CS 101
Computer Programming and utilization

IIT BOMBAY

Dr Deepak B Phatak
Subrao Nilekani Chair Professor
Department of CSE, Kanwal Rekhi Building
IIT Bombay

Session 10, Applications of two-dimensional arrays

Friday, September 2, 2011

Overview

- More applications of multidimensional arrays
  - Matrix multiplication
  - Magic square
- Processing digital images
  - Representation of digital images
  - Histogram
  - Contrast enhancement
    - Histogram equalization
    - Recalculation of pixel values
- Program to implement histogram equalization

Matrix Multiplication

Given: m x n matrix A; n x p matrix B,

The matrix product C = AB will be m × p matrix.

$$C[i][j] = \sum_{k=0}^{n-1} A[i][k] * B[k][j]$$

Matrix multiplication …

```
...
    for (i=0; i<m; i=i+1) {
        for (j=0; j<p; j=j+1) {
            C[i][j]=0;
            for (k=0; k<n; k=k+1) {
                C[i][j]=C[i][j]  +  A[i][k]*B[k][j];
             }
         }
     }
...
```

Magic squares

- A magic square is an n x n square matrix containing unique positive integers, where the sum of elements of every row and every column is same

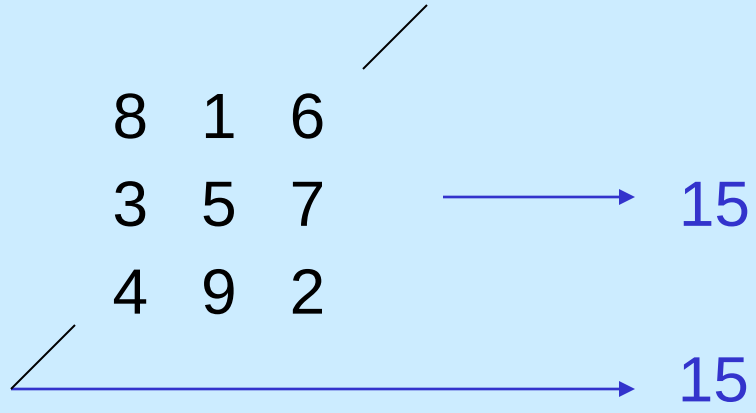
  8 1 6

  3 5 7

  4 9 2

Magic squares

- Additionally, sum of elements on each of the main diagonals is also the same as above



- Known to Chinese (Lo Shu square)
- 3 x 3 magic squares were known to Indians since Vedic times

Determine if the given matrix is a normal magic square

- We note the following:
  - Any n x n matrix will have n rows, n columns
  - It will have two diagonals
- To test given matrix for being a magic square
  - First find out what should be the sum

  sum for basic magic square = n * (n*n +1) /2
  - Then calculate all the row and column sums
  - calculate the sums for two diagonals
  - If any of these sums is not as desired
    - The given matrix is not magic square
- We will need arrays of size n for these sums
  - Now check if each of these sums is equal to the required sum

Program

```
 int square[20][20], N, i, j, sum;
 int  rsum[20], csum[20], d1sum =0, d2sum =0;
/*  Read N, matrix elements; initialize rsum[] and csum[] */
cout << " Give value of N " << endl; cin >> N
cout  << Give elements of the matrix" << endl;
for (i=0; i<N; i++) {
    for (j=0; j< N; j++) {
        cin >> square[i][j];
    }
}
for  (i=0; i < N; i++){
    rsum[i] = 0;
    jsum[i] = 0;
}
```

Program

```
 /* calculate the sum for this being N x N magic square */
sum = N * (N*N +1) /2;

/* find the row sums  and check against required sum*/
for (i=0; i< N; i++){
      for (j=0; j < N; j++){
           rsum[i] += square[i][j];
      }
      if (rsum[i] != sum) {
         cout << " Not a magic square" << endl;   return 1;
      }
}
```

Program …

```
/* find the column sums  and check against required sum*/
 for (j=0; j< N; j++){
      for (i=0; i < N; i++){
           csum[j] += square[i][j];
      }
      if (csum[j] != sum) {
         cout << " Not a magic square" << endl;   return 1;
      }
 }
 /* calculate the sums of two diagonals */
    for (i=0; i< N; i ++) d1sum += square[i][i];
    for (j=0; j < N; j ++) d2sum += square[j][N-j-1];
```

Program …

```
   /* Now check if these diagonal sums are correct or not */
   if (d1sum != sum) {
         cout << " Not a magic square" << endl;   return 1;
   }
   if (d2sum != sum) {
         cout << " Not a magic square" << endl;   return 1;
   }
/* If we reach this point, then the square is a magic square */

   cout << "Given matrix is a magic square" << endl;
   return 0;
```

Alternate program

```
/* All these summations can be done in a single nested iteration */
 int square[20][20], N, i, j, sum;
 int  rsum[20], csum[20], d1sum =0, d2sum =0;
 /* read N and all the matrix elements, initialize rsum[] and csum[] */
 sum = N * (N*N +1) /2;
 for (i=0; i< N; i++){
     for (j=0; j < N; j++){
         rsum[i] += square[i][j];
         csum[i] += square[j][i];  // Notice the “trick”
     }
     d1sum += square[i][i];
     d2sum += square[i][N-i-1];
 }
```

How are terms of a row added?

For a particular row i, the program iteratively calculates

rsum[i] = rsum[i] + square[i][j];

Thus, for i = 1; j = 0, 1, and 2

			initially	rsum[1]	= 0
	8	1	6	square[1][0]	= 3
[1][j]	3	5	7	square[1][1]	= 5
	4	9	2	square[1][2]	= 7
			Finally	rsum[1]	=15

How are terms of a column added?

For a particular column i, the program iteratively calculates

csum[i] = csum[i] + square[j][i];

Thus, for i = 1; j = 0, 1, and 2

		initially	csum[1]	= 0
8	1	6	square[1][0]	= 1
3	5	7	square[1][1]	= 5
4	9	2	square[1][2]	= 9
	[j][1]		Finally csum[1]	=15

How are diagonal terms added?

d1sum+= square[i][i] will add

8	1	6	square[0][0]	= 8
3	5	7	square[1][1]	= 5
4	9	2	square[2][2]	= 2

15

d2sum+= square[i][N-i-1] will add

8	1	6	square[0][2]	= 6
3	5	7	square[1][1]	= 5
4	9	2	square[2][1]	= 4

15

Digital images (from wikipedia)

- Digital images are a collection of pixel values
  - Pixel: A Picture element. Value represents light intensity
- These are arranged in an array (H x W)

Digital images

- Each pixel value can be represented by
  - 1 bit (m : monochrome, 0 or 1 value, e.g. black and white)
  - 8 bits (g : grayscale, 0 black to 255 white)
  - 24 bits (c: Red, Blue, Green, each one byte)
- One can have 16 million colours!
  - Capacity of a human eye is limited to a small range from 200 to 2000 colours

Digital images ...

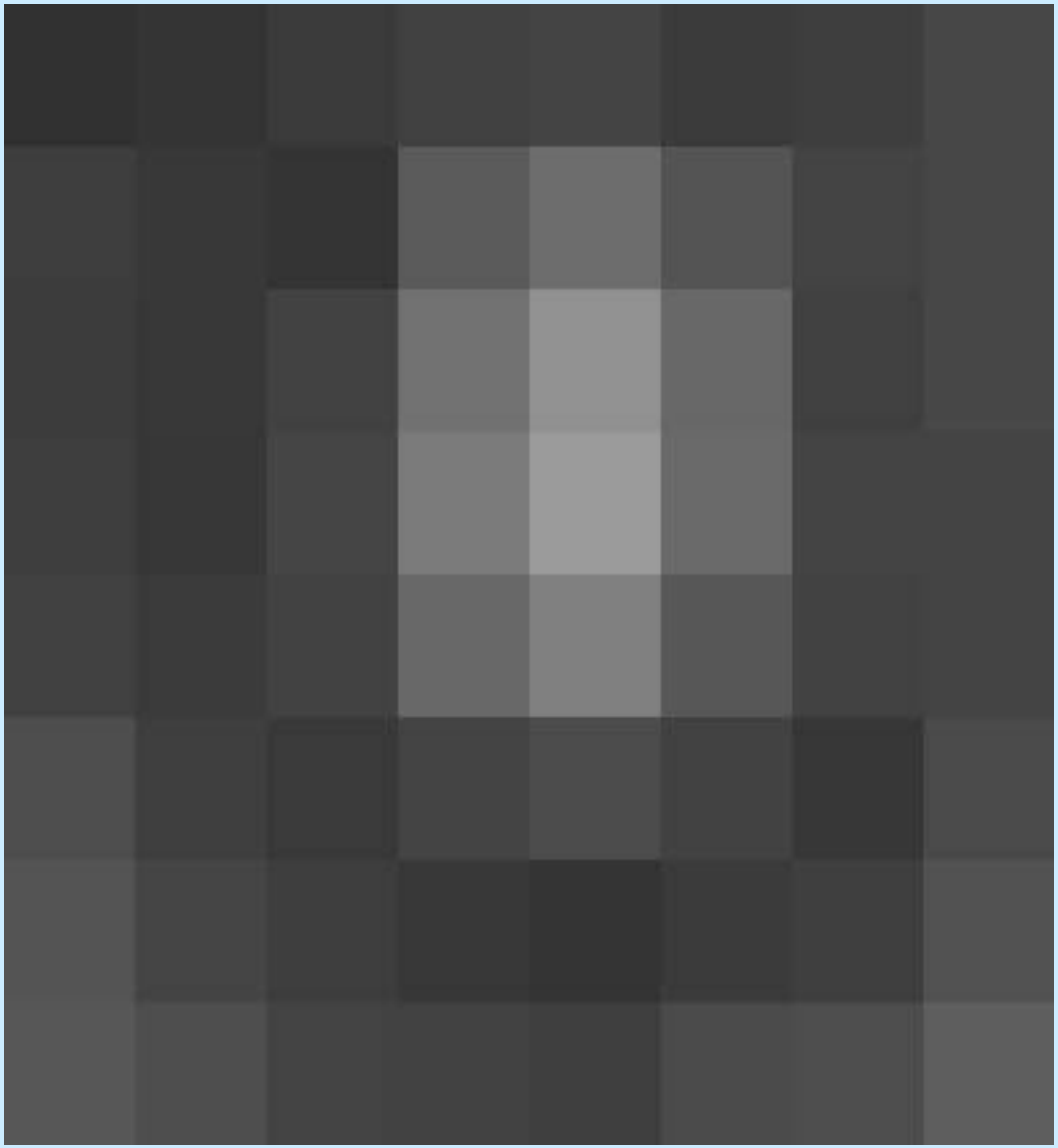
- To store information about an image in a file, we need values of Height, Width, the type of colors present, and values for each pixel for every color present
- Monochrome (grayscale) fingerprint images have small size
  - (500 x 300) bytes
- For large images, compression is necessary
  - to keep the file size within limits
  - 12 M pixel camera can produce 36 M bytes in an image
- Compression can be either lossy or lossless
- Several file formats have evolved

  raw, png, bmp, tiff, giff, jpeg, xmp
- Refer to wikipedia (Image_file_formats)

Images and histograms

- Pixel values of digital images can be read in a matrix
  - Matrix elements can be processed further
- Thus each element of a matrix for a grayscale image would contain a value between 0 and 255
  - type/size short int or char (1 byte)
- Histogram is a term from statistics, used to denote frequencies or count of number of times an event or incidence occurs
- In case of images, a histogram table indicates how many times a particular value occurs in the image
  - For each possible value, the number of pixels in the image having that value
- Why is histogram important?

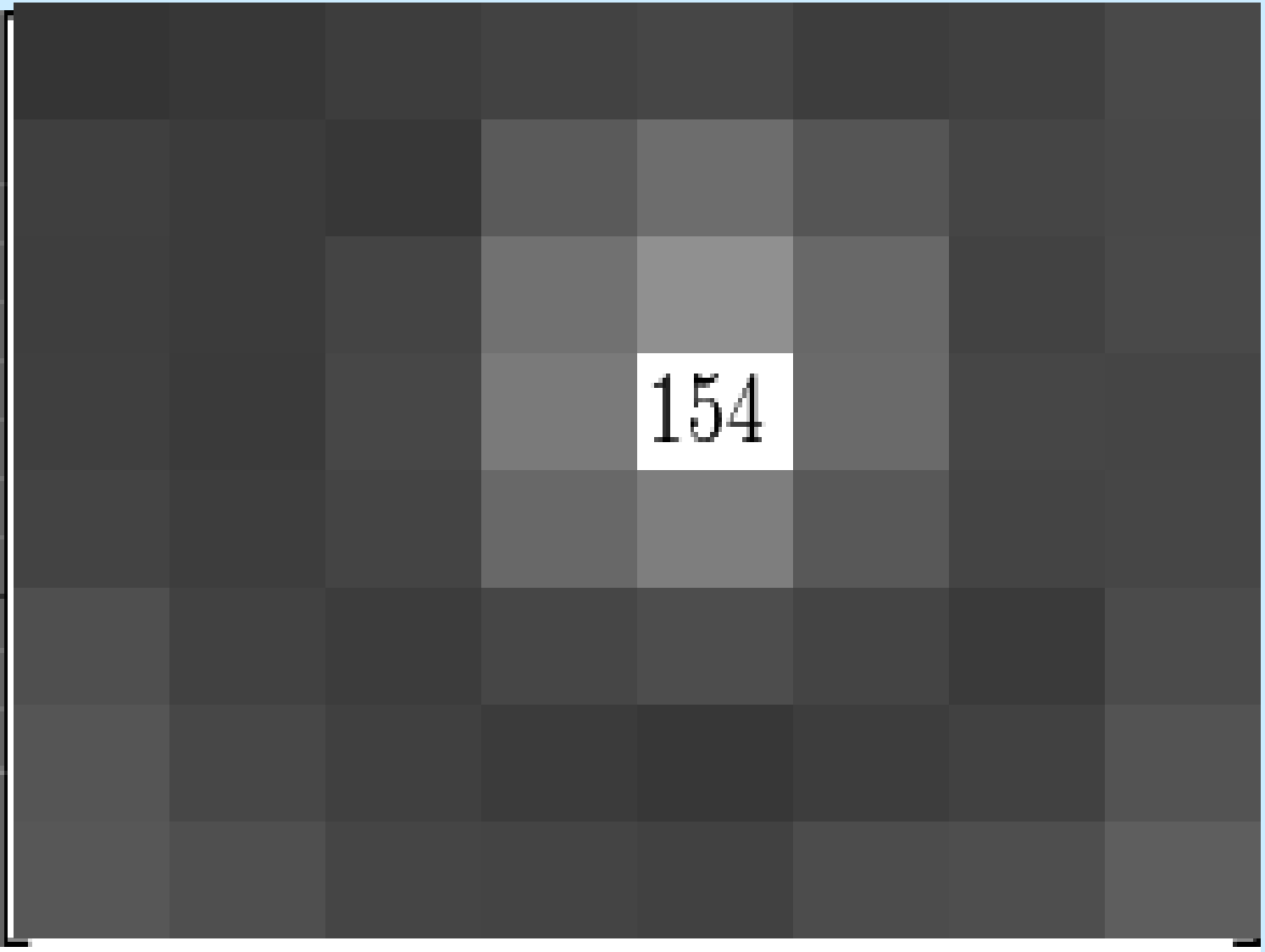
A sample image of size 8 pixel x 8 pixel



Pixel values in the sample image

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

Pixel values in the sample image



Histogram values (shown for non-zero pixels)

Val	n	Val	n	Val	n	Val	n	Val	n
52	1	64	2	72	1	85	2	113	1
55	3	65	3	73	2	87	1	122	1
58	2	66	2	75	1	88	1	126	1
59	3	67	1	76	1	90	1	144	1
60	1	68	5	77	1	94	1	154	1
61	4	69	3	78	1	104	2		
62	1	70	4	79	2	106	1		
63	2	71	2	83	1	109	1		

Values are between 52 and 154. Inadequate contrast!

What do we wish to do?

- The histogram is concentrated in a narrow range
  - 52 to 154
- Whereas the possible values are from 0 to 255
- If we can ‘stretch’ the histogram to cover all possible values
  - This is called histogram ‘equalization’
  - We should get a better contrast
- We need to ‘map’ existing values to new values
  - A value ‘v’ should be mapped by a suitable function to h (v)
- How?
  - We use the cumulative distribution function (cdf) of a histogram

Cumulative Distribution Function (cdf)

V	c	V	c	V	c	V	c	V	c
52	1	64	19	72	40	85	51	113	60
55	4	65	22	73	42	87	52	122	61
58	6	66	24	75	43	88	53	126	62
59	9	67	25	76	44	90	54	144	63
60	10	68	30	77	45	94	55	154	64
61	14	69	33	78	46	104	57		
62	15	70	37	79	48	106	58		
63	17	71	39	83	49	109	59		

Histogram equalization

- The equalization formula to calculate new value for any existing pixel value v

$$h(v) = \text{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$

- "Equalization" formula for example image
  - L = 256, M = N = 8, minimum cdf is 1

$$h(v) = \text{round}\left(\frac{cdf(v) - 1}{63} \times 255\right)$$

Histogram equalization …

- For example, the cdf of 78 is 46
  - So a pixel value 78 will be ‘equalized’ using the formula:

H(78) = round ((46 -1)/63) * 255) =

= round (0.714286 * 255)

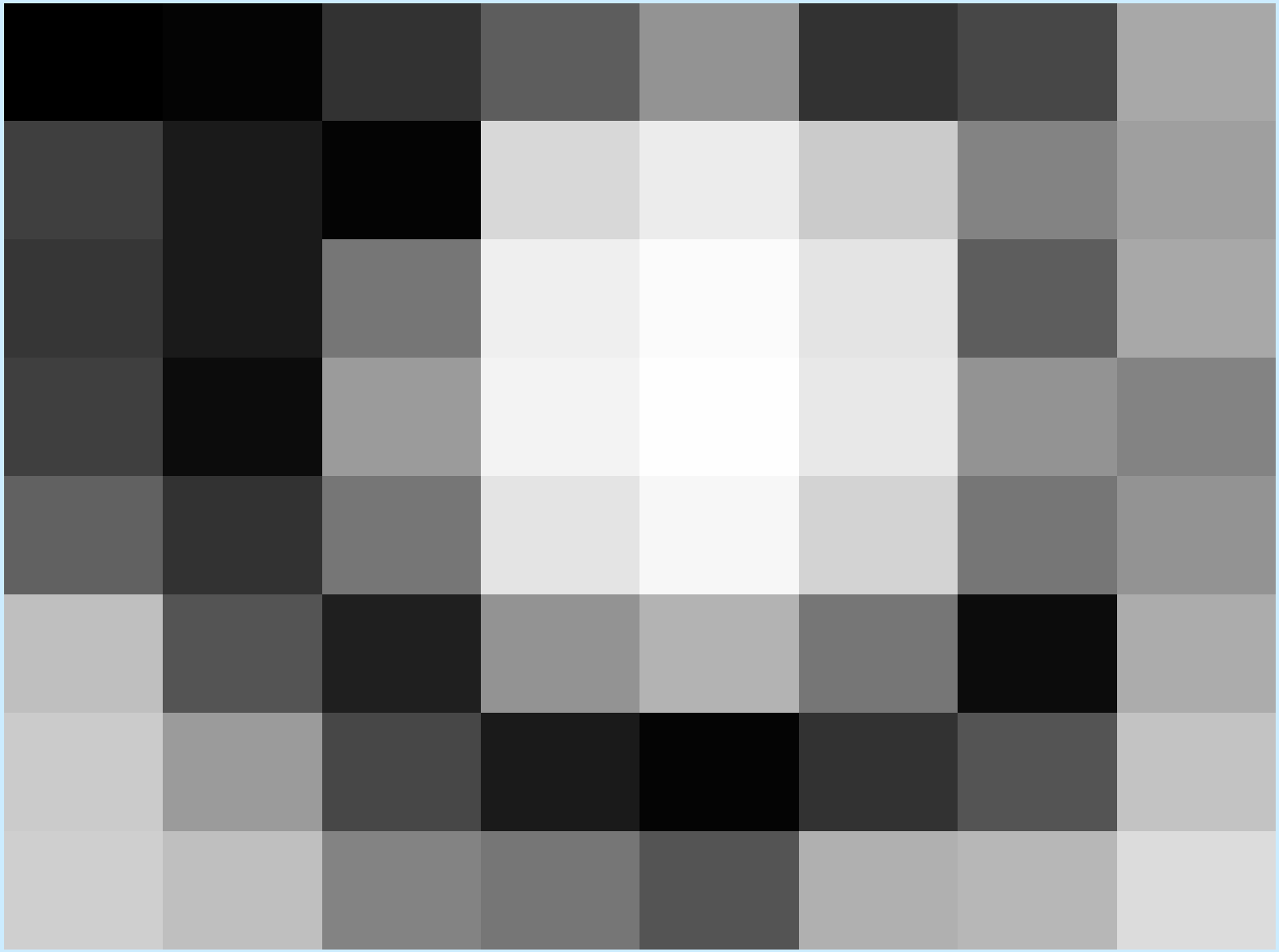
= 182

  - This will give us a new ‘equalized’ value
  - Each pixel in the image, which has a value 78, will now have this new value
- We calculate such new value for each pixel

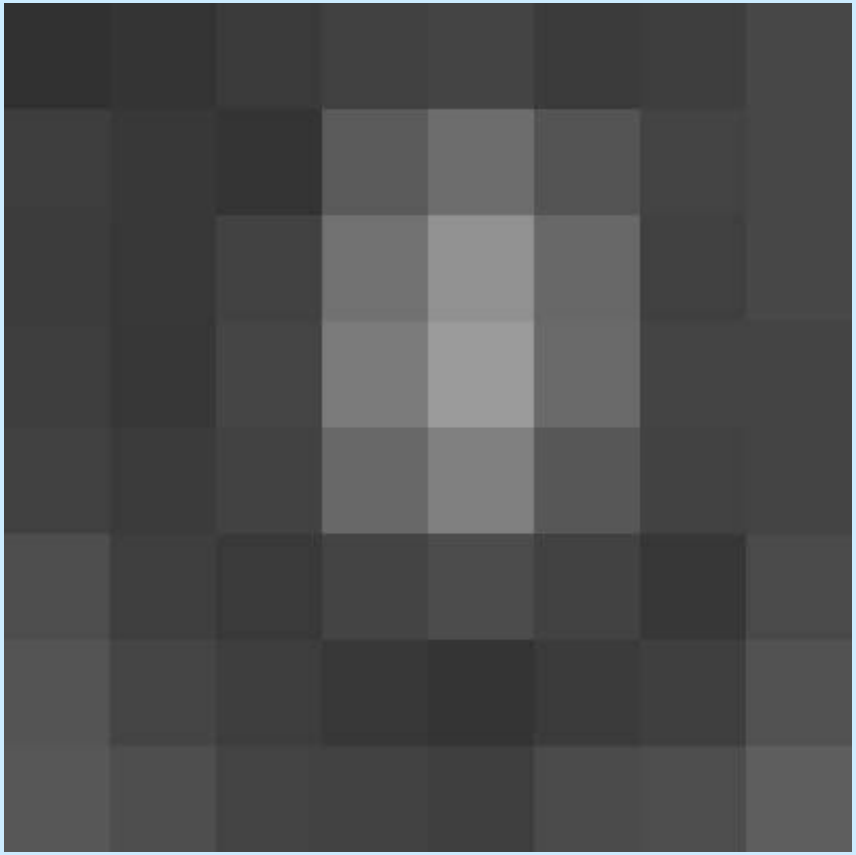
Pixel values after histogram equalization

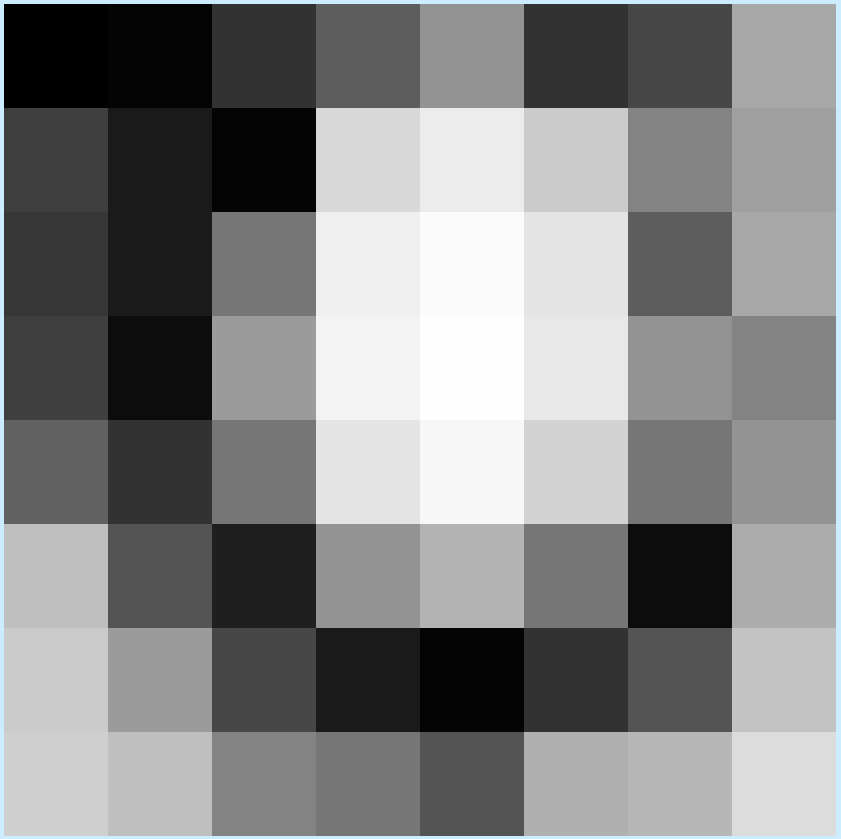
$$\begin{bmatrix} 0 & 12 & 53 & 93 & 146 & 53 & 73 & 166 \\ 65 & 32 & 12 & 215 & 235 & 202 & 130 & 158 \\ 57 & 32 & 117 & 239 & 251 & 227 & 93 & 166 \\ 65 & 20 & 154 & 243 & 255 & 231 & 146 & 130 \\ 97 & 53 & 117 & 227 & 247 & 210 & 117 & 146 \\ 190 & 85 & 36 & 146 & 178 & 117 & 20 & 170 \\ 202 & 154 & 73 & 32 & 12 & 53 & 85 & 194 \\ 206 & 190 & 130 & 117 & 85 & 174 & 182 & 219 \end{bmatrix}$$

Enhancement of Contrast



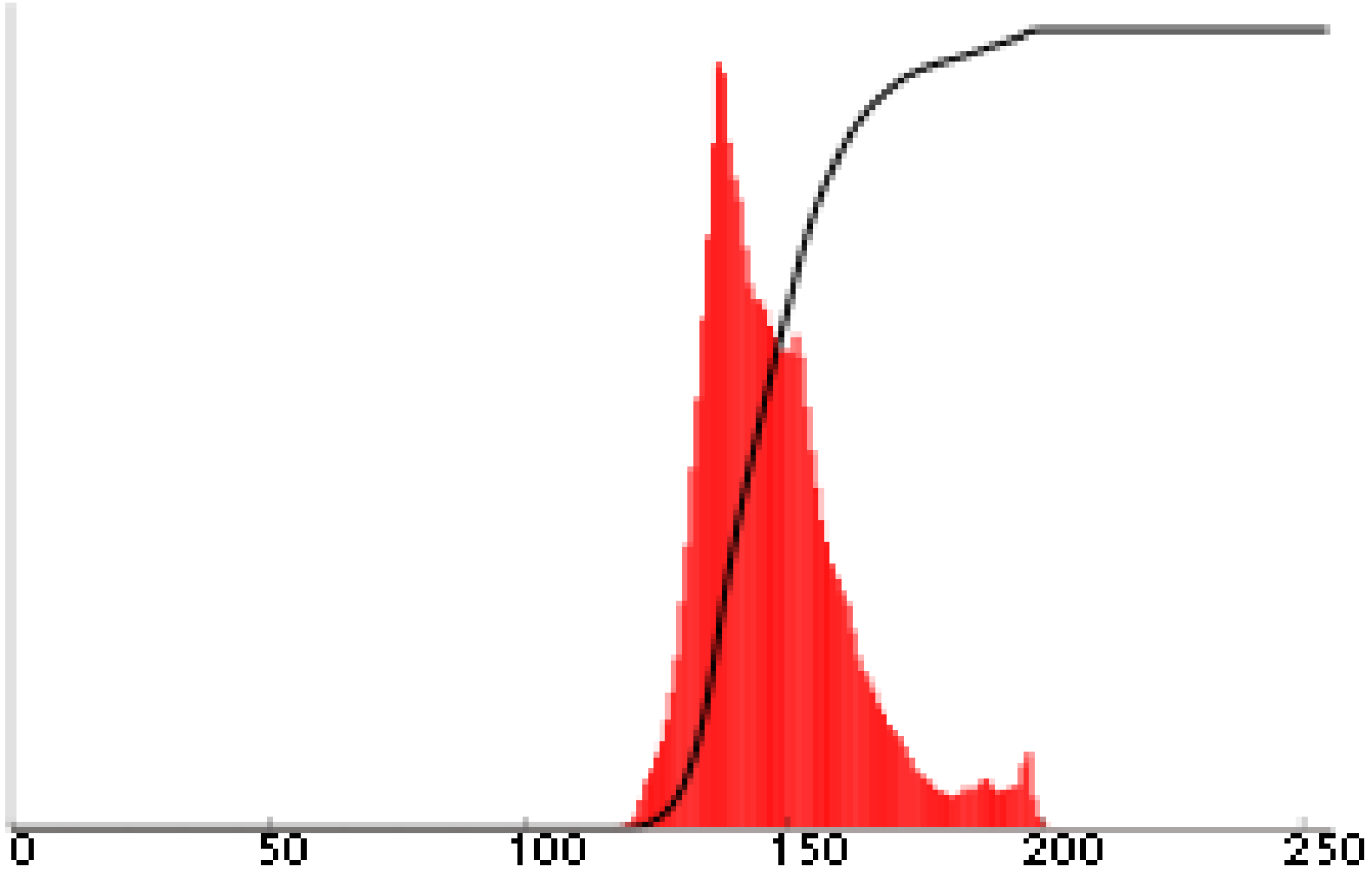
Original and new pictures for comparison



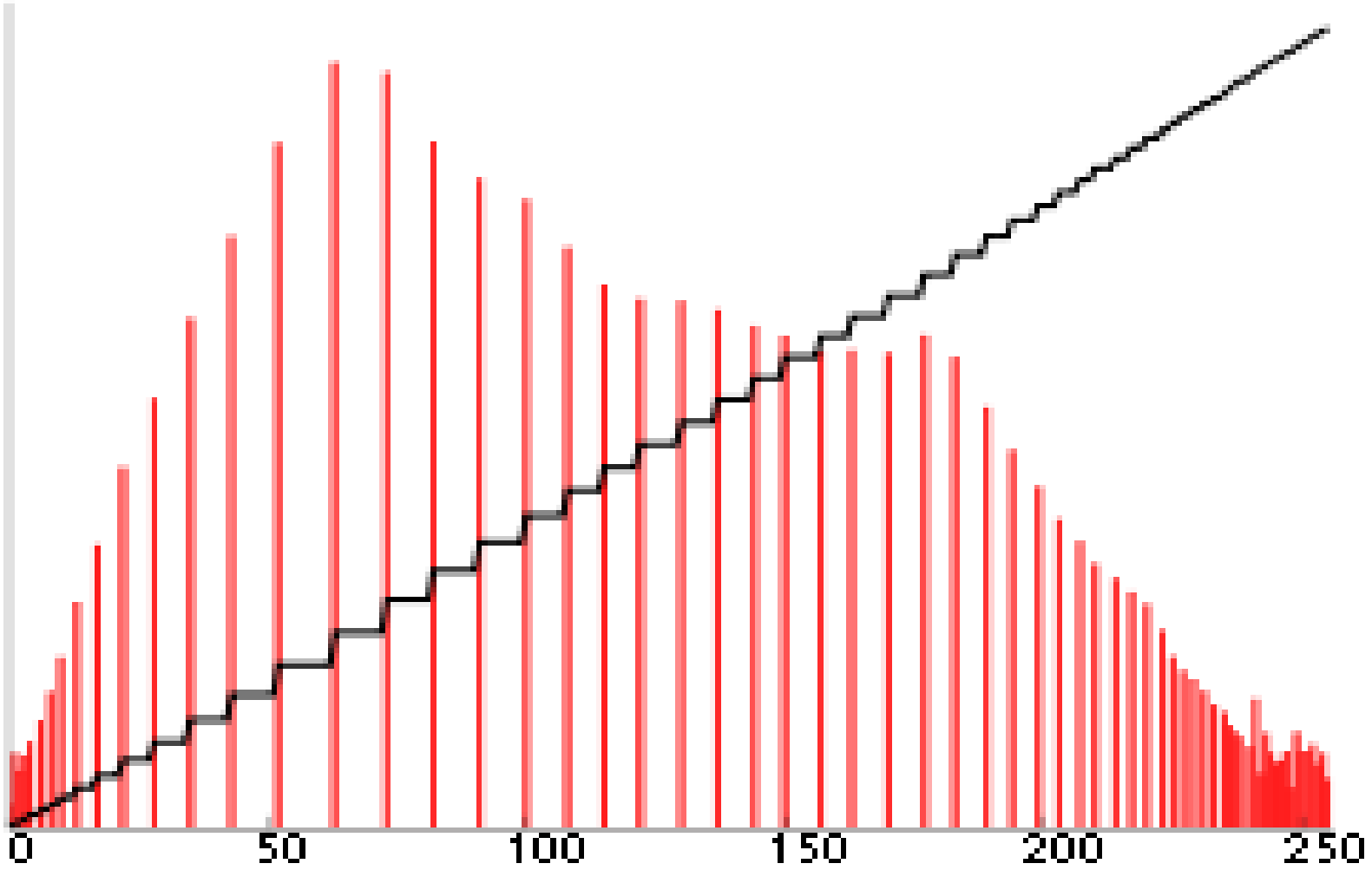


Another grayscale picture

Histogram and cdf

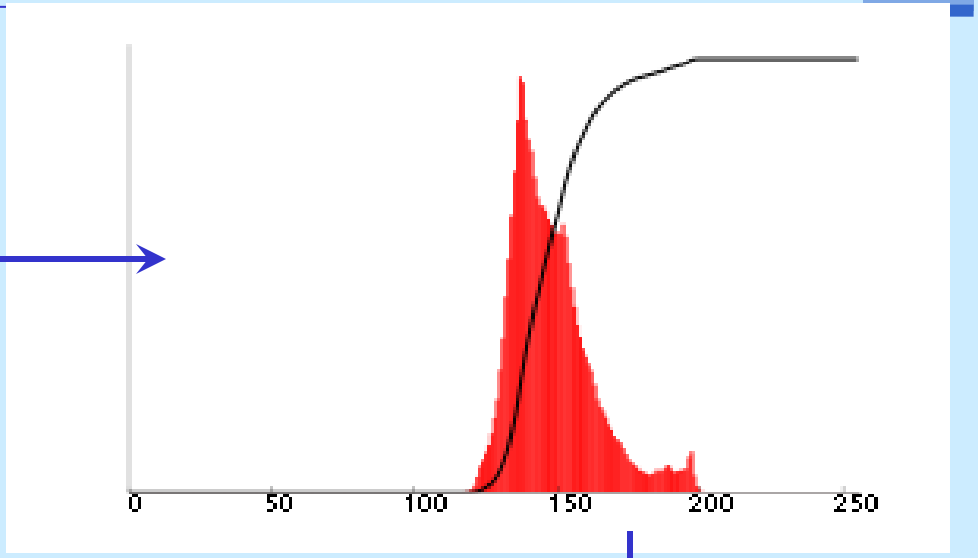


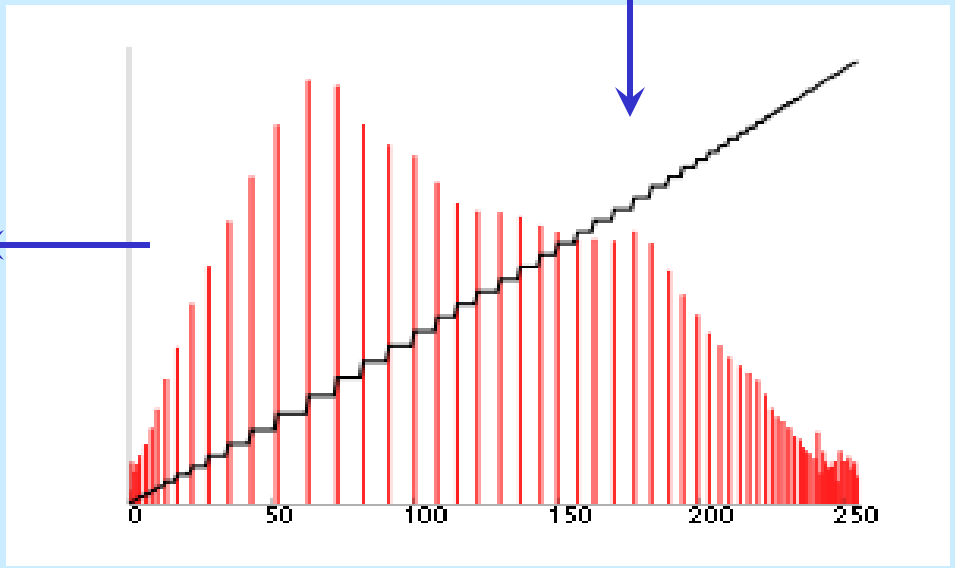
“Equalized” histogram and cdf



Picture with enhanced contrast

Original picture, equalization, modified picture





Calculating entries in the histogram table

Index	Value
0	0
1	0
.	.
.	
68 →	5
69	3
.	.
.	.
255	0

Pixel values in the image

52	55	61	66	70	61	64	73
63	59	55	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	68	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

Program for histogram equalization

IIT BOMBAY

```
/*
  program 2011_08_30_histogram_equalization.cpp

  A program which reads the pixel intensities for a grayscale image,
  calculates the histogram and the cumulative distribution function,
  and finally recalculates the pixel values such that the histogram is
  equalized. This procedure gives us an image with better contrast.

  Procedure and formula based on the material in wikipedia
*/
```

Program ...

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
  int i,j,value=0,min=0,max=0,foundflag=0,equalization,M,N;
  int image[500][500], newimage[500][500];
  int histogram[256], cdf[256], equalizer[256];
  cout << " Give the image size: Height M and width N ";
  cin >> M >> N;
  for(i=0;i<M;i++)
     for(j=0;j<N;j++)
        cin>>image[i][j];
```

Program ...

```
 for(i=0;i<256;i++){    //initialize all table elements to 0
    histogram[i]=0;
    cdf[i]=0;
    equalizer[i]=0;
 }
/* calculate histogram table entries  */
for (i=0; i<M; i++) {
    for (j=0;j<N;j++){
          histogram[ image[i][j] ]++;   // principle of associative array
    }
}
```

Program ...

```
  cout<<"Histogram values"<<endl;
  for(i=0;i<256;i++)  cout<<i<<“ "<<histogram[i]<<"\t";
  cout<<endl
  /* calculate cdf table entries */
  cdf[0] = histogram[0];
  for(i=1;i<256;i++){
       cdf[i]= cdf[i-1] + histogram[i];
  }
  /* Find the minimum value in cdf table*/
  min = 2000  ;  // any artificial value > 255
  for (i=0; i < 256; i ++){
        if (cdf[i] < min) min =cdf[i];
}
```

```
/*  Calculate entries in the equalizer table */
  cout<<endl;
  cout<<"Equalization Values"<<endl;
  for (i=0; i<256;i++){
      equalizer[i]=round((float)(cdf[i]-min)/(M*N-min)*(256-1));
      cout<<i<<" "<<equalizer[i]<<"\t ";
  }
  cout<<endl;
  ------
```

Calculating modified pixel values

```
 /*   Calculate entries in the newimage array */
for(i=0;i<M;i++){
    for(j=0;j<N;j++){
        newimage[i][j] = equalizer[ image[i][j] ];
    }
}
```

Image of a finger print