ECE 63700 Laboratory:

Image Filtering

Shang Hsuan (Sean) Lee

January 19, 2024

1 C Programing

No reports needed.

2 Displaying and Exporting Images in Python

No reports needed.

3 FIR Low Pass Filter

In this problem, a simple low pass filter given by the 9×9 point spread function:

$$h(m,n) = \begin{cases} 1/81 & \text{for } |m| \leq 4 \text{ and } |n| \leq 4 \\ 0 & \text{otherwise} \end{cases}$$

3.1 A derivation of the analytical expression for $H(e^{j\mu}, e^{j\nu})$

By DSFT formula:

$$\begin{aligned} H(e^{j\mu}, e^{j\nu}) &= \sum_{n=-4}^{4} \sum_{m=-4}^{4} \frac{1}{81} e^{-j2\pi(m\mu+n\nu)} \\ &= \frac{1}{81} \sum_{n=-4}^{4} e^{-j2\pi n\nu} \sum_{m=-4}^{4} e^{-j2\pi m\mu} \end{aligned}$$

With the formula:

$$\sum_{k=a}^{b} ar^k = \frac{a(1-r^{b-a+1})}{1-r}, \text{ for } r \neq 1$$

Apply it to $H(e^{j\mu}, e^{j\nu})$:

$$H(e^{j\mu}, e^{j\nu}) = \frac{1}{81} * \frac{1-e^{-j2\pi\nu*9}}{1-e^{-j2\pi\nu}} * \frac{1-e^{-j2\pi\mu*9}}{1-e^{-j2\pi\mu}}$$

3.2 Use Python's matplotlib to plot the magnitude of the frequency response $|H(e^{j\mu}, e^{j\nu})|$

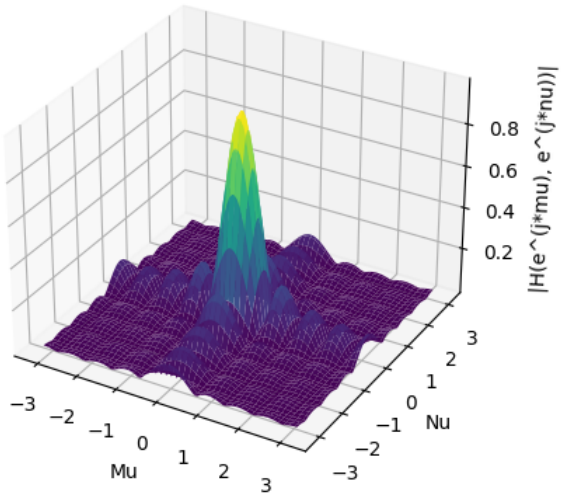

Figure 1: Magnitude of $H(e^{j\mu}, e^{j\nu})$

3.3 The original input image

The practice for this section will be running on this colored image

Figure 2: Input Image

3.4 The filtered color image

This is the output image of this practice

Figure 3: Filtered Output Image

3.5 C Code Listing

The following is the code I wrote for this part of the lab, the rest are the same as the given code example:

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

void error(char *name);

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

  if ( argc != 2 ) 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 != 'c' ) {
  fprintf ( stderr, "error:  image must be 24-bit color\n" );
  exit ( 1 );
}

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

/* Apply filter h(m, n) = 1/81*/
for ( i = 4; i < input_img.height - 4; i++)
for ( j = 4; j < input_img.width - 4; j++) {
  int red = 0;
  int green = 0;
  int blue  = 0;
  for (int n = -4; n <= 4; n++)
  for (int m = -4; m <= 4; m++) {
      red += input_img.color[0][i+m][j+n];
      green += input_img.color[1][i+m][j+n];
      blue += input_img.color[2][i+m][j+n];
  }

  output_img.color[0][i][j] = red / 81;
  output_img.color[1][i][j] = green / 81;
  output_img.color[2][i][j] = blue / 81;
}

/* open output image file */
if ( ( fp = fopen ( "lpf_output.tif", "wb" ) ) == NULL ) {
  fprintf ( stderr, "cannot open file lpf_output.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_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);
}
```

4 FIR Sharpening Filter

A low pass filter $h(m,n)$ is given as:

$$h(m,n) = \begin{cases} 1/25 & \text{for } |m| \leq 2 \text{ and } |n| \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

An unsharp mask filter $g(m,n)$ is then given by:

$$g(m,n) = \delta(m,n) + \lambda(\delta(m,n) - h(m,n))$$

4.1 A derivation of the analytical expression for $H(e^{j\mu}, e^{j\nu})$

By DSFT formula:

$$\begin{aligned} H(e^{j\mu}, e^{j\nu}) &= \sum_{n=-2}^{2} \sum_{m=-2}^{2} \frac{1}{25} e^{-j2\pi(m\mu+n\nu)} \\ &= \frac{1}{25} \sum_{n=-2}^{2} e^{-j2\pi n\nu} \sum_{m=-2}^{2} e^{-j2\pi m\mu} \end{aligned}$$

With the formula:

$$\sum_{k=a}^{b} ar^k = \frac{a(1-r^{b-a+1})}{1-r}, \text{ for } r \neq 1$$

Apply it to $H(e^{j\mu}, e^{j\nu})$:

$$H(e^{j\mu}, e^{j\nu}) = \frac{1}{25} * \frac{1-e^{-j2\pi\nu*5}}{1-e^{-j2\pi\nu}} * \frac{1-e^{-j2\pi\mu*5}}{1-e^{-j2\pi\mu}}$$

4.2 A derivation of the analytical expression for $G(e^{j\mu}, e^{j\nu})$

With the result in section 4.1, $G(e^{j\mu}, e^{j\nu})$ can be derived as:

$$\begin{aligned} G(e^{j\mu}, e^{j\nu}) &= 1 + \lambda(1 - H(e^{j\mu}, e^{j\nu})) \\ &= 1 + \lambda(1 - \frac{1}{25} * \frac{1 - e^{-j2\pi\nu*5}}{1 - e^{-j2\pi\nu}} * \frac{1 - e^{-j2\pi\mu*5}}{1 - e^{-j2\pi\mu}}) \end{aligned}$$

4.3 Use Python's matplotlib to plot the magnitude of the frequency response $|H(e^{j\mu}, e^{j\nu})|$

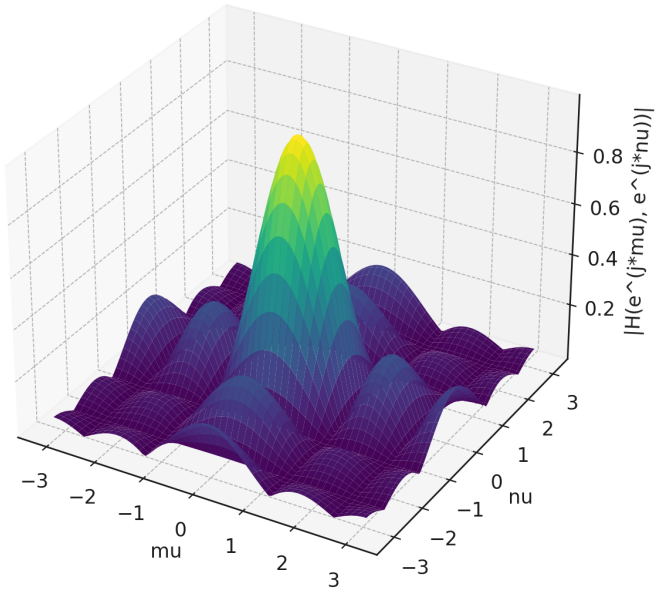

Figure 4: Magnitude of $H(e^{j\mu}, e^{j\nu})$

4.4 Use Python's matplotlib to plot $|G(e^{j\mu}, e^{j\nu})|$ for $\lambda = 1.5$

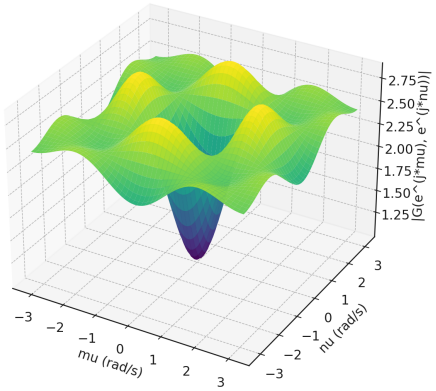

Figure 5: Magnitude of $G(e^{j\mu}, e^{j\nu})$

4.5 The original input image

The practice for this section will be running on this blurred image

Figure 6: Input Image

4.6 The sharpened color image

This is the output image of this practice

Figure 7: Sharpened Output Image, $\lambda = 1.5$

4.7 C Code Listing

The following is the code I wrote for this part of the lab, the rest are the same as the given code example:

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

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

void error(char *name);
double clip(double px);

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

  if ( argc != 2 ) 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 != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
  }

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

  /* Apply filter g(m, n) = 1 + 1.5(1 - 1/25x)*/
  for ( i = 2; i < input_img.height - 2; i++)
  for ( j = 2; j < input_img.width - 2; j++) {
    int red = 0;
    int green = 0;
    int blue  = 0;
    for (int n = -2; n <= 2; n++)
    for (int m = -2; m <= 2; m++) {
        red += input_img.color[0][i+m][j+n];
```

```
            green += input_img.color[1][i+m][j+n];
            blue += input_img.color[2][i+m][j+n];
        }

        int temp_red = input_img.color[0][i][j] + 1.5 * (input_img.color[0][i][j] - r
        int temp_green = input_img.color[1][i][j] + 1.5 * (input_img.color[1][i][j] -
        int temp_blue = input_img.color[2][i][j] + 1.5 * (input_img.color[2][i][j] -

        output_img.color[0][i][j] = clip(temp_red);
        output_img.color[1][i][j] = clip(temp_green);
        output_img.color[2][i][j] = clip(temp_blue);
    }

    /* open output image file */
    if ( ( fp = fopen ( "sf_output.tif", "wb" ) ) == NULL ) {
        fprintf ( stderr, "cannot open file sf_output.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_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 writes out the sherpened image\n");
    printf("with the name 'sf_output.tiff'.\n");
    exit(1);
}

double clip(double px)
{
    if (px > 255)  return 255;
    if (px < 0)  return 0;
    return px;
}
```

5 IIR Filter

Let $h(m,n)$ be the impulse response of an IIR filter with corresponding difference equation:

$$y(m,n) = 0.01x(m,n) + 0.9[y(m-1,n) + y(m,n-1)] - 0.81y(m-1,n-1)$$

5.1 A derivation of the analytical expression for $H(e^{j\mu}, e^{j\nu})$

To find $H(e^{j\mu}, e^{j\nu})$, first perform a z-transformation on $y(m,n)$:

$$Y(z_1,z_2) = 0.01X(z_1,z_2) + 0.9z_1^{-1}Y(z_1,z_2) + 0.9z_2^{-1}Y(z_1,z_2) - 0.81z_1^{-1}z_2^{-1}Y(z_1,z_2)$$

With $z_1 = e^{j\mu}$ and $z_2 = e^{j\nu}$, we can get $H(e^{j\mu}, e^{j\nu})$:

$$\begin{aligned} H(e^{j\mu}, e^{j\nu}) = H(z_1,z_2) &= \frac{Y(z_1,z_2)}{X(z_1,z_2)} \\ &= \frac{0.01}{1 - 0.9z_1^{-1} - 0.9z_2^{-1} - 0.81z_1^{-1}z_2 - 1} \\ &= \frac{0.01}{1 - 0.9e^{-j\mu} - 0.9e^{-j\nu} - 0.81e^{-j\mu}e^{-j\nu}} \end{aligned}$$

5.2 A plot of the magnitude of the frequency response $|H(e^{j\mu}, e^{j\nu})|$

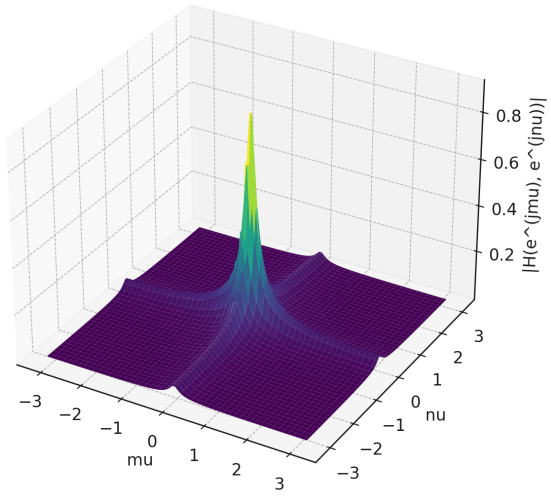

Figure 8: Magnitude of $H(e^{j\mu}, e^{j\nu})$

5.3 An image of the point spread function

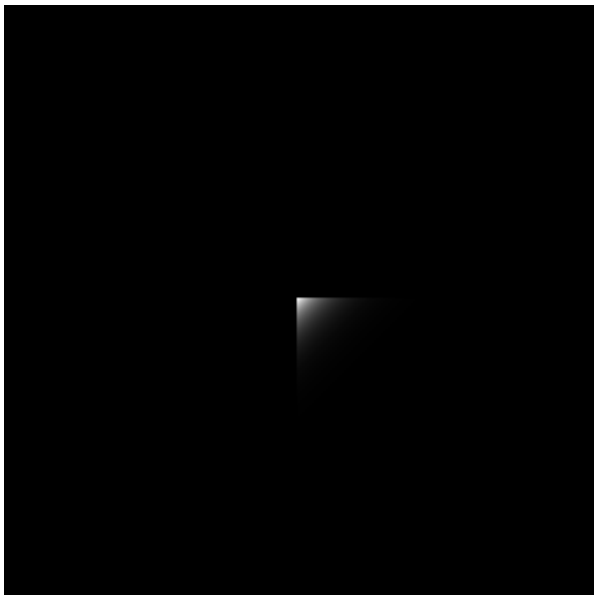

Figure 9: Point Spread Function

5.4 The filtered output color image

The output is generated by applying the IIR Filter on the same input image as section 3.

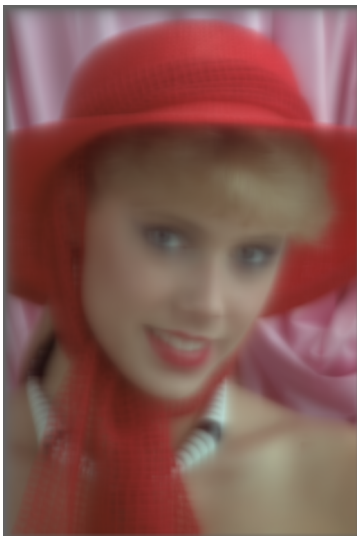

Figure 10: Filtered Output Image

5.5 C Code Listing

The following is the code I wrote for this part of the lab, the rest are the same as the given code example:

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

void error(char *name);
double clip(double px);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, output_img;
  double **red,**green, **blue;
  int i,j;

  if ( argc != 2 ) 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 != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
  }

  /* Allocate image of double precision floats */
  red = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  green = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  blue = (double **)get_img(input_img.width,input_img.height,sizeof(double));

  /* copy red component to double array */
  for ( i = 0; i < input_img.height; i++ )
  for ( j = 0; j < input_img.width; j++ ) {
    red[i][j] = input_img.color[0][i][j];
  }
```

```
/* copy green component to double array */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  green[i][j] = input_img.color[1][i][j];
}

/* copy blue component to double array */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  blue[i][j] = input_img.color[2][i][j];
}

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

/* Apply IIR filter starting at [1][1]*/
for ( i = 1; i < input_img.height; i++ )
for ( j = 1; j < input_img.width; j++ ) {
  red[i][j] = 0.01*input_img.color[0][i][j] + 0.9*red[i][j-1] + 0.9*red[i-1][j]
  green[i][j] = 0.01*input_img.color[1][i][j] + 0.9*green[i][j-1] + 0.9*green[i
  blue[i][j] = 0.01*input_img.color[2][i][j] + 0.9*blue[i][j-1] + 0.9*blue[i-1]
  }

for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  output_img.color[0][i][j] = clip(red[i][j]);
  output_img.color[1][i][j] = clip(green[i][j]);
  output_img.color[2][i][j] = clip(blue[i][j]);
}

/* open output image file */
if ( ( fp = fopen ( "iir_output.tif", "wb" ) ) == NULL ) {
  fprintf ( stderr, "cannot␣open␣file␣sf_output.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_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 writes out the sherpened image\n");
    printf("with the name 'sf_output.tiff'.\n");
    exit(1);
}

double clip(double px)
{
  if (px > 255)  return 255;
  if (px < 0)  return 0;
  return px;
}
```