Unit 2-1. Digital Image Representation and Processing

CS 3570

Shang-Hong Lai

CS Dept., NTHU

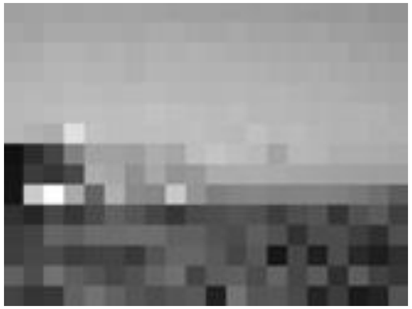
Content

- Bitmapped images
- Image Data Types
- Aliasing
- Dithering
- Interpolation
- Image transform

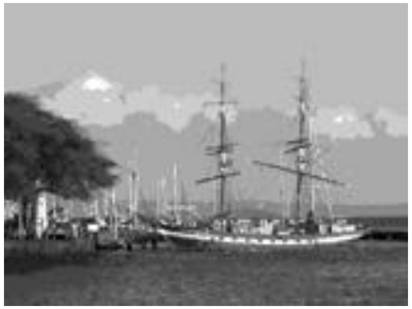
Bitmaps - digitization

- ***Pixel (picture element)***
  - A number representing the color at position (*r*, *c*) in the bitmap, where *r* is the row and *c* is the column.
- **Sampling**
  - Under-sampled images may contain blocky and unclear effect.
- **Quantization**
  - Low-bit depth can result in patchiness of color

Original



Sampling



Quantization

Bitmaps – Sampling and Quantization

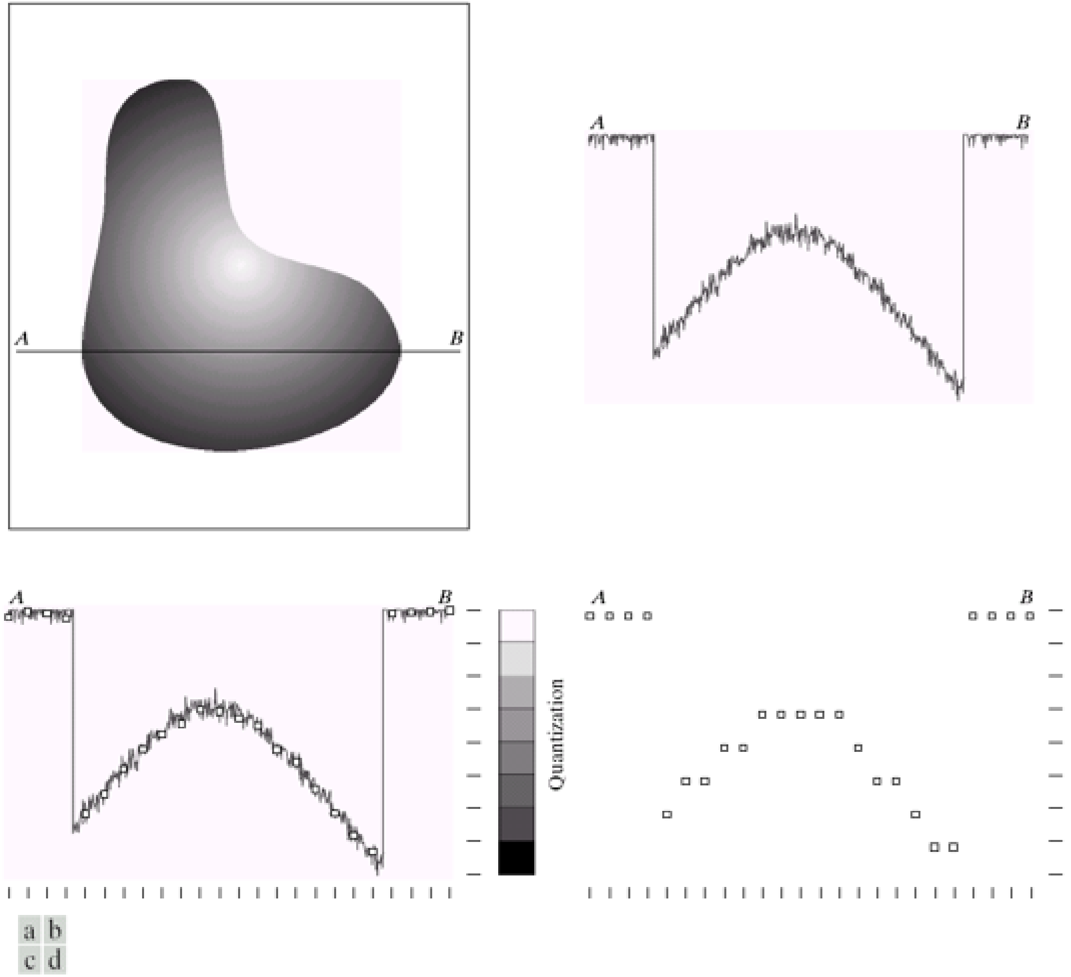


FIGURE 2.16 Generating a digital image. (a) Continuous image. (b) A scan line from *A* to *B* in the continuous image, used to illustrate the concepts of sampling and quantization. (c) Sampling and quantization. (d) Digital scan line.

Sampling :

- Digitizing the coordinate values.
- During the sampling step, you need to set a sampling rate.

Quantization :

- Digitizing the amplitude values.
- During the quantization step, you need to set bit depth.

- Sampling rate: How frequent to take a data.
- Bit depth: Each sample is represented with a fixed number of bits.

Bitmaps – Sampling and Quantization

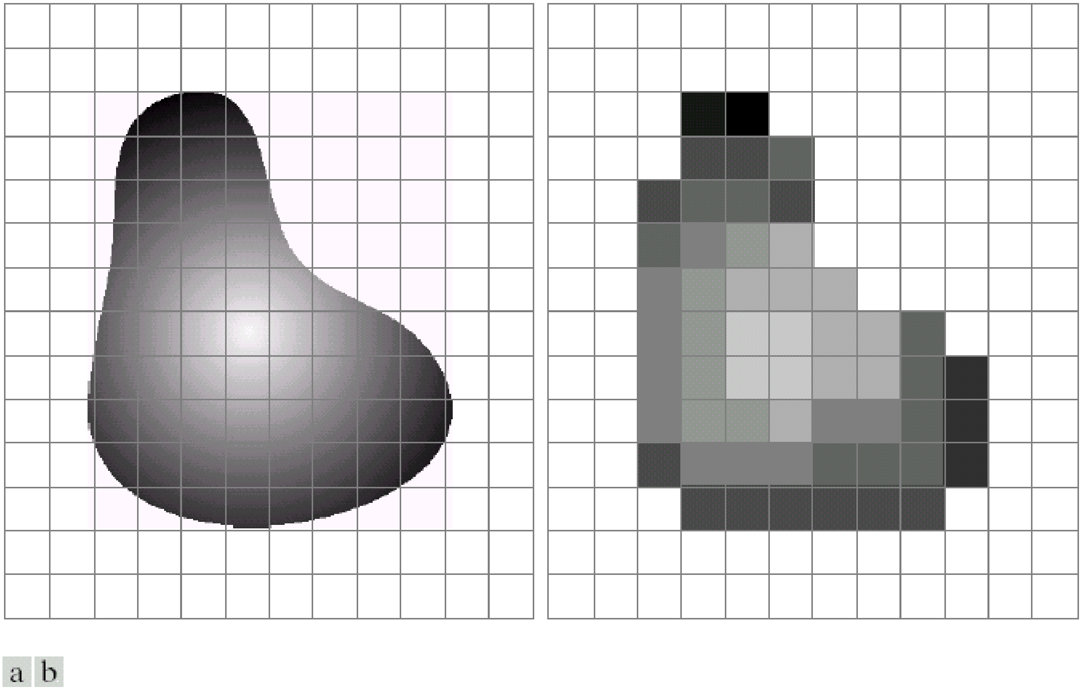
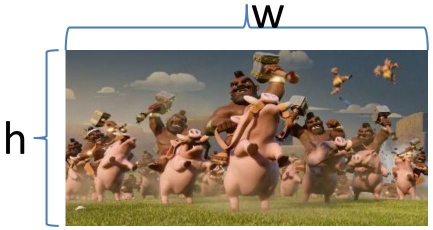


FIGURE 2.17 (a) Continuos image projected onto a sensor array. (b) Result of image sampling and quantization.

Bitmaps – pixel dimensions & resolution

- ***Pixel dimensions***
  - The number of pixels horizontally (*i.e.*, width, *w*) and vertically (*i.e.*, height, *h*), denoted $w \times h$



- ***Resolution***
  - The number of pixels in an image per unit of spatial measure.
  - Resolution can be measured in ***pixels per inch*** **(PPI).** A 200 PPI image will print out using 200 pixels to determine the colors for each inch.
  - Resolution of a printer is a matter of how many dots of color it can print over an area. A common measurement is ***dots per inch*** **(DPI)**.

Image Data Types

- Common image data types:
  - 1-bit image
  - 8-bit gray-level image
  - 24-bit color image
  - 8-bit color image.

- The most common data types for graphics and image file formats —
  - 24-bit color and 8-bit color.

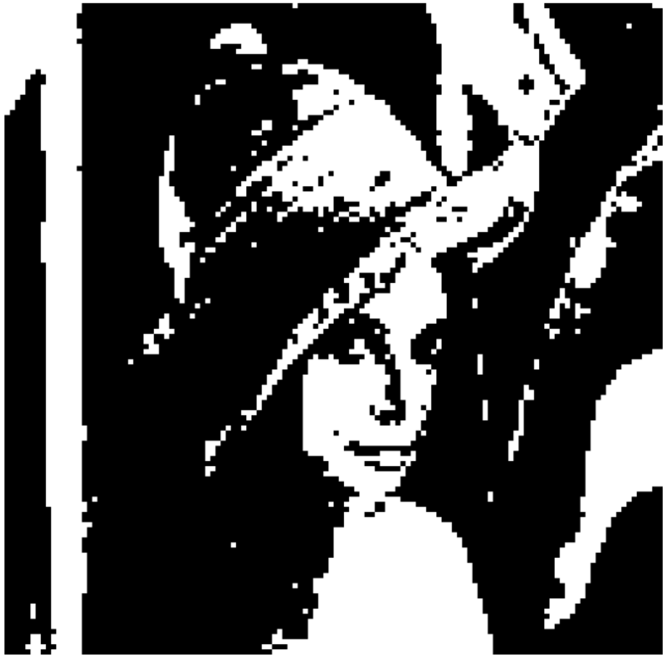
- Some formats are restricted to particular hardware / operating system platforms, while others are "cross-platform" formats.

- Even if some formats are not cross-platform, there are conversion applications that will recognize and translate formats from one system to another.

- Most image formats incorporate some variation of a compression technique due to the large storage size of image files.

1-Bit Images

- Each pixel is stored as a single bit (0 or 1), so also referred to as binary image.
- Such an image is also called a 1-bit monochrome image since it contains no color.

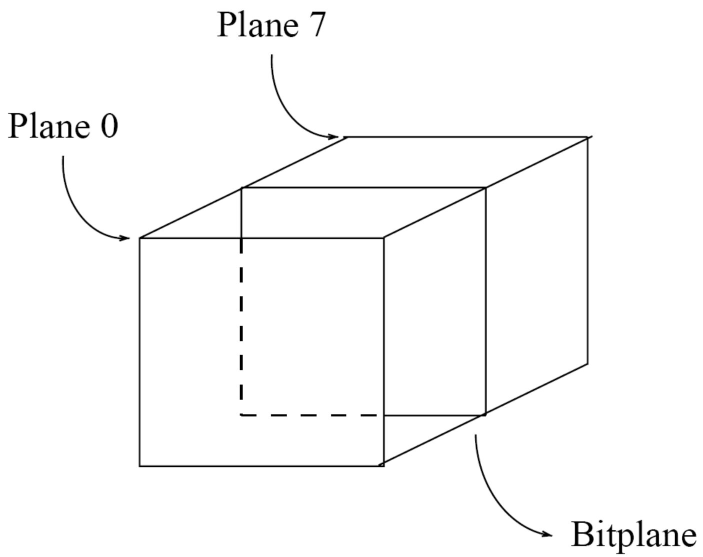


Monochrome 1-bit Lena image.

8-bit Gray-level Images

- Each pixel has a gray-value between 0 and 255, which is represented by a single byte.

- An 8-bit image can be thought of as a set of 1-bit bit-planes, where each plane consists of a 1-bit contribution to the image.



Bit-planes for 8-bit grayscale image.

8-bit Gray-level Images

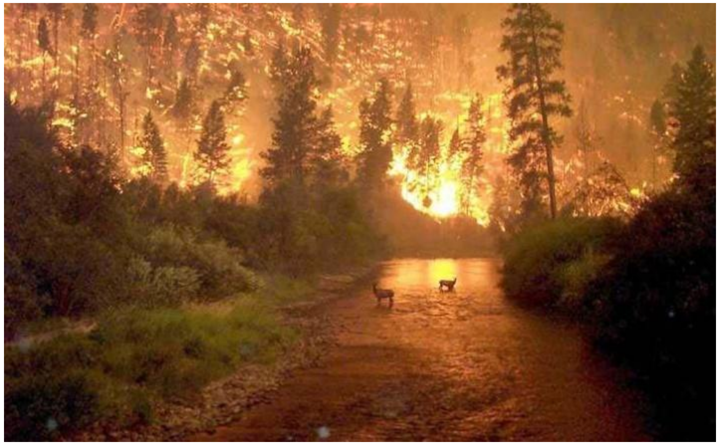
- Each pixel is usually stored as a byte (a value between 0 to 255), so a 640 x 480 grayscale image requires 300 kB of storage.
- When an image is printed, the basic strategy of dithering is used, which trades intensity resolution for spatial resolution to provide ability to print multi-level images on 2-level (1-bit) printers.

Grayscale image of Lena.

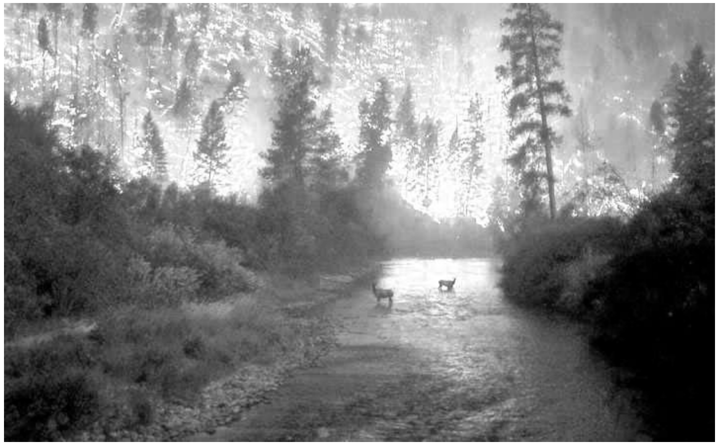
24-bit Color Images

- In a color 24-bit image, each pixel is represented by three bytes, usually representing RGB.
  - This format supports 256 x 256 x 256 possible combined colors, or a total of 16,777,216 possible colors.
  - However, it results in a storage penalty: a 640 x 480 24-bit color image would require 921.6 kB of storage without any compression.
- An important point: many 24-bit color images are actually stored as 32-bit images, with the extra byte of data for each pixel used to store an alpha value representing special effect information (e.g., transparency).

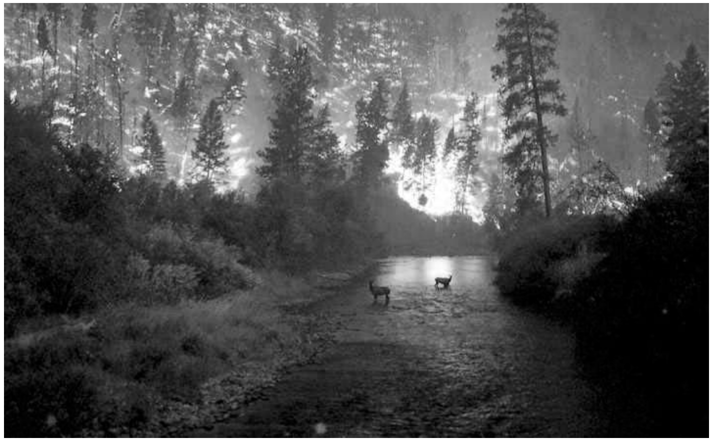
Example of 24-bit Color Image



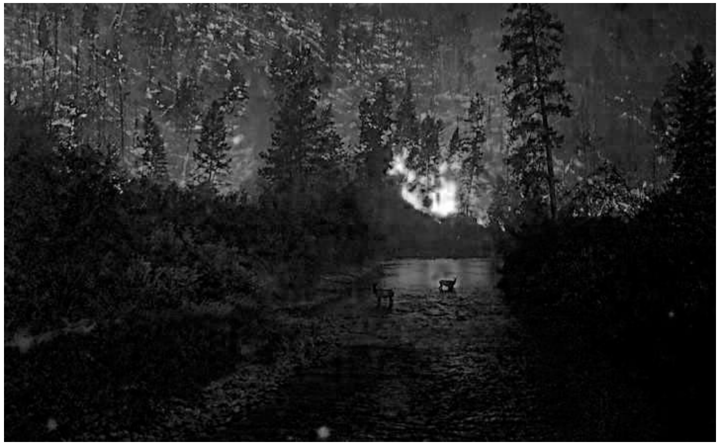
24-bit color image forestfire.bmp



R-channel



G-channel



B-channel

Higher Bit-depth Images

- More information about the scene being imaged can be gained by using more accuracy for pixel depth (64 bits, say); or by using special cameras that view more than just three colors (i.e., RGB).
  - could use invisible light (e.g., infra-red, ultraviolet) for security cameras: “dark flash”.
  - use higher-dimensional medical images of skin (> 3-D) to diagnose skin carcinoma.
  - in satellite imaging, use high-D to obtain types of crop growth, etc.

- Such images are called multispectral (more than 3 colors) or hyperspectral (a great many image planes, say 224 colors for satellite imaging).

8-Bit Color Images

- Many systems can make use of 8 bits of color information (the so-called “256 colors”) in producing a screen image.

- Such image files use the concept of a lookup table to store color information.
    - Basically, the image stores not color, but instead just a set of bytes, each of which is actually an index into a table with 3-byte values that specify the color for a pixel with that lookup table index.

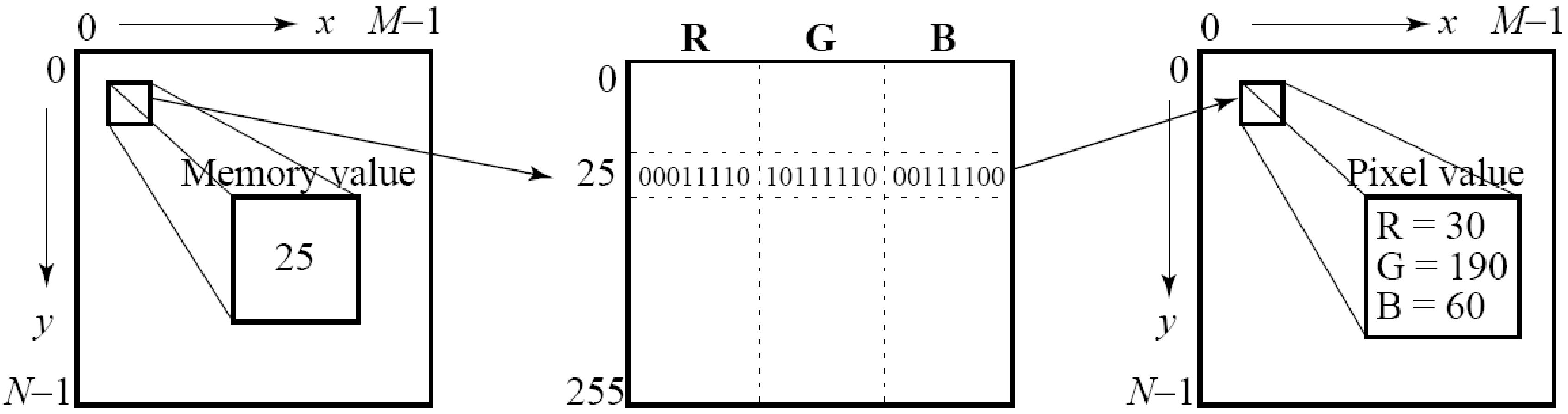
Example of 8-Bit Color Image

Example of 8-bit color image.

- Note the great savings in space for 8-bit images, over 24-bit ones: a 640 x 480 8-bit color image only requires 300 kB of storage, compared to 921.6 kB for a color image (again, without any compression applied).

Color Look-up Tables (LUTs)

- The idea used in 8-bit color images is to store only the index, or code value, for each pixel. Then, e.g., if a pixel stores the value 25, the meaning is to go to row 25 in a color look-up table (LUT).



Color LUT for 8-bit color images.

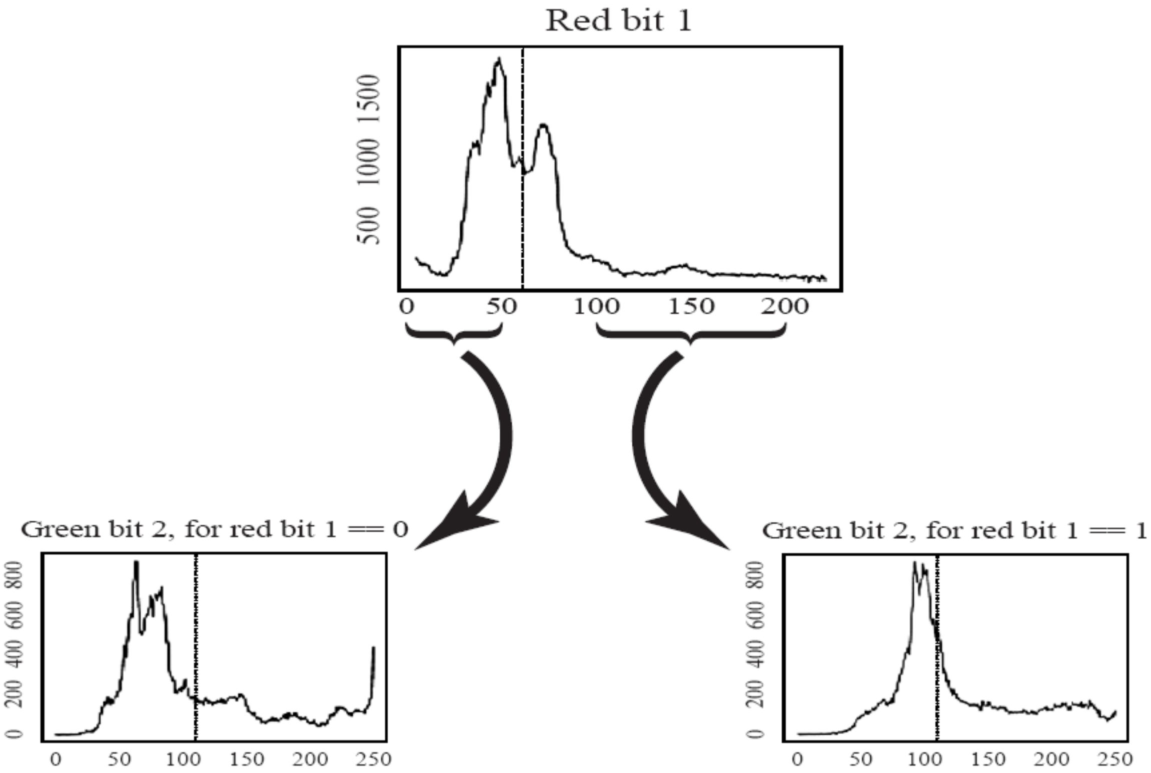
How to devise a color look-up table

- The most straightforward way to make 8-bit look-up color out of 24-bit color would be to divide the RGB cube into equal slices in each dimension.
    - The centers of each of the resulting cubes would serve as the entries in the color LUT, while simply scaling the RGB ranges 0..255 into the appropriate ranges would generate the 8-bit codes.
    - Since humans are more sensitive to R and G than to B, we could shrink the R range and G range 0..255 into the 3-bit range 0..7 and shrink the B range down to the 2-bit range 0..3, thus making up a total of 8 bits.
    - To shrink R and G, we could simply divide the R or G byte value by (256/8)=32 and then truncate. Then each pixel in the image gets replaced by its 8-bit index and the color LUT serves to generate 24-bit color.

Median-Cut Algorithm

- Median-cut algorithm: A simple alternate solution that does a better job for this color reduction problem.
  - The idea is to sort the R byte values and find their median; then values smaller than the median are labelled with a “0” bit and values larger than the median are labelled with a “1” bit.
  - This type of scheme will indeed concentrate bits where they most need to differentiate between high populations of close colors.
  - One can most easily visualize finding the median by using a histogram showing counts at position 0..255.

Median-Cut Algorithm



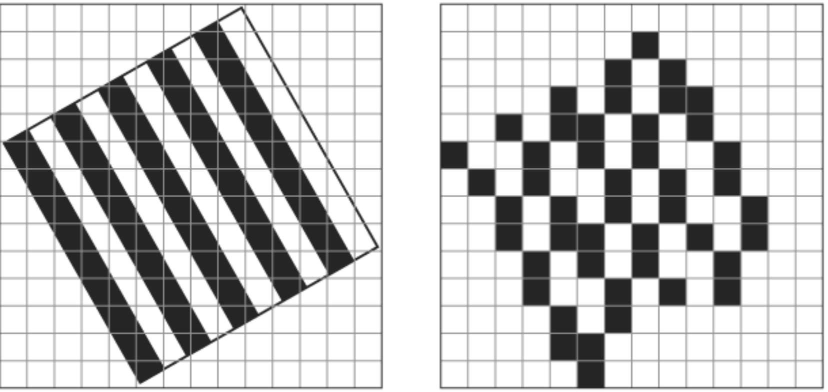
Histogram of R bytes for the 24-bit color image “forestfire.bmp” results in a 0 or 1 bit label for every pixel. For the second bit of the color table index being built, take R values less than the R median and label just those pixels as 0 or 1 according to whether their G value is less than or greater than the median of the G value, ... Continuing over R, G, B for 8 bits gives a color LUT 8-bit index.

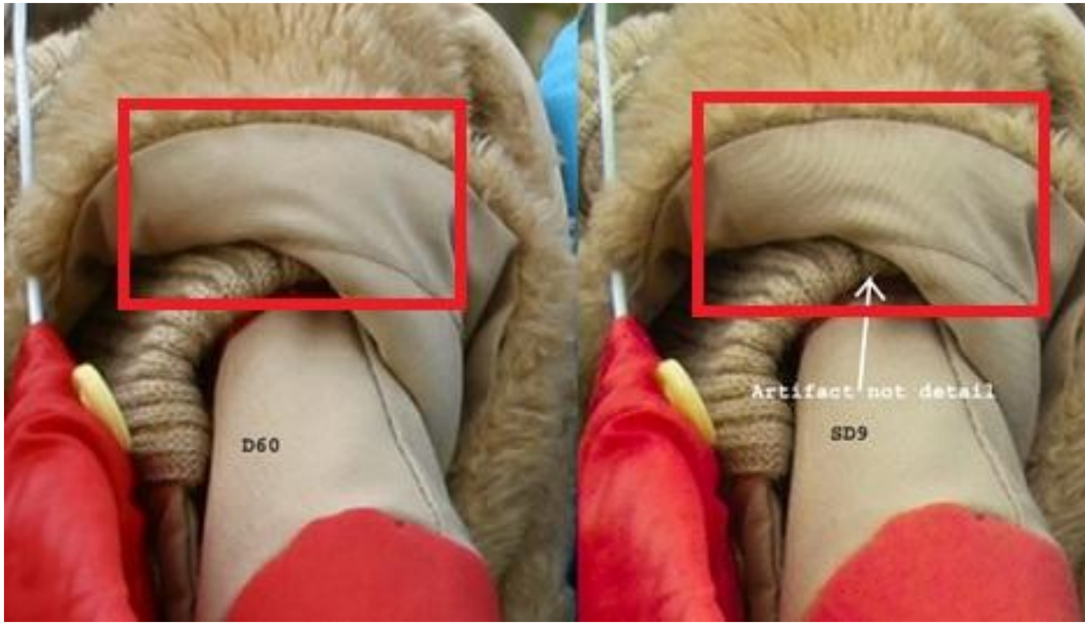
Popular Image File Formats

- 8-bit GIF: one of the most popular formats because of its historical connection to the WWW and HTML markup language as the first image type recognized by net browsers.
- JPEG: currently the most important common file format.
- PNG: most popular lossless image format.
- TIFF: flexible file format due to the addition of tags.
- BMP: standard image file format for Windows.
- PS and PDF: vector based language, popular in publishing and academia
- Many others

Aliasing - Moiré patterns

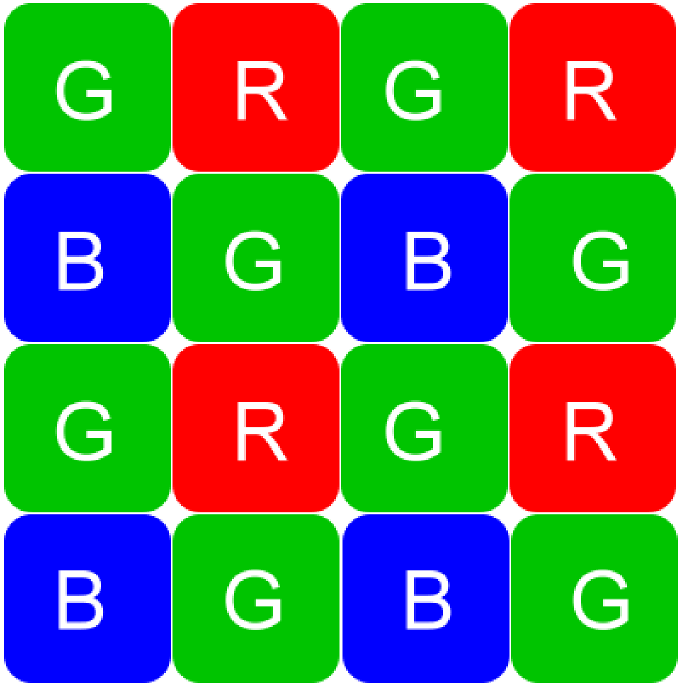
- **Aliasing**
  - It is cause by the discrete nature of pixels *(Sampling Error)*.
- **Moiré patterns** (also called **moiré effect**)
  - The sampling rate for the digital image is not high enough to capture the frequency of the pattern





Bayer Color Filter

- A less expensive method of color detection uses an array to detect only one color component at each photosite.
- Interpolation is then used to derive the other two color components based on information from neighboring sites.
- A ***color filter array* (CFA)** for arranging RGB color filters on a square grid of photosensors.
- Twice as many green sensors as blue or red sensors. Because the human eye is more sensitive to green.

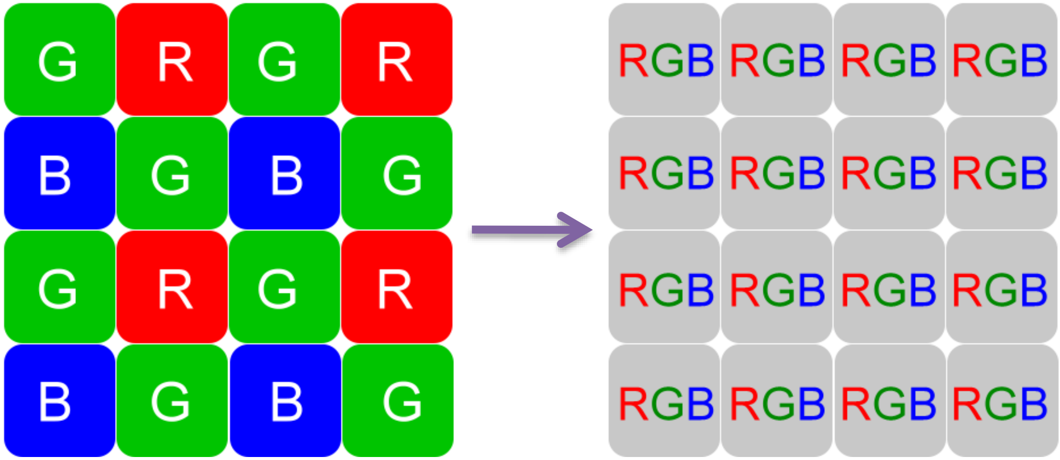


G	R	G	R
B	G	B	G
G	R	G	R
B	G	B	G

Bayer color filter

Demosaicing

- The interpolation algorithm for deriving the two missing color channels at each photo-site is called ***demosaicing***
- Algorithm
  - Nearest neighbor
  - Linear
  - Cubic
  - Cubic spline



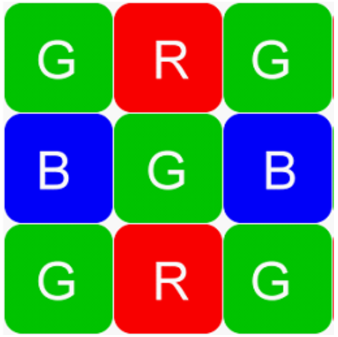
G	R	G	R
B	G	B	G
G	R	G	R
B	G	B	G

RGB	RGB	RGB	RGB
RGB	RGB	RGB	RGB
RGB	RGB	RGB	RGB
RGB	RGB	RGB	RGB

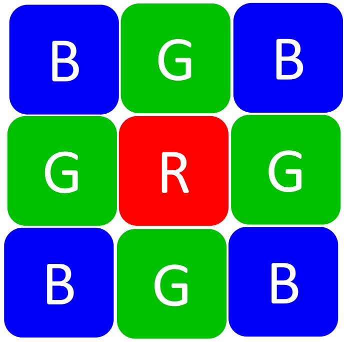
Nearest neighbor algorithm

ALGORITHM 2.1 - NEAREST NEIGHBOR ALGORITHM

```
algorithm nearest_neighbor{
  for each photosite (i,j) where the photosite detects color c1 {
    for each c2 ∈ {red,green,blue} such that c2 ≠ c1 {
      S = the set of nearest neighbors of site (i,j) that have color c2
      set the color value for c2 at site (i,j) equal to the average of the color values
      of c2 at the sites in S
    }
  }
}
```



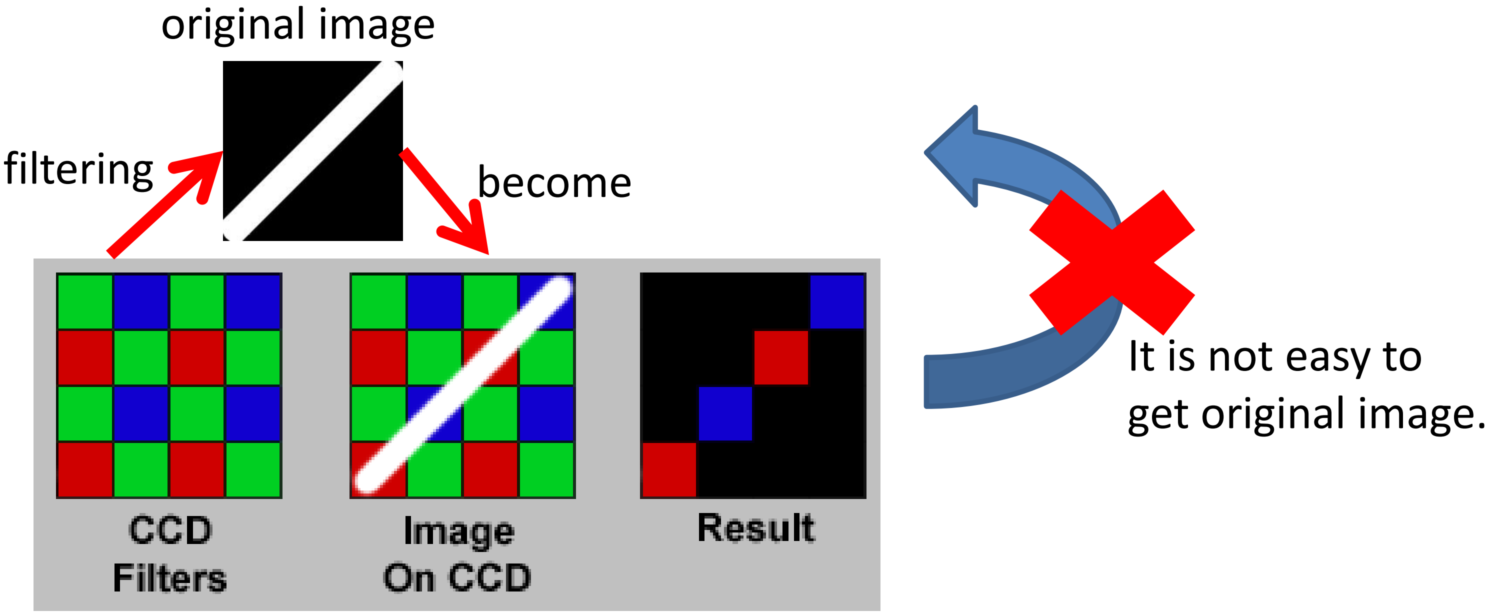
(a) Determining R or B from the center G photosite entails an average of two neighboring sites



(b) Determining B from the center R photosite entails an average of four neighboring sites diagonal from R

Color Aliasing

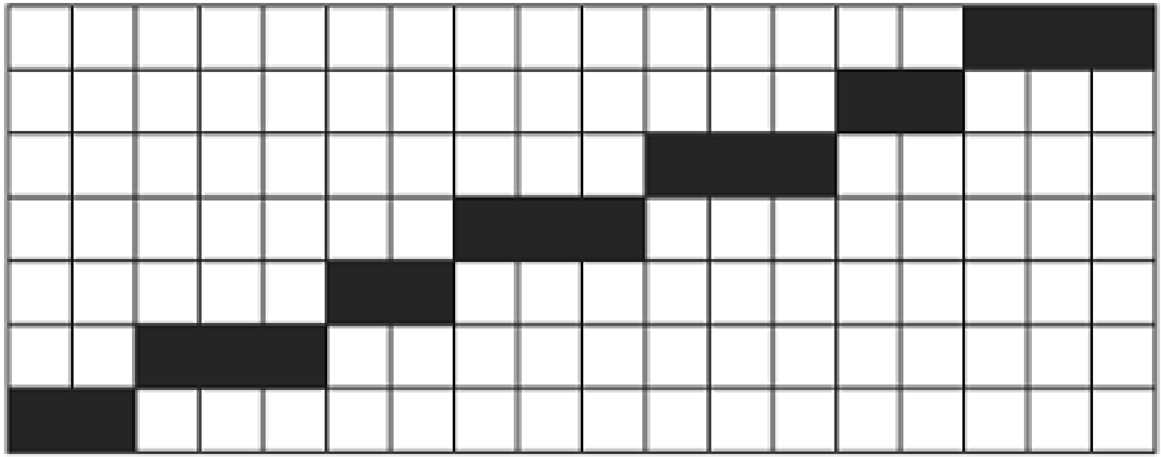
- Interpolation can’t give perfect reproduction of the scene being photographed, and occasionally color aliasing results from the process, detected as moiré patterns, streaks, or spots of color not present in the original scene.
- Example: you photograph a white line on the black background



It is not easy to get original image.

Aliasing - Jagged Edges

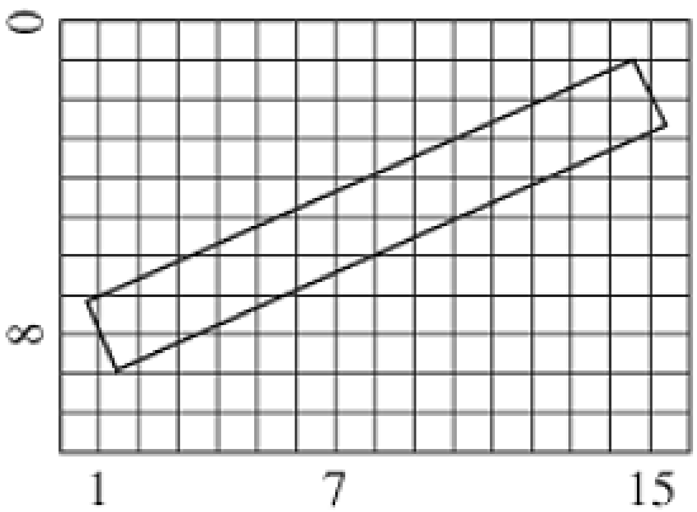
- ***Aliasing*** used to describe the jagged edges along lines or edges that are drawn at an angle across a computer screen
- A line on a computer screen is made up of discrete units: ***pixels***



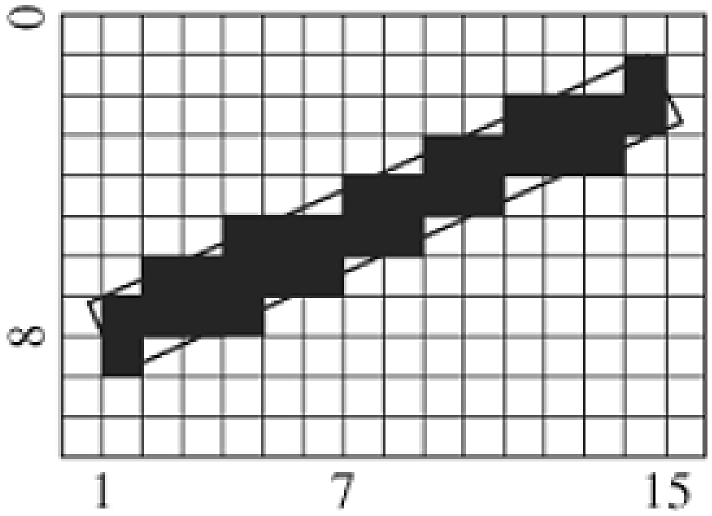
Aliasing of a line that is one pixel wide

Algorithm for drawing a line

- Left figure shows a line that is two pixels wide going from point (8, 1) to point (2, 15)
- In right figure a pixel is colored black if at least half its area is covered by the two-pixel line



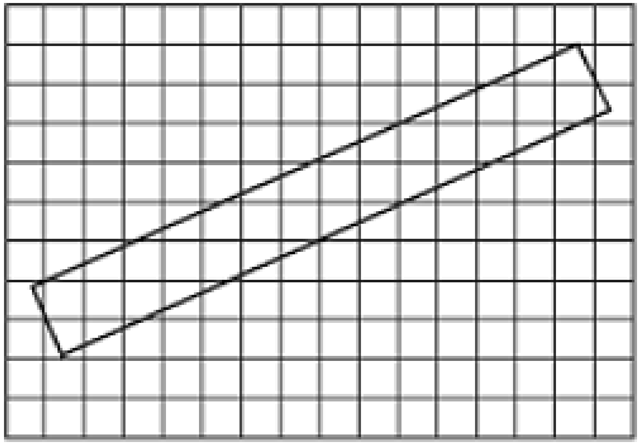
Drawing a line two pixels wide



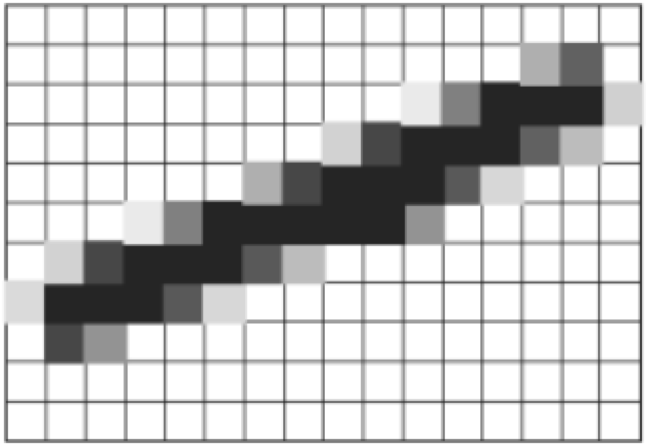
Line two pixels wide, aliased

Anti-aliasing

- **Anti-aliasing** is a technique for reducing the jaggedness of lines or edges caused by aliasing
- The idea is to color a pixel with a shade of its designated color in proportion to the amount of the pixel that is covered by the line or edge



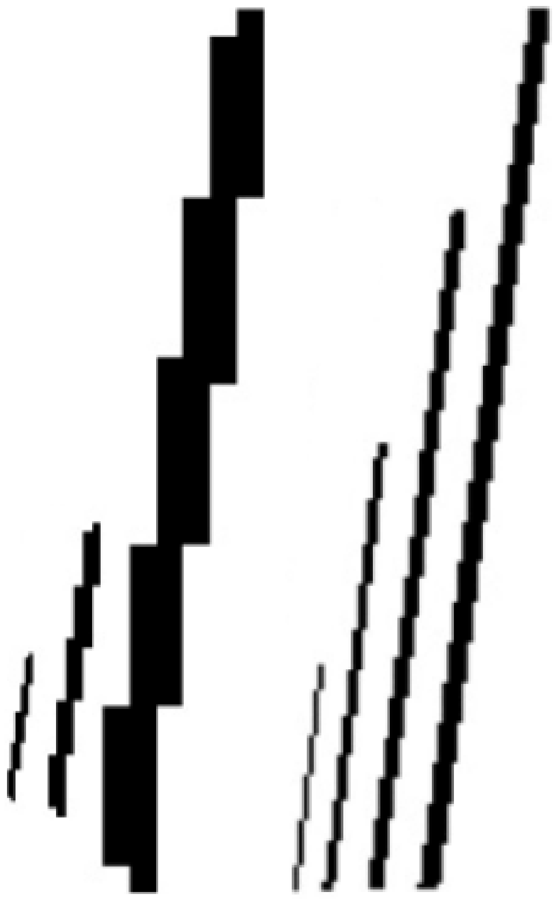
a line two pixels wide



Line two pixels wide, anti-aliased

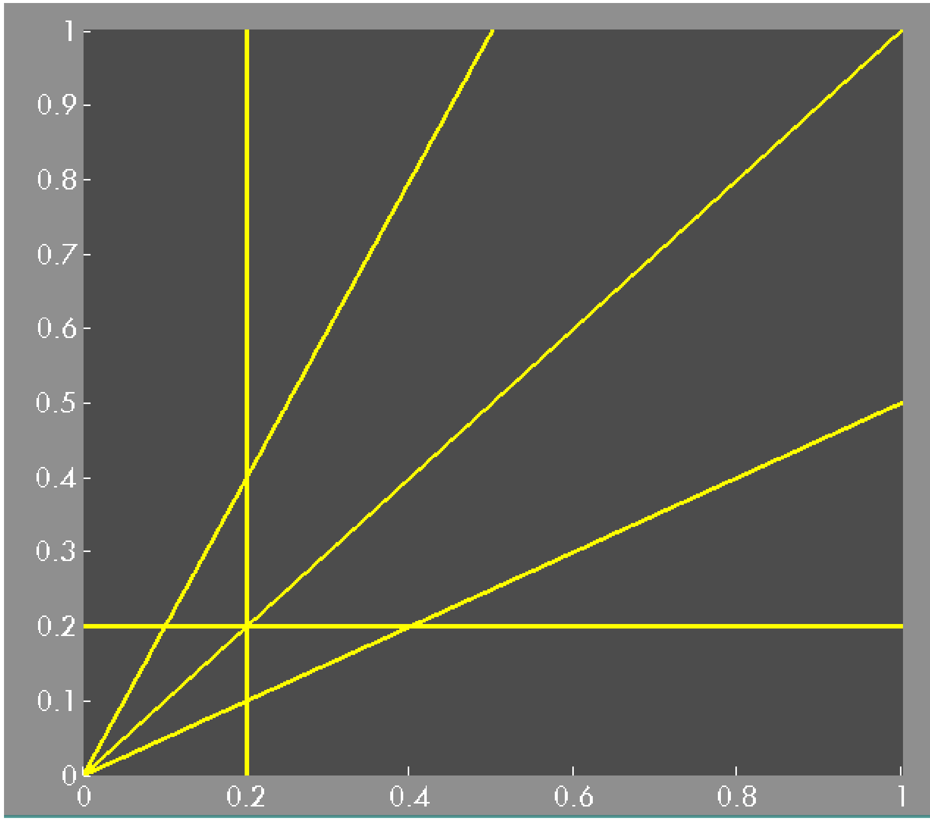
Aliasing of bitmap & vector graph

- Vector graphics suffer less from aliasing problems than do bitmap images in that vector graphics images can be resized without loss of resolution
- **Upsampling**
  - The image is later enlarged by increasing the number of pixels

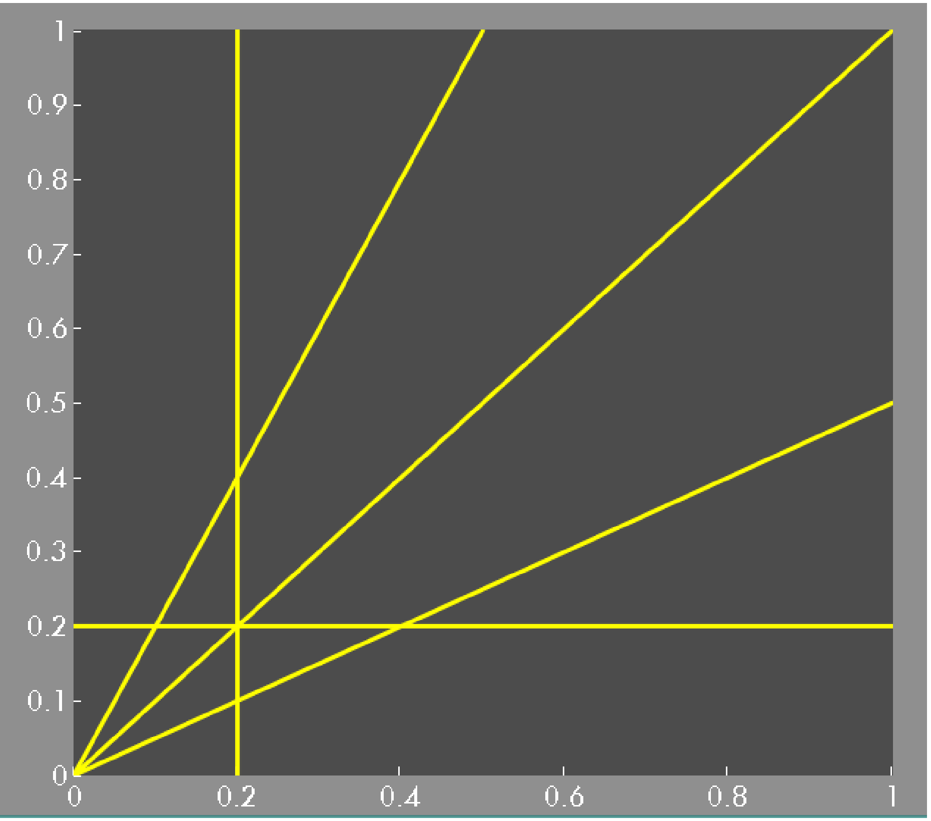


MATLAB Line smoothing

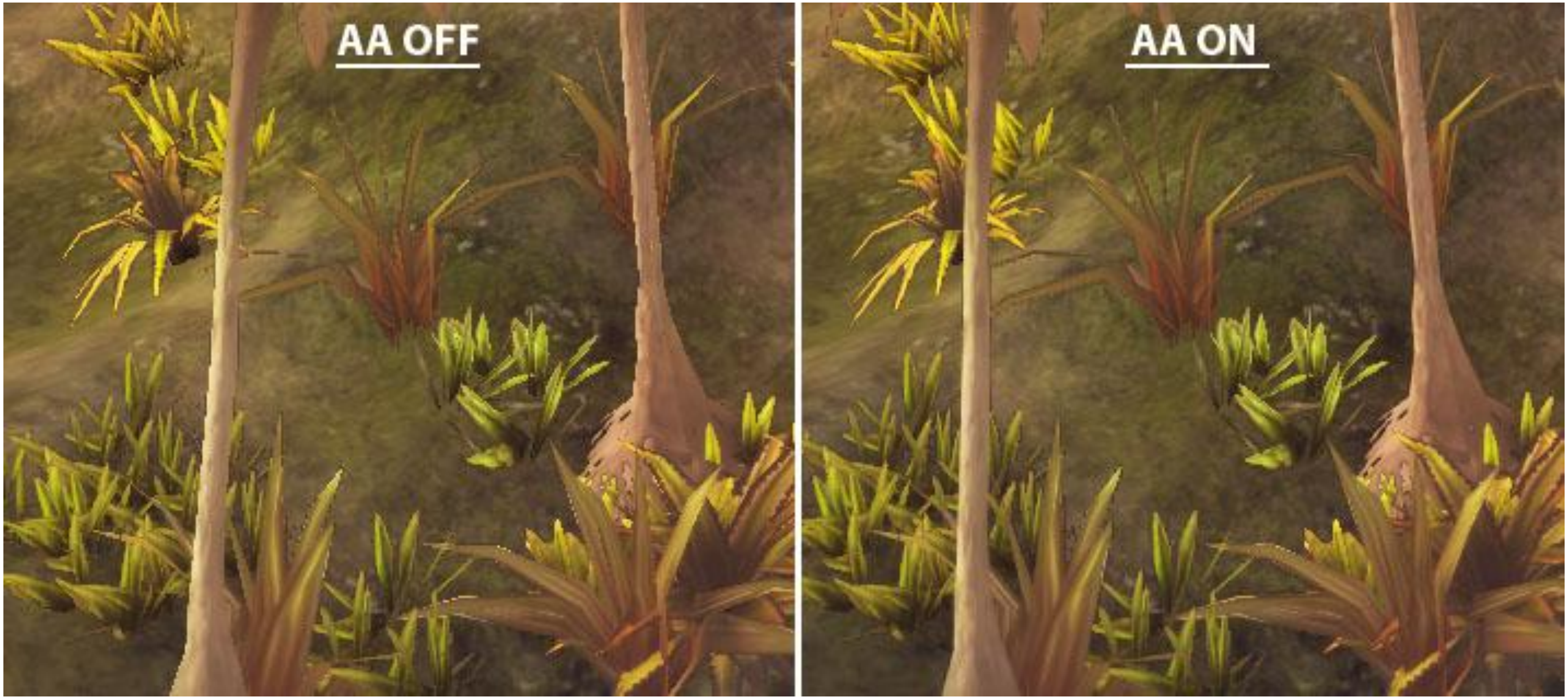
Ex: plot([0, 1], [0, 1], 'LineSmoothing', 'off');



Ex: plot([0, 1], [0, 1], 'LineSmoothing','on');



Aliasing V.S. Anti-aliasing



Dithering

- ***Dithering*** is a technique for simulating colors that are unavailable in a palette by using available colors that are blended by the eye so that they look like the desired colors

- Dithering is helpful when you change an image from RGB mode to indexed color because it makes it possible to reduce the bit depth of the image, and thus the file size, without greatly changing the appearance of the original image

Dithering

- Dithering is used to calculate patterns of dots such that values from 0 to 255 correspond to patterns that are more and more filled at darker pixel values, for printing on a 1-bit printer.
- The main strategy is to replace a pixel value by a larger pattern, say 2 x 2 or 4 x 4, such that the number of printed dots approximates the varying-sized disks of ink used in analog, in halftone printing (e.g., for newspaper photos).
  - Half-tone printing is an analog process that uses smaller or larger filled circles of black ink to represent shading, for newspaper printing.
  - For example, if we use a 2 x 2 dither matrix:

$$\begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix}$$

we can first re-map image values in 0..255 into the new range 0..4 by (integer) dividing by 256/5. Then, e.g., if the pixel value is 0 we print nothing, in a 2 x 2 area of printer output. But if the pixel value is 4 we print all four dots.

- The rule is:
  - If the intensity is > the dither matrix entry then print an on dot at that entry location: replace each pixel by an n x n matrix of dots.

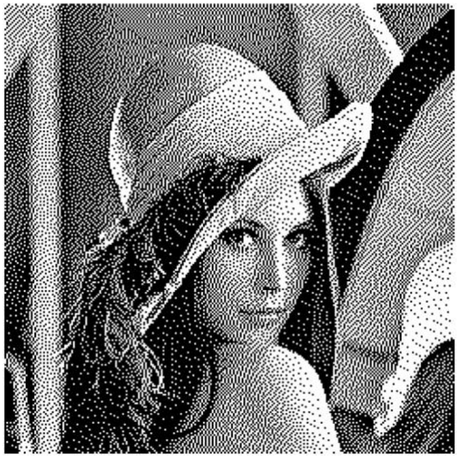
- Note that the image size may be much larger, for a dithered image, since replacing each pixel by a 4 x 4 array of dots, makes an image 16 times as large.

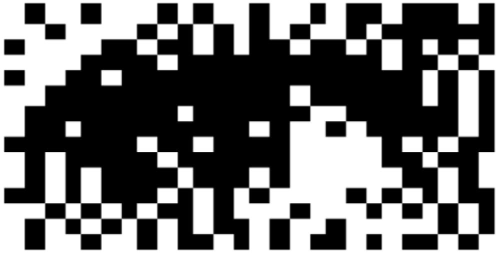
Ordered Dither

- A clever trick can get around this problem. Suppose we wish to use a larger, 4 x 4 **dither matrix**, such as

$$\begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

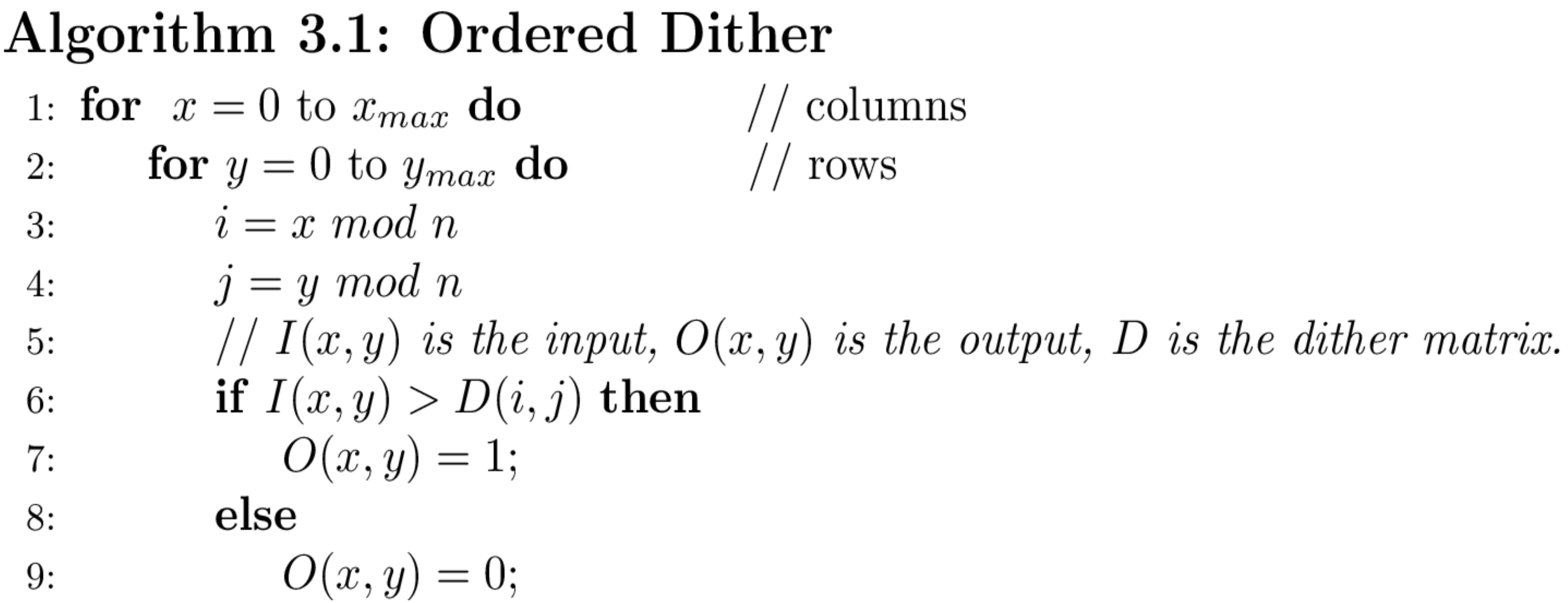
- An **ordered dither** consists of turning on the printer output bit for a pixel if the intensity level is greater than the particular matrix element just at that pixel position.





Ordered Dither

- An algorithm for Ordered Dither with n x n dither matrix:



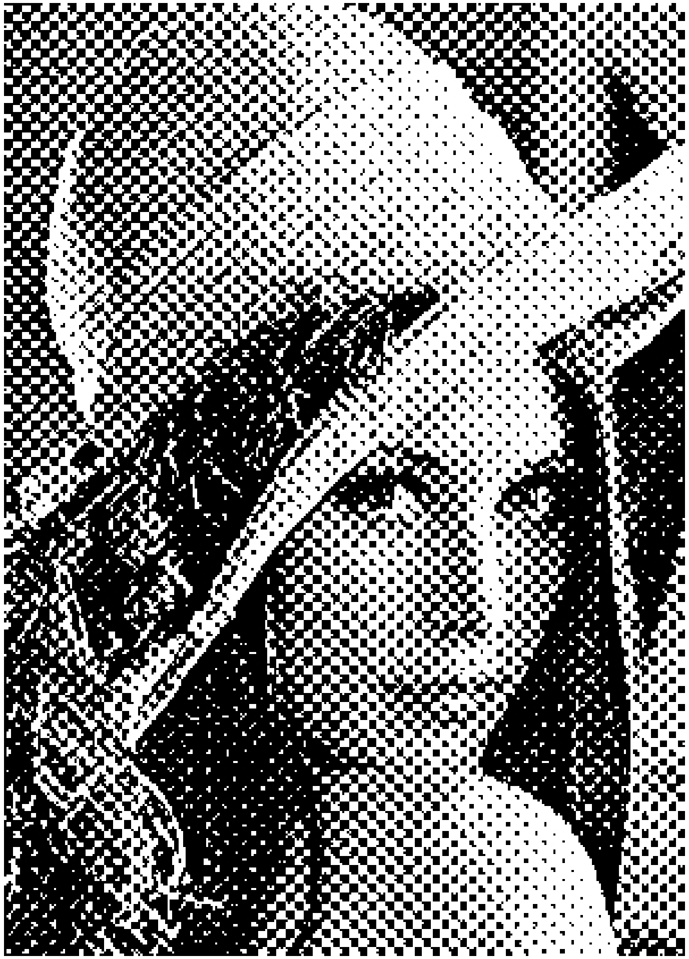
Algorithm 3.1: Ordered Dither

```
for $x = 0$ to $x_{max}$ do                // columns
    for $y = 0$ to $y_{max}$ do            // rows
        $i = x \; mod \; n$
        $j = y \; mod \; n$
        // $I(x, y)$ is the input, $O(x, y)$ is the output, $D$ is the dither matrix.
        if $I(x, y) > D(i, j)$ then
            $O(x, y) = 1;$
        else
            $O(x, y) = 0;$
```

Pattern Dithering

Threshold matrix

$$\begin{bmatrix} 200 & 250 & 100 \\ 220 & 150 & 200 \\ 10 & 150 & 50 \end{bmatrix}$$



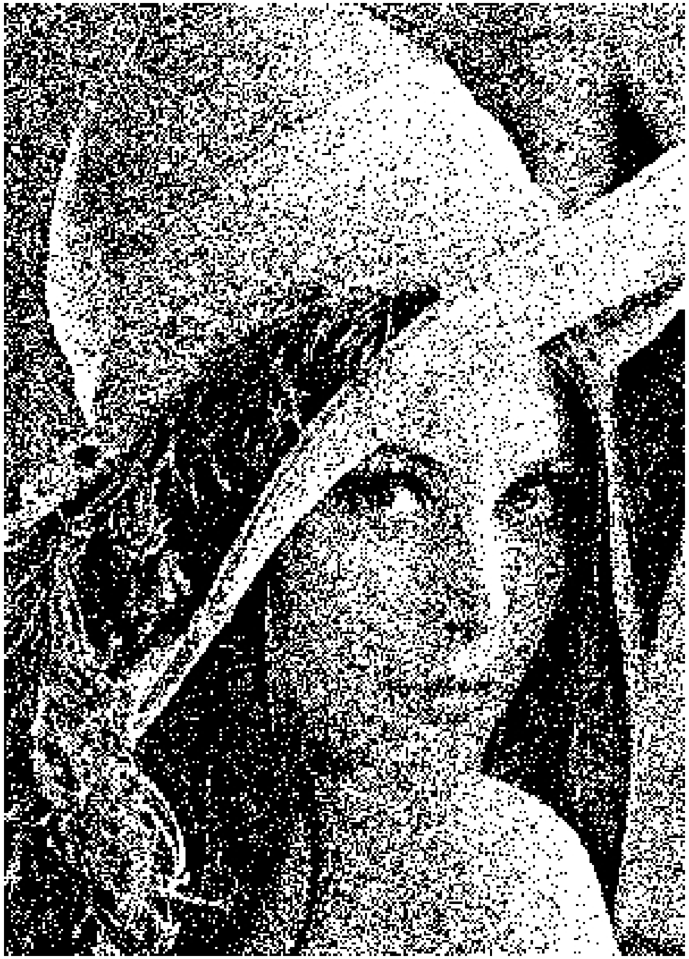
Noise Dithering

- Also called ***random dithering***

1. Generate a random number from 0 to 255
2. If the pixel's color value is greater than the number then it is white, otherwise black
3. Repeat step 2 for each pixel in the image

- Crude and "noisy"

Example of Noise Dithering



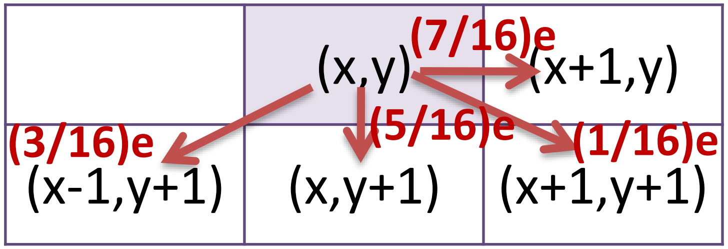
Error Diffusion Dithering

- Also called ***Floyd-Steinberg algorithm***
- Disperse the error or difference between a pixel's original value and the color (or grayscale) value available
- Alleviate the error accumulation problem in dithering

Error Diffusion Dithering

1. Define the mask, for example:
2. For each pixel p
   1. Define e: if p < 128, e = p; otherwise e = p - 255
   2. Change the value of neighbor pixel

	p	7
3	5	1

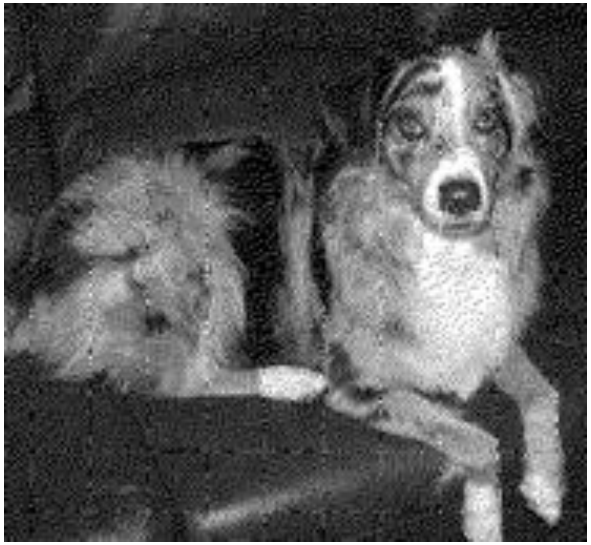


p(x+1,y) = p(x+1,y) + (7/16)e
p(x-1,y+1) = p(x-1,y+1) + (3/16)e
p(x,y+1) = p(x,y+1) + (5/16)e
p(x+1,y+1) = p(x+1,y+1) + (1/16)e

- When the mask is moved to the right by one pixel, the next step will operate on a pixel that has possibly been changed in a previous step

Error Diffusion Dithering

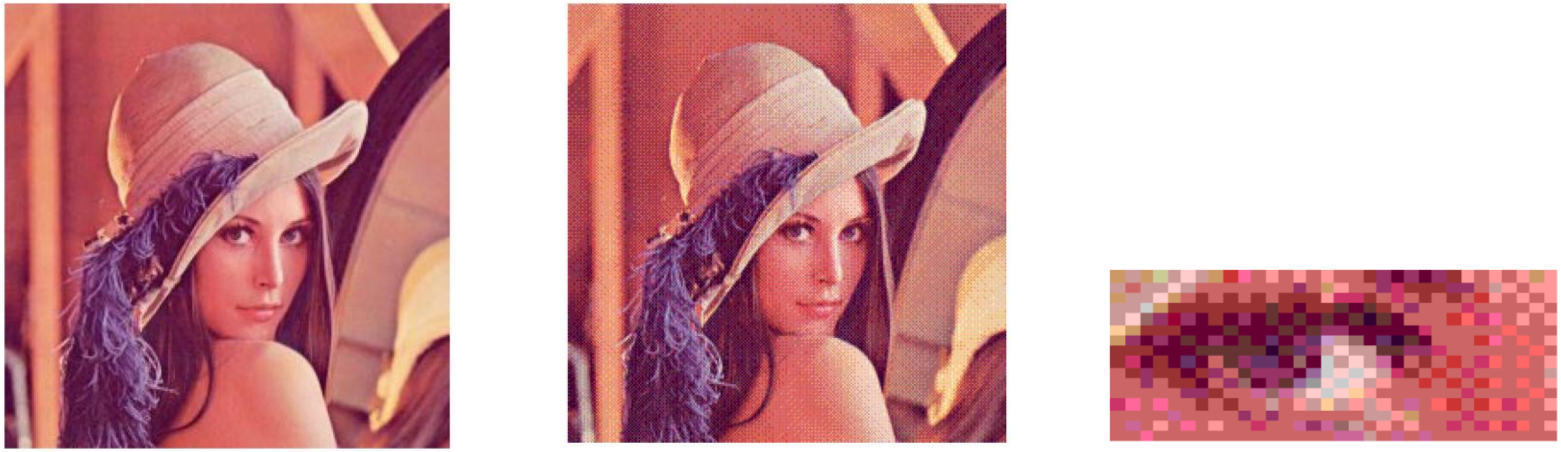
- After the error has been distributed over the whole image, the pixels are processed a second time. This time for each pixel, if the pixel value is less than 128, it is changed to a 0 in the dithered image. Otherwise it is changed to a 1.



After dithering

Color Dithering

- Fig. 3.10 (a) shows a 24-bit color image of “Lena”, and Fig. 3.10 (b) shows the same image reduced to only 5 bits via dithering. A detail of the left eye is shown in Fig. 3.10 (c).



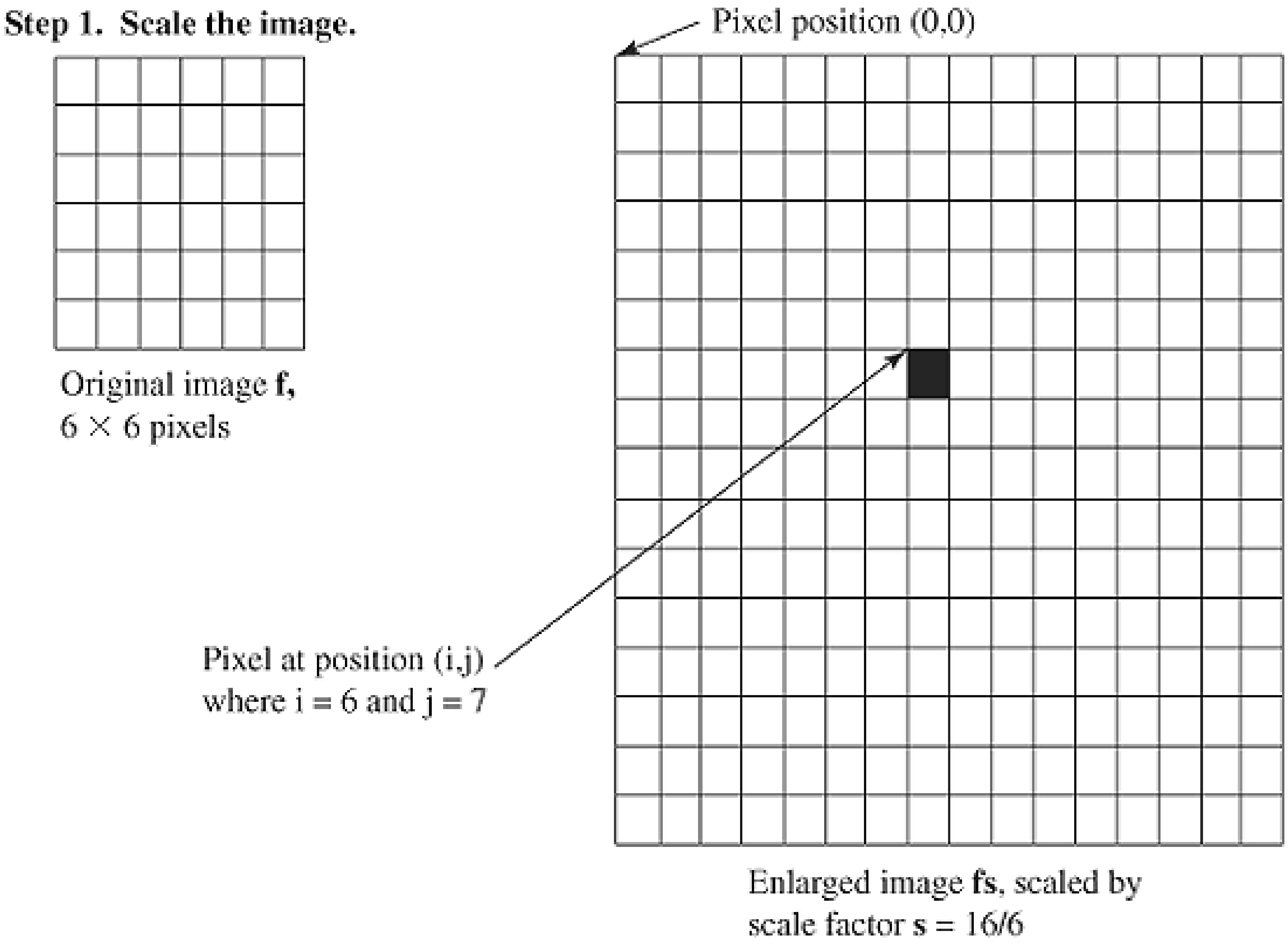
(a) (b) (c)

Fig. 3.10: (a): 24-bit color image “lena.bmp”. (b): Version with color dithering. (c): Detail of dithered version.

Interpolation

- There are interpolation methods for resampling that give better results than simple replication or discarding of pixels
- ***Interpolation*** is a process of estimating the color of a pixel based on the colors of neighboring pixels
  - Nearest neighbor
  - Bilinear
  - Bicubic

The first two steps in resampling - 1



Step 1. Scale the image.

Pixel position (0,0)

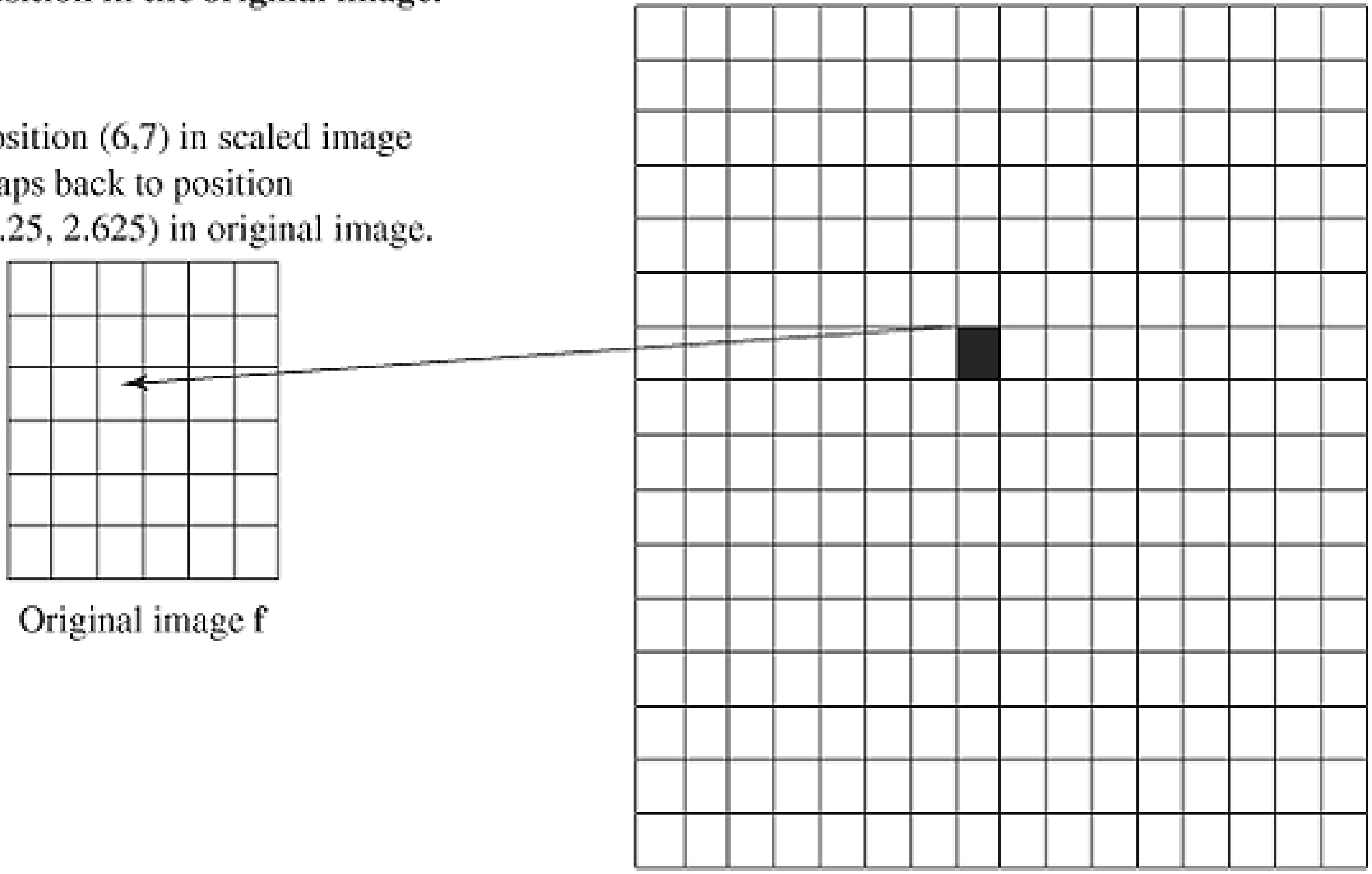
Original image **f**,
6 × 6 pixels

Pixel at position (i,j)
where i = 6 and j = 7

Enlarged image **fs**, scaled by
scale factor **s** = 16/6

The first two steps in resampling - 2

Step 2. Map each pixel in the scaled image back to a position in the original image.



Position (6,7) in scaled image maps back to position (2.25, 2.625) in original image.

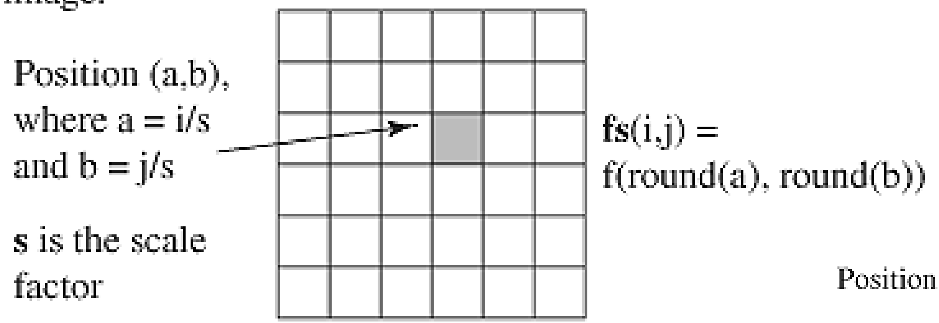
Original image f

Scaled image fs

Nearest neighbor interpolation

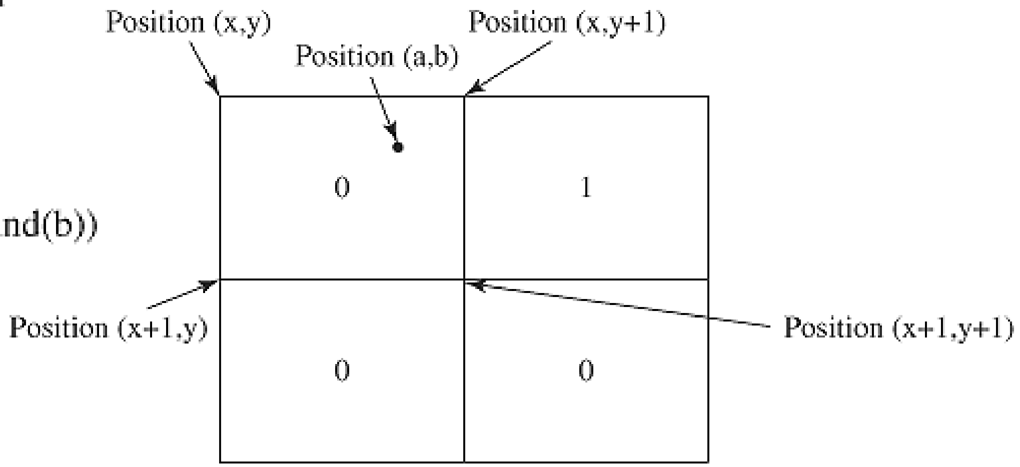
- ***Nearest neighbor interpolation*** simply rounds down to find one close pixel whose value is used for ***fs***(*i, j*)

The **nearest neighbor algorithm** assigns to **fs**(i,j) the color value f(round(a), round(b)) from the original image.



The nearest neighbor is marked in gray.

Nearest neighbor interpolation



Position with coordinates closest to both a and b gets a 1 in the convolution kernel.

Bilinear interpolation

- ***Bilinear interpolation*** uses four neighbors and makes ***fs***(*i, j*) a weighted sum of their color values. The contribution of each pixel toward the color of ***fs***(*i, j*) is a function of how close the pixel's coordinates are to (*a, b*).

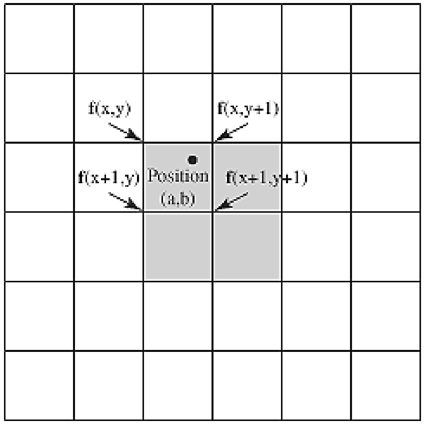
Bilinear interpolation uses an average color value of the four pixels surrounding position (a,b) in the original image. Each neighbor's contribution to the color is based on how close it is to (a,b). Let

x = floor(a)
y = floor(b)

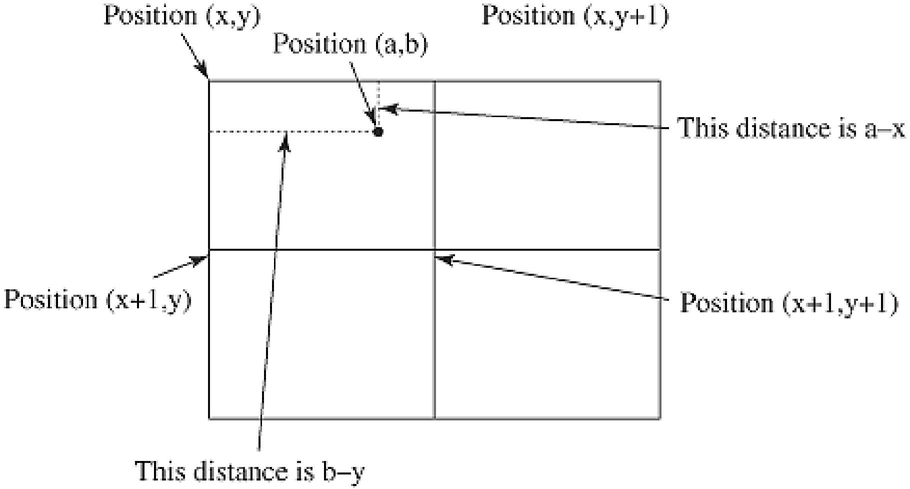
Then the pixels surrounding position (a,b) are

f(x, y)
f(x+1, y)
f(x+1, y+1)
f(x, y+1)

Neighborhood is shown in gray.



Bilinear interpolation



The color of the pixel in image **fs** is a weighted average of the four neighboring pixels. Weights come from each pixel's proximity to (a,b).

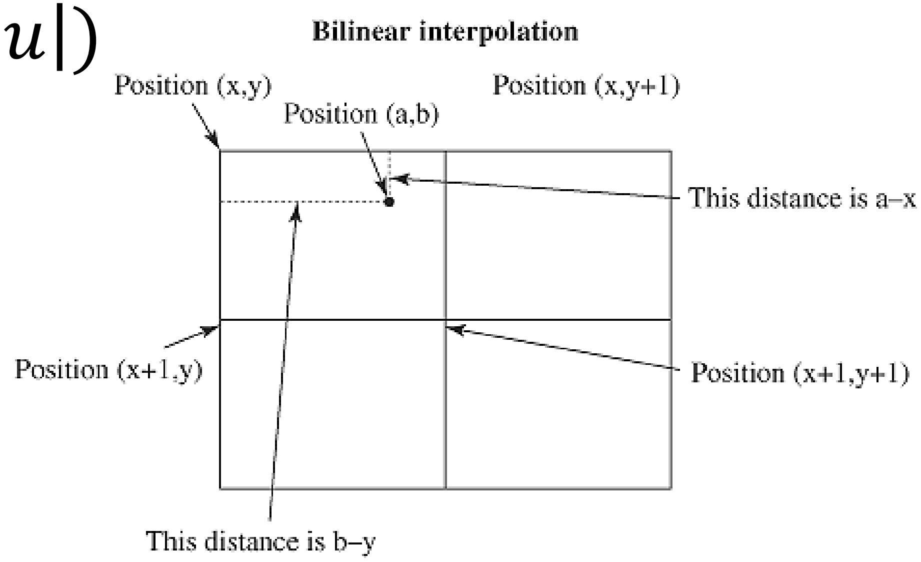
Bilinear interpolation

- 2 x 2 convolution mask H

$$t(m,n) = a - (x+m), 0 \le m \le 1, 0 \le n \le 1$$

$$u(m,n) = b - (y+n), 0 \le m \le 1, 0 \le n \le 1$$

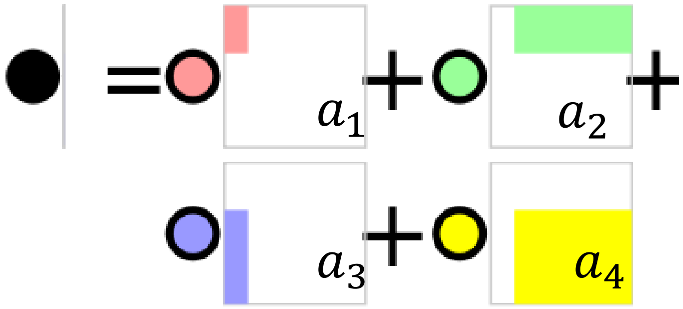
$$H(m,n) = (1-|t|)(1-|u|)$$

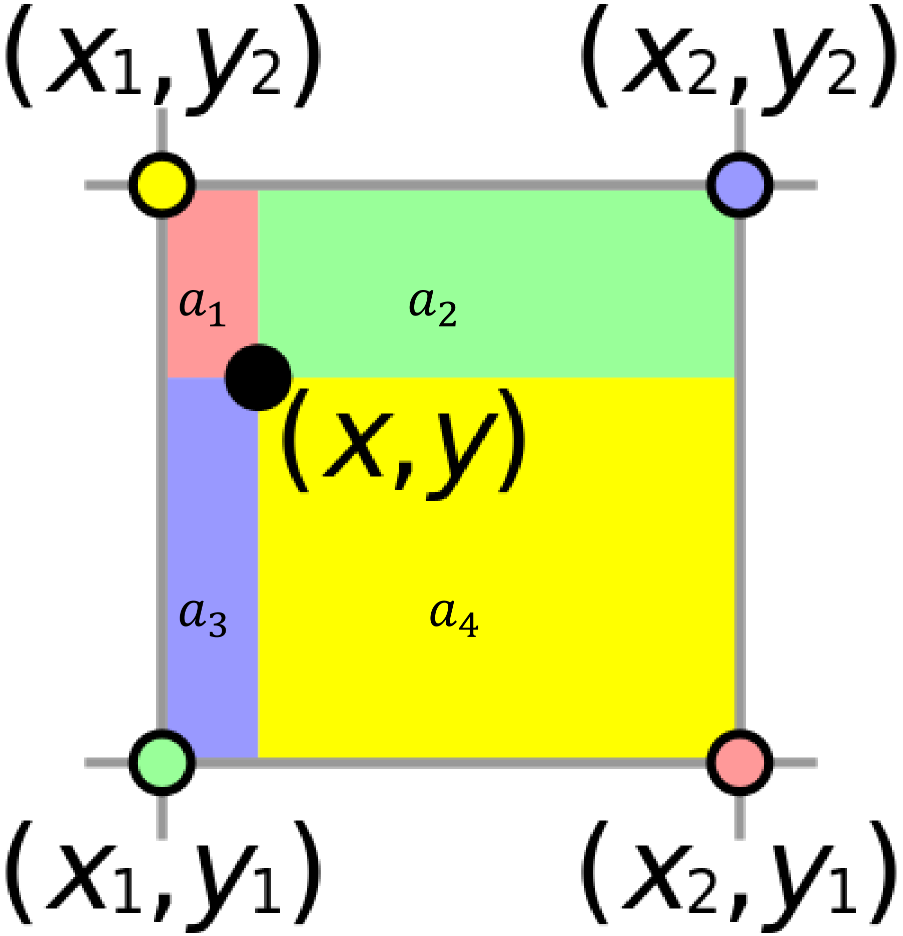


The color of the pixel in image **fs** is a weighted average of the four neighboring pixels. Weights come from each pixel's proximity to (a,b).

Bilinear interpolation

- Value of (x, y) can be determined:

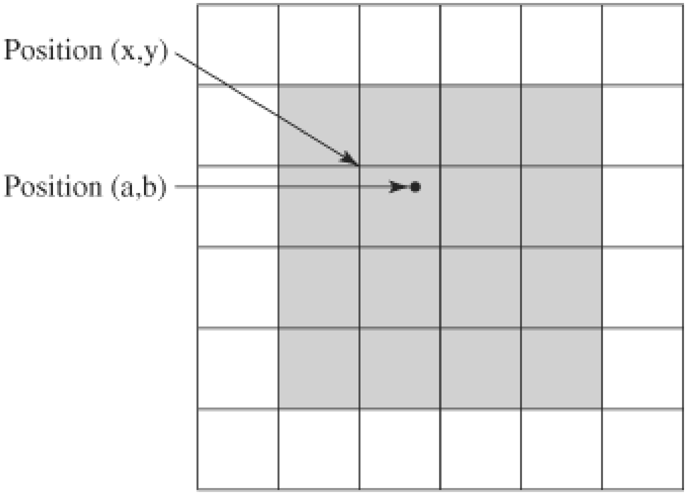




Bicubic interpolation

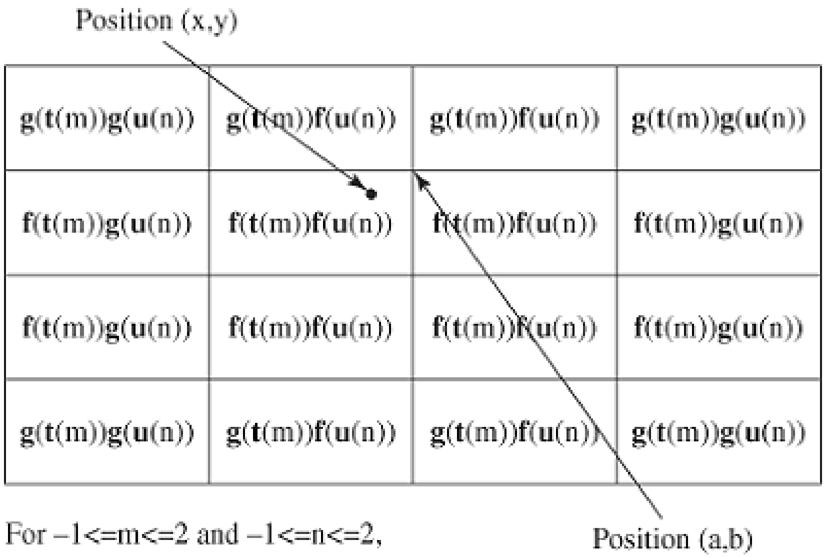
- ***Bicubic interpolation*** uses a neighborhood of sixteen pixels to determine the value of ***fs***(*i, j*)

Bicubic interpolation uses an "average" color value of the 16 pixels surrounding position (a,b) in the original image. The weight of each neighbor's contribution is based on a cubic equation that accounts for how close each neighbor is.



Neighbors are shaded in gray. The neighborhood of (a,b) extends from x–1 to x+2 in the vertical direction and from y–1 to y+2 in the horizontal direction.

Bicubic interpolation



g(t(m))g(u(n))	g(t(m))f(u(n))	g(t(m))f(u(n))	g(t(m))g(u(n))
f(t(m))g(u(n))	f(t(m))f(u(n))	f(t(m))f(u(n))	f(t(m))g(u(n))
f(t(m))g(u(n))	f(t(m))f(u(n))	f(t(m))f(u(n))	f(t(m))g(u(n))
g(t(m))g(u(n))	g(t(m))f(u(n))	g(t(m))f(u(n))	g(t(m))g(u(n))

4x4 convolution mask

For $-1 <= m <= 2$ and $-1 <= n <= 2$,

$t(m) = a - (x + m)$

$u(n) = b - (y + n)$

$f(t(m)) = 1 - 2t(m)^2 + |t(m)|^3$

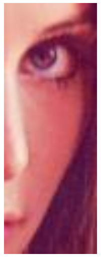
$f(u(n)) = 1 - 2u(n)^2 + |u(n)|^3$

$g(t(m)) = 4 - 8|t(m)| + 5t(m)^2 - |t(m)|^3$

$g(u(n)) = 4 - 8|u(n)| + 5u(n)^2 - |u(n)|^3$

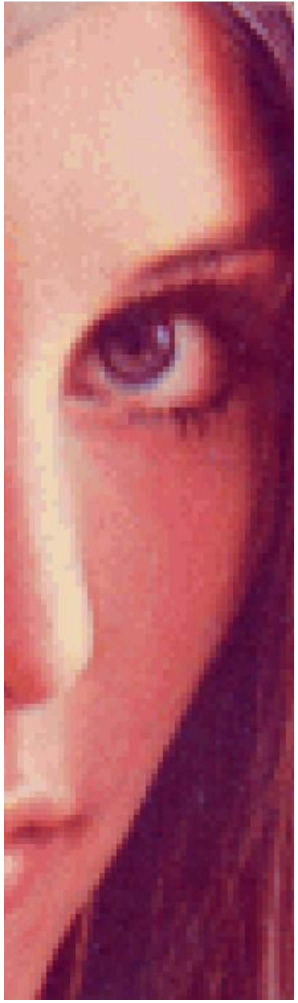
Interpolation Comparison

1x



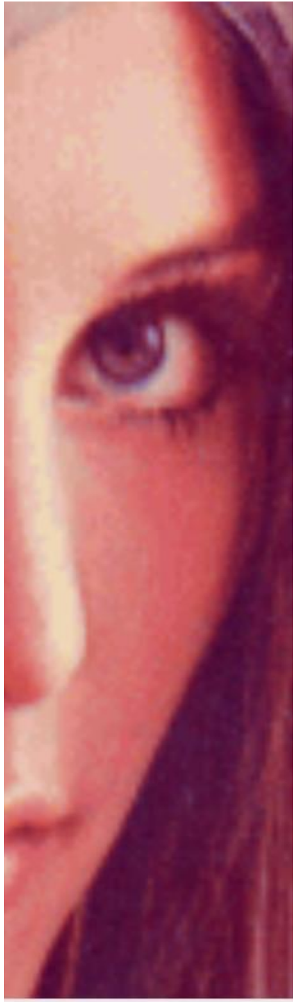
Original

8x



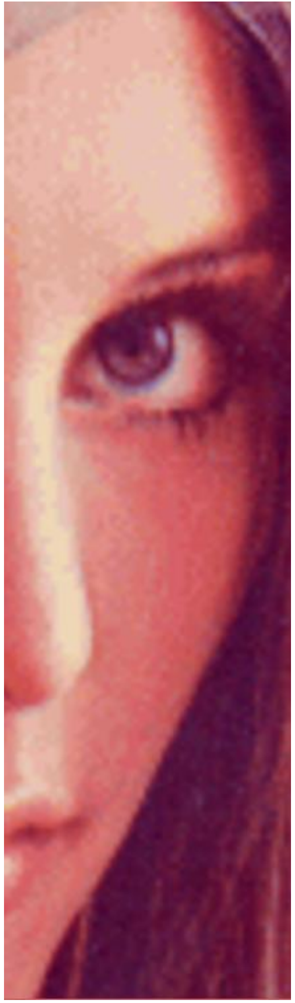
Nearest neighbor

8x



bilinear

8x



bicubic

Image Transform

- An **image transform** is a process of changing the color or grayscale values of image pixels
- Image transforms can be divided into two types
  - ***Pixel point processing***
    a pixel value is changed based only on its original value, without reference to surrounding pixels
  - ***Spatial filtering***
    changes a pixel's value based on the values of neighboring pixels

Histogram

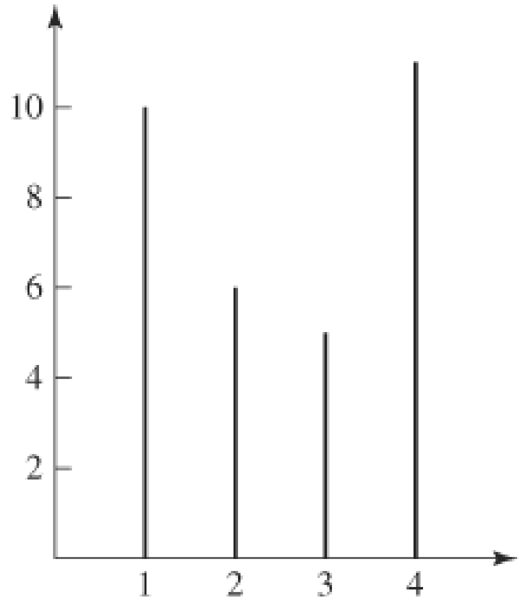
- A ***histogram*** is a discrete function that describes frequency distribution; that is, it maps a range of discrete values to the number of instances of each value in a group of numbers.
- Let v_i be the number of instances of value i in the set of numbers. Let *min* be the minimum allowable value of i, and let *max* be the maximum. Then the ***histogram function*** is defined as

$$h(i) = v_i \quad \text{for } min \le i \le max.$$

Histogram

- Simple histogram

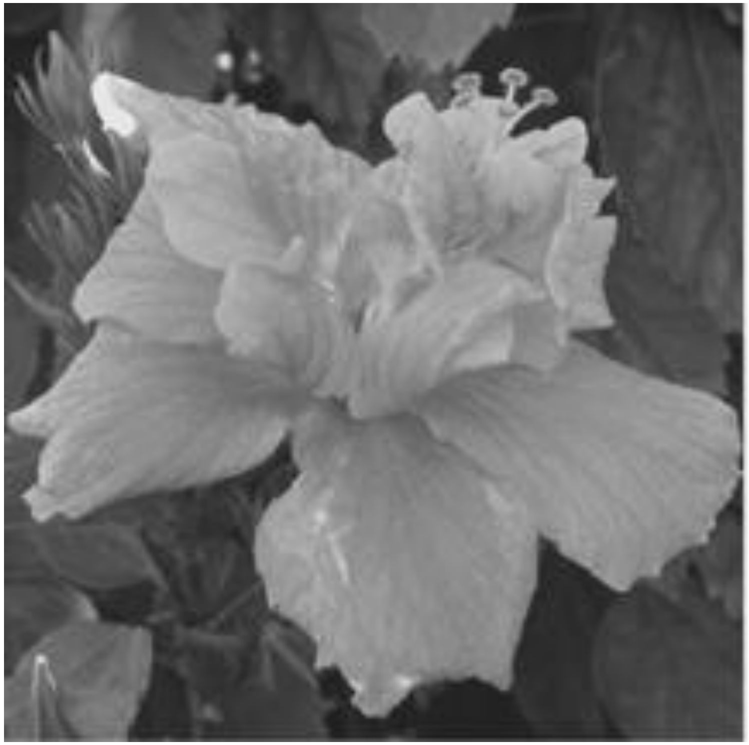
  a group of 32 students identified by class (1 for freshman, 2 for sophomore, 3 for junior, and 4 for senior)

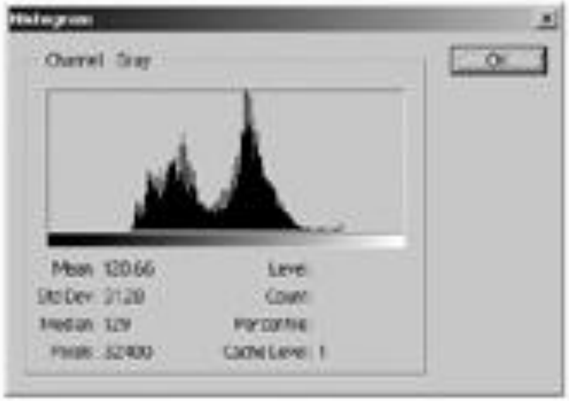


- There are ten freshmen, six sophomores, five juniors, and 11 seniors

Histogram

- An image histogram maps pixel values to the number of instances of each value in the image



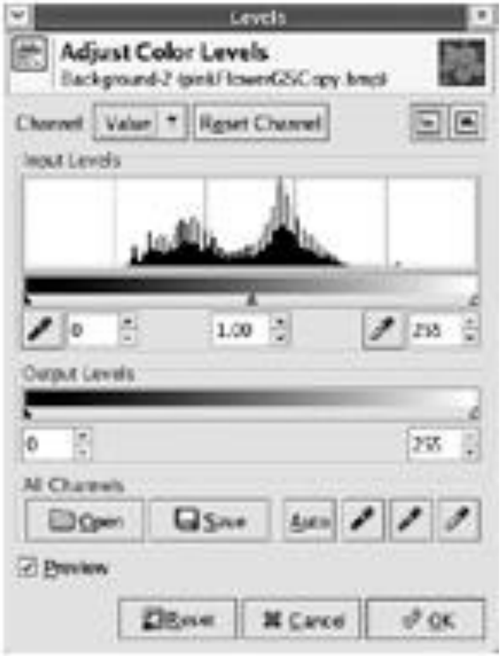


Histogram

Channel: Gray

Mean: 120.66
Std Dev: 31.20
Median: 129
Pixels: 32400

Level:
Count:
Percentile:
Cache Level: 1



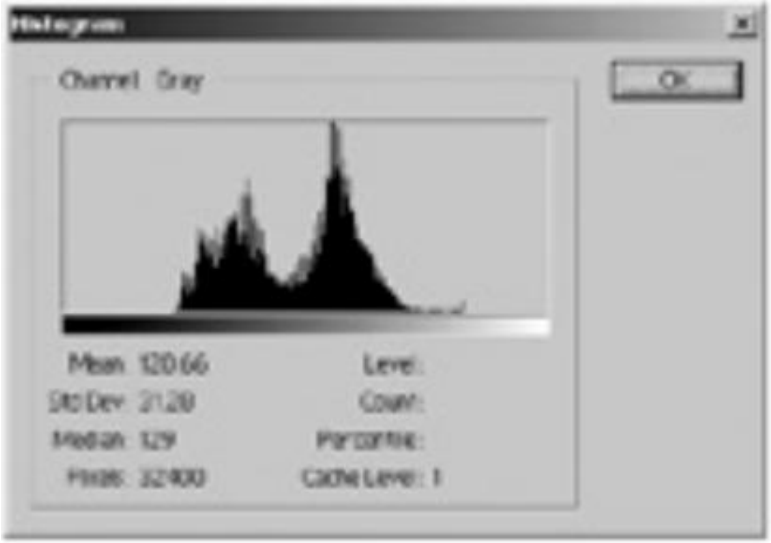
Levels

Adjust Color Levels
Background-2 (pinkFlowerGSCopy.bmp)

Channel: Value
Input Levels: 0 1.00 255
Output Levels: 0 255
☑ Preview

Histogram

- **Mean or average**
  - $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
- **Median**
  - A ***median*** is a value x such that at most half of the values in the sample population are less than x and at most half are greater.



Channel: Gray

Mean: 120.66 Level:
Std Dev: 21.20 Count:
Median: 129 Percentile:
Pixels: 32400 Cache Level: 1

- **Standard deviation**
  - $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(xi - \overline{x})^2}$

A large standard deviation implies that most of the pixel values are relatively far from the average, so the values are pretty well spread out over the possible range

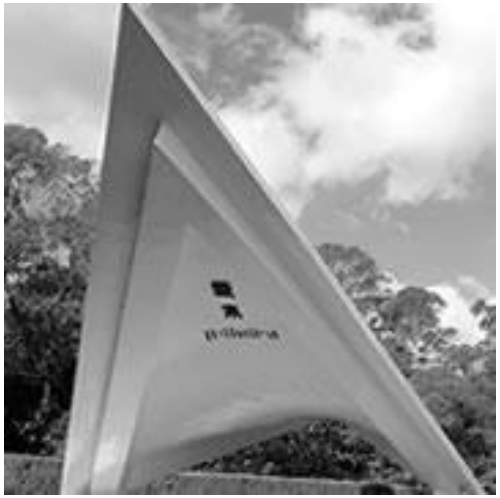
Luminance histogram

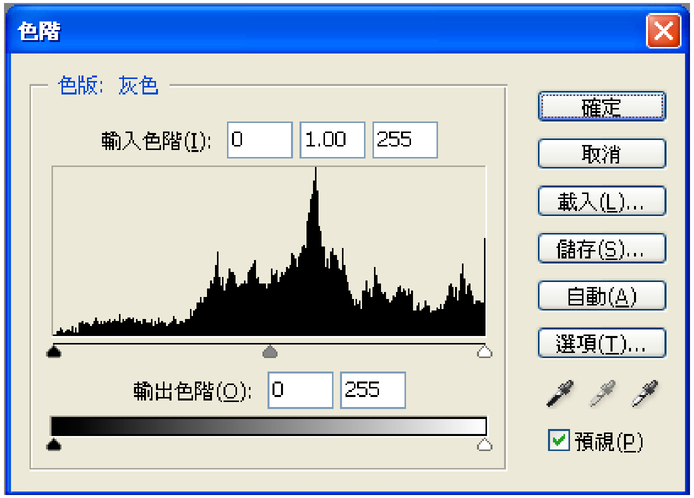
- Some scanners or image processing programs give you access to a ***luminance histogram*** (also called a ***luminosity histogram***) corresponding to a color image

$$L = 0.299R+0.587G+0.114B$$

- Among the three color channels, the human eye is most sensitive to green and least sensitive to blue

Example of Histograms





色階

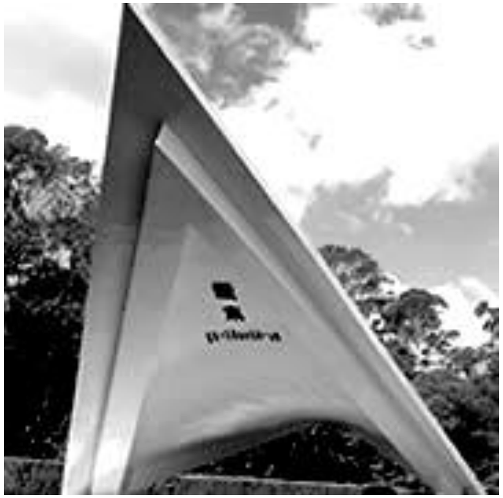
色版: 灰色

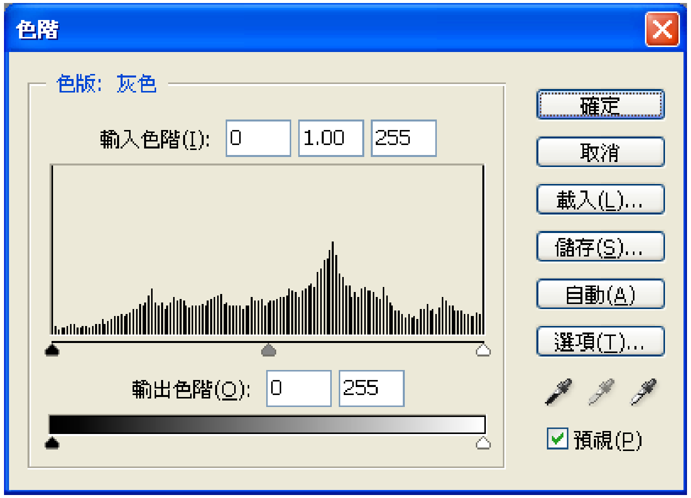
輸入色階(I): 0 1.00 255

輸出色階(O): 0 255

☑ 預視(P)

A histogram that can be manipulated to adjust brightness and contrast (from Photoshop)





色階

色版: 灰色

輸入色階(I): 0 1.00 255

輸出色階(O): 0 255

☑ 預視(P)

Histogram of image after contrast adjustment (from Photoshop)

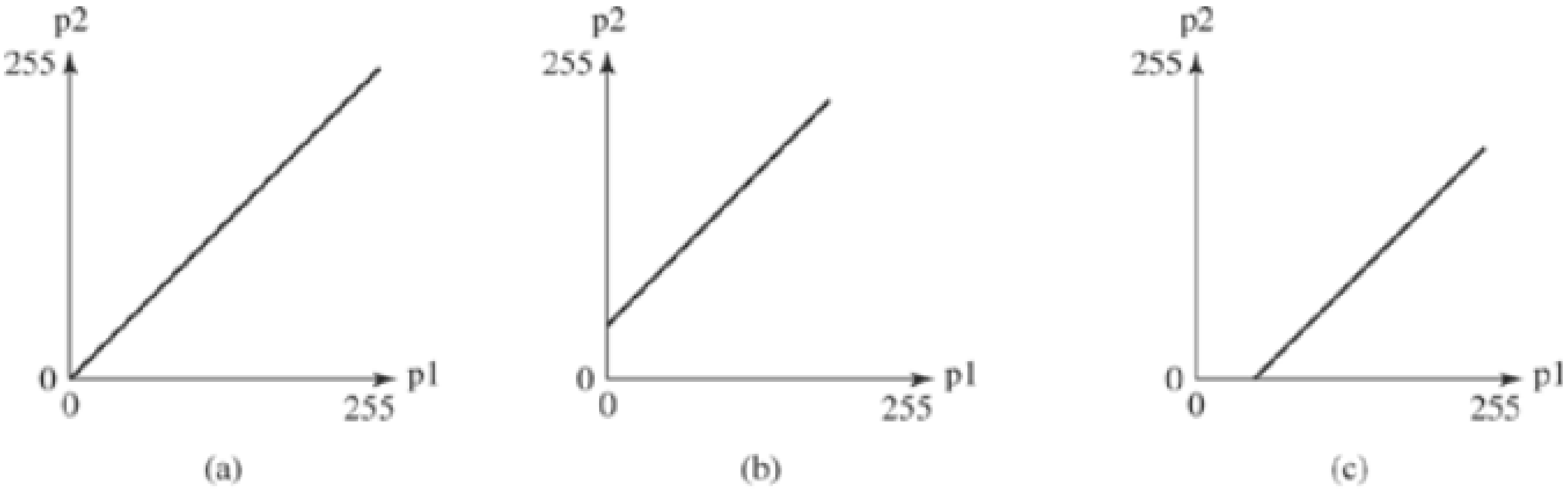
Transformation Function

- In programs like Photoshop and GIMP, the Curves feature allows you to think of the changes you make to pixel values as a transform function
- We define a ***transform*** as a function that changes pixel values

$$g(x, y) = T(f(x, y)) \text{ , } p_2 = T(p_1)$$

  - $f(x, y)$ is the pixel value at that position (x, y) in the original image. Abbreviate $f(x,y)$ as p_1
  - T is the transformation function
  - $g(x, y)$ is the transformed pixel value. Abbreviate $g(x,y)$ as p_2

Curve Representation

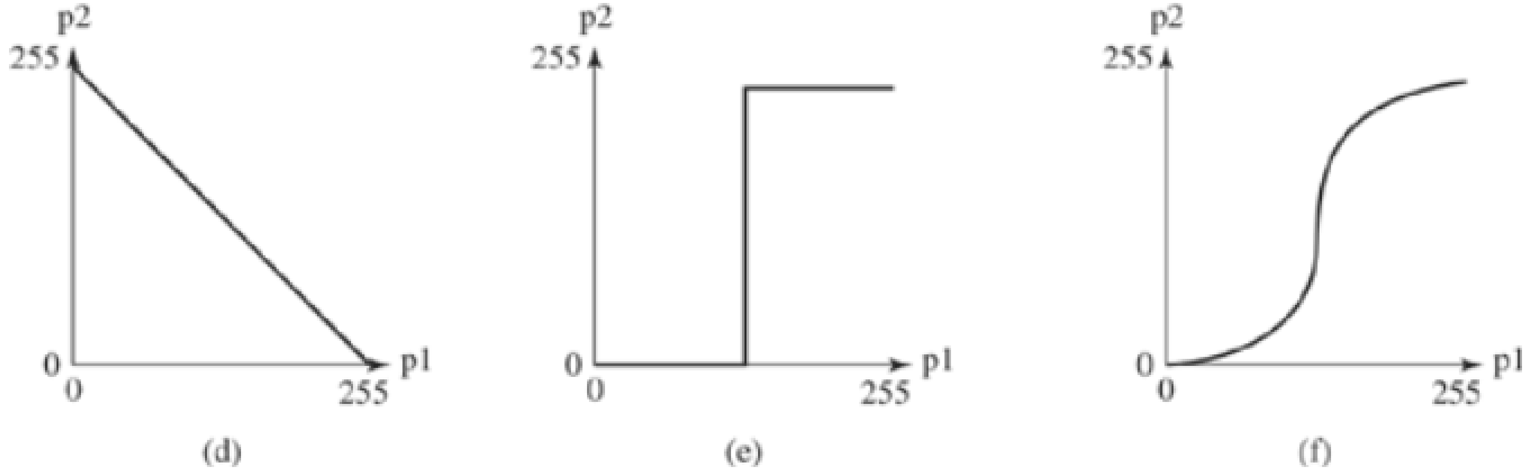


a) The transform doesn't change the pixel values. The output equals the input.

b) The transform lightens all the pixels in the image by a constant amount.

c) The transform darkens all the pixels in the image by a constant amount.

Curve Representation



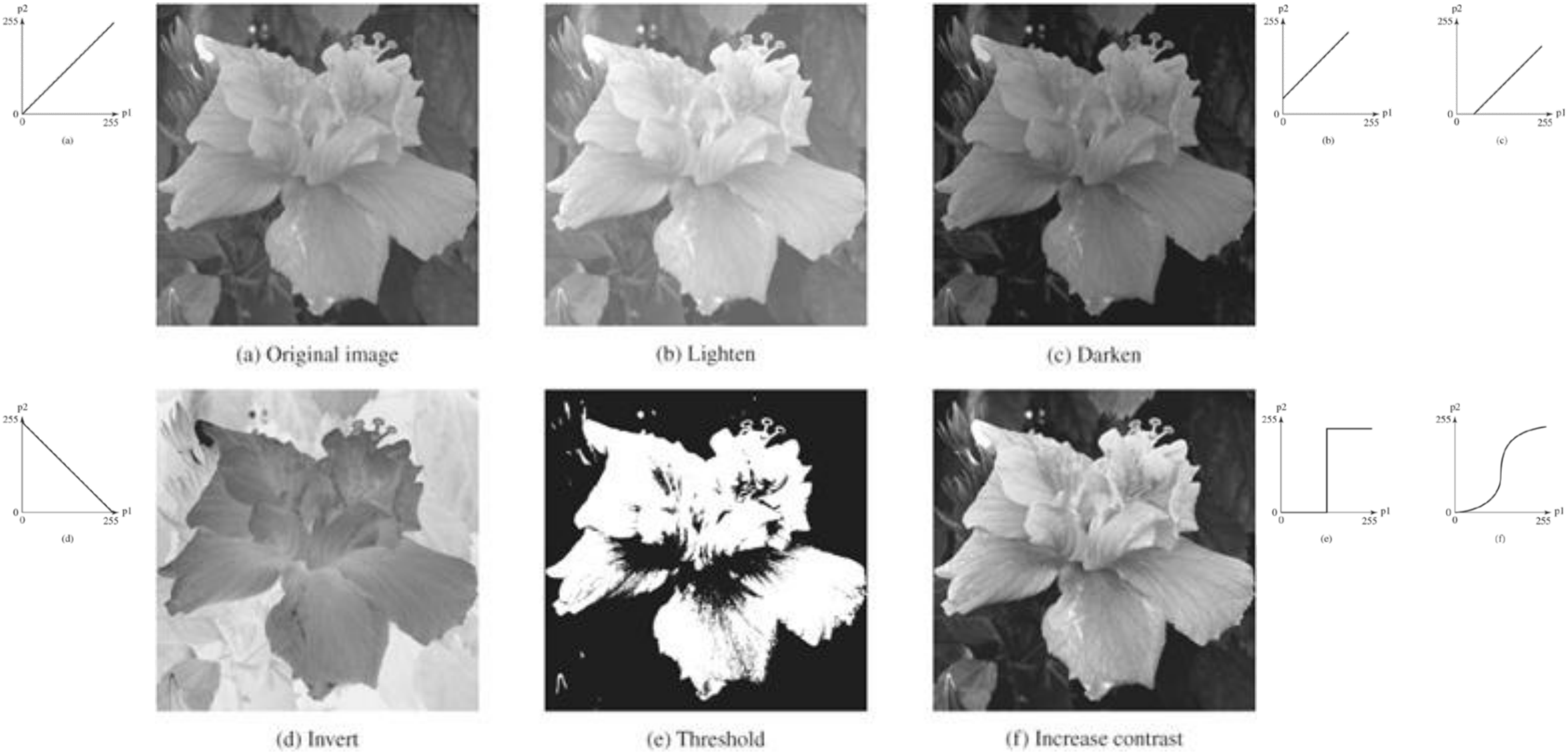
d) The transform inverts the image, reversing dark pixels for light ones.

e) The transform is a threshold function, which makes all the pixels either black or white. A pixel with a value below 128 becomes black, and all the rest become white.

f) The transform increases contrast. Darks become darker and lights become lighter.

Intensity Transformation

- Adjusting contrast and brightness with curve function



(a) Original image

(b) Lighten

(c) Darken

(d) Invert

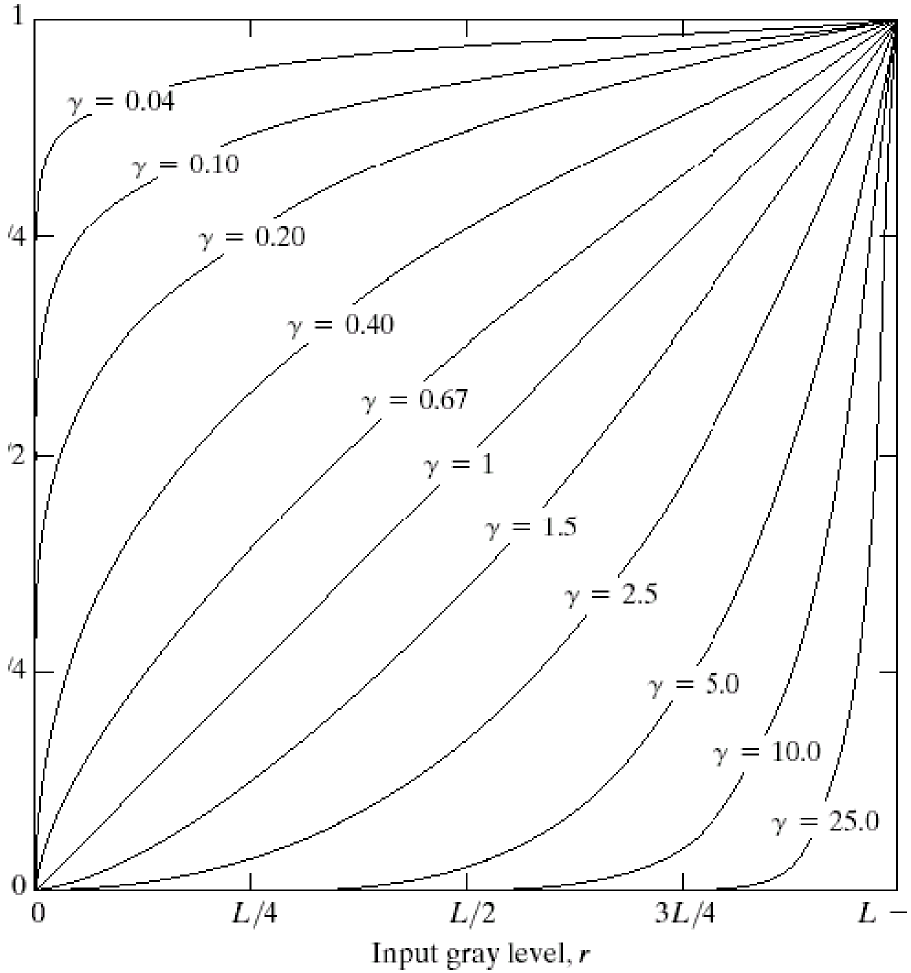
(e) Threshold

(f) Increase contrast

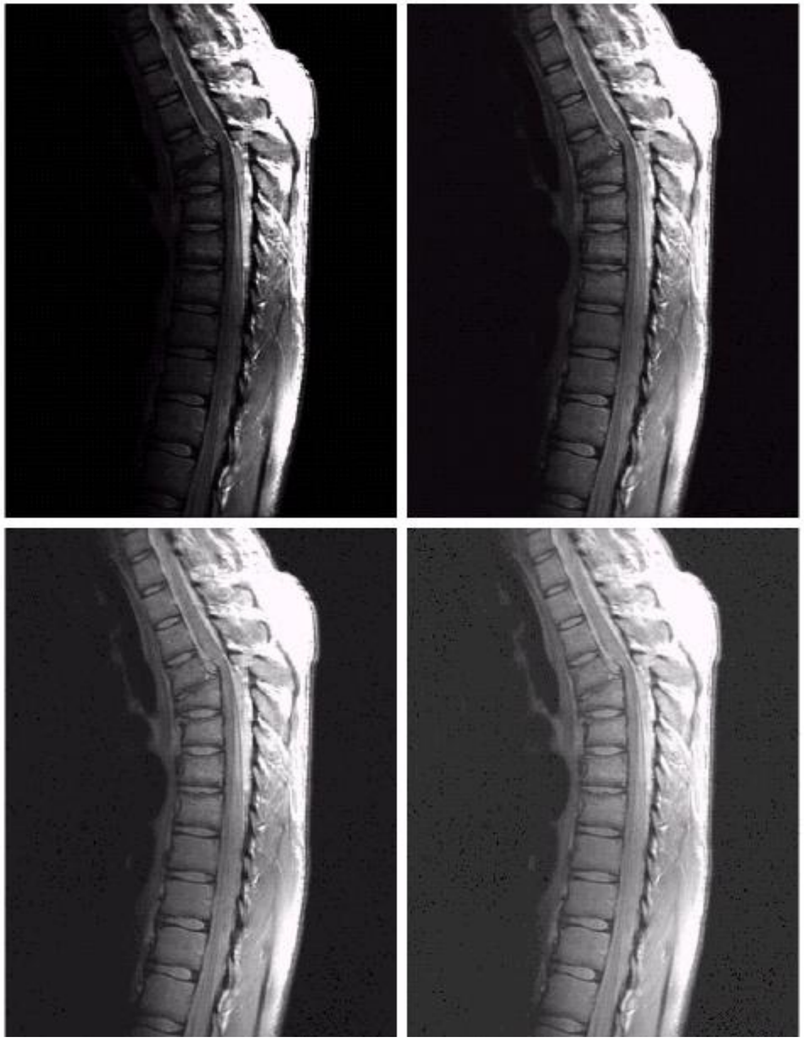
Gamma Transform

- The ***gamma value*** γ is an exponent that defines a nonlinear relationship between the input level *r* and the output level *s* for a pixel transform function.

$$s = r^{\gamma}\ , 0 \leq s \leq 1$$



Gamma Transform -Examples



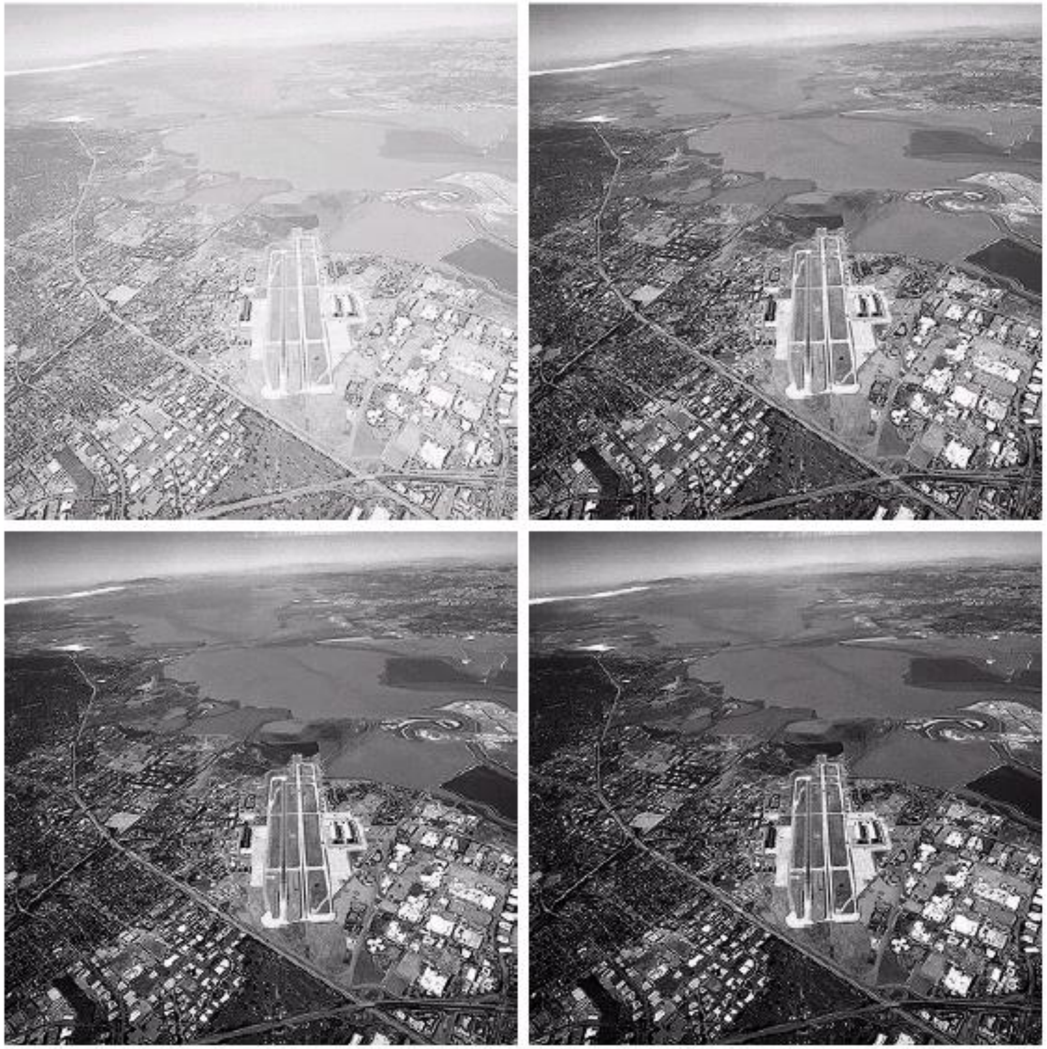
a b
c d

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.)

Gamma Transform - Examples

a b
c d

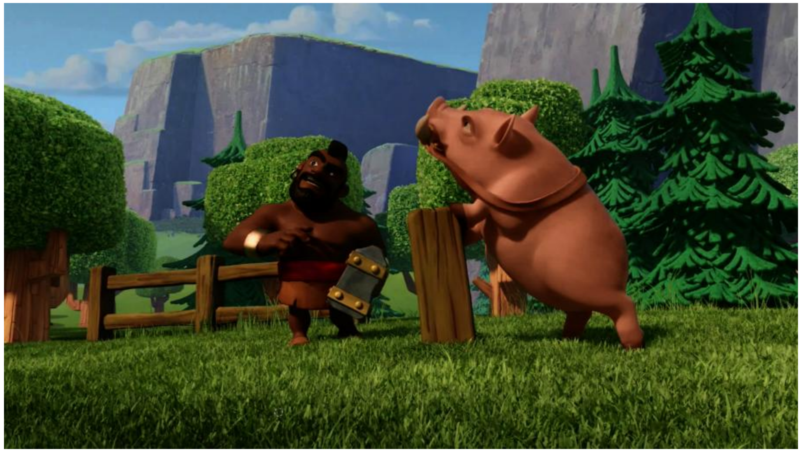
FIGURE 3.9 (a) Aerial image. (b)–(d) Results of applying the transformation in Eq. (3.2-3) with $c = 1$ and $\gamma = 3.0, 4.0,$ and 5.0, respectively. (Original image for this example courtesy of NASA.)



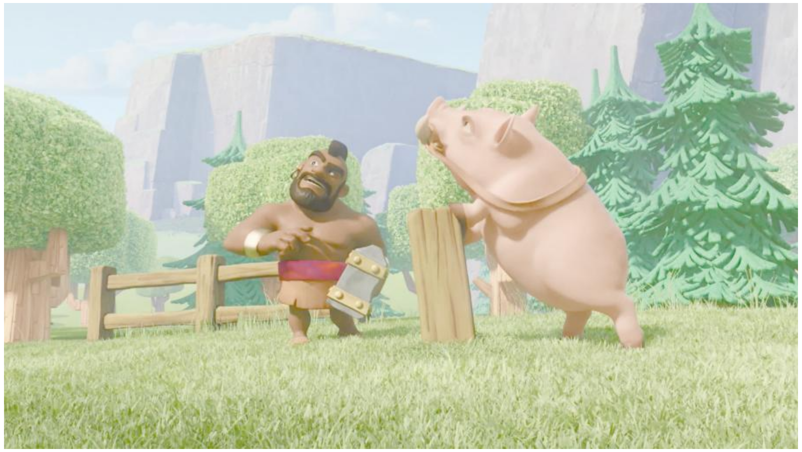
Gamma Transform -Examples

Original image

Gamma = 3



Gamma = 0.5

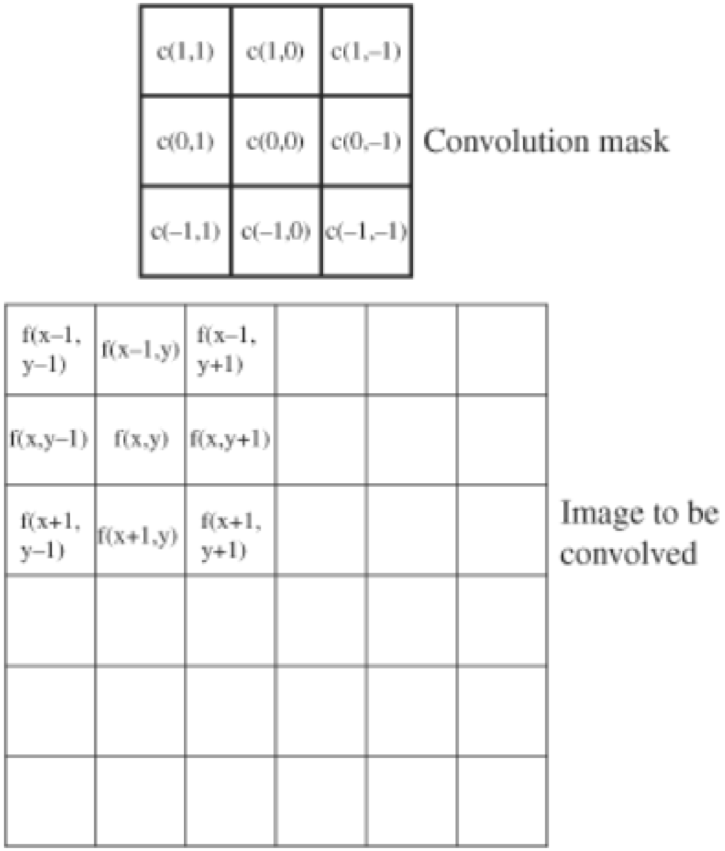


Filter

- A ***filter*** is an operation performed on digital image data to sharpen, smooth, or enhance some feature, or in some other way modify the image
- ***Filtering in the frequency domain*** is performed on image data that is represented in terms of its frequency components
- ***Filtering in the spatial domain*** is performed on image data in the form of the pixel's color values

Convolution

- Spatial filtering is done by a mathematical operation called ***convolution***, where each output pixel is computed as a weighted sum of neighboring input pixels
- Convolution is based on a matrix of coefficients called a ***convolution mask***. The mask is also sometimes called a ***filter***.



c(1,1)	c(1,0)	c(1,–1)
c(0,1)	c(0,0)	c(0,–1)
c(–1,1)	c(–1,0)	c(–1,–1)

f(x–1, y–1)	f(x–1,y)	f(x–1, y+1)			
f(x,y–1)	f(x,y)	f(x,y+1)			
f(x+1, y–1)	f(x+1,y)	f(x+1, y+1)			

1. Apply convolution mask to upper left corner of image.
2. Move mask to the right one pixel and apply again.
3. Continue applying mask to all pixels, moving left to right and top to bottom across image.

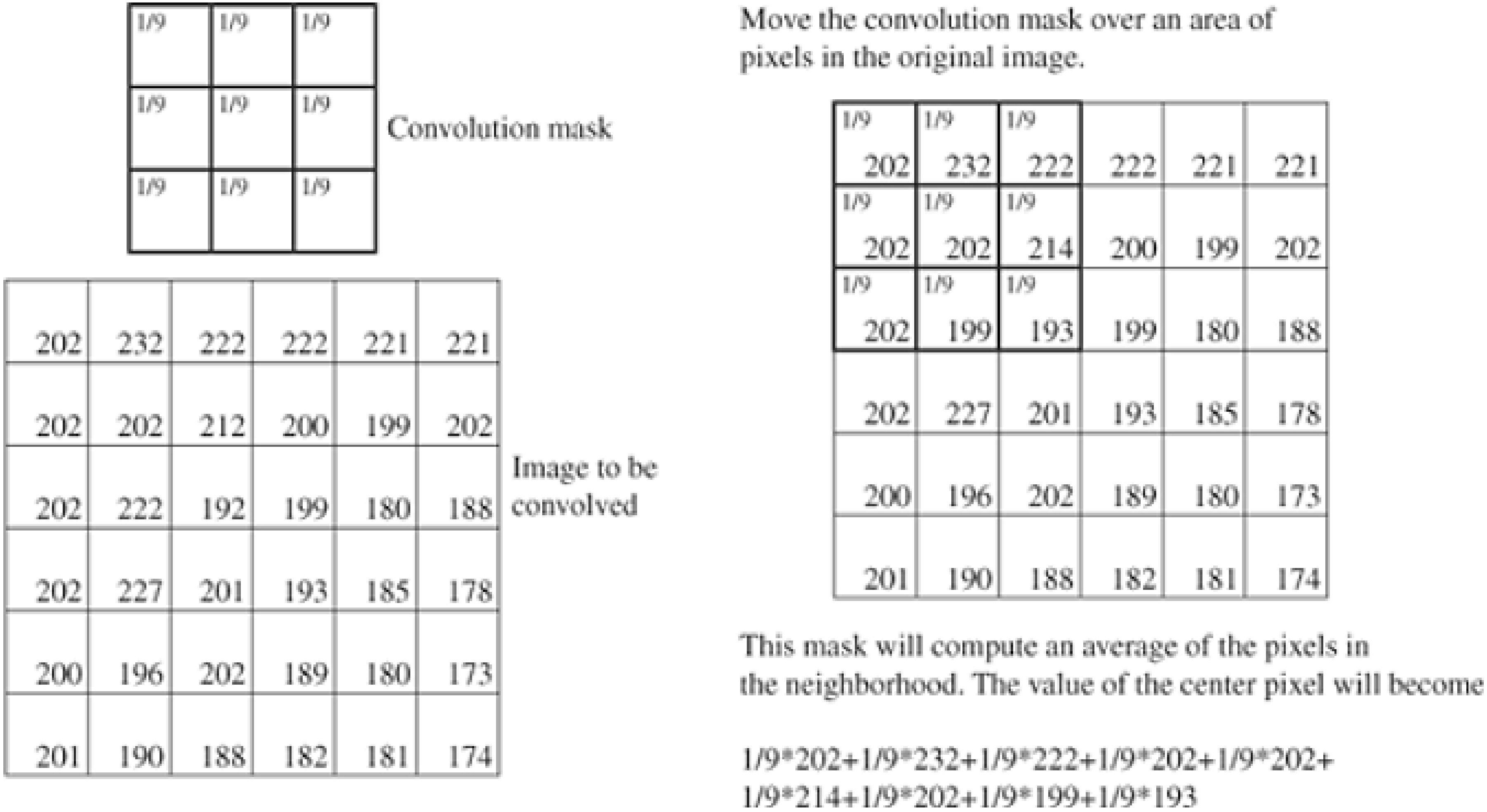
Convolution

- Let *f*(*x*, *y*) be an *M* × *N* image and *c*(*v*, *w*) be an *m* × *n* mask. Then the equation for a linear convolution is

  $$g(x,y) = \sum_{v=-i}^{i} \sum_{w=-j}^{j} c(v,w) f(x-v, y-w)$$

  where *i* = (*m* − 1)/2 and *j* = (*n* − 1)/2.
- Assume *m* and *n* are odd. This equation is applied to each pixel *f*(*x*, *y*) of an image, for 0 ≤ *x* ≤ *M* − 1 and 0 ≤ *y* ≤ *N* − 1. (If *x* − *v* < 0, *x* − *v* ≥ *M*, *y* − *w* < 0, or *y* − *w* ≥ *N*, then *f*(*x*, *y*) is undefined. These are edge cases, discussed below.)

Convolution for averaging pixels



1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Convolution mask

202	232	222	222	221	221
202	202	212	200	199	202
202	222	192	199	180	188
202	227	201	193	185	178
200	196	202	189	180	173
201	190	188	182	181	174

Image to be convolved

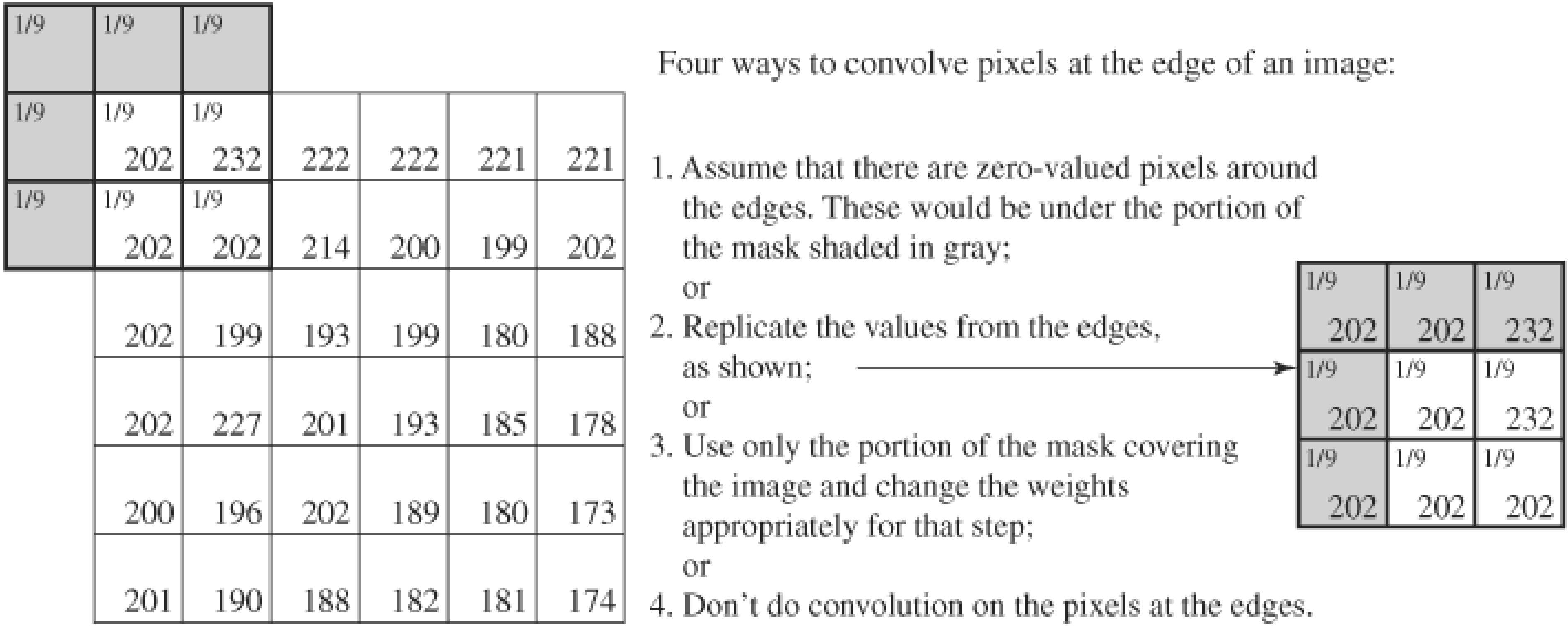
Move the convolution mask over an area of pixels in the original image.

1/9 202	1/9 232	1/9 222	222	221	221
1/9 202	1/9 202	1/9 214	200	199	202
1/9 202	1/9 199	1/9 193	199	180	188
202	227	201	193	185	178
200	196	202	189	180	173
201	190	188	182	181	174

This mask will compute an average of the pixels in the neighborhood. The value of the center pixel will become

1/9*202+1/9*232+1/9*222+1/9*202+1/9*202+
1/9*214+1/9*202+1/9*199+1/9*193

Handling edges in convolution



Four ways to convolve pixels at the edge of an image:

1. Assume that there are zero-valued pixels around the edges. These would be under the portion of the mask shaded in gray;
   or
2. Replicate the values from the edges, as shown;
   or
3. Use only the portion of the mask covering the image and change the weights appropriately for that step;
   or
4. Don't do convolution on the pixels at the edges.

1/9	1/9	1/9				
1/9	1/9 202	1/9 232	222	222	221	221
1/9	1/9 202	1/9 202	214	200	199	202
	202	199	193	199	180	188
	202	227	201	193	185	178
	200	196	202	189	180	173
	201	190	188	182	181	174

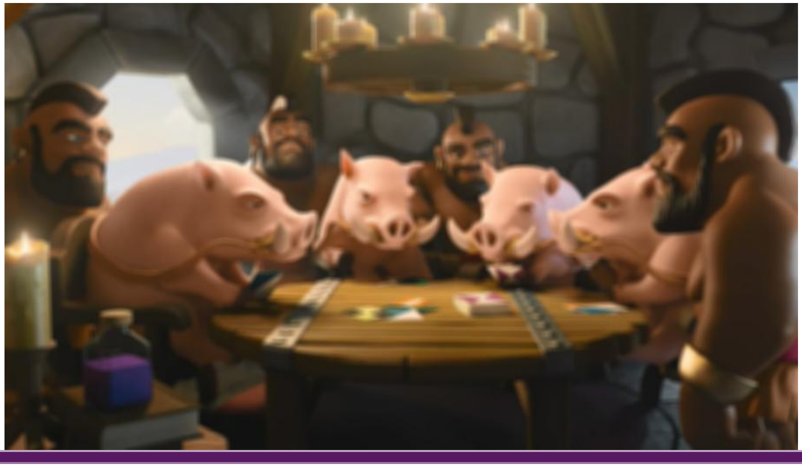
1/9 202	1/9 202	1/9 232
1/9 202	1/9 202	1/9 232
1/9 202	1/9 202	1/9 202

Average filter

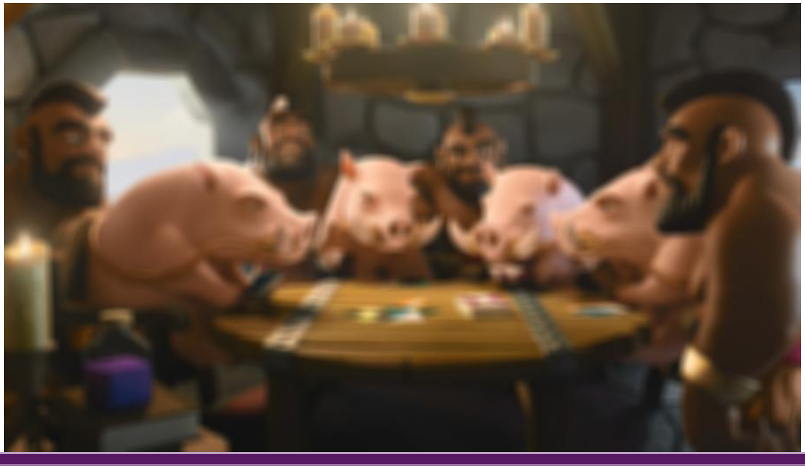
- Output pixel is the mean of its kernel neighbors.
- Here shows the effects of average filters with different mask size.

Original image

Mask size: 15



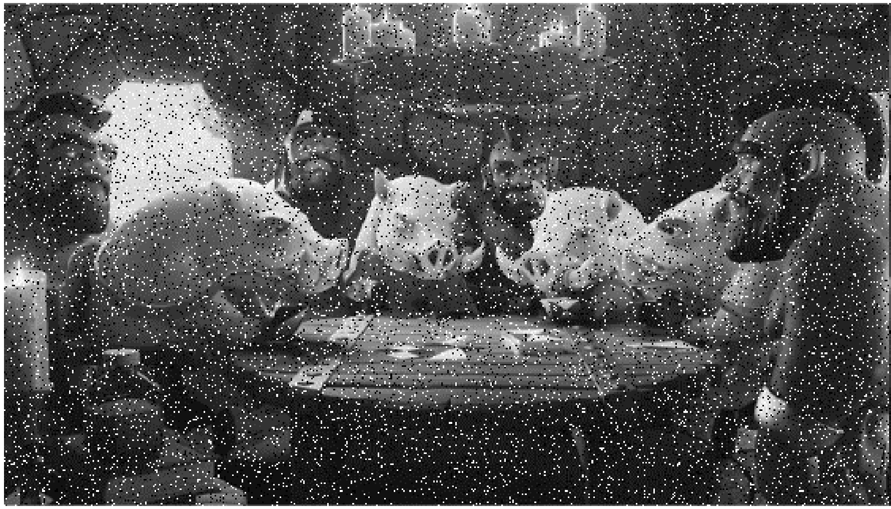
Mask size: 25



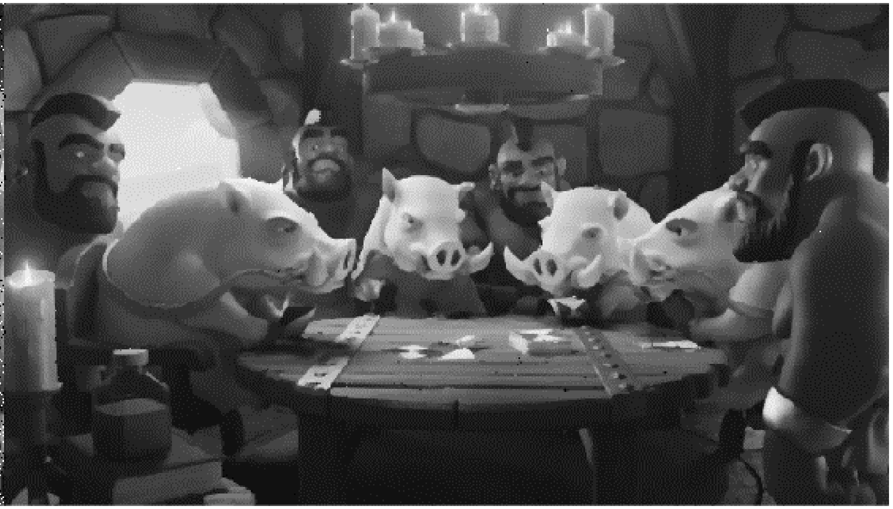
Median filter

- Output pixel is the median of its neighboring pixels in mask
- Median filter is able to perform salt-and-pepper noise reduction.

Original image



After applying median filter



Unsharp mask

- The name is misleading because this filter actually sharpens images
- The pixel values in the original image are doubled, and the blurred version of the image is subtracted from this

	2	

–

	1	
1	–3	1
	1	

=

	–1	
–1	5	–1
	–1	

2*original – Blur mask = Unsharp mask

Original image

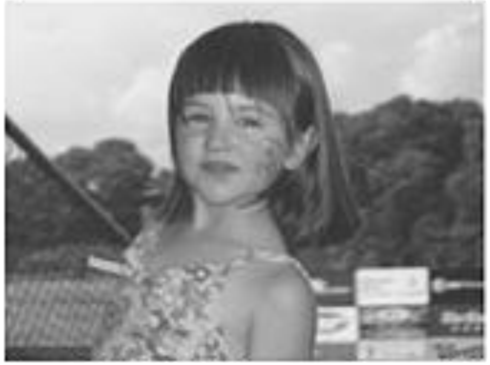


Image with blur filter applied

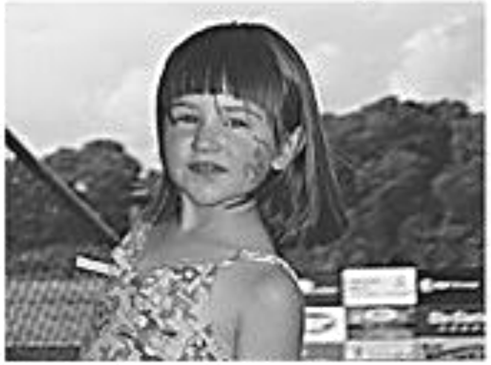


Image with unsharp mask applied

Edge-detection filter

- Sobel filter detects the edge, making that edge white while everything else is black

1	1	1
0	0	0
–1	–1	–1

Mask

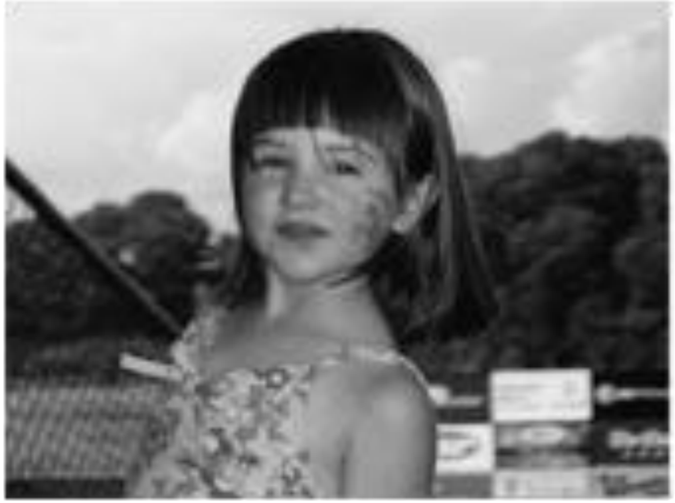
255	255	255	255
255	255	255	255
255	255	255	255
0	0	0	0
0	0	0	0
0	0	0	0

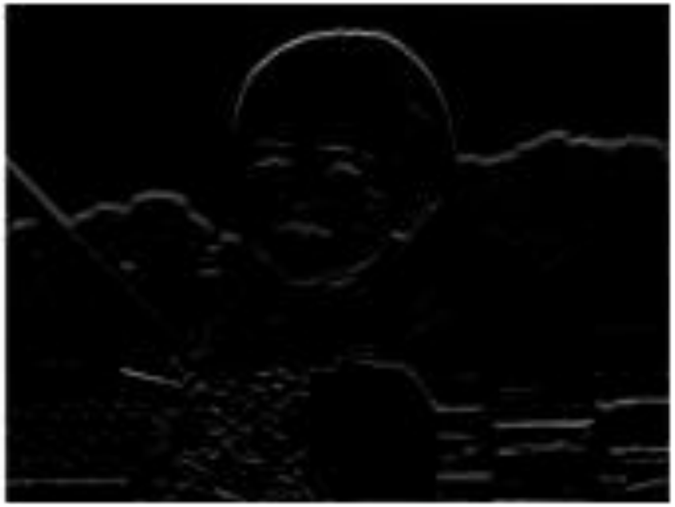
Block **a**

0	0	0	0
0	0	0	0
255	255	255	255
255	255	255	255
0	0	0	0
0	0	0	0

Block **b**

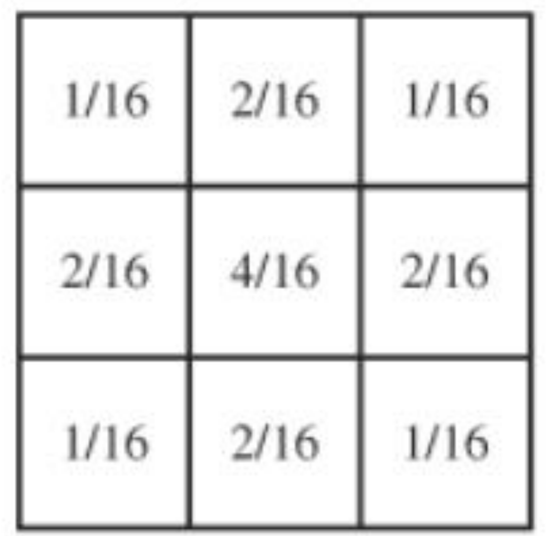
The mask above applied to block **a** of pixels yields block **b**.





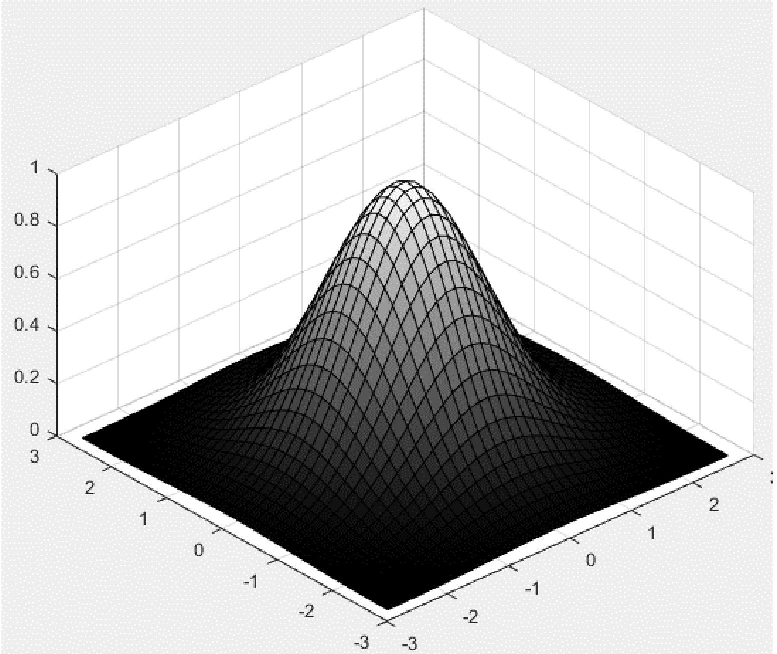
Gaussian filter

- An alternative for smoothing is to use a ***Gaussian filter***, where the coefficients in the convolution mask get smaller as you move away from the center of the mask



1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

Gaussian convolution mask



Summary

- Image representation
- Basic image processing
  - Dithering
  - Interpolation
  - Intensity transformation
  - Image convolution