Guangqing Yuan

Computer Vision

Section:CSCI381-224

Project 3

Morphological operations

Due date: 3/11/24

step 0: imgFile, structFile, dilateOutFile, erodeOutFile, openingOutFile, closingOutFile, prettyPrintFile open step 1: numImgRows, numImgCols, imgMin, imgMax read from imgFile numStructRows, numStructCols, structMin, structMax read from structFile rowOrigin, colOrigin read from strucFile

step 2: zeroFramedAry, structAry, morphAry, tempAry dynamically allocate // see description in the above

step 3: zero2DAry(zeroFramedAry, rowSize, colSize) // see description in the above

step 4: loadImg (imgFile, zeroFramedAry) // see description in the above prettyPrint (zeroFramedAry, prettyPrintFile) // with captions, say what your are printing.

step 5: zero2DAry(structAry, numStructRows, numStructCols) loadstruct (structFile, structAry) // see description in the above prettyPrint (structAry, prettyPrintFile) // with captions.

step 6: zero2DAry(morphAry, rowSize, colSize) ComputeDilation (zeroFramedAry, morphAry) // see algorithm below AryToFile (morphAry, dilateOutFile) // see description in the above prettyPrint (morphAry, prettyPrintFile) // with captions.

step 7: zero2DAry(morphAry, rowSize, colSize) ComputeErosion (zeroFramedAry, morphAry) // see algorithm below AryToFile (morphAry, erodeOutFile) prettyPrint (morphAry, prettyPrintFile) //with captions.

step 8: zero2DAry(morphAry, rowSize, colSize) ComputeOpening (zeroFramedAry, morphAry, tempAry) // see algorithm below AryToFile (morphAry, openingOutFile) prettyPrint (morphAry, prettyPrintFile) //with captions.

step 9: zero2DAry(morphAry, rowSize, colSize) ComputeClosing (zeroFramedAry, morphAry, tempAry) // see algorithm below AryToFile (morphAry, closingOutFile) prettyPrint (morphAry, prettyPrintFile) //with captions.

step 10: close all files

```
Source Code.
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;
class Morphology {
public:
  int numImgRows, numImgCols, imgMin, imgMax;
  int numStructRows, numStructCols, structMin, structMax, rowOrigin, colOrigin;
  vector<vector<int>>> zeroFramedAry, morphAry, tempAry, structAry;
  Morphology(string imgFilePath, string structFilePath) {
    // Load image and structuring element from file
    loadImg(imgFilePath);
    loadStruct(structFilePath);
  }
  //initialize to 0.
  void initArrays(int rows, int cols) {
    zeroFramedAry.resize(rows, vector<int>(cols, 0));
    morphAry.resize(rows, vector<int>(cols, 0));
    tempAry.resize(rows, vector<int>(cols, 0));
  }
  void zero2DAry(vector<vector<int>>& Ary, int nRows, int nCols) {
    for (int i = 0; i < nRows; ++i) {
      for (int j = 0; j < nCols; ++j) {
         Ary[i][j] = 0;
      }
    }
  }
  /// load imgFile to zeroFramedAry inside of frame, begins at (rowOrigin, colOrigin)
  void loadImg(string filePath) {
    ifstream inFile(filePath);
    inFile >> numImgRows >> numImgCols >> imgMin >> imgMax;
    initArrays(numImgRows + 2, numImgCols + 2);
    for (int i = 1; i \le numImgRows; i++) {
      for (int j = 1; j \le numImgCols; j++) {
         inFile >> zeroFramedAry[i][j];
      }
```

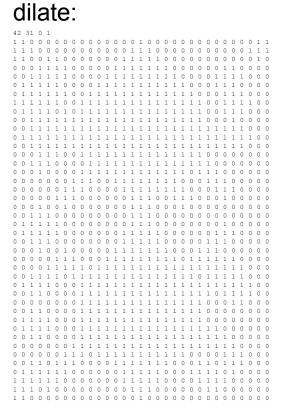
```
}
    inFile.close();
  }
  //load structFile to structAry
  void loadStruct(string filePath) {
    ifstream inFile(filePath);
    inFile >> numStructRows >> numStructCols >> structMin >> structMax >> rowOrigin >>
colOrigin;
    structAry.resize(numStructRows, vector<int>(numStructCols, 0));
    for (int i = 0; i < numStructRows; i++) {
      for (int j = 0; j < numStructCols; j++) {
         inFile >> structAry[i][j];
      }
    inFile.close();
  }
  void ComputeDilation(vector<vector<int>>& inAry, vector<vector<int>>& outAry) {
    for (int i = 1; i \le numImgRows; i++) {
      for (int j = 1; j \le numImgCols; j++) {
         if (inAry[i][j] == 1) onePixelDilation(i, j, inAry, outAry);
      }
    }
  }
  void ComputeErosion(vector<vector<int>>& inAry, vector<vector<int>>& outAry) {
    for (int i = 1; i \le numImgRows; i++) {
      for (int j = 1; j \le numImgCols; j++) {
         onePixelErosion(i, j, inAry, outAry);
      }
    }
  }
  void ComputeOpening(vector<vector<int>>& inAry, vector<vector<int>>& outAry,
vector<vector<int>>& tmp) {
    ComputeErosion(inAry, tmp);
    ComputeDilation(tmp, outAry);
  }
  void ComputeClosing(vector<vector<int>>& inAry, vector<vector<int>>& outAry,
vector<vector<int>>& tmp) {
    ComputeDilation(inAry, tmp);
```

```
ComputeErosion(tmp, outAry);
  }
  void onePixelDilation(int i, int j, vector<vector<int>>& inAry, vector<vector<int>>& outAry)
{
    for (int row = 0; row < numStructRows; row++) {</pre>
      for (int col = 0; col < numStructCols; col++) {</pre>
         if (structAry[row][col] == 1) {
           outAry[i + row - rowOrigin][j + col - colOrigin] = 1;
         }
      }
    }
  }
  void onePixelErosion(int i, int j, vector<vector<int>>& inAry, vector<vector<int>>& outAry)
{
    bool match = true;
    for (int row = 0; row < numStructRows && match; row++) {
      for (int col = 0; col < numStructCols; col++) {</pre>
         if (structAry[row][col] == 1 && inAry[i + row - rowOrigin][j + col - colOrigin] == 0) {
           match = false;
           break;
         }
      }
    if (match) outAry[i][j] = 1;
  void AryToFile(string filePath, vector<vector<int>>& ary) {
    ofstream outFile(filePath);
    // Output the image header (numImgRows, numImgCols, imgMin, imgMax) to outFile
    outFile << numImgRows << " " << numImgCols << " " << imgMin << " " << imgMax <<
endl;
    // Output the ary to outFile excluding the framed borders
    for (int i = 1; i \le numImgRows; i++) {
      for (int j = 1; j \le numImgCols; j++) {
         outFile << ary[i][j] << " ";
       outFile << endl;
    outFile.close();
  }
  void prettyPrint(vector<vector<int>>& ary, ofstream& outFile) {
    outFile << "Original Image:" << endl;
```

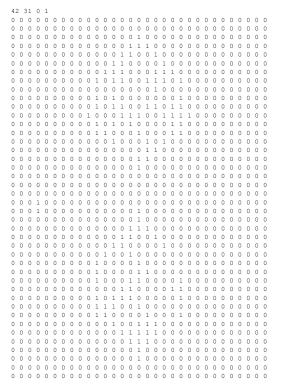
```
outFile << numImgRows << " " << numImgCols << " " << imgMin << " " << imgMax <<
endl;
    // if Ary [i, j] == 0 output ". " // a period follows by a blank
    // else output "1 " // 1 follows by a blank
    for (int i = 1; i \le numImgRows; i++) {
      for (int j = 1; j \le numImgCols; j++) {
         if (ary[i][j] == 0) {
           outFile << ". ";
         }
         else {
           outFile << "1 ";
         }
      }
      outFile << endl;
    }
    outFile << endl;
  }
};
int main(int argc, char* argv[]) {
  if (argc != 8) {
    cout << "Usage: " << argv[0] << " <imgFile> <structFile> <dilateOutFile> <erodeOutFile>
<openingOutFile> <closingOutFile> rettyPrintFile>" << endl;</pre>
    return 1;
  ofstream prettyPrintFile(argv[7]);
  Morphology morphology(argv[1], argv[2]);
  // Preparing arrays for operations
  vector<vector<int>> tempArray(morphology.numImgRows + 2,
vector<int>(morphology.numImgCols + 2, 0));
  vector<vector<int>> dilateArray(morphology.numImgRows + 2,
vector<int>(morphology.numImgCols + 2, 0));
  vector<vector<int>> erodeArray(morphology.numImgRows + 2,
vector<int>(morphology.numImgCols + 2, 0));
  vector<vector<int>> openingArray(morphology.numImgRows + 2,
vector<int>(morphology.numImgCols + 2, 0));
  vector<vector<int>> closingArray(morphology.numImgRows + 2,
vector<int>(morphology.numImgCols + 2, 0));
  morphology. zero 2D Ary (morphology. zero Framed Ary, morphology. num Img Rows, \\
morphology.numImgCols);
```

```
morphology.loadImg(argv[1]);
  morphology.prettyPrint(morphology.zeroFramedAry, prettyPrintFile);
  // Dilation
  morphology.zero2DAry(dilateArray, morphology.numImgRows, morphology.numImgCols);
// Reset dilateArray
  morphology.ComputeDilation(morphology.zeroFramedAry, dilateArray);
  morphology.AryToFile(argv[3], dilateArray);
  // Erosion
  morphology.zero2DAry(erodeArray, morphology.numImgRows, morphology.numImgCols);
// Reset erodeArray
  morphology.ComputeErosion(morphology.zeroFramedAry, erodeArray);
  morphology.AryToFile(argv[4], erodeArray);
  // Opening
  morphology.zero2DAry(openingArray, morphology.numImgRows,
morphology.numImgCols); // Reset openingArray
  morphology.ComputeOpening(morphology.zeroFramedAry, openingArray, tempArray);
  morphology.AryToFile(argv[5], openingArray);
  // Closing
  morphology.zero2DAry(closingArray, morphology.numImgRows, morphology.numImgCols);
// Reset closingArray
  morphology.ComputeClosing(morphology.zeroFramedAry, closingArray, tempArray);
  morphology.AryToFile(argv[6], closingArray);
  return 0;
}
```

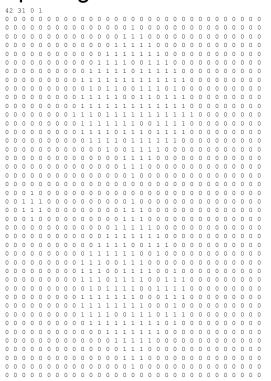
1st run



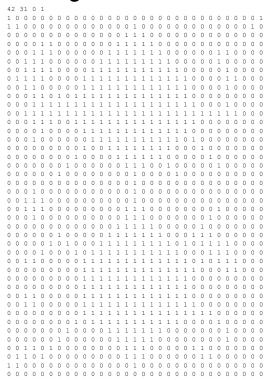
erodeOutFile



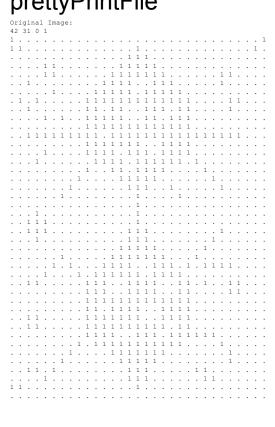
openingOutFile



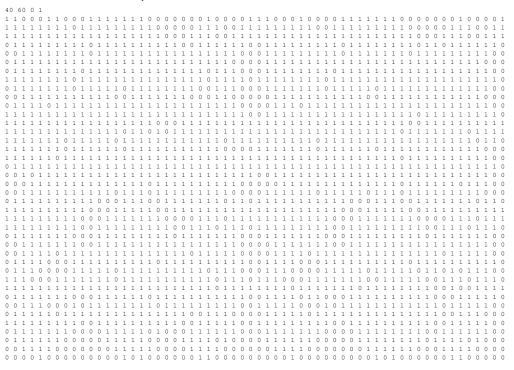
closingOutFIle



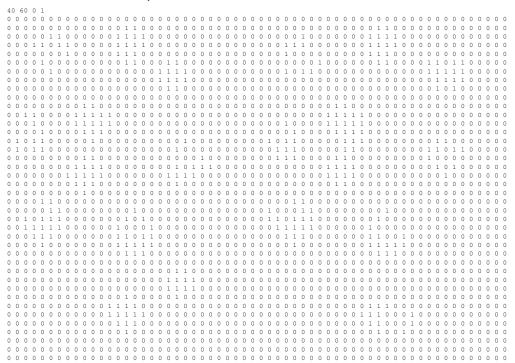
prettyPrintFile



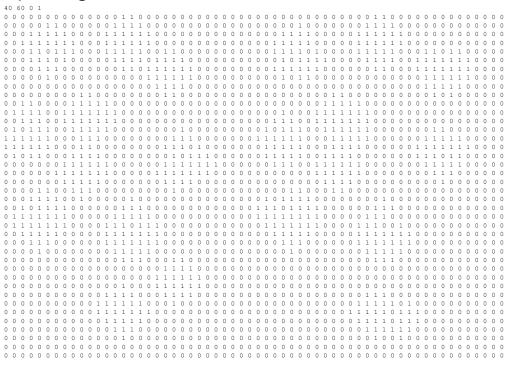
2nd run dilateOutFile,



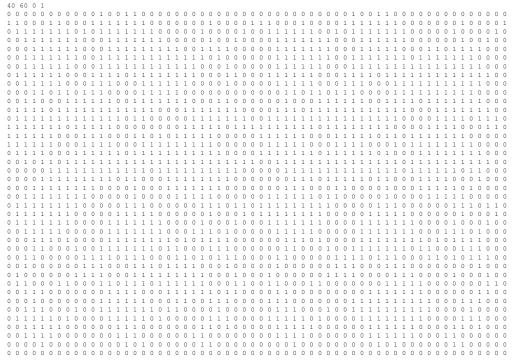
erodeOutFile,



openingOutFile,



closingOutFile,



prettyPrintFile

3rd run