

# Задание по курсу: “Обработка изображений”

## Склейка изображений

### 1. Изображение для проверки алгоритма создание пирамид Гаусса и Лапласа.

Для проверки работы алгоритма было использовано изображение птицы в формате png (рис. 1). Изображение представлено в виде трех цветовых каналов Red, Blue и Green. Перед построением пирамид изображение было переведено в цветовой формат YUV. Так как каналы U и V отвечают преимущественно за цветовую составляющую, работа производилась над Y каналом, который отвечает за яркость изображения.



Рис. 1. Оригинальное изображение для проверки алгоритма

Преобразование в YUV формат производилось по следующим формулам:

$$\begin{aligned}y &= 0.2126*r + 0.7152*g + 0.0722*b \\u &= -0.0999*r - 0.3360*g + 0.4360*b \\v &= 0.6150*r - 0.5586*g - 0.0563*b\end{aligned}$$

где  $r$ ,  $g$ ,  $b$  – красный, зеленый и синий канал соответственно.

### 2. Построение гауссовской пирамиды.

Алгоритм построения гауссовской пирамиды изображения приведен в лекциях к курсу “Обработка изображений”.

При построении гауссовской пирамиды размер изображения уменьшается на  $k$  пикселей по каждому из измерений, где  $k = (\sigma^2 + 1)$ , а  $\sigma$ , в свою очередь, коэффициент гауссовского фильтра. Поэтому при построении гауссовской пирамиды изображений использовался метод зеркалирования границ на величину  $k$  с помощью функции `pad()` из библиотеки NumPy.

Для более быстрой реализации гауссовской фильтрации была использована библиотека `stypes` и реализация свертки с ядром производилась в функции `convolve2d_fl()`, которая вызывается из динамической библиотеки `conv.so` (исходный код приведен в Приложении 1).

Исходный код функции для построения гауссовской пирамиды приведен ниже в Листинге 1.

Результаты работы функции *gauss\_pyramid()* приведены в Таблице 1 для трех различных коэффициентах  $\sigma$ . Количество слоев равно 5.


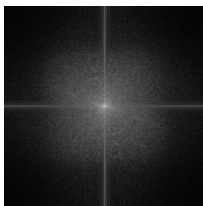

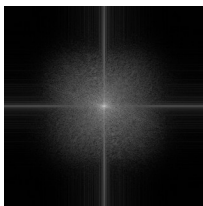

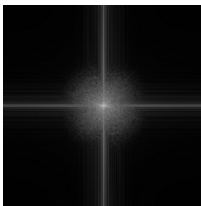
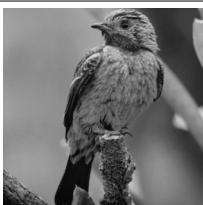
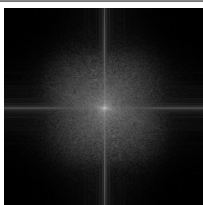
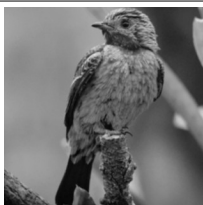
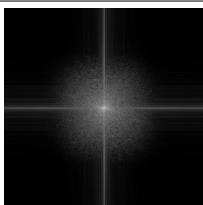
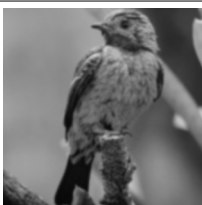
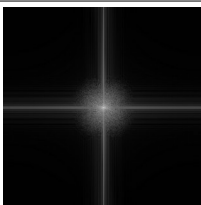
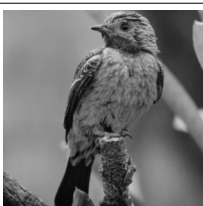
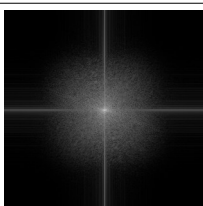
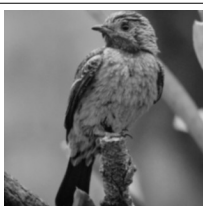
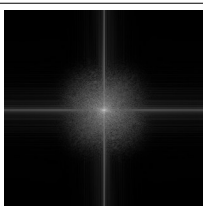
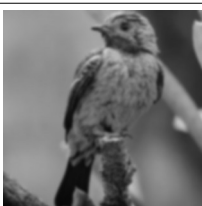
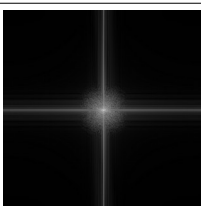
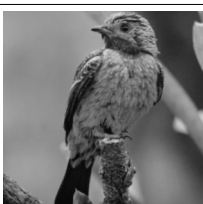
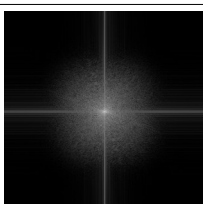
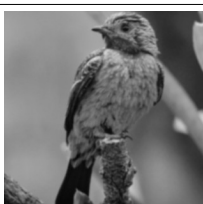
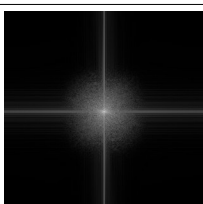
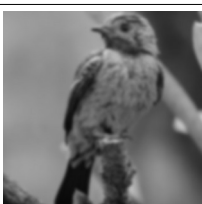
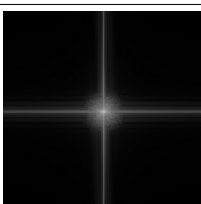
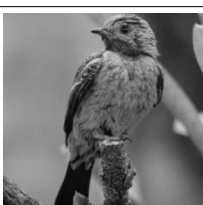
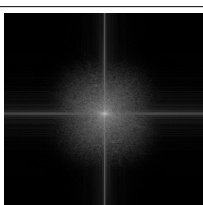
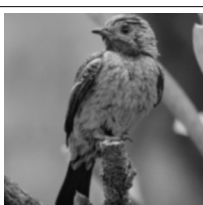
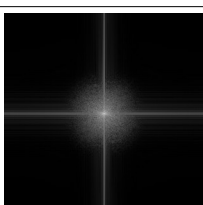
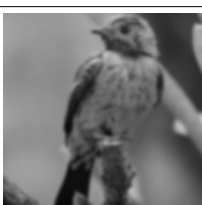
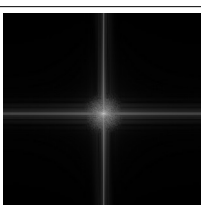
$\sigma = 0.66$		$\sigma = 1$		$\sigma = 2$	
					
					
					
					
					

Таблица 1. Построение гауссовской пирамиды при разных значениях  $\sigma$ ,  $n = 5$

Для частотного представления использовалась функции *fft.fftshift()* и *fft.fft2()* библиотеки *scipy*.

### 3. Построение лапласовской пирамиды.

Функция построения лапласовской пирамиды использует функцию построения гауссовской пирамиды. После того как построена гауссовская пирамида лапласовская пирамида получается последовательным вычитанием слоев гауссовской пирамиды. Последний элемент лапласовской пирамиды равен последнему элементу гауссовской пирамиды.

Результаты работы функции *laplas\_pyramid()* приведены в Таблице 2 для трех различных коэффициентах  $\sigma$ . Количество слоев равно 5. Для наглядности

приводятся визуализация частот в сравнении с гауссовской пирамидой. Как можно видеть, диапазон частот сужается в гауссовской пирамиды, а лапласовская пирамида представляет собой их разность.

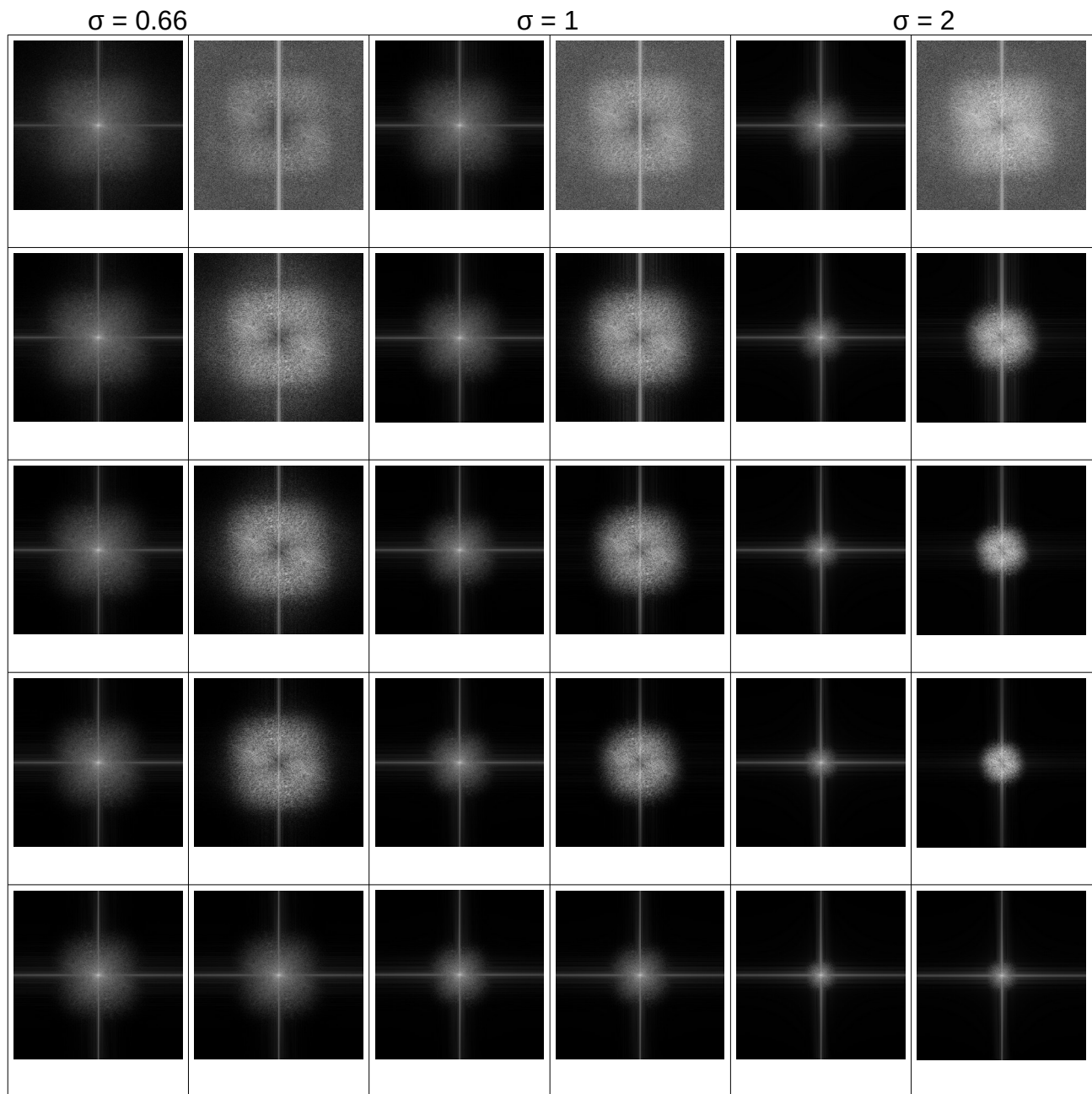


Таблица 2. Построение лапласовской пирамиды при разных значениях  $\sigma$ .  $n = 5$



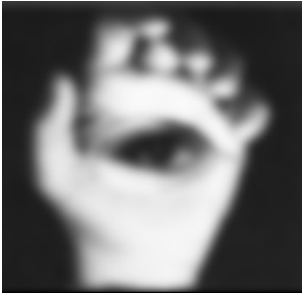


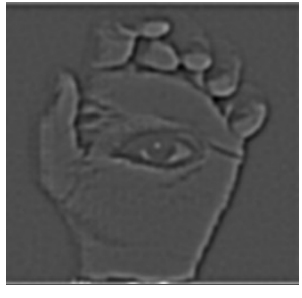

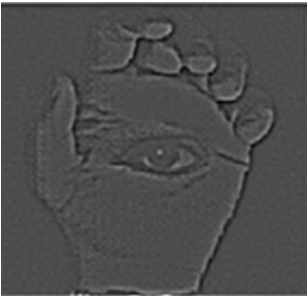
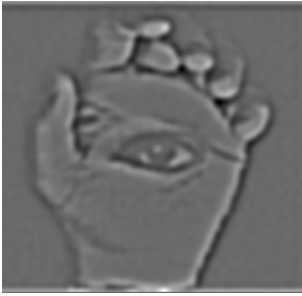

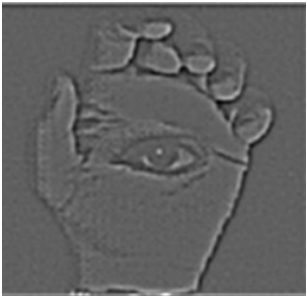
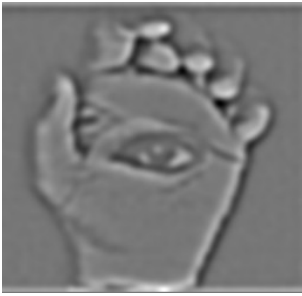


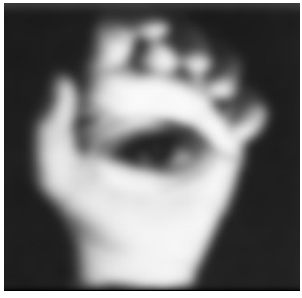
В таблице 2 продемонстрировано как с увеличением  $\sigma$  влияние фильтрации на каждом слое увеличивается. Частоты на последнем слое идентичны гауссовскому представлению, т.к. последний слой лапласовской пирамиды представляет собой копию последнего слоя в гауссовской пирамиде.

4. Склейка тестового изображения

$\sigma = 0.66$

$\sigma = 1$

$\sigma = 2$

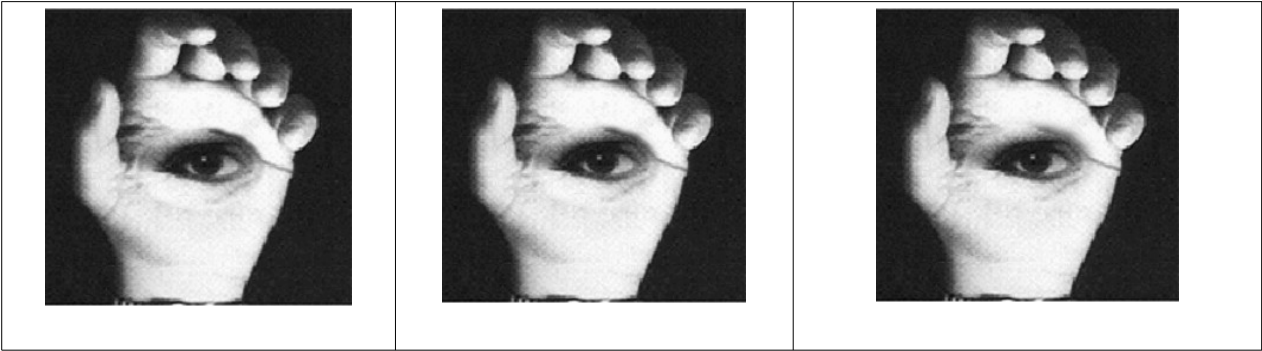
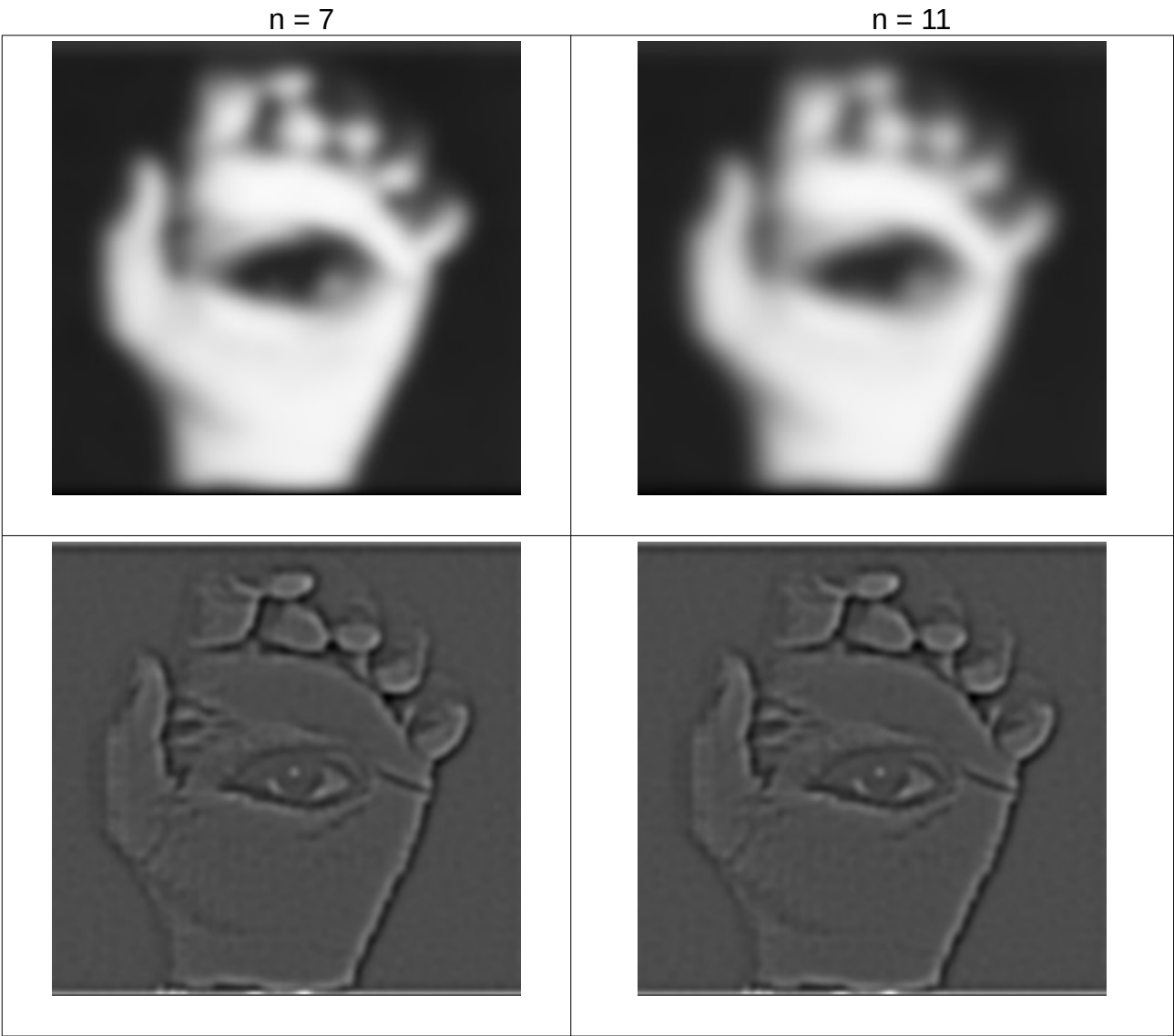
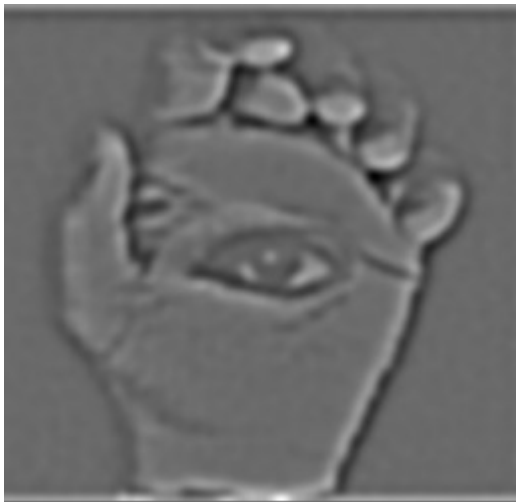


Таблица 3. Склейка тестового набора при разных значениях  $\sigma$  и фиксированном  $n = 5$











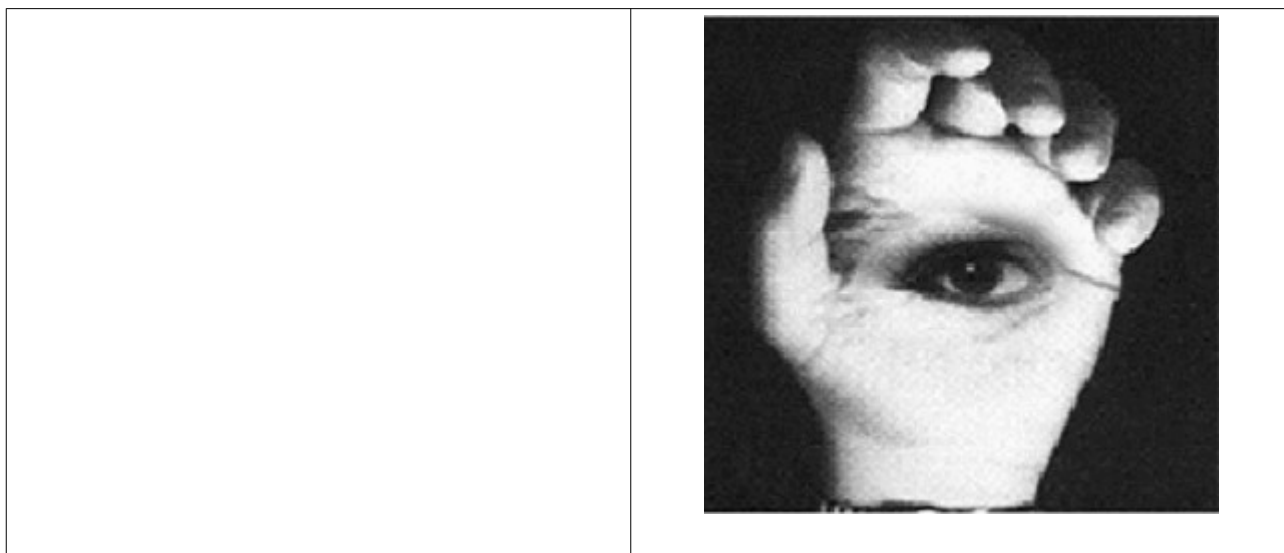
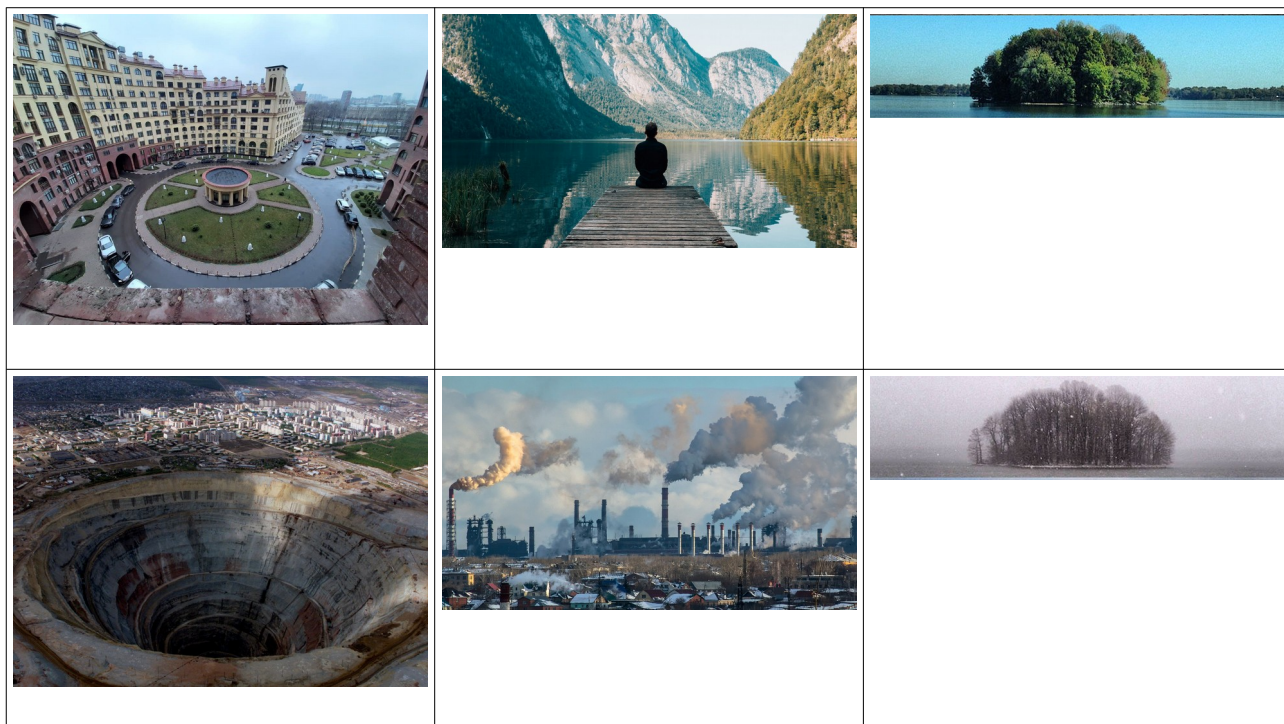


Таблица 4. Склейка тестового набора при фиксированном значении  $\sigma=2$  и разном количестве слоев  $n(7, 11)$ .

Из трех различных комбинаций  $\sigma$  и  $n$ , наилучшими результатами обладает склейка с параметрами  $\sigma = 2$  и  $n = 7$ , т.к. количество слоев  $n = 11$  не дает заметных улучшений, однако при обработке больших изображений работает дольше.

## 5. Склейка трех наборов изображений

Ниже приведены результаты использование склейки изображений с параметрами  $\sigma = 2$  и  $n = 7$ . Показаны исходные изображения, маски и результат. Склейка производилась без перевода изображений в формат YUV, результаты получились со склейкой цветовой компонент



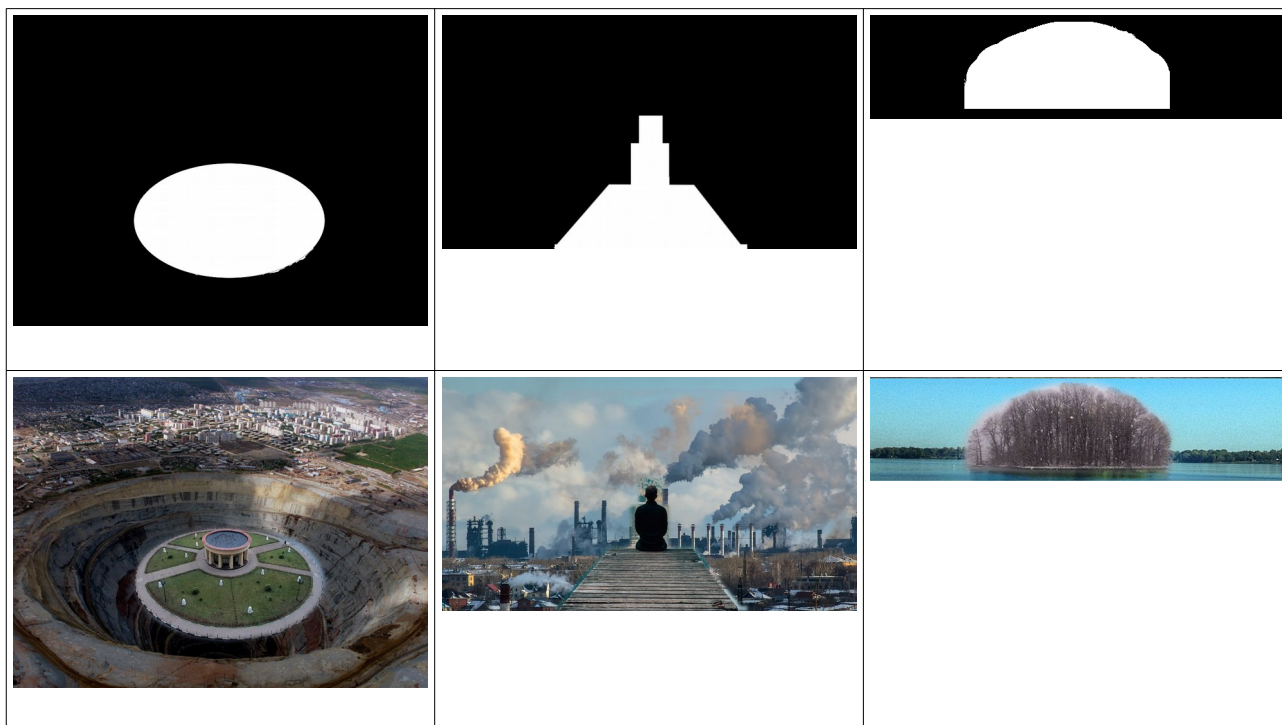


Таблица 5. Склейка произвольных изображений

Ниже приведены изображения в полный размер для оценки визуальных артефактов.



Рис. 2. Склейка изображений





Рис. 3. Слейка изображений



Рис. 4. Слейка изображений

### Листинг 1. Построение гауссовской пирамиды

```
1.     def fast_convolve(part, kern):
2.         h = part.shape[0]
3.         w = part.shape[1]
4.
5.         part = np.ascontiguousarray(part, dtype=np.float32)
6.
7.         val = conv.convolve2d_fl(h, w, part, kern);
8.         val = np.float64(val)
9.         return val
10.
11.    def g_filter_fast(img1, ar):
12.        img = img1.copy()
13.
14.        k = ar.shape[0]
15.        kd2 = k//2
16.
17.        img = np.pad(img, kd2, mode = 'symmetric')
18.        height = img.shape[0]
19.        width = img.shape[1]
20.
21.        out = np.zeros((height-kd2*2, width-kd2*2))
22.
23.        y = -1
24.        for i in range(kd2+1, height-kd2, 1):
25.            y += 1
26.            x = -1
27.            for j in range(kd2+1, width-kd2+1, 1):
28.                x +=1
29.                start = (i-(kd2+1), j-(kd2+1))
30.                end = (i+kd2, j+kd2)
31.                out[y, x] = fast_convolve(img[start[0]:end[0], \
start[1]:end[1]], ar)
32.        return out
33.
34.    def gauss_pyramid(img, sigma, n_layers):
35.        if n_layers is 0:
36.            return 0
37.
38.        ar = create_kernel(sigma)
39.        layer = []
40.        layer.append(g_filter_fast(img, ar))
41.        for i in range(1, n_layers):
42.            layer.append(g_filter_fast(layer[i-1], ar))
43.        return layer
```

## Приложение 1. Код библиотеки conv.so

```
1. #include <stdio.h>
2. #include <float.h>
3.
4. double convolve2d_fl(int part_h, int part_w, float part[][part_w], float
    kern[][part_w]);
5.
6. double convolve2d_fl(int part_h, int part_w, float part[][part_w], float
    kern[][part_w]){
7. double sum = 0.0;
8.
9. for(int i = 0; i < part_h; ++i)
10.     for(int j = 0; j < part_w; ++j)
11.         sum += part[i][j] * kern[i][j];
12.
13.     return sum;
14. }
```

## Приложение 2. Полный код для склейки цветных изображений

```
1. import math
2. import numpy as np
3.
4. from ctypes import *
5. conv = CDLL('./conv.so')
6.
7. conv.convolve2d_fl.restype = c_double
8. conv.convolve2d_fl.argtypes = [
9.     c_int,
10.    c_int,
11.    np.ctypeslib.ndpointer(dtype=np.float32,      ndim=2,
    flags='C_CONTIGUOUS'),
12.    np.ctypeslib.ndpointer(dtype=np.float32,      ndim=2,
    flags='C_CONTIGUOUS')]
13.
14.     from skimage.io import imread, imsave, imshow
15.     import skimage.transform as sk_t
16.     import scipy.signal
17.
18.     from skimage import img_as_float32, img_as_ubyte
19.
20.     def gauss(coef, x, y):
21.         return ((math.exp((-x*x-y*y)/(2*coef*coef)))/(2*math.pi*coef*coef
    ))
22.
23.     def create_kernel(coef) :
24.         k = round(coef*6+1)
25.         k_bound = k//2
26.         sum = 0.0
```

```

27.         ar = []
28.         ar = np.zeros((k, k), dtype=np.float32)
29.
30.         for i in range(0, 2*k_bound+1, 1):
31.             for j in range(0, 2*k_bound+1, 1):
32.                 x = (k_bound) - i
33.                 y = (k_bound) - j
34.                 ar[i, j] = gauss(coef, x, y)
35.                 sum = sum + ar[i, j]
36.
37.         for i in range(-k_bound, k_bound+1, 1):
38.             for j in range(-k_bound, k_bound+1, 1):
39.                 ar[i, j] = ar[i, j]/sum
40.
41.         return ar
42.
43.     def fast_convolve(part, kern):
44.         h = part.shape[0]
45.         w = part.shape[1]
46.
47.         part = np.ascontiguousarray(part, dtype=np.float32)
48.
49.         val = conv.convolve2d_fl(h, w, part, kern);
50.         val = np.float64(val)
51.         return val
52.
53.     def g_filter_fast(img1, ar):
54.         img = img1.copy()
55.
56.         k = ar.shape[0]
57.         kd2 = k//2
58.         img = np.pad(img, kd2, mode = 'symmetric')
59.
60.         height = img.shape[0]
61.         width = img.shape[1]
62.
63.         out = np.zeros((height-kd2*2, width-kd2*2))
64.
65.         y = -1
66.         for i in range(kd2+1, height-kd2, 1):
67.             y += 1
68.             x = -1
69.             for j in range(kd2+1, width-kd2+1, 1):
70.                 x +=1
71.                 start = (i-(kd2+1), j-(kd2+1))
72.                 end = (i+kd2, j+kd2)
73.                 out[y, x] = fast_convolve(img[start[0]:end[0],
start[1]:end[1]], ar)
74.
75.         return out
76.
77.
78.     def gauss_pyramid(img, sigma, n_layers):
79.         if n_layers is 0:

```

```

80.         return 0
81.
82.         ar = create_kernel(sigma)
83.         layer = []
84.         layer.append(g_filter_fast(img, ar))
85.
86.         for i in range(1, n_layers):
87.             print(i)
88.             layer.append(g_filter_fast(layer[i-1], ar))
89.
90.         return layer
91.
92. def laplas_pyramid(img, sigma, n_layers):
93.     laplas = []
94.
95.     gauss = gauss_pyramid(img, sigma, n_layers)
96.     temp = img - gauss[0]
97.     laplas.append(temp)
98.
99.     for i in range(1, n_layers-1):
100.         temp = gauss[i-1] - gauss[i]
101.         laplas.append(temp)
102.
103.     temp = gauss[n_layers-1]
104.     laplas.append(temp)
105.
106.     return laplas, gauss
107.
108. img_a = imread('examples/hole/a.png')
109. img_b = imread('examples/hole/b.png')
110. mask = imread('examples/hole/mask.png')
111. mask = (mask > 128)
112.
113. img_fa = img_as_float32(img_a)
114. img_fb = img_as_float32(img_b)
115. img_mask = img_as_float32(mask)
116.
117. r_a = img_fa[:, :, 0].copy()
118. g_a = img_fa[:, :, 1].copy()
119. b_a = img_fa[:, :, 2].copy()
120.
121. r_b = img_fb[:, :, 0].copy()
122. g_b = img_fb[:, :, 1].copy()
123. b_b = img_fb[:, :, 2].copy()
124.
125. r_mask = img_mask[:, :, 0].copy()
126. g_mask = img_mask[:, :, 1].copy()
127. b_mask = img_mask[:, :, 2].copy()
128. sigma = 2
129. layers = 7
130.
131. # y channel
132. la, ga = laplas_pyramid(r_a, sigma, layers)
133. lb, gb = laplas_pyramid(r_b, sigma, layers)

```

```

134.     gm = gauss_pyramid(r_mask, sigma, layers)
135.
136.     ly = [] #result pyramid
137.
138.     for i in range(0, layers):
139.         temp = gm[i]*la[i]+(1-gm[i])*lb[i]
140.         ly.append(temp)
141.
142.     y_img = ly[0]
143.     imsave('examples/hole/out' + str(1) + '.png', ly[i])
144.
145.     for i in range(1, layers):
146.         imsave('examples/hole/out' + str(i+1) + '.png', ly[i])
147.         y_img += ly[i]
148.
149.     imsave('examples/hole/r_channel.png', y_img)
150.
151.     # v channel
152.     la, ga = laplas_pyramid(g_a, sigma, layers)
153.     lb, gb = laplas_pyramid(g_b, sigma, layers)
154.     gm = gauss_pyramid(g_mask, sigma, layers)
155.     lv = [] #result pyramid
156.
157.     for i in range(0, layers):
158.         temp = gm[i]*la[i]+(1-gm[i])*lb[i]
159.         lv.append(temp)
160.
161.     v_img = lv[0]
162.
163.     for i in range(1, layers):
164.         v_img += lv[i]
165.
166.     imsave('examples/hole/g_channel.png', v_img)
167.
168.     # u channel
169.     la, ga = laplas_pyramid(b_a, sigma, layers)
170.     lb, gb = laplas_pyramid(b_b, sigma, layers)
171.     gm = gauss_pyramid(b_mask, sigma, layers)
172.     lu = [] #result pyramid
173.
174.     for i in range(0, layers):
175.         temp = gm[i]*la[i]+(1-gm[i])*lb[i]
176.         lu.append(temp)
177.
178.     u_img = lu[0]
179.
180.     for i in range(1, layers):
181.         u_img += lu[i]
182.
183.     imsave('examples/hole/b_channel.png', u_img)
184.
185.     out_img = np.zeros((y_img.shape[0], y_img.shape[1], 3),
dtype=np.float32)
186.     out_img[:, :, 0] = y_img

```



```
187.     out_img[:, :, 1] = v_img
188.     out_img[:, :, 2] = u_img
189.
190.     out_img = np.clip(out_img, 0, 1)
191.
192.     img2 = img_as_ubyte(out_img)
193.     imsave('examples/hole/out_img.png', img2)
194.
195.     imshow(img2)
```