**Assignment 2**

**Part 1 source code**

(PS:read dilation\_erosion.cpp for more)

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>

#include <fstream>

using namespace cv;

using namespace std;

const int MIN\_BOUNDARY = 90;

//for debug

int succTimes = 0;

int failTimes = 0;

bool getSize(int &size)

{

if (cin >> size)

{

if (size >= 1 && size <= 10)

{

return true;

}

}

return false;

}

bool getBool(bool\* array, int height, int width, int row, int col)

{

if (row < 0 || row >= height || col < 0 || col >= width)

{

cout << "getBool error!\n";

system("pause");

exit(0);

}

return array[row\*width + col];

}

void setBool(bool\* array, int height, int width, int row, int col, bool value)

{

if (row < 0 || row >= height || col < 0 || col >= width)

{

cout << "getBool error!\n";

system("pause");

exit(0);

}

array[row\*width + col] = value;

}

void displayMatrix(bool\* matrix, int height, int width)

{

fstream logfs("log.txt", ios::out | ios::app);

logfs << "display matrix: \n";

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

{

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

{

logfs << (int)getBool(matrix, height, width, i, j) << " ";

}

logfs << endl;

}

}

bool isContain(bool\* imgMatrix, int height, int width, bool\* structure, int size, int transx, int transy)

{

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

{

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

{

if (getBool(structure, size, size, i, j) != getBool(imgMatrix, height, width, i + transx, j + transy))

{

failTimes++;

return false;

}

}

}

succTimes++;

return true;

}

bool isHit(bool\* imgMatrix, int height, int width, bool\* structure, int size, int transx, int transy)

{

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

{

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

{

if ((i + transx) >= 0 && (i + transx) < height && (j + transy) >= 0 && (j + transy) < width)

{

if (getBool(structure, size, size, i, j) == getBool(imgMatrix, height, width, i + transx, j + transy))

{

return true;

}

}

}

}

return false;

}

bool\* erosion(bool\* imgMatrix, int height, int width, bool\* structure, int size)

{

bool\* result = new bool[height\*width];

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

{

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

{

setBool(result, height, width, i, j, 0);

}

}

//the point at the boundary is impossible

for (int i = 0; i < height - size + 1; i++)

{

for (int j = 0; j < width - size + 1; j++)

{

if (isContain(imgMatrix, height, width, structure, size, i, j))

{

setBool(result, height, width, i, j, 1);

}

//default value is 0

/\*else

{

setBool(result, height, width, i, j, 0);

}\*/

}

}

return result;

}

bool\* dilation(bool\* imgMatrix, int height, int width, bool\* structure, int size)

{

bool\* result = new bool[height\*width];

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

{

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

{

setBool(result, height, width, i, j, 0);

}

}

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

{

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

{

if (isHit(imgMatrix, height, width, structure, size, i, j))

{

setBool(result, height, width, i, j, 1);

}

//default value is 0

/\*else

{

setBool(result, height, width, i, j, 0);

}\*/

}

}

return result;

}

int main()

{

//get size

int size;

cout << "We use a square as a structuring element and the left-up corner is origin.\n"

<< "Please input an integer (1 - 10) as size of the square: ";

if (getSize(size))

{

cout << "\nThe size of square is: " << size << endl;

}

else

{

cout << "\nInout is invalid!\n";

system("pause");

exit(0);

}

//initialize the structuring element

bool\* structure = new bool[size\*size];

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

{

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

{

setBool(structure, size, size, i, j, 1);

}

}

//generate the matrix of img

IplImage\* img = cvLoadImage("lena-binary.bmp", 0);

bool\* imgMatrix = new bool[img->height\*img->width];

for (int row = 0; row < img->height; row++)

{

uchar\* ptr = (uchar\*)img->imageData + row\*img->widthStep;

for (int col = 0; col < img->width; col++)

{

if (ptr[col]>MIN\_BOUNDARY)//white

{

setBool(imgMatrix, img->height, img->width, row, col, 1);//1 means have data or means contained by the foreground

}

else

{

setBool(imgMatrix, img->height, img->width, row, col, 0);

}

}

}

//displayMatrix(structure, size, size);

//displayMatrix(imgMatrix, img->height, img->width);

//choose operation

int mode = 0;

cout << "Please choose a operation(1-erosion 2-dilation): ";

cin >> mode;

cout << endl;

//get result

bool\* result = NULL;

if (mode == 1)

{

result = erosion(imgMatrix, img->height, img->width, structure, size);

}

else if (mode == 2)

{

result = dilation(imgMatrix, img->height, img->width, structure, size);

}

else

{

cout << "Operation invalid!\n";

system("pause");

exit(0);

}

//draw result image

IplImage\* resultImg = cvCloneImage(img);

for (int row = 0; row < resultImg->height; row++)

{

uchar\* ptr = (uchar\*)resultImg->imageData + row\*resultImg->widthStep;

for (int col = 0; col < resultImg->width; col++)

{

if (getBool(result, resultImg->height, resultImg->width, row, col))

{

ptr[col] = 255;//white

}

else

{

ptr[col] = 0;//black

}

}

}

//save result image

if (mode == 1)

{

cvSaveImage("result\_erosion.bmp", resultImg);

}

else

{

cvSaveImage("result\_dilation.bmp", resultImg);

}

//release space

delete structure;

delete imgMatrix;

delete result;

cvReleaseImage(&img);

cvReleaseImage(&resultImg);

system("pause");

return 0;

}

**Part 2 definition**

(PS:read comment of dilation\_erosion.cpp for more info)

Basic definition:

Const int MIN\_BOUNDARY = 90//diminish the effect of background

bool\* structure = new bool[size\*size]//square structure element

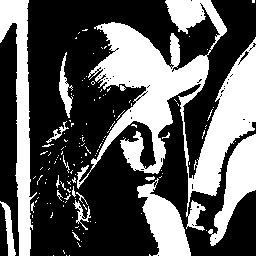
IplImage\* img = cvLoadImage("lena-binary.bmp", 0)//origin image

cvSaveImage("result\_erosion.bmp", resultImg)//erosion image

cvSaveImage("result\_dilation.bmp", resultImg)//dilation image

**Part 3 output**

1.origin image



2.dilation image



1. erosion image

