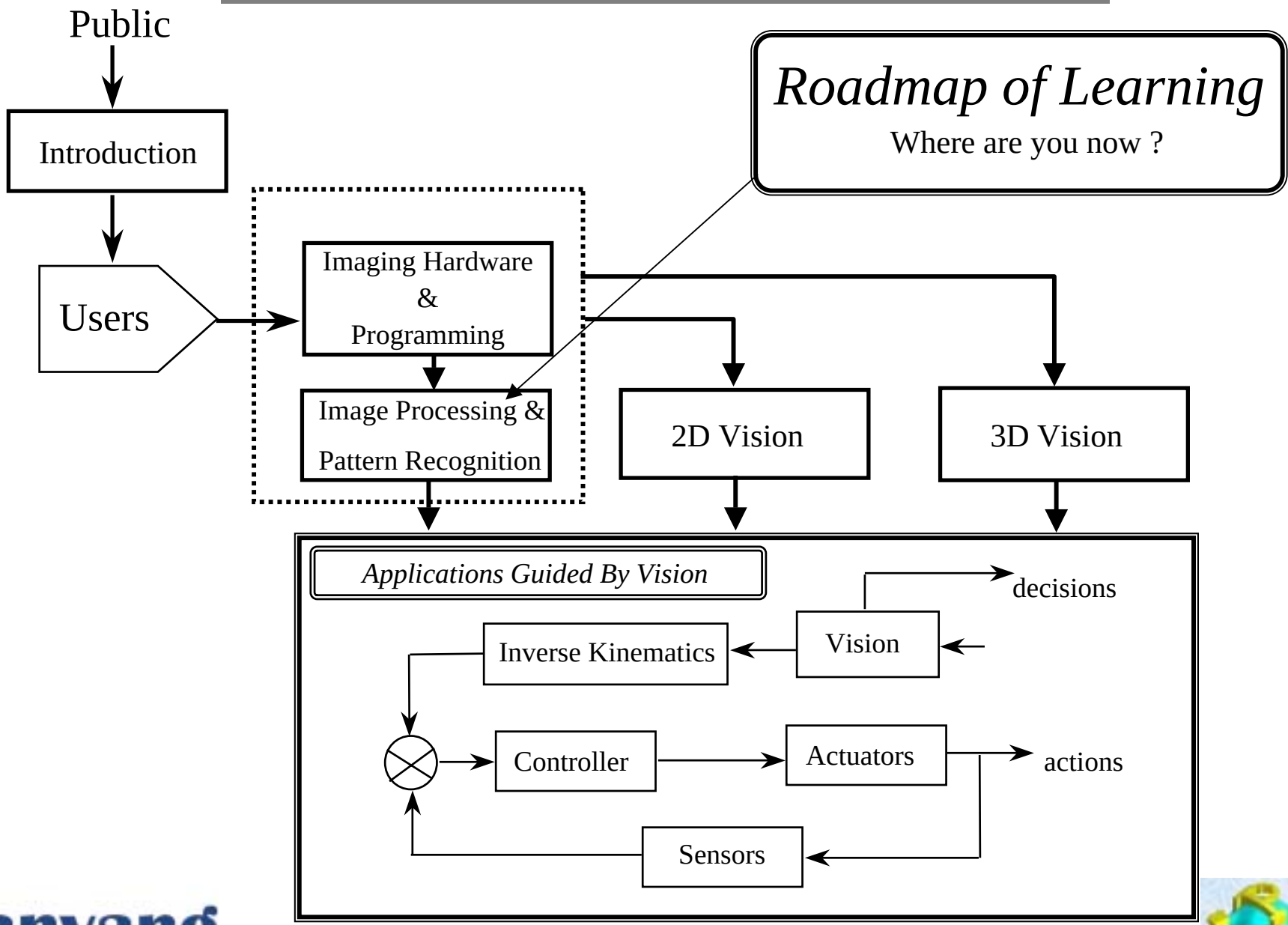
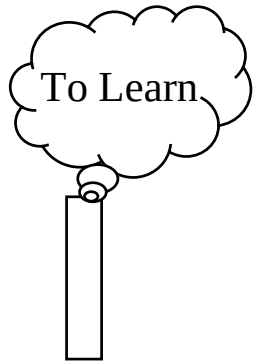


CONTENT



Chapter 4: Fundamentals of Image Processing

4.1 Point-Based Processing Techniques

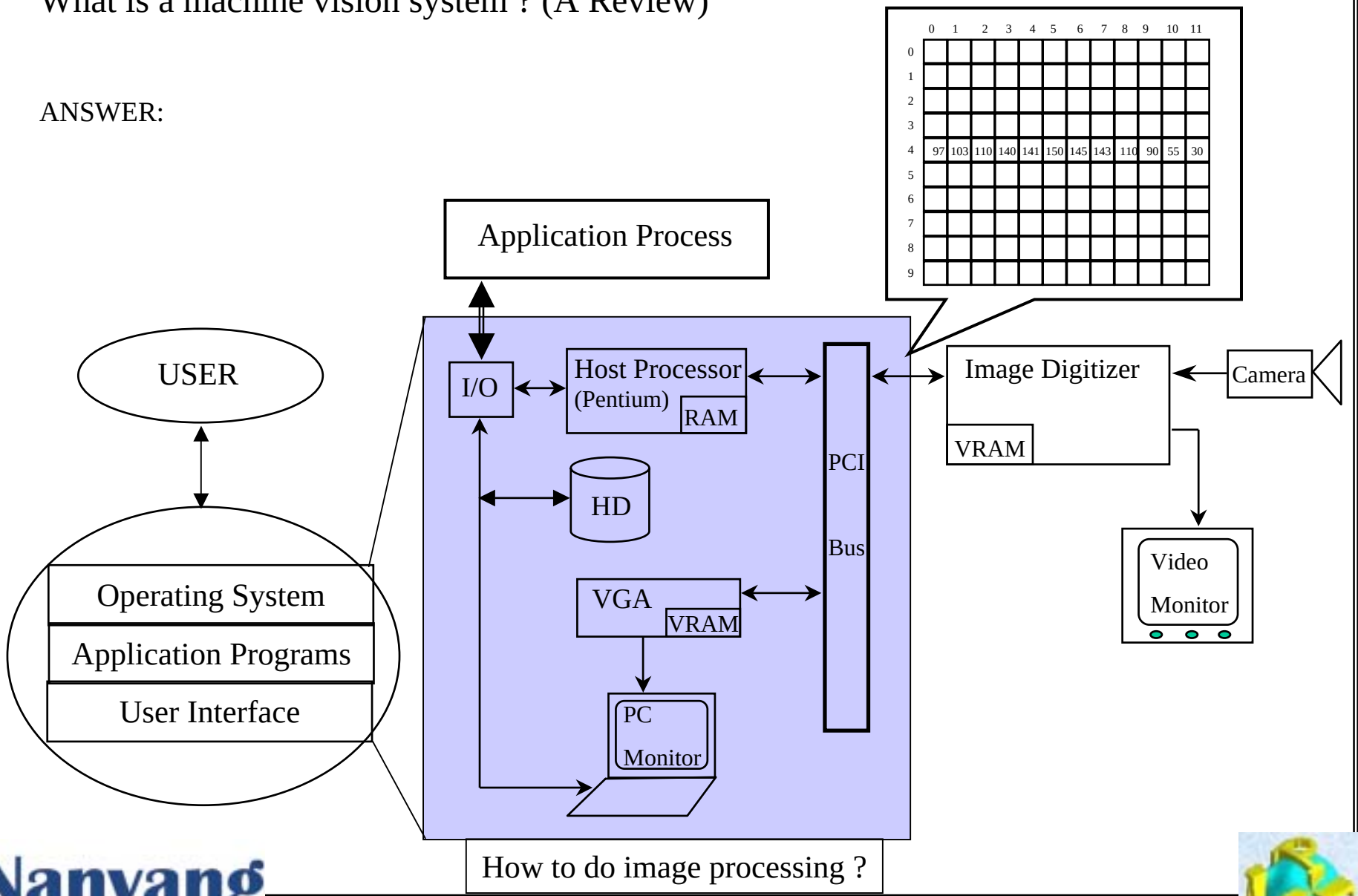
4.2 Frame-Based Processing Techniques

4.3 Area-Based Processing Techniques

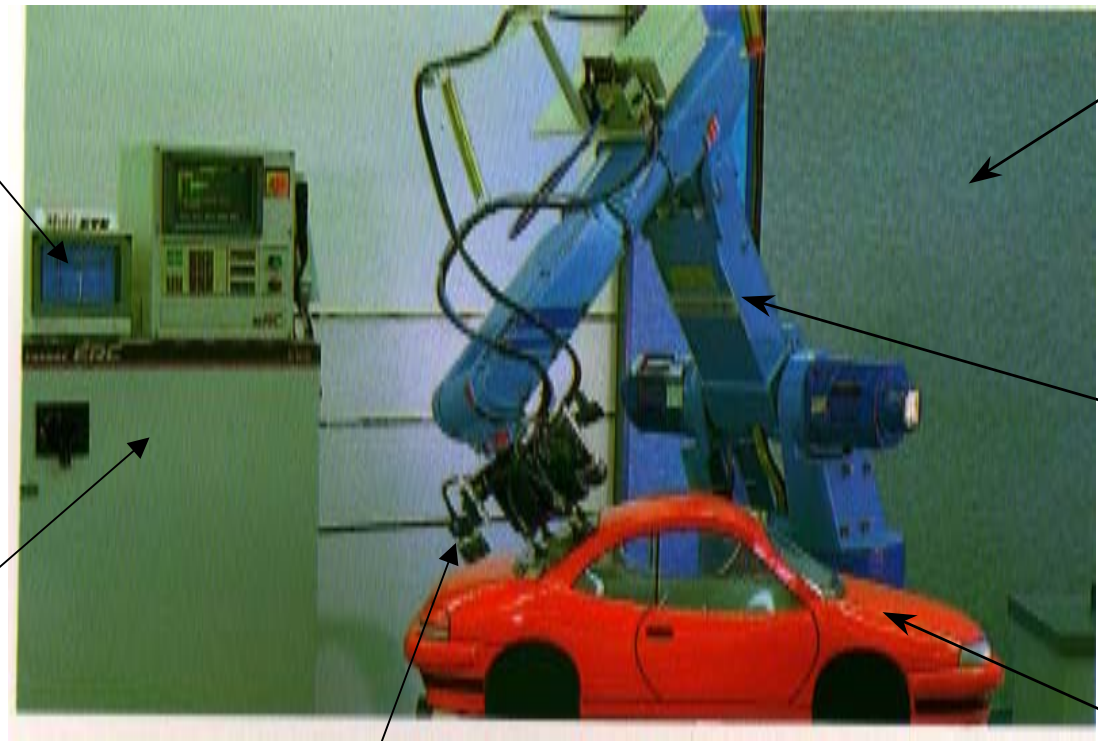
4.4 Image Primitive Extraction

What is a machine vision system ? (A Review)

ANSWER:



Why is there a need to do image processing ?



Why is there a need to do image processing ?

ANSWER:

1. An image may contains a lot of objects. But, not all of them are relevant to an application.

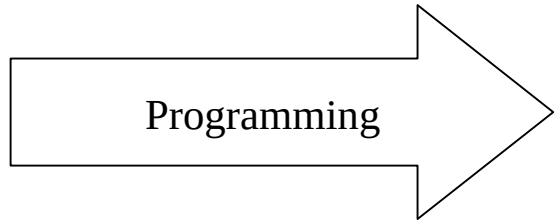
2. There is a need to detect the object of interest.

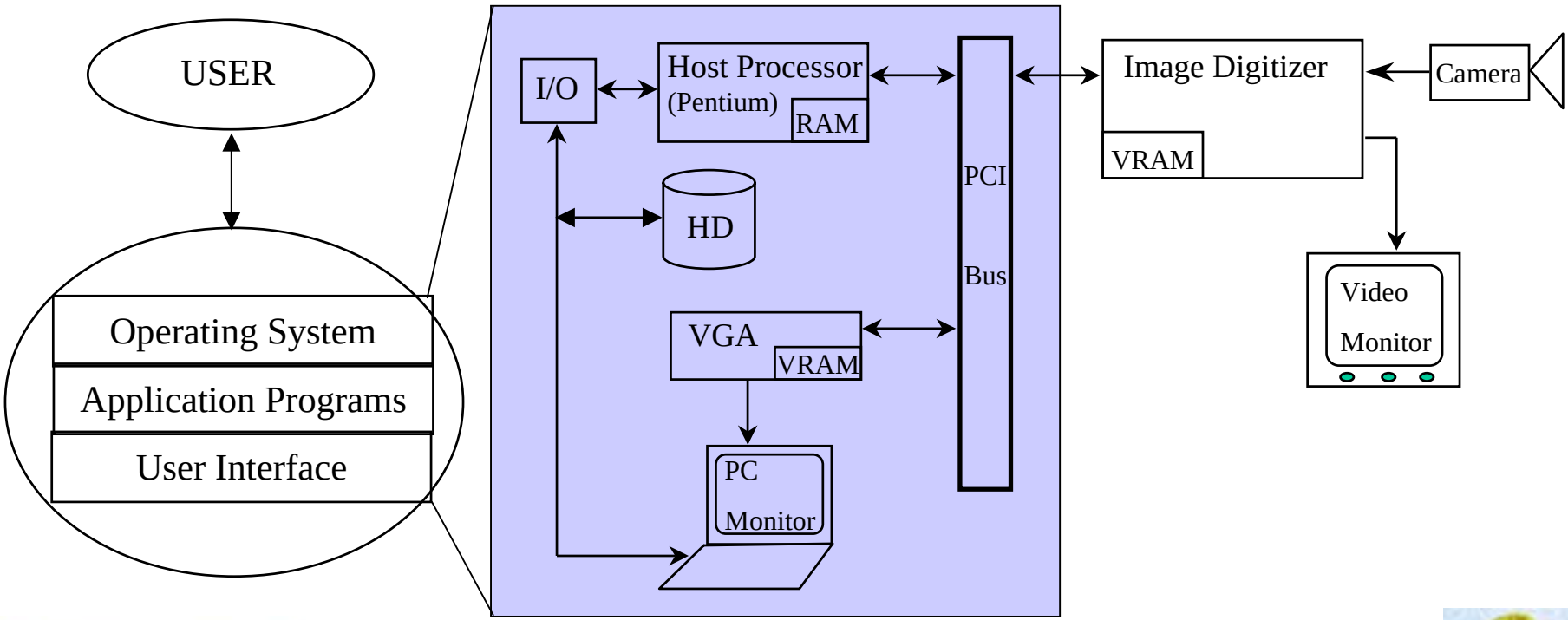
3. The only way to identify an object in an image is to compute the characteristics that are unique to the object of interest so that this object can be distinguished from other objects according to its image characteristics.

How to do image processing ?

ANSWER:

1. To use computing system.
2. To choose appropriate image processing algorithms.
3. To implement these algorithms.



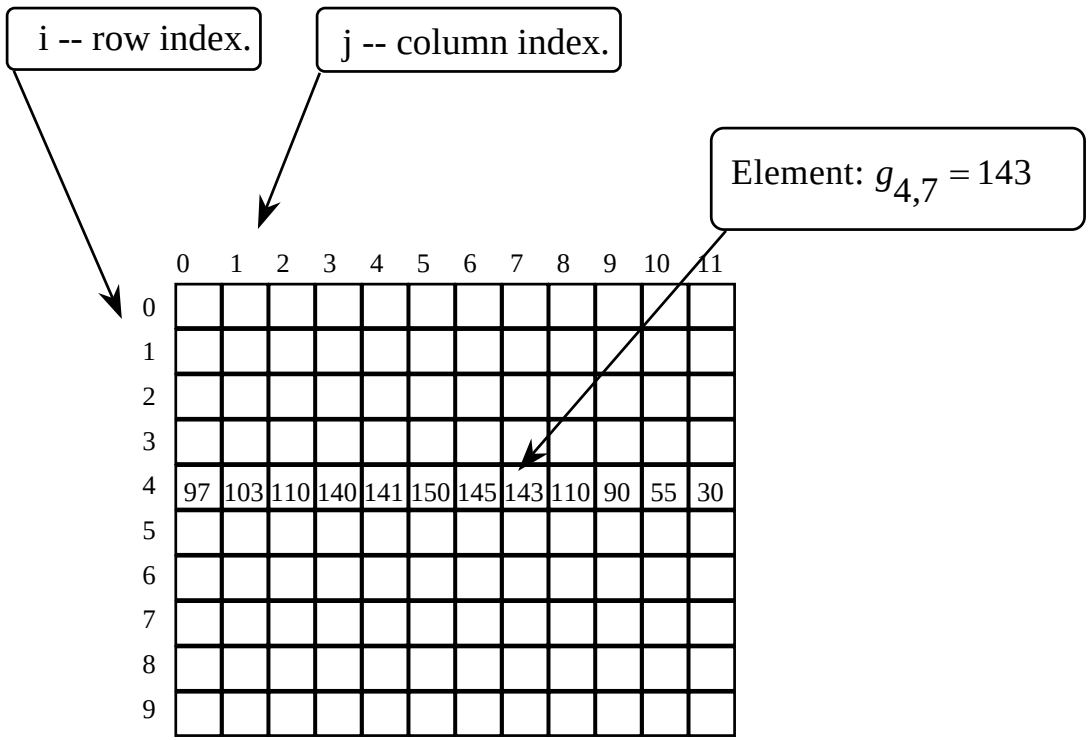


How is a digital image represented in a computing system ?

ANSWER: 1. A digital image is represented by a two-dimensional matrix.

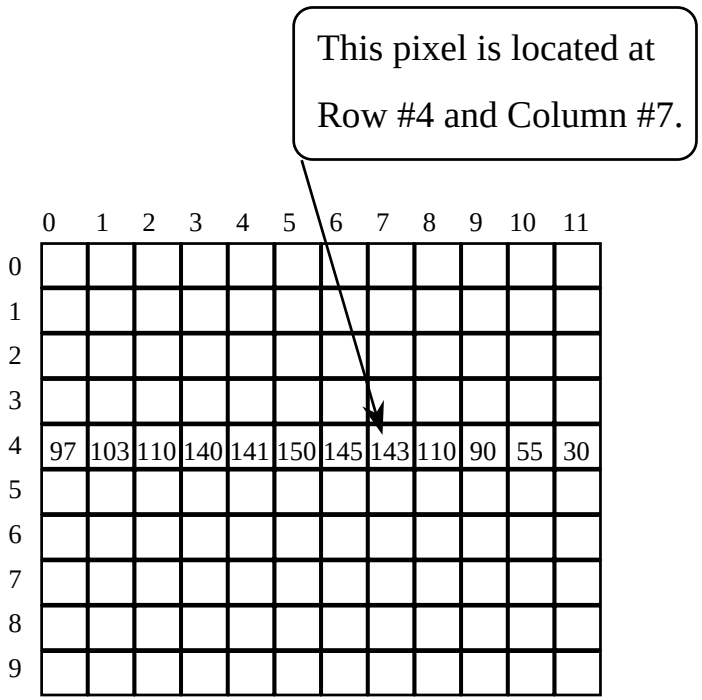
2. The elements in the image matrix are called “pixels”.

$$I = \{ g_{i,j} \mid \quad 0 \le i \le n;\ 0 \le j \le m \}$$



3. The position of a pixel in terms of column and row is called "pixel location":

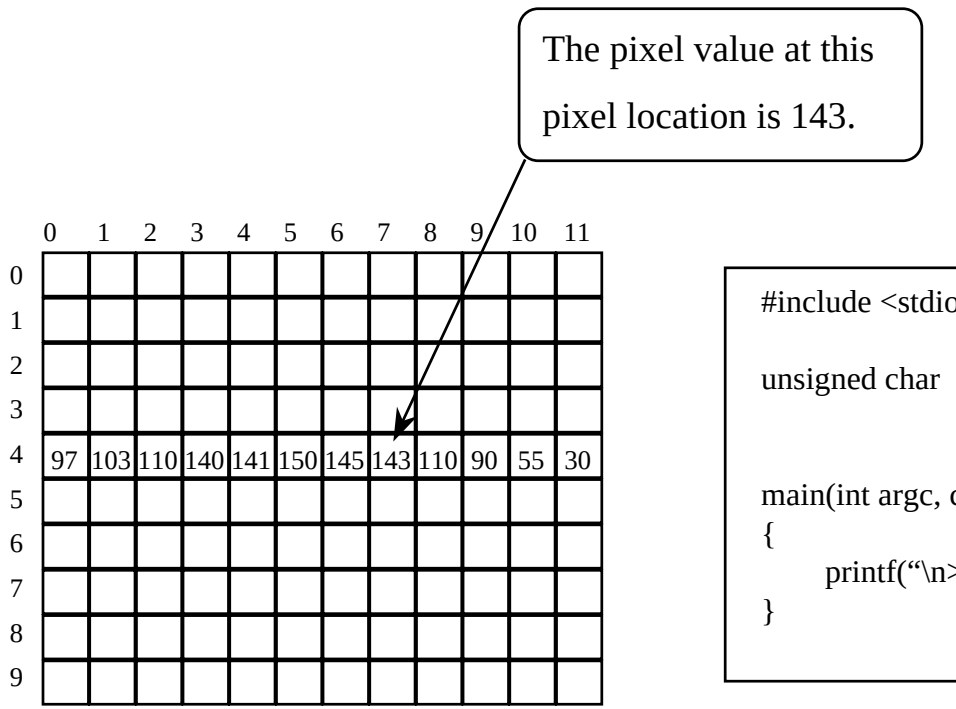
$$I = \{ g_{i,j} \mid \quad 0 \le i \le n,\ 0 \le j \le m \}$$




```
#include <stdio.h>

unsigned char  image[512*512] ;

main(int argc, char **argv)
{
        image[4*512+7] = 143 ;
}
```

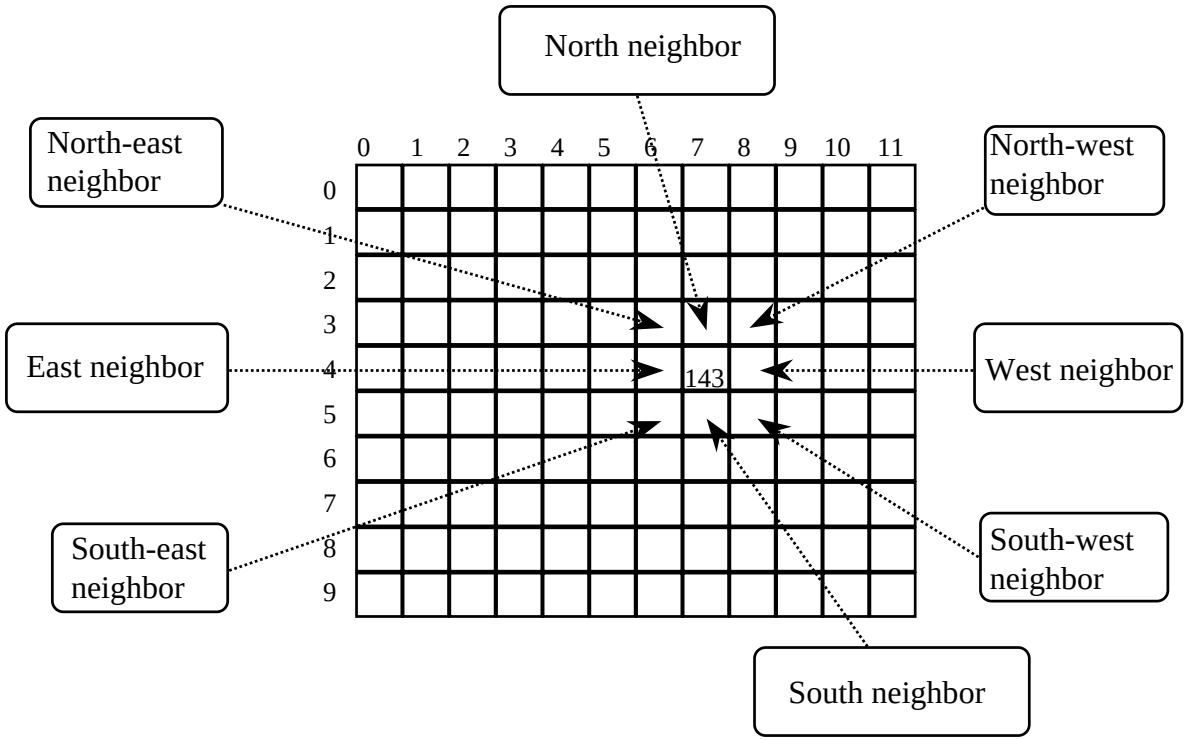

4. The gray-level at a pixel location is called “intensity value” or “pixel value”:

$$I = \{ g_{i,j} \mid \quad 0 \le i \le n,\ 0 \le j \le m \}$$



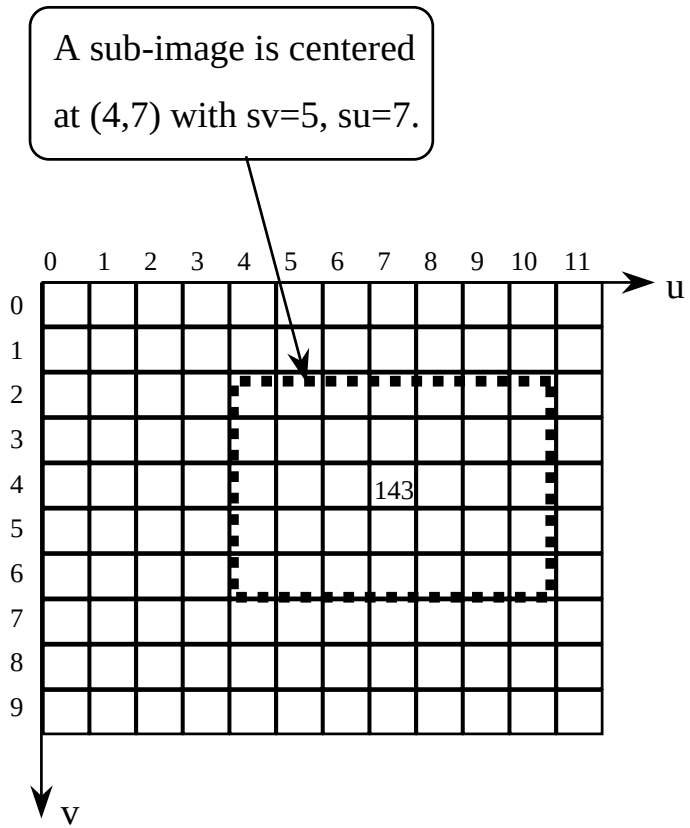

```
#include <stdio.h>

unsigned char  image[512*512] ;


main(int argc, char **argv)
{
      printf("\n> Pixel value at (4, 7) = %d", image[4*512+7])  ;
}
```


5. A pixel may have up to eight neighbors that are the adjacent pixels around it:




```
#include <stdio.h>

unsigned char  image[512*512] ;

main(int argc, char **argv)
{
      printf("\n> Pixel value at (5, 8) = %d", image[5*512+8])  ;
}
```

6. An object usually occupies a sub-area of an image. We call a sub-image a "window" or "region" in the image plane: $\text{Sub}(I, i, j) = \{ (g^{sub}_{s,t} = g_{i-sv/2+s,\, j-su/2+t}) \mid 0 \le s < sv, 0 \le t < su \}$




```
#include <stdio.h>

unsigned char  image[512*512] ;
unsigned char subimage[5*7] ;

main(int argc, char **argv)
{
        int   row, column, sv, su ;
        sv = 5 ; su = 7 ;
        for (row = 0; row < sv; row++)
        {
                for (column = 0 ; column < su; column++)
                        subimage[row*7+column] =
                                image[(4-sv/2+row)*512+(7-su/2+column)] ;
         }
}
```


How to process digital images ?

ANSWER:

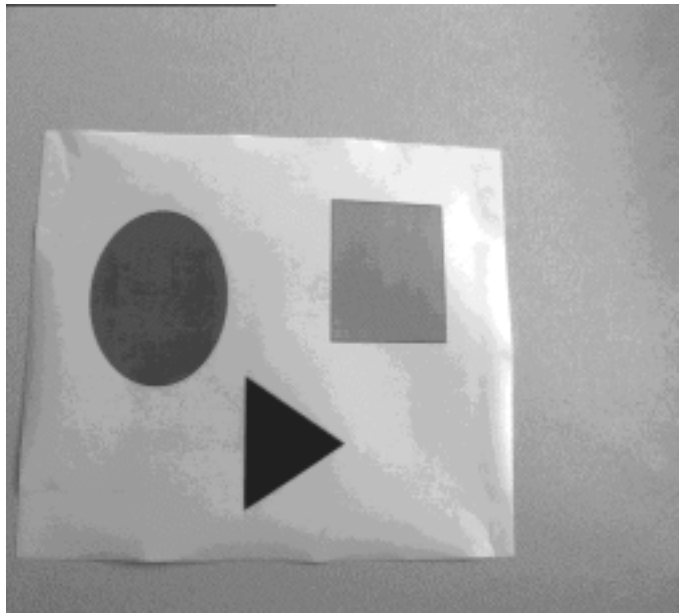
You are going to learn:

1. Point-based image processing techniques

2. Frame-based image processing techniques

3. Area-based image processing techniques

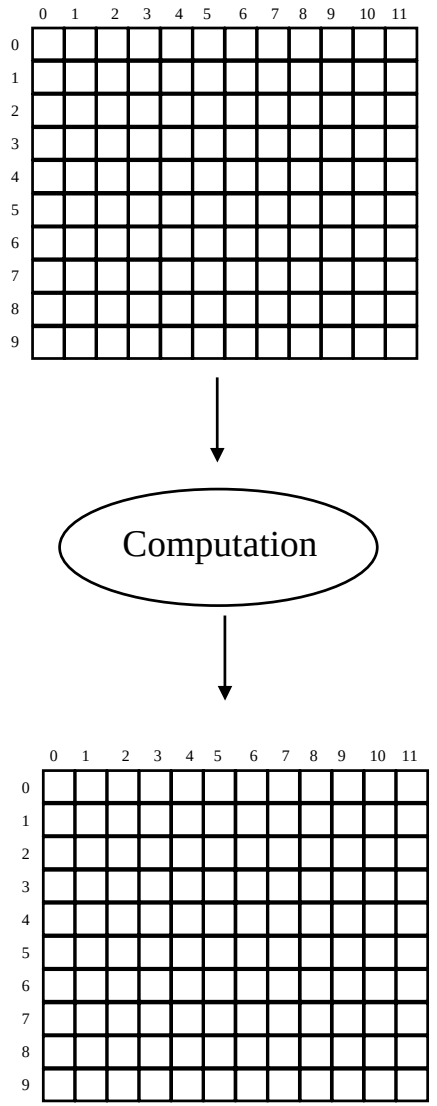
4. Image primitive extraction



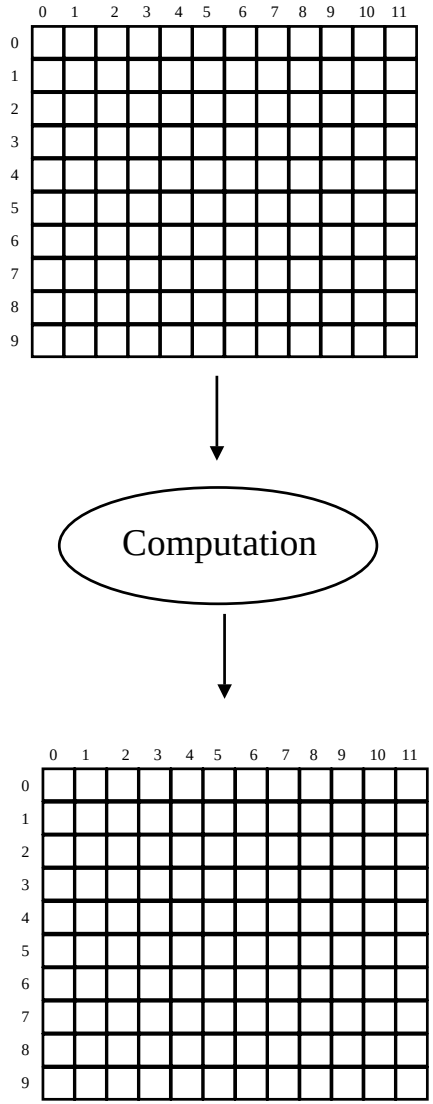
What is a Point-based image processing technique ?

ANSWER:

It refers to the individual computations carried out on each pixel location without considering the neighborhood relationship of the pixels.



Point-based Image Processing Techniques



1. Arithmetic operation(s):

Input image : $I^{in} = \{g_{i,j}^{in} \mid 0 \le i < n,\ 0 \le j < m\}$

Output image : $I^{out} = \{g_{i,j}^{out} \mid 0 \le i < n,\ 0 \le j < m\}$

Operation :

$$g_{i,j}^{out} = a \bullet g_{i,j}^{in} + b.$$

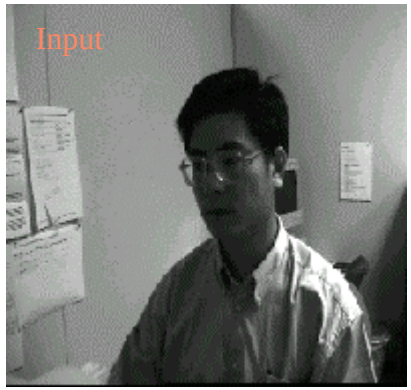
When (a=-1, b=255), this will create an inverted image.

When (a>1, b<-1), this will enhance the image contrast.

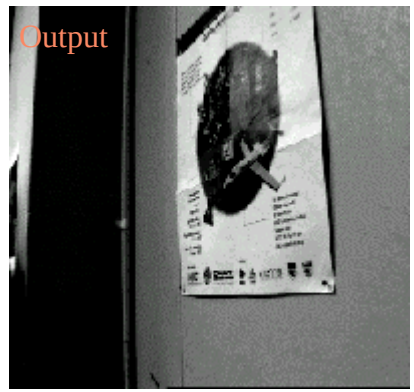
```
#include <stdio.h>

unsigned char  image_in[512*512] ;
unsigned char  image_out[512*512] ;

static void  ReadInImage()
{
}

main(int argc, char **argv)
{
        int   row, column, a, b ;

        a = -1 ; b = 255 ;
        ReadInImage() ;

        for (row = 0; row < 512; row++)
        {
                for (column = 0 ; column < 512; column++)
                        image_out[row*512+column] = (unsigned char)
                                (a*image_in[row*512+column] + b) ;
         }
}
```


Example

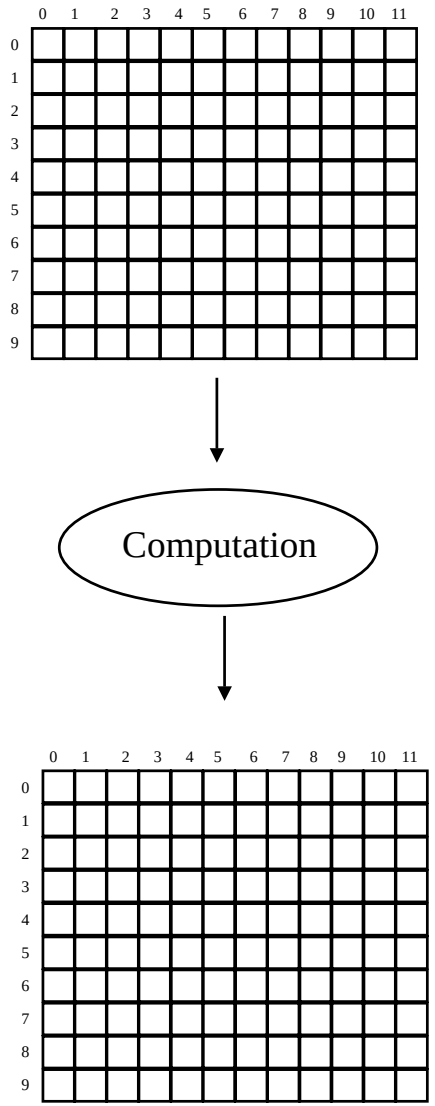
Inverted Image



Contrast Enhancement



Point-based Image Processing Techniques



2. Look-up-table operation:

Input image : $I^{in} = \{g_{i,j}^{in} \mid 0 \le i < n,\ 0 \le j < m\}$

Output image : $I^{out} = \{g_{i,j}^{out} \mid 0 \le i < n,\ 0 \le j < m\}$

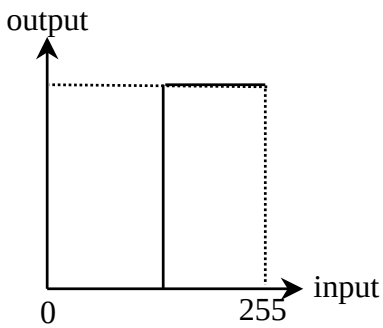
Look - up - table : $LUT = \{V_k \mid 0 \le k < 256\}$

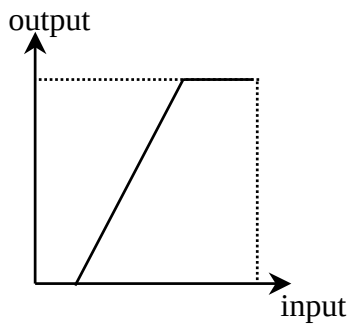
Operation :

$$\begin{cases} k = g_{i,j}^{in} \\ g_{i,j}^{out} = V_k \end{cases}$$

This operation uses the value of a pixel from the input image as the index to a vector of fixed size (256 elements if the pixel value varies from 0 to 255), and sets the value of the corresponding pixel of the output image with the vector element pointed by the index.

Input pixel value	Output pixel value
0	0
1	0
2	2
3	5
4	6
5	7
….	
255	255

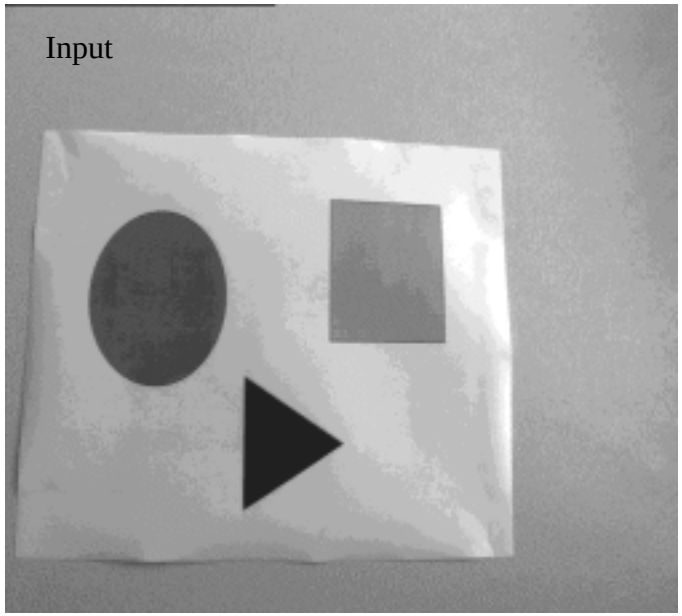


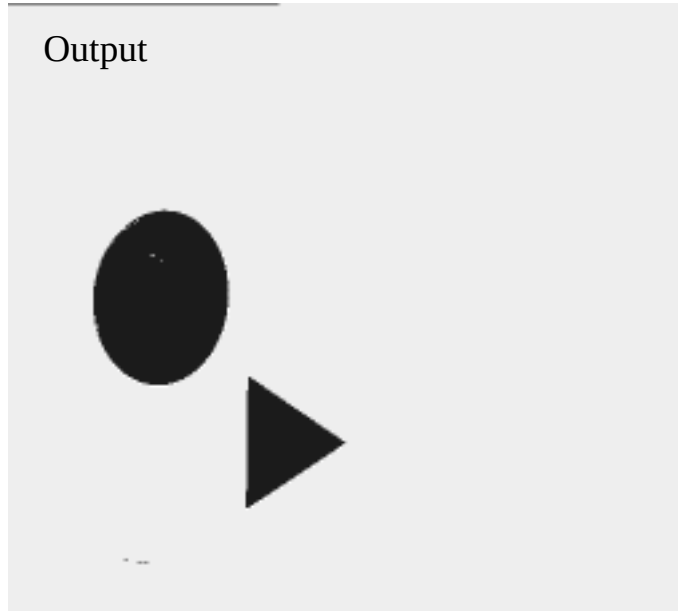


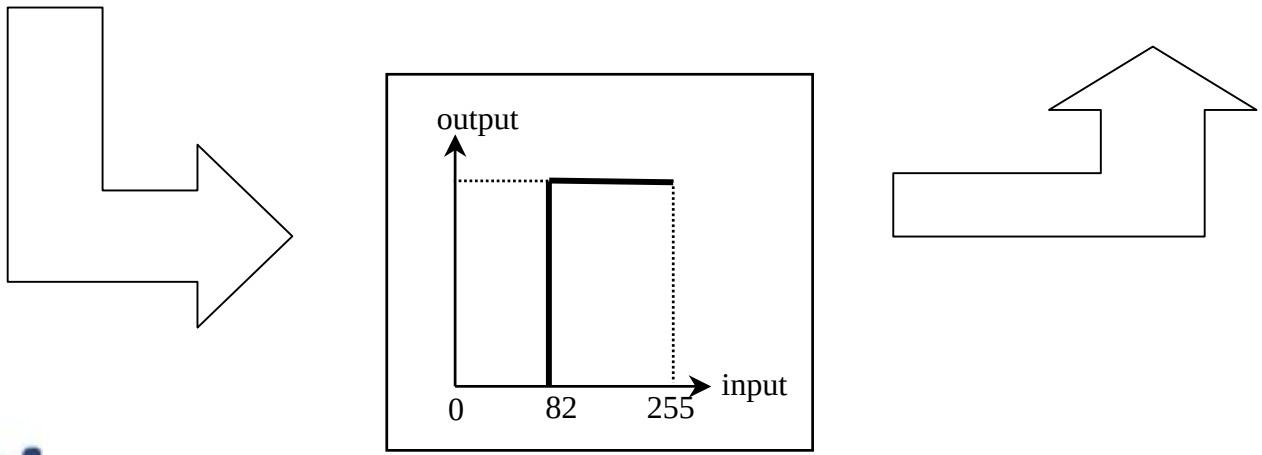
Sample Program

```
#include <stdio.h>

unsigned char  image_in[512*512], image_out[512*512] ;
unsigned char  LUT[256] ;

main(int argc, char **argv)
{
        int   row, column, index ;

        for (row = 0; row < 512; row++)
        {
                for (column = 0 ; column < 512; column++)
                {
                        index = image_in[row*512+column] ;
                        image_out[row*512+column] = LUT[index] ;
                 }
         }
}
```


Example





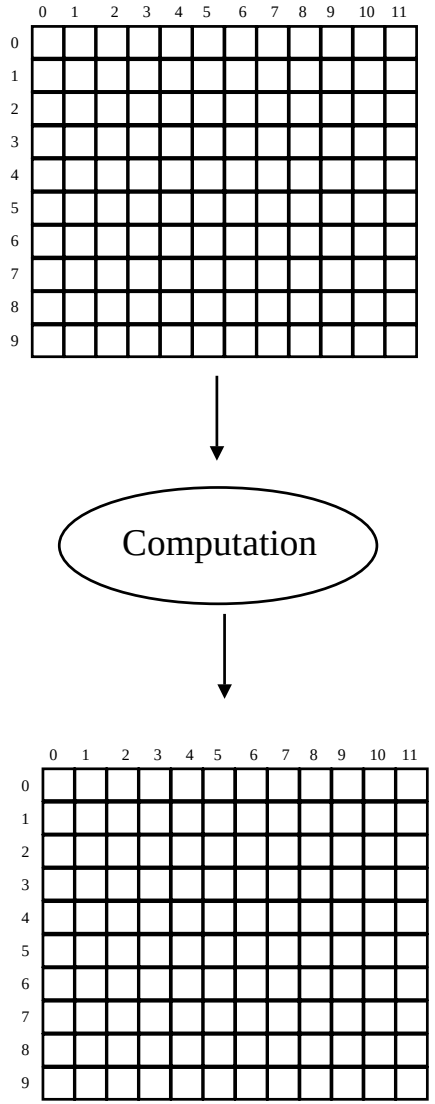


Point-based Image Processing Techniques

3. Histogram operations:

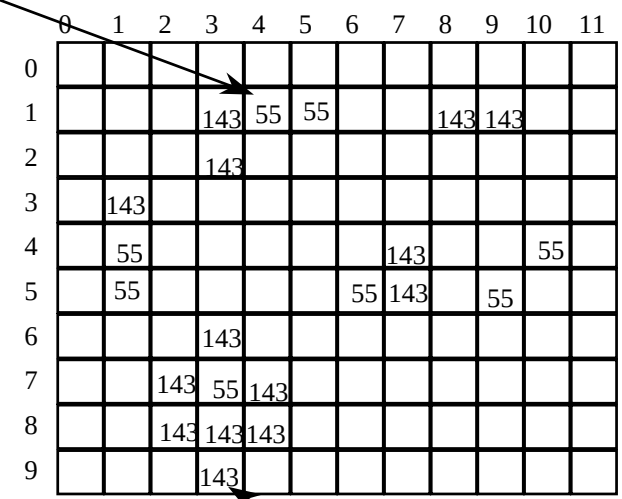
The pixel value varies from 0 to 255 if a pixel value is encoded in 8 bits. For each pixel value, we can count the number of its occurrence in an image. Therefore, we have a vector of such occurrence numbers for all pixel values. This vector is called “image histogram” that shows the intensity profile of an image.

$$H = \{h_i \mid 0 \le i \le 255\}$$



$$H = \{h_i \mid 0 \le i \le 255\}$$

Example 1: The occurrence number for the pixel value “55” is 8.



Example 2: The occurrence number for the pixel value “143” is 14.

Histogram Operation One: "Image Thresholding"

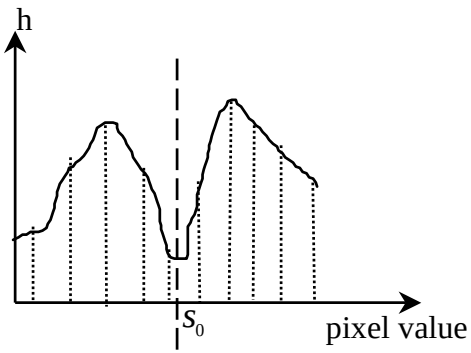
To use histogram to determine a valley point and use the corresponding pixel value as the threshold to divide the pixels into two groups: 1) the pixels of "0" value and 2) the pixels of "255" value.

Input image : $I^{in} = \{g_{i,j}^{in} \mid 0 \le i < n,\ 0 \le j < m\}$

Output image : $I^{out} = \{g_{i,j}^{out} \mid 0 \le i < n,\ 0 \le j < m\}$

Operation :

$$g_{i,j}^{out} = \begin{cases} 0. & \text{if } g_{i,j}^{in} < s_0 \\ 255. & \text{if } g_{i,j}^{in} \ge s_0 \end{cases}$$

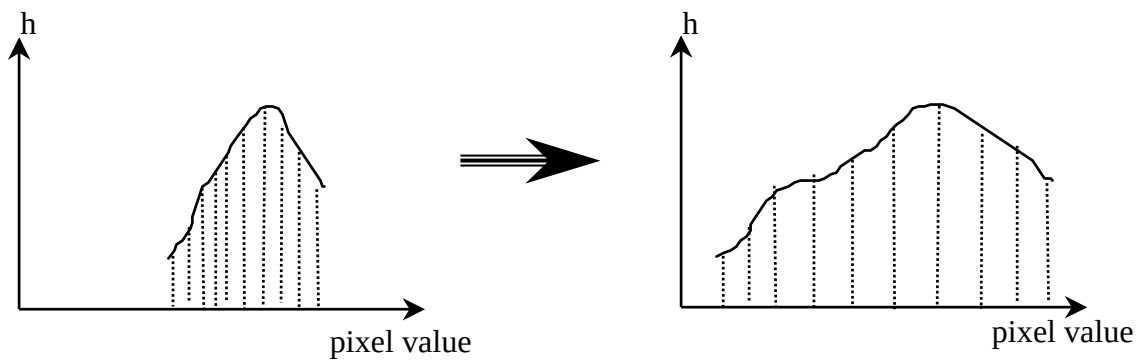


Sample Program

```
#include <stdio.h>

unsigned char  image_in[512*512] ;

main(int argc, char **argv)
{
        int   row, column, threshold ;

        threshold = s0 ;

        for (row = 0 ; row < 512 ; row++)
                for (column = 0 ; column < 512 ; column++)
                {
                        if (image_in[row*512+column] < s0)
                                image_in[row*512+column = 0 ;
                        else
                                image_in[row*512+column] = 255 ;
                }
}
```


Histogram Operation Two: “Histogram Equalization”

To spread the “intensity distribution” across the range [0,255] so as to improve the visual effect of an image.



Input image : $I^{in} = \{g_{i,j}^{in} \mid 0 \le i < n,\ 0 \le j < m\}$

Output image : $I^{out} = \{g_{i,j}^{out} \mid 0 \le i < n,\ 0 \le j < m\}$

Operation (Three Steps) :

1. To compute the histogram : $H = \{h_i \mid 0 \le i < 256\}$

2. To compute a look - up - table :

$$LUT = \{V_j = 255 * (\sum_{i=0}^{j} h_i) / (\sum_{i=0}^{255} h_i) \mid 0 \le j < 256\}$$

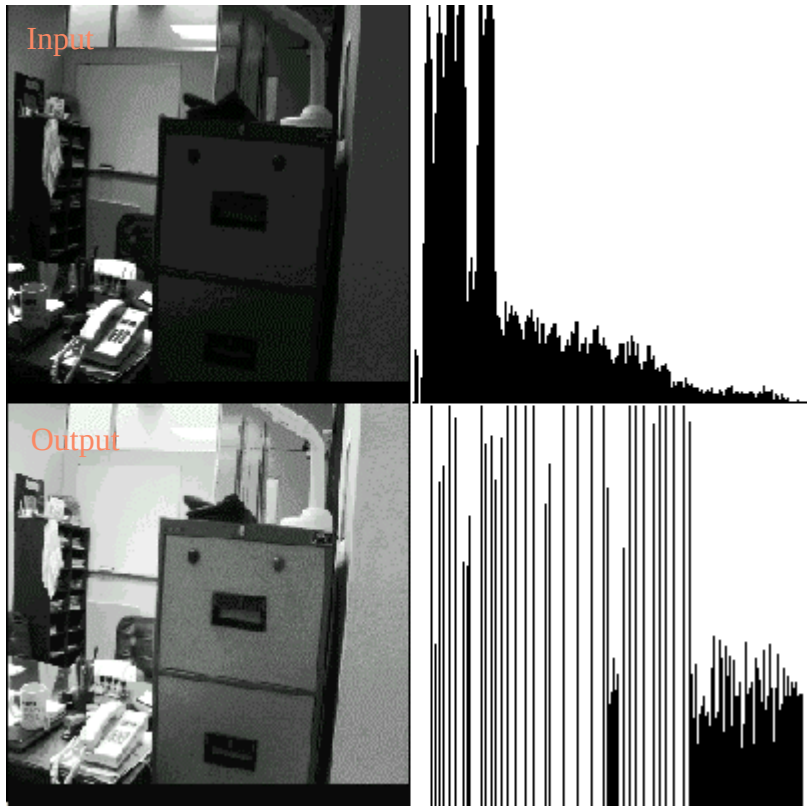
3. To apply the look - up - table to the input image.

Sample Program

```
static  int   histogram[256] ;
static  int   LUT[256] ;

static void CreateLookUpTable()
{
        int   pixel_value, index ;
        int   image_size ;

        image_size = resolution_x*resolution_y ;
        for (index = 0 ; index < 256 ; index++)
        {
                LUT[index] = 0 ;
                for (pixel_value = 0 ; pixel_value <= index; pixel_value++)
                {
                        LUT[index] = LUT[index] + histogram[pixel_value] ;
                }
                LUT[index] = 255*LUT[index]/image_size ;
         }

}
```


Result

Histogram Equalization



SUMMARY

1. A digital image is represented by a two-dimensional matrix of pixels:
   * A pixel has a location: (row, column) indices.
   * A pixel has a value: intensity, gray-level, pixel value.
   * A pixel has eight neighbors.

2. An object of interest usually occupies a sub-image.

3. It is necessary to process images in order to compute image characteristics that allow to distinguish an object of interest from the rest of image.

4. The basic Point-based processing techniques are:
   * Arithmetic operations.
   * Look-up-table operations.
   * Histogram operations.