CSCI 381 (CPP)

Section 31

Project 9.1 Morphology

Essam Yousry

Due Date of soft copy: 05/08/2018

Due Date of hard copy: 05/10/2018

Algorithm Steps in main

Step 0: open all files

( numRowsImg, numColsImg, minImg, maxImg ) 🡨 get from input1

( numRowsStrctElem, numColsStrctElem, minStrctElem, maxStrctElem ) 🡨 get from input2

( rowOrigin, colOrigin) 🡨 get from input2

Step 1: computeFrameSize

Step 2: - dynamically allocate imgAry with extra rows and extra columns

- loadImage // load input file to imgAry

- zeroFrameImg ()

- prettyPrint (imgAry) // pretty print imgAry to the \*console\* of "Input Image"

- dynamically allocate morphAry with extra rows and extra columns

Step 3:

- dynamically allocate structElemAry

- loadstruct // load input2 file to structElem array

- prettyPrint (structElemAry)// pretty print to the \*console\* of "Structuring Element"

Step 4: - initMorphAry( ) // initialize morphAry to zero

- call dilation // see your lecture note

- prettyPrint (morphAry) // pretty print to the \*console\* the result of dilation

- outputResult //write the delation result to Output1 (argv[3])

Step 5: - initMorphAry( ) // initialize morphAry to zero

- call erosion // see your lecture note

- prettyPrint (morphAry) // pretty print to the \*console\* the result of erosion

- outputResult //write the delation result to Output2 (argv[4])

Step 6: - initMorphAry( ) // initialize morphAry to zero

- call closing // By call those two morphological ops one after the other

- prettyPrint (morphAry) // pretty print to the \*console\* the result of closing

- outputResult //write the closing result to Output3 (argv[5])

Step 7: - initMorphAry( ) // initialize morphAry to zero

- call opening // By call those two morphological ops one after the other

- prettyPrint (morphAry) // pretty print to the \*console\* with "Opening Result"

- outputResult //write the opening result to Output4 (argv[6])

Step 8: close all files

Source Code

#include <iostream>

#include <fstream>

using namespace std;

void initMorphAry(int \*\*morphAry, int rowSize, int colSize){

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

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

            morphAry[i][j] = 0;

        }

    }

}

void loadImage(int \*\*imgAry, int numRowsImg, int numColsImg, int numRowsStructElem, int numColsStructElem, ifstream &Input1){

    for (int i = numRowsStructElem; i < numRowsImg + numRowsStructElem; i++){

        for (int j = numColsStructElem; j < numColsImg + numColsStructElem; j++){

            Input1 >> imgAry[i][j];

        }

    }

}

void loadStruct(int \*\*structElemAry, int numRowsStructElem, int numColsStructElem, ifstream &Input2){

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

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

            Input2 >> structElemAry[i][j];

        }

    }

}

void zeroFrameImgAry(int \*\*imgAry, int rowSize, int colSize){

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

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

            imgAry[i][0] = 0;

            imgAry[rowSize - 1][j] = 0;

            imgAry[0][j] = 0;

            imgAry[i][colSize - 1] = 0;

        }

    }

}

void prettyPrintToConsole(int \*\*array, int row, int col){

    for (int i = 0; i < row; i++){

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

            if (array[i][j] > 0){

                cout << array[i][j];

                cout << ' ';

            }

            else{

                cout << ' ';

                cout << ' ';

            }

        }

        cout << endl;

    }

    cout << endl;

}

void outputResult(int \*\*array, int row, int col, ofstream &Output){

    for (int i = 0; i < row; i++){

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

            Output << array[i][j];

            Output << ' ';

        }

        Output << endl;

    }

    Output << endl;

}

void dilation(int \*\*structElemAry, int \*\*imgAry, int \*\*morphAry, int numRowsStructElem, int numColsStructElem, int numRowsImg, int numColsImg, int rowOrigin, int colOrigin){

    for (int i = numRowsStructElem; i < numRowsImg + numRowsStructElem; i++){

        for (int j = numColsStructElem; j < numColsImg + numColsStructElem; j++){

            if (imgAry[i][j] > 0){

                for (int structRow = 0; structRow < numRowsStructElem; structRow++){

                    for (int structCol = 0; structCol < numColsStructElem; structCol++){

                        if (structElemAry[structRow][structCol] == 1){

                            morphAry[i - rowOrigin + structRow][j - colOrigin + structCol] = structElemAry[structRow][structCol];

                        }

                    }

                }

            }

        }

    }

}

void erosion(int \*\*structElemAry, int \*\*imgAry, int\*\* morphAry, int numRowsStructElem, int numColsStructElem, int numRowsImg, int numColsImg, int rowOrigin, int colOrigin){

    for (int i = numRowsStructElem; i < numRowsImg + numRowsStructElem; i++){

        for (int j = numColsStructElem; j < numColsImg + numColsStructElem; j++){

            if (imgAry[i][j] > 0){

                bool check = true;

                for (int structRow = 0; structRow < numRowsStructElem; structRow++){

                    for (int structCol = 0; structCol < numColsStructElem; structCol++){

                        if ((structElemAry[structRow][structCol] == 1) && (imgAry[i - rowOrigin + structRow][j - colOrigin + structCol] == 0)) check = false;

                    }

                }

                if (check == true) morphAry[i][j] = 1;

                else morphAry[i][j] = 0;

            }

        }

    }

}

void closing(int \*\*structElemAry, int \*\*imgAry, int \*\*morphAry, int numRowsStructElem, int numColsStructElem, int numRowsImg, int numColsImg, int rowOrigin, int colOrigin){

    dilation(structElemAry, imgAry, morphAry, numRowsStructElem, numColsStructElem, numRowsImg, numColsImg, rowOrigin, colOrigin);

    erosion(structElemAry, imgAry, morphAry, numRowsStructElem, numColsStructElem, numRowsImg, numColsImg, rowOrigin, colOrigin);

}

void opening(int \*\*structElemAry, int \*\*imgAry, int \*\*morphAry, int numRowsStructElem, int numColsStructElem, int numRowsImg, int numColsImg, int rowOrigin, int colOrigin){

    erosion(structElemAry, imgAry, morphAry, numRowsStructElem, numColsStructElem, numRowsImg, numColsImg, rowOrigin, colOrigin);

    dilation(structElemAry, imgAry, morphAry, numRowsStructElem, numColsStructElem, numRowsImg, numColsImg, rowOrigin, colOrigin);

}

void computeFrameSize(int &rowFrameSize, int &colFrameSize, int numRowsStructElem, int numColsStructElem){

    rowFrameSize = (numRowsStructElem \* 2);

    colFrameSize = (numColsStructElem \* 2);

}

int main (int argc, char \*argv[])

{

    int vars[4];

    int vars2[6];

    int numRowsImg, numColsImg, minImg, maxImg;

    int numRowsStructElem, numColsStructElem, minStructElem, maxStructElem;

    int rowOrigin, colOrigin;

    int rowFrameSize, colFrameSize;

    ifstream Input1;

    Input1.open(argv[1]);

    ifstream Input2;

    Input2.open(argv[2]);

    for(int i = 0; i < 4; i++){

        Input1 >> vars[i];

    }

    numRowsImg = vars[0];

    numColsImg = vars[1];

    minImg = vars[2];

    maxImg = vars[3];

    for (int i = 0; i < 6; i++){

        Input2 >> vars2[i];

    }

    numRowsStructElem = vars2[0];

    numColsStructElem = vars2[1];

    minStructElem = vars2[2];

    maxStructElem = vars2[3];

    rowOrigin = vars2[4];

    colOrigin = vars2[5];

    ofstream Output1;

    Output1.open(argv[3]);

    ofstream Output2;

    Output2.open(argv[4]);

    ofstream Output3;

    Output3.open(argv[5]);

    ofstream Output4;

    Output4.open(argv[6]);

    computeFrameSize(rowFrameSize, colFrameSize, numRowsStructElem, numColsStructElem);

    int rowSize = numRowsImg + rowFrameSize;

    int colSize = numColsImg + colFrameSize;

    int \*\*imgAry = new int \*[rowSize];

    for (int i = 0; i < rowSize; i++)

        imgAry[i] = new int [colSize];

    loadImage(imgAry, numRowsImg, numColsImg, numRowsStructElem, numColsStructElem, Input1);

    zeroFrameImgAry(imgAry, rowSize, colSize);

    cout << endl;

    cout << "Input Image";

    cout << endl;

    prettyPrintToConsole(imgAry, rowSize, colSize);

    int \*\*morphAry = new int\*[rowSize];

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

        morphAry[i] = new int [colSize];

    }

    int \*\*structElemAry = new int\*[numRowsStructElem];

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

        structElemAry[i] = new int [numColsStructElem];

    }

    loadStruct(structElemAry, numRowsStructElem, numColsStructElem, Input2);

    cout << "Structuring Element";

    cout << endl;

    cout << endl;

    prettyPrintToConsole(structElemAry, numRowsStructElem, numColsStructElem);

    initMorphAry(morphAry, rowSize, colSize);

    cout << "Dilation Result";

    cout << endl;

    dilation(structElemAry, imgAry, morphAry, numRowsStructElem, numColsStructElem, numRowsImg, numColsImg, rowOrigin, colOrigin);

    prettyPrintToConsole(morphAry, rowSize, colSize);

    Output1 << rowSize << ' ' << colSize << ' ' << minImg << ' ' << maxImg << endl;

    outputResult(morphAry, rowSize, colSize, Output1);

    initMorphAry(morphAry, rowSize, colSize);

    cout << "Erosion Result";

    cout << endl;

    erosion(structElemAry, imgAry, morphAry, numRowsStructElem, numColsStructElem, numRowsImg, numColsImg, rowOrigin, colOrigin);

    prettyPrintToConsole(morphAry, rowSize, colSize);

    Output2 << rowSize << ' ' << colSize << ' ' << minImg << ' ' << maxImg << endl;

    outputResult(morphAry, rowSize, colSize, Output2);

    initMorphAry(morphAry, rowSize, colSize);

    cout << "Closing Result";

    cout << endl;

    closing(structElemAry, imgAry, morphAry, numRowsStructElem, numColsStructElem, numRowsImg, numColsImg, rowOrigin, colOrigin);

    prettyPrintToConsole(morphAry, rowSize, colSize);

    Output3 << rowSize << ' ' << colSize << ' ' << minImg << ' ' << maxImg << endl;

    outputResult(morphAry, rowSize, colSize, Output3);

    initMorphAry(morphAry, rowSize, colSize);

    cout << "Opening Result";

    cout << endl;

    opening(structElemAry, imgAry, morphAry, numRowsStructElem, numColsStructElem, numRowsImg, numColsImg, rowOrigin, colOrigin);

    prettyPrintToConsole(morphAry, rowSize, colSize);

    Output4 << rowSize << ' ' << colSize << ' ' << minImg << ' ' << maxImg << endl;

    outputResult(morphAry, rowSize, colSize, Output4);

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

        delete imgAry[i];

    }

    delete [] imgAry;

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

        delete morphAry[i];

    }

    delete [] morphAry;

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

        delete structElemAry[i];

    }

    delete [] structElemAry;

    Input1.close();

    Input2.close();

    Output1.close();

    Output2.close();

    Output3.close();

    Output4.close();

    return 0;

}

Output















