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/final_dolphin.jpg
ex3/externals/final_lafa.jpg
ex3/externals/final_mask_dolphin.jpg
ex3/externals/final_mask_refa.png
ex3/externals/final_refa.jpg
ex3/externals/final_water.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:

# ex3-yishai.hazi
sol3.py
externals:
final_dolphin.jpg
final_mask_dolphin.jpg
final_water.jpg
final_refa.jpg
final_mask_refa.png
final_lafa.jpg
answer_q1.txt
answer_q2.txt
answer_q3.txt
README.md


=== Answers to questions ===

Answer to Q1:
By multiplying each level with a different value - we can control of "how much" we take from the frequencies.
For example, if we multiply the higher levels of the pyramids with small value, it means less high frequencies
in the final image.

Answer to Q2:
It looks like that if the filter size is bigger, it causes the blended image to be blur, thats probably because
at each reduction (in the pyramid) we blur a larger range of pixels.

Answer to Q3:
More levels that the pyramid has, results in a smoother and accurate image. The reason for that is that
each level "merge" different set of frequencies, so if the pyramid has more levels, more frequencies are merged. (the higher
=== 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.woFmwe/impr/ex3/yishai.hazi/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

```
# ex3-yishai.hazi
sol3.py
externals:
final_dolphin.jpg
final_mask_dolphin.jpg
final_water.jpg
final_refa.jpg
final_mask_refa.png
final_lafa.jpg
answer_q1.txt
answer_q2.txt
answer_q3.txt
README.md
```

3 ex3/answer q1.txt

```
By multiplying each level with a different value - we can control of "how much" we take from the frequencies.
For example, if we multiply the higher levels of the pyramids with small value, it means less high frequencies
in the final image.
```

4 ex3/answer q2.txt

It looks like that if the filter size is bigger, it causes the blended image to be blur, thats probably because at each reduction (in the pyramid) we blur a larger range of pixels.

5 ex3/answer q3.txt

```
More levels that the pyramid has, results in a smoother and accurate image. The reason for that is that
each level "merge" different set of frequencies, so if the pyramid has more levels, more frequencies are merged. (the higher
```

6 ex3/sol3.py

```python
import os
import numpy as np
import skimage.color as color
import imageio
import scipy.ndimage.filters
import matplotlib.pyplot as plt


def normalize(image):
    """
    A function that normalize the elements inside
    a numpy matrix, and change their type to float64.
    :param image: a numpy matrix
    :return: the matrix with all elements normalized
    to the range: [0, 1].
    """
    image = image.astype('float64')
    return image / 255


def read_image(filename, representation):
    """
    A function that reads a file according
    to a given representation.
    :param filename: The file name.
    :param representation: 1 represents rgb,
    2 represents gray_scale.
    :return: An image according to the
    given representation.
    """
    image = imageio.imread(filename)
    if representation == 2 or image.ndim != 3:
        if np.any(image > 1):
            return normalize(image)
    elif representation == 1:
        return color.rgb2gray(image)


def normalize_gaussian_filter(filter_vec):
    """
    A function that normalize the gaussian
    filter.
    :param filter_vec: The filter.
    :return: the normalized filter.
    """
    return filter_vec / np.sum(filter_vec)


def build_gaussian_filter(filter_size):
    """
    A function that build a gaussian filter
    according to a given size.
    :param filter_size: odd integer that
    represents the desired size.
    :return: The gaussian filter as
    a numpy array with shape: [1, filter_size].
    (without normalization)
    """
    number_of = filter_size - 2
    if filter_size > 1:
        small_kernel = np.array([1, 1])
    else:
        small_kernel = np.array([1])
    the_filter = small_kernel
    for i in range(number_of):
        the_filter = np.convolve(the_filter, small_kernel)
    return np.reshape(the_filter, (1, filter_size))


def convolve(im, filter_vec):
    """
    A function that convolve an image with
    a given filter.
    :param im: The given image as a grayscale
    image with double values in [0, 1].
    :param filter_vec: The filter as a row vector.
    :return: The image after the deletion.
    """
    im_after_rows_convolution = scipy.ndimage.filters.convolve(
        im, filter_vec)
    return scipy.ndimage.filters.convolve(
        im_after_rows_convolution, np.transpose(filter_vec))


def delete_elements(im):
    """
    A function that delete all the elements
    in the odd indexes. (as well as all the
    odd rows)
    :param im: The given image.
    :return: The image after the deletion.
    """
    odd_indexes_of_rows = np.arange(1, im.shape[0] + 1, 2)
    odd_indexes_of_cols = np.arange(1, im.shape[1] + 1, 2)
    im_after_horizontal_clean = np.delete(im,
                                          odd_indexes_of_rows, axis=0)
    im_after_vertical_clean = np.delete(im_after_horizontal_clean,
                                        odd_indexes_of_cols,
                                        axis=1)
    return im_after_vertical_clean


def reduce(im, filter_vec):
    """
    A function that creates a smaller image
    (smaller by factor of 2) of the given image.
    :param im: The given image.
    :param filter_vec: The given filter
    that blur the image.
    :return: The smaller image.
    """
    im_after_convolution = convolve(im, filter_vec)
    return delete_elements(im_after_convolution)


def build_gaussian_pyramid(im, max_levels, filter_size):
    """
    A function that construct a Gaussian
    pyramid of a given image.
    :param im: A grayscale image with double
    values in [0, 1].
    :param max_levels: The maximal number of levels1
    in the resulting pyramid.
    :param filter_size: The size of the Gaussian filter
    (an odd scalar that represents a squared filter) to be used
    in constructing the pyramid filter.
    :return: The resulting pyramid pyr as a standard python array,
    and filter_vec - which is a row vector of shape (1, filter_size).
    """
    filter_vec = build_gaussian_filter(filter_size)
    filter_vec = normalize_gaussian_filter(filter_vec)
    pyr = [im]
    smallest_image = im
    max_levels -= 1
    while max_levels >= 1 and smallest_image.shape[0] > 16 \
            and smallest_image.shape[1] > 16:
        smallest_image = reduce(smallest_image, filter_vec)
        pyr.append(smallest_image)
        max_levels -= 1
    return pyr, filter_vec


def zero_pad(im):
    """
    A function that expand an image by
    adding zeros in the odd indexes (and odd
    rows)
    :param im: The given image as a grayscale
    image with double values in [0, 1].
    :return: The image after extension.
    """
    odd_indexes_in_rows = np.arange(1, im.shape[0] + 1)
    odd_indexes_in_cols = np.arange(1, im.shape[1] + 1)
    im = np.insert(im, odd_indexes_in_rows, 0, axis=0)
    im = np.insert(im, odd_indexes_in_cols, 0, axis=1)
    return im


def expand(im, filter_vec):
    """
    A function that expand an image by
    adding zeros to it and then blur the
    result with the given filter.
    :param im: The given image as a grayscale
    image with double values in [0, 1].
    :param filter_vec:
    :return: The given filter we use for blur.
    (a row vector of shape (1, filter_size).)
    """
    im = zero_pad(im)
    im_after_convolve = convolve(im, filter_vec)
    return im_after_convolve


def build_laplacian_pyramid(im, max_levels, filter_size):
    """
    A function that construct a  Laplacian pyramid
    of a given image.
    :param im: A grayscale image with double
    values in [0, 1].
    :param max_levels: The maximal number of levels1
    in the resulting pyramid.
    :param filter_size: The size of the Gaussian filter
    (an odd scalar that represents a squared filter) to be used
    in constructing the pyramid filter.
    :return:The resulting pyramid pyr as a standard python array,
    and filter_vec - which is row vector of shape (1, filter_size).
    """
    gaussian_pyr, filter_vec = build_gaussian_pyramid(im, max_levels,
                                                      filter_size)
    pyr = []
    for i in range(len(gaussian_pyr) - 1):
        pyr.append(gaussian_pyr[i] - expand(gaussian_pyr[i + 1],
                                            filter_vec * 2))
    pyr.append(gaussian_pyr[-1])
```

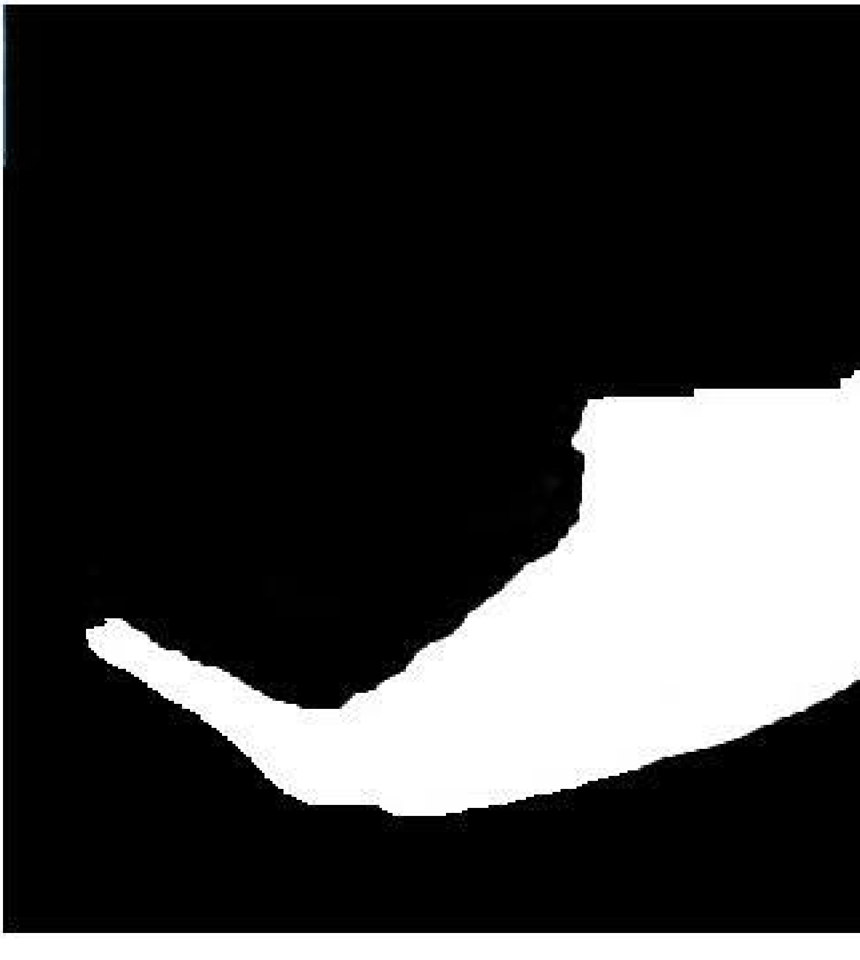
```python
    return pyr, filter_vec


def laplacian_to_image(lpyr, filter_vec, coeff):
    """
    A function that construct an image from its
    Laplacian Pyramid.
    :param lpyr: The Laplacian pyramid
    :param filter_vec: The filter that is generated
    by the function: "build_laplacian_pyramid".
    :param coeff: A python list.  Each level i of the
    laplacian pyramid is multiplied by its corresponding
    coefficient coeff[i].
    :return: The original image.
    """
    for i in range(len(lpyr)):
        lpyr[i] = lpyr[i] * coeff[i]
    image = lpyr[-1]
    for i in range(len(lpyr) - 1, 0, -1):
        image = expand(image, filter_vec * 2) + lpyr[i - 1]
    return image


def stretch_image(image):
    """
    A function that stretch the values of
    the given image to the range: [0, 1].
    :param image: The given image.
    :param min_value: A given minimum.
    :param max_value: A given maximum.
    :return: The stretched image.
    """
    return (image - np.min(image)) / (np.max(image) - np.min(image))


def render_pyramid(pyr, levels):
    """
    A function that creates a single black image
    in which the pyramid levels of the given pyramid
    pyr are stacked horizontally.
    :param pyr: The given pyramid (a Gaussian
    or Laplacian pyramid)
    :param levels: the number of levels of the pyramid.
    :return: A black image in which the pyramid levels of the
    given pyramid pyr are stacked horizontally.
    """
    res = stretch_image(pyr[0])
    for i in range(1, min(levels, len(pyr))):
        black_image = [[0] * pyr[i].shape[1]] * pyr[i].shape[0] * (2 ** i - 1)
        stretched_image = stretch_image(pyr[i])
        smaller_image_with_black_part = np.vstack(
            (stretched_image, black_image))
        res = np.hstack((res, smaller_image_with_black_part))
    return res


def display_pyramid(pyr, levels):
    """
    A function that use render_pyramid to
    internally render and then display
    the stacked pyramid image.
    :param pyr: A Gaussian or Laplacian pyramid.
    :param levels: The number of levels in pyr.
    :return: None.
    """
    image = render_pyramid(pyr, levels)
    plt.imshow(image, cmap='gray')
    plt.show()
```

```


def blend(pyr_im1, pyr_im2, pyr_mask):
    """
    A function that blends 2 Laplasian pyramids
    according to a mask pyramid.
    :param pyr_im1: First Laplasian pyramid.
    :param pyr_im2: Second Laplasian pyramid.
    :param pyr_mask: Mask pyramid.
    :return: The blended pyramid.
    """
    blended_pyr = []
    for i in range(len(pyr_mask)):
        blended_pyr.append(pyr_mask[i] * pyr_im1[i] +
                           (1 - pyr_mask[i]) * pyr_im2[i])
    return blended_pyr


def pyramid_blending(im1, im2, mask, max_levels, filter_size_im,
                     filter_size_mask):
    """
    A function that do a pyramid blending as described in the lecture.
    :param im1: First image to be blended.
    :param im2: Second image to be blended.
    :param mask: A boolean mask containing True and False representing
    which parts of im1 and im2 should appear in the resulting im_blend.
    :param max_levels: The max_levels parameter when we generating the
    Gaussian and Laplacian pyramids.
    :param filter_size_im:  The size of the Gaussian filter
    (an odd scalar that represents a squared filter) which defining the
    filter used in the construction of the Laplacian pyramids of im1 and im2.
    :param filter_size_mask: The size of the Gaussian filter
    (an odd scalar that represents a squared filter) which defining the
    filter used in the construction of the Gaussian pyramid of mask.
    :return: The blended image as a valid gray scale image in the range [0, 1].
    """

    pyr_im1, filter_vec = build_laplacian_pyramid(im1, max_levels,
                                                  filter_size_im)
    pyr_im2, filter_vec = build_laplacian_pyramid(im2, max_levels,
                                                  filter_size_im)
    pyr_mask, filter_vec_mask = build_gaussian_pyramid(mask.astype(np.float64),
                                                       max_levels,
                                                       filter_size_mask)

    blended_pyr = blend(pyr_im1, pyr_im2, pyr_mask)
    im_blend = laplacian_to_image(blended_pyr, filter_vec,
                                  [1] * len(blended_pyr))
    return stretch_image(im_blend)


def relpath(filename):
    """
    A function that concatenate the
    current path of the image with
    the given path.
    :param filename: The given path.
    :return: The concatenate path.
    """
    return os.path.join(os.path.dirname(__file__), filename)


def reshape_image(image, up, down, left, right):
    """
    A function that cuts parts of the images
    in the examples so they would look better.
    :param image: A given image.
    :param up: Number of rows to cut from the
    upper side.
    :param down: Number of rows to cut from the
    lower side.
    :param left: Number of rows to cut from the
    left side.
    :param right: Number of rows to cut from the
    right side.
    :return: The image after the cutting.
    """
    image = image[up: image.shape[0] - down]
    indexes_in_rows_left = np.arange(0, left)
    image = np.delete(image, indexes_in_rows_left, axis=1)
    indexes_in_rows_right = np.arange(image.shape[1] - right - 1,
                                      image.shape[1])
    return np.delete(image, indexes_in_rows_right, axis=1)


def show_images(im1, im2, mask, im_blend):
    """
    A function that plot the images of the
    examples in one plot.
    :param im1: The first image.
    :param im2: The second image.
    :param mask: The mask.
    :param im_blend: The blended image.
    :return: None.
    """
    fig = plt.figure()

    a1 = fig.add_subplot(221)
    a2 = fig.add_subplot(222)
    a3 = fig.add_subplot(223)
    a4 = fig.add_subplot(224)

    a1.imshow(im1)
    a2.imshow(im2)
    a3.imshow(mask, cmap='gray')
    a4.imshow(im_blend)

    a1.set_axis_off()
    a2.set_axis_off()
    a3.set_axis_off()
    a4.set_axis_off()

    plt.show()


def blending_example(image1, image2, mask, example):
    """
    A function that makes the blending for the examples.
    :param image1: The first image.
    :param image2: The second image.
    :param mask: The mask.
    :param example: 1 for blending_example1,
    2 for blending_example2.
    :return: im1, im2, mask, im_blend
    """
    im1 = read_image(image1, 2)
    im2 = read_image(image2, 2)
    mask = read_image(mask, 1)
    im_blend = im1.copy()

    red_pixels = pyramid_blending(im1[:, :, 0], im2[:, :, 0],
                                  mask, 3, 5, 3)
    green_pixels = pyramid_blending(im1[:, :, 1], im2[:, :, 1],
                                    mask, 3, 5, 3)
    blue_pixels = pyramid_blending(im1[:, :, 2], im2[:, :, 2],
                                   mask, 3, 5, 3)
    im_blend[:, :, 0] = red_pixels
    im_blend[:, :, 1] = green_pixels
    im_blend[:, :, 2] = blue_pixels

    if example == 2:
        im1 = reshape_image(im1, 75, 75, 200, 200)
        im2 = reshape_image(im2, 75, 75, 200, 200)
        mask = reshape_image(mask, 75, 75, 200, 200)
        im_blend = reshape_image(im_blend, 75, 75, 200, 200)
    else:
        im1 = reshape_image(im1, 100, 100, 100, 100)
        im2 = reshape_image(im2, 100, 100, 100, 100)
        mask = reshape_image(mask, 100, 100, 100, 100)
        im_blend = reshape_image(im_blend, 100, 100, 100, 100)

    show_images(im1, im2, mask, im_blend)

    return im1, im2, mask.astype(np.bool), im_blend


def blending_example1():
    """
    A function that perform pyramid blending
    on the first set of images.
    :return: The first image, the second image, the mask,
    the blended image.
    """
    refa = relpath("externals//final_refa.jpg")
    lafa = relpath("externals//final_lafa.jpg")
    mask = relpath("externals//final_mask_refa.png")
    return blending_example(refa, lafa, mask, 1)


def blending_example2():
    """
    A function that perform pyramid blending
    on the second set of images.
    :return: The first image, the second image, the mask,
    the blended image.
    """
    dolphin = relpath("externals//final_dolphin.jpg")
    water = relpath("externals//final_water.jpg")
    mask = relpath("externals//final_mask_dolphin.jpg")
    return blending_example(dolphin, water, mask, 2)
```

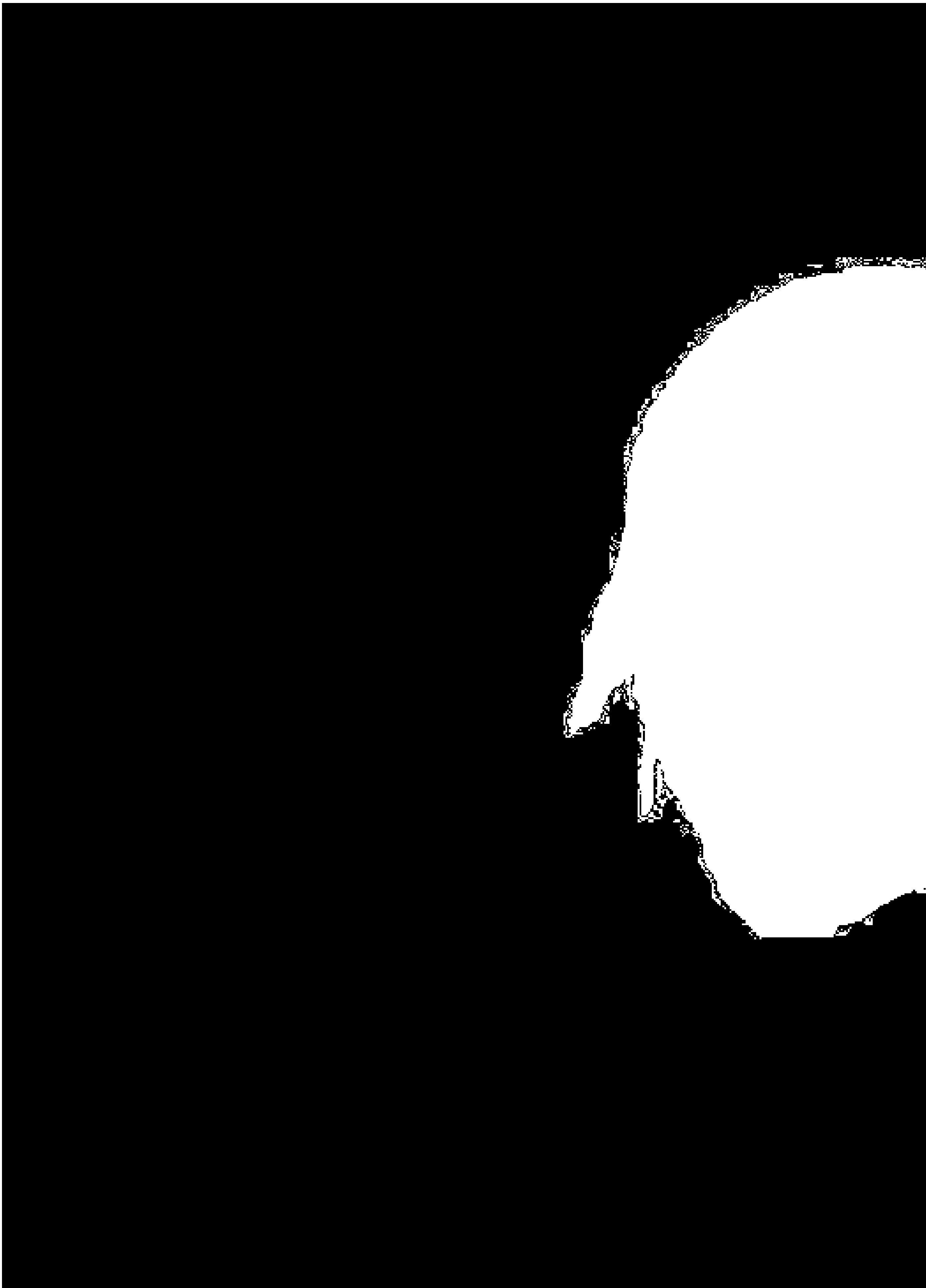
7 ex3/externals/final dolphin.jpg

8 ex3/externals/final lafa.jpg

9 ex3/externals/final mask dolphin.jpg



10 ex3/externals/final mask refa.png



11 ex3/externals/final refa.jpg

12 ex3/externals/final water.jpg