Edge Preserved Blurring

Instructor	Dr. Wesley Snyder
Student	Pratheek Bramhasamudra Mallikarjuna
ID	200065594
E-mail	pbramha@ncsu.edu

Table of Contents

Introduction

Images captured in real day life have lot of imperfections, like noise due to optics and sensors. Performing certain image processing operations on these images can enhance them. One such operation employed is image blurring. Image blurring is nothing but replacing every pixel of the image by some sort of average of its neighborhood. By, doing so, image is smoothened and effects of noise are less prominent.

However, blurring the image not only smoothens the effects of noise, it also smoothens the edges, rendering the image less sharp. To mitigate this problem, one must employ a technique, which smoothens the image while preserving the edges. This can be achieved by using variable conductance Diffusion equation to iteratively smoothen the non-edge parts of the image.

For this we have to find a characteristic function with has low value at the edges and high value at the non-edges to act as a variable conductance of the diffusion equation. From this diffusion equations with respect to time is calculated and iteratively added to the image. The amount of blurring is determined by number of iterations and the factor of diffusion equation added to the image. This project deals with implementation of the above method on various equations.

Approach

The diffusion equation is given by following formula,

$$\frac{df}{dt} = \frac{\partial}{\partial x}\left(c\frac{\partial f}{\partial x}\right) + \frac{\partial}{\partial y}(c\frac{\partial f}{\partial y})$$

Conductance *c* should be small at edges and high at non-edges. Since the gradient operator has high values at edges, a negative exponential of the gradient operator will suit our requirement. For this reason, we select the following conductance value, *c*,

$$c = e^{-\left(\frac{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}{\tau}\right)}$$

If the conductance were a constant, then blurring with diffusion at time instant *t* would be equivalent to Gaussian blurring with $\sigma = \sqrt{2ct}$. The choice of τ varies with different images since the gradient operator gives different values for different images depending on the edginess of the image. For this project, value of τ is determined by trial and error method.

The derivatives with respect to x and y are calculated by convoluting the image with following kernels.

$$h_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$h_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

After calculating the diffusion equation we iteratively add it to the image as follows,

$$f(t + \alpha) = f(t) + \alpha\frac{df}{dt}$$

Number of iterations performed depends on the amount of blurring required and scaling factor α. If α is large, convergence to the required blurriness will be faster at the

cost of quality. If α is small, it will take more iteration to achieve the required blurriness with better quality.

Results

Following are the results for edge preserved blurring on angio.ifs.

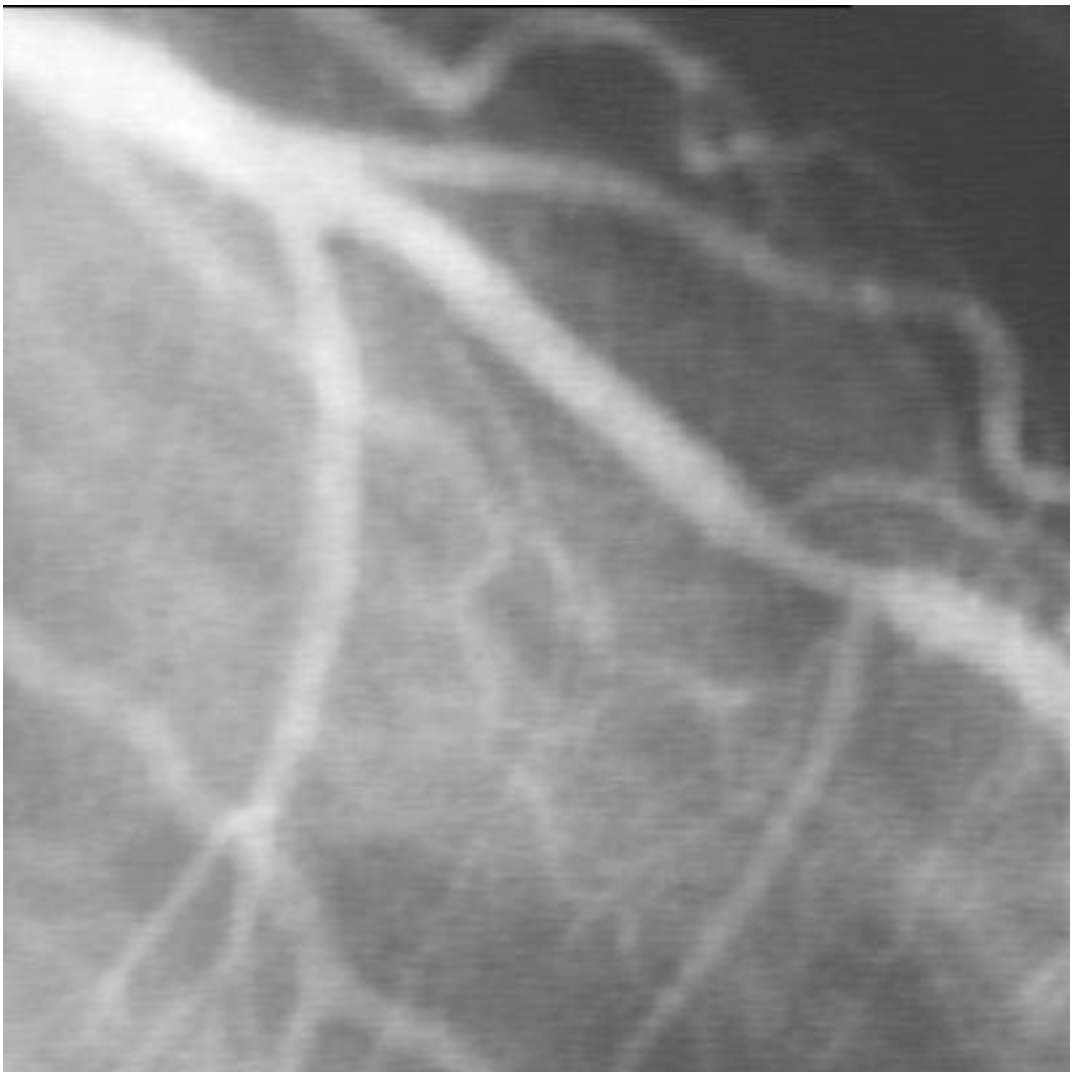

Figure 1: Angio Original

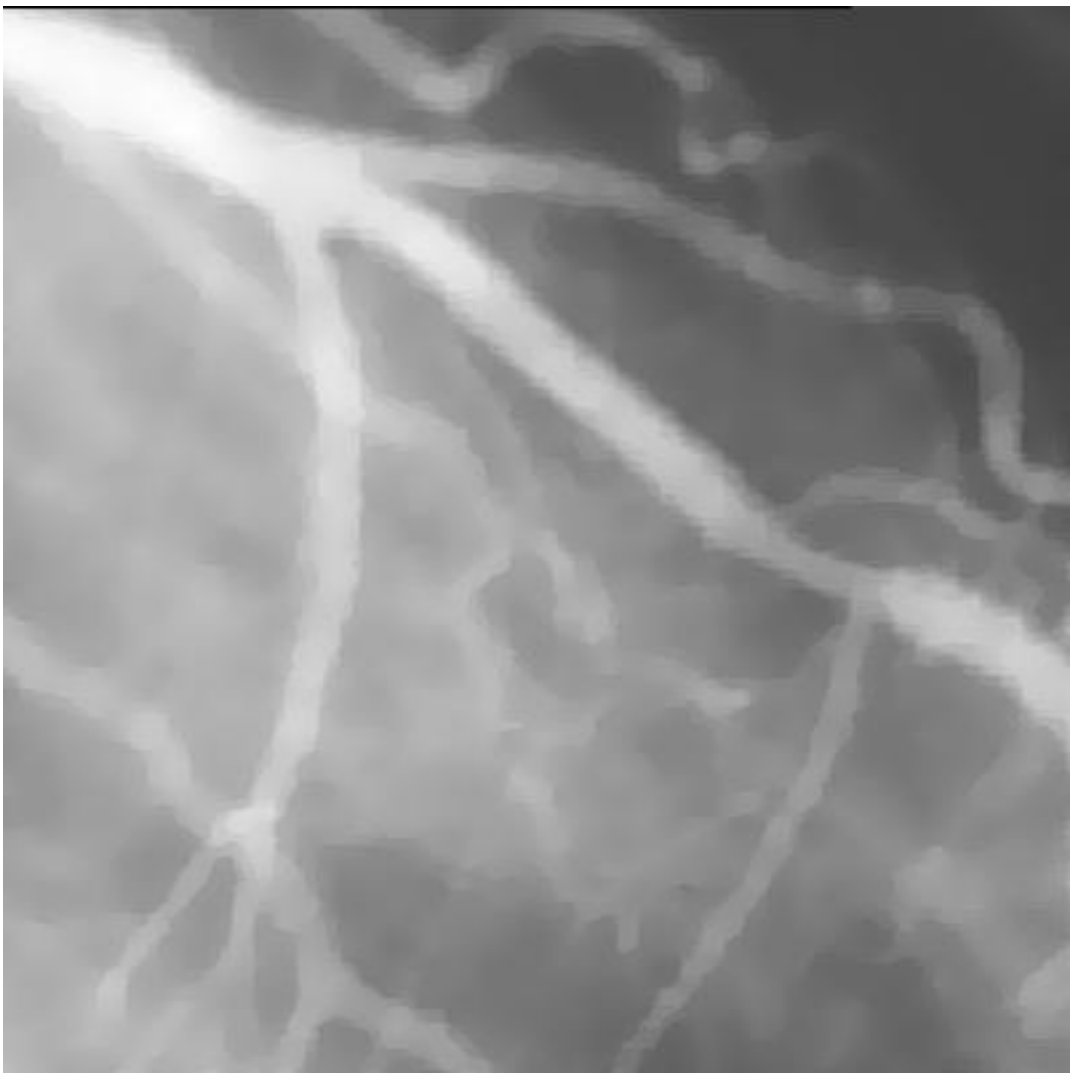

Figure 2: Angio, After Blurring τ = 5, α = 10, Iteration = 100

Following are the results for edge preserved blurring on angio128.ifs.

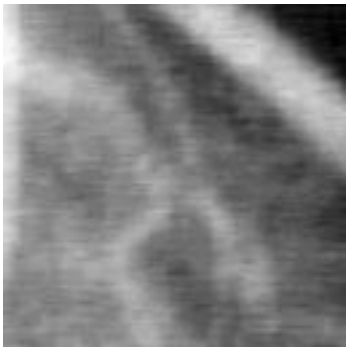

Figure 3: Angio128, Original

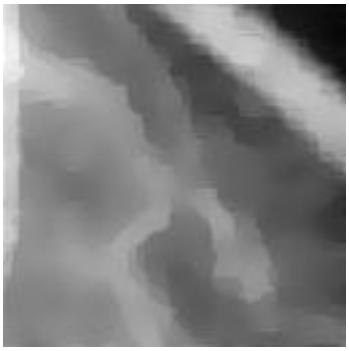

Figure 4: Angio128, After Blurring, $\tau = 5$, $\alpha = 10$, Iteration = 100

Following are the results for edge preserved blurring on BLUR1.V1.ifs.

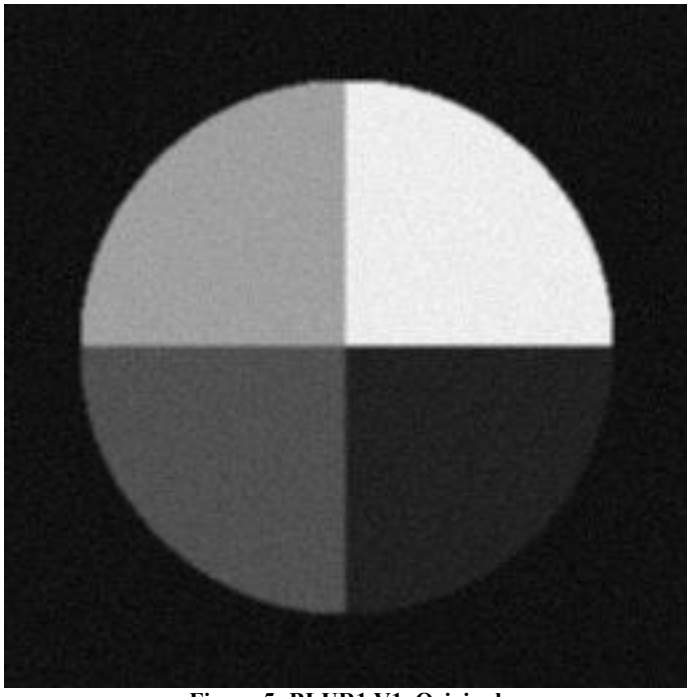

Figure 5: BLUR1.V1, Original

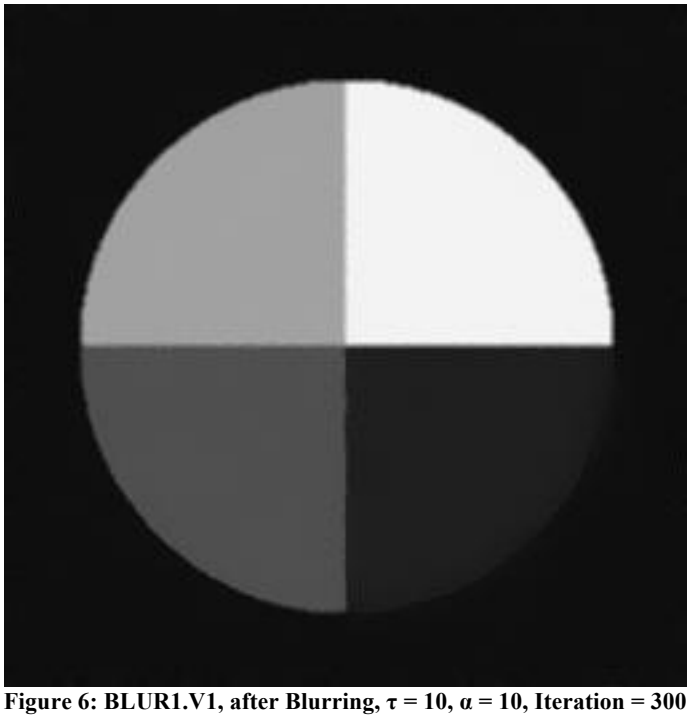

Figure 6: BLUR1.V1, after Blurring, $\tau = 10$, $\alpha = 10$, Iteration = 300

Following are the results for edge preserved blurring on lymphoma.ifs.

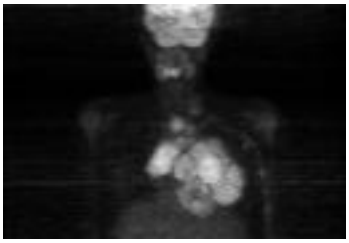

Figure 7: Lymphoma, Original

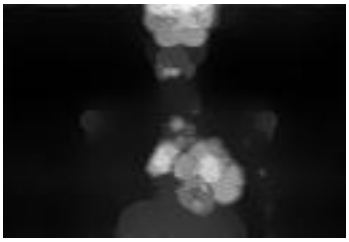

Figure 8: Lymphoma, after Blurring, τ = 50000, α = 10, Iteration = 100

Conclusion

From the result images, we can see that images are being blurred without affecting the edges much. However, choice of τ, α and number of iteration varied for different images. This poses challenges to generalize the algorithm without much of the user intervention. Moreover, this method is computationally intensive and might not be viable for large images, which require large number of iteration. This opens up the opportunities to parallelize the tasks to improve performance.

Code

```
/*
 * main.cpp
 *
 *  Created on: Jan 24, 2015
 *      Author: pratheek
 */

#include <iostream>
#include <stdio.h>
#include <math.h>
#include "ifs.h"
#include "flip.h"
#include "castTypes.h"
#include "image.h"
using namespace std;

int main() {

    char *a = "angio.ifs";
    Image temp((char *) a);
    int len[3];
    len[0] = 2;//temp.dimension;
    len[1] = temp.col;
    len[2] = temp.row;
    Image givenImage(len[0], len[2], len[1]);
    givenImage.read((char *) a);

    Image         tem(givenImage),c(givenImage),         cdx(givenImage),         cdy(givenImage),
givenImageX(givenImage), givenImageY(givenImage),
            givenImageXPlusY(givenImage);

    for (int i = 0; i < 100; i++) {
        cout << i << endl;
        cdx = givenImage;
        cdy = givenImage;
        givenImageX = givenImage;
        givenImageY = givenImage;
        c.reset(0);
        givenImageXPlusY.reset(0);


        cdx.diffx();
        cdy.diffy();
        cdx.mul(cdx);
        cdy.mul(cdy);
        c.add(cdx);
        c.add(cdy);
        c.div(-50000);
        c.expo();
        givenImageX.diffx();
        givenImageY.diffy();
        givenImageX.mul(c);
        givenImageY.mul(c);
        givenImageX.diffx();
        givenImageY.diffy();
        givenImageXPlusY.add(givenImageX);
        givenImageXPlusY.add(givenImageY);

        givenImageXPlusY.div(10);
```

```
        givenImage.add(givenImageXPlusY);
    }

    tem.sub(givenImage);
    givenImage.write((char *) "givenImageBlurred.ifs");
    return 0;
}



/*
 * image.cpp
 *
 *  Created on: Jan 24, 2015
 *      Author: pratheek
 */
#include <iostream>
#include <stdio.h>
#include <math.h>
#include "ifs.h"
#include "flip.h"
#include "../inc/image.h"
#include "castTypes.h"
#include"image.h"

using namespace std;

/*****************************************************************************************************
********************
 * FUNCTION              : Image
 * CLASS         : Image
 * TYPE                  : Constructor
 * IN                    : Dimensions of the Image
 * OUT                   : None
 * PROCESS               : Allocates memory for Image data
 *

*****************************************************************************************************
******************/
Image::Image(int dim, int _row, int _col) {

    dimension = 0;
    row = 0;
    col = 0;
    img = NULL;

    if (dim <= 0 || _row < 0 || _col < 0) {
        cout << "ER >> Size of the image is negative!" << endl;
        ERROR_LOG;
    } else {
        dimension = dim;
        row = _row;
        col = _col;

        img = new double*[row];

        for (int i = 0; i < row; i++)
            img[i] = new double[col];

        if (NULL == img) {
            cout << "ER >> Memory allocation failed" << endl;
            ERROR_LOG;
        } else {
            // Initialize image to black image
            for (int x = 0; x < row; x++) {
                for (int y = 0; y < col; y++) {
                    img[x][y] = 0;
```

```
                }
            }
        }
    }
}

/*****************************************************************************************************
********************
 * FUNCTION              : Image
 * CLASS         : Image
 * TYPE                  : Copy constructor
 * IN                    : copy object
 * OUT                   : None
 * PROCESS               : Allocates new img and copies values form old
 *

*****************************************************************************************************
******************/
Image::Image(const Image &obj) {

    this->dimension = obj.dimension;
    this->col = obj.col;
    this->row = obj.row;

    this->img = new double*[row];

    for (int i = 0; i < row; i++)
        this->img[i] = new double[col];

    if (NULL == img) {
        cout << "ER >> Memory allocation failed" << endl;
        ERROR_LOG;
    } else {
        // Initialize image to black image
        for (int x = 0; x < row; x++) {
            for (int y = 0; y < col; y++) {
                this->img[x][y] = obj.img[x][y];
            }
        }
    }
}
/*****************************************************************************************************
********************
 * FUNCTION     : Image
 * CLASS        : Image
 * TYPE         : Constructor
 * IN           : None
 * OUT          : None
 * PROCESS      : Reads an image from disk
 *

*****************************************************************************************************
******************/
Image::Image(char * fileName) {
    if ( NULL == fileName) {
        ERROR_LOG;
    } else {
        IFSIMG temp;
        int *len;
        temp = ifspin((char *) fileName);
        len = ifssiz(temp);
        dimension = len[0];
        col = len[1];
        row = len[2];

        img = new double*[row];

        for (int i = 0; i < row; i++)
            img[i] = new double[col];
```

```
        for (int x = 0; x < row; x++) {
            for (int y = 0; y < col; y++) {
                img[x][y] = ifsigp(temp, x, y);
            }
        }
    }
}
/*****************************************************************************************************
*********************
 * FUNCTION              : ~Image
 * CLASS         : Image
 * TYPE                  : Destructor
 * IN                    : None
 * OUT                   : None
 * PROCESS               : Deallocates memory for Image data
 *

*****************************************************************************************************
******************/
Image::~Image() {

    if (NULL != img) {
        for (int i = 0; i < row; i++) {
            delete[] img[i];
            img[i] = NULL;
        }
        delete[] img;
        img = NULL;
    }

    row = 0;
    col = 0;
    dimension = 0;
}
/*****************************************************************************************************
*********************
 * FUNCTION     : =
 * CLASS        : Image
 * TYPE         : object copy operator
 * IN           : copy object
 * OUT          : None
 * PROCESS      : Allocates new img and copies values form old
 *

*****************************************************************************************************
******************/
Image & Image::operator=(const Image &obj) {

    if (NULL != img) {
        for (int i = 0; i < row; i++) {
            delete[] img[i];
            img[i] = NULL;
        }
        delete[] img;
        img = NULL;
    }

    this->dimension = obj.dimension;
    this->col = obj.col;
    this->row = obj.row;

    this->img = new double*[row];

    for (int i = 0; i < row; i++)
        this->img[i] = new double[col];

    if (NULL == img) {
        cout << "ER >> Memory allocation failed" << endl;
```

```
        ERROR_LOG;
    } else {
        // Initialize image to black image
        for (int x = 0; x < row; x++) {
            for (int y = 0; y < col; y++) {
                this->img[x][y] = obj.img[x][y];
            }
        }
    }

    return *this;
}
/*****************************************************************************************************
********************
 * FUNCTION     : reset
 * CLASS        : Image
 * TYPE         : member function
 * IN           : None
 * OUT          : None
 * PROCESS      : Reads an image from disk
 *

*****************************************************************************************************
******************/
imgerr Image::reset(double resetTo) {
    if ( NULL == img) {
        return FAIL;
    } else {
        for (int x = 0; x < row; x++) {
            for (int y = 0; y < col; y++) {
                img[x][y] = resetTo;
            }
        }
    }
    return SUCC;
}
/*****************************************************************************************************
********************
 * FUNCTION             : read
 * CLASS        : Image
 * TYPE                 : member function
 * IN                   : None
 * OUT                  : None
 * PROCESS              : Reads an image from disk
 *

*****************************************************************************************************
******************/
imgerr Image::read(char * fileName) {
    if ( NULL == fileName) {
        return FAIL;
    } else {
        int *len;
        IFSIMG temp;
        temp = ifspin((char *) fileName);
        len = ifssiz(temp);

        if (NULL != img) {
            for (int i = 0; i < row; i++) {
                delete[] img[i];
                img[i] = NULL;
            }
            delete[] img;
            img = NULL;
        }

        dimension = len[0];
        col = len[1];
        row = len[2];
```

```
        this->img = new double*[row];

        for (int i = 0; i < row; i++)
            this->img[i] = new double[col];

        if (NULL == img) {
            cout << "ER >> Memory allocation failed" << endl;
            ERROR_LOG;
        } else {
            // Initialize image to black image
            for (int x = 0; x < row; x++) {
                for (int y = 0; y < col; y++) {
                    this->img[x][y] = ifsigp(temp, x, y);
                }
            }
        }
    }
    return SUCC;
}
/*****************************************************************************************************
********************
 * FUNCTION              : write
 * CLASS        : Image
 * TYPE                  : member function
 * IN                    : None
 * OUT                   : None
 * PROCESS               : Writes an image to the disk
 *

*****************************************************************************************************
******************/
imgerr Image::write(char * fileName) {
    IFSIMG temp;
    int len[3];
    len[0] = dimension; /*image to be created is two dimensional */
    len[1] = col; /*image has 128 columns */
    len[2] = row; /*image has 128 rows */

    temp = ifscreate((char *) "double", len, IFS_CR_ALL, 0);/*image is unsigned 8bit */
    if (NULL == temp) {
        cout << "ER >> ifscreate failed" << endl;
        ERROR_LOG;
    } else {
        // Initialize image to black image
        for (int x = 0; x < row; x++)
            for (int y = 0; y < col; y++) {
                ifsipp(temp, x, y, img[x][y]);
            }
    }

    ifspot(temp, fileName); /*write image 2 to disk */
    return SUCC;
}

/*****************************************************************************************************
********************
 * FUNCTION              : convl
 * CLASS        : Image
 * TYPE                  : member function
 * IN                    : Kernel, kernel dimensions
 * OUT                   : None
 * PROCESS               : Performs convolution with the given kernel
 *

*****************************************************************************************************
******************/
imgerr Image::convl(float **kernel, int kernelX, int kernelY) {
    float sum = 0;
```

```
    float kernelSum = 0;
    if ((0 == kernelX % 2) || (0 == kernelY % 2)) {
        cout << "Kernel dimensions should be ODD" << endl;
        ERROR_LOG;
        return FAIL;
    }

    for (int x = 0; x < kernelX; x++) {
        for (int y = 0; y < kernelY; y++) {
            kernelSum += llabs(kernel[x][y]);
        }
    }
    int boundryX = (kernelX - 1) >> 1;
    int boundryY = (kernelY - 1) >> 1;
    double **tempImage = NULL;

    tempImage = new double*[row]();
    for (int i = 0; i < row; i++) {
        tempImage[i] = new double[col]();
    }

    for (int x = boundryX; x < row - boundryX; x++) {
        for (int y = boundryY; y < col - boundryY; y++) {
            for (int opX = -boundryX; opX <= boundryX; opX++) {
                for (int opY = boundryY; opY >= -boundryY; opY--) {
                    sum += img[x + opX][y - opY] * kernel[opX + boundryX][boundryY - opY];
                }
            }
            tempImage[x][y] = sum / kernelSum;
            sum = 0;
        }
    }

    for (int x = boundryX; x < row - boundryX; x++) {
        for (int y = boundryY; y < col - boundryY; y++) {
            img[x][y] = tempImage[x][y];
        }
    }

    for (int i = 0; i < row; i++) {
        delete[] tempImage[i];
    }
    delete[] tempImage;

    return SUCC;
}

/*******************************************************************************************************
********************
 * FUNCTION              : getValue
 * CLASS         : Image
 * TYPE                  : member function
 * IN                    : row, col
 * OUT                   : None
 * PROCESS               : Return image value at row, col
 *

*******************************************************************************************************
******************/
double Image::getValue(int _row, int _col) {
    return img[_row][_col];
}

/*******************************************************************************************************
********************
 * FUNCTION              : setValue
 * CLASS         : Image
 * TYPE                  : member function
 * IN                    : value, row, col
```

```
 * OUT                  : None
 * PROCESS              : Sets image value at row, col with input value
 *

**********************************************************************************************
******************/
imgerr Image::setValue(int _row, int _col, double value) {

    img[_row][_col] = value;
    return SUCC;
}

/**********************************************************************************************
*********************
 * FUNCTION     : add
 * CLASS        : Image
 * TYPE         : member function
 * IN           : input Image
 * OUT          : None
 * PROCESS      : Adds input image to current image
 *

**********************************************************************************************
******************/
imgerr Image::add(const Image &inImg) {

    if ((this->row == inImg.row) && (this->col == inImg.col) && (this->dimension ==
inImg.dimension)) {
        for (int x = 0; x < row; x++) {
            for (int y = 0; y < col; y++) {
                img[x][y] = img[x][y] + inImg.img[x][y];
            }
        }
    } else {
        ERROR_LOG;
        return FAIL;
    }
    return SUCC;
}
/**********************************************************************************************
*********************
 * FUNCTION     : sub
 * CLASS        : Image
 * TYPE         : member function
 * IN           : input Image
 * OUT          : None
 * PROCESS      : Subtracts input image from current image
 *

**********************************************************************************************
******************/
imgerr Image::sub(Image inImg) {

    if ((this->row == inImg.row) && (this->col == inImg.col) && (this->dimension ==
inImg.dimension)) {
        for (int x = 0; x < row; x++) {
            for (int y = 0; y < col; y++) {
                img[x][y] = img[x][y] - inImg.img[x][y];
            }
        }
    } else {
        ERROR_LOG;
        return FAIL;
    }
    return SUCC;
}
/**********************************************************************************************
*********************
 * FUNCTION     : mul
```

```
 * CLASS        : Image
 * TYPE         : member function
 * IN           : input Image
 * OUT          : None
 * PROCESS      : multiplies current image by input image
 *

**********************************************************************************************
******************/
imgerr Image::mul(Image inImg) {

    if ((this->row == inImg.row) && (this->col == inImg.col) && (this->dimension ==
inImg.dimension)) {
        for (int x = 0; x < row; x++) {
            for (int y = 0; y < col; y++) {
                img[x][y] = img[x][y] * inImg.img[x][y];
            }
        }
    } else {
        ERROR_LOG;
        return FAIL;
    }
    return SUCC;
}
/**********************************************************************************************
*********************
 * FUNCTION     : div
 * CLASS        : Image
 * TYPE         : member function
 * IN           : divisor
 * OUT          : None
 * PROCESS      : divides current image by divisor
 *

**********************************************************************************************
******************/
imgerr Image::div(double divisor) {

    if (0 != divisor) {
        for (int x = 0; x < row; x++) {
            for (int y = 0; y < col; y++) {
                img[x][y] = img[x][y] / divisor;
            }
        }
    } else {
        ERROR_LOG;
        return FAIL;
    }
    return SUCC;
}
/**********************************************************************************************
*********************
 * FUNCTION     : exp
 * CLASS        : Image
 * TYPE         : member function
 * IN           : divisor
 * OUT          : None
 * PROCESS      : calculates the exponential of each pixel
 *

**********************************************************************************************
******************/
imgerr Image::expo() {

    for (int x = 0; x < row; x++) {
        for (int y = 0; y < col; y++) {
            img[x][y] = exp(img[x][y]);
        }
    }
```

```
    return SUCC;
}
/*****************************************************************************************************
********************
 * FUNCTION     : diffx
 * CLASS        : Image
 * TYPE         : member function
 * IN           : input Image
 * OUT          : None
 * PROCESS      : Differentiates current image in x direction
 *

*****************************************************************************************************
******************/
imgerr Image::diffx() {
    float **kernel;
    float hx[3][3] = { { -1, 0, 1 }, { -2, 0, 2 }, { -1, 0, 1 } };
    kernel = new float*[3];

    for (int i = 0; i < 3; i++)
        kernel[i] = new float[3];

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            kernel[i][j] = hx[i][j];
        }
    }

    convl((float **) kernel, 3, 3);

    for (int i = 0; i < 3; i++) {
        delete[] kernel[i];
    }
    delete[] kernel;
    return SUCC;
}
/*****************************************************************************************************
********************
 * FUNCTION     : diffy
 * CLASS        : Image
 * TYPE         : member function
 * IN           : input Image
 * OUT          : None
 * PROCESS      : Differentiates current image in y direction
 *

*****************************************************************************************************
******************/
imgerr Image::diffy() {
    float **kernel;
    float hy[3][3] = { { -1, -2, -1 }, { 0, 0, 0 }, { 1, 2, 1 } };
    kernel = new float*[3];

    for (int i = 0; i < 3; i++)
        kernel[i] = new float[3];

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            kernel[i][j] = hy[i][j];
        }
    }

    convl((float **) kernel, 3, 3);

    for (int i = 0; i < 3; i++) {
        delete[] kernel[i];
    }
    delete[] kernel;
    //fldy(img, img);
```

```
    return SUCC;
}
/* Nothing beyond this */


/*
 * image.h
 *
 *  Created on: Jan 24, 2015
 *      Author: pratheek
 */

#ifndef INC_IMAGE_H_
#define INC_IMAGE_H_

#include <stdio.h>
#include "ifs.h"
//#include "flip.h"

typedef enum _imgerr {
    FAIL = -1,
    SUCC = 0
} imgerr;

class Image {
public:
    double **img = NULL; /*Declare pointers to headers */
    int dimension = 0;
    int row = 0;
    int col = 0;

    Image(int dim, int _row, int _col);
    Image(const Image &obj);
    Image(char * fileName);
    ~Image();
    Image & operator=(const Image &obj);
    imgerr reset(double resetTo);
    imgerr read(char*);
    imgerr write(char*);
    imgerr convl(float **kernel, int kernelX, int kernelY);
    double getValue(int _row, int _col);
    imgerr setValue(int _row, int _col, double value);
    imgerr add(const Image &obj);
    imgerr sub(Image inImg);
    imgerr mul(Image inImg);
    imgerr div(double divisor);
    imgerr expo();
    imgerr diffx();
    imgerr diffy();
};

#endif /* INC_IMAGE_H_ */
/*
 * castTypes.h
 *
 *  Created on: Jan 24, 2015
 *      Author: pratheek
 */

#ifndef INC_CASTTYPES_H_
```

```
#define INC_CASTTYPES_H_
#include<iostream>
using namespace std;

#define ERROR_LOG cout<<"DB >>"<<__LINE__<<" "<<__FUNCTION__<<" "<<__FILE__<<endl

#endif /* INC_CASTTYPES_H_ */

/*
 * main.h
 *
 *  Created on: Jan 24, 2015
 *      Author: pratheek
 */

#ifndef INC_MAIN_H_
#define INC_MAIN_H_

#endif /* INC_MAIN_H_ */

CC = g++

CFLAGS = -g -O0 -Wall -Wextra
INC =   -I/Users/pratheek/ifsHOME/MacX64/hdr/\
                -I../inc

CFLAGS += $(INC)
CFLAGS += -I/Users/pratheek/ifsHOME/MacX64/hrd/opencv

SRC =   ../src/main.cpp\
                ../src/image.cpp

OBJ = $(SRC:.cpp=.o)
LIBS =  /Users/pratheek/ifsHOME/MacX64/ifslib/libflip.a
LIBS+=  /Users/pratheek/ifsHOME/MacX64/ifslib/libifs.a
LIBS+=  /Users/pratheek/ifsHOME/MacX64/ifslib/libiptools.a
LIBS+=  /Users/pratheek/ifsHOME/MacX64/ifslib/libqsyn.a



OUT = ../bin/sample_image.out

all: $(OBJ) $(LIBS)
        $(CC) $(CFLAGS)  -o $(OUT) $(OBJ) $(LIBS)
        rm -f *.o

$(OBJ): $(SRC)
        $(CC) $(CFLAGS) -c $(SRC) $(INC)

clean:
        rm -f *.o
        rm -f *.out
        rm -f ../bin/*.out
        #rm -f ../*.ifs
```