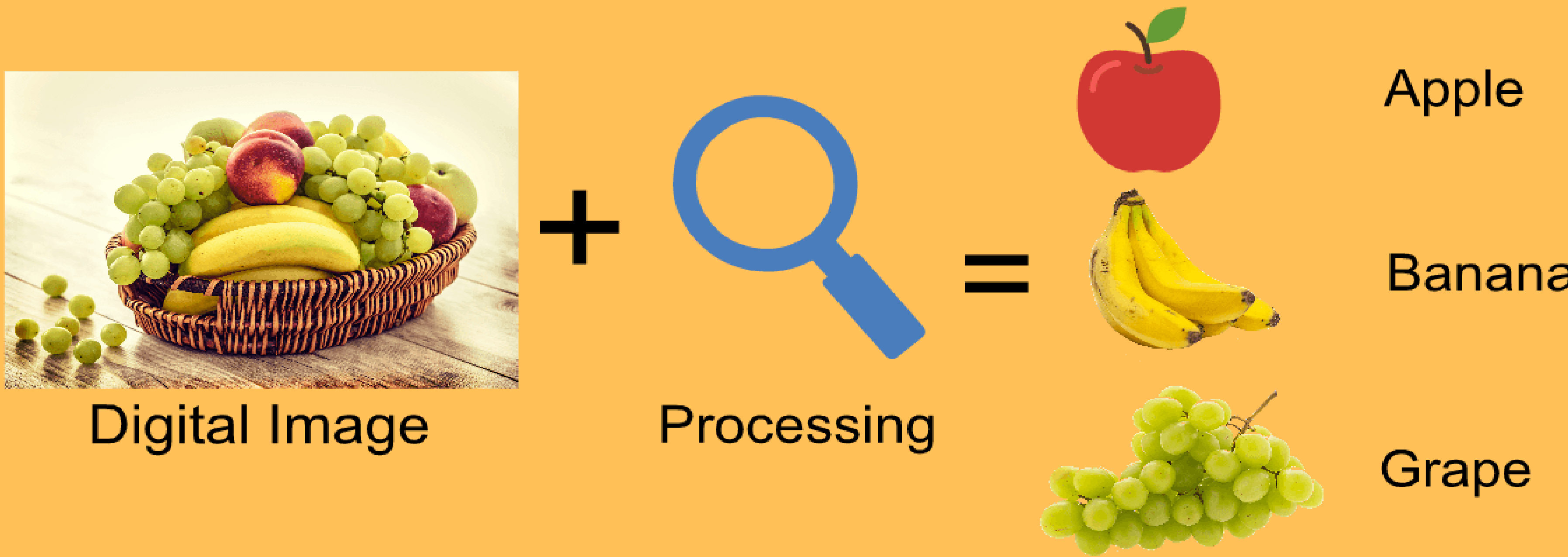
Digital Image Processing



Digital Image Processing

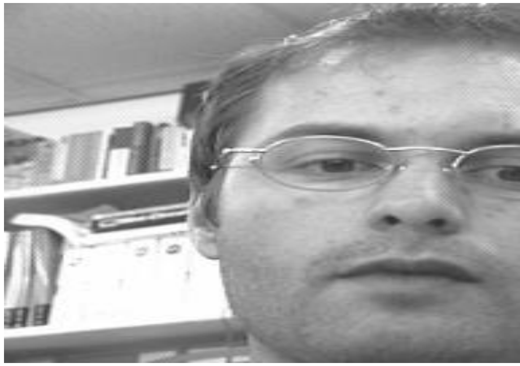
Lec. 1: Introduction2

Assist. Prof. Dr. Saad Albawi

Quantization Examples

- Bi-level (black & white) image (fax)
  - $s = 0$ or 1
- 8-bit color image (photograph)
  - $0 \leq r,g,b \leq 255$
- 10-bit color image (movie)
  - $0 \leq r,g,b \leq 1023$
- 12-bit intensity image (X-ray)
  - $0 \leq s \leq 4095$
- Multi-spectral image (satellite)
  - $0 \leq c1,c2,\ldots,c7 \leq 255$

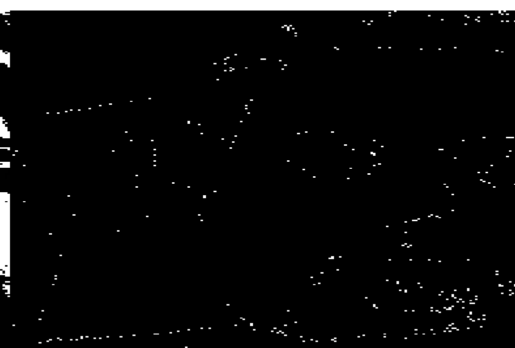
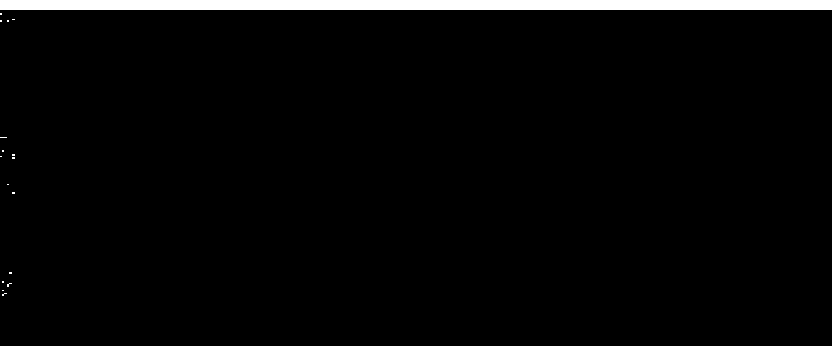
Effect of grey level resolution



8 bits | 7 bits | 6 bits



5 bits | 4 bits | 3 bits

2 bits | 1 bit | 0 bits !!!

Effect of reducing the gray-level resolution

Decreasing the gray-level resolution of a digital image may result in what is known as *false contouring*. This effect is caused by the use of an insufficient number of gray levels in smooth areas of a digital image.

To illustrate the false contouring effect, we reduce the number of gray levels of the 256-level image shown in Figure 2.6(a) from 256 to 2. The resulted images are shown in the figures 2.6(b) through (h). This can be achieved by reducing the number of bits from $k = 7$ to $k = 1$ while keeping the spatial resolution constant at 452×374 pixels.

We can clearly see that the 256-, 128-, and 64-level images are visually identical. However, the 32-level image shown in Figure 2.6(d) has an almost imperceptible set of very fine ridgelike structures in areas of smooth gray levels (particularly in the skull).False contouring generally is quite visible in images displayed using 16 or less uniformly spaced gray levels, as the images in Figures 2.6(e) through (h) show.

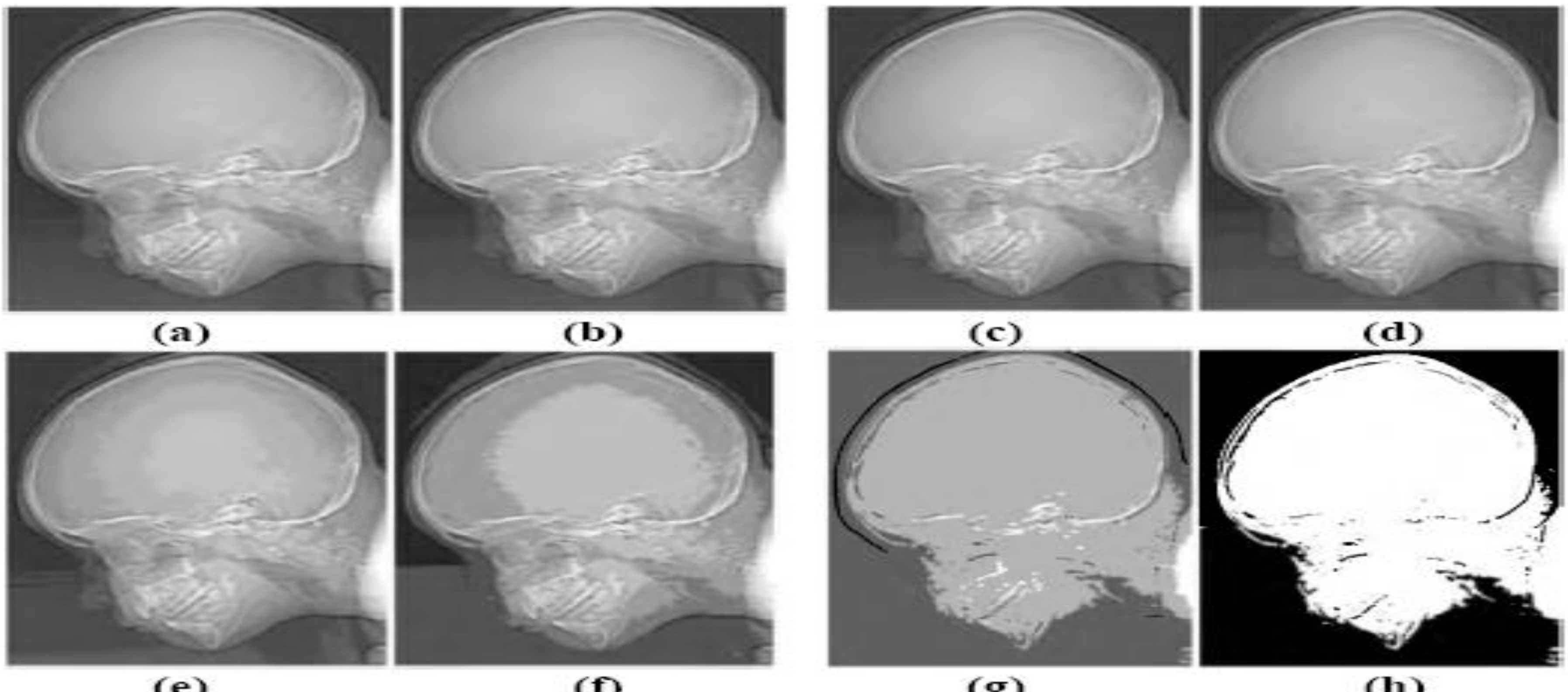


Figure 2.6 (a) 452×374, 256-level image. (b)-(h) Image displayed in 128, 64, 32, 16, 8, 4, and 2 gray levels, while keeping the spatial resolution constant.

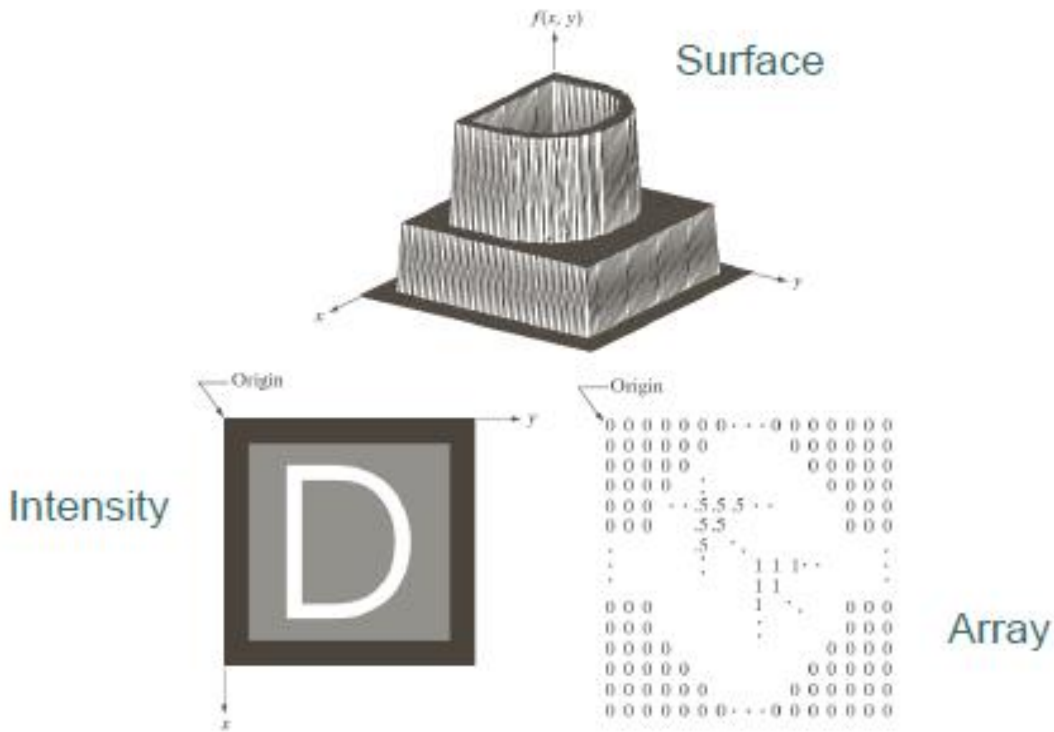
Images as Functions

- We can think of an **image** as a function, *f*, from R^2 to R:
  - $f(x, y)$ gives the **intensity** at position (x, y)
  - Realistically, we expect the image only to be defined over a rectangle, with a finite range:
    - $f: [a,b] \times [c,d] \rightarrow [0,1]$

- A color image is just three functions pasted together. We can write this as a "vector-valued" function:

$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

Image Representation



Intensity Representation

NB: There is no universally accepted convention or notation. Always check carefully!

Array Representation

- Mathematical notation

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

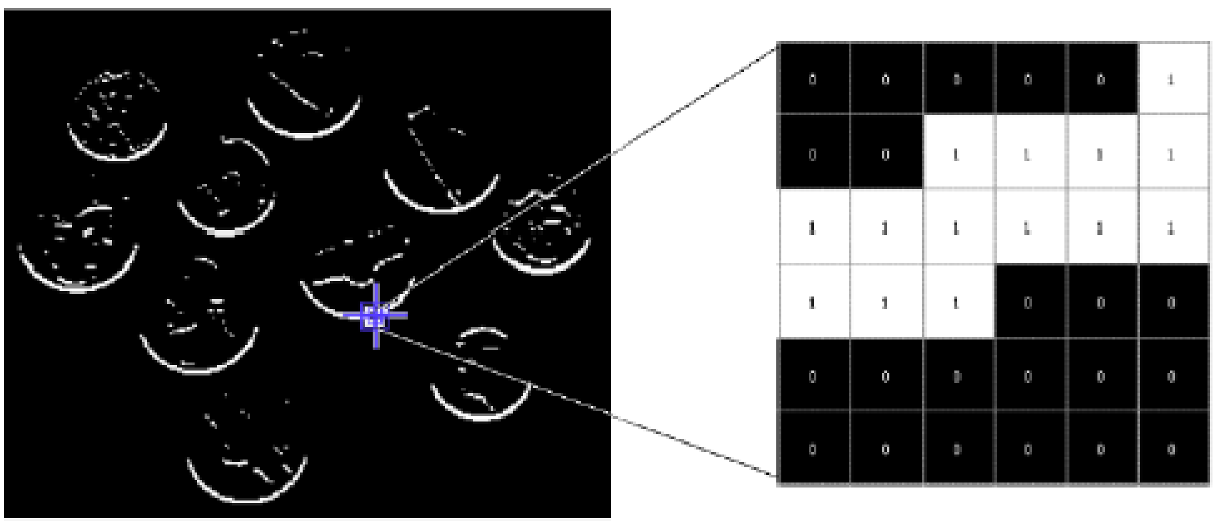
Array Representation

- MATLAB notation

$$\texttt{f(p,q)} = \begin{bmatrix} \texttt{f(1,1)} & \texttt{f(1,2)} & \cdots & \texttt{f(1,N)} \\ \texttt{f(2,1)} & \texttt{f(2,2)} & \cdots & \texttt{f(2,N)} \\ \vdots & \vdots & & \vdots \\ \texttt{f(M,1)} & \texttt{f(M,2)} & \cdots & \texttt{f(M,N)} \end{bmatrix}$$

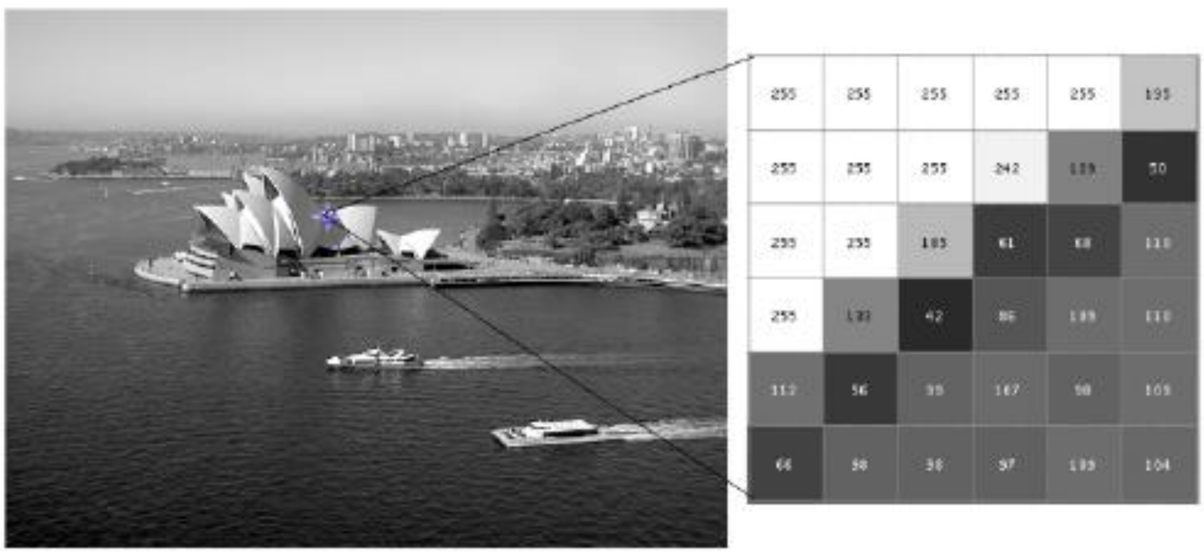
Digital image representation

- Binary (1-bit) images
  - 2D array, one bit per pixel, a 0 *usually* means "black" and a 1 means "white".
  - In MATLAB: binary images are represented using a `logical` array of 0s and 1s.



Digital image representation

- Gray-level (8-bit) images
  - 2D array, 8 bits per pixel, a 0 usually means "black" and a 255 means "white".
  - In MATLAB: intensity images can be represented using different data types (or classes): **`uint8`**, **`uint16`**, or



What is digital image?

The image consists of finite number of pixels (f(x,y))

Every pixel Is an intersection تقاطع between a row and a column.

every pixel has intensity كثافة

pixel

☝ **Ex:**

f(4,3)= 123

Refers to a pixel existing on the intersection between row 4 with column 3, and its intensity is 123.

Digital image representation

- Color images
  - **RGB representation**: each pixel is usually represented by a 24-bit number containing the amount of its Red (R), Green (G), and Blue (B) components (8 bits per component)
  - **Indexed representation**: a 2D array contains indices to a color palette (or look-up table, LUT).

Digital image representation

(a) 24-bit (true color) RGB image

(b) R

(c) G

(d) B

2^24=

16M colors

(a)

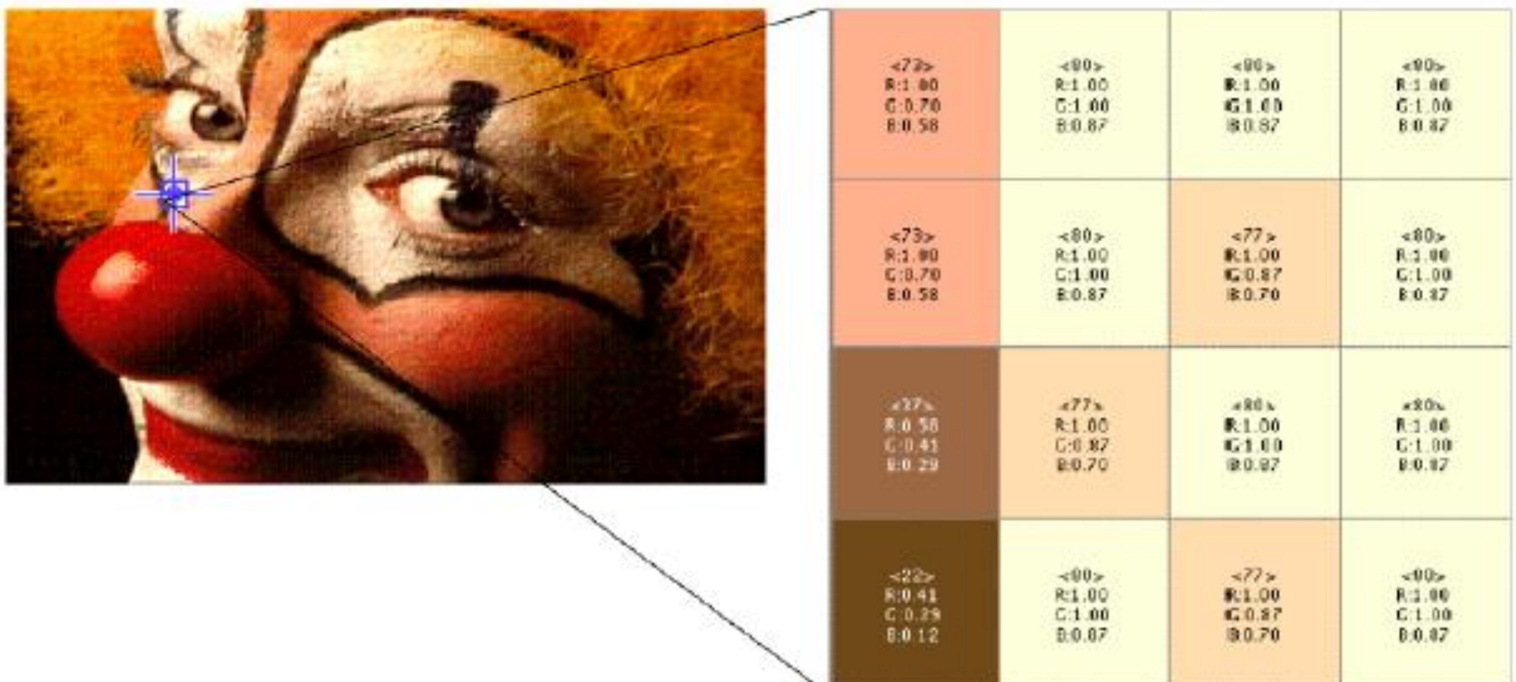
(b)

(c)

(d)

Digital image representation

- Indexed color images: old hardware can not display 16M colors. Use a pointer to a color palette (color map) . Typically 256 colors.



Mathematical representation of Digital Images

There are two categories of algebraic operations applied to images:

- Arithmetic
- Logic

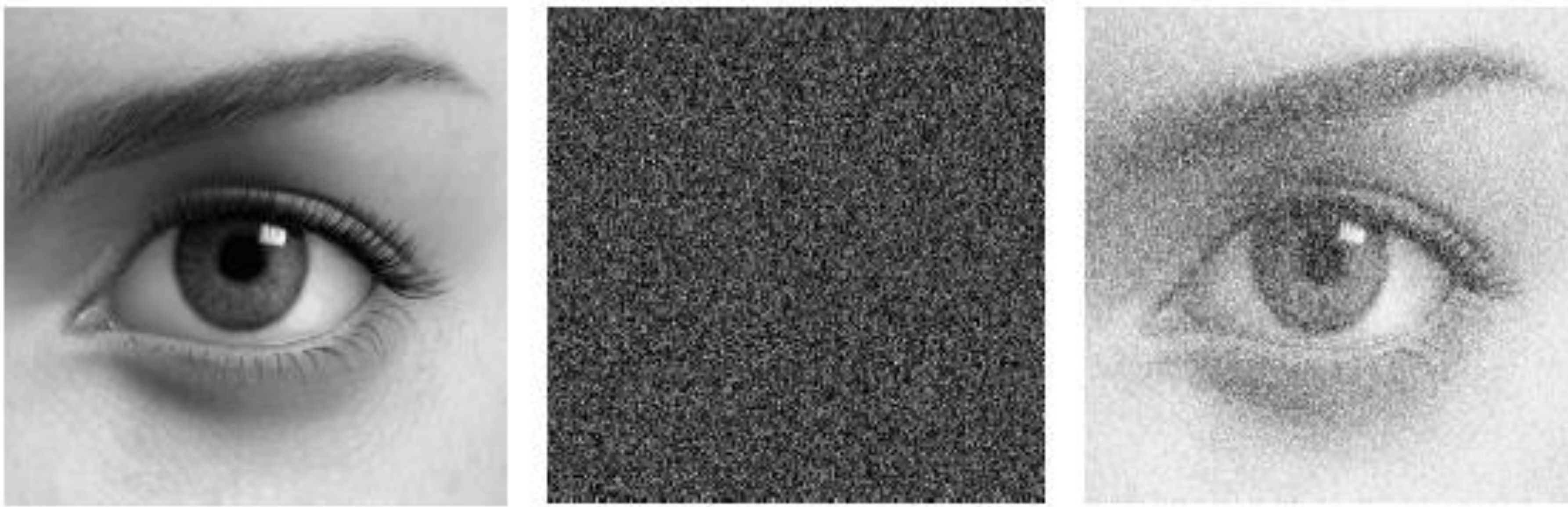
These operations are performed on a pixel-by-pixel basis between two or more images, except for the NOT logic operation which requires only one image. For example, to add images I_1 and I_2 to create I_3:

$$I_3(x,y) = I_1(x,y) + I_2(x,y)$$

$$I_1 = \begin{bmatrix} 3 & 4 & 7 \\ 3 & 4 & 5 \\ 2 & 4 & 6 \end{bmatrix} \qquad I_2 = \begin{bmatrix} 6 & 6 & 6 \\ 4 & 2 & 6 \\ 3 & 5 & 5 \end{bmatrix}$$

$$I_3 = \begin{bmatrix} 3+6 & 4+6 & 7+6 \\ 3+4 & 4+2 & 5+6 \\ 2+3 & 4+5 & 6+5 \end{bmatrix} = \begin{bmatrix} 9 & 10 & 13 \\ 7 & 6 & 11 \\ 5 & 9 & 11 \end{bmatrix}$$

- **Addition** is used to combine the information in two images. Applications include development of image restoration algorithms for modeling additive noise and special effects such as image morphing in motion pictures as shown in the figures below.



(a) Original image (b) Gaussian noise (c) Addition of images (a) and (b)

Image addition (adding noise to the image)

(a) First Original

(b) Second Original

(c) Addition of images (a) and (b)

Figure 3.3 Image addition (image morphing example)

- **Subtraction** of two images is often used to detect motion. For example, in a scene when nothing has changed, the image resulting from the subtraction is filled with zeros(black image). If something has changed in the scene, subtraction produces a nonzero result at the location of movement as shown in the figure below.

(a) Original scene

(b) Same scene at a later time

(c) Subtracting image (b) from (a). Only moving objects appear in the resulting image

Image subtraction

- **Multiplication and division** are used to adjust the brightness of an image. Multiplying the pixel values by a number greater than one will brighten the image, and dividing the pixel values by a factor greater than one will darken the image. An example of brightness adjustment is shown in the figure below.

(a) Original image (b) Image multiplied by 2 (c) Image divided by 2

Image multiplication and division

Logical Operations

The logic operations AND, OR, and NOT form a complete set, meaning that any other logic operation (XOR, NOR, NAND) can be created by a combination of these basic elements. They operate in a bit-wise fashion on pixel data.

The AND and OR operations are used to perform *masking* operation; that is; for selecting subimages in an image, as shown in the figure below. Masking is also called *Region of Interest (ROI)* processing.

(a) Original image (b) AND image mask (c) Resulting image, (a) AND (b)

(d) Original image (e) OR image mask (f) Resulting image, (d) OR (e)

Image masking

The NOT operation creates a negative of the original image ,as shown in the figure below. by inverting each bit within each pixel value.

(a) Original image (b) NOT operator applied to image (a)

Complement image

Image files Format

Image files consists of two parts:

- A ***header*** found at the start of the file and consisting of parameters regarding:
  - ✓ Number of rows (height)
  - ✓ Number of columns (width)
  - ✓ Number of bands (i.e. colors)
  - ✓ Number of bits per pixel (bpp)
  - ✓ File type
- Image ***data*** which lists all pixel values (vectors) on the first row, followed by 2nd row, and so on.

Digital Image File Formats

Types of image data are divided into two primary categories: bitmap and vector.

- *Bitmap images* (also called raster images) can be represented as 2-dimensional functions *f(x,y)*, where they have pixel data and the corresponding gray-level values stored in some file format.
- *Vector images* refer to methods of representing lines, curves, and shapes by storing only the key points. These key points are sufficient to define the shapes. The process of turning these into an

Most of the types of file formats fall into the category of bitmap images, for example:

- PPM (Portable Pix Map) format
- TIFF (Tagged Image File Format)
- GIF (Graphics Interchange Format)
- JPEG (Joint Photographic Experts Group) format
- BMP (Windows Bitmap)
- PNG (Portable Network Graphics)
- XWD (X Window Dump)

Image Histogram

- The distribution of gray levels in an image convey some useful information on the image content.

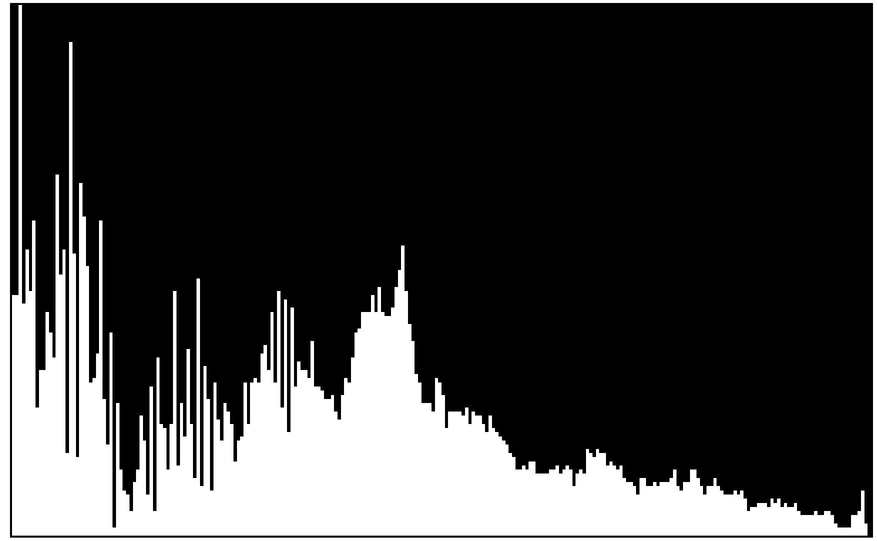
- For any image f of size mxn and Gray Level resolution k, the histogram of h is a discrete function defined on the set {0, 1, …, 2^k-1} of gray values such that h(i) is th number of pixels in the image f which have the gray value i.

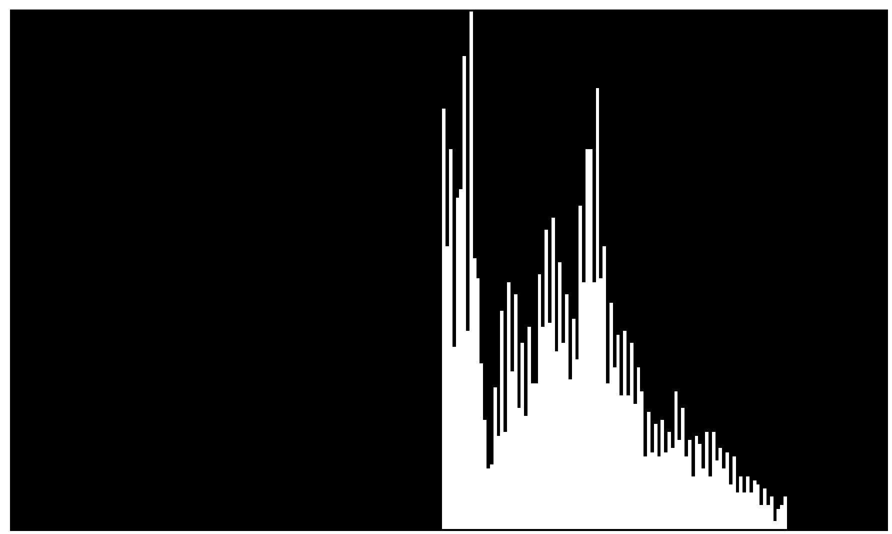
- It is customary to "normalise" a histogram by dividing h(i) by the total number of pixels in the image, i.e. use the probability distribution:

$$p(i) = h(i)/mn.$$

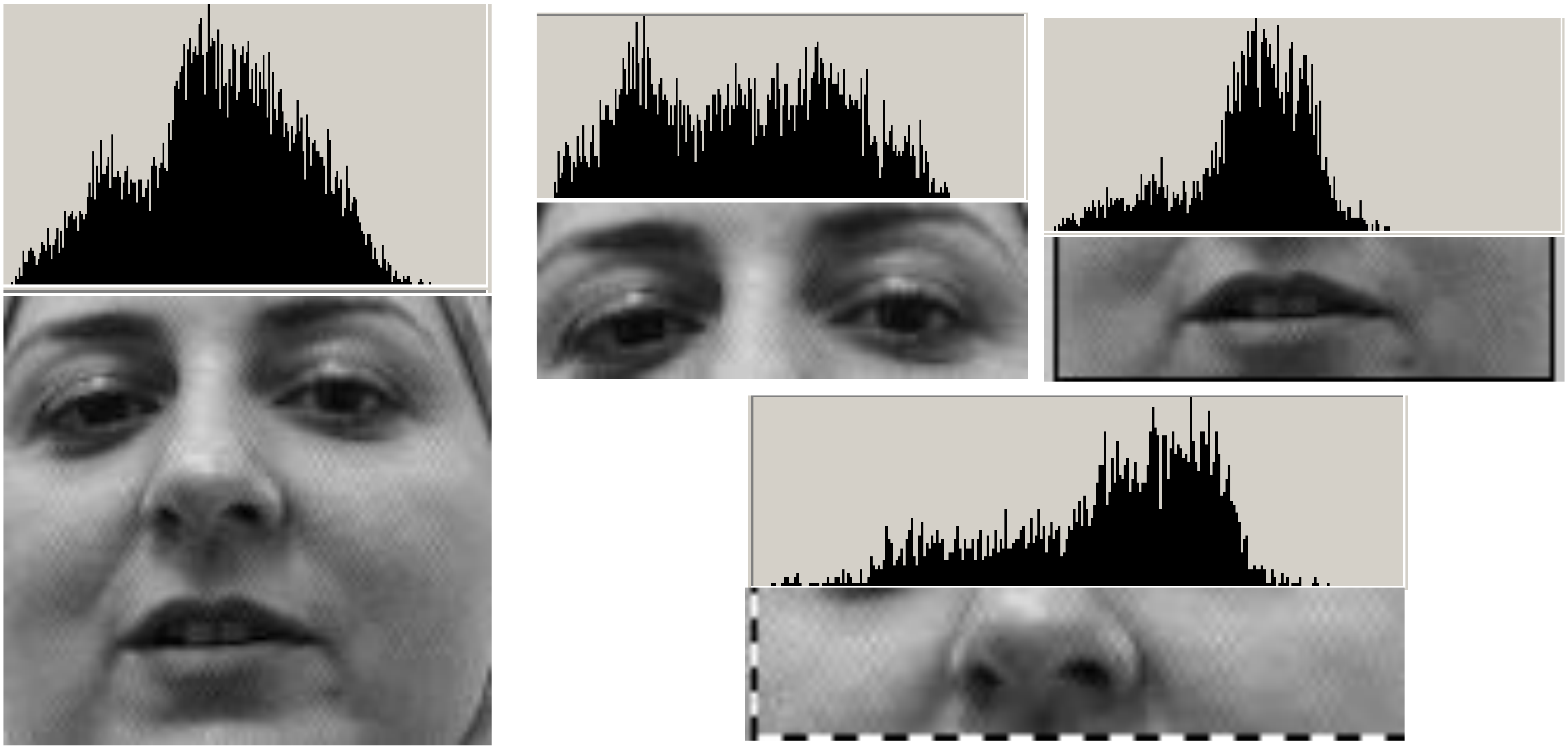
- Histograms are used in numerous processing operations.

Histograms - Examples





Local Vs. Global Histograms – Image Features



- ❑ Histograms for parts of an image provide useful tools for feature analysis.
- ❑ Local Histograms provide more information on image content than the global histogram.

Compression

- Most image file formats employ some type of compression.
- Compression methods can be:
  - **Lossy**: a tolerable degree of deterioration is introduced in visual quality.
  - **Lossless**: image is encoded in its full quality.
- As a general guideline:
  - **Lossy** compression should be used for general purpose photographic images
  - **Lossless** compression should be used for images in which no loss of detail may be tolerable (e.g., space images and medical images).

Image Aspect Ratio

Height in cm (H)

Width in cm (W)

Image Aspect Ratio = image width/image height = W/H

Pixel Aspect Ratio

Number of lines (N)

Number of pixels per line (M)

Pixel Aspect Ratio = pixel width/pixel height = WN/HM

Overview of Image Processing Operations

• ***Operations in the Spatial Domain*: Here, arithmetic calculations and/or logical operations are performed on the original pixel values. They can be further divided into three types.**

Operations in a Transform Domain: Here, the image undergoes a mathematical transformation—such as Fourier transform (FT) or discrete cosine transform (DCT)—and the image processing algorithm works in the transform domain. Example: frequency-domain filtering techniques

Global (Point) Operations

Point operations apply the same mathematical function, often called *transformation function*, to all pixels, regardless of their location in the image or the values of their neighbors. Transformation functions in the spatial domain can be expressed as

$$g(x, y) = T$$

$$f\,(x, y)$$

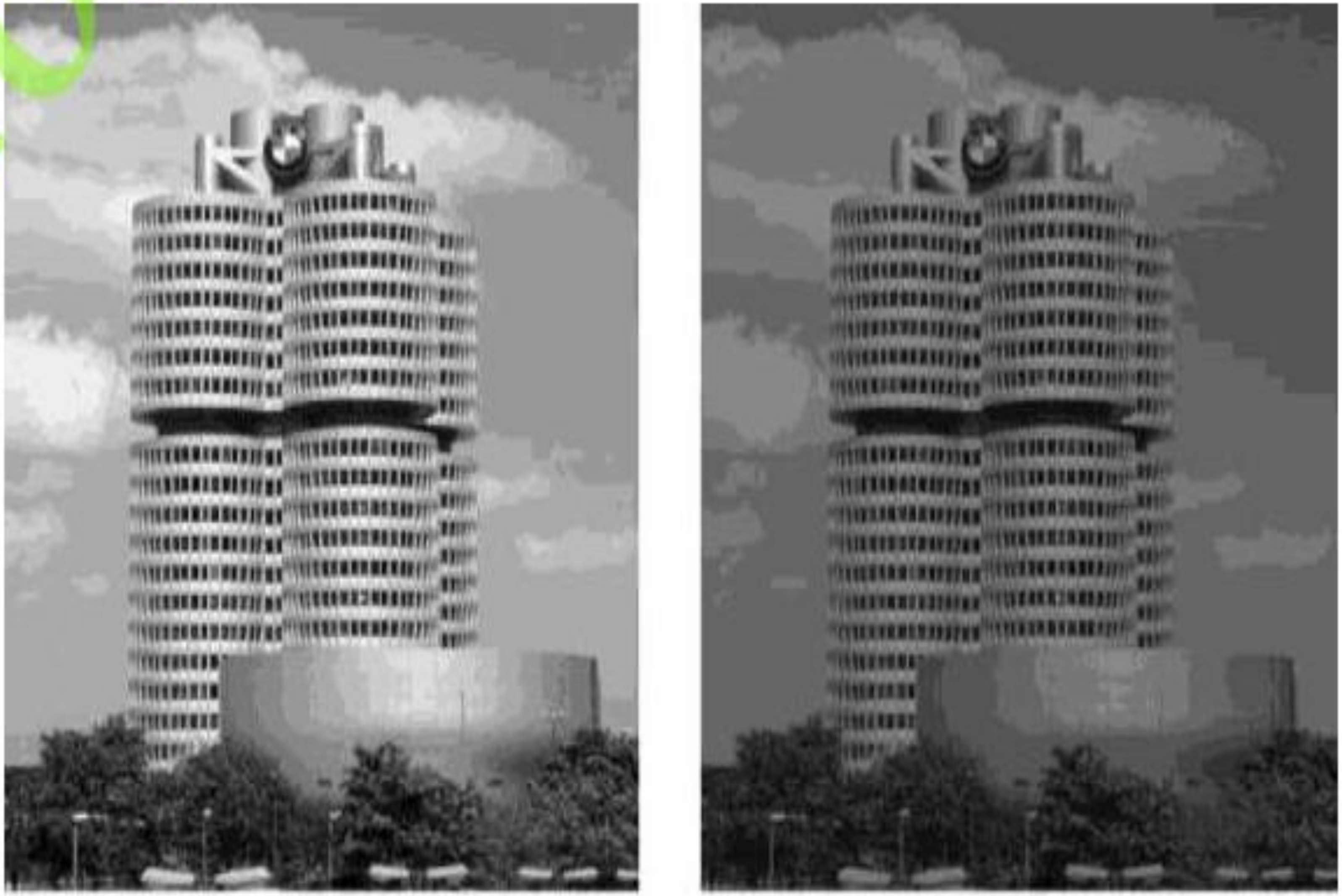
where $g(x, y)$ is the processed image, $f\,(x, y)$ is the original image, and T is an operator on $f\,(x, y)$.

Since the actual coordinates do not play any role in the way the transformation function processes the original image, a shorthand notation can be used:

$$s = T\,[r]$$

where r is the original gray level and s is the resulting gray level after processing. Figure bellow shows an example of a transformation function used to reduce the overall

intensity of an image by half: $s = r/2$. Chapter 8 will discuss point operations and transformation functions in more detail.



(a) (b)

FIGURE 2.9 Example of intensity reduction using a transformation function: (a) original image; (b) output image.

Neighborhood-Oriented Operations

Neighborhood-oriented (also known as *local* or *area*) operations consist of determining the resulting pixel value at coordinates (x, y) as a function of its original value and the value of (some of) its neighbors, typically using a *convolution* operation. The convolution of a source image with a small 2D array (known as *window*, *template*, *mask*, or *kernel*) produces a destination image in which each pixel value depends on its original value and the value of (some of) its neighbors. The convolution mask determines which neighbors are used as well as the relative weight of their original values. Masks are normally 3 × 3, such as the one shown in Figure bellow.

W_1	W_2	W_3
W_4	W_5	W_6
W_7	W_8	W_9

A 3 × 3 convolution mask, whose generic weights are $W_1, \ldots, W_9$.

Operations Combining Multiple Images

There are many image processing applications that combine two images, pixel by pixel, using an arithmetic or logical operator, resulting in a third image, *Z*:

$$X \; opn \; Y = Z$$

where *X* and *Y* may be images (arrays) or scalars, *Z* is necessarily an array, and *opn* is a binary mathematical (+, −, ×, /) or logical (AND, OR, XOR) operator. Figure bellow shows schematically how pixel-by-pixel operations work.

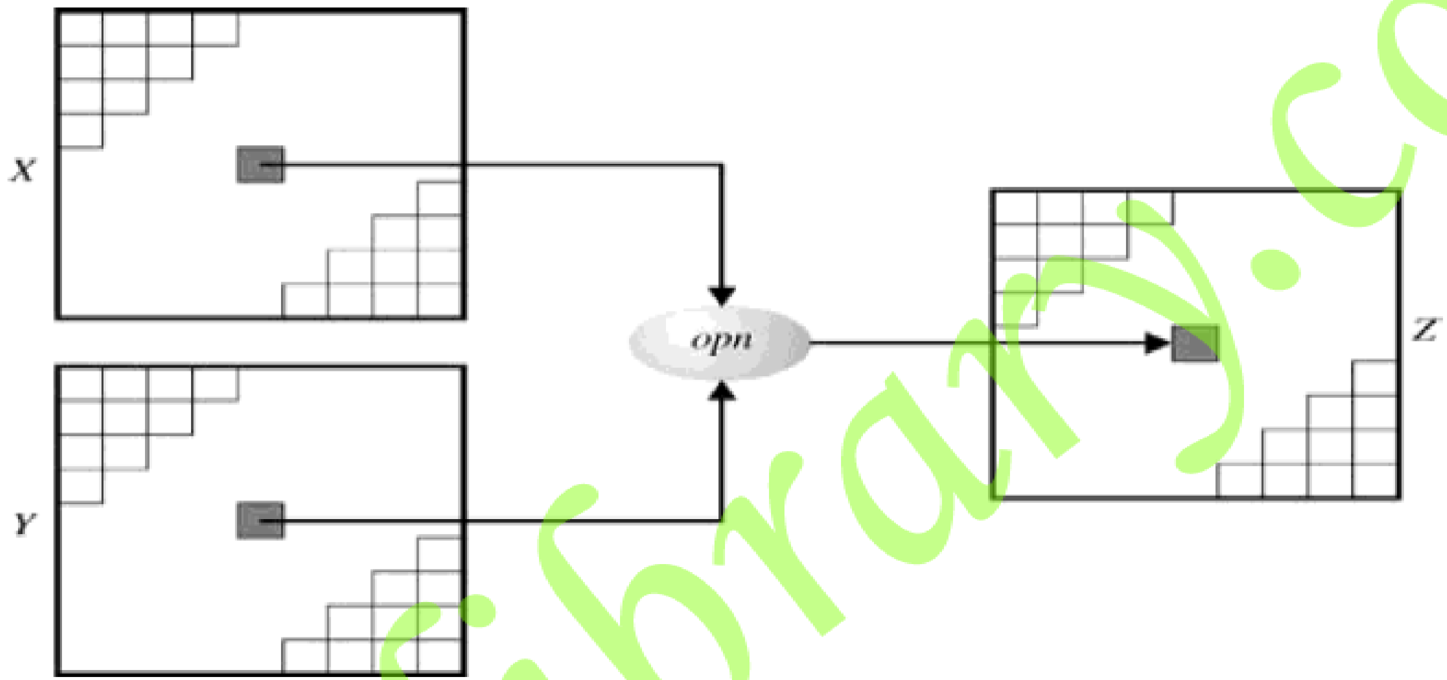
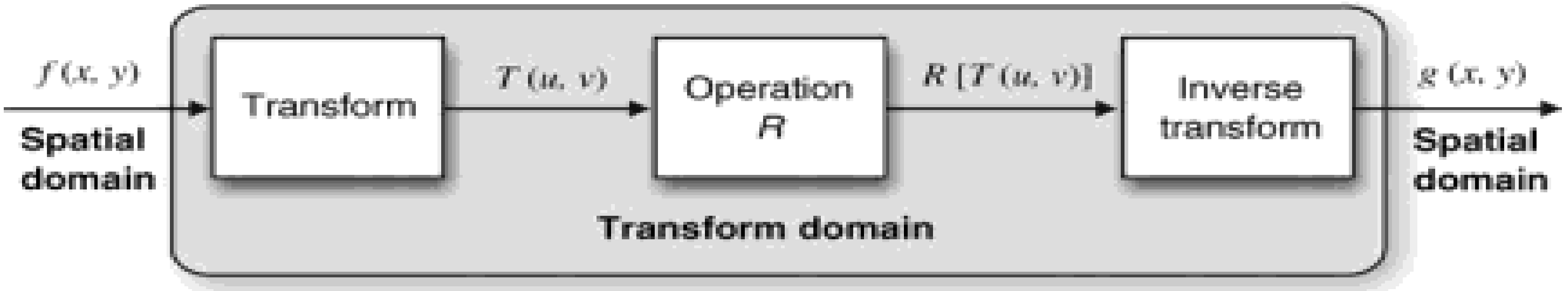


FIGURE 2.11 Pixel-by-pixel arithmetic and logic operations.

Operations in a Transform Domain

A *transform* is a mathematical tool that allows the conversion of a set of values to another set of values, creating, therefore, a new way of representing the same information. In the field of image processing, the original domain is referred to as *spatial domain*, whereas the results are said to lie in the *transform domain*.

The motivation for using mathematical transforms in image processing stems from the fact that some tasks are best performed by transforming the input images, applying selected algorithms in the transform domain, and eventually applying the inverse transformation to the result Figure bellow. This is what happens when we filter an image in the 2D frequency domain using the FT and its inverse



Operations in a transform domain.

Translation, Scaling, Rotation and Perspective Projection of image

A geometric operation can be described mathematically as the process of transforming an input image $f(x, y)$ into a new image $g(x^t, y^t)$ by modifying the *coordinates* of image pixels:

$$f(x, y) \rightarrow g(x^t, y^t) \tag{7.1}$$

that is, the pixel value originally located at coordinates (x, y) will be relocated to coordinates (x^t, y^t) in the output image.

To model this process, a *mapping function* is needed. The mapping function specifies the new coordinates (in the output image) for each pixel in the input image:

$$(x^t, y^t) = T(x, y) \tag{7.2}$$

This mapping function is an arbitrary 2D function. It is often specified as two separate functions, one for each dimension:

$$x^t = T_x(x, y) \tag{7.3}$$

and

$$y^t = T_y(x, y) \tag{7.4}$$

where T_x and T_y are usually expressed as polynomials in x and y. The case where T_x and T_y are linear combinations of x and y is called *affine transformation* (or *affine mapping*):

$$x^{t} = a_0 x + a_1 y + a_2 \tag{7.5}$$

$$y^{t} = b_0 x + b_1 y + b_2 \tag{7.6}$$

Equations (7.5) and (7.6) can also be expressed in matrix form as follows:

$$\begin{bmatrix} x^{t} \\ y^{t} \\ 1 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{7.7}$$

Affine mapping transforms straight lines to straight lines, triangles to triangles, and rectangles to parallelograms. Parallel lines remain parallel and the distance ratio

TABLE 7.1 Summary of Transformation Coefficients for Selected Affine Transformations

Transformation	a_0	a_1	a_2	b_0	b_1	b_2
Translation by δ_x, δ_y	1	0	δ_x	0	1	δ_y
Scaling by a factor $[s_x, s_y]$	s_x	0	0	0	s_y	0
Counterclockwise rotation by angle θ	$\cos\theta$	$\sin\theta$	0	$-\sin\theta$	$\cos\theta$	0
Shear by a factor $[\mathrm{sh}_x, \mathrm{sh}_y]$	1	sh_y	0	sh_x	1	0

7.4.2 Translation

Translation of an input image $f(x, y)$ with respect to its Cartesian origin to produce an output image $g(x^t, y^t)$ where each pixel is displaced by $[\sigma_x, \sigma_y]$ (i.e., $x^t = x + \sigma_x$ and $y^t = y + \sigma_y$) consists of a special case of affine transform (as discussed in Section 7.2). In Tutorial 7.2 (page 142), you will use `maketform` and `imtransform` to perform image translation.

7.4.3 Rotation

Rotation of an image constitutes another special case of affine transform (as discussed in Section 7.2). Consequently, image rotation can also be accomplished using `maketform` and `imtransform`.

The IPT also has a specialized function for rotating images, `imrotate`. Similar to `imresize`, `imrotate` allows the user to specify the interpolation method used: nearest-neighbor (the default method), bilinear, or bicubic. It also allows specification of the size of the output image. In Tutorials 7.1 (page 138), and 7.2 (page 142), you will explore the `imrotate` function.

7.4.4 Cropping

The IPT has a function for cropping images, imcrop, which crops an image to a specified rectangle. The crop rectangle can be specified interactively (with the mouse) or its coordinates be passed as parameters to the function. In Tutorial 7.1 (page 138), you will experiment with both options for using this function.

7.4.5 Flipping

The IPT has two functions for flipping matrices (which can also be used for raster images, of course): flipud—which flips a matrix up to down—and fliplr—which flips a matrix left to right. In Tutorial 7.1 (page 138), you will experiment with both functions.

7.5.1 Warping

Warping can be defined as the "transformation of an image by reparameterization of the 2D plane" [FDHF$^+$05]. Warping techniques are sometimes referred to as *rubber sheet transformations*, because they resemble the process of applying an image to a sheet of rubber and stretching it according to a predefined set of rules.

The *quadratic warp* is a particular case of *polynomial warping*, where the transformed coordinates (x^t, y^t) for a pixel whose original coordinates are (x, y) are given by the following equations:

$$x^t = a_0x^2 + a_1y^2 + a_2xy + a_3x + a_4y + a_5 \tag{7.8}$$

$$y^t = b_0x^2 + b_1y^2 + b_2xy + b_3x + b_4y + b_5 \tag{7.9}$$

7.5.2 Nonlinear Image Transformations

Nonlinear image transformations usually involve a conversion from rectangular to polar coordinates followed by a deliberate distortion of the resulting points.

Twirling The twirl transformation causes an image to be rotated around an anchor point of coordinates (x_c, y_c) with a space-variant rotation angle: the angle has a value of α at the anchor point and decreases linearly with the radial distance from the center. The effect is limited to a region within the maximum radius r_{max}. All pixels outside this region remain unchanged.

Since this transformation uses backward mapping, we are interested in the equations for the *inverse* mapping function:

$$T_x^{-1} : x = \begin{cases} x_c + r\cos(\theta) & \text{for } r \le r_{max} \\ x^t & \text{for } r > r_{max} \end{cases} \tag{7.10}$$

Image Manipulation

Image Filtering: Change range (brightness)

$$g(x,y) = T_r(f(x,y))$$

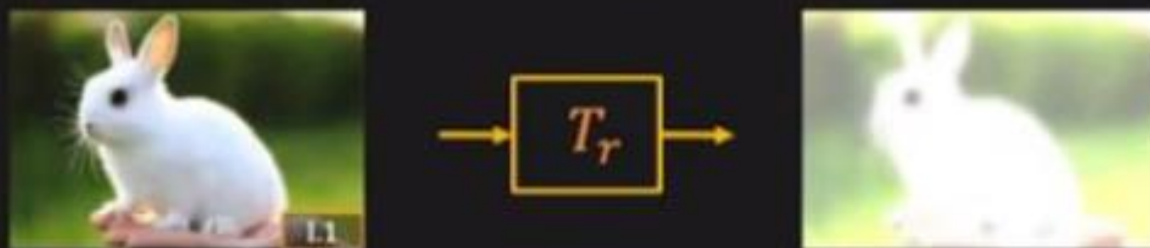
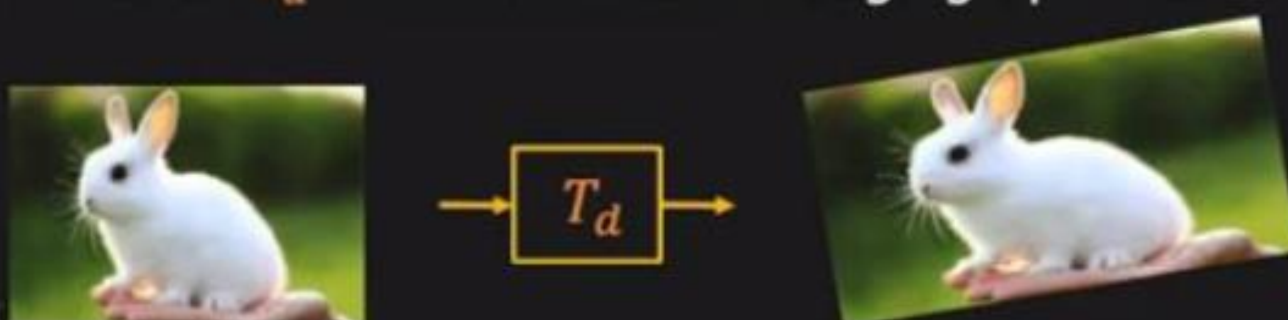


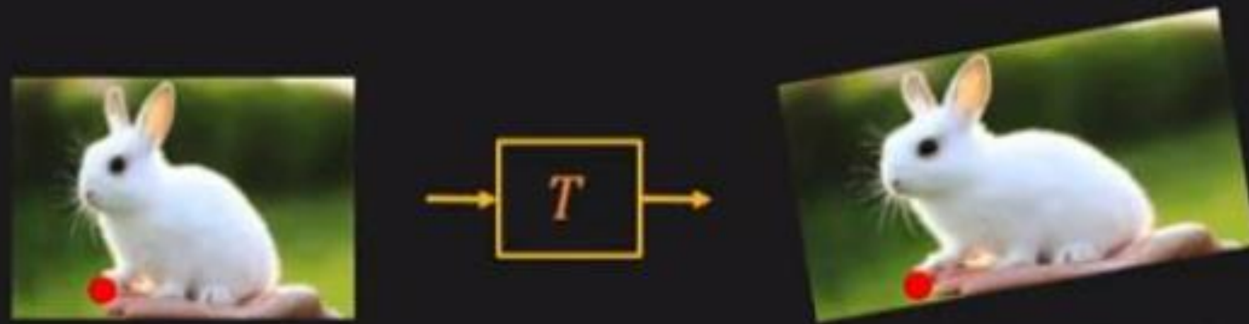
Image Warping: Change domain (location)

$$g(x,y) = f\left(T_d(x,y)\right)$$

Transformation T_d is a coordinate changing operator



2x2 Linear Transformations

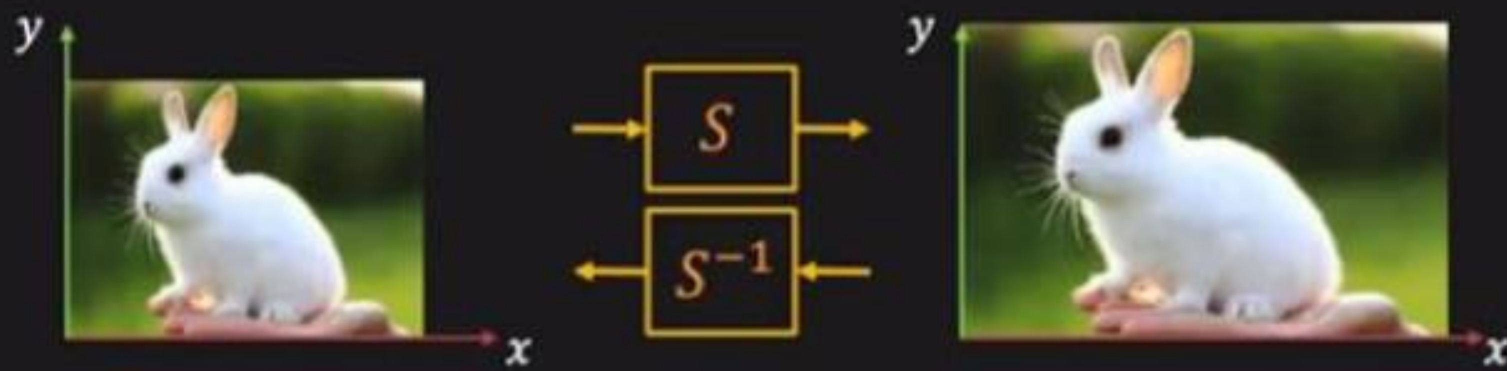


$\mathbf{p}_1 = (x_1, y_1)$ $\mathbf{p}_2 = (x_2, y_2)$

T can be represented by a matrix.

$$\mathbf{p}_2 = T\mathbf{p}_1 \qquad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = T \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \qquad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Scaling (Stretching or Squishing)



Forward:

$$x_2 = ax_1 \qquad y_2 = by_1$$

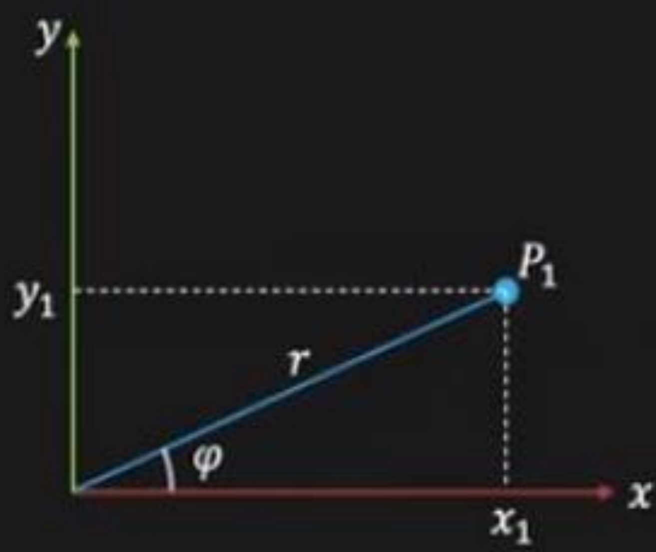
$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Inverse:

$$x_1 = \frac{1}{a} x_2 \qquad y_1 = \frac{1}{b} y_2$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = S^{-1} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1/a & 0 \\ 0 & 1/b \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

2D Rotation



$$x_1 = r\cos(\varphi)$$
$$y_1 = r\sin(\varphi)$$

$$x_2 = r\cos(\varphi + \theta)$$
$$x_2 = r\cos\varphi\cos\theta - r\sin\varphi\sin\theta$$
$$x_2 = x_1\cos\theta - y_1\sin\theta$$

$$y_2 = r\sin(\varphi + \theta)$$
$$y_2 = r\cos\varphi\sin\theta + r\sin\varphi\cos\theta$$
$$y_2 = x_1\sin\theta + y_1\cos\theta$$

Rotation

Forward:

$$x_2 = x_1 cos\theta - y_1 sin\theta$$

$$y_2 = x_1 sin\theta + y_1 cos\theta$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = R \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

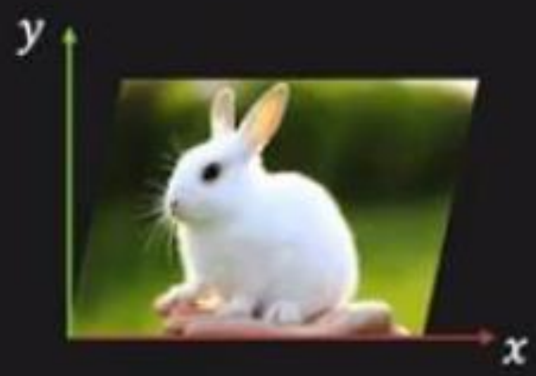
Inverse:

$$x_1 = x_2 cos\theta + y_2 sin\theta$$

$$y_1 = -x_2 sin\theta + y_2 cos\theta$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = R^{-1} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

Skew



Horizontal Skew:

$$x_2 = x_1 + m_x y_1$$
$$y_2 = y_1$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S_x \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & m_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Vertical Skew:

$$x_2 = x_1$$
$$y_2 = m_y x_1 + y_1$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S_x \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ m_y & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Mirror

Mirror about Y-axis:

$$x_2 = -x_1$$
$$y_2 = y_1$$

$$M_y = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Mirror about line $y = x$:

$$x_2 = y_1$$
$$y_2 = x_1$$

$$M_{xy} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Translation

$$x_2 = x_1 + t_x \qquad y_2 = y_1 + t_y$$

Can translation be expressed as a 2x2 matrix? **No.**

Global Warping/Transformation

Translation Rotation Scaling and Aspect

$$\boxed{g(x,y) = f\big(T(x,y)\big)}$$

Affine Projective Barrel

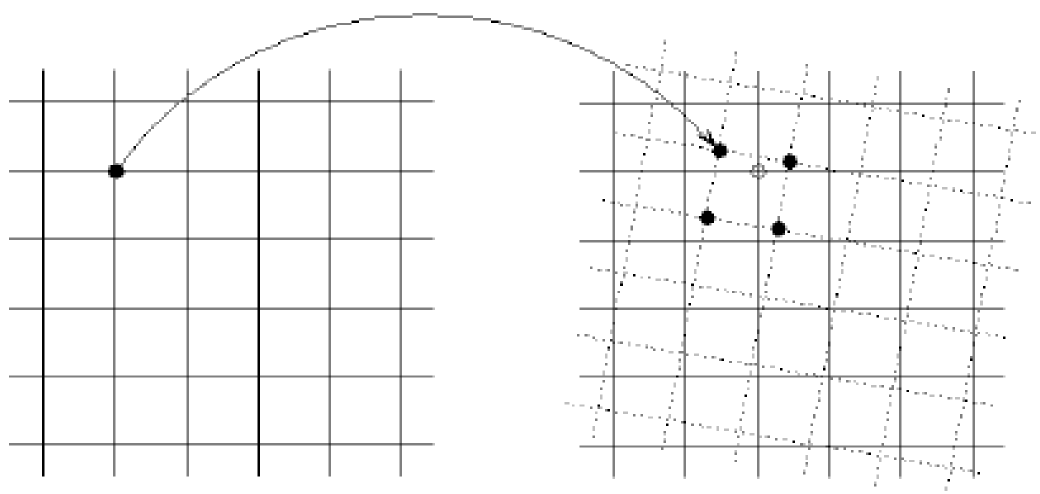
Transformation T is the same over entire domain

Often can be described by just a few parameters

Spatial Transformation

In a spatial transformation each point (x, y) of image A is mapped to a point (u, v) in a new coordinate system.

$$u = f_1(x, y)$$
$$v = f_2(x, y)$$



Mapping from (x, y) to (u, v) coordinates. A digital image array has an implicit grid that is mapped to discrete points in the new domain. These points may not fall on grid points in the new domain.

Affine Transformation

An affine transformation is any transformation that preserves collinearity (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation).

In general, an affine transformation is a composition of rotations, translations, magnifications, and shears.

$$u = c_{11}x + c_{12}y + c_{13}$$

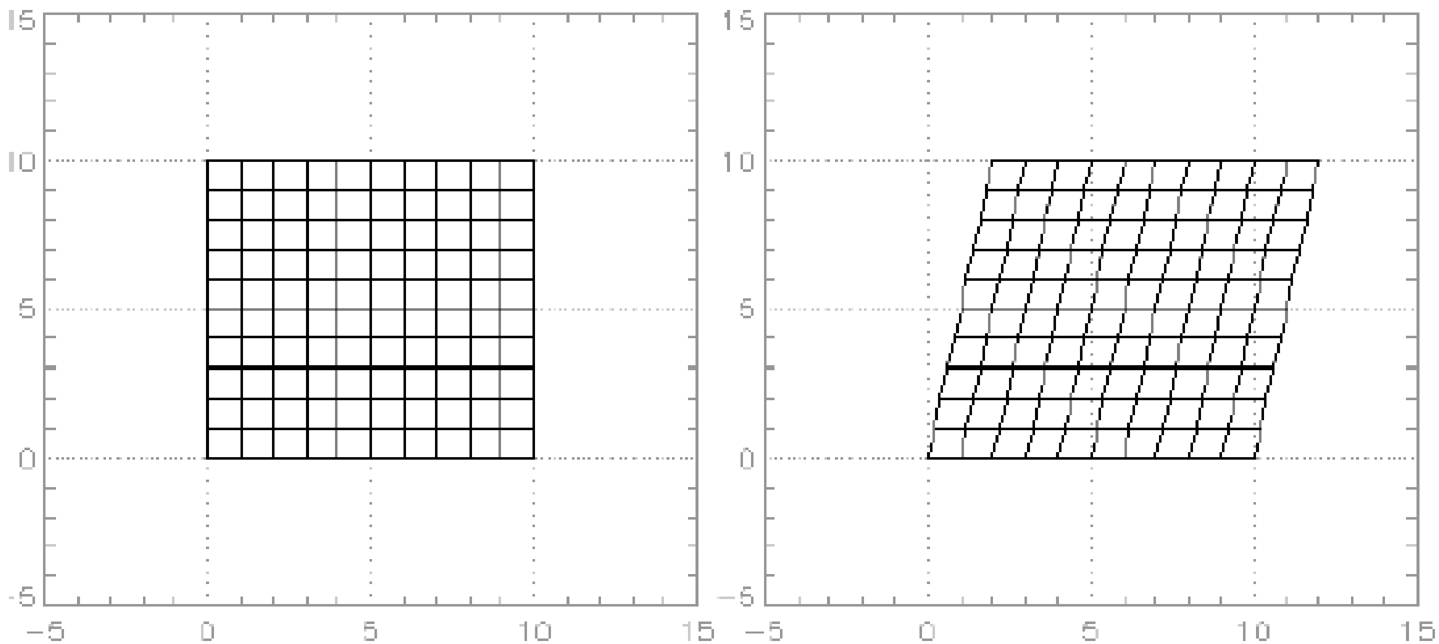
$$v = c_{21}x + c_{22}y + c_{23}$$

c_{13} and c_{23} affect translations, c_{11} and c_{22} affect magnifications, and the combination affects rotations and shears.

Affine Transformation

A shear in the x direction is produced by

$$u = x + 0.2y$$
$$v = y$$

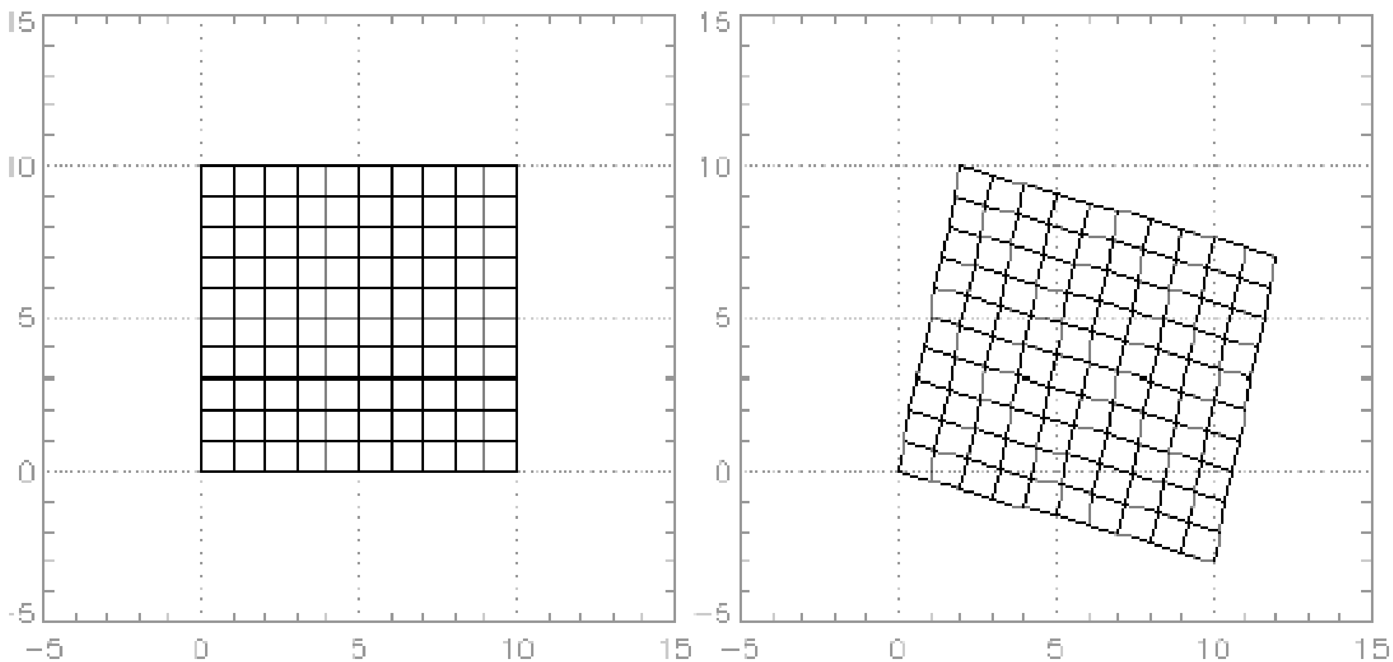


Affine Transformation

This produces as both a shear and a rotation.

$$u = x + 0.2y$$

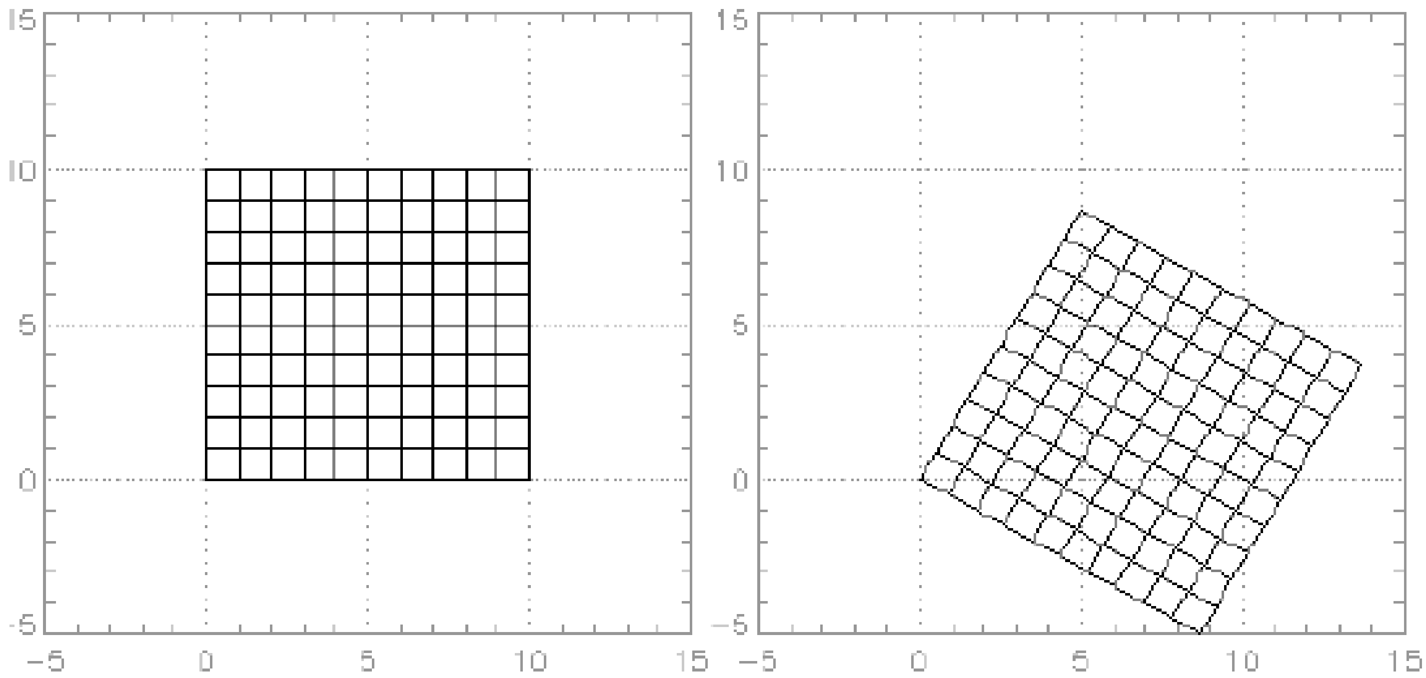
$$v = -0.3x + y$$



Affine Transformation

A rotation is produced by θ is produced by

$$u = x\cos\theta + y\sin\theta$$
$$v = -x\sin\theta + y\cos\theta$$



EXAMPLE 7.1

Generate the affine transformation matrix for each of the operations below: (a) rotation by 30°; (b) scaling by a factor 3.5 in both dimensions; (c) translation by [25, 15] pixels;

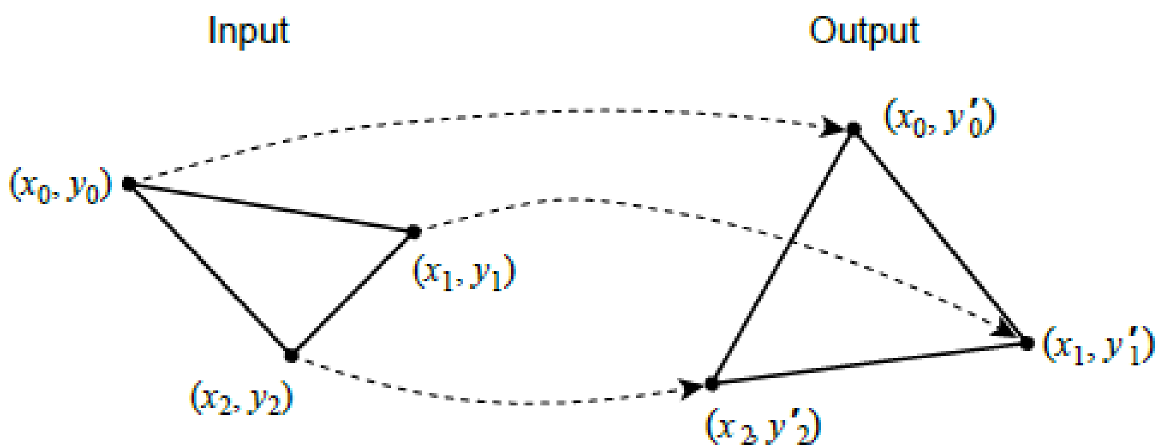


FIGURE 7.2 Mapping one triangle onto another by an affine transformation.

(d) shear by a factor [2, 3]. Use MATLAB to apply the resulting matrices to an input image of your choice.

Solution

Plugging the values into Table 7.1, we obtain the following:
(a) Since $\cos 30^\circ = 0.866$ and $\sin 30^\circ = 0.500$:

$$\begin{bmatrix} 0.866 & -0.500 & 0 \\ 0.500 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(b)

$$\begin{bmatrix} 3.5 & 0 & 0 \\ 0 & 3.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(c)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 25 & 15 & 1 \end{bmatrix}$$

(d)

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

Combinations of Transforms

Complex affine transforms can be constructed by a sequence of basic affine transforms.

Transform combinations are most easily described in terms of matrix operations. To use matrix operations we introduce *homogeneous coordinates*. These enable all affine operations to be expressed as a matrix multiplication. Otherwise, translation is an exception.

The affine equations are expressed as

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

An equivalent expression using matrix notation is

$$\mathbf{q} = \mathbf{T}\mathbf{p}$$

where $\mathbf{q}$, $\mathbf{T}$ and $\mathbf{q}$ are the defined above.

Transform Operations

The transformation matrices below can be used as building blocks.

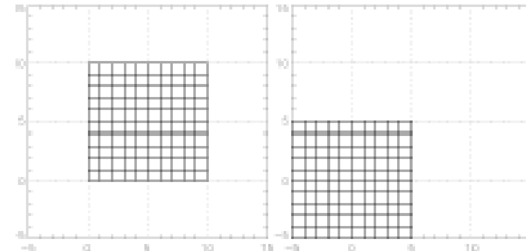
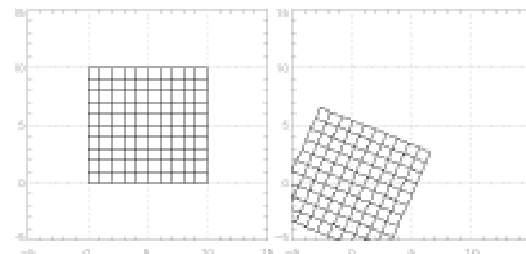
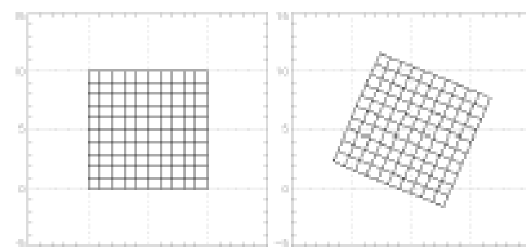
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Translation by } (x_0, y_0)$$

$$\mathbf{T} = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Scale by } s_1 \text{ and } s_2$$

$$\mathbf{T} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Rotate by } \theta$$

You will usually want to translate the center of the image to the origin of the coordinate system, do any rotations and scalings, and then translate it back.

Combined Transform Operations

Operation	Expression	Result
Translate to Origin	$\mathbf{T}_1 = \begin{bmatrix} 1.00 & 0.00 & -5.00 \\ 0.00 & 1.00 & -5.00 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$	
Rotate by 23 degrees	$\mathbf{T}_2 = \begin{bmatrix} 0.92 & 0.39 & 0.00 \\ -0.39 & 0.92 & 0.00 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$	
Translate to original location	$\mathbf{T}_3 = \begin{bmatrix} 1.00 & 0.00 & 5.00 \\ 0.00 & 1.00 & 5.00 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$	

Composite Affine Transformation

The transformation matrix of a sequence of affine transformations, say $\mathbf{T}_1$ then $\mathbf{T}_2$ then $\mathbf{T}_3$ is

$$\mathbf{T} = \mathbf{T}_3\mathbf{T}_2\mathbf{T}_3$$

The composite transformation for the example above is

$$\mathbf{T} = \mathbf{T}_3\mathbf{T}_2\mathbf{T}_1 = \begin{bmatrix} 0.92 & 0.39 & -1.56 \\ -0.39 & 0.92 & 2.35 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$$

Any combination of affine transformations formed in this way is an affine transformation.

The inverse transform is

$$\mathbf{T}^{-1} = \mathbf{T}_1^{-1}\mathbf{T}_2^{-1}\mathbf{T}_3^{-1}$$

If we find the transform in one direction, we can invert it to go the other way.

Composite Affine Transformation RST

Suppose that you want the composite representation for translation, scaling and rotation (in that order).

$$H = RST = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_0 & 0 & 0 \\ 0 & s_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & x_1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_0\cos\theta & s_1\sin\theta & s_0x_0\cos\theta + s_1x_1\sin\theta \\ -s_0\sin\theta & s_1\cos\theta & s_1x_1\cos\theta - s_0x_0\sin\theta \end{bmatrix}$$

Given the matrix H one can solve for the five parameters.

How to Find the Transformation

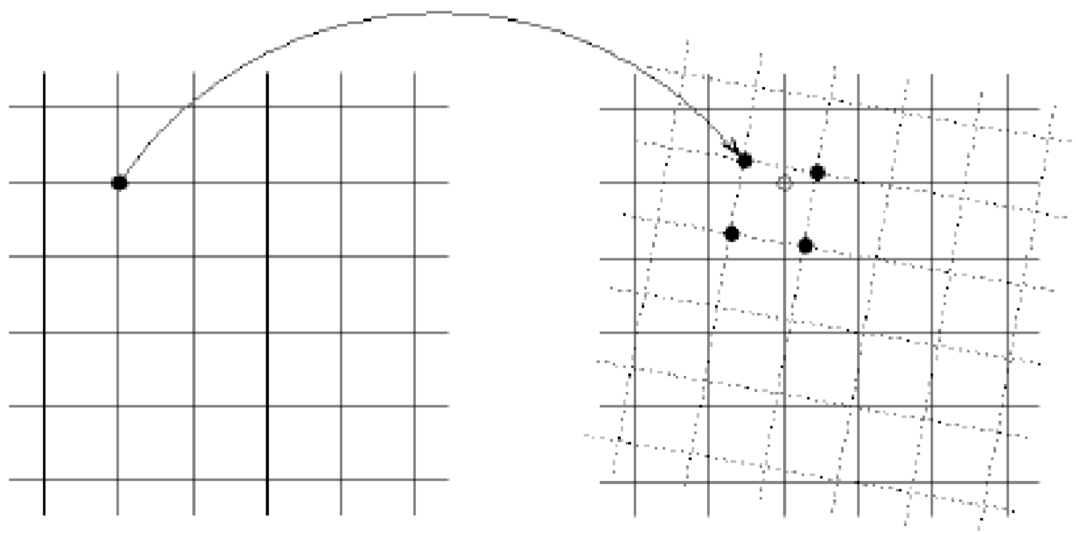
Suppose that you are given a pair of images to align. You want to try an affine transform to register one to the coordinate system of the other. How do you find the transform parameters?

Image A

Image B

Interpolation

Interpolation is needed to find the value of the image at the grid points in the target coordinate system. The mapping T locates the grid points of A in the coordinate system of B, but those grid points are not on the grid of B.

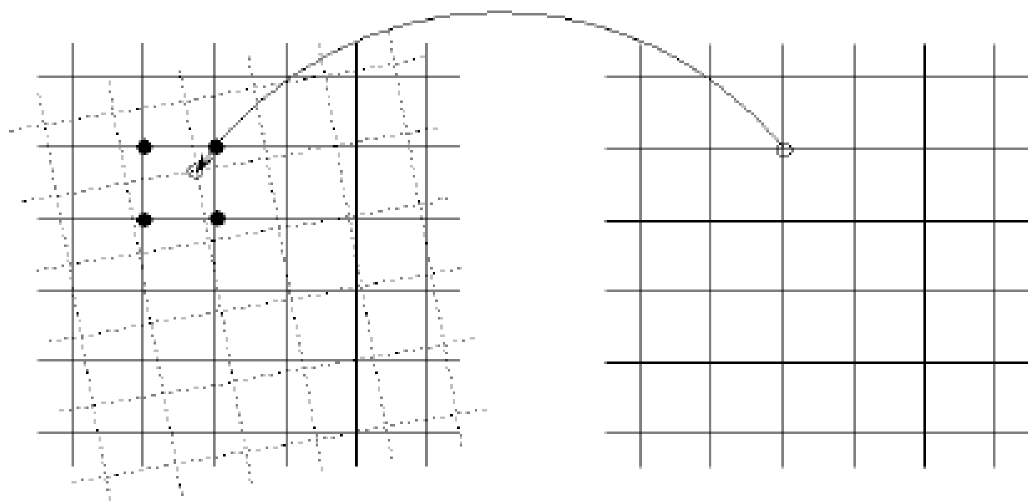


To find the values on the grid points of B we need to interpolate from the values at the projected locations.

Finding the closest projected points to a given grid point can be computationally expensive.

Inverse Projection

Projecting the grid of B into the coordinate system of A maintains the known image values on a regular grid. This makes it simple to find the nearest points for each interpolation calculation.



Let $\mathbf{Q}_g$ be the homogeneous grid coordinates of B and let $\mathbf{H}$ be the transformation from A to B. Then

$$\mathbf{P} = \mathbf{H}^{-1}\mathbf{Q}_g$$

represents the projection from B to A. We want to find the value at each point $\mathbf{P}$ given from the values on $\mathbf{P}_g$, the homogeneous grid coordinates of A.

Methods of Interpolation

There are several common methods of interpolation:

- Nearest neighbor – simplest and fastest
- Triangular – Uses three points from a bounding triangle. Useful even when the known points are not on a regular grid.
- Bilinear – Uses points from a bounding rectangle. Useful when the known points are on a regular grid.

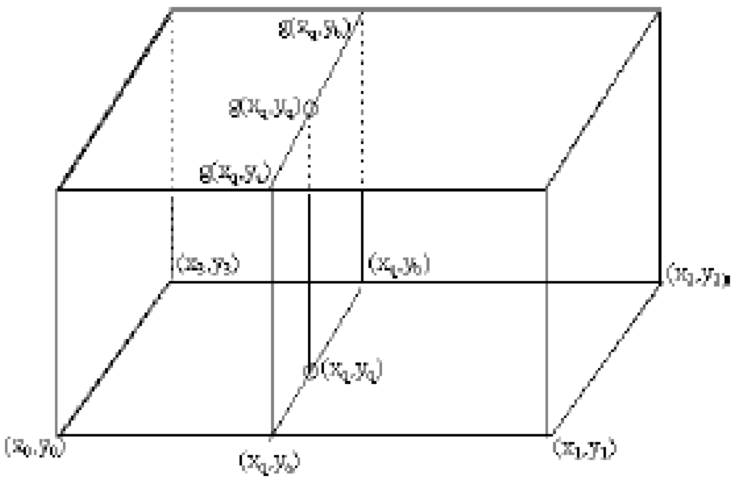
Bilinear Interpolation

Suppose that we want to find the value $g(q)$ at a point q that is interior to a four-sided figure with vertices $\{p_0, p_1, p_2, p_3\}$. Assume that these points are in order of progression around the figure and that p_0 is the point farthest to the left.

1. Find the point (x_q, y_a) between p_0 and p_1. Compute $g(x_q, y_a)$ by linear interpolation between $f(p_0)$ and $f(p_1)$.

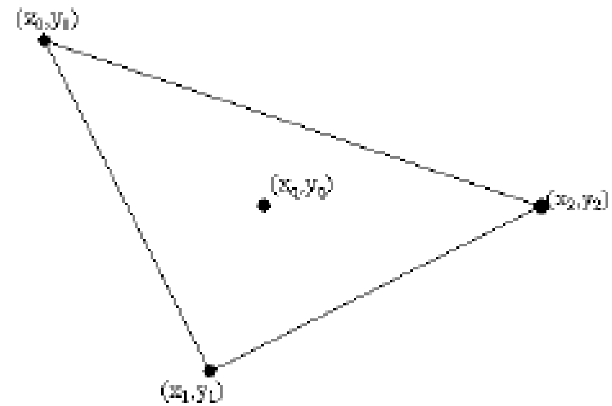
2. Find the point (x_q, y_b) between p_3 and p_2. Compute $g(x_q, y_b)$ by linear interpolation between $f(p_3)$ and $f(p_2)$.

3. Linearly interpolate between $g(x_q, y_a)$ and $g(x_q, y_b)$ to find $g(x_q, y_q)$.



Triangular Interpolation

Let (x_i, y_i, z_i) $i = 0, 1, 2$ be three points that are not collinear. These points form a triangle. Let (x_q, y_q) be a point inside the triangle. We want to compute a value z_q such that (x_q, y_q, z_q) falls on a plane that contains (x_i, y_i, z_i), $i = 0, 1, 2$.



This interpolation will work even if q is not within the triangle, but, for accuracy, we want to use a bounding triangle.

The plane is described by an equation

$$z = a_0 + a_1 x + a_2 y$$

The coefficients must satisfy the three equations that correspond to the corners of the triangle.

Triangular Interpolation

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

In matrix notation we can write

$$\mathbf{z} = \mathbf{C}\mathbf{a}$$

so that

$$\mathbf{a} = \mathbf{C}^{-1}\mathbf{z}$$

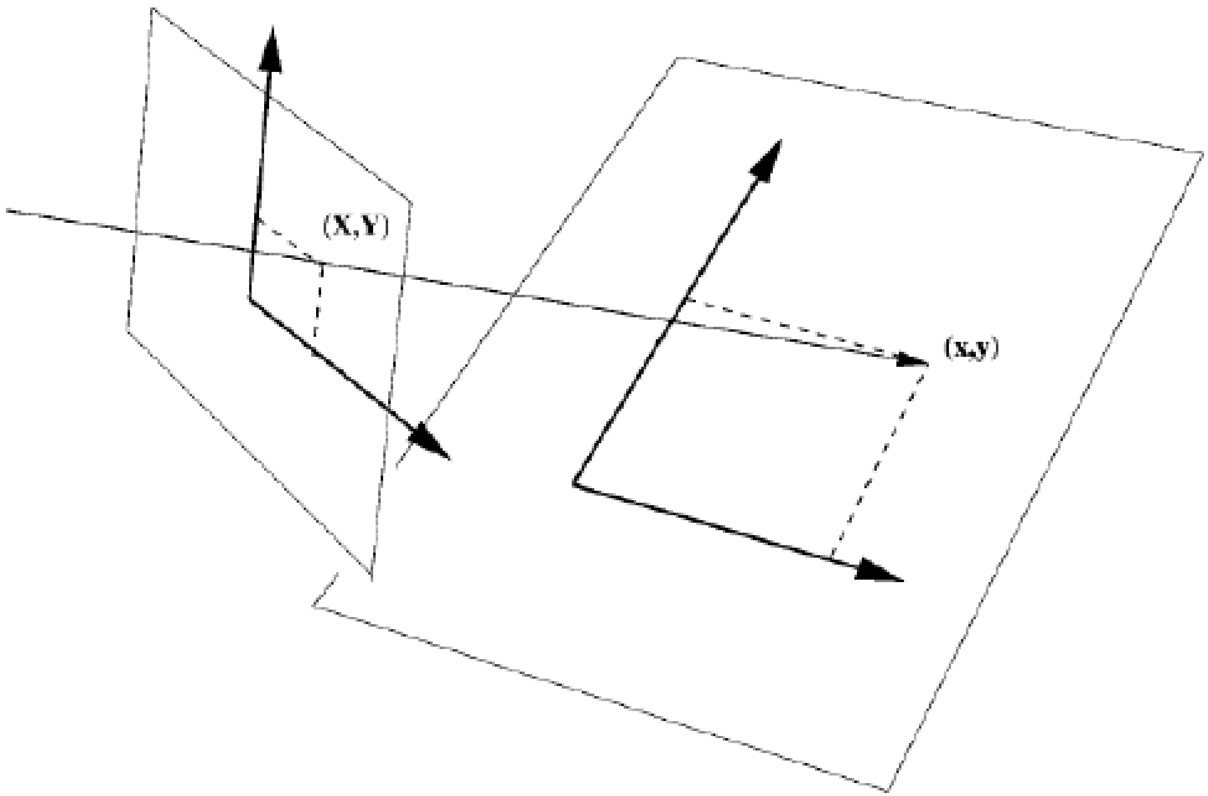
The matrix $\mathbf{C}$ is nonsingular as long as the triangle corners do not fall along a line. Then the value z_q is given by

$$z_q = \begin{bmatrix} 1 & x_q & y_q \end{bmatrix} \mathbf{a} = \begin{bmatrix} 1 & x_q & y_q \end{bmatrix} \mathbf{C}^{-1}\mathbf{z}$$

Since $\mathbf{C}$ depends only upon the locations of the triangle corners, (x_i, y_i), $i = 0, 1, 2$ it can be computed once the triangles are known. This is useful in processing large batches of images.

Projective Transform

The projective transform can handle changes caused by a tilt of the image plane relative to the object plane.



Projective Transform

The perspective transformation maps (X, Y, Z) points in 3D space to (x, y) points in the image plane.

$$x_i = \frac{-fX_0}{Z_0 - f} \quad \text{and} \quad y_i = \frac{-fY_0}{Z_0 - f}$$

Suppose that A and B are images taken at different camera angles. Projection of one image plane onto the other to correct the relative tilt requires a projective transform.

$$u = \frac{ax + by + c}{gx + hy + 1} \quad \text{and} \quad v = \frac{dx + ey + f}{gx + hy + 1}$$

This eight-parameter transform maps (x, y) points in A to (u, v) points in B.

Projective Transform

The coefficients can be computed if $n \geq 4$ matching points are known in A and B. Arrange the equations as

$$ax_i + by_i + c = gx_iu_i + hy_iu_i + u_i$$

$$dx_i + ey_i + f = gx_iv_i + hy_iv_i + v_i$$

$$\begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -x_0u_0 & -y_0u_0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -x_0v_0 & -y_0v_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ \vdots & & & & & & \vdots & \vdots \\ x_{n-1} & y_{n-1} & 1 & 0 & 0 & 0 & -x_{n-1}u_{n-1} & -y_{n-1}u_{n-1} \\ 0 & 0 & 0 & x_{n-1} & y_{n-1} & 1 & -x_{n-1}v_{n-1} & -y_{n-1}v_{n-1} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} u_0 \\ v_0 \\ u_1 \\ v_1 \\ \vdots \\ u_{n-1} \\ v_{n-1} \end{bmatrix}$$

The parameters can be found by multiplying both sides with the pseudo-inverse of the big matrix of coordinate terms.

and

$$T_y^{-1}: y = \begin{cases} y_c + r\sin(\theta) & \text{for } r \leq r_{\max} \\ y^t & \text{for } r > r_{\max} \end{cases} \tag{7.11}$$

where

$$d_x = x^t - x_c, \quad r = \sqrt{d_x^2 + d_y^2}$$

$$d_y = y^t - y_c, \quad \theta = \arctan(d_x, d_y) + \alpha \cdot \left(\frac{r_{\max} - r}{r_{\max}}\right)$$

Rippling The ripple transformation causes a local wave-like displacement of the image along both directions, x and y. The parameters for this mapping function are the (nonzero) period lengths L_x, L_y (in pixels) and the associated amplitude values A_x, A_y. The inverse transformation function is given by the following:

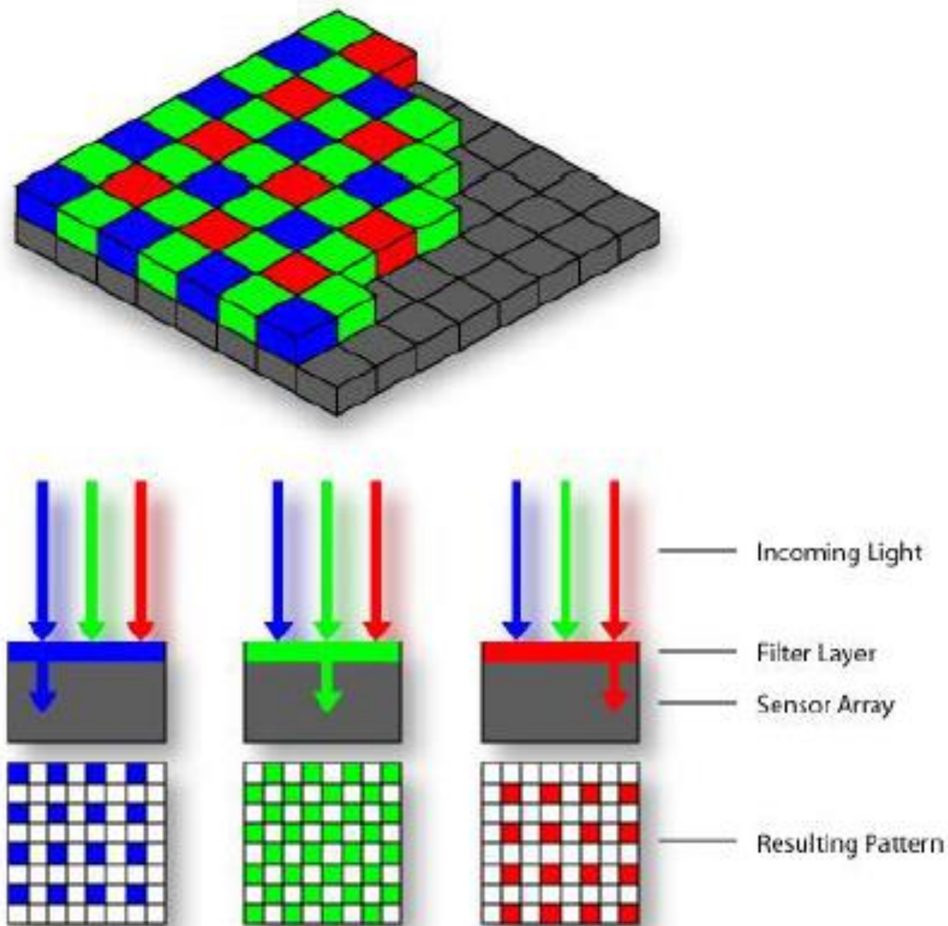
$$T_x^{-1}: x = x^t + A_x \cdot \sin\left(\frac{2\pi \cdot y^t}{L_x}\right) \tag{7.12}$$

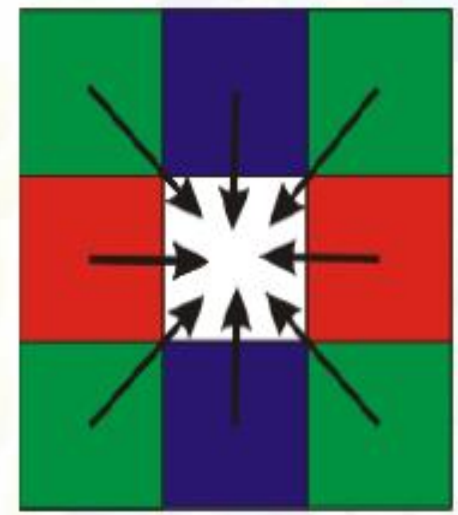
$$T_y^{-1}: y = y^t + A_y \cdot \sin\left(\frac{2\pi \cdot x^t}{L_y}\right) \tag{7.13}$$

FIGURE 7.6 Image deformation effects using *Photo Booth*.

Color Spaces:

- Practical Color Sensing: Bayer Grid.

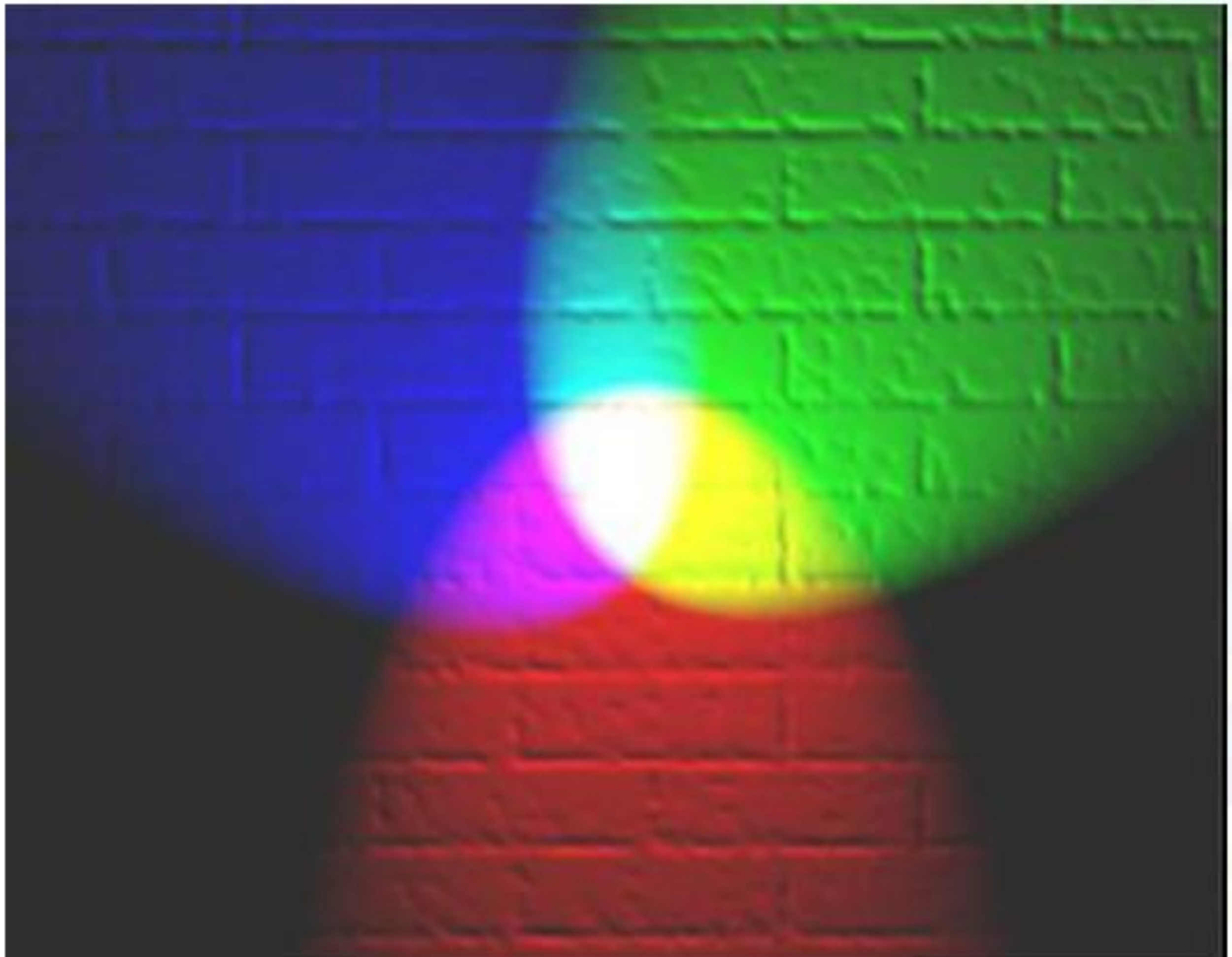




- Estimate RGB at ‘G’ cells from neighboring values

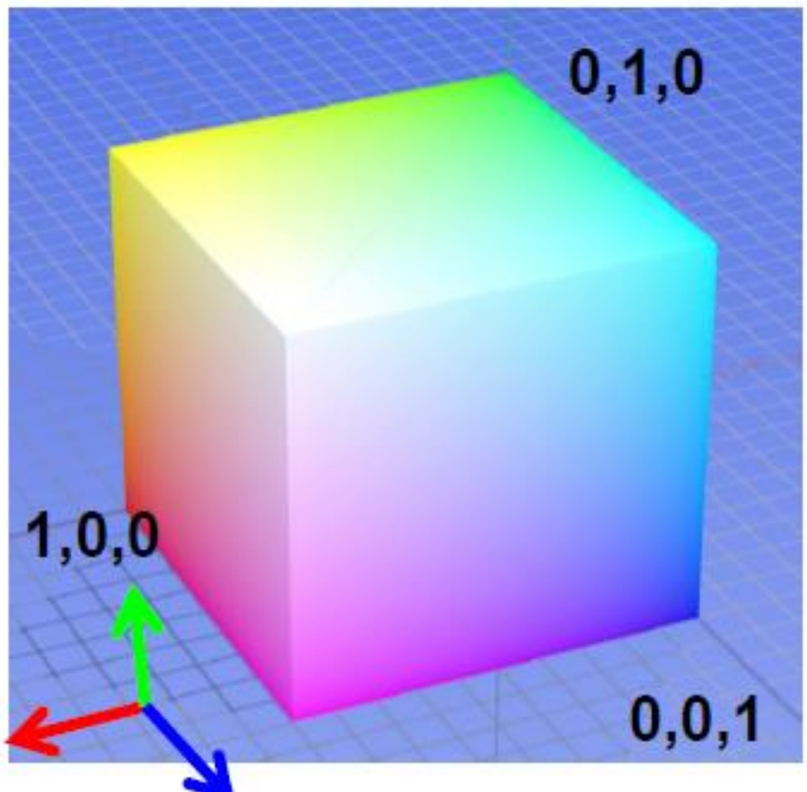
Color Sensing in Camera (RGB):

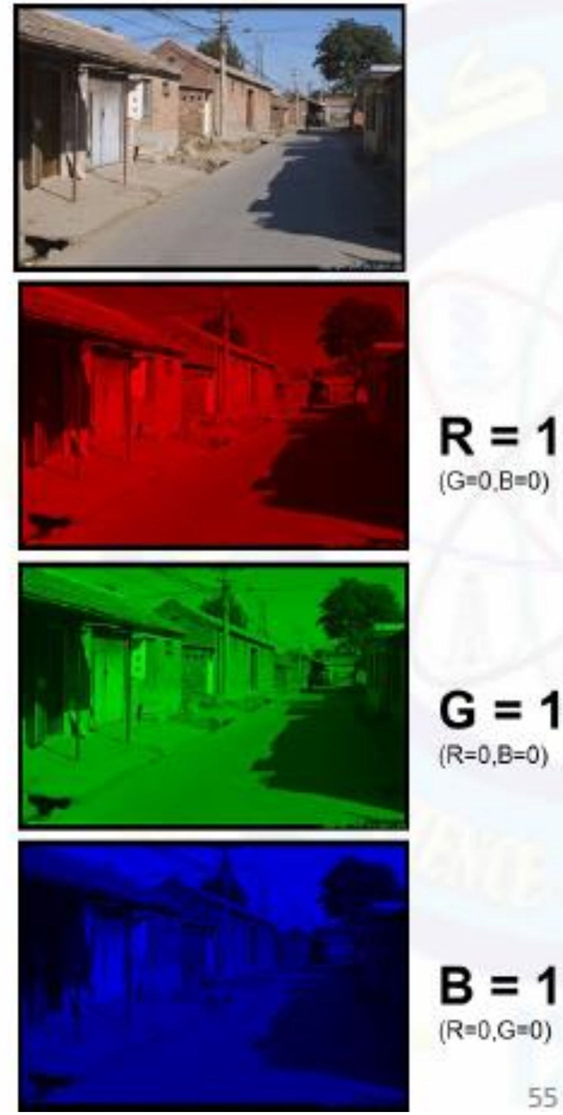
- Practical Color Sensing: Bayer Grid.



Color Sensing in Camera (RGB):

- Default color space:
  - Any color = r*R + g*G + b*B.
  - Strongly correlated channels.
  - Non-perceptual.

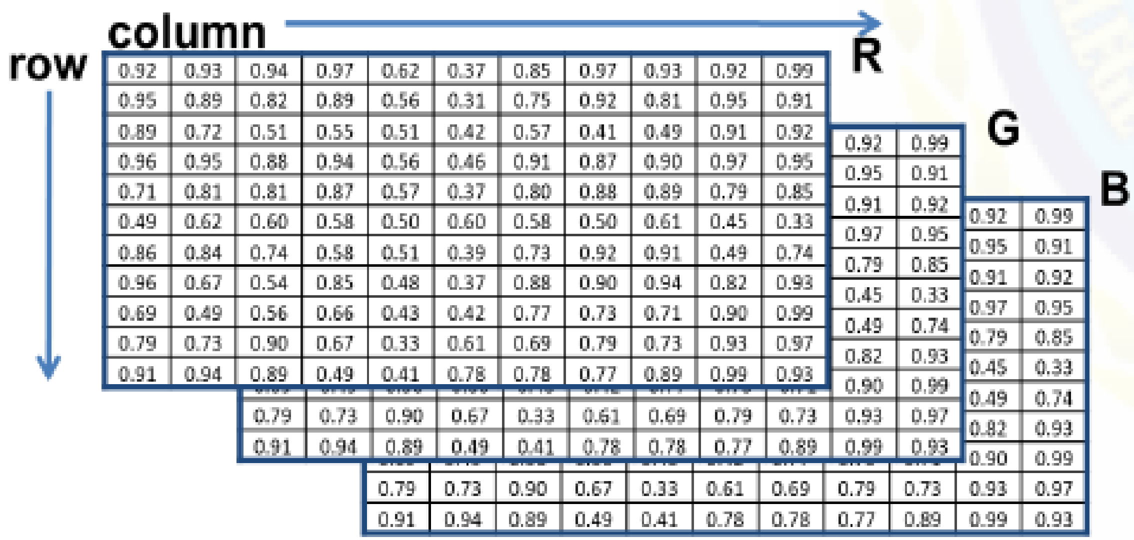




Color Image (RGB):

Color Image (RGB):

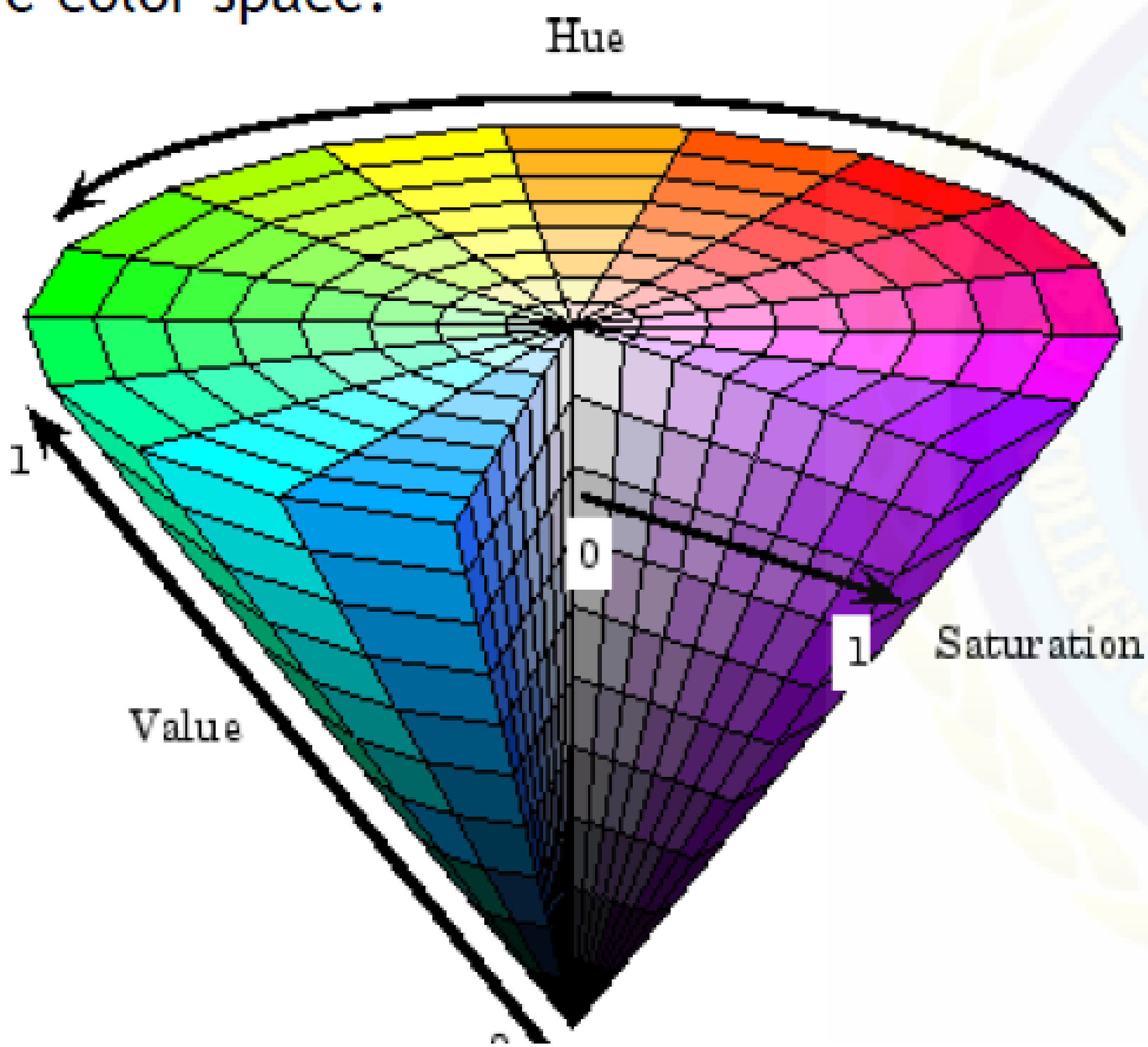
- Images represented as a matrix.
- Suppose we have a NxM RGB image called “im”.
  - im(1,1,1) = top-left pixel value in R-channel.
  - im(y, x, b) = y pixels down, x pixels to right in the bth channel.
  - im(N, M, 3) = bottom-right pixel in B-channel
- imread(filename) returns a uint8 image (values 0 to 255).
  - Convert to double format (values 0 to 1) with im2double



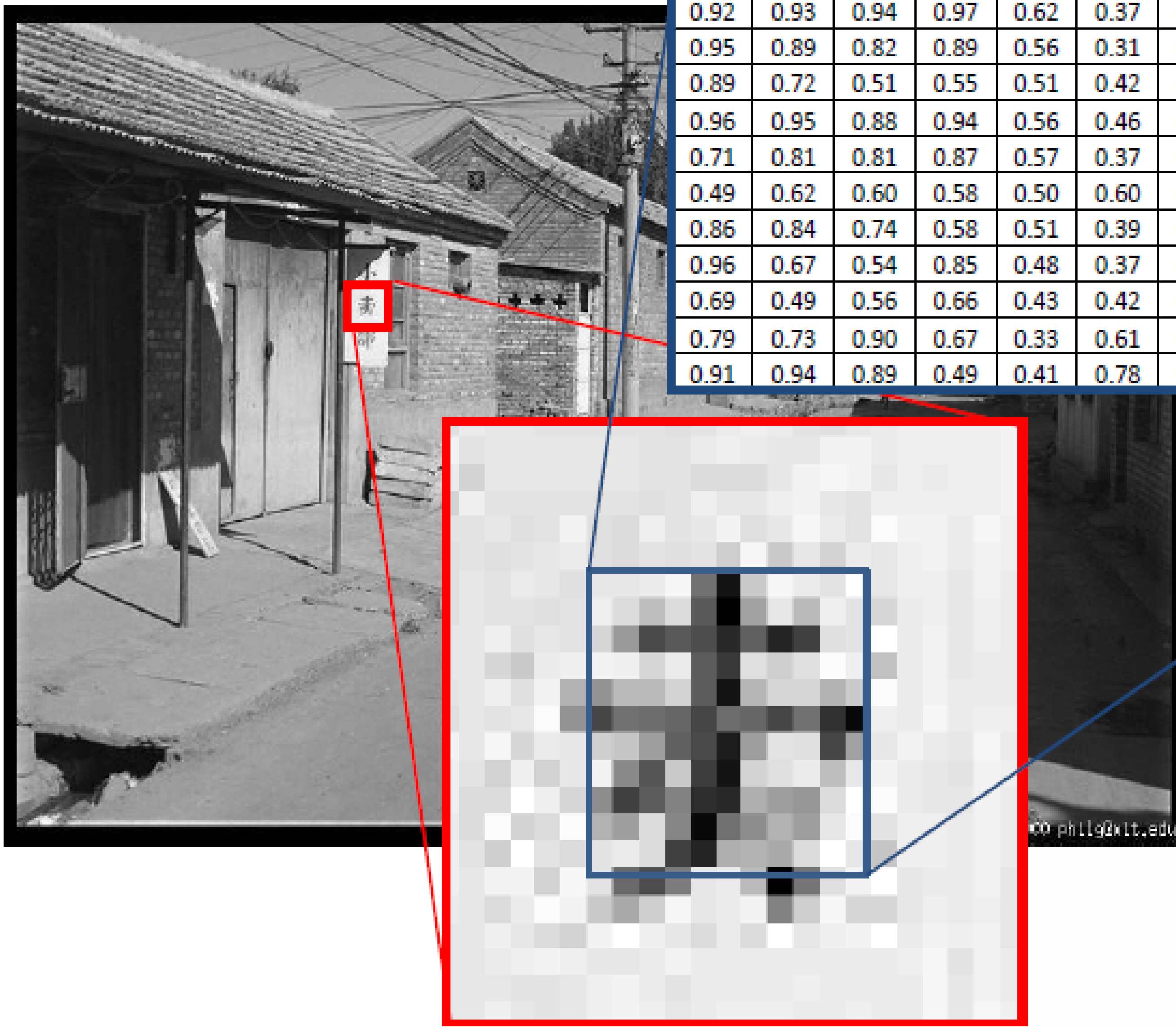
0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

Color spaces: HSV:

- Intuitive color space:



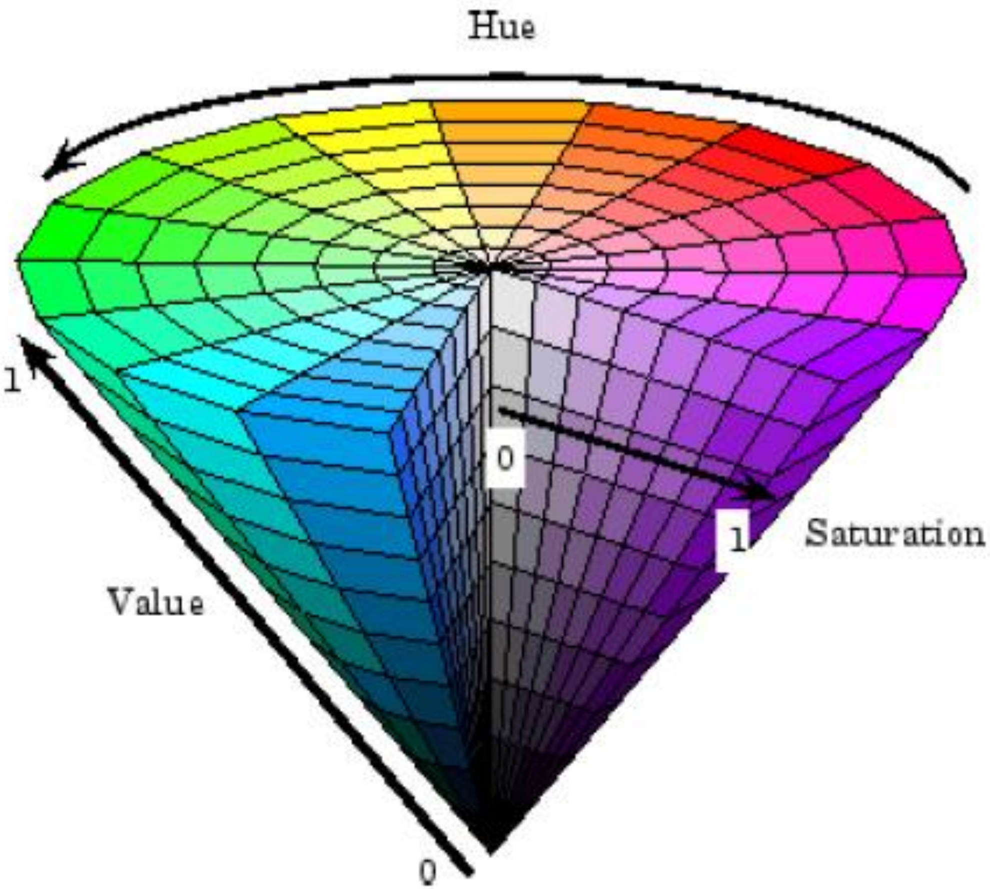
Back to grayscale intensity:

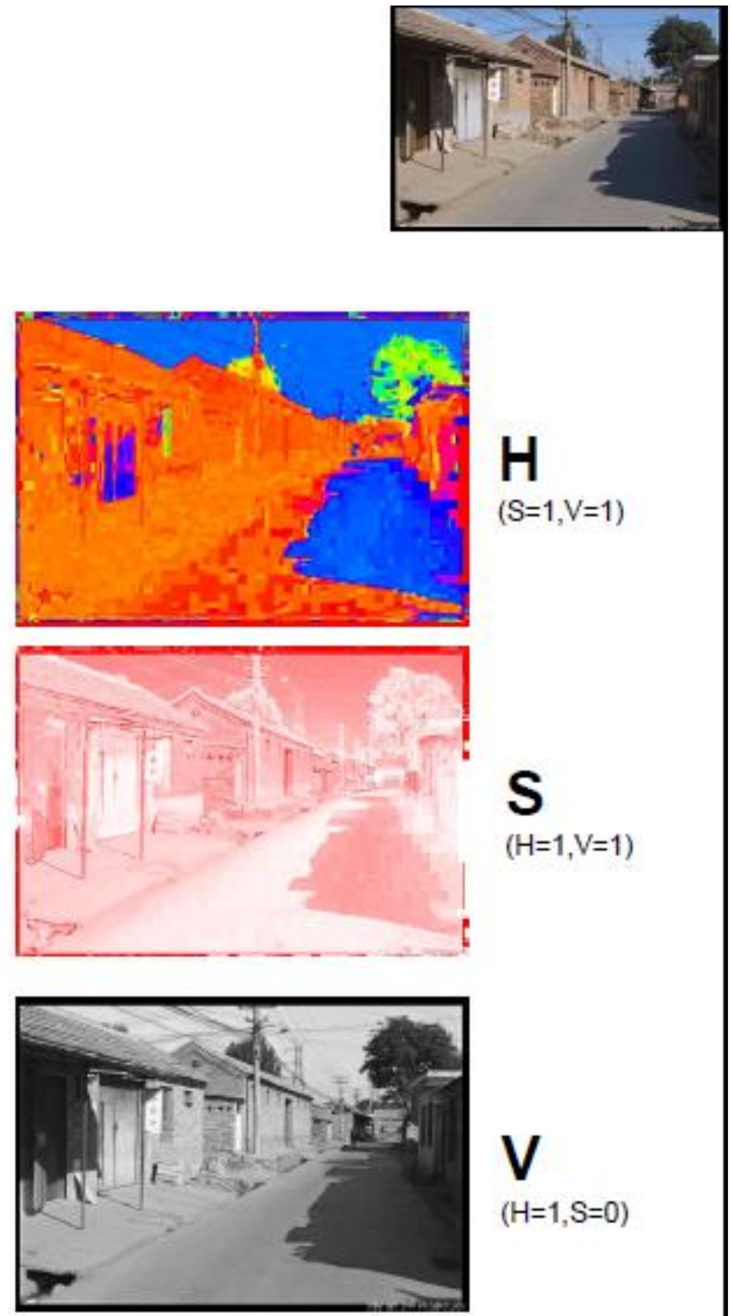


0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

Color spaces: HSV

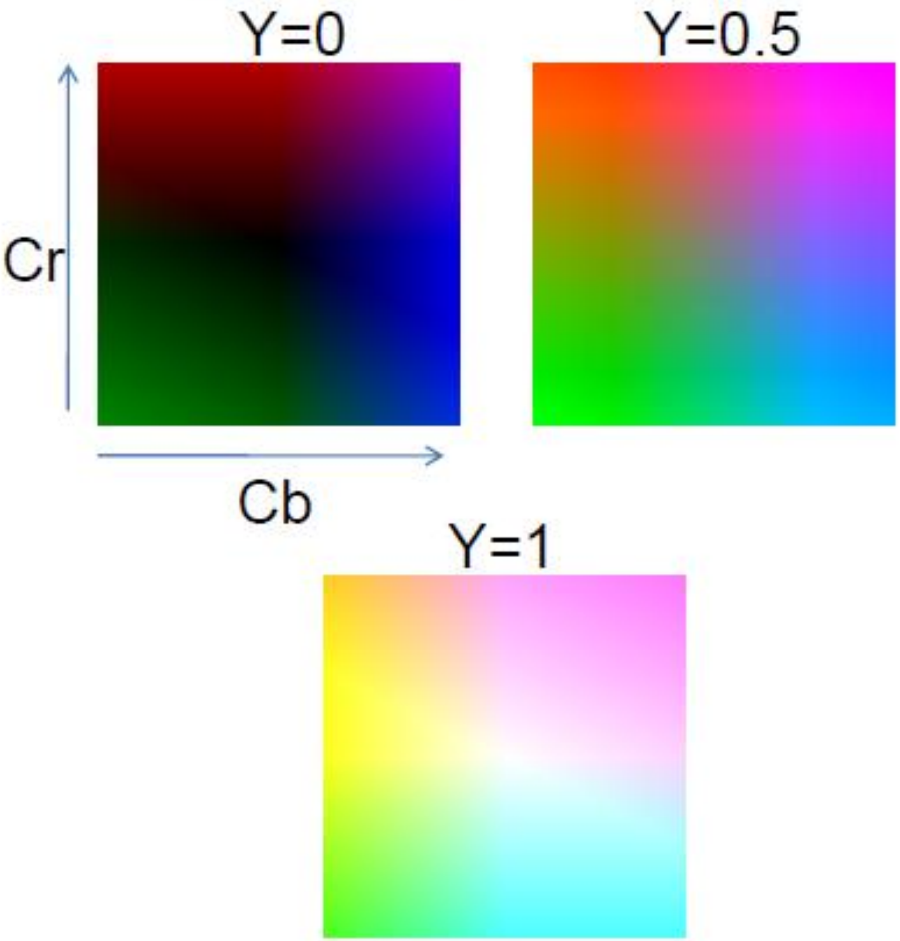
Intuitive color space

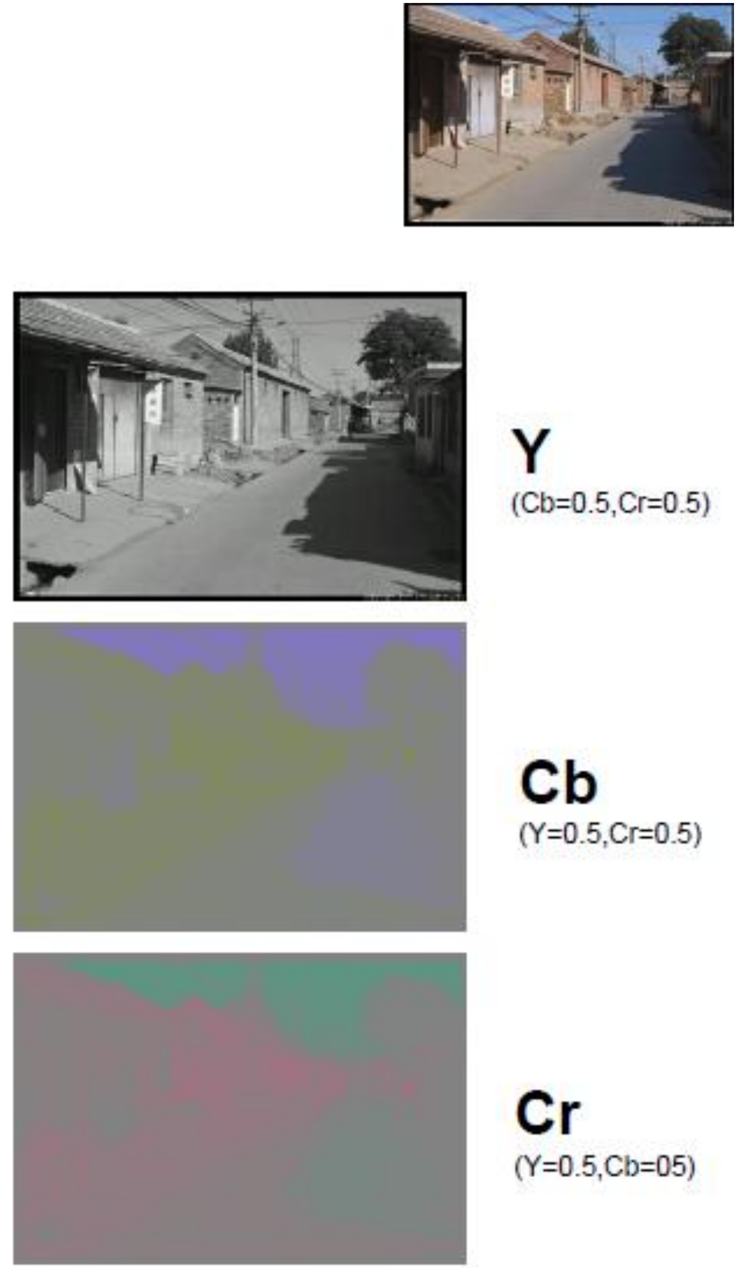




Color spaces: YCbCr

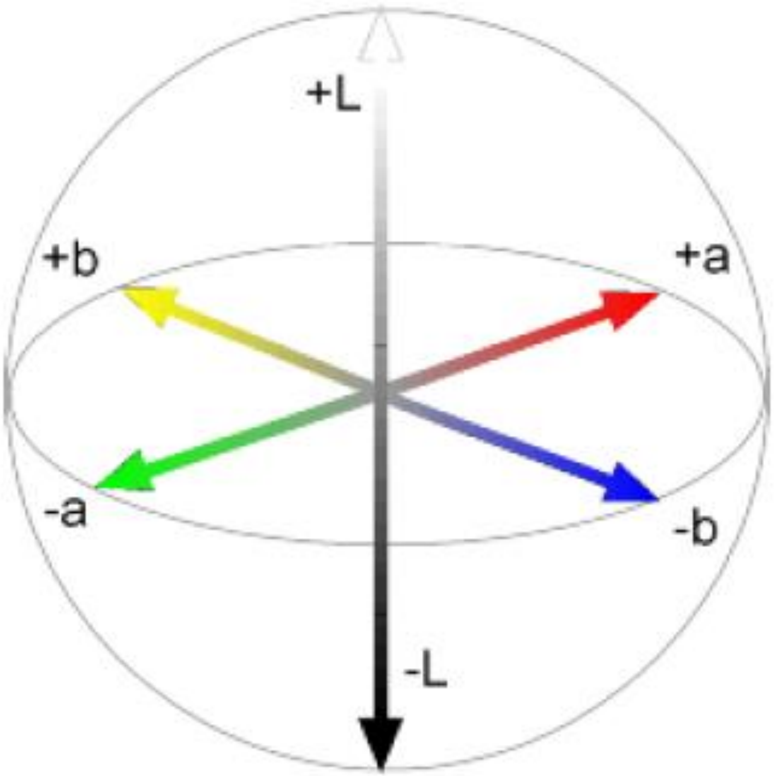
Fast to compute, good for compression, used by TV





Color spaces: L*a*b*

"Perceptually uniform"* color space



L
(a=0,b=0)

a
(L=65,b=0)

b
(L=65,a=0)

James Hays