Point Operations

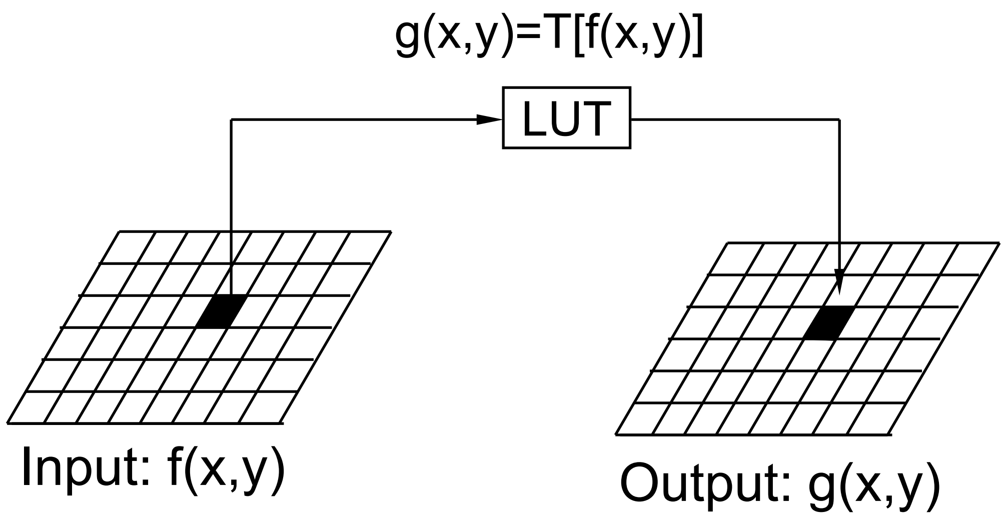
Prof. George Wolberg
Dept. of Computer Science
City College of New York

Objectives

- In this lecture we describe point operations commonly used in image processing:
  - Thresholding
  - Quantization (aka posterization)
  - Gamma correction
  - Contrast/brightness manipulation
  - Histogram equalization/matching

Point Operations

- Output pixels are a function of only one input point: $g(x,y) = T[f(x,y)]$
- Transformation T is implemented with a lookup table:
  - An input value indexes into a table and the data stored there is copied to the corresponding output position.
  - The LUT for an 8-bit image has 256 entries.

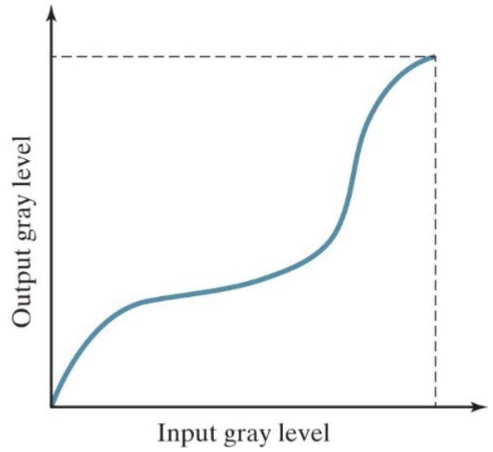


Graylevel Transformations

- **Point transformation**: changes a pixel's value without changing its location.

$$I'(x, y) = f(I(x, y))$$

Figure 3.7 A graylevel transformation maps input gray levels to output gray levels. Based on http://www.unit.eu/cours/videocommunication/Point_Transformation_histogram.pdf



ALGORITHM 3.5 Transform gray levels of an image

TRANSFORMGRAYLEVELS(I, f)

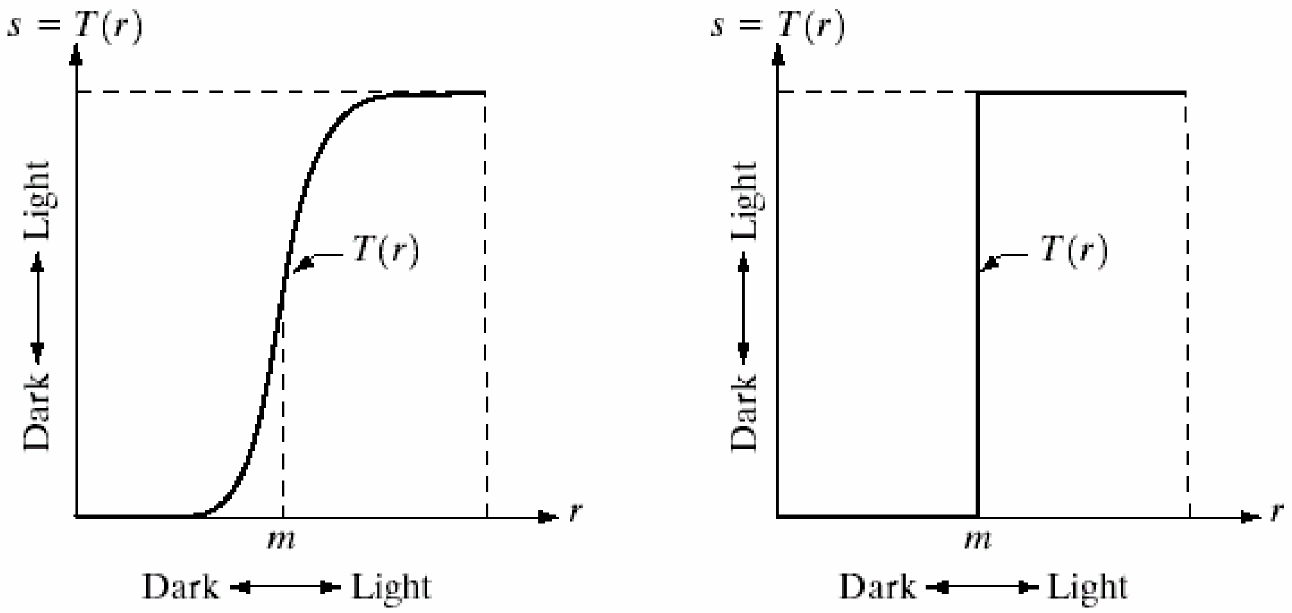
Input: grayscale image I, graylevel mapping f
Output: transformed image I'

```
1  for (x, y) ∈ I do
2      I'(x, y) ← f(I(x, y))
3  return I'
```

Graylevel Transformation *T*

Contrast enhancement:
Darkens levels below *m*
Brightens levels above *m*

Thresholding:
Replace values below *m* to black (0)
Replace values above *m* to white (255)



a b

FIGURE 3.2 Gray-level transformation functions for contrast enhancement.

Thresholding

- **Thresholding:** takes a grayscale image and sets every output pixel to 1 if its input gray level is above a certain threshold, or to 0 otherwise:

$$I'(x,y) = \begin{cases} 1 & \text{if } I(x,y) > \tau \\ 0 & \text{otherwise} \end{cases}$$

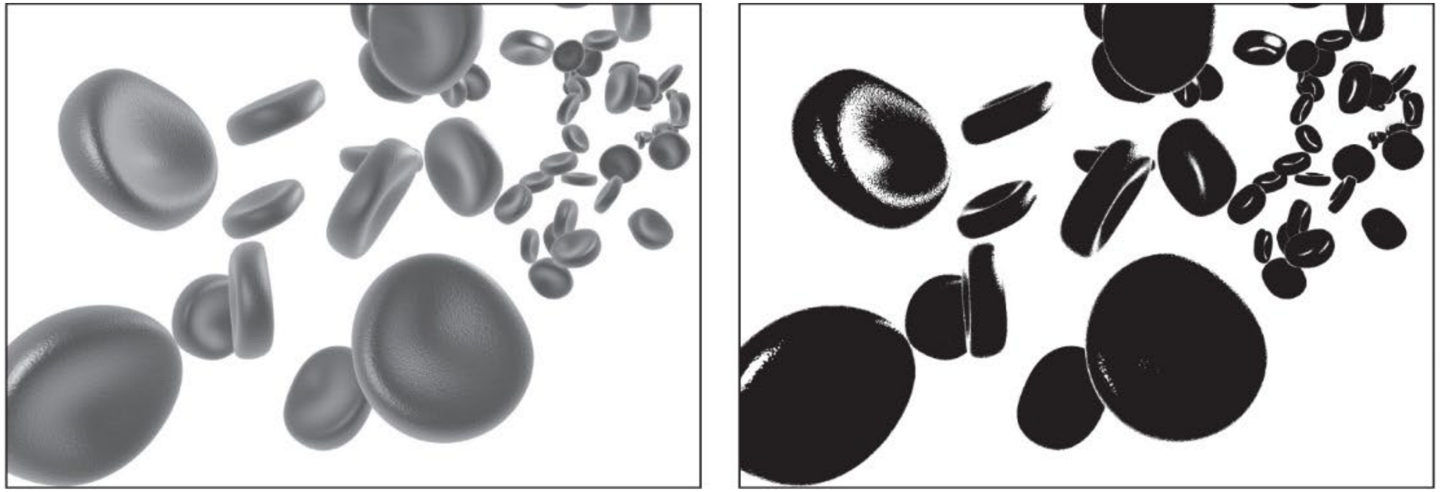
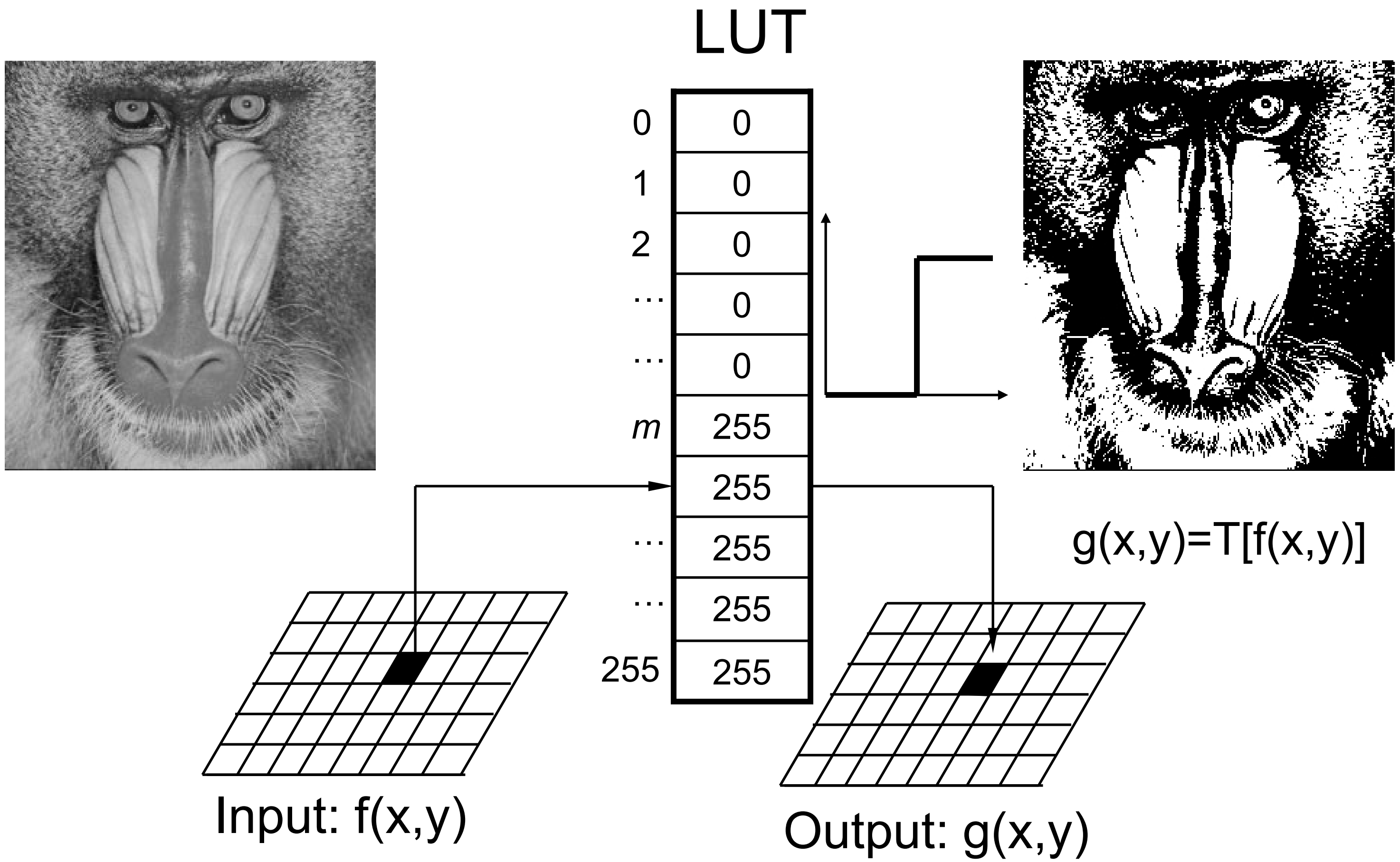


Figure 3.14 An 8-bit grayscale image (left), and the binarized result obtained by thresholding with $\tau = 150$ (right).

Ciprian Stremtan / Shutterstock.com

Lookup Table: Threshold

- Init LUT with samples taken from thresholding function T



Threshold Program

- Straightforward implementation:

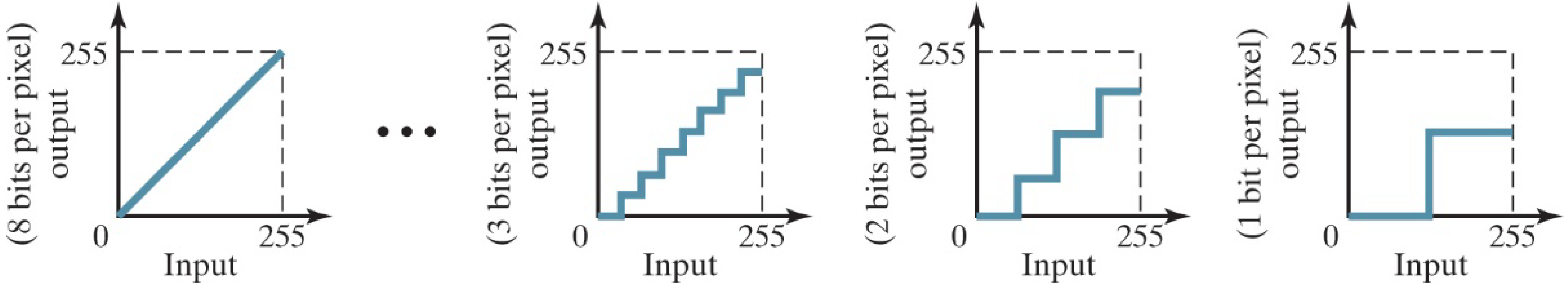
```
// iterate over all pixels
for(i=0; i<total; i++) {
  if(in[i] < thr)      out[i] = BLACK;
  else                 out[i] = WHITE;
}
```

- Better approach: exploit LUT to avoid `total` comparisons:

```
// init lookup tables
for(i=0; i<thr; i++) lut[i] = BLACK;
for(; i<MXGRAY; i++) lut[i] = WHITE;

// iterate over all pixels
for(i=0; i<total; i++) out[i] = lut[in[i]];
```

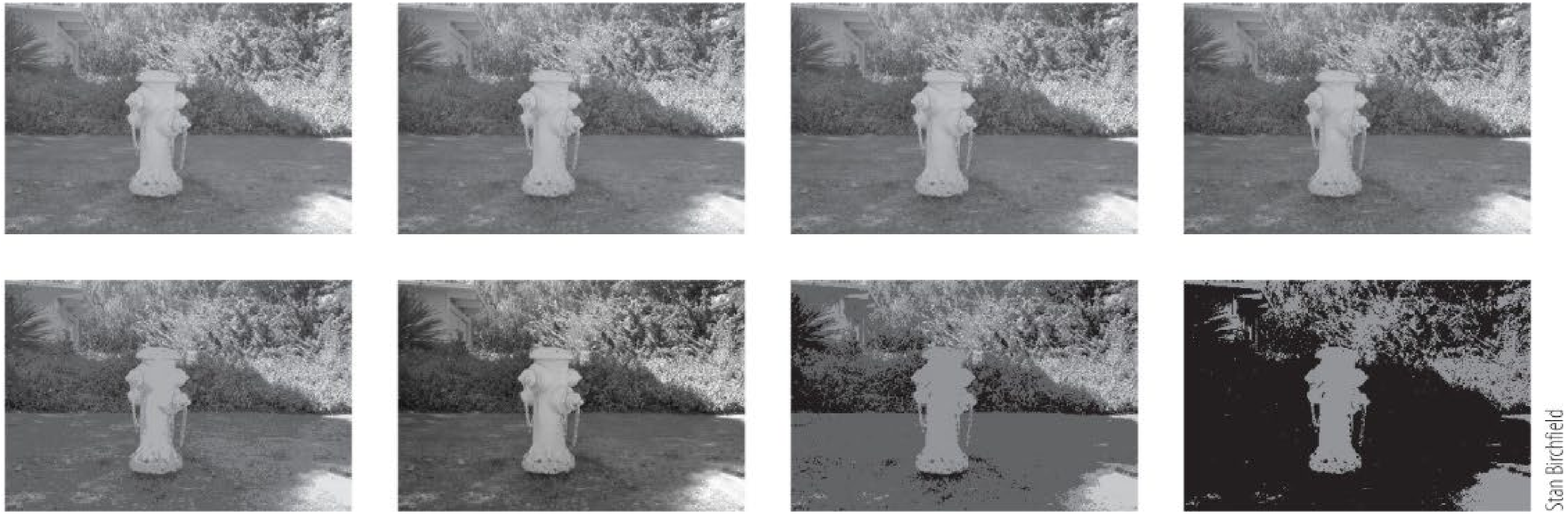
Quantization

Figure 3.16 Quantization discards the lower-order bits via a staircase function, with the number of stairs determined by the number of bits retained. From right to left: Only 1 bit is retained, so the gray levels in the dark half (less than 128) map to 0, while the gray levels in the bright half (above 127) map to 128 (binary: 10000000); 2 bits are retained, so gray levels are mapped to either 0, 64, 128, or 192; 3 bits are retained, so all gray levels are mapped to either 0, 32, 64, 96, 128, 160, 192, or 224; all 8 bits are retained (no quantization, the identity function).



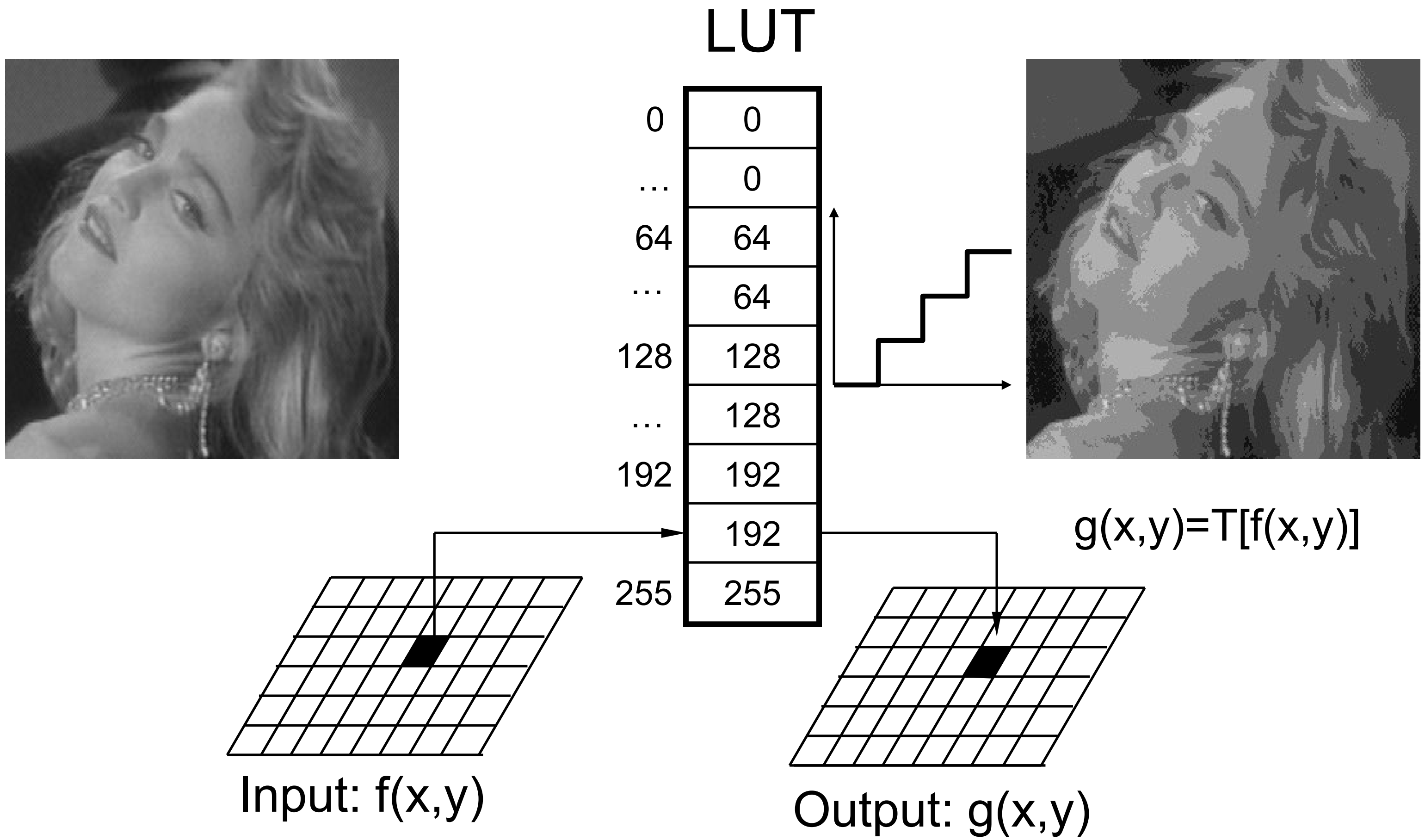
Quantization

Figure 3.17 From left-to-right and top-to-bottom: An original 8-bit-per-pixel image of a fire hydrant and its quantized versions with 7, 6, 5, 4, 3, 2, and 1 bit per pixel. Note that the image is quite recognizable with as few as 3 bits per pixel.



Lookup Table: Quantization

- Init LUT with samples taken from quantization function T



Quantization Program

- Straightforward implementation:

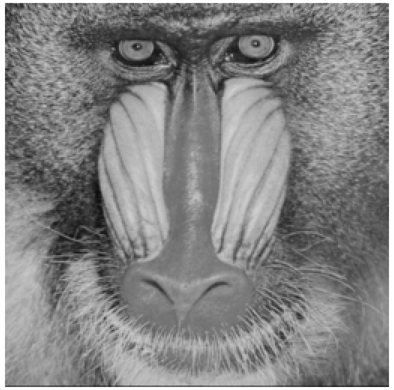
```
// iterate over all pixels
scale = MXGRAY / levels;
for(i=0; i<total; i++)
       out[i] = scale * (int) (in[i]/scale);
```

- Better approach: exploit LUT to avoid `total` mults/divisions:

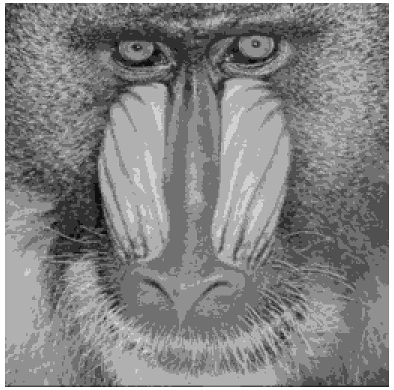
```
// init lookup tables
scale = MXGRAY / levels;
for(i=0; i<MXGRAY; i++)
       lut[i] = scale * (int) (i/scale);

// iterate over all pixels
for(i=0; i<total; i++) out[i] = lut[in[i]];
```

Quantization Artifacts

- False contours associated with quantization are most noticeable in smooth areas
- These artifacts are obscured in highly textured regions



Original image

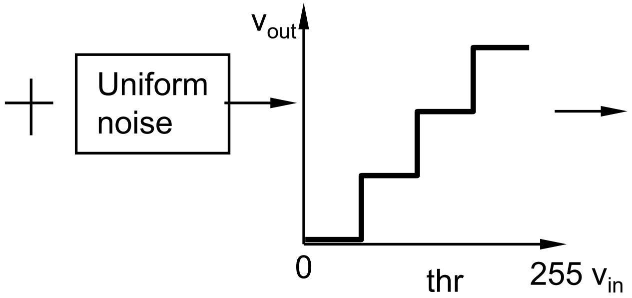


Quantized to 8 levels

Dither Signal

- Reduce quantization error by adding uniformly distributed white noise (dither signal) to the input image prior to quantization.
- Dither hides objectional artifacts.
- To each pixel of the image, add a random number in the range [-*m, m*], where *m* is MXGRAY/quantization-levels.

8 bpp (256 levels)



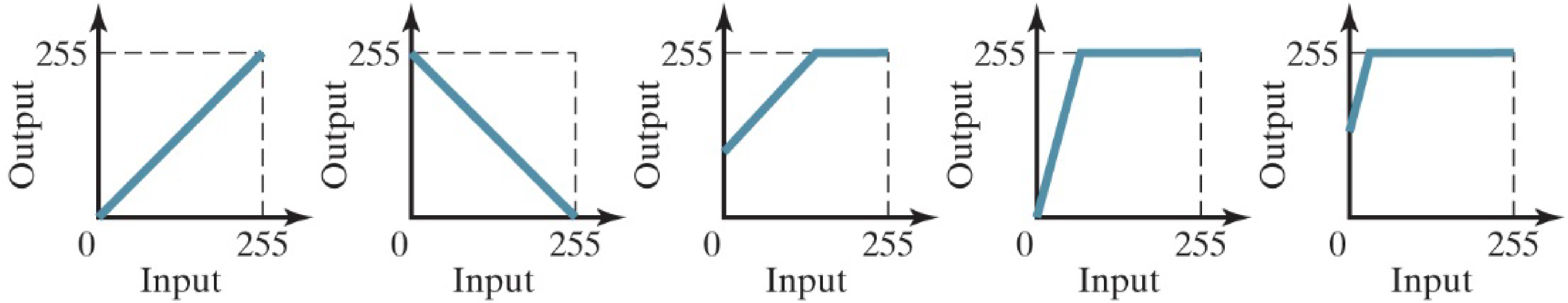
3 bpp (8 levels)

Comparison

Arithmetic Operations

- A useful class of graylevel transformations is the set of arithmetic operations, depicted in graphical form in the following figure:

Figure 3.8 Arithmetic graylevel transformations. From left to right: identity, inversion, addition (bias), multiplication (gain), and gain-bias transformation, where saturation arithmetic prevents the output from exceeding the valid range. Note that the slope remains 1 under addition, while the mapping passes through the origin under multiplication.



Saturation Arithmetic

Clamp arithmetic operation to lie in [0, 255] range:

[216 171 134 97 52] + 100 = [255 255 234 197 152]

[216 171 134 97 52] - 100 = [116 71 34 0 0]

ALGORITHM 3.6 Apply gain and bias to a grayscale image, using saturation arithmetic

ApplyGainAndBias(I, b, c)

Input: grayscale image I, constants b, c

Output: grayscale image I' with increased brightness and contrast

for $(x, y) \in I$ **do**

  $I'(x, y) \leftarrow \text{Min}(\text{Max}(\text{Round}(I(x, y) * c + b), 0), 255)$

return I'

Gain-Bias Transformations

ALGORITHM 3.6 Apply gain and bias to a grayscale image, using saturation arithmetic

APPLYGAINANDBIAS(I, b, c)

Input: grayscale image I, constants b, c
Output: grayscale image I' with increased brightness and contrast

for $(x, y) \in I$ **do**
    $I'(x, y) \leftarrow \text{MIN}(\text{MAX}(\text{ROUND}(I(x, y) * c + b), 0), 255)$
return I'

Figure 3.10 Improving image quality by applying gain or bias to an image. From left to right: Original image, brightened image by adding a constant value $(b = 50)$, higher contrast image by multiplying a constant value $(c = 2.5)$. Source: Movie *Hoop Dreams*

Piecewise-Linear Transformation Functions

- Advantage: The form of piecewise functions can be arbitrarily complex
- Disadvantage: Their specification requires considerably more user input

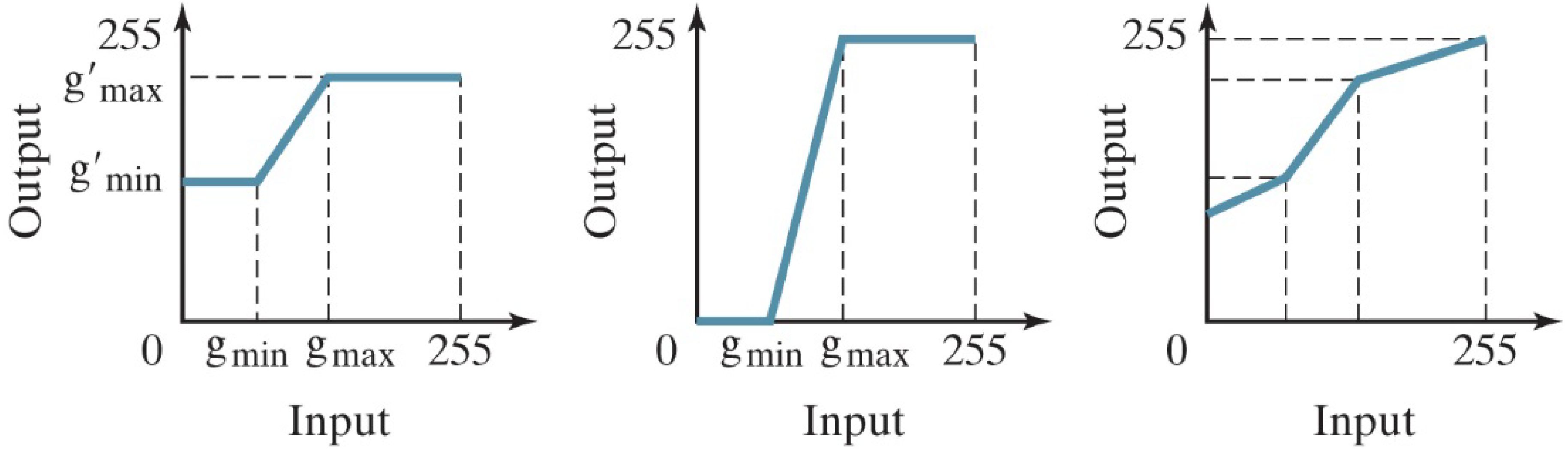
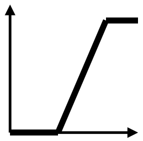
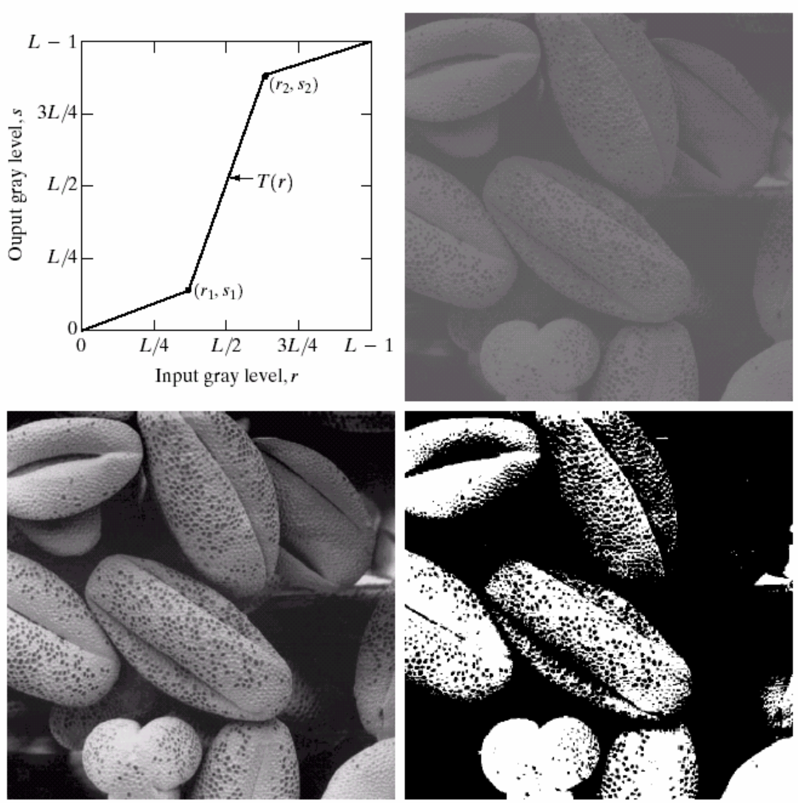


Figure 3.11 Left: Linear contrast stretching maps all the gray levels between g_{min} and g_{max} to the range g'_{min} to g'_{max}. Middle: If $g'_{min} = 0$ and $g'_{max} = 255$, then the full output range is used. RIGHT: A piecewise linear contrast stretch can model any graylevel transformation with arbitrary precision.

Linear Contrast Stretching

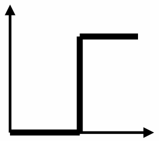
- Low contrast may be due to poor illumination, a lack of dynamic range in the imaging sensor, or even a wrong setting of a lens aperture during acquisition.
- Applied contrast stretching: $(r_1,s_1) = (r_{min},0)$ and $(r_2,s_2) = (r_{max},L-1)$





a b
c d

FIGURE 3.10 Contrast stretching. (a) Form of transformation function. (b) A low-contrast image. (c) Result of contrast stretching. (d) Result of thresholding. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra, Australia.)



Linear Contrast Stretching Transformation

- **Linear contrast stretch:** A transformation that specifies a line segment that maps gray levels between g_{min} and g_{max} in the input image to the gray levels g'_{min} and g'_{max} in the output image according to a linear function:

$$I'(x,y) = \frac{g'_{\max} - g'_{\min}}{g_{\max} - g_{\min}}(I(x,y) - g_{\min}) + g'_{\min}$$

Linear Contrast Stretching Examples

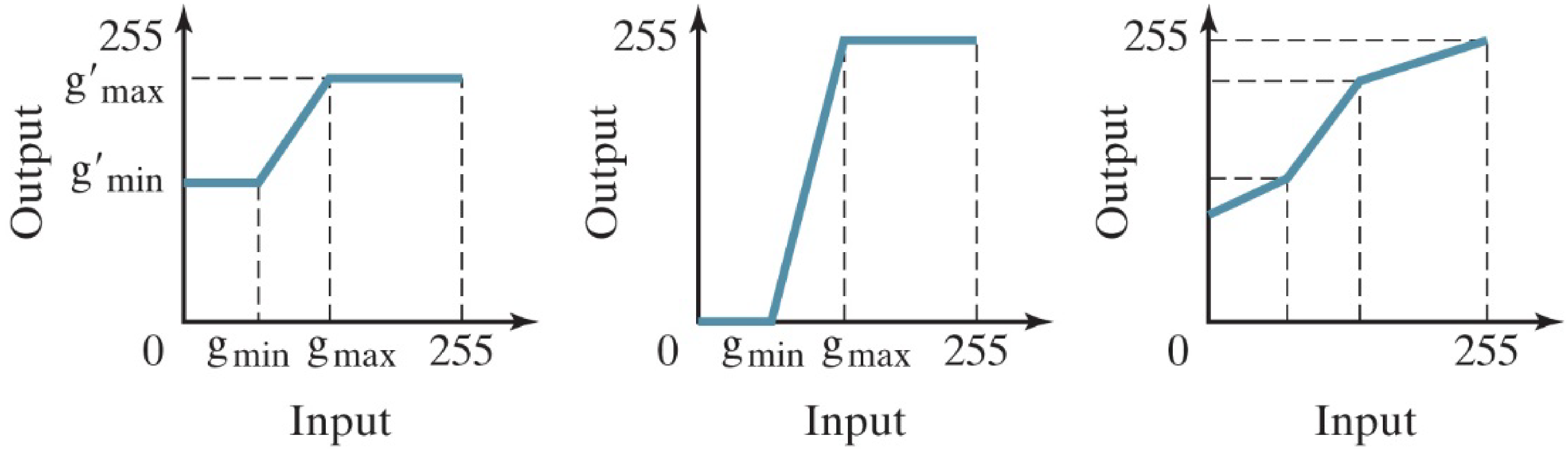


Figure 3.11 Left: Linear contrast stretching maps all the gray levels between g_{min} and g_{max} to the range g'_{min} to g'_{max}. Middle: If $g'_{min} = 0$ and $g'_{max} = 255$, then the full output range is used. RIGHT: A piecewise linear contrast stretch can model any graylevel transformation with arbitrary precision.

Analytic Transformations

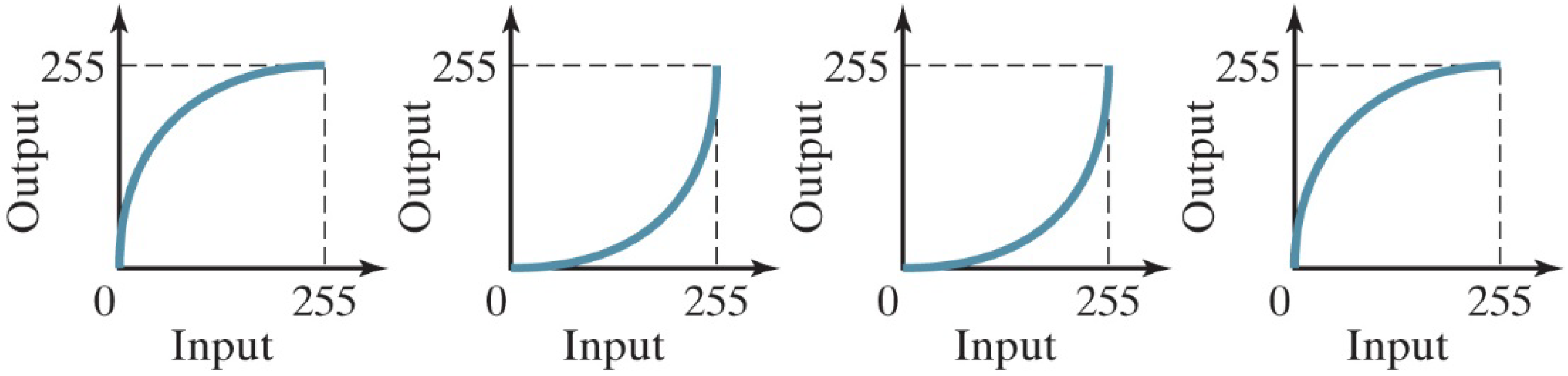
- Graylevel transformations can be specified using analytic functions such as the logarithm, exponential, or power functions:

$$I'(x, y) = \log(I(x, y))$$

$$I'(x, y) = \exp(I(x, y))$$

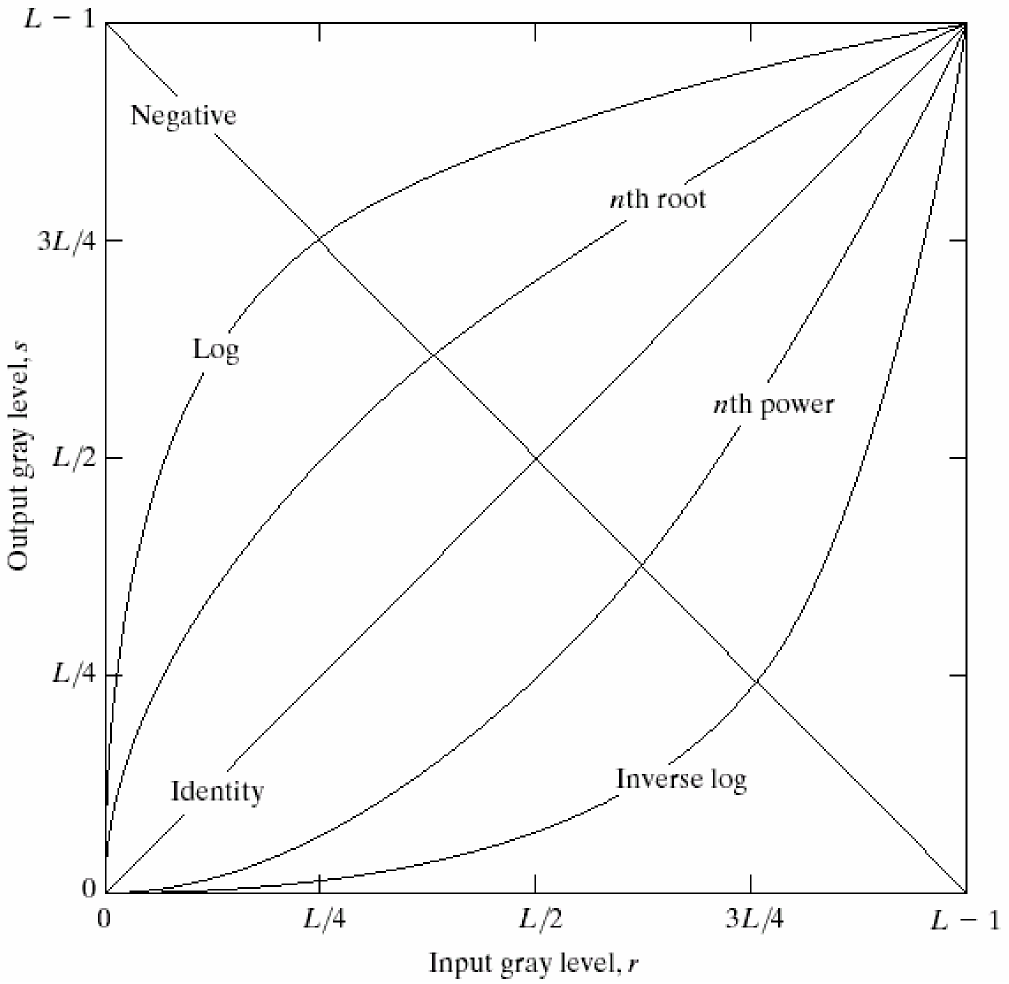
$$I'(x, y) = (I(x, y))^{\gamma}$$

Figure 3.13 Analytic graylevel mapping. From left to right: logarithm, exponential, gamma expansion ($\gamma = 2$), and gamma compression ($\gamma = 0.5$). All transformations are monotonically nondecreasing.



Analytic Transformation Examples

FIGURE 3.3 Some basic gray-level transformation functions used for image enhancement.



- Linear function
  - Negative and identity transformations
- Logarithmic function
  - Log and inverse-log transformations
- Power-law function
  - n^{th} power and n^{th} root transformations

Image Negatives

- Negative transformation : $\mathbf{s = (L-1) - r}$
- Reverses the intensity levels of an image.
- Suitable for enhancing white or gray detail embedded in dark regions of an image, especially when black area is large.

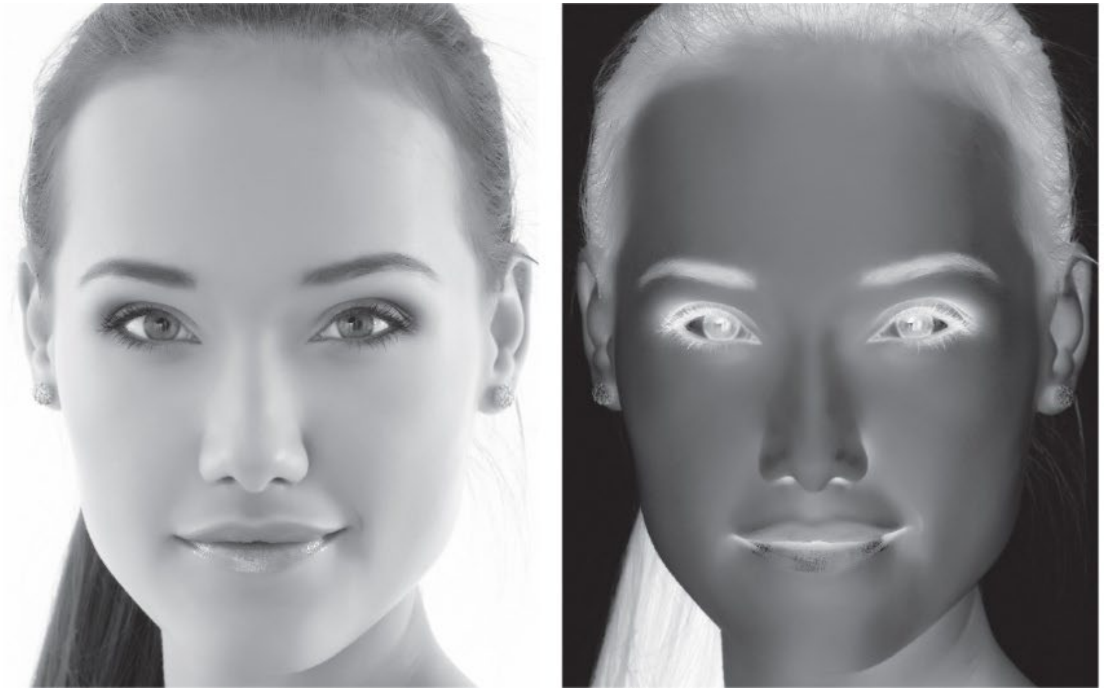


Figure 3.9 An 8-bit grayscale image (left), and the inverted image obtained by subtracting each pixel from 255 (right).

vita khorzhevska / Shutterstock.com

Log Transformations

- Log transformation : $s = c \log (1+r)$
- c is constant and $r \geq 0$
- Log curve maps a narrow range of low graylevels in input into a wider range of output levels.
- Expands range of dark image pixels while shrinking bright range.
- Inverse log expands range of bright image pixels while shrinking dark range.

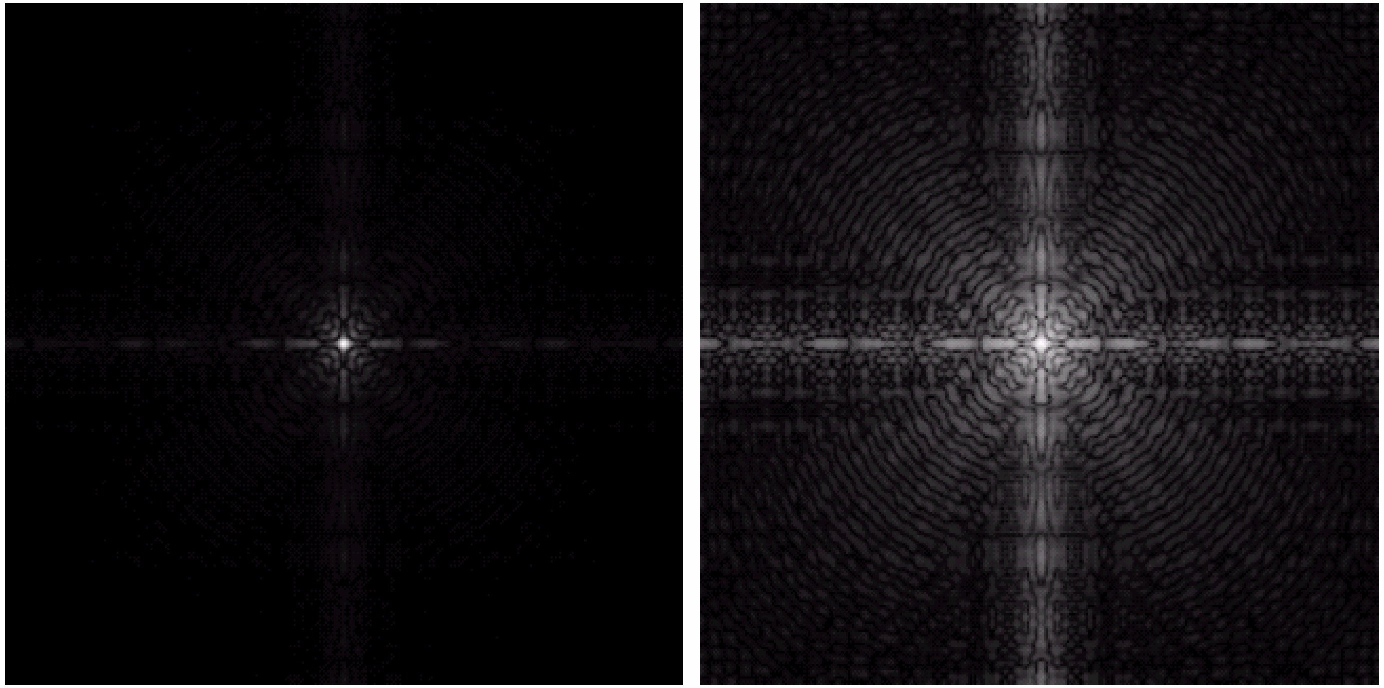
Example of Logarithm Image

- Fourier spectrum image can have intensity range from 0 to 10^6 or higher.
- Log transform lets us see the detail dominated by large intensity peak.

- Must now display [0,6] range instead of [0,10^6] range.
- Rescale [0,6] to the [0,255] range.

a b

FIGURE 3.5 (a) Fourier spectrum. (b) Result of applying the log transformation given in Eq. (3.2-2) with $c = 1$.



Power-Law Transformations

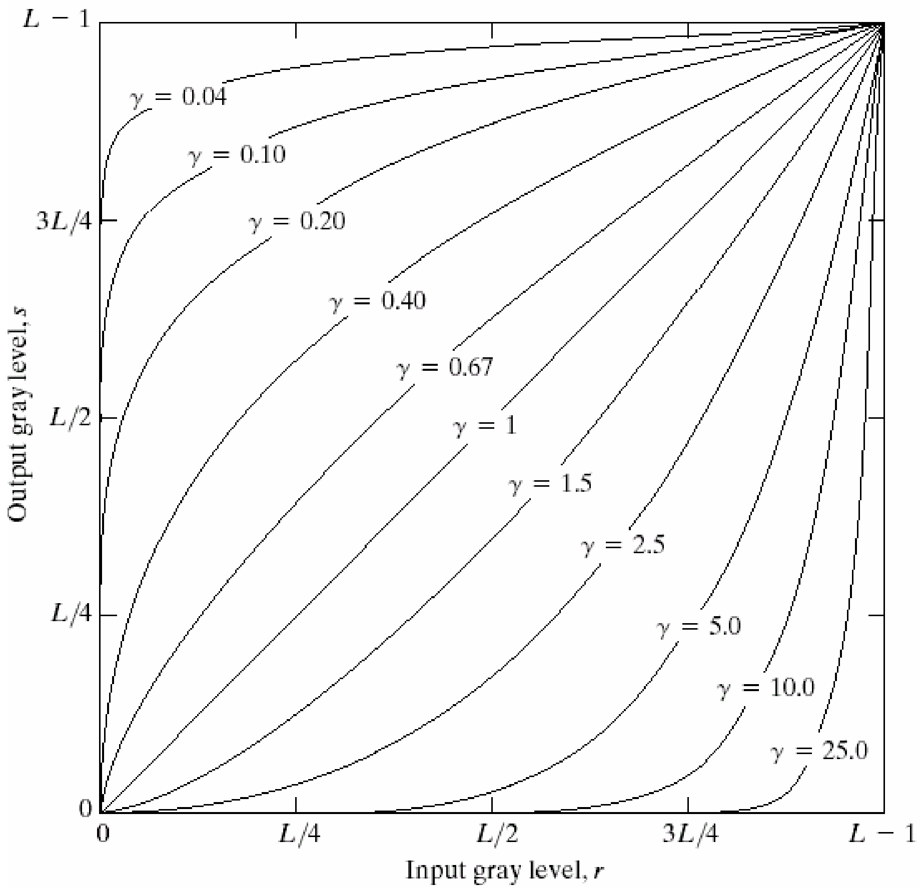


FIGURE 3.6 Plots of the equation $s = cr^\gamma$ for various values of γ ($c = 1$ in all cases).

$$s = cr^\gamma$$

- c and γ are positive constants
- Power-law curves with fractional values of γ map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels.
- $c = \gamma = 1$ ⇨ identity function

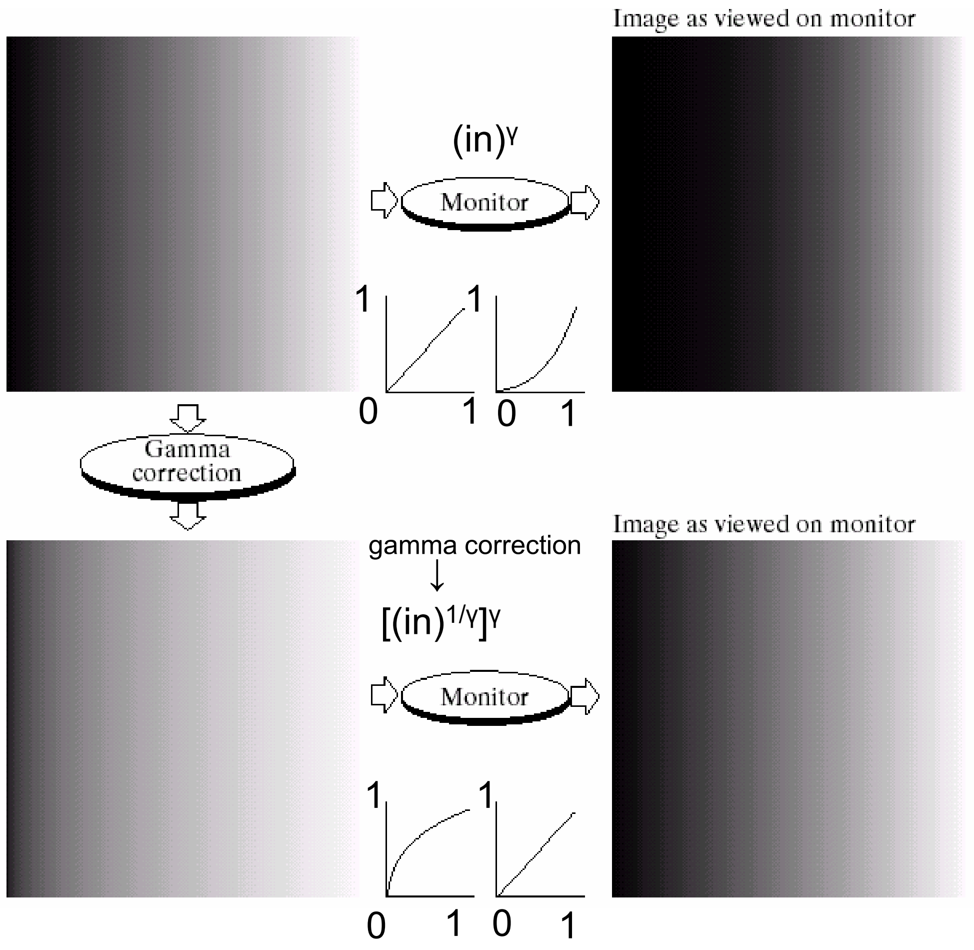
Gamma Correction

- Gamma correction is an operation that is rooted in the nonlinear mapping of intensities displayed by cathode ray tubes (CRTs).
- CRTs worked by shooting electrons at the phosphor that coated the CRT monitor. The excited phosphor would display the pixel at an intensity proportional to the input value.
- The monitor would produce *out* = in^{γ}, where $\gamma = 2.2$ and *in* is in [0,1] range.
- Since images appeared darker than desired, gamma correction achieved a perceptually desirable result by altering the input to brighter values that would appear correct after passing through the nonlinear mapping of the display.

Gamma Correction Example

a b
c d

FIGURE 3.7
(a) Linear-wedge gray-scale image.
(b) Response of monitor to linear wedge.
(c) Gamma-corrected wedge.
(d) Output of monitor.

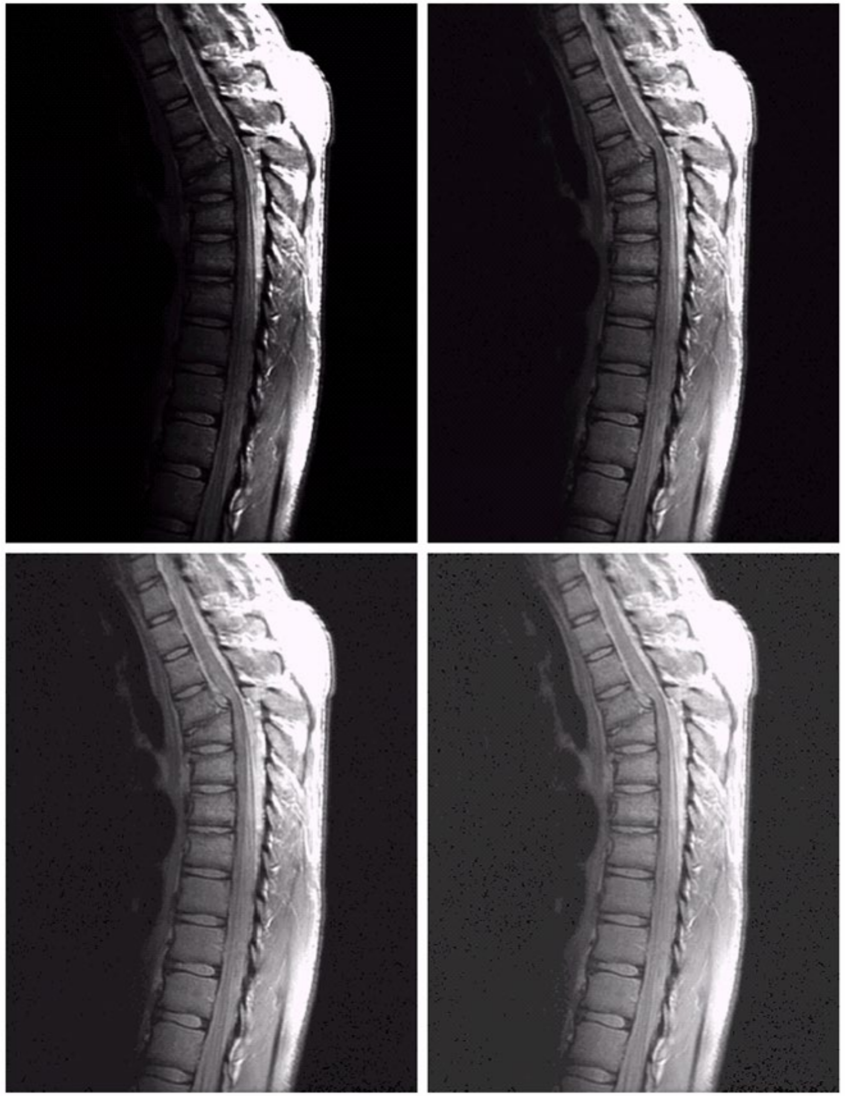


- Cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with γ varying from 1.8 to 2.5
- This darkens the picture.
- Gamma correction is done by preprocessing the image before inputting it to the monitor.

Gamma Correction Program

```
// init lookup table
double exponent = 1.0 / gamma;    // gamma correction exponent
for(i=0; i<=MaxGray; i++){
        // 1) normalize input graylevel i into [0,1] range
        // 2) raise i to the 1/gamma for gamma correction
        // 3) restore the [0,1] range back to [0,max]
        lut[i] = MaxGray * pow(((double) i/MaxGray), exponent);
}

// iterate over all pixels
for(i=0; i<total; i++) out[i] = lut[in[i]];
```

Example: MRI



a b
c d

FIGURE 3.8 (a) Magnetic resonance (MR) image of a fractured human spine. (b)–(d) Results of applying the transformation in Eq. (3.2-3) with $c = 1$ and $\gamma = 0.6, 0.4,$ and 0.3, respectively. (Original image for this example courtesy of Dr. David R. Pickens, Department of Radiology and Radiological Sciences, Vanderbilt University Medical Center.)

(a) Dark MRI. Expand graylevel range for contrast manipulation ⇨ $\gamma < 1$

(b) $\gamma = 0.6$, c=1

(c) $\gamma = 0.4$ (best result)

(d) $\gamma = 0.3$ (limit of acceptability)

When γ is reduced too much, the image begins to reduce contrast to the point where it starts to have a “washed-out” look, especially in the background

Example: Aerial Image

a b
c d

FIGURE 3.9 (a) Aerial image. (b)–(d) Results of applying the transformation in Eq. (3.2-3) with $c = 1$ and $\gamma = 3.0, 4.0,$ and 5.0, respectively. (Original image for this example courtesy of NASA.)

Washed-out image. Shrink graylevel range ⇨ $\gamma > 1$

(b) $\gamma = 3.0$ (suitable)

(c) $\gamma = 4.0$ (suitable)

(d) $\gamma = 5.0$

(High contrast; the image has areas that are too dark; some detail is lost)

Graylevel Slicing

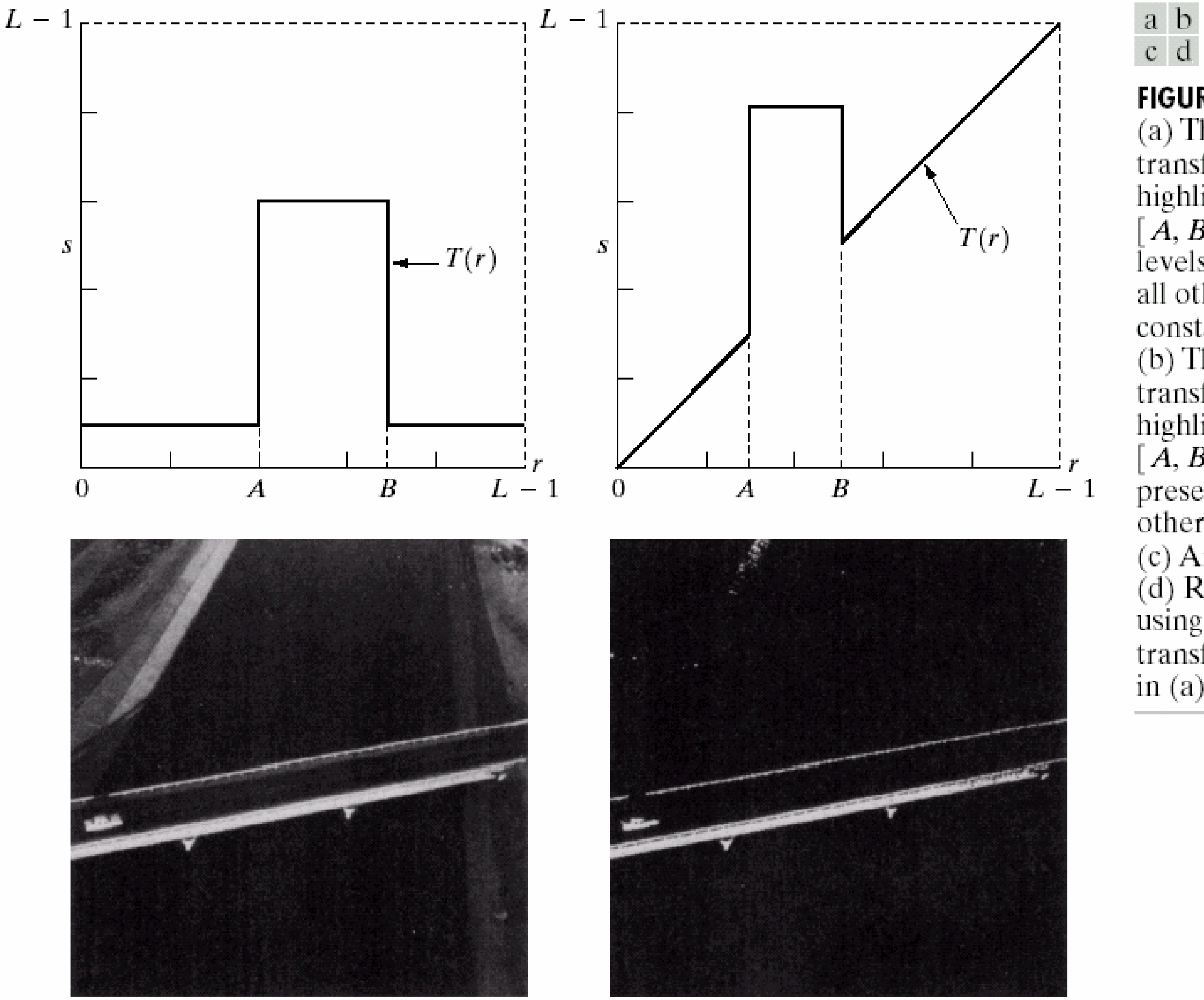
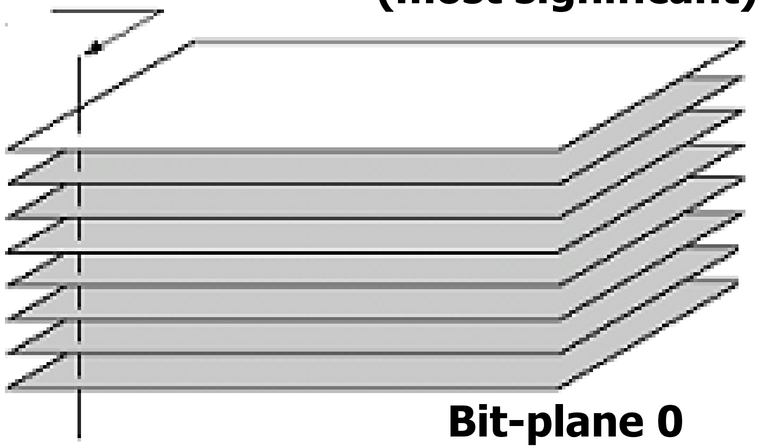


FIGURE 3.11 (a) This transformation highlights range [*A*, *B*] of gray levels and reduces all others to a constant level. (b) This transformation highlights range [*A*, *B*] but preserves all other levels. (c) An image. (d) Result of using the transformation in (a).

Bit-plane slicing

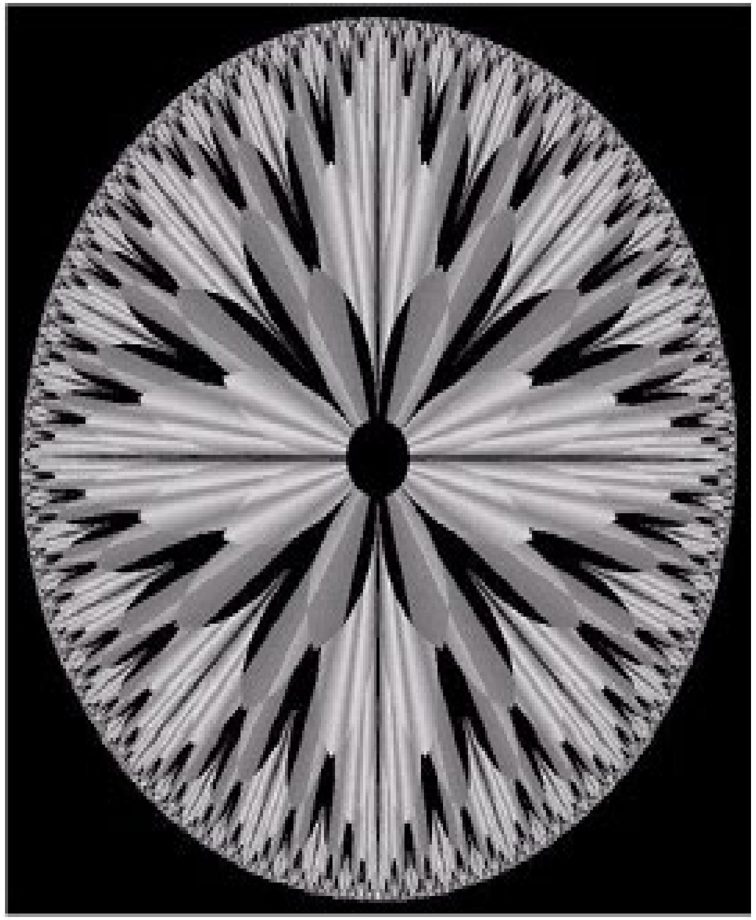


- Highlighting the contribution made to total image appearance by specific bits
- Suppose each pixel is represented by 8 bits

  Higher-order bits contain the majority of the visually significant data

  Useful for analyzing the relative importance played by each bit of the image

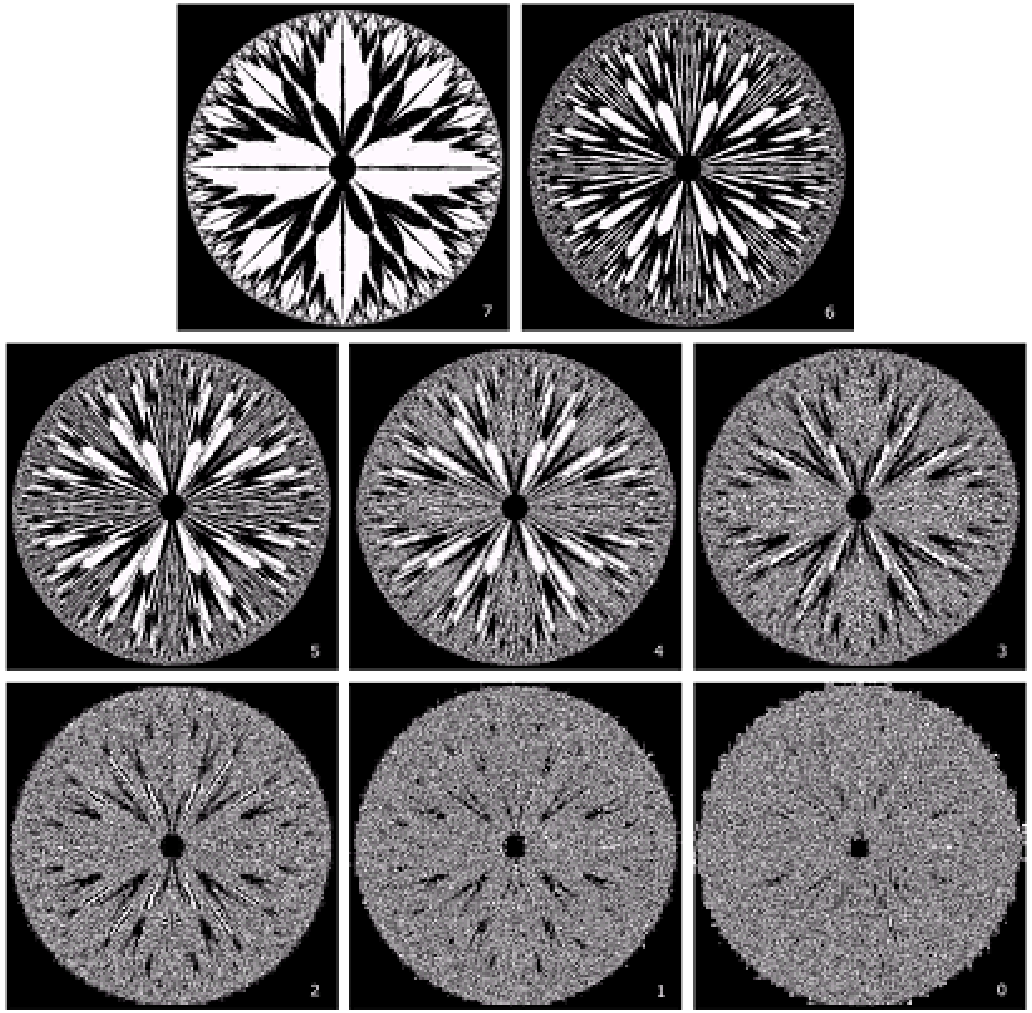
Example



An 8-bit fractal image

- The (binary) image for bit-plane 7 can be obtained by processing the input image with a thresholding graylevel transformation.
  - Map all levels between 0 and 127 to 0
  - Map all levels between 129 and 255 to 255

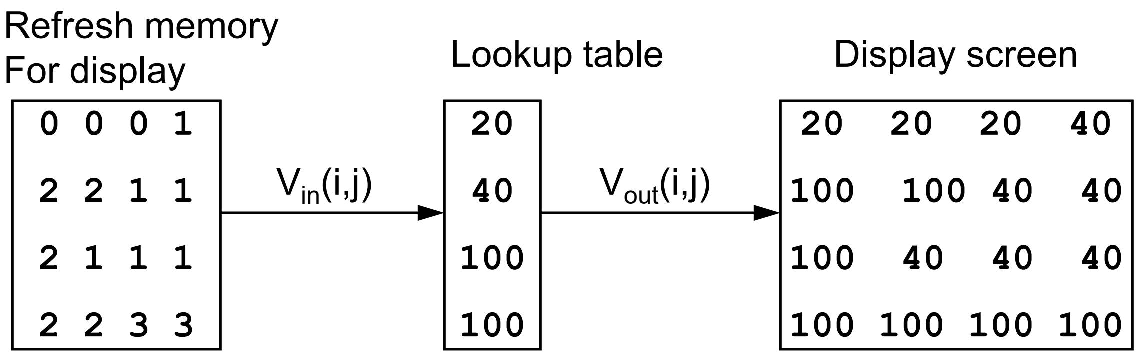
8-Bit Planes



<table>
<tr><td colspan="3">Bit-plane 7</td><td colspan="3">Bit-plane 6</td></tr>
<tr><td colspan="2">Bit-plane 5</td><td colspan="2">Bit-plane 4</td><td colspan="2">Bit-plane 3</td></tr>
<tr><td colspan="2">Bit-plane 2</td><td colspan="2">Bit-plane 1</td><td colspan="2">Bit-plane 0</td></tr>
</table>

Hardware LUTs

- All point operations can be implemented by LUTs.
- Hardware LUTs operate on the data as it is being displayed.
- It's an efficient means of applying transformations because changing display characteristics only requires loading a new table and not the entire image.
- For a 1024x1024 8-bit image, this translates to 256 entries instead of one million.
- LUTs do not alter the contents of original image (nondestructive).

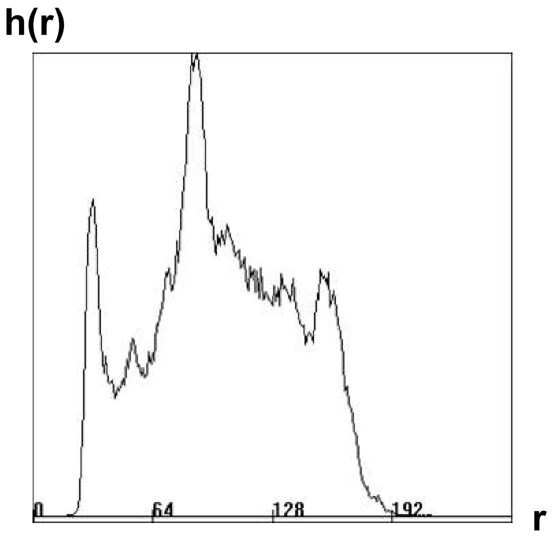


Graylevel Histograms

- **Histogram:** a simple but powerful technique for visualizing the statistical properties of an image.
  - The space in which the data resides is divided into bins, and the histogram records the number of occurrences in each bin.
    - **Graylevel histogram:** a histogram of image gray levels
    - **Normalized histogram:** computed from the histogram by simply dividing each value by the total number of pixels in the image
    - **Probability density function (PDF):** the normalized histogram

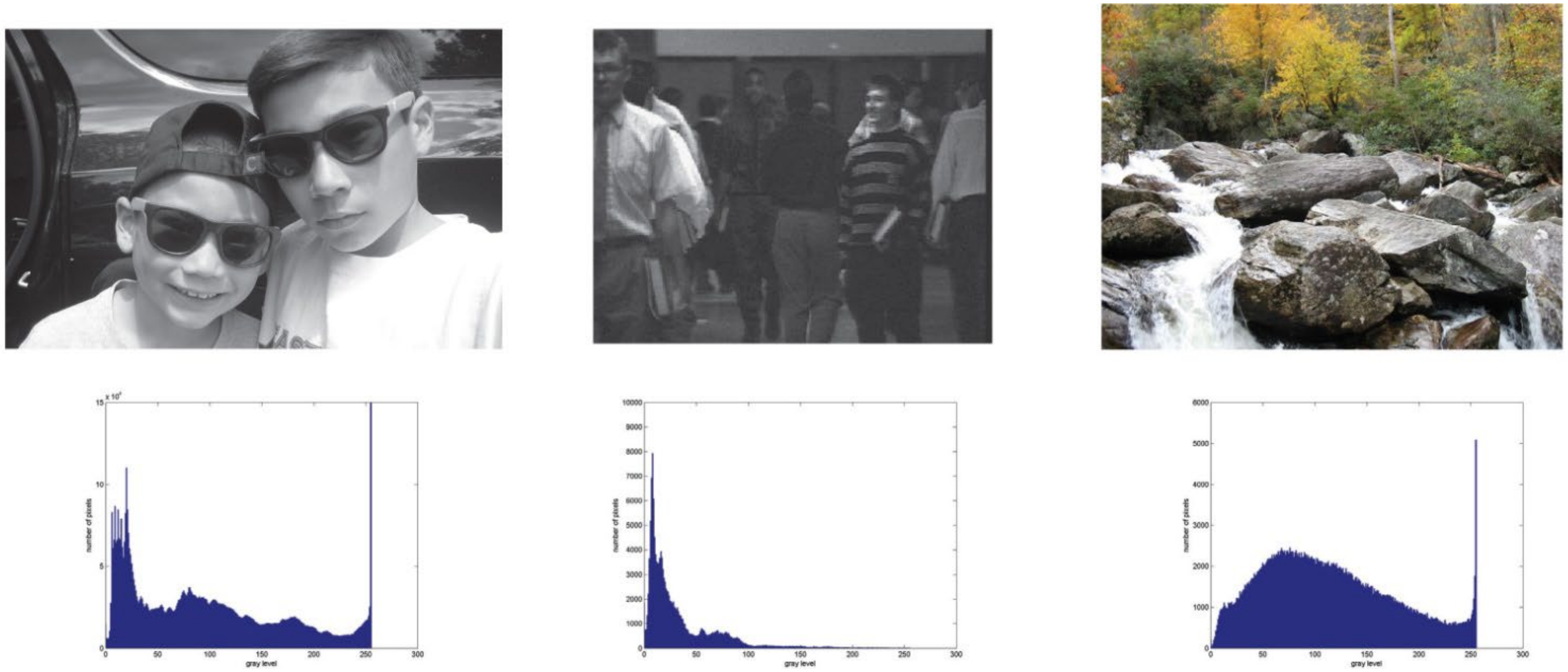
Graylevel Histogram

- A histogram of a digital image with gray levels in the range [0, L-1] is a discrete function $h(r_k) = n_k$
  - r_k : the k^{th} gray level
  - n_k : the number of pixels in the image having gray level r_k
- The sum of all histogram entries is equal to the total number of pixels in the image.



Histogram Examples

Figure 3.20 Images and their histograms. From left to right: an image with high contrast and many dark or bright pixels, a dark image with low contrast, and another high contrast image with good exposure. Note the spikes at 255 in the first and last images, indicating pixel saturation.



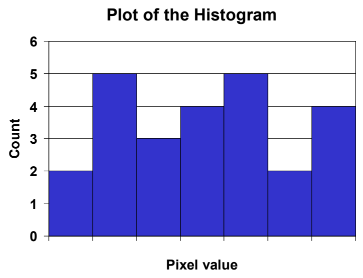
Stan Birchfield, Source: Movie *Hoop Dreams*, Jessica Birchfield

Histogram Evaluation

5x5 image

2	3	4	4	6
1	2	4	5	6
1	1	5	6	6
0	1	3	3	4
0	1	2	3	4

Graylevel	Count
0	2
1	5
2	3
3	4
4	5
5	2
6	4
Total	25



Histogram evaluation:

```
for(i=0; i<MXGRAY; i++) H[i] = 0;
for(i=0; i<total; i++) H[in[i]]++;
```

Normalized Histogram

- Divide each histogram entry at gray level r_k by the total number of pixels in the image, n

$$p(r_k) = n_k / n$$

- $p(r_k)$ gives an estimate of the probability of occurrence of gray level r_k
- The sum of all components of a normalized histogram is equal to 1.

Pseudocode

ALGORITHM 3.7 Compute the graylevel histogram of an image

ComputeHistogram(I)

Input: grayscale image I
Output: graylevel histogram h

for $\ell \leftarrow 0$ **to** 255 **do**
    $h[\ell] \leftarrow 0$
for $(x, y) \in I$ **do**
    $h[I(x, y)] \leftarrow_{+} 1$ ➤ $h[I(x,y)] \leftarrow h[I(x,y)] + 1$
return h

ALGORITHM 3.8 Compute the normalized graylevel histogram of an image

ComputeNormalizedHistogram(I)

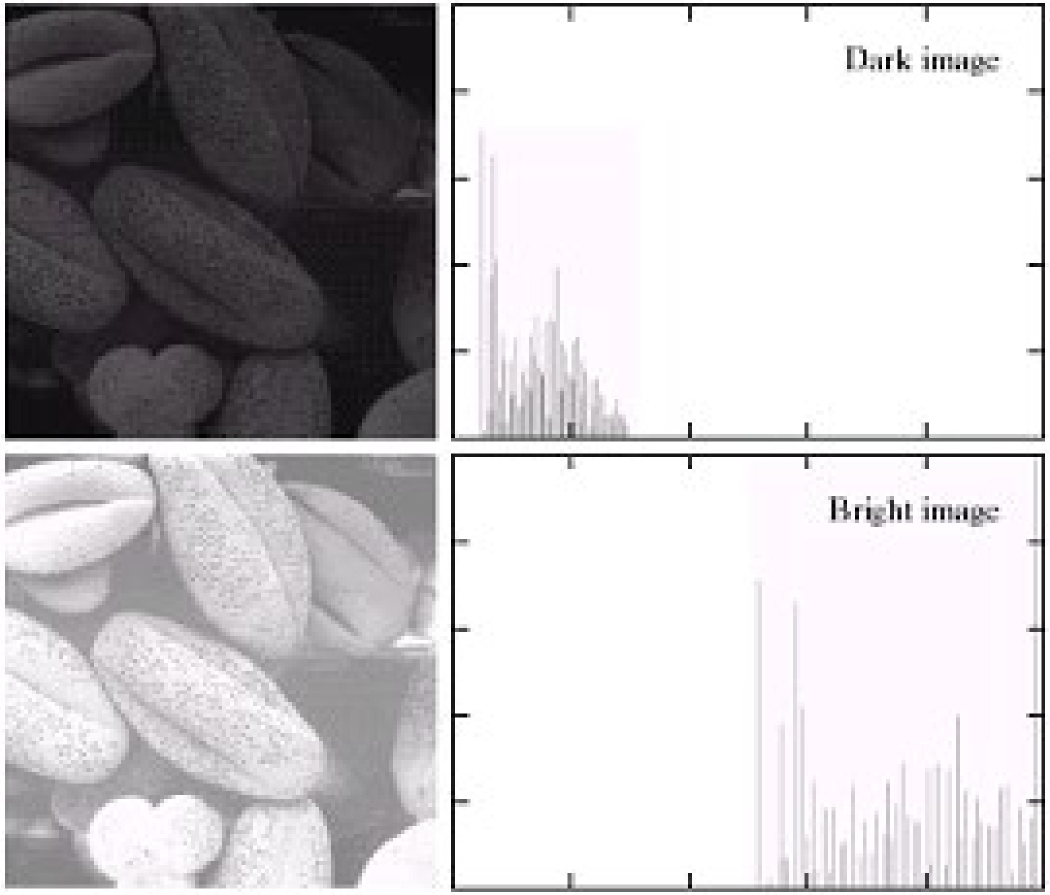
Input: grayscale image I
Output: normalized graylevel histogram $\bar{h}$

$h \leftarrow$ ComputeHistogram(I)
$n \leftarrow$ width * height
for $\ell \leftarrow 0$ **to** 255 **do**
    $\bar{h}[\ell] \leftarrow h[\ell]/n$
return $\bar{h}$

Histogram Processing

- Basic for numerous spatial domain processing techniques.
- Used effectively for image enhancement:
  - Histogram stretching
  - Histogram equalization
  - Histogram matching
- Information inherent in histograms also is useful in image compression and segmentation.

Example: Dark/Bright Images



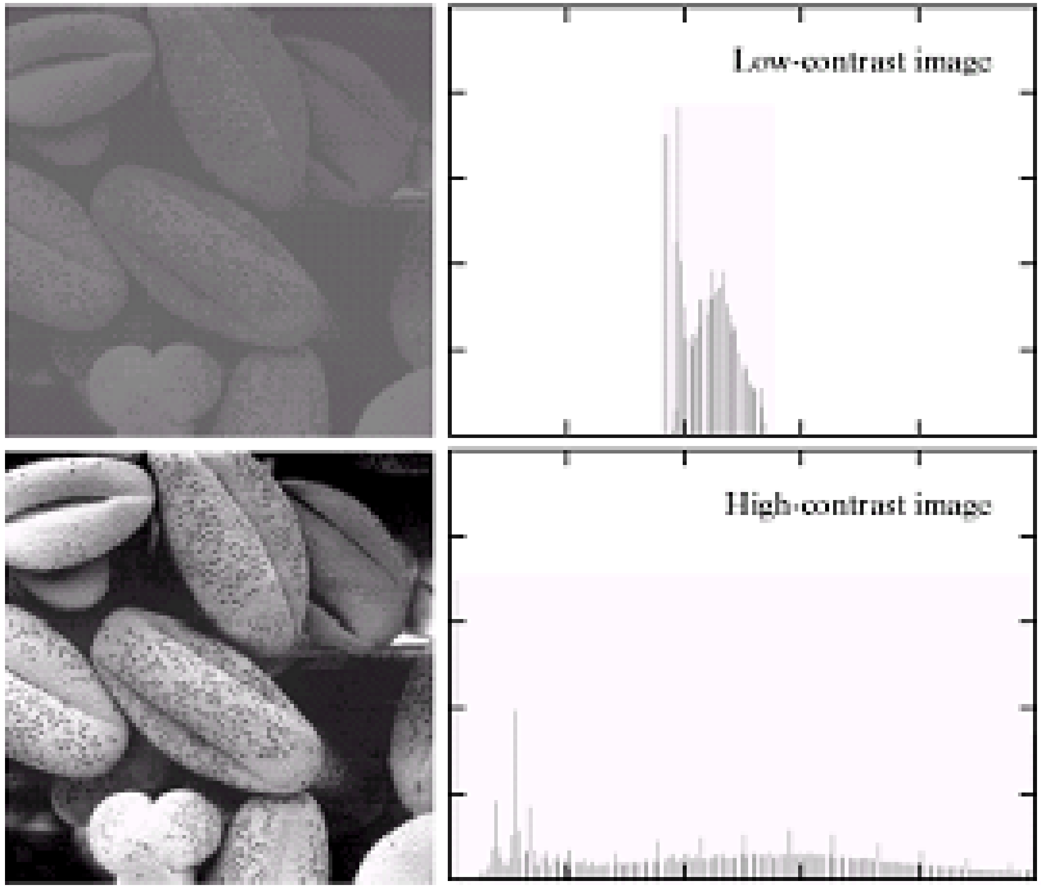
Dark image

Components of histogram are concentrated on the low side of the gray scale.

Bright image

Components of histogram are concentrated on the high side of the gray scale.

Example: Low/High Contrast Images



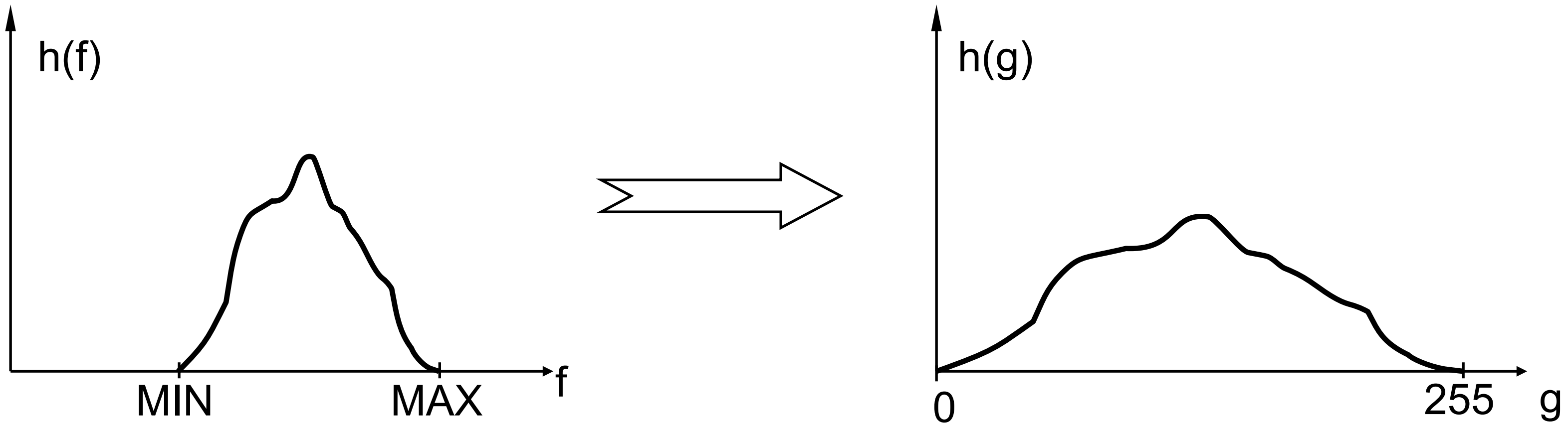
Low-contrast image

histogram is narrow and centered toward the middle of the gray scale

High-contrast image

histogram covers broad range of the gray scale and the distribution of pixels is not too far from uniform, with very few vertical lines being much higher than the others

Histogram Stretching (1)



3) Rescale to [0,255] range

1) Slide histogram down to 0

$$g = \frac{255(f - MIN)}{MAX - MIN}$$

2) Normalize histogram to [0,1] range

Histogram Stretching (2)

This is an instance of linear contrast stretching where the input range $[g_{min}, g_{max}]$ = [MIN,MAX] is mapped to output range $[g'_{min}, g'_{max}]$ = [0, 255].

$$I'(x,y) = \frac{g'_{\max} - g'_{\min}}{g_{\max} - g_{\min}}\left(I(x,y) - g_{\min}\right) + g'_{\min}$$

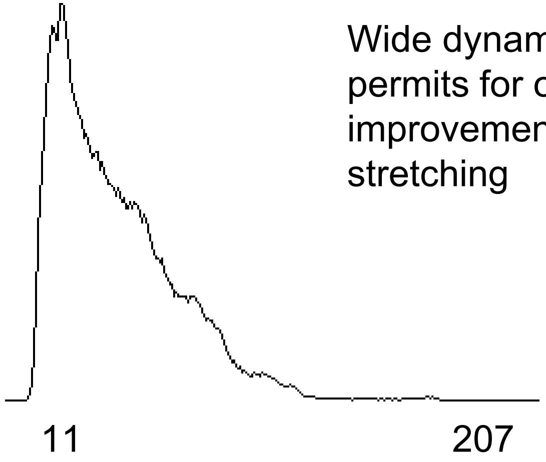
3) Rescale to [0,255] range

1) Slide histogram down to 0

$$g = \frac{255(f - MIN)}{MAX - MIN}$$

2) Normalize histogram to [0,1] range

Example (1)



Wide dynamic range permits for only a small improvement after histogram stretching

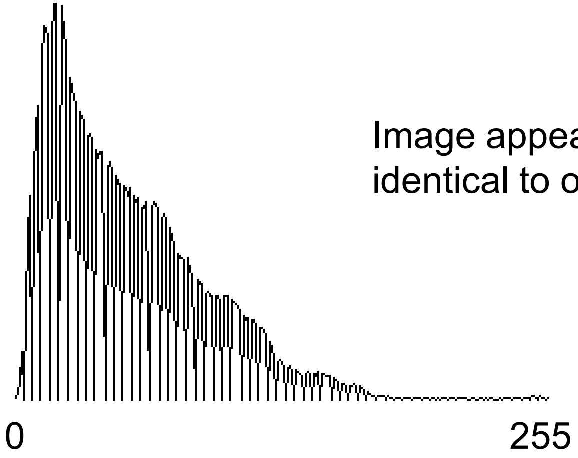
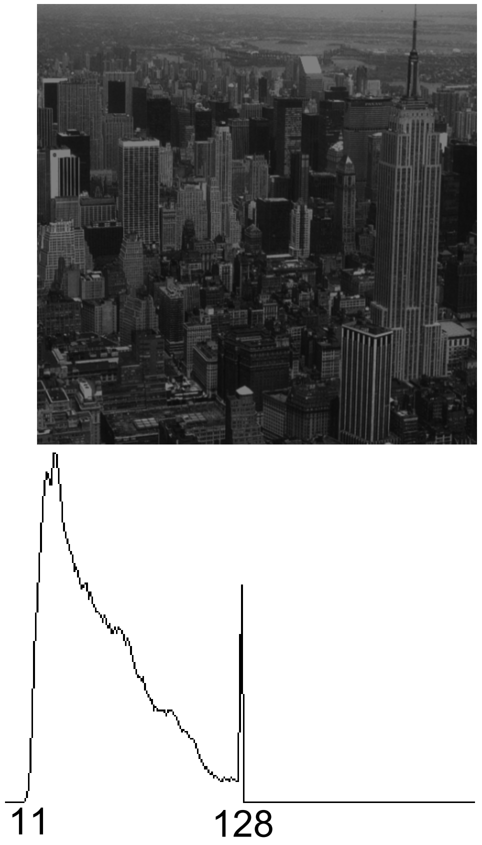
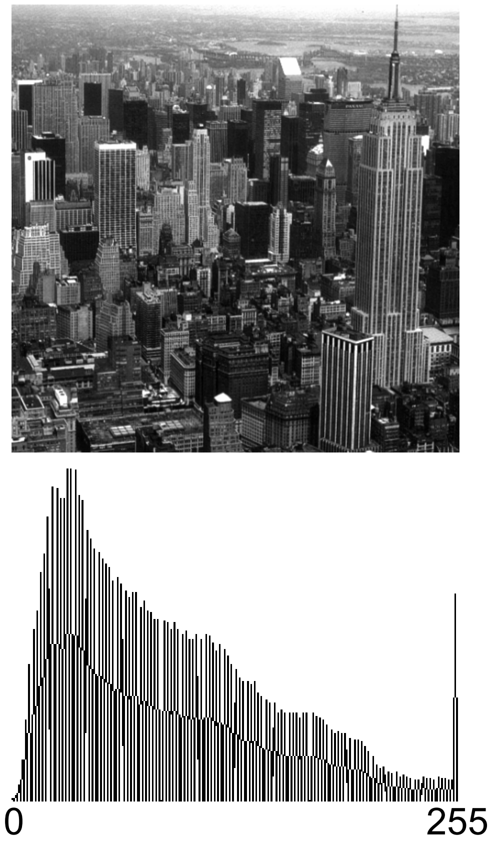


Image appears virtually identical to original

Example (2)

- Improve effectiveness of histogram stretching by clipping intensities first

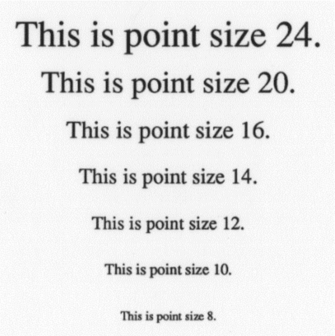


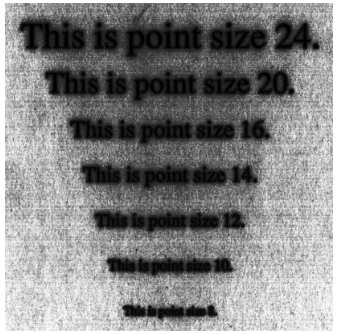


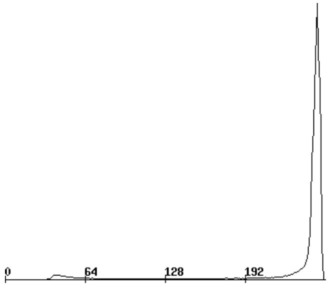
Flat histogram: every graylevel is equally present in image

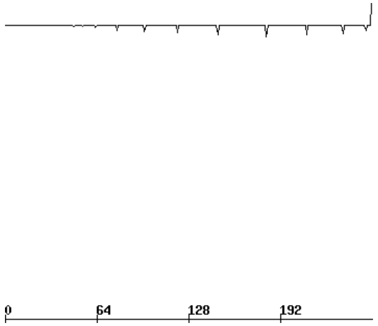
Histogram Equalization

- Produce image with flat histogram
- All graylevels are equally likely
- Appropriate for images with wide range of graylevels
- Inappropriate for images with few graylevels (see below)









Example (1)

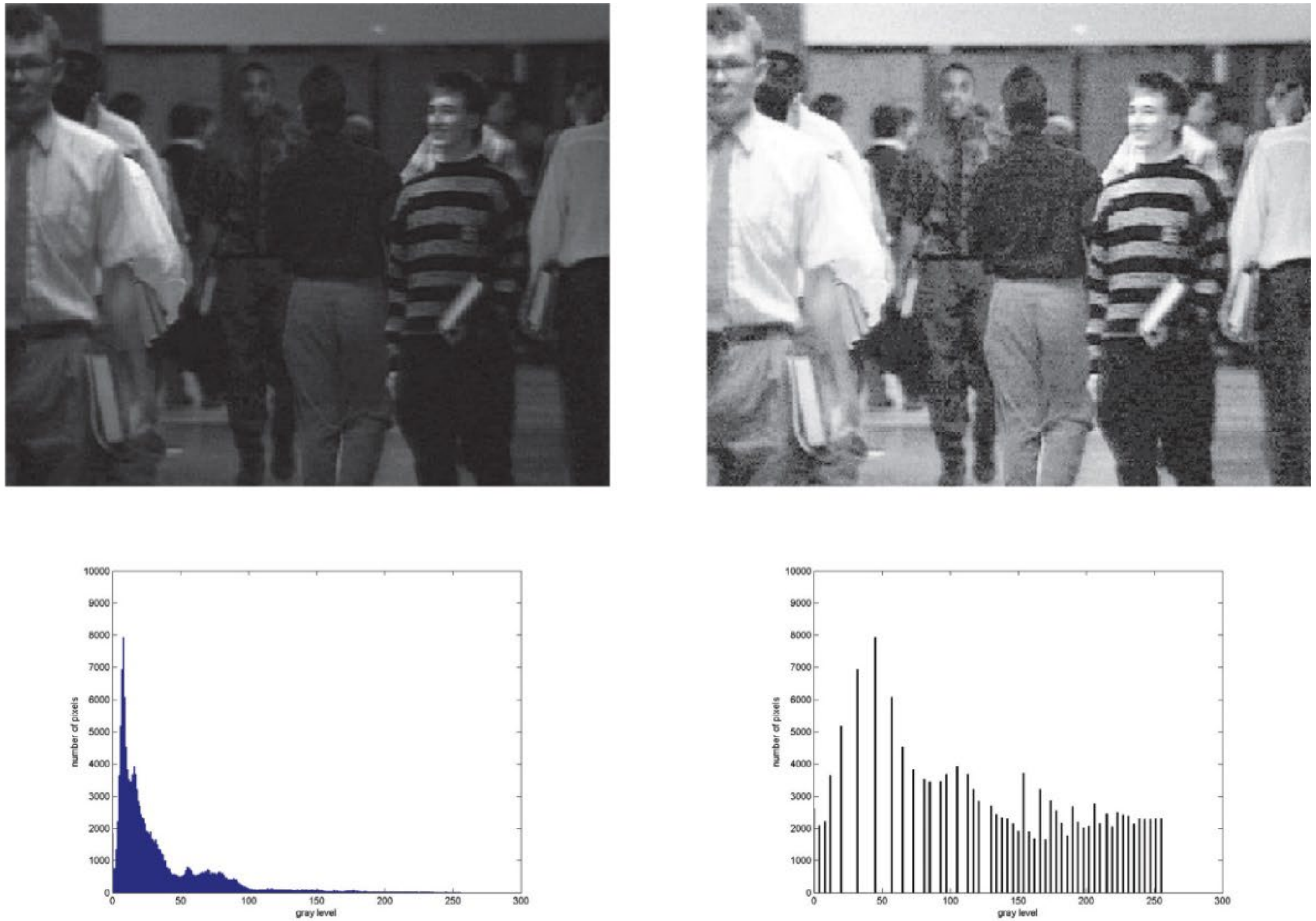
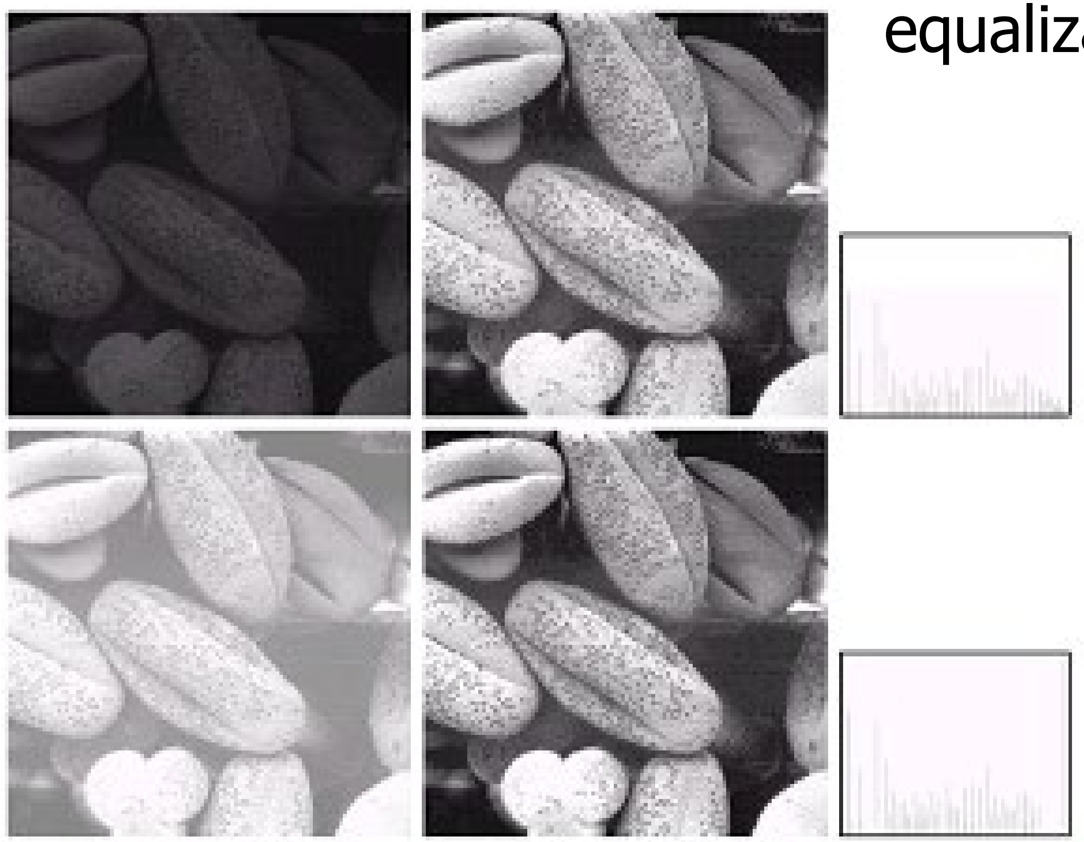


Figure 3.21 The result of histogram equalization applied to an image. The increase in contrast is noticeable. The normalized histogram of the result is much flatter than the original histogram, but it is not completely flat due to discretization effects. Source: Movie *Hoop Dreams*.

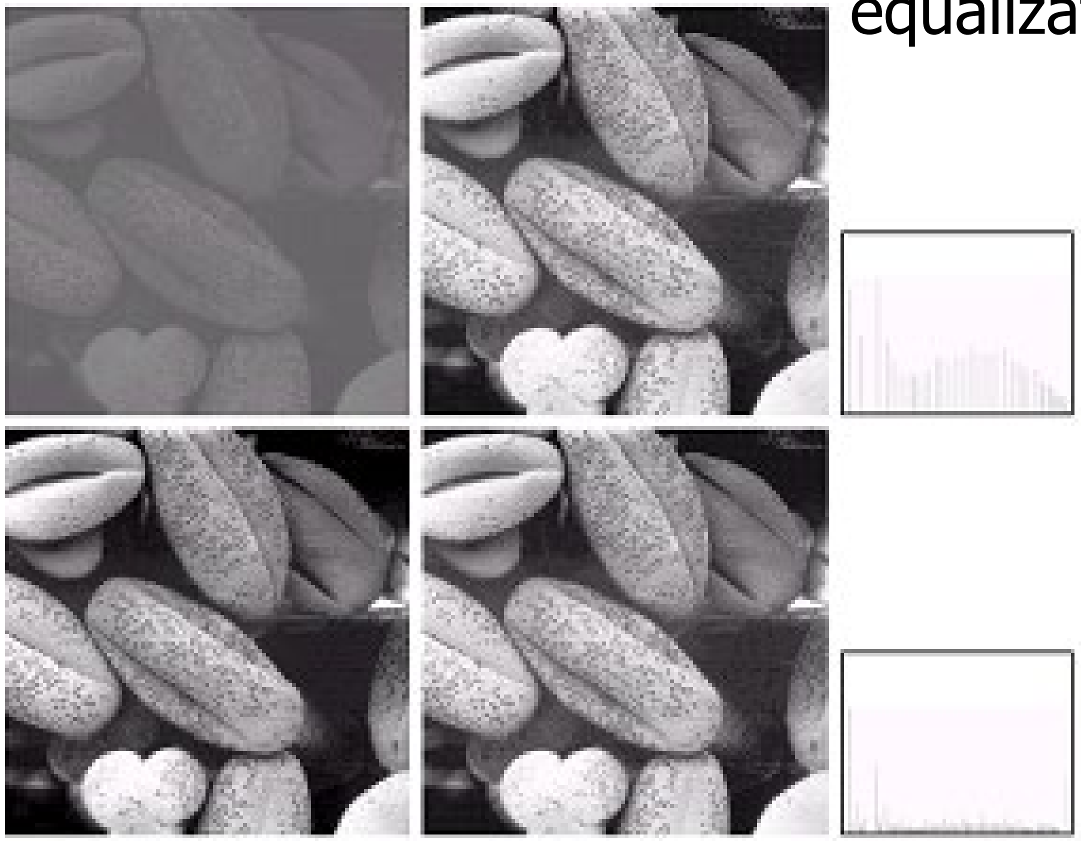
Example (2)

before after Histogram equalization



Example (3)

before after Histogram equalization



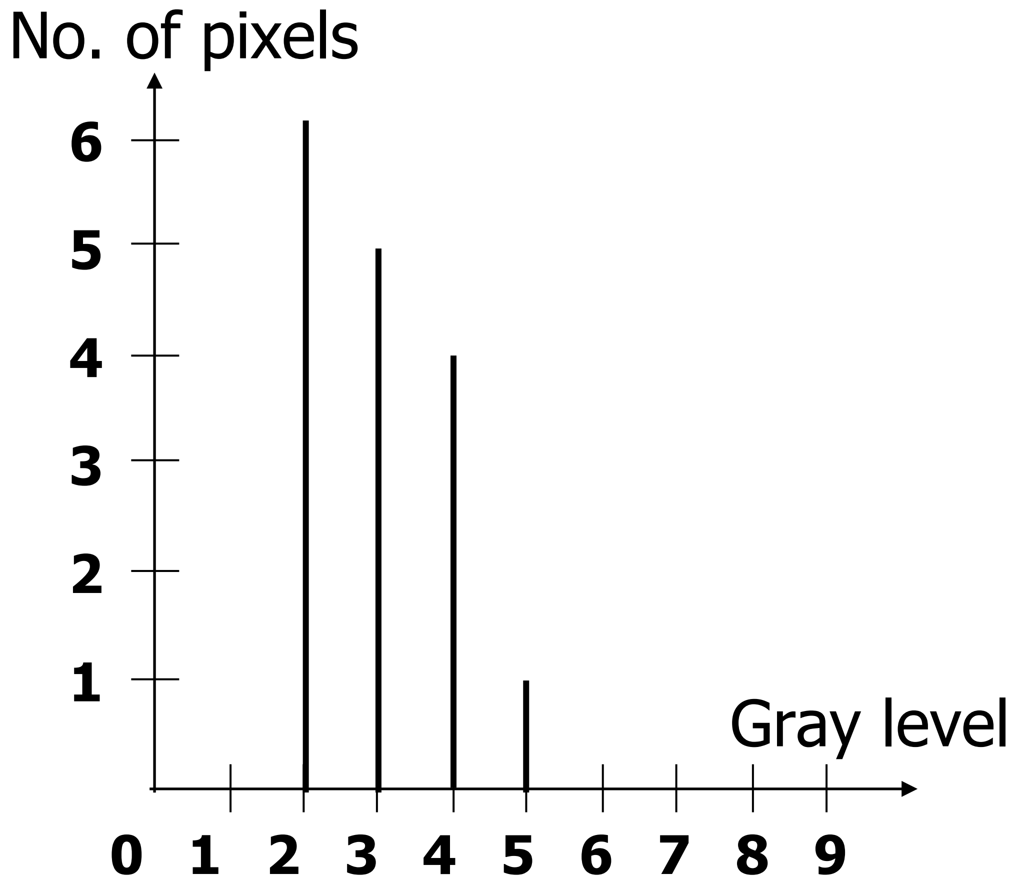
The quality is not improved much because the original image already has a wide graylevel scale

Implementation (1)

2	3	3	2
4	2	4	3
3	2	3	5
2	4	2	4

4x4 image

Gray scale = [0,9]



histogram

Implementation (2)

	Gray Level(j)	0	1	2	3	4	5	6	7	8	9
	No. of pixels	0	0	6	5	4	1	0	0	0	0
Cumulative histogram	$\sum_{j=0}^{k} n_j$	0	0	6	11	15	16	16	16	16	16
Cumulative Distribution Function (CDF)	$s = \sum_{j=0}^{k} \frac{n_j}{n}$	0	0	6/16	11/16	15/16	16/16	16/16	16/16	16/16	16/16
	s* x *9	0	0	3.3 ≈3	6.1 ≈6	8.4 ≈8	9	9	9	9	9

Implementation (3)

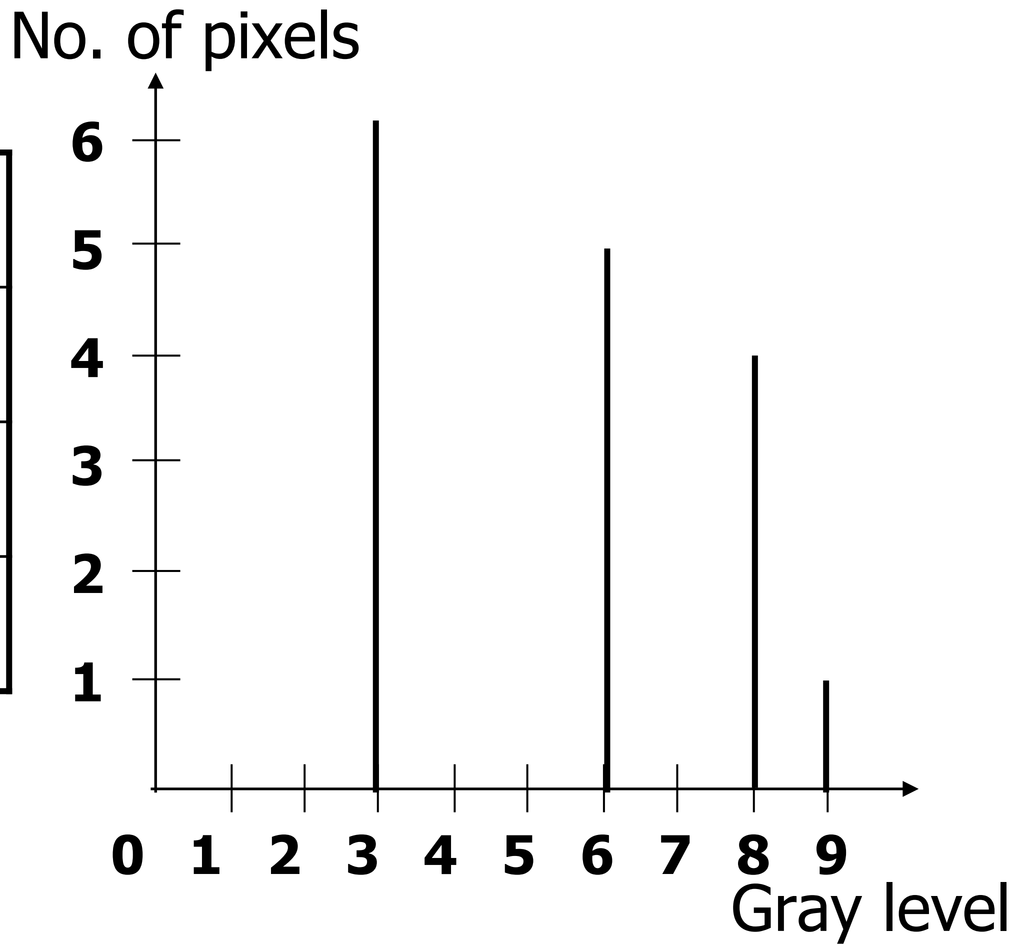
2	3	3	2
4	2	4	3
3	2	3	5
2	4	2	4

Input image

3	6	6	3
8	3	8	6
6	3	6	9
3	8	3	8

Output image

Gray scale = [0,9]



Histogram equalization

Pseudocode

ALGORITHM 3.9 Perform histogram equalization on an image

HISTOGRAMEQUALIZE(I)

Input: grayscale image I
Output: histogram-equalized grayscale image I' with increased contrast

```
h̄ ← ComputeNormalizedHistogram(I)
c̄ ← RunningSum(h̄)
for (x, y) ∈ I do
    I′(x, y) ← Round(255 * c̄[I(x, y)])
return I′
```

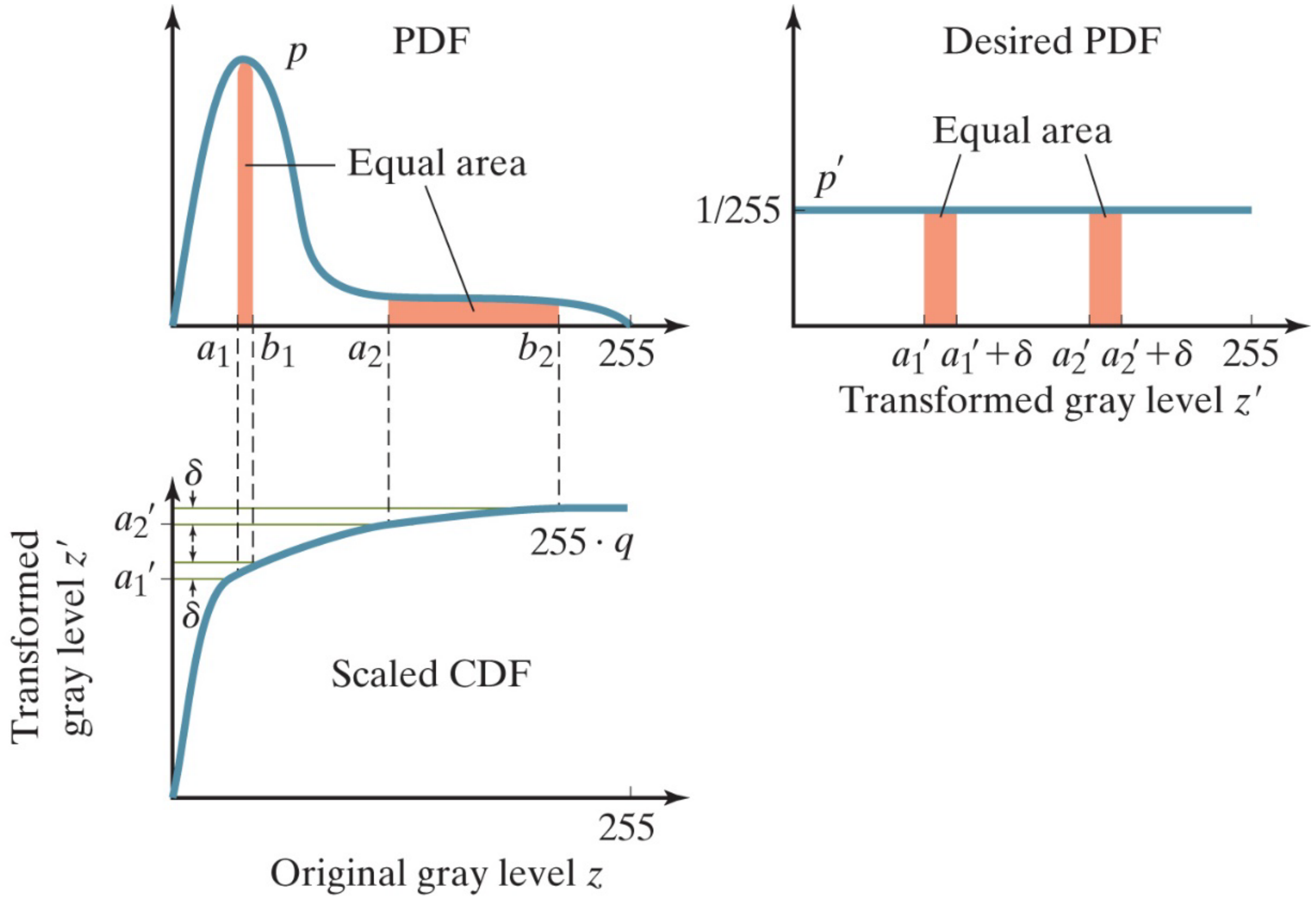
RUNNINGSUM(a)

Input: 1D array a of *length* values
Output: 1D running sum s of array

```
s[0] ← a[0]
for k ← 1 to length − 1 do
    s[k] ← s[k − 1] + a[k]
return s
```

Why It Works

Figure 3.22 Why histogram equalization works. In this example, the histogram of the original image is heavily weighted toward darker pixels. If we let the CDF be the mapping from the old gray level to the new one, the new PDF is flat and therefore weights all gray levels equally. This is because any interval of width δ in the new histogram captures the same number $(\delta/255)$ of pixels in the original image. In this example the area within each orange region is identical. Note that discretization effects have been ignored for this illustration.



Note (1)

- Histogram equalization distributes the graylevels to reach maximum gray (white) because the cumulative distribution function equals 1 when $0 \leq r \leq L\text{-}1$
- If $\sum_{j=0}^{k} n_j$ is slightly different among consecutive k , those graylevels will be mapped to (nearly) identical values as we have to produce an integer grayvalue as output
- Thus, the discrete transformation function cannot guarantee a one-to-one mapping

Note (2)

- The implementation given above is widely interpreted as histogram equalization.
- It is readily implemented with a LUT.
- It does not produce a strictly flat histogram.
- There is a more accurate solution. However, it may require a one-to-many mapping that cannot be implemented with a LUT.

Histogram Equalization Objective

Objective: we want a uniform histogram.

Rationale: maximize image entropy.

$$h_1(v_{out}) = \text{constant} = \frac{total}{MXGRAY}$$

$$= h_{avg}$$

$$c_1(v_{out}) = (v_{out} + 1) * h_{avg}$$

This is a special case of histogram matching.

Perfectly flat histogram: *H[i] = total/MXGRAY* for $0 \leq i < MXGRAY$.

If $H[v] = k * h_{avg}$ then v must be mapped onto k different levels, from v_1 to v_k. This is a one-to many mapping.

Histogram Equalization Mappings

Rule 1: Always map v onto $(v_1+v_k)/2$. (This does not result in a flat histogram, but one where brightness levels are spaced apart).

Rule 2: Assign at random one of the levels in $[v_1,v_k]$. This can result in a loss of contrast if the original histogram had two distinct peaks that were far apart (i.e., an image of text).

Rule 3: Examine neighborhood of pixel, and assign it a level from $[v_1,v_k]$ which is closest to neighborhood average. This can result in bluriness; more complex.

Rule (1) creates a lookup table beforehand.

Rules (2) and (3) are runtime operations.

Rule 2: Implementation (1)

```
void histoEqRand(imagePtr I1, imagePtr I2) {
        // copy image header of input image I1 to output image I2
        IP_copyImageHeader(I1, I2);

        // total number of pixels
        int total = I1->width() * I1->height();

        // scale dynamic range of I1 to full intensity range [0,MaxGray], where MaxGray=255
        IP_embedRange(I1, 0., (double) MaxGray, I2);

        // declarations for input and output image channel pointers
        ChannelPtr<uchar> p1, p2;
        int type;       // datatype of returned channel pointer
        int histo[MaxGray+1];

        // Note: IP_getChannel(I, ch, p1, type) gets pointer p1 of channel ch in image I.
        // The pixel datatype (e.g., uchar, short, …) of that channel is returned in type.
        // It is ignored here since we assume that our input images consist exclusively of uchars.
        // IP_getChannel() returns 1 when channel ch exists; 0 otherwise.

        // visit all channels and evaluate output image
        for(int ch=0; IP_getChannel(I1, ch, p1, type); ch++) { // get input pointer
                IP_getChannel(I2, ch, p2, type);        // get output pointer

                // compute histogram
                for(int i=0; i<=MaxGray; i++) histo[i] = 0;     // clear histogram
                for(int i=0; i<total;  i++)     histo[p1[i]]++; // eval  histogram
```

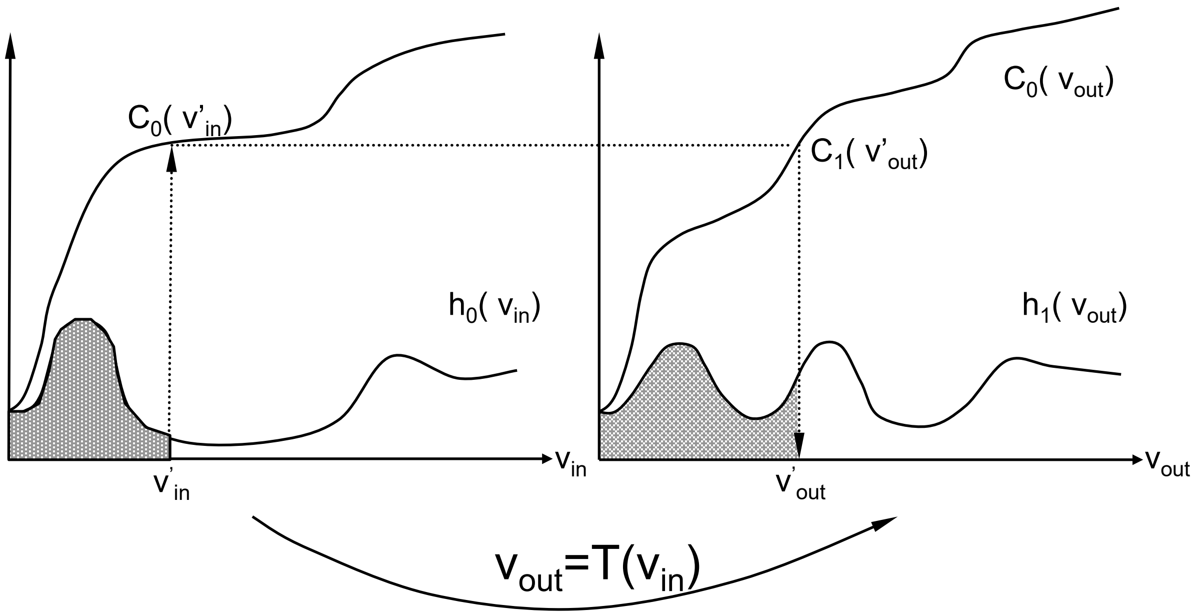
Rule 2: Implementation (2)

```
int right=0;                                    // right end of interval
int left [MaxGray+1], width[MaxGray+1]; // left end and width of target interval for each input graylevel
long Hsum = 0;                                  // cumulative histogram summation
long Havg = total / (MaxGray+1);                // value for each entry of target histogram

// evaluate remapping of all input gray levels: each input gray value maps to an interval of valid output values.
// The endpoints of the intervals are left[] and left[]+width[].

for(int i=0; i<=MaxGray; i++) {
        left[i] = right;                        // left end of grayscale interval into which i maps
        Hsum += histo[i];                       // compute cumulative histogram summation
        while(Hsum>Havg && R<MaxGray) {         // make interval wider
                Hsum -= Havg;                   // adjust Hsum
                right++;                        // update right end of interval
        }
        width[i] = right - left[i] + 1;         // width of interval
}

// visit all input pixels and remap the intensities
for(int i=0; i<total; i++) {
        if(width[p1[i]] == 1) p2[i] = left[p1[i]];
        else {          // p1[i] spills over into width[] possible values
                        // randomly pick from 0 to width[i]
                        int r = ((double) rand() / RAND_MAX) * width[p1[i]];   // 0 <= r < width[i]
        }               p2[i] = left[p1[i]] + r;
}
        }
}
```

Note

- Histogram equalization has a disadvantage:
  it can generate only one type of output image.
- With histogram specification we can specify the shape of the histogram that we wish the output image to have.
- It doesn't have to be a uniform histogram.
- Histogram specification is a trial-and-error process.
- There are no rules for specifying histograms, and one must resort to analysis on a case-by-case basis for any given enhancement task.

Histogram Matching



In the figure above, *h()* refers to the histogram, and *c()* refers to its cumulative histogram. Function *c() is a monotonically increasing function defined as:*

$$c(v) = \int_0^v h(u)du$$

Histogram Matching Rule

Let $v_{out} = T(v_{in})$ If T() is a unique, monotonic function then

$$\int_0^{v_{out}} h_1(u)du = \int_0^{v_{in}} h_0(u)du$$

This can be restated in terms of the histogram matching rule:

$$c_1(v_{out}) = c_0(v_{in})$$

Where $c_1(v_{out})$ = # pixels $\leq v_{out}$, and $c_0(v_{in})$ = # pixels $\leq v_{in}$..

This requires that $$v_{out} = c_1^{-1}(c_0(v_{in}))$$

which is the basic equation for histogram matching techniques.

Pseudocode

ALGORITHM 3.10 Perform histogram matching on an image

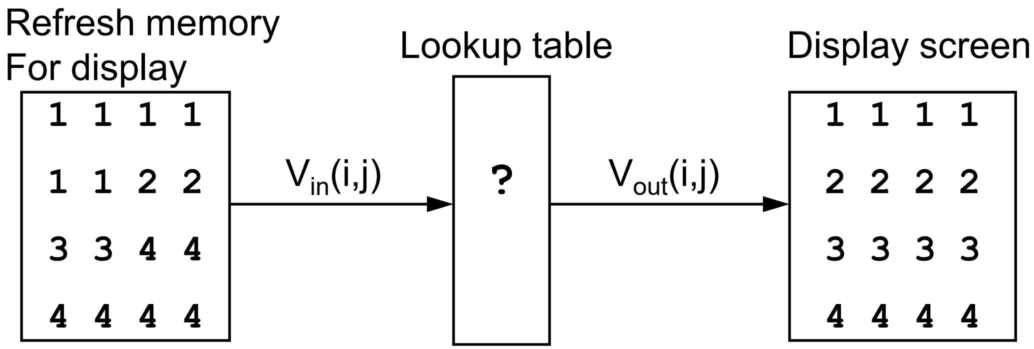
HISTOGRAMMATCH(I, h_{ref})

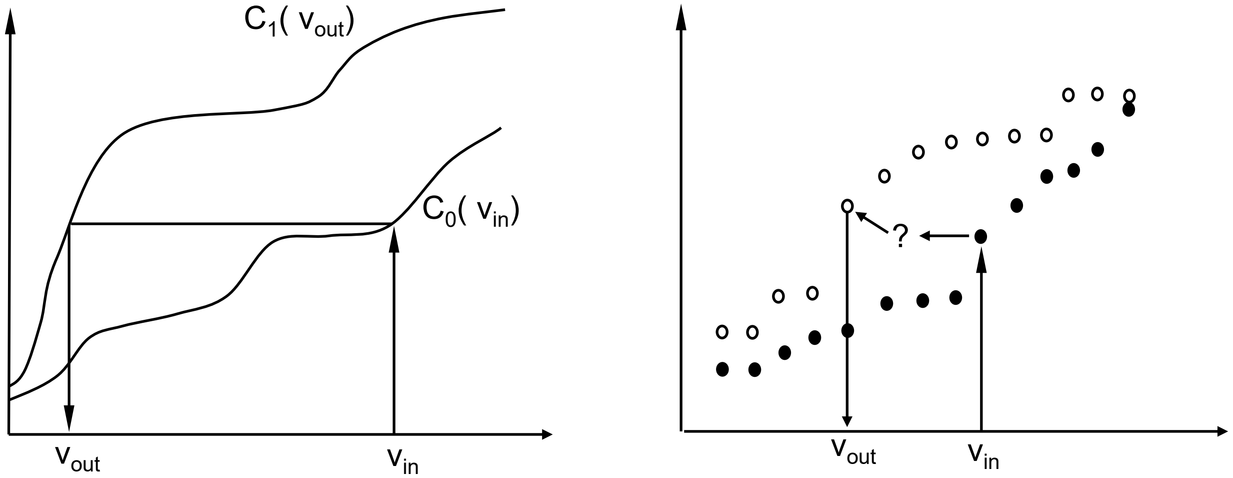
Input: grayscale image I, reference normalized graylevel histogram $\bar{h}_{ref}$
Output: grayscale image I' whose normalized graylevel histogram closely matches $\bar{h}_{ref}$

$\bar{h} \leftarrow$ COMPUTENORMALIZEDHISTOGRAM(I) ➤ Compute the normalized histogram of the image,
$\bar{c} \leftarrow$ RUNNINGSUM$(\bar{h})$ then compute the CDF of the image,
$\bar{c}_{ref} \leftarrow$ RUNNINGSUM$(\bar{h}_{ref})$ as well as the desired CDF.
for $\ell \leftarrow 0$ **to** 255 **do** ➤ For each possible gray level ℓ, set $f[\ell] \approx \bar{c}_{ref}^{-1}[\bar{c}(\ell)]$.
  $\ell' \leftarrow 255$ This is done by finding ℓ' such that $\bar{c}_{ref}[\ell'] \approx \bar{c}[\ell]$,
  repeat and setting $f[\ell] = \ell'$. To handle discretization effects,
    $f[\ell] \leftarrow \ell'$ ℓ' is set to the maximum possible gray level,
    $\ell' \leftarrow_{-} 1$ then repeatedly decremented
  while $\ell' \geq 0$ AND $\bar{c}_{ref}[\ell'] > \bar{c}[\ell]$ until $\bar{c}_{ref}[\ell'] \leq \bar{c}[\ell]$.
for $(x, y) \in I$ **do** ➤ Once the mapping f has been determined,
  $I'(x, y) \leftarrow f[I(x, y)]$ it is applied to all pixels in the image.
return I'

Histograms are Discrete

- Impossible to match all histogram pairs because they are discrete.





Continuous case **Discrete case**

Problems with Discrete Case

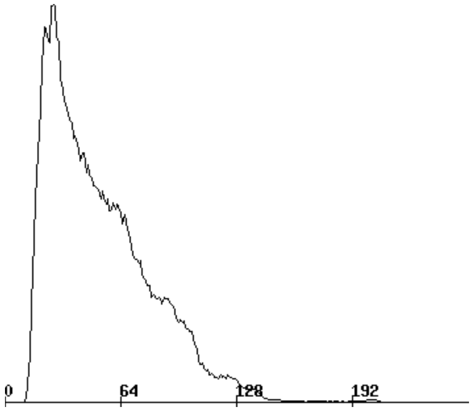
- The set of input pixel values is a discrete set, and all the pixels of a given value are mapped to the same output value. For example, all six pixels of value one are mapped to the same value so it is impossible to have only four corresponding output pixels.

- No inverse for c_1 in $v_{out}= c_1^{-1}(c_0(v_{in}))$ because of discrete domain. Solution: choose v_{out} for which $c_1(v_{out})$ is closest to $c_0(v_{in})$.

- $v_{in} \rightarrow v_{out}$ such that $|c_1(v_{out}) - c_0(v_{in})|$ is a minimum

Histogram Matching Example (1)

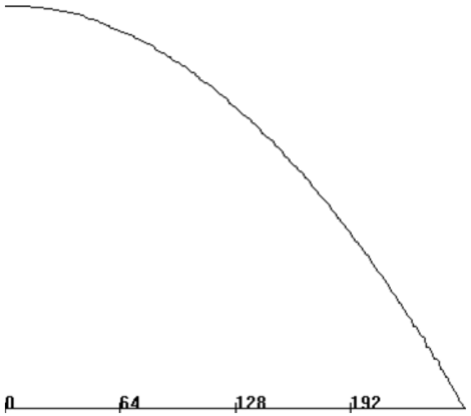
Input image



Input Histogram

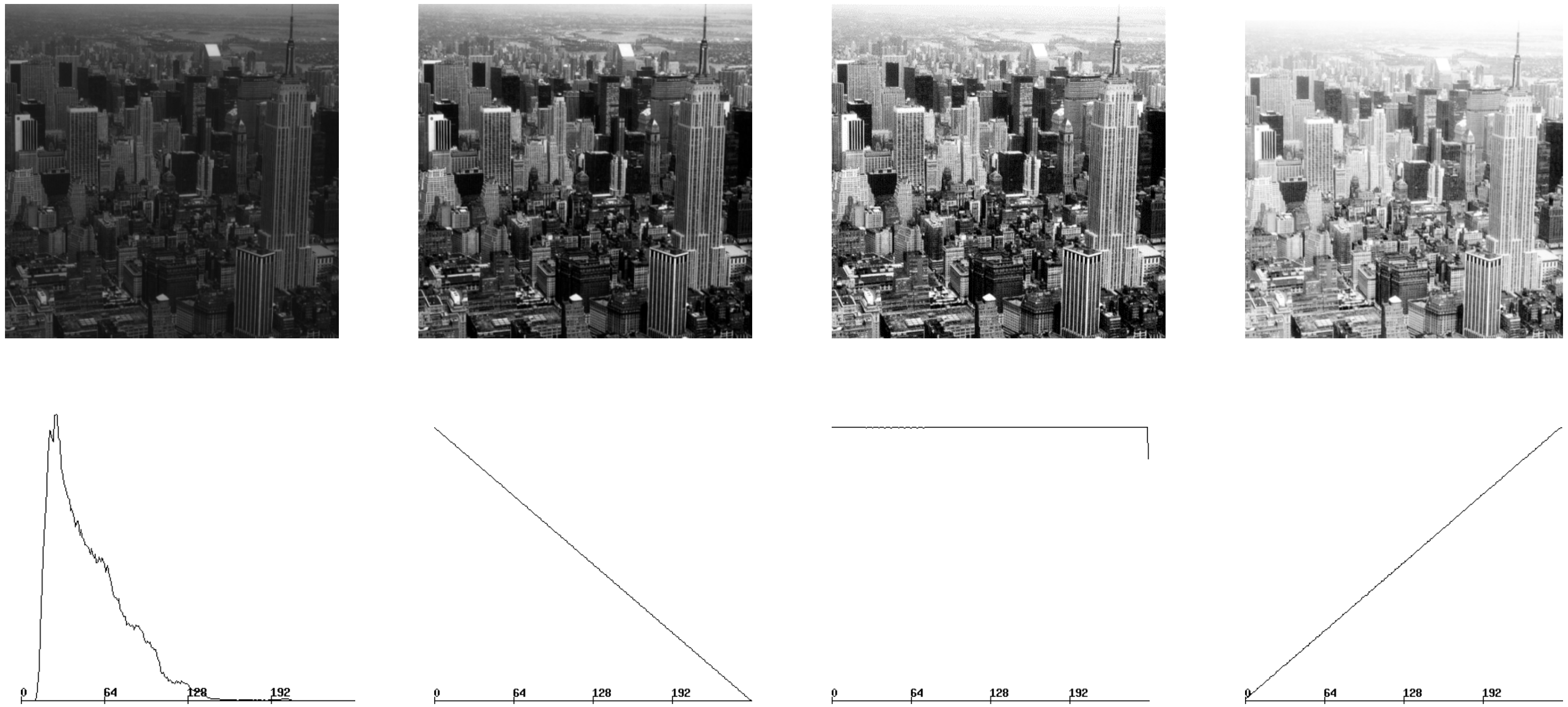
Histogram match

Target Histogram



Output image

Histogram Matching Example (2)



Implementation (1)

```
void histoMatch(imagePtr I1, ImagePtr targetHisto, imagePtr I2)
{
        // copy image header of input image I1 to output image I2
        IP_copyImageHeader(I1, I2);

        // total number of pixels
        int total1 = I1->width() * I1->height();

        // scale dynamic range of I1 to full intensity range [0,MaxGray], where MaxGray=255
        IP_embedRange(I1, 0., (double) MaxGray, I2);

        // declarations for input and output image channel pointers
        ChannelPtr<uchar> p1, p2, lutp;
        int type;       // datatype of returned channel pointer
        int histo1[MaxGray+1], histo2[MaxGray+1];

        // Note: IP_getChannel(I, ch, p1, type) gets pointer p1 of channel ch in image I.
        // The pixel datatype (e.g., uchar, short, …) of that channel is returned in type.
        // It is ignored here since we assume that our input images consist exclusively of uchars.
        // IP_getChannel() returns 1 when channel ch exists; 0 otherwise.

        // visit all channels and evaluate output image
        for(int ch=0; IP_getChannel(I1, ch, p1, type); ch++) { // get input pointer
                IP_getChannel(I2, ch, p2, type);          // get output pointer

                // compute histogram
                for(int i=0; i<=MaxGray; i++) histo1[i] = 0;    // clear histogram
                for(int i=0; i<total;  i++)       histo1[p1[i]]++; // eval  histogram
```

Implementation (2)

```
IP_getChannel(targetHisto, 0, lutp, type);
histo2 = (int *) &lutp[0];

// compute sum of target histogram for normalization
int total2 = 0;
for(int i=0; i<=MaxGray; i++)
		total2 += histo2[i];

// scale histo2 to conform with dimensions of I1
double scale = (double) total1 / total2;
if(scale != 1) {
    int sum = 0;
    for(int i=0; i<=MaxGray; i++) {
		histo2[i] = ROUND(histo2[i] * scale);

		// update histo2[] if cumulative histogram overshoots due to rounding operations
		sum += histo2[i];
		if(sum > total1) {				// check for overshoot
			histo2[i] -= (sum – total1);		// clamp last non-zero histo2[]
			for(; i <=MaxGray; i++) histo2[i] = 0;	// clear the remainder of histo2[]
		}
    }
}
```

Implementation (3)

```
int r=0;                                   // right end of interval
int left [MaxGray+1];                      // left end and width of target interval for each input graylevel
long Hsum = 0;                             // cumulative histogram summation

for(int i=0; I <=  MaxGray; i++) {
        left[i] = r;                       // left end of grayscale interval into which i maps
        Hsum += histo1[i];                 // compute cumulative histogram summation
        while(Hsum>histo2[r] && r<MaxGray) {          // compute width of interval
                Hsum -= histo2[r];                    // adjust Hsum as interval widens
                r++;                                  // update
        }
        right[i] = r;                                 // save right end of interval
}

// clear histo1[] and reuse it below
for(int i=0; i <= MaxGray; i++) histo1[i] = 0;

// visit all input pixels and remap the intensities
int p;
for(int i=0; i<total; i++) {
        p = left[p1[i]];
        if(histo1[p] < histo2[p])    // mapping satisfies target histogram (histo2)
                p2[i] = p;
        else    p2[i] = p = left[p1[i]] = MIN(p+1, right[p1[i]]);
        histo1[p]++;
}
}
}
```

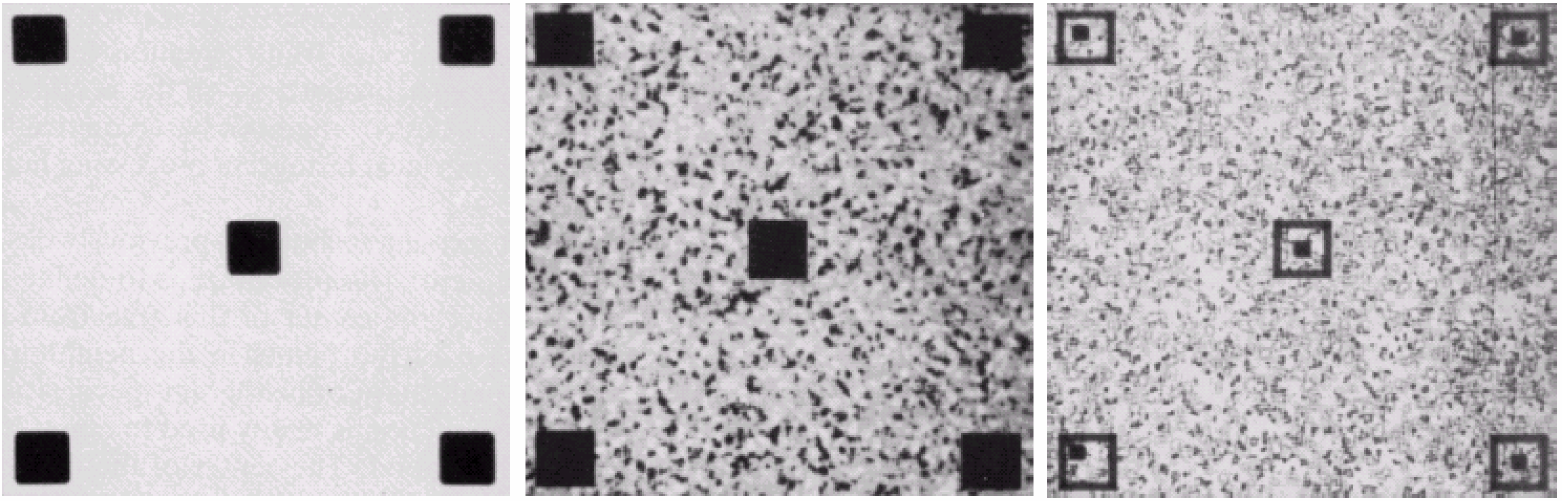
Local Pixel Value Mappings

- Histogram processing methods are global, in the sense that pixels are modified by a transformation function based on the graylevel content of an entire image.
- We sometimes need to enhance details over small areas in an image, which is called a local enhancement.
- Solution: apply transformation functions based on graylevel distribution within pixel neighborhood.

General Procedure

- Define a square or rectangular neighborhood.
- Move the center of this area from pixel to pixel.
- At each location, the histogram of the points in the neighborhood is computed and histogram equalization, histogram matching, or other graylevel mapping is performed.
- Exploit easy histogram update since only one new row or column of neighborhood changes during pixel-to-pixel translation.
- Another approach used to reduce computation is to utilize nonoverlapping regions, but this usually produces an undesirable checkerboard effect.

Example: Local Enhancement



a b c

FIGURE 3.23 (a) Original image. (b) Result of global histogram equalization. (c) Result of local histogram equalization using a 7 × 7 neighborhood about each pixel.

a) Original image (slightly blurred to reduce noise)
b) global histogram equalization enhances noise & slightly increases contrast but the structural details are unchanged
c) local histogram equalization using 7x7 neighborhood reveals the small squares inside of the larger ones in the original image.

Definitions (1)

$$\mu(x,y) = \frac{1}{n}\sum_{i,j} f(i,j)$$ **mean**

$$\sigma(x,y) = \sqrt{\frac{1}{n}\sum_{i,j}(f(i,j)-\mu(x,y))^2}$$ **standard deviation**

- Let $p(r_i)$ denote the normalized histogram entry for grayvalue r_i for $0 \le i < L$ where L is the number of graylevels.
- It is an estimate of the probability of occurrence of graylevel r_i.
- Mean m can be rewritten as

$$m = \sum_{i=0}^{L-1} r_i \, p(r_i)$$

Definitions (2)

- The nth moment of *r* about its mean is defined as

$$\mu_n(r) = \sum_{i=0}^{L-1} (r_i - m)^n p(r_i)$$

- It follows that:

$\mu_0(r) = 1$	**0th moment**
$\mu_1(r) = 0$	**1st moment**
$\mu_2(r) = \sum_{i=0}^{L-1} (r_i - m)^2 p(r_i)$	**2nd moment**

- The second moment is known as variance $\sigma^2(r)$
- The standard deviation is the square root of the variance.
- The mean and standard deviation are measures of average grayvalue and average contrast, respectively.

Example: Statistical Differencing

- Produces the same contrast throughout the image.
- Stretch $f(x, y)$ away from or towards the local mean to achieve a balanced local standard deviation throughout the image.
- σ_0 is the desired standard deviation and it controls the amount of stretch.
- The local mean can also be adjusted:

$$g(x,y) = \alpha m_0 + (1-\alpha)\mu(x,y) + (f(x,y) - \mu(x,y))\frac{\sigma_0}{\sigma(x,y)}$$

- m_0 is the mean to force locally and α controls the degree to which it is forced.
- To avoid problems when $\sigma(x, y) = 0$,

$$g(x,y) = \alpha m_0 + (1-\alpha)\mu(x,y) + (f(x,y) - \mu(x,y))\frac{\beta\sigma_0}{\sigma_0 + \beta\sigma(x,y)}$$

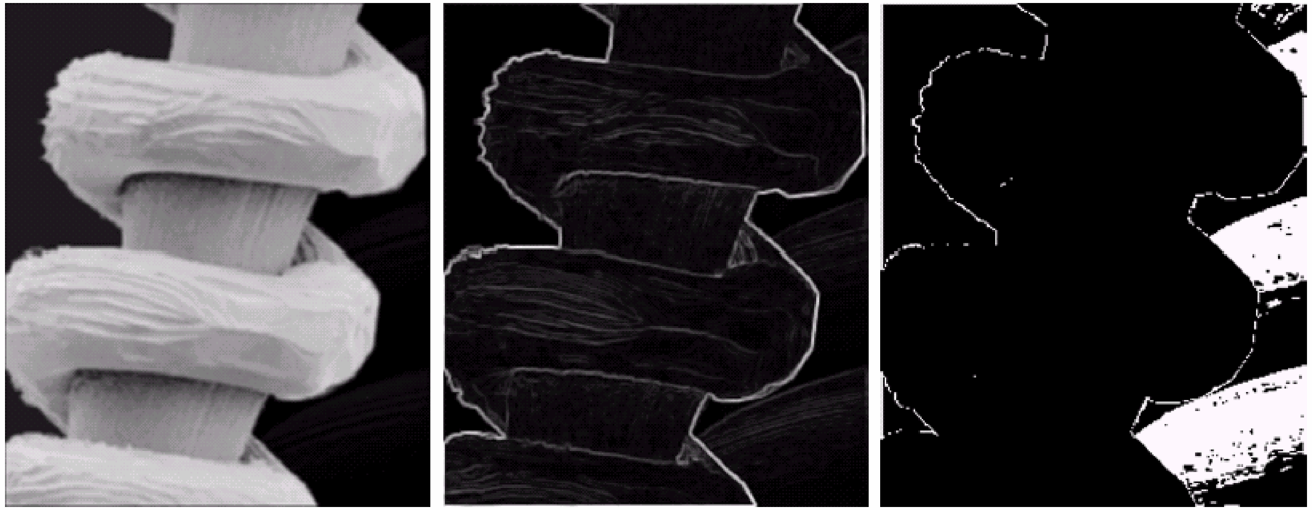
- Speedups can be achieved by dividing the image into blocks (tiles), exactly computing the mean and standard deviation at the center of each block, and then linearly interpolating between blocks in order to compute an approximation at any arbitrary position. In addition, the mean and standard deviation can be computed incrementally.

Example: Local Statistics (1)

FIGURE 3.24 SEM image of a tungsten filament and support, magnified approximately 130×. (Original image courtesy of Mr. Michael Shaffer, Department of Geological Sciences, University of Oregon, Eugene).

The filament in the center is clear.
There is another filament on the right side that is darker and hard to see.
Goal: enhance dark areas while leaving the light areas unchanged.

Example: Local Statistics (2)



a b c

FIGURE 3.25 (a) Image formed from all local means obtained from Fig. 3.24 using Eq. (3.3-21). (b) Image formed from all local standard deviations obtained from Fig. 3.24 using Eq. (3.3-22). (c) Image formed from all multiplication constants used to produce the enhanced image shown in Fig. 3.26.

Solution: Identify candidate pixels to be dark pixels with low contrast.
Dark: local mean < k_0*global mean, where $0 < k_0 < 1$.
Low contrast: k_1*global variance < local variance < k_2 * global variance, where $k_1 < k_2$.
Multiply identified pixels by constant E>1. Leave other pixels alone.

Example: Local Statistics (3)

FIGURE 3.26 Enhanced SEM image. Compare with Fig. 3.24. Note in particular the enhanced area on the right side of the image.

Results for E=4, k_0=0.4, k_1=0.02, k_2=0.4. 3x3 neighborhoods used.

Enhancement

- Point operations are used to enhance an image.
- Processed image should be more suitable than the original image for a specific application.
- Suitability is application-dependent.
- A method which is quite useful for enhancing one image may not necessarily be the best approach for enhancing another image.
- Very subjective

Two Enhancement Domains

- Spatial Domain: (image plane)
  - Techniques are based on direct manipulation of pixels in an image
- Frequency Domain:
  - Techniques are based on modifying the Fourier transform of an image
- There are some enhancement techniques based on various combinations of methods from these two categories.

Enhanced Images

- For human vision
  - The visual evaluation of image quality is a highly subjective process.
  - It is hard to standardize the definition of a good image.
- For machine perception
  - The evaluation task is easier.
  - A good image is one which gives the best machine recognition results.
- A certain amount of trial and error usually is required before a particular image enhancement approach is selected.