ECE572 – Digital Image Processing

Project2 – Image Enhancement Point Processing

Name: Sang-hyeb(Sam) Lee
NetID: slee91
Student ID: 000330428

98+10

Abstract

The objectives of this project are to implement certain point processing algorithms for image enhancement purpose, and also explore characteristics of each filter by generating sample images using each technique. For this project, I implemented eight basic point processing algorithms, histogram equalization, Gaussian noise, and local histogram equalization techniques. The result shows that interesting photos can be generated with these filters with very little effort.

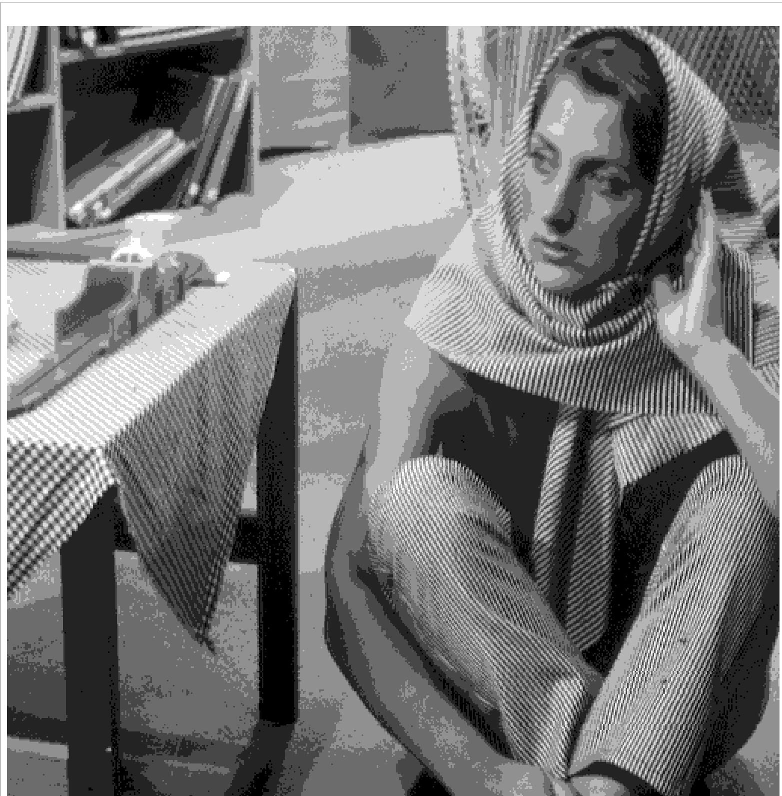
Result)

1) An explanation of the difference between image sampling and image quantization. From two aspects, define image resolution.

Answer)

When converting an analogous image to a digital form, we have to sample the image in both coordinates and in amplitude. Sampling refers to the process of digitizing the coordinate values whereas quantization refers to the process of digitizing the amplitudes.

From quantization point of view, we can vary the number of bits used to quantize a pixel intensity. This is called intensity resolution. Intensity resolution refers to the smallest discernible change in intensity level. Below is a series of photos with different quantization level.

 Quantization level =2(1bit)	 *Quantization level = 8(3bits)*

From sampling point of view, we can use spatial resolution to denote image resolution. Spatial resolution is a measure of the smallest discernible detail in an image. It is generally stated using with line pairs per unit distance and pixels per unit distance. Important thing about spatial resolution is that, spatial resolution must be stated with respect to spatial units to be meaningful.

2) Explain Pros and cons between vector representation and bitmap representation.

Answer)
1) Vector representation

PROS)	1) Smaller image size 2) can support text searching. 3) Suitable for animation because it is easier to produce variations of an initial sketch. 4) Lines and curves of images remain sharp even as the image is scaled.
CONS)	1) Only handles simple line drawings, shades, and shadings 2) Hard to apply special effects which change the overall color balance of the image such as blurring.

2) Bitmap representation

PROS)	1) easily handle images with complex colors, shades, and shapes. 2) Easier
CONS)	1) Significantly larger image size 2) Fixed resolution. Zooming in on a bitmap image results in a pixelated look.

3) Comment on the effect by using different parameters of the power-law transformation. Can power-law transformation replace log transformation?

Answer) Power-law transformation is defined as:

$$s = c * r^{\Upsilon}$$

where c and ϒ are positive constants.

When the value of ϒ is less than 1, the power-law function maps a narrow range of dark input values into a wider range of output values but it maps wider range of bright input values into a narrower range of output values. When the value of ga ϒ greater than 1, the power-law function maps a narrow range of bright input values into a wider range of output values, with the opposite being true for darker values of input levels. When the values of ϒ and C are 1, intensity values do not change. The value of c can be thought as a scaler.

Comment on the similarity between log transformation and power-law transformation.

Answer) log-transformation and power-law transformation with fractional values of gamma both maps a narrow range of dark input values into a wider range of output values, with the opposite being true for brighter values of input level. Unlike power-law function.

Can power-law transformation replace log transformation?

Answer)

Both achieve similar effect. However, power-law transformation can't replace log transformation because log-transformation has an important characteristic, called dynamic range compression. It compresses the dynamic range of images with large variations in pixel values.

4) Describe what bitplane slicing does.

Bitplane slicing refers to a process of creating multiple bitplane images where each bitplane image highlights the contribution made to the total image appearance by specific bit. For example, an 8-bit image can be divided into 8 different bitplane images where each bitplane image corresponds to a specific bit. Below is an example of a bitmap slicing 8-bit image.

Original image	Bit plane 1(Least significant bit)	bit plane 2
bit plane 3	bit plane 4	bit plane 5
bit plane 6	bit plane 7	bit plane 8

Does the most significant bit always contribute the most?

Answer)

No. The most significant bit does not always contribute the most. It depends on the image.

5) Comment on the difference (pros/cons) between contrast stretching and histogram equalization. Can they replace each other?

Answer) Contrast stretching is simple to implement compared to histogram equalization. However, it is restricted to a linear mapping of input to output values. Therefore resulting image is less harsh and tends to avoid artificial appearance of equalized images. With contrast stretching, you need to manually determine parameters to achieve desired effects. This can be cumbersome; however, you have more control of the process. Contrast stretching can also be applied to a restricted range of of the input values.

Histogram equalization is relatively harder to implement, and histogram equalization works automatically. This can be an advantage of histogram equalization because the process is automatic; however, it can also be a disadvantage because you do not have control over it. There are many techniques such as histogram specification to overcome this problem. Another disadvantage of histogram equalization is that it is applied to the whole image.

No. They can't replace each other. They produce similar result but not the same result. For example, contrast stretching is only restricted to mapping a linear mapping of of input to output values.

6) Show the derivation for histogram equalization.

Answer)
Let r denote intensity values in the original image and s denote intensity values in the enhanced image. We want to find a monotonically increasing pixel brightness transformation s = T(r) such that the resulting histogram is uniform.

The intensity levels in an image may be considered as random variables in the interval [0,L-1] where L-1 is the maximum intensity value representing white. And we can use Probability Density Function(PDF) to represent PDF of r and s that,

$$p_s(s)=p_r(r)\left|\frac{dr}{ds}\right| \quad \text{— (a)}$$

If we use the cumulative distribution function as the transformation function T(r), we get

$$s=T(r)=(L-1)\int_0^r p_r(w)\,dw \quad \text{- (b)}$$

where w is a dummy variable of the integral, and the right side of this equation is Cumulative Distribution Function(CDF) of random variable r. T(r) is monotonically increasing function because the area under the function always increases as the value of PDF p_r is always positive. And the value of s is within the range [0,L-1] as the integral evaluates to 1.

$\left|\frac{ds}{dr}\right|$ in equation (a) can be expressed as below.

$$\left|\frac{ds}{dr}\right|=\frac{dT(r)}{dr} \quad \text{- (c)}$$

And, we substitute the value of T(r) into equation (c) using the result from equation (b), then we get

$$|\frac{dT(r)}{dr}|=(L-1)\frac{d}{dr}[\int_0^r p_r(w)dw]=(L-1)p_r(r) \quad \text{-(d)}$$

We can put the result from equation (d) into equation (a)

$$p_s(s)=p_r(r)|\frac{dr}{ds}|=p_r(r)|\frac{1}{(L-1)p_r(r)}|=\frac{1}{L-1} \quad \text{-(e)}$$

Since s is in the range [0,L-1], we can infer from the equation (e) such that $P_s(s)$ is a uniform probability density function.

Since $P_s(s)$ is a uniform probability density function, we know that performing the transformation function in (b) will give us a random variable, s with uniform PDF.

For discrete values, we have

$$p_r(r_k)=\frac{n_k}{MN} \quad \text{and} \quad s_k=T(r_k)=(L-1)\sum_{i=1}^{K} p_r(r_j)=\frac{(L-1)}{MN}\sum_{i=1}^{K} n_i$$

where k = 0,1,2,3,....,L-1.

T(r) is a monotonically increasing function as n_j is always positive. The value of s is always in the range of [0,L-1] because $\frac{1}{MN}*\sum_{i=1}^{K} n_i=1$

7) Prove equations 2.6-6 and 2.6-7 on page 75 to help understand the image averaging algorithm.

Answer)

Proof of equation 2.6-6)

From equation 2.6-4, we know that g(x,y) is formed by adding a noise η(x,y) to the original image f(x,y).

$$g(x,y)=f(x,y)+\eta(x,y)$$

Equation 2.6-5 states that an image $\bar{g}(x,y)$ is formed by averaging K different noisy images.

$$\bar{g}(x,y)=\frac{1}{K}\sum_{i=1}^{K} g_i=\frac{1}{K}\sum_{i=1}^{K} f_i+\frac{1}{K}\sum_{i=1}^{K} \eta_i$$

We apply E{} on the both sides, then we get

$$E\{\bar{g}(x,y)\}=\frac{1}{K}\sum_{i=1}^{K} E\{f_i\}+\frac{1}{K}\sum_{i=1}^{K} E\{\eta_i\}$$

$E\{f_i\}$ is f_i since all the f_i are the same image. And $E\{n_i\}$ is 0 since the noise has a zero mean. Then, we get

$$E\{\bar{g}(x,y)\}=f(x,y)$$

Therefore, the equation 2.6-6 is valid.

Proof of equation 2.6-7)

From equation 2.6-5, we know that

$$\bar{g}(x,y)=\frac{1}{K}\sum_{i=1}^{K} g_i=\frac{1}{K}\sum_{i=1}^{K} f_i+\frac{1}{K}\sum_{i=1}^{K} \eta_i$$

According to random variable theory, the variance of the sum of uncorrelated random variable is the sum of the variances of those variables.

$$\sigma^2_{\bar{g}(x,y)} = \frac{1}{K^2}[\sigma^2_{f_1} + \sigma^2_{f_2} + \sigma^2_{f_3} \ldots + \sigma^2_{f_K}] + \frac{1}{K^2}[\sigma^2_{\eta_1} + \sigma^2_{\eta_2} + \sigma^2_{\eta_3} \ldots + \sigma^2_{\eta_K}]$$

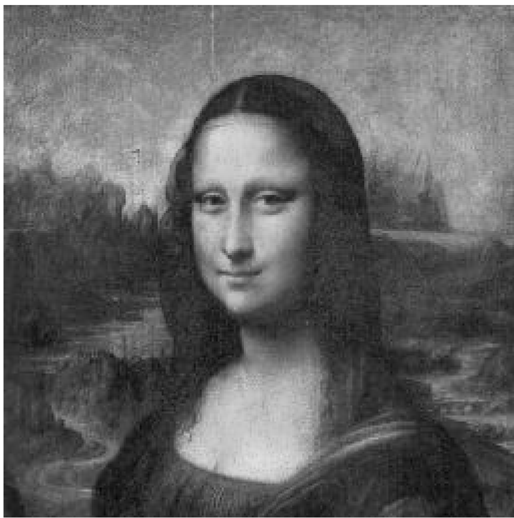
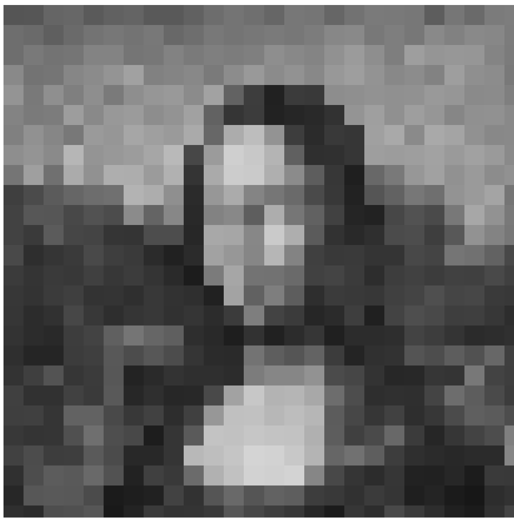
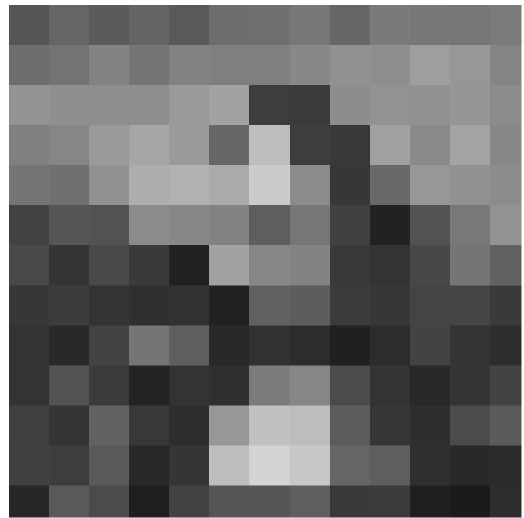
All the f_i has a variance of 0 since they are all constant and η is uncorrelated random variable so each $\sigma^2_{\eta_i}$ is sample of the noise with variance σ^2_{η}, we get,

$$\sigma^2_{\bar{g}(x,y)} = \frac{K}{K^2}[\sigma^2_{\eta}] = \frac{1}{K}\sigma^2_{\eta}$$

Therefore, the equation 2.6-7 is valid.

Results from Task 1

Task 1.1) sampling()

 Figure 1.1a: Original Image	 Figure 1.2b: downsampling by 5(s=5)
 Figure 1.2b: downsampling by 10(s=10)	 Figure 1.2b: downsampling by 20(s=20)

Task 1.1) quantization()

 Figure 1.3a: Original Image	 Figure 1.3b: Quantization with 2 levels(q=2)

Figure 1.3c: Quantization with 4 levels

Figure 1.3d: Quantization with 8 levels(q=8)

Task 1.2) logtran() & powerlaw()

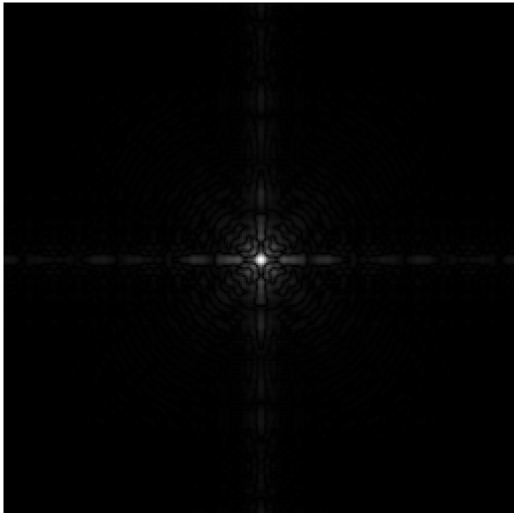


Figure 1.2a: original image

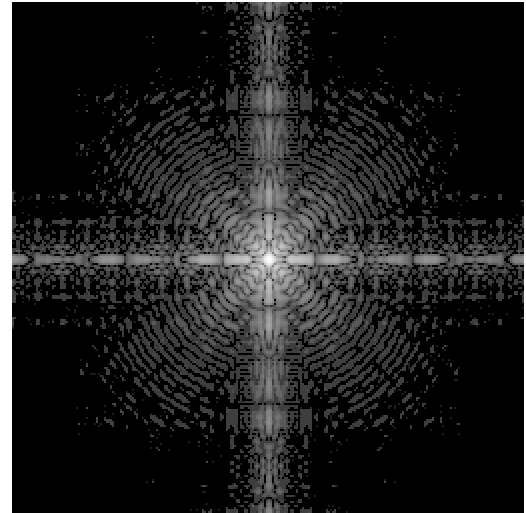


Figure 1.2b: log-transformation(c=1)

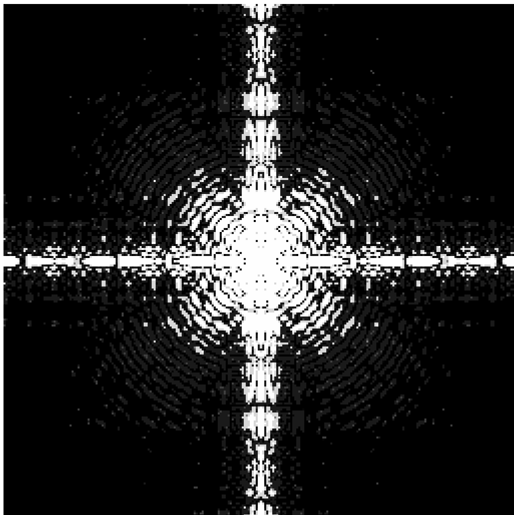


Figure 1.2c: powerlaw(c=1,r=3)

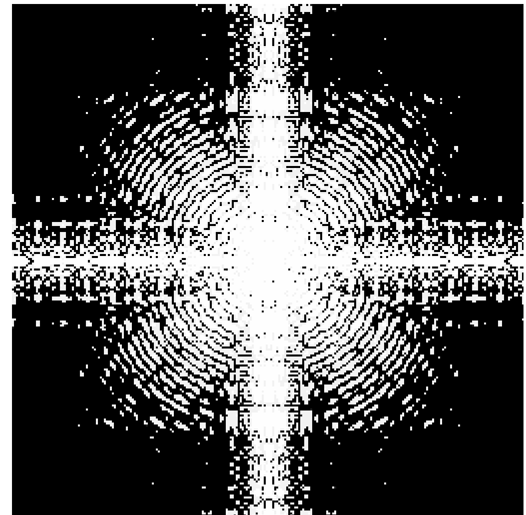


Figure 1.2d: powerlaw(c=1,r=5)

Task 1.3)
Here is a histogram of bunker image.

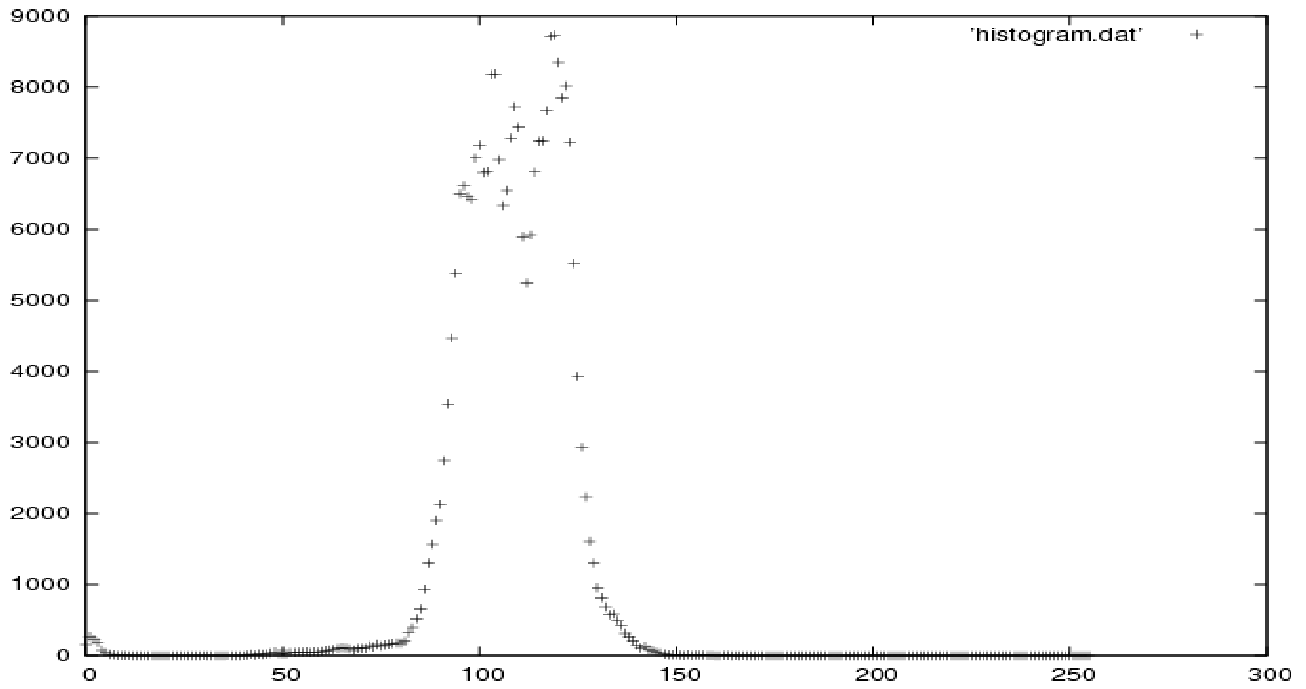
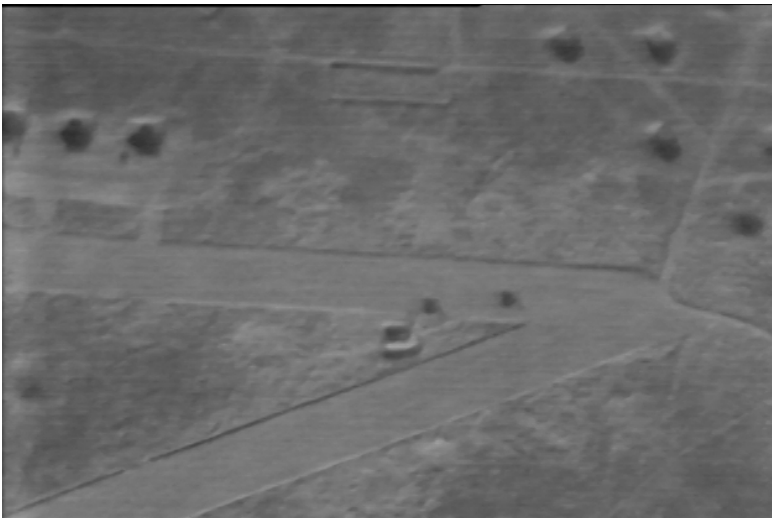
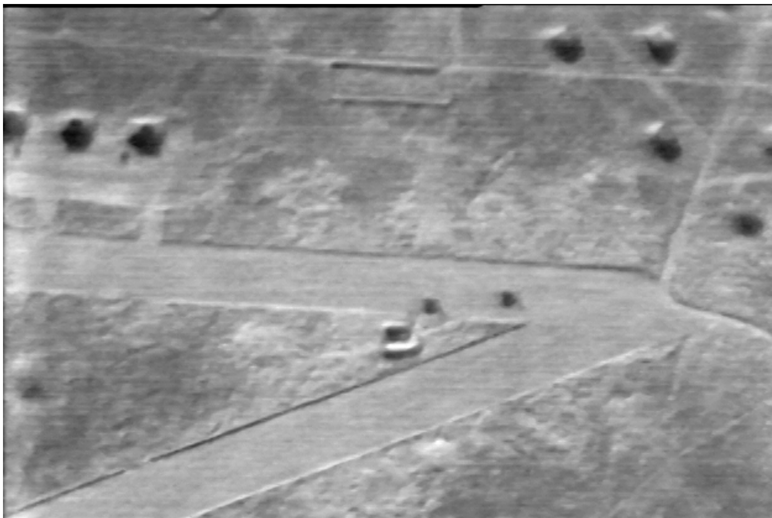


Figure 1.3a: Histogram of original image

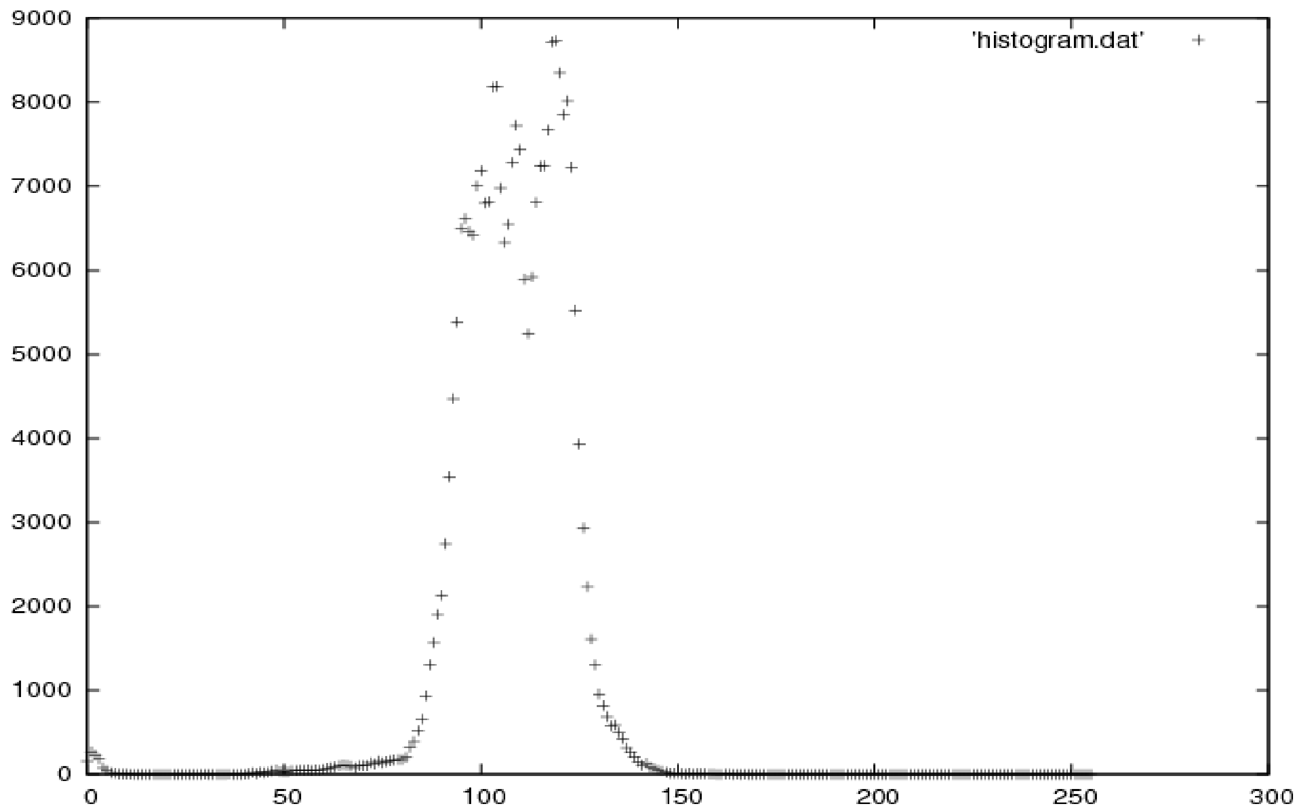
As you can see, its intensity levels are clustered around 70~135. I used the value of 2.3 for the slope and -100 for x-intercept to generate a new image. Below is a histogram of new image.

Figure 1.4 b: Histogram of contrast stretched image (m=2.3, b=-100)

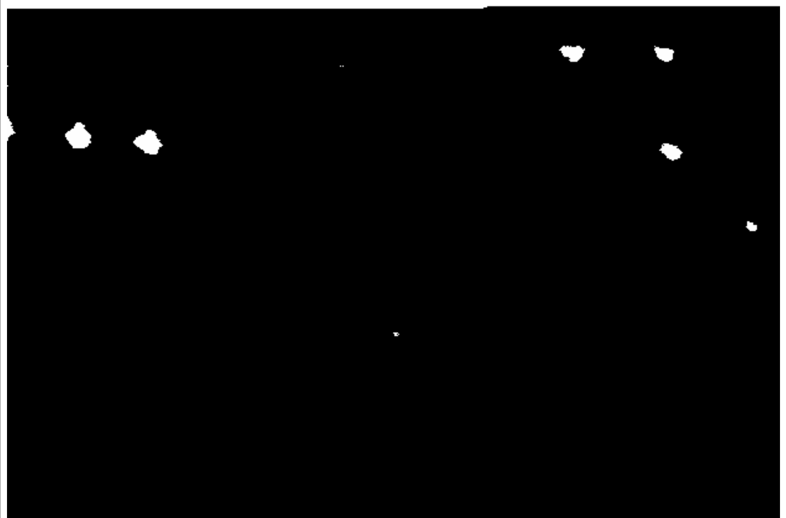
The histogram of a new image is more spread out and it is centered around 125. Original image and contrast stretched image are displayed below:

 Figure 1.3c: Original Image	 Figure 1.3d: Contrast stretched image (m=2.3, b=-100)

Task 1.4)
Here is a histogram of bunker image.

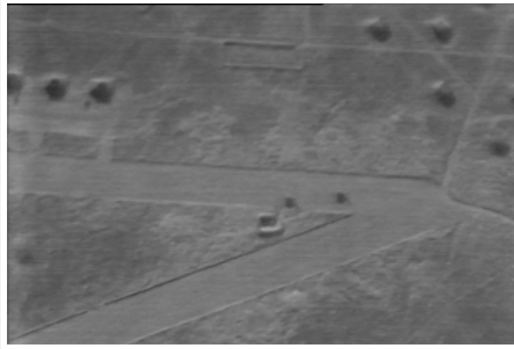
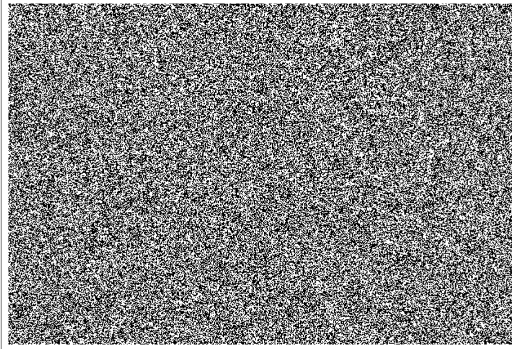
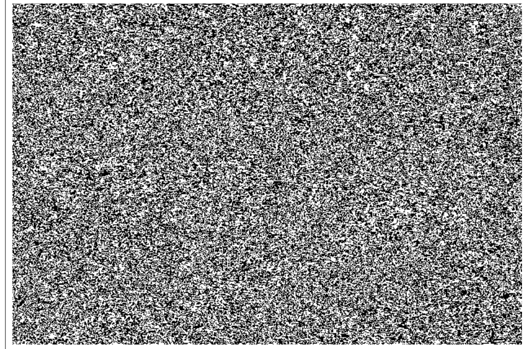
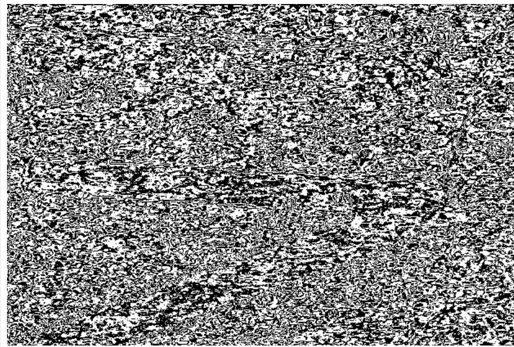
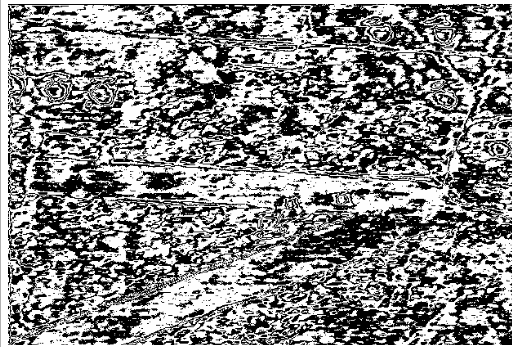
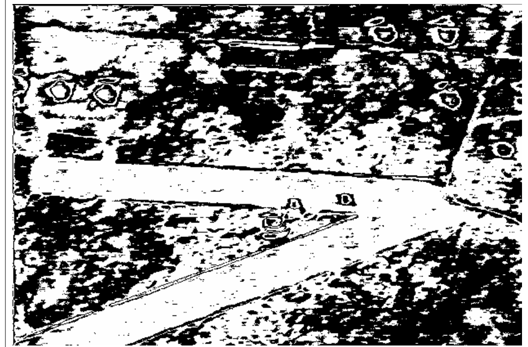
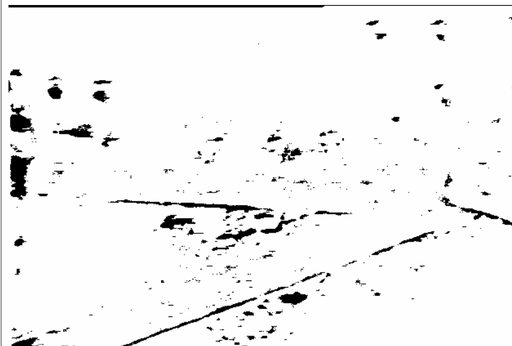
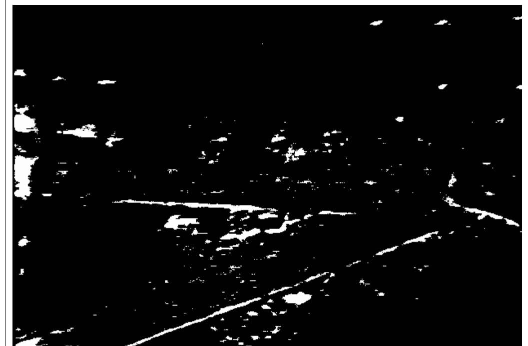


Photos generated with threshold() function are displayed below. Threshold() function highlights all the values in the range of interest. The pictures in the first column keeps other intensity level unchanged whereas the picture in the second column set their values to 0. We used the range of (0,69) in the first row, and the range of (0,55) in the second row.

	
(A=0,B=69, keepintensity=1)	(A=0,B=69, keepintensity=0)

(A=0,B=55, keepintensity=1)	(A=0,B=55, keepintensity=0)

Task 1.5) Original image of bunker.pgm and Bit planes 1 through 8 with bit plane 1 corresponding to the least significant bit

<table>
<tr><td>
Original image</td><td>
Bit plane 1(Least significant bit)</td><td>
bit plane 2</td></tr>
<tr><td>
bit plane 3</td><td>
bit plane 4</td><td>
bit plane 5</td></tr>
<tr><td>
bit plane 6</td><td>
bit plane 7</td><td>
bit plane 8</td></tr>
</table>

Task 2)Histogram Equalization

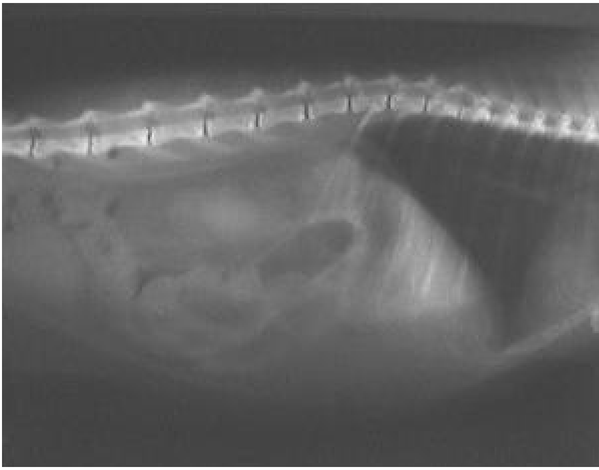


Figure 2.1a: Original Image

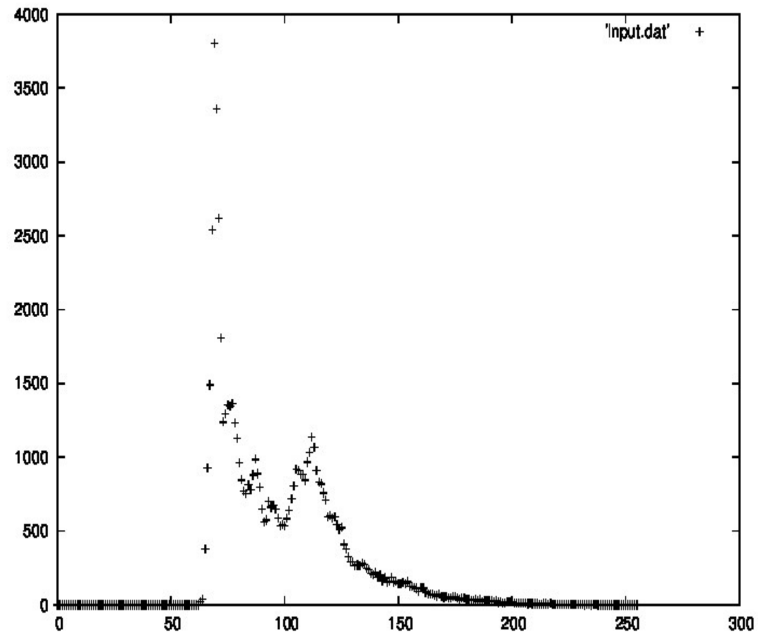


Figure 2.1b: Histogram of Original Image

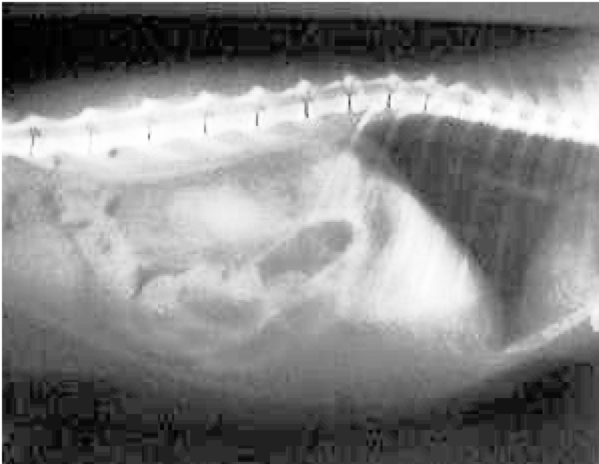


Figure 2.2a: Enhanced Image

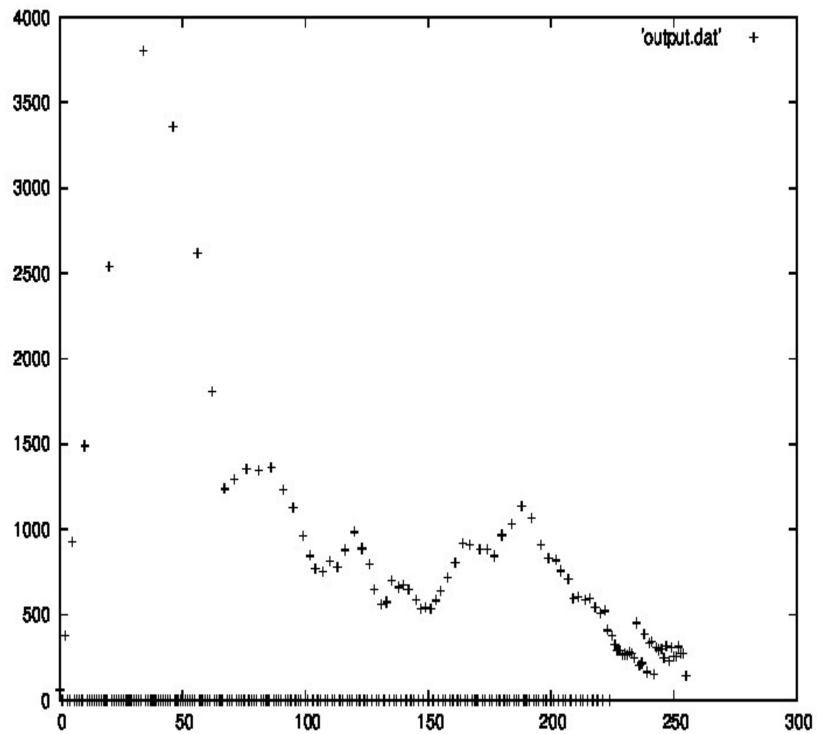


Figure 2.2b: Histogram of Enhanced Image

I implemented printHistogram(Image,char*) function in utility.cpp to print a histogram data of given image to a file. Histograms were generated with gnuplot as directed on the project2 description document.

Task3) Gaussian noise

Here is an original image and five noisy images generated by adding Gaussian noise with 50 standard deviation.

 Original Image	 Image #1 with Gaussian noise(sd=50)	 Image #2 with Gaussian noise(sd=50)
 Image #3 with Gaussian noise(sd=50)	 Image #4 with Gaussian noise(sd=50)	 Image #5 with Gaussian noise(sd=50)

*Image average of 5 noisy images

Averaged Image of 5 noisy images

This image was generated by averaging five noisy images. As you can see, there is less noise in the averaged image compared to all those five noisy images. This is because Gaussian noise has a zero mean.

I increased the number of noisy images and generated an averaged images:
1) Image average of 10 noisy images (standard deviation sd=50)

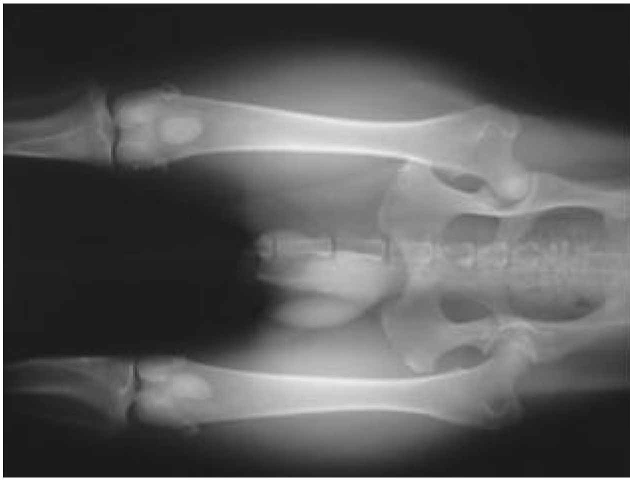
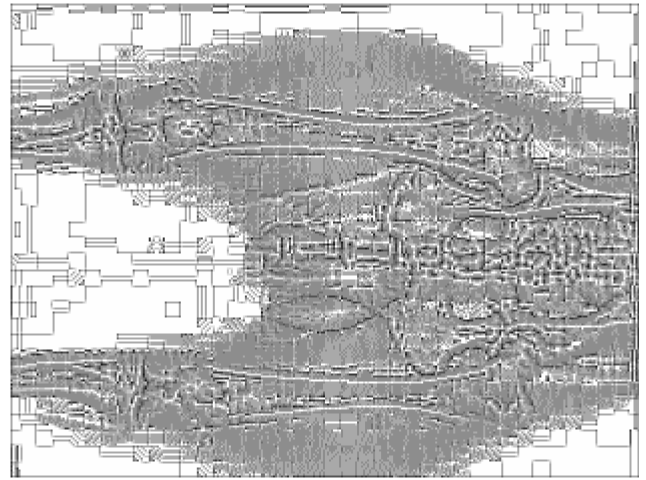
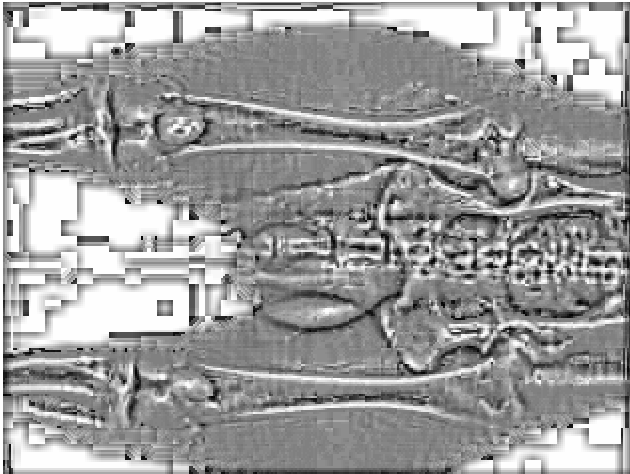
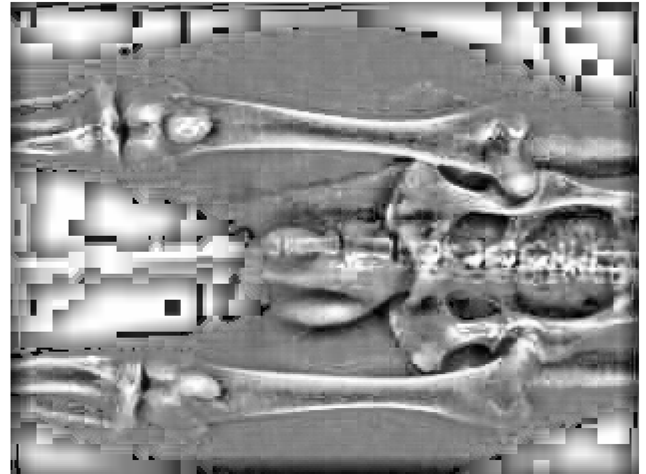
Averaged Image of 10 noisy images

2) Image average of 50 noisy images (standard deviation sd=50)

Averaged Image of 50 noisy Images

As you can see, the enhancement effect is more pleasing with more number of noisy images.

Bonus) local histogram equalization.

<table>
<tr>
<td>
Figure B.1a: original image</td>
<td>
Figure B.1b: local histogram equalization with 3x3mask</td>
</tr>
<tr>
<td>
Figure B.1c: local histogram equalization with 9x9 mask</td>
<td>
Figure B.1c: local histogram equalization with 21x21mask</td>
</tr>
<tr>
<td>
Figure B.1d: local histogram equalization with 101x101 mask</td>
<td></td>
</tr>
</table>

Conclusion)

In this project, a few point processing algorithms, a histogram equalization, and image averaging technique were studied and also implemented. Many photos were generated with each technique to study their characteristics. I found histogram equalization technique very interesting. It was cumbersome to implement it but its effect was impressive as it did not require any manipulation like other point processing techniques. Overall, I have learned a great deal about point processing algorithms and a histogram equalization in this project.

```
/**********************************************************
 * addNoise.cpp - noise generating function
 *
 *   - gaussianNoise: generate Gaussian noise
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 09/09/2011
 *
 * Modified:
 **********************************************************/

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include "utility.h"

using namespace std;

/**
 * Gaussian noise function. It generates Gaussian noise with zero mean and
 * sigma standard deviation using BOX-MULLER METHOD.
 *
 * @param inimg input image
 * @param sd standard devication
 * @return an image with gaussian noise
 */
Image gaussianNoise(Image &inimg,float sd) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;
  double uniform1=0, uniform2=0; //two uniformly distributed random numbers
  double normal1=0,normal2=0;
  double actual1=0,actual2=0;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  outimg.createImage(nr, nc, ntype);

  //apply gaussian noise
  //this code is written for readability not for optimization
  for (i=0; i<nr; i+=2)
    for (j=0; j<nc; j++)
      for (k=0; k<nchan; k++) {
        //1) generates two uniformly distributed random numbers
        while(uniform1 == 0.0)  uniform1 = 1.0 *random()/RAND_MAX;
        uniform2 = 1.0 *random()/RAND_MAX;

        //2) Uses BOX-MULLER technique of inverse transformation to turn
        //   uniformly distributed random numbers into two unit normal random numbe
rs.
        normal1=sqrt(-2*log(uniform1)) * cos(2 * PI * uniform2);
        normal2=sqrt(-2*log(uniform1)) * sin(2 * PI * uniform2);

        //3) modify unit normal random numbers according to given mean and variance
.
        //   we use a zero mean and sigma standard devision
        //   we use boundaryCheck function I defined for project1 to ensure that th
e pixel intensity doesn't go over bounary.
        //   since we have two different normal random numbers, we apply each of th
em to different rows.
        outimg(i,j,k) = boundaryCheckF(inimg(i,j,k)+(int)(normal1*sd),0,255);
        if(i!=nr-1) outimg(i+1,j,k) = boundaryCheckF(inimg(i+1,j,k)+(int)(normal2*s
d),0,255);

      }

  return outimg;
}
```

```
/**********************************************************
 * pointProcessing.cpp - point processing functions for grey-sclae images
 *
 *   - sampling: image downsample/subsample
 *   - quantization: image quantization
 *   - logtran: log transformation
 *   - powerlaw: power-law(gamma) transformation
 *   - cs: contrast streching
 *   - threshold: grey-level slicing in threshold
 *   - bitplane: bitplane slicing
 *   - histeq: histogram equalization
 *   - lohisteq: local histogram equalization
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 09/02/2011
 *
 * Modified:
 **********************************************************/
#include "Image.h"
#include "Dip.h"
#include "utility.h"
#include <iostream>
#include <cstdlib>
#include <cstring>

using namespace std;

/**
 * Contrast stretching. s = m*r + b where
 * s: enhanced pixel intensity
 * r: original pixel intensity
 * @param inimg Input image
 * @param m Slope
 * @param b Intercept
 * @return Contrast stretched image.
 */
Image cs(Image &inimg, float m, float b) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  outimg.createImage(nr, nc, ntype);

  // perform contrast stretching
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++)
      for (k=0; k<nchan; k++)  {
        outimg(i,j,k) = boundaryCheckF((m * inimg(i,j,k) + b),0,L);
      }
  return outimg;
}
```

need to have a user input start and end intensity

```
/**
 * Local histogram equalization with given neighborhood size
 * It works by as below:
 * 1) Define a neighborhood and move its center from pixel to pixel
 * 2) At each location, the histogram of the points in the neighboorhood is compute
d.
 * 3) Histogram equalization transformation function is obtained
 * 4) Use this function to map the intensity of the pixel centered in the neighborh
ood
 * @param inimg input image
 * @param neighbor size of neighborhood
 * @return an image with local histogram equalization
 */
Image lohisteq(Image &inimg,int neighbor) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;
  int h[(int)L+1];
  int halfneighbor = (neighbor-1)/2;
  int starti,startj;
  int endi,endj;
  int li,lj;

 //alloate memory
  nr = inimg.getRow(); //height-row
  nc = inimg.getCol(); //width-column
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  outimg.createImage(nr, nc, ntype);

  //For each pixel, apply local histogram equalization
  for(i=0;i<nr;i++) {
    for(j=0;j<nc;j++) {

      //clear histogram array
      memset(h,0, (int)(L+1) * sizeof(int));

      //1) compute the histogram of the points in the neighboorhood
      //work out the range of pixels in the neighboorhood
      //make sure it does not go out of image range
      starti = i- halfneighbor<0?0:i-halfneighbor;
      startj = j- halfneighbor<0?0:j-halfneighbor;
      endi = i + halfneighbor>=nr?nr-1:i+halfneighbor;
      endj = j + halfneighbor>=nc?nc-1:j+halfneighbor;

      for (li=starti; li<=endi; li++)
        for (lj=startj; lj<=endj; lj++)
          h[(int)inimg(li,lj,0)]++;

      //2)conver to CDF
      for(li=1;li<=L;li++)
        h[li]+=h[li-1];
```

```
      //3)assign pixel value at the center
      outimg(i,j,0) = roundf((float)h[(int)inimg(i,j,0)]*L / (neighbor*neighbor) );
    }
  }

  return outimg;
}

/**
 * Histogram equalization
 *
 * @param inimg input image
 * @return an image with histogram eqaulization
 */
Image histeq(Image &inimg) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;
  int h[(int)L+1];

 //alloate memory
  nr = inimg.getRow();
```

```
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();
  memset(h,0, (int)(L+1) * sizeof(int));

  outimg.createImage(nr, nc, ntype);

  //1) count histogram function h
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++)
      h[(int)inimg(i,j,0)]++;

   //2)conver to CDF
  for(i=1;i<=L;i++)
    h[i]+=h[i-1];

  //3)assign pixel value
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++)
      outimg(i,j,0) = roundf((float)h[(int)inimg(i,j,0)]*L / (nr*nc) );

  return outimg;
}

/**
 * bitplane-slicing creates seprate images highlighting the contribution made
 * to the total image apperance by specific bits.
 *
 * @param inimg Input image
 * @return an NBIT-size array of images which correspods to each bit.
 */
Image* bitplane(Image &inimg) {
  Image* outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  //creates an NBIT-array of images
  outimg = new Image[NBIT]; //created in heap
  for(i=0;i<NBIT;i++) outimg[i].createImage(nr, nc, ntype);

  //perform bitmap slicing
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++)
      for (k=0; k<NBIT; k++) //for each bit, extract appropriate bit using the bit
mask
        outimg[k](i,j,0) = (int)inimg(i,j,0) & (0x1<<k);
  return outimg;
}
```

or (1 & (inimg(i,j,0)>>k))

```
/**
 * Grey-level Slicing
 * s = L if A<=R<=B
 * s = 0 or r otherwise (r when you want to keep all other intensity levels unchang
ed)
 * s: enhanced pixel intensity
 * r: original pixel intensity
 * [A,B]: range to highlight
 * @param inimg Input image
 * @param A lower limit of range
 * @param B upper limit of range
 * @param keepintensity flag used to decide whether to keep all other intensity lev
els unchanged. DEFAULT=TRUE
 * @return image with grey-level slicing effect
 */
Image threshold(Image &inimg, float A, float B,bool keepintensity) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  outimg.createImage(nr, nc, ntype);

  //perform grey-level slicing
  // s = L if A<=R<=B
  // s = (0 or r) otherwise (s = r when you want to keep all other intensity levels
 unchanged)
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++)
        outimg(i,j,0) = inimg(i,j,0)>=A&&inimg(i,j,0)<=B?L:((keepintensity)?inimg(i
,j,0):0);
  return outimg;
}



/**
 * Power-law Transformation. s = c*r^g where
 * s: enhanced pixel intensity
 * r: original pixel intensity
 * c: constant
 * g: gamma value
 * @param inimg Input image
 * @param c constant
 * @param g gamma value
 * @return image with powerlaw transformation effect.
 */
Image powerlaw(Image &inimg, float c, float g) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  outimg.createImage(nr, nc, ntype);

  //perform log transformation using s = c * r^g
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++)
        outimg(i,j,0) = c * powf(inimg(i,j,0),g);

  return outimg;
}

/**
 * Log transformation. s = c*log(1+r) where
 * s: enhanced pixel intensity
 * r: original pixel intensity
 * c: constant
 * @param inimg Input image
```

```
 * @param c constant
 * @return image with log-transformation effect.
 */
Image logtran(Image &inimg, float c) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  outimg.createImage(nr, nc, ntype);

  //perform log transformation using s = c * log(1+r)
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++)
        outimg(i,j,0) = c * logf(inimg(i,j,0)+1);

  return outimg;
}

/**
 * Quantization to q quantization levels using the equation
 * s = floor(r / L+1) * q
 * where s is the enhanced pixel intensity
 *       r is the original pixel intensity
 *       L is the maximum component level
 *       q is new quantization level
 *       floor(x) gives largest integral value not greater than x
 * @param inimg Input image
 * @param q required quantization levels
 * @return new image with caricature effect.
 */
Image quantization(Image &inimg,int q) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  outimg.createImage(nr, nc, ntype);

  // perform quantization
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++)
      //use the equation s = floor((r/L) * (q-1))
      outimg(i,j,0) = floor(inimg(i,j,0)/(float)(L+1)*(float)q);

  return outimg;
}

/**
 * Downsampling effect by s where s>1
 *
 * @param inimg Input image
 * @param s ratio to sample by
 * @return new image with caricature effect.
 */
Image sampling(Image &inimg,int s) {
  Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;
  int icounter; //for loop counter used for downsampling
  int jcounter;

  // allocate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();

  outimg.createImage(nr, nc, ntype);

  // perform downsampling by s
  for (i=0; i<nr; i+=s) //increment by s
    for (j=0; j<nc; j+=s) //increment by s
      //same intensity within range (i,j)~(i+s-1,j+s-1)
      for (icounter=i;icounter<s+i && icounter<nr;icounter++)
        for(jcounter=j;jcounter<s+j && jcounter<nc;jcounter++)
            outimg(icounter,jcounter,0) = inimg(i,j,0);


  return outimg;
}
```

$\frac{r.q}{L+1} \cdot \frac{L}{q-1}$

-2

```
/****************************************************************
 * testbs.cpp: test code for bitplane-scaling
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 ****************************************************************/

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>

using namespace std;

#define Usage "testgs inimg outimg_prefix \n"

int main(int argc, char **argv)
{
  Image inimg, *outimg;    // the original image
  char* filename;
  Image resimg; //rescaled image
  int i;

  // check if the number of arguments on the command line is correct
  if (argc < 2) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  // test the caricature function
  outimg = bitplane(inimg);

  //prepare the buffer to hold file name
  filename = (char*)malloc(strlen(argv[2])+10);

  // output the image
  for(i=0;i<NBIT;i++) {
    //creates a name
    sprintf(filename,"%s%d.pgm",argv[2],i); //it always ends with pgm as bit-slicin
g is only applied to gray-scale image in our code.
    resimg = rescale(outimg[i]);
    writeImage(resimg, filename);
  }

  return 0;
}
```

```
// test code for contrast stretching

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
#include "utility.h"

using namespace std;

#define Usage "testcs inimg outimg slope intercept\n"

int main(int argc, char **argv)
{
  Image inimg, outimg;    // the original image
  float m, b;

  // check if the number of arguments on the command line is correct
  if (argc < 5) {
    cout << Usage;
    exit(3);
  }

  // read in command-line arguments
  m = atof(argv[3]);
  b = atof(argv[4]);

  // read in image
  inimg = readImage(argv[1]);

  string filename(argv[1]);
  string ext=".dat";

  // print out the histogram of input image
  printHistogram(inimg,(filename+ext).c_str());

  // test the contrast stretching function
  outimg = cs(inimg, m, b);

  //print out the histogram of output image
  filename = string(argv[2]);
  printHistogram(outimg,(filename+ext).c_str());
  // output the image
  writeImage(outimg, argv[2]);

  return 0;
}
```

```
/**************************************************************
 * testgn.cpp: test code for gaussian noise
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 ***************************************************************/

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>

using namespace std;

#define Usage "testgs inimg outimg_prefix sd numberofImages \n"

int main(int argc, char **argv)
{
  Image inimg, outimg;    // the original image
  char* filename;
  int i;
  int numberOfImages;
  Image avgimg; //average image;
  float sd;

  // check if the number of arguments on the command line is correct
  if (argc < 5) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  // read in number of images
  numberOfImages = atoi(argv[4]);

  //read in standard deviation
  sd = atof(argv[3]);

  // output filename = out
  filename = (char*)malloc(strlen(argv[2])+10);

  //print out the given number of images with gaussian noise.
  for(i=0;i<numberOfImages;i++) {
    // apply gaussian noise
    outimg = gaussianNoise(inimg,sd);

    //prepare the buffer to hold file name
    filename = (char*)malloc(strlen(argv[2])+10);

    //creates a file name
    sprintf(filename,"%s%d.pgm",argv[2],i);
    writeImage(outimg, filename);

    //add all images for image averaging effect later
    if(i==0) avgimg = outimg;
    else avgimg= avgimg + outimg;
  }
```

```
  //apply image averaging by dividing sum of all images by the number of images
  avgimg = avgimg / numberOfImages;
  sprintf(filename,"%s_av_%d.pgm",argv[2],numberOfImages);
  writeImage(avgimg,filename);


  return 0;
}
```

```
/***************************************************************
 * testgs.cpp: test code for grey-scaling(thresholding)
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 ***************************************************************/

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
#include "utility.h"

using namespace std;

#define Usage "testgs inimg outimg A B flag\n"

int main(int argc, char **argv)
{
  Image inimg, outimg;    // the original image
  float A;
  float B;
  bool keepintensity=true;

  // check if the number of arguments on the command line is correct
  if (argc < 6) {
    cout << Usage;
    exit(3);
  }

  // read in command-line arguments
  A = atof(argv[3]);
  B = atof(argv[4]);
  keepintensity = atoi(argv[5])==1; //true when 1. otherwise false

  // read in image
  inimg = readImage(argv[1]);

  string filename(argv[1]);
  string ext=".dat";

  // print out the histogram of input image
  printHistogram(inimg,(filename+ext).c_str());


  // test the thresholding
  outimg = threshold(inimg, A,B,keepintensity);

  // output the image
  writeImage(outimg, argv[2]);

  return 0;
}
```

```
/***************************************************************
 * tesths.cpp: test code for histogram equalization
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 ***************************************************************/

#include "Image.h"
#include "Dip.h"
#include "utility.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>

using namespace std;

#define Usage "tesths inimg outimg \n"

int main(int argc, char **argv)
{
  Image inimg, outimg;    // the original image

  // check if the number of arguments on the command line is correct
  if (argc < 2) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);
  // print the histogram of input image
  printHistogram(inimg,"input.dat");
  // test the histogram
  outimg = histeq(inimg);

  // print out function
  writeImage(outimg, argv[2]);
  // print the histogram of output image
  printHistogram(outimg,"output.dat");
  return 0;
}
```

```
/*************************************************************
 * testlhs.cpp: test code for local histogram equalization
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 *************************************************************/

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>

using namespace std;

#define Usage "testlhs inimg outimg neighborSize\n"

int main(int argc, char **argv)
{
  Image inimg, outimg;    // the original image
  int neighbor;
  // check if the number of arguments on the command line is correct
  if (argc < 2) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  // read in neighbor size
  neighbor = atoi(argv[3]);

  // test the histogram
  outimg = lohisteq(inimg,neighbor);

  // print out function
  writeImage(outimg, argv[2]);

  return 0;
}
```

```
/***************************************************************
 * testlt.cpp: test code for log transformation
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 09/02/11
 *
 ***************************************************************/

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>

using namespace std;

//message to be printed when incorrect argument is given
#define Usage "testlt inimg outimg c\n"

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

  Image inimg, outimg;    // the original image, output image
  Image resimg; //rescaled image
  float c;

  // check if the number of arguments on the command line is correct
  if (argc < 4) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  // read in a consant
  c = atof(argv[3]);

  // test the negative effect function
  outimg = logtran(inimg,c);

  //rescale the image
  resimg = rescale(outimg);

  // output the image
  writeImage(resimg, argv[2]);

  return 0;
}
```

```
/***************************************************************
 * testni.cpp: test code for negative image
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 ***************************************************************/

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
using namespace std;

//message to be printed when incorrect argument is given
#define Usage "testni inimg outimg\n"

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

  Image inimg, outimg;    // the original image

  // check if the number of arguments on the command line is correct
  if (argc < 3) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  // test the negative effect function
  outimg = ni(inimg);

  // output the image
  writeImage(outimg, argv[2]);

  return 0;
}
```

```
/***************************************************************
 * testpl.cpp: test code for power-law transformation
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 09/02/11
 *
 ***************************************************************/

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>

using namespace std;

//message to be printed when incorrect argument is given
#define Usage "testpl inimg outimg c gamma\n"

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

  Image inimg, outimg;    // the original image, output image
  float c; //constant
  float gamma; //gamma value

  // check if the number of arguments on the command line is correct
  if (argc < 5) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  // read in a consant
  c = atof(argv[3]);
  gamma = atof(argv[4]);

  // test the negative effect function
  outimg = powerlaw(inimg,c,gamma);

  // output the image
  writeImage(outimg, argv[2]);

  return 0;
}
```

```
/*************************************************************
 * testqt.cpp: test quantization
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 *************************************************************/

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
using namespace std;

//message to be printed when incorrect argument is given
#define Usage "testqt inimg outimg quantization\n"

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

  Image inimg, outimg;    // the original image, image with effect
  Image resimg;
  int q;

  // check if the number of arguments on the command line is correct
  if (argc < 4) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  // quantization level
  q = atoi(argv[3]);

  // test the negative effect function
  outimg = quantization(inimg,q);

  // rescale the image
  resimg = rescale(outimg);

  // output the image
  writeImage(resimg, argv[2]);

  return 0;
}
```

```
/***************************************************************
 * testsampling.cpp: test downsampling
 *
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 08/31/11
 *
 ***************************************************************/

#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <cstdlib>
using namespace std;

//message to be printed when incorrect argument is given
#define Usage "testsampling inimg outimg ratio\n"

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

  Image inimg, outimg;    // the original image
  int s;

  // check if the number of arguments on the command line is correct
  if (argc < 4) {
    cout << Usage;
    exit(3);
  }

  // read in image
  inimg = readImage(argv[1]);

  //ratio to downsample by
  s = atoi(argv[3]);

  // test the negative effect function
  outimg = sampling(inimg,s);

  // output the image
  writeImage(outimg, argv[2]);

  return 0;
}
```

```
/*********************************************************
 * utility.cpp - commonly used supporting functions
 *
 *   - polarToCartesian : converts polar to cartesian coordinate
 *   - cartesianToPolar : converts cartesian to polar coordinate
 *   - boundaryCheck    : ensures that given number is within the given range
 *   - boundaryCheckF   : float-version of boundaryCheck
 *   - printHistogram   : print histogram
 * Author: Sang-hyeb(Sam) Lee (C) slee91@utk.edu
 *
 * Created: 01/11/08
 *
 * Modified:
 *  09/02/11: add boundaryCheckF() implementation by Sanghyeb Lee
 *            add printhisogram() implementation by Sanghyeb Lee
 ********************************************************/
#include "Image.h"
#include "Dip.h"
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <assert.h>
#include <cstring>
#include "utility.h"

using namespace std;

/* Print histogram
 * it prints out histogram of given image to the given file
 * @param inimg input image
 */
void printHistogram(Image& inimg, const char* outputfile) {
    Image outimg;
  int i, j, k;
  int nr, nc, ntype, nchan;
  int h[(int)L+1];
  ofstream file;

 //alloate memory
  nr = inimg.getRow();
  nc = inimg.getCol();
  ntype = inimg.getType();
  nchan = inimg.getChannel();
  memset(h,0, (int)(L+1) * sizeof(int));

  //open file for writing
  file.open(outputfile);
 //only accepts single-channel image.
  if (nchan>1) {
    cout << "printHistogram: can only handle single-channel image\n";
    exit(3);
  }
  outimg.createImage(nr, nc, ntype);

  //open file

  //compute histogram h
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++)
      h[(int)inimg(i,j,0)]++;

  //print histogram to the output file
  for(i=0;i<=L;i++) {
    file<<i<<" "<<h[i]<<endl;
  }

}

/**
 * converts cartesian coordinate to polar coordinate
 *  r = sqrt(x^2+y^2)
 *  a = atan(y/x)
 *  where r is the radius (distance from the center) and a is the angle
 *  and (x,y) is the cartesian co-ordinate of the raster point.
 * @param x the x-coordinate of the point (int)
 * @param y the y-coordinate of the point (int)
 * @param rp the address of the float variable to hold radius
 * @param ap the address of the float variable to hold angle
 */
void cartesianToPolar(int x,int y,float *rp,float *ap) {
  *rp = sqrt(pow(x,2)+pow(y,2));
  *ap = atan2f(y,x);
}
/**
 * converts polar coordinate to cartesian coordinate
 *  x = r * cos(a)
 *  y = -r * sin(a)
 *  where r is the radius (distance from the center) and a is the angle
 *  and (x,y) is the cartesian co-ordinate of the raster point.
 * @param r the radius (float)
 * @param a the angle  (float)
 * @param xp the address of the int variable to hold x-coordinate
 * @param yp the address of the int variable to hold y-coordinate
 */
void polarToCartesian(float r,float a,int* xp,int *yp) {
  *xp = roundf(r * cosf(a));
  *yp = roundf(r * sinf(a));
}

/**
 * check if the given int number is within the given range. Otherwise, change
 * the number to either maximum or minimum so that it is in the range.
 * @param x  the number to check  (float)
 * @param lower the lower limit of the range
 * @param upper the upper limit of the range
 * @return number in the range
 */
int boundaryCheck(int x,int lower,int upper) {
  if(x<lower) return lower;
  else if(x>upper) return upper;
  else return x;
}

/**
 * check if the given float number is within the given range. Otherwise, change
 * the number to either maximum or minimum so that it is in the range.
 * @param x  the number to check  (float)
 * @param lower the lower limit of the range
 * @param upper the upper limit of the range
 * @return number in the range
 */
float boundaryCheckF(float x,float lower,float upper) {
  if(x<lower) return lower;
  else if(x>upper) return upper;
  else return x;
}
```

```
/********************************************************************
 * Dip.h - header file of the Image processing library
 *
 *   - cs: contrast stretching
 *   - ni: negative image
 *   - fe: fish eye effect
 *   - cr: caricature effect
 *   - sampling: downsampling
 *   - quantization: quantization
 *   - logtran: log transformation
 *   - powerlaw: power-law transformation
 *   - cs: contrast streching(TO BE DONE)
 *   - threshold: grey-level slicing
 *   - bitplane: bitplane slicing
 *   - histeq(): histogram equalization
 *   - gaussianNoise(): gaussian noise
 *   - lohisteq(): local histogram equalization
 * Author: Hairong Qi, hqi@utk.edu, ECE, University of Tennessee
 *
 * Created: 01/22/06
 *
 * Modification:
 *  - 08/31/11: add ni() implementation by Sanghyeb Lee
 *              add fe() implementation by Sanghyeb Lee
 *              add cr() implementation by Sanghyeb Lee
 *  - 09/02/11: add sampling() implementation by Sanghyeb Lee
 *              add quantization() implementation by Sanghyeb Lee
 *              add logtran() implementation by Sanghyeb Lee
 *              add powerlaw() implementation by Sangheyb Lee
 *              add threshold() implementation by Sanghyeb Lee
 *              add bitplame() implementation by Sanghyeb Lee
 *  - 09/09/11  add histeq() implementation by Sanghyeb Lee
 *              add gaussianNoise() implementation by Sanghyeb Lee
 *              add lohisteq() implementation by Sanghyeb Lee
 ********************************************************************/

#ifndef DIP_H
#define DIP_H

#include "Image.h"

#define PI 3.1415926


//////////////////////////////////////////////
// point-based image enhancement processing

Image cs(Image &,                     // contrast stretching
         float,                       // slope
         float);                      // intercept

Image ni(Image &);                    // negative image

Image fe(Image &,                     // fish eye effect
         float);                      // R - scale constant


Image cr(Image &,                     // caricature effect
         float);                      // R - scale constant


Image sampling(Image &,               // Downsampling
               int);                  // s - ratio to downsample by


Image quantization(Image &,           // Quantization
                   int);              // q - required quantization level

Image logtran(Image &,                // log transformation
              float);                 // c - constant

Image powerlaw(Image &,               // power-law transformation
               float,                 // c - constant
               float);                // gamma value

Image threshold(Image &,              // grey-slicing function
                float,                // start range A
                float,                // end range B
                bool c=true);         // leave other intensity default

Image* bitplane(Image &inimg);       // bitplane-slicing function
                                     // returns Image[NBIT] arrays

Image histeq(Image &inimg);          // Histogram equalization

Image gaussianNoise(Image &inimg,  //gaussian noise
                    float sd);     //standard deviation

Image lohisteq(Image &inimg,         // local histogram equalization
               int);                  // size of neighborhood

#endif
```

```
/*******************************************************************
 * utility.h - header file of the utility functions
 *
 * Author: Sang-hyeb Lee, slee91@utk.edu, ECE, University of Tennessee
 *
 * Created: 08/31/11
 *
 * Modification:
 *
 *******************************************************************/
#ifndef UTILITY_H
#define UTILITY_H

//////////////////////////////////////////////
// coordinate conversion

//polar to cartesian coordinate
void polarToCartesian(float r, //radius in polar coordinate
                      float a, //angle in polar coordinate
                      int* xp, //address of the variable to hold x-coordinate
                      int *yp); //address of the variable to hold y-coordinate

//cartesian to polar coordinate
void cartesianToPolar(int x, //x-coordinate
                      int y, //y-coordinate
                      float *rp, //address of the variable to hold radius
                      float *ap); //addresss of the variable to hold angle

//////////////////////////////////////////////
// misc

//ensures that given int number is within the range.
int boundaryCheck(int x, //number to check for
                  int lower, //lower limit of the range
                  int upper); //upper limit of the range

//ensures that given float number is within the range.
float boundaryCheckF(float x, //number to check for
                  float lower, //lower limit of the range
                  float upper); //upper limit of the range

//print out histogram to the given file
void printHistogram(Image& inimg, //input image
                    const char* outputfile); //output filename
#endif
```