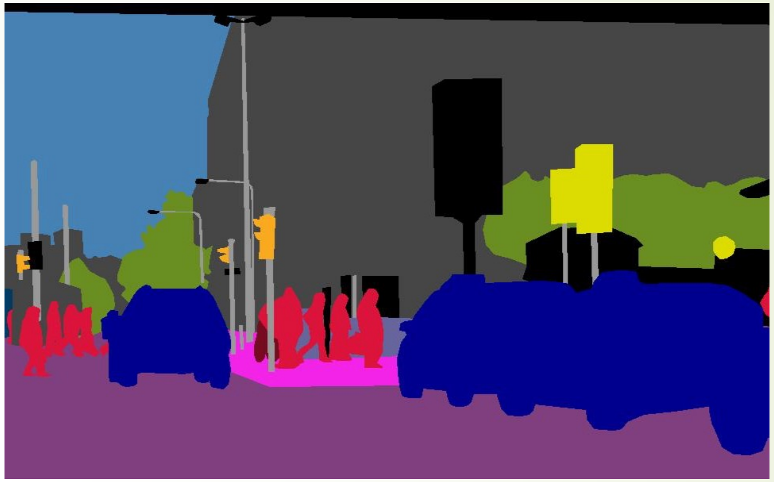
FUNDAMENTALS OF IMAGE AND VIDEO PROCESSING

Part 5: Middle-level image processing

Middle-level image processing

- In this section we introduce intermediate-level operations, whose purpose is to **convert the image into more meaningful** (semantically richer) **representations**
  - For instance, starting from a pixel-level representation we want to describe the image in terms of contained "objects"
  - Each object could be described by its shape, color, texture, etc.
  - At this level, we are still not able to define what an object is, but just its **appearance**.
- Middle-level processing produces descriptions that are easier to use for high-level systems
  - Based on descriptors, high-level could go further in semantic interpretation, trying to make sense of the image content

Middle-level processing: example



- Starting from the image on the left, we want to describe the objects contained in the scene (right)
  - We could, e.g., extract areas that are uniform according to some properties, and describe them in terms of shape and texture
  - For instance, the blue 'blob' on the left could be described as compact shape with sharp angles, mostly convex with a major concavity on the bottom, with a dominant dark-red color.
  - We are still unable to associate such blob to the concept of 'car'

Image source: A. Kirillov et al., CVPR 2019

From middle- to high-level

- Later on, the blobs could be associated to visual concepts
  - This requires sophisticated machine learning techniques

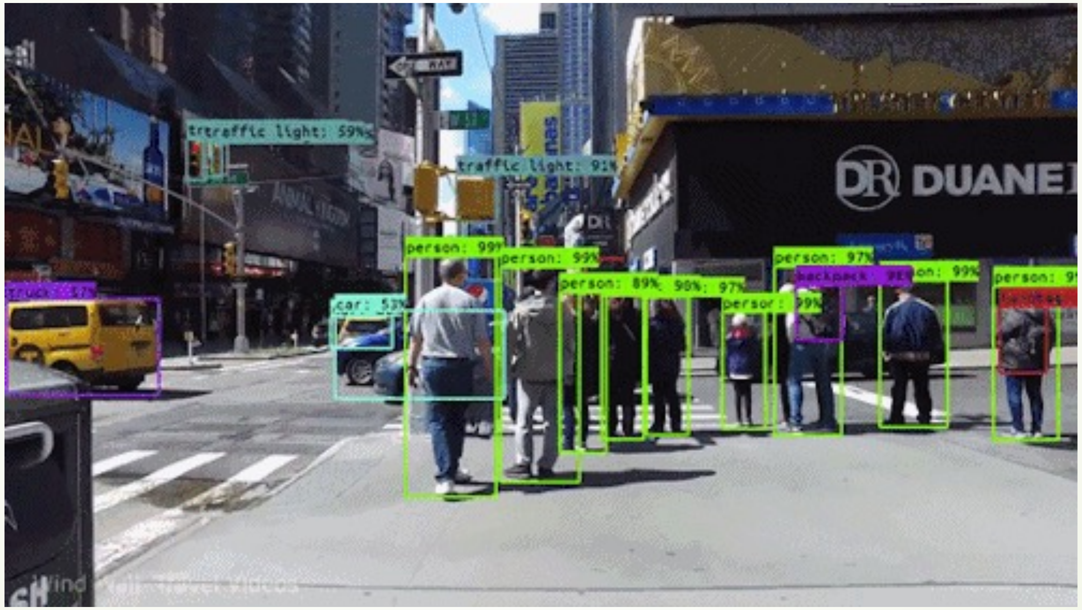


Image source: towardsdatascience.com

5

Image descriptors

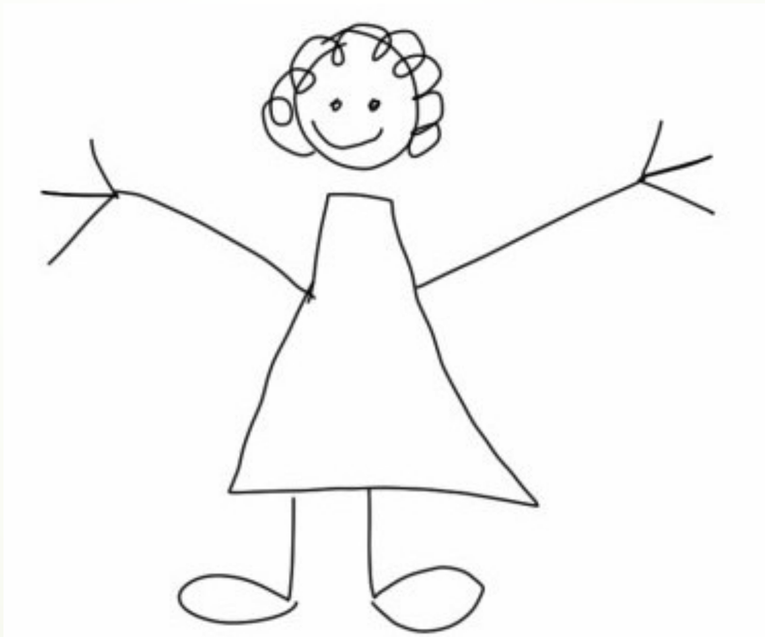
- We'll introduce four main types of descriptors:
  - **Contours**: boundaries of objects
  - **Regions**: shapes associated to objects
  - **Textures**: arrangement of colors on the object surface
  - **Structures**: groups of objects with spatial relationships
- The four descriptors are somewhat complementary
  - Contours enclose regions
  - Regions contain textures
  - Groups of regions/contours create structures
- The techniques used to extract such descriptors from images, however, are quite different

The process of extracting image descriptors

- We distinguish 3 main phases:
  - **Detection**: it is the action of revealing the desired descriptor from the image (e.g., detecting edge points to produce an edge-map)
  - **Representation**: it is the action of associating an appropriate description to each detected primitive (e.g., representing a chain of edge point as a 2D curve, or contour)
  - **Feature extraction**: it is the action of associating a set of quantitative parameters to each detected primitive (e.g., representing a contour in terms of length, shape, closeness, curvature, etc.)
- Each phase can be implemented in different ways (algorithms), with relevant pros and cons

Contours

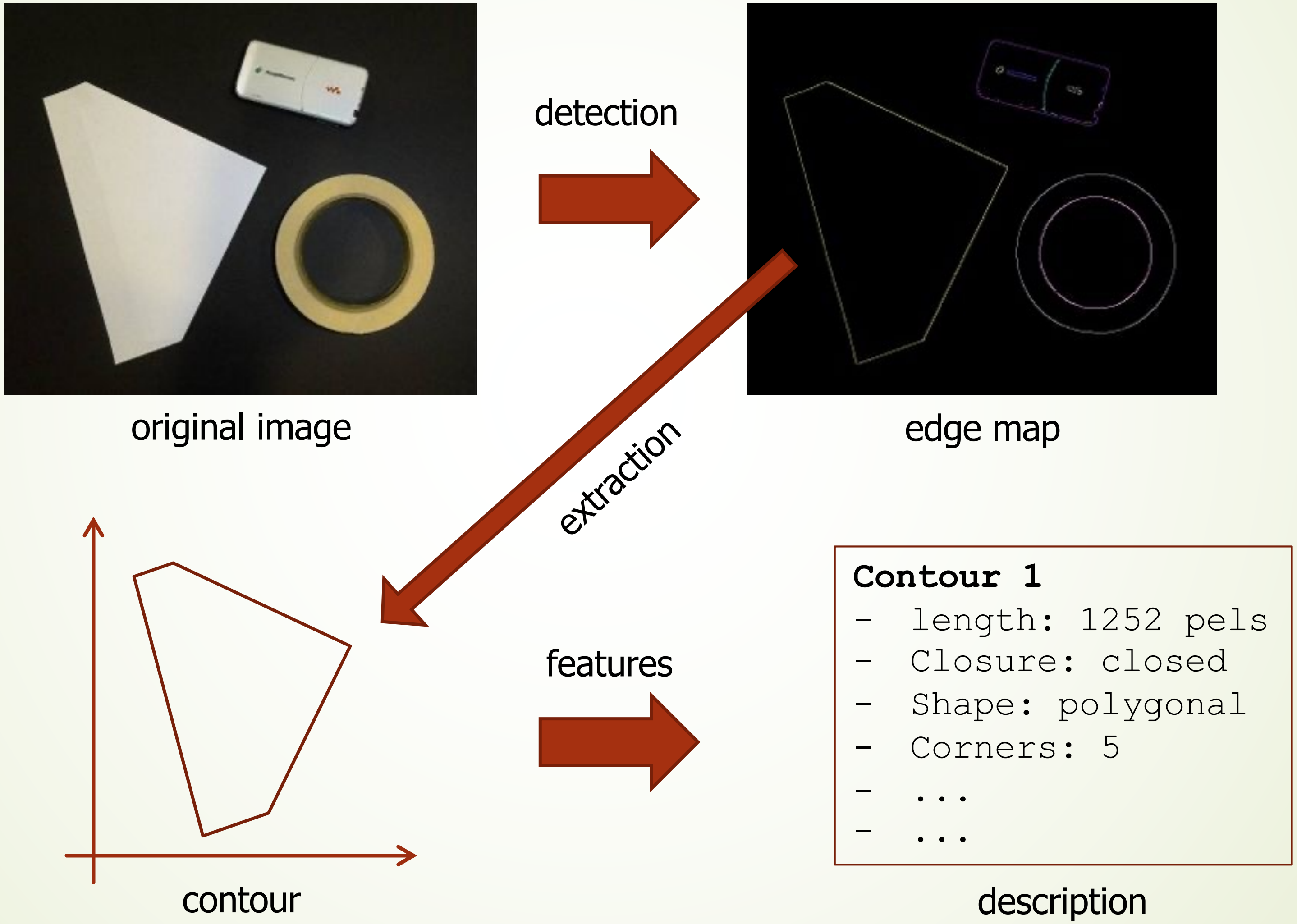
- The first descriptors we introduce are image contours
- The **expressivity of contours** in representing the nature of objects is rather evident



Contour extraction process

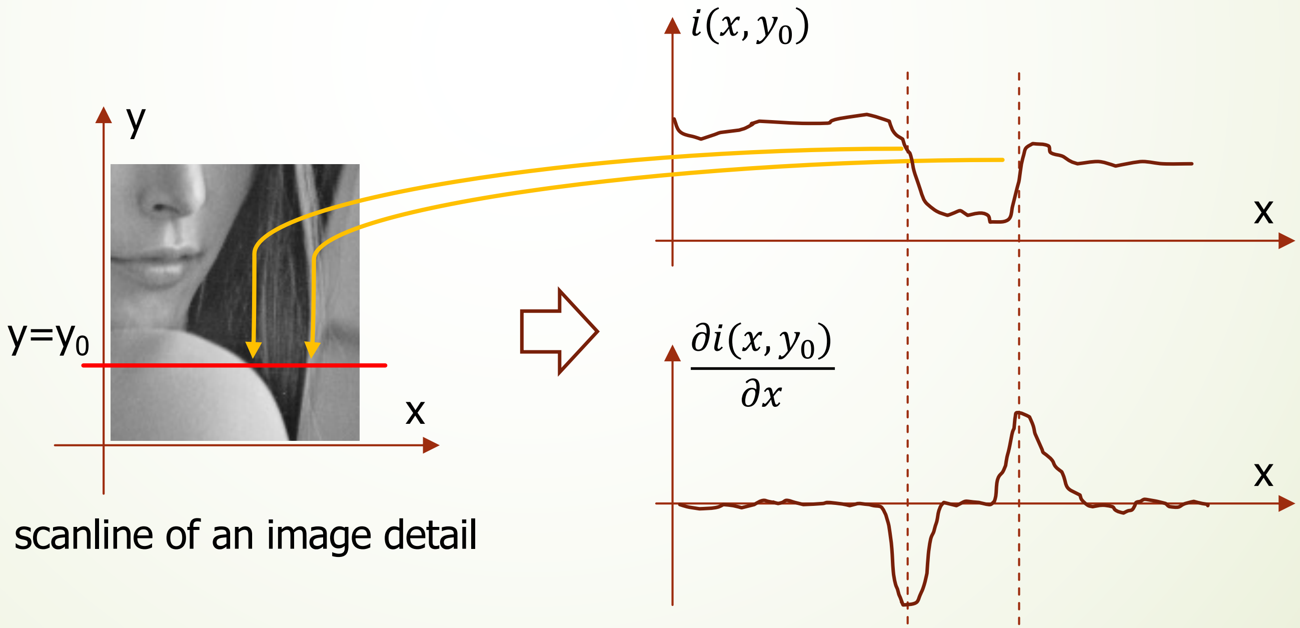
- As we've seen, we distinguish 3 phases:
  - **Edge detection**: we convert an image (grayscale or color) into a binary map of edges
  - **Contour extraction**: we scan the edge points that have been detected at the previous step to create connected chains of contours (curves)
  - **Feature extraction**: we associate a set of parameters to each contour chain

Contour extraction: example



Edge detection

- Edges are characterized by steep luminance/color variations that are present in an image in the presence of object borders
  - Since edges are associated to steep variations of the image function, they can be detected by analyzing spatial high-frequencies
  - Typical approaches are therefore based on the use of gradient filters



scanline of an image detail

Sobel edge detector: example

x derivative

y derivative

modulus

12

Sobel pseudo-code

```
int img[N][M], edges[N][M]                  // input, output images
int grad_x[N][M], grad_y[N][M]              // directional gradient images
int kernel_x = [[1,2,1],[0,0,0],[-1,-2,-1]]  // Sobel FIR x direction
int kernel_y = [[1,0,-1],[2,0,-2],[1,0,-1]]  // Sobel FIR y direction

load (img)
grad_x = convolve(img, kernel_x)            // FIR filtering with kernel x
grad_y = convolve(img, kernel_y)            // FIR filtering with kernel y
for n in 0...N-1 {                          // raster scan image
   for m in 0...M-1 {                       // " "
     tmp = (grad_x[n][m]^2+grad_y[n][m]^2)   // square mod
     if (tmp > THR) edges[n][m] = 1          // thresholding
     else edges[n][m] = 0                    // " "
     }}
```

Gradient operators: pros and cons

- PROS
  - Easy to implement
  - Limited complexity (two 3x3 FIR filters)
  - Relatively accurate if image is not too noisy
- CONS
  - Hardly detect weak contours
  - Disconnected and thick contours (depends on thresholding)
  - Imprecise localization

Canny edge detector

- John Canny proposed to address the problem of optimal edge detection in a formal way
- An "optimal" detector should provide:
  - low misdetection (catch as many real edges as possible)
  - good localization (identify the center of the edge)
  - low rate of false edges (edges should be marked only once and noise should not create false edges)
- To this end Canny proposed a method based on the optimization of a functional, defined as the sum of 4 exponential terms
  - In actual implementations, the above optimization is approximated by a more traditional sequence of filters

Canny: typical implementation

- The detector follows 5 steps:
  - Apply a **Gaussian filter** to remove noise (typically, a 5x5 FIR kernel)
  - Calculate the **gradients along x and y** (similar to Sobel), and associate to each point an intensity and an angle (quantized to 4 directions)
  - Perform **non-maximum suppression**, a kind of thinning where the edge strength is compared to the neighbors along the edge direction and only the point with larger gradient is selected
  - Apply **double thresholding**: pixels are marked as strong edges, weak edges or suppressed according to a lower and an upper threshold
  - Edge **tracking by hysteresis**: weak edges are preserved only if they are 8-connected to at least a strong one

Canny edge detection: example

Canny detector: pros and cons

- PROS
  - Very good localization
  - Thin contours
  - High-continuity of lines, also for weak contours
  - Highly resistant to noise
- CONS
  - Several parameters/thresholds
  - Relatively complex

Contour representation & description

- Edge detectors produce in output binary images
- This is not yet a true contour representation
  - We still need to process the edge image and extract the contour chains, in the form of 2D curves.
- To this purpose, we need appropriate algorithms to follow sequences of connected contour points
- The most known algorithm is the so-called **Freeman chain code**

Textures

- The last descriptor we introduce is the texture
- Textures complement contours and shapes in revealing (or hiding) the nature of an object

Textures

- A texture is the visual appearance of a surface. It can be thought as a spatial arrangement of colors (pattern) showing some kind of regularity
  - In the simplest case, it can be a uniform color
- It is a complementary feature that, associated to a shape (contour or region), completes the description of an object

Examples of textures

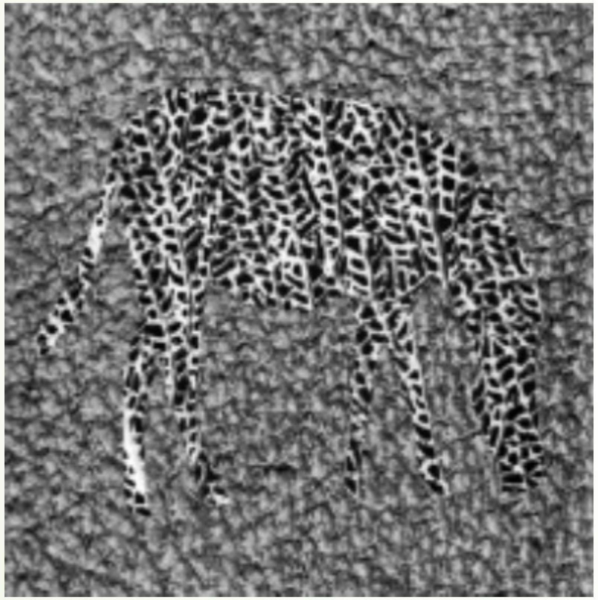
Source: Brodatz texture database

Texture analysis

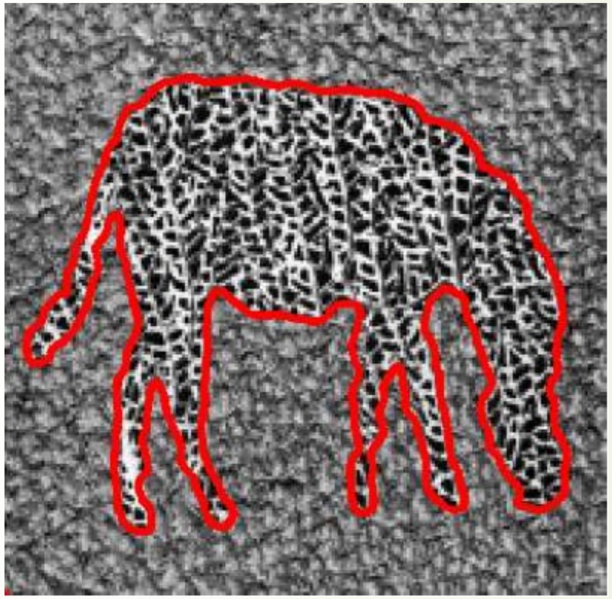
- We typically don't "detect" textures, but we rather analyze them to:
  - define parameters that can identify a given texture (e.g., to evaluate texture homogeneity in segmentation)
  - detect texture irregularities (e.g., finding anomalies in textured surfaces for visual inspection purposes)
  - classify textures (e.g., in object detection, when the texture is a characteristic feature of a given object)
  - define models to synthesize similar textures (e.g., in computer graphics and virtual reality)

Texture analysis: examples

- Using texture homogeneity feature in segmentation



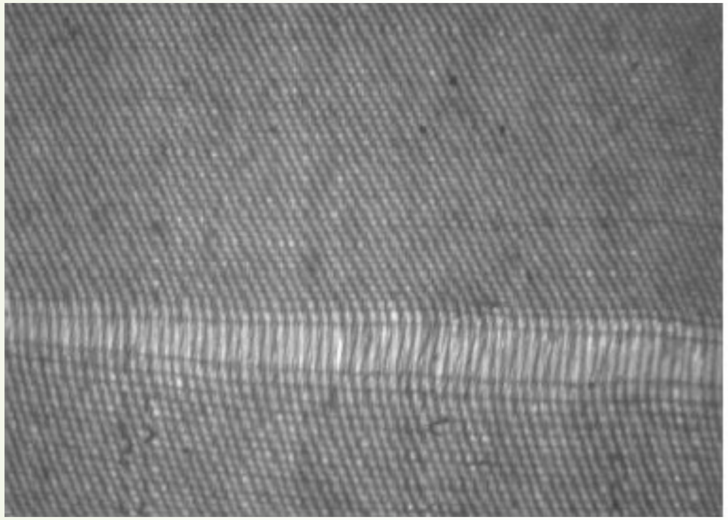
Artificial test image



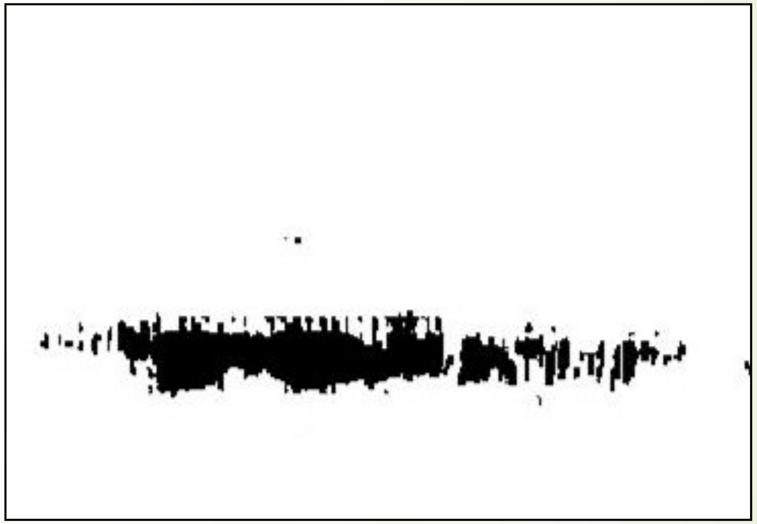
Segmentation based
on texture similarity

Texture analysis: examples

- Texture fault detection



A sample of tissue
with a defect



Defect detection basd
on texture anomaly

Texture analysis: example

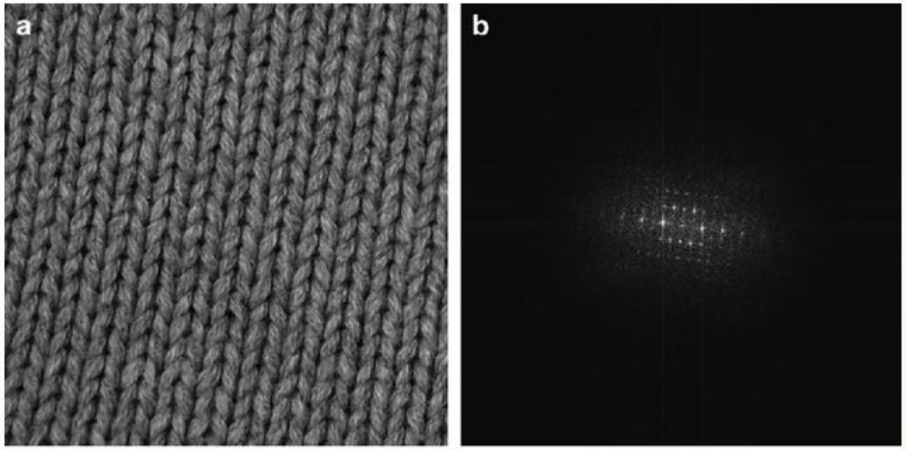
- Texture synthesis

Texture sample

Synthesized pattern with
texture sample properties

Transform-based methods

- Transforms (e.g., DFT but not only) provide information about the energy distribution of an image in the frequency domain
  - Looking at transform coefficients I can perceive if an image contains higher or lower frequencies, as well as their dominant directions
  - This is very much related to the textures present in the image, in fact, textures produce frequency peaks in specific zones of the transform



texture sample transformed sample

Source: G. Dougherty, M.A.Haidekker, "Medical Image Processing: Techniques and Applications", Springer, 2011