1. Подготовка

In [10]:
```
from google.colab import drive

drive.mount('content/')
```

```
Drive already mounted at content/; to attempt to forcibly remount, call drive.mount("content/", force_remount=T
rue).
```

In [9]:
```
!pip install nbconvert > 0
```

In [11]:
```
!jupyter nbconvert --to html /content/content/MyDrive/Colab_Notebooks/4_glue_images.ipynb
```

```
[NbConvertApp] Converting notebook /content/content/MyDrive/Colab_Notebooks/4_glue_images.ipynb to html
[NbConvertApp] Writing 9433881 bytes to /content/content/MyDrive/Colab_Notebooks/4_glue_images.html
```

Импортируем необходимые модули:

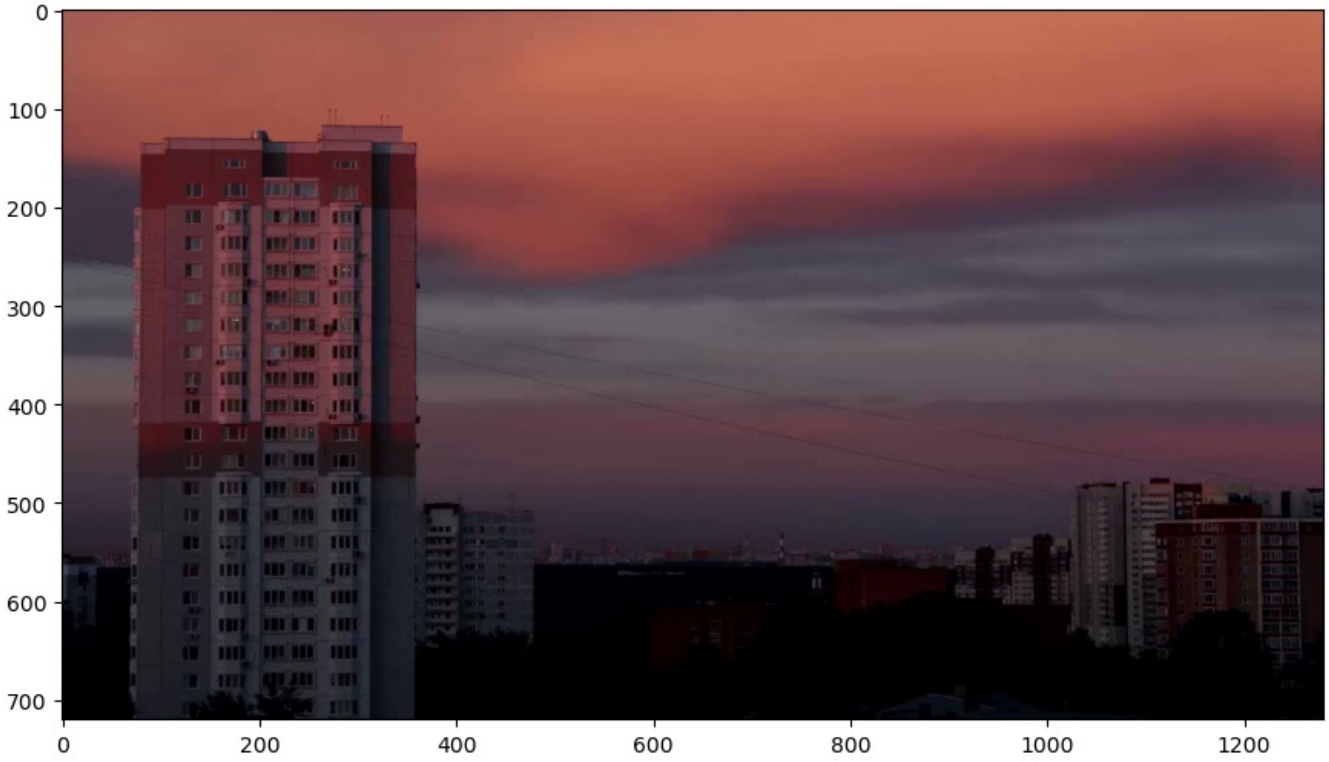
In [ ]:
```
from skimage import img_as_float, exposure
from skimage.color import rgb2yuv, yuv2rgb
from skimage.io import imshow, imread
import numpy as np
from numpy.fft import fftshift, fft2
from scipy.signal import convolve2d
import matplotlib.pyplot as plt
%matplotlib inline
```

Изначально будем работать с пространством **YUV**, где первые канал является яркостью (значения в [0;1]), а другие два - цветовые компоненты (значения в [-1;1]). Исходная ф-ция **imshow** работает с пространством **RGB**, потому напишем свой вариант, учитывая количество каналов в изначальном изображении и переходы к разным пространства с помощью **rgb2yuv** и **yuv2rgb**.

In [ ]:
```
# аналог imread
def imread_yuv(filename):
    img = rgb2yuv(imread(filename))
    return img
```

In [ ]:
```
# аналог imshow
def imshow_yuv(img, size=(15, 12), several=False, blocks=(1, 1)):

    fig = plt.figure(figsize = size)

    imgs_number = 1
    if several:
        imgs_number = len(img)

    for i in range(min(blocks[0]*blocks[1], imgs_number)):
        fig.add_subplot(blocks[1], blocks[0], i+1)

        if several:
            if len(img[i].shape) == 3:
                # yuv2rgb переводит картинку во float Значения, т.е. [0;1]
                plt.imshow(np.clip(yuv2rgb(img[i]), 0, 1))
            else:
                plt.imshow(np.clip(img[i], 0, 1))
        else:
            plt.imshow(np.clip(yuv2rgb(img), 0, 1))
```

In [ ]:
```
img = imread_yuv('https://i.ytimg.com/vi/MOuEFccsNKA/maxresdefault.jpg')
imshow_yuv(img, size=(12, 6))
```



2. Гауссовская пирамида

Сначала построим функцию ядра гауссовского фильтра и саму функцию свертки, учитывающую кол-во каналов изображения.

In [ ]:
```
def gauss(sigma, x, y):
    return 1 / (2 * np.pi * sigma**2) * np.exp((-x**2 -y**2) / (2 * sigma**2))
```

In [ ]:
```
def gauss_kernel(sigma):
    k = round(3 * sigma) * 2 + 1
    ker = np.zeros((k, k))
    center = k // 2

    for x in range(-center, center + 1):
        for y in range(-center, center + 1):
            ker[y + center][x + center] = gauss(sigma, x, y)

    return ker / np.sum(ker)
```

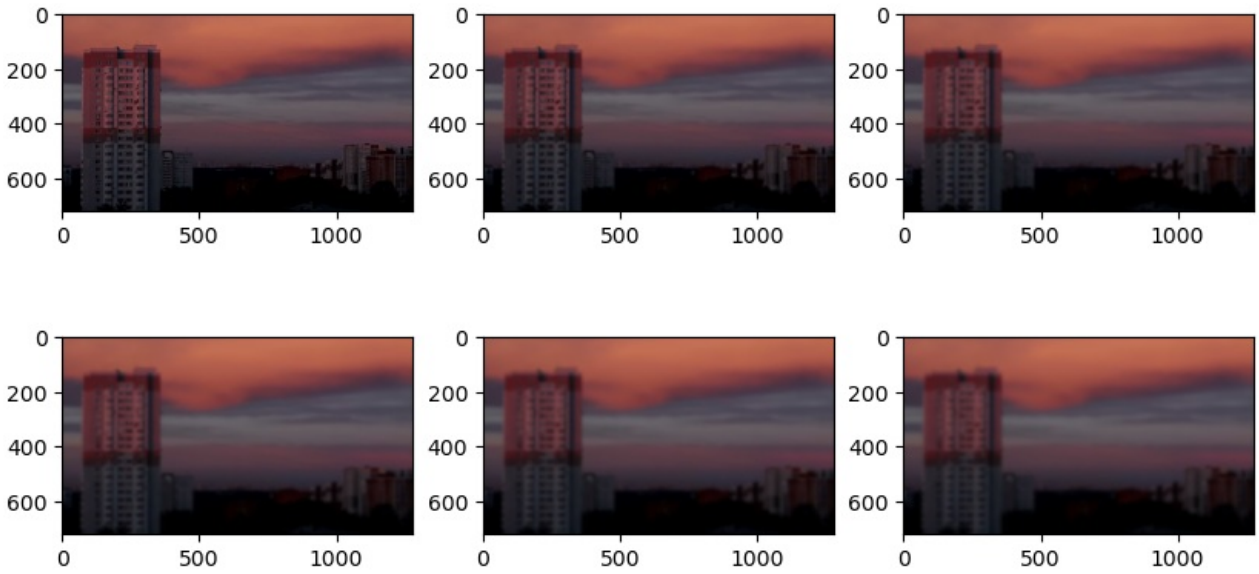
Для свертки используем функцию **convolve2d**, позволяющую регулировать параметры свертки. При каждой операции будем сохранять размер изображения за счет зеркалирования границ матриц (параметры *same* и *symm*).

In [ ]:
```
def convolution(img, kernel):

    if len(img.shape) == 2:
        return convolve2d(img, kernel, mode='same', boundary='symm')

    ch1, ch2, ch3 = img[:, :, 0], img[:, :, 1], img[:, :, 2]
    new_ch1 = convolve2d(ch1, kernel, mode='same', boundary='symm')
    new_ch2 = convolve2d(ch2, kernel, mode='same', boundary='symm')
    new_ch3 = convolve2d(ch3, kernel, mode='same', boundary='symm')
    return np.dstack((new_ch1, new_ch2, new_ch3))
```

In [ ]:
```
def gauss_pyramid(img, sigma, n_layers):
    pyramid = [img]
    ker = gauss_kernel(sigma)

    for i in range(n_layers):
        step = pyramid[-1]
        new_img = convolution(step, ker)
        pyramid.append(new_img)

    return pyramid
```

Изображения гауссовской пирамиды для **sigma** = 1 из 5 слоев (первым изображением идет оригинальное фото, а потом пирамида).

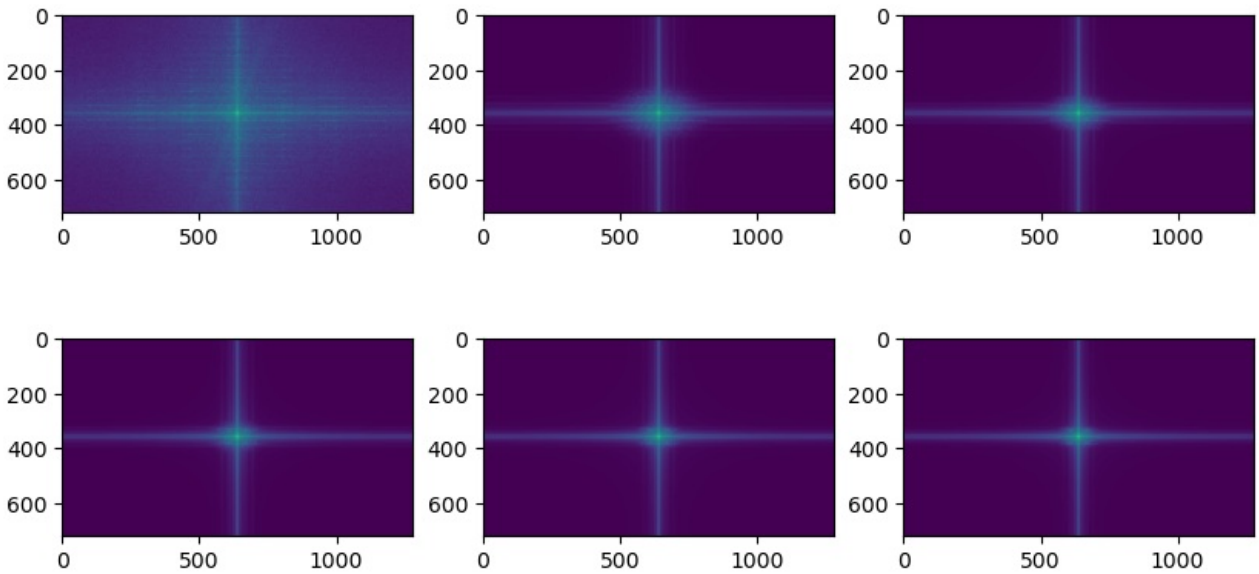
In [ ]:
```
img_g = gauss_pyramid(img, 3, 5)
imshow_yuv(img_g, size=(10, 5), several=True, blocks=(3, 2))
```



Аналогично функции **imshow_yuv** построим функцию **отображения Фурье**, показывающей частоты изображения, а именно первого канала **YUV** пространства.

In [ ]:
```
def imshow_fft(img, size=(15, 12), several=False, blocks=(1, 1)):

    fig = plt.figure(figsize = size)

    imgs_number = 1
    if several:
        imgs_number = len(img)

    for i in range(min(blocks[0]*blocks[1], imgs_number)):
        fig.add_subplot(blocks[1], blocks[0], i+1)

        if several:
            if len(img[i].shape) == 3:
                fft_img = np.log(1 + abs(fftshift(fft2(img[i][:,:,0]))))
            else:
                fft_img = np.log(1 + abs(fftshift(fft2(img[i]))))
        else:
            fft_img = np.log(1 + abs(fftshift(fft2(img[:,:,0]))))

        plt.imshow(fft_img)
```

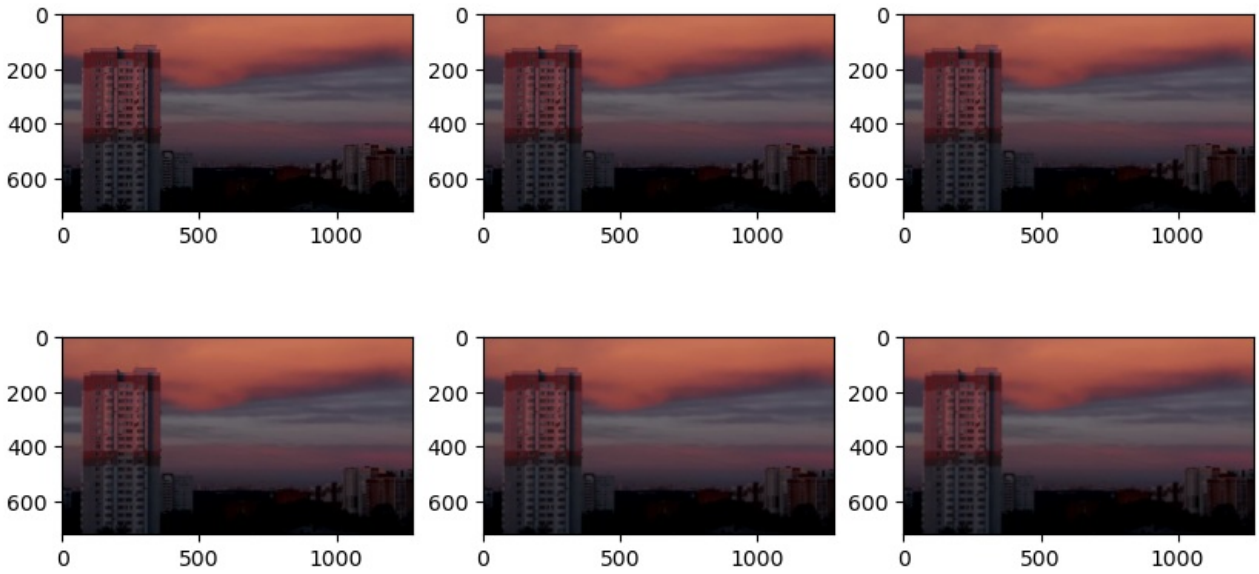
In [ ]:
```
imshow_fft(img_g, size=(10, 5), several=True, blocks=(3, 2))
```



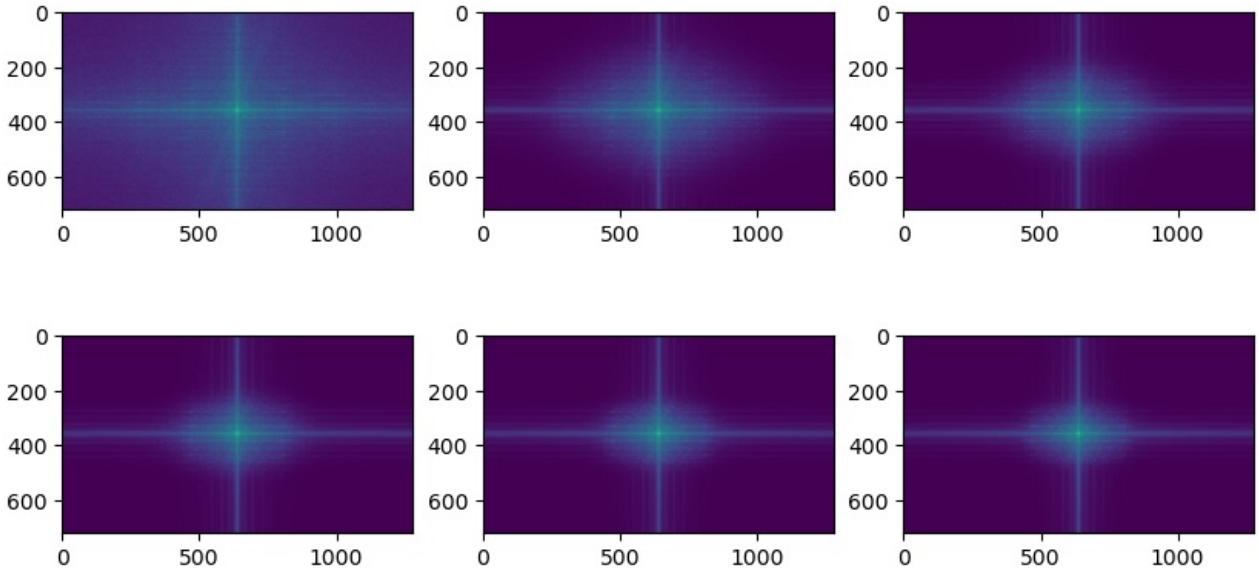
Можно убедиться, что диапазон частот сужается. Теперь построим пирамиды для трех различных значений **sigma**.

sigma = 1

In [ ]:
```
img_g_test1 = gauss_pyramid(img, 1, 5)
imshow_yuv(img_g_test1, size=(10, 5), several=True, blocks=(3, 2))
```

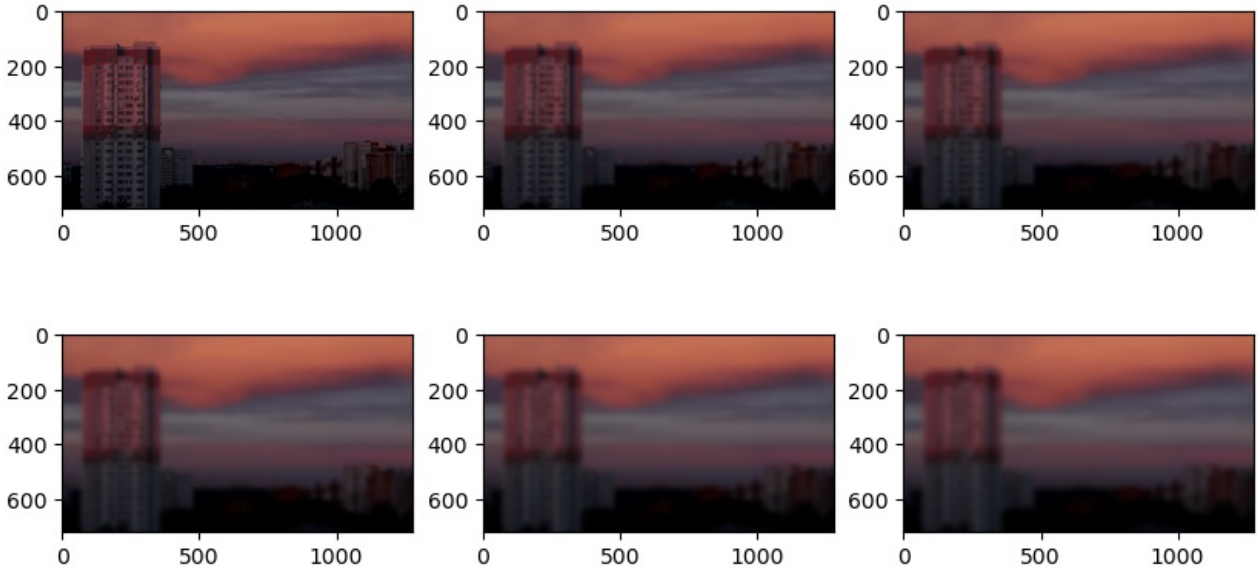


In [ ]:
```
imshow_fft(img_g_test1, size=(10, 5), several=True, blocks=(3, 2))
```

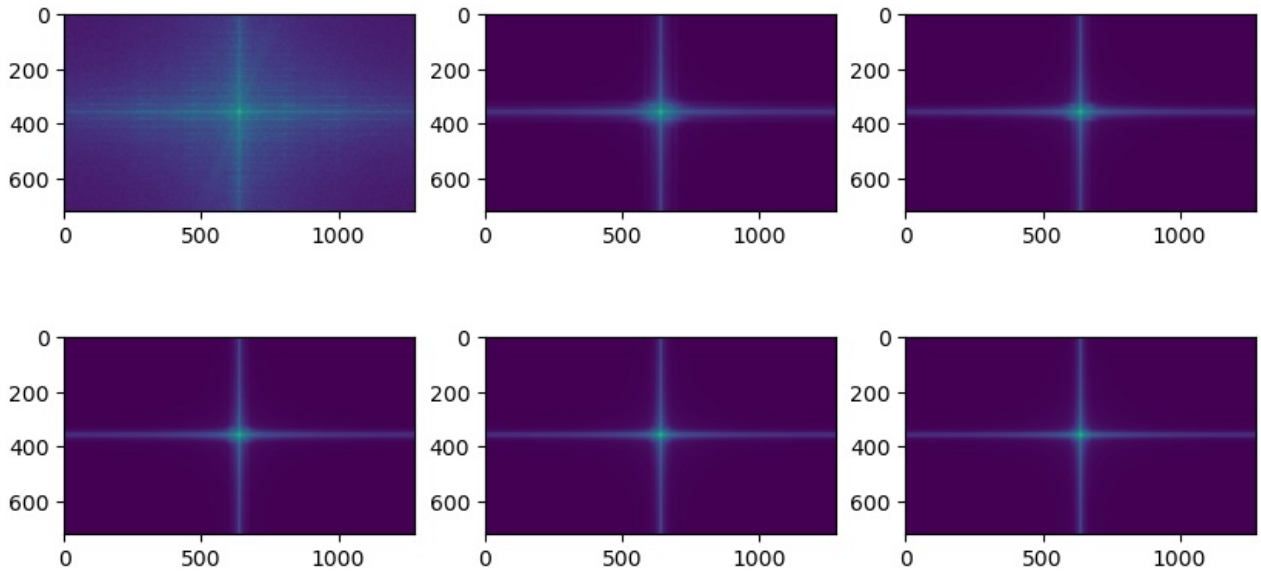


sigma = 5

In [ ]:
```
img_g_test2 = gauss_pyramid(img, 5, 5)
imshow_yuv(img_g_test2, size=(10, 5), several=True, blocks=(3, 2))
```



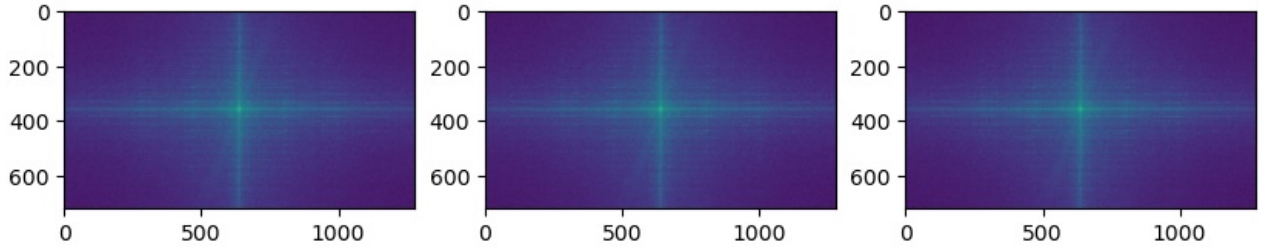
In [ ]:
```
imshow_fft(img_g_test2, size=(10, 5), several=True, blocks=(3, 2))
```

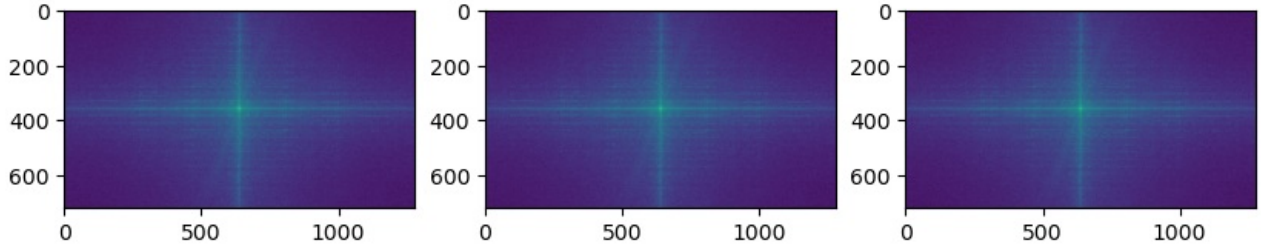


sigma = 0.3

In [ ]:
```
img_g_test3 = gauss_pyramid(img, 0.3, 5)
imshow_yuv(img_g_test3, size=(10, 5), several=True, blocks=(3, 2))
```

In [ ]:
```
imshow_fft(img_g_test3, size=(10, 5), several=True, blocks=(3, 2))
```





3. Лапласовская пирамида

Зададим функцию построения лапласовской пирамиды. Для оптимизации вычислений добавим возможность уже на входе

загрузить гауссовскую пирамиду.

In [ ]:
```
def laplas_pyramid(img, sigma, n_layers, pyramid_g=None):

    if pyramid_g is None:
        pyramid_g = gauss_pyramid(img, sigma, n_layers)

    pyramid_l = [img]

    for i in range(n_layers - 1):
        new_img = pyramid_g[i] - pyramid_g[i + 1]
        pyramid_l.append(new_img)

    pyramid_l.append(pyramid_g[-1])
    return pyramid_l
```

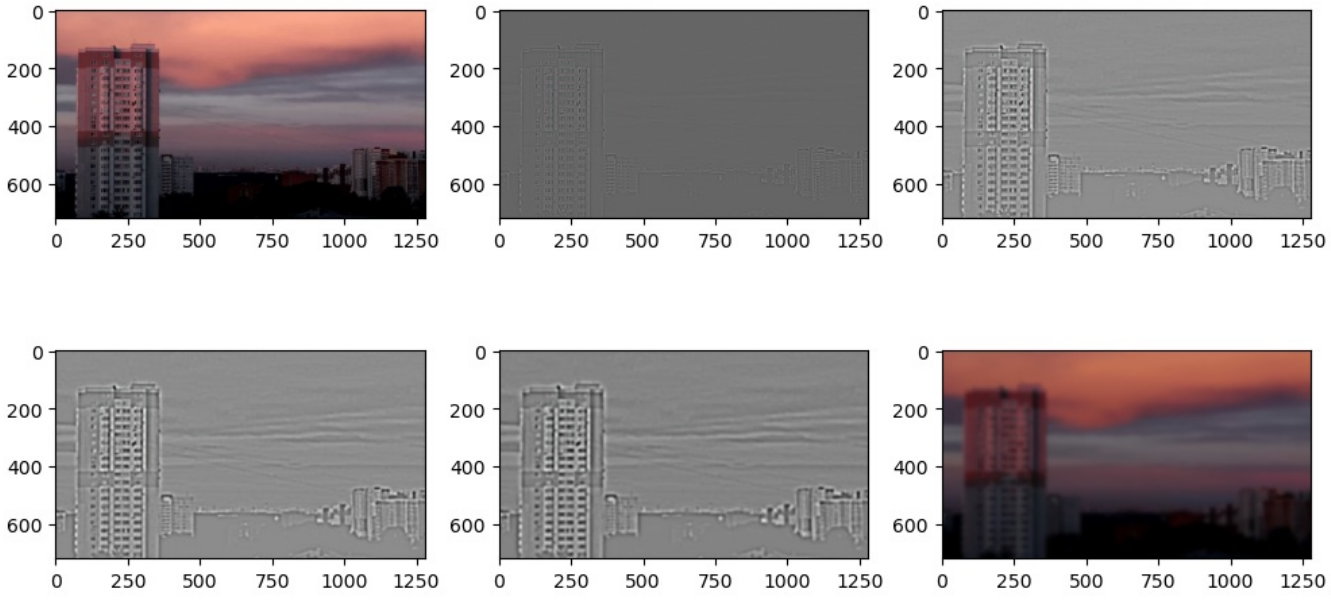
Для лучшего визуального восприятия элементов лапласовской пирамиды можно **нормализировать** изображения (они будут достаточно блеклы и почти не видны), т.е. изменить диапазон значений интенсивности пикселей. Будем нормализировать YUV диапазон только по Y каналу, т.к. он отвечает за интенсивность. В самом решении нормализация нигде не используется. Нормализацию выполним порсдеством соответствующей функции **exposure.rescale_intensity**, указав диапазон новых значений пикселей.

In [ ]:
```
# Нормализация изображения в пространстве цветов YUV
def normalize_yuv(img):

    # Нормализация компонента Y
    normalized_y = exposure.rescale_intensity(img[:, :, 0], in_range='image', out_range=(0, 1))

    # Обновление компонента Y в исходном изображении
    normalized_yuv_image = img.copy()
    normalized_yuv_image[:, :, 0] = normalized_y

    return normalized_yuv_image
```

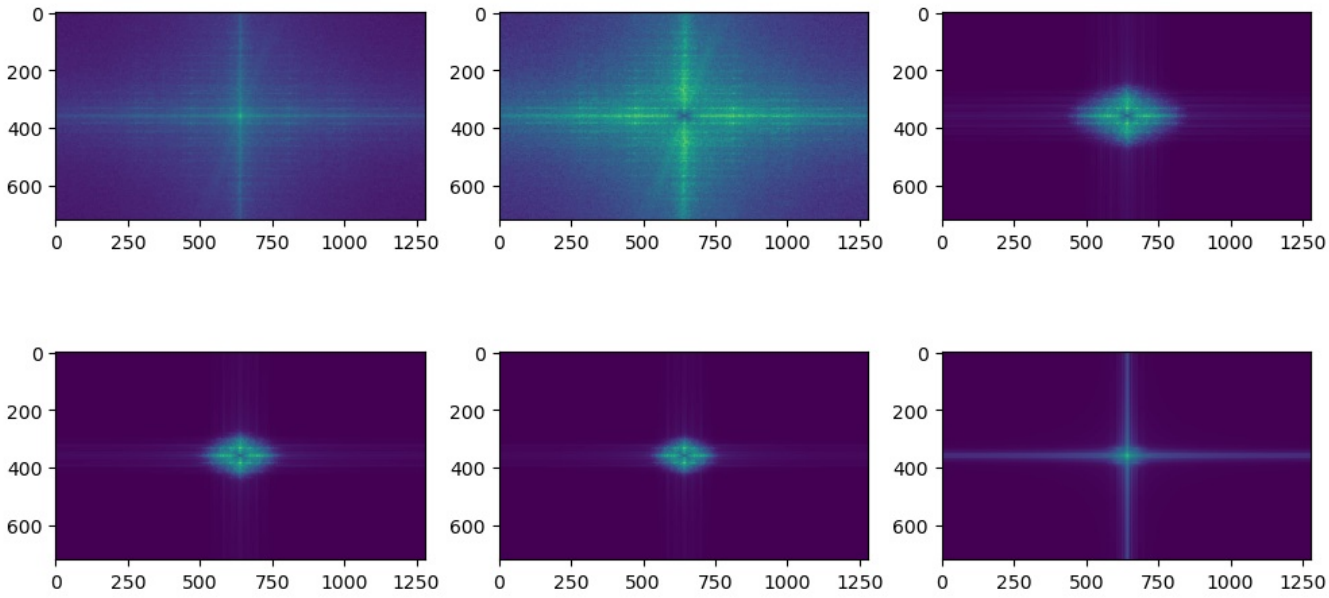
In [ ]:
```
# нормализация для выборочных изображений в последовательности
def normalize_several_yuv(imgs, start=0, end=None):
    if not end:
        end = len(imgs)
    if end < 0:
        end += len(imgs)

    n = imgs.copy()
    for i in range(start, end):
        n[i] = normalize_yuv(imgs[i])

    return n
```

На вход будем подавать уже готовые гауссовские пирамиды с прошлых тестов.

In [ ]:
```
img_l = laplas_pyramid(img, 3, 5, img_g)

n = normalize_several_yuv(img_l, 0, -1)
imshow_yuv(n, several=True, size=(12, 6), blocks=(3, 2))
```



In [ ]:
```
imshow_fft(img_l, size=(12, 6), several=True, blocks=(3, 2))
```



Можно убедиться, что при сложении всех элементов лапласовской пирамиды будет получена исходная. Напишем соответствующую функцию с демонстрацией всех промежуточных шагов.

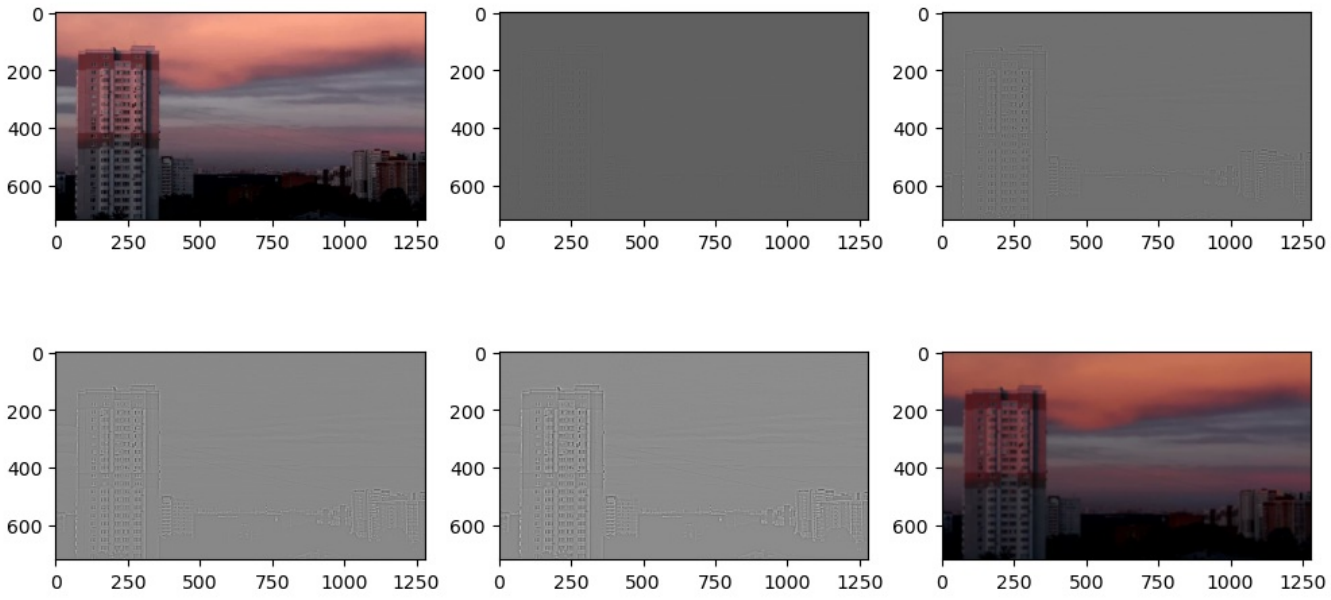
In [ ]:
```
def imsum(imgs):
    s = imgs[0]
    steps = [imgs[0]]
    for i in range(1, len(imgs)):
        s += imgs[i]
        steps.append(imgs[i])
    return s, steps
```

In [ ]:
```
sum_img, steps = imsum(img_l[1:])
imshow_yuv(sum_img, size=(8, 6))
```

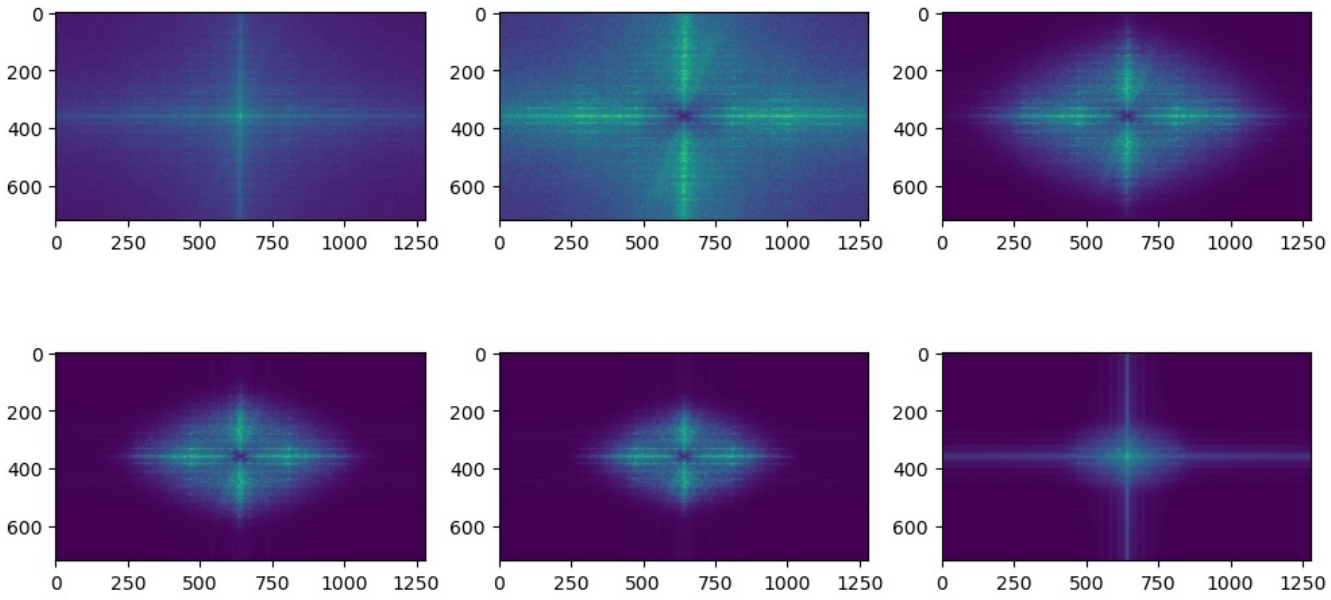
Проведем аналогичные эксперименты для разных значений **sigma**.

sigma = 1

In [ ]:
```
img_l_test1 = laplas_pyramid(img, 1, 5, img_g_test1)

n = normalize_several_yuv(img_l_test1, 0, -1)
imshow_yuv(n, several=True, size=(12, 6), blocks=(3, 2))
```

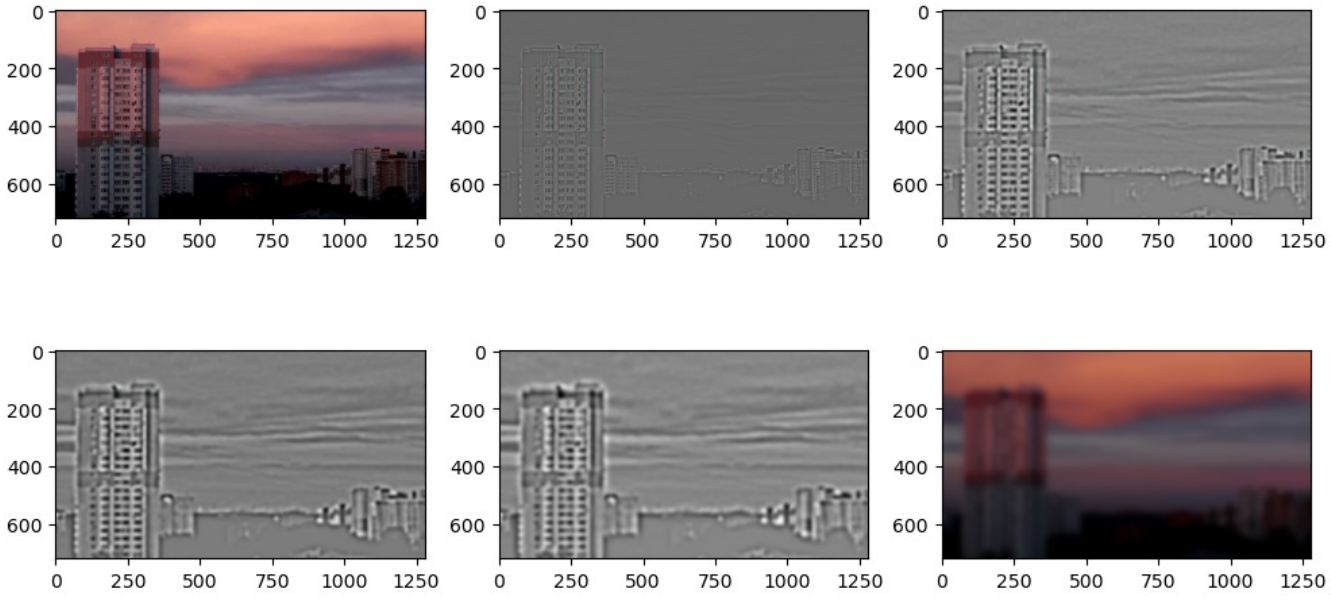


In [ ]:
```
imshow_fft(img_l_test1, size=(12, 6), several=True, blocks=(3, 2))
```

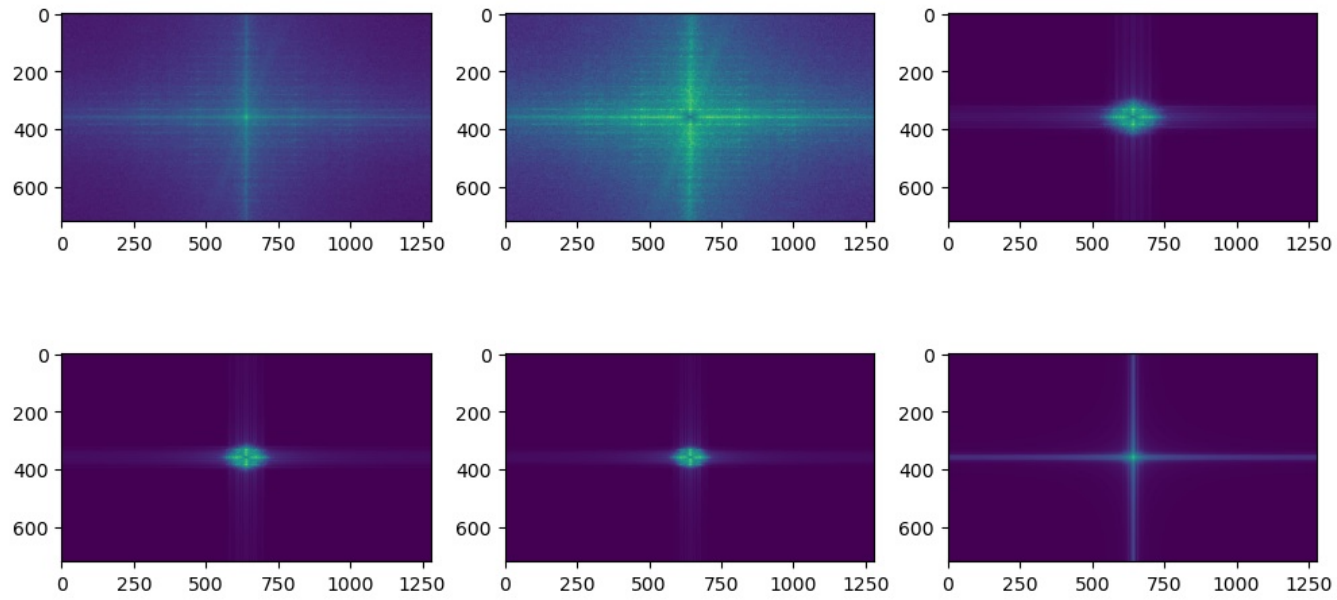


sigma = 5

In [ ]:
```
img_l_test2 = laplas_pyramid(img, 5, 5, img_g_test2)

n = normalize_several_yuv(img_l_test2, 0, -1)
imshow_yuv(n, several=True, size=(12, 6), blocks=(3, 2))
```

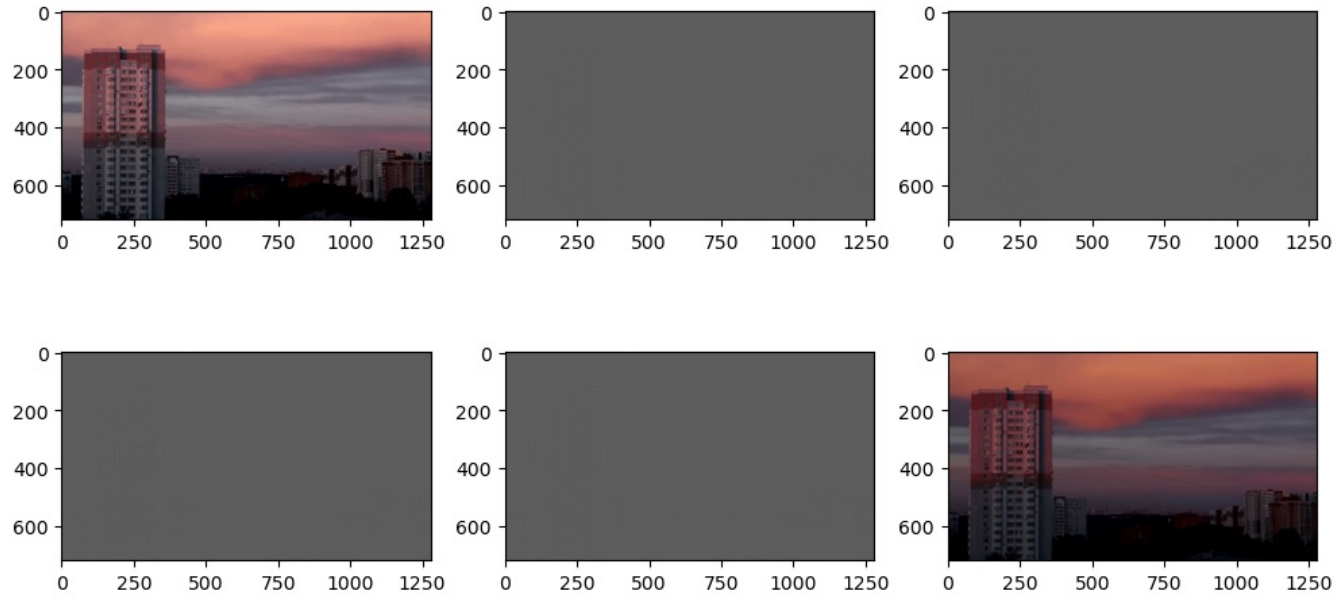


In [ ]:
```
imshow_fft(img_l_test2, size=(12, 6), several=True, blocks=(3, 2))
```

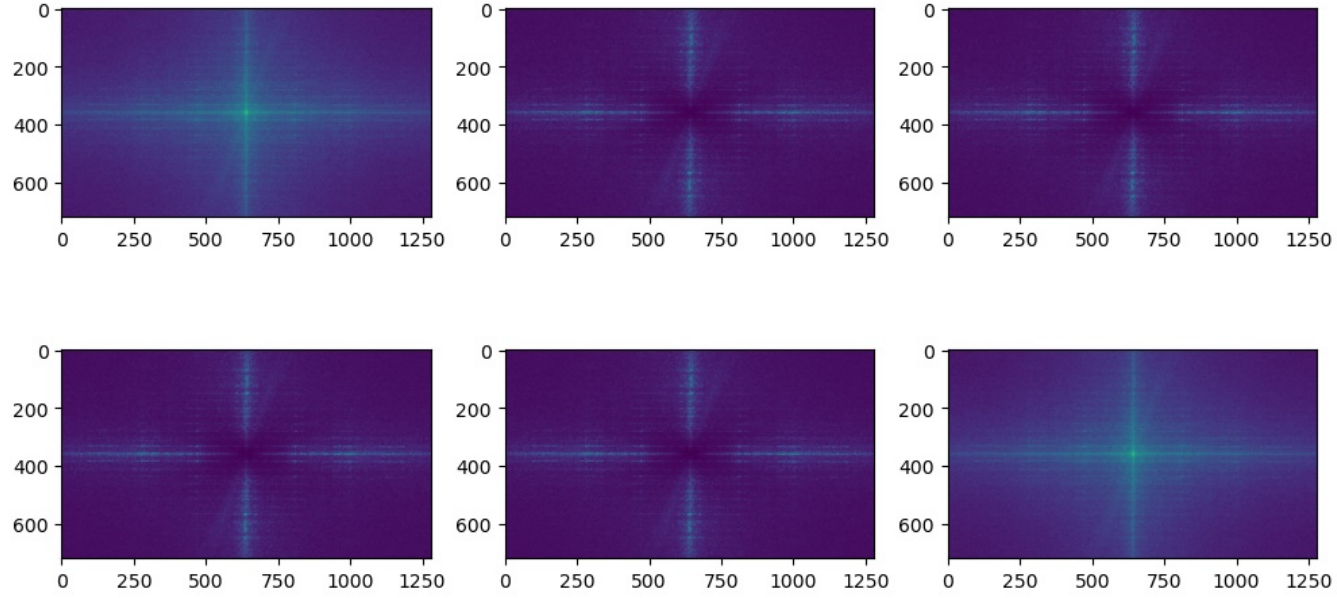


sigma = 0.3

In [ ]:
```
img_l_test3 = laplas_pyramid(img, 0.3, 5, img_g_test3)

n = normalize_several_yuv(img_l_test3, 0, -1)
imshow_yuv(n, several=True, size=(12, 6), blocks=(3, 2))
```



In [ ]:
```
imshow_fft(img_l_test3, size=(12, 6), several=True, blocks=(3, 2))
```



4. Склейка изображений

Напишем функцию бинаризации маски, где в итоге будем использовать только ее первый канал в пространстве **YUV**, т.к. нам важны значения интенсивности пикселей. Надо учитывать, что изначально все значения там лежат в диапазоне [0;1]. Также напишем отдельно функцию склейки, которую мы будем проводить для каждого канала по отдельности. Будем выводить готовое изображение и все элементы лапласовской пирамиды, т.е. каждую итерацию склейки.

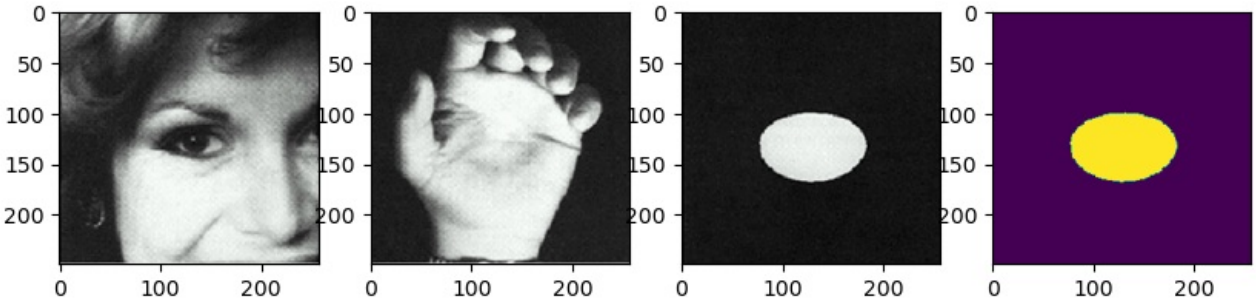
In [ ]:
```
def binarize(img):
    mask = img[:, :, 0]
    return (mask > 0.5).astype('float32')
```

In [ ]:
```
def glue(img1, img2, mask, sigma, n_layers):
    LA = laplas_pyramid(img1, sigma, n_layers)
    LB = laplas_pyramid(img2, sigma, n_layers)

    mask = binarize(mask)
    GM = gauss_pyramid(mask, sigma, n_layers)

    steps = []
    # диапазон сдвинут, т.к. в пирамидах 1-ое изображение - оригинал
    for i in range(1, n_layers+1):
        GM[i] = np.dstack((GM[i], GM[i], GM[i]))
        step = GM[i] * LA[i] + (1 - GM[i]) * LB[i]
        steps.append(step)

    return sum(steps), steps
```

Сначала проведем тесты на готовом сете изображений.

In [ ]:
```
img_a = imread_yuv('a.png')
img_b = imread_yuv('b.png')
img_mask = imread_yuv('mask.png')
```

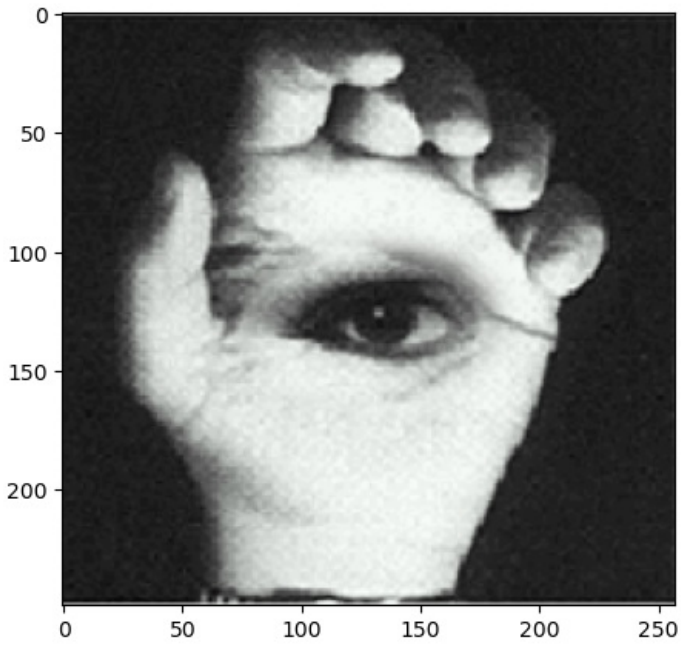
In [ ]:
```
imshow_yuv([img_a, img_b, img_mask, binarize(img_mask)], size=(10, 5), several=True, blocks=(4, 1))
```



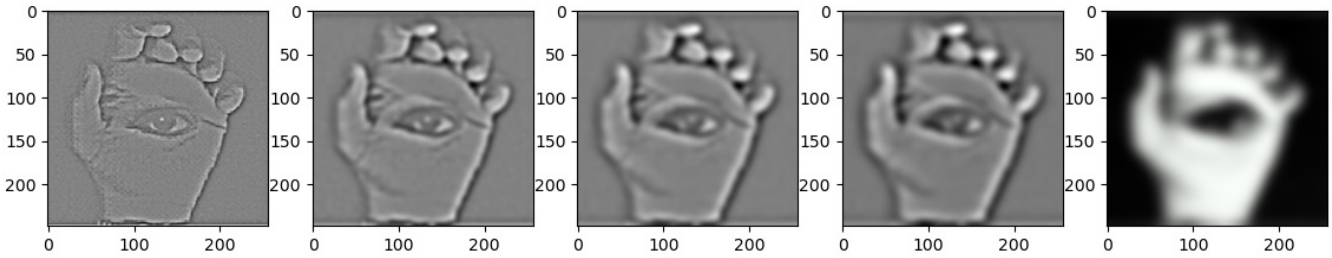
Попробуем комбинированно фиксировать значения sigma и кол-ва слоев.

sigma = 1 n_layers = 5

In [ ]:
```
res, steps = glue(img_a, img_b, img_mask, 3, 5)
imshow_yuv(res, size=(10, 5))
```

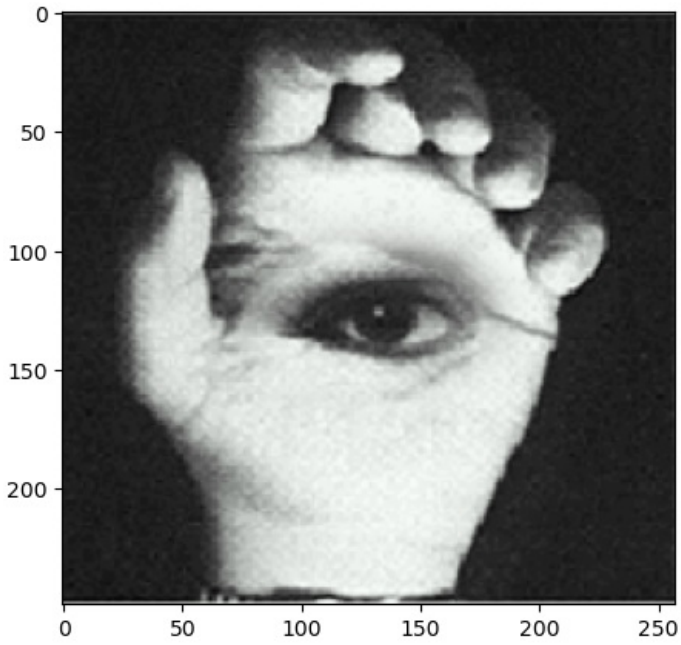


In [ ]:
```
n = normalize_several_yuv(steps)
imshow_yuv(n, size=(14, 8), several=True, blocks=(5, 2))
```

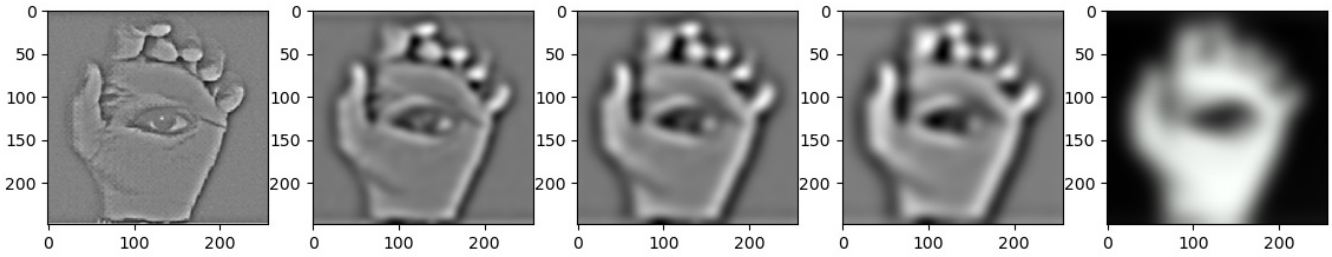


sigma = 5 n_layers = 5

In [ ]:
```
res, steps = glue(img_a, img_b, img_mask, 5, 5)
imshow_yuv(res, size=(10, 5))
```

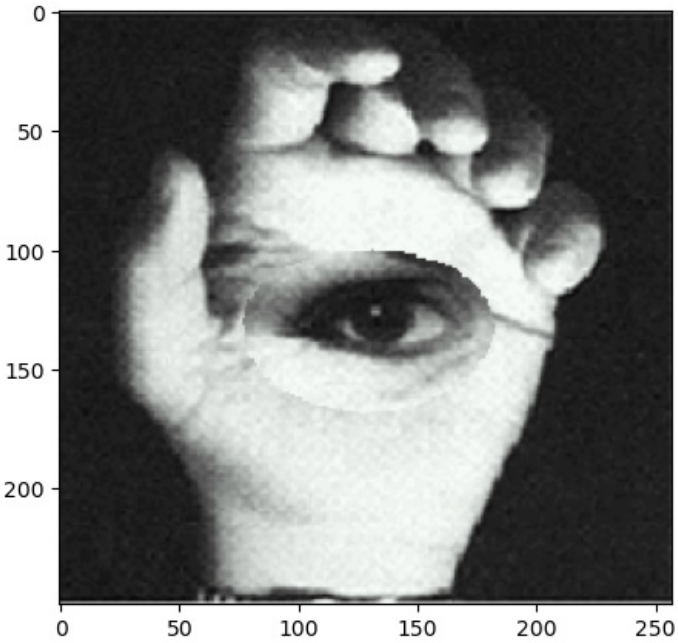


In [ ]:
```
n = normalize_several_yuv(steps)
imshow_yuv(n, size=(14, 8), several=True, blocks=(5, 2))
```

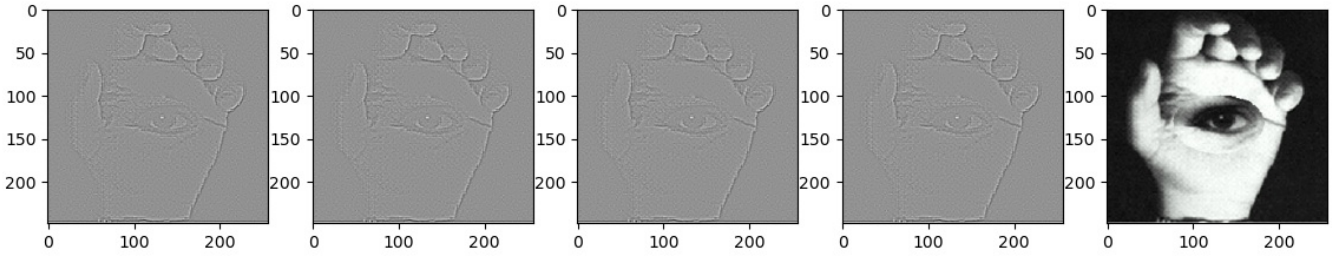


steps = 0.3 n_layers = 5

In [ ]:
```
res, steps = glue(img_a, img_b, img_mask, 0.3, 5)
imshow_yuv(res, size=(10, 5))
```

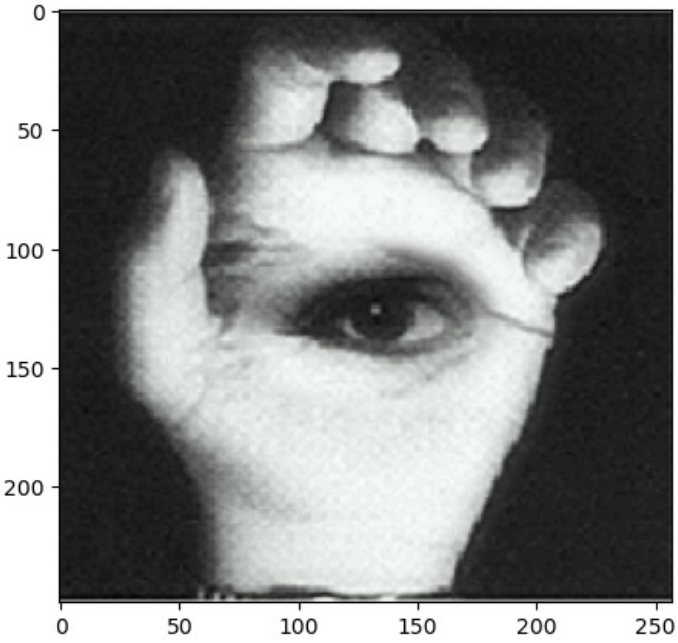


In [ ]:
```
n = normalize_several_yuv(steps)
imshow_yuv(n, size=(14, 8), several=True, blocks=(5, 2))
```

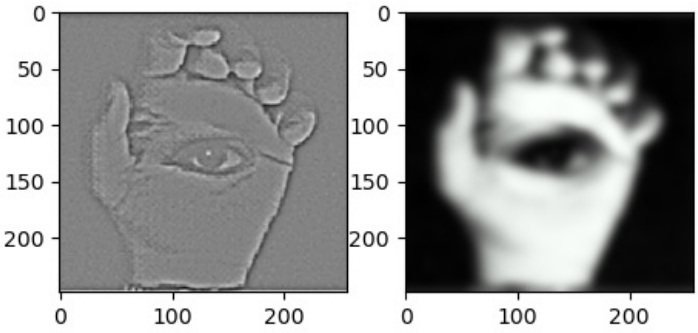


sigma = 3 n_layers = 2

In [ ]:
```
res, steps = glue(img_a, img_b, img_mask, 3, 2)
imshow_yuv(res, size=(10, 5))
```

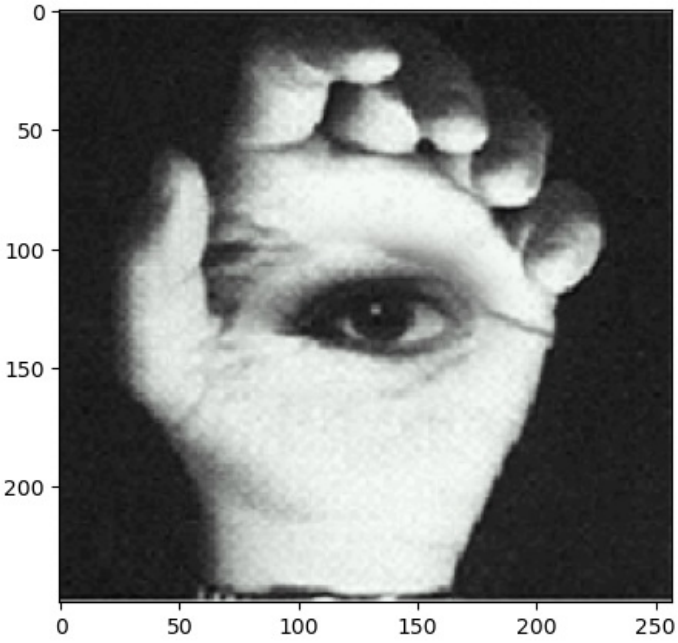


In [ ]:
```
n = normalize_several_yuv(steps)
imshow_yuv(n, size=(14, 8), several=True, blocks=(5, 2))
```

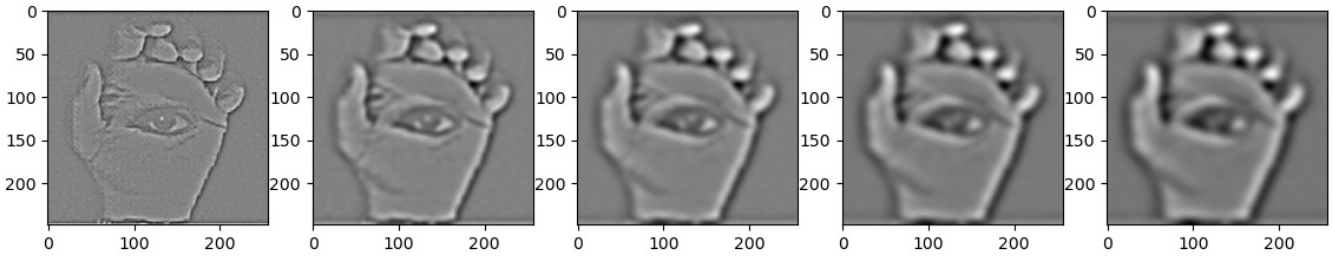


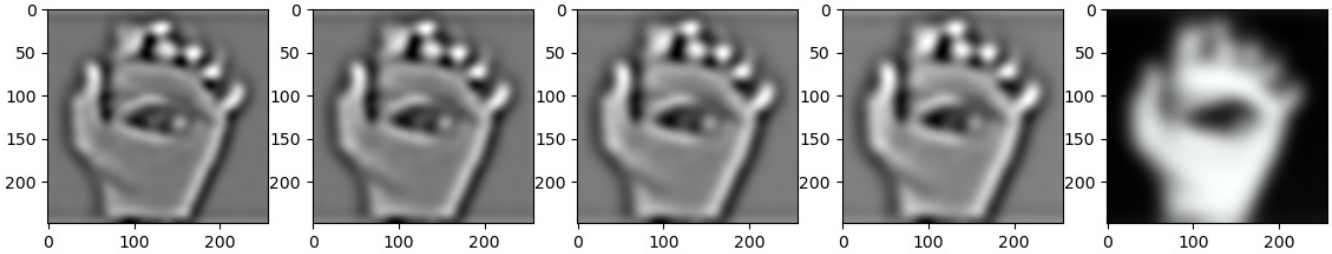
sigma = 3 n_layers = 10

In [ ]:
```
res, steps = glue(img_a, img_b, img_mask, 3, 10)
imshow_yuv(res, size=(10, 5))
```



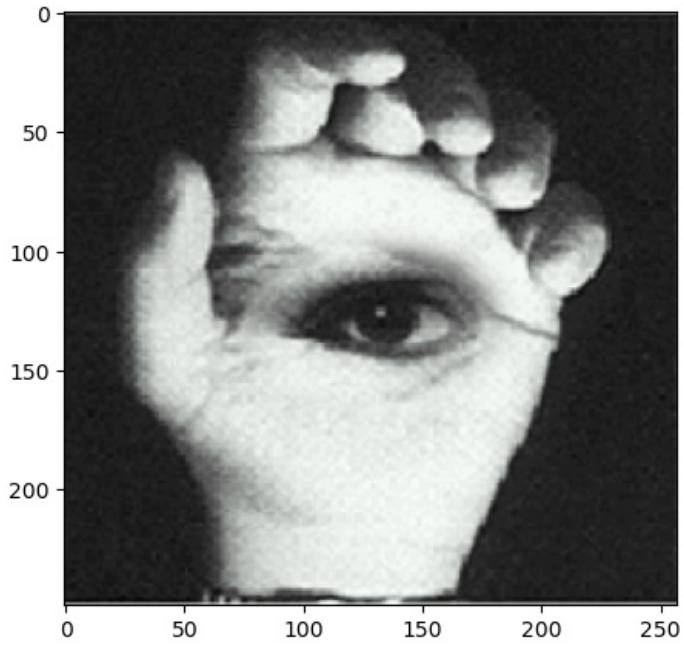
In [ ]:
```
n = normalize_several_yuv(steps)
imshow_yuv(n, size=(14, 8), several=True, blocks=(5, 2))
```



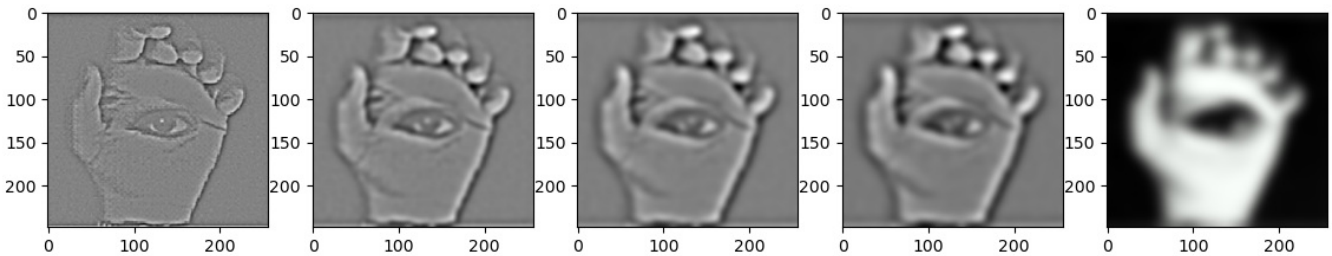


sigma = 3 n_layers = 5

In [ ]:
```
res, steps = glue(img_a, img_b, img_mask, 3, 5)
imshow_yuv(res, size=(10, 5))
```

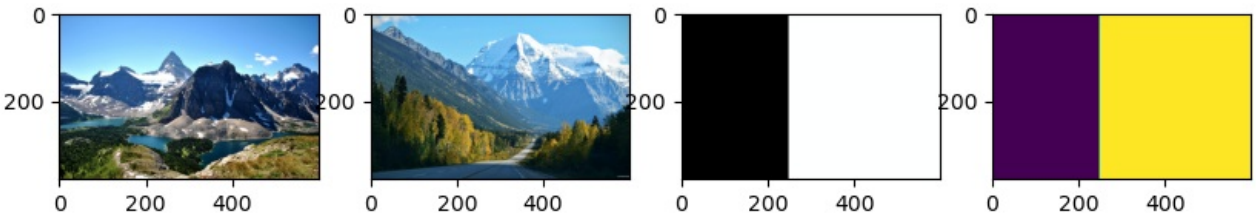


In [ ]:
```
n = normalize_several_yuv(steps)
imshow_yuv(n, size=(14, 8), several=True, blocks=(5, 1))
```

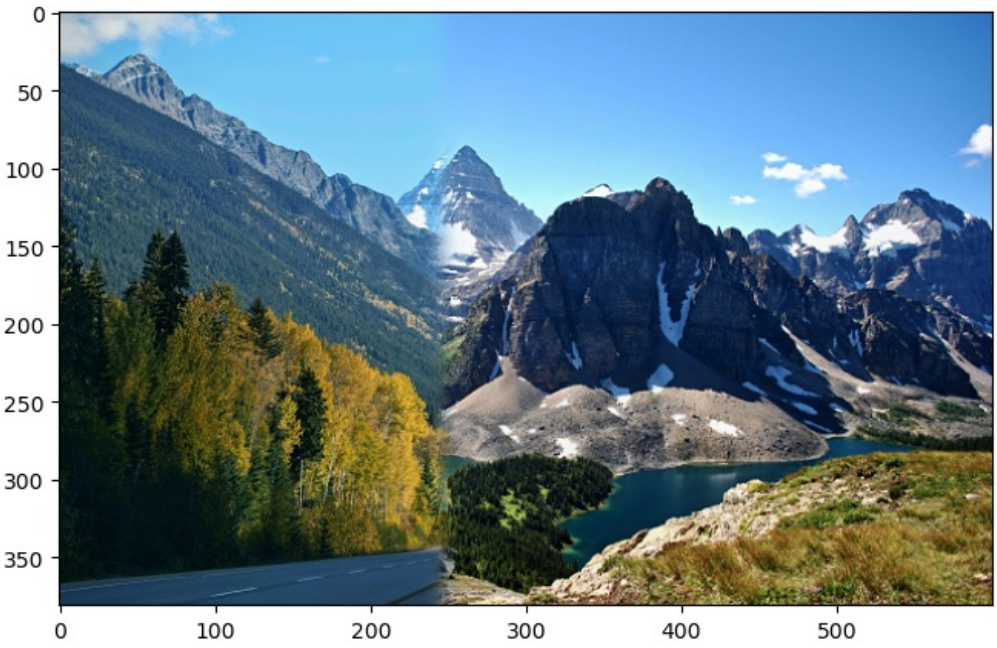


5. Свои примеры

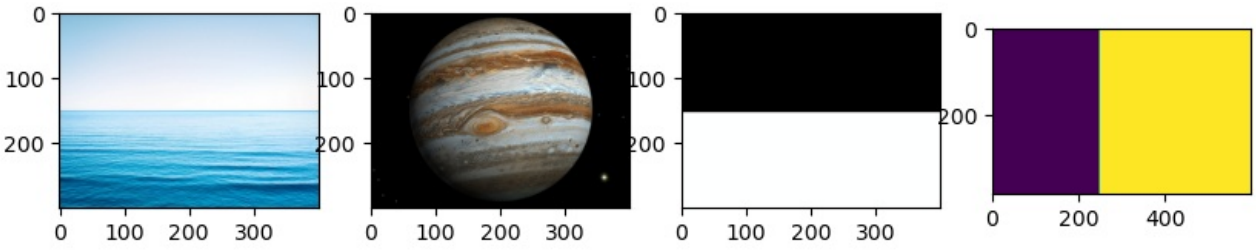
In [ ]:
```
test1 = imread_yuv('test1.png')
test1 = test1[1:, :, :]
test2 = imread_yuv('test2.png')
mask1 = imread_yuv('mask1.png')

imshow_yuv([test1, test2, mask1, binarize(mask1)], size=(10, 5), several=True, blocks=(4, 1))
```



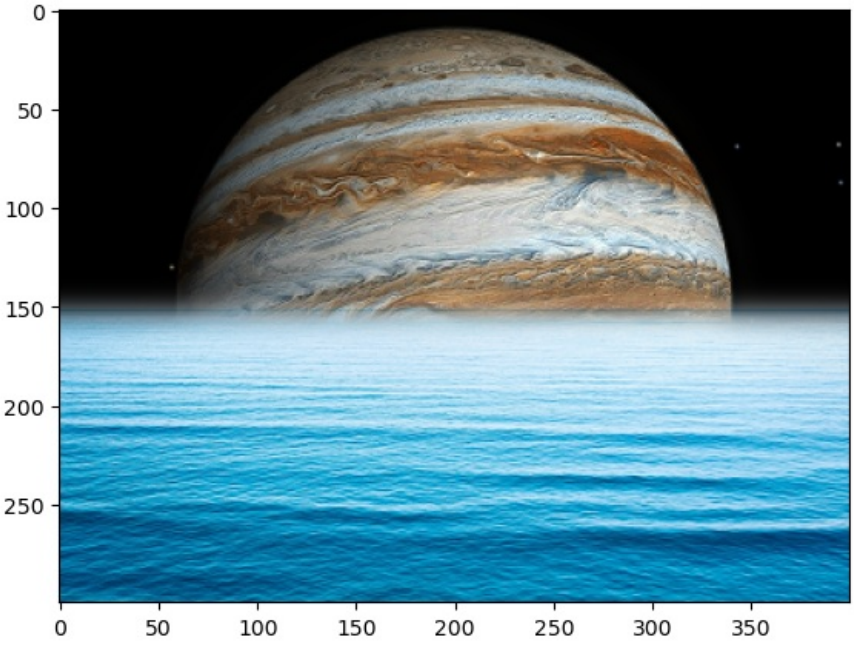
In [ ]:
```
res, _ = glue(test1, test2, mask1, 3, 5)
imshow_yuv(res, size=(10, 5))
```



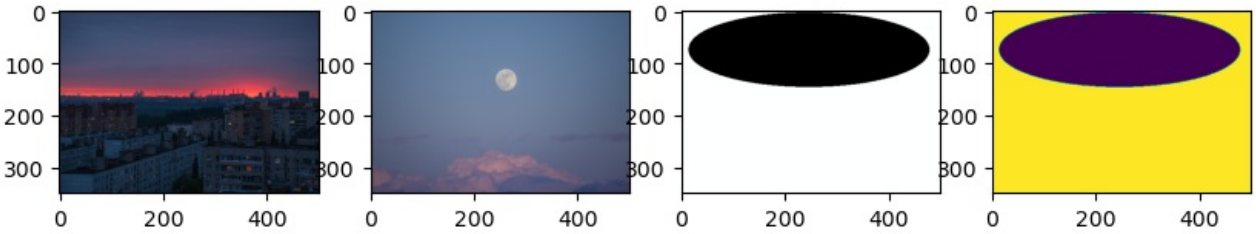
In [ ]:
```
test3 = imread_yuv('test3.png')
test4 = imread_yuv('test4.png')
mask2 = imread_yuv('mask2.png')

imshow_yuv([test3, test4, mask2, binarize(mask1)], size=(10, 5), several=True, blocks=(4, 1))
```



In [ ]:
```
res, _ = glue(test3, test4, mask2, 3, 5)
imshow_yuv(res, size=(10, 5))
```



In [ ]:
```
test5 = imread_yuv('test6.png')
test6 = imread_yuv('test5.png')
mask3 = imread_yuv('Plz.png')

imshow_yuv([test5, test6, mask3, binarize(mask3)], size=(10, 5), several=True, blocks=(4, 1))
```



In [ ]:
```
res, _ = glue(test5, test6, mask3, 3, 5)
imshow_yuv(res, size=(10, 5))
```

In [ ]: