Project 2.2: Average Filtering (C++)

CSC 381 Rafael Carmilema

Due Date: Sep / 08 /2016

III. Algorithm for 3 by 3 averaging filter  
  
  
step 0: - open the input file and output file  
          - read the image header, the four numbers  
          - dynamically allocate mirrorFramedAry  
          - dynamically alloicate tempAry  
  
step 1: read the input file and load onto mirrowframeAry begin at [1,1]  
  
step 2: mirrowFramed the mirrorFramedAr  
  
step 3: process the MirrorframedAry, from left to right and top to bottom  
          begin at (1, 1)  
  
        - neighborAry <- load MirrorframedAry[i,j]'s 3 X 3 neighborhoods  
          - tempAry[i,j] <-- compute the averaging of neighborAry  
  
          - keep tracking the newMin and newMax of tempAry  
  
step 4: repeat step 3 until all pixels are processed  
  
step 5: output to AVG3X3Out.txt the image header (numRows, numCols, newMin, newMax)  
  
step 6: output to AVG3X3Out.txt from tempAry, begin at [1,1], without the pixels on the frame  
  
step 7: close input file and AVG3X3Out.txt

#include <iostream>

#include <fstream>

#include <string>

#include <sstream>

using namespace std;

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

class MedianFilter{

private:

int countarr=0;

int row;

int col;

int min;

int max;

int newmin;

int newmax;

int\*\* mirrorFramedAry;

int\*\* tempAry;

int neighborAry[9];

int\*\* thrArray;

int globalcount=0;

public:

MedianFilter(int r, int c, int mi, int ma){

row=r;

col=c;

min=mi;

max=ma;

// allocate 2d dynamic array for mirrorFramedAry

thrArray = new int\*[r];

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

thrArray[i]= new int [c];

}

//initialize the 2d threshold array

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

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

thrArray[i][j]=0;

}

}

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

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

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

}

//ini

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

for(int j=0;j <c+2;j++){

mirrorFramedAry[i][j]=0;

}

}

//allocate temp

tempAry = new int\*[r+2];

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

tempAry[i]= new int [c+2];

}

//init

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

for(int j=0; j<c+2;j++){

tempAry[i][j]=0;

}

}

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

neighborAry[i]=9999; // just edit

cout<<neighborAry[i]<<" ";

}

cout<<endl;

}

void readintoMirrorArr(int value, int r, int c){

mirrorFramedAry[r+1][c+1]=value;

}

void MirrorFramed(){

for(int k = 0; k<= row+1;k++){

mirrorFramedAry[k][0]= mirrorFramedAry[k][1];

mirrorFramedAry[k][col+1]= mirrorFramedAry[k][col];

}

for(int p=0; p<= col+1; p++){

mirrorFramedAry[0][p]= mirrorFramedAry[1][p];

mirrorFramedAry[row+1][p]= mirrorFramedAry[row][p];

}

}

void processMirrorFramed(){

cout<<"Test"<<endl;

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

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

getNeighbors(i,j);

globalcount++;

//cout<<mirrorFramedAry[i][j];

}

//cout<<endl;

}

}

//3x3 grid here

void printNeighborArr(){

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

cout<<neighborAry[i]<<" ";

}

cout<<endl;

}

void getNeighbors(int i, int j){

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

for(int columns=j-1;columns<=j+1;columns++){

//cout<<

loadNeightbors(mirrorFramedAry[rows][columns]);

countarr++;

//cout<<countarr;

}

}

selectionSort(i, j);

if(globalcount<=10){

printNeighborArr();

}

}

void selectionSort(int a, int b){

int min;

int loc;

int temp;

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

/\* if(i==5){

break;

}\*/

min=neighborAry[i];

loc=i;

for(int j=i+1;j<9;j++)

{

if(min>neighborAry[j])

{

min=neighborAry[j];

loc=j;

}

}

temp=neighborAry[i];

neighborAry[i]=neighborAry[loc];

neighborAry[loc]=temp;

}

//track min and max

tempAry[a][b]=neighborAry[4];

if(a==1&&b==1){

newmin=tempAry[a][b];

newmax=tempAry[a][b];

}

else{

if(tempAry[a][b]>newmax){

newmax=tempAry[a][b];

}

if(tempAry[a][b]<newmin){

newmin=tempAry[a][b];

}

}

}

void loadNeightbors(int value){

if(countarr==9){

countarr=0;

}

neighborAry[countarr]=value;

}

void printTempFramed(ofstream& outfile1){

outfile1<<row<<" "<<col<<" "<<newmin<<" "<<newmax<<endl;

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

cout<<i<<" ";

}

cout<<endl;

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

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

cout<<tempAry[i][j];

outfile1<<tempAry[i][j]<<" ";

}

cout<<endl;

outfile1<<endl;

}

}

void printMirrorFramed(){

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

for(int j=0; j<col+2;j++){

cout<<mirrorFramedAry[i][j];

}

cout<<endl;

}

}

};

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

//cout<<"hello "<<endl;

if ( argc < 2 ) // argc should be 2 for correct execution

// We print argv[0] assuming it is the program name

cout<<"usage: "<< argv[0] <<" <filename>\n";

else {

// We assume argv[1] is a filename to open

ifstream the\_file ( argv[1] );

// Always check to see if file opening succeeded

if ( !the\_file.is\_open() )

cout<<"Could not open file\n";

else {

int count =0;

int string;

int row;

int col;

int min;

int max;

int\*\* inputArray;

ofstream outfile1;

outfile1.open(argv[2]);

while(the\_file>>string){

count++;

if(count==1){

row=string;

cout<<row<<endl;

}

else if(count==2){

col=string;

cout<<col<<endl;

}

else if(count == 3){

min=string;

cout<<min<<endl;

}

else if(count ==4){

max=string;

cout<<max<<endl;

break;

}

else{

break;

}

}

inputArray= new int\*[row];

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

inputArray[i]= new int[col];

}

for(int r = 0; r<row;r++){

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

inputArray[r][c]=0;

}

}

MedianFilter test(row,col,min,max);

for(int r= 0; r<row;r++){

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

while(the\_file>>string){

inputArray[r][c]=string;

test.readintoMirrorArr(string,r ,c);

//cout<<r<<" "<< c<<" "<<string<<endl;

break;

}

}

// cout<<endl;

}

//test.printMirrorFramed();

test.MirrorFramed();

// cout<<endl;

// test.printMirrorFramed();

//cout<<endl;

test.processMirrorFramed();

test.printTempFramed(outfile1);

the\_file.close();

outfile1.close();

}//else

}

return 0;

}

Histogram of Average

(0):0

(1):7+++++++

(2):80 +'s

(3):80 +'s

(4):80 +'s

(5):71+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

(6):48++++++++++++++++++++++++++++++++++++++++++++++++

(7):80 +'s

(8):80 +'s

(9):57+++++++++++++++++++++++++++++++++++++++++++++++++++++++++

(10):19+++++++++++++++++++

(11):30++++++++++++++++++++++++++++++

(12):41+++++++++++++++++++++++++++++++++++++++++

(13):49+++++++++++++++++++++++++++++++++++++++++++++++++

(14):33+++++++++++++++++++++++++++++++++

(15):17+++++++++++++++++

(16):28++++++++++++++++++++++++++++

(17):43+++++++++++++++++++++++++++++++++++++++++++

(18):32++++++++++++++++++++++++++++++++

(19):6++++++

(20):12++++++++++++

(21):12++++++++++++

(22):20++++++++++++++++++++

(23):11+++++++++++

(24):14++++++++++++++

(25):8++++++++

(26):6++++++

(27):14++++++++++++++

(28):17+++++++++++++++++

(29):15+++++++++++++++

(30):6++++++

(31):15+++++++++++++++

(32):36++++++++++++++++++++++++++++++++++++

(33):21+++++++++++++++++++++

(34):14++++++++++++++

(35):5+++++

(36):11+++++++++++

(37):22++++++++++++++++++++++

(38):21+++++++++++++++++++++

(39):21+++++++++++++++++++++

(40):13+++++++++++++

(41):23+++++++++++++++++++++++

(42):29+++++++++++++++++++++++++++++

(43):43+++++++++++++++++++++++++++++++++++++++++++

(44):41+++++++++++++++++++++++++++++++++++++++++

(45):32++++++++++++++++++++++++++++++++

(46):24++++++++++++++++++++++++

(47):21+++++++++++++++++++++

(48):20++++++++++++++++++++

(49):15+++++++++++++++

(50):7+++++++

(51):6++++++

(52):3+++

(53):1+

(54):80 +'s

Pretty Print

1

1

1111111111

1111111111111

11111111111111

11111111111111

11111111111111 1

1111111111111111 11

1111111111111111 111

1111111111111111 11

1111111111111111 11

11111111111111111 1

111111111111111111 1111111

111111111111111111 111 11

111111111 11111111 11111

111111 1111111 111

11111 11111111 11

1111111111111111

11111111111111

1111111111111

111111111111111

11

111