UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE

Lecture notes:

DIGITAL IMAGE PROCESSING

Pasi Fränti

Abstract: The course introduces to digital image processing techniques such as histogram manipulation, filtering, thresholding, segmentation and shape detection. They are basic components in many applications in image analysis, visual communication, satellite and medical imaging.

Joensuu
8.9. 2003

TABLE OF CONTENTS:

DIGITAL IMAGE PROCESSING

"A picture needs more than thousand operations"

1 DIGITAL IMAGE REPRESENTATION 3

1.1 DIGITIZATION 3
1.2 COLOR IMAGE MODELS 6
1.3 SUMMARY OF IMAGE TYPES 13

2 BASIC IMAGE PROCESSING OPERATIONS 14

2.1 CHANGING RESOLUTION 14
2.2 GRAY-LEVEL TRANSFORMS 16
2.3 FILTERING 21
2.4 DETECTION OF DISCONTINUITIES 27

3 SEGMENTATION 31

3.1 REGULAR BLOCK DECOMPOSITION 31
3.2 SPLITTING AND MERGING 34
3.3 REGION GROWING 35
3.4 EDGE FOLLOWING ALGORITHM 36
3.5 EDGE LINKING 38
3.6 THRESHOLDING 39
3.7 HOUGH TRANSFORM 42

4 BINARY IMAGE PROCESSING 47

4.1 BOUNDARY EXTRACTION 47
4.2 DISTANCE TRANSFORM 48
4.3 SKELETONIZING 51
4.4 COMPONENT LABELING 53
4.5 HALFTONING 53
4.6 MORPHOLOGICAL OPERATIONS 57

5 COLOR IMAGE PROCESSING 61

5.1 COLOR QUANTIZATION OF IMAGES 61
5.2 PSEUDO-COLORING 63

6 APPLICATIONS 66

6.1 RASTER TO VECTOR CONVERSION 66
6.2 OPTICAL CHARACTER RECOGNITION 66
6.3 CELL PARTICLE MOTION ANALYSIS 69
6.4 MAMMOGRAPHY CALCIFICATION DETECTION 71

LITERATURE 73

1 Digital image representation

Digital image is a finite collection of discrete samples (*pixels*) of any observable object. The pixels represent a two- or higher dimensional "view" of the object, each pixel having its own discrete value in a finite range. The pixel values may represent the amount of visible light, infra red light, absortation of x-rays, electrons, or any other measurable value such as ultrasound wave impulses. The image does not need to have any visual sense; it is sufficient that the samples form a two-dimensional spatial structure that may be illustrated as an image.

The images may be obtained by a digital camera, scanner, electron microscope, ultrasound stethoscope, or any other optical or non-optical sensor. Examples of digital image are:

- digital photographs
- satellite images
- radiological images (x-rays, mammograms)
- binary images, fax images, engineering drawings

Computer graphics, CAD drawings, and vector graphics in general are not considered in this course even though their reproduction is a possible source of an image. In fact, one goal of intermediate level image processing may be to reconstruct a model (e.g. vector representation) for a given digital image.

1.1 Digitization

Digital image consists of $N \times M$ pixels, each represented by k bits. A pixel can thus have 2^k different values typically illustrated using a different shades of gray, see Figure 1.1. In practical applications, the pixel values are considered as integers varying from 0 (black pixel) to 2^k-1 (white pixel).

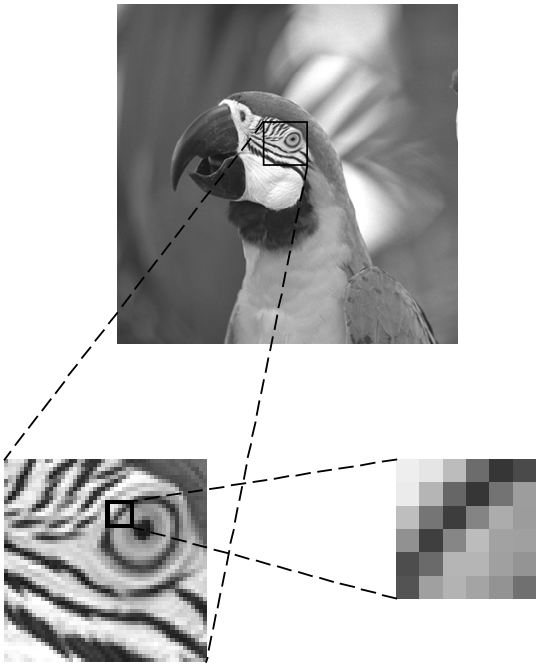


Figure 1.1: Example of digital image.

The images are obtained through a digitization process, in which the object is covered by a two-dimensional sampling grid. The main parameters of the digitization are:

- *Image resolution*: the number of samples in the grid.
- *pixel accuracy*: how many bits are used per sample.

These two parameters have a direct effect on the image quality (Figure 1.2) but also to the storage size of the image (Table 1.1). In general, the quality of the images increase as the resolution and the bits per pixel increase. There are a few exceptions when reducing the number of bits increases the image quality because of increasing the contrast. Moreover, in an image with a very high resolution only very few gray-levels are needed. In some applications it is more important to have a high resolution for detecting details in the image whereas in other applications the number of different levels (or colors) is more important for better outlook of the image. To sum up, if we have a certain amount of bits to allocate for an image, it makes difference how to choose the digitization parameters.

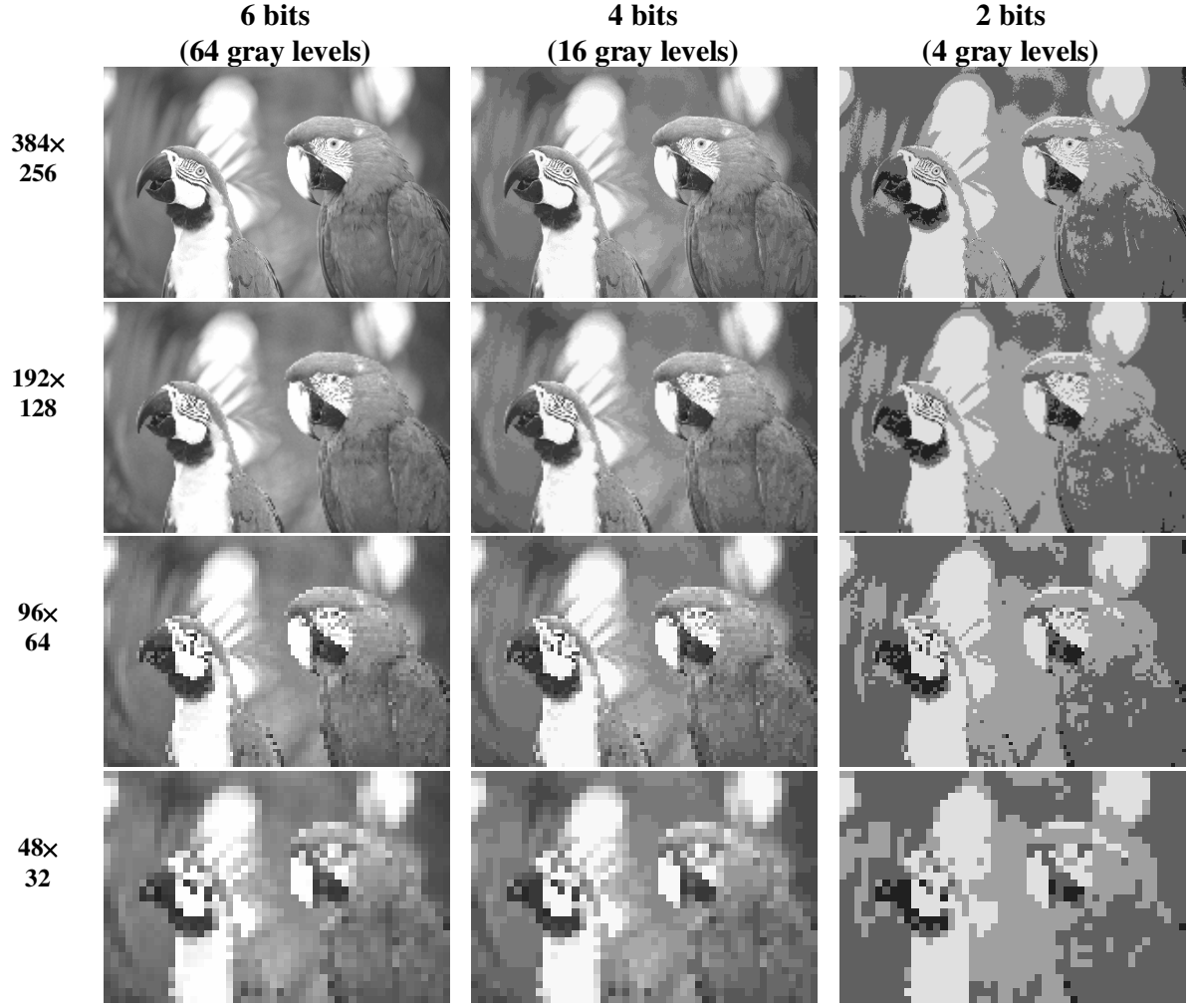


Figure 1.2: The effect of resolution and pixel accuracy to image quality.

Table 1.1 Memory consumption (bytes) with the different parameter values.

Resolution	Bits per pixel: 1	2	4	6	8
32 × 32	128	256	512	768	1,024
64 × 64	512	1,024	2,048	3,072	4,096
128 × 128	2,048	4,096	8,192	12,288	16,384
256 × 256	8,192	16,384	32,768	49,152	65,536
512 × 512	32,768	65,536	131,072	196,608	262,144
1024 × 1024	131,072	262,144	524,288	786,432	1,048,576

The properties of human eye implies some upper limits. For example, it is known that the human eye can observe at most one thousand different gray levels in ideal conditions, but in any practical situations 8 bits per pixel (256 gray level) is usually enough. The required levels decreases even further as the resolution of the image increases (Figure 1.3). In a laser quality printing, as in this lecture notes, even 6 bits (64 levels) results in quite satisfactory result. On the other hand, if the application is e.g. in medical imaging or in cartography, the visual quality is not the primary concern. For example, if the pixels represent some physical measure and/or the image will be analyzed by a computer, the additional accuracy may be useful. Even if human eye cannot detect any differences, computer analysis may recognize the differences.

The requirement of the spatial resolution depends both on the usage of the image and the image content. If the default printing (or display) size of the image is known, the scanning resolution can be chosen accordingly so that the pixels are not seen and the image appearance is not jagged (blocky). However, the final reproduction size of the image is not always known but images are often achieved just for "later use". Thus, once the image is digitized it will most likely (according to Murphy's law) be later edited and enlarged beyond what was allowed by the original resolution. The image content sets also some requirements to the resolution. If the image has very fine structure exceeding the sampling resolution, it may cause so-called aliasing effect (Figure 1.4) where the digitized image has patterns that does not exists in the original (Figure 1.5).

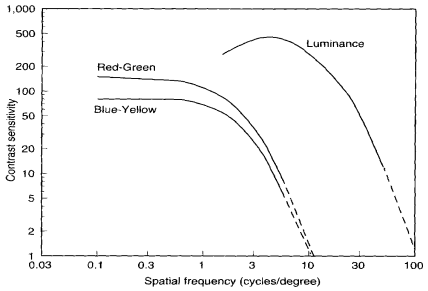


Figure 1.3: Sensitivity of the eye to intensity changes.

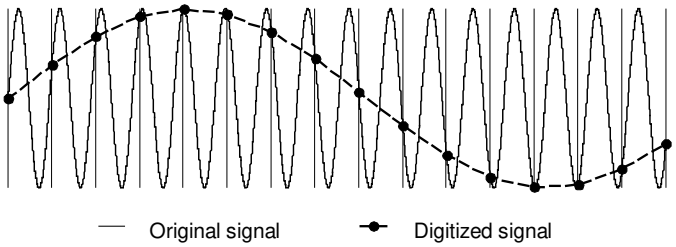


Figure 1.4: Aliasing effect for 1-dimensional signal.

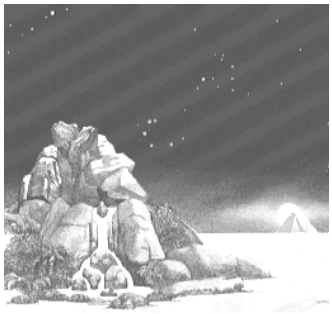


Figure 1.5: Aliasing effect (Moiré) for an image.

1.2 Color image models

Visible light is composed of relatively narrow band of frequencies in the electromagnetic energy spectrum - approximately between 400 and 700 nm. A green object, for example, reflects light with wavelength primarily in the 500 to 570 nm range, while absorbing most of the energy at other wavelengths. A white object reflects light that is relatively balanced in all visible wavelengths.

According to the theory of the human eye, all colors are seen as variable combinations of the three so-called *primary colors* red (*R*), green (*G*), and blue (*B*). For the purpose of standardization, the CIE (Commission Internationale de l'Eclairage - International Commission on Illumination) designated in 1931 the following specific wavelength values to the primary colors:

blue (*B*) = 435.8 nm
green (*G*) = 546.1 nm
red (*R*) = 700.0 nm

The primary colors can be added to produce the secondary colors magenta (red + blue), cyan (green + blue), and yellow (red + green), see Figure 1.6. Mixing all the three primary colors results in white. Color television reception is based on this three color system with the additive nature of light. There exists several useful color models: RGB, CMY, YUV, YIQ, and HSI - just to mention a few.

RGB color model

The colors of the RGB model can be described as a triple (*R*, *G*, *B*), so that $R, G, B \in [0,1]$. The RGB color space can be considered as a three-dimensional unit cube, in which each axis represents one of the primary colors, see Figure 1.6. Colors are points inside the cube defined by its coordinates. The primary colors thus are red=(1,0,0), green=(0,1,0), and blue=(0,0,1). The secondary colors of RGB are cyan=(0,1,1), magenta=(1,0,1) and yellow=(1,1,0). The nature of the RGB color system is additive in the sense how adding colors makes the image brighter. Black is at the origin, and white is at the corner where $R=G=B=1$. The gray scale extends from black to white along the line joining these two points. Thus a shade of gray can be described by (x,x,x) starting from black=(0,0,0) to white=(1,1,1).

The colors are often normalized as given in (1.1). This normalization guarantees that $r+g+b=1$.

$$r = \frac{R}{(R+B+G)}$$
$$g = \frac{G}{(R+B+G)} \qquad (1.1)$$
$$b = \frac{B}{(R+B+G)}$$

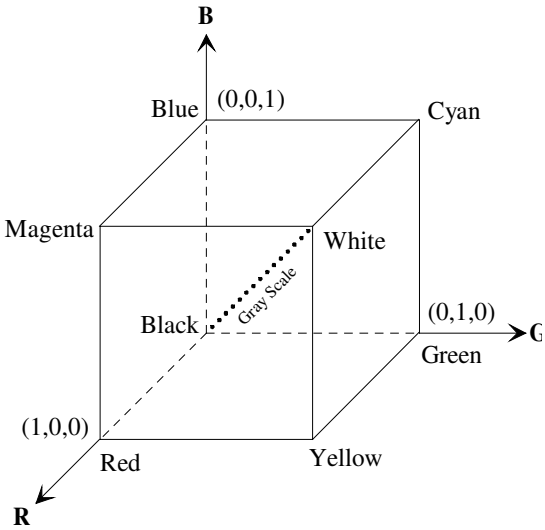


Figure 1.6: RGB color cube.

CMY color model

The CMY color model is closely related to the RGB model. Its primary colors are *C* (cyan), *M* (magenta), and *Y* (yellow). I.e. the secondary colors of RGB are the primary colors of CMY, and vice versa. The RGB to CMY conversion can be performed by

$$C = 1 - R$$
$$M = 1 - G \qquad (1.2)$$
$$Y = 1 - B$$

The scale of *C*, *M* and *Y* also equals to unit: $C, M, Y \in [0,1]$. The CMY color system is used in offset printing in a subtractive way, contrary to the additive nature of RGB. A pixel of color Cyan, for example, reflects all the RGB other colors but red. A pixel with the color of magenta, on the other hand, reflects all other RGB colors but green. Now, if we mix cyan and magenta, we get blue, rather than white like in the additive color system.

YUV color model

The basic idea in the YUV color model is to separate the color information apart from the brightness information. The components of YUV are:

$$
\begin{aligned}
Y &= 0.3 \cdot R + 0.6 \cdot G + 0.1 \cdot B \\
U &= B - Y \\
V &= R - Y
\end{aligned}
\tag{1.3}
$$

Y represents the *luminance* of the image, while U,V consists of the color information, i.e. *chrominance*. The luminance component can be considered as a gray-scale version of the RGB image, see Figure 1.7 for an example. The advantages of YUV compared to RGB are:

- The brightness information is separated from the color information
- The correlations between the color components are reduced.
- Most of the information is collected to the Y component, while the information content in the U and V is less. This can be seen in Figure 1.7, where the contrast of the Y component is much greater than that of the U and V components.

The latter two properties are beneficial in image compression. This is because the correlations between the different color components are reduced, thus each of the components can be compressed separately. Besides that, more bits can be allocated to the Y component than to U and V. The YUV color system is adopted in the JPEG image compression standard.

YIQ color model

YIQ is a slightly different version of YUV. It is mainly used in North American television systems. Here Y is the luminance component, just as in YUV. I and Q correspond to U and V of YUV color systems. The RGB to YIQ conversion can be calculated by

$$
\begin{aligned}
Y &= 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \\
I &= 0.596 \cdot R - 0.275 \cdot G - 0.321 \cdot B \\
Q &= 0.212 \cdot R - 0.523 \cdot G + 0.311 \cdot B
\end{aligned}
\tag{1.4}
$$

The YIQ color model can also be described corresponding to YUV:

$$
\begin{aligned}
Y &= 0.3 \cdot R + 0.6 \cdot G + 0.1 \cdot B \\
I &= 0.74 \cdot V - 0.27 \cdot U \\
Q &= 0.48 \cdot V + 0.41 \cdot U
\end{aligned}
\tag{1.5}
$$

HSI color model

The HSI model consists of *hue* (H), *saturation* (S), and *intensity* (I). Intensity corresponds to the luminance component (Y) of the YUV and YIQ models. Hue is an attribute associated with the dominant wavelength in a mixture of light waves, i.e. the dominant color as perceived by an observer. Saturation refers to relative purity of the amount of white light mixed with hue. The advantages of HSI are:

- The intensity is separated from the color information (the same holds for the YUV and YIQ models though).
- The hue and saturation components are intimately related to the way in which human beings perceive color.

Whereas RGB can be described by a three-dimensional cube, the HSI model can be described by the color triangle shown in Figure 1.8. All colors lie inside the triangle whose vertices are defined by the three initial colors. Let us draw a vector from the central point of the triangle to the color point *P*. Then hue (*H*) is the angle of the vector with respect to the red axis. For example 0° indicates red color, 60° yellow, 120° green, and so on. Saturation (*S*) is the degree to which the color is undiluted by white and is proportional to the distance to the center of the triangle.

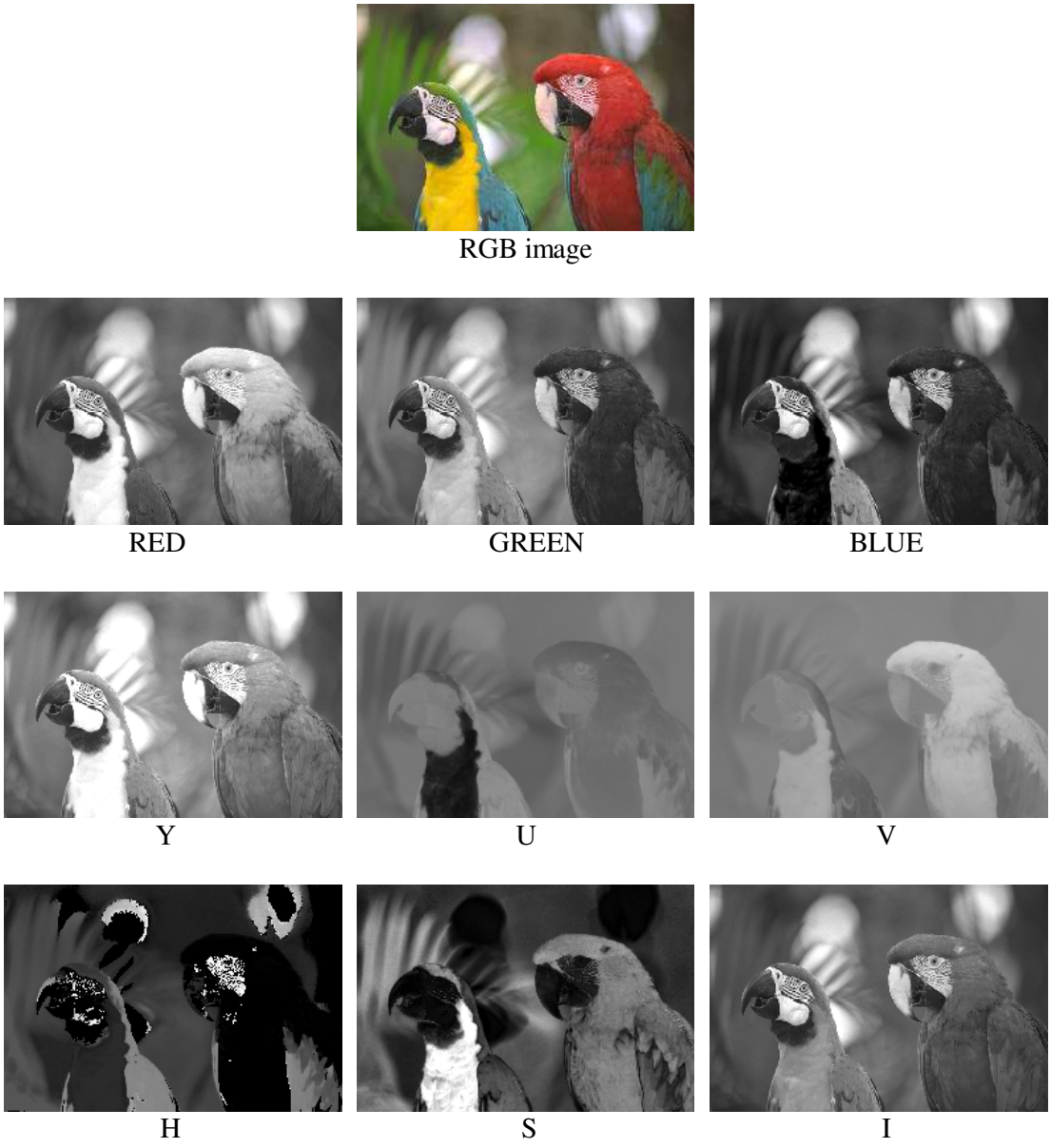


Figure 1.7: Color components of test image Parrots in RGB, YUV, and HSI color space.

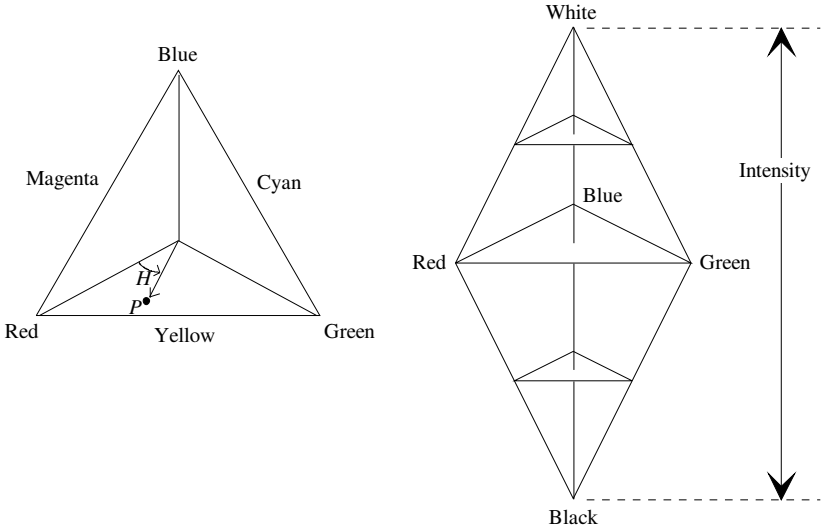


Figure 1.8: (left) HSI color triangle; (right) HSI color solid.

HSI can be obtained from the RGB colors as follows. Intensity of HSI model is defined:

$$I = \frac{1}{3} \cdot (R + G + B) \tag{1.6}$$

Let W=(1/3, 1/3, 1/3) be the central point of the triangle, see Figure 1.9. Hue is then defined as the angle between the line segments WP_R and WP. In vector form, this is the angle between the vectors (p_R-w) and (p-w). Here (p-w) = (r-1/3, g-1/3, b-1/3), and (p_R-w) = (2/3, -1/3, -1/3). The inner product of two vectors are calculated $(x) \cdot (y) = \|x\| \|y\| \cos\alpha$, where α denotes the angle between the vectors. The following equations thus holds for $0° \le H \le 180°$:

$$(p - w) \cdot (p_R - w) = \|p - w\| \|p_R - w\| \cos H \tag{1.7}$$

The left part of the equation is:

$$\begin{aligned} (p - w) \cdot (p_R - w) &= \frac{2}{3}\left(r - \frac{1}{3}\right) - \frac{1}{3}\left(g - \frac{1}{3}\right) - \frac{1}{3}\left(b - \frac{1}{3}\right) \\ &= \frac{2r - g - b}{3} \\ &= \frac{2R - G - B}{3(R + G + B)} \end{aligned} \tag{1.8}$$

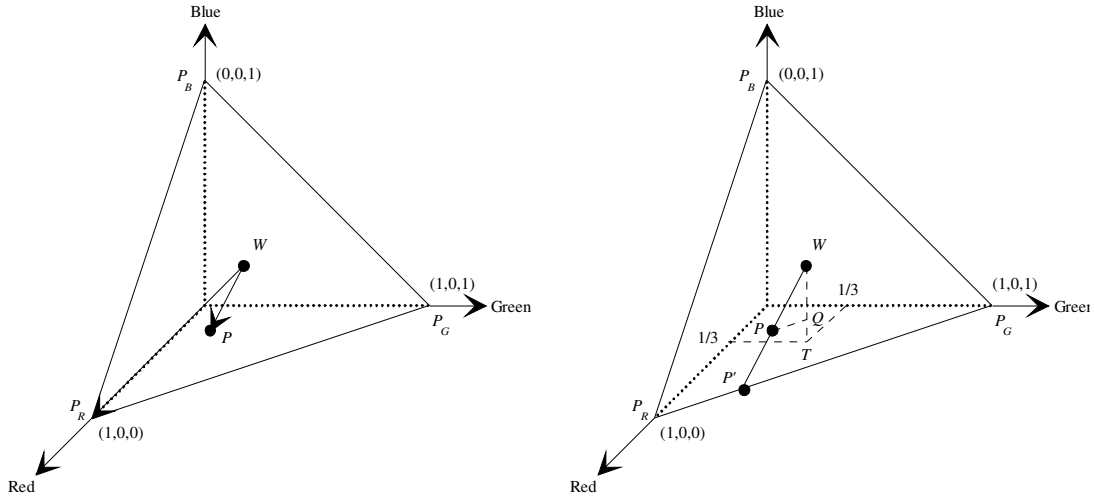


Figure 1.9: Details of the HSI color triangle needed to obtain expressions for hue and saturation.

The right part evaluates as follows:

$$\begin{aligned}\|p-w\| &= \sqrt{\left(r-\frac{1}{3}\right)^2+\left(g-\frac{1}{3}\right)^2+\left(b-\frac{1}{3}\right)^2} \\ &= \sqrt{\frac{3\left(R^2+G^2+B^2\right)-(R+G+B)^2}{3(R+G+B)^2}}\end{aligned} \tag{1.9}$$

$$\|p_R-w\| = \sqrt{\left(\frac{2}{3}\right)^2+\left(-\frac{1}{3}\right)^2+\left(-\frac{1}{3}\right)^2} = \sqrt{\frac{2}{3}} \tag{1.10}$$

Now we can determine H:

$$\begin{aligned}H &= \cos^{-1}\frac{(p-w)\cdot(p_R-w)}{\|p-w\|\|p_R-w\|} \\ &= \cos^{-1}\left[\frac{2(R-G-B)}{3(R+B+G)}\sqrt{\frac{9(R+G+B)^2}{6(R+G+B)-2(R+G+B)^2}}\right] \\ &= \cos^{-1}\frac{\frac{1}{2}(R-G)+\frac{1}{2}(R-B)}{\sqrt{(R-G)^2+(R-B)(G-B)}}\end{aligned} \tag{1.11}$$

Equation (1.11) yields values of H in the interval $0° \le H \le 180°$. If $B > G$, then H has to be greater than 180°. So, whenever $B > G$, we simply let $H = 360° - H$. Finally, H is normalized to the scale [0,1]:

$$H' = \frac{H}{360^\circ} \tag{1.12}$$

The next step is to derive an expression for saturation S in terms of a set of RGB values. It is defined as the relative distance between the points P and W in respect to the distance between P' and W:

$$S = \frac{|WP|}{|WP'|} \tag{1.13}$$

Here P' is obtained by extending the line WP until it intersects the nearest side of the triangle, see Figure 1.9. Let T be the projection of P onto the rg plane, parallel to the b axis and let Q be the projection of P onto WT, parallel to the rg plane. The equation then evaluates to:

$$S = \frac{|WP|}{|WP'|} = \frac{|WQ|}{|WT|} = \frac{|WT| - |QT|}{|WT|} = 1 - \frac{|QT|}{|WT|} = 1 - \frac{b}{1/3} = 1 - 3b \tag{1.14}$$

The same examination can be made for whatever side of the triangle that is closest to the point P. Thus S can be obtained for any point lying on the HSI triangle as follows:

$$\begin{aligned} S &= 1 - \min(3r, 3g, 3b) \\ &= 1 - \min\left[\frac{3R}{(R+G+B)}, \frac{3G}{(R+G+B)}, \frac{3B}{(R+G+B)}\right] \\ &= 1 - \frac{3}{(R+G+B)}\min(R,G,B) \end{aligned} \tag{1.15}$$

The RGB to HSI conversion can be summarized as follows:

$$\begin{aligned} H &= \frac{1}{360^\circ} \cdot \cos^{-1}\left[\frac{1/2 \cdot [(R-G)+(R-B)]}{\sqrt{(R-G)^2 + (R-B)(G-B)}}\right], \text{ if } B \le G \\ H &= 1 - \frac{1}{360^\circ} \cdot \cos^{-1}\left[\frac{1/2 \cdot [(R-G)+(R-B)]}{\sqrt{(R-G)^2 + (R-B)(G-B)}}\right], \text{ otherwise} \\ \mathrm{S} &= 1 - \frac{3}{(R+G+B)} \cdot \min(R,G,B) \\ I &= \frac{1}{3} \cdot (R+G+B) \end{aligned} \tag{1.16}$$

The HSI to RGB conversion is left as an exercise, see [Gonzalez & Woods, 1992, p.229-235] for details.

1.3 Summary of image types

In the following sections we focus on gray-scale images because it is the basic image type. All other image types can be illustrated as a gray-scale or a set of gray-scale images. In principle, the image processing methods designed for gray-scale images can be straightforwardly generalized to color and video images by processing each color plane, or image frame separately. Color images consists of several (commonly three) color components (typically RGB or YUV), whereas the number of image frames in a video sequence is practically speaking unlimited.

Good quality photographs needs 24 bits per pixel, which can represent over 16 million different colors. However, in some simple graphical applications the number of colors can be greatly reduced. This would put less memory requirements to the display devices and usually speeds up the processing. For example, 8 bits (256 colors) is often sufficient to represent the icons in Windows desktop if the colors are properly chosen. A straightforward allocation of the 8 bits, e.g. (3, 3, 2) bits to red, green and blue respectively, is not sufficient. Instead, a *color palette* of 256 specially chosen colors is generated to approximate the image. Each original pixel is mapped to its nearest palette color and the index of this color is stored instead of the *rgb*-triple (Figure 1.10). GIF images are examples of color palette images.

Binary image is a special case of gray-scale image with only two colors: black and white. Binary images have applications in engineering, cartography and in fax machines. They may also be a result of a thresholding, and therefore a point of further analysis of another non-binary image. Binary images have relatively small storage requirements and using the standard image compression algorithms they can be stored very efficiently. There are special image processing methods suitable only for binary images. The main image types are summarized in Table 1.2.

Table 1.2: Summary of image types.

Image type	Typical bpp	No. of colors	Common file formats
Binary image	1	2	JBIG, PCX, GIF, TIFF
Gray-scale	8	256	JPEG, GIF, PNG, TIFF
Color image	24	$16.6 \cdot 10^6$	JPEG, PNG, TIFF
Color palette image	8	256	GIF, PNG
Video image	24	$16.6 \cdot 10^6$	MPEG

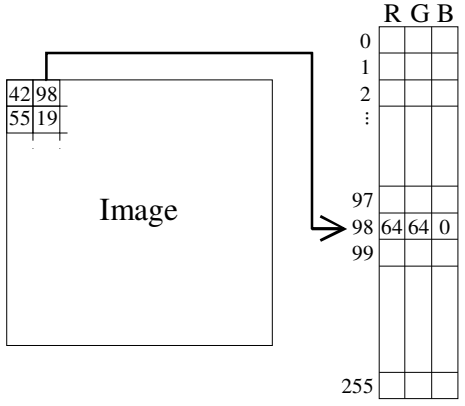


Figure 1.10: Look-up table for a color palette image.

2 Basic image processing operations

In this section we study techniques for resolution enhancement and reduction, adjusting color distribution (histogram and LUT operations), and spatial operations such as filtering and edge detection. We operate with 8 bpp gray-scale images each constrained to the range [0, 255], unless otherwise noted. Generalization to color images will be considered in Section 5.

2.1 Changing resolution

A reduced resolution version of a given image is sometimes needed for a preview purpose, for example. A preview image (or *thumbnail*) must be small enough to allow fast access but also with sufficient quality so that the original image is still recognizable. A smaller resolution copy is also needed if the image is embedded into another application (or printed) using smaller size as the full resolution would allow. Sometimes the image resolution may be reduced just for saving memory.

Resolution reduction is formally defined as follows: given an image of $N \times M$ pixels, generate an image of size $N/c \times M/c$ pixels (where c is a zooming factor) so that the visual content of the image is preserved as well as possible. There are two alternative strategies: (1) *sub sampling*, (2) *averaging*. In both cases, the input image is divided into blocks of $c \times c$ pixels. For each block, one representative pixel is generated to the output image (Figure 2.1). In sub sampling, any of the input pixels is chosen, e.g. the upper leftmost pixel in the block. In averaging, the pixel depends on the values of all input pixels in the block. It could be chosen as the average, weighted average, or the median of the pixels. Averaging results in smoother image whereas sub sampling preserves more details.

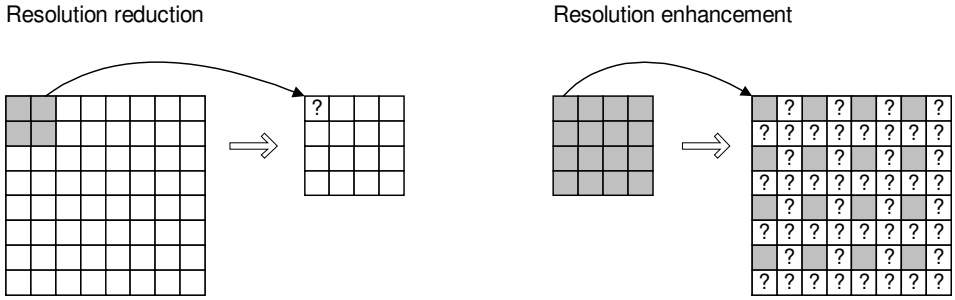


Figure 2.1: Changing image resolution by the factor of two.

The resolution of an image must sometimes be increased, e.g. when the image is zoomed for viewing. For each input pixels there are $c \times c$ output pixels to be generated. This is opposite to resolution reduction (see Figure 2.1). A straightforward method simply takes copies of the input pixel but this results in a jagged (blocky) image where the pixels are clearly visible. A more sophisticated method known as *bilinear interpolation* generates the unknown pixel values by taking the linear combination of the four nearest known pixel values (Figure 2.2). The value of a pixel $x_{i,j}$ at the location (i,j) can be calculated as:

$$f(x) = a + \Delta i(b-a) + \Delta j(c-a) + \Delta i \Delta j(a-b-c+d) \qquad (2.1)$$

where a, b, c, d are the nearest known pixel values to x; Δi and Δj define the relative distance from a to x (varying from 0 to 1). The result is a smoother image compared to straightforward copying of a single pixel value. Examples of resolution reduction and resolution increasing by interpolation is illustrated in Figures 2.3 and 2.4.

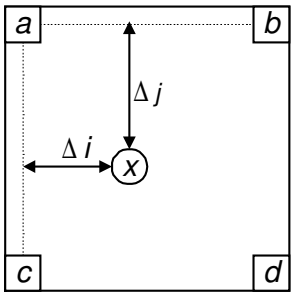


Figure 2.2: Bilinear interpolation of pixel x using the four known pixels a, b, c, d.

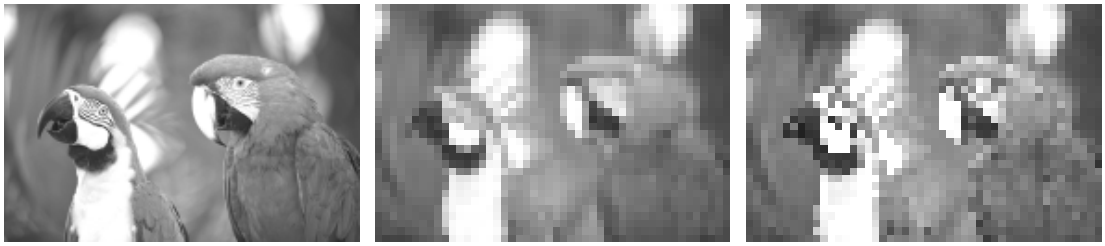


Figure 2.3: Example of resolution reduction 384×256 → 48×32: original image (left), resolution reduced by averaging (middle), and by sub sampling (right).

Figure 2.4: Bilinear interpolation using two 48×32 images obtained using averaging (top), sub sampling (below). Interpolated images: original (left), 92×64 (middle), 192×128 (right).

2.2 Gray-level transforms

A general gray-level transform can be described as

$$y = f(x) \tag{2.2}$$

where x is the original pixel value and y is the result after transform (Figure 2.5). The function f depends only on the pixel value, and some global information in the image given by the frequency distribution of the pixels (i.e. *histogram*). The transform can also use prior knowledge of the image given by the user of the image processing application. The transform, however, is independent from the neighboring pixel values.

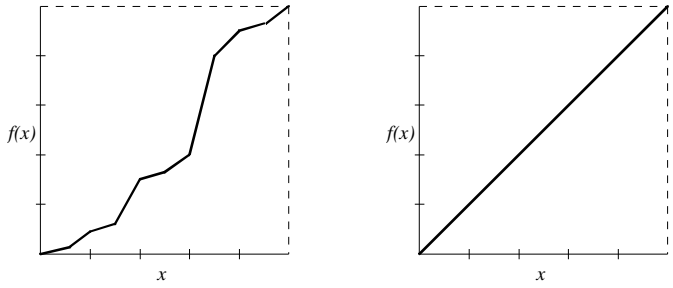


Figure 2.5: Illustration of arbitrary transform (left), identity transform (right).

Constant addition and negation

The simplest form of global transform are *constant addition* (also known as *DC-shift*) and *negation*, see Figures 2.6. The former is used to enhance dark images. The latter can be used for displaying medical images and photographs on screen with monochrome positive film with the idea of using the resulting negatives as normal slides.

Constant addition: $f(x) = x + c$ (2.3)

Negation: $f(x) = c - x$ (2.4)

Contrast stretching

The visible quality of a low contrast image can be improved by *contrast stretching*. This is based on an assumption that the dynamic scale, or the relevant information of the image is concentrated between two pixel values x_1 and x_2, and that these values are already prior knowledge. The scale of the histogram in the range $[x_1, x_2]$ is enlarged while the scales below x_1 and above x_2 are compressed, see Figure 2.6.

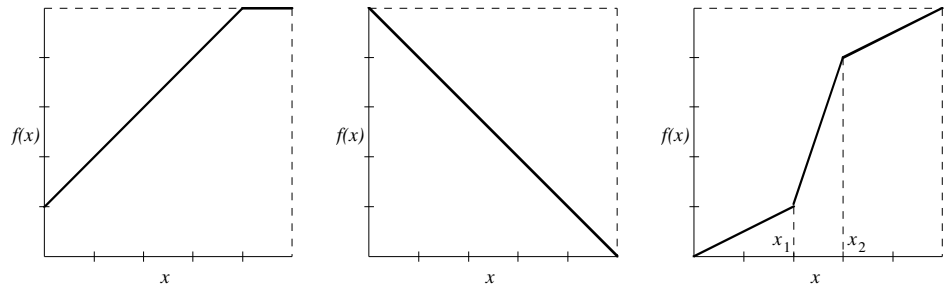


Figure 2.6: Constant addition (left), negation (middle), contrast stretching (right).

Range compression

A counter example to the previous situation appear when the dynamic range of an image far exceeds the capability of the display device. In this case only the brightest parts of the image are visible. An effective way to compress the dynamic range of pixel values is to perform the following intensity transform:

$$\textit{Range compression}: \qquad f(x) = c \cdot \log(\, 1 + |x| \,) \tag{2.5}$$

Gray-level slicing

Suppose that the gray-level values that are of particular interest are known. These pixel values can then be separated from the background by the *gray-level slicing technique*, see Figure 2.7. The method assigns a bright pixel value to the pixels of interest, and a dark value to the rest of the pixels belonging to the "background". In another variant the so-called background pixels are leaved untouched, see Figure 2.7. In the former case, the gray-level slicing technique thus performs a kind of *thresholding* (see the next paragraph).

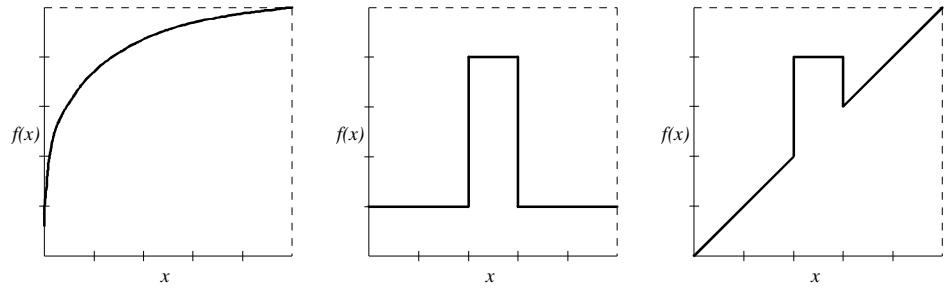


Figure 2.7: Range compression (left), variants of gray-level slicing (middle and right).

Quantization and global thresholding

Quantization is a technique where the number of gray-levels are reduced. Let's suppose an image consisting of 8 bits per pixel and thus having 256 different gray levels. The image can be quantized for example to 16 levels simply by taking the 4 most significant bits of the pixel values. This operation performs a *uniform quantization*, where the range of each gray-level value is equal, see Figure 2.8. The applications of quantization can be found in image compression. It can also be used as a basic tool in image segmentation, or to help a human observer to detect possible objects in an image that is otherwise not seen because of the smoothness of the image. Quantization generates artificial edges into the images which may be

of help in the analysis. Quantization is also necessary when displaying 256-level gray-scale images on a VGA-display that can only show a maximum of 64 gray-levels.

Thresholding performs a two-level quantization of the image, see Figure 2.8. The purpose of thresholding is to classify the image pixels according to some threshold criterion. The operation splits the pixels into two (or more) groups in the same manner as the gray-level slicing.

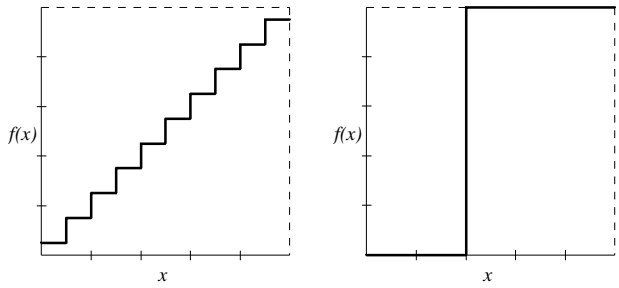


Figure 2.8. Quantization (left), thresholding (right).

Histogram equalization

Sometimes the histogram of an image contains mostly dark pixels; this is the case of an insufficiently exposed photograph (Figure 2.9). The image can be enhanced by constant addition but *histogram equalization* is generally more efficient technique for this purpose (Figure 2.10). It is also applicable whenever the contrast of the image is too small for whatever reason. The idea of the method is to spread the histogram as evenly as possible over the full intensity scale. This is done by calculating cumulative sums of the pixel samples for each gray level value x in the histogram. The sum implies the number of gray levels that should be allocated to the range [0, x], and is proportional to the cumulative frequency $t(x)$, and to the total number of gray levels g:

$$f(x) = g \cdot \frac{t(x)}{n} - 1 \qquad (2.6)$$

Here n is the total frequency, i.e. the number of pixels in the image. An example of histogram equalization is given in Table 2.1 and in Figure 2.10.

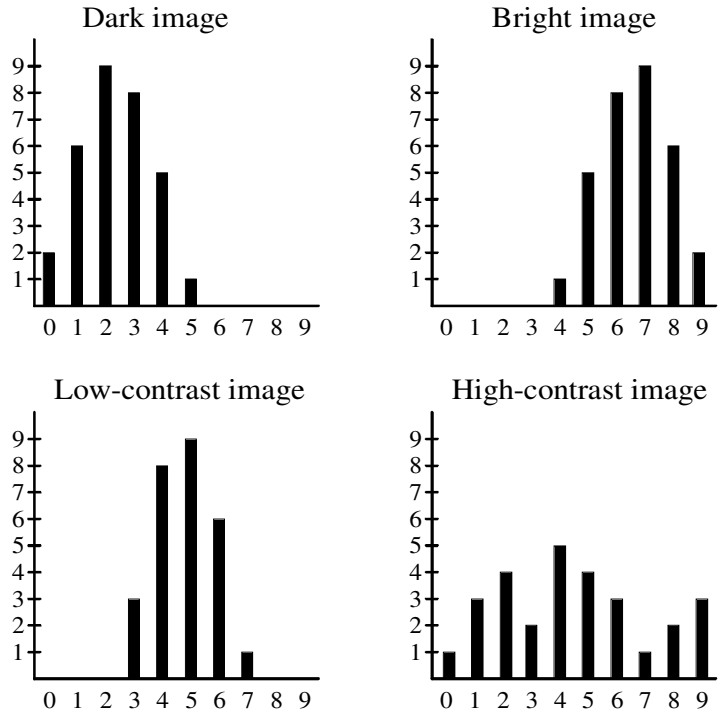


Figure 2.9: Histograms corresponding to four basic image types.

Table 2.1: Example of histogram equalization.

Original value x	**Frequency** $\#(x)$	**Cumulative frequency** $t(x)$	**New value** $f(x)$
0	1	1	0
1	9	10	2
2	8	18	5
3	6	24	7
4	1	25	7
5	1	26	8
6	1	27	8
7	1	28	8
8	2	30	9
9	0	30	9

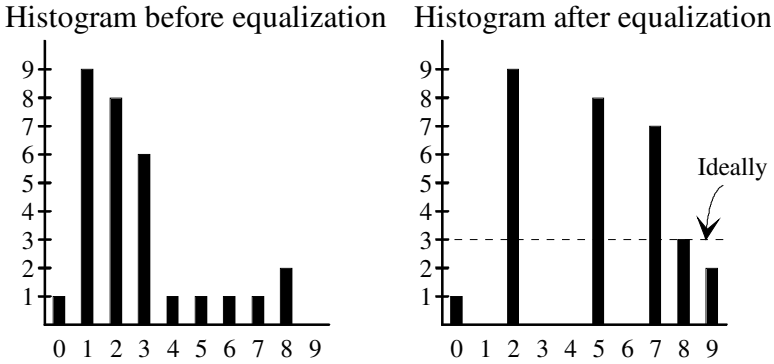


Figure 2.10: Example of histogram equalization.

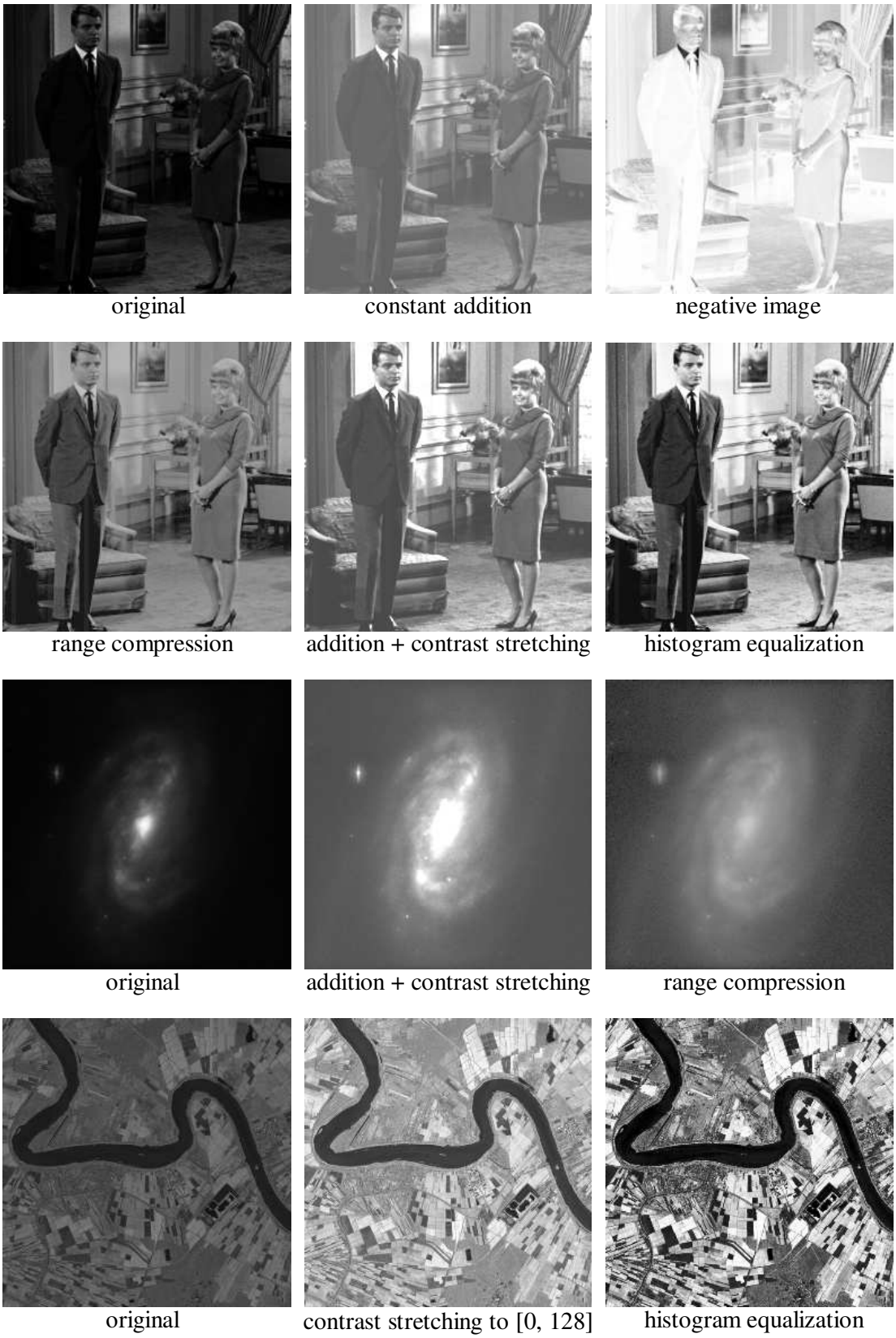


Figure 2.11: Illustrations of some gray-level transforms. Contrast stretching was applied with the factor k=2, range compression factor was c=25, and addition constant 100.

2.3 Filtering

Filtering is an image processing operation where the value of a pixel depends on the values of its neighboring pixels. Each of the pixels are processed separately with a predefined *window* (or *template,* or *mask*). Weighted sum of the pixels inside the window is calculated using the weights given by a mask, see Figure 2.12. The result of the sum replaces the original value in the processed image:

$$f(x) = \sum_{i=1}^{9} w_i \cdot x_i \tag{2.7}$$

In the case of border pixels, the part of the mask lying outside of the image is assumed to have the same pixel values as that of the border pixels, see Figure 2.13. Note that the filtering is a parallel operation, i.e. the neighboring values used in the calculations are always taken from the original image, not from the processed image.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figure 2.12: General mask for filtering with a 3×3 window.

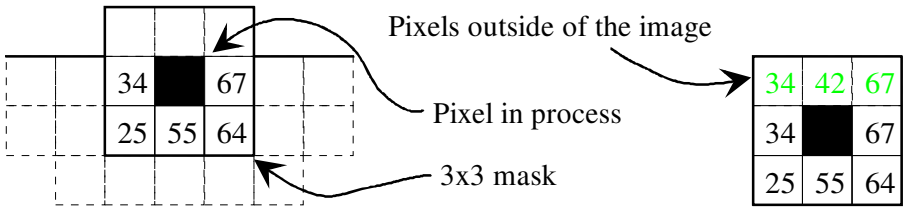


Figure 2.13: Example of filtering operation in the case of border pixel.

Low-pass filtering

Low-pass filtering (or *averaging filtering*, or *smoothing*) reduces the *high frequency components* (or *noise*) in the image by averaging the pixel values over a small region (block). see Figure 2.14. This reduces noise and makes the image generally smoother, especially near the edges. The level of smoothing can be changed by increasing the size of the window, see Figure 2.15.

High-pass filtering

High-pass filtering is the opposite operation to low-pass filtering. The *low frequency components* are eliminated and only the high frequency components in the image are retained.

The operation can be applied in image enhancement by adding the result of the filtering to the original image. This is known as *sharpening*. It enhances the pixels near edges and makes it easier to observe details in the image, see Figure 2.16. The use of negative weights in the mask may result in negative values, thus the pixel values must be scaled back to [0, 255].

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

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 2.14: Masks for low-pass (left) and high-pass filters (right).

Median filtering

Low-pass and high-pass filters (as well as many other filters presented in Section 2.3) are in the class of *linear filters*; they can always be described by a weighting mask. *Median filtering*, on the other hand, belong to a class of *rank filters*. Here the pixels within the window are *ranked* (or *sorted*) and the result of the filtering is chosen according to the ordering of the pixel values. In median filtering the new value of a pixel is the median of the pixel values in the window. The parameter of the filtering is the size and the shape of the filtering window (mask).

The median filter is used for removing noise. It can remove isolated impulsive noise and at the same time it preserves the edges and other structures in the image. Contrary to average filtering it does not smooth the edges. Two examples of the so-called *salt and pepper noise* and their removal are shown in Figure 2.17. In the left image each pixel has been affected by the noise with a probability of 5 %, and in the right example with a probability of 20 %. A pixel that is affected by the noise is replaced either with a black or a white pixel.

Detailed examples of the three filtering schemes are demonstrated in Table 2.2.

Table 2.2: Pixel-level examples of smoothing, sharpening and median filtering.

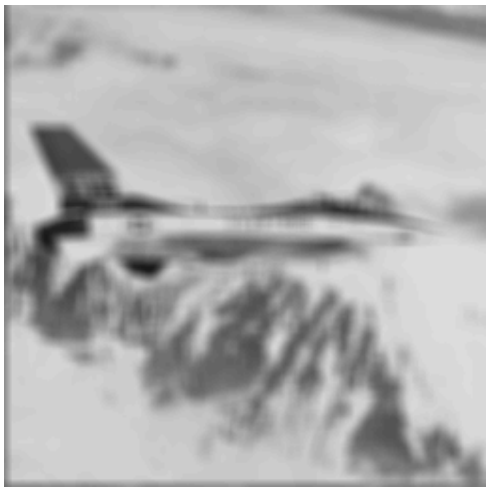
ORIGINAL IMAGE	SMOOTHING	SHARPENING	MEDIAN FILTERING
Flat			
10 10 10 10 10	10 10 10 10 10	10 10 10 10 10	10 10 10 10 10
10 10 10 10 10	10 10 10 10 10	10 10 10 10 10	10 10 10 10 10
10 10 10 10 10	10 10 10 10 10	10 10 10 10 10	10 10 10 10 10
10 10 10 10 10	10 10 10 10 10	10 10 10 10 10	10 10 10 10 10
10 10 10 10 10	10 10 10 10 10	10 10 10 10 10	10 10 10 10 10
Random texture			
11 13 18 16 16	12 14 14 15 15	1 8 50 23 28	11 13 13 16 16
14 11 13 12 12	14 15 15 15 14	10 0 0 0 0	14 14 13 16 16
19 18 19 12 17	15 15 14 14 13	52 44 62 0 53	14 14 13 13 12
14 15 13 15 10	15 16 15 15 14	2 7 0 16 0	14 15 15 15 14
13 13 19 15 14	13 15 15 15 13	9 0 53 16 19	13 13 15 15 14
Two impulses			
10 10 10 10 10	11 11 11 10 10	0 0 0 10 10	10 10 10 10 10
10 **20** 10 10 10	11 11 11 10 10	0 100 0 10 10	10 10 10 10 10
10 10 10 10 10	11 11 12 11 11	0 0 0 0 0	10 10 10 10 10
10 10 10 **20** 10	10 10 11 11 11	10 10 0 100 0	10 10 10 10 10
10 10 10 10 10	10 10 11 11 11	10 10 0 0 0	10 10 10 10 10
Line (horizontal)			
10 10 10 10 10	10 10 10 10 10	10 10 10 10 10	10 10 10 10 10
10 10 10 10 10	13 13 13 13 13	0 0 0 0 0	10 10 10 10 10
20 20 20 20 20	13 13 13 13 13	80 80 80 80 80	10 10 10 10 10
10 10 10 10 10	13 13 13 13 13	0 0 0 0 0	10 10 10 10 10
10 10 10 10 10	10 10 10 10 10	10 10 10 10 10	10 10 10 10 10
Edge (vertical)			
10 10 10 **20 20**	10 10 13 17 20	10 10 0 50 10	10 10 10 20 20
10 10 10 **20 20**	10 10 13 17 20	10 10 0 50 10	10 10 10 20 20
10 10 10 **20 20**	10 10 13 17 20	10 10 0 50 10	10 10 10 20 20
10 10 10 **20 20**	10 10 13 17 20	10 10 0 50 10	10 10 10 20 20
10 10 10 **20 20**	10 10 13 17 20	10 10 0 50 10	10 10 10 20 20
Edge (horizontal)			
10 10 10 10 10	10 10 10 10 10	10 10 10 10 10	10 10 10 10 10
10 10 10 10 10	10 10 10 10 10	10 10 10 10 10	10 10 10 10 10
10 10 10 10 10	13 13 13 13 13	0 0 0 0 0	10 10 10 10 10
20 20 20 20 20	17 17 17 17 17	50 50 50 50 50	20 20 20 20 20
20 20 20 20 20	20 20 20 20 20	10 10 10 10 10	20 20 20 20 20
Edge (diagonal)			
10 10 10 10 10	10 10 10 11 12	10 10 10 0 0	10 10 10 10 10
10 10 10 10 **20**	10 10 11 13 16	10 10 0 0 60	10 10 10 10 20
10 10 10 **20 20**	10 11 13 17 19	10 0 0 50 30	10 10 10 20 20
10 10 **20 20 20**	11 13 17 19 20	0 0 50 30 20	10 10 20 20 20
10 **20 20 20 20**	12 16 19 20 20	0 60 30 20 20	10 20 20 20 20
Slope (horizontal)			
10 12 14 16 18	11 12 14 16 17	4 12 14 16 24	10 12 14 16 18
10 12 14 16 18	11 12 14 16 17	4 12 14 16 24	10 12 14 16 18
10 12 14 16 18	11 12 14 16 17	4 12 14 16 24	10 12 14 16 18
10 12 14 16 18	11 12 14 16 17	4 12 14 16 24	10 12 14 16 18
10 12 14 16 18	11 12 14 16 17	4 12 14 16 24	10 12 14 16 18

Original image "airplane" (512×512×8)

Smoothed by 3×3 averaging filter

Smoothed by 5×5 averaging filter

Smoothed by 7×7 averaging filter

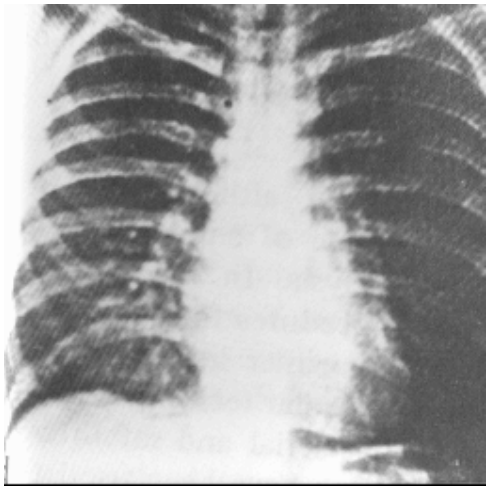


Smoothed by 15×15 averaging filter

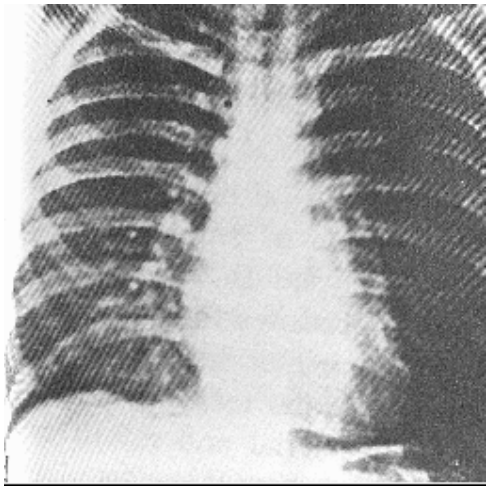
Figure 2.15: Example of smoothing.

Original image "airplane" (512×512×8)

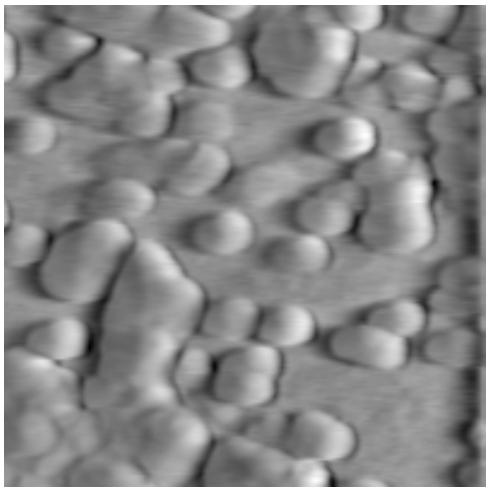
Sharpened by 3×3 window



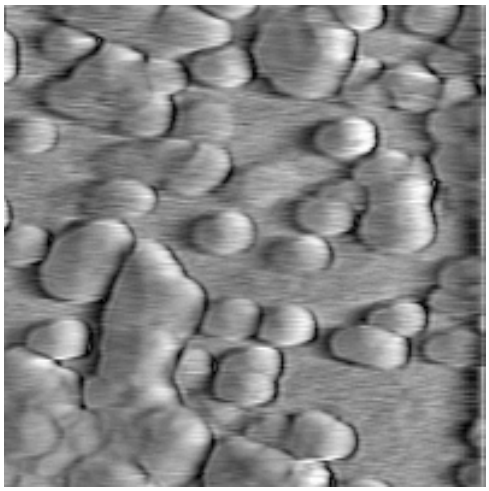
Original image "xray" (512×512×8)



Sharpened by 3×3 window



Original image "sample-1" (200×200×8)



Sharpened by 3×3 window

Figure 2.16: Example of sharpening.

Salt & pepper noise (5 %)

Salt & pepper noise (20 %)

Smoothed by 3×3 window

Smoothed by 3×3 window

Median filtered by 3×3 window

Median filtered by 3×3 window

Figure 2.17: Example of noise removing by smoothing and median filtering.

2.4 Detection of discontinuities

Discontinuities (such as isolated points, thin lines and edges in the image) can be detected by using similar masks as in the low- and high-pass filtering. The absolute value of the weighted sum given by equation (2.7) indicates how strongly that particular pixel corresponds to the property described by the mask; the greater the absolute value, the stronger the response.

Point detection

The mask for detecting *isolated point*s is given in Figure 2.18. A point can be defined to be isolated if the response by the masking exceed a predefined threshold:

$$|f(x)| > T \tag{2.8}$$

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 2.18: Mask for detecting isolated points.

Line detection

In principle, *line detection* is identical to point detection. Instead of one mask, four different masks must be used to cover the four primary directions (horizontal, vertical and two diagonals), see Figure 2.19. Note that the result of the operation only indicates for that particular pixel only, whether it is in a line (corresponding to its local neighborhood). More global approach to line detection is given in Section 3.

Horizontal

-1	-1	-1
2	2	2
-1	-1	-1

+45°

-1	-1	2
-1	2	-1
2	-1	-1

Vertical

-1	2	-1
-1	2	-1
-1	2	-1

-45°

2	-1	-1
-1	2	-1
-1	-1	2

Figure 2.19 Masks for line detection.

Edge detection

The most important operation to detect discontinuities is *edge detection*. Edge is defined here as the boundary between two regions with relatively distinct gray-level properties. Therefore it can be given by the first derivate. This is done by calculating the *gradient* of the pixel relative to its neighborhood. A good approximation of the first derivate is given by the two *Sobel operators* of Figure 2.20 with the advantage of a smoothing effect. Because derivatives enhance noise, the smoothing effect is a particularly attractive feature of the Sobel operators.

The gradients are calculated separately for horizontal and vertical directions:

$$G_x = (x_7 + 2x_8 + x_9) - (x_1 + 2x_2 + x_3) \tag{2.9}$$

$$G_y = (x_3 + 2x_6 + x_9) - (x_1 + 2x_4 + x_7) \tag{2.10}$$

It is a common practice to approximate the overall gradient by taking the sum of the absolute values of these two:

$$\nabla f \approx |G_x| + |G_y| \tag{2.11}$$

The *direction* of the gradient is:

$$\alpha = \tan^{-1}\left(\frac{G_x}{G_y}\right) \tag{2.12}$$

The second derivate can be approximated by the *Laplacian* mask given in Figure 2.21. The drawbacks of *Laplacian* are its sensitivity to noise and incapability to detect the direction of the edge. On the other hand, because it is the second derivate it produces double peak (positive and negative impulse). By Laplacian, we can detect whether the pixel lies in the dark or bright side of the edge. This property can be used in image segmentation.

Horizontal edge

-1	-2	-1
0	0	0
1	2	1

Vertical edge

-1	0	1
-2	0	2
-1	0	1

Figure 2.20: Sobel masks for edge detection.

0	-1	0
-1	4	-1
0	-1	0

Figure 2.21: Mask for Laplacian (second derivate).

Table 2.3: Pixel-level examples of point, line and edge detection.

ORIGINAL IMAGE					POINT DETECTION					LINE (x) DETECTION					EDGE (x) DETECTION				
Flat																			
10	10	10	10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	10	10	10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	10	10	10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	10	10	10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	10	10	10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Random noise																			
11	13	18	16	16	-10	-5	32	7	12	-4	4	11	13	12	7	-6	-16	-17	-16
14	11	13	12	12	-4	-37	-15	-27	-22	-13	-22	-24	-24	-22	29	19	3	-6	-1
19	18	19	12	17	33	26	**43**	-15	36	30	32	19	21	21	4	8	7	4	-3
14	15	13	15	10	-12	-8	-22	1	-34	-9	-17	-10	-20	-19	-23	-16	-2	3	-6
13	13	19	15	14	-4	-15	34	1	5	-4	3	4	10	8	-5	1	10	10	12
Two impulses																			
10	10	10	10	10	-10	-10	-10	0	0	-10	-10	-10	0	0	10	20	10	0	0
10	**20**	10	10	10	-10	**80**	-10	0	0	20	20	20	0	0	0	0	0	0	0
10	10	10	10	10	-10	-10	-20	-10	-10	-10	-10	-20	-10	-10	-10	-20	0	20	10
10	10	10	**20**	10	0	0	-10	**80**	-10	0	0	20	20	20	0	0	0	0	0
10	10	10	10	10	0	0	-10	-10	-10	0	0	-10	-10	-10	0	0	-10	-20	-10
Line (horizontal)																			
10	10	10	10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	10	10	10	10	-30	-30	-30	-30	-30	-30	-30	-30	-30	-30	**40**	**40**	**40**	**40**	**40**
20	**20**	**20**	**20**	**20**	**60**	**60**	**60**	**60**	**60**	**60**	**60**	**60**	**60**	**60**	0	0	0	0	0
10	10	10	10	10	-30	-30	-30	-30	-30	-30	-30	-30	-30	-30	**-40**	**-40**	**-40**	**-40**	**-40**
10	10	10	10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Edge (vertical)																			
10	10	10	**20**	**20**	0	0	-30	30	0	0	0	0	0	0	0	0	0	0	0
10	10	10	**20**	**20**	0	0	-30	30	0	0	0	0	0	0	0	0	0	0	0
10	10	10	**20**	**20**	0	0	-30	30	0	0	0	0	0	0	0	0	0	0	0
10	10	10	**20**	**20**	0	0	-30	30	0	0	0	0	0	0	0	0	0	0	0
10	10	10	**20**	**20**	0	0	-30	30	0	0	0	0	0	0	0	0	0	0	0
Edge (horizontal)																			
10	10	10	10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	10	10	10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	10	10	10	10	-30	-30	-30	-30	-30	-30	-30	-30	-30	-30	**40**	**40**	**40**	**40**	**40**
20	**20**	**20**	**20**	**20**	30	30	30	30	30	30	30	30	30	30	**40**	**40**	**40**	**40**	**40**
20	**20**	**20**	**20**	**20**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Edge (diagonal)																			
10	10	10	10	10	0	0	0	-10	-20	0	0	0	-10	-20	0	0	0	10	30
10	10	10	10	**20**	0	0	-10	-30	**40**	0	0	-10	0	10	0	0	10	30	**40**
10	10	10	**20**	**20**	0	-10	-30	30	10	0	-10	0	0	10	0	10	30	30	10
10	10	**20**	**20**	**20**	-10	-30	30	10	0	-10	0	0	10	0	10	30	30	10	0
10	**20**	**20**	**20**	**20**	-20	**40**	10	0	0	10	10	10	0	0	10	20	10	0	0
Slope (horizontal)																			
10	12	14	16	18	-6	0	0	0	6	0	0	0	0	0	0	0	0	0	0
10	12	14	16	18	-6	0	0	0	6	0	0	0	0	0	0	0	0	0	0
10	12	14	16	18	-6	0	0	0	6	0	0	0	0	0	0	0	0	0	0
10	12	14	16	18	-6	0	0	0	6	0	0	0	0	0	0	0	0	0	0
10	12	14	16	18	-6	0	0	0	6	0	0	0	0	0	0	0	0	0	0

Original image

G_x-gradient

G_y-gradient

$|G_x|+|G_y|$

Figure 2.22: Example of Sobel operators.

3 Segmentation

Segmentation is often the first phase in image analysis. Its primary purpose is to divide the image into certain basic elements. What these elements (segments) are, depends on the application. In air-to-ground images it may be sufficient to separate a road from the environment and the vehicles moving on the road. Automatic image segmentation is one of the most difficult tasks in image processing.

Segmentation algorithms for monochrome images (gray-scale images) generally are based on one or two basic properties of gray-level values: *discontinuity* and *similarity*. The basic image processing operations for detecting discontinuities was presented in Section 2.3. The similarity-based segmentation algorithms are based either on *thresholding*, *region growing*, or *region splitting and merging*. Formally image segmentation can be defined as a process that divides the image R into *regions* R_1, R_2, ... R_n such that:

- The segmentation is complete: $\bigcup_{i=1}^{n} R_i = R$
- Each region is uniform.
- The regions do not overlap each other: $R_i \bigcap R_j = \varnothing,\ \forall i \neq j$
- The pixels of a region have a common property: $P(R_i) = TRUE$
- Neighboring regions have different properties: $P(R_i) \neq P(R_j),\ \forall R_i, R_j$ are neighbors.

Let continue by presenting a few techniques that are used in image segmentation.

3.1 Regular block decomposition

The easiest way to segment an image is to divide it into blocks with regular shapes. The shape is often fixed throughout the image and the segmentation is based on varying the size of the blocks. These techniques are often found in image compression methods that do not require a perfect segmentation of the image, rather than aims at minimizing the contrast (variance) within the blocks.

Quadtree segmentation

Quadtree segmentation begins with only one segment which is the entire image. Let the size of the image be $n \times n$ pixels. The image is then recursively split into four equally sized sub-blocks of size $n/2 \times n/2$ until the blocks meet a predefined *uniformity criterion*. This criterion is generally the *variance* of the pixels in the block. The block is considered homogenous if its variance is below a predefined threshold ε, otherwise it is divided into sub blocks. See Figure 3.1 for an example of a quadtree, and Figure 3.2 for a quadtree segmentation applied to the image *Camera*.

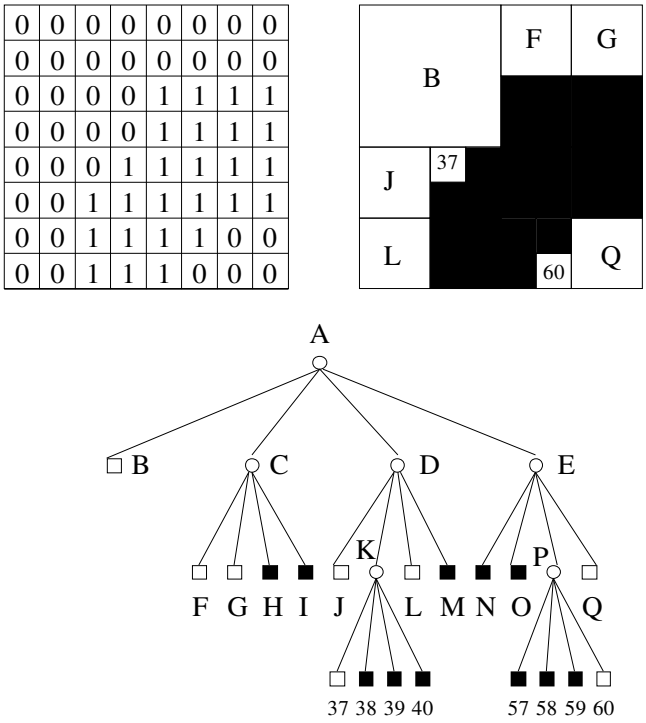


Figure 3.1: Example of quadtree data structure.

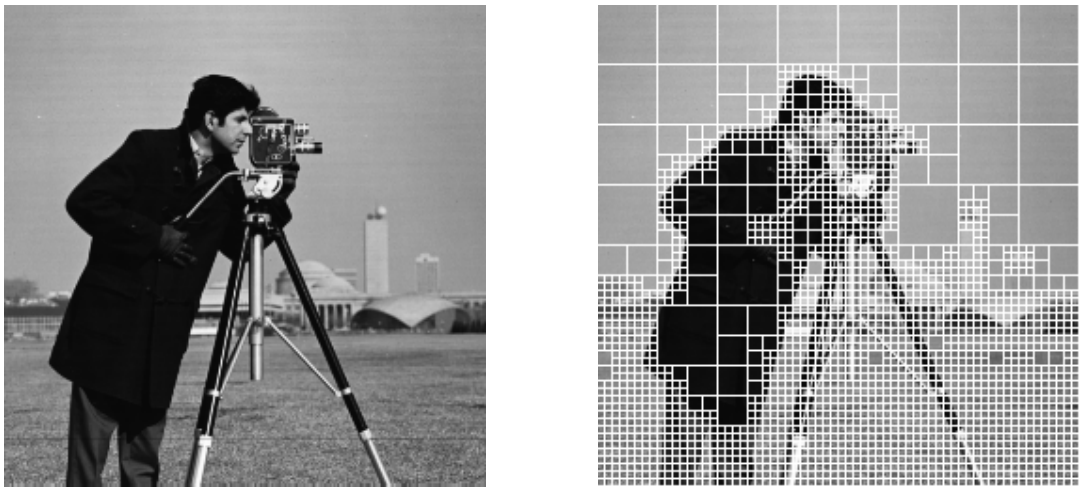


Figure 3.2: Example of quadtree segmentation based on variance criterion.

Binary tree segmentation

Binary-tree segmentation is closely related to quadtree segmentation. Here the region splitting is performed by dividing the blocks to not four but two equal sized sub blocks. The direction of the splitting varies so that the blocks at the first level (=the whole image) are divided in horizontal direction. At the second level the splitting is performed in vertical direction, then at the third level again in horizontal direction, and so on. By this way the blocks are always square, or rectangular with the dimension of 2:1.

The threshold criterion can also be variance as in quadtree segmentation. Any other suitable criterion, however, can be applied both in quadtree and binary tree segmentation, as well as in the polygon segmentation to be presented next.

Polygon segmentation

A generalization of binary tree segmentation is established if the blocks are allowed to be split within any directions, not just in horizontal and vertical directions. A straight line can be drawn from any border pixel of the region to any other border pixel, see Figure 3.3. This line splits the region into two sub regions. The resulting regions are always polygons, thus the method is called *polygon segmentation*.

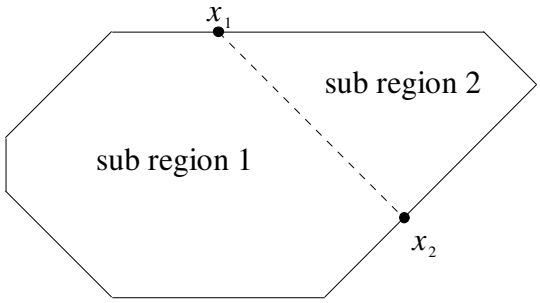


Figure 3.3: Example of splitting a polygon block.

To select the best possible line for splitting, all possible border pixel pairs must be considered. Among the several candidate lines, the one is chosen which yields smallest overall variance of the two resulting sub blocks. This is because the aim of the segmentation is to minimize the variance of each block. The operation, on the other hand, is computationally demanding and requires

$$\frac{n \cdot (n-1)}{2} = O\left(n^2\right) \tag{3.1}$$

tests in the worst case, where n is the number of border pixels in the region. A variant of the polygon segmentation limits the possible directions of the lines to four in the set of (0°, 45°, 90 °, 135°). By this way we only need to test at most 4 counter-pairs for each border pixel of the block, taking $O(n)$ time in total. The lines can still begin anywhere in the region border. The resulting regions are always polygons having at most 8 border lines. See Figure 3.4 for an example of polygon segmentation of this kind. The degree of freedom in the splitting is always a trade-off between a good segmentation and computation time.

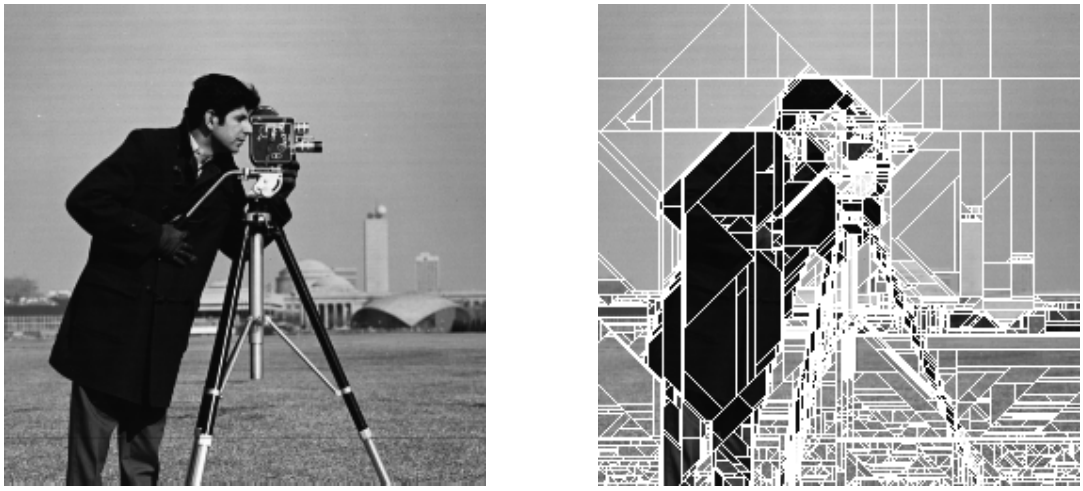


Figure 3.4: Example of polygon segmentation based on variance criterion.

3.2 Splitting and merging

The quadtree segmentation, for example, is a simple but still powerful tool for image decomposition. The resulting segmentation, on the other hand, contains neighboring regions having similar values, thus contradicting the last requirement of the image segmentation (neighboring regions should have different properties). The regular block decomposition techniques of Section 3.1. thus are not sufficient alone, at least if the formal requirements of segmentation are wanted to be met. To overcome this drawback, the block decomposition techniques can be augmented by a second phase called *merging*. The overall algorithm is thus called *splitting and merging* technique.

An example of quadtree-based split and merge is given in Figure 3.5. In the first phase the image is decomposed into four sub blocks (a), which are then further divided, except the top leftmost block (b). At the third phase the splitting phase is completed (c). At the merging phase the neighboring blocks that are uniform and have the same color are merged together. The segmentation results to only two regions (d), which is ideal in this favorable example.

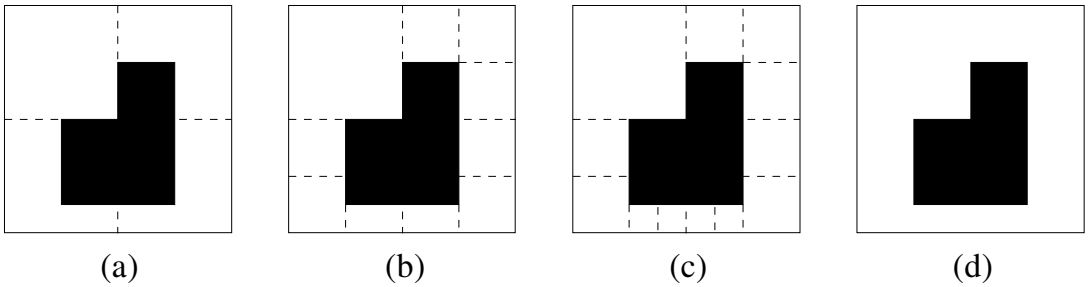


Figure 3.5: Example of quadtree based splitting and merging.

3.3 Region growing

Region growing (or *pixel aggregation*) starts from an *initial region*, which can be a group of pixels (or a single pixel) called *seed points*. The region is then extended by including new pixels into it. The included pixels must be neighboring pixels to the region and they must meet a *uniform criterion*. The growing continues until a *stopping rule* takes effect, see Figure 3.6 as an example. The region growing has several practical problems:

- how to select the seed points
- growing rule (or uniform criterion)
- stopping rule.

All of these parameters depend on the application. In the infra-red images in military applications the interesting parts are the areas which are hotter compared to their surroundings, thus a natural choice for the seed points would be the brightest pixel(s) in the image. In interactive applications the seed point(s) can be given by the user who gives (by pointing with the mouse, for example) any pixels inside the object, and the computer then finds the desired area by region growing algorithm.

The growing rule can be the differences of pixels. The value of the pixel in consideration can be compared to the value of its nearest neighbor inside the region. If the difference does not exceed a predefined threshold the pixel is attended to the region. The comparison can also be made to the average value of the region, though the local comparison criteria has found to be superior in the C*ell Particle Motion Analysis project*, see Section 6. Another criterion is to examine the variance of the region, or gradient of the pixel. Several growing criteria worth consideration are summarized in the following:

Global criteria:

- *Difference* between $x_{i,j}$ and the average value of the region.
- The *variance* σ^2 of the region if $x_{i,j}$ is attended to it.

Local criteria:

- *Difference* between $x_{i,j}$ and the nearest pixel inside the region.
- The *variance* of the local neighborhood of pixel $x_{i,j}$. If the variance is low enough (meaning that the pixel belongs to a homogenous area) it will be attended to the region.
- *Gradient* (see Section 2.3) of the pixel. If it is small enough (meaning that the pixel does not belong to an edge) it will be attended to the region.

The growing will stop automatically when there are no more pixels that meet the uniform criterion. The stopping can also take effect whenever the size of the region gets too large (assuming a priori knowledge of the object's preferable size). The shape of the object can also be used as a stopping rule, however, this requires some form of pattern recognition and prior knowledge of the object's shape.

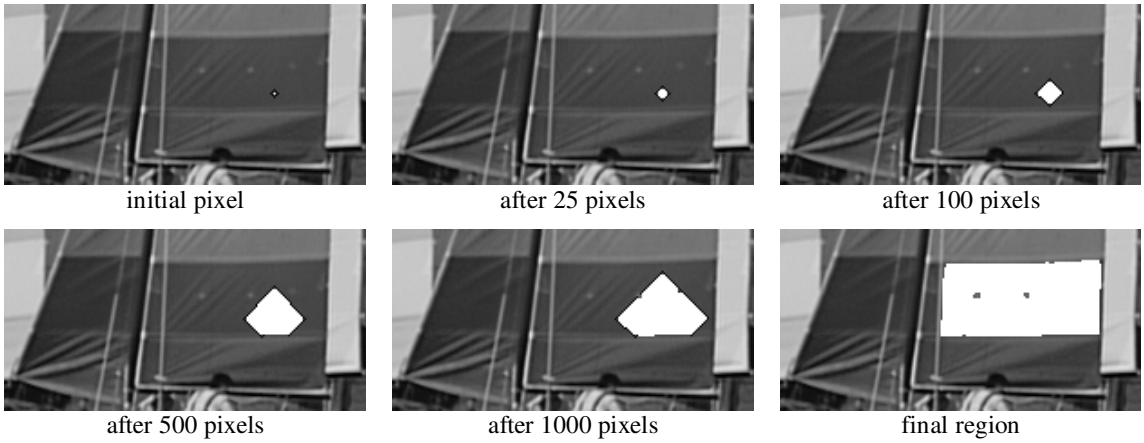


Figure 3.6: Example of region growing.

3.4 Edge following algorithm

If it is known that an object in an image has a discrete edge all around it, it is then possible to follow the edge around the object and get back to the beginning. It is assumed that a position on the edge of a region has been identified (i, j) which is the starting point of the algorithm. A simple *edge following* algorithm based on Sobel gradients is described as follows.

Algorithm for edge following:

1. Check the neighbors of the starting pixel:
   (a) Choose the three with the greatest (absolute) gradient magnitude.
   (b) Sort the pixels into descending order and store them into an array as a triple.
   (c) Choose the one with the greatest magnitude.
   (d) Set *dir* as the direction according to Figure 3.7.
   (e) Go one step forward in direction *dir*.
2. REPEAT UNTIL back at starting pixel, or list is empty:
   (a) Mark the current pixel as *visited*.
   (b) Check the three neighboring pixels in directions: *dir*, *dir*-1, *dir*+1 (mod 8).
   (c) Sort the pixels into descending order and store them into an array as a triple.
   (d) Choose the one with the greatest magnitude.
   (e) REPEAT UNTIL suitable pixel found:
      IF the candidate pixel is unvisited and its gradient is high enough, THEN
         - Pixel OK
         - Remove the pixel from the list
      ELSE IF more candidate pixels in the triple, THEN
         - Take the next pixel from the triple
      ELSE
         - Remove the empty triple
         - Move one position back in the array
   (f) Go one step forward in direction *dir*.

0	1	2
7	*	3
6	5	4

* current pixel

Figure 3.7: Numbering of the directions.

An example of the edge following algorithm is given in Table 3.1. Here the pixel (x,y) with the greatest gradient is chosen as the seed: (4,4)=76. A threshold value 20 is assumed (only gradients exceeding this threshold are considered as edge pixels). At the first stage the three pixels with the greatest gradients are chosen: (4,3), (5,3), and (3,4) in that order. When two or three pixels have equal values, *row-major order* is assumed in the ordering. The pixel (4,3) is first chosen, however, the following pixels (3,2), (4,2), and (5,2) fail the test and pixel (4,3) is therefore dropped. The next candidate (5,3) produces candidates (6,3), (5,2), and (6,2), from which the first is chosen. The detailed execution of the algorithm is given in Table 3.2.

Table 3.1: Example of edge following algorithm: original image superimposing the object (left); gradients of the pixels superimposing the result of the edge following (right).

Original image										
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1	8	9	10	11	12	13	14	15	16	17
2	9	10	11	12	13	14	15	16	17	18
3	10	11	12	13	14	15	16	17	18	19
4	11	12	13	**24**	**25**	**26**	**27**	18	19	20
5	12	13	14	**25**	**26**	**27**	**28**	19	20	21
6	13	14	15	**26**	**27**	**28**	19	20	21	22
7	14	15	16	**27**	**28**	19	20	21	22	23
8	15	16	17	18	19	20	21	22	23	24
9	16	17	18	19	20	21	22	23	24	25
10	17	18	19	20	21	22	23	24	25	26

Sobel gradients										
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1	-	-	-	-	-	-	-	-	-	-
2	-	16	16	16	16	16	16	16	16	-
3	-	16	36	56	**56**	**56**	40	20	16	-
4	-	16	56	**76**	56	56	**60**	40	16	-
5	-	16	**56**	56	16	4	**44**	24	16	-
6	-	16	**56**	56	4	**44**	44	4	16	-
7	-	16	40	**60**	**44**	44	4	16	16	-
8	-	16	20	40	24	4	16	16	16	-
9	-	16	16	16	16	16	16	16	16	-
10	-	-	-	-	-	-	-	-	-	-

Table 3.2: Detailed execution of the edge following algorithm.

Step	**1st candidate**	**2nd candidate**	**3rd candidate**
1	~~**(4,3)=56**~~	**(5,3)=56**	(3,4)=56
2	~~**(3,2)=16**~~	~~**(4,2)=16**~~	~~**(5,2)=16**~~
3	**(6,3)=56**	(5,2)=16	(6,2)=16
4	**(7,4)=60**	(7,3)=40	(7,2)=16
5	**(7,5)=44**	(8,4)=40	(8,5)=24
6	**(6,6)=44**	(7,6)=44	(8,6)=4
7	**(5,7)=44**	(6,7)=44	(5,6)=4
8	**(4,7)=60**	(4,8)=40	(5,8)=24
9	**(3,6)=56**	(3,7)=40	(3,8)=20
10	**(3,5)=56**	(2,5)=16	(2,6)=16
11	**(4,4)=76**	(3,4)=56	(2,4)=16

3.5 Edge linking

Edge pixels can be linked simply by comparing their gradient information. *Edge linking* uses the gradient information given by Sobel operation. Two pixels are defined to be *similar* if their gradient magnitudes and their gradient directions are close enough to each other:

$$\left|\nabla f(i_1, j_1) - \nabla f(i_2, j_2)\right| \leq T \tag{3.2}$$

$$\left|\alpha(i_1, j_1) - \alpha(i_2, j_2)\right| \leq A \tag{3.3}$$

Here T and A are predefined threshold values. Pixels found to be similar are linked together by drawing a straight line between them. The algorithm operates in a local 5×5, or 7×7 pixel window. The center pixel is compared against the other pixels in the window. If the compared pixels are similar they are linked together. The window is then moved by one position to right and the center pixel of the next window is processed. Note that the algorithm considers the edge pixels only, i.e. the pixels whose gradient magnitude does not exceed a predefined threshold are ignored.

See Figure 3.8 for an example of edge linking. Here the resulting image is post-processed by deleting isolated short elements and linking edge segments separated by small breaks.

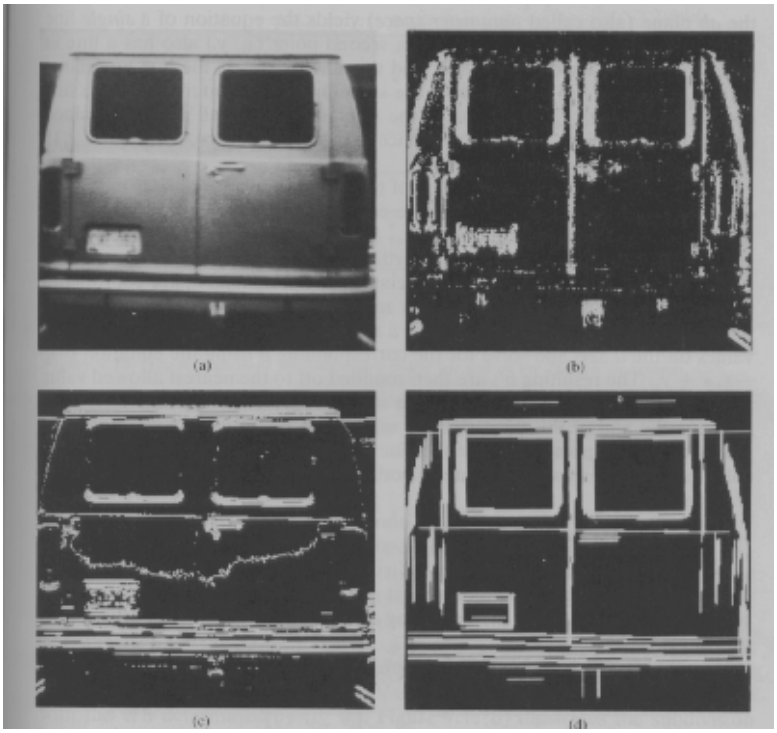


Figure 3.8: Example of edge linking: (a) original image; (b) G_y-component; (c) G_x-component; (d) result of edge linking.

3.6 Thresholding

Suppose that the image consists of one or more objects and background, each having distinct gray-level values. The purpose of *thresholding* is to separate these areas from each other by using the information given by the histogram of the image. If the object and the background have clearly uniform gray-level values, the object can easily be detected by splitting the histogram using a suitable threshold value, see Figure 3.9. The threshold is considered as a function of:

$$T = g\left(i, j, x_{i,j}, p(i,j)\right) \tag{3.4}$$

where i and j are the coordinates of the pixel, $x_{i,j}$ is the intensity value of the pixel, and $p(i, j)$ is a function describing the neighborhood of the pixel (e.g. the average value of the neighboring pixels). The thresholding can be performed by:

$$f(x,y) = \begin{cases} 1 & \text{if } x_{i,j} > T \\ 0 & \text{if } x_{i,j} \le T \end{cases} \tag{3.5}$$

An example of thresholding is shown in Figure 3.10, and the corresponding histogram in Figure 3.11.

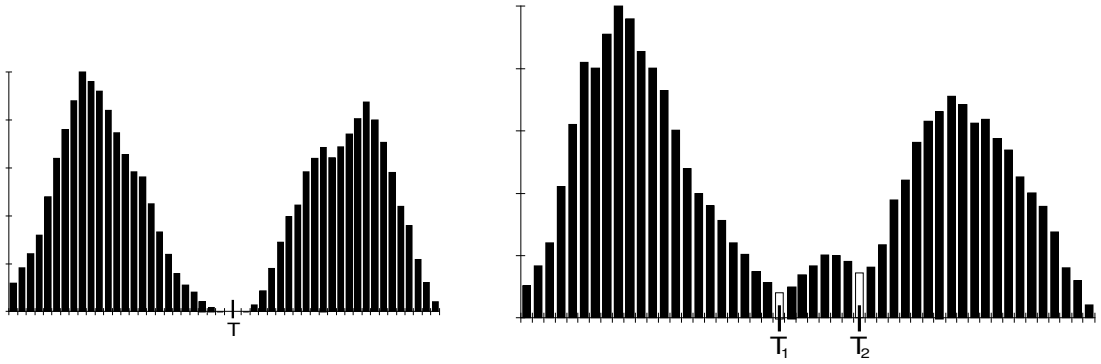


Figure 3.9: Histogram partitioned with one threshold (left),and with two thresholds right).

The thresholding can be classified according to the parameters it depends on:

Thresholding:	**Effective parameters:**
Global	$x_{i,j}$
Local	$x_{i,j}$, $p(i, j)$
Dynamic	$x_{i,j}$, $p(i, j)$, i , j

In practice, the processed images are rarely taken in perfect illumination conditions, thus the object is hardly recognizable only by analyzing the global histogram of the image. An image can be defined as consisting of two components: *illuminance* $I(i, j)$ and *reflectance* $R(i, j)$. The first component corresponds to the amount of light striking the object at a given position. The latter component corresponds to the surface of the object; how much it reflects light. The perceived image is the product of these two components: $x_{i,j} = I(i, j) \times R(i, j)$, see Figure 3.12. If the illumination component is known, it can be easily eliminated, but this is rarely the case.

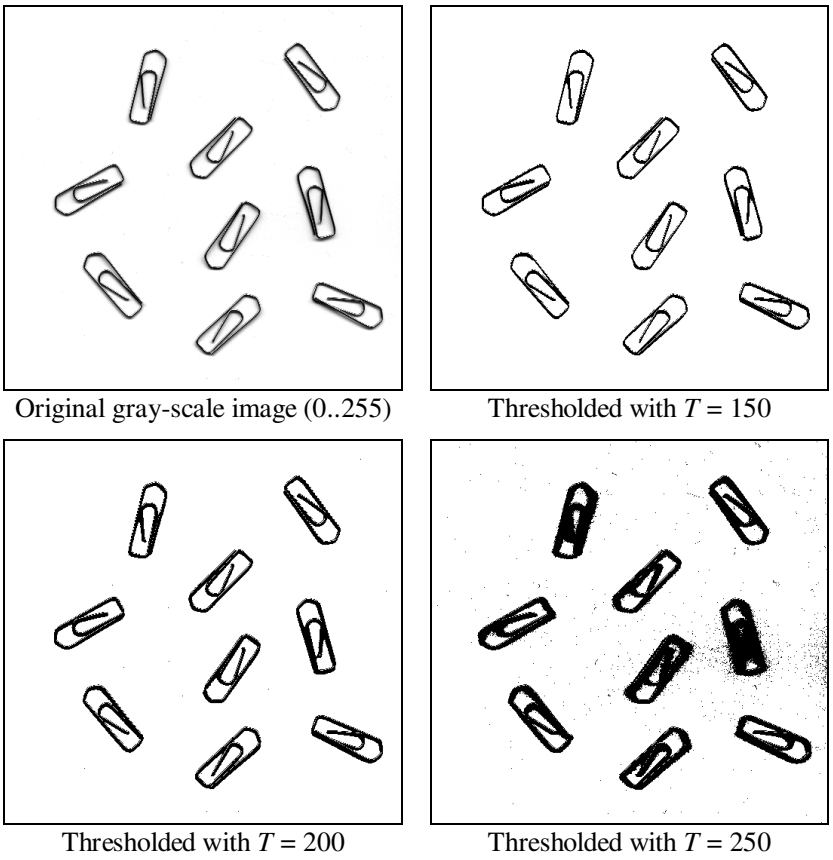


Figure 3.10: Example of thresholding with fixed threshold value.

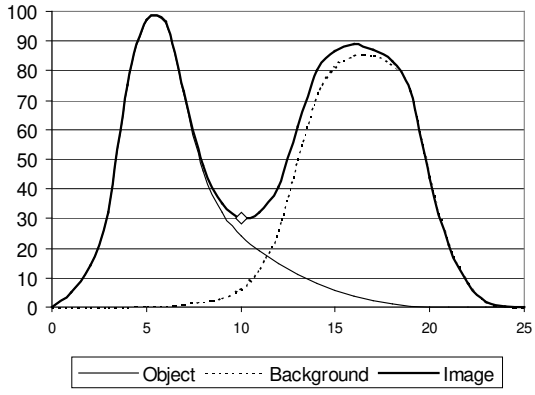


Figure 3.11: Example of histogram with two overlapping peaks.

For a given histogram, a threshold value can be calculated by an iterative method as described below. The method aims at minimizing the pixel variances inside the two segments. The algorithm starts by taking the average value of the pixels as a tentative threshold T. The image is the segmented into two parts and the average values ($\overline{x_1}$ and $\overline{x_2}$) of the two segments are calculated (n_1 and n_2 are the number of pixels in the two segments). New threshold value T is taken as the average of $\overline{x_1}$ and $\overline{x_2}$, and the segmentation is redone. The process is then

repeated until the threshold value T (and therefore segmentation) does not change. The iterative method does not necessarily give the optimal threshold value but it is practical, and in most cases sufficiently good.

Iterative method for calculating threshold:

Initialize: $T = \overline{x}$
REPEAT
- Segment the image using T.
- Calculate segment averages: $\overline{x_1} = \frac{1}{n_1} \sum_{x_i \le T} x_i$ and $\overline{x_2} = \frac{1}{n_2} \sum_{x_i > T} x_i$
- Recalculate threshold: $T = \frac{\overline{x_1} + \overline{x_2}}{2}$

UNTIL T does not change.

Dynamic thresholding works. The first phase of the algorithm is to divide the image into 7×7 blocks. For each block a local histogram is calculated, and the blocks having a bimodal histogram (as in Figure 3.12b) are further examined. At the second phase threshold values are calculated for each of these blocks using the method presented in [Gonzalez & Woods, 1992, pp. 447-450]. Thresholds for the other (non-bimodal) blocks are then determined by using *bilinear interpolation* (extract formulation of interpolation is passed here). The threshold of each block is assigned to the center pixel of the block, and a pixel-level interpolation is used to derive thresholds for the rest of the pixels on the basis of the thresholds of the center pixels. At this stage each pixel of the image has its own threshold value. Finally, the image pixels are thresholded with (3.5) according to their threshold values.

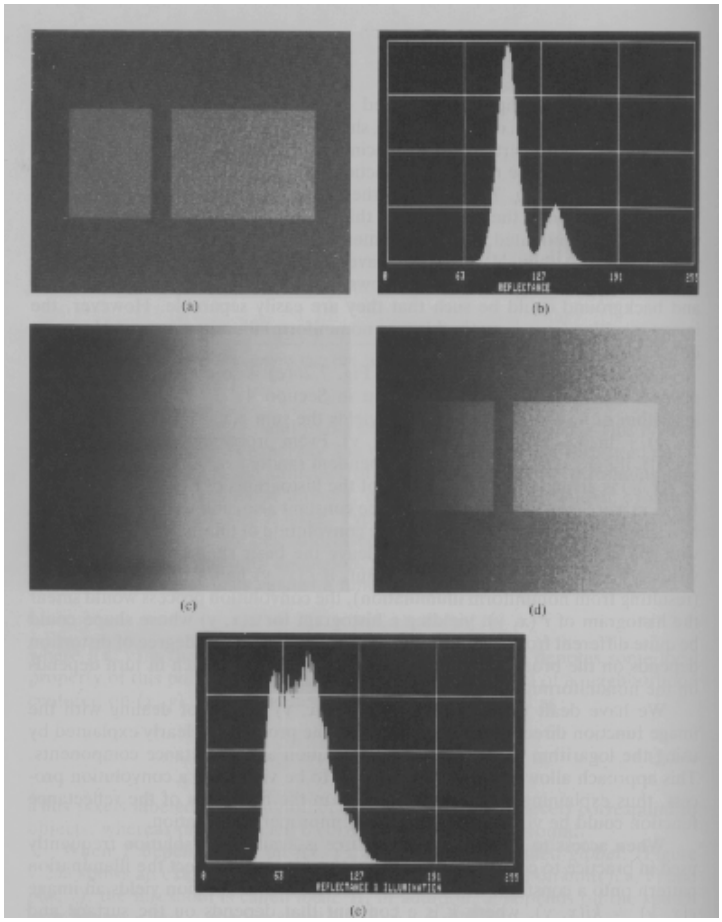


Figure 3.11: (a) Computer generated reflectance function; (b) histogram of reflectance function; (c) computer generated illumination function; (d) image produced by the product of the illumination and reflectance function; (e) histogram of the image.
(From: Gonzalez and Woods [1992])

3.7 Hough transform

Suppose that we have an image consisting of several samples of a straight line, see Figure 3.12. Hough [1962] proposed a method (commonly referred as *Hough transform*) for finding the line (or lines) among these samples (or edge pixels). Consider a point (x_i, y_i). There is an infinite number of lines passing through this point, however, they all can be described by

$$y_i = a \cdot x_i + b \tag{3.6}$$

This means that all the lines passing (x_i, y_i) are defined by two parameters, a and b. The equation (3.6) can be rewritten as

$$b = -x_i \cdot a + y_i \tag{3.7}$$

Now, if we consider b as a function of a, where x_i and y_i are two constants, the parameters can be represented by a single line in the ab-plane. The Hough transform is a process where each

pixel sample in the original *xy*-plane is transformed to a line in the *ab*-plane. Consider two pixels (x_1, y_1) and (x_2, y_2). Suppose that we draw a line *L* across these points. The transformed lines of (x_1, y_1) and (x_2, y_2) in the *ab*-plane intersect each other in the point (a', b'), which is the description of the line *L*, see Figure 3.13. In general, the more lines cross point (a, b) the stronger indication that is that there is a line $y = a \cdot x + b$ in the image.

To implement the Hough transform a two-dimensional matrix is needed, see Figure 3.15. In each cell of the matrix there is a counter of how many lines cross that point. Each line in the *ab*-plane increases the counter of the cells that are along its way. A problem in this implementation is that both the slope (a) and intercept (b) approach infinity as the line approaches the vertical. One way around this difficulty is to use the normal representation of a line:

$$x \cdot \cos\alpha + y \cdot \sin\alpha = d \tag{3.8}$$

Here *d* represents the shortest distance between origin and the line. α represents the angle of the shortest path in respect to the *x*-axis. Their corresponding ranges are $\alpha \in [0, \sqrt{2}D]$, and $\alpha \in$ [-90°, 90°], where *D* is the distance between corners in the image.

Although the focus has been on straight lines, the Hough transform is applicable to any other shape. For example, the points lying on the circle

$$(x - c_1)^2 + (y - c_2)^2 = {c_3}^2 \tag{3.9}$$

can be detected by using the approach just discussed. The basic difference is the presence of three parameters (c_1, c_2, c_3), which results in a 3-dimensional parameter space.

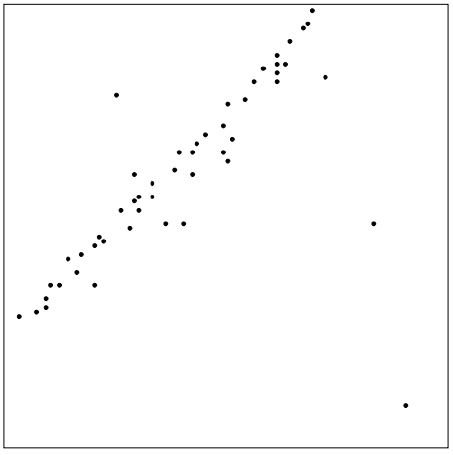


Figure 3.12: Image containing samples of a straight line.

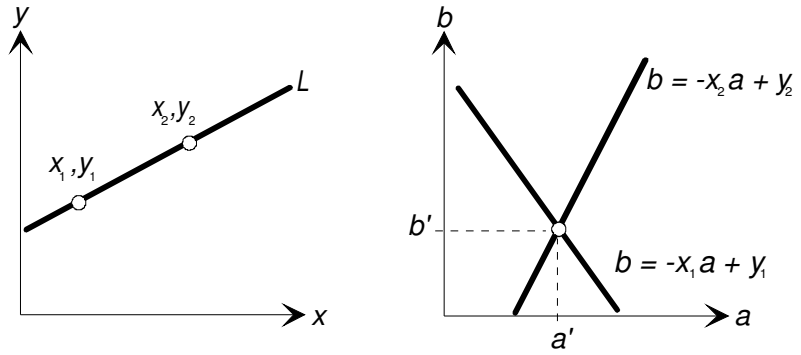


Figure 3.13: Hough transform: *xy*-plane (left); *ab*-plane (right).

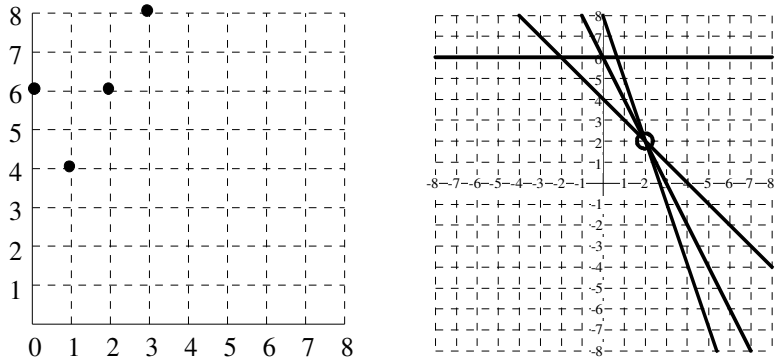


Figure 3.14: Small example of Hough transform.

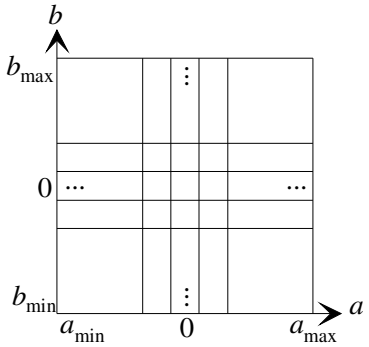


Figure 3.15: Quantization of the parameter plane for use in Hough transform.

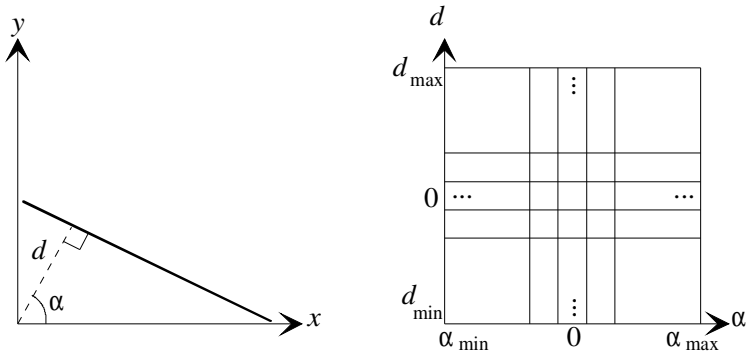


Figure 3.16: Normal representation of a line (left); quantization of the αd-plane into cells.