ECE 637 Lab 1 Report

Image Filtering

Xihui Wang

Section 3 - FIR Low Pass Filter:

3.1 A derivation of the analytical expression for H(e^{ju}, e^{jv}).

We already know that

$$h(m,n) = \begin{cases} \frac{1}{81} & for\ |m| \le 4, |n| \le 4 \\ 0 & otherwise \end{cases}$$

Then by discrete space Fourier Transform(DSFT), we could get

$$H(e^{j\mu}, e^{j\upsilon}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m,n) e^{-j(\mu m + \upsilon n)}$$

$$= \sum_{n=-4}^{4} \sum_{m=-4}^{4} \frac{1}{81} e^{-j(\mu m + \upsilon n)}$$

$$= \frac{1}{81} \frac{\sin\left(\frac{9}{2}\mu\right) \sin\left(\frac{9}{2}\upsilon\right)}{\sin\left(\frac{1}{2}\mu\right) \sin\left(\frac{1}{2}\upsilon\right)}$$

3.2 A plot of $\left|H\left(e^{j\mu}, e^{jv}\right)\right|$.

Plotting the magnitude of $H(e^{j\mu}, e^{jv})$ using Matlab, and then we could get a graph as shown in Fig 3-2.

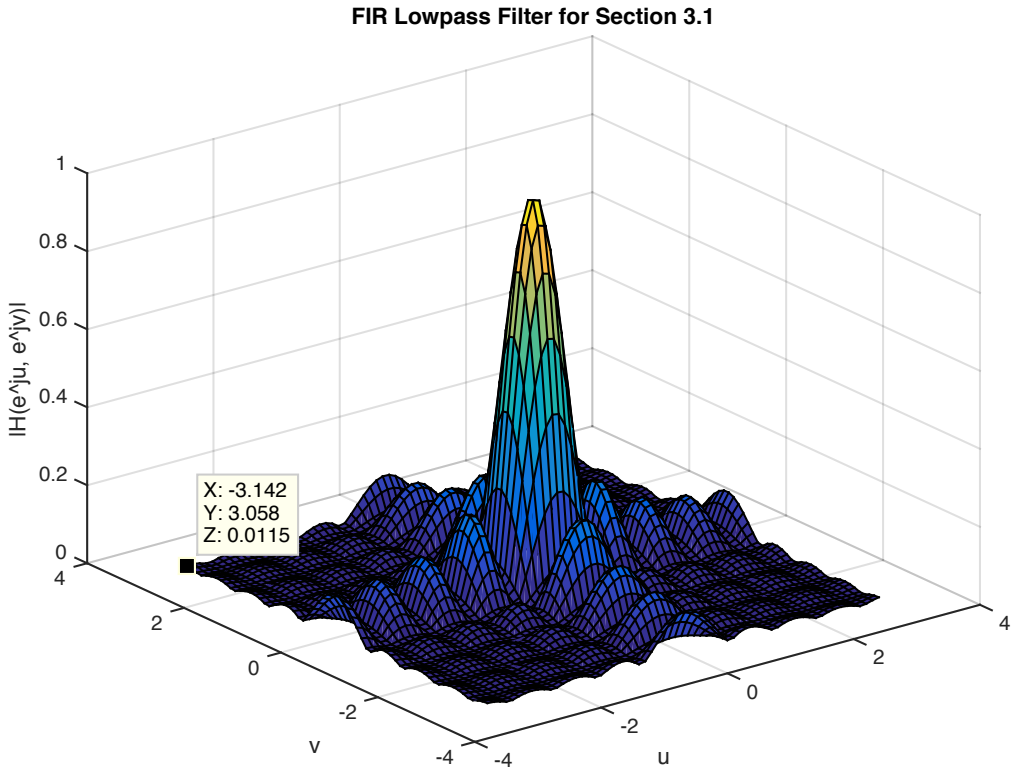

Fig 3-2 Magnitude of the Frequency Response over the region $[-\pi, \pi] \times [-\pi, \pi]$

3.3 The color image in img03.tif.

Fig 3-3 The color image in img03.tif

3.4 The filtered color image.

Fig 3-4 The filtered color image

3.5 A listing of the C code.

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

void error(char *name);
double clipping(double pixel);


int main (int argc, char **argv)
{
   FILE *fp;
   struct TIFF_img input_img, green_img, color_img;
   double **img1,**img2,**img3; // red, green, blue
   int32_t i,j,k,l;

   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 */
  img1 = (double
**)get_img(input_img.width+8,input_img.height+8,sizeof(double));//red
  img2 = (double
**)get_img(input_img.width+8,input_img.height+8,sizeof(double));//gre
en
  img3 = (double
**)get_img(input_img.width+8,input_img.height+8,sizeof(double));//blu
e

  /* copy green component to double array */
  for(i = 0;i<input_img.height+8;i++)
  for(j = 0;j<input_img.width+8;j++){
      img1[i][j] =
(((i>3)&&(i<input_img.height+4))&&((j>3)&&(j<input_img.width+4)))?
input_img.color[0][i-4][j-4]:0;
      img2[i][j] =
(((i>3)&&(i<input_img.height+4))&&((j>3)&&(j<input_img.width+4)))?
```

```
input_img.color[1][i-4][j-4]:0;
          img3[i][j] =
(((i>3)&&(i<input_img.height+4))&&((j>3)&&(j<input_img.width+4)))?
input_img.color[2][i-4][j-4]:0;
   }



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

   double r, g, b;
   double h = 1.0/81; //for -4 to 4, m&n

   for ( i = 0; i < input_img.height; i++ )
   for ( j = 0; j < input_img.width; j++ ) {
      r = 0.0;
      g = 0.0;
      b = 0.0;

      //-4 to 4
      for (k = -4; k <5; k++ )
      for ( l = -4; l < 5; l++ ){
         //take the sum
         //h=1/81
         r = r + h*img1[i-k+4][j-l+4];
         g = g + h*img2[i-k+4][j-l+4];
         b = b + h*img3[i-k+4][j-l+4];
         }
```

```
    //y(m,n)
    color_img.color[0][i][j] = clipping(r);
    color_img.color[1][i][j] = clipping(g);
    color_img.color[2][i][j] = clipping(b);
    }


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

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

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

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

free_img( (void**)img1 );
free_img( (void**)img2 );
free_img( (void**)img3 );
```

```
    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 horizontally filters the green component, adds
noise,\n");
        printf("and writes out the result as an 8-bit image\n");
        printf("with the name 'green.tiff'.\n");
        printf("It also generates an 8-bit color image,\n");
        printf("that swaps red and green components from the input
image");
        exit(1);
}

double clipping(double pixel)
{
    if (pixel > 255)    return 255;
    if (pixel < 0)    return 0;
    return pixel;
}
```

Section 4 - FIR Sharpening Filter:

4.1 A derivation of the analytical expression for H(e^{ju}, e^{jv}).

$$h(m,n) = \begin{cases} \frac{1}{25} & for\ |m| \le 2, |n| \le 2 \\ 0 & otherwise \end{cases}$$

Then by discrete space Fourier Transform(DSFT), we could get

$$H(e^{j\mu}, e^{j\upsilon}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m,n) e^{-j(\mu m + \upsilon n)}$$

$$= \sum_{n=-2}^{2} \sum_{m=-2}^{2} \frac{1}{25} e^{-j(\mu m + \upsilon n)}$$

$$= \frac{1}{25} \frac{\sin\left(\frac{5}{2}\mu\right) \sin\left(\frac{5}{2}\upsilon\right)}{\sin\left(\frac{1}{2}\mu\right) \sin\left(\frac{1}{2}\upsilon\right)}$$

4.2 A derivation of the analytical expression for G(e^{ju}, e^{jv}).

Since $g(m,n) = \delta(m,n) + \lambda\big(\delta(m,n) - h(m,n)\big)$, $\mathcal{F}\big(\delta(m,n)\big) = 1$, and it is an linear equation. Therefore, we can conclude that:

$$G(e^{j\mu}, e^{j\upsilon}) = 1 + \lambda\Big(1 - H(e^{j\mu}, e^{j\upsilon})\Big) = 1 + \lambda(1 - \frac{1}{25} \frac{\sin\left(\frac{5}{2}\mu\right) \sin\left(\frac{5}{2}\upsilon\right)}{\sin\left(\frac{1}{2}\mu\right) \sin\left(\frac{1}{2}\upsilon\right)})$$

4.3 A plot of |H(e^{ju}, e^{jv})|.

Plotting the magnitude of $H(e^{j\mu}, e^{j\upsilon})$ using Matlab, and then we could get a graph as shown in Fig 4-3.

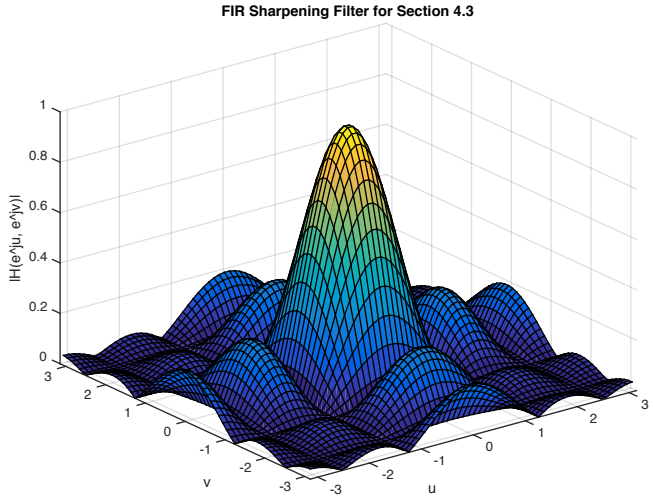

Fig 4-3 Magnitude of the Frequency Response over the region $[-\pi, \pi] \times [-\pi, \pi]$

4.4 A plot of |G(e^{ju}, e^{jv})| for $\lambda = 1.5$.

Plotting the magnitude of $G(e^{j\mu}, e^{j\upsilon})$ using Matlab, and then we could get a graph as shown in Fig 4-4.

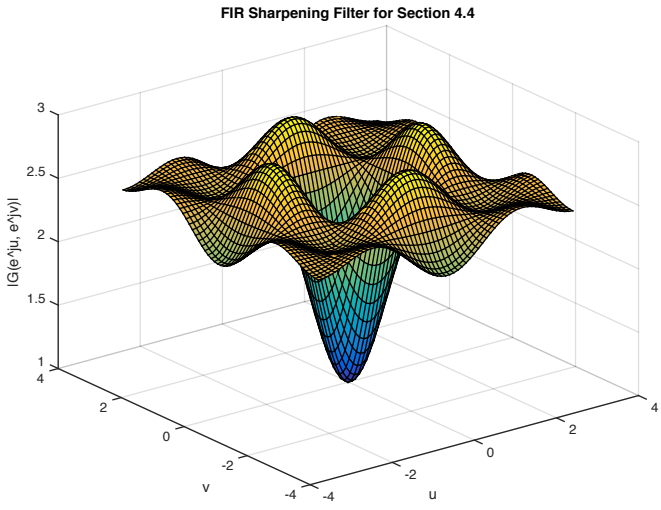

Fig 4-4 Magnitude of G(e^{ju}, e^{jv}) for $\lambda = 1.5$ over the region $[-\pi, \pi] \times [-\pi, \pi]$

4.5 The input color image imgblur.tif.

Fig 4-5 The input color image imgblur.tif

4.6 The output sharpened color image for $\lambda = 1.5$.

Fig 4-6 The output sharpened color image for $\lambda = 1.5$

4.7 A listing of the C code.

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

void error(char *name);
double clipping(double pixel);


int main (int argc, char **argv)
{
   FILE *fp;
   struct TIFF_img input_img, green_img, color_img;
   double **img1,**img2,**img3; // red, green, blue
   int32_t i,j,k,l;
   double lambda = 1.5;
   double h = 1.0/25; //for -4 to 4, m&n


   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 */
   img1 = (double
**)get_img(input_img.width+4,input_img.height+4,sizeof(double));//red
   img2 = (double
**)get_img(input_img.width+4,input_img.height+4,sizeof(double));//gre
en
   img3 = (double
**)get_img(input_img.width+4,input_img.height+4,sizeof(double));//blu
e

   /* copy green component to double array */
   for(i = 0;i<input_img.height+4;i++)
   for(j = 0;j<input_img.width+4;j++){
         img1[i][j] =
(((i>1)&&(i<input_img.height+2))&&((j>1)&&(j<input_img.width+2)))?
```

```
input_img.color[0][i-2][j-2]:0;
          img2[i][j] =
(((i>1)&&(i<input_img.height+2))&&((j>1)&&(j<input_img.width+2)))?
input_img.color[1][i-2][j-2]:0;
          img3[i][j] =
(((i>1)&&(i<input_img.height+2))&&((j>1)&&(j<input_img.width+2)))?
input_img.color[2][i-2][j-2]:0;
   }

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

   double r, g, b;

   for ( i = 0; i < input_img.height; i++ )
   for ( j = 0; j < input_img.width; j++ ) {
      r = 0.0;
      g = 0.0;
      b = 0.0;

      //-4 to 4
      for (k = -2; k < 3; k++ )
      for (l = -2; l < 3; l++ ){
         //take the sum
         r = r + h*img1[i-k+2][j-l+2];
         g = g + h*img2[i-k+2][j-l+2];
         b = b + h*img3[i-k+2][j-l+2];
         }
```

```
    r = ((lambda+1)*img1[i+2][j+2] - lambda*r);
    g = ((lambda+1)*img2[i+2][j+2] - lambda*g);
    b = ((lambda+1)*img3[i+2][j+2] - lambda*b);

    //y(m,n)
    color_img.color[0][i][j] = clipping(r);
    color_img.color[1][i][j] = clipping(g);
    color_img.color[2][i][j] = clipping(b);
    }



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

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

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

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

```
    free_img( (void**)img1 );
    free_img( (void**)img2 );
    free_img( (void**)img3 );

    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 horizontally filters the green component, adds
noise,\n");
        printf("and writes out the result as an 8-bit image\n");
        printf("with the name 'green.tiff'.\n");
        printf("It also generates an 8-bit color image,\n");
        printf("that swaps red and green components from the input
image");
        exit(1);
}



double clipping(double pixel)
{
    if (pixel > 255)    return 255;
    if (pixel < 0)    return 0;
    return pixel;
}
```

Section 5 - IIR Filter:

5.1 A derivation of the analytical expression for H(e^{ju}, e^{jv}).

$$y(m,n) = 0.01x(m,n) + 0.9\big(y(m-1,n) + y(m,n-1)\big) - 0.81y(m-1,n-1)$$

Therefore, the Z transform of the above equation is

$$Y(e^{j\mu}, e^{j\upsilon}) = 0.01X(e^{j\mu}, e^{j\upsilon}) + e^{-j\mu}0.9Y(e^{j\mu}, e^{j\upsilon}) + e^{-j\upsilon}0.9Y(e^{j\mu}, e^{j\upsilon}) - e^{-j\mu}e^{-j\upsilon}0.81Y(e^{j\mu}, e^{j\upsilon})$$

Since $H(e^{j\mu}, e^{j\upsilon}) = \frac{Y(e^{j\mu}, e^{j\upsilon})}{X(e^{j\mu}, e^{j\upsilon})}$

By simplifying the above equation, we get

$$H(e^{j\mu}, e^{j\upsilon}) = \frac{0.01}{1 - 0.9e^{-j\mu} - 0.9e^{-j\upsilon} + 0.81e^{-j\mu}e^{-j\upsilon}}$$

5.2 A plot of $|H(e^{ju}, e^{jv})|$.

Plotting the magnitude of $H(e^{j\mu}, e^{jv})$ using Matlab, and then we could get a graph as shown in Fig 5-2.

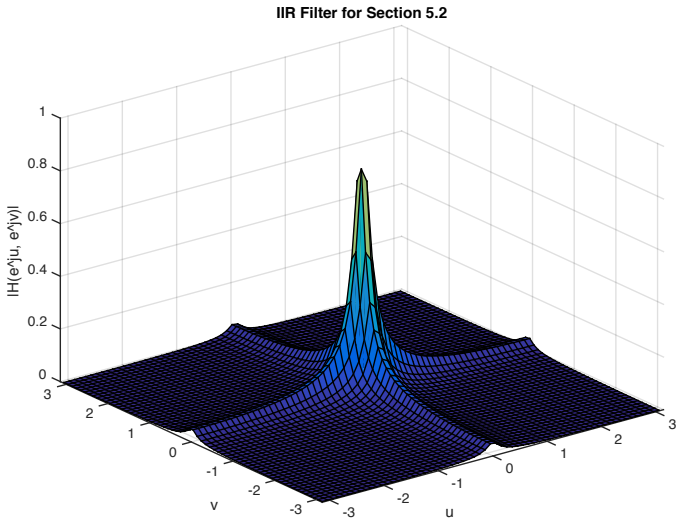

Fig 5-2 Magnitude of the Frequency Response over the region $[-\pi, \pi] \times [-\pi, \pi]$

5.3 An image of the point spread function.

Applying the difference equation to a 265x265 image of the form $x(m,n) = \delta(m-127, n-127)$, and then export the result to a TIFF file. We could get an image as shown in Fig 5-3.

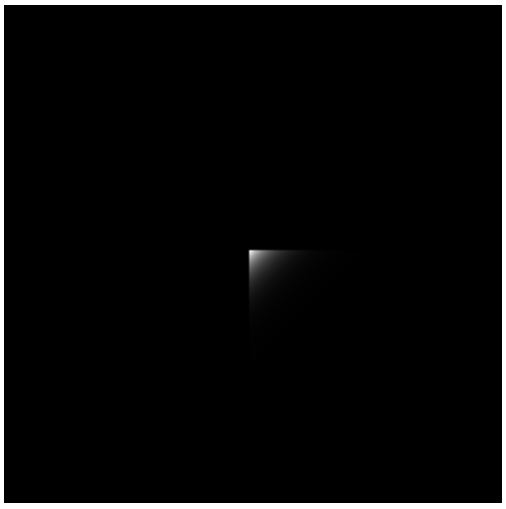

Fig 5-3 An image of the point spread function

5.4 The filtered output color image.

Fig 5-4-2 The original input image

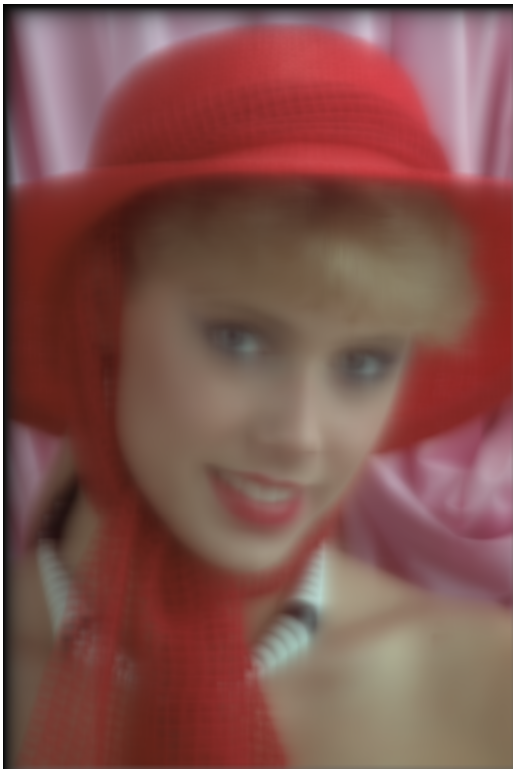

Fig 5-4-2 The filtered output color image

5.5 A listing of the C code.

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

void error(char *name);
double clipping(double pixel);


int main (int argc, char **argv)
{
   FILE *fp;
   struct TIFF_img input_img, green_img, color_img;
   double **img1,**img2,**img3; // red, green, blue
   int32_t i,j,k,l;


   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 */
    img1 = (double
**)get_img(input_img.width+1,input_img.height+1,sizeof(double));//red
    img2 = (double
**)get_img(input_img.width+1,input_img.height+1,sizeof(double));//gre
en
    img3 = (double
**)get_img(input_img.width+1,input_img.height+1,sizeof(double));//blu
e

    /* copy green component to double array */
    for(i = 0;i<input_img.height+1;i++)
    for(j = 0;j<input_img.width+1;j++){
            img1[i][j] = 0;
            img2[i][j] = 0;
            img3[i][j] = 0;
    }
```

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

   for ( i = 0; i < input_img.height; i++ )
   for ( j = 0; j < input_img.width; j++ ) {
      img1[i+1][j+1] =
0.01*input_img.color[0][i][j]+0.9*img1[i][j+1]+0.9*img1[i+1][j]-
0.81*img1[i][j];
      img2[i+1][j+1] =
0.01*input_img.color[1][i][j]+0.9*img2[i][j+1]+0.9*img2[i+1][j]-
0.81*img2[i][j];
      img3[i+1][j+1] =
0.01*input_img.color[2][i][j]+0.9*img3[i][j+1]+0.9*img3[i+1][j]-
0.81*img3[i][j];
      }

   for ( i = 0; i < input_img.height; i++ )
   for ( j = 0; j < input_img.width; j++ ) {
            //y(m,n)
      color_img.color[0][i][j] = clipping(img1[i+1][j+1]);
      color_img.color[1][i][j] = clipping(img2[i+1][j+1]);
      color_img.color[2][i][j] = clipping(img3[i+1][j+1]);
   }


   /* open color image file */
   if ( ( fp = fopen ( "color.tif", "wb" ) ) == NULL ) {
         fprintf ( stderr, "cannot open file color.tif\n");
```

```
        exit ( 1 );
    }

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

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

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

    free_img( (void**)img1 );
    free_img( (void**)img2 );
    free_img( (void**)img3 );

    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 horizontally filters the green component, adds
noise,\n");
    printf("and writes out the result as an 8-bit image\n");
```

```
        printf("with the name 'green.tiff'.\n");
        printf("It also generates an 8-bit color image,\n");
        printf("that swaps red and green components from the input
image");
        exit(1);
}


double clipping(double pixel)
{
    if (pixel > 255)    return 255;
    if (pixel < 0)    return 0;
    return pixel;
}
```