ECE5554 – Computer Vision
Lecture 2a – Pixel Operations

Creed Jones, PhD

BRADLEY DEPARTMENT
OF ELECTRICAL & COMPUTER ENGINEERING

VIRGINIA TECH

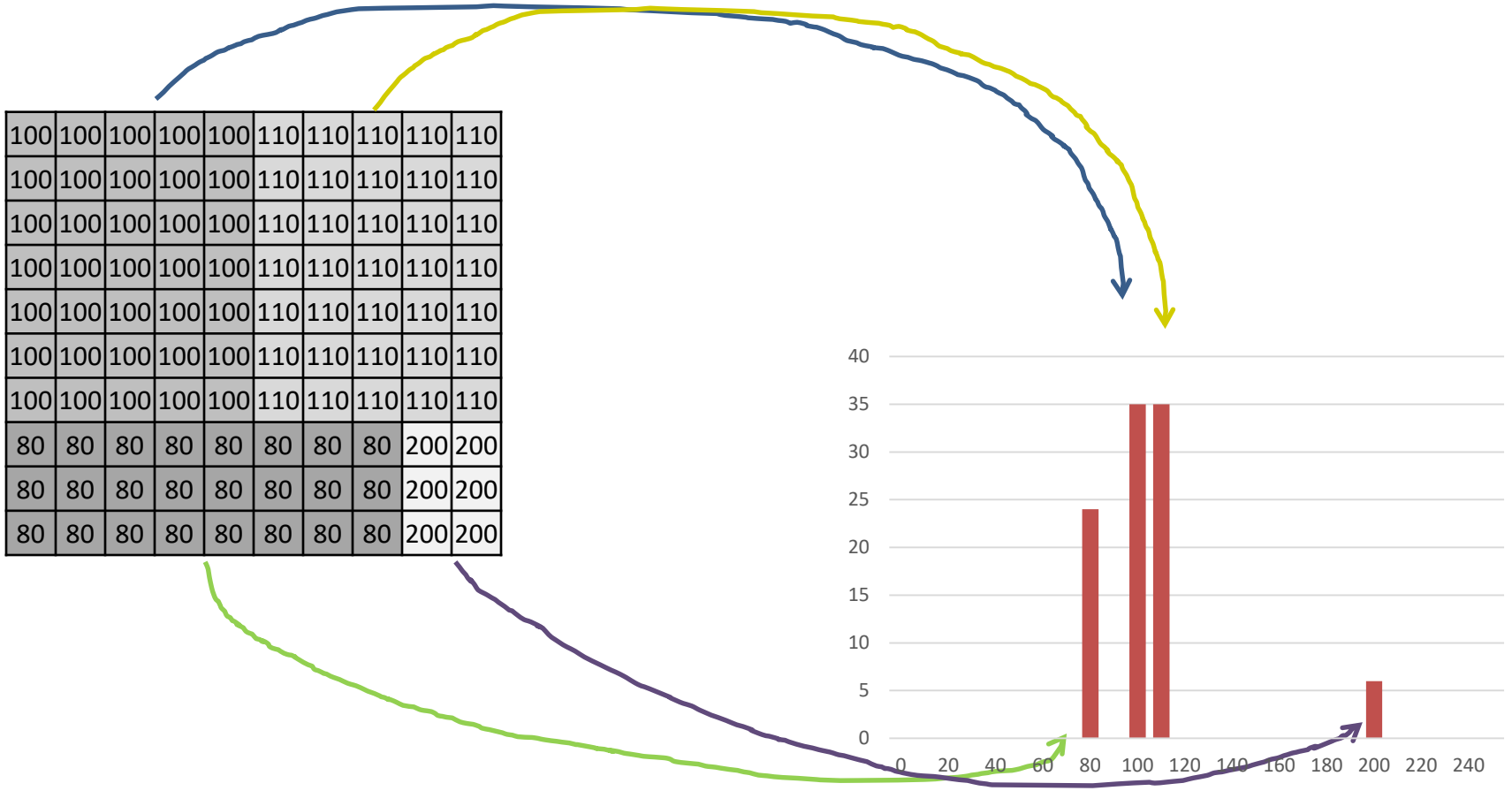
Course updates

- HW1
  - due this Thursday, July 13, 11:59 PM
- Quiz 1 is TOMORROW, July 12 (open 6 PM to 11:59 PM)
  - Covers lectures 1 (1a, 1b & 1c) and 2 (2a, 2b & 2c)
  - Don't forget!
- GTA office hours
  - Dhanush Dinesh will be available via zoom on Monday 6PM - 8PM and on Thursday at 10 AM - 12 PM
  - Meeting ID: 895 5961 7255
  - Passcode: 824650

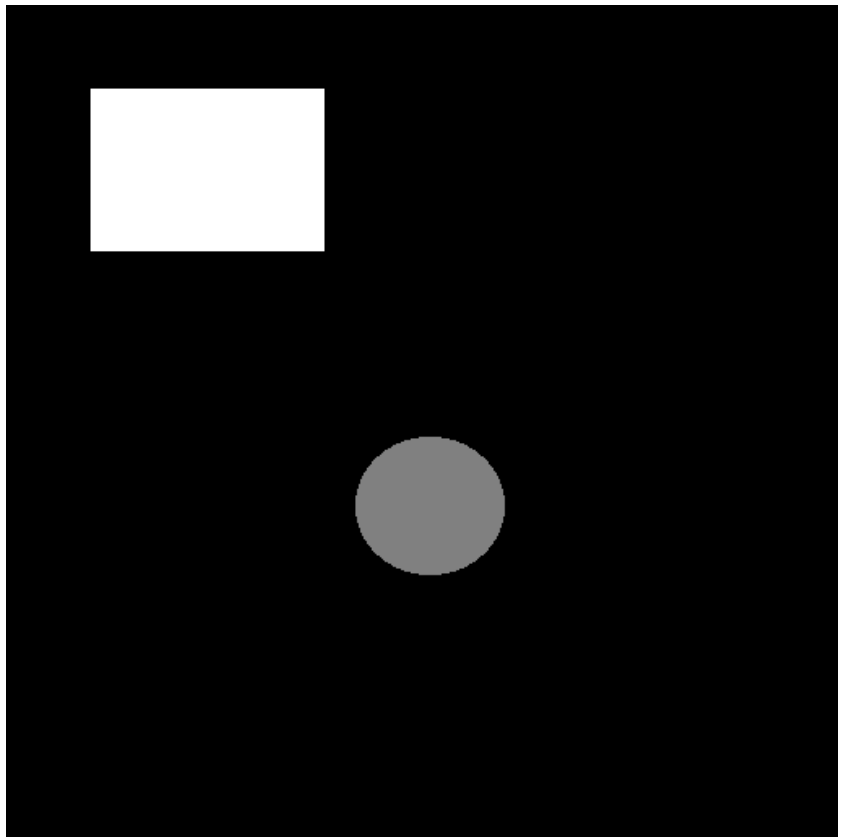
Today's Objectives

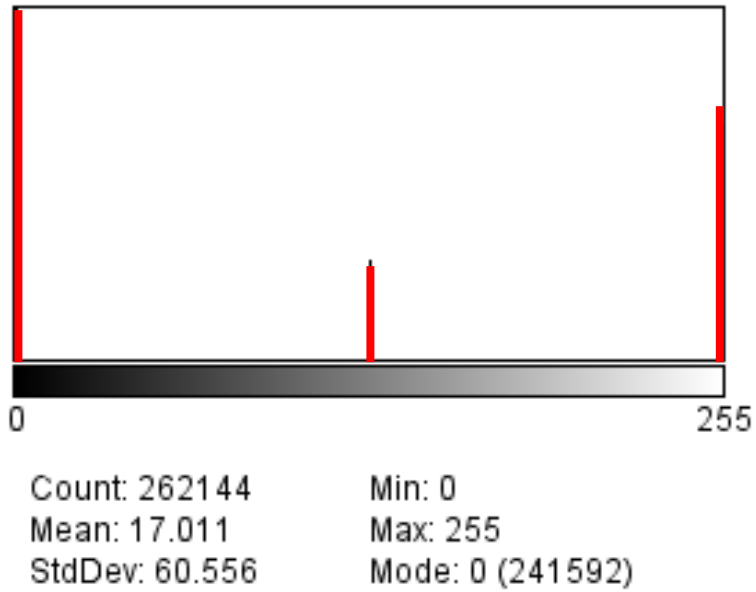
- Intensity histograms
- Image Statistics
- Histogram Equalization
- Pixel Operations

An image histogram is a vector where each entry is the frequency of occurrence of the corresponding gray-level value; it's a bar plot of brightness counts

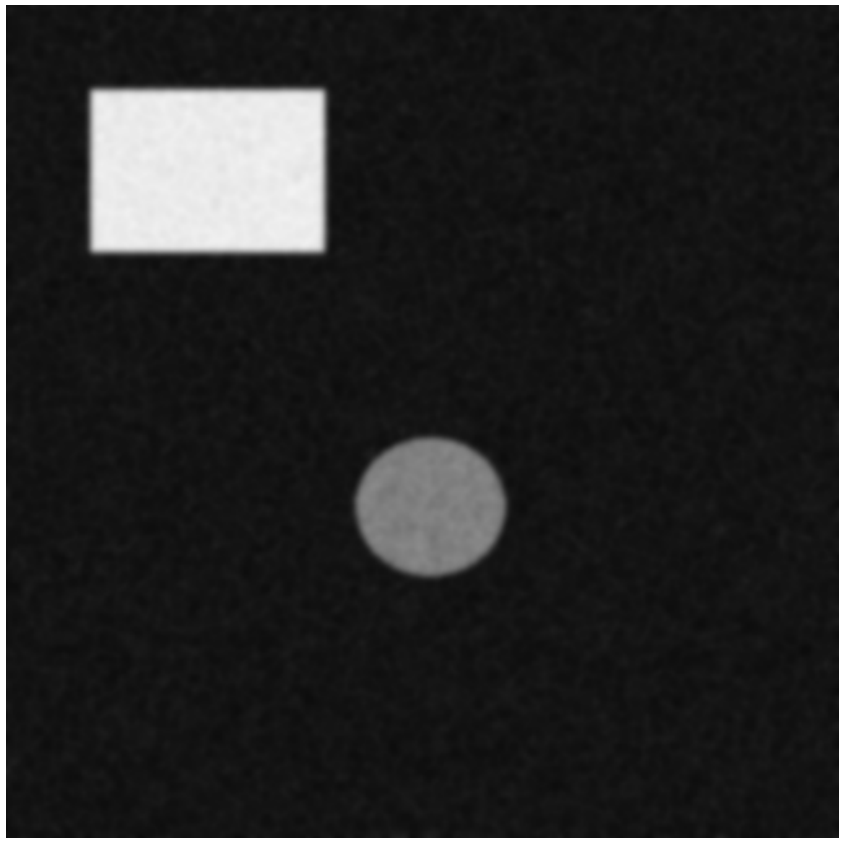


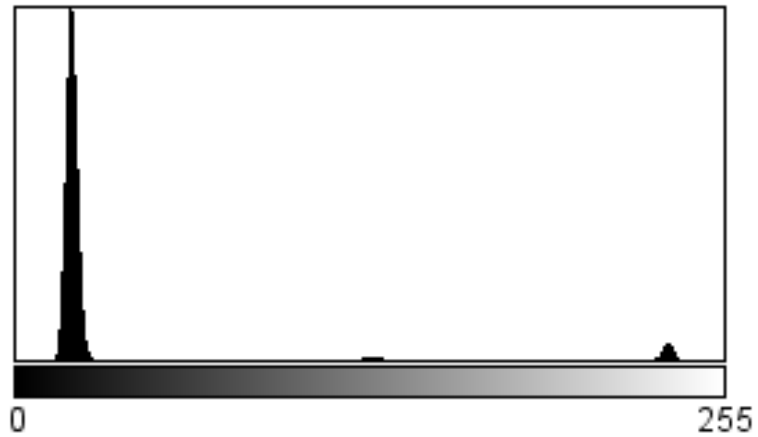
Regions of distinct intensity are easily seen in the image histogram – note that the area of each region is apparent (the height of the histogram is pixel count)





Noise and image smoothing (blur) cause the lobes in the histogram to spread; the noise that was added to this image is normally distributed





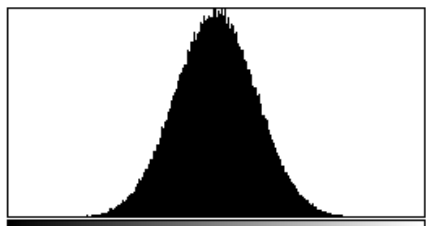
Count: 262144
Mean: 34.057
StdDev: 50.218

Min: 10
Max: 243
Mode: 19 (42839)

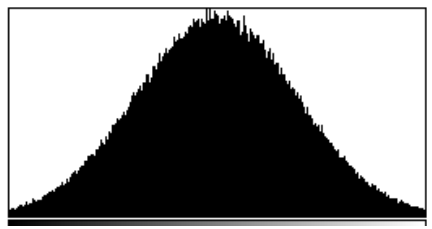
Varying levels of additive Gaussian noise (higher standard deviations, wider distributions) cause more image degradation

original image

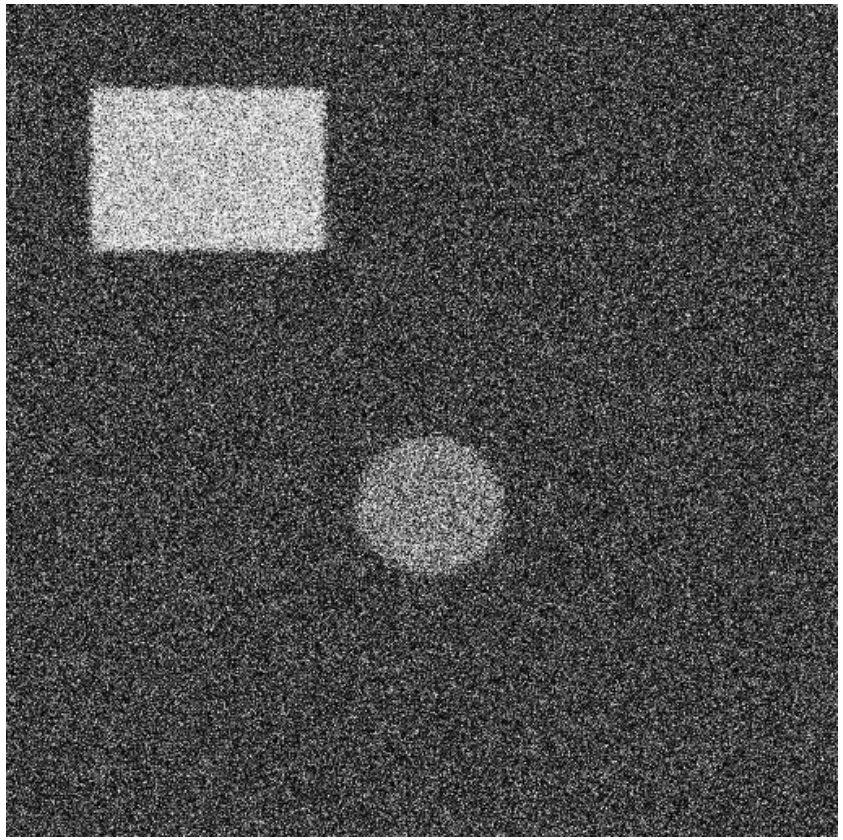
Gaussian noise –
σ = 25.0

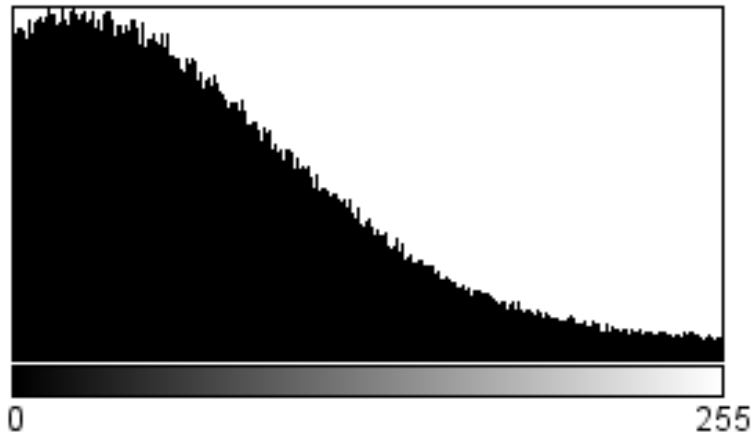


Gaussian noise –
σ = 50.0



A large amount of noise causes the lobes corresponding to regions to spread farther – the Gaussian shape of the noise distribution is easily seen





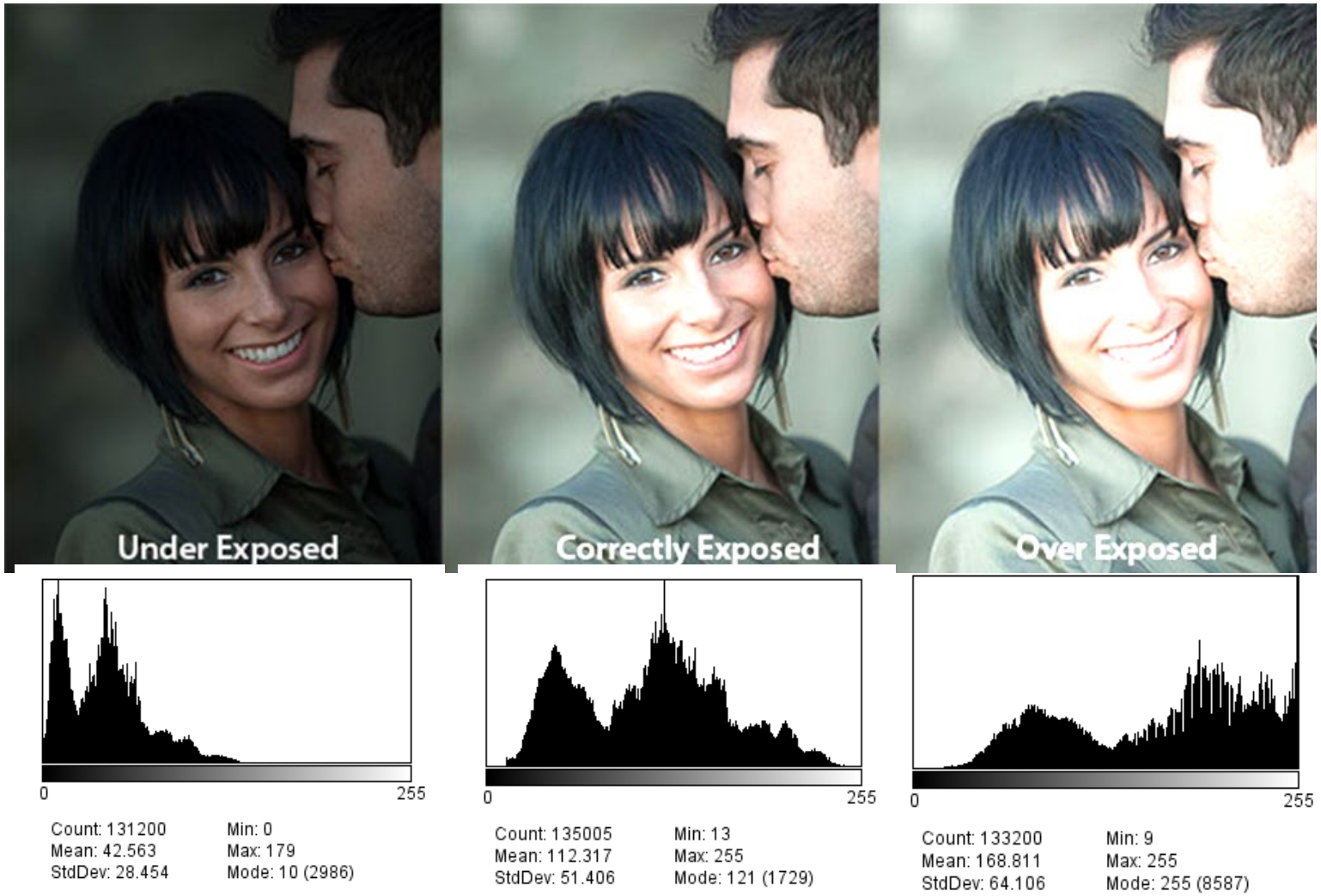
Count: 262144
Mean: 74.958
StdDev: 56.285

Min: 0
Max: 255
Mode: 12 (2194)

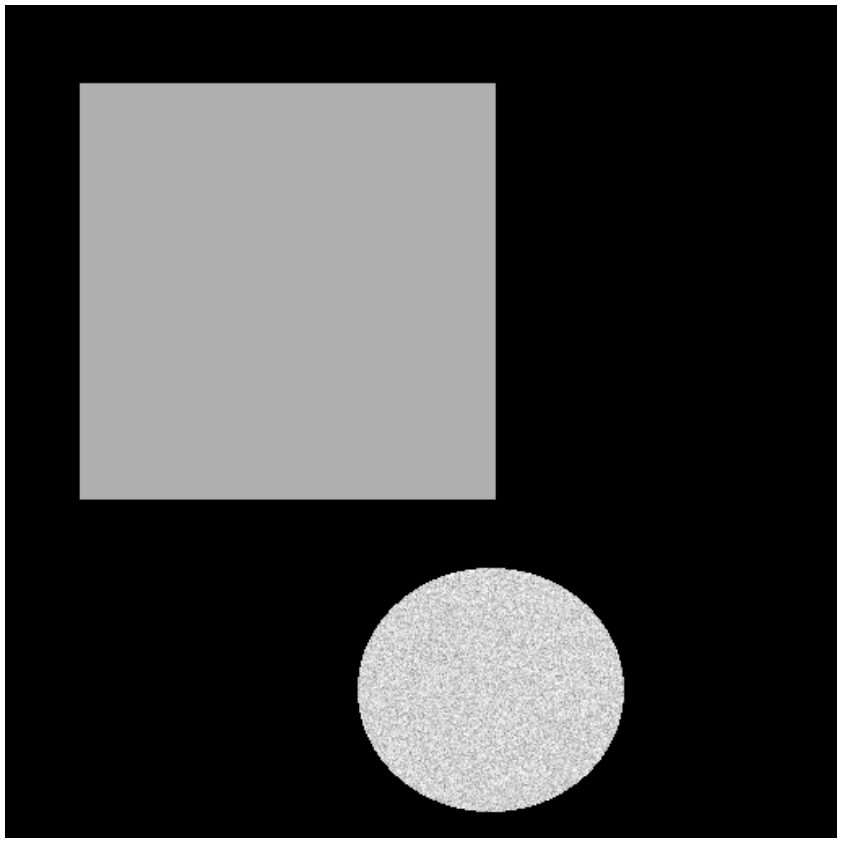
Histograms can be computed by adding the total number of pixels having each distinct gray level – location of the pixels is ignored

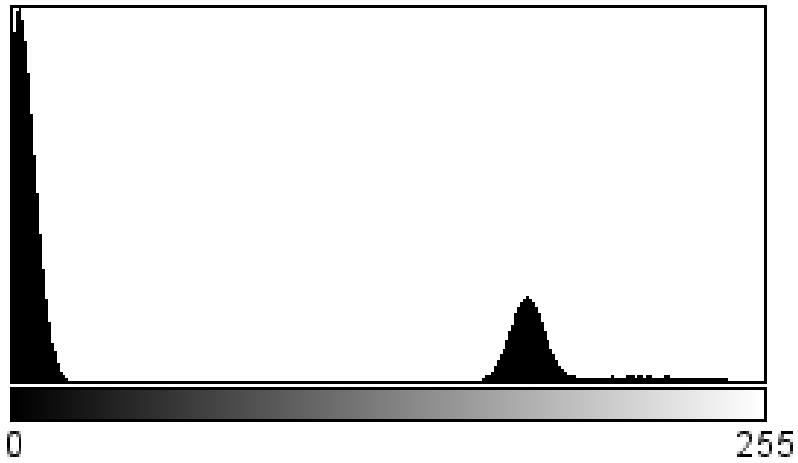
```
def getHist(image, histlen):
    row,col = image.shape
    H = np.zeros((histlen), int)
    for r in range(row):
        for c in range(col):
            H[image[r,c]] = H[image[r,c]] + 1
    return H


…
img = cv2.imread('C:\\Data\\hieroglyphics.png', cv2.IMREAD_GRAYSCALE)
histo = getHist(img, 256)
```

The histogram quickly reveals images that are underexposed, properly exposed or overexposed



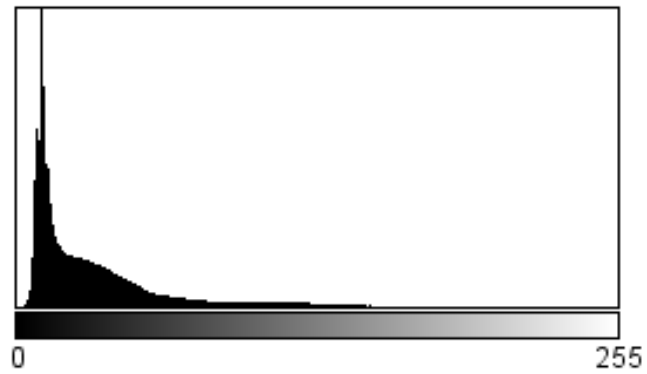
When objects in the image are clearly defined by different gray levels, the histogram reveals the areas of the objects in pixel count





Count: 262144
Mean: 62.449
StdDev: 84.213

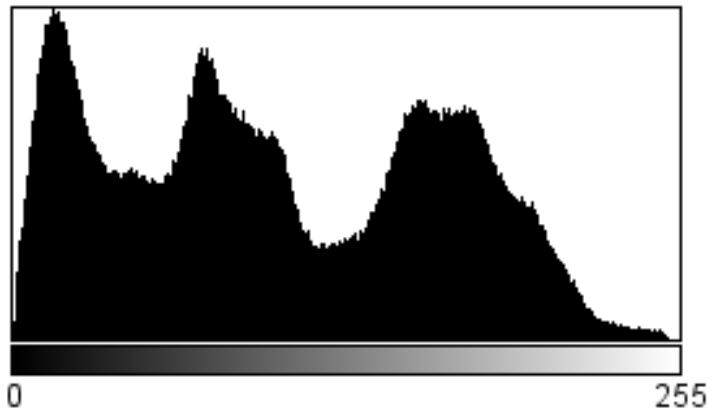
Min: 0
Max: 255
Mode: 2 (20075)



Count: 1225000
Mean: 36.922
StdDev: 39.459

Min: 0
Max: 250
Mode: 10 (94324)

Rembrandt van Rijn, *Jacob blessing the Children of Joseph*



Count: 838614
Mean: 101.429
StdDev: 63.027

Min: 0
Max: 254
Mode: 15 (6891)

Jan Vermeer, *Young Woman with a Water Pitcher*

The gray-level mean and variance can be computed from the image histogram – and we can find the median as well

Mean:

$$\mu = \frac{1}{RC}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I(r,c)$$

Variance:

$$\sigma^2 = \frac{1}{RC}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1}[I(r,c)-\mu]^2$$

or

$$\sigma^2 = \left[\frac{1}{RC}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1}[I(r,c)]^2\right] - \mu^2$$

Standard Deviation:

$$\sigma = \sqrt{\sigma^2}$$

Mean:

$$\mu = \frac{1}{RC}\sum_{i=0}^{G-1} i \cdot H(i)$$

Variance:

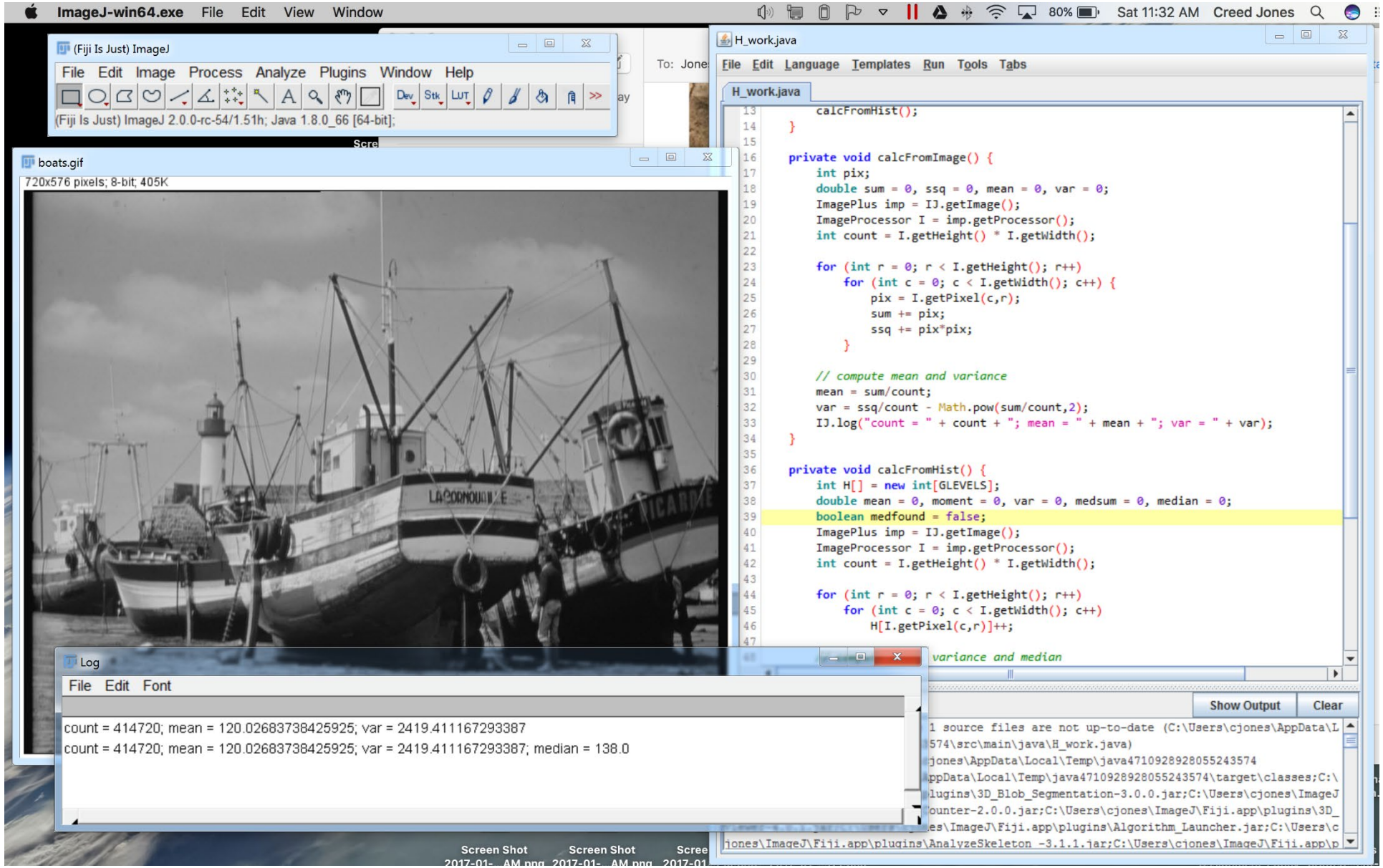
$$\sigma^2 = \left[\frac{1}{RC}\sum_{i=0}^{G-1} i^2 \cdot H(i)\right] - \mu^2$$

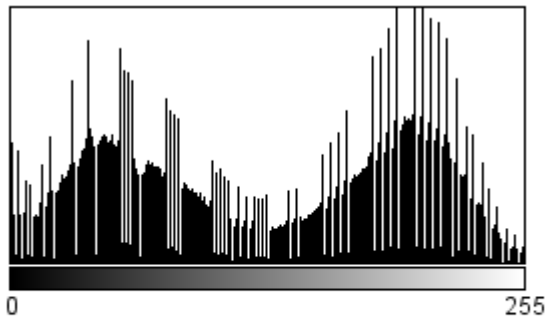
Standard Deviation:

$$\sigma = \sqrt{\sigma^2}$$

Median:

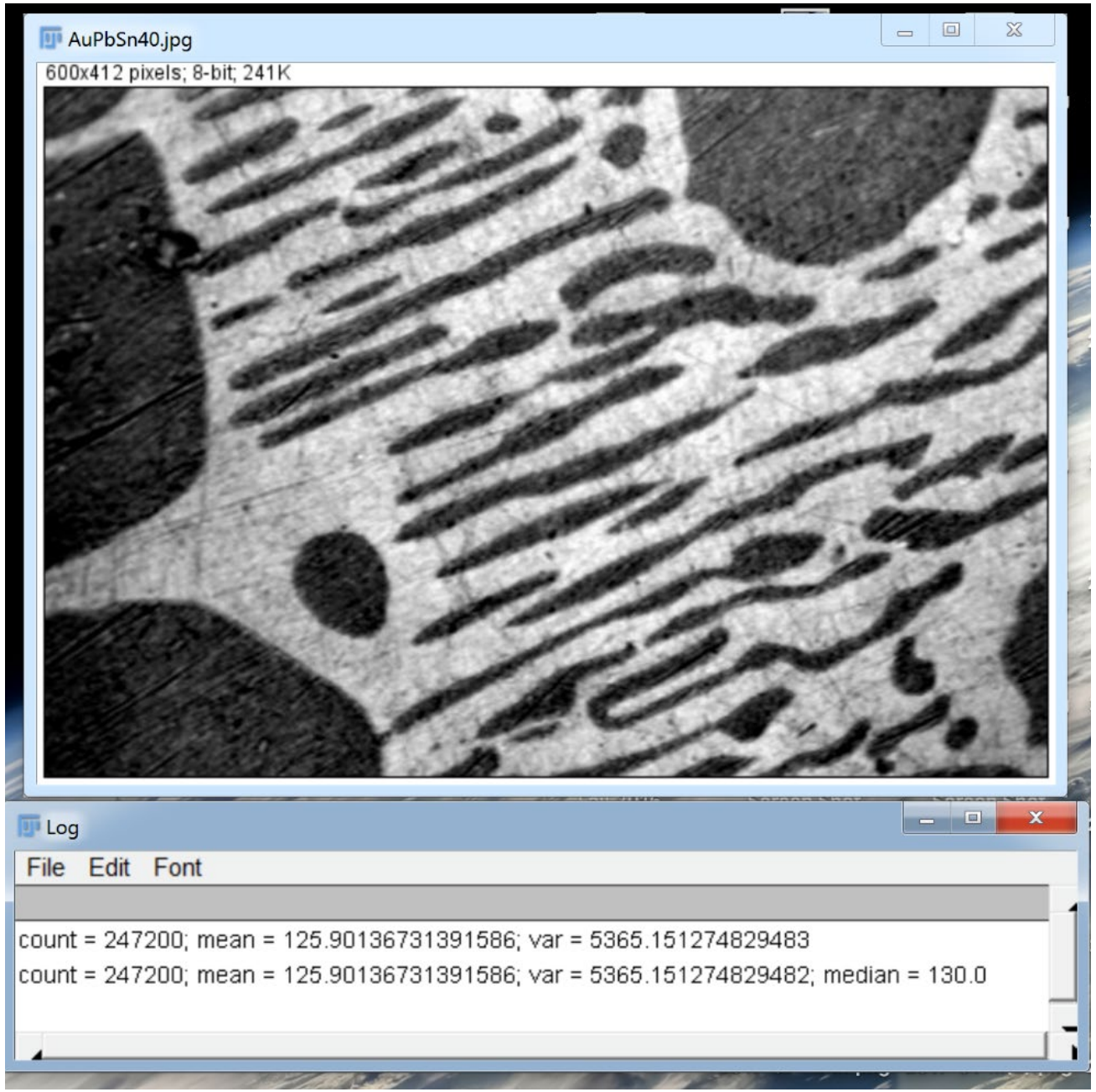
$$m = min\left\{ i \left| \sum_{k=0}^{i} H(k) \geq \frac{RC}{2} \right.\right\}$$

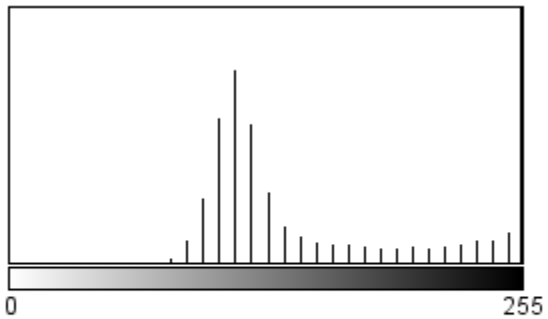

ImageJ-win64.exe File Edit View Window
80% Sat 11:32 AM Creed Jones
(Fiji Is Just) ImageJ
File Edit Image Process Analyze Plugins Window Help
(Fiji Is Just) ImageJ 2.0.0-rc-54/1.51h; Java 1.8.0_66 [64-bit];
boats.gif
720x576 pixels; 8-bit; 405K
H_work.java
File Edit Language Templates Run Tools Tabs
H_work.java
calcFromHist();
}
private void calcFromImage() {
int pix;
double sum = 0, ssq = 0, mean = 0, var = 0;
ImagePlus imp = IJ.getImage();
ImageProcessor I = imp.getProcessor();
int count = I.getHeight() * I.getWidth();
for (int r = 0; r < I.getHeight(); r++)
for (int c = 0; c < I.getWidth(); c++) {
pix = I.getPixel(c,r);
sum += pix;
ssq += pix*pix;
}
// compute mean and variance
mean = sum/count;
var = ssq/count - Math.pow(sum/count,2);
IJ.log("count = " + count + "; mean = " + mean + "; var = " + var);
}
private void calcFromHist() {
int H[] = new int[GLEVELS];
double mean = 0, moment = 0, var = 0, medsum = 0, median = 0;
boolean medfound = false;
ImagePlus imp = IJ.getImage();
ImageProcessor I = imp.getProcessor();
int count = I.getHeight() * I.getWidth();
for (int r = 0; r < I.getHeight(); r++)
for (int c = 0; c < I.getWidth(); c++)
H[I.getPixel(c,r)]++;
variance and median
Show Output
Clear
Log
File Edit Font
count = 414720; mean = 120.02683738425925; var = 2419.411167293387
count = 414720; mean = 120.02683738425925; var = 2419.411167293387; median = 138.0



Count: 247200
Mean: 125.901
StdDev: 73.247

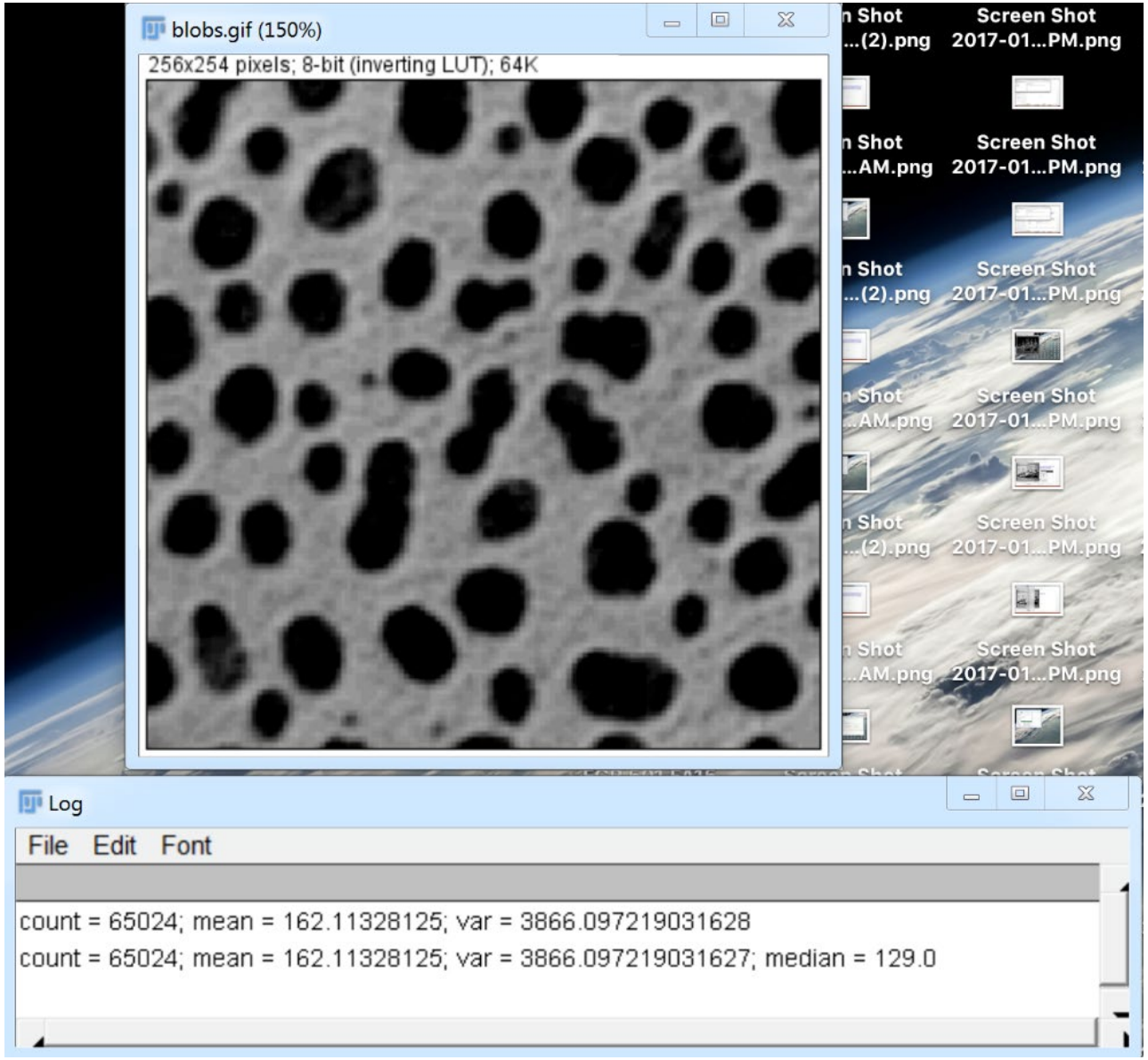
Min: 0
Max: 255
Mode: 201 (3164)





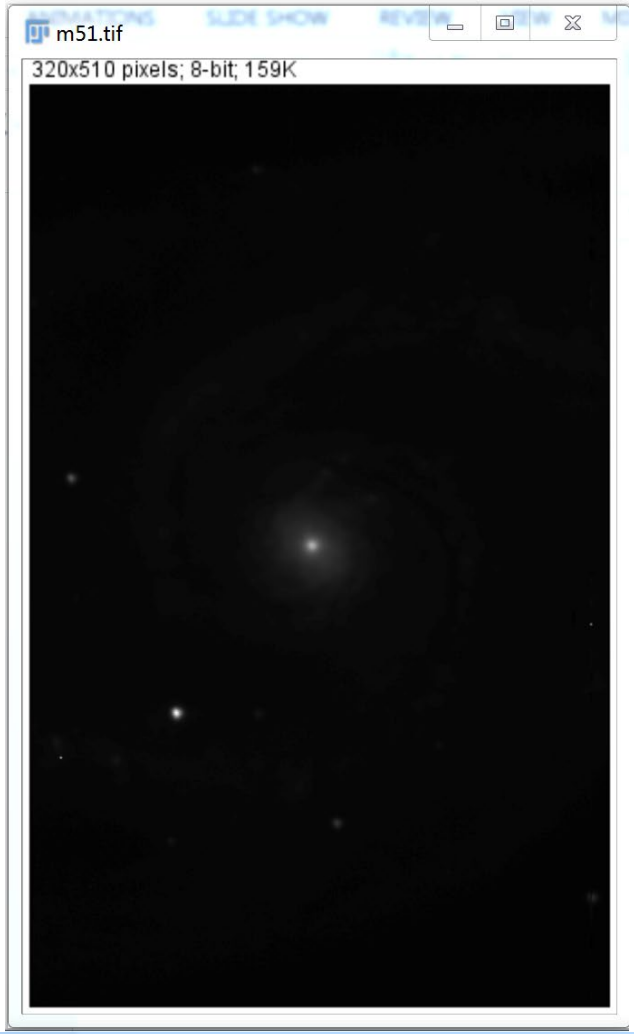
Count: 65024
Mean: 162.113
StdDev: 62.178

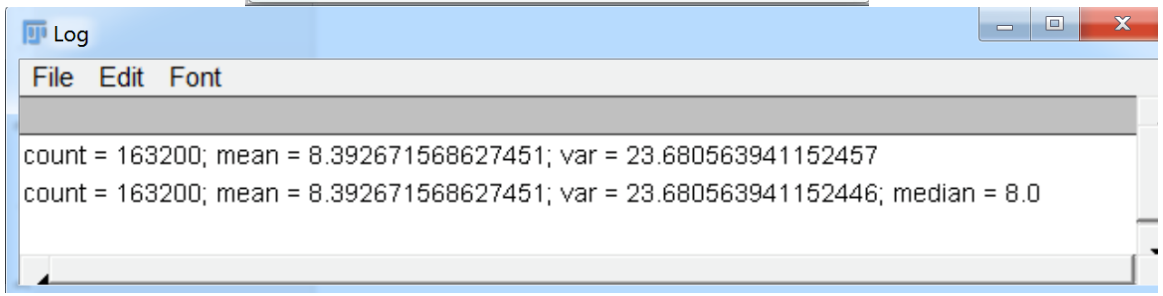
Min: 72
Max: 255
Mode: 255 (13723)



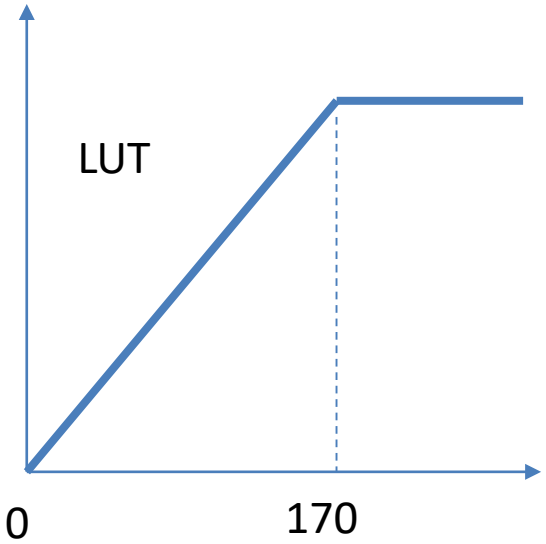
Count: 163200
Mean: 8.393
StdDev: 4.866

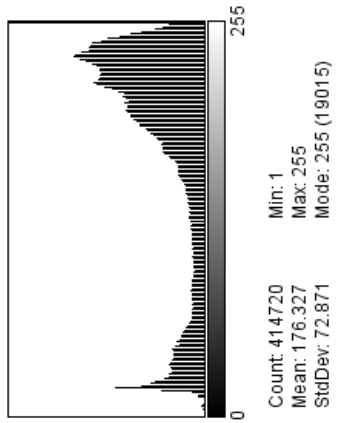
Min: 0
Max: 255
Mode: 7 (44085)

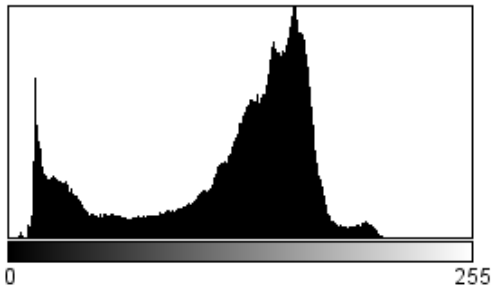




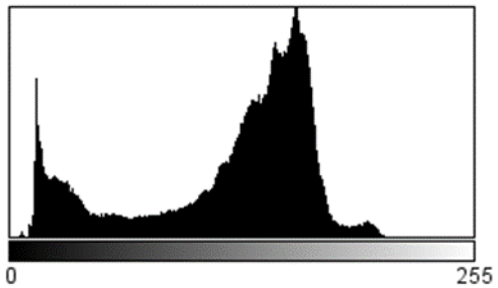
Realize that the output histogram can be computed by transforming the input histogram through the look-up table…





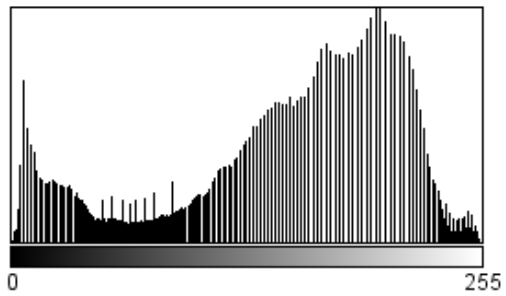


…so we can design a lookup table by specifying what the output histogram should look like; one approach is to make the histogram as flat as possible – this is called histogram equalization



Count: 414720
Mean: 120.027
StdDev: 49.188

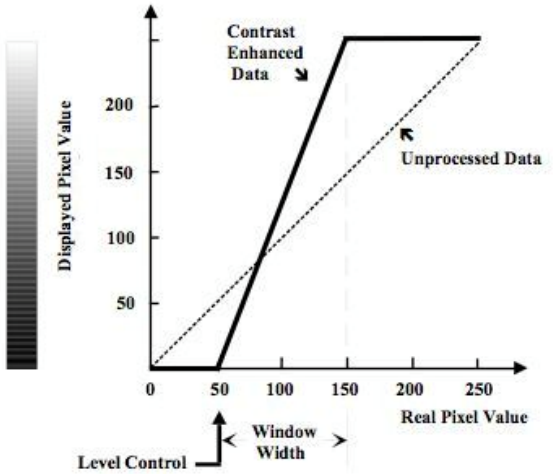
Min: 3
Max: 220
Mode: 157 (7480)



Count: 414720
Mean: 139.107
StdDev: 67.852

Min: 0
Max: 255
Mode: 198 (7480)

First, let's look at a formulaic way to express the contents of a linear LUT



$$LUT[i] = \begin{cases} 0, & if\ i < a \\ (i-a)\dfrac{255}{b-a}, & if\ (a \le i \le b) \\ 255, & if\ i > b \end{cases}$$

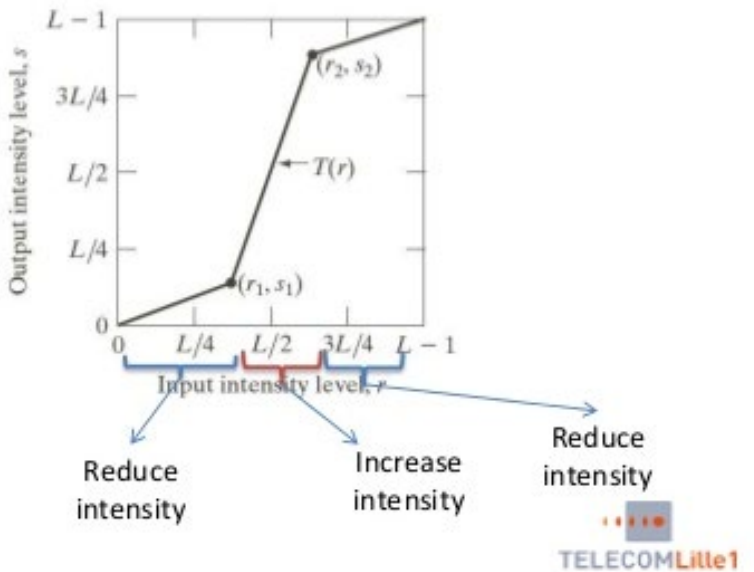
idx - *input*	I(idx) - *output*
0	0
…	0
50	0
51	1*(255/100) = (int)2.55 = 3
52	2*(255/100) = (int)5.1 = 5
53	3*(255/100) = (int)7.65 = 8
…	…
148	98*(255/100) = (int)249.9 = 250
149	99*(255/100) = (int)252.45 = 252
150	255
…	255
255	255

We may want to tailor a piecewise linear transformation to certain gray level regions – to increase contrast in certain ranges

Piecewise linear transformations

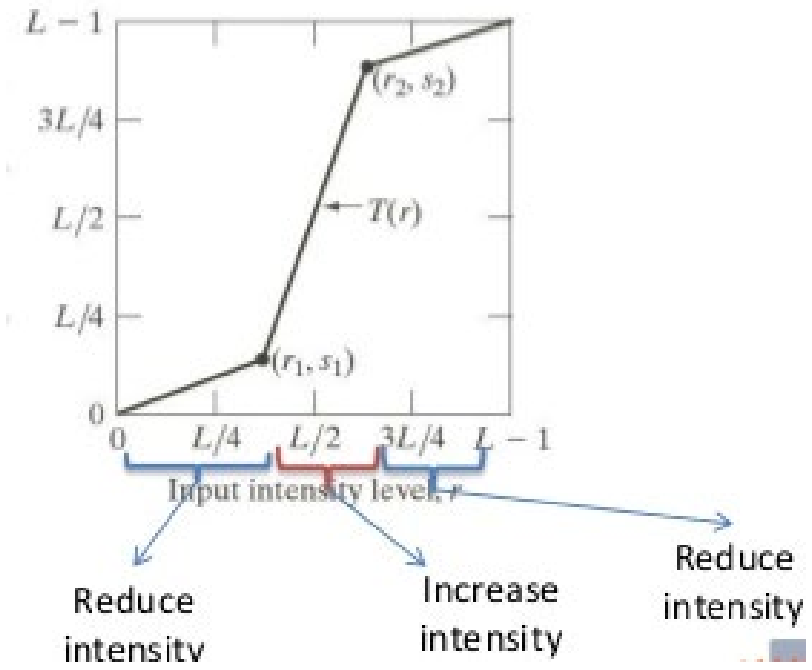
- Contrast stretching
  - Low contrast images can result from poor illumination

Low contrast image



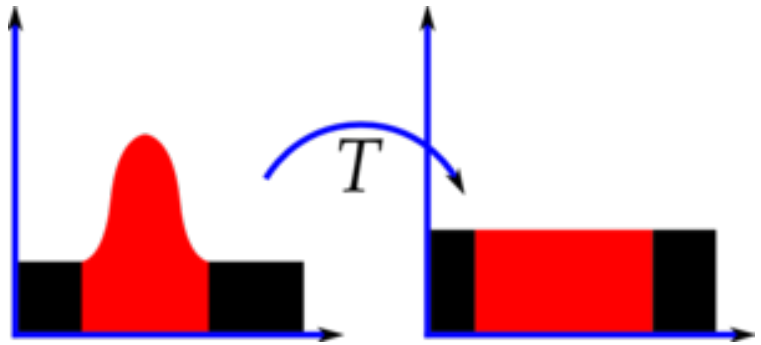
17

To do this, the LUT is calculated in pieces, using linear interpolation between each known point



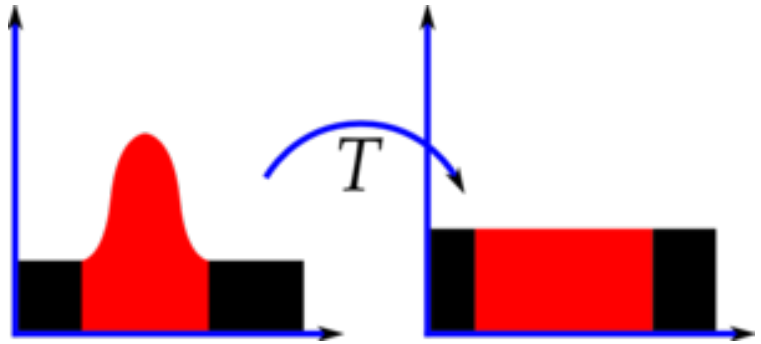
idx - *input*	I(idx) - *output*
0	0
1	(int)(1*s_1/r_1)
…	…
r_1-1	(int)((r_1-1)*s_1/r_1)
r_1	s_1
r_1+1	(int)(s_1 + (1)*(s_2-s_1)/(r_2-r_1))
…	…
r_2-1	(int)(s_1 + (r_2-r_1-1)*(s_2-s_1)/(r_2-r_1))
r_2	s_2
r_2+1	(int)(s_2 + (1)*(255-s_2)/(255-r_2))
…	
255	255

Histogram equalization is the adjusting of intensity values (usually via LUT) so that the output histogram is equal in height - ideally



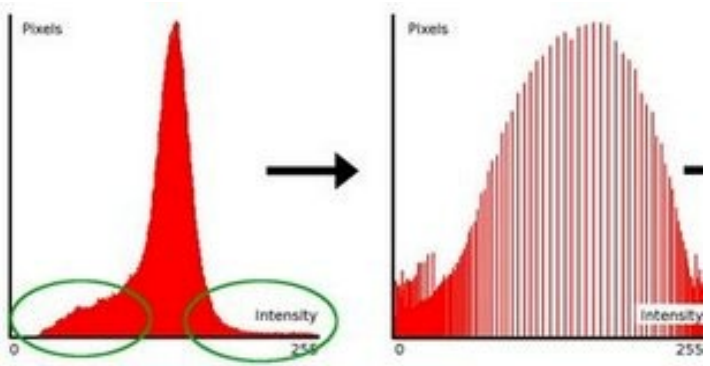
Though this is the idea – with a result that has a flat histogram – this rarely happens because of the limitations of mapping from and to quantized intensity values

Histogram equalization is the adjusting of intensity values (usually via LUT) so that the output histogram is equal in height - ideally

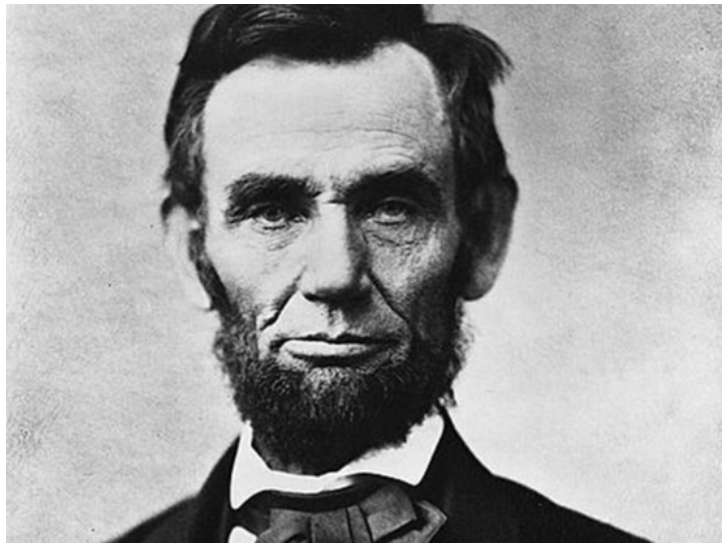


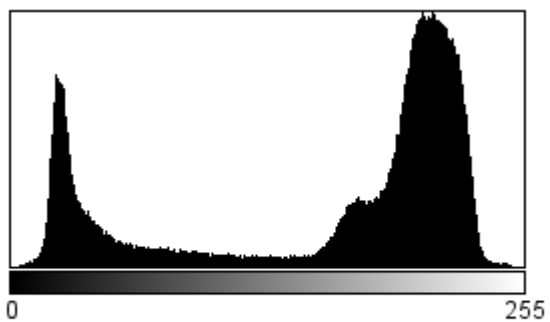
Though this is the idea – with a result that has a flat histogram – this rarely happens because of the limitations of mapping from and to quantized intensity values

This is more usually the result

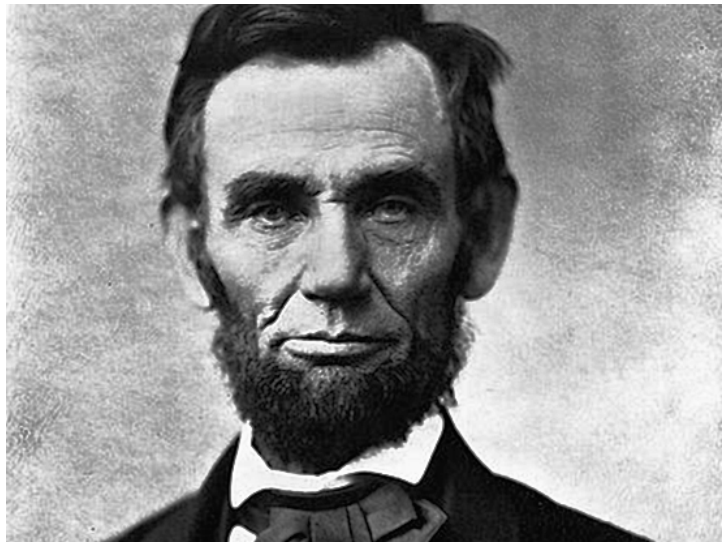


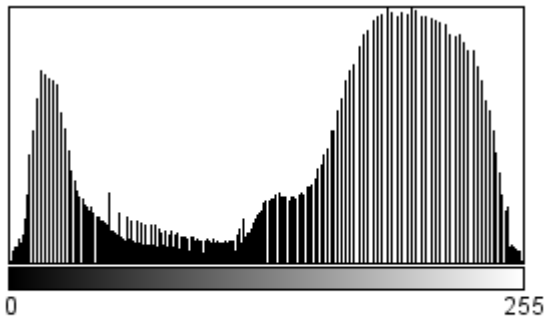
The precise objective is a resulting image where the histogram has equal area over each (reasonably sized) portion of the intensity range





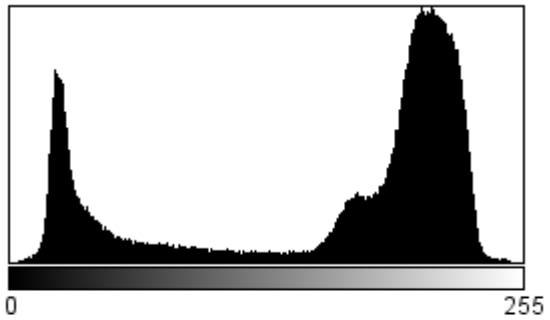
Count: 227150
Mean: 157.677
StdDev: 74.624
Min: 0
Max: 255
Mode: 210 (3775)



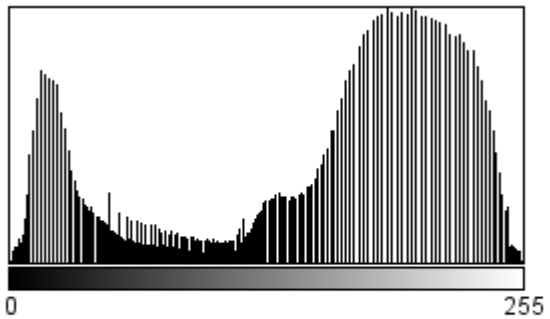


Count: 227150
Mean: 145.341
StdDev: 74.390
Min: 0
Max: 255
Mode: 200 (3775)

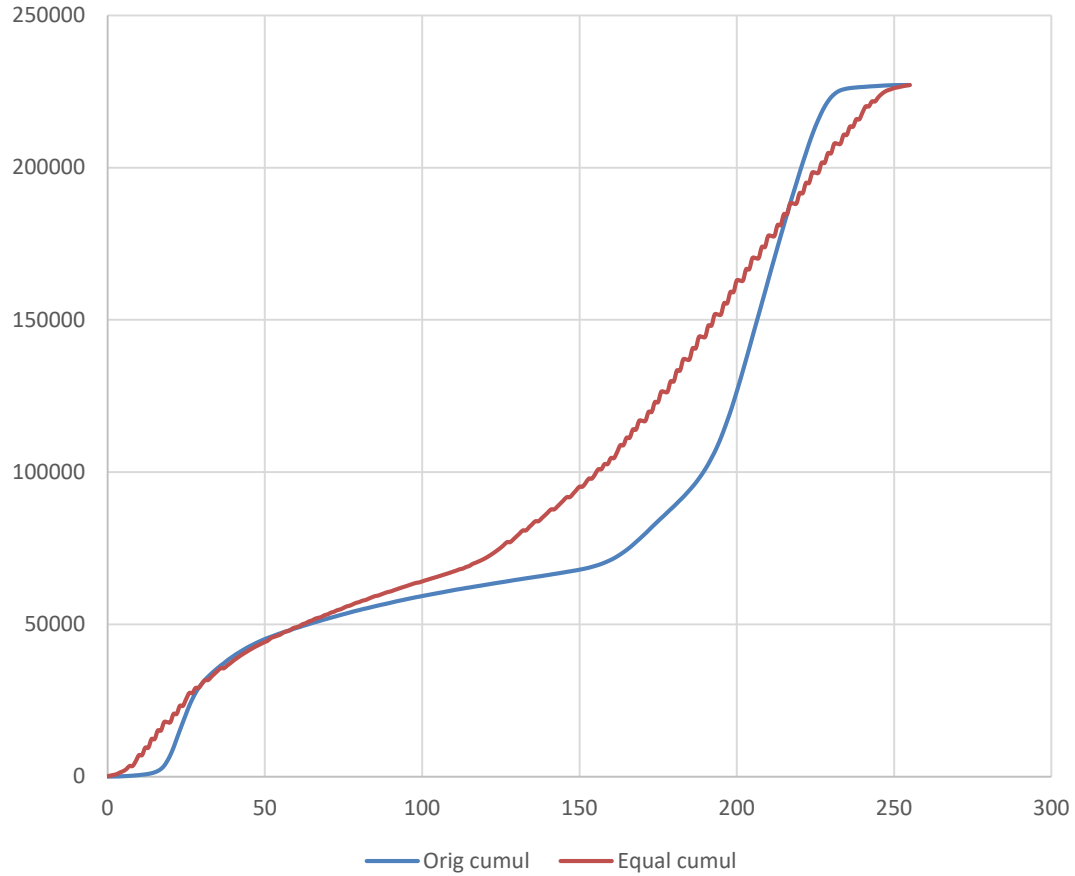
The cumulative histogram (where $CH[i] = \sum_{k=0}^{i} H[k]$) of the equalized image will be close (well, closer) to a line



Count: 227150 Min: 0
Mean: 157.677 Max: 255
StdDev: 74.624 Mode: 210 (3775)

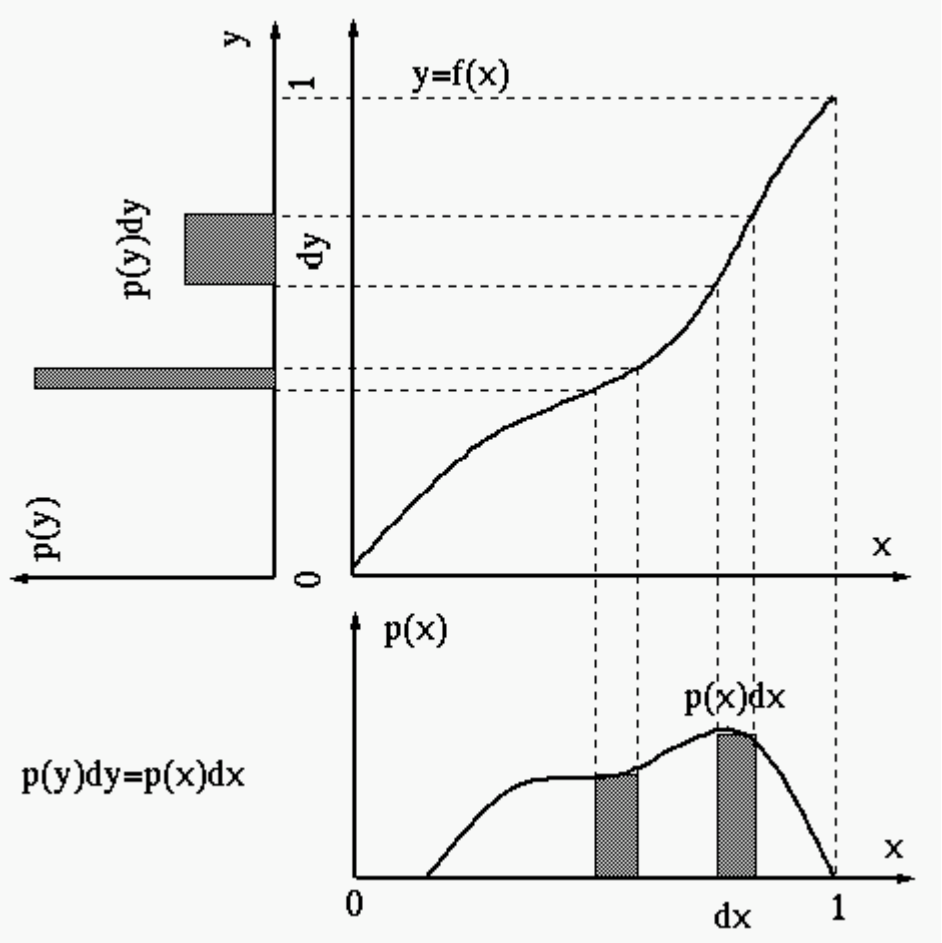


Count: 227150 Min: 0
Mean: 145.341 Max: 255
StdDev: 74.390 Mode: 200 (3775)

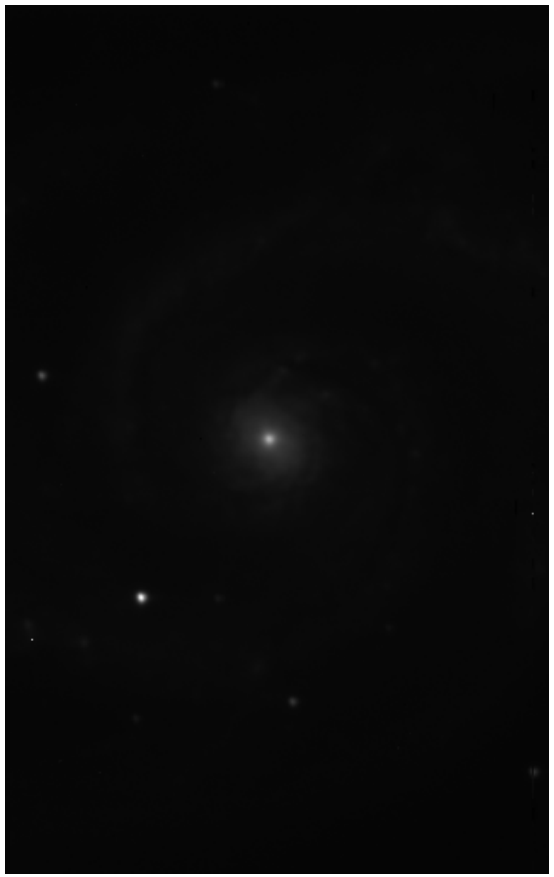


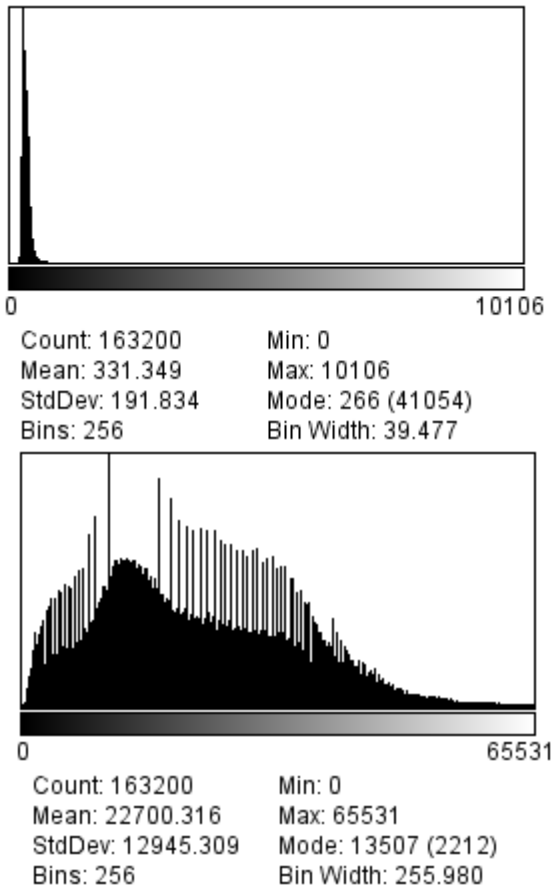
One way to fill the LUT to accomplish histogram equalization: successive piecewise linear functions for a set of regions in the input histogram

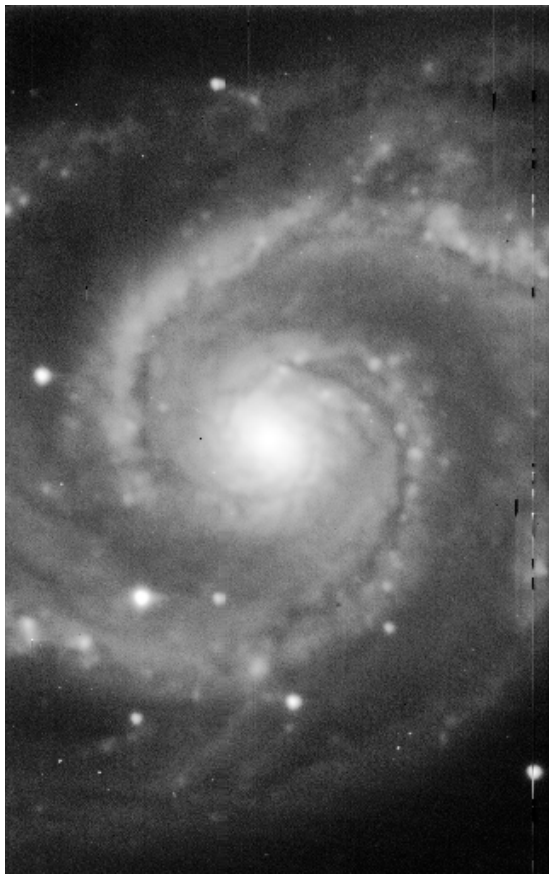
- The bottom 1/n of the output range should be a linear progression through whatever portion of the input range contains 1/n of the image pixels
- The next 1/n should progress through the portion of the input range containing the next 1/n of the image pixels
- and so on…

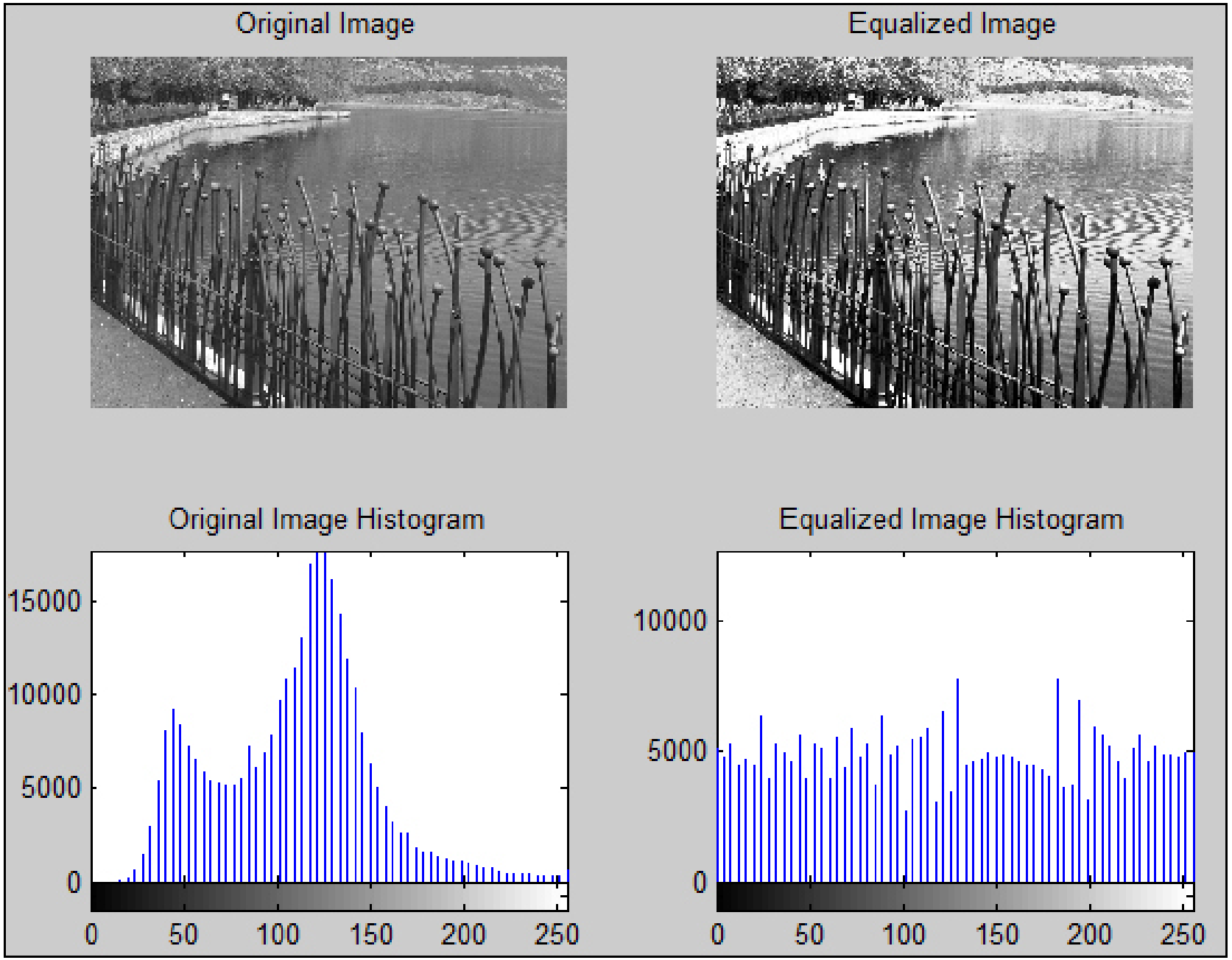


When we perform histogram equalization on the 16-bit image, the comb effect on the output is much reduced and visual result is better (though the improvement may be difficult to see)



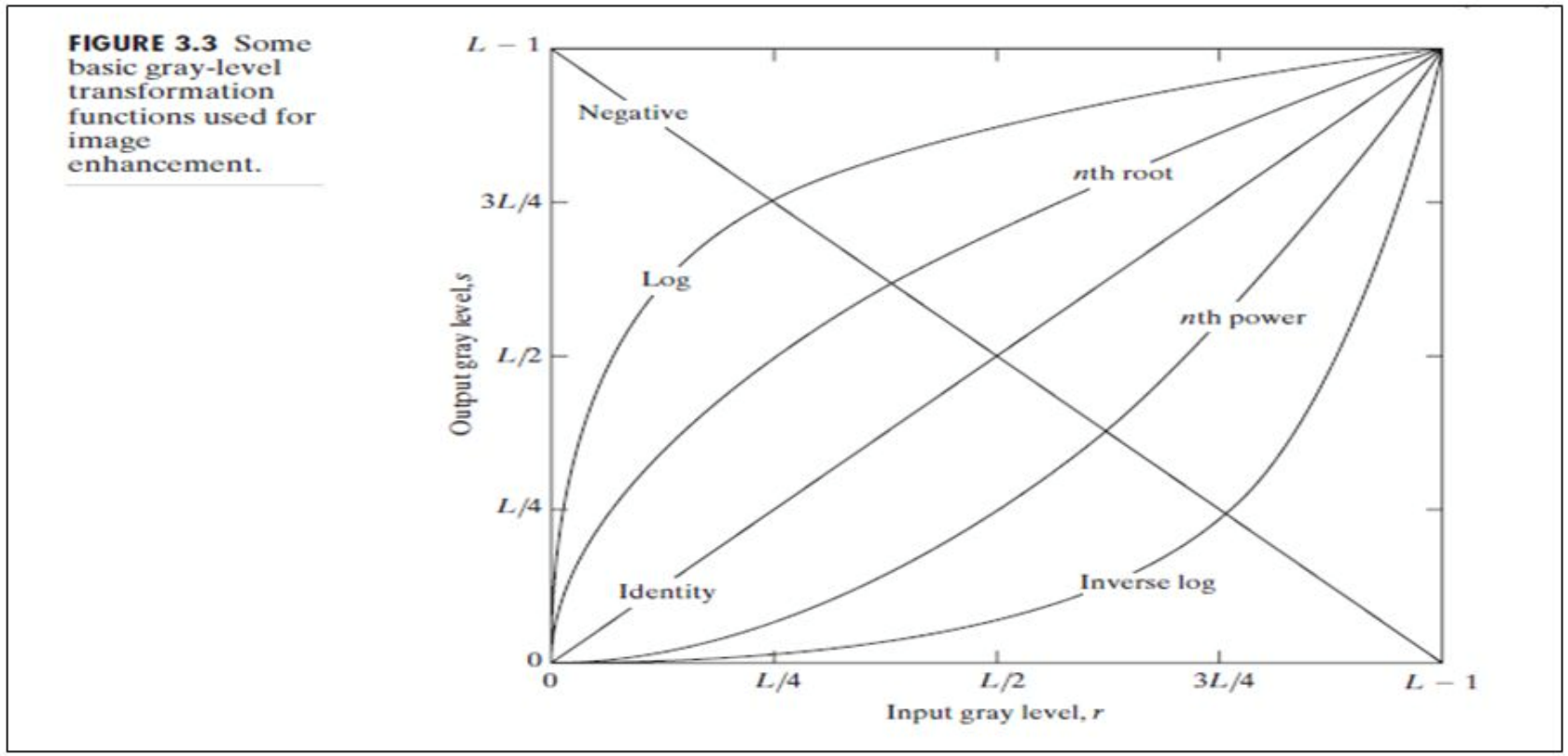





Original Image
Equalized Image
Original Image Histogram
Equalized Image Histogram
15000
10000
5000
0
0
50
100
150
200
250
10000
5000
0
0
50
100
150
200
250

Some Basic Intensity (Gray-level) Transformation Functions

Consider the following figure, which shows three basic types of functions used frequently for image enhancement:



Today's Objectives

- Intensity histograms
- Image Statistics
- Histogram Equalization
- Pixel Operations