

# Contents

1	Basic Test Results	2
2	ex3/README.md	4
3	ex3/answer q1.txt	5
4	ex3/answer q2.txt	6
5	ex3/answer q3.txt	7
6	ex3/sol3.py	8
7	ex3/externals/final dolphin.jpg	15
8	ex3/externals/final lafa.jpg	16
9	ex3/externals/final mask dolphin.jpg	18
10	ex3/externals/final mask refa.png	19
11	ex3/externals/final refa.jpg	21
12	ex3/externals/final water.jpg	23

# 1 Basic Test Results

```
1  ex3/
2  ex3/README.md
3  ex3/answer_q1.txt
4  ex3/answer_q2.txt
5  ex3/answer_q3.txt
6  ex3/externals/
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
13 ex3/sol3.py
14 Ex3 Presubmission Script
15 =====
16
17
18     Disclaimer
19     -----
20     The purpose of this script is to make sure that your code is compliant
21     with the exercise API and some of the requirements
22     The script does not test the quality of your results.
23     Don't assume that passing this script will guarantee that you will get
24     a high grade in the exercise
25
26 === Check Submission ===
27
28 README file:
29
30 # ex3-yishai.hazi
31 sol3.py
32 externals:
33 final_dolphin.jpg
34 final_mask_dolphin.jpg
35 final_water.jpg
36 final_refa.jpg
37 final_mask_refa.png
38 final_lafa.jpg
39 answer_q1.txt
40 answer_q2.txt
41 answer_q3.txt
42 README.md
43
44
45 === Answers to questions ===
46
47 Answer to Q1:
48 By multiplying each level with a different value - we can control of "how much" we take from the frequencies.
49 For example, if we multiply the higher levels of the pyramids with small value, it means less high frequencies
50 in the final image.
51
52 Answer to Q2:
53 It looks like that if the filter size is bigger, it causes the blended image to be blur, thats probably because
54 at each reduction (in the pyramid) we blur a larger range of pixels.
55
56 Answer to Q3:
57 More levels that the pyramid has, results in a smoother and accurate image. The reason for that is that
58 each level "merge" different set of frequencies, so if the pyramid has more levels, more frequencies are merged. (the higher
59 === Load Student Library ===
```

```

60
61 Loading...
62
63 === Section 3.1 ===
64
65 Trying to build Gaussian pyramid...
66     Passed!
67 Checking Gaussian pyramid type and structure...
68     Passed!
69 Trying to build Laplacian pyramid...
70     Passed!
71 Checking Laplacian pyramid type and structure...
72     Passed!
73
74 === Section 3.2 ===
75
76 Trying to build Laplacian pyramid...
77     Passed!
78 Trying to reconstruct image from pyramid... (we are not checking for quality!)
79     Passed!
80 Checking reconstructed image type and structure...
81     Passed!
82
83 === Section 3.3 ===
84
85 Trying to build Gaussian pyramid...
86     Passed!
87 Trying to render pyramid to image...
88     Passed!
89 Checking structure of returned image...
90     Passed!
91 Trying to display image... (if DISPLAY env var not set, assumes running w/o screen)
92     Passed!
93
94 === Section 4 ===
95
96 Trying to blend two images... (we are not checking the quality!)
97     Passed!
98 Checking size of blended image...
99     Passed!
100 Tring to call blending_example1()...
101     Passed!
102 Checking types of returned results...
103     Passed!
104 Tring to call blending_example2()...
105 /tmp/bodek.woFmwe/impr/ex3/yishai.hazi/gitsub/testdir/test:8: DeprecationWarning: `imread` is deprecated!
106 `imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
107 Use ``imageio.imread`` instead.
108     im = imread(filename)
109     Passed!
110 Checking types of returned results...
111     Passed!
112
113
114 === Presubmission Completed Successfully ===
115
116
117 Please go over the output and verify that there were no failures / warnings.
118 Remember that this script tested only some basic technical aspects of your implementation.
119 It is your responsibility to make sure your results are actually correct and not only
120 technically valid.

```

## 2 ex3/README.md

```
1  # ex3-yishai.hazi
2  sol3.py
3  externals:
4  final_dolphin.jpg
5  final_mask_dolphin.jpg
6  final_water.jpg
7  final_refa.jpg
8  final_mask_refa.png
9  final_lafa.jpg
10 answer_q1.txt
11 answer_q2.txt
12 answer_q3.txt
13 README.md
```

## 3 ex3/answer q1.txt

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

## 4 ex3/answer q2.txt

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

## 5 ex3/answer q3.txt

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

## 6 ex3/sol3.py

```
1  import os
2  import numpy as np
3  import skimage.color as color
4  import imageio
5  import scipy.ndimage.filters
6  import matplotlib.pyplot as plt
7
8
9  def normalize(image):
10     """
11     A function that normalize the elements inside
12     a numpy matrix, and change their type to float64.
13     :param image: a numpy matrix
14     :return: the matrix with all elements normalized
15     to the range: [0, 1].
16     """
17     image = image.astype('float64')
18     return image / 255
19
20
21  def read_image(filename, representation):
22     """
23     A function that reads a file according
24     to a given representation.
25     :param filename: The file name.
26     :param representation: 1 represents rgb,
27     2 represents gray_scale.
28     :return: An image according to the
29     given representation.
30     """
31     image = imageio.imread(filename)
32     if representation == 2 or image.ndim != 3:
33         if np.any(image > 1):
34             return normalize(image)
35     elif representation == 1:
36         return color.rgb2gray(image)
37
38
39  def normalize_gaussian_filter(filter_vec):
40     """
41     A function that normalize the gaussian
42     filter.
43     :param filter_vec: The filter.
44     :return: the normalized filter.
45     """
46     return filter_vec / np.sum(filter_vec)
47
48
49  def build_gaussian_filter(filter_size):
50     """
51     A function that build a gaussian filter
52     according to a given size.
53     :param filter_size: odd integer that
54     represents the desired size.
55     :return: The gaussian filter as
56     a numpy array with shape: [1, filter_size].
57     (without normalization)
58     """
59     number_of = filter_size - 2
```



```

60     if filter_size > 1:
61         small_kernel = np.array([1, 1])
62     else:
63         small_kernel = np.array([1])
64     the_filter = small_kernel
65     for i in range(number_of):
66         the_filter = np.convolve(the_filter, small_kernel)
67     return np.reshape(the_filter, (1, filter_size))
68
69
70 def convolve(im, filter_vec):
71     """
72     A function that convolve an image with
73     a given filter.
74     :param im: The given image as a grayscale
75     image with double values in [0, 1].
76     :param filter_vec: The filter as a row vector.
77     :return: The image after the deletion.
78     """
79     im_after_rows_convolution = scipy.ndimage.filters.convolve(
80         im, filter_vec)
81     return scipy.ndimage.filters.convolve(
82         im_after_rows_convolution, np.transpose(filter_vec))
83
84
85 def delete_elements(im):
86     """
87     A function that delete all the elements
88     in the odd indexes. (as well as all the
89     odd rows)
90     :param im: The given image.
91     :return: The image after the deletion.
92     """
93     odd_indexes_of_rows = np.arange(1, im.shape[0] + 1, 2)
94     odd_indexes_of_cols = np.arange(1, im.shape[1] + 1, 2)
95     im_after_horizontal_clean = np.delete(im,
96                                           odd_indexes_of_rows, axis=0)
97     im_after_vertical_clean = np.delete(im_after_horizontal_clean,
98                                         odd_indexes_of_cols,
99                                         axis=1)
100     return im_after_vertical_clean
101
102
103 def reduce(im, filter_vec):
104     """
105     A function that creates a smaller image
106     (smaller by factor of 2) of the given image.
107     :param im: The given image.
108     :param filter_vec: The given filter
109     that blur the image.
110     :return: The smaller image.
111     """
112     im_after_convolution = convolve(im, filter_vec)
113     return delete_elements(im_after_convolution)
114
115
116 def build_gaussian_pyramid(im, max_levels, filter_size):
117     """
118     A function that construct a Gaussian
119     pyramid of a given image.
120     :param im: A grayscale image with double
121     values in [0, 1].
122     :param max_levels: The maximal number of levels1
123     in the resulting pyramid.
124     :param filter_size: The size of the Gaussian filter
125     (an odd scalar that represents a squared filter) to be used
126     in constructing the pyramid filter.
127     :return: The resulting pyramid pyr as a standard python array,

```

```

128     and filter_vec - which is a row vector of shape (1, filter_size).
129     """
130     filter_vec = build_gaussian_filter(filter_size)
131     filter_vec = normalize_gaussian_filter(filter_vec)
132     pyr = [im]
133     smallest_image = im
134     max_levels -= 1
135     while max_levels >= 1 and smallest_image.shape[0] > 16 \
136           and smallest_image.shape[1] > 16:
137         smallest_image = reduce(smallest_image, filter_vec)
138         pyr.append(smallest_image)
139         max_levels -= 1
140     return pyr, filter_vec
141
142
143 def zero_pad(im):
144     """
145     A function that expand an image by
146     adding zeros in the odd indexes (and odd
147     rows)
148     :param im: The given image as a grayscale
149     image with double values in [0, 1].
150     :return: The image after extension.
151     """
152     odd_indexes_in_rows = np.arange(1, im.shape[0] + 1)
153     odd_indexes_in_cols = np.arange(1, im.shape[1] + 1)
154     im = np.insert(im, odd_indexes_in_rows, 0, axis=0)
155     im = np.insert(im, odd_indexes_in_cols, 0, axis=1)
156     return im
157
158
159 def expand(im, filter_vec):
160     """
161     A function that expand an image by
162     adding zeros to it and then blur the
163     result with the given filter.
164     :param im: The given image as a grayscale
165     image with double values in [0, 1].
166     :param filter_vec:
167     :return: The given filter we use for blur.
168     (a row vector of shape (1, filter_size).)
169     """
170     im = zero_pad(im)
171     im_after_convolve = convolve(im, filter_vec)
172     return im_after_convolve
173
174
175 def build_laplacian_pyramid(im, max_levels, filter_size):
176     """
177     A function that construct a Laplacian pyramid
178     of a given image.
179     :param im: A grayscale image with double
180     values in [0, 1].
181     :param max_levels: The maximal number of levels1
182     in the resulting pyramid.
183     :param filter_size: The size of the Gaussian filter
184     (an odd scalar that represents a squared filter) to be used
185     in constructing the pyramid filter.
186     :return: The resulting pyramid pyr as a standard python array,
187     and filter_vec - which is row vector of shape (1, filter_size).
188     """
189     gaussian_pyr, filter_vec = build_gaussian_pyramid(im, max_levels,
190                                                       filter_size)
191     pyr = []
192     for i in range(len(gaussian_pyr) - 1):
193         pyr.append(gaussian_pyr[i] - expand(gaussian_pyr[i + 1],
194                                           filter_vec * 2))
195     pyr.append(gaussian_pyr[-1])

```

```

196     return pyr, filter_vec
197
198
199 def laplacian_to_image(lpyr, filter_vec, coeff):
200     """
201     A function that construct an image from its
202     Laplacian Pyramid.
203     :param lpyr: The Laplacian pyramid
204     :param filter_vec: The filter that is generated
205     by the function: "build_laplacian_pyramid".
206     :param coeff: A python list. Each level i of the
207     laplacian pyramid is multiplied by its corresponding
208     coefficient coeff[i].
209     :return: The original image.
210     """
211     for i in range(len(lpyr)):
212         lpyr[i] = lpyr[i] * coeff[i]
213     image = lpyr[-1]
214     for i in range(len(lpyr) - 1, 0, -1):
215         image = expand(image, filter_vec * 2) + lpyr[i - 1]
216     return image
217
218
219 def stretch_image(image):
220     """
221     A function that stretch the values of
222     the given image to the range: [0, 1].
223     :param image: The given image.
224     :param min_value: A given minimum.
225     :param max_value: A given maximum.
226     :return: The stretched image.
227     """
228     return (image - np.min(image)) / (np.max(image) - np.min(image))
229
230
231 def render_pyramid(pyr, levels):
232     """
233     A function that creates a single black image
234     in which the pyramid levels of the given pyramid
235     pyr are stacked horizontally.
236     :param pyr: The given pyramid (a Gaussian
237     or Laplacian pyramid)
238     :param levels: the number of levels of the pyramid.
239     :return: A black image in which the pyramid levels of the
240     given pyramid pyr are stacked horizontally.
241     """
242     res = stretch_image(pyr[0])
243     for i in range(1, min(levels, len(pyr))):
244         black_image = [[0] * pyr[i].shape[1]] * pyr[i].shape[0] * (2 ** i - 1)
245         stretched_image = stretch_image(pyr[i])
246         smaller_image_with_black_part = np.vstack(
247             (stretched_image, black_image))
248         res = np.hstack((res, smaller_image_with_black_part))
249     return res
250
251
252 def display_pyramid(pyr, levels):
253     """
254     A function that use render_pyramid to
255     internally render and then display
256     the stacked pyramid image.
257     :param pyr: A Gaussian or Laplacian pyramid.
258     :param levels: The number of levels in pyr.
259     :return: None.
260     """
261     image = render_pyramid(pyr, levels)
262     plt.imshow(image, cmap='gray')
263     plt.show()

```

```

264
265
266 def blend(pyr_im1, pyr_im2, pyr_mask):
267     """
268     A function that blends 2 Laplasian pyramids
269     according to a mask pyramid.
270     :param pyr_im1: First Laplasian pyramid.
271     :param pyr_im2: Second Laplasian pyramid.
272     :param pyr_mask: Mask pyramid.
273     :return: The blended pyramid.
274     """
275     blended_pyr = []
276     for i in range(len(pyr_mask)):
277         blended_pyr.append(pyr_mask[i] * pyr_im1[i] +
278                             (1 - pyr_mask[i]) * pyr_im2[i])
279     return blended_pyr
280
281
282 def pyramid_blending(im1, im2, mask, max_levels, filter_size_im,
283                     filter_size_mask):
284     """
285     A function that do a pyramid blending as described in the lecture.
286     :param im1: First image to be blended.
287     :param im2: Second image to be blended.
288     :param mask: A boolean mask containing True and False representing
289     which parts of im1 and im2 should appear in the resulting im_blend.
290     :param max_levels: The max_levels parameter when we generating the
291     Gaussian and Laplacian pyramids.
292     :param filter_size_im: The size of the Gaussian filter
293     (an odd scalar that represents a squared filter) which defining the
294     filter used in the construction of the Laplacian pyramids of im1 and im2.
295     :param filter_size_mask: The size of the Gaussian filter
296     (an odd scalar that represents a squared filter) which defining the
297     filter used in the construction of the Gaussian pyramid of mask.
298     :return: The blended image as a valid gray scale image in the range [0, 1].
299     """
300
301     pyr_im1, filter_vec = build_laplacian_pyramid(im1, max_levels,
302                                                    filter_size_im)
303     pyr_im2, filter_vec = build_laplacian_pyramid(im2, max_levels,
304                                                    filter_size_im)
305     pyr_mask, filter_vec_mask = build_gaussian_pyramid(mask.astype(np.float64),
306                                                         max_levels,
307                                                         filter_size_mask)
308
309     blended_pyr = blend(pyr_im1, pyr_im2, pyr_mask)
310     im_blend = laplacian_to_image(blended_pyr, filter_vec,
311                                   [1] * len(blended_pyr))
312     return stretch_image(im_blend)
313
314
315 def relpath(filename):
316     """
317     A function that concatenate the
318     current path of the image with
319     the given path.
320     :param filename: The given path.
321     :return: The concatenate path.
322     """
323     return os.path.join(os.path.dirname(__file__), filename)
324
325
326 def reshape_image(image, up, down, left, right):
327     """
328     A function that cuts parts of the images
329     in the examples so they would look better.
330     :param image: A given image.
331     :param up: Number of rows to cut from the

```

```

332     upper side.
333     :param down: Number of rows to cut from the
334     lower side.
335     :param left: Number of rows to cut from the
336     left side.
337     :param right: Number of rows to cut from the
338     right side.
339     :return: The image after the cutting.
340     """
341     image = image[up: image.shape[0] - down]
342     indexes_in_rows_left = np.arange(0, left)
343     image = np.delete(image, indexes_in_rows_left, axis=1)
344     indexes_in_rows_right = np.arange(image.shape[1] - right - 1,
345                                     image.shape[1])
346     return np.delete(image, indexes_in_rows_right, axis=1)
347
348
349 def show_images(im1, im2, mask, im_blend):
350     """
351     A function that plot the images of the
352     examples in one plot.
353     :param im1: The first image.
354     :param im2: The second image.
355     :param mask: The mask.
356     :param im_blend: The blended image.
357     :return: None.
358     """
359     fig = plt.figure()
360
361     a1 = fig.add_subplot(221)
362     a2 = fig.add_subplot(222)
363     a3 = fig.add_subplot(223)
364     a4 = fig.add_subplot(224)
365
366     a1.imshow(im1)
367     a2.imshow(im2)
368     a3.imshow(mask, cmap='gray')
369     a4.imshow(im_blend)
370
371     a1.set_axis_off()
372     a2.set_axis_off()
373     a3.set_axis_off()
374     a4.set_axis_off()
375
376     plt.show()
377
378
379 def blending_example(image1, image2, mask, example):
380     """
381     A function that makes the blending for the examples.
382     :param image1: The first image.
383     :param image2: The second image.
384     :param mask: The mask.
385     :param example: 1 for blending_example1,
386     2 for blending_example2.
387     :return: im1, im2, mask, im_blend
388     """
389     im1 = read_image(image1, 2)
390     im2 = read_image(image2, 2)
391     mask = read_image(mask, 1)
392     im_blend = im1.copy()
393
394     red_pixels = pyramid_blending(im1[:, :, 0], im2[:, :, 0],
395                                 mask, 3, 5, 3)
396     green_pixels = pyramid_blending(im1[:, :, 1], im2[:, :, 1],
397                                 mask, 3, 5, 3)
398     blue_pixels = pyramid_blending(im1[:, :, 2], im2[:, :, 2],
399                                 mask, 3, 5, 3)

```

```

400     im_blend[:, :, 0] = red_pixels
401     im_blend[:, :, 1] = green_pixels
402     im_blend[:, :, 2] = blue_pixels
403
404     if example == 2:
405         im1 = reshape_image(im1, 75, 75, 200, 200)
406         im2 = reshape_image(im2, 75, 75, 200, 200)
407         mask = reshape_image(mask, 75, 75, 200, 200)
408         im_blend = reshape_image(im_blend, 75, 75, 200, 200)
409     else:
410         im1 = reshape_image(im1, 100, 100, 100, 100)
411         im2 = reshape_image(im2, 100, 100, 100, 100)
412         mask = reshape_image(mask, 100, 100, 100, 100)
413         im_blend = reshape_image(im_blend, 100, 100, 100, 100)
414
415     show_images(im1, im2, mask, im_blend)
416
417     return im1, im2, mask.astype(np.bool), im_blend
418
419 def blending_example1():
420     """
421     A function that perform pyramid blending
422     on the first set of images.
423     :return: The first image, the second image, the mask,
424     the blended image.
425     """
426     refa = relpath("externals//final_refa.jpg")
427     lafa = relpath("externals//final_lafa.jpg")
428     mask = relpath("externals//final_mask_refa.png")
429     return blending_example(refa, lafa, mask, 1)
430
431
432
433 def blending_example2():
434     """
435     A function that perform pyramid blending
436     on the second set of images.
437     :return: The first image, the second image, the mask,
438     the blended image.
439     """
440     dolphin = relpath("externals//final_dolphin.jpg")
441     water = relpath("externals//final_water.jpg")
442     mask = relpath("externals//final_mask_dolphin.jpg")
443     return blending_example(dolphin, water, mask, 2)

```

7 ex3/externals/final dolphin.jpg



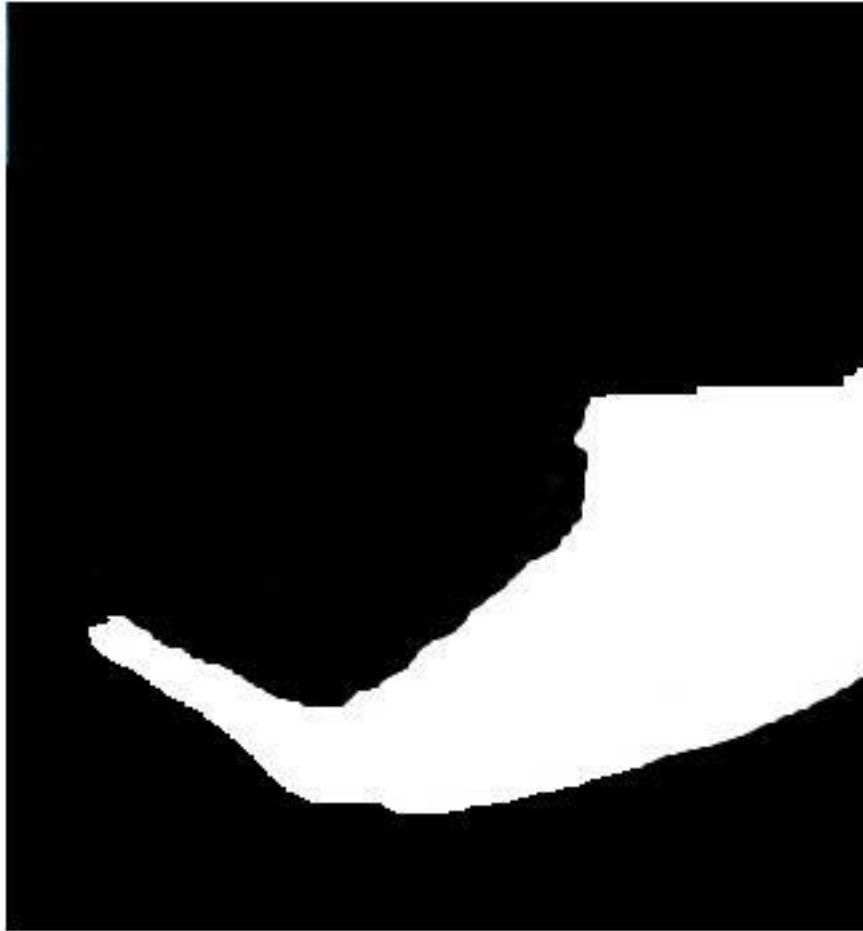




8 ex3/externals/final lafa.jpg

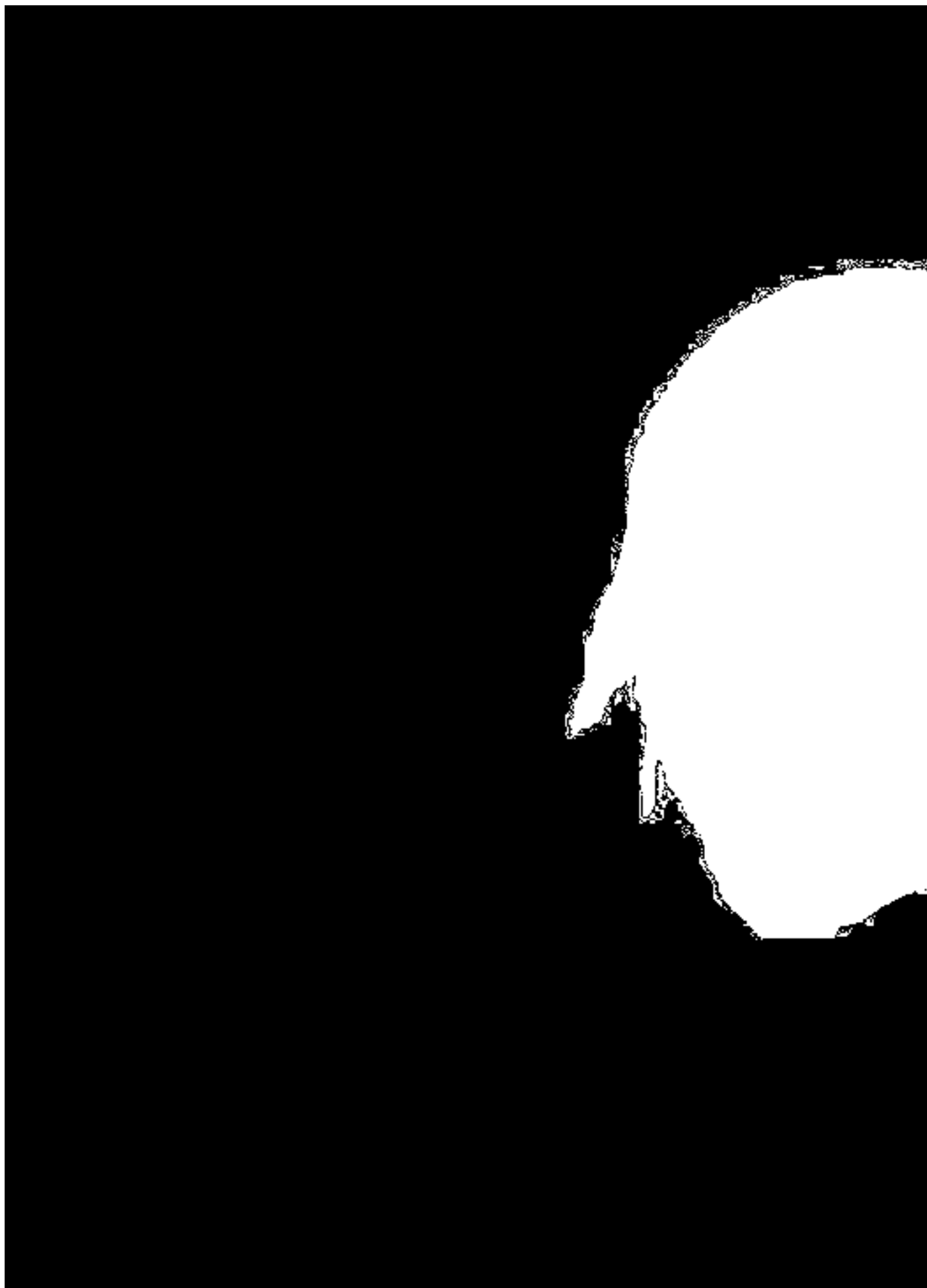


9 ex3/externals/final mask dolphin.jpg





10 ex3/externals/final mask refa.png





11 ex3/externals/final refa.jpg



12 ex3/externals/final water.jpg

