**Пункт 1.**

В качестве изображения для экспериментов используем фотографии, приведенные к заданию - фото лица, фото руки, маска для склейки

**Пункт 2.**

В данном задании необходимо определить функцию построения гауссовской пирамиды.

Код:

```
def gaussian_pyramid(img, sigma, n_layers):
    channels_amount = 1 if len(img.shape) == 2 else img.shape[2]
    if channels_amount == 1:
        pyramid = [img]
        for i in range(n_layers - 1):
            last = pyramid[-1]
            gaussian_pic = gaussian(last, sigma)
            pyramid.append(gaussian_pic)
        return pyramid

    pyramid = [img]
    for i in range(n_layers - 1):
        last = pyramid[-1]
        gaussian_pic = gaussian(last, sigma, multichannel=True)
        pyramid.append(gaussian_pic)
    return pyramid
```

Функция принимает изображение, сигму и число слоев. Для удобства предусмотрена работа с мультиканальным изображением, поэтому делается отдельная проверка. Гауссовский фильтр реализован в соответствующем модуле - skimage.filters

Для удобства создания лапласовской пирамиды в результирующий массив добавляется само исходное изображение.

Опробуем работу функции. Зафиксируем число слоев - 3 (с учетом начального слоя - исходное изображение). Используем разные значения сигма. Положим по очереди $\sigma = \{0.4,\ 1,\ 2\}$

Код для запуска и формирования графика частот:
Тесты проводились на картинке части лица (приложена к степу)

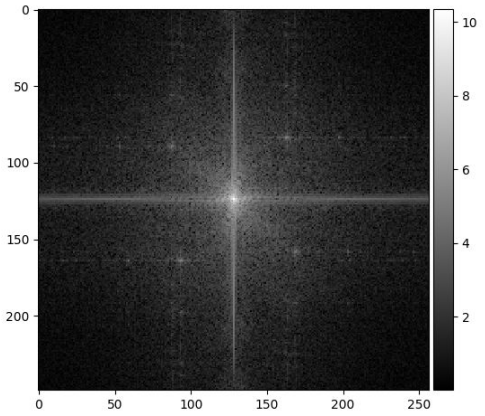
```
img = color.rgb2grey(io.imread('sources/face_part.png'))
pyramid = gaussian_pyramid(img, sigma=0.4, n_layers=3)
ffts = fourier_transformer(pyramid)
for f in ffts:
    io.imshow(f, cmap='gray')
    plt.show()
```

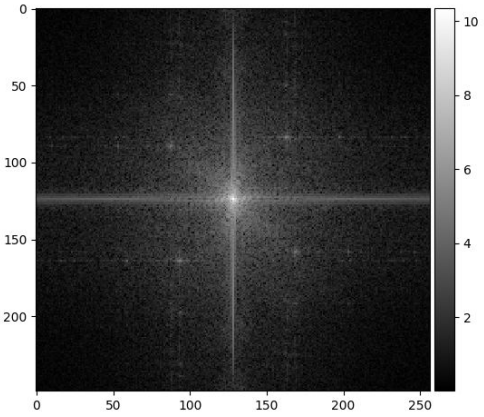
```
def fourier_transformer(gaussian_pyr):
    return [
        log(
            1 + abs(fftshift(fft2(layer)))
        )
        for layer in gaussian_pyr
    ]
```

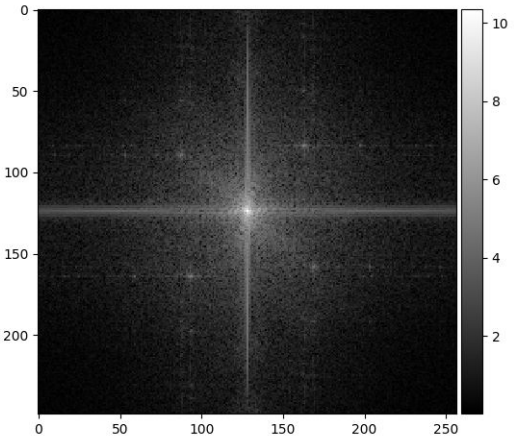
Результаты:

1. sigma = 0.4

Как было рассказано на лекциях, применение гауссовского фильтра уменьшает диапазон частот, так, что они становятся меньше и меньше, сжимаясь к началу системы счисления. 0.4 дает легкое сглаживание, уменьшая диапазон частот равномерно (от слоя к слою)

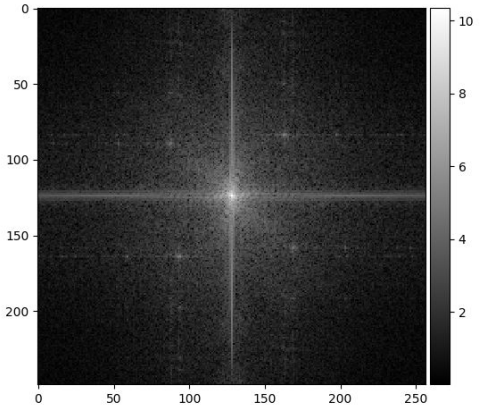


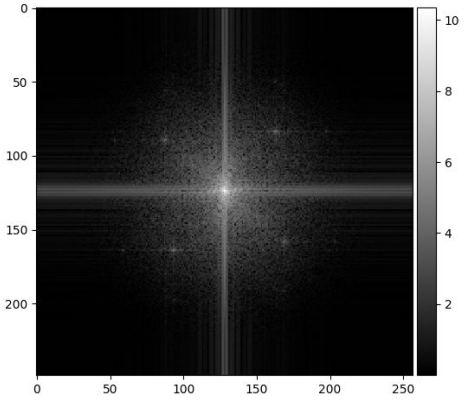



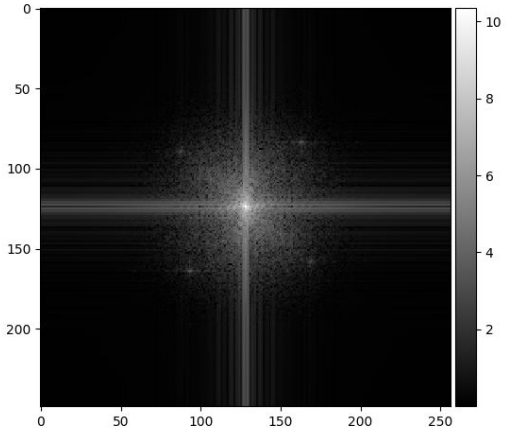



2. sigma = 1

Применение 1 в качестве сигмы дает заметное сглаживание существенно уменьшая диапазон частот уже на втором слое пирамиды.

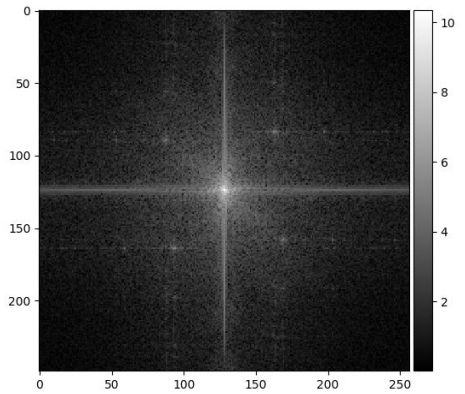


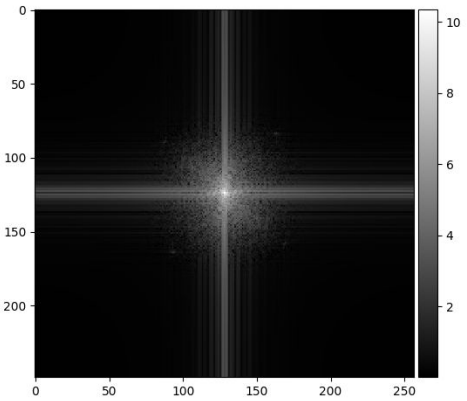



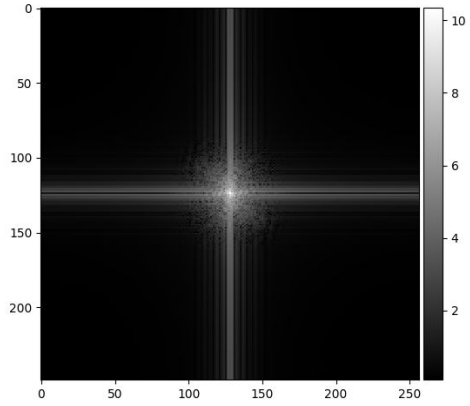



3. sigma = 2

Такое значение еще значительнее и быстрее размывает картинку.



**Пункт 3.**

В задании требуется проделать то же, что и в предыдущем пункте, но уже с пирамидой лапласа.

Функция построения пирамиды по изображению, сигме и числу слоев:

```
def laplas_pyramid(img, sigma, n_layers):
    g_pyramid = gaussian_pyramid(img, sigma, n_layers)
    l_pyramid = []
    for i in range(0, len(g_pyramid)):
        if i + 1 < len(g_pyramid):
            l_pyramid.append(g_pyramid[i] - g_pyramid[i + 1])
        else:
            l_pyramid.append(g_pyramid[i])
    return l_pyramid
```

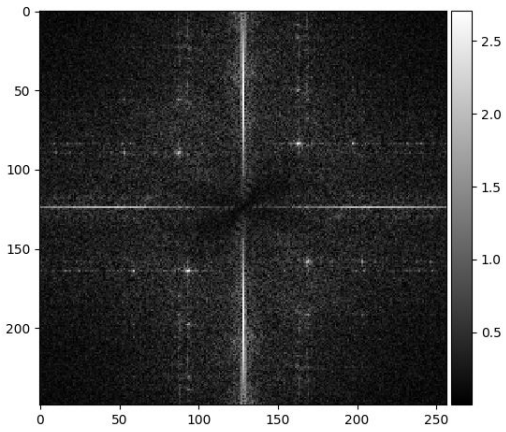
Проведем аналогичные опыты с пирамидой при тех же сигма, но уже 4 слоях. Это сделано потому, последний слой пирамиды будем последним слоем гаусовской пирамиды и для наглядности увеличим число слоев.

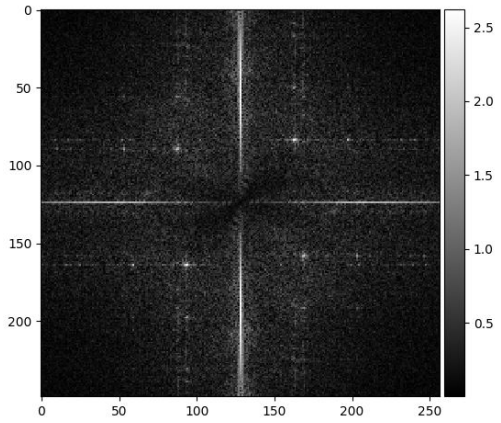
Код запуска:

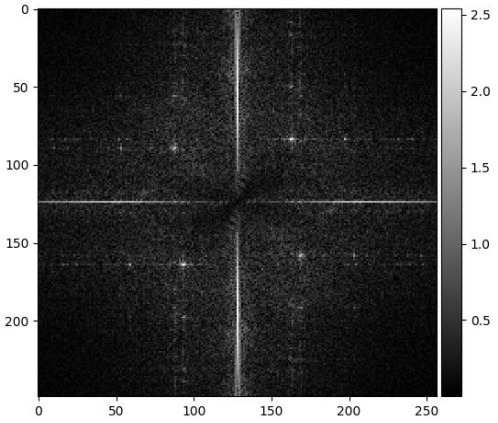
```
pyramid = laplas_pyramid(img, sigma=0.4, n_layers=3)
ffts = fourier_transformer(pyramid)
for f in ffts:
    io.imshow(f, cmap='gray')
    plt.show()
```

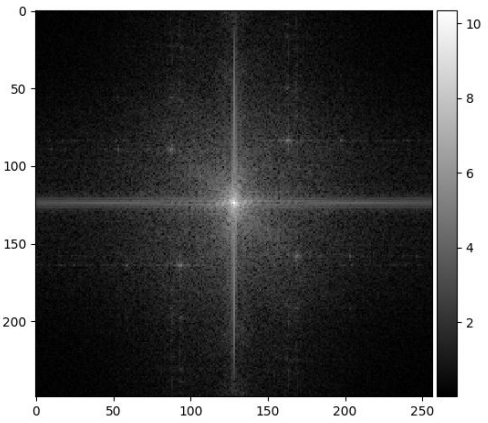
Результаты:

1. sigma = 0.4

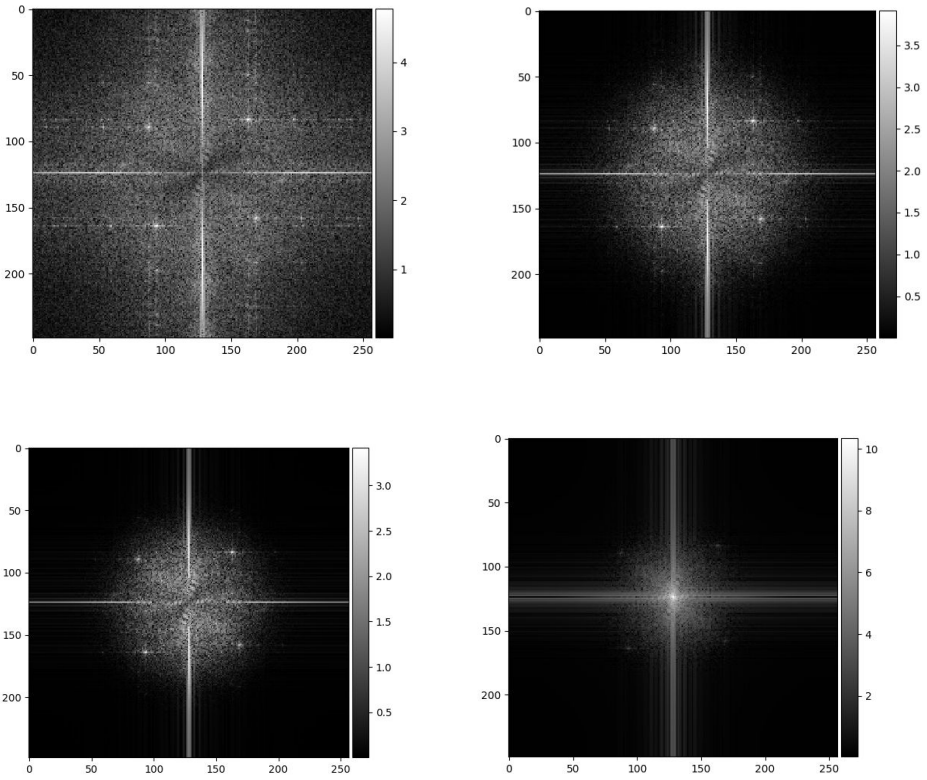


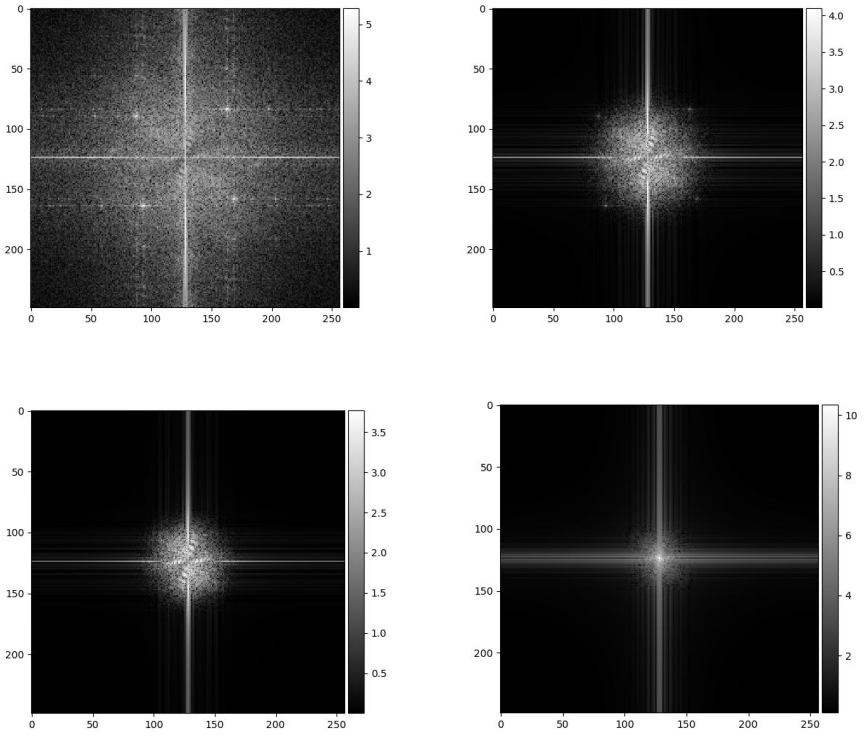



2. sigma = 1



3. sigma = 2



**Пункт 4.**

На том же наборе изображений, что и в предыдущих пунктах, протестируем итоговый результат - коллаж.

Функция, возвращающая совмещенную картинку по двум исходным и маске.

```
def blending_channel(la, lb, mask):
   height, width = la[0].shape
   result = numpy.zeros((height, width))
   for layer_a, layer_b, mask_layer in zip(la, lb, mask):
       result += mask_layer * layer_a + (1 - mask_layer) * layer_b
   return result


def blending_color(img_one, img_two, mask, img_sigma, mask_sigma,
n_layers):
   r_one, g_one, b_one = img_one[:, :, 0], img_one[:, :, 1], img_one[:, :,
2]
   r_two, g_two, b_two = img_two[:, :, 0], img_two[:, :, 1], img_two[:, :,
2]
   mask = gaussian_pyramid(mask, mask_sigma, n_layers)
   r = blending_channel(laplas_pyramid(r_one, img_sigma, n_layers)[1:],
                        laplas_pyramid(r_two, img_sigma, n_layers)[1:],
mask[1:])

   g = blending_channel(laplas_pyramid(g_one, img_sigma, n_layers)[1:],
                        laplas_pyramid(g_two, img_sigma, n_layers)[1:],
mask[1:])

   b = blending_channel(laplas_pyramid(b_one, img_sigma, n_layers)[1:],
                        laplas_pyramid(b_two, img_sigma, n_layers)[1:],
mask[1:])
   return numpy.dstack((r, g, b))
```

Первая функция осуществляет преобразование канала, вторая - преобразование цветного изображения (то есть поканальное применение) первой функции и формирует итоговое изображение. Для маски и картинок используются две разные сигмы.

Код для запуска:

```
mask = color.rgb2grey(io.imread('sources/mask_two.png'))
mask[mask > 0.5] = 1
mask[mask < 0.5] = 0
result = blending_color(io.imread('sources/a_two.png'),
io.imread('sources/b_two.png'),mask, 1, 7, 6)
io.imshow(result)
plt.show()
```

Зафиксируем размытие для картинок на 1, будем менять сигму для маски:

сигма = 1

сигма = 4

сигма = 5

сигма = 7

При сигма = 4 и 7 получаем хороший результат склейки.

Теперь будем менять число слоев. Зафиксируем сигму для картинок и маски на 1. Возьмем $\{5,\ 7,\ 9\}$ слоев

число слоев = 5

число слоев = 7

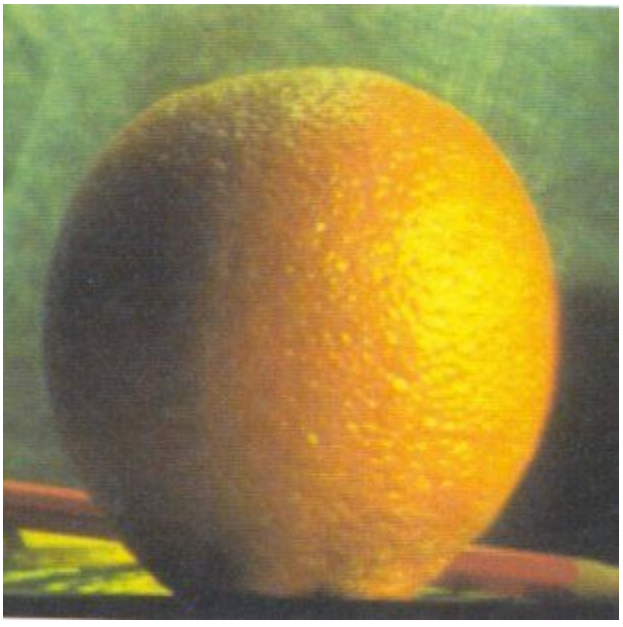
число слоев = 9

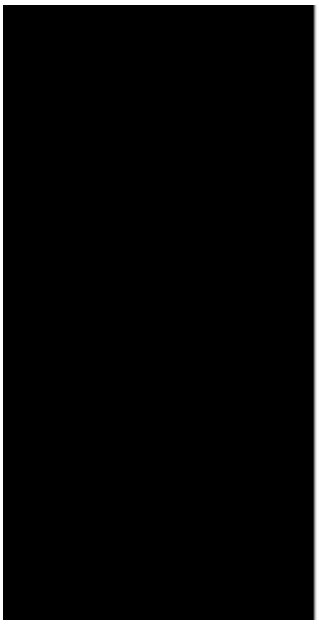
Последний результат дает хорошее изображение.

**Пункт 5.**

В данном пункте нужно самостоятельно 3 коллажа.

Канонический пример с яблоком и апельсином:
Сигма для картинок = 1, для маски = 2, число слоев = 6





Результат:

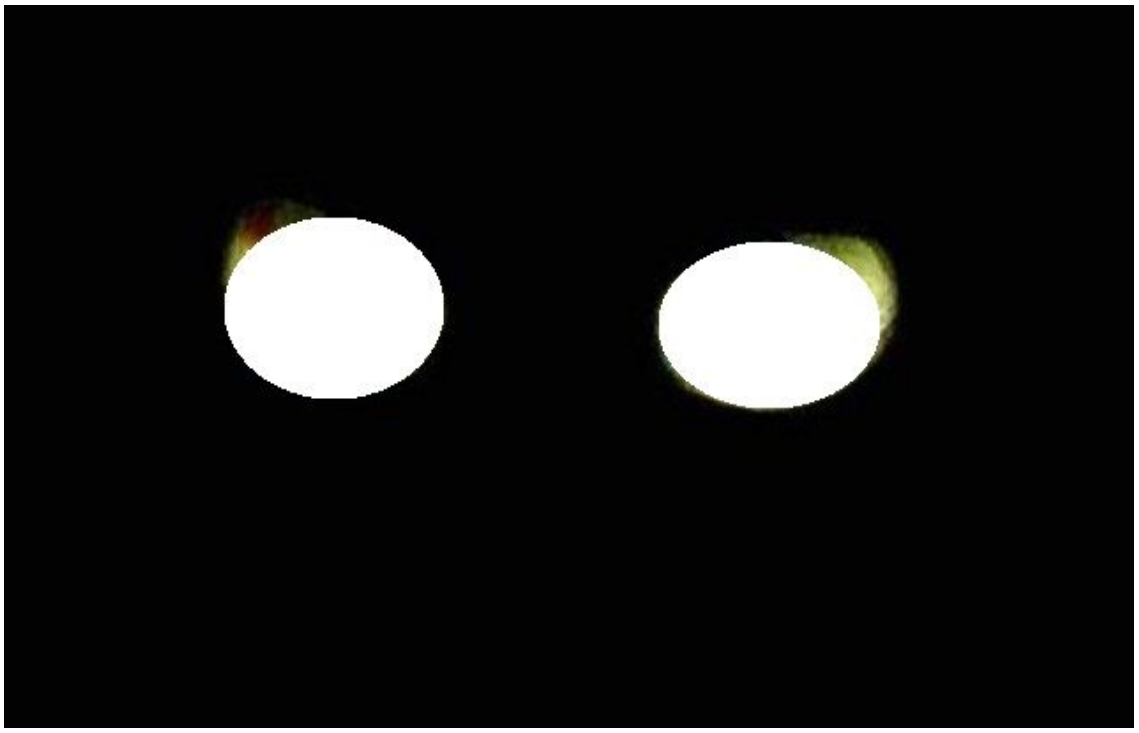
Опыт второй:



Результат:


Опыт третий:

маску делал в paint, позже в коде делал threshold 0.8, чтобы покрыть весь глаз



Результат: