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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Романов В.В.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Крамаренко А.А.

**Цель:** изучить методы фильтрации и размытия изображения.

**Ход работы:**

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.

Алгоритм выглядит следующим образом:

1. Устанавливаем размер матрицы свертки и среднее квадратичное отклонение;
2. Заполняем матрицу свертки значениями функции Гаусса с математическим ожиданием, равным координатам центра матрицы;
3. Нормируем матрицу так, чтобы сумма элементов равнялась 1;
4. Создаем копию изображения;
5. Для каждого внутреннего пикселя копии рассчитываем новое значение насыщенности пикселя по формуле (1) и записываем это значение в пиксель нового изображения.

Загружаем изображение для работы с ним. На рисунке 1 показано исходное изображение.



Рисунок 1 – Исходное изображение.

С помощью встроенных функций Python преобразуем исходное изображение в черно-белый формат и используем размытие Гаусса. На рисунке 2 представлен результат работы программы.

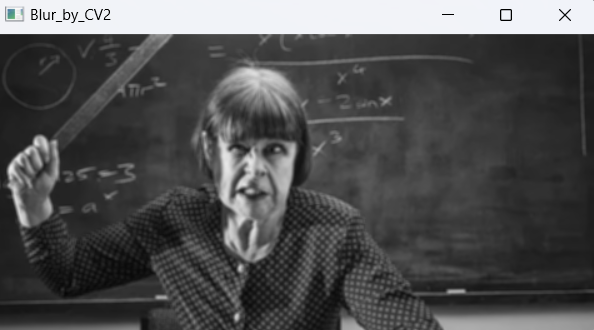


Рисунок 2 – Размытие OpenCV.

Размеры ядра свертки и стандартного отклонения можно устанавливать вручную, поэтому посмотрим результаты работы программы на разных значениях параметров. На рисунках 3-4 изображены результаты при размере ядра 3 на 3 с отклонением 50, а также 11 на 11 с отклонением 9.



Рисунок 3 – Размытие 3х3 и стандартным отклонением 50.

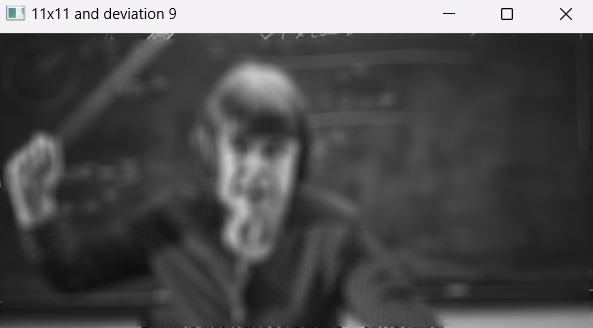


Рисунок 4 – Размытие 11х11 и стандартным отклонением 9.

Кроме того, был переписан код из ЯП Python в C++ для реализации размытия Гаусса. Код исходников на 2-ух разных языках программирования приведен ниже в листинге программы.

**Вывод:** изучил методы фильтрации и размытия изображений.

**Листинг программы**

**Файл GaussBlur.py**

import cv2  
import numpy as np  
  
def main():  
  
 *# Задание №1*  
img = cv2.imread(r'test.jpg', cv2.IMREAD\_GRAYSCALE)  
  
 standard\_deviation = 9  
 kernel\_size = 11  
  
 imgBlur\_1 = AnotherGaussianBlur(img, kernel\_size, standard\_deviation)  
 cv2.imshow('Original', img)  
 cv2.imshow(str(kernel\_size) + 'x' + str(kernel\_size) + ' and deviation ' + str(standard\_deviation), imgBlur\_1)  
  
 *# Задание №4*  
standard\_deviation = 50  
 kernel\_size = 3  
 imgBlur\_2 = AnotherGaussianBlur(img, kernel\_size, standard\_deviation)  
 cv2.imshow(str(kernel\_size) + 'x' + str(kernel\_size) + ' and deviation ' + str(standard\_deviation), imgBlur\_2)  
  
 *# Задание №5*  
imgBlur\_CV2 = cv2.GaussianBlur(img, (kernel\_size, kernel\_size), standard\_deviation)  
 cv2.imshow('Blur\_by\_CV2', imgBlur\_CV2)  
 cv2.waitKey(0)  
  
*# Задание №3*  
def AnotherGaussianBlur(img, kernel\_size, standard\_deviation):  
 kernel = np.ones((kernel\_size, kernel\_size)) *# Первоначальное ядро свёртки*  
a = b = (kernel\_size + 1) // 2 *# Вычисление центрального элемента матрицы (определения пикселя в фокусе)*  
  
 *# Построение матрицы свёртки*  
for i in range(kernel\_size):  
 for j in range(kernel\_size):  
 kernel[i, j] = gauss(i, j, standard\_deviation, a, b) *# Вычисление функции Гаусса*  
print(kernel)  
  
 *# Задание №2*  
sum = 0  
 for i in range(kernel\_size):  
 for j in range(kernel\_size):  
 sum += kernel[i, j]  
 for i in range(kernel\_size):  
 for j in range(kernel\_size):  
 kernel[i, j] /= sum  
 print(kernel)  
  
 *# Применение операции свёртки к изображению*  
imgBlur = Convolution(img, kernel)  
 return imgBlur  
  
*# Реализация операции свёртки*  
def Convolution(img, kernel):  
 kernel\_size = len(kernel)  
 imgBlur = img.copy()  
 *# Начальные координаты для итераций по пикселям*  
x\_start = kernel\_size // 2  
 y\_start = kernel\_size // 2  
 for i in range(x\_start, imgBlur.shape[0] - x\_start):  
 for j in range(y\_start, imgBlur.shape[1] - y\_start):  
 *# Операция свёртки - каждый пиксель умножается на соответствующий элемент ядра свертки и суммирование произведений*  
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[i + k, j + l] \* kernel[k + (kernel\_size // 2), l + (kernel\_size // 2)]  
 imgBlur[i, j] = val  
 return imgBlur  
  
def gauss(x, y, omega, a, b):  
  
 *# Вычисляем удвоенный квадрат параметра omega для упрощения дальнейших вычислений*  
omegaIn2 = 2 \* omega \*\* 2  
  
 *# m1 - первая часть формулы гауссовой функции, которая зависит от omegaIn2.*  
 *# Она вычисляет нормализационный множитель так, чтобы интеграл от гауссовой функции по всей плоскости был = 1.*  
 *# Это обеспечивает, что сумма всех значений гауссовой функции будет равна 1.*  
m1 = 1 / (np.pi \* omegaIn2)  
  
 *# m2 - вторая часть формулы гауссовой функции, которая вычисляет экспоненциальное значение с отрицательным аргументом.*  
 *# (x-a) \*\* 2 и (y-b) \*\* 2 - квадраты расстояний от точки (x, y) до центра гауссовой функции (a, b).*  
 *# Делим на omegaIn2 и берем экспоненту от отрицательного результата для определения того,*  
 *# насколько быстро значение гауссовой функции убывает с расстоянием от центра (a, b).*  
m2 = np.exp(-((x - a) \*\* 2 + (y - b) \*\* 2) / omegaIn2)  
  
 *# Возвращаем окончательное значение гауссовой функции для заданных координат (x, y).*  
 *# Это значение будет наибольшим в центре (a, b) и уменьшается с удалением от центра согласно форме гауссовой кривой*  
return m1 \* m2  
  
main()

**Файл GaussBlur.cpp**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

// Реализация функции Гаусса

double gauss(double x, double y, double omega, double a, double b) {

double omegaIn2 = 2 \* pow(omega, 2);

double m1 = 1 / (3.14 \* omegaIn2);

double m2 = exp(-(pow((x - a), 2) + pow((y - b), 2)) / omegaIn2);

return m1 \* m2;

}

// Реализация фильтра Гаусса

vector<vector<double>> gaussianBlur(vector<vector<double>>& img, int kernel\_size, double standard\_deviation) {

int a = (kernel\_size + 1) / 2;

int b = (kernel\_size + 1) / 2;

// Построение матрицы ядра свертки

vector<vector<double>> kernel(kernel\_size, vector<double>(kernel\_size));

for (int i = 0; i < kernel\_size; ++i) {

for (int j = 0; j < kernel\_size; ++j) {

kernel[i][j] = gauss(i, j, standard\_deviation, a, b);

}

}

// Нормализация матрицы ядра свертки

double sum = 0;

for (int i = 0; i < kernel\_size; ++i) {

for (int j = 0; j < kernel\_size; ++j) {

sum += kernel[i][j];

}

}

for (int i = 0; i < kernel\_size; ++i) {

for (int j = 0; j < kernel\_size; ++j) {

kernel[i][j] /= sum;

}

}

// Применение операции свертки

int img\_height = img.size();

int img\_width = img[0].size();

vector<vector<double>> img\_blurred(img\_height, vector<double>(img\_width));

for (int i = a; i < img\_height - a; ++i) {

for (int j = b; j < img\_width - b; ++j) {

double val = 0;

for (int k = -a; k <= a; ++k) {

for (int l = -b; l <= b; ++l) {

val += img[i + k][j + l] \* kernel[k + a][l + b];

}

}

img\_blurred[i][j] = val;

}

}

return img\_blurred;

}

int main() {

vector<vector<double>> img = { {1.0, 2.0, 3.0, 4.0, 5.0},

{6.0, 7.0, 8.0, 9.0, 10.0},

{11.0, 12.0, 13.0, 14.0, 15.0},

{16.0, 17.0, 18.0, 19.0, 20.0},

{21.0, 22.0, 23.0, 24.0, 25.0} };

int kernel\_size = 5;

double standard\_deviation = 100.0;

vector<vector<double>> img\_blurred = gaussianBlur(img, kernel\_size, standard\_deviation);

// Вывод размытого изображения

for (const auto& row : img\_blurred) {

for (const auto& pixel : row) {

cout << pixel << " ";

}

cout << endl;

}

return 0;

}