МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ТВ

ОТЧЁТ

по лабораторной работе № 3

по дисциплине «Цифровая обработка изображений»

Тема: ЦИФРОВАЯ ФИЛЬТРАЦИЯ ИЗОБРАЖЕНИЙ

Студенты гр. 9105 _____ Шаривзянов Д. Р.

_____ Басманов А. А.

Преподаватель _____ Поздеев А. А.

Санкт-Петербург

2024

ЦИФРОВАЯ ФИЛЬТРАЦИЯ ИЗОБРАЖЕНИЙ

Цели работы: знакомство с фильтрацией изображений посредством разработки и применения фильтров сглаживания, выделения границ, повышения резкости и прочих.

Код программы:

```
#include <iostream>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <vector>

using namespace std;
using namespace cv;

struct Image{
    //default
    Mat bgr;
    Mat gray;

    //smoothing
    Mat gauss3;
    Mat gauss5;
    Mat median3;

    //custom
    Mat mosaic3;

    //sharpening
    Mat aperture_cor_3;

    //edges
    Mat dog;
    Mat canny;
    Mat sobel_vert;         // в отчёт
    Mat sobel_hor;
    Mat scharr_vert;        // в отчёт
    Mat scharr_hor;         // в отчёт
    Mat prewitt_vert;
    Mat prewitt_hor;
};

void gauss3(const Mat &src, Mat &dst){
    dst = Mat::zeros(src.size(), CV_8U);
    float k = 36; // коэффициент нормировки
    float Fk[3][3] = { {1,4,1}, {4,16,4}, {1,4,1} }; // маска фильтра
    for (int i = 1; i < src.cols - 1; i++)
        for (int j = 1; j < src.rows - 1; j++) {
        uchar pix_value = src.at<uchar>(j, i);
        // далее производим свертку
        float Rez = 0;
        for (int ii = -1; ii <= 1; ii++)
            for (int jj = -1; jj <= 1; jj++) {
                uchar blurred = src.at<uchar>(j + jj, i + ii);
                Rez += Fk[ii + 1][jj + 1] * blurred;
```

```
            }
        uchar blurred = Rez / k; // осуществляем нормировку
        dst.at<uchar>(j, i) = blurred;
    }
}

void gauss5(const Mat &src, Mat &dst){
    dst = Mat::zeros(src.size(), CV_8U);
    float k = 273; // коэффициент нормировки
    float Fk[5][5] = {
    {1, 4,  7,  4,  1},
    {4, 16, 26, 16, 4},
    {7, 26, 41, 26, 7},
    {4, 16, 26, 16, 4},
    {1, 4,  7,  4,  1}
    }; // маска фильтра
    for (int i = 2; i < src.cols - 2; i++)
        for (int j = 2; j < src.rows - 2; j++) {
            uchar pix_value = src.at<uchar>(j, i);
            float Rez = 0;
            for (int ii = -2; ii <= 2; ii++)
                for (int jj = -2; jj <= 2; jj++) {
                    uchar blurred = src.at<uchar>(j + jj, i + ii);
                    Rez += Fk[ii + 2][jj + 2] * blurred;
                }
            uchar blurred = Rez / k; // осуществляем нормировку
            dst.at<uchar>(j, i) = blurred;
    }
}

void dog(const Mat &srcGauss1, const Mat &srcGauss2, Mat &dst, int coeff){
    dst = Mat::zeros(srcGauss1.size(), CV_8U);
    absdiff(srcGauss1, srcGauss2, dst);
    dst*=coeff;
}

void mosaic3(const Mat &src, Mat &dst){
    dst = Mat::zeros(src.size(), CV_8U);
    float k = 9; // коэффициент нормировки
    float Fk[3][3] = { {1,1,1}, {1,1,1}, {1,1,1} }; // маска фильтра
    for (int i = 1; i < src.cols - 1; i+=3)
        for (int j = 1; j < src.rows - 1; j+=3) {
        // далее производим свертку
        float Rez = 0;
        uchar mosaiced;
        for (int ii = -1; ii <= 1; ii++)
            for (int jj = -1; jj <= 1; jj++) {
                mosaiced = src.at<uchar>(j + jj, i + ii);
                Rez += Fk[ii + 1][jj + 1] * mosaiced;
            }

        for (int ii = -1; ii <= 1; ii++)
            for (int jj = -1; jj <= 1; jj++) {
                mosaiced = Rez / k; // осуществляем нормировку
                dst.at<uchar>(j + jj, i + ii) = mosaiced;
            }
    }
}

void aperture_cor3(const Mat &src, Mat &dst){
    // выделяем память под выходное изображение
    dst = Mat::zeros(src.size(), CV_8U);
```

```
    // коэффициент нормировки
    float k = 3;
    // матрица коэффициентов фильтра
    float Fk[3][3] = { {-1,-1,-1}, {-1,11,-1}, {-1,-1,-1} };
    // перебираем все пиксели входного изображения
    for (int i = 1; i < src.cols - 1; i++)
        for (int j = 1; j < src.rows - 1; j++) {
        // вычисляем свертку пикселя входного изображения по маске фильтра
        float Rez = 0;
        for (int ii = -1; ii <= 1; ii++)
            for (int jj = -1; jj <= 1; jj++) {
                int apertured = src.at<uchar>(j + jj, i + ii);
                Rez += Fk[ii + 1][jj + 1] * apertured;
            }
        int apertured_norm = Rez / k;
        dst.at<uchar>(j, i) = saturate_cast<uchar>(apertured_norm);
    }
}

void median3(const Mat &src, Mat &dst){
    dst = Mat::zeros(src.size(), CV_8U);
    for (int i = 1; i < src.cols - 1; i++)
        for (int j = 1; j < src.rows - 1; j++) {
            vector<uchar> vec_median;
            for (int ii = -1; ii <= 1; ii++)
                for (int jj = -1; jj <= 1; jj++)
                    vec_median.push_back(src.at<uchar>(j + jj, i + ii));
            sort(vec_median.begin(), vec_median.end());
            dst.at<uchar>(j, i) = vec_median.at(4);
        }
}

void sobel_vert(const Mat &src, Mat &dst){
    dst = Mat::zeros(src.size(), CV_8U);
    int Fk[3][3] = { {-1,0,1}, {-2,0,2}, {-1,0,1} }; // маска фильтра
    for (int i = 1; i < src.cols - 1; i++)
        for (int j = 1; j < src.rows - 1; j++) {
            float dst_pix = 0;
            for (int ii = -1; ii <= 1; ii++)
                for (int jj = -1; jj <= 1; jj++)
                    dst_pix += Fk[ii + 1][jj + 1] * src.at<uchar>(j + jj, i + ii);
            dst.at<uchar>(j, i) = saturate_cast<uchar>(abs(dst_pix));
    }
}

void sobel_hor(const Mat &src, Mat &dst){
    dst = Mat::zeros(src.size(), CV_8U);
    int Fk[3][3] = { {-1,-2,-1}, {0,0,0}, {1,2,1} }; // маска фильтра
    for (int i = 1; i < src.cols - 1; i++)
        for (int j = 1; j < src.rows - 1; j++) {
            float dst_pix = 0;
            for (int ii = -1; ii <= 1; ii++)
                for (int jj = -1; jj <= 1; jj++)
                    dst_pix += Fk[ii + 1][jj + 1] * src.at<uchar>(j + jj, i + ii);
            dst.at<uchar>(j, i) = saturate_cast<uchar>(abs(dst_pix));
    }
}

void scharr_hor(const Mat &src, Mat &dst){
    dst = Mat::zeros(src.size(), CV_8U);
    int Fk[3][3] = { {3,10,3}, {0,0,0}, {-3,-10,-3} }; // маска фильтра
    for (int i = 1; i < src.cols - 1; i++)
```

```
        for (int j = 1; j < src.rows - 1; j++) {
            float dst_pix = 0;
            for (int ii = -1; ii <= 1; ii++)
                for (int jj = -1; jj <= 1; jj++)
                    dst_pix += Fk[ii + 1][jj + 1] * src.at<uchar>(j + jj, i + ii);
            dst.at<uchar>(j, i) = saturate_cast<uchar>(abs(dst_pix));
    }
}

void scharr_vert(const Mat &src, Mat &dst){
    dst = Mat::zeros(src.size(), CV_8U);
    int Fk[3][3] = { {3,0,-3}, {10,0,-10}, {3,0,-3} }; // маска фильтра
    for (int i = 1; i < src.cols - 1; i++)
        for (int j = 1; j < src.rows - 1; j++) {
            float dst_pix = 0;
            for (int ii = -1; ii <= 1; ii++)
                for (int jj = -1; jj <= 1; jj++)
                    dst_pix += Fk[ii + 1][jj + 1] * src.at<uchar>(j + jj, i + ii);
            dst.at<uchar>(j, i) = saturate_cast<uchar>(abs(dst_pix));
    }
}

void prewitt_vert(const Mat &src, Mat &dst){
    dst = Mat::zeros(src.size(), CV_8U);
    int Fk[3][3] = { {1,0,-1}, {1,0,-1}, {1,0,-1} }; // маска фильтра
    for (int i = 1; i < src.cols - 1; i++)
        for (int j = 1; j < src.rows - 1; j++) {
            float dst_pix = 0;
            for (int ii = -1; ii <= 1; ii++)
                for (int jj = -1; jj <= 1; jj++)
                    dst_pix += Fk[ii + 1][jj + 1] * src.at<uchar>(j + jj, i + ii);
            dst.at<uchar>(j, i) = saturate_cast<uchar>(abs(dst_pix));
    }
}

void prewitt_hor(const Mat &src, Mat &dst){
    dst = Mat::zeros(src.size(), CV_8U);
    int Fk[3][3] = { {1,1,1}, {0,0,0}, {-1,-1,-1} }; // маска фильтра
    for (int i = 1; i < src.cols - 1; i++)
        for (int j = 1; j < src.rows - 1; j++) {
            float dst_pix = 0;
            for (int ii = -1; ii <= 1; ii++)
                for (int jj = -1; jj <= 1; jj++)
                    dst_pix += Fk[ii + 1][jj + 1] * src.at<uchar>(j + jj, i + ii);
            dst.at<uchar>(j, i) = saturate_cast<uchar>(abs(dst_pix));
    }
}

void lab3(const Mat &img_bgr){
    Image img;
    img.bgr = img_bgr;
    resize(img.bgr, img.bgr, Size(300, 300), 0, 0, INTER_CUBIC);
    imshow("image bgr", img.bgr);

    cvtColor(img.bgr, img.gray, COLOR_BGR2GRAY);
    imshow("image gray", img.gray);
    imwrite("../../Images/Lab 3/image gray.jpg", img.gray);

    gauss3(img.gray, img.gauss3);
    // imshow("gauss 3", img.gauss3);
    imwrite("../../Images/Lab 3/gauss 3.jpg", img.gauss3);
```

```
    gauss5(img.gray, img.gauss5);
    // imshow("gauss 5", img.gauss5);
    imwrite("../../Images/Lab 3/gauss 5.jpg", img.gauss5);

    mosaic3(img.gray, img.mosaic3);
    // imshow("mosaic 3", img.mosaic3);
    imwrite("../../Images/Lab 3/mosaic 3.jpg", img.mosaic3);

    aperture_cor3(img.gray, img.aperture_cor_3);
    // imshow("aperture_cor 3", img.aperture_cor_3);
    imwrite("../../Images/Lab 3/aperture_cor 3.jpg", img.aperture_cor_3);

    median3(img.gray, img.median3);
    // imshow("median 3", img.median3);
    imwrite("../../Images/Lab 3/median 3.jpg", img.median3);

    dog(img.gauss3, img.gauss5, img.dog, 7);
    // imshow("dog", img.dog);
    imwrite("../../Images/Lab 3/dog.jpg", img.dog);

    Canny(img.gray, img.canny, 100, 200);
    // imshow("canny", img.canny);
    imwrite("../../Images/Lab 3/canny.jpg", img.canny);

    sobel_vert(img.gray, img.sobel_vert);
    imshow("sobel vertical", img.sobel_vert);
    imwrite("../../Images/Lab 3/sobel vertical.jpg", img.sobel_vert);

    sobel_hor(img.gray, img.sobel_hor);
    imshow("sobel horizontal", img.sobel_hor);
    imwrite("../../Images/Lab 3/sobel horizontal.jpg", img.sobel_hor);

    scharr_vert(img.gray, img.scharr_vert);
    imshow("scharr vertical", img.scharr_vert);
    imwrite("../../Images/Lab 3/scharr vertical.jpg", img.scharr_vert);

    scharr_hor(img.gray, img.scharr_hor);
    imshow("scharr horizontal", img.scharr_hor);
    imwrite("../../Images/Lab 3/scharr horizontal.jpg", img.scharr_hor);

    prewitt_vert(img.gray, img.prewitt_vert);
    imshow("prewitt vertical", img.prewitt_vert);
    imwrite("../../Images/Lab 3/prewitt vertical.jpg", img.prewitt_vert);

    prewitt_hor(img.gray, img.prewitt_hor);
    imshow("prewitt horizontal", img.prewitt_hor);
    imwrite("../../Images/Lab 3/prewitt horizontal.jpg", img.prewitt_hor);

    waitKey();
}
```

1. Исходное изображение

Рис 1. Исходное полутоновое изображение.

2. Результаты фильтрации изображения 1-й группой фильтров

Рис. 2. Фильтр Гаусса с апертурой 3x3.

Рис. 3. Фильтр Гаусса с апертурой 5x5.

Рис. 4. Фильтр «Мозаика».

Рис. 5. Фильтр апертурной коррекции.

Рис. 6. Медианный фильтр.

3. Результаты фильтрации изображения 2-й группой фильтров

Рис. 7. Разность гауссиан.

Рис. 8. Фильтр Канни.

Рис. 9. Оператор Щарра для горизонтальных контуров.

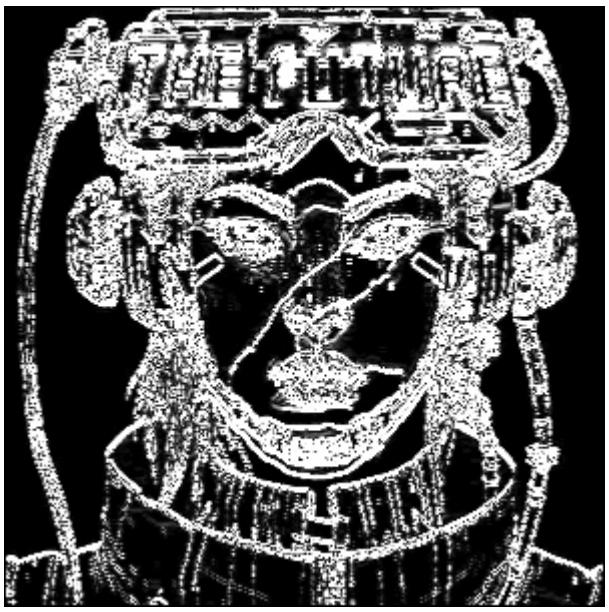

Рис. 10. Оператор Щарра для вертикальных контуров.

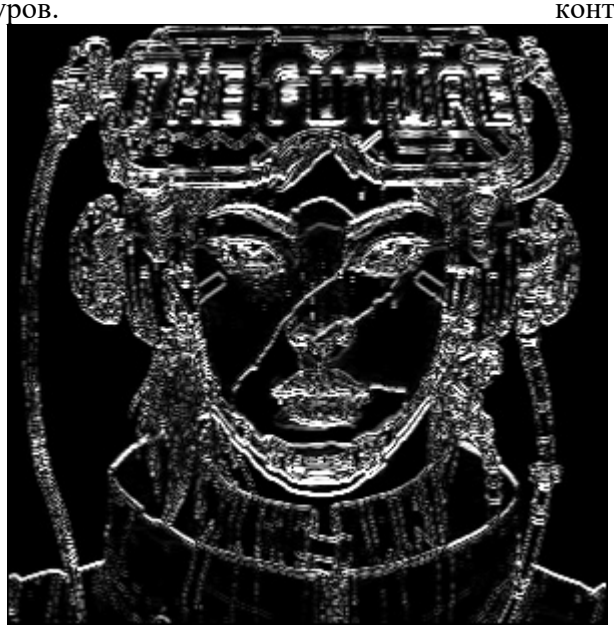

Рис. 11. Оператор Собела для вертикальных контуров.

Выводы: модификация значения пиксела с учётом его соседей (фильтрация) – весьма интересное явление! В зависимости от весов соседей фильтрация позволяет сгладить, увеличить резкость или выделить контура изображения. При сглаживании для фокуса (центрального элемента маски) вычисляют взвешенное среднее значение соседей. При увеличении резкости происходит усиление скачков яркости на границах объекта. А при выделении контуров и вовсе приходится брать производную с конечными приращениями (разностную производную) в зависимости от типа контура (горизонтальный, вертикальный, наклонный).