Unit No 3: Segmentation, Texture & Motion Analysis

- Segmentation: Edge Detection (Prewitt, Sobel, Canny), Optimum Edge Detection, Thresholding techniques, Region-based segmentation.

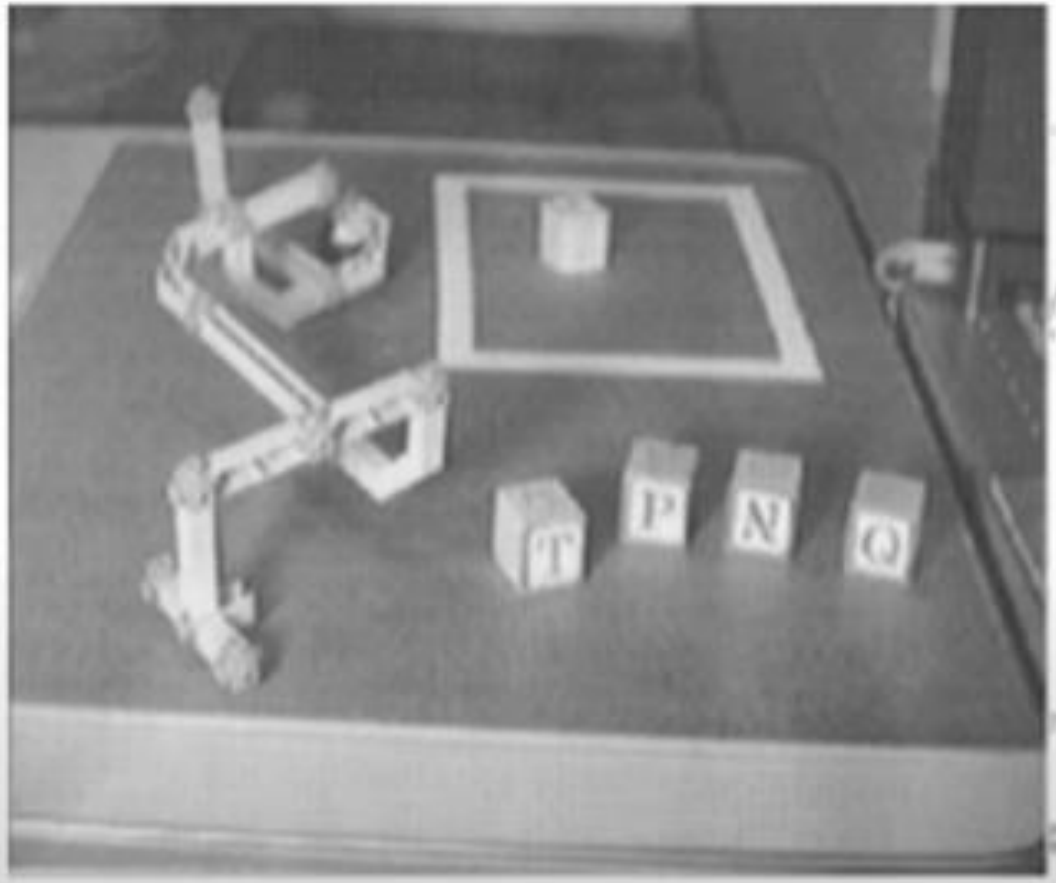
- Texture Analysis: Introduction to texture in images, Statistical texture analysis methods: Gray Level Co-occurrence Matrix (GLCM), Local Binary Patterns (LBP); Filter-based texture analysis methods: Gabor filters, Laws' texture energy measures; Texture-based segmentation.

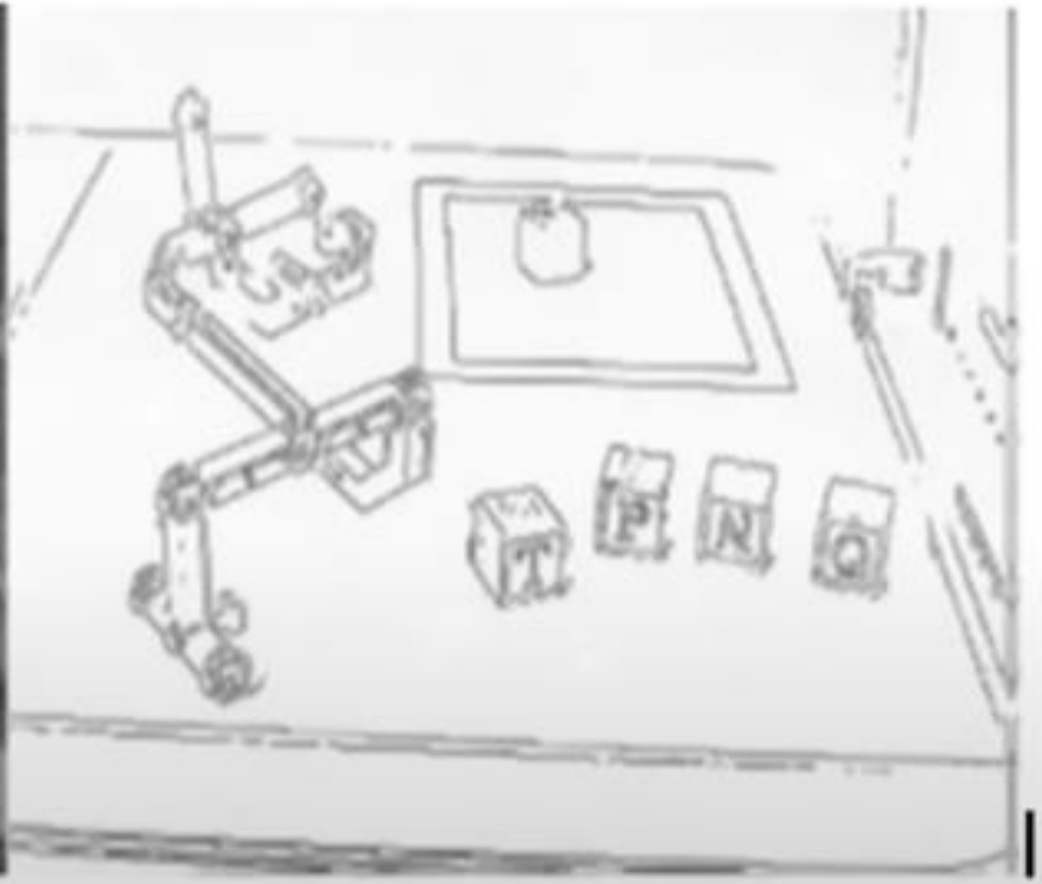
- Motion Analysis: Optical flow estimation, Lucas-Kanade method, Horn-Schunck method, Background subtraction, Dense optical flow using Deep Learning (FlowNet), Motion-based segmentation

EDGE DETECTION- Robert, Prewitt, Sobel

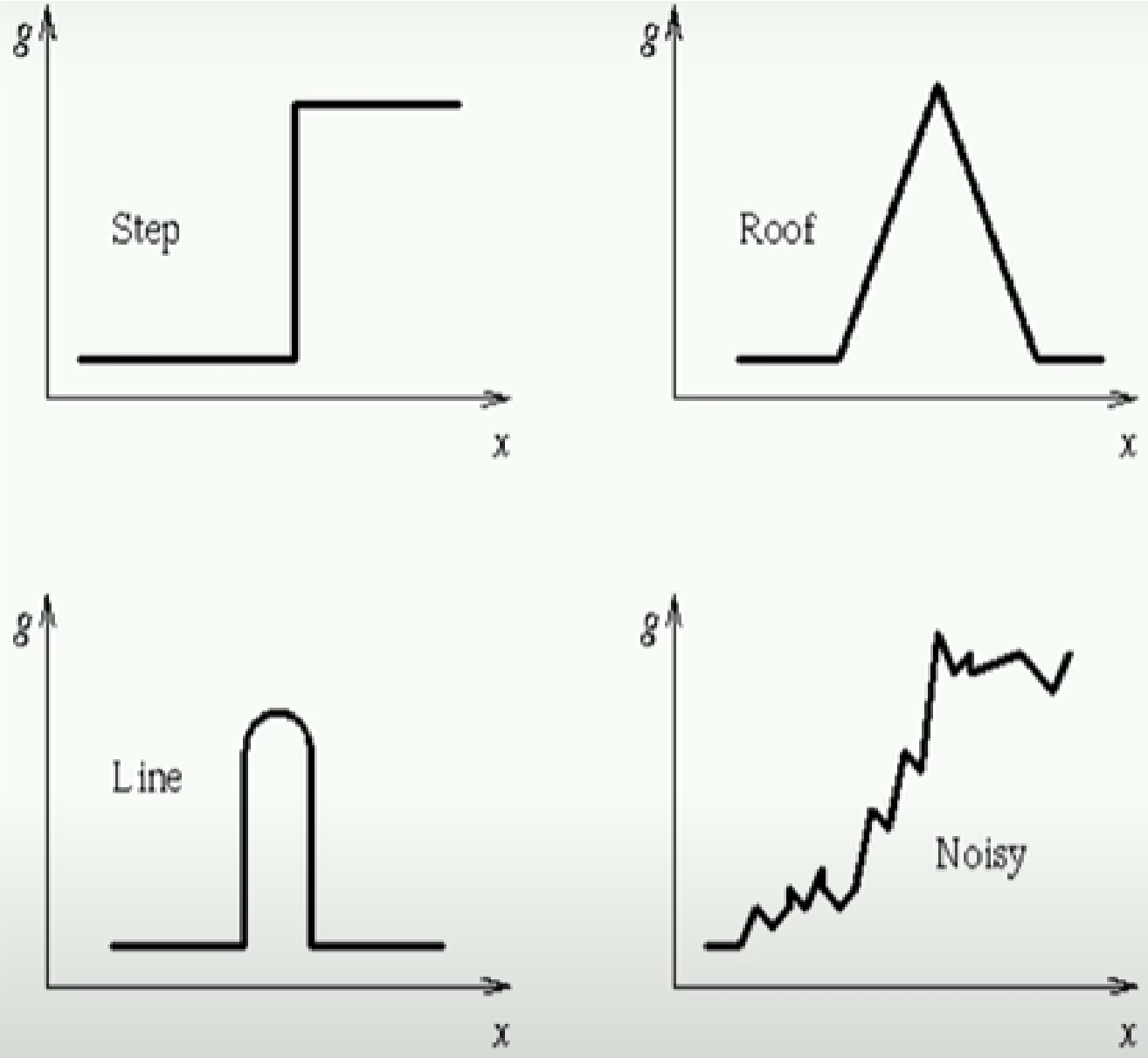
- Edges are those places in an image corresponds to an object boundaries.
- It makes an abrupt change in the intensity of pixels
- And also produces discontinuity in image brightness or contrast
- Edge information is found by looking at the relationship a pixel has with its neighbourhoods.
- If a pixels grey level values is similar to those around it, there is probably not an edge at that point.
- If a pixel has neighbourhood with widely varying gray levels .it may present an edge point.

- Use of Edge Detection – Extracting information about the image. E.g. location of objects present in the image, their shape, size, image sharpening and enhancement





- Types of edge: Generally edges are of
  - Step Edge
  - Ramp Edge
  - Line Edge
  - Roof Edge



Typical edge profiles.

- Steps in Edge Detection

(1) **Smoothing:** suppress as much noise as possible, without destroying true edges.

(2) **Enhancement:** apply differentiation to enhance the quality of edges (i.e., sharpening)

(3) **Thresholding:** determine which edge pixels should be discarded as noise and which should be retained (i.e., threshold edge magnitude).

(4) **Localization:** determine the exact edge location.

- METHODS OF EDGE DETECTION

- First Order Derivative / Gradient Methods
  - Roberts Operator
  - Sobel Operator
  - Prewitt Operator
- Second Order Derivative
  - Laplacian
  - Laplacian of Gaussian
  - Difference of Gaussian

- An image gradient is a directional change in the intensity or color in an image. It includes both magnitude and direction.
  - Image gradient may be used to extract information from images
  - The magnitude M of the gradient is defined by

$$M=\sqrt{Gx^2} + Gy^2$$

The direction is given by

$$\theta = \tan^{-1}(Gx/ Gy)$$

Sobel Edge Detector

- The 3X3 convolution mask smoothes the image by some amount , hence it is less susceptible to noise. But it produces thicker edges. So edge localization is poor
- Convolution Mask

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

Gx

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

Gy

- The mask can be applied separately to image to produce separate measurement of the gradient component in each orientation .These can be combined together to find the absolute magnitude of the gradient at each point and orientation of that gradient

- Figure Shows a 3 * 3 region of an image

Z1	Z2	Z3
Z4	Z5	Z6
Z7	Z8	Z9

- Sobel Operator

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

Gx

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

Gy

$$Gx = (Z3+2Z6+Z9) - (Z1+2Z4+Z7)$$

$$Gy = (Z1+2Z2+Z3) - (Z7+2Z8+Z9)$$

$$M = \sqrt{Gx^2 + Gy^2}$$

•Sobel operator goes for averaging and emphasizes on the pixel closer to the center of the mask. It is less affected by noise and is one of the most popular Edge Detectors

PREWITT OPERATOR

- It is similar to the Sobel operator but uses slightly different masks
- Convolution Mask

Gx=

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

Gy=

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

- Figure Shows a 3 * 3 region of an image

Z1	Z2	Z3
Z4	Z5	Z6
Z7	Z8	Z9

Gx= (Z3+Z6+Z9) –(Z1+Z4+Z7)

Gy=(Z1+Z2+Z3)-(Z7+Z8+Z9)

$M=\sqrt{Gx^2 + Gy^2}$

Canny Edge Detector

- To understand the canny edge detector, the reader has to go through the following sequence.
- Canny is not like other traditional edge detectors, it is just not masking or hovering on the input image matrix.
- Instead, it is detailed and has steps to follow.
- This section shall provide complete input on the same clearly, step by step.

- Canny edge detection technique, not just is a plain edge detection technique. It has an additional feature. It also suppresses the noise while detecting the edges flawlessly.
- Let's go step by step:

1. Conversion to the Grayscale

- Let us take a sample image and proceed with the conversion. We have converted the input RGB image to a Grayscale image. One can refer to the below figure to understand the same.

Input Image:

Grayscale Converted Image:

2. Gaussian Blur

- It is an operator, which helps in removing the noise in the input image. This noise removed image shall enable further processing to be smooth and flawless. The sigma value has to be set appropriately for better results.

3. Intensity Gradient Calculation

- We shall go back to the basics. Sobel filter is to be used in this process. Let's understand what an edge is all about. Sudden intensity change is the edge and in fact, the intensity change of the pixel is the edge.
- Now the Sobel operator has to applied over the input image and the steps and sequences remain the same as the process explained in the Sobel Edge Detection process. The resultant Sobel operated image is presented below. The is referred as Gradient Magnitude of the image.

- We preferred sobel operator and it is the general approach. But, it is not a mandated rule to always go with sobel operator. It can be any gradient operator and the result should be the gradient magnitude of the image.
- The resulting Gradient Approximation can be calculated with

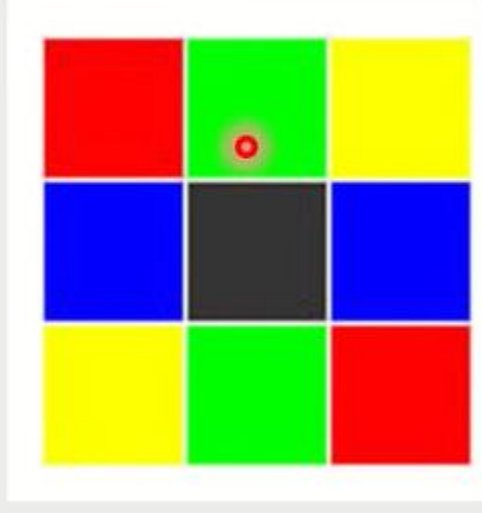
$$G = \sqrt{Gx^2 + Gy^2}$$

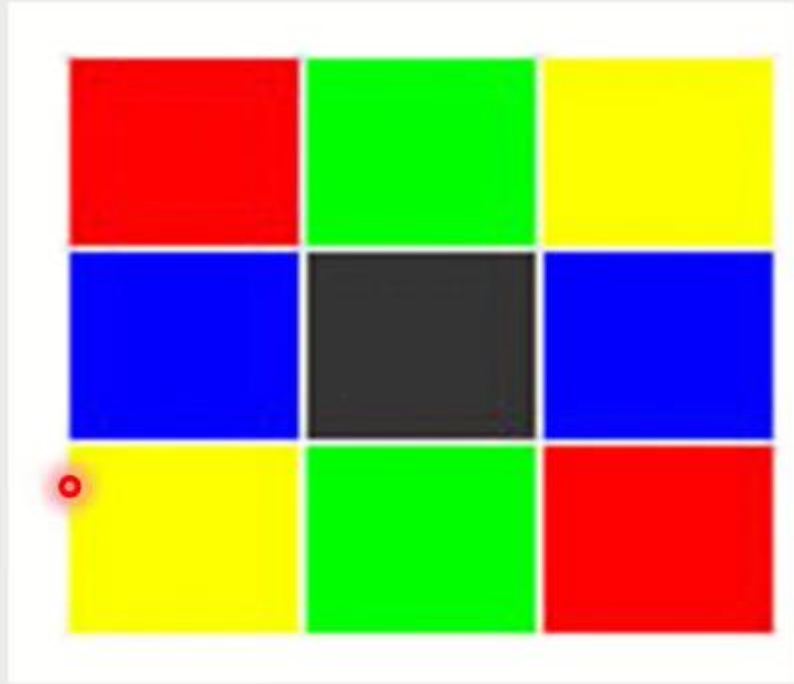
- The G will be compared against the threshold and with which, one can understand the taken point is an edge or not.
- The formula for finding the edge direction is just: Theta = inv tan (Gy / Gx)

4. Non Maximum Suppression – This is the next step in the sequence. The gradient magnitude operators discussed in the previous stage normally obtains thick edges. But, the final image is expected to have thin edges.

Hence, the process Non Maximum Suppression shall enable us to derive thing edges from thicker one through the following steps:

- We have the edge direction already available with us. The subsequent step is to relate the identified edge direction to a DIRECTION that can be sketched in the image. i.e. ideally, it is a prediction of how the movement of edges could happen.
- An example is always handy and we have taken 3 * 3 matrix as a reference. It's all about the colors and the below is to be visualized as 3 * 3 matrix for the scenario being discussed.





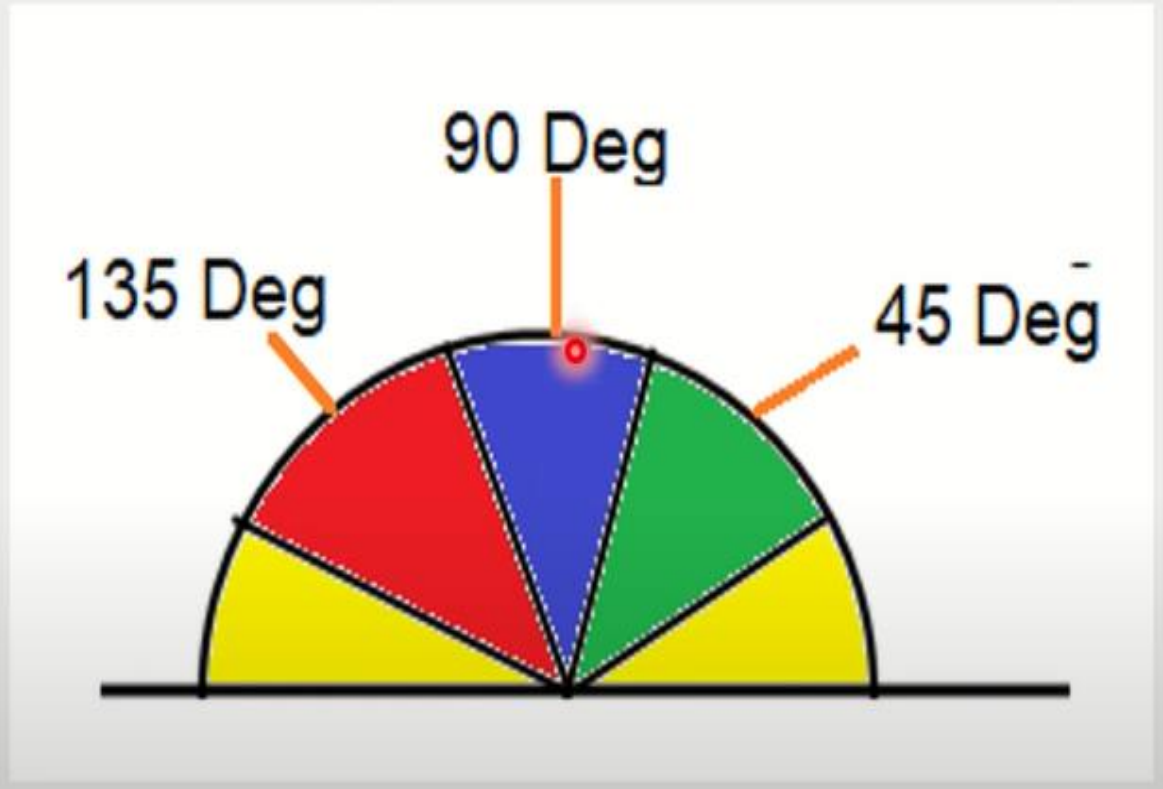
The possible directions could be of movement could be,

North to south:

East to west:

Both the diagonals:

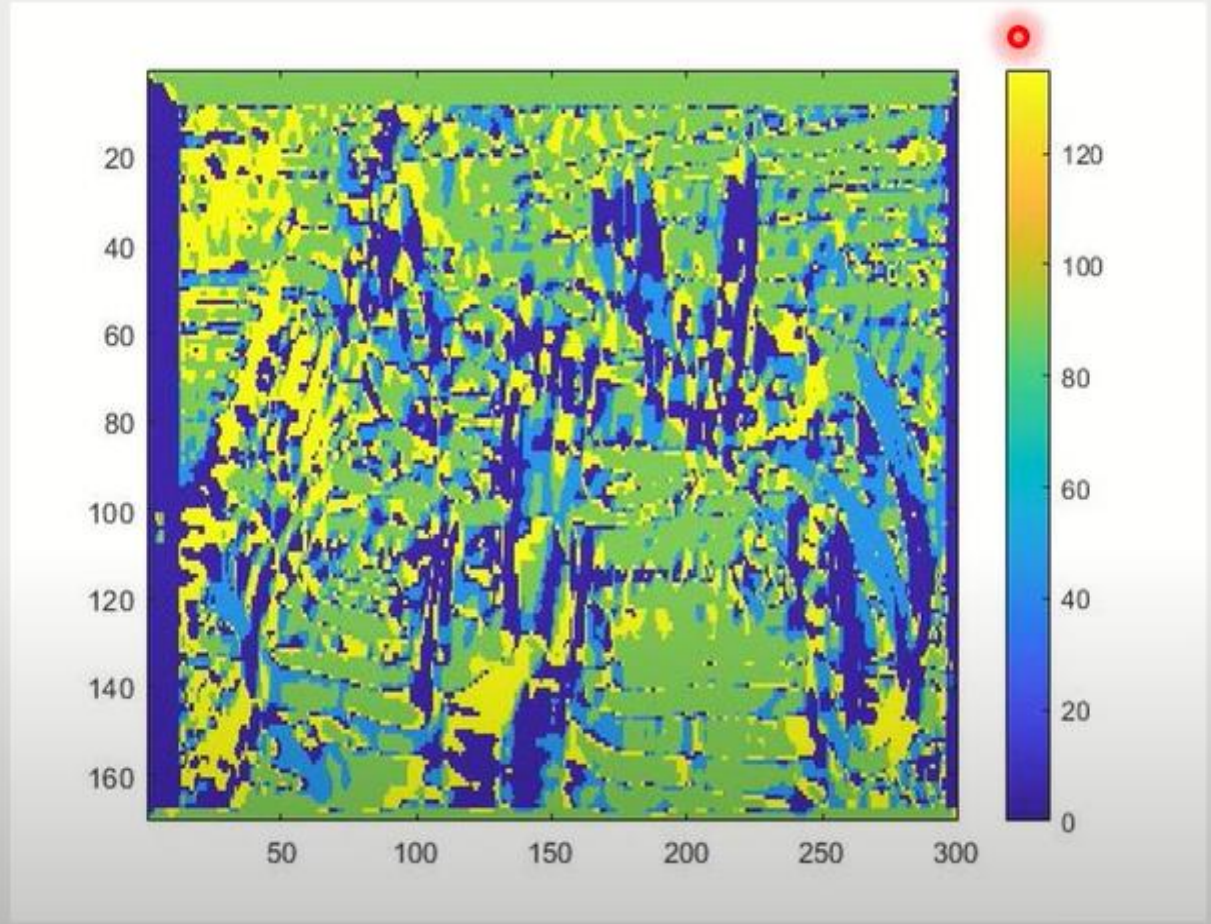
- The Centre cell is the region of interest for us. It is important to understand this point explained below.
- There can be only 4 possible directions for any pixel. They are
  - *0 Degrees*
  - *45 Degrees*
  - *90 Degrees and*
  - *135 Degrees*
- Hence, it forces us to a situation where the edge has to be definitely oriented to one of these four directions.
- This is kind of approximation where if the orientation angle is observed to be 5 degrees then it is taken as 0 degrees. Similarly, if it is 43 degrees, it shall be made as 45 degrees.
- For ease of understanding we have drawn a semicircle with color shading. It is representing 180 degrees. (But, actual scenario is for 360 degrees.)



With the above picture as reference the following rules are to be framed:

1. Any edge which come under the yellow range is set to 0 degrees. (Which means from 0 to 22.5 degree and 157.5 to 180 degrees are set to 0 degrees)
2. Any edge which come under the green range is all set to 45 degrees. (Which means 22.5 degrees to 67.5 degrees is set as 45 degrees)
3. Any edge coming under the blue range is all set to 90 degrees. (Which means 67.5 degrees to 112.5 degrees is set as 45 degrees)
4. The last remains easy to understand. Any edge coming under the red range is all set to 135 degrees. (Which means 112.5 degrees to 157.5 degrees is set as 135 degrees)

- After this process, direction of the edges is mapped to any of the 4 directions mentioned above. The input image now shall be like the one presented below where directions of the edges are appropriately mapped.



- The last step in the process comes now.
- The edge directions are all determined and the non-maximum suppression is to be applied.
- Non maximum suppression as the name suggests, is a process where suppression of the pixels to zero which cannot be considered as an edge is carried out. This shall enable the system to generate a thin line in the output image as shown below.
- These results are obtained before the thresholding and as expected the next stage to perform thresholding and smoothing.

5. Thresholding – A must to do process

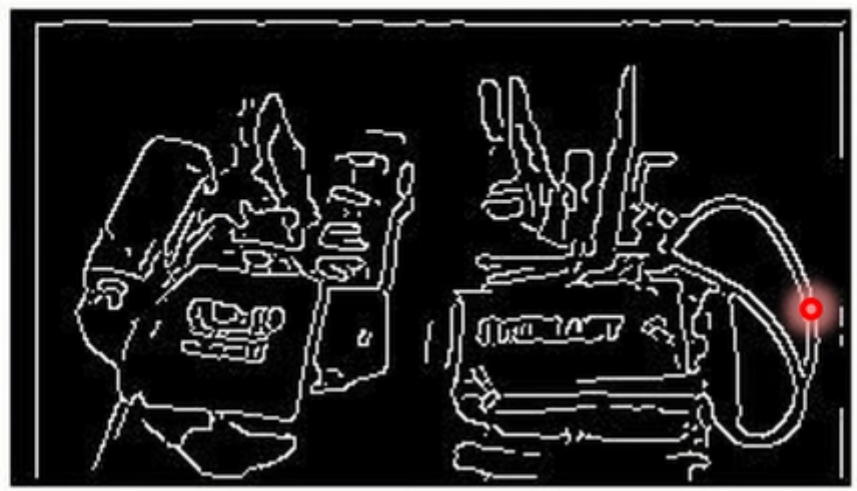
- As one could see from the previous stage results, Non maximum thresholding has not provided us excellent results.
  - *There is still some noise. The image even raises a threat in mind that some of the edges shown may not really be so and some edges could be missed in the process. Hence, there has to be a process to address this challenge. The process to be followed is thresholding.*
- We need to go with double thresholding and in this process we have to set two thresholds. One, a high and another a low. What is this?
- Simple assume high threshold value as 0.8. Any pixel with a value above 0.8 is to be seen as a stronger edge.
  - *Another threshold, the lower one can be 0.2. In that case, any pixel below this value is not an edge at all and hence set them all to 0.*
- Now comes the next question, what about the values in between?
  - *I mean from 0.2 to 0.8? They may or may not be an edge. They are referred as a weak edge. There has to be a process to determine which of the weak edges are actual edges so as to not to miss them.*

6. Edge Tracking

- As discussed in the previous section, it is important for us to now understand which of the weaker edges are actual edges.
- Simple approach has to be followed. We can call the weak edges connected to strong edges as strong/actual edges and retain them. Weak edges which are not connected to stronger ones are to be removed.

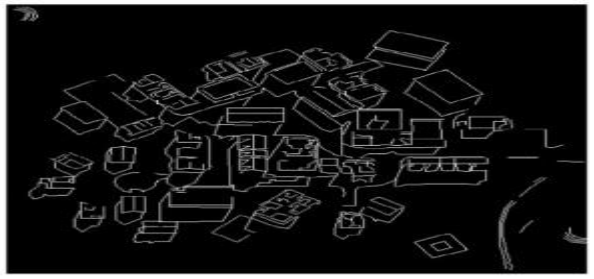
7. The final cleansing

- All the remaining weak edges can be removed and that is it. The process is complete. Once this process is done, we could get the following output image as the result.

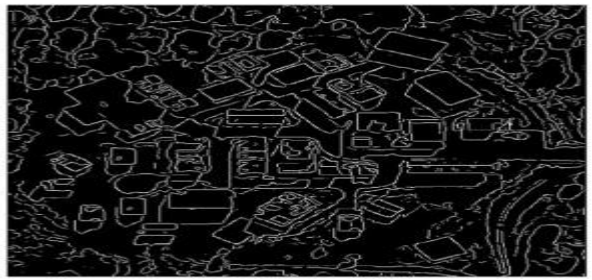


Optimum Edge Detection

(a)



(b)



(c)

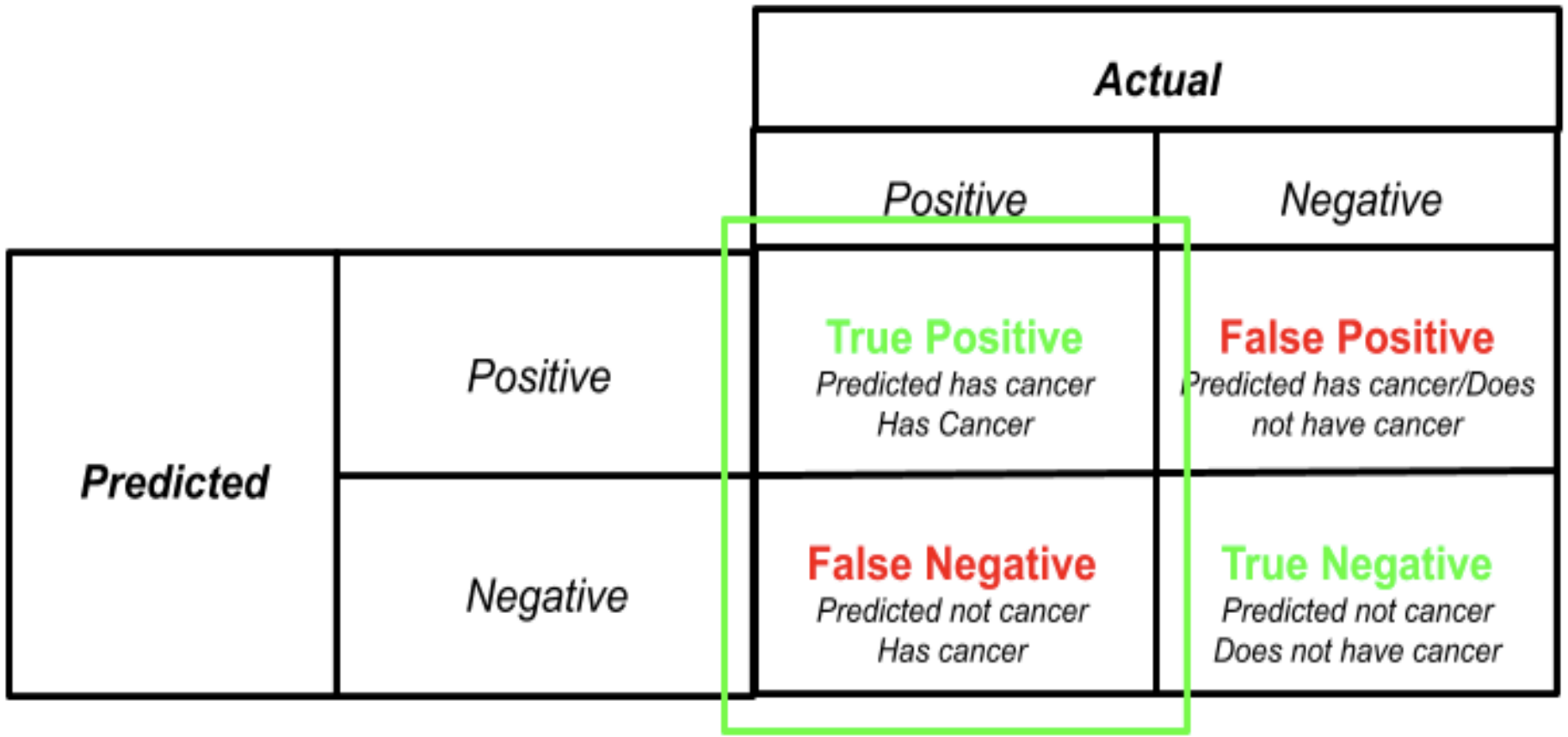
(d)

Optimum Edge Detection

- Optimum edge detection refers to the process of detecting edges or boundaries between objects or regions in an image with the goal of achieving the best possible results.
- The term "optimum" suggests that the edge detection technique is designed to produce accurate and meaningful edge information while minimizing errors, noise

key concepts and goals of optimum edge detection

- **Edge Detection Purpose**: The primary purpose of edge detection in image processing is to enhance the visibility of object boundaries or transitions in an image. These boundaries are often areas of rapid intensity change, where pixel values transition from one region or object to another.
- **Sharpness and Accuracy**: Optimum edge detection aims to detect edges with sharpness and accuracy. This means that the detected edges should align closely with the true boundaries in the image, providing a clear representation of object shapes and locations.
- **Minimizing Noise**: Image data can be noisy, containing random fluctuations in pixel values. An optimum edge detection algorithm should minimize the impact of noise, ensuring that detected edges are not heavily influenced by these fluctuations.
- **Reducing False Positives**: False positives occur when an edge is detected where there is no actual object boundary in the image. Optimum edge detection algorithms should minimize the occurrence of false positives, as these can lead to incorrect interpretations of the image.


Actual
Positive
Negative
Predicted
Positive
Negative
True Positive
Predicted has cancer
Has Cancer
False Positive
Predicted has cancer/Does not have cancer
False Negative
Predicted not cancer
Has cancer
True Negative
Predicted not cancer
Does not have cancer

key concepts and goals of optimum edge detection

- **Orientation and Thickness**: In many cases, edge detection is not just about finding the presence of an edge but also determining its orientation and thickness. Optimum edge detection techniques may provide information about the direction of the edge (e.g., horizontal, vertical, diagonal) and the width of the edge region.

- **Edge Connectivity**: Optimum edge detection considers the connectivity of edges. This means that detected edges should form continuous curves or contours that outline objects in the image. Disconnected or fragmented edges are less desirable.

- **Adaptability**: Different images and applications may require different edge detection approaches. An optimum edge detection algorithm may be adaptable and tunable to perform well across a variety of image types and conditions.

Thresholding techniques

What's it?

- As discussed in brief, segmentation algorithms based on thresholding approach are suitable for images where there is distinct difference between object and background.
- The goal of thresholding based segmentation algorithms is to divide an image into two distinct regions (object and background) directly based on intensity values and/or properties of these values.
- This is viewed as one of the simplest techniques for the image segmentation. Let us understand things in detail.

Contd.,

- There is an important component called threshold involved and based on the threshold T, one could go ahead with three types of thresholding based segmentation algorithms and they are:
  - Segmentation algorithm based on global threshold
  - Segmentation algorithm based on variable threshold and
  - Segmentation algorithm based on multiple threshold.

Segmentation algorithm based on global threshold

$$g(x,y) = \begin{cases} 1 \; if \; f(x,y) > T \\ 0 \; if \; f(x,y) \leq T \end{cases}$$

Where,

- $g\,(x,y)$ represents the output image (segmented image);
- $f\,(x,y)$ represents the input image and T represents threshold.

- When the intensity distribution of object and background pixels in an image are sufficiently distinct, segmentation can be done by using a single global threshold over the entire image.
- Consider an image $f\,(x,y)$ with light objects on a dark background. To extract objects from the image, select a threshold T.
- A pixel in an image is considered as part of the object, if the pixel intensity is greater than threshold T.
- Similarly, if the pixel intensity is less than or equal to threshold T, that pixel is considered as the background component.

Segmentation algorithm based on variable threshold

- Here, as the name suggests, the value for the T is a variable.
- That is, the value of T can vary over the span of entire image.
- There are two types, Local threshold and the next one is adaptive threshold.
- The local threshold T's value is totally dependent on the neighbourhood of the X and Y.
- Whereas, in the adaptive thresholding, the value of T is a function of X and Y.

Segmentation algorithm based on multiple threshold

- Multiple thresholding is used when there are two light objects on a dark background in an image. Consider an image $f\ (x, y)$ with two light objects: object 1 with intensity a and object 2 with intensity b and dark background with intensity c.
- To extract these two objects, consider two thresholds T_1 and T_2.
- Let T_1 denote threshold for object 1 and let T_2 denote threshold for object 2. A pixel is said to belong to object 1 if its intensity is greater than T_1 and less than or equal to T_2.
- A pixel is said to belong to object 2 if its intensity is greater than T_2 . A pixel is said to belong to background if its intensity is less than T_1.

Contd.,

$$g(x,y) = \begin{cases} a & \text{if } T_1 < f(x,y) \leq T_2 \\ b & \text{if } f(x,y) > T_2 \\ c & \text{if } f(x,y) \leq T_1 \end{cases}$$

Here $g\ (x,y)$ denotes segmented image ;

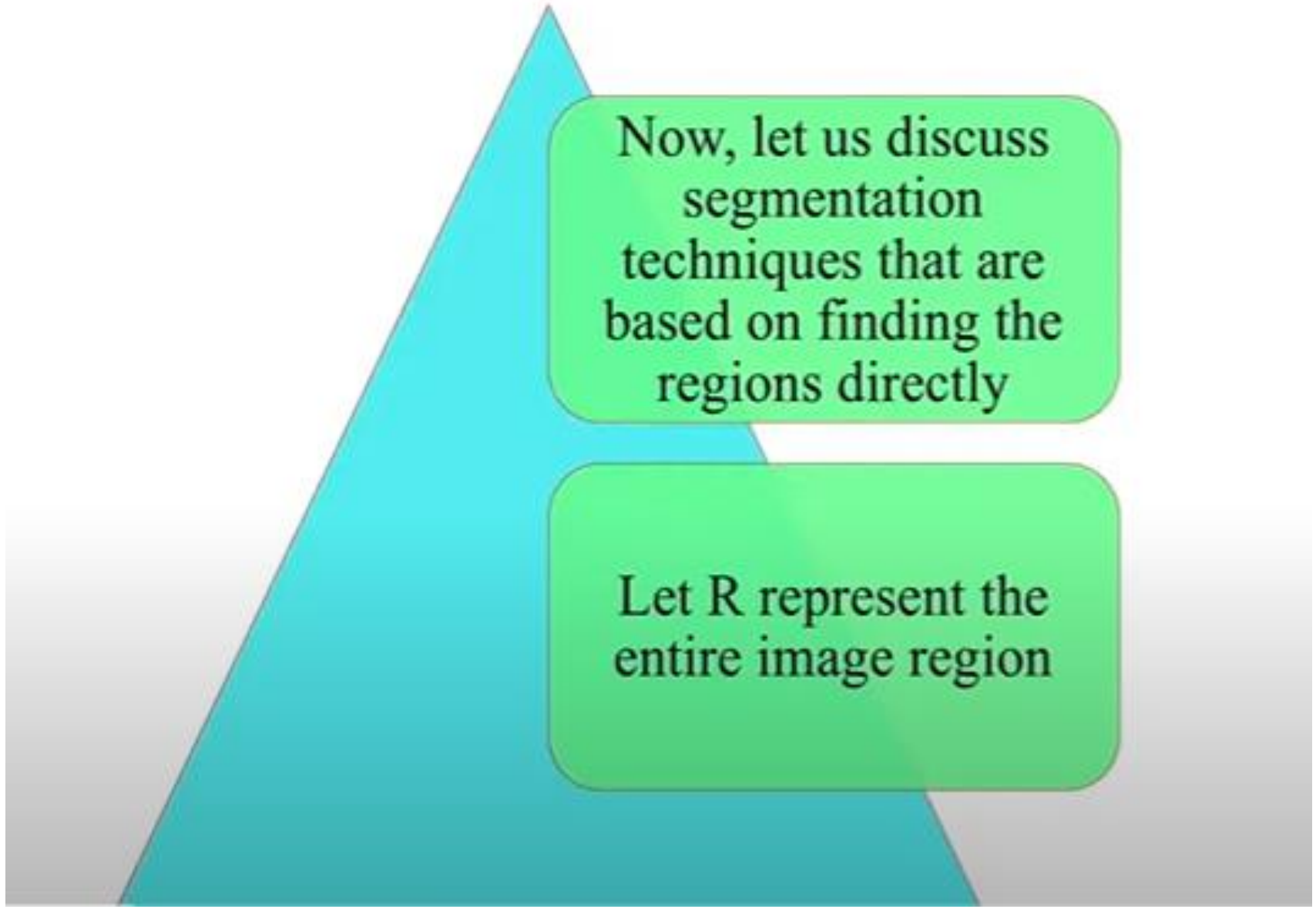
a denotes intensities of object 1; b denotes intensities of object 2 ; c denotes intensity of background. T1 denotes threshold for object 1 and T2 denotes threshold for object 2.

Topic Name : Region Based Segmentation

The objective of segmentation is to partition an image into regions

We have approached this problem by finding boundaries between regions based on discontinuities in gray levels

Also it is accomplished via thresholds based on the distribution of pixel properties, such as gray-level values or color


Now, let us discuss segmentation techniques that are based on finding the regions directly
Let R represent the entire image region

- We may view segmentation as a process that partitions R into n sub-regions, R1, R2, . . . , Rn such that
  - *a.* $\bigcup_{i=1}^{n} R_i$ =R
  - *b.* R_i is a connected region, I = 1,2,…,n
  - *c.* $R_i \cap R_j = \emptyset$ for all j,i ≠ j
  - d. P(R_i) = TRUE for i = 1,2,…,n
  - e. P($R_i \cap R_j$) = FALSE for i ≠ j
- Here, P(R_i) is a logical predicate defined over the points in set R_i and φ is the null set

Condition (a) indicates that the segmentation must be complete; i.e. every pixel must be in a region

Condition (b) requires that points in a region must be connected in some predefined sense

Condition (c) indicates that the regions must be disjoint

- Condition (d) deals with the properties that must be satisfied by the pixels in a segmented region
- For example P(R_i) = TRUE if all pixels in R_i have the same gray level
- Finally, condition (e) indicates that regions R_i and R_j are different in the sense of predicate P

Region Growing

Region growing is a procedure that groups pixels or subregions into larger regions based on predefined criteria

The basic approach

- To start with a set of "seed" points and from these grow regions by appending to each seed those neighboring pixie's that have properties similar to the seed

Region Growing

Selecting a set of one or more starting points often can be based on the nature of the problem

When a priori information is not available, the procedure is to compute at every pixel the same set of properties that ultimately will be used to assign pixels to regions during the growing process

If the result of these computations shows clusters of values, the pixels whose properties place them near the centroid of these clusters can be used as seeds

Region Growing

The selection of similarity criteria depends not only on the problem under consideration, but also on the type of image data available

For example, the analysis of land-use satellite imagery depends heavily on the use of color

When the images are monochrome, region analysis must be carried out with a set of descriptors based on gray levels and spatial properties (such as moments or texture)

Region Growing

- Descriptors alone can yield misleading results if connectivity or adjacency information is not used in the region-growing process
- For example, visualize a random arrangement of pixels with only three distinct gray-level values
- Grouping pixels with the same gray level to form a "region" without paying attention to connectivity would yield a segmentation result that is meaningless

Region Growing

Another problem in region growing is the formulation of a stopping rule

Basically, growing a region should stop when no more pixels satisfy the criteria for inclusion in that region

Criteria such as gray level, texture, and color, are local in nature and do not take into account the ‘history’ of region growth

Region Growing

Additional criteria that increase the power of a region-growing algorithm utilize the concept of size, likeness between a candidate pixel and the pixels grown so far and the shape of the region being grown

Region Growing : Example

Figure (a) shows an X-ray image of a weld (the horizontal dark region) containing several cracks and porosities (the bright, white streaks running horizontally through the middle of the image)

(a) Image showing defective welds

Region Growing : Example

We wish to use region growing to segment the regions of the weld failures

- Determine the initial seed points

We selected as starting points all pixels having values of 255

The points thus extracted from the original image are shown in Fig.(b)

Region Growing : Example

(b) Seed points

Region Growing : Example

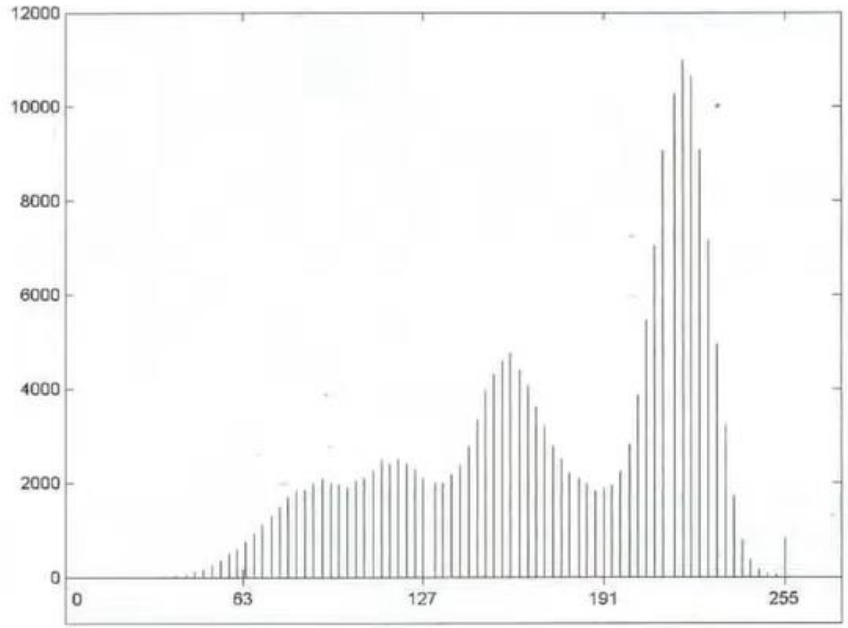
Note that many of the points are clustered into seed regions

- Choose criteria for region growing

We chose two criteria for a pixel to be annexed to a region:

- The absolute gray-level difference between any pixel and the seed had lo be less than 65

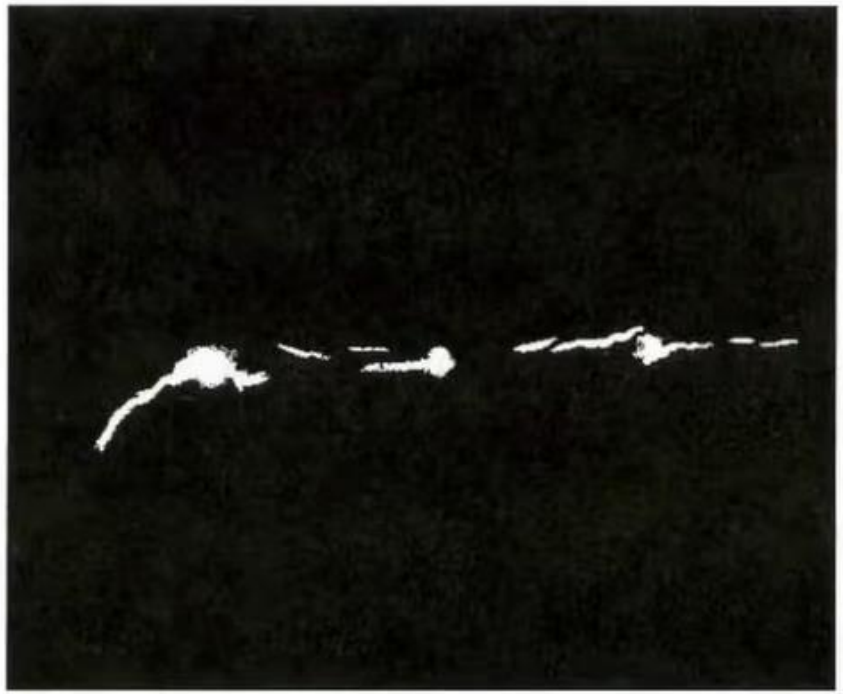
This number is based on the histogram shown in figure



Histogram of previous figure (a)

Region Growing : Example

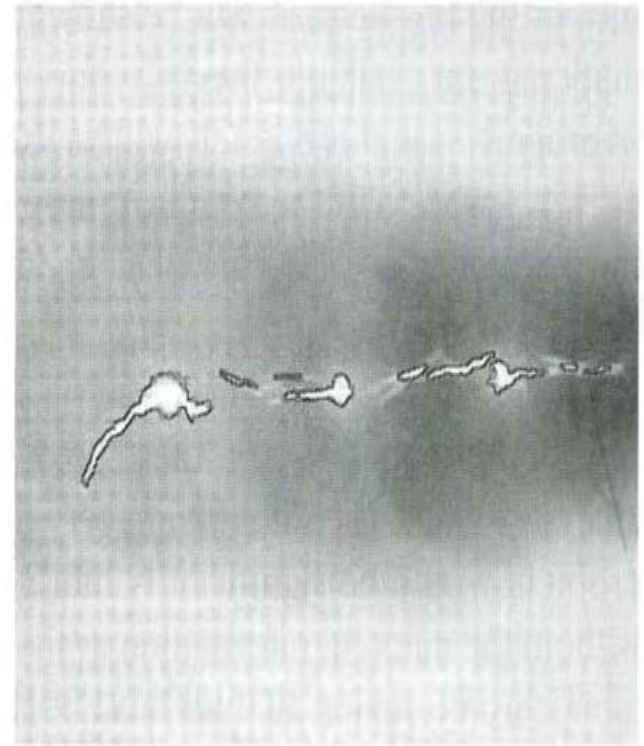
- To be included in one of the regions, the pixel had to be 8-connected to at least one pixel in that region
- If a pixel was found to be connected to more than one region, the regions were merged
- Figure (c) shows the regions that resulted by starting with the seeds in Fig.(b) and utilizing the criteria defined



(c) Result of region growing

Region Growing : Example

Superimposing the boundaries of these regions on the original image, we obtain Fig.(d) which reveals that the region-growing procedure did indeed segment the defective welds with an acceptable degree of accuracy



(d) Boundaries of segmented defective welds (in black)

Region Splitting and Merging

Let R represent the entire image region and select a predicate P

Segmenting R is to subdivide it successively into smaller and smaller quadrant regions so that, for any region R_i, P(R_i) = TRUE

If P(R) = FALSE, we divide the image into quadrants

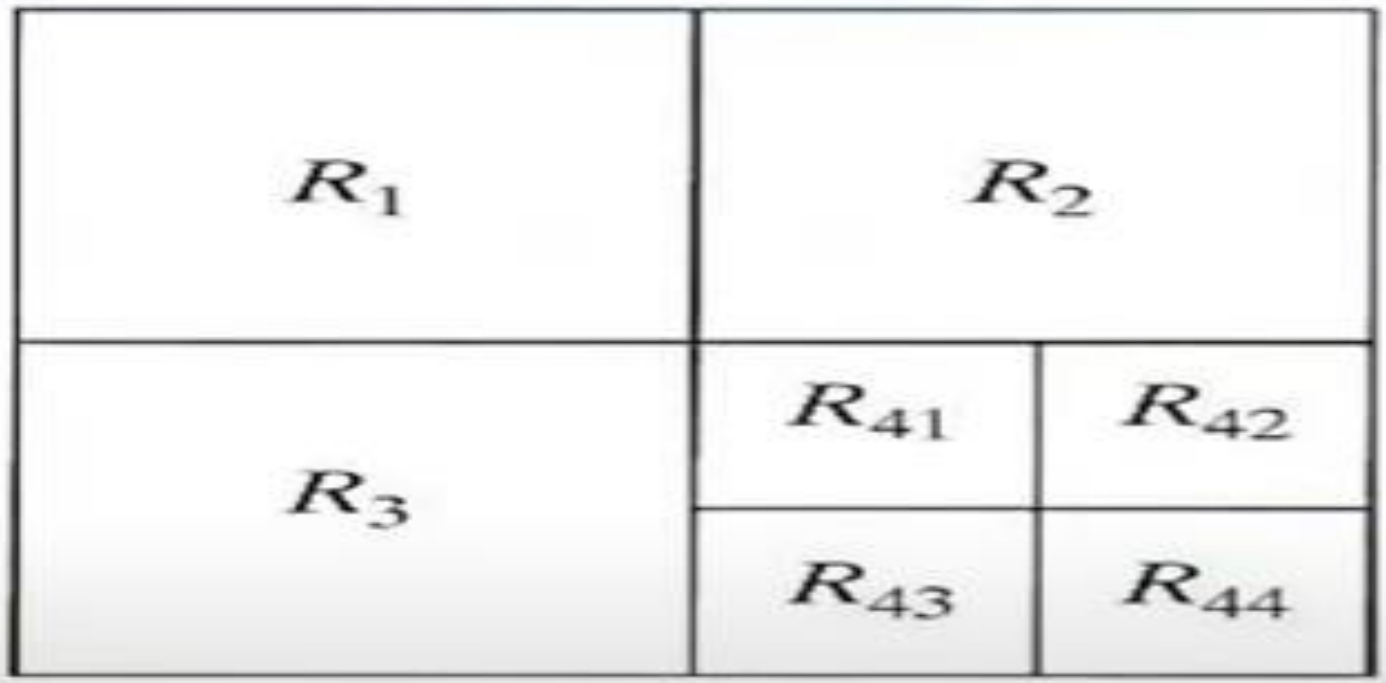
Region Splitting and Merging

If P is FALSE for any quadrant, we subdivide that quadrant into sub quadrants, and so on

This particular splitting technique has a convenient representation in the form of a quadtree as illustrated in figure

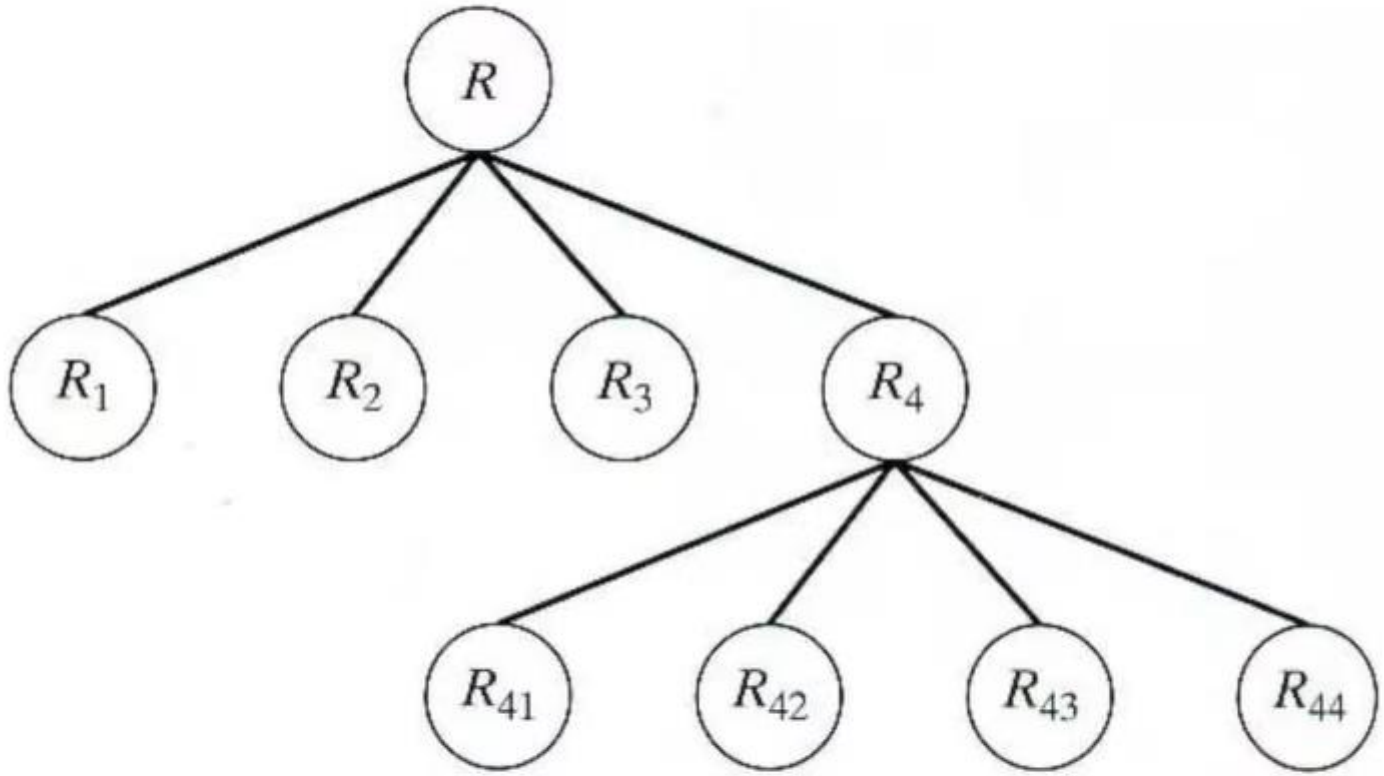
The root of the tree corresponds to the entire image and that each node corresponds to a subdivision

Region Splitting and Merging



(a) Partitioned image

Region Splitting and Merging



(b) Corresponding quad-tree

Region Splitting and Merging

- If only splitting were used, the final partition likely would contain adjacent regions with identical properties
- This drawback may be remedied by allowing merging, as well as splitting
- Two adjacent regions R_j and R_k are merged only if $\mathrm{P}(R_j \cup R_k) =$ TRUE

Region Splitting and Merging

Summary

1. Split into four disjoint quadrants any region R_i for which $P(R_i) =$ FALSE

2. Merge any adjacent regions R_j and R_k for which $P(R_j \cup R_k) =$ TRUE

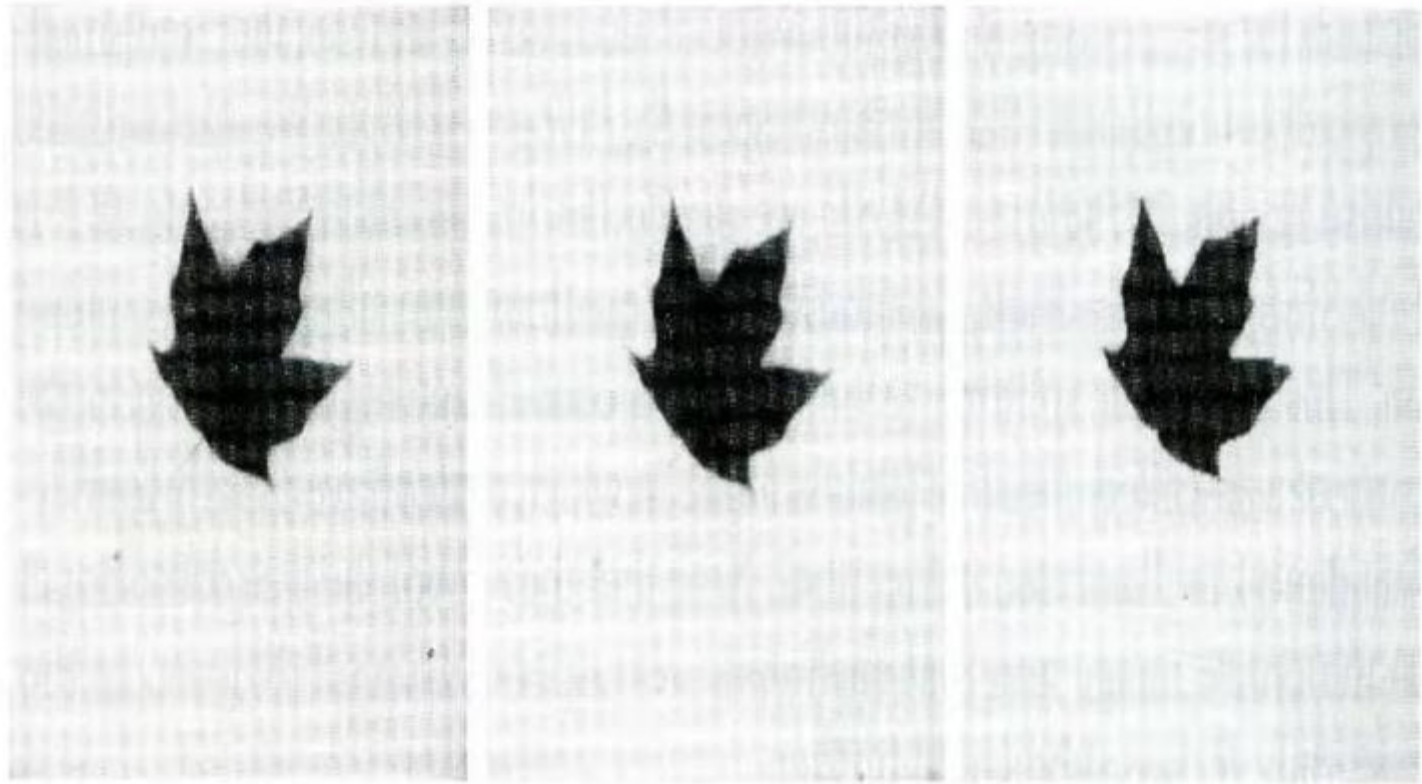
3. Stop when no further merging or splitting is possible

Region Splitting and Merging

The merged regions may be of different sizes

The principal advantage of this approach is that it uses the same quadtree for splitting and merging, until the final merging step

Region Splitting and Merging : Example



(a) Original image
(b) Result of split and merge procedure
(c) Result of thresholding (a)

Region Splitting and Merging : Example

- Figure (a) shows a simple image
- We define P(R_i) = TRUE if at least 80% of the pixels in R_i have the property $|z_j - m_i| \leq 2\sigma_i$, where z_j is the gray level of the j^{th} pixel in R_i, m_i is the mean gray level of that region, and σ_i is the standard deviation of the gray levels in R_i
- Splitting and merging was done using the algorithm outlined

Region Splitting and Merging : Example

The result is shown in Fig.(b)

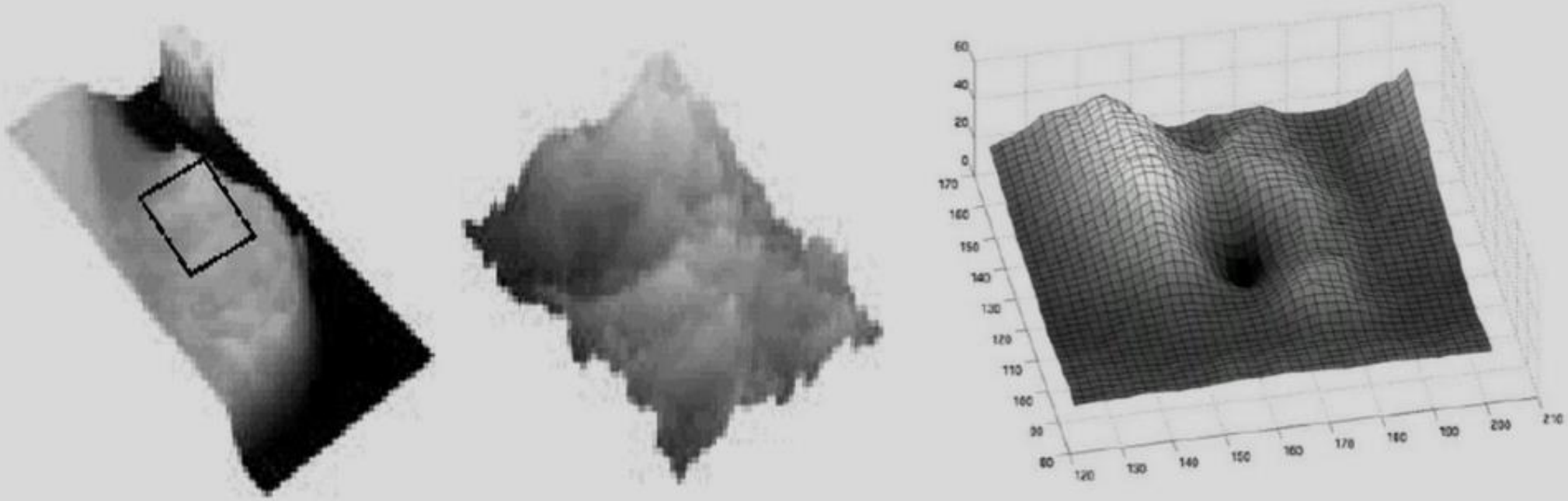
The image shown in Fig.(c) was obtained by thresholding Fig.(a), with a threshold placed midway between the two principal peaks of the histogram

What is Texture?

- Repeating pattern of local variations in image intensity.
- Characterized by the spatial distribution of intensity levels in a neighborhood.
- Cannot be defined for a point.
- A feature used to partition images into regions of interest and to classify those regions.
- Provides information in the spatial arrangement of colours or intensities in an image.

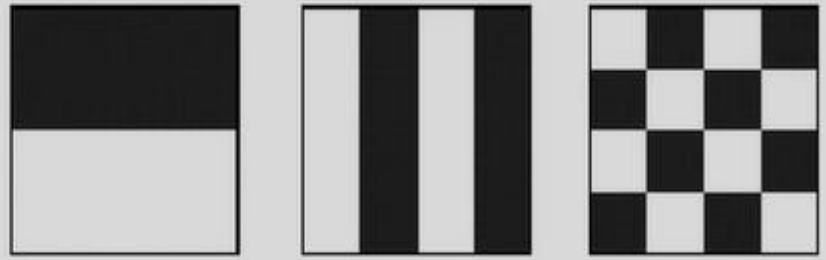
What is Texture?

- Texture is a repeating pattern of local variations in image intensity:



What is Texture?

- For example, an image has a 50% black and 50% white distribution of pixels.



- Three different images with the same intensity distribution, but with different textures.

Texture

Texture consists of texture primitives or texture elements, sometimes called ***texels*** (a group of pixels having homogenous property)

– Texture can be described as fine, coarse, grained, smooth, etc.

– Such features are found in the tone and structure of a texture.

– Tone is based on pixel intensity properties in the ***texel***, while structure represents the spatial relationship between ***texels***.

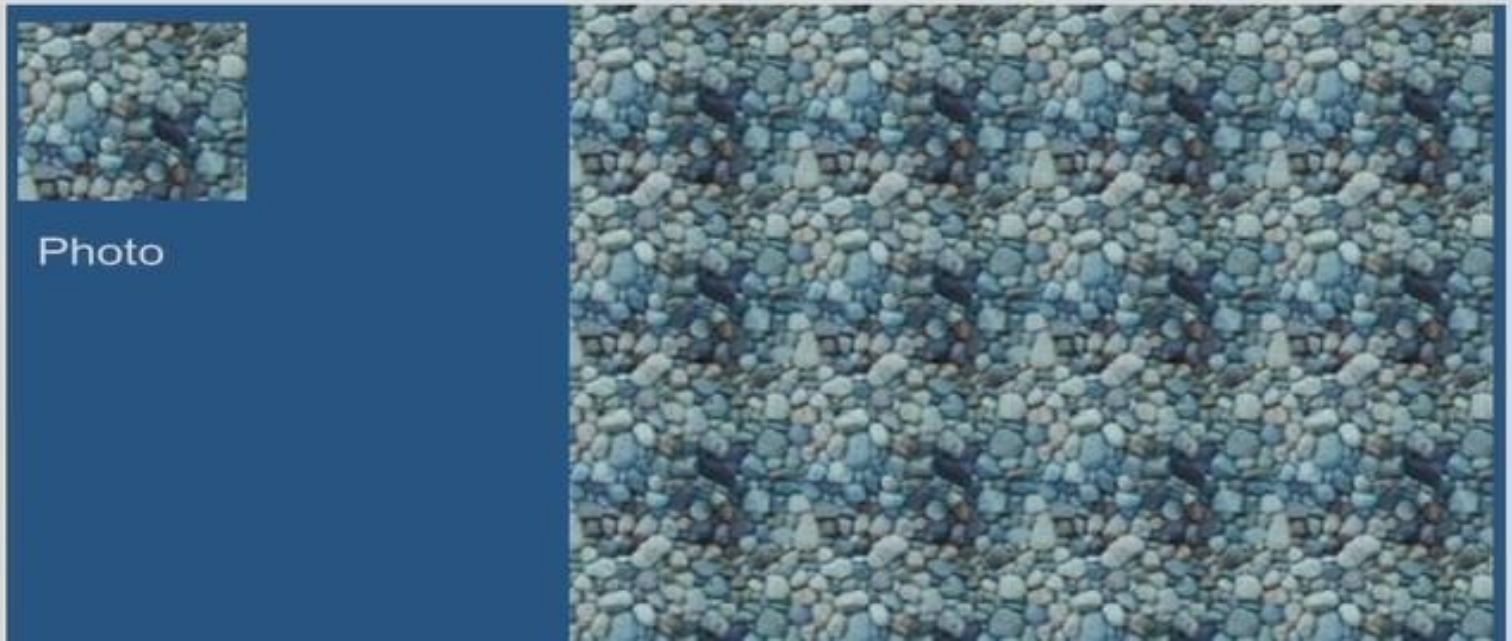
– If ***texels*** are **small** and tonal differences between texels are large a fine texture results.

– If ***texels*** are **large** and consist of several pixels, a coarse texture results.

Texture Analysis

- Four primary issues in texture analysis:

  - ***texture classification***
  - ***texture segmentation***
  - ***texture synthesis***
  - ***shape from texture***



- ***Texture classification*** is concerned with identifying a given textured region from a given set of texture classes.

  Each of these regions has unique texture characteristics.

  Statistical methods are extensively used.

  (*e.g. GLCM, contrast, entropy, homogeneity*)

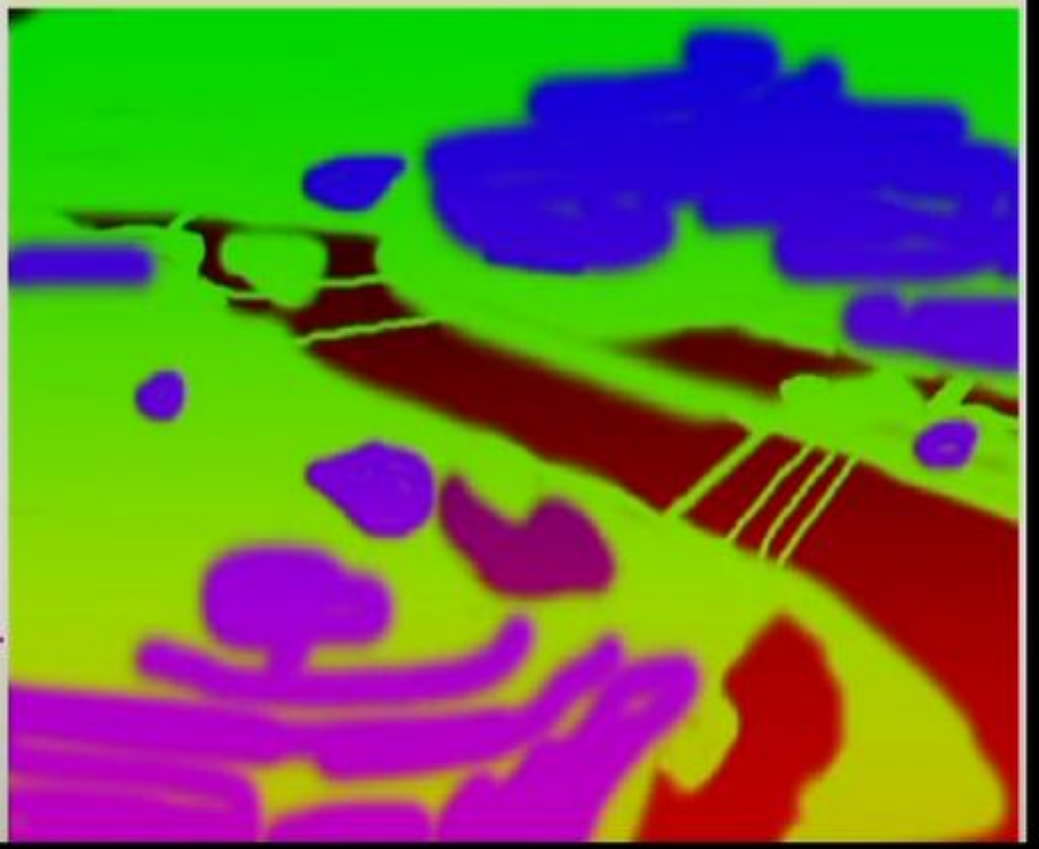
- ***Texture segmentation*** is concerned with automatically determining the boundaries between various texture regions in an image. Partition into different regions where the texture is homogenous.

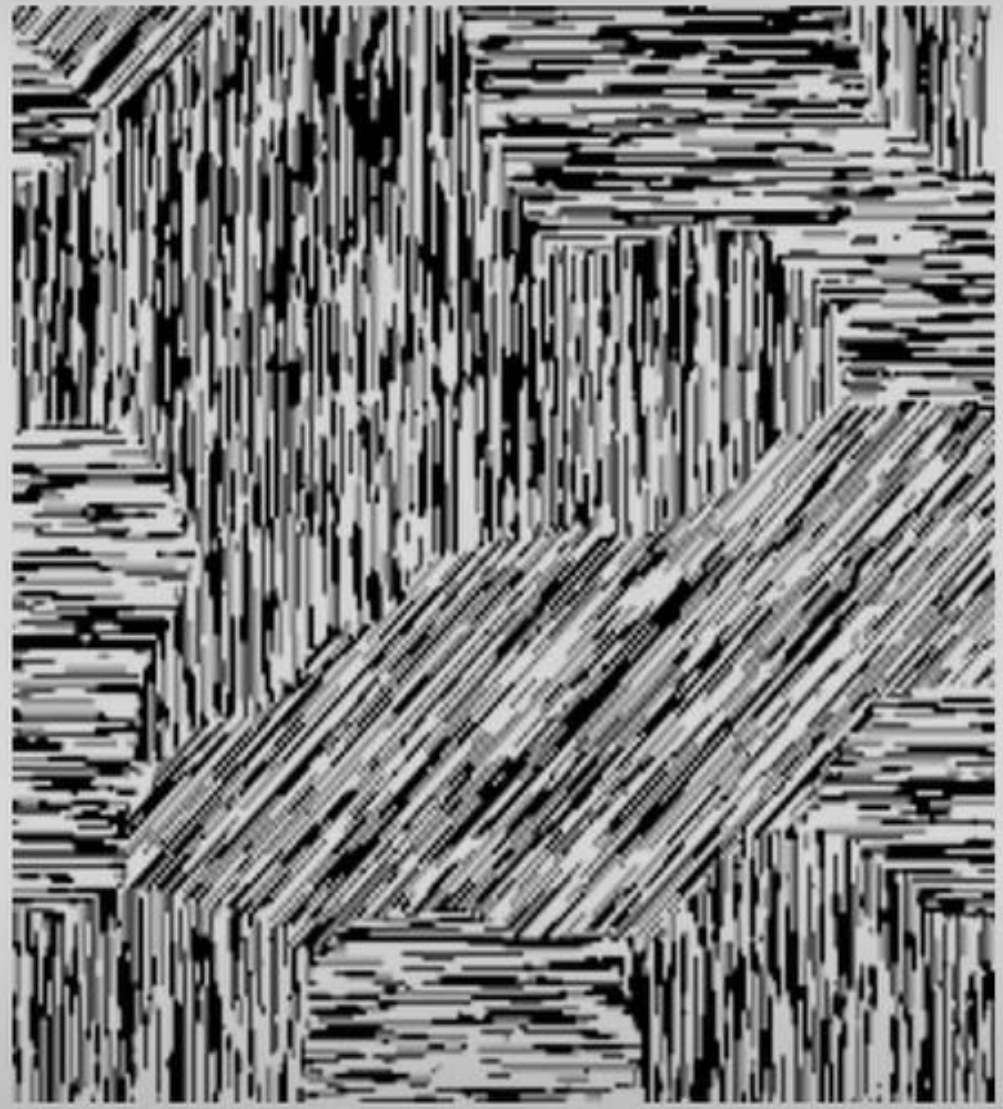
- **Texture synthesis** is the process of algorithmically constructing a large digital image from a small digital sample image by taking advantage of its structural content. Given a finite sample of some textures, the goal is to synthesize other samples from that texture.

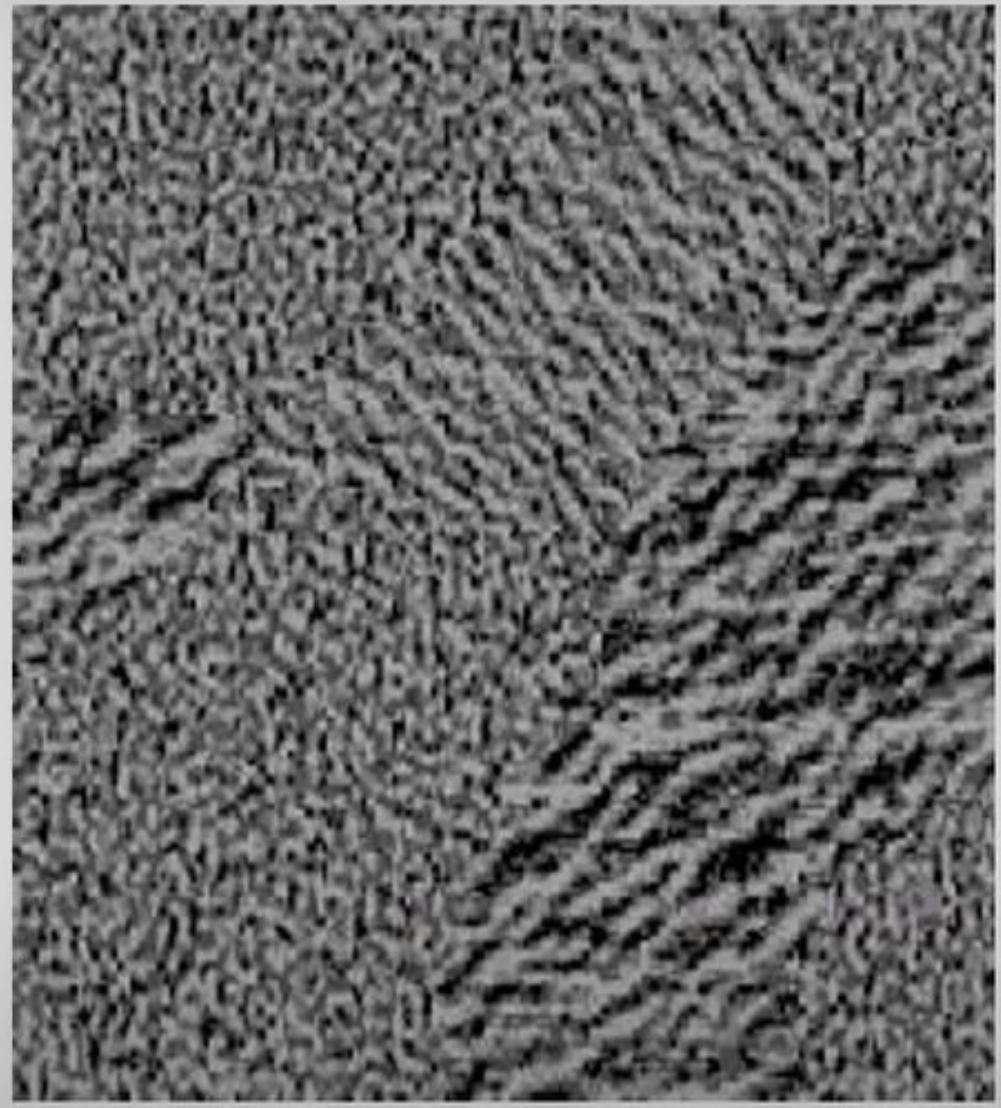
- **Shape from texture:** Texture pattern variations give cue to estimate shape of a surface.

input

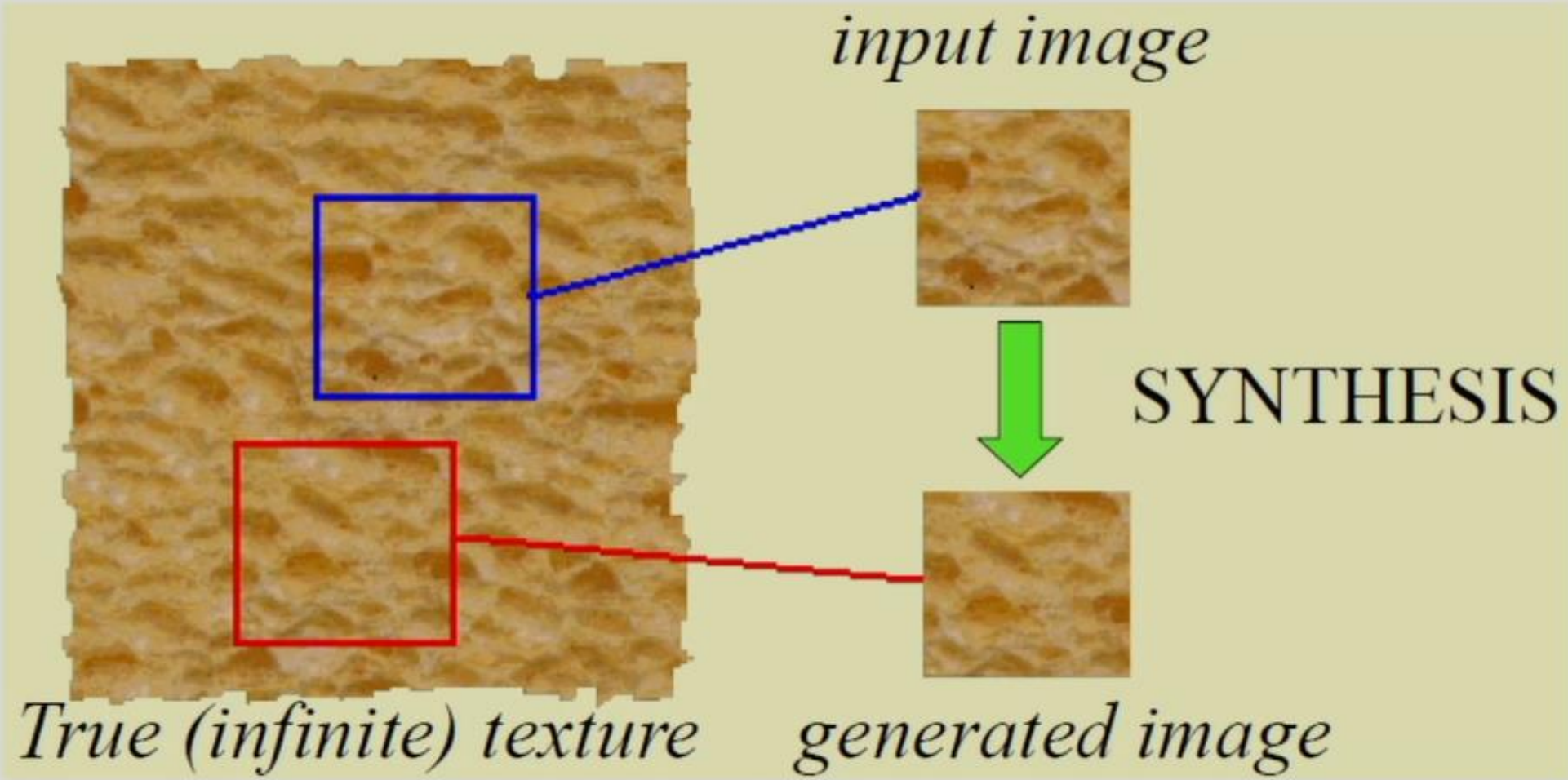
texture segmentation



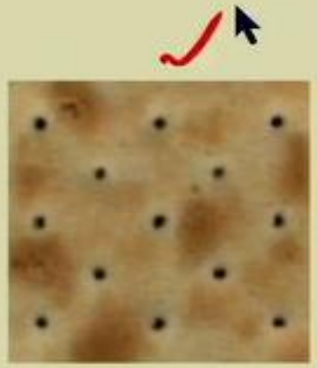


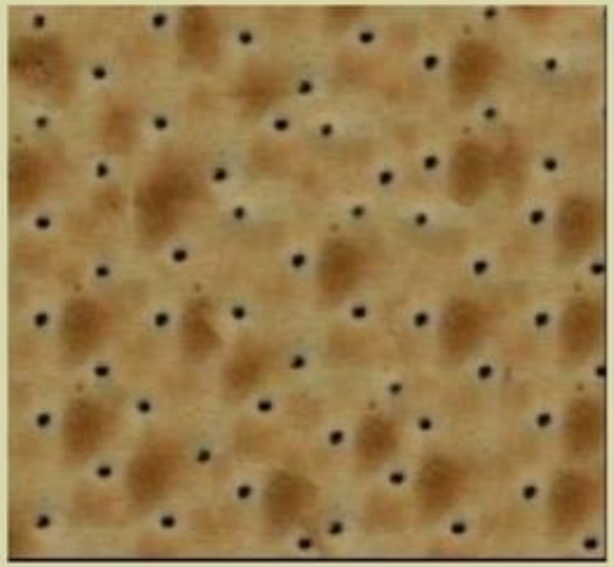


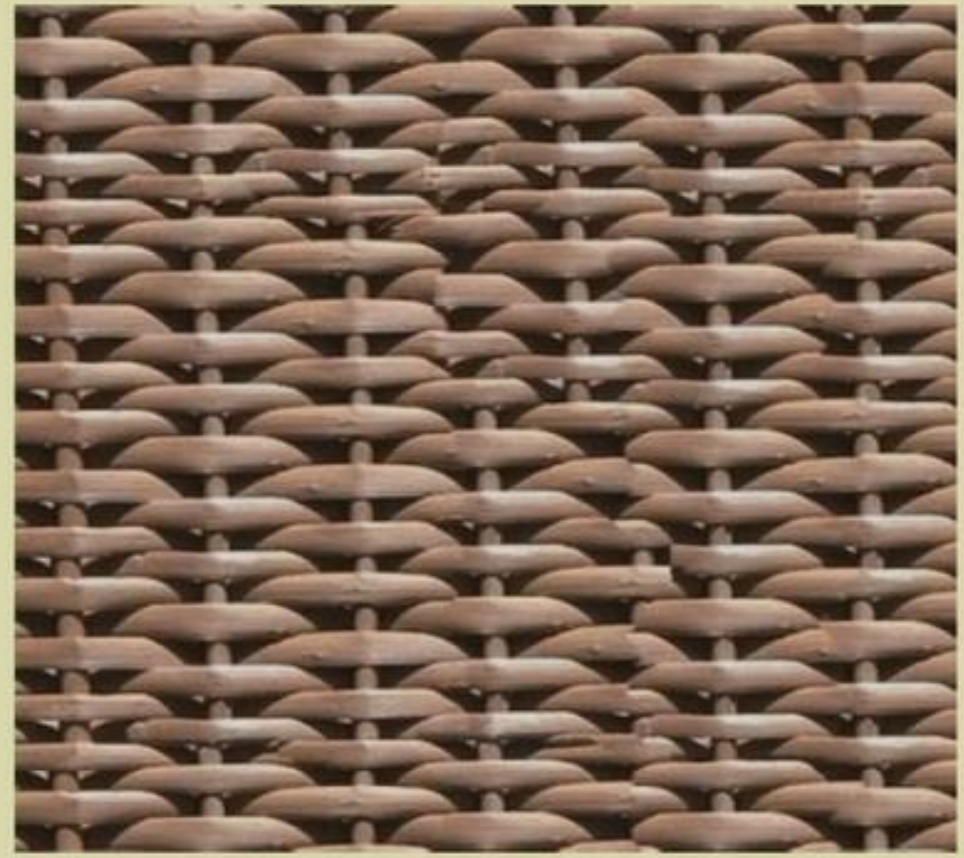
The Goal of Texture Synthesis

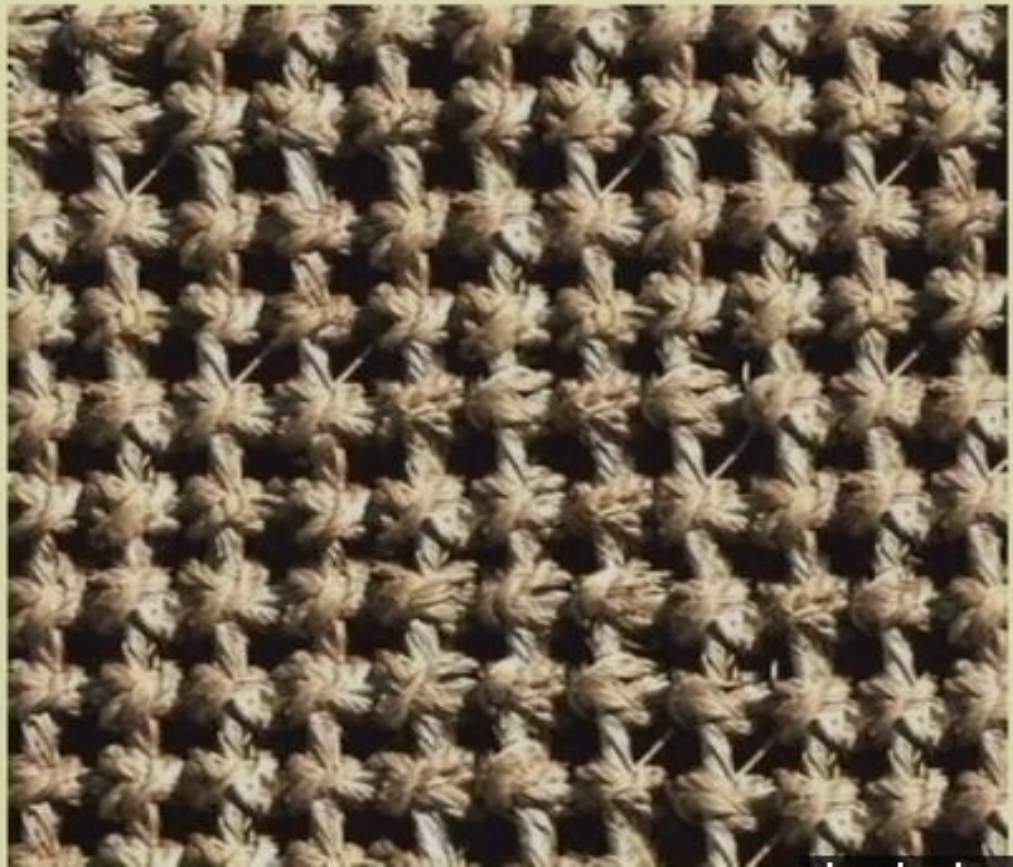


Given a finite sample of some texture, the goal is to synthesize other samples from that same texture



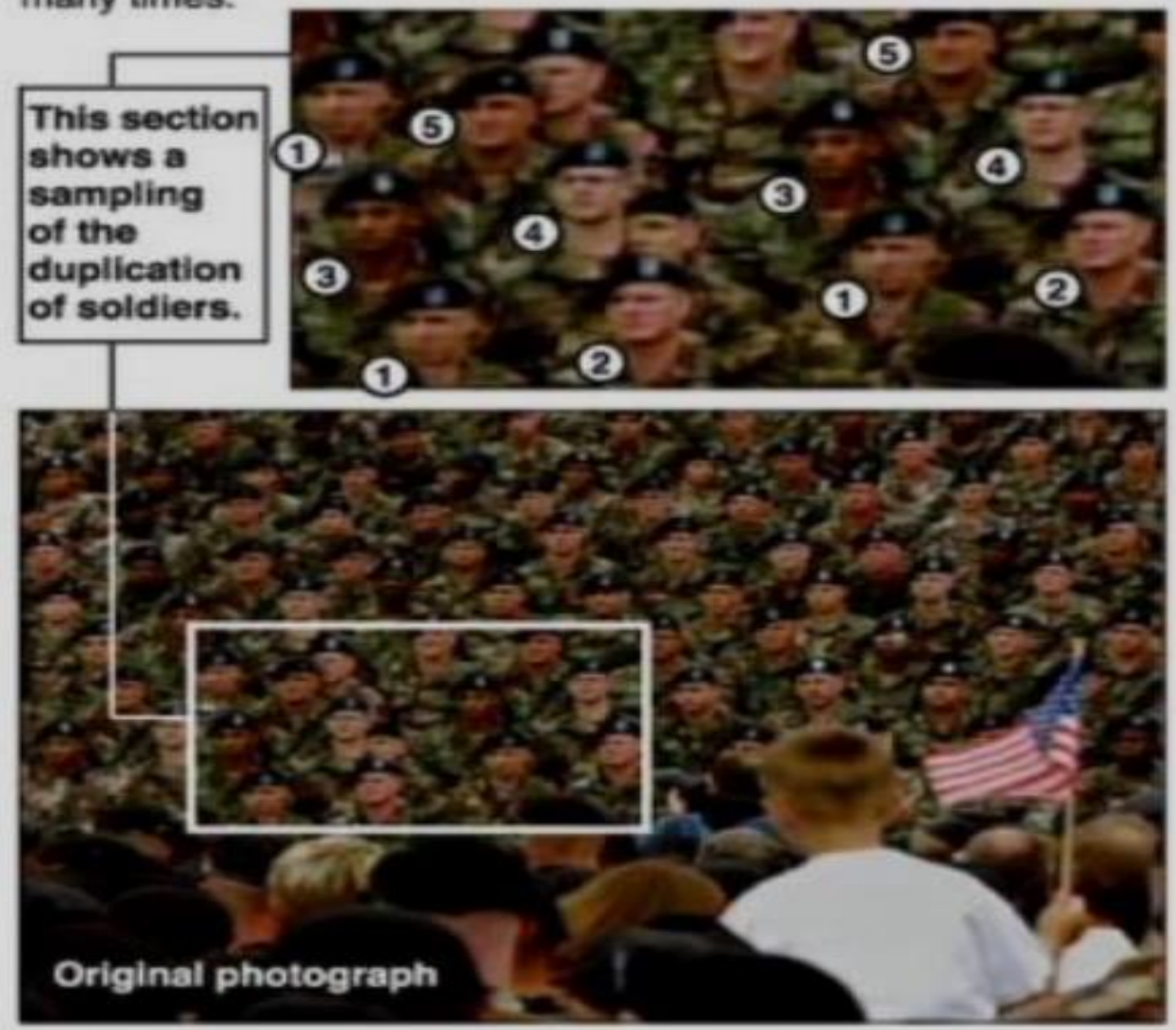






Bush campaign digitally altered TV ad

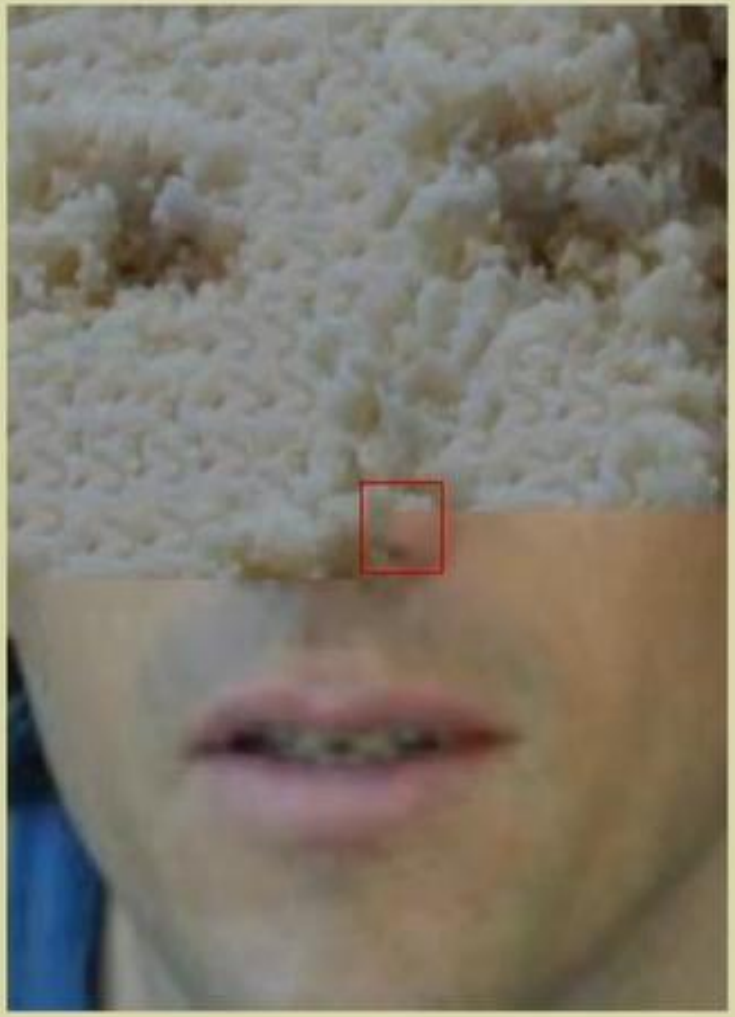
President Bush's campaign acknowledged Thursday that it had digitally altered a photo that appeared in a national cable television commercial. In the photo, a handful of soldiers were multiplied many times.

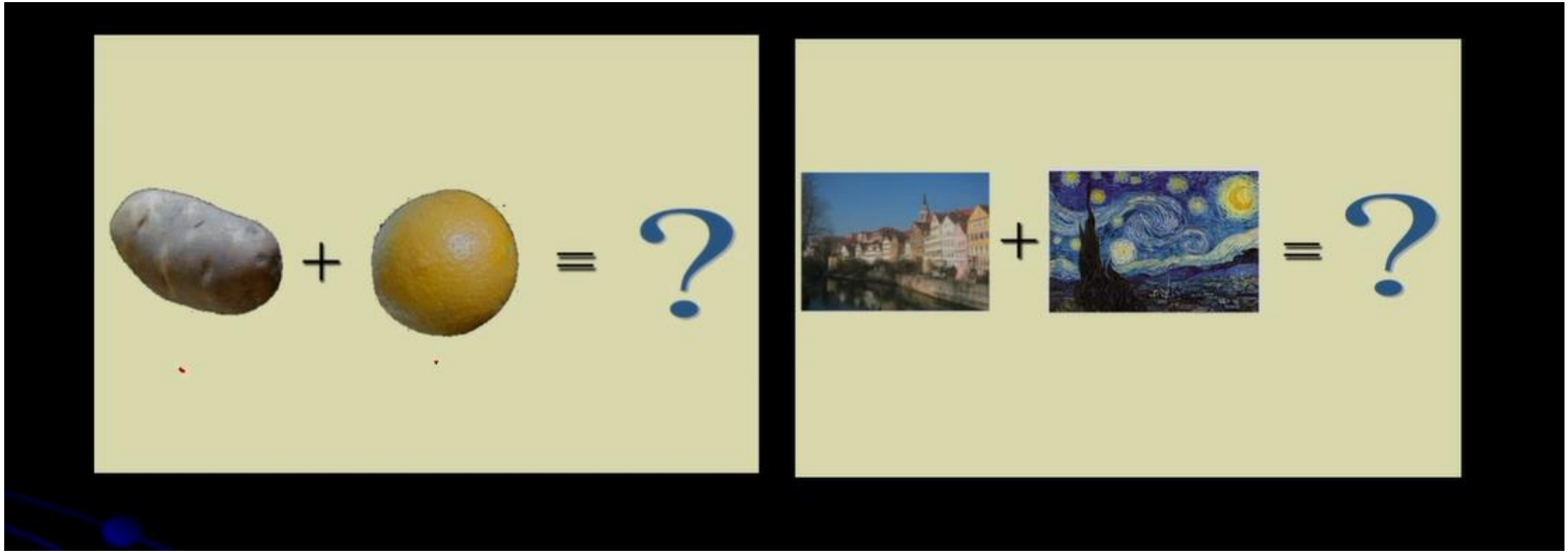


AP

Texture Transfer

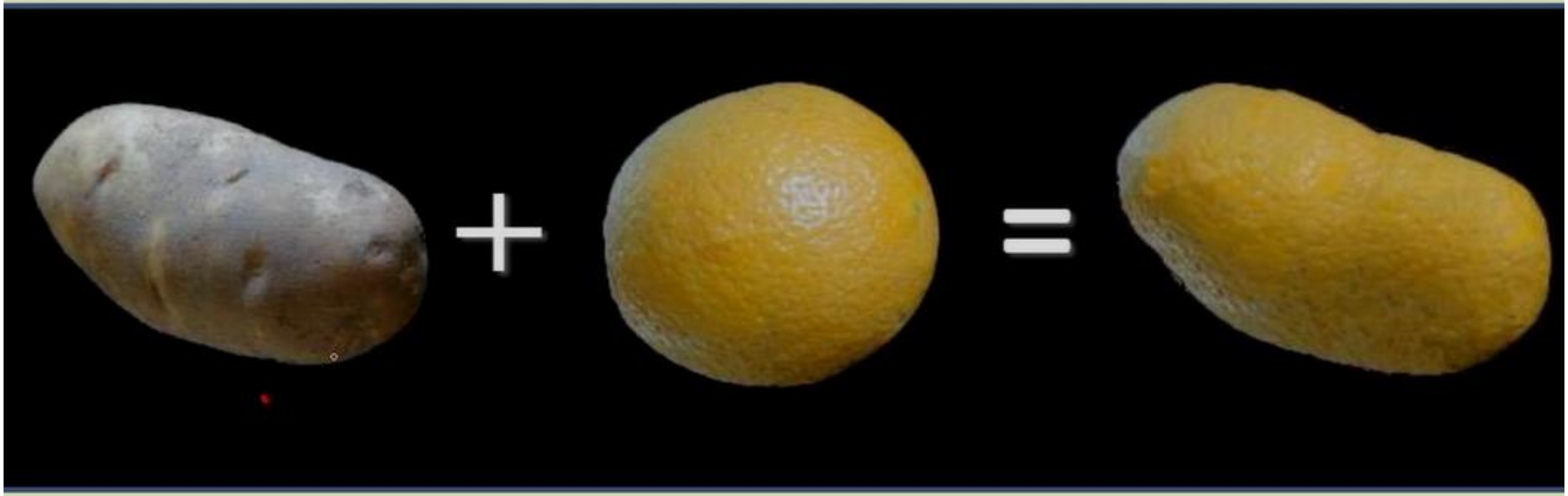
- Take the texture from one image and “paint” it onto another object

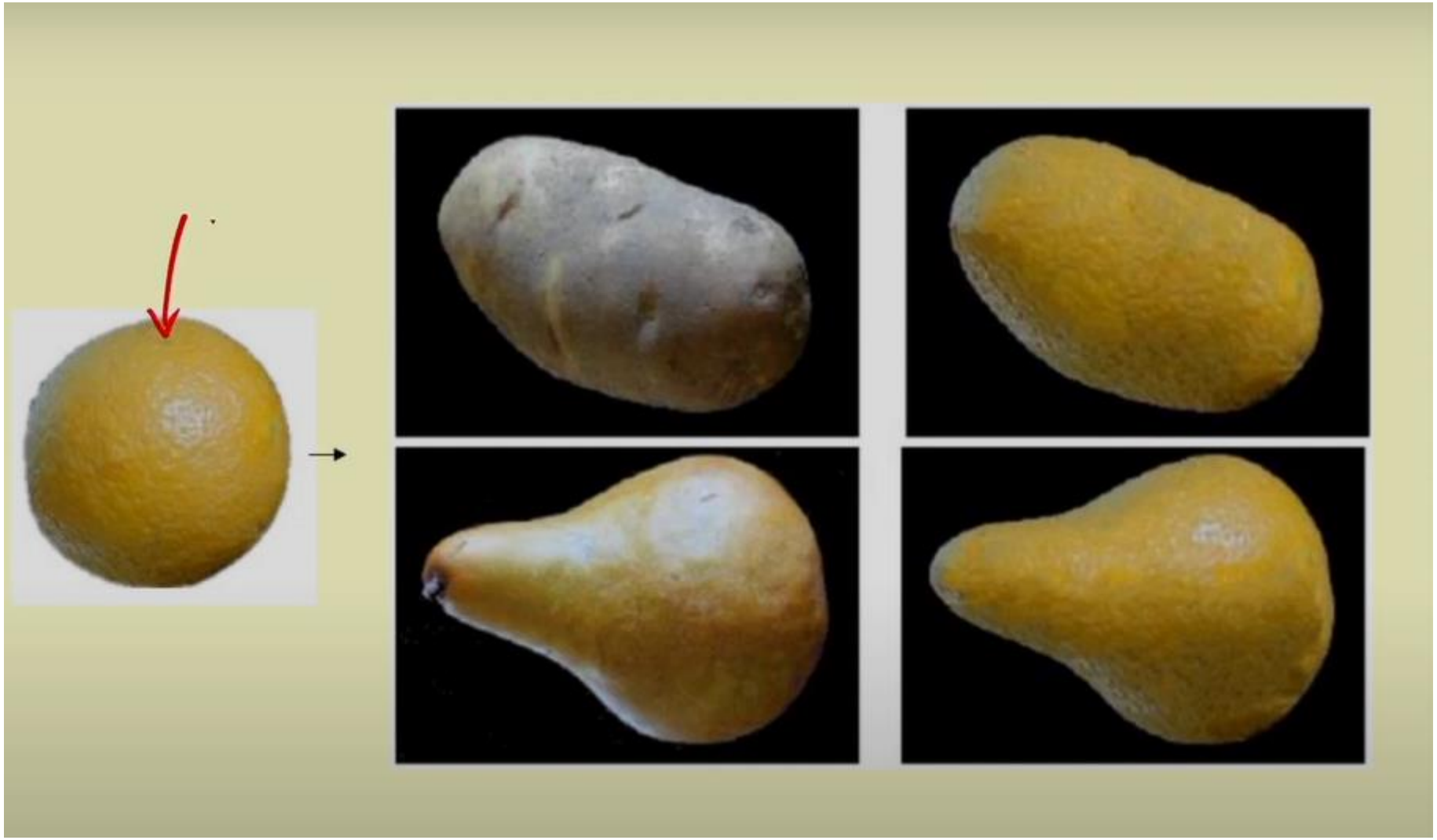




Application: Texture Transfer

- Try to explain one object with bits and pieces of another object:





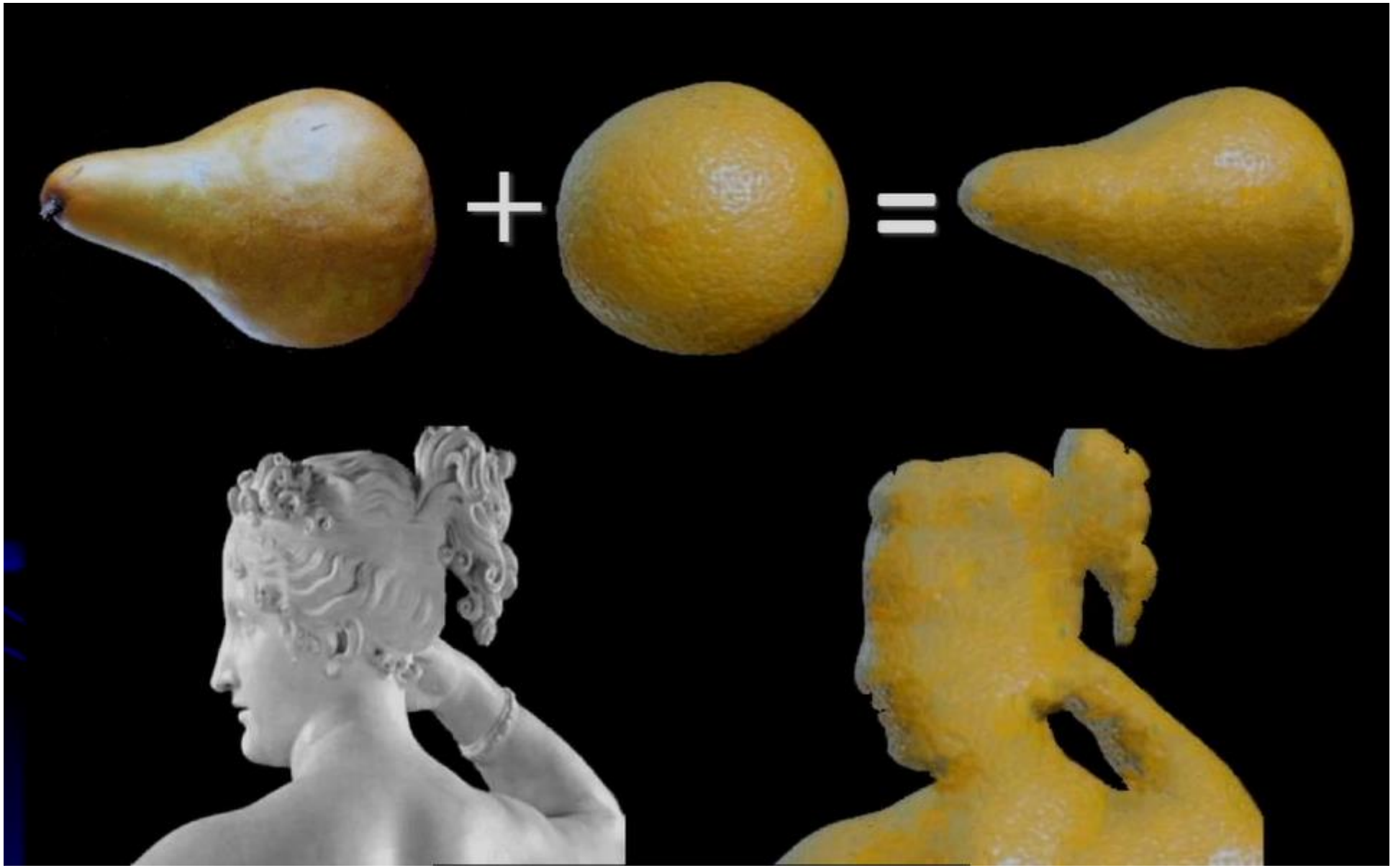
Texture Transfer

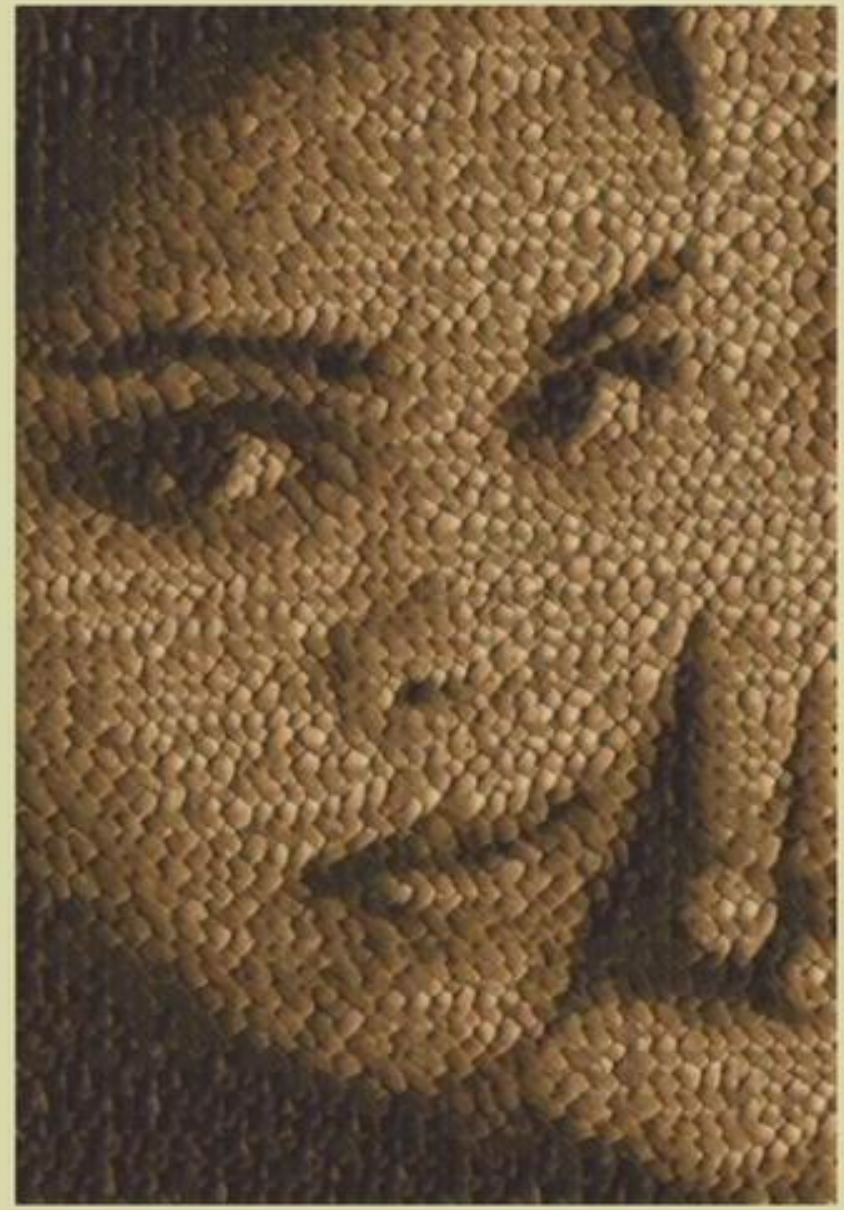
Constraint

Texture sample


+

=



Defining Texture

There are ***three approaches*** to defining exactly what texture is:

- ***Structural*** : texture is a set of primitive texels in some regular or repeated relationship. When the size of the texture primitive is large, first determine the shape and properties of the basic primitive and the rules which govern the placement of these primitives, forming macrotextures.

- ***Statistical*** : texture is a quantitative measure of the arrangement of intensities in a region.

  This set of measurements is called a *feature vector*.

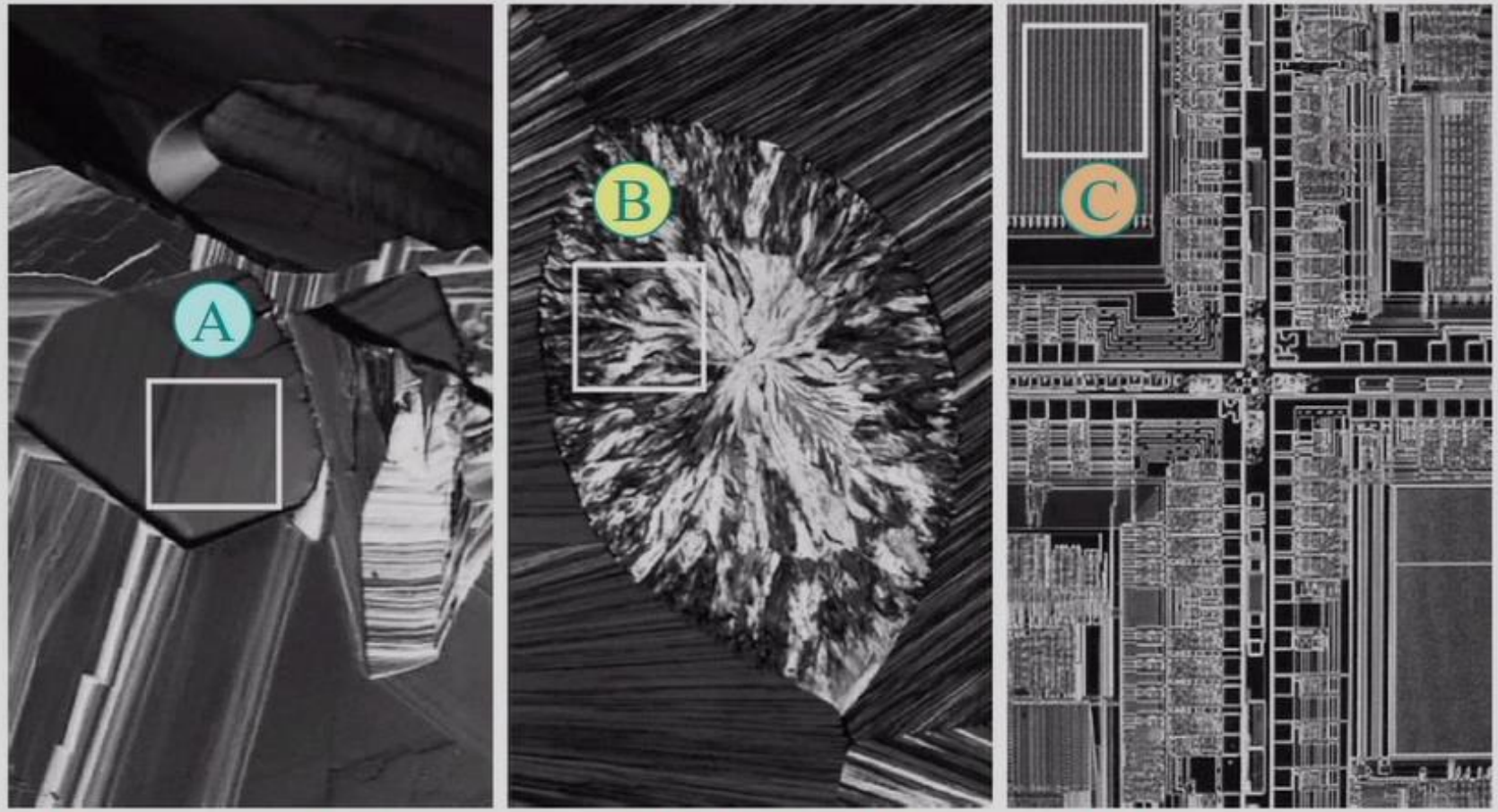
  Statistical methods are particularly useful when the texture primitives are small, resulting in microtextures.

- ***Spectral***: Fourier Transform for texture representation.

Texture Descriptors

Purpose: to describe "texture" of the region.

Examples: optical microscope images:



Superconductor (smooth texture)

Cholesterol (coarse texture)

Microprocessor (regular texture)

Statistical Approaches for Texture Descriptors

We can use statistical moments computed from an image histogram:

$$\mu_n(z) = \sum_{i=0}^{L-1} (z_i - m)^n \, p(z_i)$$

z = intensity

$p(z)$ = PDF or histogram of z

where $$m = \sum_{i=0}^{L-1} z_i p(z_i)$$

Example: The 2nd moment = variance → measure "smoothness"

The 3rd moment → measure "skewness"

The 4th moment → measure "uniformity" (flatness)

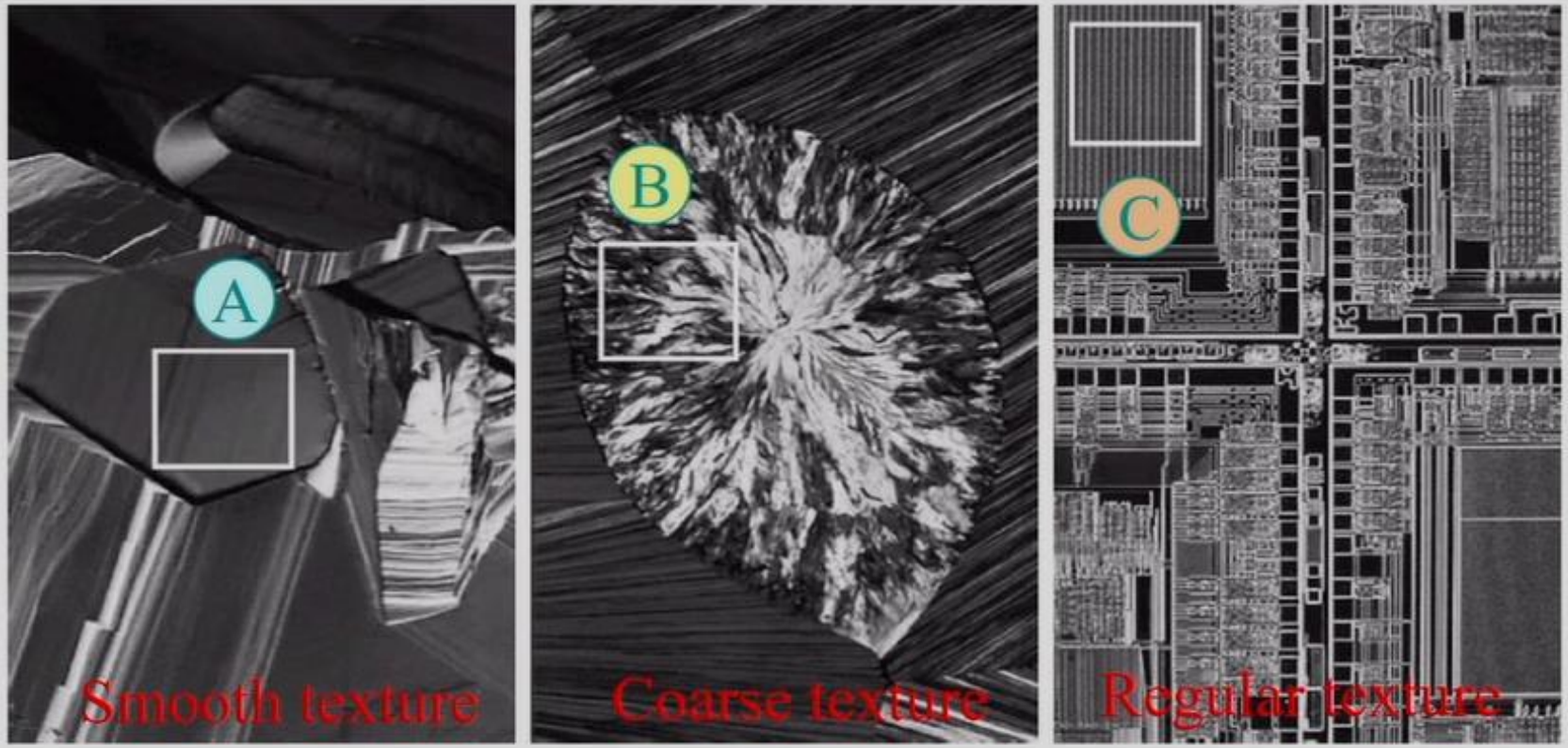
Variance $\sigma^2 = \sum_{i=0}^{L-1} (z_i - m)^2 p(z_i)$ **Roughness factor** $R = 1 - \frac{1}{1+\sigma^2}$

[Information of Smoothness R=0 & Coarseness R=1]

Skewness parameter $\mu_3(z) = \sum_{i=0}^{L-1} (z_i - m)^3 p(z_i)$ **Direction of intensity change.**

Average entropy $e(z) = -\sum_{i=0}^{L-1} p(z_i) \log_2 p(z_i)$ **[Measure of variability]**

Statistical Approaches for Texture Descriptors



	Texture	Mean	Standard deviation	R (normalized)	Third moment	Uniformity	Entropy
A	Smooth	82.64	11.79	0.002	−0.105	0.026	5.434
B	Coarse	143.56	74.63	0.079	−0.151	0.005	7.783
C	Regular	99.72	33.73	0.017	0.750	0.013	6.674

Gray Level Co-occurrence

- The statistical measures described so far are easy to calculate, but do not provide any information about the repeating nature of texture.

- A gray level co-occurrence matrix (**GLCM**) contains information about the positions of pixels having similar gray level values.

GLCM

- A co-occurrence matrix is a two-dimensional array, ***P***, in which both the rows and the columns represent a set of possible image values.

- A GLCM $P_d[i,j]$ is defined by first specifying a displacement vector $\underline{d}=(dx,dy)$ and counting all pairs of pixels separated by ***d*** having gray levels *i* and *j*.

GLCM

- The **GLCM** is defined by:

$$P_d[i, j] = n_{ij}$$

– where n_{ij} is the number of occurrences of the pixel values ***(i,j)*** lying at distance ***d*** in the image.

– The co-occurrence matrix P_d has dimension $n \times n$, where *n* is the number of gray levels in the image.

GLCM

For example, if ***d***=*(1,1)*

2	1	2	0	1
0	2	1	1	2
0	1	2	2	0
1	2	2	0	1
2	0	1	0	1

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

0 1 2 (i)

0 1 2 (j)

there are 16 pairs of pixels in the image which satisfy this spatial separation. Since there are only three gray levels, ***P[i,j]*** is a **3×3** matrix.

GLCM

Algorithm:

- Count all pairs of pixels in which the first pixel has a value ***i***, and its matching pair displaced from the first pixel by ***d*** has a value of ***j***.

- This count is entered in the $\mathbf{i^{th}}$ row and $\mathbf{j^{th}}$ column of the matrix $P_d[i,j]$

- Note that $P_d[i,j]$ is not symmetric, since the number of pairs of pixels having gray levels ***[i,j]*** does not necessarily equal the number of pixel pairs having gray levels ***[j,i]***.

Normalized GLCM

- The elements of $P_d[i,j]$ can be normalized by dividing

each entry by the total number of pixel pairs *i.e.,* total no. of point pairs (*n*) in the image that satisfies $P_d[i,j]$. Divide each & every element of $P_d[i,j]$ *by n* to get the matrix *N(i,j)* .

Normalized GLCM $N[i,j]$, defined by:

$$N[i,j] = \frac{P_d[i,j]}{\sum_i \sum_j P_d[i,j]}$$

which normalizes the co-occurrence values to lie between 0 and 1, and allows them to be thought of as probabilities *i.e., N(i,j) is an estimate of the joint probability distribution.*

Numeric Features of GLCM

- Gray level co-occurrence matrices capture properties of a texture but they are not directly useful for further analysis, such as the comparison of two textures.

- Numeric features are computed from the co-occurrence matrix that can be used to represent the texture more compactly.
  - Maximum probability.
  - Moments.
  - Contrast.
  - Uniformity & Homogeneity.
  - Entropy

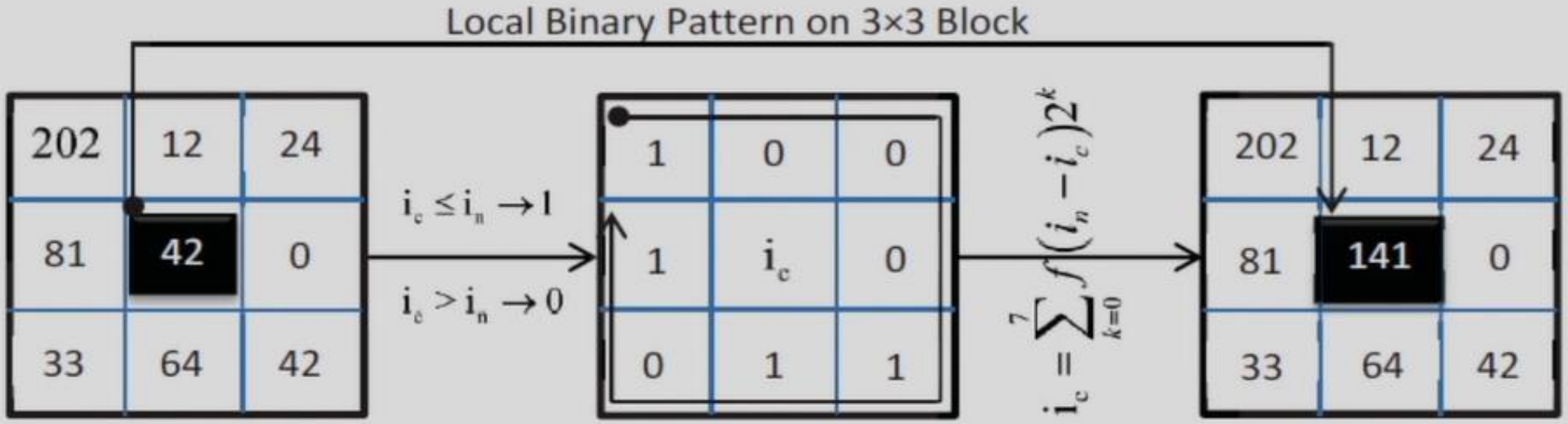
Quantitative Texture Measures

- Numeric quantities or statistics that describe a texture can be calculated from the intensities (or colors) themselves
- One problem with deriving texture measures from co-occurrence matrices is how to choose the displacement vector **d**.
- The choice of the displacement vector is an important parameter in the definition of the GLCM.
- Occasionally the GLCM is computed from several values of **d** and the one which maximizes a statistical measure computed from P[i,j] is used.

WHERE LOCAL BINARY PATTERNS USED

- Local binary pattern (LBP) is a popular technique used for image/face representation and classification.
- Local Binary Patterns (LBPs) have been used for a wide range of applications
  1. Face detection
  2. Face recognition
  3. Facial expression recognition
  4. Remote sensing
  5. Texture classification
  6. Object detection systems

Local binary patterns



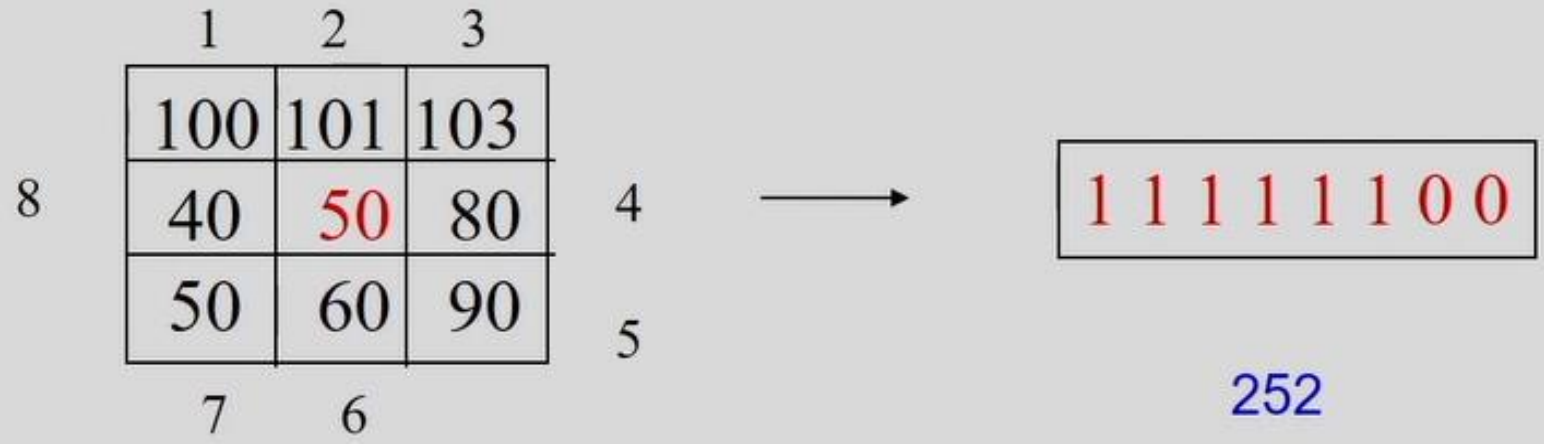
$$LBP_{P,R}(x_c, y_c) = \sum_{k=0}^{P-1} f(i_k - i_c)2^k$$

where, i_k and i_c are gray-level intensity values of the k^{th} neighboring pixel and the central pixel, respectively.

$$f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

Local Binary **Pattern** Measure

- For each pixel p, create an 8-bit number $b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7\ b_8$, where $b_i = 0$ if neighbor *i* has value less than or equal to *p*'s value and 1 otherwise.
- Convert these 8-bit strings to integers.
- Represent the texture in the image (or a region) by the histogram of these numbers.



Gabor filter

A 2D Gabor function consists of a sinusoidal plane wave of a certain frequency and orientation modulated by a Gaussian envelope.

$$\psi_{\omega,\theta}(x,y) = \frac{1}{2\pi\sigma_x\sigma_y}.G_\theta(x,y).S_{\omega,\theta}(x,y)$$

where,

$$G_\theta(x,y) = \exp\left\{-\left(\frac{(x\cos\theta + y\sin\theta)^2}{2\sigma_x^2} + \frac{(-x\sin\theta + y\cos\theta)^2}{2\sigma_y^2}\right)\right\},$$

$$S_{\omega,\theta}(x,y) = \left[\exp\{i(\omega x cos\theta + \omega y sin\theta)\} - \exp\left\{-\frac{\omega^2\sigma^2}{2}\right\}\right]$$

The 2D Gaussian function is used to control the spatial spread of the Gabor filter. The variance parameters of the Gaussian determines the spread along the *x* and *y* directions. Additionally, there is an orientation parameter for this.

$$\psi_{\omega,\theta}(x,y) = \frac{1}{2\pi\sigma_x\sigma_y}.G_\theta(x,y).S_{\omega,\theta}(x,y)$$

where,

$$G_\theta(x,y) = \exp\left\{-\left(\frac{(x\cos\theta + y\sin\theta)^2}{2\sigma_x^2} + \frac{(-x\sin\theta + y\cos\theta)^2}{2\sigma_y^2}\right)\right\},$$

$$S_{\omega,\theta}(x,y) = \left[\exp\{i(\omega x cos\theta + \omega y sin\theta)\} - \exp\left\{-\frac{\omega^2\sigma^2}{2}\right\}\right]$$

- $G_\theta(x,y)$ is a 2D Gaussian function.
- $S_{\omega,\theta}(x,y)$ is a 2D sinusoid function.
- (x,y) corresponds to a spatial location of an image. In this location, the Gabor filter is centered.
- ω corresponds to the frequency parameter of the 2D sinusoid $S_{\omega,\theta}(x,y)$.
- σ^2_{dir} is the variance of the Gaussian function along either x or y direction. The variance fixes the region around the center of the filter for its operation.

- The two sinusoidal components of the Gabor filters are generated by a 2D complex sinusoid.

- The local spatial frequency content of the image intensity variations (texture) in an image can be extracted by applying these two sinusoids.

- There are real and imaginary components of the complex sinusoid. The two components are phase shifted by π/2 radian.

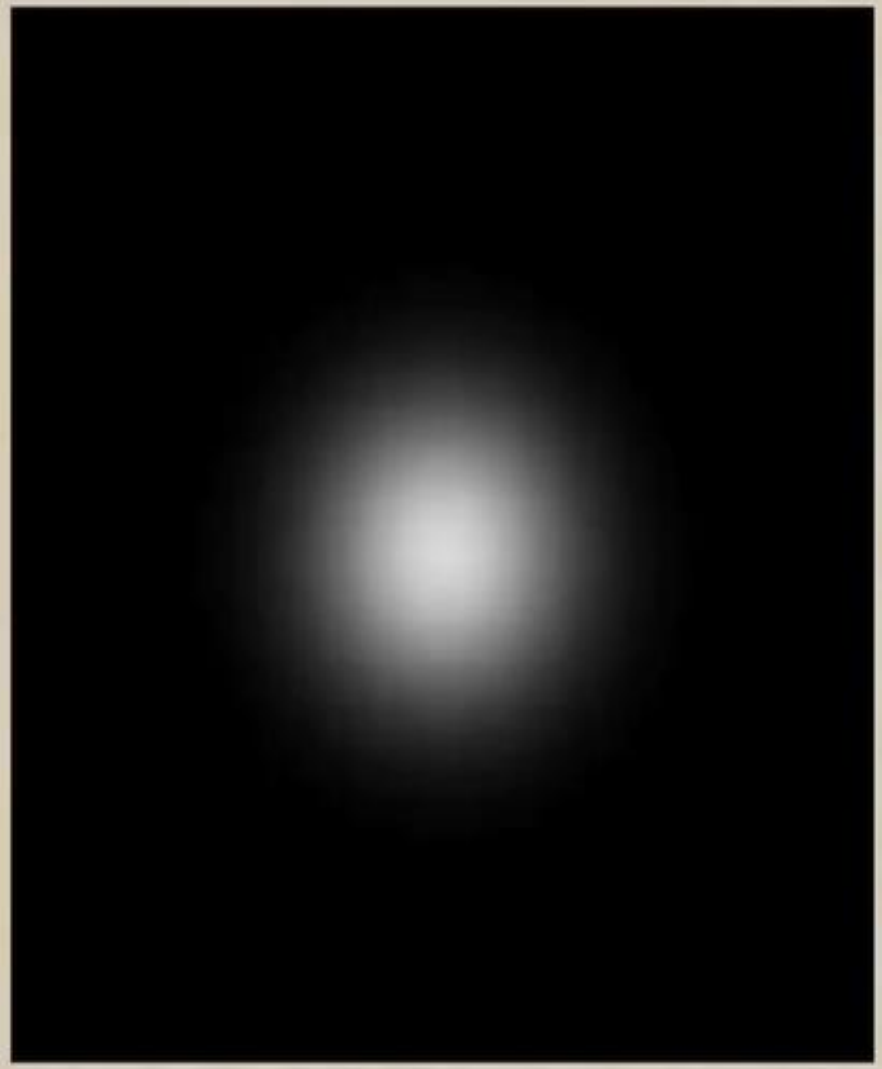
- Gaussian and the sinusoid functions are multiplied to get the complex Gabor filter.

$$\Re\{\psi_{\omega,\theta}(x,y)\} = \frac{1}{2\pi\sigma^2}.G_\theta(x,y).\Re\{S_{\omega,\theta}(x,y)\},$$

$$Im\{\psi_{\omega,\theta}(x,y)\} = \frac{1}{2\pi\sigma^2}.G_\theta(x,y).\Im\{S_{\omega,\theta}(x,y)\}$$

The real and imaginary parts of the Gabor filter are separately applied to a particular location *(x, y)* of an image *f(x,y)* to extract Gabor features

$$C_\psi(x,y) = \sqrt{(f(x,y) * \Re\{\psi_{\omega,\theta}(x,y)\})^2 + (f(x,y) * Im\{\psi_{\omega,\theta}(x,y)\})^2}$$



A typical Gaussian filter with $\sigma=30$

A typical Gabor filter with $\sigma=30$, $\omega=3.14$ and $\theta=45°$

2-D Gabor filter:

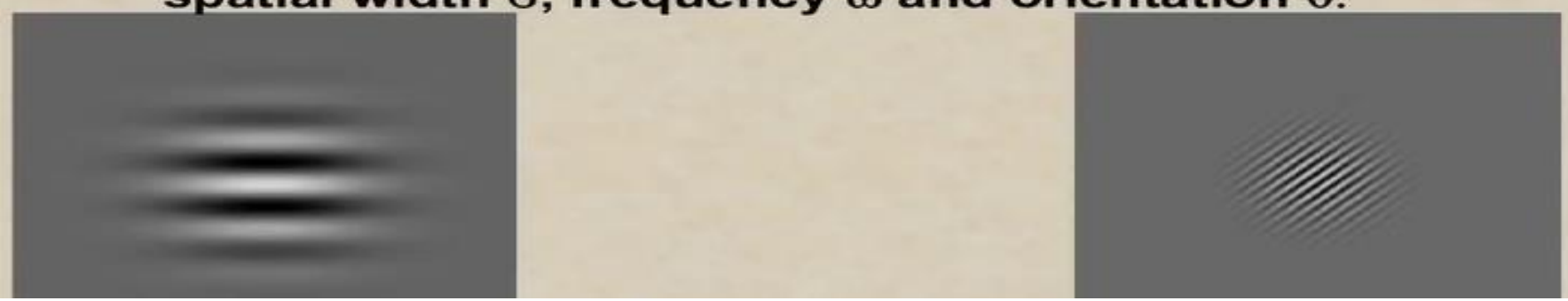
$$f(x, y, \omega, \theta, \sigma_x, \sigma_y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left[\frac{-1}{2}\left(\left(\frac{x}{\sigma_x}\right)^2 + \left(\frac{y}{\sigma_y}\right)^2\right) + j\omega(x\cos\theta + y\sin\theta)\right]$$

where

σ	is the spatial spread
ω	is the frequency
θ	is the orientation

Use a bank of Gabor filters at multiple scales and orientations to obtain filtered images.

Gabor filters with different combinations of spatial width σ, frequency ω and orientation θ.



Laws' Texture Energy Features

- Signal-processing-based algorithms use texture filters applied to the image to create filtered images from which texture features are computed.

- The Laws Algorithm
  - Filter the input image using texture filters.
  - Compute texture energy by summing the absolute value of filtering results in local neighborhoods around each pixel.
  - Combine features to achieve rotational invariance.

Law's texture masks (1)

L5	(Level)	=	[	1	4	6	4	1	]
E5	(Edge)	=	[	-1	-2	0	2	1	]
S5	(Spot)	=	[	-1	0	2	0	-1	]
R5	(Ripple)	=	[	1	-4	6	-4	1	]

- (L5) (Gaussian) gives a center-weighted local average
- (E5) (gradient) responds to row or col step edges
- (S5) (LOG) detects spots
- (R5) (Gabor) detects ripples

Law's texture masks (2)

Creation of 2D Masks

- **1D Masks are "multiplied" to construct 2D masks:** mask E5L5 is the "product" of E5 and L5 –

$$\underbrace{\begin{bmatrix} -1 \\ -2 \\ 0 \\ 2 \\ 1 \end{bmatrix}}_{\text{E5}} \times \underbrace{\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}}_{\text{L5}} = \underbrace{\begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}}_{\text{E5L5}}$$

Texture-based segmentation

Texture-based segmentation

- Texture-based segmentation is a technique used in image processing and computer vision to partition an image into regions or segments based on the texture characteristics of the underlying objects or surfaces.

- Texture refers to the visual patterns and variations in pixel intensity or color within an image region, and it can be an important feature for distinguishing between different materials, surfaces, or objects.

- Texture-based segmentation aims to identify and group image pixels or regions that share similar texture properties

Key Concepts in Texture-Based Segmentation

- **Texture Features:** Texture-based segmentation relies on extracting texture features from an image. These features capture the statistical and structural information related to the distribution of pixel values or color variations in local image neighbourhoods. Common texture features include texture energy, co-occurrence matrices, local binary patterns, and Gabor filters.
- **Local Analysis:** Texture analysis is often performed locally, considering small neighbourhoods of pixels within an image. These local analysis windows or regions move across the image, allowing for the characterization of texture variations at different scales and orientations.
- **Texture Homogeneity:** One of the main assumptions in texture-based segmentation is that objects or regions with similar textures exhibit a degree of homogeneity in their texture features. Therefore, pixels within the same object or surface tend to have similar texture characteristics, making texture a useful criterion for segmentation.

Steps in Texture-Based Segmentation

- **Feature Extraction:** Compute texture features for each pixel or local image region. Common features include texture energy, entropy, contrast, and co-occurrence matrices, among others.
- **Feature Representation:** Represent the extracted features in a suitable feature space. Each pixel's feature vector characterizes its texture properties. This step may involve dimensionality reduction or feature selection techniques.
- **Clustering:** Apply a clustering algorithm to group pixels or regions with similar texture features together. Common clustering algorithms used in texture-based segmentation include k-means clustering, hierarchical clustering, and spectral clustering.
- **Region Labeling:** Assign labels or identifiers to the clusters, which represent segmented regions in the image. These labels can be used to identify and distinguish different objects or surfaces based on their textures.
- **Post-processing:** Depending on the application, additional post-processing steps may be applied to refine the segmentation results. These steps can include noise reduction, boundary smoothing, or region merging/splitting.

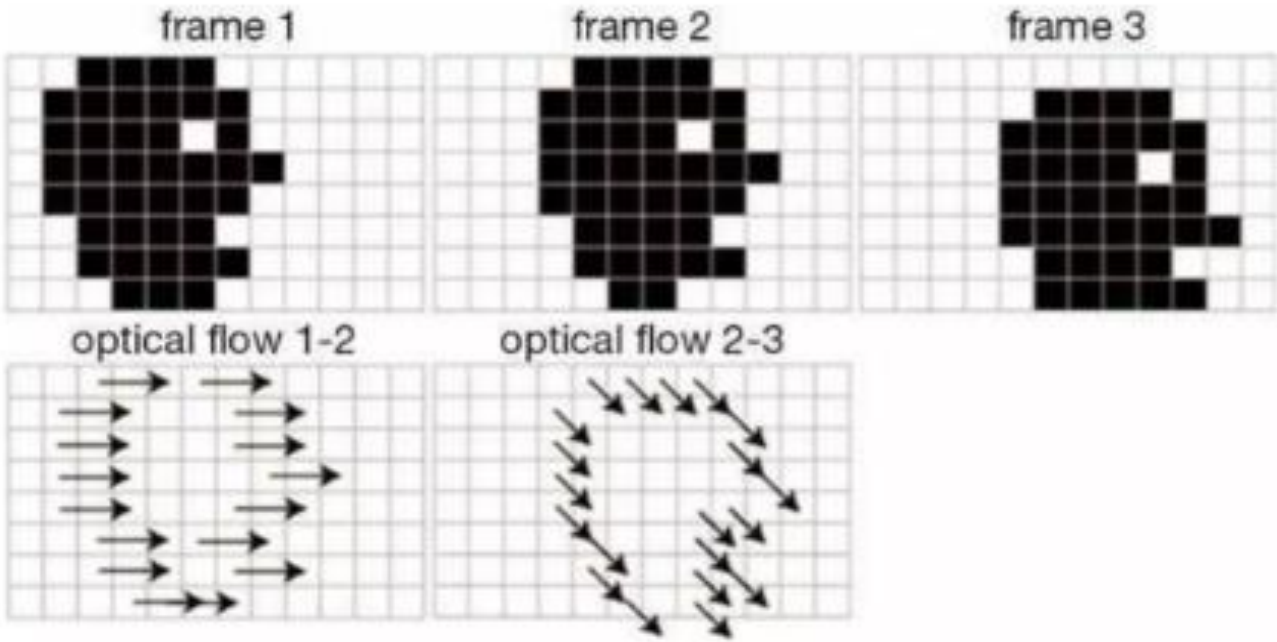
What is *optical flow estimation*?

- Optical flow...

  represents the **apparent motion of objects.**

- Optical flow estimation...

  can predict the **movement of objects in a video.**



Optical flow estimation

- Optical flow estimation is a technique used in computer vision and motion analysis to track the motion of objects in a video sequence.
- It provides information about how pixels in consecutive frames of a video are moving relative to each other.
- This information is crucial for various applications, such as object tracking, video stabilization, and scene understanding

- **Basic Concept**: Optical flow is the apparent motion of objects in an image or video due to the relative motion between the camera and the scene. It is computed by tracking the movement of small image regions (typically pixels) from one frame to the next. The result is a vector field where each vector represents the displacement of a pixel between frames.

- **Assumptions**: Optical flow estimation assumes several key assumptions, including:

  - Brightness constancy: It assumes that the brightness of a pixel remains constant as it moves between frames. This is often referred to as the brightness constancy constraint.
  - Spatial coherence: Neighboring pixels tend to have similar motion vectors. This assumption helps in regularizing the optical flow field.

- **Optical Flow Equation**: The optical flow equation is derived from these assumptions and is typically expressed as follows:

Ixu+Iyv+It=0Ixu+Iyv+It=0

Where:

•IxIx and IyIy are the spatial gradients of the image intensity in the x and y directions.

•uu and vv are the horizontal and vertical components of the optical flow vector.

•ItIt is the temporal gradient, representing the change in intensity over time.

- **Methods for Estimation**: Several methods can be used to estimate optical flow. Some common approaches include:

•**Lucas-Kanade Method**: This is a local optical flow estimation technique that assumes a small motion between frames and solves the optical flow equation for a local image patch.

•**Horn-Schunck Method**: This is a global approach that imposes smoothness constraints on the optical flow field and solves for flow vectors across the entire image simultaneously.

•**Block Matching**: This method divides the image into blocks and matches corresponding blocks between frames to estimate motion.

- **Challenges**: Optical flow estimation can be challenging in the presence of occlusions (a process where by something is hidden), motion discontinuities, and non-rigid deformations (refers to the change in size or shape of an object). Handling these complexities often requires advanced techniques and robust algorithms.

- **Applications**: Optical flow estimation has numerous applications, including:

  - Object tracking: Tracking the movement of objects in videos.
  - Video compression: Efficient video coding by transmitting only the motion vectors.
  - Image stabilization: Reducing the effects of camera shake in videos.
  - Scene understanding: Extracting information about the 3D structure and motion of a scene.

Optical flow estimation is **important**

- Widely used by insects and birds

- Practical usage
  - Analyze motion
  - Avoid collision
  - Assist in navigation

- Real-world Applications
  - Video/Motion classification
  - Navigation assistance
    - Self driving cars
    - Drones

Lucas–Kanade method

- The Lucas-Kanade method is a widely used optical flow estimation technique in computer vision and image processing.
- It is a local method for estimating the optical flow vectors (motion vectors) of small image regions or pixels between two consecutive frames in a video sequence.
- The method is named after its inventors, Bruce D. Lucas and Takeo Kanade, who introduced it in their 1981 paper "An Iterative Image Registration Technique with an Application to Stereo Vision."

- **Objective**: The primary goal of the Lucas-Kanade method is to estimate the apparent motion (optical flow) of a small patch or neighborhood of pixels in an image between two frames. It assumes that the motion in this small region is approximately constant and therefore seeks to find the horizontal (u) and vertical (v) components of the motion vector for this patch.
- **Assumptions**:

- **Brightness Constancy**: The method assumes that the brightness of a pixel or a small patch remains constant as it moves from one frame to another. In other words, it assumes that the intensity of a pixel doesn't change significantly due to motion.
- **Spatial Smoothness**: The method also assumes that neighboring pixels in the image have similar motion vectors. This assumption helps in regularizing the optical flow field and reduces noise.

- Consider an Image I(x,y). For smaller motion, the new image can be represented as:

$$H(x,y) = I(x+u,y+v)$$

- Where (u,v) represents the displacement of the pixel.

- Solving this equation using Taylors Expansion we obtain the Lucas-Kanade equation :

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$A^T A$ $\qquad$ $A^T b$

Algorithm

Step 1 : Compute the Image x and Image y derivatives.

Step 2 : Compute the difference Image It = Image 1 – Image 2.

Step 3 : Smoothen the image components Ix, Iy and It.

Step 4 : Solve the Linear Equations for each pixel and calculate the Eigen values.

Step 5 : Depending on Eigen values obtained, solve the equations using Cramer's rule.

Step 6 : Plot the optical Flow vectors.

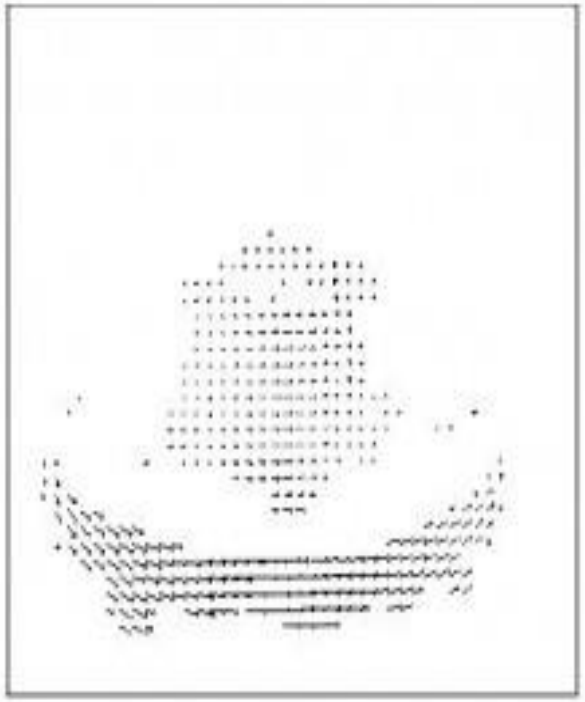
➢ **Advantages**:

- The Lucas-Kanade method is computationally efficient and suitable for real-time applications.
- It works well when motion is relatively small within a local region.

➢ **Limitations**:

- It assumes that the motion is well approximated as constant within the small region, which may not hold for large motions or non-rigid deformations.
- It does not handle occlusions or large displacements effectively.
- The method can be sensitive to noise.

Horn-Schunck method



Horn-Schunck method

- The Horn-Schunck method, also known as the Horn-Schunck optical flow method, is a global approach for estimating optical flow in a video sequence.
- This method was developed by Robert B. Horn and Brian G. Schunck in the late 1980s. Unlike the Lucas-Kanade method, which is a local approach, the Horn-Schunck method estimates the optical flow for all points in an image simultaneously.
- This global estimation approach makes it suitable for scenarios with large motion and significant changes in the scene.

➢ **Objective**: The primary goal of the Horn-Schunck method is to estimate the optical flow vectors (motion vectors) for all pixels in an image between two consecutive frames. It aims to find the horizontal (u) and vertical (v) components of the motion vector at each pixel.

➢ **Assumptions**:

- **Brightness Constancy**: Similar to the Lucas-Kanade method, the Horn-Schunck method assumes that the brightness of a pixel remains constant as it moves from one frame to another.
- **Smoothness Constraint**: It assumes that neighboring pixels have similar motion vectors, imposing a smoothness constraint on the optical flow field. In other words, the motion field should vary smoothly across the image.

- **Algorithm**: The Horn-Schunck method formulates the estimation problem as an energy minimization problem, where the goal is to minimize a global energy functional. The energy functional is defined as the sum of squared differences between the gradients of the estimated flow field and the temporal gradient (change in intensity) between frames. The optimization problem seeks to minimize this energy functional subject to the smoothness constraint.
- Here are the key steps:

- **Calculate Image Gradients**: Compute the spatial gradients of image intensity (I_x and I_y) for both frames.
- **Initialize Flow Field**: Initialize the horizontal (u) and vertical (v) components of the motion vectors at each pixel. This can be done with initial guesses or starting with zero values.
- **Iterative Optimization**:
  - In each iteration, update the motion vectors (u and v) to minimize the energy functional.
  - The update is performed by solving a system of partial differential equations (PDEs) derived from the energy functional.
  - These PDEs take into account the brightness constancy and smoothness constraints.
- **Convergence**: Repeat the iterative optimization until the motion field converges to a solution. Convergence can be determined based on criteria such as the change in motion vectors between iterations or the energy decrease.

❑**Advantages**:

- The Horn-Schunck method provides a global estimation of optical flow, which makes it suitable for handling large displacements and non-rigid motions.
- It is capable of handling complex scenes with significant motion variations.

❑**Limitations**:

- It is computationally more intensive compared to local methods like Lucas-Kanade.
- It may not perform well in the presence of large motion discontinuities or occlusions.
- The accuracy of the results can be affected by the choice of parameters and initializations.

Background subtraction

(a)

(b)

(c)

(d)

Background subtraction

- Background subtraction is a fundamental technique in computer vision and image processing used to extract objects or regions of interest from a video sequence by separating them from the background.

- This technique is commonly employed in applications such as object tracking, motion detection, surveillance, and video segmentation.

- The primary idea behind background subtraction is to identify and isolate the parts of an image or video frame that have changed over time, assuming that the background remains relatively static.

Background subtraction works

- **Background Modeling**: The first step in background subtraction is to create a model of the background scene. This is typically done by capturing several initial frames of the video when there are no foreground objects or motion of interest. These frames are used to build a statistical model of the background, which could be represented as a single background image or as a probability distribution for each pixel's intensity values.
- **Frame Subtraction**: Once the background model is established, each new frame of the video is compared to the background model to detect changes. This is done by subtracting the current frame from the background model.
  - If the difference between a pixel's intensity in the current frame and the corresponding pixel in the background model exceeds a predefined threshold, that pixel is considered part of the foreground. Otherwise, it is part of the background.
- **Thresholding**: Thresholding is an essential step in background subtraction. It helps distinguish between the foreground and the background by setting a pixel as foreground when the absolute difference in intensity values exceeds a certain threshold. The choice of threshold depends on the specific application and the characteristics of the video.

- **Post-processing**: After the initial background subtraction, post-processing techniques may be applied to refine the results. These techniques can include noise reduction, morphological operations (e.g., erosion and dilation), and contour analysis to group pixels into connected regions.
- **Object Detection**: Once the foreground regions have been identified, objects or regions of interest can be detected by analyzing the connected components in the binary mask obtained after thresholding and post-processing. Each connected component corresponds to a potential object or region that has moved or changed in the scene.
- **Update Background Model**: To adapt to gradual changes in the background, many background subtraction algorithms incorporate a mechanism to update the background model over time. This helps maintain an accurate representation of the background even in scenarios with slow lighting changes or gradual scene variations.

Dense optical flow using Deep Learning (FlowNet)

Dense optical flow using Deep Learning (FlowNet)

Introduction to FlowNet:

•FlowNet is a deep learning architecture designed to perform dense optical flow estimation.

•It was introduced in the paper "FlowNet: Learning Optical Flow with Convolutional Networks.“

FlowNet Architecture:

•FlowNet consists of two main variants: FlowNetSimple and FlowNetCorr.

•FlowNetSimple:

- It includes input image pairs, which are typically grayscale images.
- Convolutional layers are used for feature extraction.
- Siamese network structure shares weights between input frames.
- Additional convolutional layers capture spatial relationships and motion patterns.
- Upsampling layers increase the resolution of the predicted flow field.
- The output layer produces the dense optical flow field with vectors (u, v) for each pixel.

Training FlowNet:

•FlowNet is trained using a large dataset of optical flow ground truth data.

•Common loss functions for training include L1 loss or smooth L1 loss.

•The model's weights are updated via backpropagation and gradient descent to minimize the loss.

Inference with FlowNet:

•After training, FlowNet can be used for optical flow estimation on new image pairs.

•Given a pair of frames, the model predicts the optical flow field, which can be used in various applications.

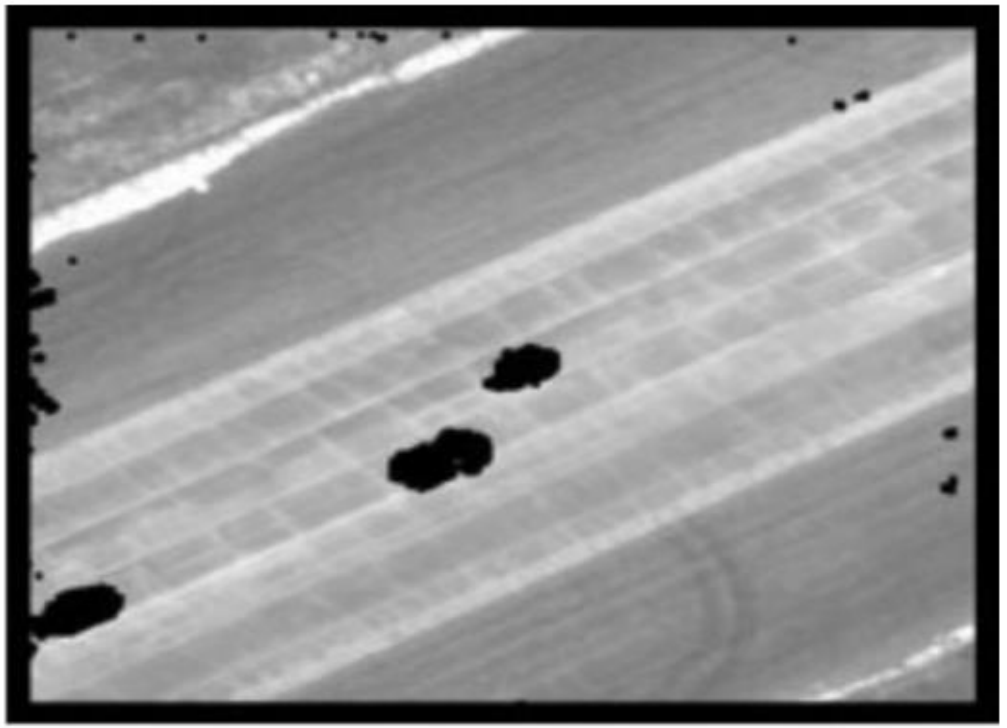
Applications of FlowNet:

•FlowNet has found applications in diverse fields, including:

- Autonomous vehicles for motion estimation.
- Robotics for object tracking and navigation.
- Video analysis and understanding for scene dynamics.

Motion-based segmentation

(a) Original image



(b) Motion segmentation result

1. Introduction to Motion-Based Segmentation:

•Motion-based segmentation is a computer vision technique used to separate objects in a video stream based on their motion characteristics.

•It's an essential tool for applications like object tracking, activity recognition, and video analysis.

2. Key Concepts:

•**Motion Patterns:** Different objects in a scene can exhibit various motion patterns, such as translation, rotation, scaling, and deformation.

•**Background vs. Foreground:** The goal is to distinguish between the background (stationary objects or scene) and the foreground (moving objects).

3. Techniques for Motion-Based Segmentation

- **Frame Differencing:**
  - Subtracting consecutive video frames to identify pixels that have changed.
  - Simple and computationally efficient but sensitive to noise.
- **Optical Flow:**
  - Estimates the motion vector for each pixel to identify moving objects.
  - Effective for dense motion fields but may struggle with occlusions and abrupt changes.
- **Background Subtraction:**
  - Identifying the background and foreground by modeling the background using historical frames.
  - Can be based on simple methods like frame averaging or more advanced techniques .
- **Motion Energy Image:**
  - Computing the energy or magnitude of motion vectors for each pixel.
  - Effective in identifying areas with significant motion.
- **Deep Learning Approaches:**
  - Employing convolutional neural networks (CNNs) or recurrent neural networks (RNNs) for motion-based segmentation.
  - Deep learning models can automatically learn complex motion patterns and adapt to different scenarios.

Applications

- Object tracking in surveillance and autonomous vehicles.
- Action recognition in sports and human-computer interaction.
- Video analytics for security and traffic monitoring.
- Augmented reality and virtual reality experiences.