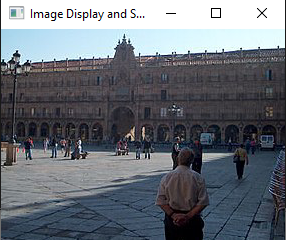
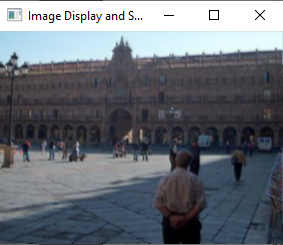
Bryce Chalfant

Starting off with the theory of the code, my code takes in an image reads it and displays it. The varying functions that you can do to edit the image are binary, greyscale, color, median, edge detection, and sobel edge detection. The purpose of this code is to manipulate and edit the image on display in real time. The original picture in my program looks like this:

My median function works by having a nested for loop that will iterate through the rows and columns of the pixel table except for the first and last of each. At each x, y coordinate, the function will look at all eight of the pixels neighbors and sum their red , green, and blue values and divide it by 9 which will then be assigned to the original x,y pixel’s red, green, and blue value respectively. This should cause a sort of blur affect which is meant to help eliminate noise that maybe in the image itself. If the frame of the window of neighbor pixels is increased, then we’ll see an increase in blurring as we’ll get a larger sample size to set as the pixel’s values. Here is the result of my median function with a 3x3 window 

My prewitt function is used in edge detection, so what this means is that when we run this function it will show where there are edges in the picture. This is done by applying a 3x3 mask that you apply to the pixel and its surrounding neighbors. This means that we will be getting the gradient difference between the current pixel and it’s neighbors. This means that if there isn’t a change in rgb then the output will be zero but if there is a change then that must mean there is an edge between this pixel and it’s neighbors. My code albeit messy does this by taking the absolute value of red values from each of the nine pixel with the mask applied to the values and summing them. I did this for the remaining two colors as well. After I summed the red, green, and blue value and checked if it was greater than 255, or less than 0 and capped it out if they were. Finally, I set the current pixel’s rgb values to the sum value. The result wasn’t perfect but I felt like it was good enough after struggling with this for a few hours. Here is my result:

Finally, I decided to try and implement the sobel operator, but that one did not come out as well as the other. The theory behind the sobel operator is that by using the second derivative we should get clearer edges than before with the priwitt operator since it’s only the first derivative. I tried a different approach to sobel than I did with priwitt because I felt like I couldn’t use the same method since this had a different mask to apply. I tried to calculate the gradient of x and the gradient of y all together instead of just doing the separate colors. So I applied the -1,-2,-1 mask the same was as in priwitt but something must have been wrong because the image didn’t have clearer edges. I’m almost positive that it was due to how I tried to implement it. Here’s how my sobel function looked: 

#include <stdlib.h>

#include <stdio.h>

#include <malloc.h>

#include <stdio.h>

#include <stdlib.h>

#include <GL/glut.h>

#include <FI/FreeImage.h>

#define FILENAME "4by3.jpg"

char outFileName[] = "outimg.tif"; // file name to be saved

//the pixel structure

typedef struct {

GLubyte r, g, b;

} pixel;

//the global structure

typedef struct {

pixel \*data;

int w, h;

} glob;

glob global;

FIBITMAP \*gimage;

enum { MENU\_FILTER, MENU\_SAVE, MENU\_GRAY, MENU\_COLOR, MENU\_ROTATE, MENU\_QUIT,MENU\_BOX, MENU\_MEDIAN, MENU\_EDGE, MENU\_SOBEL};

//read image

pixel \*read\_img(char \*name, int \*width, int \*height) {

FIBITMAP \*image;

int i, j, pnum;

RGBQUAD aPixel;

pixel \*data;

if ((image = FreeImage\_Load(FIF\_JPEG, name, 0)) == NULL) {

return NULL;

}

\*width = FreeImage\_GetWidth(image);

\*height = FreeImage\_GetHeight(image);

gimage = image;// share the pointer

data = (pixel \*)malloc((\*height)\*(\*width) \* sizeof(pixel \*));

pnum = 0;

for (i = 0; i < (\*height); i++) {

for (j = 0; j < (\*width); j++) {

FreeImage\_GetPixelColor(image, j, i, &aPixel);

data[pnum].r = (aPixel.rgbRed);

data[pnum].g = (aPixel.rgbGreen);

data[pnum++].b = (aPixel.rgbBlue);

}

}

FreeImage\_Unload(image);

return data;

}//read\_img

//write\_img

void write\_img(char \*name, pixel \*data, int width, int height) {

FIBITMAP \*image;

RGBQUAD aPixel;

int i, j;

image = FreeImage\_Allocate(width, height, 24, 0, 0, 0);

if (!image) {

perror("FreeImage\_Allocate");

return;

}

for (i = 0; i < height; i++) {

for (j = 0; j < width; j++) {

aPixel.rgbRed = data[i\*width + j].r;

aPixel.rgbGreen = data[i\*width + j].g;

aPixel.rgbBlue = data[i\*width + j].b;

FreeImage\_SetPixelColor(image, j, i, &aPixel);

}

}

if (!FreeImage\_Save(FIF\_TIFF, image, name, 0)) {

perror("FreeImage\_Save");

}

FreeImage\_Unload(image);

}//write\_img

/\*draw the image ‐ it is already in the format openGL requires for

glDrawPixels\*/

void display\_image(void)

{

// draw the image

glDrawPixels(global.w, global.h, GL\_RGB, GL\_UNSIGNED\_BYTE,

(GLubyte\*)global.data);

glFlush();

}//display\_image()

void draw\_triangle(void) // draw a triangle at a random position

{

glDrawPixels(global.w, global.h, GL\_RGB,

GL\_UNSIGNED\_BYTE, (GLubyte\*)global.data);

glBegin(GL\_TRIANGLES);

glColor3f(1.0, 0, 0);

glVertex2i(rand() % global.w, rand() % global.h);

glColor3f(0, 1.0, 0);

glVertex2i(rand() % global.w, rand() % global.h);

glColor3f(0, 0, 1.0);

glVertex2i(rand() % global.w, rand() % global.h);

glEnd();

glFlush();

}

//A simple thresholding filter.

void toBinary(pixel\* Im, int myIm\_Width, int

myIm\_Height) {

int x, y;

for (x = 0; x < myIm\_Width; x++)

for (y = 0; y < myIm\_Height; y++) {

if (Im[x + y \* myIm\_Width].g > 128) // thresholding

{

Im[x + y \* myIm\_Width].r = 255;

Im[x + y \* myIm\_Width].g = 255;

Im[x + y \* myIm\_Width].b = 255;

}

else

{

Im[x + y \* myIm\_Width].r = 0;

Im[x + y \* myIm\_Width].g = 0;

Im[x + y \* myIm\_Width].b = 0;

}

}

glutPostRedisplay(); // Tell glut that the image has been updated and needs to be redrawn

}//binary

// gray level conversion

void toGray(pixel\* Im, int myIm\_Width, int

myIm\_Height) {

int x, y;

int grayval = 0;

for (x = 0; x < myIm\_Width; x++)

for (y = 0; y < myIm\_Height; y++) {

grayval = (Im[x + y \* myIm\_Width].r +

Im[x + y \* myIm\_Width].g + Im[x + y \*

myIm\_Width].b) / 3;

Im[x + y \* myIm\_Width].r = grayval;

Im[x + y \* myIm\_Width].g = grayval;

Im[x + y \* myIm\_Width].b = grayval;

}

glutPostRedisplay();

}//gray

void median(pixel\* Im, int myIm\_Width, int myIm\_Height) {

for (int x = 1; x < myIm\_Width; x++) {

for (int y = 1; y < myIm\_Height; ++y) {

//red

Im[x + y \* myIm\_Width].r = (Im[(x - 1) + (y + 1) \* myIm\_Width].r + Im[(x)+(y + 1) \* myIm\_Width].r + Im[(x + 1) + (y + 1) \* myIm\_Width].r +

Im[(x - 1) + (y)\* myIm\_Width].r + Im[(x)+(y)\* myIm\_Width].r + Im[(x + 1) + (y)\* myIm\_Width].r + Im[(x - 1) + (y - 1) \* myIm\_Width].r +

Im[(x)+(y - 1) \* myIm\_Width].r + Im[(x + 1) + (y - 1) \* myIm\_Width].r) / 9;

//blue

Im[x + y \* myIm\_Width].b = (Im[(x - 1) + (y + 1) \* myIm\_Width].b + Im[(x)+(y + 1) \* myIm\_Width].b + Im[(x + 1) + (y + 1) \* myIm\_Width].b +

Im[(x - 1) + (y)\* myIm\_Width].b + Im[(x)+(y)\* myIm\_Width].b + Im[(x + 1) + (y)\* myIm\_Width].b + Im[(x - 1) + (y - 1) \* myIm\_Width].b +

Im[(x)+(y - 1) \* myIm\_Width].b + Im[(x + 1) + (y - 1) \* myIm\_Width].b) / 9;

//green

Im[x + y \* myIm\_Width].g = (Im[(x - 1) + (y + 1) \* myIm\_Width].g + Im[(x)+(y + 1) \* myIm\_Width].g + Im[(x + 1) + (y + 1) \* myIm\_Width].g +

Im[(x - 1) + (y)\* myIm\_Width].g + Im[(x)+(y)\* myIm\_Width].g + Im[(x + 1) + (y)\* myIm\_Width].g + Im[(x - 1) + (y - 1) \* myIm\_Width].g +

Im[(x)+(y - 1) \* myIm\_Width].g + Im[(x + 1) + (y - 1) \* myIm\_Width].g) / 9;

}

}

glutPostRedisplay();

}

void prewitt(pixel\* Im, int myIm\_Width, int myIm\_Height) {

GLbyte sum;

for (int x = 1; x < myIm\_Width; x++) {

for (int y = 1; y < myIm\_Height; ++y) {

//red

GLbyte red = abs(((Im[(x - 1) + y \* myIm\_Width].r - Im[(x + 1) + y \* myIm\_Width].r)/2 + (Im[(x - 1) + (y+1) \* myIm\_Width].r - Im[(x + 1) + (y+1) \* myIm\_Width].r) / 2 +

(Im[(x - 1) +( y-1) \* myIm\_Width].r - Im[(x + 1) + (y-1) \* myIm\_Width].r) / 2)/3) +

abs(((Im[x + (y+1) \* myIm\_Width].r - Im[x + (y-1) \* myIm\_Width].r)/2 + (Im[(x-1) + (y + 1) \* myIm\_Width].r - Im[(x-1) + (y - 1) \* myIm\_Width].r) / 2 +

(Im[(x+1) + (y + 1) \* myIm\_Width].r - Im[(x+1) + (y - 1) \* myIm\_Width].r) / 2)/3);

//blue

GLbyte blue = abs(((Im[(x - 1) + y \* myIm\_Width].b - Im[(x + 1) + y \* myIm\_Width].b) / 2 + (Im[(x - 1) + (y + 1) \* myIm\_Width].b - Im[(x + 1) + (y + 1) \* myIm\_Width].b) / 2 +

(Im[(x - 1) + (y - 1) \* myIm\_Width].b - Im[(x + 1) + (y - 1) \* myIm\_Width].b) / 2) / 3) +

abs(((Im[x + (y + 1) \* myIm\_Width].b - Im[x + (y - 1) \* myIm\_Width].b) / 2 + (Im[(x - 1) + (y + 1) \* myIm\_Width].b - Im[(x - 1) + (y - 1) \* myIm\_Width].b) / 2 +

(Im[(x + 1) + (y + 1) \* myIm\_Width].b - Im[(x + 1) + (y - 1) \* myIm\_Width].b) / 2) / 3);

//green

GLbyte green = abs(((Im[(x - 1) + y \* myIm\_Width].g - Im[(x + 1) + y \* myIm\_Width].g) / 2 + (Im[(x - 1) + (y + 1) \* myIm\_Width].g - Im[(x + 1) + (y + 1) \* myIm\_Width].g) / 2 +

(Im[(x - 1) + (y - 1) \* myIm\_Width].g - Im[(x + 1) + (y - 1) \* myIm\_Width].g) / 2) / 3) +

abs(((Im[x + (y + 1) \* myIm\_Width].g - Im[x + (y - 1) \* myIm\_Width].g) / 2 + (Im[(x - 1) + (y + 1) \* myIm\_Width].g - Im[(x - 1) + (y - 1) \* myIm\_Width].g) / 2 +

(Im[(x + 1) + (y + 1) \* myIm\_Width].g - Im[(x + 1) + (y - 1) \* myIm\_Width].g) / 2) / 3);

/\*

GLbyte gradientx = abs((

((Im[(x - 1) + (y+1) \* myIm\_Width].r + Im[(x - 1) + (y+1) \* myIm\_Width].g + Im[(x - 1) + (y+1) \* myIm\_Width].b) - (Im[(x + 1) + (y+1) \* myIm\_Width].r + Im[(x + 1) + (y+1) \* myIm\_Width].g + Im[(x + 1) + (y+1) \* myIm\_Width].b)) + //row1

((Im[(x-1) + y \* myIm\_Width].r +Im[(x-1) + y \* myIm\_Width].g + Im[(x-1) + y \* myIm\_Width].b) - (Im[(x+1) + y \* myIm\_Width].r + Im[(x+1) + y \* myIm\_Width].g + Im[(x+1) + y \* myIm\_Width].b)) + //row2

((Im[(x - 1) + (y - 1) \* myIm\_Width].r + Im[(x - 1) + (y - 1) \* myIm\_Width].g + Im[(x - 1) + (y - 1) \* myIm\_Width].b) - (Im[(x + 1) + (y - 1) \* myIm\_Width].r + Im[(x + 1) + (y - 1) \* myIm\_Width].g + Im[(x + 1) + (y - 1) \* myIm\_Width].b))));//row3

GLbyte gradienty = abs((

((Im[(x - 1) + (y + 1) \* myIm\_Width].r + Im[(x - 1) + (y + 1) \* myIm\_Width].g + Im[(x - 1) + (y + 1) \* myIm\_Width].b) - (Im[(x - 1) + (y - 1) \* myIm\_Width].r + Im[(x - 1) + (y - 1) \* myIm\_Width].g + Im[(x - 1) + (y - 1) \* myIm\_Width].b)) +//collumn 1

((Im[(x) + (y + 1) \* myIm\_Width].r + Im[(x) + (y + 1) \* myIm\_Width].g + Im[(x) + (y + 1) \* myIm\_Width].b) - (Im[(x) + (y - 1) \* myIm\_Width].r + Im[(x) + (y - 1) \* myIm\_Width].g + Im[(x) + (y - 1) \* myIm\_Width].b)) / 2 + // collumn 2

((Im[(x - 1) + (y + 1) \* myIm\_Width].r + Im[(x - 1) + (y + 1) \* myIm\_Width].g + Im[(x - 1) + (y + 1) \* myIm\_Width].b) - (Im[(x + 1) + (y - 1) \* myIm\_Width].r + Im[(x + 1) + (y - 1) \* myIm\_Width].g + Im[(x + 1) + (y - 1) \* myIm\_Width].b))));//column 3

sum = gradientx + gradienty;

\*/

sum = (red + green + blue);

sum = sum > 255 ? 255 : sum;

sum = sum < 0 ? 0 : sum;

Im[x + y \* myIm\_Width].r = sum;

Im[x + y \* myIm\_Width].b = sum;

Im[x + y \* myIm\_Width].g = sum;

}

}

glutPostRedisplay();

}

void sobel(pixel\* Im, int myIm\_Width, int myIm\_Height) {

GLbyte sum;

for (int x = 1; x < myIm\_Width; x++) {

for (int y = 1; y < myIm\_Height; ++y) {

GLbyte gradientx = abs((

(-(Im[(x - 1) + (y + 1) \* myIm\_Width].r + Im[(x - 1) + (y + 1) \* myIm\_Width].g + Im[(x - 1) + (y + 1) \* myIm\_Width].b) - 2 \* (Im[(x) + (y + 1) \* myIm\_Width].r + Im[(x) + (y + 1) \* myIm\_Width].g + Im[(x) + (y + 1) \* myIm\_Width].b) -(Im[(x + 1) + (y + 1) \* myIm\_Width].r + Im[(x + 1) + (y + 1) \* myIm\_Width].g + Im[(x + 1) + (y + 1) \* myIm\_Width].b)) + //row1

(-(Im[(x - 1) + y \* myIm\_Width].r + Im[(x - 1) + y \* myIm\_Width].g + Im[(x - 1) + y \* myIm\_Width].b) - 2\* (Im[(x + 1) + y \* myIm\_Width].r + Im[(x) + y \* myIm\_Width].g + Im[(x) + y \* myIm\_Width].b) - (Im[(x+1) + y \* myIm\_Width].r + Im[(x + 1) + y \* myIm\_Width].g + Im[(x + 1) + y \* myIm\_Width].b)) + //row2

(-(Im[(x - 1) + (y - 1) \* myIm\_Width].r + Im[(x - 1) + (y - 1) \* myIm\_Width].g + Im[(x - 1) + (y - 1) \* myIm\_Width].b) - 2\* (Im[(x) + (y - 1) \* myIm\_Width].r + Im[(x) + (y - 1) \* myIm\_Width].g + Im[(x) + (y - 1) \* myIm\_Width].b) - (Im[(x + 1) + (y - 1) \* myIm\_Width].r + Im[(x + 1) + (y - 1) \* myIm\_Width].g + Im[(x + 1) + (y - 1) \* myIm\_Width].b))));//row3

GLbyte gradienty = abs((

(-(Im[(x - 1) + (y + 1) \* myIm\_Width].r + Im[(x - 1) + (y + 1) \* myIm\_Width].g + Im[(x - 1) + (y + 1) \* myIm\_Width].b) - 2\* (Im[(x - 1) + (y) \* myIm\_Width].r + Im[(x - 1) + (y) \* myIm\_Width].g + Im[(x - 1) + (y) \* myIm\_Width].b) - (Im[(x - 1) + (y - 1) \* myIm\_Width].r + Im[(x - 1) + (y - 1) \* myIm\_Width].g + Im[(x - 1) + (y - 1) \* myIm\_Width].b)) +//collumn 1

(-(Im[(x)+(y + 1) \* myIm\_Width].r + Im[(x)+(y + 1) \* myIm\_Width].g + Im[(x)+(y + 1) \* myIm\_Width].b) -2 \* (Im[(x)+(y) \* myIm\_Width].r + Im[(x)+(y) \* myIm\_Width].g + Im[(x)+(y) \* myIm\_Width].b) - (Im[(x)+(y-1) \* myIm\_Width].r + Im[(x)+(y - 1) \* myIm\_Width].g + Im[(x)+(y - 1) \* myIm\_Width].b)) / 2 + // collumn 2

(-(Im[(x - 1) + (y + 1) \* myIm\_Width].r + Im[(x - 1) + (y + 1) \* myIm\_Width].g + Im[(x - 1) + (y + 1) \* myIm\_Width].b) - 2 \* (Im[(x + 1) + (y) \* myIm\_Width].r + Im[(x + 1) + (y) \* myIm\_Width].g + Im[(x + 1) + (y) \* myIm\_Width].b) - (Im[(x + 1) + (y - 1) \* myIm\_Width].r + Im[(x + 1) + (y - 1) \* myIm\_Width].g + Im[(x + 1) + (y - 1) \* myIm\_Width].b))));//column 3

sum = gradientx + gradienty;

sum = gradientx + gradienty;

sum = sum > 255 ? 255 : sum;

sum = sum < 0 ? 0 : sum;

Im[x + y \* myIm\_Width].r = sum;

Im[x + y \* myIm\_Width].b = sum;

Im[x + y \* myIm\_Width].g = sum;

}

}

}

void drawBox() {

glDrawPixels(global.w, global.h, GL\_RGB,

GL\_UNSIGNED\_BYTE, (GLubyte\*)global.data);

glBegin(GL\_LINE\_LOOP);

glColor3f(1, 0, 0); glVertex2i(175, 100); glVertex2i(200, 100);//down x horizontal

glColor3f(1, 0, 0); glVertex2i(200, 100); glVertex2i(200, 75);//down y vertical

glColor3f(1, 0, 0); glVertex2i(200, 75); glVertex2i(175, 75);//back to x

glColor3f(1, 0, 0); glVertex2i(175, 75); glVertex2i(175, 100);//up y

glEnd();

glFlush();

}

/\*glut keyboard function\*/

void keyboard(unsigned char key, int x, int y)

{

switch (key)

{

case 0x1B: case'q': case 'Q':

exit(0); break;

case's': case'S':

printf("SAVING IMAGE: outimg.tif\n");

write\_img(outFileName, global.data, global.w,

global.h);

break;

case 'g': case 'G':

toGray(global.data, global.w, global.h);

break;

case'b': case'B':

toBinary(global.data, global.w, global.h);

break;

case'c': case'C':

global.data = read\_img((char\*)FILENAME,

&global.w, &global.h); display\_image();

break;

case't': case'T':

draw\_triangle();

break;

case'd': case'D':

drawBox();

break;

case'm': case 'M':

median(global.data, global.w,global.h);

break;

case'e': case 'E':

prewitt(global.data, global.w, global.h);

break;

case 'f': case 'F':

sobel(global.data, global.w, global.h);

break;

}

}

//keyboard

//Glut menu callback function

void menuFunc(int value)

{

switch (value) {

case MENU\_QUIT: { exit(0); }break;

case MENU\_SAVE:

{

printf("SAVING IMAGE: outimg.tif\n");

write\_img(outFileName, global.data, global.w, global.h);

}

break;

case MENU\_GRAY:

{

toGray(global.data, global.w, global.h);

}

break;

case MENU\_COLOR: {

global.data = read\_img((char\*)FILENAME, &global.w, &global.h);

display\_image();

}

break;

case MENU\_FILTER:

{

toBinary(global.data, global.w, global.h);

}

break;

case MENU\_ROTATE:

{

gimage = FreeImage\_Rotate(gimage, 30);

RGBQUAD aPixel;

int pnum = 0;

for (int i = 0; i < global.h; i++) {

for (int j = 0; j < global.w; j++) {

FreeImage\_GetPixelColor(gimage, j, i, &aPixel);

global.data[pnum].r = (aPixel.rgbRed);

global.data[pnum].g = (aPixel.rgbGreen);

global.data[pnum++].b = (aPixel.rgbBlue);

}

}

display\_image();

}

break;

case MENU\_BOX:

{

drawBox();

display\_image();

}

break;

case MENU\_MEDIAN:

{

median(global.data, global.w, global.h);

}

break;

case MENU\_EDGE:

{

prewitt(global.data, global.w, global.h);

}

case MENU\_SOBEL:

{

sobel(global.data, global.w, global.h);

}

break;

}

}//menuFunc

void show\_keys()

{

printf("Q:quit\nB:Binary\nC:Color\nG:Gray\nT:Draw a triangle\nD:Draw a Box\nM:Median\nE:Edge\nF:Sobel\nS:save\n");

}

//Glut menu set up

void init\_menu()

{

int sub\_menu = glutCreateMenu(&menuFunc);

glutAddMenuEntry("Gray", MENU\_GRAY);

glutAddMenuEntry("Color", MENU\_COLOR);

glutAddMenuEntry("Binary", MENU\_FILTER);

glutAddMenuEntry("Rotate", MENU\_ROTATE);

glutAddMenuEntry("Median", MENU\_MEDIAN);

glutAddMenuEntry("Edge", MENU\_EDGE);

glutAddMenuEntry("Sobel", MENU\_SOBEL);

int main\_menu = glutCreateMenu(&menuFunc);

glutAddSubMenu("Modify", sub\_menu);

glutAddMenuEntry("Box",MENU\_BOX);

glutAddMenuEntry("Save", MENU\_SAVE);

glutAddMenuEntry("Quit", MENU\_QUIT);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

int main(int argc, char\*\* argv)

{

global.data = read\_img((char\*)FILENAME, &global.w,

&global.h);

if (global.data == NULL)

{

printf("Error loading image file %s\n",

FILENAME);

return 1;

}

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowSize(global.w, global.h);

glutCreateWindow("Image Display and Simple Filtering");

glutDisplayFunc(display\_image);

glutKeyboardFunc(keyboard);

glMatrixMode(GL\_PROJECTION);

glOrtho(0, global.w, 0, global.h, 0, 1);

init\_menu();

show\_keys();

glutMainLoop();

return 0;

}