ECE 637 Digital Image Processing Laboratory: Neighborhoods and Connected Components

Yang Wang

February 2, 2016

1 Area Fill

In this section, a C program that fills in an area of connected pixels in a image is written. C subroutine to find the connected neighbors of a specfic pixel and to find all the pixels connected to a specfic pixel is implemented.

1.1 Plot image im22gd2.tif

Figure 1: Original img22gd2.tif

1.2 Plot image showing the connected set for $s = (67, 45)$, and $T = 2$

Figure 2: $s = (67, 45)$, and $T = 2$

1.3 Plot image showing the connected set for $s = (67, 45)$, and $T = 1$

As we can see, when T gets smaller, there is less connected set.

Figure 3: $s = (67, 45)$, and $T = 1$

1.4 Plot image showing the connected set for $s = (67, 45)$, and $T = 3$

As we can see, when T gets bigger, there are more connected sets.

Figure 4: $s = (67, 45)$, and $T = 3$

1.5 Code listing

1.5.1 areafill.c

```
#include "defs.h"

int main (int argc, char **argv) {
    FILE *fp;
    struct TIFF_img input_img;
    struct pixel s;
    int i, j, numcon;
    int ClassLabel = 1;
    double T;

    if (argc != 6) 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:  image must be grayscale\n");
    exit(1);
}

/* get the other parameters */
sscanf(argv[2], "%d", &(s.m));
sscanf(argv[3], "%d", &(s.n));
sscanf(argv[4], "%lf", &T);

unsigned int **seg = (unsigned int
↪  **)get_img(input_img.width,
                     input_img.height, sizeof(unsigned
                     ↪  int));

ConnectedSet(s, T, input_img.mono, input_img.width,
↪  input_img.height,
             ClassLabel, seg, &numcon);

for (i = 0; i < input_img.height; i++) {
    for (j = 0; j < input_img.width; j++) {
        if (seg[i][j] == ClassLabel) {
            input_img.mono[i][j] = 0;
        } else {
            input_img.mono[i][j] = 255;
        }
    }
}

free_img((void *)seg);

/* open output image file */
if ((fp = fopen(argv[5], "wb")) == NULL) {
    fprintf(stderr, "cannot open file output.tif\n");
    exit(1);
}

/* write output image */
if(write_TIFF(fp, &input_img)) {
```

```
        fprintf(stderr, "error writing TIFF file %s\n", argv[5]);
        exit(1);
    }

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

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

    return(0);
}
```

1.5.2 defs.c

Note: Section 2 uses the same subroutines written in this code listing.

```
#include "defs.h"

void ConnectedNeighbors(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int *M,
    struct pixel c[4]);

void ConnectedSet(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int ClassLabel,
    unsigned int **seg,
    int *NumConPixels);

void ConnectedNeighbors(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int *M,
    struct pixel c[4]) {
```

```
    *M = 0;

    if ((s.n-1) >= 0 && abs(img[s.m][s.n] - img[s.m][s.n-1]) <=
    ↪  T) {
        c[*M].m = s.m;
        c[*M].n = s.n - 1;
        (*M)++;
    }
    if ((s.n+1) < width && abs(img[s.m][s.n] - img[s.m][s.n+1])
    ↪  <= T) {
        c[*M].m = s.m;
        c[*M].n = s.n + 1;
        (*M)++;
    }
    if ((s.m-1) >= 0 && abs(img[s.m][s.n] - img[s.m-1][s.n]) <=
    ↪  T) {
        c[*M].m = s.m - 1;
        c[*M].n = s.n;
        (*M)++;
    }
    if ((s.m+1) < height && abs(img[s.m][s.n] - img[s.m+1][s.n])
    ↪  <= T) {
        c[*M].m = s.m + 1;
        c[*M].n = s.n;
        (*M)++;
    }
}

void ConnectedSet(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int ClassLabel,
    unsigned int **seg,
    int *NumConPixels) {
    struct pixelList *head, *tail, *tmp;
    struct pixel c[4];
    int M, i;

    (*NumConPixels) = 0;
    head = (struct pixelList *)malloc(sizeof(struct pixelList));
    head->pixel.m = s.m;
    head->pixel.n = s.n;
    head->next = NULL;
```

```
    tail = head;

    /*************************************************
     * start with the chosen pixel
     * build a linked list of the pixel that is not yet
     * labeled, the next starting pixel will be the head
     * of the built linked list
     *************************************************/

    while (head != NULL) {
        if (seg[head->pixel.m][head->pixel.n] != ClassLabel) {
            seg[head->pixel.m][head->pixel.n] = ClassLabel;
            (*NumConPixels)++;
            // find the connected neighbors (not-yet labled)
            ConnectedNeighbors(head->pixel, T, img, width,
             ↪  height, &M, c);

            // go through the neighbors, build the linked list
            for (i = 0; i < M; i++) {
                if (seg[c[i].m][c[i].n] != ClassLabel) {
                    tmp = (struct pixelList
                     ↪  *)malloc(sizeof(struct pixelList));
                    tmp->pixel.m = c[i].m;
                    tmp->pixel.n = c[i].n;
                    tmp->next = NULL;

                    tail->next = tmp;
                    tail = tmp;
                }
            }
        }

        // new pixel will the linked list head
        tmp = head->next;
        free(head);
        head = tmp;
    }
}

void error(char *name)
{
    printf("Very useful error message.\n\n");
    exit(1);
}
```

1.5.3 defs.h

```
#ifndef _DEFS_H_
#define _DEFS_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"
#include "typeutil.h"

struct pixel {
    int m;
    int n;
};

struct pixelList {
    struct pixel pixel;
    struct pixelList *next;
};

void ConnectedNeighbors(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int *M,
    struct pixel c[4]);

void ConnectedSet(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int ClassLabel,
    unsigned int **seg,
    int *NumConPixels);
```

```
void error(char *name);

#endif /* _DEFS_H */
```

2 Image Segmentation

In this section, instead of using only one pixel to grow the connected sets, we index through the image to extract all the connected sets in the original image. Hence, an image segmentation is performed in this section.

2.1 Plot image segmentation for $T = 1$

Figure 5: Randomly colored segmentation for $T = 1$

2.2 Plot image segmentation for $T = 2$

Figure 6: Randomly colored segmentation for $T = 2$

2.3 Plot image segmentation for $T = 3$

Figure 7: Randomly colored segmentation for $T = 3$

2.4 List of the number of regions generated for each threshold

	Number of regions
$T = 1$	36
$T = 2$	41
$T = 3$	23

2.5 Code listing

2.5.1 segmen.c

```
#include "defs.h"

int main (int argc, char **argv) {
    FILE *fp;
    struct TIFF_img input_img;
    struct pixel s;
    int i, j, numcon, segLabel;
    double T;

    if (argc != 4) error(argv[0]);

    /* we have 1 segment at the beginning */
    segLabel = 1;

    /* 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:  image must be grayscale\n");
    exit(1);
}

/* get the other parameters */
sscanf(argv[2], "%lf", &T);

unsigned int **seg = (unsigned int
↪  **)get_img(input_img.width,

                                          ↪  input_img.height,
                                         sizeof(unsigned
                                         ↪  int));

/* make sure all elements are 0 */
for (i = 0; i < input_img.height; i++) {
    for (j = 0; j < input_img.width; j++) {
        seg[i][j] = 0;
    }
}

/* go through the image */
for (i = 0; i < input_img.height; i++) {
    for (j = 0; j < input_img.width; j++) {
        if (seg[i][j] == 0) {
            s.m = i;
            s.n = j;
            ConnectedSet(s, T, input_img.mono,
            ↪  input_img.width,
```

```
                                input_img.height, segLabel, seg,
                                ↪  &numcon);
                        if (numcon > 100) {
                            /* new label created after 100 connected sets
                            ↪  */
                            segLabel++;
                        } else {
                            /* otherwise, back to 0 */
                            ConnectedSet(s, T, input_img.mono,
                            ↪  input_img.width,
                                        input_img.height, 0, seg,
                                        ↪  &numcon);
                        }
                    }
                }
            }
            for (i = 0; i < input_img.height; i++) {
                for (j = 0; j < input_img.width; j++) {
                    input_img.mono[i][j] = seg[i][j];
                }
            }

            free_img((void *)seg);

            /* open output image file */
            if ((fp = fopen(argv[3], "wb")) == NULL) {
                fprintf(stderr, "cannot open file output.tif\n");
                exit(1);
            }

            /* write output image */
            if(write_TIFF(fp, &input_img)) {
                fprintf(stderr, "error writing TIFF file %s\n", argv[3]);
                exit(1);
            }

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

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

            return(0);
        }
```