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**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Миленченко А.Р.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

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

**Цель работы:** Реализация матрицы Гаусса, тестирование полученного и встроенного метода размытия по Гауссу на выбранном изображении.

Размытие – один из базовых алгоритмов обработки изображения. Размытие строится из предположения, что каждый пиксель картинки по своему значению связан с предыдущим и не должен заметно от него отличатся. Если же так происходит, например, вокруг пикселя яркость 240, а в самом пикселе 12, то мы предполагаем, что данный пиксель – это шумы. И размытие картинки – один из главных способов нивелировать эти шумы. Базовый подход к фильтрации строится на том, что значение пикселя должно изменится под влиянием окружающих пикселей.

Для начала переведем изображение в черно-белый формат, в этом случае каждый пиксель представляет собой только одно целое неотрицательное значение длиной 8 бит – яркость. Далее рассмотрим изображение как двумерную матрицу размерности ℎ × 𝑤:

Теперь рассмотрим произвольный пиксель данной матрицы 𝑎𝑖𝑗:

Значение яркости данного пикселя будем менять с учетом всех указанных пикселей вокруг заданного. Для этого необходимо выполнить операцию свертки. Операция свертки заключается в преобразовании исходной матрицы B размерности 𝑛 × 𝑛 в числовое значение с помощью специальной матрицы ker размерности 𝑛 × 𝑛, называемой ядром свертки:

Матрица ker берется с различными значениями для разных типов фильтрации, стандартное условие – данная матрица должна иметь сумму элементов близкую к единице.

Гауссовский фильтр построен на ядре свертки, полученной как значения функции Гаусса (плотность распределения) для двумерной случайной величины:

Здесь 𝑎, 𝑏 – математическое ожидание двумерной случайной величины.

Для построения матрицы в качестве этих значений выбираем координаты центрального элемента матрицы, например, для матрицы 5 × 5 значения математического ожидания будут следующими: 𝑎 = 3, 𝑏 = 3. В качестве 𝑥, 𝑦 выбираем все индексы всех элементов матрицы поочередно. Значение 𝜎 будем считать параметром размытия по Гауссу. При каждом новом значении 𝜎 результат размытия будет отличаться.

Размерность матрицы свертки – это другой возможный параметр фильтрации.

Далее применяем операцию свертки ко всем внутренним пикселям

изображения и записываем вычисленное по формуле (1) число в качестве нового значения яркости пикселя.

Итого алгоритм кратко будет выглядеть следующим образом:

1. установить размер матрицы свертки и среднее квадратичное отклонение;
2. заполнить матрицу свертки значениями функции Гаусса с мат.
3. ожиданием, равным координатам центра матрицы;
4. нормировать матрицу так, чтобы сумма элементов равнялась 1;
5. создать копию изображения;
6. для каждого внутреннего пикселя копии рассчитать новое значение
7. насыщенности пикселя по формуле (1) и ЗАПИСАТЬ это значение в пиксель нового изображения

Задание 1. Выполнить пункты 1 и 2 алгоритма, то есть построить

матрицу Гаусса. Просмотреть итоговую матрицу для размерностей 3, 5, 7.

Задание 2. Нормировать полученную матрицу Гаусса. Протестировать

результаты на матрицах из предыдущего пункта.

Задание 3. Реализовать фильтр Гаусса средствами языка python.

Задание 4. Применить данный фильтр для двух разных значений

среднего квадратичного отклонения и двух разных размерностей матрицы свертки, сравнить результаты для ОДНОГО изображения.

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

Задание 6 Реализовать размытие Гаусса средствами любого другого языка программирования.

**Итоговые размытия:**

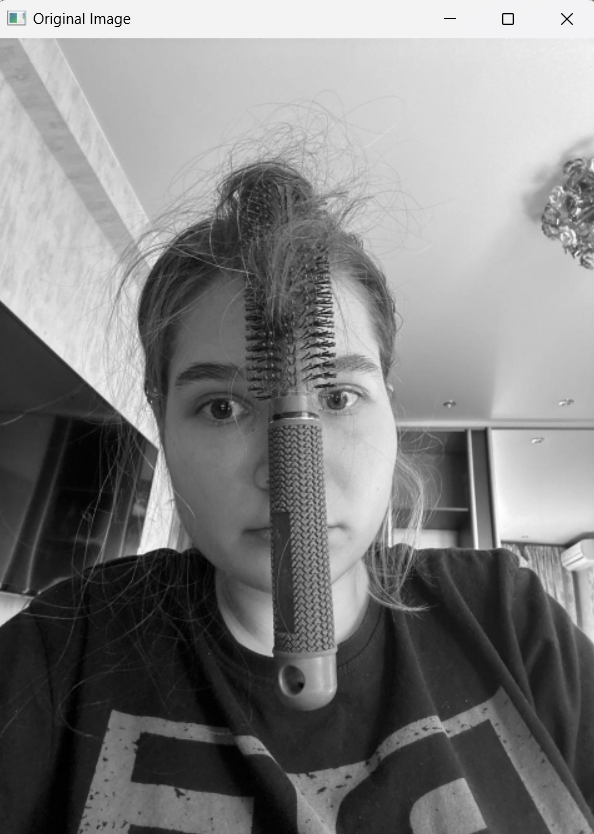


Рисунок 1- исходное изображение



Рисунок 2- размытие 3х3 и стандартным отклонением 100



Рисунок 1- размытие 11х11 и стандартным отклонением 100

**Ответы на вопросы:**

1. Принцип операции размытия изображения: Операция размытия (или фильтрации) изображения заключается в уменьшении контрастности и выделения деталей на изображении. Это применяется для сглаживания шума, уменьшения высокочастотных компонентов и создания эффекта размытия, который может быть полезным в различных задачах обработки изображений.

2. Операция свертки матрицы: Операция свертки матрицы (или ядра) 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. Матрица свертки в размытии по Гауссу: Матрица свертки в размытии по Гауссу представляет собой 2D матрицу (ядро), которая формируется на основе гауссовой функции. Эта матрица определяет, как веса пикселей изображения должны учитываться при операции размытия.

4. Алгоритм размытия по Гауссу: Для размытия по Гауссу сначала строится матрица свертки на основе гауссовой функции. Затем эта матрица нормализуется. После этого применяется операция свертки между изображением и матрицей свертки, чтобы получить размытое изображение.

5. 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. Фильтрация по Гауссу средствами 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.

**Листинг программы**

import cv2

import numpy as np

def gaussian\_function(x, y, sigma, center\_x, center\_y):

coefficient = 1 / (2 \* np.pi \* (sigma \*\* 2))

exponent = np.exp(-((x - center\_x) \*\* 2 + (y - center\_y) \*\* 2) / (2 \* (sigma \*\* 2)))

return coefficient \* exponent

def gaussian\_blur(input\_image, kernel\_dimension, sigma\_value):

kernel\_matrix = np.ones((kernel\_dimension, kernel\_dimension))

center = (kernel\_dimension - 1) / 2

# Заполнение ядра значениями функции Гаусса

for i in range(kernel\_dimension):

for j in range(kernel\_dimension):

kernel\_matrix[i, j] = gaussian\_function(i, j, sigma\_value, center, center)

print("Kernel Matrix for size: ", kernel\_dimension)

print(kernel\_matrix)

# Нормализация ядра

kernel\_sum = kernel\_matrix.sum()

kernel\_matrix /= kernel\_sum

print("Normalized Kernel Matrix: \n", kernel\_matrix, "\n")

blurred\_image = input\_image.copy()

offset = kernel\_dimension // 2

# Применение гауссового размывания

for y in range(offset, blurred\_image.shape[0] - offset):

for x in range(offset, blurred\_image.shape[1] - offset):

pixel\_value = 0

for ky in range(-offset, offset + 1):

for kx in range(-offset, offset + 1):

pixel\_value += input\_image[y + ky, x + kx] \* kernel\_matrix[ky + offset, kx + offset]

blurred\_image[y, x] = pixel\_value

return blurred\_image

# Загрузка изображения и применение фильтра

original\_image = cv2.imread("nuvotivsi.jpg", cv2.IMREAD\_GRAYSCALE)

blurred\_1 = gaussian\_blur(original\_image, 3, 100)

# blurred\_2 = gaussian\_blur(original\_image, 3, 10)

# blurred\_3 = gaussian\_blur(original\_image, 5, 100)

# blurred\_4 = gaussian\_blur(original\_image, 5, 10)

# blurred\_5 = gaussian\_blur(original\_image, 7, 10)

blurred\_6 = gaussian\_blur(original\_image, 11, 100)

# Размер для отображения

scaling\_factor = 50

new\_width = int(original\_image.shape[1] \* scaling\_factor / 100)

new\_height = int(original\_image.shape[0] \* scaling\_factor / 100)

new\_dimensions = (new\_width, new\_height)

resized\_original = cv2.resize(original\_image, new\_dimensions, interpolation=cv2.INTER\_AREA)

resized\_blur1 = cv2.resize(blurred\_1, new\_dimensions, interpolation=cv2.INTER\_AREA)

# resized\_blur2 = cv2.resize(blurred\_2, new\_dimensions, interpolation=cv2.INTER\_AREA)

# resized\_blur3 = cv2.resize(blurred\_3, new\_dimensions, interpolation=cv2.INTER\_AREA)

# resized\_blur4 = cv2.resize(blurred\_4, new\_dimensions, interpolation=cv2.INTER\_AREA)

# resized\_blur5 = cv2.resize(blurred\_5, new\_dimensions, interpolation=cv2.INTER\_AREA)

resized\_blur6 = cv2.resize(blurred\_6, new\_dimensions, interpolation=cv2.INTER\_AREA)

cv2.imshow("Original Image", resized\_original)

cv2.imshow("Kernel size 3 | Std Dev 100", resized\_blur1)

# cv2.imshow("Kernel size 3 | Std Dev 10", resized\_blur2)

# cv2.imshow("Kernel size 5 | Std Dev 100", resized\_blur3)

# cv2.imshow("Kernel size 5 | Std Dev 10", resized\_blur4)

# cv2.imshow("Kernel size 7 | Std Dev 100", resized\_blur5)

cv2.imshow("Kernel size 7 | Std Dev 100", resized\_blur6)

cv2.waitKey(0)

cv2.destroyAllWindows()