Contents

1 Basic Test Results

```
ex3/
ex3/README.md
ex3/answer_q1.txt
ex3/answer_q2.txt
ex3/answer_q3.txt
ex3/externals/
ex3/externals/daft_orig.jpg
ex3/externals/dark_mask.jpg
ex3/externals/dark_orig.jpg
ex3/externals/freddie.jpg
ex3/externals/freddie_mask.png
ex3/externals/tal.jpg
ex3/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 ===

README file:

tal.porezky
sol3.py
answer_q1.txt
answer_q2.txt
answer_q3.txt

=== Answers to questions ===

Answer to Q1:
The dark level (darkness of the image) changes. Each level is more blurry than
 the one before it, so it contains lower frequencies. If the coefficient of a
 certain level is higher then the image will be darker at the appropriate
 frequencies.
We can think of it like an equlizer of the different images of the pyramid.

Answer to Q2:
Different image filters are responsible for the "smoothness" changes between
the two images blended.
Higher filter size will result a larger "gaussian" which will be used as the
smoothing tool. The higher the filter size is, the more smooth will be the
change at the points where the two images blend.
Smaller filter size will result a smaller "gaussian" which will lead to a
rougher change (not so smooth) at the points where the two images blend.

Answer to Q3:
The higher the number of pyramid levels the less sharp the result image (the
blended image) will be.

=== Load Student Library ===

Loading...

=== 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()...
/tmp/bodek.Tz1Ldl/impr/ex3/tal.porezky/gitsub/testdir/test:8: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  im = imread(filename)
    Passed!
Checking types of returned results...
    Passed!


=== Presubmission Completed Successfully ===


    Please go over the output and verify that there were 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 ex3/README.md

```
tal.porezky
sol3.py
answer_q1.txt
answer_q2.txt
answer_q3.txt
```

3 ex3/answer q1.txt

The dark level (darkness of the image) changes. Each level is more blurry than the one before it, so it contains lower frequencies. If the coefficient of a certain level is higher then the image will be darker at the appropriate frequencies.

We can think of it like an equlizer of the different images of the pyramid.

4 ex3/answer q2.txt

```
Different image filters are responsible for the "smoothness" changes between
the two images blended.
Higher filter size will result a larger "gaussian" which will be used as the
smoothing tool. The higher the filter size is, the more smooth will be the
change at the points where the two images blend.
Smaller filter size will result a smaller "gaussian" which will lead to a
rougher change (not so smooth) at the points where the two images blend.
```

5 ex3/answer q3.txt

```
The higher the number of pyramid levels the less sharp the result image (the
blended image) will be.
```

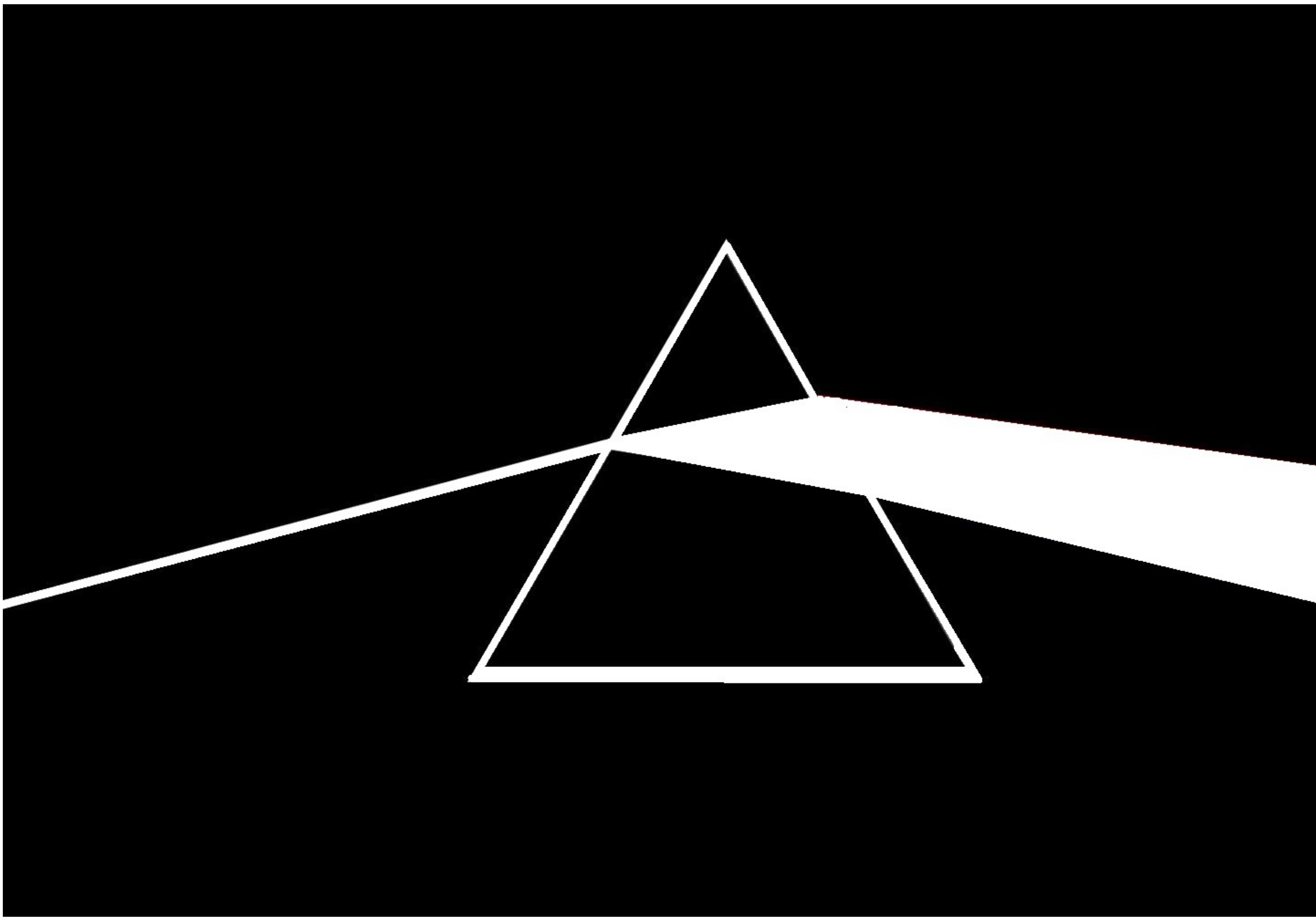
6 ex3/sol3.py

```
# ----------- Imports -----------
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
import numpy as np
from scipy.ndimage.filters import convolve as _convolve
from imageio import imread
import os

# ----------- Constants -----------
MIN_IMG_SIZE_IN_EACH_AXIS = 16
EXPEND_FACTOR = 2


# ----------- Helper functions -----------
def read_image(filename, representation):
    """
    function which reads an image ˜Lle and converts it into a given
    representation.
    :param filename: the filename of an image on disk (could be grayscale or
    RGB).
    :param representation: representation code, either 1 or 2 defining
    whether the output should be a grayscaleimage (1) or an RGB image (2).
    If the input image is grayscale, we won't call it with representation = 2.
    :return: rgb or grayscale img
    """
    image = imread(filename)
    new_image = image.astype(np.float64)
    new_image /= 255
    if representation == 1:
        new_image = rgb2gray(new_image)
    return new_image


def _im_downsample(im, blur_filter):
    """
    downsamples the image according to the blur filter given and takes
    every even pixel in the image.
    :param im: np.array image
    :param blur_filter: np.array of size (1,) of the filter
    :return: smaller sample.
    """
    im = _convolve(im, blur_filter, mode='reflect')
    im = _convolve(im, blur_filter.T, mode='reflect')
    im = im[::2, ::2]
    return im


def _im_expand(im, blur_filter):
    """
    expands the image accodring to the blur filter. Makes the image twice
    as bigger.
    :param im: np.array image
    :param blur_filter: np.array of size (1,) of the filter
    :return: bigger image.
    """
    expended_im = np.zeros(
        (im.shape[0] * EXPEND_FACTOR, im.shape[1] * EXPEND_FACTOR))
    expended_im[1::2, 1::2] = im
    blur_filter = blur_filter * EXPEND_FACTOR
    expended_im = _convolve(_convolve(expended_im, blur_filter),
                            blur_filter.T)
    return expended_im


def _get_binomial_coefficients(size):
    """
    creates gaussian vector of the given size. size must be odd.
    :param size: odd integer
    :return: gaussian vector of the given size
    """
    binomial_coefficients = np.array([1, 1], dtype=np.float64)
    for _ in range(size - 2):
        binomial_coefficients = np.convolve([1, 1], binomial_coefficients)
    binomial_coefficients = binomial_coefficients[np.newaxis, :]
    return binomial_coefficients


def _reshape_dims_of_imgs(org_im, other_im):
    """
    makes fixes to other_im so it shape will fit org_im
    :param org_im: original image.
    :param other_im: image which shape to fix.
    :return: other_image with fixed size.
    """
    if org_im.shape[0] != other_im.shape[0]:
        other_im = other_im[:-1, :]
    if org_im.shape[1] != other_im.shape[1]:
        other_im = other_im[:, :-1]
    return other_im


def _strech_im(im):
    """
    streches the image so it's values will fit 0-1 scale.
    :param im: original image.
    :return: image with values between 0 and 1.
    """
    return (im - np.min(im)) / (np.max(im) - np.min(im))


# ----------- 3.1: Image Pyramids -----------
def build_gaussian_pyramid(im, max_levels, filter_size):
    """
    constructs a gaussian pyramid.
    :param im: grayscale image with double values in [0,1].
    :param max_levels: maximal number of levels in the resulting pyramid.
    :param filter_size: the size of the gaussian filter
    :return: pyr, filter_vec. pyr - python array where the elements are
    images with different sizes. filter_vec - the gaussian filter used.
    """
    blur_filter_not_normalized = _get_binomial_coefficients(filter_size)
    blur_filter = blur_filter_not_normalized / np.sum(
        blur_filter_not_normalized)

    pyr = list()
    curr_im = im
    while ((curr_im.shape[0] >= MIN_IMG_SIZE_IN_EACH_AXIS) and
           (curr_im.shape[1] >= MIN_IMG_SIZE_IN_EACH_AXIS) and
           (len(pyr) < max_levels)):
        pyr.append(curr_im)
        curr_im = _im_downsample(curr_im, blur_filter)

    return pyr, blur_filter


def build_laplacian_pyramid(im, max_levels, filter_size):
    """
    constructs a laplacian pyramid.
    :param im: grayscale image with double values in [0,1].
    :param max_levels: maximal number of levels in the resulting pyramid.
    :param filter_size: the size of the gaussian filter
    :return: pyr, filter_vec. pyr - python array where the elements are
    images with different sizes. filter_vec - the gaussian filter used.
    """
    gauss_pyr_list, gauss_blur_filter = build_gaussian_pyramid(im,
                                                               max_levels,
                                                               filter_size)
    laplace_pyr = list()
    for pyr_idx in range(len(gauss_pyr_list) - 1):
        curr_gauss_img = gauss_pyr_list[pyr_idx]
        expended_next_gauss_im = _im_expand(gauss_pyr_list[pyr_idx + 1],
                                            gauss_blur_filter)
        # todo maybe i should delete these later.
        expended_next_gauss_im = _reshape_dims_of_imgs(curr_gauss_img,
                                                       expended_next_gauss_im)
        laplace_pyr.append(gauss_pyr_list[pyr_idx] - expended_next_gauss_im)
    laplace_pyr.append(gauss_pyr_list[-1])
    return laplace_pyr, gauss_blur_filter


# ----------- 3.2: Laplacian pyramid reconstruction -----------

def laplacian_to_image(lpyr, filter_vec, coeff):
    """
    reconstructes image from its aplacian pyramid.
    :param lpyr: return value of the build_x_pyramid function
    :param filter_vec: return value of the build_x_pyramid function
    :param coeff: python list with length same as the number of levels in
    the pyramid lpyr.
    :return: image from the lalpacian pyramid.
    """
    for i in range(len(coeff)):
        lpyr[i] = lpyr[i] * coeff[i]
    im = lpyr[-1]
    for i in range(len(lpyr) - 1, 0, -1):
        expended_im = _reshape_dims_of_imgs(lpyr[i - 1],
                                            _im_expand(im, filter_vec))
        im = (lpyr[i - 1] + expended_im)
    return im


# ----------- 3.3: Pyramid display -----------
def render_pyramid(pyr, levels):
    """
    constructes image from the 'levels' elements in pyr.
    :param pyr: return value of the build_x_pyramid function
    :param levels: how many pictures there will be in the image.
    :return: image.
    """
    im = _strech_im(pyr[0])
    for i in range(1, min(len(pyr), levels)):
        difference_y = pyr[0].shape[0] - pyr[i].shape[0]
        im = np.hstack((im, np.pad(_strech_im(pyr[i]),
                                   ((0, difference_y),
                                    (0, 0)),
                                   'constant')))
    return im


def display_pyramid(pyr, levels):
    """
    displays the image built in render_pyramid
    :param pyr: return value of the build_x_pyramid function
    :param levels: how many pictures there will be in the image.
    :return: None
```

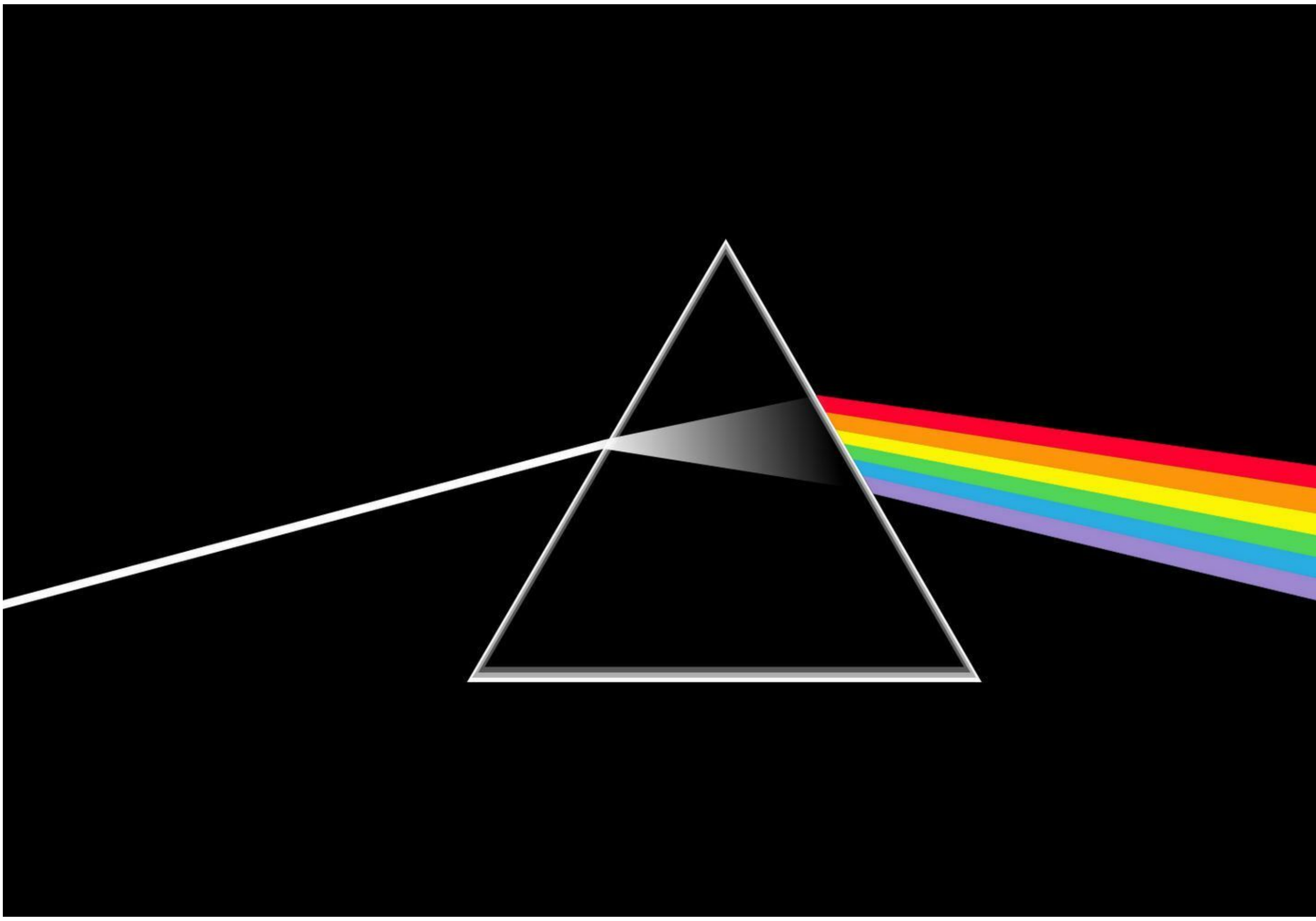
```python
    """
    res = render_pyramid(pyr, levels)
    plt.figure()
    plt.imshow(res, cmap='gray')
    plt.show()
    return


# ----------- 4: Pyramid blending -----------
def pyramid_blending(im1, im2, mask, max_levels, filter_size_im,
                     filter_size_mask):
    """
    pyramid blending as described in the lecture.
    :param im1: grayscale image to blend
    :param im2: grayscale image to blend
    :param mask: boolean mask
    :param max_levels: parameter generated from the gaussian and laplacian
    pyramids.
    :param filter_size_im: size of the gaussian filter (ood integer)
    :param filter_size_mask: size of the gaussian filter used for the mask.
    :return: the blended image.
    """
    assert (im1.shape == im2.shape == mask.shape)
    L1, filter_1 = build_laplacian_pyramid(im1,
                                           max_levels,
                                           filter_size_im)
    L2, filter_2 = build_laplacian_pyramid(im2,
                                           max_levels,
                                           filter_size_im)
    Gm, filter_m = build_gaussian_pyramid(mask.astype(np.float64),
                                          max_levels,
                                          filter_size_mask)
    Lout = list()
    for k in range(len(L1)):
        Lout.append(Gm[k] * L1[k] + (1 - Gm[k]) * L2[k])
    coeff = np.ones(len(L1)) * 1.0
    im = laplacian_to_image(Lout, filter_1, coeff)
    im = np.clip(im, 0, 1)

    return im


# ----------- 5: Your blending examples -----------

def relpath(filename):
    """
    fucntion provided by you.
    """
    return os.path.join(os.path.dirname(__file__), filename)


def _color_blending(im1, im2, mask, max_levels, filter_size_im,
                    filter_size_mask):
    """
    helper function for blending each color (red, green, blue) of the image
    :param im1: grayscale image to blend
    :param im2: grayscale image to blend
    :param mask: boolean mask
    :param max_levels: parameter generated from the gaussian and laplacian
    pyramids.
    :param filter_size_im: size of the gaussian filter (ood integer)
    :param filter_size_mask: size of the gaussian filter used for the mask.
    :return: the blended image.
    """
    r = pyramid_blending(im1[:, :, 0], im2[:, :, 0], mask, max_levels,
                         filter_size_im, filter_size_mask)
    g = pyramid_blending(im1[:, :, 1], im2[:, :, 1], mask, max_levels,
                         filter_size_im, filter_size_mask)
    b = pyramid_blending(im1[:, :, 2], im2[:, :, 2], mask, max_levels,
                         filter_size_im, filter_size_mask)
    blended_im = np.empty(im1.shape)
    blended_im[:, :, 0] = r
    blended_im[:, :, 1] = g
    blended_im[:, :, 2] = b
    return blended_im


def _plot_4_images(im1, im2, mask, blended, gray_result):
    """
    plots together the 4 images: im1, im2, mask and blended.
    :param im1: grayscale image to blend
    :param im2: grayscale image to blend
    :param mask: boolean mask
    :param blended: blended image created
    :param gray_result: True or False.
    :return: none
    """
    plt.figure()
    plt.subplot(2, 2, 1)
    plt.imshow(im1)
    plt.subplot(2, 2, 2)
    plt.imshow(im2)
    plt.subplot(2, 2, 3)
    plt.imshow(mask, cmap='gray')
    plt.subplot(2, 2, 4)
    if gray_result:
        plt.imshow(blended, cmap='gray')
    else:
        plt.imshow(blended)
    plt.show()


def blending_example1():
    """
    my own blending examples. This function puts my head on Freddie
    Mercury's head.
    :return: im1, im2, mask and the blended image.
    """
    im1 = read_image(relpath('externals/freddie.jpg'), 2)
    im2 = read_image(relpath('externals/tal.jpg'), 2)
    mask = read_image(relpath('externals/freddie_mask.png'), 1)
    mask = mask > 0.5
    blended = rgb2gray(np.clip(_color_blending(im1, im2, mask, 7, 15, 15),
                               0, 1))
    _plot_4_images(im1, im2, mask, blended, True)
    return im1, im2, mask.astype(np.bool), blended


def blending_example2():
    """
    my own blending examples. This function combains to music elements of
    artists which I really like - Pink Ployd's album "Dark side of the
    moon" and photos of "Daft Punk" robots.
    :return: im1, im2, mask and the blended image.
    """
    im1 = read_image(relpath('externals/daft_orig.jpg'), 2)
    im2 = read_image(relpath('externals/dark_orig.jpg'), 2)
    mask = read_image(relpath('externals/dark_mask.jpg'), 1)
    mask = mask < 0.5
    blended = np.clip(_color_blending(im1, im2, mask, 7, 15, 15), 0, 1)
    _plot_4_images(im1, im2, mask, blended, False)
    return im1, im2, mask.astype(np.bool), blended
```

7 ex3/externals/daft orig.jpg

8 ex3/externals/dark mask.jpg



9 ex3/externals/dark orig.jpg



10 ex3/externals/freddie.jpg

11 ex3/externals/freddie mask.png

12 ex3/externals/tal.jpg