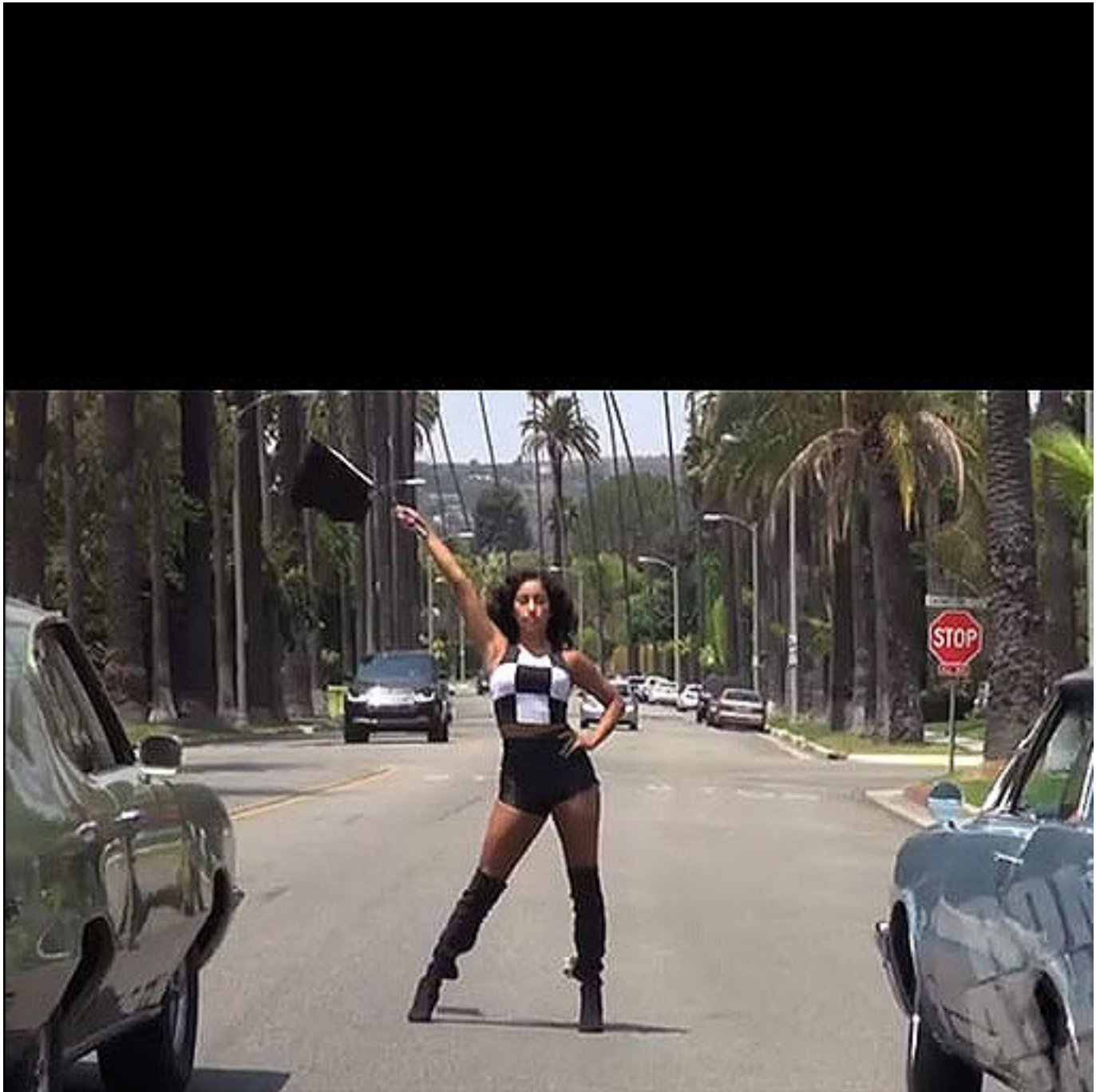


## 9 blend/mask2.jpg





10 blend/race2.jpg





11 blend/stat1.jpg



12 blend/trump1.jpg

