Computer Vision Programming Language: C++

Project #3 3x3 Filters

Andres Quintero

Due Date:

Soft copy: 2/18/2020

Hard copy: 2/20/2020

Submission:

Soft copy: 2/18/2020

Hard copy: 2/18/2020

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Main():

step 0: inFile 🡨 open

maskFile, AvgOutImg, MedianOutImg, GaussOutImg, DebugFile 🡨 open

step 1: numRows, numCols, minVal, maxVal 🡨 read from inFile

maskRows, maskCols, maskMin, maskMax 🡨 read from maskFile

step 2: dynamically allocate all 1-D and 2-D arrays

step 3: loadMask (maskAry, mask)

step 4: loadImage (mirrorFramedAry, inFile)

step 5: mirrorFraming (mirrorFramedAry)

step 6: ComputeAvgImg (avgAry, newMin, newMax)

step 7: outputAryToFile(avgAry, AvgOutImg, newMin, newMax)

step 8: computeMedianImg (medianAry, newMin, newMax)

step 9: outputAryToFile(medianAry, MedianOutImg, newMin, newMax)

step 10: computeGaussImg (GaussAry, newMin, newMax)

step 11: outputAryToFile(GaussAry, GaussOutImg, newMin, newMax)

step 12: close all files

**Source code: main.cpp**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

// Prototypes

void loadMask(int\*\* maskAry, fstream& maskFile, int maskRows, int maskCols);

void debugPrint(int\*\* array, int rows, int cols, fstream& outFile); // DEBUGSTUFF

void loadImage(int\*\* mirrorFramedAry, fstream& inFile, int rows, int cols);

void mirrorFraming(int\*\* mirrorFramedAry, int rows, int cols);

void ComputeAvgImage(int\*\* avgAry, int\*\* mirrorFramedAry, int& newMin, int& newMax, int numRows, int numCols);

int avg3x3(int i, int j, int\*\* avgAry);

void outputAryToFile(int\*\* array, fstream&outFile1, int numRows, int numCols, int& newMin, int& newMax);

void ComputeMedianImage(int\*\* medianAry, int\*\* mirrorFramedAry, int& newMin, int& newMax, int numRows, int numCols);

void loadNeighbors(int i, int j, int\* neighborAry, int\*\* mirrorFramedAry);

void sort(int\* ary);

void ComputeGaussImage(int\*\* gaussAry, int\*\* mirrorFramedAry, int\*\* maskAry, int& newMin, int& newMax, int numRows, int numCols, int maskRows, int maskCols);

int convolution(int i, int j, int\*\* mirrorFramedAry, int\*\* maskAry, int maskRows, int maskCols);

// Main

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

// Varibles

int numRows, numCols, minVal, maxVal;

int newMin, newMax;

int maskRows, maskCols, maskMin, maskMax;

// Opening Files

fstream inFile(argv[1]); // Input Image File

fstream maskFile(argv[2]); // Input Mask File

fstream AvgOutImg(argv[3], fstream::out); // AvgOutImg

fstream MedianOutImg(argv[4], fstream::out); // MedianOutImg

fstream GaussOutImg(argv[5], fstream::out); // GaussOutImg

// Reading image file header and loading values

inFile >> numRows;

inFile >> numCols;

inFile >> minVal;

inFile >> maxVal;

//Reading mask file header and loading values

maskFile >> maskRows;

maskFile >> maskCols;

maskFile >> maskMin;

maskFile >> maskMax;

// Allocate arrays

int \*\*mirrorFramedAry, \*\*avgAry, \*\*medianAry, \*\*gaussAry, \*\*maskAry;

// mirrorFramedAry

mirrorFramedAry = new int\*[numRows + 2];

for(int i = 0; i < numRows+2; i++){

mirrorFramedAry[i] = new int[numCols + 2];

}

// avgAry

avgAry = new int\*[numRows + 2];

for(int i = 0; i < numRows+2; i++){

avgAry[i] = new int[numCols + 2];

}

// medianAry

medianAry = new int\*[numRows + 2];

for(int i = 0; i < numRows+2; i++){

medianAry[i] = new int[numCols + 2];

}

// gaussAry

gaussAry = new int\*[numRows + 2];

for(int i = 0; i < numRows+2; i++){

gaussAry[i] = new int[numCols + 2];

}

// maskAry

maskAry = new int\*[numRows];

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

maskAry[i] = new int[numCols];

}

loadMask(maskAry, maskFile, maskRows, maskCols);

loadImage(mirrorFramedAry, inFile, numRows, numCols);

mirrorFraming(mirrorFramedAry, numRows, numCols);

ComputeAvgImage(avgAry, mirrorFramedAry, newMin, newMax, numRows, numCols);

outputAryToFile(avgAry, AvgOutImg, numRows, numCols, newMin, newMax);

ComputeMedianImage(medianAry, mirrorFramedAry, newMin, newMax, numRows, numCols);

outputAryToFile(medianAry, MedianOutImg, numRows, numCols, newMin, newMax);

ComputeGaussImage(gaussAry, mirrorFramedAry, maskAry, newMin, newMax, numRows, numCols, maskRows, maskCols);

outputAryToFile(gaussAry, GaussOutImg, numRows, numCols, newMin, newMax);

// Closing Files

inFile.close();

maskFile.close();

AvgOutImg.close();

MedianOutImg.close();

GaussOutImg.close();

}

// Functions

int convolution(int i, int j, int\*\* mirrorFramedAry, int\*\* maskAry, int maskRows, int maskCols){

int sum = 0;

int pixelSum = 0;

int\* maskNeighbor = new int[9];

int\* dataNeighbor = new int[9];

loadNeighbors(1, 1, maskNeighbor, maskAry);

loadNeighbors(i, j, dataNeighbor, mirrorFramedAry);

for(int h = 0; h < 9; h++){

sum += maskNeighbor[h] \* dataNeighbor[h];

pixelSum += maskNeighbor[h];

}

return sum / pixelSum;

}

void ComputeGaussImage(int\*\* gaussAry, int\*\* mirrorFramedAry, int\*\* maskAry, int& newMin, int& newMax, int numRows, int numCols, int maskRows, int maskCols){

newMin = 9999;

newMax = 0;

int i = 1, j = 1;

while(i < numRows+1){

j = 1;

while (j < numCols+1) {

gaussAry[i][j] = convolution(i, j, mirrorFramedAry, maskAry, maskRows, maskCols);

if(newMin > gaussAry[i][j]) { newMin = gaussAry[i][j]; }

if(newMax < gaussAry[i][j]) { newMax = gaussAry[i][j]; }

j++;

}

i++;

}

}

void sort(int\* array){ //bubbleSort

int i, j, temp;

for (i = 0; i < 9-1; i++){ //8 for bubbleSort

for (j = 0; j < 9-i-1; j++){

if (array[j] > array[j+1]){

//swapping

temp = array[j];

array[j] = array[j+1];

array[j+1] = temp;

}

}

}

}

void loadNeighbors(int i, int j, int\* neighborAry, int\*\* mirrorFramedAry){

int index = 0;

for(int a = -1; a < 2; a++){

neighborAry[index++] = mirrorFramedAry[i+a][j-1];

neighborAry[index++] = mirrorFramedAry[i+a][j];

neighborAry[index++] = mirrorFramedAry[i+a][j+1];

}

}

void ComputeMedianImage(int\*\* medianAry, int\*\* mirrorFramedAry, int& newMin, int& newMax, int numRows, int numCols){

newMin = 999;

newMax = 0;

int\* neighborAry = new int[9];

int i = 1, j = 1;

while(i < numRows+1){

j = 1;

while (j < numCols+1) {

loadNeighbors(i, j, neighborAry, mirrorFramedAry);

sort(neighborAry);

medianAry[i][j] = neighborAry[5];

if(newMin > medianAry[i][j]) { newMin = medianAry[i][j]; }

if(newMax < medianAry[i][j]) { newMax = medianAry[i][j]; }

j++;

}

i++;

}

}

void outputAryToFile(int\*\* array, fstream& outFile, int numRows, int numCols, int& newMin, int& newMax){

outFile << numRows << " " << numCols << " " << newMin << " " << newMax << endl;

for(int i = 1; i < numRows+1; i++){

for(int j = 1; j < numCols+1; j++){

outFile<< array[i][j] << " ";

}

outFile << endl;

}

}

int avg3x3(int i, int j, int\*\* mirrorFramedAry){

int sum = 0;

for(int a = -1; a < 2; a++){

sum += mirrorFramedAry[i+a][j-1];

sum += mirrorFramedAry[i+a][j];

sum += mirrorFramedAry[i+a][j+1];

}

return sum / 9;

}

void ComputeAvgImage(int\*\* avgAry, int\*\* mirrorFramedAry, int& newMin, int& newMax, int numRows, int numCols){

newMin = 999;

newMax = 0;

int i = 1, j = 1;

while(i < numRows+1){

j = 1;

while (j < numCols+1) {

avgAry[i][j] = avg3x3(i, j, mirrorFramedAry);

if(newMin > avgAry[i][j]) { newMin = avgAry[i][j]; }

if(newMax < avgAry[i][j]) { newMax = avgAry[i][j]; }

j++;

}

i++;

}

}

void mirrorFraming(int\*\* mirrorFramedAry, int rows, int cols){

for(int i = 0; i < cols+2; i++){

mirrorFramedAry[0][i] = mirrorFramedAry[1][i];

mirrorFramedAry[rows+1][i] = mirrorFramedAry[rows][i];

}

for(int i = 0; i < rows+2; i++){

mirrorFramedAry[i][0] = mirrorFramedAry[i][1];

mirrorFramedAry[i][cols+1] = mirrorFramedAry[i][cols];

}

}

void loadImage(int\*\* mirrorFramedAry, fstream& inFile, int rows, int cols){

int value;

for(int i = 1; i < rows+1; i++){

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

inFile >> value;

mirrorFramedAry[i][j] = value;

}

}

}

void loadMask(int\*\* maskAry, fstream& maskFile, int maskRows, int maskCols){

int value;

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

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

maskFile >> value;

maskAry[i][j] = value;

}

}

}

**Histogram of AvgOutImg:**

A screenshot of text

Description automatically generated

**PrettyPrint Threshold of AvgOutImg:**

**A screenshot of a cell phone

Description automatically generated**

**Histogram of MedianOutImg:**

**A screenshot of a cell phone

Description automatically generated**

**PrettyPrint Threshold of MedianOutImg:A close up of a logo

Description automatically generated**

**Histogram of GaussOutImg:**

**A screenshot of a cell phone

Description automatically generated**

**PrettyPrint Threshold of GaussOutImg:**

**A close up of text on a white background

Description automatically generated**