Contents

1 Basic Test Results

```
Archive:  /tmp/bodek.pPLy7U/impr/ex3_late/itamakatz/presubmission/submission
  inflating: current/README
  inflating: current/answer_q1.txt
  inflating: current/answer_q2.txt
  inflating: current/answer_q3.txt
   creating: current/images/
  inflating: current/images/im1_apple.jpg
  inflating: current/images/im1_filter.jpg
  inflating: current/images/im1_huji.jpg
  inflating: current/images/im2_eye.jpg
  inflating: current/images/im2_filter.jpg
  inflating: current/images/im2_flower.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:  itamakatz

submitted files:

==== README for ex3 ===

List of submitted files:

README - this file
answer_qt1.txt - Answer to question Q1
answer_qt2.txt - Answer to question Q2
answer_qt3.txt - Answer to question Q3
sol3.py - python3 code
images - directory containing the blending examples images:

- im1_apple.jpg
1  - im1_filter.jpg
- im1_huji.jpg

- im2_eye.jpg
2  - im2_filter.jpg
- im2_flower.jpg


* Note: Those pictures were taken by me. I like practicing photography
in my spare time and those pictures were taken from special and artistic
points of view. You are more than welcome to visit my website ik-art.com
for more photos.

=== Answers to questions ===

Answer to Q1:
Answer to question Q1:

```

```
Multiplying each level with a different value emphasizes
certain ares in the frequency domain. So we are actually
trying to control those areas of the drequency domain.



Answer to Q2:
Answer to question Q2:

When we use a bigger gaussian filter, the edges of the image
masked onto the second image are more blended in the environment,
making it hard to visualize the stitch line. With a small
gaussian filter we can vividly see the stitch because the blending
is less strong.



Answer to Q3:
Answer to question Q3:


With higher number of levels, we'll get better blend in
the environment. The reason is that we include more
low frequencies in the pyramid blending stage, which make
the blending more noticed.


=== 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

```
itamakatz

==== README for ex3 ===

List of submitted files:

README - this file
answer_qt1.txt - Answer to question Q1
answer_qt2.txt - Answer to question Q2
answer_qt3.txt - Answer to question Q3
sol3.py - python3 code
images - directory containing the blending examples images:

    - im1_apple.jpg
 1  - im1_filter.jpg
    - im1_huji.jpg

    - im2_eye.jpg
 2  - im2_filter.jpg
    - im2_flower.jpg


* Note: Those pictures were taken by me. I like practicing photography
  in my spare time and those pictures were taken from special and artistic
  points of view. You are more than welcome to visit my website ik-art.com
  for more photos.
```

3 answer q1.txt

```
Answer to question Q1:

Multiplying each level with a different value emphasizes
certain ares in the frequency domain. So we are actually
trying to control those areas of the drequency domain.
```

4 answer q2.txt

```
Answer to question Q2:

When we use a bigger gaussian filter, the edges of the image
masked onto the second image are more blended in the environment,
making it hard to visualize the stitch line. With a small
gaussian filter we can vividly see the stitch because the blending
is less strong.
```

5 answer q3.txt

```
Answer to question Q3:


With higher number of levels, we'll get better blend in
the environment. The reason is that we include more
low frequencies in the pyramid blending stage, which make
the blending more noticed.
```

6 sol3.py

```python
import os
import functools
import numpy as np
import scipy.special
from scipy.misc import imread
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from scipy.signal import convolve2d

# global parameter to plot as many figures as necessary
g_plot_index = 1

def index():
    # simulates a static variables of g_plot_index.
    # returns - number of figure before increment
    global g_plot_index
    g_plot_index += 1
    return g_plot_index - 1

def read_image(filename, representation):
    # filename - file to open as image
    # representation - is it a B&W or color image
    im = imread(filename)
    # check if it is a B&W image
    if(representation == 1):
        im = rgb2gray(im)
    # convert to float and normalize
    return (im / 255).astype(np.float32)


def relpath(filename):
    # converts relative paths to absolute
    # filename - relative path
    # returns - absolute path
    return os.path.join(os.path.dirname(__file__), filename)



def create_filter_vec(filter_size):
    # creates a binomial coefficient of length filter_size
    # filter_size - length of the coefficient array
    # returns - the binomial coefficient array

    # special case of an odd number.
    if filter_size == 1: return np.array([[0]])
    conv_ker =  np.array([[1, 1]])
    filter = conv_ker
    # using an O(logN) algorithm to compute the filter
    log2 = np.log2(filter_size - 1)
    whole = np.floor(log2).astype(np.int64)
    rest = (2**(log2) - 2**(whole)).astype(np.int64)
    for i in range(whole):
        filter = convolve2d(filter, filter).astype(np.float32)
    for i in range(rest):
        filter = convolve2d(filter, conv_ker).astype(np.float32)
    # normalize
    return (filter / np.sum(filter)).astype(np.float32)


```

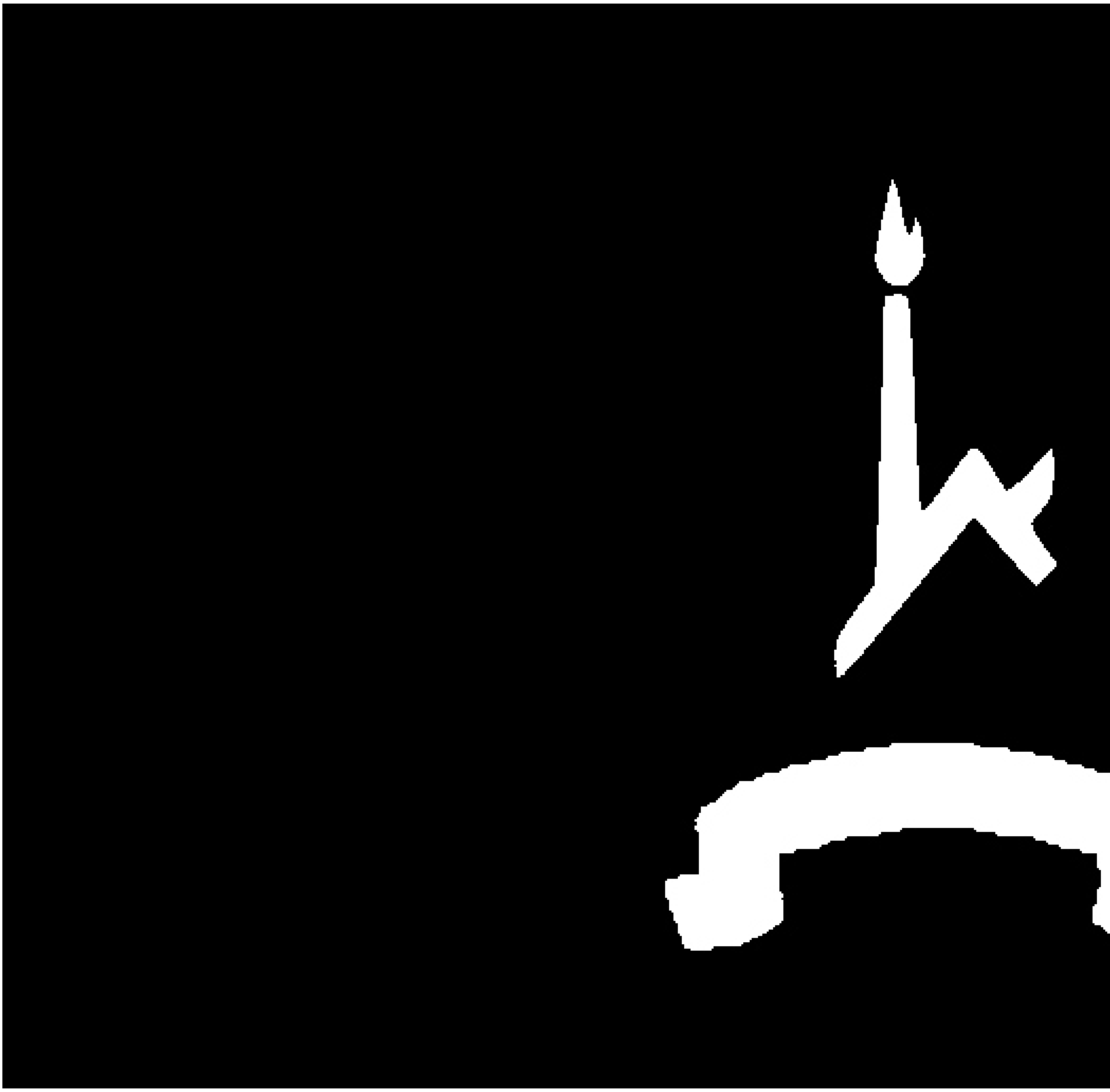
```python

def build_gaussian_pyramid(im, max_levels, filter_size):
    # calc the filter array
    filter_vec = create_filter_vec(filter_size)
    # create the entire array for better complexity
    pyr = [0] * (np.min([max_levels, np.log2(im.shape[0]).astype(np.int64) - 3,
                         np.log2(im.shape[1]).astype(np.int64) - 3]))
    pyr[0] = im
    # for each iter, use the last iter to calc the current iter. note i transpose twice. once to calc
    # the y conv and the second to flip back the image
    for i in range(1, len(pyr)):
        pyr[i] = scipy.ndimage.filters.convolve(pyr[i - 1], filter_vec, output = None, mode = 'mirror')
        pyr[i] = scipy.ndimage.filters.convolve(pyr[i].transpose(), filter_vec, output = None, mode = 'mirror')
        pyr[i] = (pyr[i].transpose()[::2, ::2]).astype(np.float32)
    return pyr, filter_vec



def stretch(elem):
    # stretching to [0,1]
    max_ =  np.max(elem)
    range_ = max_ - np.min(elem)
    return 1 - ((max_ - elem) / range_)



def expand(filter_vec, im):
    # method that helps calculate an expanded image given an image and a kernel array
    # filter_vec - kernel used to build the gaussian pyramid
    # im - image to expand
    # return - the expanded image after interpolation
    expand = np.zeros([im.shape[0] * 2, im.shape[1] * 2], dtype=np.float32)
    expand[0::2, 0::2] = im
    expand = scipy.ndimage.filters.convolve(expand, filter_vec, output=None, mode='mirror')
    expand = scipy.ndimage.filters.convolve(expand.transpose(), filter_vec, output=None, mode='mirror')
    return (expand.transpose()).astype(np.float32)



def build_laplacian_pyramid(im, max_levels, filter_size):
    # build the laplacian pyramid from a given image
    gauss_pyr, filter_vec = build_gaussian_pyramid(im, max_levels, filter_size)
    filter_vec *= 2 #  on expansion the kernel should not be completely normalized
    pyr = [0] * len(gauss_pyr) # create the entire array for better complexity
    # using functional programing to avoid a for loop
    pyr[:-1] = np.ndarray.tolist(np.array(gauss_pyr[:-1]) -  \
                                 np.array(list(map(functools.partial(expand, filter_vec), gauss_pyr[1:]))))
    pyr[-1] = gauss_pyr[-1]
    return pyr, filter_vec



def laplacian_to_image(lpyr, filter_vec, coeff):
    im = np.array([[0]]).astype(np.float32)
    # add leyers and expand for next iteration
    for i  in range(len(lpyr) - 1):
        im = expand(filter_vec, im + lpyr[-(i + 1)] * coeff[-(i + 1)])
    # mult by the coefficient
    return (im + lpyr[0] * coeff[0]).astype(np.float32)


def render_pyramid(pyr, levels):
    # calc the length of the returned matrix by the geometric progression
    length, curr = 0, float(pyr[0].shape[1])
    for i in range(levels):
        length += curr
        curr = np.ceil(curr/2)
    # return the empty matrix
```

```
    return np.zeros([pyr[0].shape[0], int(length)], dtype=np.float32)



def display_pyramid(pyr, levels):
    res = render_pyramid(pyr, levels)
    length = 0
    # find location of each layer in the res matrix
    for i in range(levels):
        res[0 : pyr[i].shape[0], length : pyr[i].shape[1] + length] = stretch(pyr[i])
        length += pyr[i].shape[1]
    # plot the resulting matrix
    plt.figure(index())
    plt.imshow(np.clip(res, 0, 1), plt.cm.gray)
    plt.show()
    return



def pyramid_blending(im1, im2, mask, max_levels, filter_size_im, filter_size_mask):
    # calc L1,L2, G1
    im1_lpyr, filter_vec = build_laplacian_pyramid(im1, max_levels, filter_size_im)
    im2_lpyr, _ = build_laplacian_pyramid(im2, max_levels, filter_size_im)
    mask_gpyr, _ = build_gaussian_pyramid(mask.astype(np.float32), max_levels, filter_size_mask)
    # calc L_out
    out_pyrl = (np.array(mask_gpyr) * np.array(im1_lpyr)) + (1 - np.array(mask_gpyr)) * np.array(im2_lpyr)
    # clip to truncate the laplacian negative values
    return np.clip(laplacian_to_image(out_pyrl, filter_vec, np.ones(len(im1_lpyr))), 0, 1)



def sub_plot(im, arg, color):
    # faster way to plot many images in one figure
    # im - im to plot
    # arg - argument for subplot
    # color - boolean if it is a color image or not.
    plt.subplot(arg)
    plt.imshow(im) if color else plt.imshow(im, plt.cm.gray)
    return


def examples(path_1, path_2, mask_path, max_levels, filter_size_im, filter_size_mask):
    # general function to plot blending examples
    # path_1 - relative path to first image
    # path_2 - relative path to second image
    # mask_path - relative path to the mask image
    # max_levels - number of layers in the pyramid
    # filter_size_im - size of im1, im2 filter
    # filter_size_mask - size of the mask filter
    # returns - [im1, im2, mask, im_blend] - the opened images and the resulting blend
    im1 = read_image(relpath(path_1), 2)
    im2 = read_image(relpath(path_2), 2)
    # mult by 255 to revert the normalization so the mask is binary
    mask = read_image(relpath(mask_path), 1) * 255
    mask[mask > 0.5] = True
    mask[mask <= 0.5] = False
    mask = mask.astype(np.bool_)
    # calc all the RGB axis
    im_blend = im1 * 0
    im_blend[:,:,0] = pyramid_blending(im1[:,:,0], im2[:,:,0], mask, max_levels, filter_size_im, filter_size_mask)
    im_blend[:,:,1] = pyramid_blending(im1[:,:,1], im2[:,:,1], mask, max_levels, filter_size_im, filter_size_mask)
    im_blend[:,:,2] = pyramid_blending(im1[:,:,2], im2[:,:,2], mask, max_levels, filter_size_im, filter_size_mask)

    # plot results
    plt.figure(index())

    sub_plot(im1, 221, True)
    sub_plot(im2, 222, True)
```

```python
    sub_plot(mask, 223, False)
    sub_plot(im_blend, 224, True)

    plt.show()
    return im1, im2, mask, im_blend



def blending_example1():
    return examples('images/im1_huji.jpg', 'images/im1_apple.jpg', 'images/im1_filter.jpg', 2, 3, 55)

def blending_example2():
    return examples('images/im2_flower.jpg', 'images/im2_eye.jpg', 'images/im2_filter.jpg', 4, 31, 55)
```

7 images/im1 apple.jpg

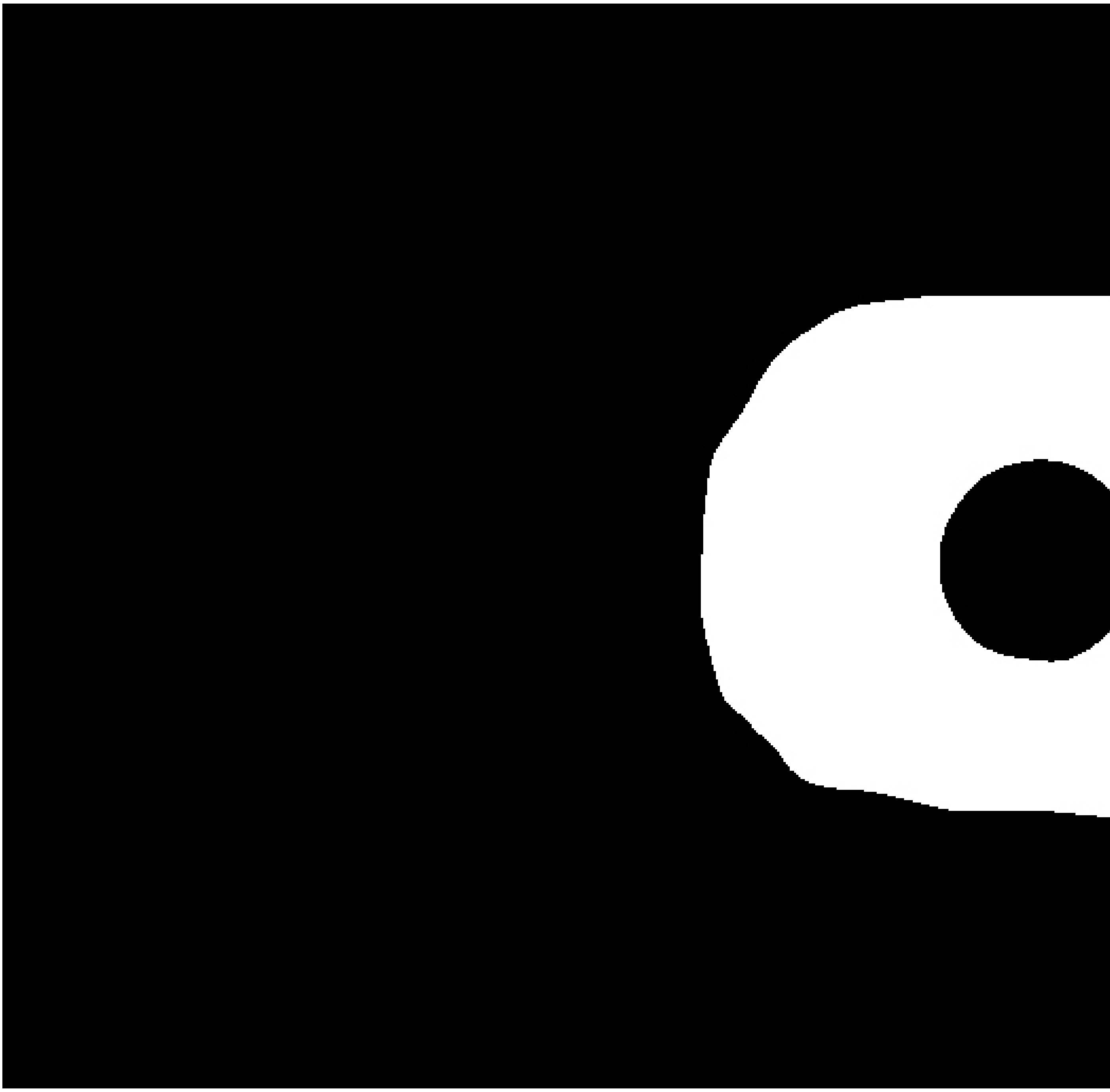
8 images/im1 filter.jpg



9 images/im1 huji.jpg

10 images/im2 eye.jpg

11 images/im2 filter.jpg



12 images/im2 flower.jpg