

```

1  //-----3 FIR Low Pass Filter-----
2  #include <math.h>
3  #include "tiff.h"
4  #include "allocate.h"
5  #include "randlib.h"
6  #include "typeutil.h"
7
8  void error(char *name);
9
10 int main (int argc, char **argv)
11 {
12     FILE *fp;
13     struct TIFF_img input_img, green_img, red_img, blue_img, color_img;
14     double **img1, **imgr, **imgb, **img2, **img3, **img4;
15     int32_t i, j, pixelg, ii, jj, pixelr, pixelb;
16
17     if ( argc != 2 ) error( argv[0] );
18
19     /* open image file */
20     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
21         fprintf ( stderr, "cannot open file %s\n", argv[1] );
22         exit ( 1 );
23     }
24
25     /* read image */
26     if ( read_TIFF ( fp, &input_img ) ) {
27         fprintf ( stderr, "error reading file %s\n", argv[1] );
28         exit ( 1 );
29     }
30
31     /* close image file */
32     fclose ( fp );
33
34     /* check the type of image data */
35     if ( input_img.TIFF_type != 'c' ) {
36         fprintf ( stderr, "error: image must be 24-bit color\n" );
37         exit ( 1 );
38     }
39
40     /* Allocate image of double precision floats */
41     img1 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
42     imgr = (double **)get_img(input_img.width, input_img.height, sizeof(double));
43     imgb = (double **)get_img(input_img.width, input_img.height, sizeof(double));
44     img2 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
45     img3 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
46     img4 = (double **)get_img(input_img.width, input_img.height, sizeof(double));
47
48     // /* Initialize the img arrays */
49     // for ( i = 0; i < input_img.height; i++ )
50     // for ( j = 0; j < input_img.width; j++ ) {
51     //     img1[i][j] = 0;
52     //     img2[i][j] = 0;
53     // }
54
55     /* copy green, red & blue component to double array */
56     for ( i = 0; i < input_img.height; i++ )
57     for ( j = 0; j < input_img.width; j++ ) {
58         img1[i][j] = input_img.color[1][i][j];
59         imgr[i][j] = input_img.color[0][i][j];
60

```

```
61     imgb[i][j] = input_img.color[2][i][j];
62 }
63
64
65 /* Filter image with the FIR Low Pass Filter */
66 for ( i = 4; i < input_img.height-4; i++ )
67 for ( j = 4; j < input_img.width-4; j++ ) {
68     // img2[i][j] = (img1[i][j-1] + img1[i][j] + img1[i][j+1])/3.0;
69     for ( ii = -4; ii <= 4; ii++ )
70     for ( jj = -4; jj <= 4; jj++ ) {
71         img2[i][j] = img2[i][j] + img1[i+ii][j+jj]/81;
72         img3[i][j] = img3[i][j] + imgr[i+ii][j+jj]/81;
73         img4[i][j] = img4[i][j] + imgb[i+ii][j+jj]/81;
74     }
75 }
76
77 /* Fill in boundary pixels */
78
79 // for ( i = 0; i < input_img.height; i++ ) {
80 //     img2[i][0] = 0;
81 //     img2[i][input_img.width-1] = 0;
82 // }
83
84 for ( i = 0; i < 4; i++ )
85 for ( j = 0; j < 4; j++ ) {
86     for ( ii = -1*i; ii <= 4; ii++ )
87     for ( jj = -1*i; jj <= 4; jj++ ) {
88         img2[i][j] = img2[i][j] + img1[i+ii][j+jj]/81;
89         img3[i][j] = img3[i][j] + imgr[i+ii][j+jj]/81;
90         img4[i][j] = img4[i][j] + imgb[i+ii][j+jj]/81;
91     }
92 }
93
94 for ( i = input_img.height-4; i < input_img.height; i++ )
95 for ( j = input_img.width-4; j < input_img.width; j++ ) {
96     for ( ii = -4; ii < input_img.height-i; ii++ )
97     for ( jj = -4; jj < input_img.width-j; jj++ ) {
98         img2[i][j] = img2[i][j] + img1[i+ii][j+jj]/81;
99         img3[i][j] = img3[i][j] + imgr[i+ii][j+jj]/81;
100        img4[i][j] = img4[i][j] + imgb[i+ii][j+jj]/81;
101    }
102 }
103
104 for ( i = 0; i < 4; i++ )
105 for ( j = input_img.width-4; j < input_img.width; j++ ) {
106     for ( ii = -1*i; ii <= 4; ii++ )
107     for ( jj = -4; jj < input_img.width-j; jj++ ) {
108         img2[i][j] = img2[i][j] + img1[i+ii][j+jj]/81;
109         img3[i][j] = img3[i][j] + imgr[i+ii][j+jj]/81;
110         img4[i][j] = img4[i][j] + imgb[i+ii][j+jj]/81;
111     }
112 }
113
114 for ( i = input_img.height-4; i < input_img.height; i++ )
115 for ( j = 0; j < 4; j++ ) {
116     for ( ii = -4; ii < input_img.height-i; ii++ )
117     for ( jj = -1*i; jj <= 4; jj++ ) {
118         img2[i][j] = img2[i][j] + img1[i+ii][j+jj]/81;
119         img3[i][j] = img3[i][j] + imgr[i+ii][j+jj]/81;
120         img4[i][j] = img4[i][j] + imgb[i+ii][j+jj]/81;
```

```
121     }
122 }
123
124
125 // /* Set seed for random noise generator */
126 // srandom2(1);
127
128 // /* Add noise to image */
129 // for ( i = 0; i < input_img.height; i++ )
130 // for ( j = 1; j < input_img.width-1; j++ ) {
131 //     img2[i][j] += 32*normal();
132 // }
133
134 /* set up structure for output achromatic image */
135 /* to allocate a full color image use type 'c' */
136 get_TIFF ( &green_img, input_img.height, input_img.width, 'g' );
137 get_TIFF ( &red_img, input_img.height, input_img.width, 'g' );
138 get_TIFF ( &blue_img, input_img.height, input_img.width, 'g' );
139
140 /* set up structure for output color image */
141 /* Note that the type is 'c' rather than 'g' */
142 get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
143
144 /* copy green, red & blue component to new images */
145 for ( i = 0; i < input_img.height; i++ )
146 for ( j = 0; j < input_img.width; j++ ) {
147     pixelg = (int32_t)img2[i][j];
148     pixelr = (int32_t)img3[i][j];
149     pixelb = (int32_t)img4[i][j];
150
151     if(pixelg>255) {
152         green_img.mono[i][j] = 255;
153     }
154     else {
155         if(pixelg<0) green_img.mono[i][j] = 0;
156         else green_img.mono[i][j] = pixelg;
157     }
158
159     if(pixelr>255) {
160         red_img.mono[i][j] = 255;
161     }
162     else {
163         if(pixelr<0) red_img.mono[i][j] = 0;
164         else red_img.mono[i][j] = pixelr;
165     }
166
167     if(pixelb>255) {
168         blue_img.mono[i][j] = 255;
169     }
170     else {
171         if(pixelb<0) blue_img.mono[i][j] = 0;
172         else blue_img.mono[i][j] = pixelb;
173     }
174 }
175
176 // /* Illustration: constructing a sample color image -- interchanging the red and
177 // green components from the input color image */
178 // for ( i = 0; i < input_img.height; i++ )
179 //     for ( j = 0; j < input_img.width; j++ ) {
180 //         color_img.color[0][i][j] = input_img.color[1][i][j];
```

```

180 //          color_img.color[1][i][j] = input_img.color[0][i][j];
181 //          color_img.color[2][i][j] = input_img.color[2][i][j];
182 //      }
183
184 /* Illustration: constructing a sample color image -- put 3 image (green,red,blue)
into 1 image */
185 for ( i = 0; i < input_img.height; i++ )
186     for ( j = 0; j < input_img.width; j++ ) {
187         color_img.color[0][i][j] = red_img.mono[i][j];
188         color_img.color[1][i][j] = green_img.mono[i][j];
189         color_img.color[2][i][j] = blue_img.mono[i][j];
190     }
191
192 // /* open green image file */
193 // if ( ( fp = fopen ( "green.tif", "wb" ) ) == NULL ) {
194 //     fprintf ( stderr, "cannot open file green.tif\n");
195 //     exit ( 1 );
196 // }
197
198 // /* write green image */
199 // if ( write_TIFF ( fp, &green_img ) ) {
200 //     fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
201 //     exit ( 1 );
202 // }
203
204 // /* close green image file */
205 // fclose ( fp );
206
207
208 /* open color image file */
209 if ( ( fp = fopen ( "color.tif", "wb" ) ) == NULL ) {
210     fprintf ( stderr, "cannot open file color.tif\n");
211     exit ( 1 );
212 }
213
214 /* write color image */
215 if ( write_TIFF ( fp, &color_img ) ) {
216     fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
217     exit ( 1 );
218 }
219
220 /* close color image file */
221 fclose ( fp );
222
223 /* de-allocate space which was used for the images */
224 free_TIFF ( &(input_img) );
225 free_TIFF ( &(green_img) );
226 free_TIFF ( &(red_img) );
227 free_TIFF ( &(blue_img) );
228 free_TIFF ( &(color_img) );
229
230 free_img( (void**)img1 );
231 free_img( (void**)img2 );
232 free_img( (void**)img3 );
233 free_img( (void**)img4 );
234 free_img( (void**)imgr );
235 free_img( (void**)imgb );
236
237 return(0);
238 }

```

```
239
240 void error(char *name)
241 {
242     printf("usage: %s image.tiff \n\n",name);
243     printf("this program reads in a 24-bit color TIFF image.\n");
244     printf("It then horizontally filters the green component, adds noise,\n");
245     printf("and writes out the result as an 8-bit image\n");
246     printf("with the name 'green.tiff'.\n");
247     printf("It also generates an 8-bit color image,\n");
248     printf("that swaps red and green components from the input image");
249     exit(1);
250 }
251
252
```