САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ПЕТРА ВЕЛИКОГО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Лабораторная работа №2**

**Методы сглаживания изображений**

**Дисциплина**: Разработка графических приложений

Выполнил студент гр. 13541/3 Балсутьев В.А.

(подпись)

Руководитель Абрамов Н.А.

(подпись)

“ ” 2018 г.

Санкт – Петербург

2018

# Содержание

[Оглавление 2](#_Toc530530503)

[Задание 3](#_Toc530530504)

[Фильтр Гаусса 3](#_Toc530530505)

[Билатеральный фильтр 3](#_Toc530530506)

[Фильтр NLMeans 3](#_Toc530530507)

[Код программы 5](#_Toc530530508)

[Фильтр Гаусса 9](#_Toc530530509)

[Билатеральный фильтр 10](#_Toc530530510)

[NLMeans фильтр 11](#_Toc530530511)

[Выводы 12](#_Toc530530512)

# Задание

Реализовать методы сглаживания изображений с использованием следующих подходов: Gaussian Blur; Bilateral Filter; Non – local means.

## Фильтр Гаусса

Фильтр Гаусса —фильтр, чьей импульсной переходной функцией является функция Гаусса. Фильтр Гаусса (Gaussian filter) обычно используется в цифровом виде для обработки двумерных сигналов (изображений) с целью снижения уровня шума. Однако при ресемплинге он дает сильное размытие изображения.

Гауссова функция (гауссиан, гауссиана, функция Гаусса) — вещественная функция, описываемая следующей формулой:



где параметры a,b,c — произвольные вещественные числа.

Гауссовы функции и их дискретные аналоги (такие, как дискретное гауссово ядро) используются в цифровой обработке сигналов, обработке изображений, синтезе звука; в частности, через гауссианы определяются гауссов фильтр и гауссово размытие.

## Билатеральный фильтр

Введённый Tomasi и Manduchi билатеральный фильтр, сохраняющий края, нашёл широкое применение во многих задачах по обработке изображений, например, фильтрация шума , редактирование текстуры и тона, оценки оптического потока. Билатеральная фильтрация также часто используется в качестве начального этапа обработки кадров, например, для задачи распознавания объектов, где необходимо отфильтровать несущественные детали и шумы при сохранении резких краев основного изображения. Основным недостатком билатеральных фильтров являются большие вычислительные затраты. Билатеральная фильтрация (двунаправленная фильтрация) - это нелинейный и не итерационный процесс, комбинирующий пространственную (domain) и яркостную (range) фильтрацию. Таким образом, учитывается не только значения интенсивности близлежащих пикселей, но и их расстояние до текущего фильтруемого пикселя. Вклад близлежащих пикселей существенен по отношению к остальным.

## Фильтр NLMeans

Это алгоритм обработки изображений для шумоподавления. В отличие от фильтров" local mean", которые принимают среднее значение группы пикселей, окружающих целевой пиксель, для сглаживания изображения, фильтрация нелокальными средствами принимает среднее значение всех пикселей в изображении, взвешенное по тому, насколько похожи эти пиксели на целевой пиксель. Это приводит к гораздо большей ясности постфильтрации и меньшей потере деталей изображения по сравнению с локальными средними алгоритмами. По сравнению с другими известными методами шумоподавления нелокальные средства добавляют "методический шум" (т. е. ошибку в процессе шумоподавления), который больше похож на белый шум, что желательно, поскольку он обычно менее тревожен в шумоизолированном продукте.

Цель весовой функции-определить, насколько тесно изображение в точке p связано с изображением в точке q. Она может принимать различные формы.

Весовая функция Гаусса



# Код программы

#include <iostream>

#include <opencv2/opencv.hpp>

#define SIGMA 0.5

using namespace cv;

using namespace std;

double gaussianFunc(int x, int y, double sigma) {

return((1 / (2 \* CV\_PI\*sigma\*sigma))\*exp(-(x\*x + y \* y) / (2 \* sigma\*sigma)));

}

void generateKernel(int size, Mat& kernel) {

kernel = Mat(size, size, CV\_32F);

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

kernel.at<float>(i, j) = gaussianFunc(i - (size - 1) / 2, j - (size - 1) / 2, SIGMA);

}

}

}

uchar countGaussValue(int x, int y, Mat& img, Mat& kernel) {

float acc = 0;

for (int i = 0; i < kernel.rows; i++) {

for (int j = 0; j < kernel.cols; j++) {

if (((y - (kernel.cols - 1) / 2) + i) >= 0 && ((y - (kernel.cols - 1) / 2) + i) < img.rows && ((x - (kernel.cols - 1) / 2) + j) >= 0 && ((x - (kernel.cols - 1) / 2) + j) < img.cols) {

acc += img.at<uchar>((y - (kernel.cols - 1) / 2) + i, (x - (kernel.cols - 1) / 2) + j)\*kernel.at<float>(i, j);

}

else continue;

}

}

return (uchar)acc;

}

Mat neighboursValues(int area, Mat& src, int x, int y) {

Mat values = Mat::zeros(area\*area, 1, CV\_8U);

for (int i = 0; i < area; i++) {

for (int j = 0; j < area; j++) {

values.at<uchar>(j + i, 0) = src.at<uchar>((x - (area - 1) / 2) + j, (y - (area - 1) / 2) + i);

}

}

return values;

}

double normOfVector(Mat& vec1, Mat& vec2) {

double norm = 0;

for (int i = 0; i < vec1.rows; i++) {

norm = norm + pow((vec1.at<uchar>(i, 0) - vec2.at<uchar>(i, 0)), 2);

}

norm = sqrt(norm);

return norm;

}

float distance(int x, int y, int i, int j) {

return float(sqrt(pow(x - i, 2) + pow(y - j, 2)));

}

double gaussian(float x, double sigma) {

return exp(-(pow(x, 2)) / (2 \* pow(sigma, 2))) / (2 \* CV\_PI \* pow(sigma, 2));

}

void applyBilateralFilter(Mat& source, Mat& filteredImage, int x, int y, int diameter, double sigmaI, double sigmaS) {

double iFiltered = 0;

double wP = 0;

int xNeighbor = 0;

int yNeighbor = 0;

int half = diameter / 2;

for (int i = 0; i < diameter; i++) {

for (int j = 0; j < diameter; j++) {

xNeighbor = x - (half - i);

yNeighbor = y - (half - j);

double gr = gaussian(source.at<uchar>(xNeighbor, yNeighbor) - source.at<uchar>(x, y), sigmaI);

double gs = gaussian(distance(x, y, xNeighbor, yNeighbor), sigmaS);

double w = gr \* gs;

iFiltered = iFiltered + source.at<uchar>(xNeighbor, yNeighbor) \* w;

wP = wP + w;

}

}

iFiltered = iFiltered / wP;

filteredImage.at<uchar>(x, y) = (uchar)iFiltered;

}

void nonLocalMeans(Mat& source, Mat& filteredImage, int x, int y, int diameter, double sigmaI) {

double iFiltered = 0;

double wP = 0;

int xNeighbor = 0;

int yNeighbor = 0;

int half = diameter / 2;

for (int i = 0; i < diameter; i++) {

for (int j = 0; j < diameter; j++) {

xNeighbor = x - (half - i);

yNeighbor = y - (half - j);

Mat vector1 = neighboursValues(5, source, x, y);

Mat vector2 = neighboursValues(5, source, xNeighbor, yNeighbor);

double vecNorm = normOfVector(vector1, vector2);

double gr = gaussian(vecNorm, sigmaI);

double w = gr;

iFiltered = iFiltered + source.at<uchar>(xNeighbor, yNeighbor) \* w;

wP = wP + w;

}

}

iFiltered = iFiltered / wP;

filteredImage.at<uchar>(x, y) = (uchar)iFiltered;

}

Mat bilateralFilterOwn(Mat& source, int diameter, double sigmaI, double sigmaS) {

Mat filteredImage = Mat::zeros(source.rows, source.cols, CV\_8U);

int width = source.cols;

int height = source.rows;

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

applyBilateralFilter(source, filteredImage, i, j, diameter, sigmaI, sigmaS);

}

}

return filteredImage;

}

Mat NLMeansFilterOwn(Mat& source, int diameter, double sigmaI) {

Mat resultImage = Mat::zeros(source.rows, source.cols, CV\_8U);

int width = source.cols;

int height = source.rows;

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

nonLocalMeans(source, resultImage, i, j, diameter, sigmaI);

}

}

return resultImage;

}

void applyGaussFilter(Mat& img, int kernelSize) {

Mat gauss;

generateKernel(kernelSize, gauss);

Size imgSize = img.size();

for (int i = 0; i < imgSize.height; i++) {

for (int j = 0; j < imgSize.width; j++) {

img.at<uchar>(i, j) = countGaussValue(j, i, img, gauss);

}

}

}

int main()

{

string imgName = "test.png";

Mat src;

src = imread(imgName, CV\_LOAD\_IMAGE\_GRAYSCALE);

cout << "TYPE: " << typeToString(src.type());

namedWindow("Initial img", WINDOW\_AUTOSIZE);

imshow("Initial img", src);

if (!src.data)

{

printf("No image data \n");

return -1;

}

Mat gaussTest;

Mat gaussTestOpenCV;

Mat bilateralTest;

Mat bilateralTestOpenCV;

Mat NLMeansTest;

Mat NLMeansTestOpenCV;

src.copyTo(gaussTest);

src.copyTo(bilateralTest);

src.copyTo(NLMeansTest);

applyGaussFilter(gaussTest, 7);

imwrite("filtered\_image\_gaussian.png", gaussTest);

GaussianBlur(src, gaussTestOpenCV, Size(7,7), 1);

imwrite("filtered\_image\_gaussian\_opencv.png", gaussTestOpenCV);

Mat bilateralFilteredImage = bilateralFilterOwn(bilateralTest, 5, 6.0, 16.0);

imwrite("filtered\_image\_bilateral.png", bilateralFilteredImage);

bilateralFilter(src, bilateralTestOpenCV, 5, 6.0, 16.0);

imwrite("filtered\_image\_bilateral\_opencv.png", bilateralTestOpenCV);

Mat NLMeansFilteredImage = NLMeansFilterOwn(src, 5, 15);

imwrite("filtered\_image\_nlmeans.png", NLMeansFilteredImage);

fastNlMeansDenoising(src, NLMeansTestOpenCV, 5, 7, 5);

imwrite("filtered\_image\_nlmeans\_opencv.png", NLMeansTestOpenCV);

waitKey(0);

}

## Фильтр Гаусса

Исходное изображение:



Sigma = 0.8:



Sigma = 1:



## Билатеральный фильтр

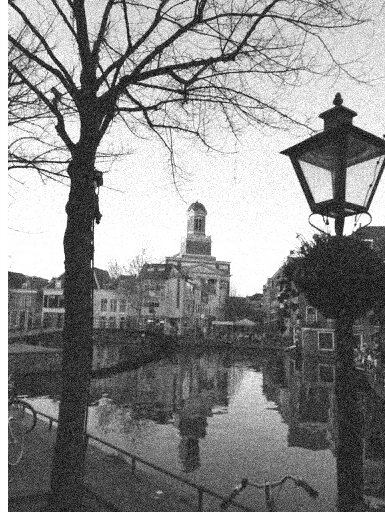
Исходное изображение:



Sigma = 6.0 :



Sigma = 15.0:



## NLMeans фильтр

Исходное изображение:



Sigma = 7.0:



Sigma = 15.0:



# Выводы

В работе были рассмотрены следующие алгоритмы сглаживания изображений:

* Фильтр Гаусса;
* Билатеральный фильтр;
* Фильтр локализированных усреднений;

В ходе работы было определено, что фильтр Гаусса дает наибольшее размытие, таким образом на изображении исчезает детализация. Билатеральный фильтр дает хороший результат при работе с шумом. По сравнению с фильтром Гаусса, билатеральный фильтр и фильтр NLMeans дают лучшую степень сглаживания без потери элементов на изображении. При сравнении работы билатерального фильтра и фильтра NLMeans было определено, что фильтр NLMeans дает лучший результат, хотя разница в работе фильтров NLMeans и билатерального не столь велика как при сравнении их работы с работой фильтра Гаусса.