#include <stdio.h>

#include <stdlib.h>

#include <fstream>

#include <opencv2/opencv.hpp>

#include <limits>

#include<string.h>

#include<sstream>

#include<conio.h>

#include <iostream>

#include <iomanip>

#include <ctime>// include this header

#include <omp.h>

using namespace cv;

using namespace std;

/\*\*

\* This function makes use of the Mat << operator.

\*/

void dump(const Mat &mat, const char\* fname)

{

ofstream filestream;

filestream.open (fname);

filestream << mat << endl << endl;

filestream.close();

}

/\*\*

\* Used to avoid noise in the image.

\*/

void applyMedian(cv::Mat &input, cv::Mat &output) {

double sigma = 6.75;

//cv::Mat gaussKernel = cv::getMedianKernel(9,sigma,CV\_32F);

cv::MedianBlur( input, output, cv::Size(3,3), 6.75); //1.5

}

/\*\*

\* This is similar to the implementation of Robert Laganière.

\* See his book: OpenCV 2 Computer Vision Application Programming Cookbook.

\*/

cv::Mat showHistogram(const cv::Mat &inImage){

cv::MatND hist;

// For a gray scale [0:255] we have 256 bins

const int bins[1] = {256};

const float hranges[2] = {0.0, 255.0};

const float\* ranges[1] = { hranges };

const int channels[1] = {1};

cv::calcHist(const\_cast<Mat\*>(&inImage),

1, // histogram from 1 image only

channels,

cv::Mat(), // no mask is used

hist, // the output histogram

1, // 1D histogram

bins,

ranges // pixel value ranges

);

// Get min and max bin values

double maxVal=0;

double minVal=0;

cv::minMaxLoc(hist, &minVal, &maxVal, 0, 0);

// The image to display the histogram

cv::Mat histImg(bins[0], bins[0], CV\_8U, cv::Scalar(255));

// Map the highest point to 95% of the histogram height to leave some

// empty space at the top

const int histHeight = bins[0];

const int maxHeight = 0.95 \* histHeight;

cv::Mat\_<float>::iterator it = hist.begin<float>();

cv::Mat\_<float>::iterator itend = hist.end<float>();

int barPosition = 0;

for ( ; it != itend; ++it) {

float histValue = (\*it);

int barHeight = ( histValue \* maxHeight ) / maxVal;

cv::line(histImg,

// start the line from the bottom, and go up based on the barHeight

// Remember the (0,0) is the top left corner

cv::Point(barPosition, histHeight),

cv::Point(barPosition, histHeight - barHeight),

cv::Scalar::all(0));

barPosition++;

}

return histImg;

}

void applyClosing( cv::Mat &binaryImage, int element\_radius = 2 ) {

int element\_type = cv::MORPH\_ELLIPSE;

// The structuring element used for dilation and erosion.

Mat element = cv::getStructuringElement( element\_type,

Size( 2\*element\_radius + 1, 2\*element\_radius+1 ),

Point( element\_radius, element\_radius ) );

dump( element, "element.data" );

cv::dilate(binaryImage, binaryImage,

element,

Point(-1, -1),

2

);

cv::erode(binaryImage, binaryImage,

element,

// Position of the anchor within the structuring element.

// The default value -1,-1 means that the anchor is at the element center

Point(-1, -1),

// Iterations: the number of times this operation is applied

2

);

}

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

{

int start\_s=clock();

cout << "Compiled with OpenCV version " << CV\_VERSION << endl;

// Mat inImage = cv::imread("C://scenic.JPG");

int x=1;

int avg=0;

int const n=7000;

//Mat image\_array[1000];

cv::Mat image\_array[8000];

//cv::Mat image\_array[500];

//cv::Mat binaryimage[8000];

//cv::Mat grayimage[1000];

cv::Mat \*grayimage=new cv::Mat[8000];

//Mat grayimage1[1000];

cv::Mat \*grayimage1=new cv::Mat[8000];

cv::Mat \*grayimage2=new cv::Mat[8000];

//Mat grayimage2[1000];

string \*s=new string[8000];

string \*str1=new string[8000];

std::string firstlevel (".jpg");

std::string secondlevel ("\*");

std::string scheme ("D://");

//std::string hostname[1000];

std::string \*hostname=new std::string[8000];

std::string url;

int a = 1;

Mat dst[1000];

stringstream \*ss=new stringstream[8000];

//ss << a;

//string str = ss.str();

//hostname=scheme+str+firstlevel;

//omp\_set\_num\_threads(5);

//printf\_s("%d\n", omp\_get\_num\_threads( ));

// #pragma omp parallel num\_threads(1)

//#pragma omp parallel for

for(int ind=1;ind<546;ind++)

{

ss[ind]<<ind;

str1[ind]=ss[ind].str();

hostname[ind]=scheme+str1[ind]+firstlevel;

image\_array[ind]=imread(hostname[ind]);

cout<<"\nImage READ"<<ind;

cvtColor(image\_array[ind], grayimage[ind], CV\_RGB2GRAY);

grayimage[ind].convertTo(grayimage[ind], CV\_8UC1);

//image\_array[ind].convertTo(grayimage[ind], CV\_8UC1);

// applyMedian( grayimage[ind], grayimage[ind] );

cv::MedianBlur( grayimage[ind], grayimage1[ind], cv::Size(3,3),6.75);

cv::threshold(grayimage1[ind], grayimage2[ind]

, 0 // the value doesn't matter for Otsu thresholding

, 255 // we could choose any non-zero value. 255 (white) makes it easy to see the binary image "| cv::THRESH\_BINARY\_INV"

, cv::THRESH\_OTSU);

avg = avg + countNonZero(grayimage2[ind]);

image\_array[ind].release();

grayimage[ind].release();

grayimage1[ind].release();

grayimage2[ind].release();

int stop\_s=clock();

// if (ind==50 || ind==100 || ind==500 || ind==1000)

cout << " Tme: " << (stop\_s-start\_s)/double(CLOCKS\_PER\_SEC)\*1000 <<"ms"<< endl;

}

cout<<"\nAverage Segmented Pixels:"<<avg/545;

//int stop\_s=clock();

//cout << "time: " << (stop\_s-start\_s)/double(CLOCKS\_PER\_SEC)\*1000 << endl;

//image\_array[0]=imread("C://aa.JPG");

//image\_array[0]=imread(hostname);

//image\_array[1]=imread("C://binary.JPG");

/\* for(int ind=1;ind<2;ind++)

{

imshow("dem1",image\_array[ind]);

}\*/

// imshow("Original", inImage);

// Mat histgram = showHistogram(inImage);

// imshow("Histogram", histgram);

// Mat grayImage;

// cvtColor(inImage, grayImage, CV\_RGB2GRAY);

// applyMedian( grayImage, grayImage );

// The Otsu thresholding algorithm works well when the histogram has a bimodal distribution.

// It will find the threshold value that maximizes the extra-class variance while

// keeping a low intra-class variance.

// cv::Mat binaryImage;

/\* cv::threshold(grayImage, binaryImage

, 0 // the value doesn't matter for Otsu thresholding

, 255 // we could choose any non-zero value. 255 (white) makes it easy to see the binary image

, cv::THRESH\_OTSU | cv::THRESH\_BINARY\_INV); \*/

// int avg = countNonZero(binaryImage);

//cout<<"\n"<<avg;

// applyClosing( binaryImage, 2 );

// imshow("Binary", binaryImage);

/\*cv::Mat outImage = cv::Mat::zeros( inImage.rows, inImage.cols, inImage.type() );

inImage.copyTo( outImage, binaryImage );

imshow("Result", outImage);\*/

getch();

waitKey(0);

return EXIT\_SUCCESS;

}