**LAB 3 REPORT – Aaron Bruner**

The purpose of this lab is to implement thinning, branchpoint and endpoint detection to recognize letters in an image of text. The lab instructions laid out 2 steps for us to follow. The first step was to read the input image, msf image, and ground truth.

Text

Description automatically generated

We are using the MSF image from lab 3 which is shown above.

Using a threshold value of 200 we can see how it thins out the image.  
Original Image:Qr code

Description automatically generated Image after Thinning: Qr code

Description automatically generated

For this example, we get a result of 4 end points, 0 branches. TP = 151 and FP = 1109 for this example.

The ideal T is where TP is the highest and so the ideal T should be 244.

Threshold[0] TP = 151 FP = 1111

Threshold[1] TP = 151 FP = 1111

Threshold[2] TP = 151 FP = 1111

Threshold[3] TP = 151 FP = 1111

Threshold[4] TP = 151 FP = 1111

Threshold[5] TP = 151 FP = 1111

Threshold[6] TP = 151 FP = 1111

Threshold[7] TP = 151 FP = 1111

Threshold[8] TP = 151 FP = 1111

Threshold[9] TP = 151 FP = 1111

Threshold[10] TP = 151 FP = 1111

Threshold[11] TP = 151 FP = 1111

Threshold[12] TP = 151 FP = 1111

Threshold[13] TP = 151 FP = 1111

Threshold[14] TP = 151 FP = 1111

Threshold[15] TP = 151 FP = 1111

Threshold[16] TP = 151 FP = 1111

Threshold[17] TP = 151 FP = 1111

Threshold[18] TP = 151 FP = 1111

Threshold[19] TP = 151 FP = 1111

Threshold[20] TP = 151 FP = 1111

Threshold[21] TP = 151 FP = 1111

Threshold[22] TP = 151 FP = 1111

Threshold[23] TP = 151 FP = 1111

Threshold[24] TP = 151 FP = 1111

Threshold[25] TP = 151 FP = 1111

Threshold[26] TP = 151 FP = 1111

Threshold[27] TP = 151 FP = 1111

Threshold[28] TP = 151 FP = 1111

Threshold[29] TP = 151 FP = 1111

Threshold[30] TP = 151 FP = 1111

Threshold[31] TP = 151 FP = 1111

Threshold[32] TP = 151 FP = 1111

Threshold[33] TP = 151 FP = 1111

Threshold[34] TP = 151 FP = 1111

Threshold[35] TP = 151 FP = 1111

Threshold[36] TP = 151 FP = 1111

Threshold[37] TP = 151 FP = 1111

Threshold[38] TP = 151 FP = 1111

Threshold[39] TP = 151 FP = 1111

Threshold[40] TP = 151 FP = 1111

Threshold[41] TP = 151 FP = 1111

Threshold[42] TP = 151 FP = 1111

Threshold[43] TP = 151 FP = 1111

Threshold[44] TP = 151 FP = 1111

Threshold[45] TP = 151 FP = 1111

Threshold[46] TP = 151 FP = 1111

Threshold[47] TP = 151 FP = 1111

Threshold[48] TP = 151 FP = 1111

Threshold[49] TP = 151 FP = 1111

Threshold[50] TP = 151 FP = 1111

Threshold[51] TP = 151 FP = 1111

Threshold[52] TP = 151 FP = 1111

Threshold[53] TP = 151 FP = 1111

Threshold[54] TP = 151 FP = 1111

Threshold[55] TP = 151 FP = 1111

Threshold[56] TP = 151 FP = 1111

Threshold[57] TP = 151 FP = 1111

Threshold[58] TP = 151 FP = 1111

Threshold[59] TP = 151 FP = 1111

Threshold[60] TP = 151 FP = 1111

Threshold[61] TP = 151 FP = 1111

Threshold[62] TP = 151 FP = 1111

Threshold[63] TP = 151 FP = 1111

Threshold[64] TP = 151 FP = 1111

Threshold[65] TP = 151 FP = 1111

Threshold[66] TP = 151 FP = 1111

Threshold[67] TP = 151 FP = 1111

Threshold[68] TP = 151 FP = 1111

Threshold[69] TP = 151 FP = 1111

Threshold[70] TP = 151 FP = 1111

Threshold[71] TP = 151 FP = 1111

Threshold[72] TP = 151 FP = 1111

Threshold[73] TP = 151 FP = 1111

Threshold[74] TP = 151 FP = 1111

Threshold[75] TP = 151 FP = 1111

Threshold[76] TP = 151 FP = 1111

Threshold[77] TP = 151 FP = 1111

Threshold[78] TP = 151 FP = 1111

Threshold[79] TP = 151 FP = 1111

Threshold[80] TP = 151 FP = 1111

Threshold[81] TP = 151 FP = 1111

Threshold[82] TP = 151 FP = 1111

Threshold[83] TP = 151 FP = 1111

Threshold[84] TP = 151 FP = 1111

Threshold[85] TP = 151 FP = 1111

Threshold[86] TP = 151 FP = 1111

Threshold[87] TP = 151 FP = 1111

Threshold[88] TP = 151 FP = 1111

Threshold[89] TP = 151 FP = 1111

Threshold[90] TP = 151 FP = 1111

Threshold[91] TP = 151 FP = 1111

Threshold[92] TP = 151 FP = 1111

Threshold[93] TP = 151 FP = 1111

Threshold[94] TP = 151 FP = 1111

Threshold[95] TP = 151 FP = 1111

Threshold[96] TP = 151 FP = 1111

Threshold[97] TP = 151 FP = 1111

Threshold[98] TP = 151 FP = 1111

Threshold[99] TP = 151 FP = 1111

Threshold[100] TP = 151 FP = 1111

Threshold[101] TP = 151 FP = 1111

Threshold[102] TP = 151 FP = 1111

Threshold[103] TP = 151 FP = 1111

Threshold[104] TP = 151 FP = 1111

Threshold[105] TP = 151 FP = 1111

Threshold[106] TP = 151 FP = 1111

Threshold[107] TP = 151 FP = 1111

Threshold[108] TP = 151 FP = 1111

Threshold[109] TP = 151 FP = 1111

Threshold[110] TP = 151 FP = 1111

Threshold[111] TP = 151 FP = 1111

Threshold[112] TP = 151 FP = 1111

Threshold[113] TP = 151 FP = 1111

Threshold[114] TP = 151 FP = 1111

Threshold[115] TP = 151 FP = 1111

Threshold[116] TP = 151 FP = 1111

Threshold[117] TP = 151 FP = 1111

Threshold[118] TP = 151 FP = 1111

Threshold[119] TP = 151 FP = 1111

Threshold[120] TP = 151 FP = 1111

Threshold[121] TP = 151 FP = 1111

Threshold[122] TP = 151 FP = 1111

Threshold[123] TP = 151 FP = 1111

Threshold[124] TP = 151 FP = 1111

Threshold[125] TP = 151 FP = 1111

Threshold[126] TP = 151 FP = 1111

Threshold[127] TP = 151 FP = 1111

Threshold[128] TP = 151 FP = 1111

Threshold[129] TP = 151 FP = 1111

Threshold[130] TP = 151 FP = 1111

Threshold[131] TP = 151 FP = 1111

Threshold[132] TP = 151 FP = 1111

Threshold[133] TP = 151 FP = 1111

Threshold[134] TP = 151 FP = 1111

Threshold[135] TP = 151 FP = 1111

Threshold[136] TP = 151 FP = 1111

Threshold[137] TP = 151 FP = 1111

Threshold[138] TP = 151 FP = 1111

Threshold[139] TP = 151 FP = 1111

Threshold[140] TP = 151 FP = 1111

Threshold[141] TP = 151 FP = 1111

Threshold[142] TP = 151 FP = 1111

Threshold[143] TP = 151 FP = 1111

Threshold[144] TP = 151 FP = 1111

Threshold[145] TP = 151 FP = 1111

Threshold[146] TP = 151 FP = 1111

Threshold[147] TP = 151 FP = 1111

Threshold[148] TP = 151 FP = 1111

Threshold[149] TP = 151 FP = 1111

Threshold[150] TP = 151 FP = 1111

Threshold[151] TP = 151 FP = 1111

Threshold[152] TP = 151 FP = 1111

Threshold[153] TP = 151 FP = 1111

Threshold[154] TP = 151 FP = 1111

Threshold[155] TP = 151 FP = 1111

Threshold[156] TP = 151 FP = 1111

Threshold[157] TP = 151 FP = 1111

Threshold[158] TP = 151 FP = 1111

Threshold[159] TP = 151 FP = 1111

Threshold[160] TP = 151 FP = 1111

Threshold[161] TP = 151 FP = 1111

Threshold[162] TP = 151 FP = 1111

Threshold[163] TP = 151 FP = 1111

Threshold[164] TP = 151 FP = 1111

Threshold[165] TP = 151 FP = 1111

Threshold[166] TP = 151 FP = 1111

Threshold[167] TP = 151 FP = 1109

Threshold[168] TP = 151 FP = 1109

Threshold[169] TP = 151 FP = 1109

Threshold[170] TP = 151 FP = 1109

Threshold[171] TP = 151 FP = 1109

Threshold[172] TP = 151 FP = 1109

Threshold[173] TP = 151 FP = 1109

Threshold[174] TP = 151 FP = 1109

Threshold[175] TP = 151 FP = 1109

Threshold[176] TP = 151 FP = 1109

Threshold[177] TP = 151 FP = 1109

Threshold[178] TP = 151 FP = 1109

Threshold[179] TP = 151 FP = 1109

Threshold[180] TP = 151 FP = 1109

Threshold[181] TP = 151 FP = 1109

Threshold[182] TP = 151 FP = 1109

Threshold[183] TP = 151 FP = 1109

Threshold[184] TP = 151 FP = 1109

Threshold[185] TP = 151 FP = 1109

Threshold[186] TP = 151 FP = 1109

Threshold[187] TP = 151 FP = 1109

Threshold[188] TP = 151 FP = 1109

Threshold[189] TP = 151 FP = 1109

Threshold[190] TP = 151 FP = 1109

Threshold[191] TP = 151 FP = 1109

Threshold[192] TP = 151 FP = 1109

Threshold[193] TP = 151 FP = 1109

Threshold[194] TP = 151 FP = 1109

Threshold[195] TP = 151 FP = 1109

Threshold[196] TP = 151 FP = 1109

Threshold[197] TP = 151 FP = 1109

Threshold[198] TP = 151 FP = 1109

Threshold[199] TP = 151 FP = 1109

Threshold[200] TP = 151 FP = 1109

Threshold[201] TP = 151 FP = 1109

Threshold[202] TP = 151 FP = 1109

Threshold[203] TP = 151 FP = 1109

Threshold[204] TP = 151 FP = 1109

Threshold[205] TP = 151 FP = 1109

Threshold[206] TP = 151 FP = 1109

Threshold[207] TP = 151 FP = 1109

Threshold[208] TP = 151 FP = 1109

Threshold[209] TP = 151 FP = 1109

Threshold[210] TP = 151 FP = 1109

Threshold[211] TP = 151 FP = 1109

Threshold[212] TP = 151 FP = 1109

Threshold[213] TP = 151 FP = 1109

Threshold[214] TP = 151 FP = 1109

Threshold[215] TP = 151 FP = 1109

Threshold[216] TP = 151 FP = 1109

Threshold[217] TP = 151 FP = 1109

Threshold[218] TP = 151 FP = 1109

Threshold[219] TP = 151 FP = 1109

Threshold[220] TP = 151 FP = 1109

Threshold[221] TP = 151 FP = 1109

Threshold[222] TP = 151 FP = 1109

Threshold[223] TP = 151 FP = 1109

Threshold[224] TP = 151 FP = 1109

Threshold[225] TP = 151 FP = 1109

Threshold[226] TP = 151 FP = 1109

Threshold[227] TP = 151 FP = 1109

Threshold[228] TP = 151 FP = 1109

Threshold[229] TP = 151 FP = 1109

Threshold[230] TP = 151 FP = 1109

Threshold[231] TP = 151 FP = 1109

Threshold[232] TP = 151 FP = 1109

Threshold[233] TP = 151 FP = 1109

Threshold[234] TP = 151 FP = 1109

Threshold[235] TP = 151 FP = 1109

Threshold[236] TP = 151 FP = 1109

Threshold[237] TP = 151 FP = 1109

Threshold[238] TP = 151 FP = 1109

Threshold[239] TP = 151 FP = 1109

Threshold[240] TP = 151 FP = 1109

Threshold[241] TP = 151 FP = 1109

Threshold[242] TP = 151 FP = 1109

Threshold[243] TP = 151 FP = 1109

Threshold[244] TP = 151 FP = 1109

Threshold[245] TP = 146 FP = 1047

Threshold[246] TP = 146 FP = 1047

Threshold[247] TP = 137 FP = 1017

Threshold[248] TP = 137 FP = 1017

Threshold[249] TP = 137 FP = 1017

Threshold[250] TP = 53 FP = 406

Threshold[251] TP = 53 FP = 406

Threshold[252] TP = 53 FP = 406

Threshold[253] TP = 53 FP = 406

Threshold[254] TP = 53 FP = 406

Threshold[255] TP = 0 FP = 0

**Chart, line chart

Description automatically generated**

Source Code:

/\* File : lab3.c

Author: Aaron Bruner

Class : ECE - 4310 : Introduction to Computer Vision

Term : Fall 2022

Description: The purpose of this lab is to implement thinning, branchpoint and endpoint detection to recognize letters in an image of text.

Required Files:

\* parenthood.ppm

\* parenthood\_e\_template.ppm

\* parenthood\_gt.txt

\* msf\_e.ppm

Bugs:

\* Currently none

\*/

#define True 1

#define False 0

#define DEBUG False

#define T 255 // Upper limit for thresholding

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

struct groundTruth {

char letter;

int x; // COLUMN

int y; // ROW

};

unsigned char\* readImage( int\* ROWS, int\* COLS, char\* source);

unsigned char\* createImage(int size);

void thin(unsigned char\* srcImage);

void edgeNonEdgeTransitions(unsigned char\* img, int \*transitions, int \*neighbors, int \*passNum4, int r, int c);

void branchEndPoints(unsigned char\* img, int\* isE);

char\* sourceImageDir = "parenthood.ppm";

char\* templateImageDir = "parenthood\_e\_template.ppm";

char\* groundTruthDir = "parenthood\_gt.txt";

char\* msfeImageDir = "msf\_e.ppm";

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

{

unsigned char\* sourceImage, \*templateImage, \*msf\_eImage, \*thresholdImage;

struct groundTruth\* truth;

char temp, gtLetter;

int temp1, temp2, fileRows = 0; // Number of rows in the ground truth file

int i = 0, j = 0, sourceROWS, sourceCOLS, templateROWS, templateCOLS, msfe\_ROWS, msfe\_COLS;

int dr = 7, dc = 4, found = False, TP = 0, FP = 0, threshLocation = 0, isE = False, endBranch = 0, eRows = 15, eCols = 9;

int gtR = 0, gtC = 0, index = 0, end = 0, branch = 0;

FILE\* fpt, \*TPFPfpt;

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

/\* STEP 1: Read in source, msf image and ground truth \*/

/\* \* User provides no arguments (argc == 1) then we default to specified files \*/

/\* \* User provides 4 arguments (argc == 5) then we open provided files \*/

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

//printf("Step 1:\n");

if (argc == 1) {

//printf("Performing matched filter on images [%s] and [%s] using ground truth [%s] and MSF image [%s]\n", sourceImageDir, templateImageDir, groundTruthDir, msfeImageDir);

//printf("\t\* Reading in source image...");

sourceImage = readImage(&sourceROWS, &sourceCOLS, sourceImageDir); //printf("\t[SUCCESS]\n");

//printf("\t\* Reading in template image...");

templateImage = readImage(&templateROWS, &templateCOLS, templateImageDir); //printf("\t[SUCCESS]\n");

//printf("\t\* Reading in MSF\_e image...");

msf\_eImage = readImage(&msfe\_ROWS, &msfe\_COLS, msfeImageDir); //printf("\t[SUCCESS]\n");

// Read in CSV/TXT file

//printf("\t\* Opening ground truth file...");

fpt = fopen(groundTruthDir, "r");

//fpt == NULL ? printf("Failed to open %s\n", groundTruthDir), exit(0) : printf("\t[SUCCESS]\n");

}

else if (argc == 5)

{

printf("Performing matched filter on images [%s] and [%s] using ground truth [%s] and MSF image [%s]\n", argv[1], argv[2], argv[4], argv[3]);

printf("\t\* Reading in source image...");

sourceImage = readImage(&sourceROWS, &sourceCOLS, argv[1]); printf("\t[SUCCESS]\n");

printf("\t\* Reading in template image...");

templateImage = readImage(&templateROWS, &templateCOLS, argv[2]); printf("\t[SUCCESS]\n");

printf("\t\* Reading in MSF\_e image...");

msf\_eImage = readImage(&msfe\_ROWS, &msfe\_COLS, argv[3]); printf("\t[SUCCESS]\n");

// Read in CSV/TXT file

printf("\t\* Opening ground truth file...");

fpt = fopen(argv[4], "r");

fpt == NULL ? printf("Failed to open %s\n", argv[3]), exit(0) : printf("\t[SUCCESS]\n");

}

else

{

printf("Incorrect number of arguments...\nUsage: ./lab2 (sourceImage.ppm) (templateImage.ppm) (msf\_e.ppm) (groundTruth.txt)\n");

exit(0);

}

while ((i = fscanf(fpt, "%c %d %d\n", &temp, &temp1, &temp2)) && !feof(fpt))

if (i == 3) fileRows += 1;

//printf("\t\* Found %d number of rows in the ground truth file\n", fileRows);

//printf("\t\* Allocating space for ground truth file...");

truth = calloc(fileRows, sizeof(struct groundTruth)); //printf("\t[SUCCESS]\n");

rewind(fpt); // Return to the beginning of the file

//printf("\t\* Scanning in values from ground truth file...");

for (i = 0; i <= fileRows && !feof(fpt); i++)

{

fscanf(fpt, "%c %d %d\n", &truth[i].letter, &truth[i].x, &truth[i].y);

}

fclose(fpt);

//printf("\t[Read in %d rows]\n", i - 1);

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

/\* STEP 2: Looping through the following steps for a range of T \*/

/\* a) Loop through the ground truth letter locations \*/

/\* i) Check a 9x15 pixel area centered at the ground truth location. If any pixel in \*/

/\* the msf image is greater then the threshold, consider the letter "detected". \*/

/\* If none of the pixels in the 9 x 15 area are greater than the threshold, \*/

/\* consider the letter "not detected". \*/

/\* ii) If the letter is "not detected" continue to the next letter \*/

/\* iii) Create a 9 x 15 pixel image that is a copy of the area centered at the ground \*/

/\* truth location(center of letter) from the original image. \*/

/\* iv) Threshold this image at 128 to create a binary image. \*/

/\* v) Thin the thresholded image down to single-pixel wide components. \*/

/\* vi) Check all remaining pixels to determine if they are branch-points or endpoints. \*/

/\* vii) If there are not exactly 1 branch-point and 1 endpoint, do not further consider \*/

/\* this letter(it becomes “not detected”) \*/

/\* b) Count up the number of FP (letters detected that are not 'e') and TP (number of \*/

/\* letters detected that are 'e'). \*/

/\* c) Output the total TP and FP for each T. \*/

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

//printf("Step 2:\n");

thresholdImage = createImage(eCols \* eRows);

// Open to write to clear file

TPFPfpt = fopen("TPFP.txt", "w"); TPFPfpt == NULL ? (printf("Failed to open TPFP.txt.\n"), exit(0)) : TPFPfpt;

// Threshold values

for (i = 0; i <= T; i++, branch = end = TP = FP = found = isE = False)

{

// File rows from the GT

for (j = 0; j < fileRows; j++)

{

gtLetter = truth[j].letter; gtR = truth[j].y; gtC = truth[j].x;

// iii)

for (int r = -dr; r <= dr; r++)

{

for (int c = -dc; c <= dc; c++, index++)

{

if (msf\_eImage[(r + gtR) \* msfe\_COLS + (c + gtC)] > i) found = True;

thresholdImage[index] = sourceImage[(r + gtR) \* sourceCOLS + (c + gtC)];

}

}

// iv)

for (index = 0; index < eRows \* eCols; index++) thresholdImage[index] = thresholdImage[index] > 128 ? 0 : 255;

// v)

thin(thresholdImage);

// vi)

branchEndPoints(thresholdImage, &isE);

// b)

found ? (gtLetter == 'e' ? TP++ : FP++) : (isE ? (gtLetter == 'e' ? TP++ : TP) : FP);

}

// c)

fprintf(TPFPfpt, "%d %d %d\n", i, TP, FP);

}

fclose(TPFPfpt);

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

}

/// <summary>

/// This is a thinning algorithm designed based on both the lecture notes 'Edge properties' and the

/// Zhang-Suen thinning algorithm found at the link below.

/// https://rosettacode.org/wiki/Zhang-Suen\_thinning\_algorithm#C

///

/// This function uses a helper function edgeNonEdgetransitions which detects if there are any edge pixes

/// and returns them in the variables transitionCount, neighborCount and passNum4 for each row and column.

/// </summary>

/// <param name="srcImage">The source image which is a 9x15 which contains a letter to be thinned</param>

void thin(unsigned char\* srcImage)

{

const int rowE = 15, colE = 9;

const unsigned char ON = 255, OFF = 0;

int erasure = True, transitionCount = 0, neighborCount = 0, passNum4 = 0;

unsigned char\* copy = createImage(rowE \* colE);

//Lecture notes: Edge properties | Describes the Zhang-Suen thinning algorithm

// https://rosettacode.org/wiki/Zhang-Suen\_thinning\_algorithm#C

// P9 P2 P3 \_ A \_

// P8 P1 P4 C X B

// P7 P6 P5 \_ D \_

//1. Pass through the image looking at each pixel X.

do

{

for (int i = 0; i < rowE \* colE; i++) copy[i] = (unsigned char)0; // Reset the copy image to all black pixels

erasure = False; // Only repeat if we find a marked pixel

for (int r = 1; r < rowE; r++)

{

for (int c = 1; c < colE; c++, transitionCount = 0, neighborCount = 0)

{

//2. Count the number of edge->non-edge transitions in CW (or CCW) order around the pixel X (r,c)

// Only need to check pixes that are ON

if (srcImage[r \* colE + c] != OFF) edgeNonEdgeTransitions(srcImage, &transitionCount, &neighborCount, &passNum4, r, c);

//5. The edge pixel is marked for erasure if it has

// a) exactly 1 edge->non - edge transition,

// b) 2 <= edge neighbors <= 6, and

// c) passes item #4

if (transitionCount == 1 && neighborCount >= 3 && neighborCount <= 7 && passNum4 == 1)

{

// Mark pixel for erasure

copy[r \* colE + c] = ON;

erasure = True;

}

}

}

//6. Once all pixels have been scanned, erase those marked, and repeat

// (back to step 1) until no pixels are marked for erasure.

for (int index = 0; index < colE \* rowE; index++)

{

copy[index] == ON ? srcImage[index] = OFF : False;

}

} while (erasure == True);

// Free the copy image

free(copy);

return;

}

/// <summary>

/// Determining if the letter is 'e' if it has one end and one branch

/// </summary>

/// <param name="img">The image of the letter in which we are scanning</param>

/// <param name="isE">Logical bool which is either true if the letter is 'e' and false if it is not</param>

void branchEndPoints(unsigned char\* img, int \*isE)

{

const int rowE = 15, colE = 9;

const unsigned char ON = 255, OFF = 0;

int edge = 0, last = False, branch = 0, end = 0;

unsigned char N = OFF, E = OFF, S = OFF, W = OFF, NE = OFF, SE = OFF, SW = OFF, NW = OFF;

//Lecture notes: Edge properties

// P9 P2 P3 \_ A \_

// P8 P1 P4 C X B

// P7 P6 P5 \_ D \_

for (int r = 1; r < rowE; r++)

{

for (int c = 1; c < colE; c++, edge = 0)

{

//2. Count the number of branch and end points in CW (or CCW) order around the pixel X (r,c)

// Only need to check pixes that are ON

if (img[r \* colE + c] == ON)

{

// Checking in a clockwise rotation of the following image (r,c)

// NW N NE | (-1,-1) (-1,0) (-1,+1) | (-9,-1) (-9,0) (-9,+1)

// W X E | ( 0,-1) (0, 0) ( 0,+1) | ( 0,-1) (0, 0) ( 0,+1)

// SW S SE | (+1,-1) (+1,0) (+1,+1) | (+9,-1) (+9,0) (+9,+1)

// We'll check North -> North East -> East -> South East -> South -> South West -> West -> North West

// Moving up one row = -9 | Moving down one row = +9 | Moving left = -1 | Moving right = +1

N = img[(r - 1) \* colE + c]; NE = img[(r - 1) \* colE + (c + 1)];

E = img[r \* colE + (c + 1)]; SE = img[(r + 1) \* colE + (c + 1)];

S = img[(r + 1) \* colE + c]; SW = img[(r + 1) \* colE + (c - 1)];

W = img[r \* colE + (c - 1)]; NW = img[(r - 1) \* colE + (c - 1)];

//3. Count the number of edge neighbor pixels

// Check N edges

N == ON ? last = True : (last = False);

// Check NE edges

NE == ON ? last = True : ((last == True ? edge++ : edge), last = False);

// Check E edges

E == ON ? last = True : ((last == True ? edge++ : edge), last = False);

// Check SE edges

SE == ON ? last = True : ((last == True ? edge++ : edge), last = False);

// Check S edges

S == ON ? last = True : ((last == True ? edge++ : edge), last = False);

// Check SW edges

SW == ON ? last = True : ((last == True ? edge++ : edge), last = False);

// Check W edges

W == ON ? last = True : ((last == True ? edge++ : edge), last = False);

// Check NW edges

NW == ON ? last = True : ((last == True ? edge++ : edge), last = False);

// Check NW -> N edges

N && last ? edge++ : edge;

// https://stackoverflow.com/questions/63938495/branch-points-of-the-skeleton

// Endpoint -- has exactly one edge->non-edge transition

// Branchpoint -- has more than two edge->non-edge transitions

edge == 1 ? end++ : end;

edge > 2 ? branch++ : branch;

}

}

}

// Part vii)

(end == 1 && branch == 1) ? \*isE = True : (\*isE = False);

return;

}

/// <summary>

/// Checking for edge->non-edge transitions in a clockwise motion centered at X which is r,c

/// </summary>

/// <param name="img">The 9 x 15 image we're looping over</param>

/// <param name="transitions">The number of Edge to non-edge transitions</param>

/// <param name="neighbors">Count of neighboring pixels (on)</param>

/// <param name="r"></param>

/// <param name="c"></param>

void edgeNonEdgeTransitions(unsigned char\* img, int \*transitions, int \*neighbors, int \*passNum4, int r, int c)

{

const int COLS = 9;

const unsigned char ON = 255, OFF = 0;

int last = False;

unsigned char N = OFF, E = OFF, S = OFF, W = OFF, NE = OFF, SE = OFF, SW = OFF, NW = OFF;

// Checking in a clockwise rotation of the following image (r,c)

// NW N NE | (-1,-1) (-1,0) (-1,+1) | (-9,-1) (-9,0) (-9,+1)

// W X E | ( 0,-1) (0, 0) ( 0,+1) | ( 0,-1) (0, 0) ( 0,+1)

// SW S SE | (+1,-1) (+1,0) (+1,+1) | (+9,-1) (+9,0) (+9,+1)

// We'll check North -> North East -> East -> South East -> South -> South West -> West -> North West

// Moving up one row = -9 | Moving down one row = +9 | Moving left = -1 | Moving right = +1

N = img[(r - 1) \* COLS + c]; NE = img[(r - 1) \* COLS + (c + 1)];

E = img[r \* COLS + (c + 1)]; SE = img[(r + 1) \* COLS + (c + 1)];

S = img[(r + 1) \* COLS + c]; SW = img[(r + 1) \* COLS + (c - 1)];

W = img[r \* COLS + (c - 1)]; NW = img[(r - 1) \* COLS + (c - 1)];

//3. Count the number of edge neighbor pixels

// Check N edges

N == ON ? ((\*neighbors)++, last = True) : (last = False);

// Check NE edges

NE == ON ? ((\*neighbors)++, last = True) : ((last == True ? (\*transitions)++ : \*transitions), last = False);

// Check E edges

E == ON ? ((\*neighbors)++, last = True) : ((last == True ? (\*transitions)++ : \*transitions), last = False);

// Check SE edges

SE == ON ? ((\*neighbors)++, last = True) : ((last == True ? (\*transitions)++ : \*transitions), last = False);

// Check S edges

S == ON ? ((\*neighbors)++, last = True) : ((last == True ? (\*transitions)++ : \*transitions), last = False);

// Check SW edges

SW == ON ? ((\*neighbors)++, last = True) : ((last == True ? (\*transitions)++ : \*transitions), last = False);

// Check W edges

W == ON ? ((\*neighbors)++, last = True) : ((last == True ? (\*transitions)++ : \*transitions), last = False);

// Check NW edges

NW == ON ? ((\*neighbors)++, last = True) : ((last == True ? (\*transitions)++ : \*transitions), last = False);

// Check NW -> N edges

(N && last) ? (\*transitions)++ : (\*transitions);

//4. Check that at least one of the North, East, or (West and South) are not edge pixels

// A or B or (C and D) != edge

(N == OFF || E == OFF || (S == OFF && W == OFF)) ? (\*passNum4 = True) : (\*passNum4 = False);

return;

}

/// <summary>

/// The readImage function is designed to take a file name as the source and reads all of the data into a new image.

/// </summary>

/// <param name="ROWS"> Number of rows in the source image </param>

/// <param name="COLS"> Number of columns in the source image </param>

/// <param name="source"> File name that we're needing to open and read data from </param>

/// <returns> The function returns an array of values which makes up our image </returns>

unsigned char\* readImage(int\* ROWS, int\* COLS, char\* source)

{

int BYTES, readHeaderReturn;

static char header[80];

// Open image for reading

FILE \*fpt = fopen(source, "rb");

if (fpt == NULL) {

printf("Failed to open file (%s) for reading.\n", source);

exit(0);

}

/\* read image header (simple 8-bit greyscale PPM only) \*/

if (fscanf(fpt, "%s %d %d %d\n", header, &\*COLS, &\*ROWS, &BYTES) != 4 || strcmp(header, "P5") != 0 || BYTES != 255)

{

fclose(fpt);

printf("Image header corrupted.\n");

exit(0);

}

unsigned char\* destination = createImage((\*ROWS)\*(\*COLS)); // Create an empty image that is large enough for ROWS x COLS bytes

fread(destination, 1, (\*ROWS) \* (\*COLS), fpt);

fclose(fpt);

return destination;

}

/// <summary>

/// createImage allocates memory for our image array.

/// </summary>

/// <param name="size"> Number of bytes that are needing to be allocated for our image </param>

/// <returns> An array with 'size' number of bytes allocated for our image use</returns>

unsigned char\* createImage(int size)

{

unsigned char\* newImage = (unsigned char\*)calloc(size, sizeof(unsigned char));

if (newImage == NULL) {

printf("Unable to allocate %d bytes of memory.\n", size);

exit(0);

}

return newImage;

}