Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**Дисциплина: Алгоритмы цифровой обработки мультимедиа**

**Тема: Методы фильтрации и размытия изображения**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Эзри

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Крамаренко

**Цель работы:** реализовать алгоритм размытия по Гауссу.

**Ход работы:**

**Задание 1.** Выполнить пункты 1 и 2 алгоритма, то есть построить матрицу Гаусса. Просмотреть итоговую матрицу для размерностей 3, 5, 7.

Была реализована функция Гаусса  и построены матрицы свёртки размерностей 3x3, 5x5 и 7x7 со значением среднеквадратического отклонения, равным 3. Можно заметить, что значения больше в центре и меньше по краям.

Kernel size: 3

Standard deviation: 3

[[0.01582423 0.01672824 0.01582423]

[0.01672824 0.01768388 0.01672824]

[0.01582423 0.01672824 0.01582423]]

Kernel size: 5

Standard deviation: 3

[[0.01133856 0.01339492 0.01416015 0.01339492 0.01133856]

[0.01339492 0.01582423 0.01672824 0.01582423 0.01339492]

[0.01416015 0.01672824 0.01768388 0.01672824 0.01416015]

[0.01339492 0.01582423 0.01672824 0.01582423 0.01339492]

[0.01133856 0.01339492 0.01416015 0.01339492 0.01133856]]

Kernel size: 7

Standard deviation: 3

[[0.00650554 0.00858856 0.01014619 0.01072582 0.01014619 0.00858856 0.00650554]

[0.00858856 0.01133856 0.01339492 0.01416015 0.01339492 0.01133856 0.00858856]

[0.01014619 0.01339492 0.01582423 0.01672824 0.01582423 0.01339492 0.01014619]

[0.01072582 0.01416015 0.01672824 0.01768388 0.01672824 0.01416015 0.01072582]

[0.01014619 0.01339492 0.01582423 0.01672824 0.01582423 0.01339492 0.01014619]

[0.00858856 0.01133856 0.01339492 0.01416015 0.01339492 0.01133856 0.00858856]

[0.00650554 0.00858856 0.01014619 0.01072582 0.01014619 0.00858856 0.00650554]]

**Задание 2.** Нормировать полученную матрицу Гаусса. Протестировать результаты на матрицах из предыдущего пункта.

Чтобы нормировать матрицу, поделим каждый её элемент на сумму значений всех элементов этой матрицы: kernel /= np.sum(kernel).

Kernel size: 3

Standard deviation: 3

[[0.1069973 0.11310982 0.1069973 ]

[0.11310982 0.11957153 0.11310982]

[0.1069973 0.11310982 0.1069973 ]]

Sum of kernel elements: 1.0

Kernel size: 5

Standard deviation: 3

[[0.0317564 0.03751576 0.03965895 0.03751576 0.0317564 ]

[0.03751576 0.04431963 0.04685151 0.04431963 0.03751576]

[0.03965895 0.04685151 0.04952803 0.04685151 0.03965895]

[0.03751576 0.04431963 0.04685151 0.04431963 0.03751576]

[0.0317564 0.03751576 0.03965895 0.03751576 0.0317564 ]]

Sum of kernel elements: 1.0

Kernel size: 7

Standard deviation: 3

[[0.01129725 0.01491455 0.01761946 0.01862602 0.01761946 0.01491455 0.01129725]

[0.01491455 0.01969008 0.02326108 0.02458993 0.02326108 0.01969008 0.01491455]

[0.01761946 0.02326108 0.02747972 0.02904957 0.02747972 0.02326108 0.01761946]

[0.01862602 0.02458993 0.02904957 0.03070911 0.02904957 0.02458993 0.01862602]

[0.01761946 0.02326108 0.02747972 0.02904957 0.02747972 0.02326108 0.01761946]

[0.01491455 0.01969008 0.02326108 0.02458993 0.02326108 0.01969008 0.01491455]

[0.01129725 0.01491455 0.01761946 0.01862602 0.01761946 0.01491455 0.01129725]]

Sum of kernel elements: 1.0

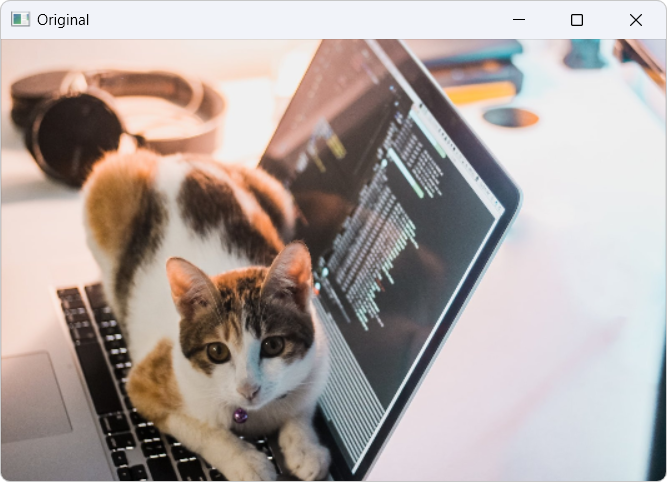
**Задание 3.** Реализовать фильтр Гаусса средствами языка Python.

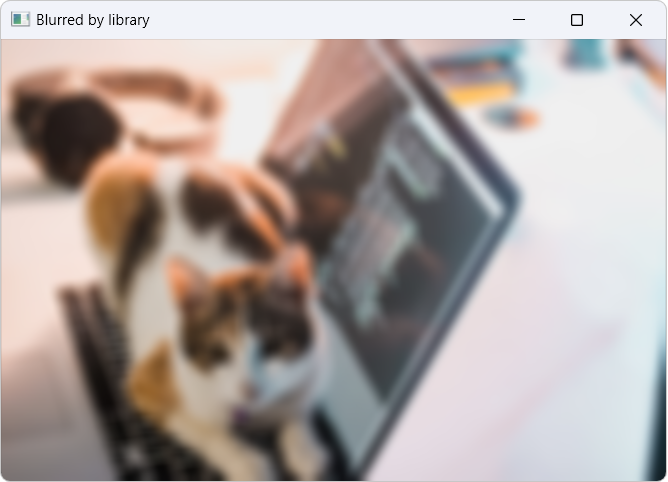
Для этого была реализована функция gaussian\_blur, которая принимает чёрно-белое изображение (двумерный массив), размер матрицы ядра свёртки и стандартное отклонение в качестве параметров, строит матрицу Гаусса, нормирует её, создаёт копию изображения и для каждого внутреннего пикселя изображения рассчитывает новое значение яркости пикселя по формуле и записывает это значение в пиксель нового изображения.

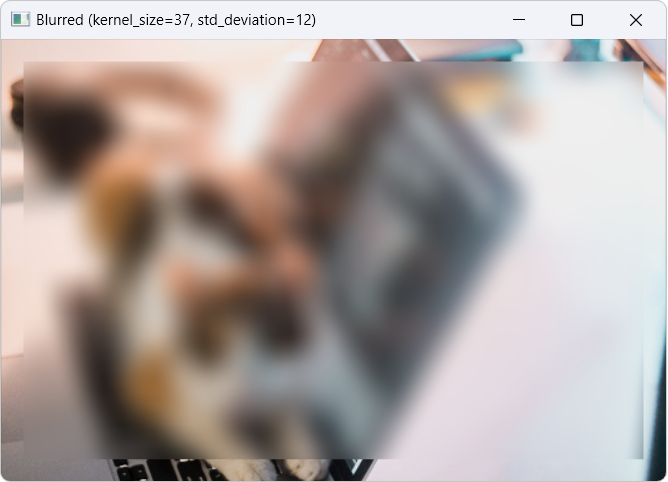
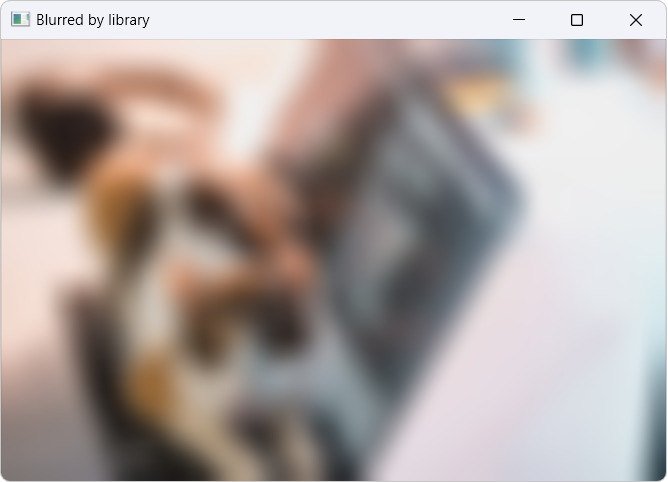
Также я реализовал размытие цветных изображений – для этого каждый из каналов размывается по отдельности и затем три размытых изображения собираются в одно цветное изображение.

**Задание 4.** Применить данный фильтр для двух разных значений среднего квадратичного отклонения и двух разных размерностей матрицы свёртки, сравнить результаты для ОДНОГО изображения.

Верхнее изображения – исходное. Ниже – размерность 15x15, стандартное отклонение 5, ещё ниже – размерность 37x37, стандартное отклонение 12. Слева – моя реализация, справа – из библиотеки OpenCV.



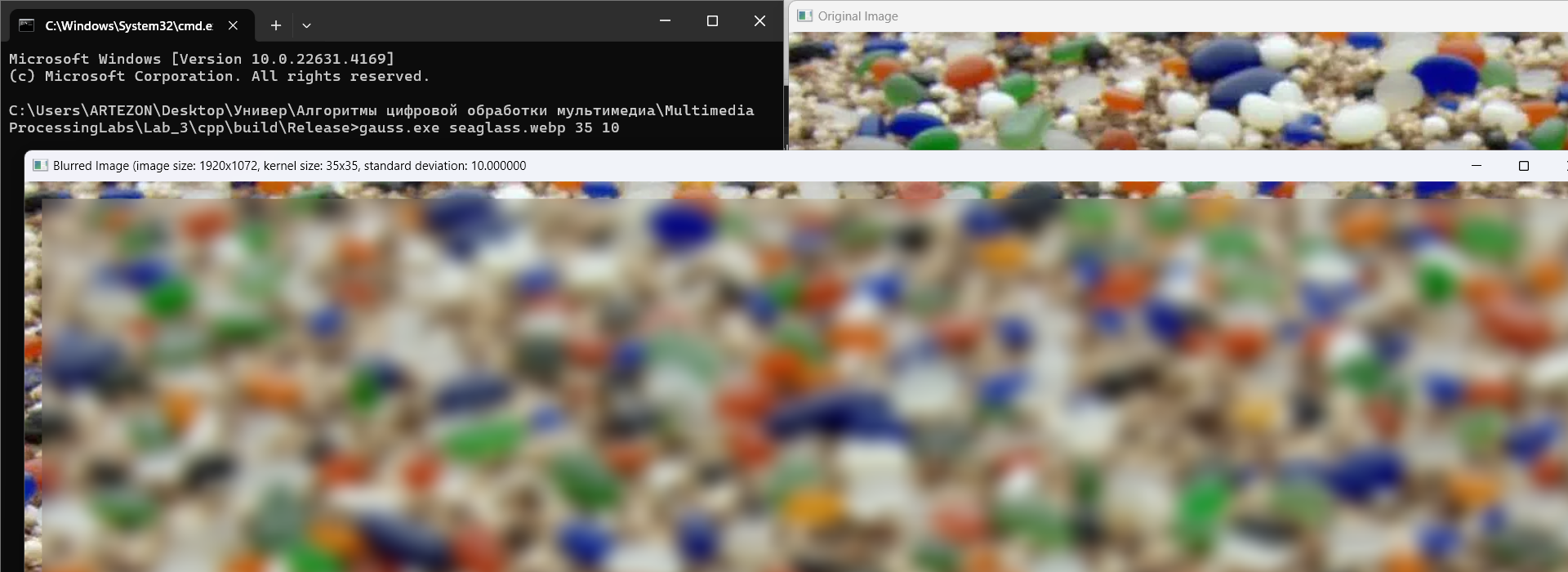
 

**Задание 5.** Реализовать размытие Гаусса встроенным методом библиотеки OpenCV, сравнить результаты с Вашей реализацией.

Размытие встроенным методом библиотеки OpenCV выполняется следующим образом: img\_blur\_lib = cv2.GaussianBlur(img, (kernel\_size, kernel\_size), std\_deviation). Сравнение с моей реализацией выше.

**Задание 6.** (самостоятельно) Реализовать размытие Гаусса средствами любого другого языка программирования.

Алгоритм размытия по Гауссу был реализован на C++. Для открытия изображений была использована библиотека OpenCV для C++ и система сборки CMake. Алгоритм проводит операции над стандартными векторами (каждый вектор содержит вектор целых чисел). Программа показывает результаты при помощи функции imshow. Программа принимает аргументы: gauss.exe <Путь к изображению> <Размерность матрицы свёртки> <Среднеквадратическое отклонение>.



**Листинг программы**

Файл Lab\_3\_Task\_1.py

import numpy as np  
  
  
def gauss(x, y, sigma, a, b):  
 two\_sigma\_squared = 2 \* sigma \* sigma  
 return np.exp(-((x - a) \*\* 2 + (y - b) \*\* 2) / two\_sigma\_squared) / (np.pi \* two\_sigma\_squared)  
  
  
def generate\_kernel(kernel\_size, std\_deviation):  
 kernel = np.zeros((kernel\_size, kernel\_size))  
 a = b = kernel\_size // 2 # математическое ожидание двумерной случайной величины  
  
 # Строим матрицу свёртки  
 for y in range(kernel\_size):  
 for x in range(kernel\_size):  
 kernel[y, x] = gauss(x, y, std\_deviation, a, b)  
  
 return kernel  
  
  
std\_deviation = 3  
for kernel\_size in (3, 5, 7):  
 print(f'\nKernel size: {kernel\_size}')  
 print(f'Standard deviation: {std\_deviation}')  
 print(generate\_kernel(kernel\_size, std\_deviation))

Файл Lab\_3\_Task\_2.py

import numpy as np  
  
  
def gauss(x, y, sigma, a, b):  
 two\_sigma\_squared = 2 \* sigma \* sigma  
 return np.exp(-((x - a) \*\* 2 + (y - b) \*\* 2) / two\_sigma\_squared) / (np.pi \* two\_sigma\_squared)  
  
  
def generate\_kernel(kernel\_size, std\_deviation):  
 kernel = np.zeros((kernel\_size, kernel\_size))  
 a = b = kernel\_size // 2 # математическое ожидание двумерной случайной величины  
  
 # Строим матрицу свёртки  
 for y in range(kernel\_size):  
 for x in range(kernel\_size):  
 kernel[y, x] = gauss(x, y, std\_deviation, a, b)  
  
 return kernel  
  
  
std\_deviation = 3  
for kernel\_size in (3, 5, 7):  
 print(f'\nKernel size: {kernel\_size}')  
 print(f'Standard deviation: {std\_deviation}')  
 kernel = generate\_kernel(kernel\_size, std\_deviation)  
 kernel /= np.sum(kernel) # нормируем матрицу  
 print(kernel)  
 print(f'Sum of kernel elements: {np.sum(kernel)}')

Файл Lab\_3\_Tasks\_3\_4\_5\_Final.py

import numpy as np  
import cv2  
  
  
def gauss(x, y, sigma, a, b):  
 two\_sigma\_squared = 2 \* sigma \* sigma  
 return np.exp(-((x - a) \*\* 2 + (y - b) \*\* 2) / two\_sigma\_squared) / (np.pi \* two\_sigma\_squared)  
  
  
def generate\_kernel(kernel\_size, std\_deviation):  
 kernel = np.zeros((kernel\_size, kernel\_size))  
 a = b = kernel\_size // 2 # математическое ожидание двумерной случайной величины  
  
 # Строим матрицу свёртки  
 for y in range(kernel\_size):  
 for x in range(kernel\_size):  
 kernel[y, x] = gauss(x, y, std\_deviation, a, b)  
  
 return kernel  
  
  
def gaussian\_blur(img, kernel\_size, std\_deviation):  
 kernel = generate\_kernel(kernel\_size, std\_deviation)  
 print(kernel)  
 print(np.sum(kernel))  
  
 kernel /= np.sum(kernel) # нормируем матрицу  
 print(kernel)  
 print(np.sum(kernel))  
  
 # Проходим через внутренние пиксели изображения и выполняем операцию свёртки.  
 # Каждый стоящий рядом пиксель изображения умножается на соответствующее значение в ядре,  
 # а затем все результаты суммируются и записываются в пиксель размытого изображения.  
 blurred = img.copy()  
 h, w = img.shape[:2]  
 half\_kernel\_size = int(kernel\_size // 2)  
 for y in range(half\_kernel\_size, h - half\_kernel\_size):  
 for x in range(half\_kernel\_size, w - half\_kernel\_size):  
 # Операция свёртки  
 val = 0  
 for k in range(-(kernel\_size // 2), kernel\_size // 2 + 1):  
 for l in range(-(kernel\_size // 2), kernel\_size // 2 + 1):  
 val += img[y + k, x + l] \* kernel[k + half\_kernel\_size, l + half\_kernel\_size]  
 blurred[y, x] = val  
  
 return blurred  
  
  
kernel\_size = 15  
std\_deviation = 5  
  
folder = '../images/'  
  
# img = cv2.imread(folder + 'seaglass.webp')  
img = cv2.imread(folder + 'cat.jpg')  
img = cv2.resize(img, [i // 3 for i in img.shape[1::-1]])  
# img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
  
if kernel\_size % 2 == 0:  
 print('Размер матрицы ядра должен быть нечётный')  
 exit(1)  
  
if img.ndim == 2:  
 img\_blur\_mine = gaussian\_blur(img, kernel\_size, std\_deviation)  
else:  
 img\_blur\_mine = np.empty\_like(img)  
 channels = img.shape[2]  
 for i in range(channels):  
 img\_blur\_mine[:, :, i] = gaussian\_blur(img[:, :, i], kernel\_size, std\_deviation)  
  
img\_blur\_lib = cv2.GaussianBlur(img, (kernel\_size, kernel\_size), std\_deviation)  
  
cv2.imshow('Original', img)  
cv2.imshow(f'Blurred (kernel\_size={kernel\_size}, std\_deviation={std\_deviation})', img\_blur\_mine)  
cv2.imshow('Blurred by library', img\_blur\_lib)  
  
# cv2.imwrite(f'blur\_{kernel\_size}\_{std\_deviation}.jpg', img\_blur\_mine)  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**Ответы на вопросы**

1. Опишите, в чем заключается принцип операции размытия изображения и зачем ее применяют?

Базовый подход к фильтрации строится на том, что значение пикселя должно изменится под влиянием окружающих пикселей. Эффект размытия широко используется для уменьшения шума изображения и снижения детализации.

2. Что такое операция свёртки матрицы?

Операция свёртки матрицы представляет собой математическую операцию, при которой ядро перемещается по элементам изображения, и для каждой позиции выполняется умножение элементов изображения на соответствующие элементы ядра, а затем суммирование результатов.

3. Каким образом строится матрица свёртки в размытии по Гауссу?

Матрица свертки в размытии по Гауссу представляет собой 2D матрицу (ядро), которая формируется на основе гауссовой функции с математическим ожиданием в центре матрицы свёртки, а затем нормализуется, чтобы сумма элементов матрицы равнялась единице.

4. Опишите алгоритм размытия по Гауссу.

1) установить размер матрицы свёртки и среднее квадратичное отклонение;

2) заполнить матрицу свёртки значениями функции Гаусса с мат. ожиданием, равным координатам центра матрицы;

3) нормировать матрицу так, чтобы сумма элементов равнялась 1;

4) создать копию изображения;

5) для каждого внутреннего пикселя копии рассчитать новое значение пикселя по формуле и записать это значение в пиксель нового изображения.

5. Укажите известные Вам параметры размытия по Гауссу.

Параметры размытия по Гауссу включают стандартное отклонение гауссовой функции (определяет степень размытия) и размер ядра (определяет, насколько далеко распространяются веса от центрального пикселя).

6. Каким образом производить фильтрацию по Гауссу средствами библиотеки OpenCV?

В OpenCV для фильтрации по Гауссу используется функция cv2.GaussianBlur(). Эта функция принимает изображение, размер ядра (kernel\_size) и стандартное отклонение (sigma) гауссовой функции в качестве аргументов и возвращает размытое изображение.