PURDUE UNIVERSITY

ECE 637 Laboratory Exercise 3
Neighborhoods and Connected Components

Tong Shen

February 2, 2017

1 AREA FILL WITH DIFFERENT THRESHOLD VALUES

In this section, we will learn the method to fill in an area of connected pixels in an image. To do this, we need to compute the set of all pixels which are connected to a specified pixel s.

1.1 GRAY SCALE IMAGE *img22gd2.tif*

As it shows in Figure 1.1

Figure 1.1: *img22gd2.tif*

1.2 Area Fill with Different Thresholds

In this section, we will calculate the connect sets in the pixel(65,47) with different thresholds of 1,2 and 3. The results are displayed in the following figures.

(a) Threshold = 1

(b) Threshold = 2

(c) Threshold = 3

Figure 1.2: Area Fill with Different Thresholds

According to the figures, with the increase of threshold, the connect set of pixel(65,47) becomes larger.

1.3 Code List

1.3.1 AreaFill.c

```
1
3  #include "ConnectSET.h"
5  void error(char *name);
7  int main(int argc, char **argv)
   {
9
   FILE *fp;
11 struct TIFF_img input_img, output_img;
   unsigned char **img;
```

```
unsigned int **seg;
int i, j;
int ClassLabel = 1;
int numCpixels;
double T;
struct pixel s;


if (argc != 5) error(argv[0]);

/* open image file */
if ((fp = fopen(argv[1], "rb")) == NULL) {
fprintf(stderr, "cannot open file %s\n", argv[1]);
exit(1);
}

/* read image */
if (read_TIFF(fp, &input_img)) {
fprintf(stderr, "error reading file %s\n", argv[1]);
exit(1);
}

/* close image file */
fclose(fp);

/* check the type of image data */
if (input_img.TIFF_type != 'g') {
fprintf(stderr, "error: must be a grayscale image\n");
exit(1);
}

/*get all parameters from input arguments*/
s.x = strtod(argv[2], NULL);
s.y = strtod(argv[3], NULL);
T = strtod(argv[4], NULL);

/*s.x = (int)argv[2];
s.y = (int)argv[3];
T = (int)argv[4];*/

/*sscanf(argv[2], "%d", &(s.x));
sscanf(argv[3], "%d", &(s.y));
sscanf(argv[4], "%lf", &T);*/



/* Allocate image of double precision floats */
seg = (unsigned int **)get_img(input_img.width, input_img.height, sizeof(unsigned int));
img = (unsigned char **)get_img(input_img.width, input_img.height, sizeof(unsigned char)
    );

/* copy img to double array */
for (i = 0; i < input_img.height; i++)
for (j = 0; j < input_img.width; j++) {
```

```
img[i][j] = (unsigned char)input_img.mono[i][j];
}

/*get the connect area using given T, and pixel location*/
connectSet(s, T, img, input_img.width, input_img.height, ClassLabel, seg, &numCpixels);



/* set up structure for output achromatic image */
/* to allocate a full color image use type 'c' */
get_TIFF(&output_img, input_img.height, input_img.width, 'g');

/*copy the result to output_img*/
for (i = 0; i < input_img.height; i++) {
for (j = 0; j < input_img.width; j++) {
if (seg[i][j] == ClassLabel) {
output_img.mono[i][j] = 0;
}
else
{
output_img.mono[i][j] = 255;
}
}
}

/* open output image file */
if ((fp = fopen("output.tif", "wb")) == NULL) {
fprintf(stderr, "cannot open file green.tif\n");
exit(1);
}

/* write output image */
if (write_TIFF(fp, &output_img)) {
fprintf(stderr, "error writing TIFF file %s\n", argv[2]);
exit(1);
}

/* close output image file */
fclose(fp);

/* de-allocate space which was used for the images */
free_TIFF(&(input_img));

free_img((void**)img);

return(0);
}
void error(char *name)
{
printf("usage:  %s  image.tiff \n\n", name);
exit(1);
}
```

1.3.2 CONNECTSET.C

```
#include "ConnectSET.h"


void connectNeighbors(
struct pixel s,
double T,
unsigned char **img,
int width,
int height,
int *M,
struct pixel c[4]) {
*M = 0;
if ((s.x - 1 >= 0) && abs(img[s.x][s.y] - img[s.x - 1][s.y]) <= T) {
c[*M].x = s.x - 1;
c[*M].y = s.y;
(*M)++;
}
if ((s.x + 1 < height) && abs(img[s.x][s.y] - img[s.x + 1][s.y]) <= T) {
c[*M].x = s.x + 1;
c[*M].y = s.y;
(*M)++;
}
if ((s.y - 1 >= 0) && abs(img[s.x][s.y] - img[s.x][s.y - 1]) <= T) {
c[*M].x = s.x;
c[*M].y = s.y - 1;
(*M)++;
}
if ((s.y + 1 < width) && abs(img[s.x][s.y] - img[s.x][s.y + 1]) <= T) {
c[*M].x = s.x;
c[*M].y = s.y + 1;
(*M)++;
}
}

void connectSet(
struct pixel s,
double T,
unsigned char **img,
int width,
int height,
int ClassLabel,
unsigned int **seg,
int *numCpixels) {
*numCpixels = 0;
struct pixel c[4];
struct Node *head, *temp, *last,*temp2;
int M;                                                    //futher modification.
head = (struct Node *)malloc(sizeof(struct Node));
head->current.x = s.x;
head->current.y = s.y;
head->next = NULL;
```

```
last = head;
int WLpixel;
WLpixel = 1;
while(WLpixel != 0){
if (seg[head->current.x][head->current.y] != ClassLabel) {
seg[head->current.x][head->current.y] = ClassLabel;
(*numCpixels)++;
connectNeighbors(head->current, T, img, width, height, &M, c);
for (int i = 0; i < M; i++) {
if (seg[c[i].x][c[i].y] != ClassLabel) {
// here use queue data structure to store the pixels waiting for processing
temp = (struct Node *)malloc(sizeof(struct Node));
temp->current.x = c[i].x;
temp->current.y = c[i].y;
temp->next = NULL;     //enqueue
WLpixel++;           //keep the number of the unprocessed pixels

last->next = temp;
last = temp;
}
}
}

// After processing, Dequeue
temp2 = head->next;
free(head);
head = temp2;
WLpixel--;
}
}
```

1.3.3 CONNECTSET.H

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <math.h>

#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

struct pixel {
int x;
int y;
};
struct Node{
```

```
18 struct pixel current;
   struct Node *next;
20 };
   void connectNeighbors(
22 struct pixel s,
   double T,
24 unsigned char **img,
   int width,
26 int height,
   int *M,
28 struct pixel c[4]);
   void connectSet(
30 struct pixel s,
   double T,
32 unsigned char **img,
   int width,
34 int height,
   int ClassLabel,
36 unsigned int **seg,
   int *numCpixels);
```

2 IMAGE SEGMENTATION WITH DIFFERENT THRESHOLDS

In this section, we will come up with a subroutine to get connect sets in an image. First of all, we need to find all the connect sets in an image with different thresholds. And then, we will get connect sets with pixels more than 100 and represent them with a sequence of numbers.

2.1 PLOT OF IMAGE SEGMENTATION

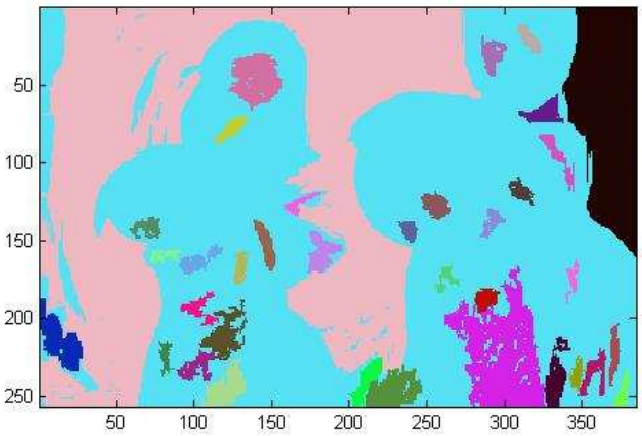

Figure 2.1: Image Segmentation(T = 1)

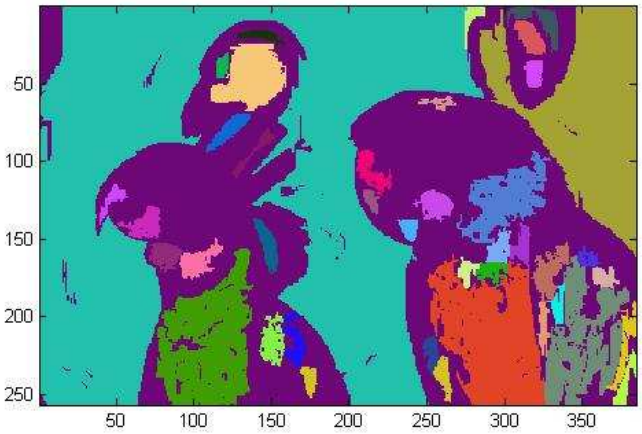

Figure 2.2: Image Segmentation(T = 2)

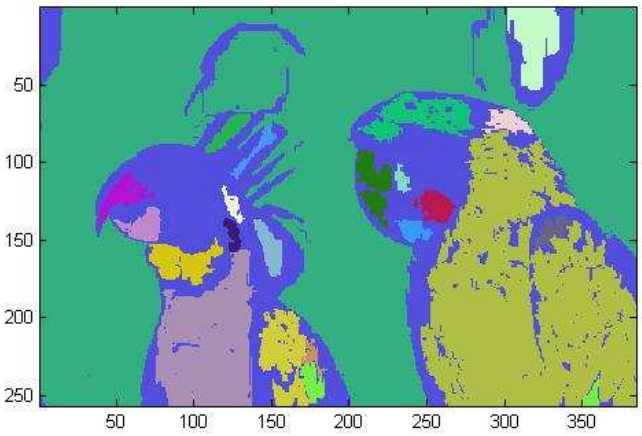

Figure 2.3: Image Segmentation(T = 3)

2.2 List of regions of the Image with Different T

	Number of Regions
T = 1	36
T = 2	41
T = 3	23

Table 2.1: List of Region of Image with different T

2.3 C code List

```
2
  #include "ConnectSET.h"
4
  void error(char *name);
6
  int main(int argc, char **argv)
8 {

10 FILE *fp;
  struct TIFF_img input_img, output_img;
12 unsigned char **img;
  unsigned int **segment;
14 int i, j;
  int ClassLabel = 1;
16 int numCpixels;
  double T;
18 struct pixel s;
```

```
20
22 if (argc != 3) error(argv[0]);
24 /* open image file */
   if ((fp = fopen(argv[1], "rb")) == NULL) {
26 fprintf(stderr, "cannot open file %s\n", argv[1]);
   exit(1);
28 }
30 /* read image */
   if (read_TIFF(fp, &input_img)) {
32 fprintf(stderr, "error reading file %s\n", argv[1]);
   exit(1);
34 }
36 /* close image file */
   fclose(fp);
38
   /* check the type of image data */
40 if (input_img.TIFF_type != 'g') {
   fprintf(stderr, "error: must be a grayscale image\n");
42 exit(1);
   }
44
   /*get all parameters from input arguments*/
46
   T = strtod(argv[2], NULL);
48
   /* Allocate image of double precision floats */
50 segment = (unsigned int **)get_img(input_img.width, input_img.height, sizeof(unsigned
       int));
   img = (unsigned char **)get_img(input_img.width, input_img.height, sizeof(unsigned char)
       );
52
   /* copy img to double array */
54 for (i = 0; i < input_img.height; i++)
   for (j = 0; j < input_img.width; j++) {
56 img[i][j] = (unsigned char)input_img.mono[i][j];
   }
58
   /*get the connect area of all pixels who do not belong to a connect set*/
60 for (i = 0; i < input_img.height; i++)
   for (j = 0; j < input_img.width; j++) {
62 if (segment[i][j] == 0)   // if already in a connect set, skip
   {
64 s.x = i;
   s.y = j;
66 connectSet(s, T, img, input_img.width, input_img.height, ClassLabel, segment, &
       numCpixels);
   if (numCpixels > 100) {   //  if only more than 100 pixels in a connectset, add
       ClassLabel.
68 ClassLabel++;
```

```
printf("%d\n", numCpixels);
}
else {
connectSet(s, T, img, input_img.width, input_img.height, 0, segment, &numCpixels);
// abanbon current connect set if num of pixel less than 100
}
}
}
ClassLabel--;
printf("%d\n", ClassLabel);

/* set up structure for output achromatic image */
/* to allocate a full color image use type 'c' */
get_TIFF(&output_img, input_img.height, input_img.width, 'g');

/*copy the result to output_img*/
for (i = 0; i < input_img.height; i++) {
for (j = 0; j < input_img.width; j++) {
output_img.mono[i][j] = segment[i][j];
}
}

/* open output image file */
if ((fp = fopen("output.tif", "wb")) == NULL) {
fprintf(stderr, "cannot open file green.tif\n");
exit(1);
}

/* write output image */
if (write_TIFF(fp, &output_img)) {
fprintf(stderr, "error writing TIFF file %s\n", argv[2]);
exit(1);
}

/* close output image file */
fclose(fp);

/* de-allocate space which was used for the images */
free_TIFF(&(input_img));

free_img((void**)img);

return(0);
}
void error(char *name)
{
printf("usage: %s image.tiff \n\n", name);
exit(1);
}
```