**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

**отчЁт**

**по лабораторной работе № 4**

**по дисциплине «Цифровая обработка изображений»**

**Тема: РАЗРАБОТКА ПРОГРАММЫ ДЛЯ**

**МОРФОЛОГИЧЕСКОЙ ФИЛЬТРАЦИИ ИЗОБРАЖЕНИЙ**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9105 |  | Шаривзянов Д. Р. |
|  |  | Басманов А. А. |
| Преподаватель |  | Поздеев А. А. |

Санкт-Петербург

2024

**РАЗРАБОТКА ПРОГРАММЫ ДЛЯ**

**МОРФОЛОГИЧЕСКОЙ ФИЛЬТРАЦИИ ИЗОБРАЖЕНИЙ**

**Цели работы.**

Целью лабораторной работы является знакомство с методами морфологической фильтрации изображений. В соответствии с заданием на моделирование необходимо разработать программу в виде консольного приложения, реализующую следующие функции:

1. Загрузку растрового изображения, преобразование его в однотонное и

вывод его на экран.

2. Преобразование изображения в бинарное и вывод его на экран.

3. Фильтрацию бинарного изображения морфологическими фильтрами.

4. Фильтрацию полутонового изображения морфологическими фильтрами.

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;

Mat binary;

// morphology

Mat erosion;

Mat dilation;

Mat close;

Mat open;

// extra

Mat contour;

Mat MG;

};

void erosion(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++) {

float min = 255;

for (int ii = -1; ii <= 1; ii++)

for (int jj = -1; jj <= 1; jj++) {

uchar Y = src.at<uchar>(j + jj, i + ii);

if (Y < min) min = Y;

}

dst.at<uchar>(j, i) = min;

}

}

void dilation(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++) {

uchar max = 0;

for (int ii = -1; ii <= 1; ii++)

for (int jj = -1; jj <= 1; jj++) {

uchar Y = src.at<uchar>(j + jj, i + ii);

if (Y > max) max = Y;

}

dst.at<uchar>(j, i) = max;

}

}

void closing(const Mat &src, Mat &dst)

{

Mat temp = Mat::zeros(src.size(), CV\_8U);

dilation(src, temp);

erosion(temp, dst);

}

void opening(const Mat &src, Mat &dst)

{

Mat temp = Mat::zeros(src.size(), CV\_8U);

erosion(src, temp);

dilation(temp, dst);

}

void contour(const cv::Mat& src, Mat &dst, int thick = 1) {

int scale = 0;

cv::Mat res\_1 = cv::Mat::zeros(src.size(), CV\_8U);

cv::Mat prep1 = src.clone();

cv::Mat output\_img = src.clone();

for (int scale = 1; scale <= thick; scale++) {

for (int i = 1; i < prep1.rows - 1; i++)

for (int j = 1; j < prep1.cols - 1; j++) {

float min = 255;

uchar pix\_value = prep1.at<uchar>(i, j);

for (int ii = -1; ii <= 1; ii++)

for (int jj = -1; jj <= 1; jj++) {

uchar Y = prep1.at<uchar>(i - ii, j - jj);

if (Y < min) min = Y;

}

res\_1.at<uchar>(i, j) = min;

}

prep1 = res\_1.clone();

}

scale = 0;

cv::Mat res\_2 = cv::Mat::zeros(src.size(), CV\_8U);

cv::Mat prep2 = src.clone();

for (int scale = 1; scale <= thick; scale++) {

for (int i = 1; i < prep2.rows - 1; i++)

for (int j = 1; j < prep2.cols - 1; j++) {

float max = 0;

uchar pix\_value = prep2.at<uchar>(i, j);

for (int ii = -1; ii <= 1; ii++)

for (int jj = -1; jj <= 1; jj++) {

uchar Y = prep2.at<uchar>(i - ii, j - jj);

if (Y > max) max = Y;

}

res\_2.at<uchar>(i, j) = max;

}

prep2 = res\_2.clone();

}

cv::subtract(prep2, prep1, output\_img);

dst = output\_img;

}

void MG(const Mat &src, Mat &dst)

{

Mat temp\_img = Mat::zeros(src.size(), CV\_8U);

Mat dilated\_img = Mat::zeros(src.size(), CV\_8U);

Mat eroded\_img = Mat::zeros(src.size(), CV\_8U);

Mat gradient\_img = Mat::zeros(src.size(), CV\_8U);

Mat eroded\_gradient\_img = Mat::zeros(src.size(), CV\_8U);

Mat sum\_img = Mat::zeros(src.size(), CV\_8U);

for (int scale = 1; scale <= 3; scale++) {

for (int i = scale; i < src.cols - scale; i++)

for (int j = scale; j < src.rows - scale; j++) {

float max = 0;

float min = 255;

for (int ii = -scale; ii <= scale; ii++)

for (int jj = -scale; jj <= scale; jj++) {

uchar Y = src.at<uchar>(j + jj, i + ii);

if (Y > max) max = Y;

if (Y < min) min = Y;

}

dilated\_img.at<uchar>(j, i) = max;

eroded\_img.at<uchar>(j, i) = min;

}

gradient\_img = dilated\_img - eroded\_img;

for (int i = scale - 1; i < gradient\_img.cols - scale + 1; i++)

for (int j = scale - 1; j < gradient\_img.rows - scale + 1; j++) {

float min = 255;

for (int ii = -scale + 1; ii <= scale - 1; ii++)

for (int jj = -scale + 1; jj <= scale - 1; jj++) {

uchar Y = gradient\_img.at<uchar>(j + jj, i + ii);

if (Y < min) min = Y;

}

eroded\_gradient\_img.at<uchar>(j, i) = min;

}

sum\_img = sum\_img + eroded\_gradient\_img;

}

dst = sum\_img / 3;

}

void lab4(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 4/image gray.jpg", img.gray);

threshold(img.gray, img.binary, 128, 255, THRESH\_BINARY);

imshow("image binary", img.binary);

imwrite("../../Images/Lab 4/image binary.jpg", img.binary);

erosion(img.binary, img.erosion);

// imshow("bin\_erosion", img.erosion);

imwrite("../../Images/Lab 4/bin\_erosion.jpg", img.erosion);

dilation(img.binary, img.dilation);

// imshow("bin\_dilation", img.dilation);

imwrite("../../Images/Lab 4/bin\_dilation.jpg", img.dilation);

closing(img.binary, img.close);

// imshow("bin\_close", img.close);

imwrite("../../Images/Lab 4/bin\_close.jpg", img.close);

opening(img.binary, img.open);

// imshow("bin\_open", img.open);

imwrite("../../Images/Lab 4/bin\_open.jpg", img.open);

erosion(img.gray, img.erosion);

// imshow("gs\_erosion", img.erosion);

imwrite("../../Images/Lab 4/gs\_erosion.jpg", img.erosion);

dilation(img.gray, img.dilation);

// imshow("gs\_dilation", img.dilation);

imwrite("../../Images/Lab 4/gs\_dilation.jpg", img.dilation);

closing(img.gray, img.close);

imshow("gs\_close", img.close);

imwrite("../../Images/Lab 4/gs\_close.jpg", img.close);

opening(img.gray, img.open);

// imshow("gs\_open", img.open);

imwrite("../../Images/Lab 4/gs\_open.jpg", img.open);

contour(img.gray, img.contour);

imshow("contour", img.contour);

imwrite("../../Images/Lab 4/contour.jpg", img.contour);

MG(img.gray, img.MG);

// bitwise\_not(img.MG, img.MG);

imshow("MG", img.MG);

imwrite("../../Images/Lab 4/MG.jpg", img.MG);

waitKey();

}

1. Исходные изображения

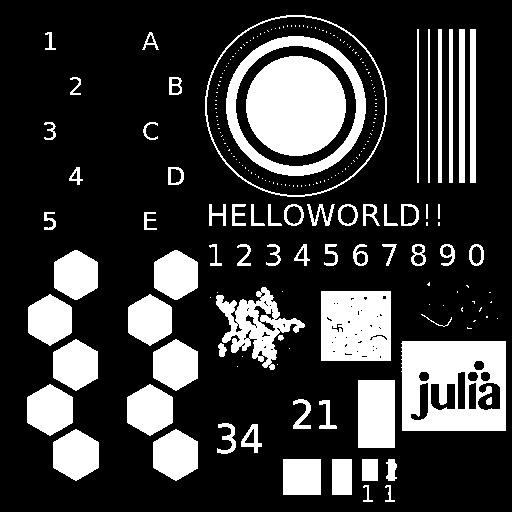


Рис 1. Исходное бинарное изображение.

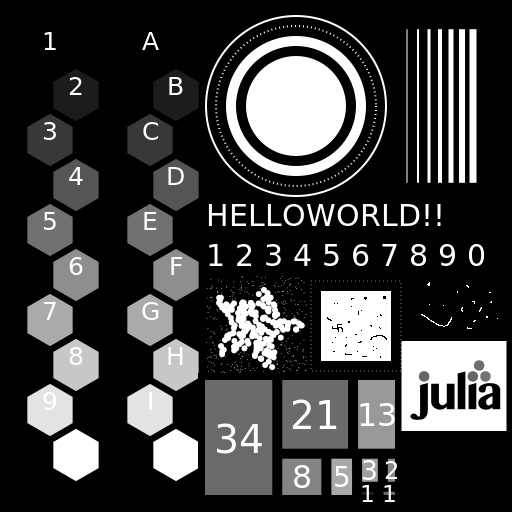


Рис 2. Исходное полутоновое изображение.

1. Результаты морфологической фильтрации бинарных изображений

|  |  |
| --- | --- |
|  |  |
| Рис. 3. Фильтр бинарной эрозии. | Рис. 4. Фильтр бинарной дилатации. |

|  |  |
| --- | --- |
|  |  |
| Рис. 5. Оператор открытия. | Рис. 6. Оператор закрытия. |

1. Результаты морфологической фильтрации полутоновых изображений

|  |  |
| --- | --- |
|  |  |
| Рис. 7. Фильтр полутоновой эрозии. | Рис. 8. Фильтр полутоновой дилатации. |
|  |  |
| Рис. 9. Оператор открытия. | Рис. 10. Оператор закрытия. |
|  |  |
|  |  |
| Рис. 9. Оператор выделения контуров. | Рис. 10. Многомасштабный морфологический градиент. |

**Выводы.**

1) Бинарную и полутоновую обработку можно осуществить одними и теми же фильтрами.

2) Эрозия удаляет шумы типа «соль» и уменьшает объекты.

3) Дилатация удаляет шумы типа «перец» и уменьшает объекты.

4) Открытие и закрытие удаляют эти же шумы, но без уменьшения.

5) Многомасштабный морфологический градиент тоже выделят контуры, но делает это не так резко, как оператор выделения контуров.

Примечание: шум – это элемент, который меньше структурного элемента; объект – это элемент, который больше или равен структурному элементу.