KMITL พระจอมเกล้าลาดกระบัง software engineering INTERNATIONAL COLLEGE

Computer Vision
13016370

Edited by:
Dr. Ukrit Watchareeruetai
International College
King Mongkut's Institute of Technology Ladkrabang

Lecture 11
Image features

- Image features
  - Color
  - Texture
  - Shape
  - Keypoints

Image features

Image features

- Human visual systems work on various types of image features.
- Each type represents different characteristics of objects/images.
- Four main types:
  - Color – Presence of brightness/color information
  - Texture – Pattern of object's surface
  - Shape – Describing the boundary of object
  - Keypoints – Distinctive locations on object

Image features: Color

Color features

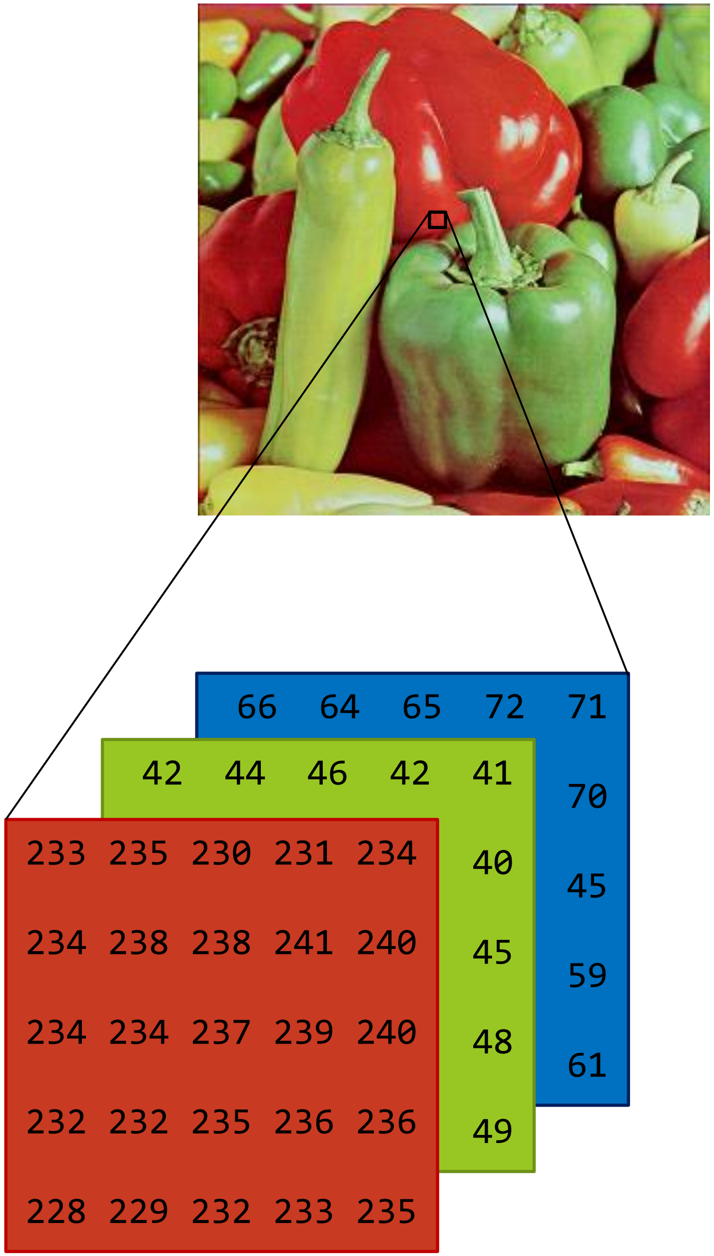
- One of the most important visual features
- Different colors are perceived due to different light reflection/absorption properties of object's surface.
- Related to the presence of color information
- Different color spaces can be used:
  - Gray-scale – monotone (represent the brightness)
  - RGB – three primary colors: red, green, and blue
  - HSV, HSI – Decouple the hue and saturation from value (related to lightness)

Grayscale

25	27	26	22	31
25	23	32	29	230
26	25	31	227	225
37	232	236	226	229
237	243	235	236	236

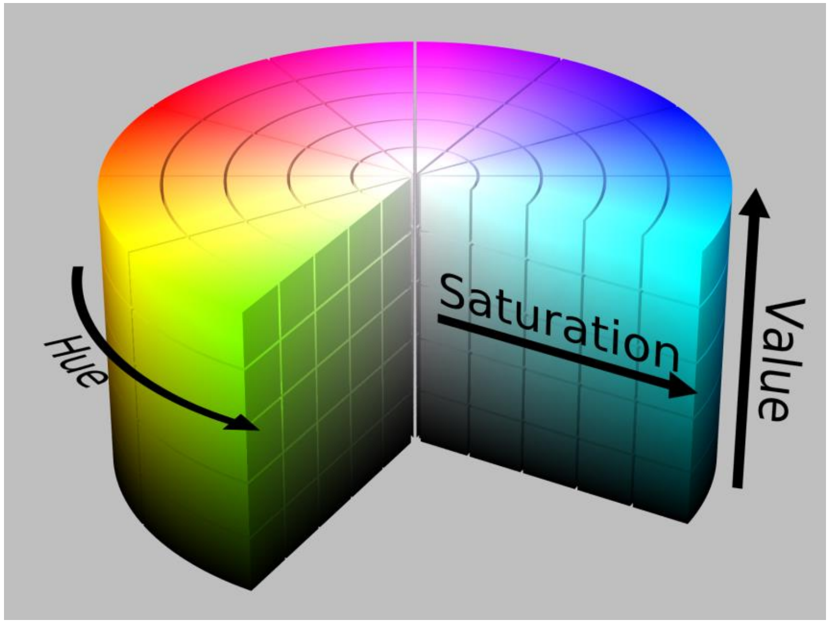
- A **grayscale image** is a 2-D array that assigns to each pixel in the array an integer value representing the intensity of that position in the image.
  - Also known as an **intensity image**
  - An 8-bit grayscale image assigns value in the range [0, 255] to each pixel.
    - **0 = black**
    - **255 = white**

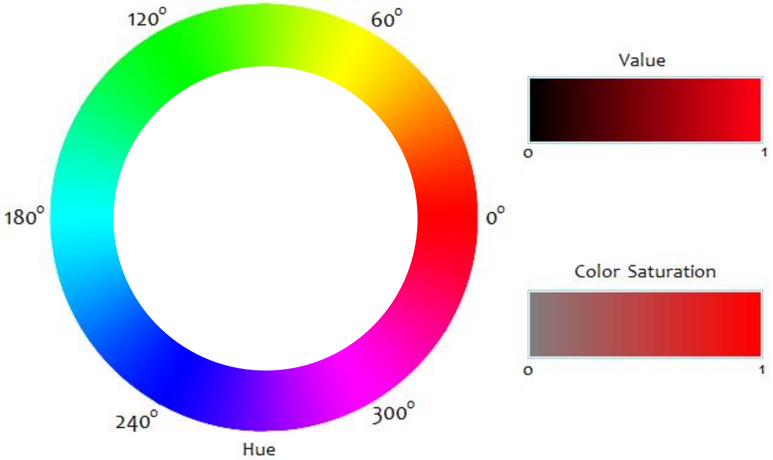
RGB color space



- In the **RGB color space**, the elements represent the **red**, **green**, and **blue** intensity of that location, respectively.
  - The value of each element is in the range [0, 255].
- Can be conceptually considered as a set of three 2-D arrays (called **bands** or **channels**)

HSV color space

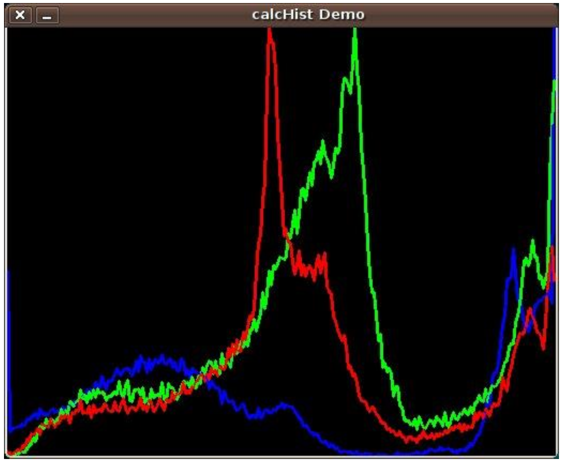




- **HSV** is another color space that can decouple color from lightness.
  - **Hue** is what most people mean by color.
    - Distinction between red and yellow
  - **Saturation** is the amount of the color that is present.
    - Distinction between red and pink
  - **Value** is strongly related to the amount of light.
    - Distinction between a dark red and light red

Color histogram

- Color histogram represents the distribution of color components in an image.
  - Calculate the histogram of each color component
  - Then concatenate to each other to form a color histogram
- Measure the frequency of each color
  - The number of times that the color appears in the image



Source: https://docs.opencv.org

Histogram matching

- Various methods to measure the similarity/distance between two given histogram $H_1(I)$ and $H_2(I)$:

Method	Formula
Correlation	$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{(H_1(I) - \bar{H}_1)^2 (H_2(I) - \bar{H}_2)^2}}$
Chi-square	$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$
Intersection	$d(H_1, H_2) = \sum_I min\left(H_1(I), H_2(I)\right)$
Bhattacharyya distance	$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H}_1 \bar{H}_2 N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$

Source: https://docs.opencv.org

cv::calcHist()

- `calcHist()` is used to calculate the histogram of an image.
  - C++: `void calcHist(const Mat* images, int nimages, const int* channels, InputArray mask, OutputArray hist, int dims, const int* histSize, const float** ranges )`
  - Python: `calcHist(images, channels, mask, histSize, range[, hist[,accumulate]]) → hist`

  - Parameters:
    - **images** – Source arrays. They all should have the same depth, CV_8U or CV_32F , and the same size. Each of them can have an arbitrary number of channels.
    - **nimages** – Number of source images.
    - **channels** – List of the dims channels used to compute the histogram. The first array channels are numerated from 0 toimages[0].channels()-1 , the second array channels are counted from images[0].channels() toimages[0].channels() + images[1].channels()-1, and so on.
    - **mask** – Optional mask. If the matrix is not empty, it must be an 8-bit array of the same size as images[i] . The non-zero mask elements mark the array elements counted in the histogram.
    - **hist** – Output histogram, which is a dense or sparse dims -dimensional array.
    - **dims** – Histogram dimensionality that must be positive and not greater than CV_MAX_DIMS (equal to 32 in the current OpenCV version).
    - **histSize** – Array of histogram sizes in each dimension.
    - **ranges** – Array of the dims arrays of the histogram bin boundaries in each dimension.

  - #include `"opencv2/imgproc/imgproc.hpp"`

Source: https://docs.opencv.org

cv::compareHist()

- `calcHist()` compares two histograms.
  - `C++: double compareHist(InputArray` **`H1`**`, InputArray` **`H2`**`, int` **`method`**`)`
  - `Python: compareHist(H1, H2, method) → retval`

  - Parameters:
    - **H1** – Histogram 1
    - **H2** – Histogram 2
    - **method** – Method to measure the distance between two histograms
      - CV_COMP_CORREL
      - CV_COMP_CHISQR
      - CV_COMP_INTERSECT
      - CV_COMP_BHATTACHARYYA

  - #include `"opencv2/imgproc/imgproc.hpp"`

Source: https://docs.opencv.org

Image features: Texture

Texture features

- A texture feature quantifies a certain characteristic of perceived texture of an image.
- Measure properties such as smoothness, coarseness, contrast
- A different texture pattern results in a different set of features.
- Important for image segmentation and object recognition

Gray-level co-occurrence matrix (GLCM)

- A co-occurrence matrix describes the probability to find a pair of pixels with gray-levels i and j at distance d in direction θ.
- Usually, four directions are considered:
  - Horizontal, diagonal, vertical, and antidiagonal (0 °, 45°, 90°, 135°)
- The number of co-occurrence matrices = the number of distances × the number of directions
- The size of each co-occurrence matrix is $L \times L$.
  - L is the number of different gray-levels in the input image.

- Co-occurrence matrices are used in a calculation of texture features.

GLCM calculation

A 5×5 image (4 gray-levels)

0	0	0	1	1
1	1	2	2	2
1	0	1	0	0
2	0	3	3	0
3	3	1	1	2

GLCM: d=1, θ=0

$$\frac{1}{40} \times \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 6 & 4 & 1 & 2 \\ 1 & 4 & 6 & 2 & 1 \\ 2 & 1 & 2 & 4 & 0 \\ 3 & 2 & 1 & 0 & 4 \end{array}$$

GLCM: d=1, θ=90

$$\frac{1}{38} \times \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 4 & 3 & 3 & 2 \\ 1 & 3 & 2 & 4 & 3 \\ 2 & 3 & 4 & 0 & 1 \\ 3 & 2 & 3 & 1 & 0 \end{array}$$

GLCM: d=2, θ=90

$$\frac{1}{28} \times \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 2 & 6 & 2 & 0 \\ 1 & 6 & 2 & 1 & 1 \\ 2 & 2 & 1 & 0 & 2 \\ 3 & 0 & 1 & 2 & 0 \end{array}$$

GLCM-based texture features

Texture feature	Formula	Description
Uniformity (energy)	$\sum_{i=0}^{L-1}\sum_{j=0}^{L-1} P(i,j)^2$	Measure the uniformity in gray-level
Contrast	$\sum_{i=0}^{L-1}\sum_{j=0}^{L-1} (i-j)^2 P(i,j)$	Measure contrast between a pixel and its neighbor over the entire image
Homogeneity	$\sum_{i=0}^{L-1}\sum_{j=0}^{L-1} \frac{P(i,j)}{1+\lvert i-j\rvert}$	Measure the spatial closeness of the distribution of elements in GLCM to the diagonal
Entropy	$-\sum_{i=0}^{L-1}\sum_{j=0}^{L-1} P(i,j)\log_2 P(i,j)$	Measure the randomness of the elements of GLCM

Source: R.C. Gonzalez and R.E. Woods., Digital Image Processing (3rd), Pearson, p.854, 2010.

Image features: Shape

Shape features

- Shape is a visual feature describing how the boundary of an object looks like.
- Representations:
  - Region-based representation
    - Binary image – 0 represents object pixel, 1 background (or vice versa)
  - Boundary-based representation
    - Contour – a sequence of 2D points along the boundary of object
    - Chain code – a sequence of code representing the relationship between two adjacent contour pixels
      - E.g., The 8-directional Freeman chain code
    - Signature – a sequence of features calculated from a contour pixel

Contour

- Image contour is a sequence of points along the boundary of an objects.

$$C(u) = (x(u), y(u))$$

  - $u = 0, 1, 2, \dots, N-1$ is the index of points on the contour.
  - N is the number of points on the contour.

cv2.findContours

- C++: `void drawContours(InputOutputArray image, InputArrayOfArrays contours, int contourIdx, const Scalar& color, int thickness=1)`
- Python: `cv2.drawContours(image, contours, contourIdx, color[,thickness]) → None`
- Parameters:
  - `image` – Destination image
  - `contours` – A set of all contours. Each is a vector of points.
  - `contourIdx` – Index of the contour to draw. If negative, all contours are drawn.
  - `color` – Color of the contours
  - `thickness` – Thickness of contours.

  - Note: For more options, please see https://docs.opencv.org.

Source: https://docs.opencv.org

cv2.drawContours

- C++: `void findContours(InputOutputArray image, OutputArrayOfArrays contours, int mode, int method)`
- Python: `cv2.findContours(image, mode, method) → contours, hierarchy`
- Parameters:
  - `image` – Single-channel 8-bit image (input)
  - `contours` – All detected contours. Each is a vector of points.
  - `hierarchy` – Optional output vector showing the relationship between contours
  - `mode` – Contour retrieval method:
    - CV_RETR_EXTERNAL – retrieves only the extreme outer contours
    - CV_RETR_LIST – retrieves all contours without hierarchy
  - `method` – Contour approximation method:
    - CV_CHAIN_APPROX_NONE – stores all the contour points
    - CV_CHAIN_APPROXSIMPLE – Horizontal, vertical, and diagonal segments are represented by their end points.
      - For example, a rectangle is represented by the four corners.

  - Note 1: For Python, replace the prefix CV_ by cv2. For example, cv2.RETR_EXTERNAL.
  - Note 2: For more options, please see https://docs.opencv.org.

Source: https://docs.opencv.org

Shape factors

Shape factors

- Useful descriptors calculated from the geometric properties of a shape
- Known as shape factors
- Can be divided into 2 main types:
  - Primary descriptors – calculated directly from the shape
    - Examples: area, perimeter, major axis, minor axis
    - Usually, invariant to translation and rotation only
  - Secondary descriptors – calculated from primary descriptors
    - Examples: aspect ratio, circularity
    - Usually, invariant to translation, rotation, and scaling

Shape descriptor	Description	Invariant property
Area (A)	Area of shape (approximately, the number of pixels in the shape)	Translation, rotation
Perimeter (P)	Boundary length (approximately, the number of pixels on the contour)	
Major axis (D_{major})	Length of the longer side of the minimum bounding box (MBB)	
Minor axis (D_{minor})	Length of the shorter side of MBB	
Diameter (D)	The longest distance between two contour points	
Radius of MBC (R_{MBC})	The radios of the minimum bounding circle (MBC)	
Aspect ratio (A_R)	Ratio of major axis to minor axis$A_R = \frac{D_{major}}{D_{minor}}$	Translation, rotation, scaling
Circularity (C)	$C = \frac{4\pi A}{P^2}$	

Moment invariants

Image moments

- The 2D moment of an image $f(x,y)$ of size $M \times N$ is defined as

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x, y)$$

  - p and q are non-negative integers (0, 1, 2, …).
  - $(p + q)$ determines the order of moment.
  - It is a particular weighted sum of pixel brightness.
    - The weights depend on the coordinate of pixels.

- This image moment can be used to describe objects.
  - Usually image segmentation is required before calculating the moments.

Image moments

- Order 0:

$$m_{00} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)$$

- Order 1:

$$m_{10} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x \times f(x, y)$$

$$m_{01} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} y \times f(x, y)$$

- Centroid:

$$(\bar{x}, \bar{y}) = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}})$$

Central moments

- The central moment μ_{pq} of order $(p+q)$ is defined as

$$\mu_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (x-\bar{x})^p (y-\bar{y})^q f(x,y)$$

  - Translation invariant
  - Computed relatively to object's centroid

- The normalized central moments:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\left(1+\frac{p+q}{2}\right)}}$$

  - Translation- and scale-invariant

Moment invariants

- Also known as Hu's invariant moments
  - Proposed by Hu in 1962
- Invariant to translation, rotation, scaling, and mirroring
- Consists of 7 terms
  - $\varphi_1 = \eta_{20} + \eta_{02}$
  - $\varphi_2 = (\eta_{20}+\eta_{02})^2 + 4\eta_{11}^2$
  - $\varphi_3 = (\eta_{30}-3\eta_{12})^2 + (3\eta_{21}-\eta_{03})^2$
  - $\varphi_4 = (\eta_{30}+\eta_{12})^2 + (\eta_{21}+\eta_{03})^2$
  - $\varphi_5 = (\eta_{30}-3\eta_{12})(\eta_{30}+\eta_{12})[(\eta_{30}+\eta_{12})^2 - 3(\eta_{21}+\eta_{03})^2] + (3\eta_{21}-\eta_{03})(\eta_{21}+\eta_{03})[3(\eta_{30}+\eta_{12})^2 - (\eta_{21}+\eta_{03})^2]$
  - $\varphi_6 = (\eta_{20}-\eta_{02})\,[(\eta_{30}+\eta_{12})^2 - (\eta_{21}+\eta_{03})^2] + 4\eta_{11}(\eta_{30}+\eta_{12})\,(\eta_{21}+\eta_{03})$
  - $\varphi_7 = (3\eta_{21}-\eta_{03})(\eta_{30}+\eta_{12})[(\eta_{30}+\eta_{12})^2 - 3(\eta_{21}+\eta_{03})^2] - (\eta_{30}-3\eta_{12})(\eta_{21}+\eta_{03})[3(\eta_{30}+\eta_{12})^2 - (\eta_{21}+\eta_{03})^2]$

cv2.moments

- C++: `Moments moments(InputArray array, bool binaryImage=false)`
- Python: `cv2.moments(array[,binaryImage])` → `retval`
- Parameters:
  - `array` – An image (single-channel), or 1D array of 2D points
  - `binaryImage` – If true, all non-zero pixels are treated as 1's.
  - `retval` – An object of `class Moments`

```
class Moments
{
public:
    Moments();
    Moments(double m00, double m10, double m01, double m20, double m11,
            double m02, double m30, double m21, double m12, double m03 );
    Moments( const CvMoments& moments );
    operator CvMoments() const;

    // spatial moments
    double  m00, m10, m01, m20, m11, m02, m30, m21, m12, m03;
    // central moments
    double  mu20, mu11, mu02, mu30, mu21, mu12, mu03;
    // central normalized moments
    double  nu20, nu11, nu02, nu30, nu21, nu12, nu03;
}
```


Source: https://docs.opencv.org

cv2.HuMoments

- C++: `Moments HuMoments(const Moments& m, OutputArray hu)`
- Python: `cv2.HuMoments(m[, hu])` → `hu`
- Parameters:
  - `m` – Input moments (computed by cv2.moments())
  - `hu` – Output Hu's moment invariants

Source: https://docs.opencv.org

Curvature scale space (CSS)

Curvature-scale space (CSS)

- A method to extract shape features from a space of curvature and scale.
  - Proposed by S. Abbasi, F. Mohktarian, and J. Kittler in 90's

- Detect distinctive points on contours at various scales
  - Use the concept of scale space
- Construct a feature called CSS image
- Invariant to translation, scaling, and easier to deal with rotation problem

Curvature

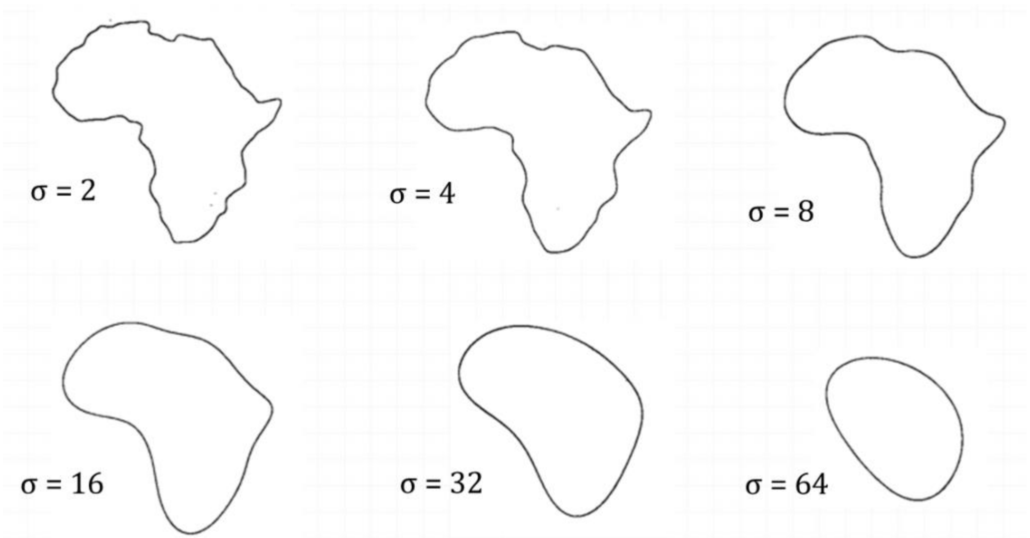
- Curvature is the amount by which a curve deviates from being straight.
  - Measure how fast a curve is changing the direction at a point
- Let $C(s) = (x(s), y(s))$ denote a contour and s the arc length.
- Curvature is then defined by:

$$k(s) = \frac{\dot{x}(s)\ddot{y}(s) - \ddot{x}(s)\dot{y}(s)}{(\dot{x}(s)^2 - \dot{y}(s)^2)^{3/2}}$$

  - Curvature $k(s)$ can be either positive, zero, or negative.

Scale space

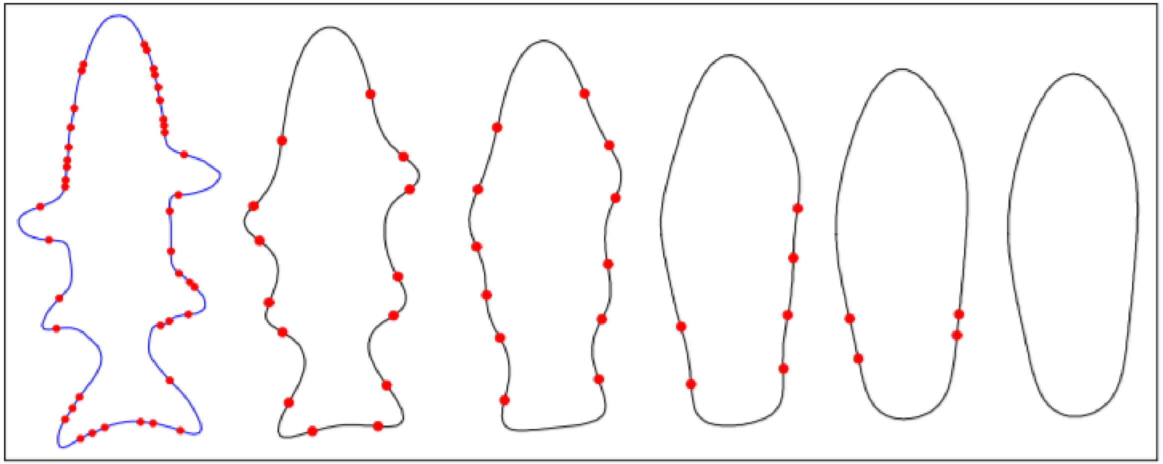
- A contour can be evolved by convoluting with 1D Gaussian function with a different scale (σ).
  - The parameter σ controls the degree of smoothness.
    - Large σ → Rough scale → Capture key structures
    - Small σ → Fine scale → Contain fine details



Source: F. Mohktarian and A.K. Machworth, “A theory of multiscale, curvature-based shape representation for planar curves,” IEEE Transaction on Pattern Analysis and Machine Intelligence, vol.14, no.8, pp.789-805, 1992.

Zero-crossing point

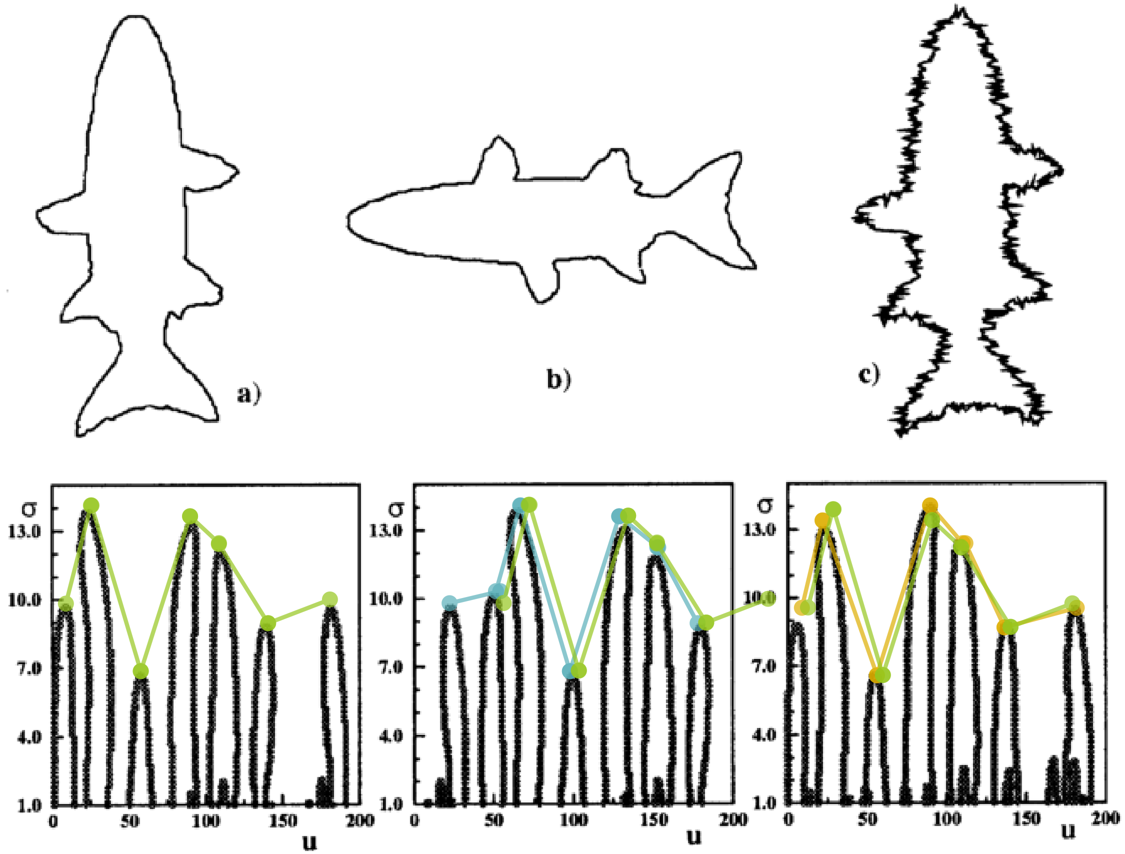
- For each scale, identify points on contour with zero curvature.
  - Known as Curvature zero-crossings.
  - Zero curvature means being straight.
  - Points that are not curve inward or outward
- The number of zero-crossing points is even (0, 2, 4, …).
  - Relatively more zero-crossing points at a fine scale
  - Two adjacent zero-crossing points merge and disappear at a higher scale.



Source: F. Mohktarian et al., "Robust and efficient shape indexing through curvature scale space," Proc. BMVC, pp.53-62, 1996.

CSS image

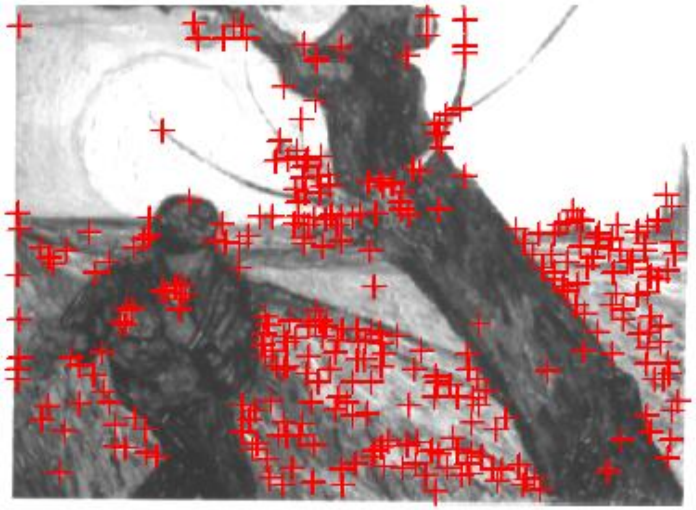
- Plotting the location of all keypoints in all scales of contour, resulting in CSS image.



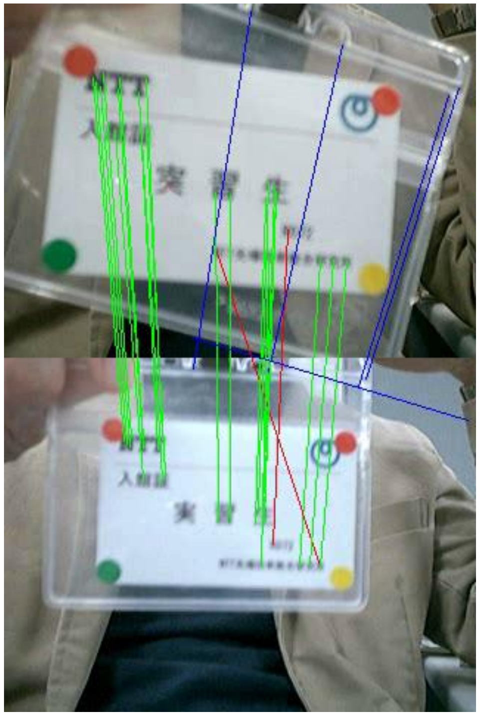
Source: S. Abbasi et al., “Curvature scale space image in shape similarity retrieval,” Multimedia Systems, vol.7, pp.467-476, 1999.

Image features: Keypoints

Keypoints



Source: Schmid et al., 2000



Source: Watchareeruetai et al., 2011

- A **keypoint** or **interest point** is a point in an image with distinct characteristics.
  - A set of keypoints can describe the structure of object being considered.
  - Can be used for image indexing, stereo matching, object recognition
- Advantages:
  - Robust against partial occlusion
  - Require no segmentation process

Scale-invariant feature transform (SIFT)

SIFT

- SIFT: Scale-Invariant Feature Transform
- Proposed by David Lowe in 1999
  - D.G. Lowe, “Object recognition from local scale-invariant features,” Proc. of 7th International Conference on Computer Vision (ICCV), 1999.
  - D.G. Lowe, “Distinctive image features from scale-invariant keypoints,” International Journal on Computer Vision (IJCV), vol.60, no.2, pp.91-110, 2004.
- Key benefits:
  - Highly distinctive keypoints
  - Efficient to compute
  - Robust against scale and rotation changes
  - Partially tolerant to illumination and affine geometry changes
- Divided into 2 main processes:
  - Keypoint detector
  - Keypoint descriptor

SIFT keypoint detector

- Scale-space images
- Generate images with different scales using Gaussian filters:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

  - $*$ - Convolution operator
  - $I(x, y)$ - Input image
  - $G(x, y, \sigma)$ - 2D Gaussian filter
  - The parameter σ controls the degree of smoothness.
    - Large σ → Rough scale → Capture key structures
    - Small σ → Fine scale → Contain fine details

SIFT keypoint detector

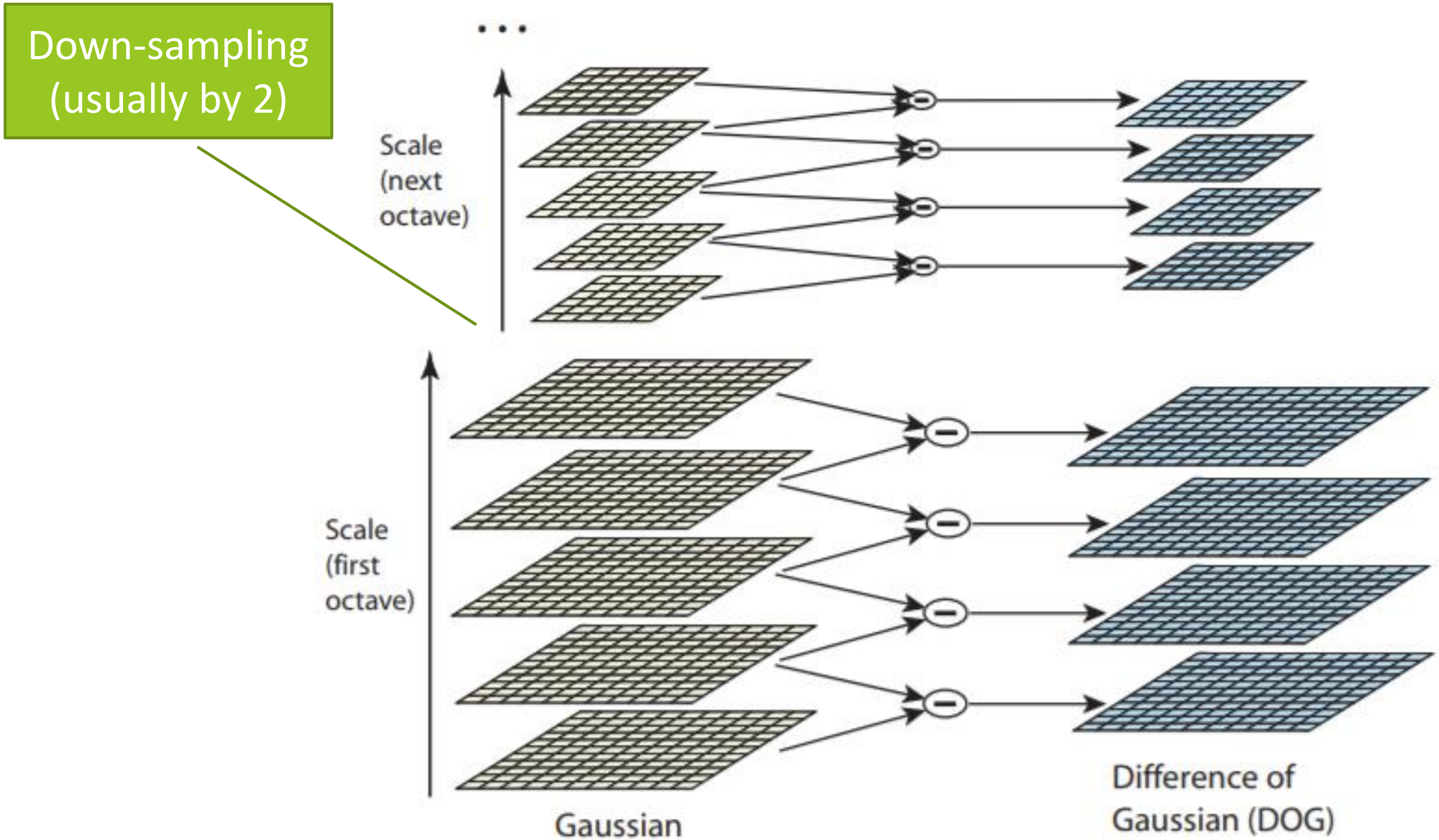
- Difference-of-Gaussian (DoG)
  - Subtract two Gaussian functions with different scales (σ)
  - Band-pass filter
    - Detect features corresponding to its frequency band

- DoG image:

$$\begin{aligned} D(x, y, \sigma) &= \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned}$$

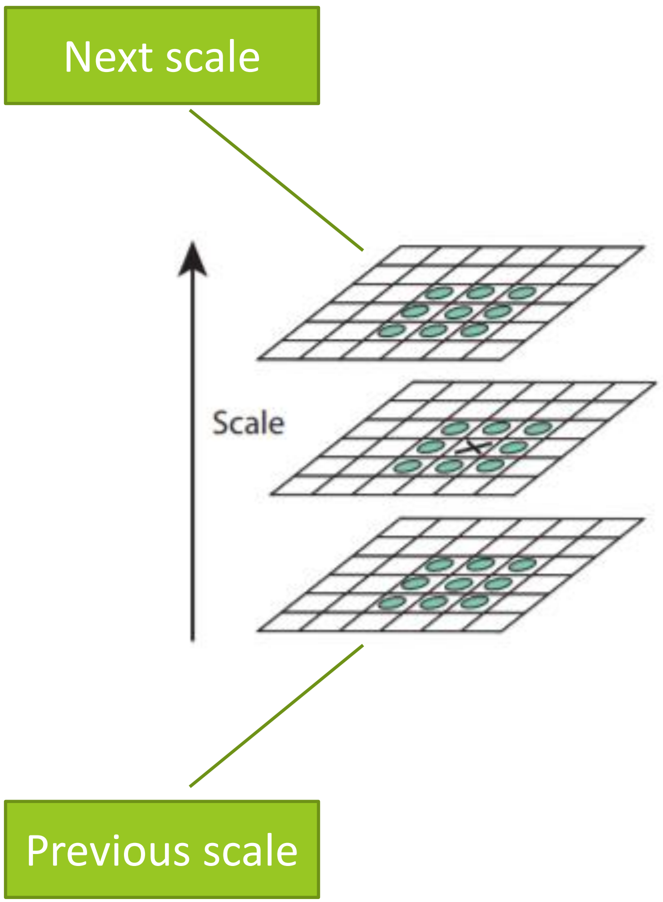
  - k – multiplicative factor

SIFT keypoint detector



Source: D.G. Lowe, "Distinctive image features from scale-invariant keypoints," IJCV, vol.60, no.2, pp.91-110, November 2004.

SIFT keypoint detector



- Scale-space extrema detection
  - Search over all scales and locations
    - Compare each pixel X with its 26 neighbors ($3 \times 3 \times 3$ space)
      - 8 neighbors from the same scale
      - 9 neighbors from the next scale
      - 9 neighbors from the previous scale
    - Check if X is an extremum
      - Maximum: X is larger than all 26 neighbors
      - Minimum: X is smaller than all 26 neighbors

Source: D.G. Lowe, “Distinctive image features from scale-invariant keypoints,” IJCV, vol.60, no.2, pp.91-110, November 2004.

Keypoint properties

- Each keypoint is associated by 4 main properties:
  - (x, y) – The location in the image
  - σ – The scale at which it is detected
  - θ – The orientation of the gradient vector:

$$\theta(x, y) = tan^{-1}\left(\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}\right)$$

  - m – The magnitude of the gradient vector:

$$m(x, y) = \sqrt{\left(L(x+1, y) - L(x-1, y)\right)^2 + \left(L(x, y+1) - L(x, y-1)\right)^2}$$

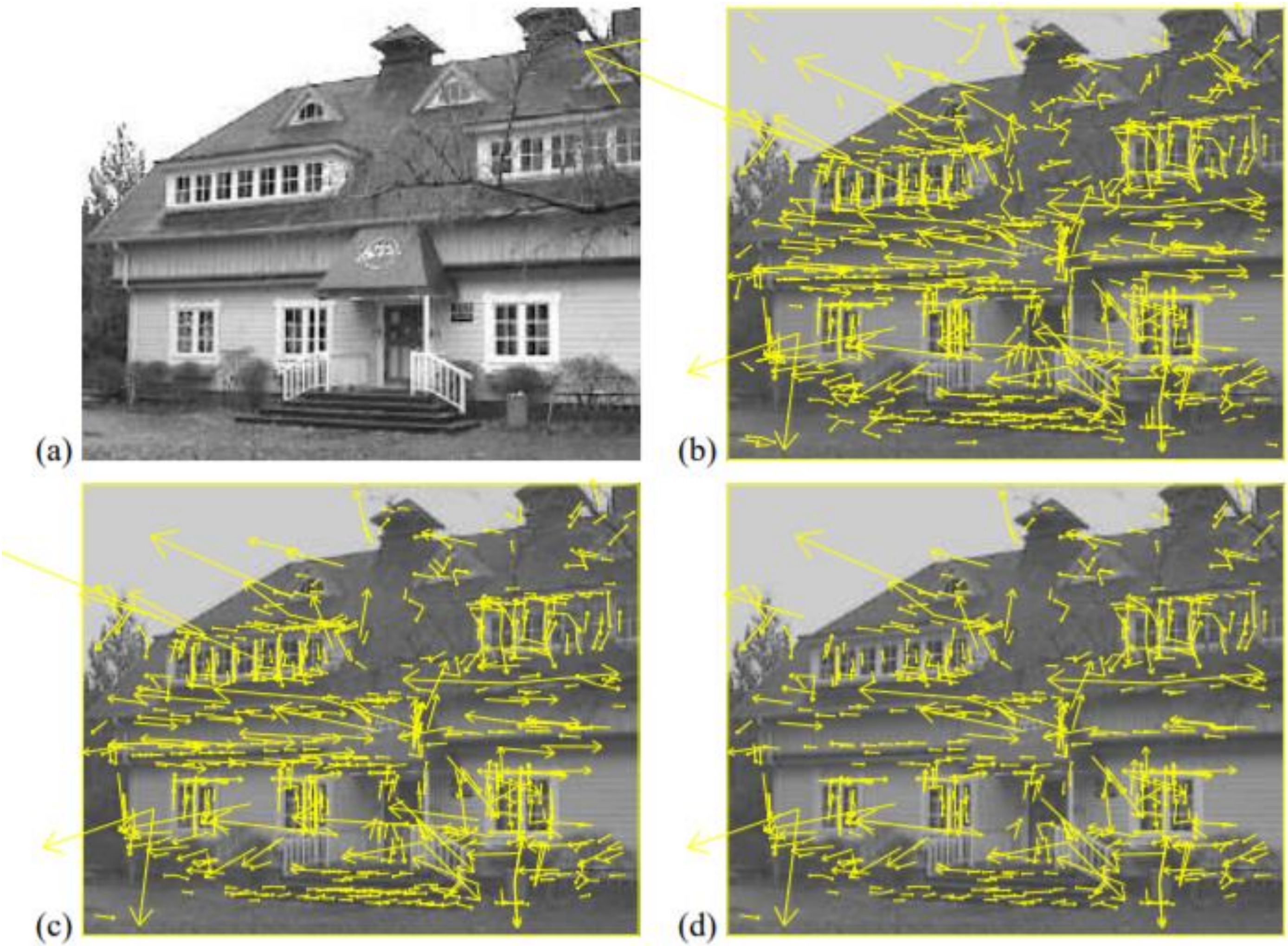
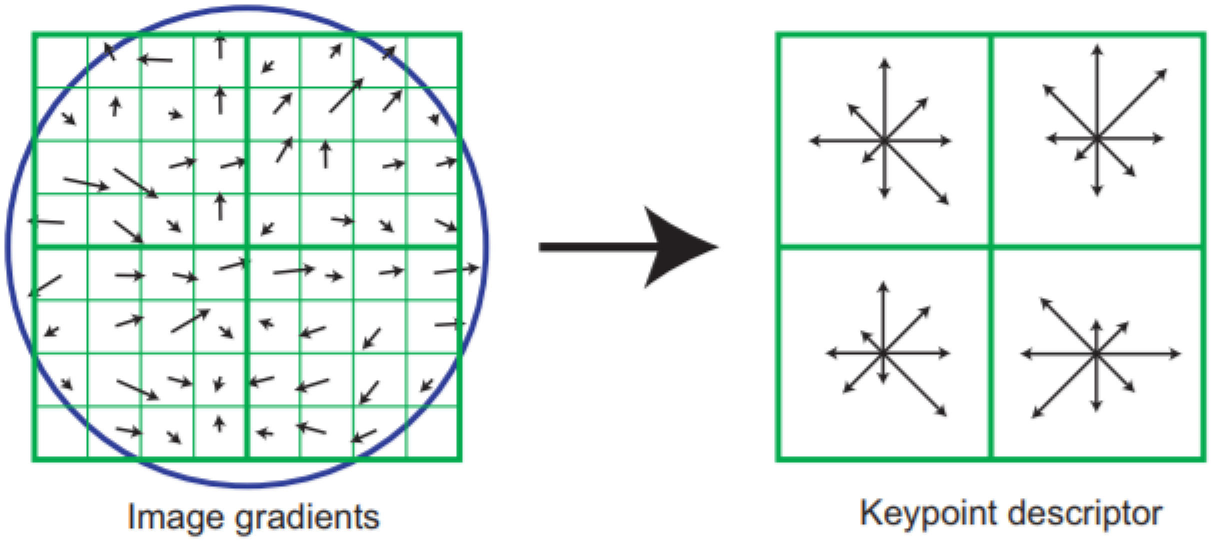


Figure 5: This figure shows the stages of keypoint selection. (a) The 233x189 pixel original image. (b) The initial 832 keypoints locations at maxima and minima of the difference-of-Gaussian function. Keypoints are displayed as vectors indicating scale, orientation, and location. (c) After applying a threshold on minimum contrast, 729 keypoints remain. (d) The final 536 keypoints that remain following an additional threshold on ratio of principal curvatures.

Source: D.G. Lowe, “Distinctive image features from scale-invariant keypoints,” IJCV, vol.60, no.2, pp.91-110, November 2004.

SIFT descriptor

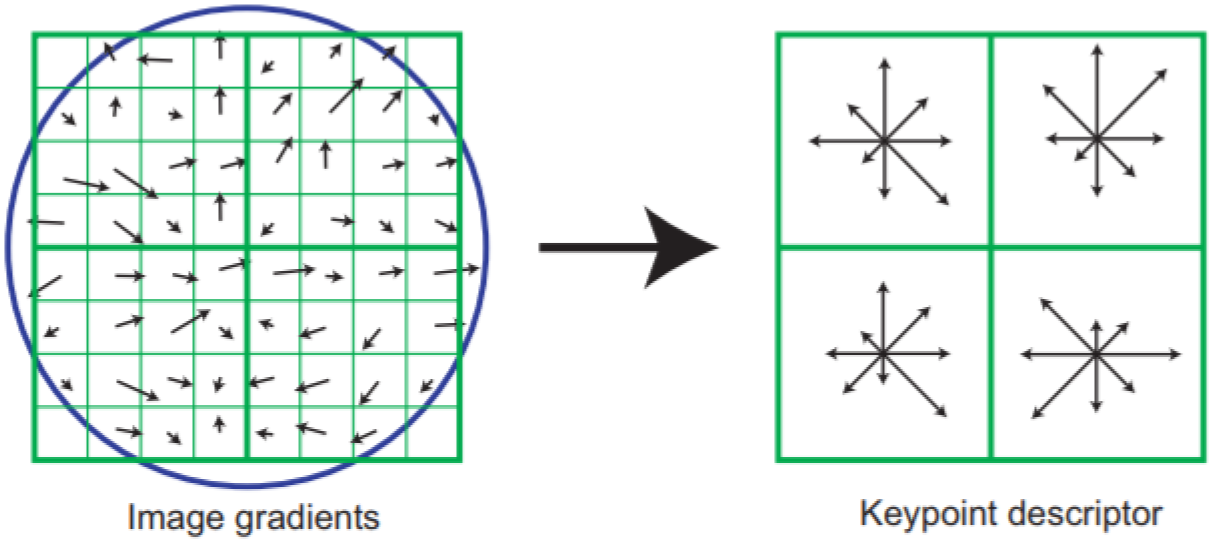
- Calculated from local area of size 16×16 pixels
  - Compute the gradient vector for each pixel
    - Rotate each gradient vector by θ (the orientation of the keypoint)
      - To make it rotation invariance
    - Gaussian weighing function with σ equal to one-half the width
      - To avoid changes caused by the position of window



Source: D.G. Lowe, "Distinctive image features from scale-invariant keypoints," IJCV, vol.60, no.2, pp.91-110, November 2004.

SIFT descriptor

- Divide the window into 16 4×4 sub-regions
  - Construct *histogram-of-gradient* in each sub-region
    - 8 bins – each corresponds to one direction (0, 45, 90, 135, 180, 225, 270, and 315°)
  - Generate 16×8 = 128-dimensional feature vector describing the keypoint



Source: D.G. Lowe, "Distinctive image features from scale-invariant keypoints," IJCV, vol.60, no.2, pp.91-110, November 2004.

References

References

- Rafael C. Gonzalez and Richard E. Woods, Digital Image Processing (3rd Edition), Prentice Hall, 2010.
- J.C. Russ, The Image Processing Handbook (6th ed.), CRC Press, 2011.
- C. Solomon and T. Breckon, Fundamentals of Digital Image Processing: A Practical Approach With Examples in MATLAB, Wiley-Blackwell, 2011.
- A. McAndrew, An Introduction to Digital Image Processing with MATLAB, Course Technology, 2004.
- D.G. Lowe, “Distinctive image features from scale-invariant keypoints,” IJCV, vol.60, no.2, pp.91-110, November 2004.
- F. Mohktarian and A.K. Machworth, “A theory of multiscale, curvature-based shape representation for planar curves,” IEEE Transaction on Pattern Analysis and Machine Intelligence, vol.14, no.8, pp.789-805, 1992.
- F. Mohktarian et al., “Robust and efficient shape indexing through curvature scale space,” Proc. BMVC, pp.53-62, 1996.
- S. Abbasi et al., “Curvature scale space image in shape similarity retrieval,” Multimedia Systems, vol.7, pp.467-476, 1999.
- https://docs.opencv.org
- https://en.wikipedia.org