Chapter #

Feature Extraction

Image Enhancement, Edge and Other Feature Detection

Zhigang Zhu
Department of Computer Science, The City College of New York, New York, NY 10031

1. INTRODUCTION

In this chapter, we will discuss the following topics:

- Image enhancement
- Mathematical techniques (convolution, Gaussian filtering)
- Edge and line detection and extraction
- Region segmentation
- Contour extraction
- Corner detection

In image enhancement, we will include (1) brightness mapping; (2) contrast stretching/enhancement; (3) histogram modification; (4) noise reduction; etc. After that, we will introduce a very useful mathematical technique called convolution, and then discuss Gaussian filtering for noise reduction. Using the same convolution concept, we can also obtain gradient information from images for edge and line detection and extraction. Finally, we will discuss three basic feature extraction problems: region segmentation, contour extraction and corner detection.

2. IMAGE ENHANCEMENT

The overall goal of image enhancement is to improve the 'visual quality' of an image, either for human viewing, for subsequent processing, or for both. Usually, there is no general theory of 'visual quality'. As a rule of

thumb, a general assumption is: if it looks better, it is better. This is often not a good assumption, but we probably don't have a better way to define visual quality in a general sense.

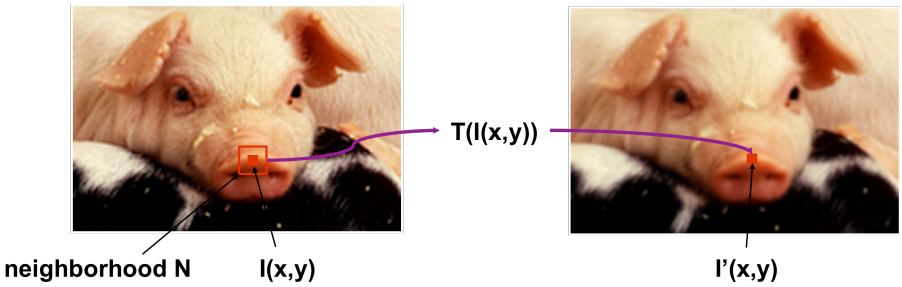


Figure 1. Spatial domain methods

There are two general approaches to image enhancement: ***spatial domain methods*** operating directly on image pixels, and ***frequency domain methods*** operating on the Fourier transform of the image. In this chapter, we will mainly focus on the spatial domain methods, even though we may mention the frequency domain methods occasionally. For information on the latter, please refer to the standard image processing literature. The spatial domain methods can be roughly represented as a transformation function T (Figure 1)

$$\mathbf{I'(x,y) = T(I(x,y))} \qquad \mathbf{(1)}$$

which turns an original image function I(x,y) into a processed image function I'(x,y). The transformation (or operation) T can be one of the following three types:

(1) ***point operations***, which transforms each individual pixel I(x,y) into a pixel in the same location;
(2) **area operations**, which transform a local area around (x,y) into a pixel at (x,y); and
(3) ***global operations***, which use all the pixels of the entire image to determine the value of the pixel at (x,y).

For area operations, a neighborhood is usually defined as a rectangular area around pixel at (x,y), and typically with an odd size: 3×3, 5×5, etc., so that the area is centered on pixel I(x,y). Many image processing algorithms rely on this basic notion.

In the following sub-sections, we will discuss a number of useful point operations, histogram modification techniques, and noise reduction methods.

Since histograms will be used in the discussions of point operations and noise reduction, we will introduce the concept of histograms and techniques for histogram modification first. Then we will discuss point operations and noise reduction. Finally, as the summary of this section, we will offer some observations on image enhancement.

2.1 Histogram modification

2.1.1 Histograms

Histograms are a simple yet very important tool for image analysis. An image shows the spatial distribution of gray values. The histogram of an image, on the other hand, discards the spatial information and shows the relative frequency of occurrence of these gray values.

Image

0	3	3	2	5	5
1	1	0	3	4	5
2	2	2	4	4	4
3	3	4	4	5	5
3	4	5	5	6	6
7	6	6	6	6	5

Gray Value	Count	Rel. Freq.
0	2	.05
1	2	.05
2	4	.11
3	6	.17
4	7	.20
5	8	.22
6	6	.17
7	1	.03
Sum=	36	1.00

Figure 2. Histogram example

A histogram plots the absolute pixel counts as a function of these gray values, as shown in Figure 2 and Figure 3. For an image with dimensions M by N, and G different levels, the histogram can be represented as H(i), i =0, 1, …, G-1, and we can easily know that

$$\sum_{i=0}^{G-1} H(i) = MN \tag{2}$$

By interpreting the *relative frequency histogram* as a probability distribution, we then have:

$$P(i) = P(I(x,y) = i) = H(i)/(MxN) \tag{3}$$

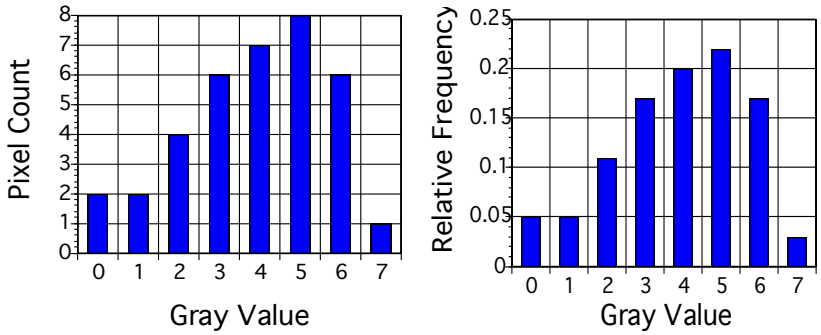


Figure 3. Histogram plotting: in pixel counts and relative frequencies

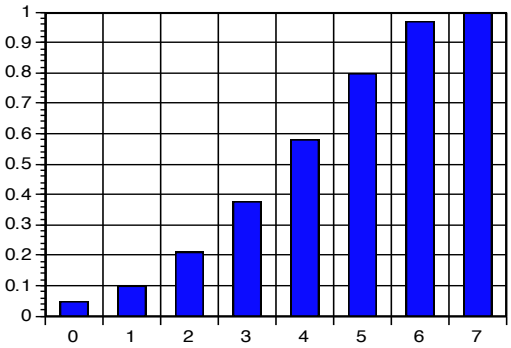


Figure 4. Cumulative histogram

Sometimes it is useful to generate the cumulative distribution function of the image (Figure 4), as

$$Q(i) = \sum_{k=0}^{i} P(k) \tag{4}$$

so that we can easily know what is the percentage of pixel numbers below a certain pixel value *i*. Often we may prefer to generate a cumulative histogram corresponding to the cumulative distribution function, as

$$CH(i) = \sum_{k=0}^{i} H(k) \tag{5}$$

if we want to use the original histogram of pixel counts.

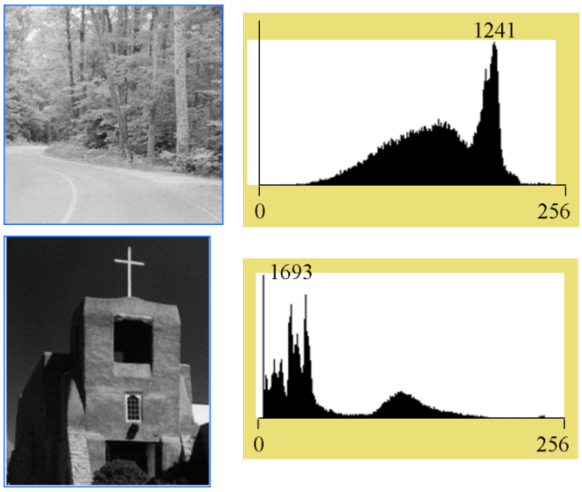


Figure 5. Histogram Examples

Figure 5 shows two images and their corresponding histograms. From their histograms we can make the following two observations. (1) Overall, the top image is brighter than the bottom image, since its histogram has higher counts of bright pixels. (2) In each of the two histograms, there is an obvious valley between two peaks, therefore roughly separating each image into two parts. Image 1 can be segmented into the darker trees and the lighter road, and image 2 into the dark sky and the brighter building front facade with the cross. We will see a few more applications that can make use of histograms below.

2.1.2 Histogram Equalization

Image histograms usually consist of peaks, valleys, and low plains. A ***peak*** at a gray level means that there are many pixels concentrated around that gray level. A ***plain*** indicates a small number of pixels distributed over a wider range of gray levels. A ***valley***, as we have seen, usually defines a range of gray levels that separate two groups of pixels with higher and lower gray levels than those valley pixels.

The goal of histogram equalization is to modify the gray levels of an image so that the histogram of the modified image is flat. This means that we have to expand pixels in peaks over a wider range of gray-levels, meanwhile "squeeze" low plains pixels into a narrower range of gray levels so that all gray values will be utilized equally. Since histogram equalization is a standard method in an image processing textbook, here we will only summarize its basic algorithm without explaining the details.

We aim to find a mapping from one set of gray-levels, *I*, to a new set, *O*. Ideally, the number of pixels, N_p, occupying each gray level should be:

$$N_p = \frac{M*N}{G} \tag{6}$$

where G is the number of gray levels, and M and N are the image dimensions. To approximate this, we will need to apply the transform

$$i = MAX\left[\, 0, \text{round}\left\{\frac{CH(j)}{N_p}\right\} - 1 \,\right] \tag{7}$$

where CH is the cumulative histogram, j is the gray value in the source image and i is the gray value in the equalized image.

Figure 6 shows a histogram equalization example. On the left, the original image and the equalized image are depicted. The latter exhibits a much fuller dynamic range than the original image. On the right, the original histogram, the transformation curve O=T(I) generated from the cumulative histogram, and the equalized histogram are plotted. It can be seen that the equalized histogram is mostly flat.

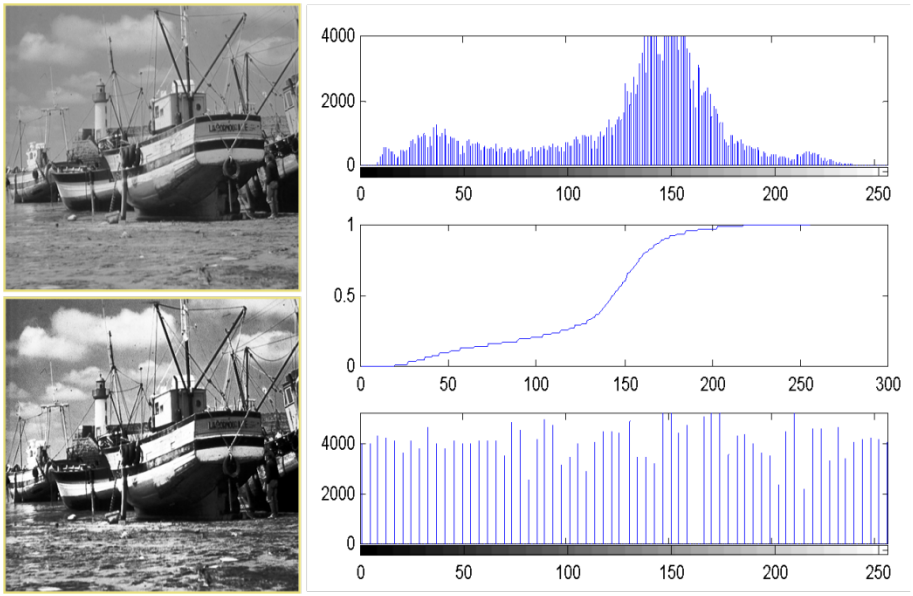


Figure 6. Histogram equalization example. On the left: original image (top) and the equalized image (bottom). On the right: the histogram of the original image (top), its cumulative histogram, i.e., the transformation function (middle), and the histogram of the equalized image (bottom).

2.2 Point Operations

2.2.1 Point Transforms: General Idea

In a point operation, an input pixel value, I, is mapped to an output pixel value, O, via the transfer function T:

$$O = T(I) \quad (8)$$

Usually the transformation function T is a monotonic, 1-1 mapping, as shown in Figure 7.

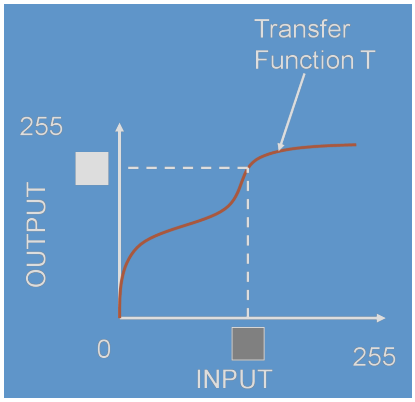


Figure 7. Point Operations

In the following, we will take a look at the most useful point operations (transforms): brightness mapping, thresholding, linear stretching, and non-linear scaling.

2.2.2 Point Transforms: Brightness mapping

Brightness mapping is in the form of O = I + g, where g is the overall gain offset of the brightness. If g equals zero, the transform is an identity transformation therefore the image is unchanged (Figure 8 left). When the value of g is positive, the image becomes brighter (Figure 8 middle), whereas the image becomes darker when the value of g is negative (Figure 8 right). Note that in reality, the output O should be clipped into the range of 0 to 1 if we use the normalized intensity range (Figure 1), or 0 to 255 if a byte is used to represent each pixel in an intensity image. In Figure 8, the histograms of the original image, the brightened image and the darkened image are also shown, illustrating the shifts of distributions of pixel values in the gray-scale range of the image. The first histogram uses [0-255] index values while the other two use normalized index values [0-1].

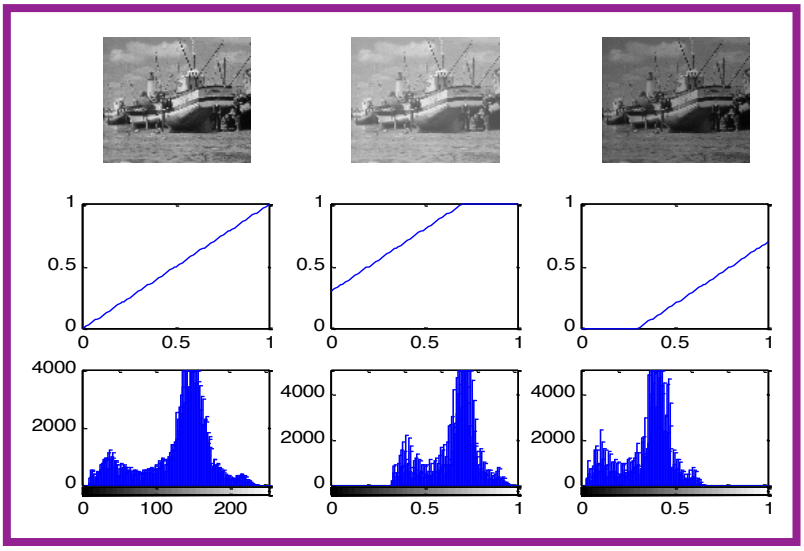


Figure 8. Brightness mapping

2.2.3 Point Transforms: Thresholding

Thresholding is a very simple but effective method for image segmentation. In Figure 9, a color image (a) is first transformed into a gray-scale image (b). By defining a threshold, the pixels whose gray levels are above the threshold are then set to 1s and those whose gray levels are below or at the threshold are set to 0s. Thus, the image is segmented into a white background and black object pixels (c).

Figure 9. Thresholding

2.2.4 Point Transforms: Linear Stretch

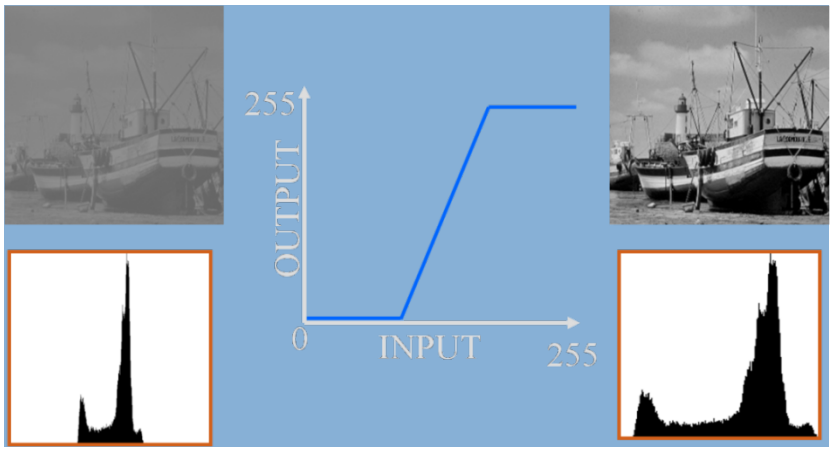


Figure 10. Linear stretching

Consider the case in which the original image only occupies a small subset (I_{min}, I_{max}) of the full range of gray values (0, K-1). We would like the new image to span the full range of gray values. The transformation will be an equation of the straight line:

$$I'(x,y) = \frac{K}{I_{max} - I_{min}} I(x,y) - \frac{K}{I_{max} - I_{min}} I_{min}$$

$$I' = mI + b$$

(9)

It is the equation of the straight line going through the point (I_{min} , 0) and (I_{max} , K), as shown in Figure 11. In practice, a look-up table O=T(I) will be generated using the above line equation, before the transformation is applied to a M by N image. This will save a lot of time in calculating the transformation, as the time complexity is reduced to O(K) from O(MN).

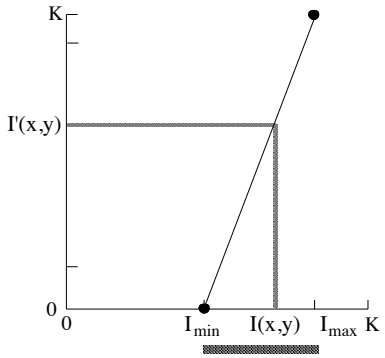


Figure 11. Linear stretching

Other practical issues occur when scaling discrete images. So far we have assumed a continuous grayscale. However, in practice, we only have discrete grayscale levels. Figure 12 shows what happens in the case of a discrete grayscale with 7 levels. There are two problems. First, the output O = T(I) may not be an integer. Usually we will round it to its nearest integer level. Second, some intensity levels could have no pixels. So the total number of intensity levels could be fewer than that of the original image. This seems to be a serious problem if the total number of levels is reduced. But if the original image has 256 levels, this is not a big issue.

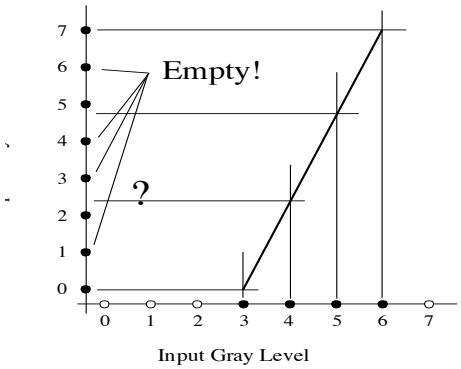


Figure 12. Scaling discrete images

2.2.5 Non-Linear Scaling: Power Law

One of the non-linear scaling methods is to use the power law function

$$O = I^{\gamma} \qquad (10)$$

The value of the power γ controls the effects of the scaling (Figure 13): When γ=1, the transformation is an identity transform; when γ < 1, the scaling is to enhance contrast in dark regions; when γ > 1 it is to enhance contrast in bright regions. Figure 14 illustrates the effect of a square root transfer: the dark region is stretched across significantly more gray values and causes the texture pattern of the background to show. Figure 15, on the other hand, displays the result of using a transfer function with γ=3: the original whitened image becomes much more pleasant with a larger dynamic range stretched to the lower gray level side.

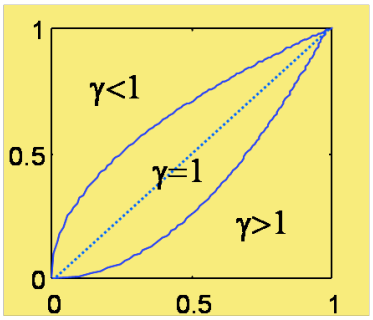


Figure 13. Non-linear scaling: power law

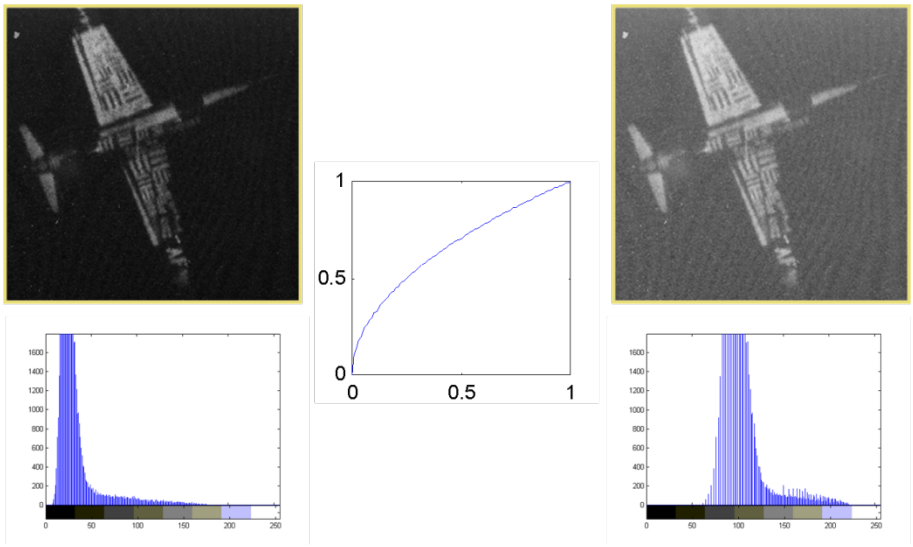


Figure 14. Square root transfer: γ=0.5

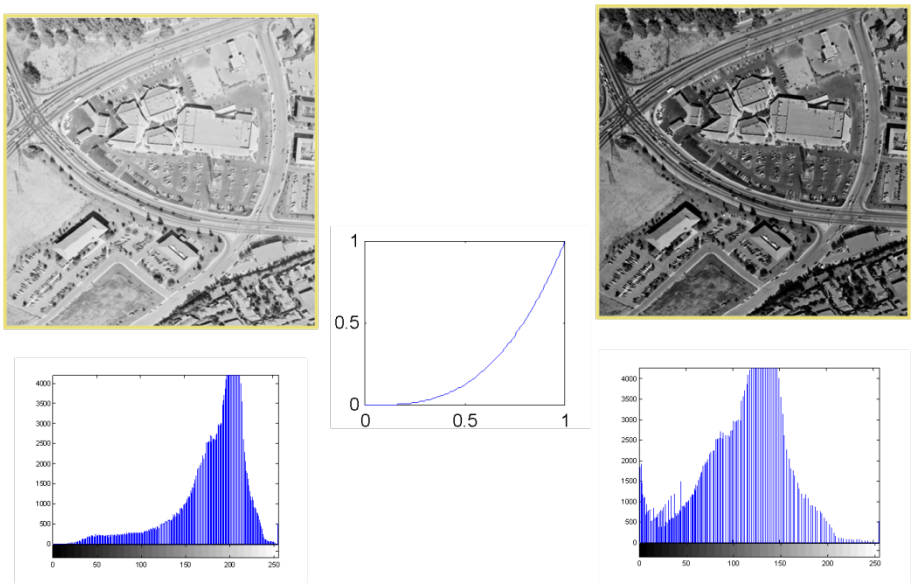


Figure 15. Power law transform: $\gamma = 3$

2.3 Noise Reduction

Now we will move on to discuss noise reduction. We will cover the following topics:

- What is noise?
- How is noise reduction performed?
  - Noise reduction from first principles
  - Neighborhood operators including:
    - linear filters (low pass, high pass)
    - non-linear filters (median)

2.3.1 Noise, Linear Filtering and Convolution

We will start with an example to show some properties of noise before presenting a noise model. Then we will introduce an important mathematical operation called convolution for implementing our linear filters. We will then show an example of how linear filtering can reduce noise within an image.

Figure 16 shows an original image, an image of random noise, and the final grainy image when adding the noise to the original image. As an example of how noise affects grayscale values, the intensity values of the pixels below the dark, vertical line are displayed in the three traces below the

images. The values of the original image, the noise, and the 'grainy' image are plotted on top, in the middle, and on the bottom, respectively.

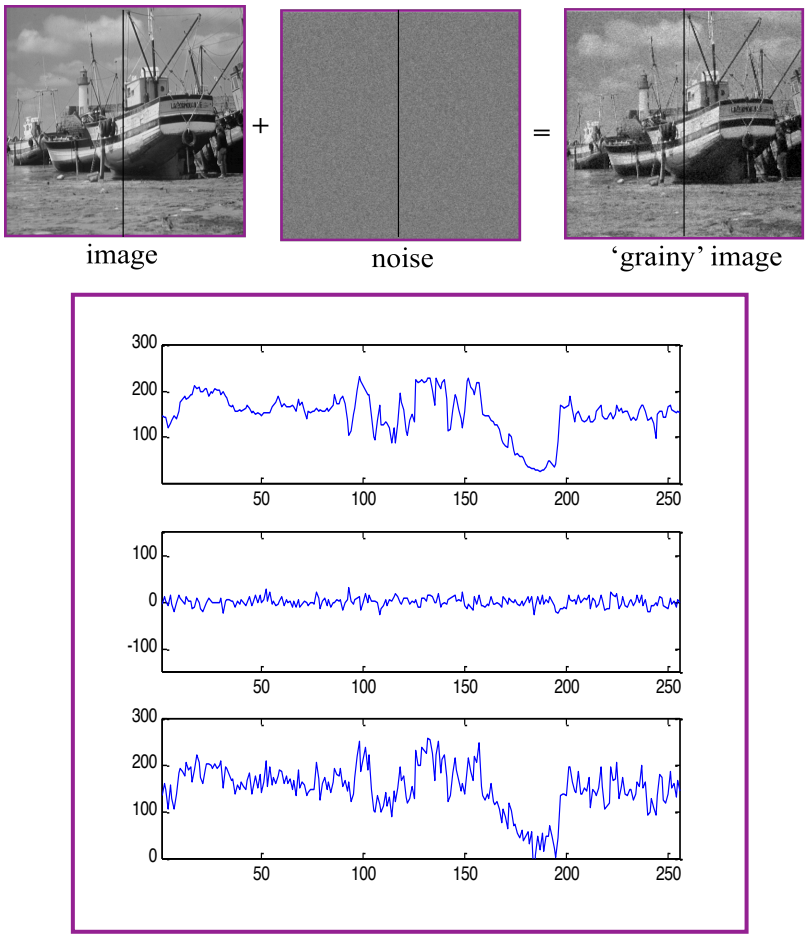


Figure 16. Additive noise

From the plots, we can see that: (1) Noise is additive; (2) Noise fluctuations are rapid; and (3) Noise represents high frequency signals. The sources of noise could be one or several of the following: The common source of noise is the charge-coupled device (CCD) chip of a camera, where electronic signal fluctuations in the CCD detector could lead to noise in images. The noise could also be introduced by thermal energy of the sensors; the situation could be worse for far infrared image sensors since those sensors detect temperature (thermal) rather than reflectance. The other noise sources could be electronics and transmission.

A typical additive **noise model** can be defined as a normal (or called Gaussian) distribution with mean noise $\mu = 0$ and a standard deviation σ

$$\eta(x) = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{1}{2}\frac{(x-\mu)^2}{\sigma}\right] \tag{11}$$

Figure 17 shows a plot of the noise distribution (histogram), shifting the mean to 128 for a 256-level image in order to display the noise image. The zero mean indicates that the additive noise is fluctuated around zero.

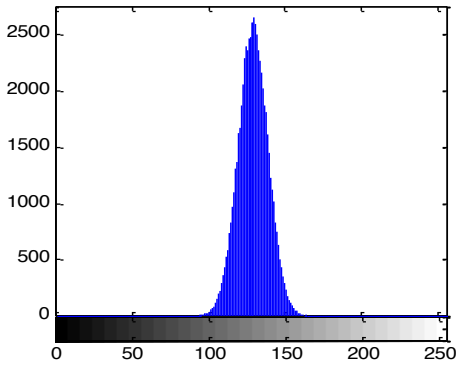


Figure 17. Plot noise histogram

How do we reduce noise? We can apply the linear filtering approach by utilizing convolution. **Convolution** is an extremely important concept in computer vision, image processing, signal processing, etc. There is a lot of related mathematics behind it even though we will not cover it here. Instead, we will provide a quick summary.

The general idea is to reduce a filtering operation to the repeated application of a mask M (or filter kernel) to the image I. The kernel can be thought of as an $l \times l$ image, and l is usually odd so that the kernel has a central pixel. In practice the following are the steps to perform a convolution-based linear filtering operation (Figure 18):

- (flip kernel both horizontally and vertically) – explanation in text below
- Align kernel center pixel with an image pixel
- Point-wise multiply each kernel pixel value with corresponding image pixel value and add results
- Resulting sum is normalized by kernel weight
- Result is the value of the pixel centered on the kernel

- Process is repeated across image

Note that special treatment is needed when kernel is near edge of input image.

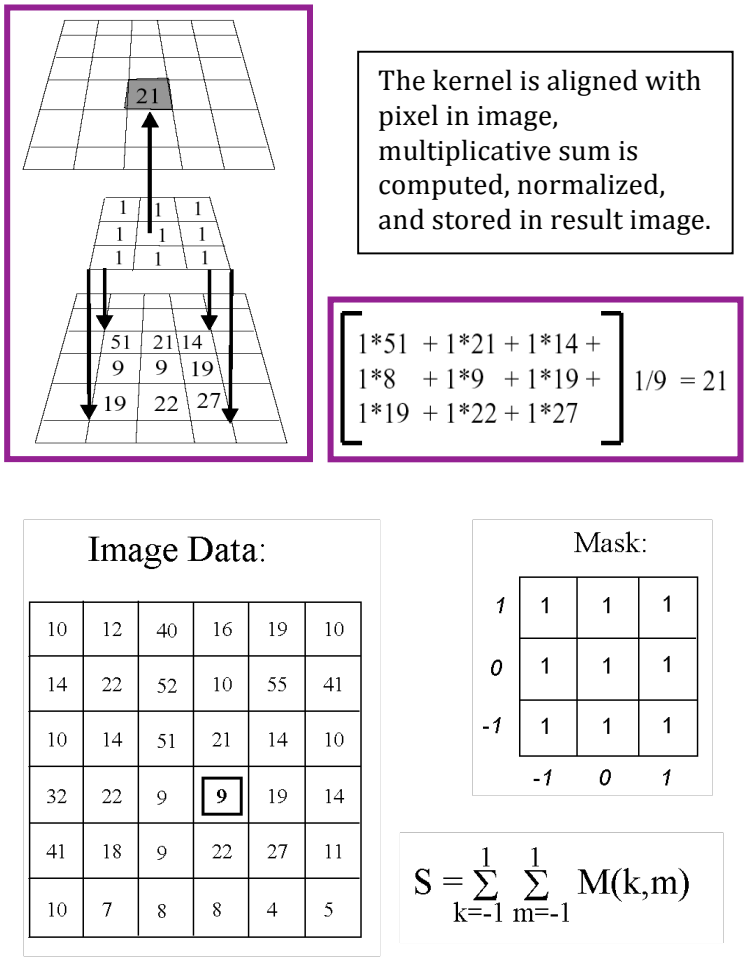


Figure 18. Illustration of convolution

Strictly speaking, the first step (flip the kernel) is required so that all the properties of convolution (below) can be maintained, as we can see in the formal definition of the convolution operation:

$$\mathbf{I_f} = \mathbf{I} \otimes \mathbf{M} : I_f(i,j) = \frac{1}{S} \sum_{k=-l}^{l} \sum_{m=-l}^{l} I(i-k, j-m) M(k,m) \qquad (12)$$

where ⊗ represents the convolution operation, S is the sum of the values of the mask, and the negative signs in (-k, -m) indicate the flip of the kernel. Using the definition in Eq. (12), there are several useful **properties of convolution:**

Commutative: The convolution of an image with a kernel is the same as the convolution of the kernel with the image:

$$f_1(t) \otimes f_2(t) = f_2(t) \otimes f_1(t)$$

Distributive: The sum of the convoluted results of an image with two separate kernels are the same as the convolution of the image with the sum of the two kernels:

$$f_1(t) \otimes [f_2(t) + f_3(t)] = f_1(t) \otimes f_2(t) + f_1(t) \otimes f_3(t)$$

Using this rule can lead to significant reduction in computation time for convolutions since two convolutions can be implemented by one convolution.

***Associative*:** The result of the convolutions of an image with two kernels consecutively is the same as convolution of the image with the convolution result of the two kernels.

$$f_1(t) \otimes [f_2(t) \otimes f_3(t)] = [f_1(t) \otimes f_2(t)] \otimes f_3(t)$$

Using this rule, we can turn one problem into the other, depending on which one is more efficient in computation.

Shift: The same amount of shift will be maintained in both the original image (or the kernel) and the result of convolution.

if $f_1(t) \otimes f_2(t) = c(t)$, then $f_1(t) \otimes f_2(t\text{-}T) = f_1(t\text{-}T) \otimes f_2(t) = c(t\text{-}T)$

Convolution with impulse:

$f(t) \otimes d(t) = f(t)$

Convolution with shifted impulse:

$f(t) \otimes d(t\text{-}T) = f(t\text{-}T)$

In practice, the kernel is usually symmetric in both the x and y directions, so for convenience, the convolution can be represented (without flipping the kernel) as

$$I_f(i,j) = \frac{1}{S} \sum_{k=-l}^{l} \sum_{m=-l}^{l} I(i+k, j+m) M(k,m) \qquad (13)$$

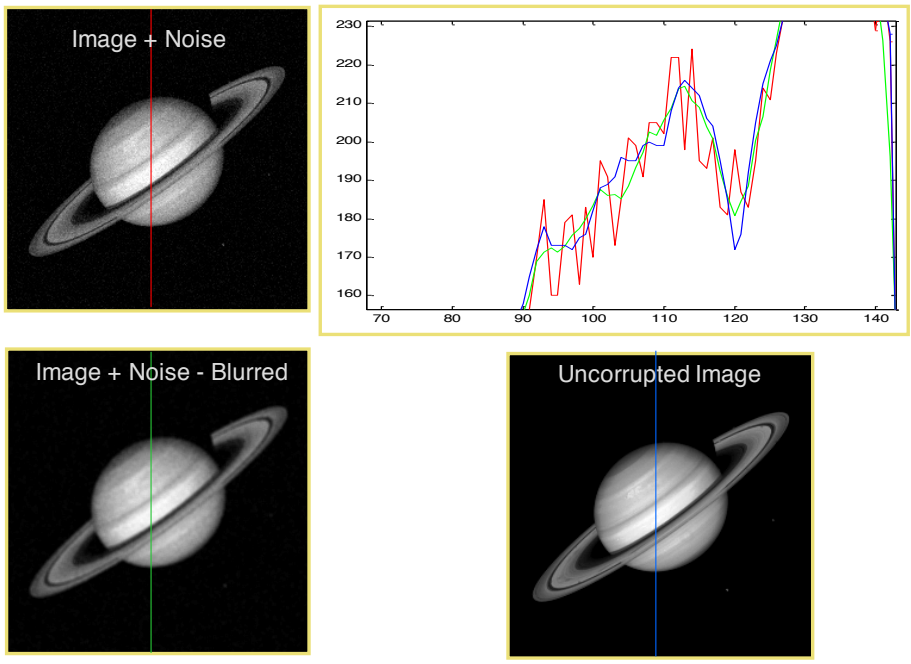


Figure 19. Noise reduction: Gaussian filtering

Figure 19 shows an example of noise reduction. The linear filtering technique using an all-one mask relies on high frequency noise fluctuations being 'blocked' by the filter. Hence, it is called a low-pass filter. The technique is very simply to implement; however, the fine detail in the image may also be smoothed. Therefore, we need a balance between preserving fine detail and reducing noise. In the following, we will introduce two techniques: median filtering as a non-linear filtering approach, and edge-preserving smoothing.

2.3.2 Median Filtering

One of the commonly used nonlinear filters is the median filter, which computes the median of points in a defined neighborhood. It has a lower tendency of blurring fine details compared to averaging, therefore it works very well for 'shot' or 'salt and pepper' noise. How does a median filter work? Here are the three steps:

- Get data from a window centered at the to-be-processed pixel
- Sort the data (from min to max)
- Put the median value in the filtered image

Note that the weighting and summation procedure in convolution is replaced by a *sorting* and *selecting* process. Figure 20 compares the performance of the low-pass and the median filters. Roughly speaking, a 1D median filter of a window size m can remove noise that is less than half of the window size m/2. Similar observation can be made for a 2D median filter.

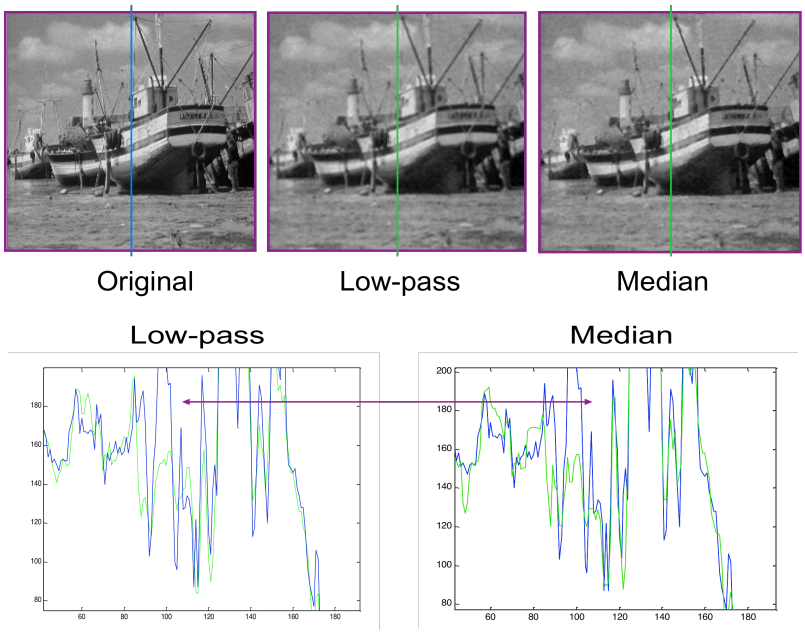


Figure 20. Low-pass: fine detail smoothed by averaging; Median: fine detail passed by filter

2.3.3 Edge-preserving smoothing

The goal of the edge-preserving smoothing is to smooth (average, blur) an image without disturbing sharpness or position of the real edges in an image. In the following paragraph, we will briefly list several such operators that have been proposed in the past.

2.3.3.1 Nagoa-Maysuyama Filter

The Nagoa-Maysuyama Filter was proposed by [REF]. Here are the basic steps (refer to Figure 21):

- Calculate the variances within nine sub-windows of a 5x5 moving window;

- Output value is the mean of the sub-window with the lowest variance.

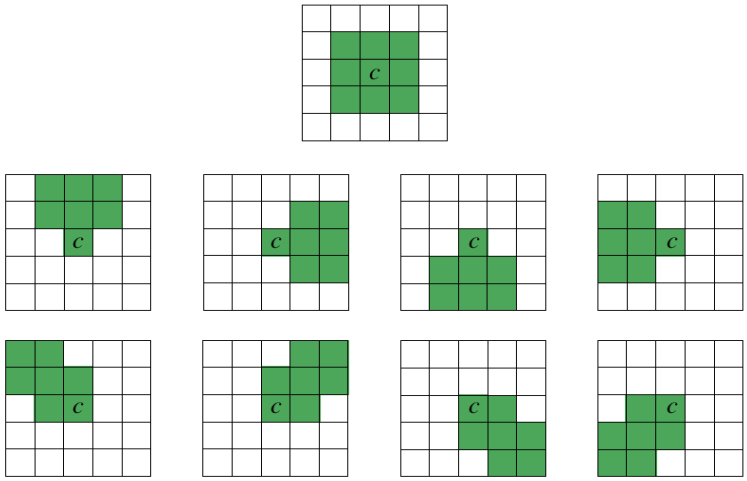


Figure 21. 2.3.3.1 Nagoa-Maysuyama Filter: nine subwindows used

2.3.3.2 Kuwahara Filter

The principle of Kuwahara Filter is the following (Figure 22):

- Divide filter mask into four regions marked by (a, b, c, d).
- In each region compute the mean brightness and the variance
- The output value of the center pixel (marked as abcd) in the window is the mean value of that region that has the smallest variance.

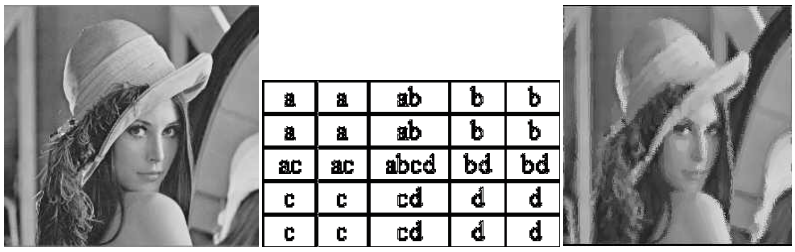


Figure 22. Kuwahara Filter

2.4 Observations on Enhancement

- Set of general techniques

- Varied goals:
  - enhance perceptual aspects of an image for human observation
  - preprocessing to aid a vision system achieve its goal
- Techniques tend to be simple, ad hoc, and qualitative
- Not universally applicable
  - results depend on image characteristics
  - determined by interactive experimentation
- Can be embedded in larger vision system
  - selection must be done carefully
  - some techniques introduce artifacts into the image data
- Developing specialized techniques for specific applications can be tedious and frustrating.

3. EDGE AND LINE DETECTION AND EXTRACTION

What's an edge? Here is a quick list of how we use the word "edge" in our daily life:

- "He was sitting on the Edge of his seat."
- "She paints with a hard Edge."
- "I almost ran off the Edge of the road."
- "She was standing by the Edge of the woods."
- "Film negatives should only be handled by their Edges."
- "We are on the Edge of tomorrow."
- "He likes to live life on the Edge."
- "She is feeling rather Edgy."

You might find that the definition of Edge is not always clear. In computer vision, an edge is usually related to a discontinuity within a local set of pixels. In Figure 23, a rendered image is shown, which exhibits four types of discontinuity that can be viewed as edges.

A: Depth discontinuity: abrupt depth change in the world;
B: Surface normal discontinuity: change in surface orientation;
C: Illumination discontinuity: shadows, lighting changes;
D: Reflectance discontinuity: surface properties, markings

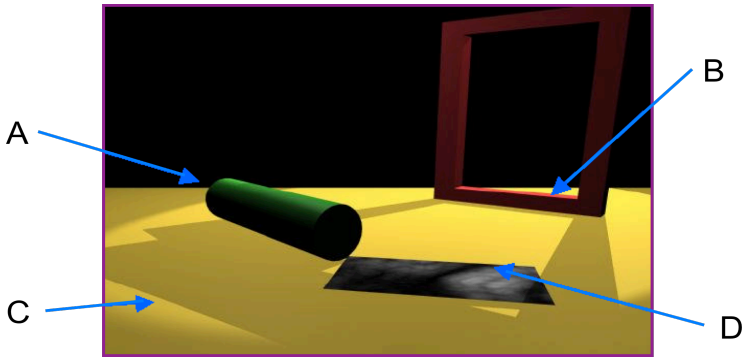


Figure 23. Illustrations of various types of discontinuity in an image. A: Depth discontinuity: abrupt depth change in the world; B: Surface normal discontinuity: change in surface orientation; C: Illumination discontinuity: shadows, lighting changes; D: Reflectance discontinuity: surface properties, markings

To make the situation worse, you might realize that human vision systems can also perceive edges that are not really there. They are called illusory edges (Figure 24).

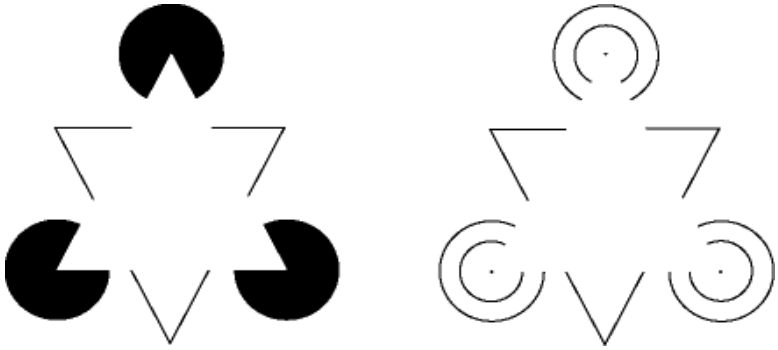


Figure 24. Kanizsa Triangles: illusory edges

Illusory edges will not be detectable by the algorithms that we will discuss. Since there are no changes in image irradiance, no image processing algorithms can directly address these situations unless we apply higher-level techniques. Computer vision algorithms can only deal with these sorts of problems by drawing on information external to the images (e.g. using perceptual grouping techniques)

In this section, we will devise computational algorithms (edge detectors) for the extraction of significant edges from the image. Unfortunately, again, what is meant by significant is unclear. It is partly defined by the context in which the edge detectors are being applied (Figure 25).

Figure 25. Panda in the wild. Left: color image; right: gradient image

3.1 Definition of Edgels and How to Detect Them

We define a local edge (or *edgel: edge element)* to be a rapid change in the function representation of an image over a small area. This implies that edgels should be detectable over a local neighborhood. Edgels are NOT yet contours, boundaries, or lines of objects. However, edgels may lend support to the existence of those structures. In other words, these structures are typically constructed from edgels.

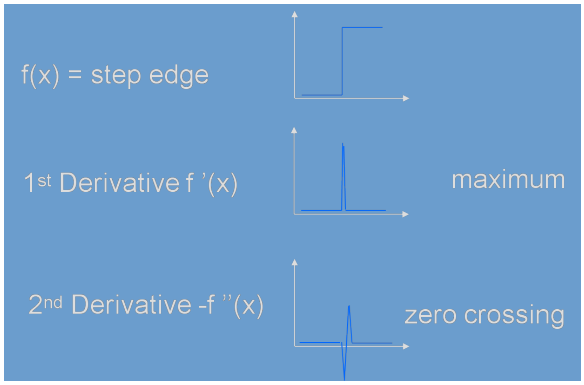


Figure 26. Using differentiation to detect edge (need a new figure)

An edgel, as a rapid change in image, results in a high local gradient. Therefore we can use differentiation to characterize this rapid change. Figure 26 illustrates the idea using a 1D function f(x). When f(x) is a step function, its first derivative f'(x) has a pulse (local maximum) at the edge of the step,

and its second derivative f”(x) exhibits a zero-crossing at the edge point. In the figure we show –f”(x) so that the curve exhibits a zero crossing from a negative pulse to a positive pulse.

With these notations, here is a brief outline of what we will discuss on edge detection:

- First order edge detectors
  - Mathematics
  - 1x2, Roberts, Sobel, Prewitt
- Canny edge detector
- Second order edge detector
  - Laplacian, LOG / DOG
- Hough Transform – detection by voting
  - Lines, circles and other shapes

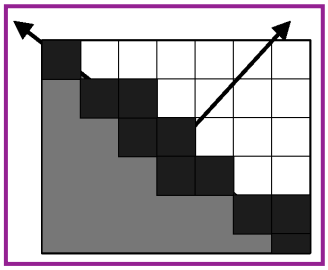


Figure 27. Quantitative edge descriptors (arrows to be labeled)

3.1.1 Quantitative Edge Descriptors

Edgels have three important properties: orientation, magnitude and position (Figure 27, Figure 28).

Edge Orientation: This can be either defined as the edge normal - unit vector either in the direction of maximum intensity change (maximum intensity gradient), or in the edge direction - unit vector perpendicular to the edge normal.

Edge Position / Center: image position at which edge is located (usually saved as binary image)

Edge Strength / Magnitude: it is related to local contrast or gradient - how rapid is the intensity variation across the edge along the edge normal.

Figure 28 shows an example. A close-up window on the right is shown for an original image on the left, which the position and the orientation of an edgel (between the sky and the sand) shown. Its strength (magnitude) along the direction of the edge normal (shown as a yellow line) is plotted on the

bottom-left chart. The further close-up on the bottom-right shows the transition area (pixel mixes) between two regions, the sky and the sand hill, which indicates localizing the edge position might not be as simple as you might have thought.

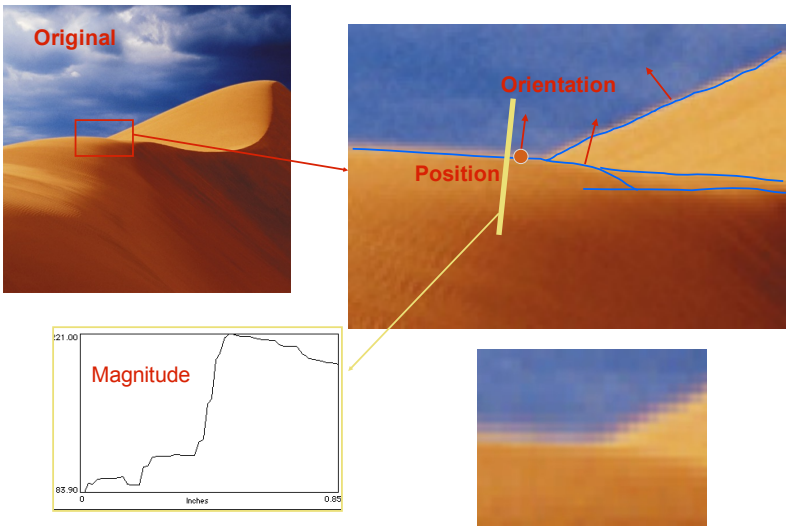


Figure 28. Edge and its properties

3.1.2 Edge Detection in a Noisy Image

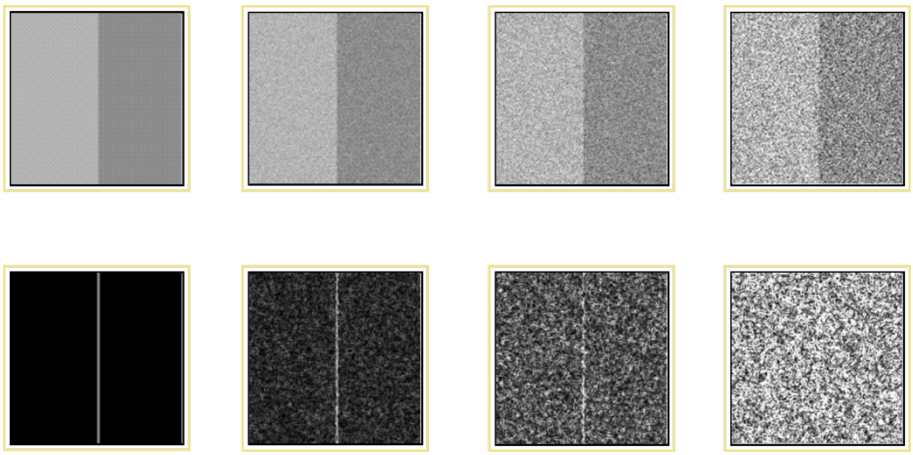


Figure 29. Edge degradation in noise

In addition to the pixelization problem (pixel mixes), things could get worse in real-world images since edges degrade with noise. Figure 29 shows how the edgels could be buried into noise when the original image of a step is corrupted by different levels of noise. Using a small local gradient operator, the edgels will not be able to be detected when the noise level is too high, even though humans can still perceive the edges. Therefore, the first step for edge detection in noisy images is noise smoothing, using the techniques we discussed in the previous section. Here are the typical steps for an edge detection algorithm.

A Typical Edge Detection Algorithm:

- Noise Smoothing
  - Suppress as much noise as possible while retaining 'true' edges
  - In the absence of other information, assume 'white' noise with a Gaussian distribution, thus using a low-pass filter
- Edge Enhancement
  - Design a filter that responds to edges; filter output is high at edge pixels and low elsewhere
- Edge Localization
  - Determine which edge pixels should be discarded as noise and which should be retained. Here are two steps to do so:
    - thin wide edges to 1-pixel width (non-maximum suppression)
    - establish minimum value to declare a local maximum from edge filter to be an edge (thresholding)

3.2 First order edge detectors

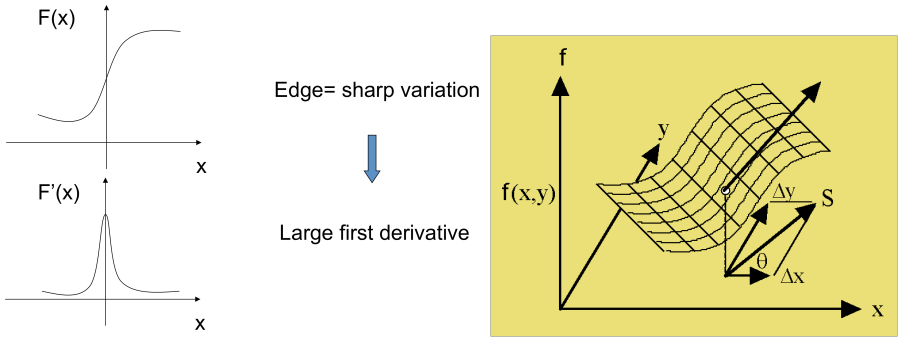


Figure 30. First order detectors

As we know, a sharp variation (edge) corresponds to a local maximum in its first order derivative (Figure 30 left). We will extend this idea to the 2D case. Assume f is a continuous function in (x,y). Then we have

$$\Delta_x = \frac{\partial f}{\partial x}, \quad \Delta_y = \frac{\partial f}{\partial y} \tag{14}$$

which are the rates of change of the function f in the x and y directions, respectively. The vector (Δ_x, Δ_y) is called the gradient of f, including the horizontal gradient and the vertical gradient components. This vector has a magnitude:

$$s = \sqrt{\Delta_x^2 + \Delta_y^2} \tag{15}$$

and an orientation:

$$\theta = \tan^{-1}\left(\frac{\Delta_y}{\Delta_x}\right) \tag{16}$$

The orientation θ is the direction of the maximum change in f, and magnitude S is the strength of that change.

This seems to be the end of the story. But unfortunately a digital image I(i,j) that a computer can process is not a continuous function. Therefore we have to look for discrete approximations to the gradient.

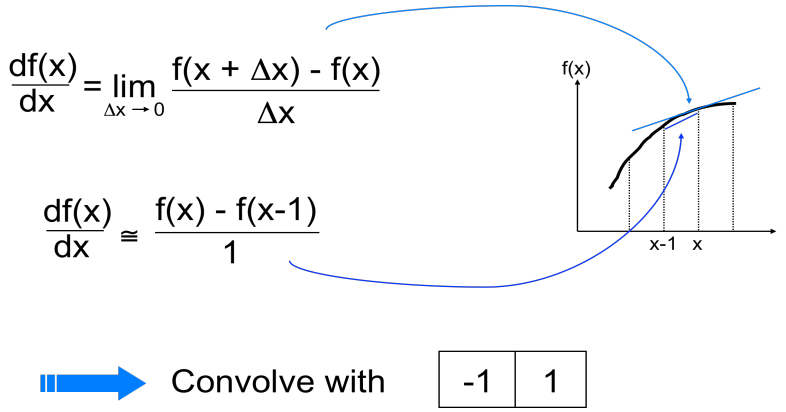
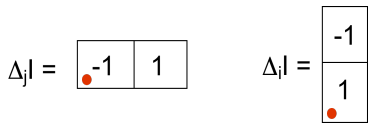


Figure 31. Discrete approximations to the gradient

Figure 31 illustrates the idea. The discrete approximation of the first order derivative turned into a difference between the current pixel and the previous pixel, which in turn can be represented as a convolution operation of the original image with a kernel [-1, 1]. In a 2D image, this leads to the 1x2 edge detector directly using pixel difference, as (Figure 32)



where the red dot represents the current pixel location of each kernel.

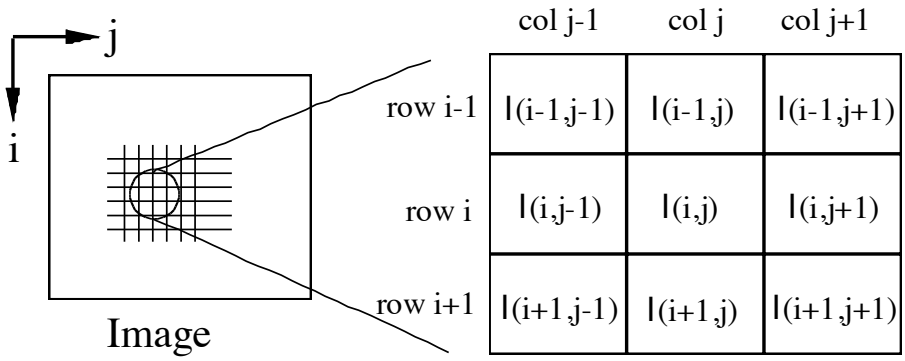


Figure 32. 2D edge detection using pixel differences

Figure 33. Edge detection using the 1x2 detector: an example

Figure 33 shows an example: the original image, the 1x2 vertical gradient image showing the vertical gradient components of all the pixels of the original image, the 1x2 horizontal gradient image (showing the horizontal gradient components), and the combined (e.g. using the gradient strength of each pixel) are shown.

While the 1x2 operator is simple, it is sensitive to image noise since the kernel relies on the differences between two consecutive pixels, in the vertical and horizontal directions, respectively. One solution to this problem is to smooth the image first, for example, using an averaging operation. Spatial averages can be computed using one of the following masks:

$$\mathbf{1/9\ x}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \mathbf{1/8\ x}\begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Combining an average mask with the 1x2 edge operator can lead to better edge detection. Figure 34 shows an example of one of the noisy images shown in Figure 29. It can be easily seen that after two iterations of 3x3 spatial averaging, the edgels along the step can be extracted. From this point on, we will call the results using 1x2 operators "gradient" images (even though sometimes they are called "edge" images in literature), whereas the thresholded edge images will be called "edge" images.

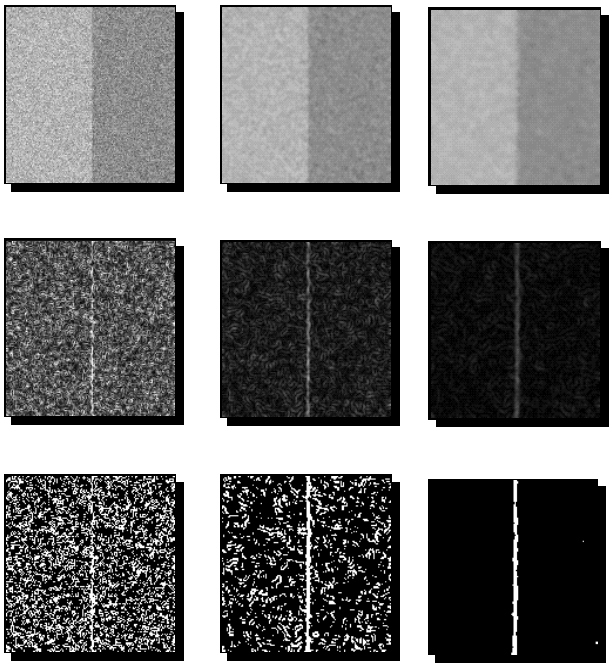


Figure 34. Edge detection in a noisy image. From top to bottom: grayscale, edge (gradient) and thresholded edge images. From left to right: original, result after 1 iteration of spatial averaging, and that after 2 iterations of spatial averaging

If you may recall, both the spatial operation and the difference operation are convolutions, therefore the consecutive applications of these two operators can also be done by generating a combined kernel with the convolution of the two kernels, and then applying the combined kernel to the original image just once for all. For example, applying the following mask is equivalent to taking the difference of the averages on two sides (top and bottom) of the central pixel:

-1	-1	-1
0	●	0
1	1	1

Note that since the average for each side of is done for three pixels, the final result for each pixel of the gradient image should be divided by a weight 3 – thus generating a *normalized* gradient image. The normalizing weight is calculated as the total weights on each side (which should be the same). This leads to the design of the Prewitt operator:

P_1=

-1	-1	-1
0	0	0
1	1	1

P_2 =

-1	0	1
-1	0	1
-1	0	1

where the weights for all pixels in averaging are ones. Figure 35 shows the results of applying the Prewitt operator on an image we have shown before.

Figure 35. Original image and the edge magnitude image

Figure 36 shows a simple global thresholding technique to obtain an edge image from the gradient magnitude image (a portion), using the histogram of the gradient image as a guideline for how many percentages of points will be

kept as edgels. In the figure, two thresholds are applied, 128 and 64. Clearly more edge points are kept when we select a lower threshold.

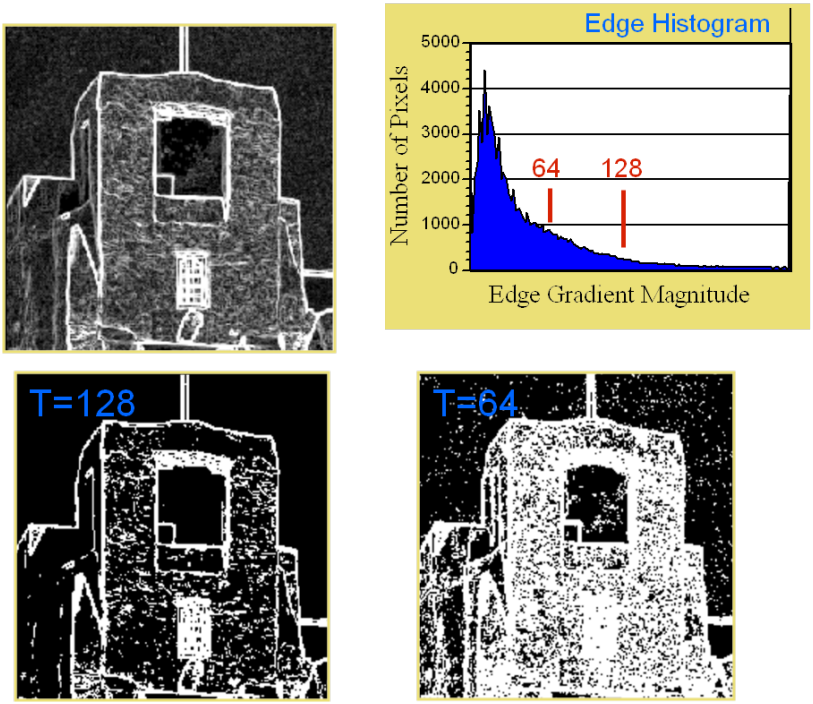


Figure 36. Edge extraction via thresholding

If we want the weights closer to the central pixel to be higher, then the Sobel operator will do the job:

$S_1 =$

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

$S_2 =$

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

Note the normalization factor for the weights would be 1/4. Sometime, we will still want to use a smaller kernels for computational reason, then a 2x2 kernel called Robert cross operator can be used:

$$\left|\begin{array}{cc} 1 & 0 \\ 0 & -1 \end{array}\right| + \left|\begin{array}{cc} 0 & 1 \\ -1 & 0 \end{array}\right|$$

Typically, only the magnitudes of the edges are calculated using the Robert operator since the two gradient components run on diagonal directions instead of the horizontal and vertical directions.

Of course edge detection kernels could be larger than 3x3, which usually lead to thicker edges, while less sensitive to noise.

Instead of obtaining edge magnitudes and orientations by calculating two gradient components using two kernels, another approach is to use eight masks aligned with the usual compass directions, select the largest response as the edge magnitude, whereas the edge orientation is the direction associated with the largest response. Figure 36 shows the eight Robinson compass masks.

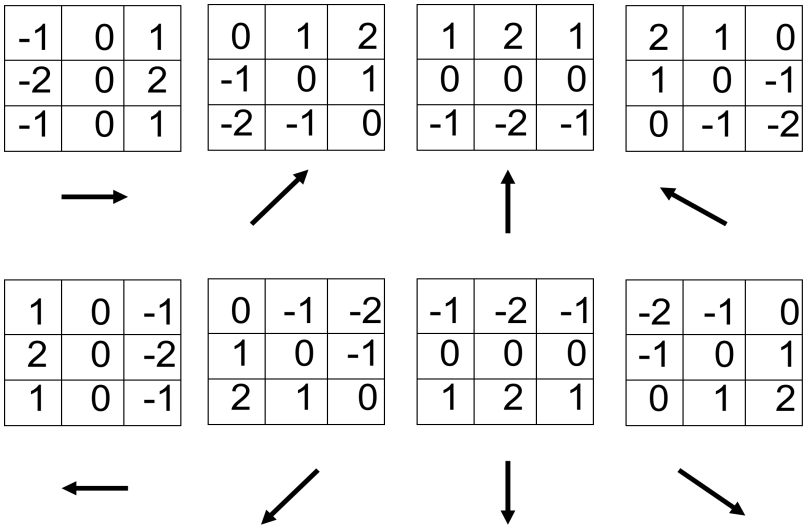


Figure 37. Robinson Compass Masks

Here we offer a theoretical analysis of edge kernels; analysis is based on a step edge inclined at an angle θ (relative to y-axis) through the center of the window.

- Error in edge contrast (magnitude)
  - Robinson/Sobel: ground truth edge contrast less than 1.6% different from that computed by the operator used.
- Error in edge direction
  - Robinson/Sobel: less than 1.5 degrees error
  - Prewitt: less than 7.5 degrees error
- Summary
  - Typically, 3 x 3 gradient operators perform better than 2 x 2.
  - Prewitt2 and Sobel perform better than any of the other 3x3 gradient estimation operators.
  - In low signal to noise ratio situations, gradient estimation operators of size larger than 3 x 3 have improved performance.

- In large masks, weighting by distance from the central pixel is beneficial.

3.3 Canny Edge Detector (TBD)

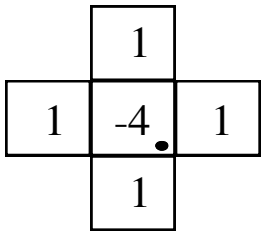
- Probably the most widely used edge detector
- LF. Canny, "A computational approach to edge detection", IEEE Trans. Pattern Anal. Machine Intelligence (PAMI), vol. PAMI vii-g, pp. 679-697, 1986.
- Based on a set of criteria that should be satisfied by an edge detector:
  - Good detection. There should be a minimum number of false negatives and false positives.
  - Good localization. The edge location must be reported as close as possible to the correct position.
  - Only one response to a single edge.

3.4 Second Order Edge Detector (TBD)

- Second derivative = rate of change of first derivative.
- Maxima of first derivative = zero crossings of second derivative.
- For a discrete function, derivatives can be approximated by differencing.
- Now consider a two-dimensional function f(x,y).
- The second partials of f(x,y) are not isotropic.
- Can be shown that the smallest possible isotropic second derivative operator is the Laplacian:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{17}$$

- Two-dimensional discrete approximation is:



- Second derivative, like first derivative, enhances noise
- Combine second derivative operator with a smoothing operator.

- Questions:
  - Nature of optimal smoothing filter.
  - How to detect intensity changes at a given scale.
  - How to combine information across multiple scales.
- Smoothing operator should be
  - 'tunable' in what it leaves behind
  - smooth and localized in image space.
- One operator which satisfies these two constraints is the Gaussian:

$$G(x,y) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\left[\frac{(x^2+y^2)}{2\sigma^2}\right]} \qquad (18)$$

- Marr and Hildreth approach:

1. Apply Gaussian smoothing using σ's of increasing size:
2. Take the Laplacian of the resulting images:
3. Look for zero crossings.

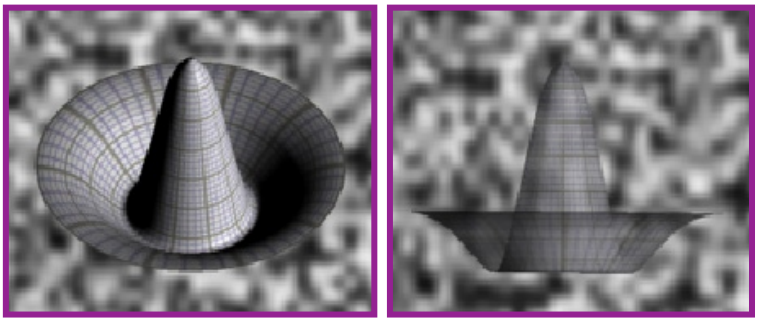
- Second expression can be written as:

$$(\nabla^2 G) \circledast I$$

- Thus, can take Laplacian of the Gaussian (LOG) and use that as the operator.

$$\nabla^2 G(x,y) = \frac{-1}{\pi\sigma^4}\left[1 - \frac{(x^2+y^2)}{2\sigma^2}\right] e^{-\left[\frac{(x^2+y^2)}{2\sigma^2}\right]}$$

- $\tilde{N}^2G$ is a circularly symmetric operator. Also called the hat or Mexican-hat operator

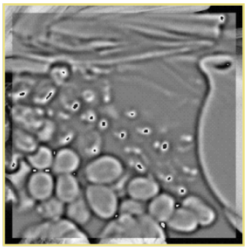


Multi-Resolution Scale Space

- Observations:

- For sufficiently different σ's, the zero crossings will be unrelated unless there is 'something going on' in the image.
- If there are coincident zero crossings in two or more successive zero crossing images, then there is sufficient evidence for an edge in the image.
- If the coincident zero crossings disappear as scales become larger, then either:
  - two or more local intensity changes are being averaged together, or
  - two independent phenomena are operating to produce intensity changes in the same region of the image but at different scales.

■ Use these ideas to produce a 'first-pass' approach to edge detection using multi-resolution zero crossing data.

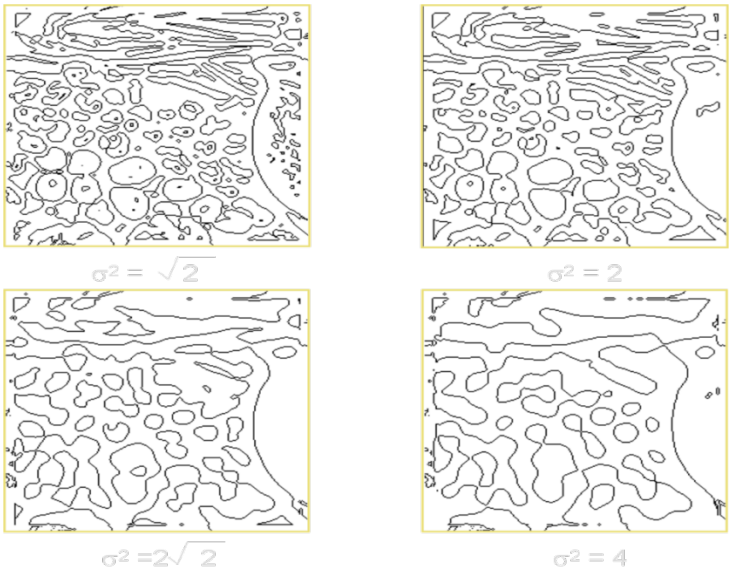


17x17 LoG Filter

Thresholded Positive

Thresholded Negative

Zero Crossings



3.5 Hough Transform: From Edgels to Structures

Given local edge elements (edgels), can we organize these into more 'complete' structures, such as straight lines? Let us first discuss how to group edge points into lines; the technique is called the Hough transform. Then the technique can be extended to other structures.

The idea behind the Hough transform is that a simple change in representations converts a *point grouping problem* into a *peak detection problem*, which is an easier problem to solve than the former one. Here is the problem: Given a set of local edge elements, with or without orientation information, how can we extract longer straight lines?

Here is the general idea of the Hough transform:

Step 1. Find an alternative space in which lines map to points.

Step 2. Each edge element 'votes' for the straight line which it may be a part of.

Step 3. Points receiving a high number of votes might correspond to actual straight lines in the image.

3.5.1 Voting in the (m,b) space

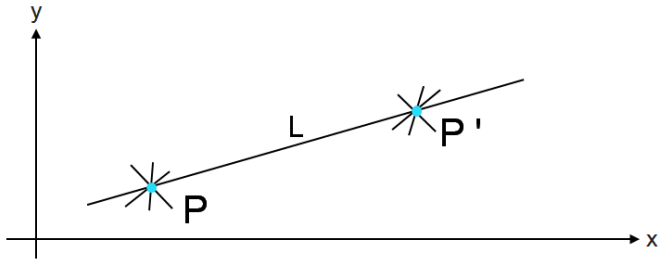


Figure 38: Hough transform: two points and two sets of lines

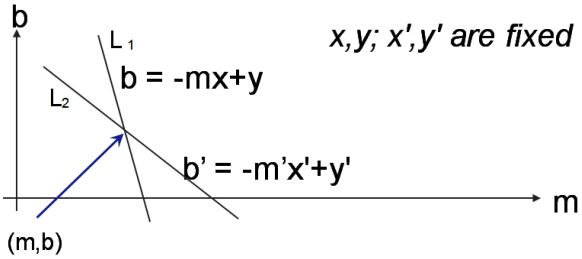


Figure 39: Hough Transform: two lines in parameter space

Consider two (edge) points, P(x,y) and P'(x',y') in image space (Figure 38). The set of all lines through P=(x,y) is y=mx + b, for appropriate choices of m and b. Similarly, we have a set of lines for P'. These two sets of lines share the same line that passes both P and P'. But the first line equation is also the equation of a line L_1 in (m,b) space, or the *parameter space*. Similarly we have the line L_2 in the parameter space for the second equation. These two lines intersect at a point (m,b), which is exactly the parameters (slope and intercept) for the line PP'.

With this we can restate the general idea of the Hough transform:

1. Construct a Hough space (m,b), which is a representation of every possible line segment in the plane.
2. Make the Hough space (m and b) discrete.
3. Let every edge point (edgel) in the image plane 'vote for' any line it might belong to.

The steps of the Line Detection Algorithm using the Hough transform are (Figure 40):

1. Quantize b and m into appropriate 'buckets', where we have to decide what the 'appropriate' intervals between the buckets (bins) are.
2. Create an accumulator array H(m,b), all of whose elements are initially zeros.

3. For each point (i,j) in the edge image for which the edge magnitude is above a specific threshold, increment all points in H(m,b) for all discrete values of m and b satisfying b = -mj+i. Note that H is a two dimensional histogram.
4. Local maxima in H corresponds to collinear edge points in the edge image, thus a line segment is detected.

Using the Hough transform, the problem of line detection in image space has been transformed into the problem of cluster detection in parameter space. Figure 41 shows an example where two lines are detected.

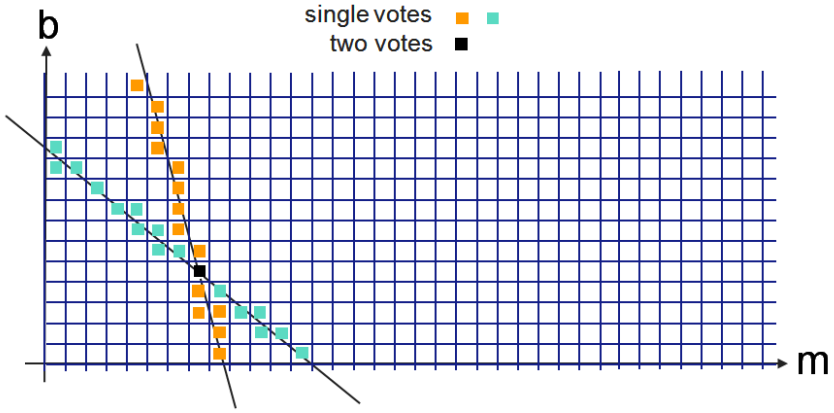


Figure 40: Voting in the parameter space

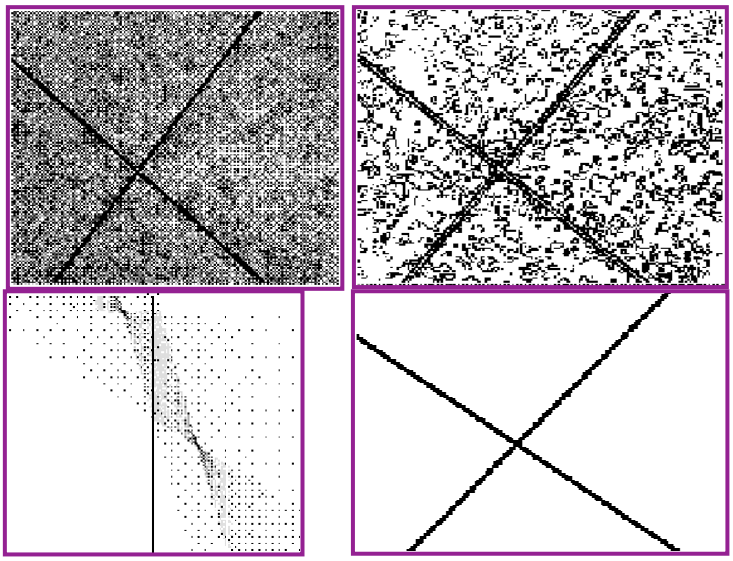


Figure 41: Row 1: original image and edge image; Row 2: accumulator array H(m,b), and the detected results corresponding the two highest peaks in H(m,b)

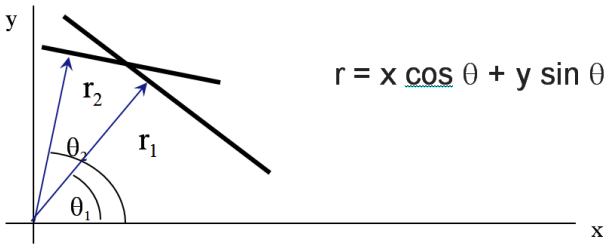


Figure 42: Polar coordinate representation

3.5.2 Voting in the (r,θ) space

In principle, the above description provides the basics about the Hough Transform. But there is a problem: vertical lines have infinite slopes therefore it is difficult to quantize m to take this into account. Hence, an alternative parameterization of a line is normally used, such as a polar coordinate representation (Figure 42).

$$r = x \cos\theta + y \sin\theta \tag{19}$$

We can see that the (r,θ) space is an efficient representation, in that

(1) it is small: only two parameters (like y=mx+b);

(2) it is finite: $0 \le r \le \text{sqrt}(\text{row}^2+\text{col}^2)$, $0 \le \theta \le 2\pi$;

(3) it is unique: only one representation per line

We should note that each edgel will generate a curve instead of a line in (r,θ) space, which is now a sinusoid (Figure 43); but the Hough transform algorithm remains valid.

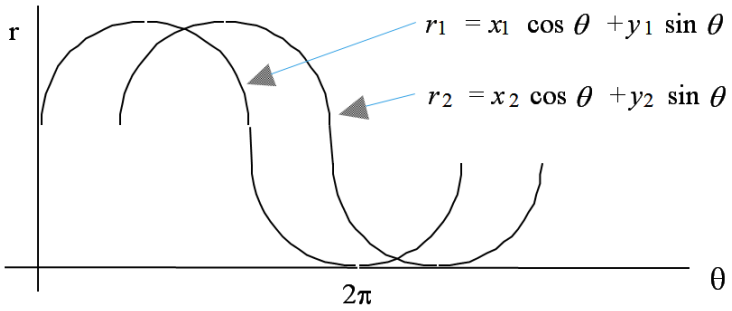


Figure 43: Voting in the (r,θ) space; note that the two sinusoidal curves have one intersection when $0 \le \theta \le 2\pi$

Figure 44 shows a real example of the Hough transform in the (r,θ) space.

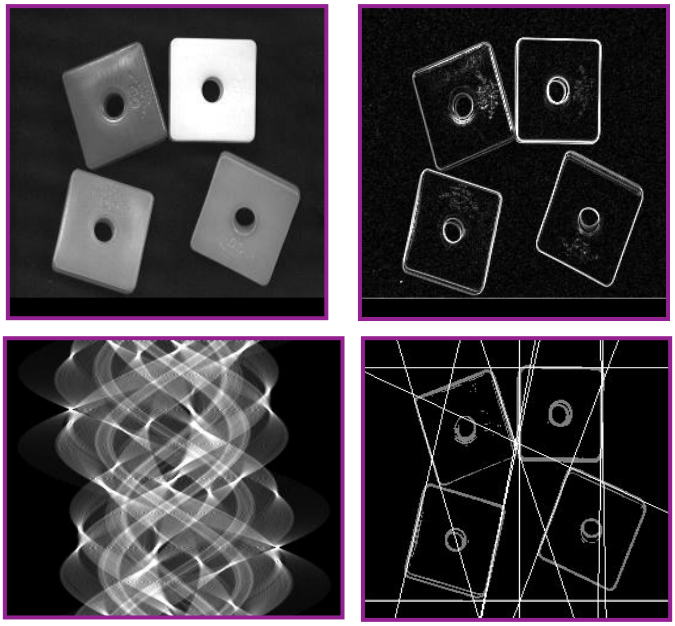


Figure 44: An example. Row 1: original image and edge image; Row 2: accumulator array H(r,θ), and the detected results corresponding the highest peaks in H(r,θ). Note that that this voting approach results in edges that may extend from one edge to the other, so some sort of post processing is necessary to match up with the actual extent of the edge in the picture.

3.5.3 Using edgel orientations

We can make one more improvement. Note that the voting technique, either in the (m,b) space or the (r,θ) space, only uses the fact that an edgel exists at point (i,j). Recall that we can also estimate orientation of the edge! The orientation of the edge provides more constraints: Using the estimate of edge orientation we could obtain θ so that each edgel now maps to a point in the Hough space. In the ideal case, three collinear points will have the same orientation, therefore having the same (r,θ).

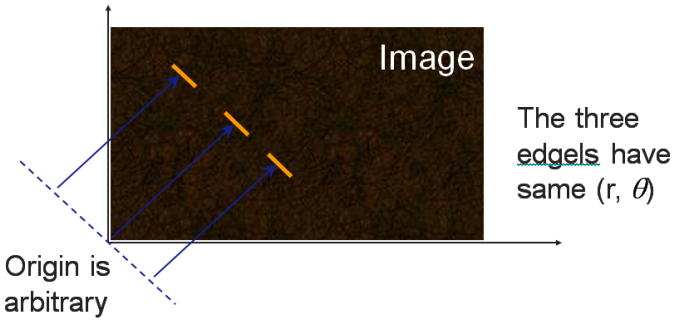


Figure 45: Using the orientation information

In practice, the orientation estimates are not accurate due to the local operators we usually use. Therefore collinear edges in Cartesian coordinate space now form point clusters in (m,b) parameter space, or the (r,θ) space. This means the average location gives the parameters of the line that fits the three edgels.

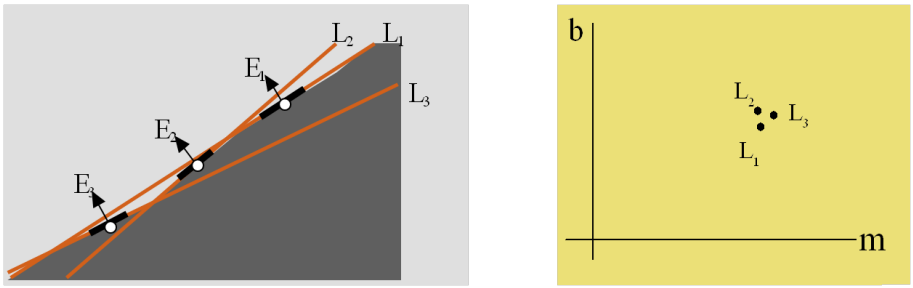


Figure 46: Collinear edges in Cartesian coordinate space (left) form point clusters in (m,b) parameter space (right)

Note that edge location (e.g., the start and the end) in the image is lost using the Hough transform. Consequently, we still need to do some image space manipulations, e.g., an edge 'connected components' algorithm. [ref] Heikki Kälviäinen, Petri Hirvonen, L. Xu and Erkki Oja, "Probabilistic and nonprobabilistic Hough Transforms: Overview and comparisons", *Image and vision computing*, Volume 13, Number 4, pp. 239-252, May 1995.

3.5.4 Generalization

The Hough transform technique can be generalized to any parameterized curve:

$$f(x,a) = 0 \tag{20}$$

where **x** = (x,y) is the image coordinates and **a** is the parameter vector in the Hough space. Success of this technique depends upon the quantization of the parameters: if it is too coarse, the maxima will be 'pushed' together. On the other hand, if it is too fine, peaks will be less defined, meaning the votes will be spread out into a cluster of bins. Also note that exponential growth in the dimensions of the accumulator array with the number of curve parameters restricts its practical application to curves with a few parameters (usually, 1, 2 or 3).

Now as an example, we show how to find a circle. A circle has three parameters: two for its center (a,b) and one for its radius r. Therefore we have

$$f(i,j,r) = (i-a)^2+(j-b)^2-r^2 = 0 \quad (21)$$

To show the principle easier, let us start with this task: Find the center of a circle with *known* radius r when given an edge image with no gradient direction information (i.e., only edge locations are known). For this task, we have **x** = (i,j), the Cartesian space, and **a** = (a,b), the parameter space (Hough space). Figure 47 illustrates how to find the circle with a known radius.

The goal is to find the center of the circle from a number of edge points (edgels) on the circle. Given an edge point at (i,j) in the image (as shown in the top-left image), equation (21) also represents a circle in the parameter space (a,b), where (i,j) is the center and r is the radius, as shown in the top-right image. Two points in the image generate two circles in the parameter space, which intersect at two points (as shown in the bottom-left image). With more edge points on the image circle, a lot of votes are accumulated in the circle center location (a,b) in the parameter space, therefore the center is found!

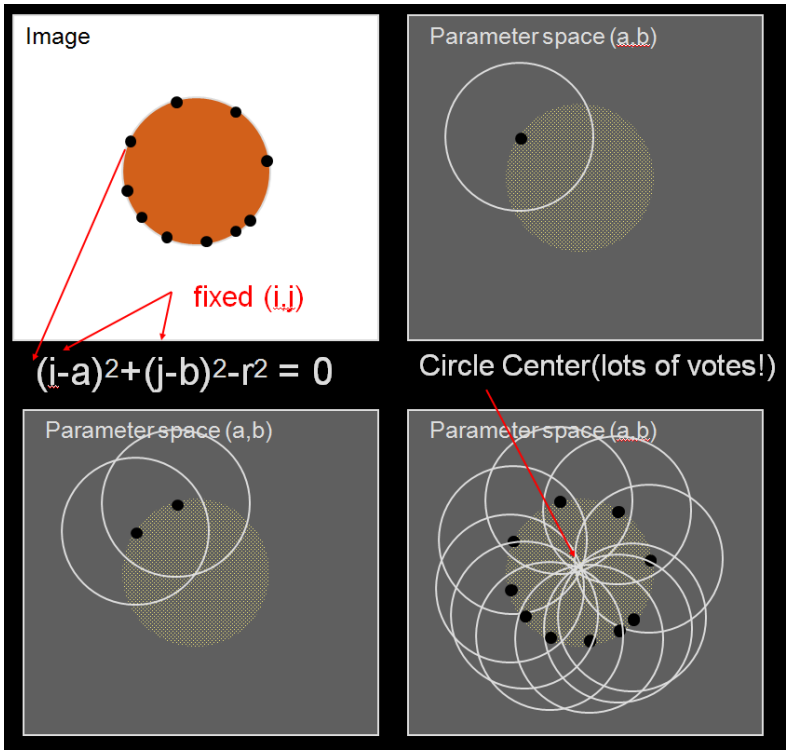


Figure 47: Finding a circle of known radius

Generally, we would like to find a circle without assuming the radius is known. In this case the accumulator array is 3-dimensional (3D), which

means the computation will be more time-consuming. However, if edge directions are known or approximately known, such as being provided by edge detectors, the computational complexity can be greatly reduced. Suppose there is a known error limit on the edge direction (say +/- 10^o), we could use this piece of information to constrain the search range of the three parameters.

The Hough transform can be extended in many ways. For more information, the readers can check out [refs]

- Ballard, D. H. Generalizing the Hough Transform to Detect Arbitrary Shapes, Pattern Recognition 13:111-122, 1981.
- Illingworth, J. and J. Kittler, Survey of the Hough Transform, Computer Vision, Graphics, and Image Processing, 44(1):87-116, 1988

4. REGION, CONTOUR AND CORNER DETECTION

4.1 Region Segmentation

4.2 Contour Extraction

4.3 Corner Detection

5. CONCLUDING REMARKS

6. QUESTIONS AND PROJECTS

6.1 Questions

6.2 Projects

1. Generate the histogram of an image you can find online. If it is a color image, first turn it into an intensity image and then generate its histogram. Try to display your histogram.

2. Apply the 1x2 operator and Sobel operator to your image and analyze the

results of the gradient magnitude images. Does the Sobel operator have any clear visual advantages over the 1x2 operator? If you subtract the 1x2 edge image from the Sobel are there any residuals? (Note: don't forget to normalize your results)

3. Generate edge maps of the above gradient maps. You may first generate a histogram of each gradient map, and only keep certain percentage of pixels (e.g. 5% of the highest gradient values) as edge pixels(edgels). Use the percentage to find a threshold for the gradient magnitudes.

4. What happens when you increase the size of the kernel to 5x5, or 7x7? Discuss computational cost (in terms of members of operations, and the real machine running times), edge detection results and sensitivity to noise, etc. Note that your larger kernel should still be an edge detector.

5. Suppose you apply the Sobel operator to each of the RGB color planes comprising the image. How might you combine these results into a color edge detector? Do the resulting edges differ from the gray scale results? How and why?

REFERENCES

Digital Image Processing
Computer Graphics
Photogrammetry