color\_img;

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #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, green\_img, red\_img,blue\_img, double \*\*img1,\*\*imgr,\*\*imgb,\*\*img2,\*\*img3,\*\*img4;  int32\_t i,j,pixelg,ii,jj,pixelr,pixelb; | |
| 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 |  |  |

1. /\* read image \*/
2. if ( read\_TIFF ( fp, &input\_img ) ) {
3. fprintf ( stderr, "error reading file %s\n", argv[1] );
4. exit ( 1 ); 29 }

30

1. /\* close image file \*/
2. fclose ( fp ); 33
3. /\* check the type of image data \*/
4. if ( input\_img.TIFF\_type != 'c' ) {
5. fprintf ( stderr, "error: image must be 24-bit color\n" );
6. exit ( 1 ); 38 }

39

1. /\* Allocate image of double precision floats \*/
2. img1 = (double \*\*)get\_img(input\_img.width,input\_img.height,sizeof(double));
3. imgr = (double \*\*)get\_img(input\_img.width,input\_img.height,sizeof(double));
4. imgb = (double \*\*)get\_img(input\_img.width,input\_img.height,sizeof(double));
5. img2 = (double \*\*)get\_img(input\_img.width,input\_img.height,sizeof(double));
6. img3 = (double \*\*)get\_img(input\_img.width,input\_img.height,sizeof(double));
7. img4 = (double \*\*)get\_img(input\_img.width,input\_img.height,sizeof(double)); 47
8. // /\* Initialize the img arrays \*/
9. // for ( i = 0; i < input\_img.height; i++ )
10. // for ( j = 0; j < input\_img.width; j++ ) { 51 // img1[i][j] = 0;

52 // img2[i][j] = 0;

53 // }

54

55

1. /\* copy green, red & blue component to double array \*/
2. for ( i = 0; i < input\_img.height; i++ )
3. for ( j = 0; j < input\_img.width; j++ ) {
4. img1[i][j] = input\_img.color[1][i][j];
5. imgr[i][j] = input\_img.color[0][i][j];

61

62 }

63

64

imgb[i][j] = input\_img.color[2][i][j];

1. /\* Filter image with the FIR Low Pass Filter \*/
2. for ( i = 4; i < input\_img.height-4; i++ )
3. 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  green components from the input color image \*/ | | | | the red and |
| 177 | // | for | ( i | = 0; i < input\_img.height; i++ ) | |
| 178 | // |  | for | ( j = 0; j < input\_img.width; j++ ) { | |
| 179 | // |  |  | 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  185 | /\*  into  for | | Illustration: constructing a sample color image -- put 3 image  1 image \*/  ( i = 0; i < input\_img.height; i++ ) | (green,red,blue) |
| 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  202  203  204  205  206  207  208  209 | // exit ( 1 );  // }  // /\* close green image file \*/  // fclose ( fp );  /\* open color image file \*/  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  217  218  219  220  221  222 | fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );  exit ( 1 );  }  /\* close color image file \*/ fclose ( fp ); | | | |
| 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 {

1. printf("usage: %s image.tiff \n\n",name);
2. printf("this program reads in a 24-bit color TIFF image.\n");
3. printf("It then horizontally filters the green component, adds noise,\n");
4. printf("and writes out the result as an 8-bit image\n");
5. printf("with the name 'green.tiff'.\n");
6. printf("It also generates an 8-bit color image,\n");
7. printf("that swaps red and green components from the input image");
8. exit(1); 250 }

251

252