ECE 63700 Laboratory:

Neighborhoods and Connected Components

Shang Hsuan (Sean) Lee

January 31, 2024

1 Area Fill

1.1 Input image img22gd2.tif

Figure 1: img22gd2.tif

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

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

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

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

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

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

1.5 C code Listing

1.5.1 AreaFill.c

```
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include "connectedNeighbors.h"

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, output_img;
```

```
char *input_file = "img22gd2.tif";

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

/* read image */
if ( read_TIFF ( fp, &input_img ) ) {
  fprintf ( stderr, "error reading file %s\n", input_file );
  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 24-bit color\n" );
  exit ( 1 );
}

/* set up structure for output grey image */
/* Note that the type is 'g' rather than 'c' */
get_TIFF ( &output_img, input_img.height, input_img.width, 'g' );

// allocate seg
unsigned int **seg = (unsigned int **)get_img(input_img.width, input_img.height
int numConPixels = 0;
struct pixel s = {.m = 67, .n = 45};
double T = 3;
int classLabel = 1;

// initialize seg
for (int i = 0; i < input_img.height; i++)
for (int j = 0; j < input_img.width; j++) {
  seg[i][j] = 0;
}

ConnectedSet(s, T, input_img.mono, input_img.width, input_img.height, classLabe

// convert to binary image
for (int i = 0; i < input_img.height; i++)
for (int 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 ( "connected_output_3.tif", "wb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file connected_output.tif\n");
    exit ( 1 );
  }

  /* write output image */
  if ( write_TIFF ( fp, &output_img ) ) {
    fprintf ( stderr, "error writing TIFF file connected_output.tif\n");
    exit ( 1 );
  }

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

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

  return(0);
}

void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then pass the input image through a low pass filter ,\n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'lpf_output.tiff '.\n");
    exit(1);
}
```

1.5.2 connectedNeighbors.c

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

struct pixel {
    int m, n; // m=row, n=col
    };

void ConnectedNeighbors(
    struct pixel s, // location of the pixel s
    double T, // threshold
    unsigned char **img,
```

```
    int width,
    int height,
    int *M, // pointer to the number of neighbors connected to s
    struct pixel c[4] // array containing the M connected neighbors to the pixel
    ) {
        unsigned char img_s = img[s.m][s.n];

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


void ConnectedSet(
    struct pixel s, // seed s
    double T, // threshold
    unsigned char **img,
    int width,
    int height,
    int ClassLabel, // integer value used to label any pixel which is connected t
    unsigned int **seg, // If a pixel is connected to s, then assign ClassLabel t
    int *NumConPixels //number of pixels which were found to be connected to s (M
    ) {
        *NumConPixels = 0; // Initialize the count of connected pixels

        // Initialize a list B
        struct pixel *B = malloc(width * height * sizeof(struct pixel));
        int B_size = 0;

        // Add s to B
        B[B_size++] = s;

        while (B_size > 0) {
```

```
            // Remove a pixel from B
            struct pixel current = B[--B_size];

            // If the current pixel is not yet labeled
            if (seg[current.m][current.n] == 0) {
                seg[current.m][current.n] = ClassLabel; // Label the current pixe
                (*NumConPixels)++; // Increase the count of connected pixels

                // Find connected neighbors
                int M = 0;
                struct pixel c[4];
                ConnectedNeighbors(current, T, img, width, height, &M, c);

                // Add connected and unlabeled neighbors to B
                for (int i = 0; i < M; i++) {
                    if (seg[c[i].m][c[i].n] == 0) {
                        B[B_size++] = c[i];
                    }
                }
            }
        }

        free(B); // Free the dynamic array used for B
    }
```

2 Image Segmentation

2.1 Image segmentation with $T = 1$, $T = 2$, and $T = 3$

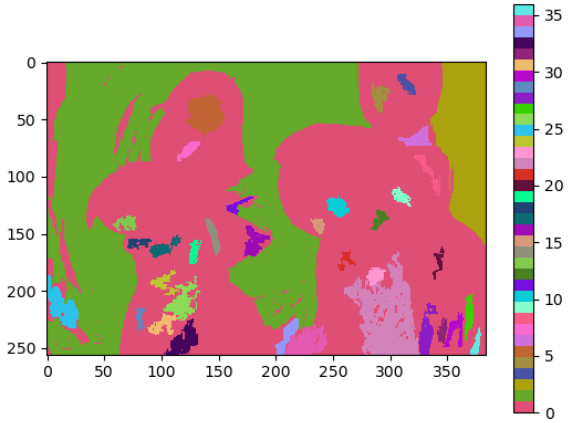

Figure 5: Randomly colored segmentation for $T = 1$

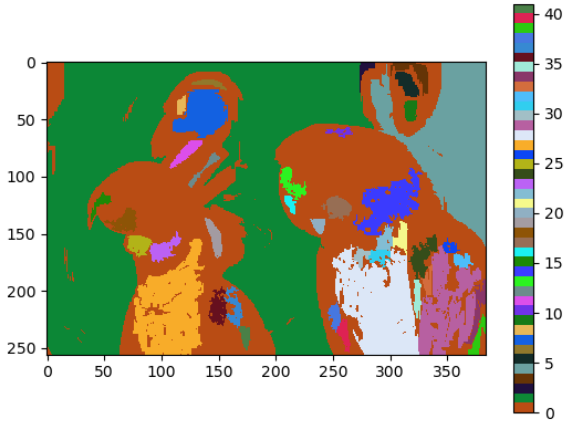

Figure 6: Randomly colored segmentation for $T = 2$

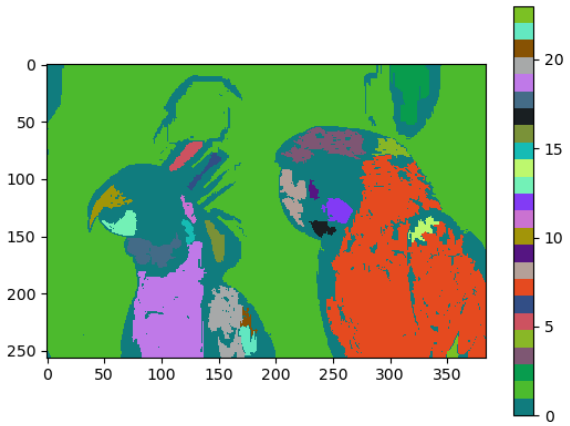

Figure 7: Randomly colored segmentation for $T = 3$

2.2 Number of regions generated with $T = 1$, $T = 2$, and $T = 3$

36 regions are generated with $T = 1$, 41 regions are generated with $T = 2$, and finally, 23 regions are generated with $T = 3$.

2.3 C code Listing

```
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include "connectedNeighbors.h"
```

```
void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, output_img;

  char *input_file = "img22gd2.tif";

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

  /* read image */
  if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", input_file );
    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 24-bit color\n" );
    exit ( 1 );
  }

  /* set up structure for output grey image */
  /* Note that the type is 'g' rather than 'c' */
  get_TIFF ( &output_img, input_img.height, input_img.width, 'g' );

  // allocate seg
  unsigned int **seg = (unsigned int **)get_img(input_img.width, input_img.height
  double T = 3;
  int classLabel = 1;

  // initialize seg
  for (int i=0; i<input_img.height; i++)
  for (int j=0; j<input_img.width; j++) {
    seg[i][j] = 0;
  }

  int numConPixels;
  for (int i=0; i<input_img.height; i++)
  for (int j=0; j<input_img.width; j++) {
    if (seg[i][j] == 0) {
        struct pixel s = {.m = i, .n = j};
```

```
        ConnectedSet(s, T, input_img.mono, input_img.width, input_img.height, cla
        // Increment label if more than 100 pixels connected
        if (numConPixels > 100) {
            classLabel ++;
        }
        else {
            // Reset label to 0 if not more than 100 pixels connected
            for (int k = 0; k < input_img.height; k++)
            for (int l = 0; l < input_img.width; l++){
                if (seg[k][l] == classLabel){
                    seg[k][l] = 0;
                }
            }

        }
    }
  }

  printf("Number of regions: %d\n", classLabel - 1);


  for (int i = 0; i < input_img.height; i++)
  for (int j = 0; j < input_img.width; j++) {
    output_img.mono[i][j] = seg[i][j];
  }


  /* open output image file */
  if ( ( fp = fopen ( "segmentation_3.tif", "wb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file segmentation.tif\n");
    exit ( 1 );
  }

  /* write output image */
  if ( write_TIFF ( fp, &output_img ) ) {
    fprintf ( stderr, "error writing TIFF file segmentation.tif\n");
    exit ( 1 );
  }

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

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

  return(0);
}
```

```
void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then pass the input image through a low pass filter ,\n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'lpf_output.tiff '.\n");
    exit(1);
}
```