Contents

1 Basic Test Results

```
Archive:  /tmp/bodek.pPLy7U/impr/ex3/liavst2/presubmission/submission
  inflating: current/README
  inflating: current/answer_q1.txt
  inflating: current/answer_q2.txt
  inflating: current/answer_q3.txt
   creating: current/blend/
  inflating: current/blend/givat2.jpg
  inflating: current/blend/mask1.jpg
  inflating: current/blend/mask2.jpg
  inflating: current/blend/race2.jpg
  inflating: current/blend/stat1.jpg
  inflating: current/blend/trump1.jpg
  inflating: current/sol3.py
ex3 presubmission script

    Disclaimer
    ----------
    The purpose of this script is to make sure that your code is compliant
    with the exercise API and some of the requirements
    The script does not test the quality of your results.
    Don't assume that passing this script will guarantee that you will get
    a high grade in the exercise

=== Check Submission ===

login:  liavst2

submitted files:

=====================================================
= README for ex3: Image Pyramids & Pyramid Blending =
=====================================================


==========================
= List Of Submitted Files =
==========================


- README - this file.
- answer_q1.txt
- answer_q2.txt
- answer_q3.txt
- sol3.py - contains the implementation of all function
- blend - directory which contains the images for examples 1,2
- stat1.jpg
- trump1.jpg     =>     for blending_example1
- mask1.jpg

- race2.jpg
- givat2.jpg     =>     for blending_example2
- mask2. jpg

=== Answers to questions ===

Answer to Q1:

Answer to Q1:
============
```

```

Laplacian pyramid is like a band pass filter. So multiplying each
level with a different value allows us to highlight certain
sections in the frequency domain, meaning that we try to control
on certain sections in the frequency domain.

Answer to Q2:

Answer to Q2:
============

The bigger the gaussian filter, the better the blending is in
the environment. The effect is noticed only along the edges of
the mask, so other parts of both images remain the same.
For bigger filter, we have less high frequencies remaining so
we get spreader combination of the pixels. Thus the edges of
the mask are more difficult to identify (As you'll see in
my blending_example1)

Answer to Q3:

Answer to Q3:
============

The more levels of the pyramid, the better blending in the environment.
As we go downwards the pyramid we get more low frequencies participating
in the reconstruction procedure, which makes the blending more softer
and absorb the edges in the environment more efficiently.

=== Section 3.1 ===

Trying to build Gaussian pyramid...
    Passed!
Checking Gaussian pyramid type and structure...
    Passed!
Trying to build Laplacian pyramid...
    Passed!
Checking Laplacian pyramid type and structure...
    Passed!

=== Section 3.2 ===

Trying to build Laplacian pyramid...
    Passed!
Trying to reconstruct image from pyramid... (we are not checking for quality!)
    Passed!
Checking reconstructed image type and structure...
    Passed!

=== Section 3.3 ===

Trying to build Gaussian pyramid...
    Passed!
Trying to render pyramid to image...
    Passed!
Checking structure of returned image...
    Passed!
Trying to display image... (if DISPLAY env var not set, assumes running w/o screen)
    Passed!

=== Section 4 ===

Trying to blend two images... (we are not checking the quality!)
    Passed!
Checking size of blended image...
    Passed!
Tring to call blending_example1()...
    Passed!
```

```
Checking types of returned results...
    Passed!
Tring to call blending_example2()...
    Passed!
Checking types of returned results...
    Passed!

=== All tests have passed ===
=== Pre-submission script done ===


    Please go over the output and verify that there are no failures/warnings.
    Remember that this script tested only some basic technical aspects of your implementation
    It is your responsibility to make sure your results are actually correct and not only
    technically valid.
```

2 README

```
liavst2

=====================================================
= README for ex3: Image Pyramids & Pyramid Blending =
=====================================================


===========================
= List Of Submitted Files =
===========================


- README - this file.
- answer_q1.txt
- answer_q2.txt
- answer_q3.txt
- sol3.py - contains the implementation of all function
- blend - directory which contains the images for examples 1,2
    - stat1.jpg
    - trump1.jpg     =>     for blending_example1
    - mask1.jpg

    - race2.jpg
    - givat2.jpg     =>     for blending_example2
    - mask2. jpg
```

3 answer q1.txt

```
Answer to Q1:
============

Laplacian pyramid is like a band pass filter. So multiplying each
level with a different value allows us to highlight certain
sections in the frequency domain, meaning that we try to control
on certain sections in the frequency domain.
```

4 answer q2.txt

```
Answer to Q2:
============

The bigger the gaussian filter, the better the blending is in
the environment. The effect is noticed only along the edges of
the mask, so other parts of both images remain the same.
For bigger filter, we have less high frequencies remaining so
we get spreader combination of the pixels. Thus the edges of
the mask are more difficult to identify (As you’ll see in
my blending_example1)
```

5 answer q3.txt

```
Answer to Q3:
============

The more levels of the pyramid, the better blending in the environment.
As we go downwards the pyramid we get more low frequencies participating
in the reconstruction procedure, which makes the blending more softer
and absorb the edges in the environment more efficiently.
```

6 sol3.py

```python
##################################################################
# FILE: sol3.py
# WRITER: Liav Steinberg
# EXERCISE : Image Processing ex3
##################################################################

import os
import numpy as np
from scipy import signal as sg
from scipy.misc import imread
from skimage.color import rgb2gray
from scipy.ndimage import filters as flt
from matplotlib import pyplot as plt


# -------------------------helpers----------------------------#


def read_image(filename, representation):
    #######################################################
    # reads an image with the given representation
    #######################################################
    image = imread(relpath(filename)).astype(np.float32) / 255
    return image if representation == 2 else rgb2gray(image)


def relpath(filename):
    #######################################################
    # returns the relative path of the image
    #######################################################
    return os.path.join(os.path.dirname(__file__), filename)


def create_gauss_filter(k_size):
    ########################################################
    # Helper function to calculate gaussian filter_vec
    ########################################################
    base = filter_vec = np.array([[1, 1]])
    for i in range(2, k_size):
        filter_vec = sg.convolve2d(filter_vec, base).astype(np.float32)
    # normalize the filter_vec
    filter_vec /= np.sum(filter_vec)
    return filter_vec


def stretch_values(pyr_element):
    #######################################################
    # stretching pyramid values to [0,1] before displaying
    #######################################################
    minimum = np.min(pyr_element)
    maximum = np.max(pyr_element)
    range_ = maximum - minimum
    return 1 - ((maximum - pyr_element) / range_)


def reduce(im, filt):
    #######################################################
    # shrinks image by a factor of 1/2
    #######################################################
```

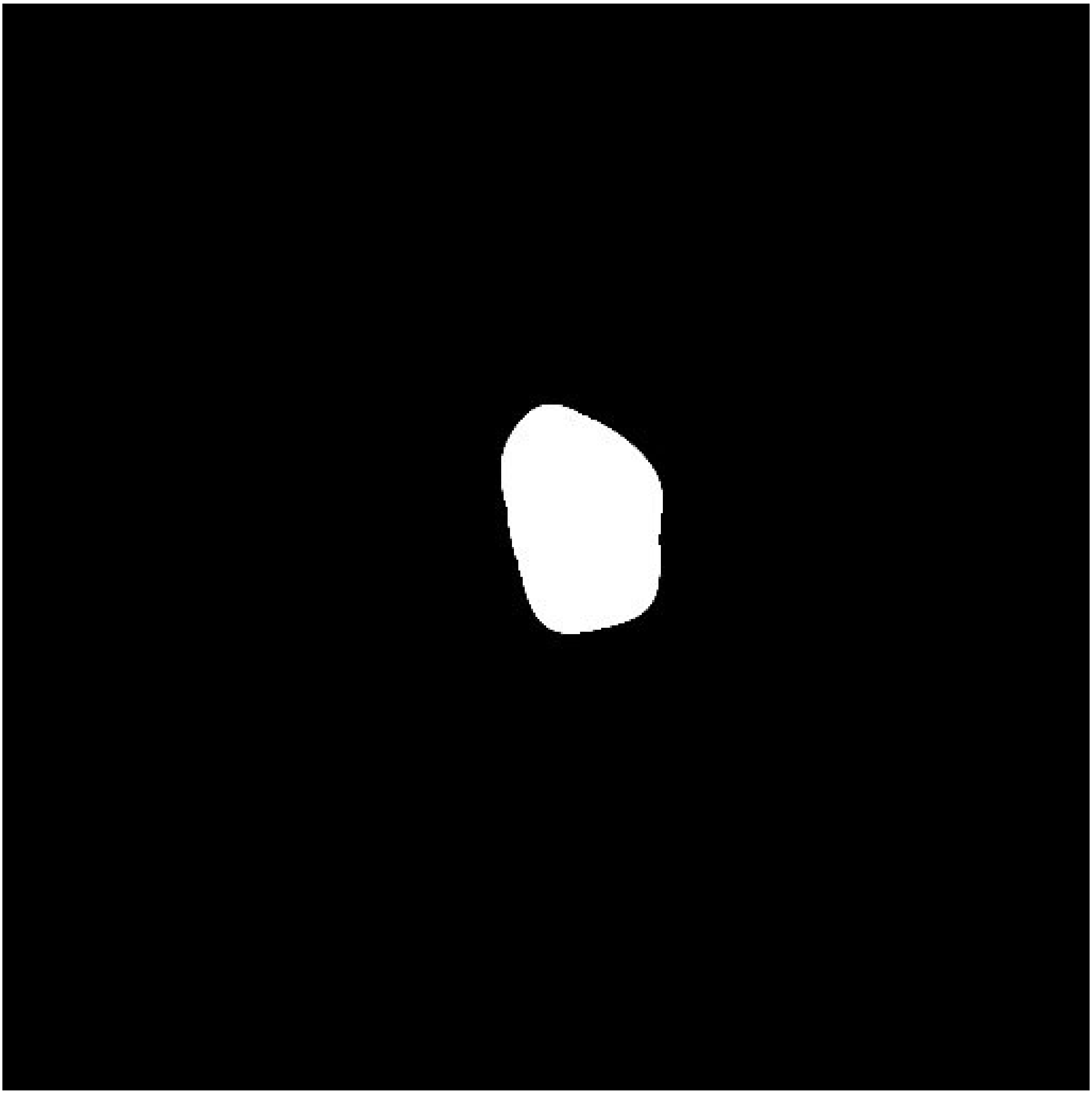
```python
    red = flt.convolve(flt.convolve(im, filt), filt.reshape(filt.size, 1))
    return red[::2, ::2]


def expand(im, filt):
    ##########################################################
    # expand image by a factor of 2
    ##########################################################
    exp = np.zeros((im.shape[0] * 2, im.shape[1] * 2), dtype=np.float32)
    exp[::2, ::2] = im[:, :]
    return flt.convolve(flt.convolve(exp, 2 * filt), 2 * filt.reshape(filt.size, 1))


def blend_images(im1, im2, mask, flt_size, pyr_size):
    ##########################################################
    # displays the blending result
    ##########################################################
    R1, G1, B1 = im1[:, :, 0], im1[:, :, 1], im1[:, :, 2]
    R2, G2, B2 = im2[:, :, 0], im2[:, :, 1], im2[:, :, 2]
    R = pyramid_blending(R2, R1, mask, pyr_size, flt_size, flt_size).astype(np.float32)
    G = pyramid_blending(G2, G1, mask, pyr_size, flt_size, flt_size).astype(np.float32)
    B = pyramid_blending(B2, B1, mask, pyr_size, flt_size, flt_size).astype(np.float32)

    im_blend = np.zeros_like(im1)
    im_blend[:, :, 0] += R
    im_blend[:, :, 1] += G
    im_blend[:, :, 2] += B

    # plotting the images
    f, ax = plt.subplots(2, 2)

    ax[0, 0].imshow(im1)
    ax[0, 0].set_title('image 1')
    ax[0, 1].imshow(im2)
    ax[0, 1].set_title('image 2')
    ax[1, 0].imshow(mask, 'gray')
    ax[1, 0].set_title('mask')
    ax[1, 1].imshow(im_blend)
    ax[1, 1].set_title('result')
    plt.show()
    return im1, im2, mask, im_blend


# -------------------------3.1----------------------------#

def build_gaussian_pyramid(im, max_levels, filter_size):
    ##########################################################
    # returns an array representing gaussian pyramid built
    # by a gaussian filter
    ##########################################################
    gauss_pyr = [im]
    filter_vec = create_gauss_filter(filter_size)
    for i in range(max_levels-1):
        im = reduce(im, filter_vec)
        # if the image has exceeded the legal size, stop
        if im.shape[0] < 16 or im.shape[1] < 16:
            break
        gauss_pyr.append(im)
    return gauss_pyr, filter_vec


def build_laplacian_pyramid(im, max_levels, filter_size):
    ##########################################################
    # returns an array representing laplacian pyramid built
    # by the algorithm from the tirgul
    ##########################################################
    gauss_pyr, filter_vec = build_gaussian_pyramid(im, max_levels, filter_size)
    lapl_pyr = []
```

```
    for i in range(max_levels-1):
        curr = gauss_pyr[i]
        exp_curr = expand(gauss_pyr[i+1], filter_vec)
        # check images sizes before subtracting
        if exp_curr.shape[0] > curr.shape[0]:
            exp_curr = np.delete(exp_curr, -1, axis=0)
        if exp_curr.shape[1] > curr.shape[1]:
            exp_curr = np.delete(exp_curr, -1, axis=1)
        # adding to the pyramid
        lapl_pyr.append(curr - exp_curr)
    # add the last gauss pyramid element
    lapl_pyr.append(gauss_pyr[-1])
    return lapl_pyr, filter_vec


# -------------------------3.2----------------------------#

def laplacian_to_image(lpyr, filter_vec, coeff):
    #######################################################
    # constructs the original image from its laplacian
    # pyramid
    #######################################################
    img = np.zeros_like(lpyr[-1])
    correct_shape = lpyr[0].shape
    for mat, co in zip(reversed(lpyr), reversed(coeff)):
        # adjustments before adding the matrices
        if img.shape[0] > mat.shape[0]:
            img = np.delete(img, -1, axis=0)
        if img.shape[1] > mat.shape[1]:
            img = np.delete(img, -1, axis=1)
        img += mat * co
        img = expand(img, filter_vec) if \
            img.shape != correct_shape else img
    return img


# -------------------------3.3----------------------------#

def render_pyramid(pyr, levels):
    #######################################################
    # calculates the height and width of the image where
    # the pyramid will be displayed
    #######################################################
    # calculating height and width of the image
    height = pyr[0].shape[0]
    cols = float(pyr[0].shape[1])
    width = 0
    for i in range(levels):
        width += cols
        cols = float(np.ceil(cols/2))
    res = np.zeros((height, int(width)))
    # rendering the pyramid
    Xbegin_pos, Xend_pos = 0, 0
    for i in range(levels):
        Xend_pos = pyr[i].shape[1]
        Ypos = pyr[i].shape[0]
        # set the image in its appropriate place in the pyramid
        res[0:Ypos, Xbegin_pos:Xbegin_pos + Xend_pos] += stretch_values(pyr[i])
        # advancing the starting position of the next image
        Xbegin_pos += Xend_pos
    return res


def display_pyramid(pyr, levels):
    #######################################################
    # displays the pyramid of a given image, amount of
    # levels deep
    #######################################################
```

```python
    res = render_pyramid(pyr, levels)
    plt.figure()
    plt.imshow(res, 'gray')
    plt.show()


# ---------------------------4------------------------------#

def pyramid_blending(im1, im2, mask, max_levels, filter_size_im, filter_size_mask):
    #######################################################
    # blends two images according to a given mask
    #######################################################
    L1, filt1 = build_laplacian_pyramid(im1, max_levels, filter_size_im)
    L2, filt1 = build_laplacian_pyramid(im2, max_levels, filter_size_im)
    G, filt2 = build_gaussian_pyramid(mask.astype(np.float32), max_levels, filter_size_mask)
    Lout = []
    for i in range(max_levels):
        curr = (G[i] * L1[i]) + ((1.0 - G[i]) * L2[i])
        Lout.append(curr)
    return np.clip(laplacian_to_image(Lout, filt1, np.ones(len(Lout))), 0, 1)


# --------------------------4.1-----------------------------#

def blending_example1():
    ####################################
    # example 1
    ####################################
    im1 = read_image(relpath('blend/stat1.jpg'), 2).astype(np.float32)
    im2 = read_image(relpath('blend/trump1.jpg'), 2).astype(np.float32)
    mask = read_image(relpath('blend/mask1.jpg'), 1)
    # some adjustments on the mask to disable dirt along the edges
    mask[mask > 0.5] = 1
    mask[mask <= 0.5] = 0
    mask = mask.astype(np.bool)
    return blend_images(im1, im2, mask, 35, 6)


def blending_example2():
    ####################################
    # example 2
    ####################################
    im1 = read_image(relpath('blend/givat2.jpg'), 2).astype(np.float32)
    im2 = read_image(relpath('blend/race2.jpg'), 2).astype(np.float32)
    mask = read_image(relpath('blend/mask2.jpg'), 1)
    # some adjustments on the mask to disable dirt along the edges
    mask[mask > 0.5] = 1
    mask[mask <= 0.5] = 0
    mask = mask.astype(np.bool)
    return blend_images(im1, im2, mask, 35, 1)


# --------------------------end-----------------------------#
```

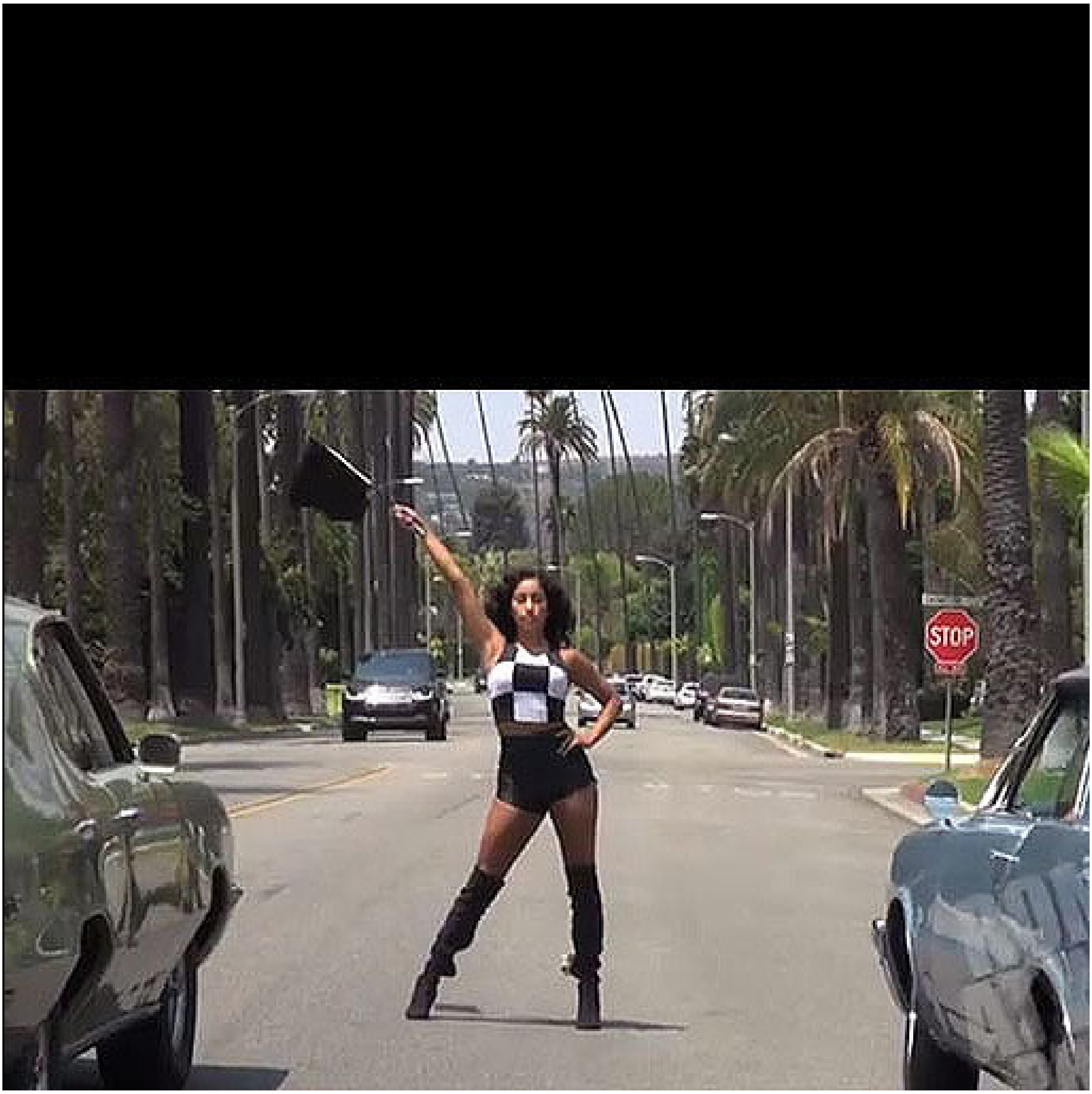
7 blend/givat2.jpg

8 blend/mask1.jpg



9 blend/mask2.jpg

10 blend/race2.jpg



11 blend/stat1.jpg

12 blend/trump1.jpg