

# 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/daft orig.jpg	13
8	ex3/externals/dark mask.jpg	15
9	ex3/externals/dark orig.jpg	16
10	ex3/externals/freddie.jpg	17
11	ex3/externals/freddie mask.png	19
12	ex3/externals/tal.jpg	21

# 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/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
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 tal.porezky
31 sol3.py
32 answer_q1.txt
33 answer_q2.txt
34 answer_q3.txt
35
36 === Answers to questions ===
37
38 Answer to Q1:
39 The dark level (darkness of the image) changes. Each level is more blurry than
40 the one before it, so it contains lower frequencies. If the coefficient of a
41 certain level is higher then the image will be darker at the appropriate
42 frequencies.
43 We can think of it like an equalizer of the different images of the pyramid.
44
45 Answer to Q2:
46 Different image filters are responsible for the "smoothness" changes between
47 the two images blended.
48 Higher filter size will result a larger "gaussian" which will be used as the
49 smoothing tool. The higher the filter size is, the more smooth will be the
50 change at the points where the two images blend.
51 Smaller filter size will result a smaller "gaussian" which will lead to a
52 rougher change (not so smooth) at the points where the two images blend.
53
54 Answer to Q3:
55 The higher the number of pyramid levels the less sharp the result image (the
56 blended image) will be.
57
58 === Load Student Library ===
59
```

```

60 Loading...
61
62 === Section 3.1 ===
63
64 Trying to build Gaussian pyramid...
65     Passed!
66 Checking Gaussian pyramid type and structure...
67     Passed!
68 Trying to build Laplacian pyramid...
69     Passed!
70 Checking Laplacian pyramid type and structure...
71     Passed!
72
73 === Section 3.2 ===
74
75 Trying to build Laplacian pyramid...
76     Passed!
77 Trying to reconstruct image from pyramid... (we are not checking for quality!)
78     Passed!
79 Checking reconstructed image type and structure...
80     Passed!
81
82 === Section 3.3 ===
83
84 Trying to build Gaussian pyramid...
85     Passed!
86 Trying to render pyramid to image...
87     Passed!
88 Checking structure of returned image...
89     Passed!
90 Trying to display image... (if DISPLAY env var not set, assumes running w/o screen)
91     Passed!
92
93 === Section 4 ===
94
95 Trying to blend two images... (we are not checking the quality!)
96     Passed!
97 Checking size of blended image...
98     Passed!
99 Tring to call blending_example1()...
100     Passed!
101 Checking types of returned results...
102     Passed!
103 Tring to call blending_example2()...
104 /tmp/bodek.Tz1Ldl/impr/ex3/tal.porezky/gitsub/testdir/test:8: DeprecationWarning: `imread` is deprecated!
105 `imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
106 Use ``imageio.imread`` instead.
107     im = imread(filename)
108     Passed!
109 Checking types of returned results...
110     Passed!
111
112
113 === Presubmission Completed Successfully ===
114
115
116 Please go over the output and verify that there were no failures / warnings.
117 Remember that this script tested only some basic technical aspects of your implementation.
118 It is your responsibility to make sure your results are actually correct and not only
119 technically valid.

```

## 2 ex3/README.md

```
1  tal.porezky
2  sol3.py
3  answer_q1.txt
4  answer_q2.txt
5  answer_q3.txt
```

### 3 ex3/answer q1.txt

```
1 The dark level (darkness of the image) changes. Each level is more blurry than
2 the one before it, so it contains lower frequencies. If the coefficient of a
3 certain level is higher then the image will be darker at the appropriate
4 frequencies.
5 We can think of it like an equalizer of the different images of the pyramid.
```

## 4 ex3/answer q2.txt

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

## 5 ex3/answer q3.txt

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

## 6 ex3/sol3.py

```
1  # ----- Imports -----
2  import matplotlib.pyplot as plt
3  from skimage.color import rgb2gray
4  import numpy as np
5  from scipy.ndimage.filters import convolve as _convolve
6  from imageio import imread
7  import os
8
9  # ----- Constants -----
10 MIN_IMG_SIZE_IN_EACH_AXIS = 16
11 EXPEND_FACTOR = 2
12
13
14 # ----- Helper functions -----
15 def read_image(filename, representation):
16     """
17     function which reads an image file and converts it into a given
18     representation.
19     :param filename: the filename of an image on disk (could be grayscale or
20     RGB).
21     :param representation: representation code, either 1 or 2 defining
22     whether the output should be a grayscaleimage (1) or an RGB image (2).
23     If the input image is grayscale, we won't call it with representation = 2.
24     :return: rgb or grayscale img
25     """
26     image = imread(filename)
27     new_image = image.astype(np.float64)
28     new_image /= 255
29     if representation == 1:
30         new_image = rgb2gray(new_image)
31     return new_image
32
33
34 def _im_downsample(im, blur_filter):
35     """
36     downsamples the image according to the blur filter given and takes
37     every even pixel in the image.
38     :param im: np.array image
39     :param blur_filter: np.array of size (1,) of the filter
40     :return: smaller sample.
41     """
42     im = _convolve(im, blur_filter, mode='reflect')
43     im = _convolve(im, blur_filter.T, mode='reflect')
44     im = im[::2, ::2]
45     return im
46
47
48 def _im_expand(im, blur_filter):
49     """
50     expands the image according to the blur filter. Makes the image twice
51     as bigger.
52     :param im: np.array image
53     :param blur_filter: np.array of size (1,) of the filter
54     :return: bigger image.
55     """
56     expended_im = np.zeros(
57         (im.shape[0] * EXPEND_FACTOR, im.shape[1] * EXPEND_FACTOR))
58     expended_im[1::2, 1::2] = im
59     blur_filter = blur_filter * EXPEND_FACTOR
```



```

60     expended_im = _convolve(_convolve(expended_im, blur_filter),
61                               blur_filter.T)
62     return expended_im
63
64
65 def _get_binomial_coefficients(size):
66     """
67     creates gaussian vector of the given size. size must be odd.
68     :param size: odd integer
69     :return: gaussian vector of the given size
70     """
71     binomial_coefficients = np.array([1, 1], dtype=np.float64)
72     for _ in range(size - 2):
73         binomial_coefficients = np.convolve([1, 1], binomial_coefficients)
74     binomial_coefficients = binomial_coefficients[np.newaxis, :]
75     return binomial_coefficients
76
77
78 def _reshape_dims_of_imgs(org_im, other_im):
79     """
80     makes fixes to other_im so it shape will fit org_im
81     :param org_im: original image.
82     :param other_im: image which shape to fix.
83     :return: other_image with fixed size.
84     """
85     if org_im.shape[0] != other_im.shape[0]:
86         other_im = other_im[: -1, :]
87     if org_im.shape[1] != other_im.shape[1]:
88         other_im = other_im[:, : -1]
89     return other_im
90
91
92 def _strech_im(im):
93     """
94     streches the image so it's values will fit 0-1 scale.
95     :param im: original image.
96     :return: image with values between 0 and 1.
97     """
98     return (im - np.min(im)) / (np.max(im) - np.min(im))
99
100
101 # ----- 3.1: Image Pyramids -----
102 def build_gaussian_pyramid(im, max_levels, filter_size):
103     """
104     constructs a gaussian pyramid.
105     :param im: grayscale image with double values in [0,1].
106     :param max_levels: maximal number of levels in the resulting pyramid.
107     :param filter_size: the size of the gaussian filter
108     :return: pyr, filter_vec. pyr - python array where the elements are
109             images with different sizes. filter_vec - the gaussian filter used.
110     """
111     blur_filter_not_normalized = _get_binomial_coefficients(filter_size)
112     blur_filter = blur_filter_not_normalized / np.sum(
113         blur_filter_not_normalized)
114
115     pyr = list()
116     curr_im = im
117     while ((curr_im.shape[0] >= MIN_IMG_SIZE_IN_EACH_AXIS) and
118           (curr_im.shape[1] >= MIN_IMG_SIZE_IN_EACH_AXIS) and
119           (len(pyr) < max_levels)):
120         pyr.append(curr_im)
121         curr_im = _im_downsample(curr_im, blur_filter)
122
123     return pyr, blur_filter
124
125
126 def build_laplacian_pyramid(im, max_levels, filter_size):
127     """

```

```

128     constructs a laplacian pyramid.
129     :param im: grayscale image with double values in [0,1].
130     :param max_levels: maximal number of levels in the resulting pyramid.
131     :param filter_size: the size of the gaussian filter
132     :return: pyr, filter_vec. pyr - python array where the elements are
133     images with different sizes. filter_vec - the gaussian filter used.
134     """
135     gauss_pyr_list, gauss_blur_filter = build_gaussian_pyramid(im,
136                                                                max_levels,
137                                                                filter_size)
138     laplace_pyr = list()
139     for pyr_idx in range(len(gauss_pyr_list) - 1):
140         curr_gauss_img = gauss_pyr_list[pyr_idx]
141         expended_next_gauss_im = _im_expand(gauss_pyr_list[pyr_idx + 1],
142                                             gauss_blur_filter)
143         # todo maybe i should delete these later.
144         expended_next_gauss_im = _reshape_dims_of_imgs(curr_gauss_img,
145                                                         expended_next_gauss_im)
146         laplace_pyr.append(gauss_pyr_list[pyr_idx] - expended_next_gauss_im)
147     laplace_pyr.append(gauss_pyr_list[-1])
148     return laplace_pyr, gauss_blur_filter
149
150
151 # ----- 3.2: Laplacian pyramid reconstruction -----
152
153 def laplacian_to_image(lpyr, filter_vec, coeff):
154     """
155     reconstructes image from its aplacian pyramid.
156     :param lpyr: return value of the build_x_pyramid function
157     :param filter_vec: return value of the build_x_pyramid function
158     :param coeff: python list with length same as the number of levels in
159     the pyramid lpyr.
160     :return: image from the lalpacian pyramid.
161     """
162     for i in range(len(coeff)):
163         lpyr[i] = lpyr[i] * coeff[i]
164     im = lpyr[-1]
165     for i in range(len(lpyr) - 1, 0, -1):
166         expended_im = _reshape_dims_of_imgs(lpyr[i - 1],
167                                             _im_expand(im, filter_vec))
168         im = (lpyr[i - 1] + expended_im)
169     return im
170
171
172 # ----- 3.3: Pyramid display -----
173 def render_pyramid(pyr, levels):
174     """
175     constructs image from the 'levels' elements in pyr.
176     :param pyr: return value of the build_x_pyramid function
177     :param levels: how many pictures there will be in the image.
178     :return: image.
179     """
180     im = _strech_im(pyr[0])
181     for i in range(1, min(len(pyr), levels)):
182         difference_y = pyr[0].shape[0] - pyr[i].shape[0]
183         im = np.hstack((im, np.pad(_strech_im(pyr[i]),
184                                   ((0, difference_y),
185                                    (0, 0)),
186                                   'constant'))))
187     return im
188
189
190 def display_pyramid(pyr, levels):
191     """
192     displays the image built in render_pyramid
193     :param pyr: return value of the build_x_pyramid function
194     :param levels: how many pictures there will be in the image.
195     :return: None

```

```

196     """
197     res = render_pyramid(pyr, levels)
198     plt.figure()
199     plt.imshow(res, cmap='gray')
200     plt.show()
201     return
202
203
204 # ----- 4: Pyramid blending -----
205 def pyramid_blending(im1, im2, mask, max_levels, filter_size_im,
206                     filter_size_mask):
207     """
208     pyramid blending as described in the lecture.
209     :param im1: grayscale image to blend
210     :param im2: grayscale image to blend
211     :param mask: boolean mask
212     :param max_levels: parameter generated from the gaussian and laplacian
213     pyramids.
214     :param filter_size_im: size of the gaussian filter (ood integer)
215     :param filter_size_mask: size of the gaussian filter used for the mask.
216     :return: the blended image.
217     """
218     assert (im1.shape == im2.shape == mask.shape)
219     L1, filter_1 = build_laplacian_pyramid(im1,
220                                           max_levels,
221                                           filter_size_im)
222     L2, filter_2 = build_laplacian_pyramid(im2,
223                                           max_levels,
224                                           filter_size_im)
225     Gm, filter_m = build_gaussian_pyramid(mask.astype(np.float64),
226                                           max_levels,
227                                           filter_size_mask)
228     Lout = list()
229     for k in range(len(L1)):
230         Lout.append(Gm[k] * L1[k] + (1 - Gm[k]) * L2[k])
231     coeff = np.ones(len(L1)) * 1.0
232     im = laplacian_to_image(Lout, filter_1, coeff)
233     im = np.clip(im, 0, 1)
234
235     return im
236
237
238 # ----- 5: Your blending examples -----
239
240 def relpath(filename):
241     """
242     fucntion provided by you.
243     """
244     return os.path.join(os.path.dirname(__file__), filename)
245
246
247 def _color_blending(im1, im2, mask, max_levels, filter_size_im,
248                    filter_size_mask):
249     """
250     helper function for blending each color (red, green, blue) of the image
251     :param im1: grayscale image to blend
252     :param im2: grayscale image to blend
253     :param mask: boolean mask
254     :param max_levels: parameter generated from the gaussian and laplacian
255     pyramids.
256     :param filter_size_im: size of the gaussian filter (ood integer)
257     :param filter_size_mask: size of the gaussian filter used for the mask.
258     :return: the blended image.
259     """
260     r = pyramid_blending(im1[:, :, 0], im2[:, :, 0], mask, max_levels,
261                          filter_size_im, filter_size_mask)
262     g = pyramid_blending(im1[:, :, 1], im2[:, :, 1], mask, max_levels,
263                          filter_size_im, filter_size_mask)

```

```

264     b = pyramid_blending(im1[:, :, 2], im2[:, :, 2], mask, max_levels,
265                           filter_size_im, filter_size_mask)
266     blended_im = np.empty(im1.shape)
267     blended_im[:, :, 0] = r
268     blended_im[:, :, 1] = g
269     blended_im[:, :, 2] = b
270     return blended_im
271
272
273 def _plot_4_images(im1, im2, mask, blended, gray_result):
274     """
275     plots together the 4 images: im1, im2, mask and blended.
276     :param im1: grayscale image to blend
277     :param im2: grayscale image to blend
278     :param mask: boolean mask
279     :param blended: blended image created
280     :param gray_result: True or False.
281     :return: none
282     """
283     plt.figure()
284     plt.subplot(2, 2, 1)
285     plt.imshow(im1)
286     plt.subplot(2, 2, 2)
287     plt.imshow(im2)
288     plt.subplot(2, 2, 3)
289     plt.imshow(mask, cmap='gray')
290     plt.subplot(2, 2, 4)
291     if gray_result:
292         plt.imshow(blended, cmap='gray')
293     else:
294         plt.imshow(blended)
295     plt.show()
296
297
298 def blending_example1():
299     """
300     my own blending examples. This function puts my head on Freddie
301     Mercury's head.
302     :return: im1, im2, mask and the blended image.
303     """
304     im1 = read_image(relpath('externals/freddie.jpg'), 2)
305     im2 = read_image(relpath('externals/tal.jpg'), 2)
306     mask = read_image(relpath('externals/freddie_mask.png'), 1)
307     mask = mask > 0.5
308     blended = rgb2gray(np.clip(_color_blending(im1, im2, mask, 7, 15, 15),
309                                   0, 1))
310     _plot_4_images(im1, im2, mask, blended, True)
311     return im1, im2, mask.astype(np.bool), blended
312
313
314 def blending_example2():
315     """
316     my own blending examples. This function combines to music elements of
317     artists which I really like - Pink Floyd's album "Dark side of the
318     moon" and photos of "Daft Punk" robots.
319     :return: im1, im2, mask and the blended image.
320     """
321     im1 = read_image(relpath('externals/daft_orig.jpg'), 2)
322     im2 = read_image(relpath('externals/dark_orig.jpg'), 2)
323     mask = read_image(relpath('externals/dark_mask.jpg'), 1)
324     mask = mask < 0.5
325     blended = np.clip(_color_blending(im1, im2, mask, 7, 15, 15), 0, 1)
326     _plot_4_images(im1, im2, mask, blended, False)
327     return im1, im2, mask.astype(np.bool), blended

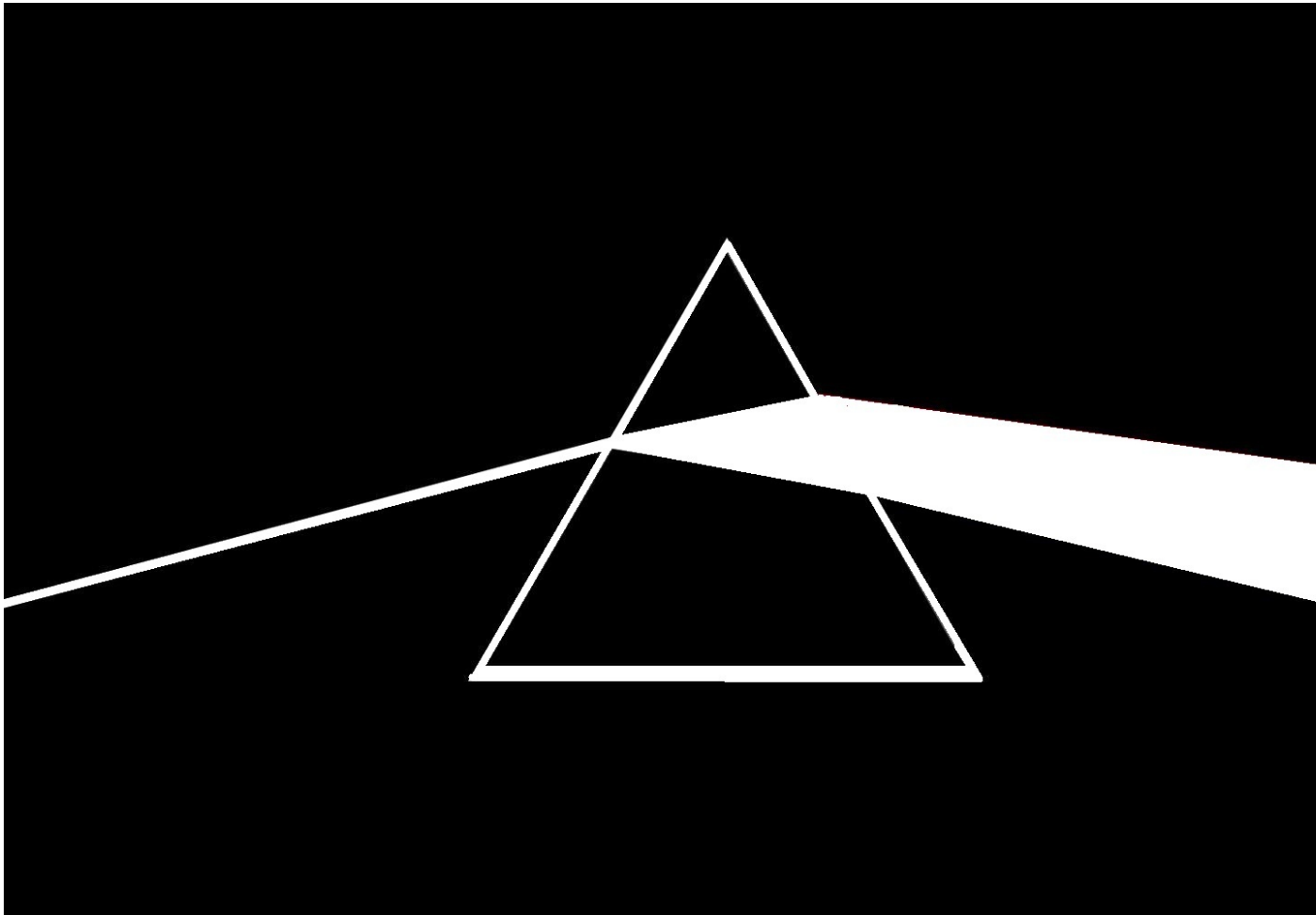
```



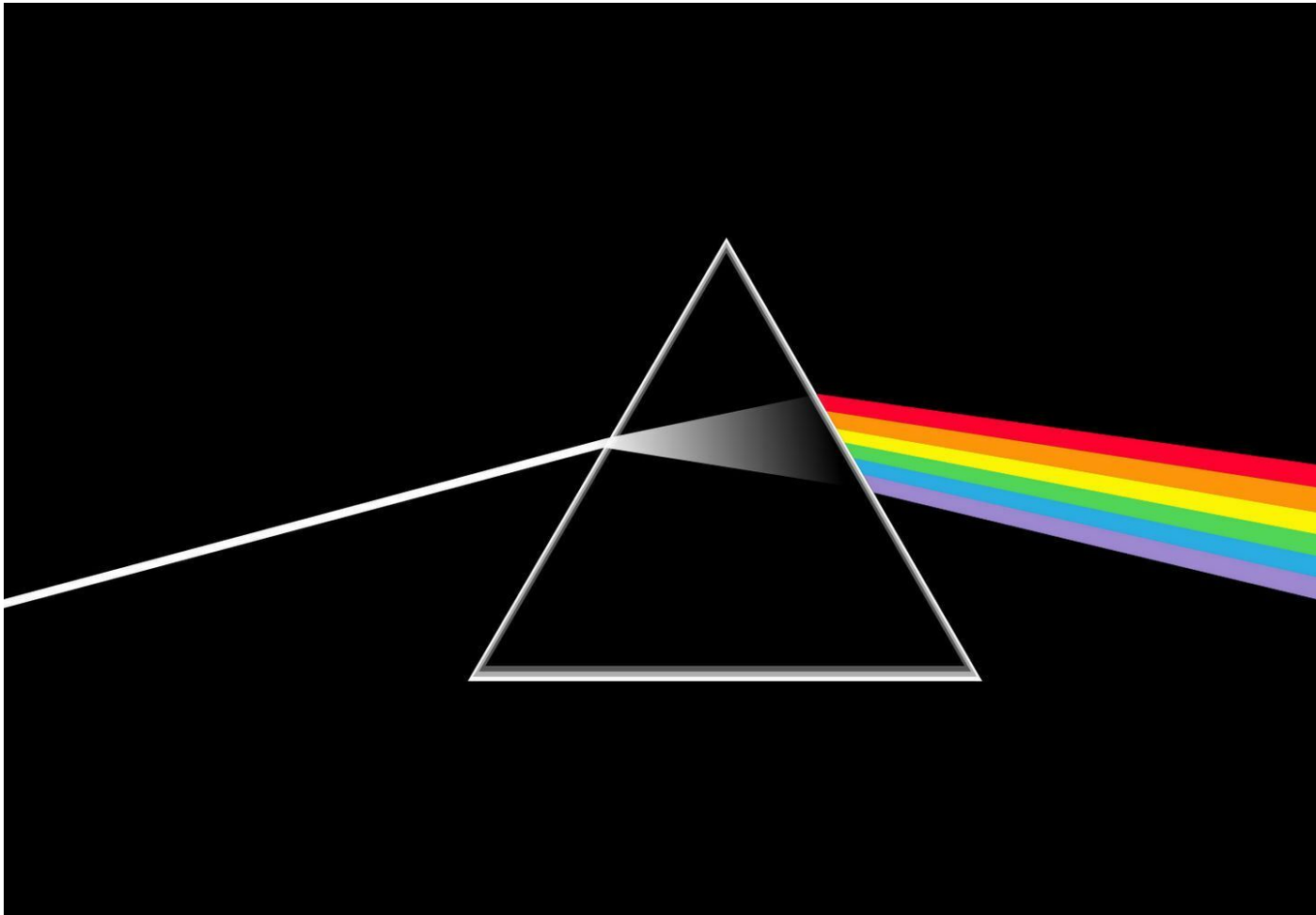
7 ex3/externals/daft orig.jpg



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**

