ECE5554 – Computer Vision
Lecture 4b – Corner Detection and SIFT

Creed Jones, PhD

BRADLEY DEPARTMENT
OF ELECTRICAL & COMPUTER ENGINEERING

VIRGINIA TECH.

Today’s Objectives

Corner detection

- Concept
- Harris corner detection
- Shi-Tomasi corner detection
- Förstner corner detection

Keypoint Detection

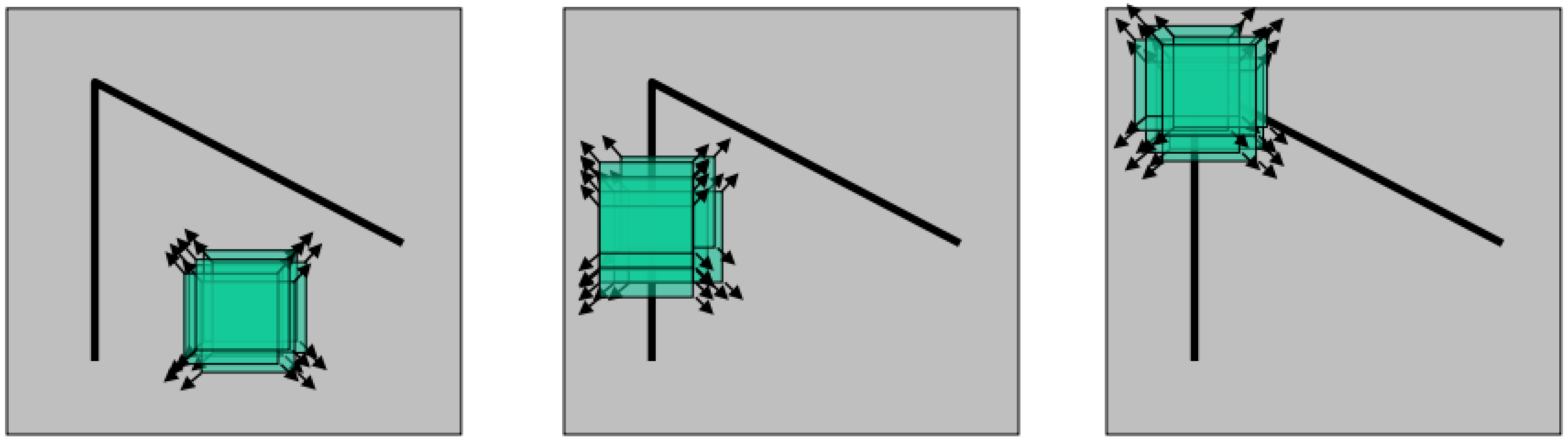
- Invariant descriptors
- SIFT method

CORNER DETECTION

Corner detection is an important function for locating the boundaries of objects in an image

- A *corner* is a location in the image where the gradient is high in multiple directions
- Corners are vertices of the outside of an object
- Knowledge of the locations of corners are especially useful for:
  - object tracking – in an image sequence
  - image understanding – what is the outline of the object
  - compact representation – an image of a rectangle can be accurately represented by four corners
  - *correspondence* between images for 3D processing
- Corner detection is often called *keypoint detection*

The Harris corner detector (or Plessey feature point detector) locates points where the spatial gradient is large in more than a single direction



"flat" region:
no change in
all directions

"edge":
no change along
the edge direction

"corner":
significant change
in all directions

The Harris corner detector finds points with significant gradient components in two orthogonal directions

1. Find approximations of the X and Y gradient components:
   $I_x = \frac{\partial I}{\partial x}(u, v)$ and $I_y = \frac{\partial I}{\partial y}(u, v)$; the Sobel kernels are suitable for this

The Harris corner detector finds points with significant gradient components in two orthogonal directions

1. Find approximations of the X and Y gradient components:

   $I_x = \frac{\partial I}{\partial x}(u, v)$ and $I_y = \frac{\partial I}{\partial y}(u, v)$; the Sobel kernels are suitable for this

2. Form the products and crossproducts of the gradients – these form the *local structure matrix M:*

   $M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$ where each component is an individual image

The Harris corner detector finds points with significant gradient components in two orthogonal directions

1. Find approximations of the X and Y gradient components:
   $I_x = \frac{\partial I}{\partial x}(u,v)$ and $I_y = \frac{\partial I}{\partial y}(u,v)$; the Sobel kernels are suitable for this
2. Form the products and crossproducts of the gradients – these form the *local structure matrix M:*
   $M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$ where each component is an individual image
3. Apply a Gaussian smoothing filter to each component image individually; the resulting matrix is called $\bar{M}$

The Harris corner detector finds points with significant gradient components in two orthogonal directions

1. Find approximations of the X and Y gradient components:
   $I_x = \frac{\partial I}{\partial x}(u,v)$ and $I_y = \frac{\partial I}{\partial y}(u,v)$; the Sobel kernels are suitable for this
2. Form the products and crossproducts of the gradients – these form the *local structure matrix M:*
   $M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$ where each component is an individual image
3. Apply a Gaussian smoothing filter to each component image individually; the resulting matrix is called $\bar{M}$
4. Define a corner strength function as:

$$Q(u,v) = \det\left(\bar{M}(u,v)\right) - \alpha\left(trace\left(\bar{M}(u,v)\right)\right)^2$$

where $\alpha \approx 0.05$

The Harris corner detector finds points with significant gradient components in two orthogonal directions

1. Find approximations of the X and Y gradient components:

   $I_x = \frac{\partial I}{\partial x}(u,v)$ and $I_y = \frac{\partial I}{\partial y}(u,v)$; the Sobel kernels are suitable for this

2. Form the products and crossproducts of the gradients – these form the *local structure matrix M:*

   $M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$ where each component is an individual image

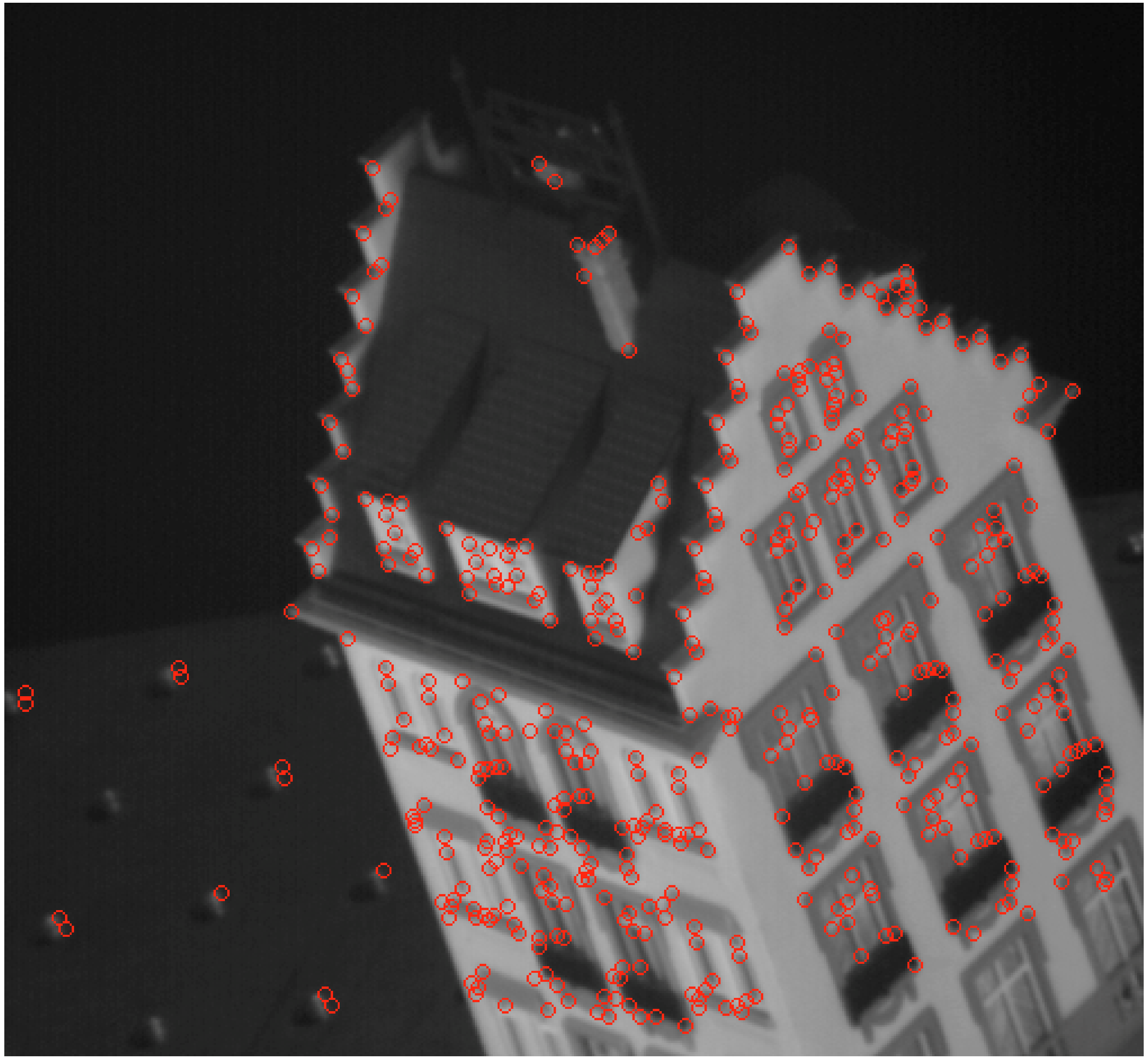
3. Apply a Gaussian smoothing filter to each component image individually; the resulting matrix is called $\bar{M}$

4. Define a corner strength function as:

   $$Q(u,v) = \det\left(\bar{M}(u,v)\right) - \alpha\left(trace\left(\bar{M}(u,v)\right)\right)^2$$

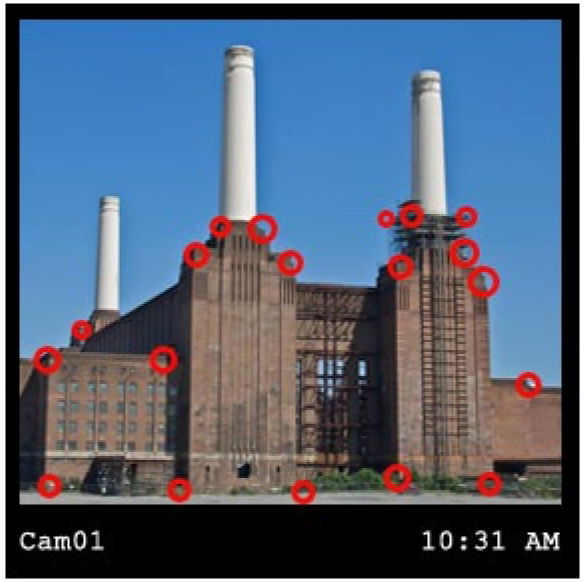
   where $\alpha \approx 0.05$

5. Maxima of the resulting image are possible corner points



Corner points of an image can be used to locate subsequent operations, identify objects, perform measurements, determine image contents…

A: Original image



B: Detected image

The Shi-Tomasi corner detector evaluates the eigenvalues of the Harris matrix

1. Find approximations of the X and Y gradient components:
   $I_x = \frac{\partial I}{\partial x}(u,v)$ and $I_y = \frac{\partial I}{\partial y}(u,v)$; the Sobel kernels are suitable for this
2. Form the products and crossproducts of the gradients – these form the *local structure matrix M:*
   $M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$ where each component is an individual image
3. Apply a Gaussian smoothing filter to each component image individually; the resulting matrix is called $\bar{M}$
4. Define a corner strength function as:
   $Q(u,v) = \min(eig_1(\bar{M}), eig_1(\bar{M}))$
5. Maxima of the resulting image are possible corner points

Quick eigenvalue/eigenvector review

The **eigenvectors** of a matrix **A** are the vectors **x** that satisfy $Ax = \lambda x$

The scalar λ is the **eigenvalue** corresponding to **x**

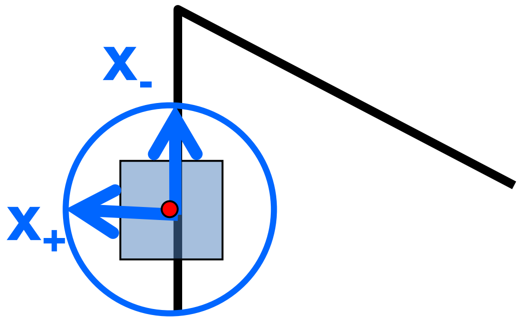
- The eigenvalues are found by solving $det(A - \lambda I) = 0$
- In our 2D case, ***A*** = ***H*** is a 2x2 matrix, so we have $det\begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} \end{bmatrix} = 0$
- The solution is $\lambda = \frac{1}{2}\left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2}\right]$

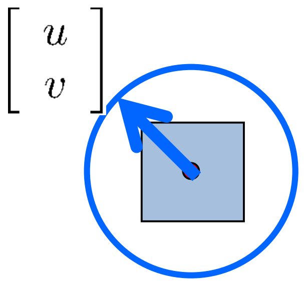
Once you know λ, you find **x** by solving $\begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = 0$

Corner detection: the math

This can be rewritten as:

$$E(u,v) = \sum_{x,y \in W} [u \quad v] \underbrace{\begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

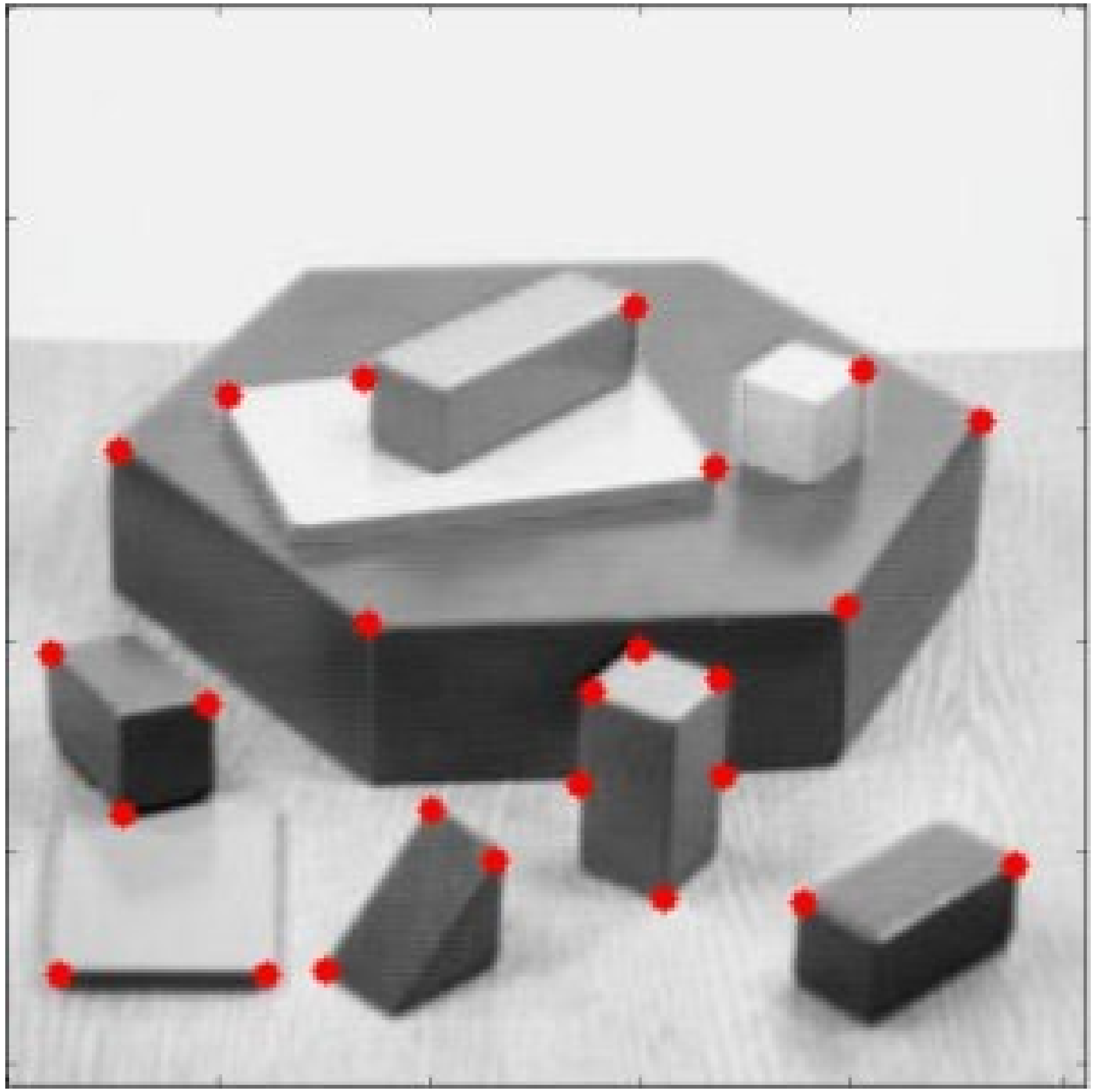




Eigenvalues and eigenvectors of H

- Define shifts with the smallest and largest change (E value)
- x_+ = direction of **largest** increase in E.
- λ_+ = amount of increase in direction x_+
- x_- = direction of **smallest** increase in E.
- λ- = amount of increase in direction x_+

$$Hx_+ = \lambda_+ x_+$$
$$Hx_- = \lambda_- x_-$$



```
import numpy as np
import scipy.interpolate as interp
import cv2

RED = (0, 0, 255)
GREEN = (0, 255, 0)

# Load an image, in gray for processing and in color for annotation
filename = "stonehenge.png"
dispImg = cv2.imread('C:\\Data\\' + filename)
img = cv2.cvtColor(dispImg, cv2.COLOR_BGR2GRAY)

# Find corners using OpenCV implementation of Shi-Tomasi
corners = cv2.goodFeaturesToTrack(img,maxCorners=100, \
         qualityLevel=0.1, minDistance=10)
corners = np.int0(corners)
for i in corners:
    x,y = i.ravel()
    cv2.circle(dispImg,(x,y),3,GREEN,-1)

cv2.imwrite('C:\\Data\\dispImg.png', dispImg)
```

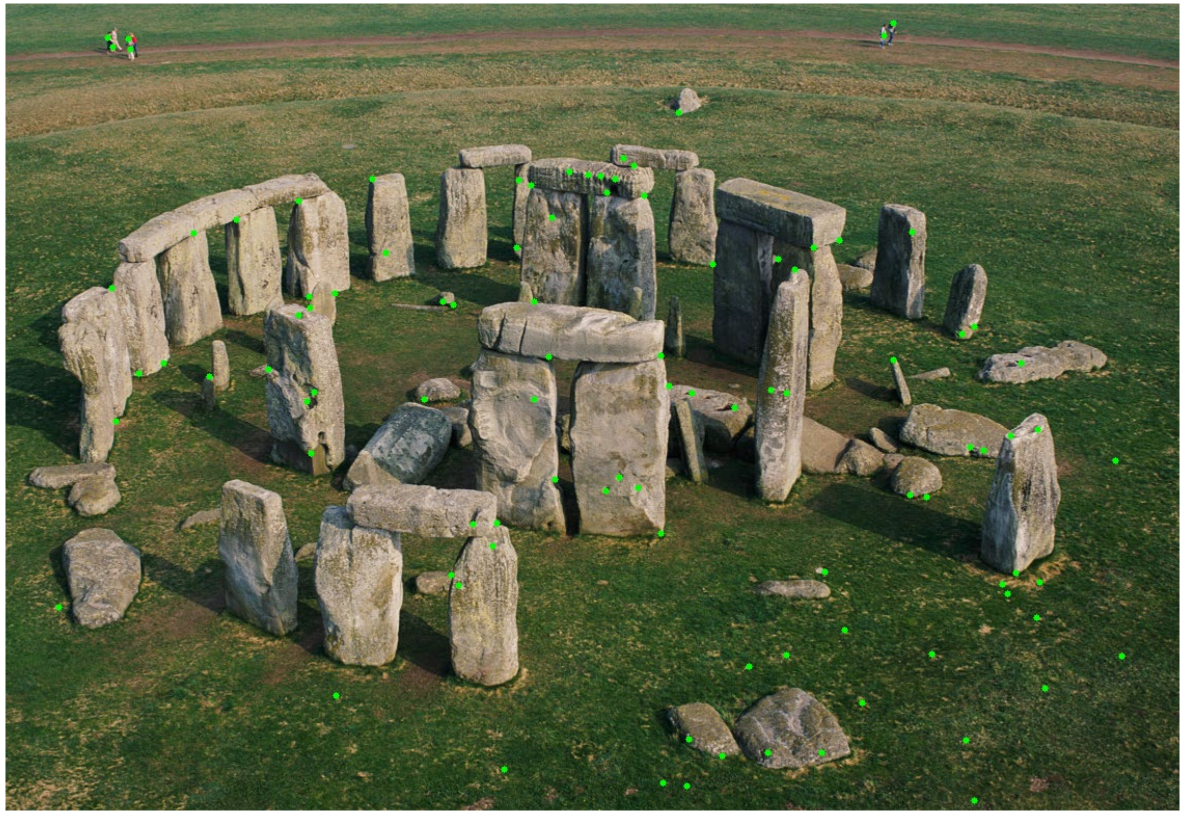
```
import numpy as np
import scipy.interpolate as interp
import cv2

RED = (0, 0, 255)
GREEN = (0, 255, 0)

# Load an image, in gray for processing and in color for annotation
filename = "stonehenge.png"
dispImg = cv2.imread('C:\\Data\\' + filename)
img = cv2.cvtColor(dispImg, cv2.COLOR_BGR2GRAY)

# Find corners using OpenCV implementation of Shi-Tomasi
corners = cv2.goodFeaturesToTrack(img,maxCorners=100, \
            qualityLevel=0.1, minDistance=10)
corners = np.int0(corners)
for i in corners:
    x,y = i.ravel()
    cv2.circle(dispImg,(x,y),3,GREEN,-1)

cv2.imwrite('C:\\Data\\dispImg.png', dispImg)
```



cornerSubPix
Refines the corner locations.

Python: cv.FindCornerSubPix(image, corners, win, zero_zone, criteria) → corners

Parameters:

image – Input image.

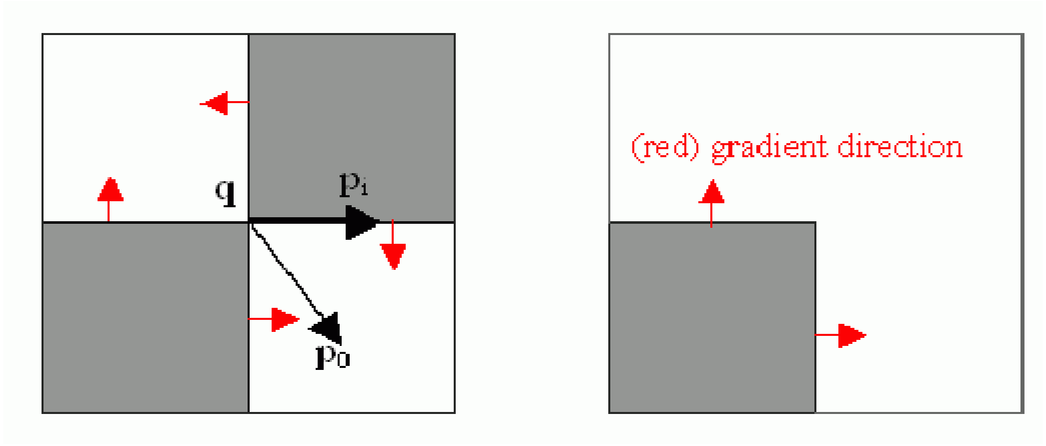
corners – Initial coordinates of the input corners and refined coordinates provided for output.

winSize – Half of the side length of the search window. For example, if winSize=Size(5,5) , then a 5*2+1 \times 5*2+1 = 11 \times 11 search window is used.

zeroZone – Half of the size of the dead region in the middle of the search zone over which the summation in the formula below is not done. It is used sometimes to avoid possible singularities of the autocorrelation matrix. The value of (-1,-1) indicates that there is no such a size.

criteria – Criteria for termination of the iterative process of corner refinement. That is, the process of corner position refinement stops either after criteria.maxCount iterations or when the corner position moves by less than criteria.epsilon on some iteration.

The function iterates to find the sub-pixel accurate location of corners or radial saddle points, as shown on the figure below.



https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html

Förstner corner detection is also based on the local structure matrix

1. Find approximations of the X and Y gradient components:
   $I_x = \frac{\partial I}{\partial x}(u,v)$ and $I_y = \frac{\partial I}{\partial y}(u,v)$; the Sobel kernels are suitable for this
2. Form the products and crossproducts of the gradients – these form the *local structure matrix M:*
   $M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$ where each component is an individual image
3. Apply a Gaussian smoothing filter to each component image individually; the resulting matrix is called $\bar{M}$
4. Find the eigenvalues of $\bar{M}^{-1}$: call them λ_1 and λ_2
5. Find the size $w = \frac{\lambda_1 \cdot \lambda_2}{\lambda_1 + \lambda_2}$ and roundness $q = 1 - \left(\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2}\right)^2$ of the error ellipses
6. Compute thresholds $w_{min} = 2\alpha\bar{w}$ and $q_{min} = \alpha$, where $\alpha \in [0.5 - 0.75]$ and $\bar{w}$ is the average over the entire image
7. Corner points are located where $w > w_{min}$ and $q > q_{min}$

INTRODUCTION TO KEYPOINTS AND THE SCALE-INVARIANT FEATURE TRANSFORM (SIFT)

There are alternatives to NGC when matching in grayscale images; the preference often depends on the application requirements and image detail level

- Other slightly different correlation coefficient formulae have been used
  - NGC uses the L_2 norm – can use others such as L_∞
  - Entropy-based similarity measures are sometimes used: $diff \log(diff)$

- Recognition using object moments, as we will see

- New methods have been developed – such as SIFT

Transformational Invariance is a desired property of a feature description

Suppose we are comparing two images I_1 and I_2

- I_2 may be a transformed version of I_1
- What kinds of transformations are we likely to encounter in practice?

We'd like to find the same features regardless of the transformation

- This is called transformational ***invariance***
- Most feature methods are designed to be invariant to
  - Translation, 2D rotation, scale
- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transformations (some are fully affine invariant)
  - Limited illumination/contrast changes

How to achieve invariance

Need both of the following:

1. Make sure your detector is invariant
   - Harris is invariant to translation and rotation
   - Scale is trickier
     - common approach is to detect features at many scales using a Gaussian pyramid (e.g., MOPS)
     - More sophisticated methods find "the best scale" to represent each feature (e.g., SIFT)
2. Design an invariant feature *descriptor*
   - A descriptor captures the information in a region around the detected feature point
   - The simplest descriptor: a square window of pixels
     - What's this invariant to?
   - Let's look at some better approaches…

Rotation invariance for feature descriptors

Find dominant orientation of the image patch

- This is given by $\mathbf{x}_+$, the eigenvector of $\mathbf{H}$ corresponding to λ_+
  - λ_+ is the *larger* eigenvalue
- Rotate the patch according to this angle

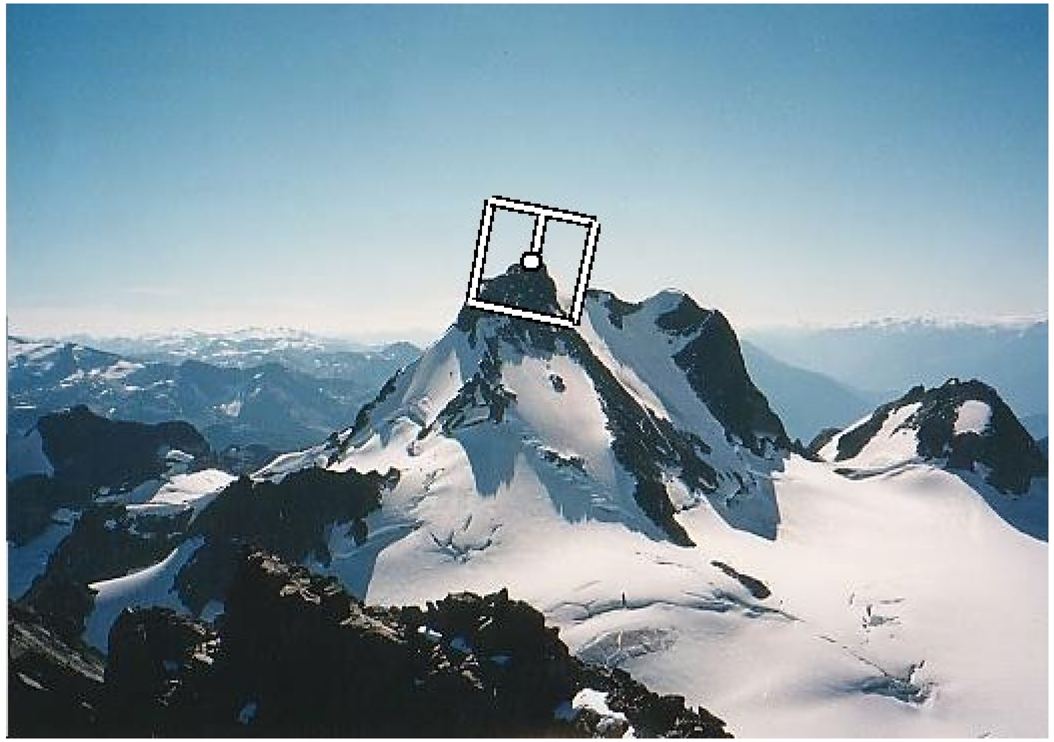
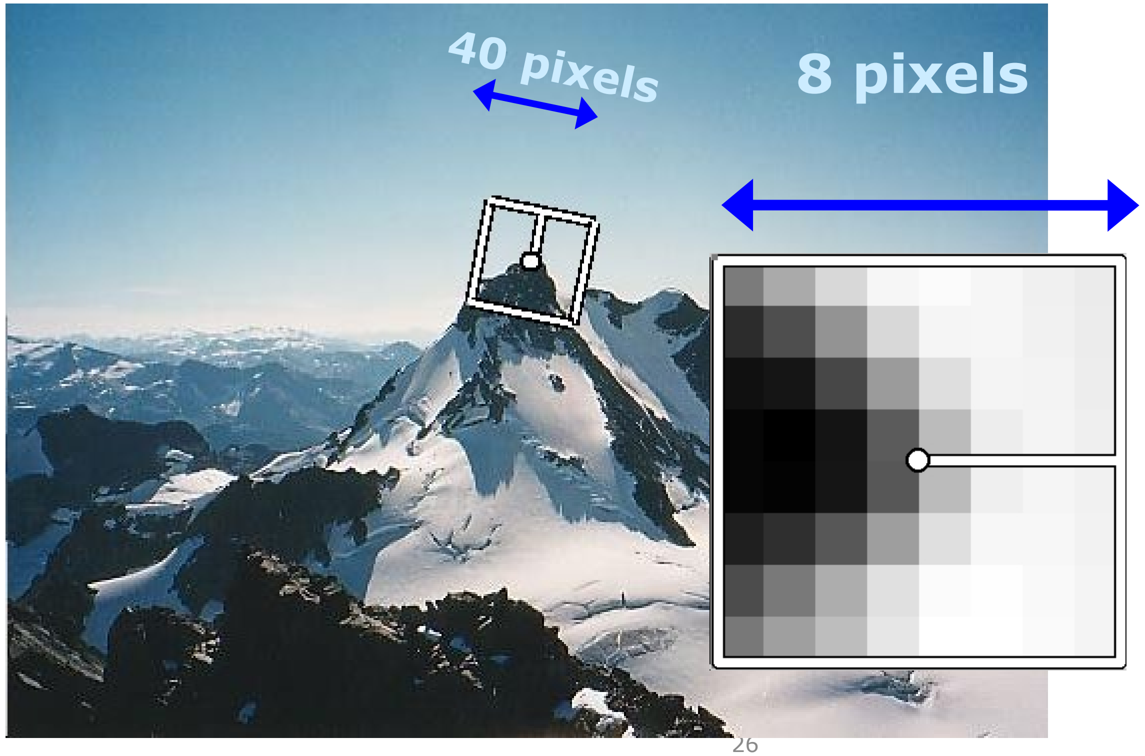


Figure by Matthew Brown

Multiscale Oriented PatcheS descriptor

Take 40x40 square window around detected feature

- Scale to 1/5 size (using prefiltering)
- Rotate to horizontal
- Sample 8x8 square window centered at feature
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window



Adapted from slide by Matthew Brown

Detections at multiple scales

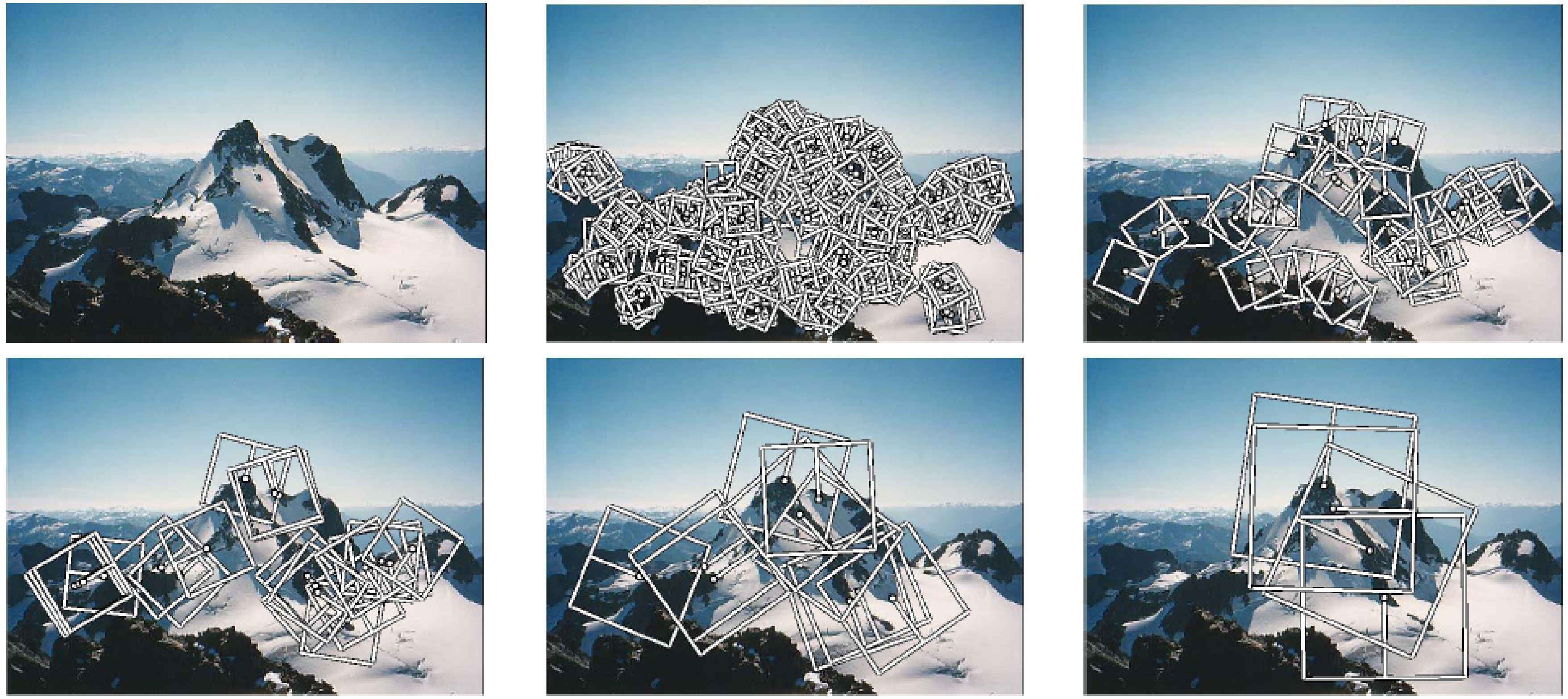
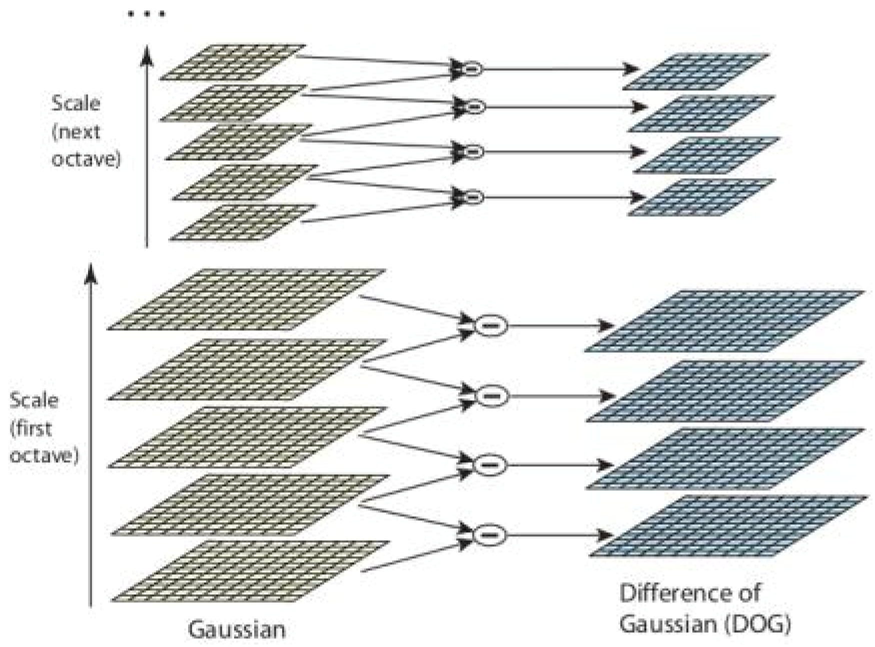


Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.

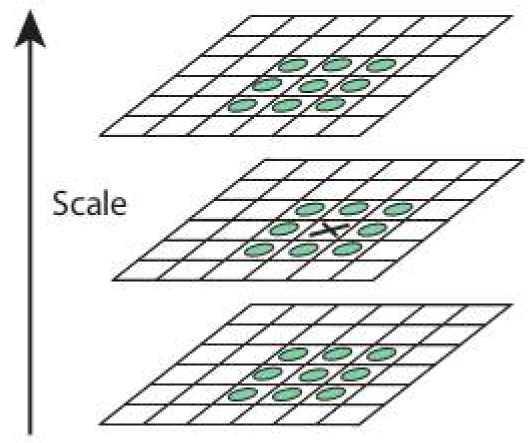
David Lowe at the University of British Columbia devised the Scale-Invariant Feature Transform (SIFT) which can be used for object matching

- First, the Difference of Gaussian (DoG) images are formed by subtracting images filtered with Gaussian kernels having different widths – generally, the σ vary by $\sqrt{2}$

- This is done for each image in a Gaussian multiscale pyramid



Descriptors or *keypoints* are defined as local maxima within the 3D space of images and scales; these keypoints are characteristic and quite stable



- Interpolation is used to more precisely locate the maximum
- Maxima caused by edge points are suppressed
- Keypoint orientations are encoded as related to local image orientation
  - so the orientation information is not sensitive to template rotation

Scale Invariant Feature Transform

Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations

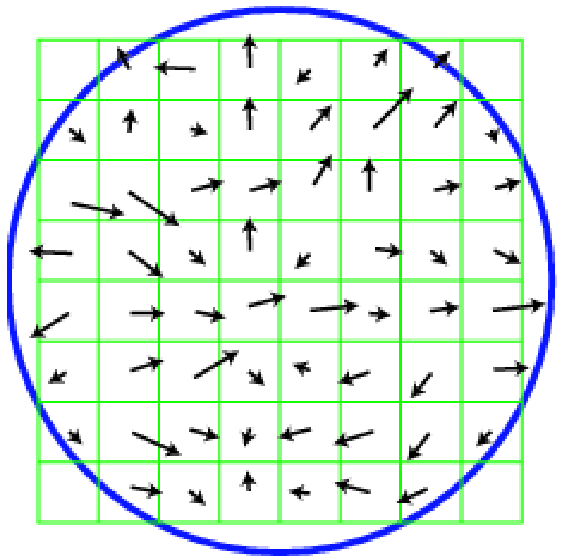
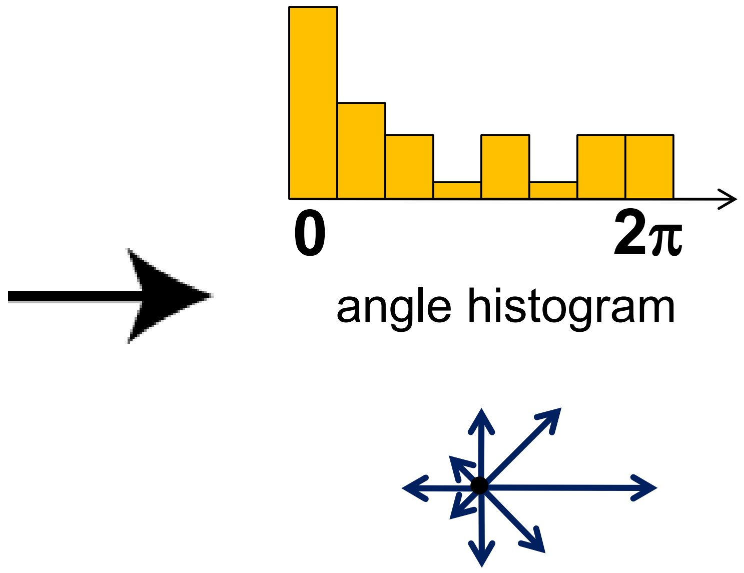


Image gradients

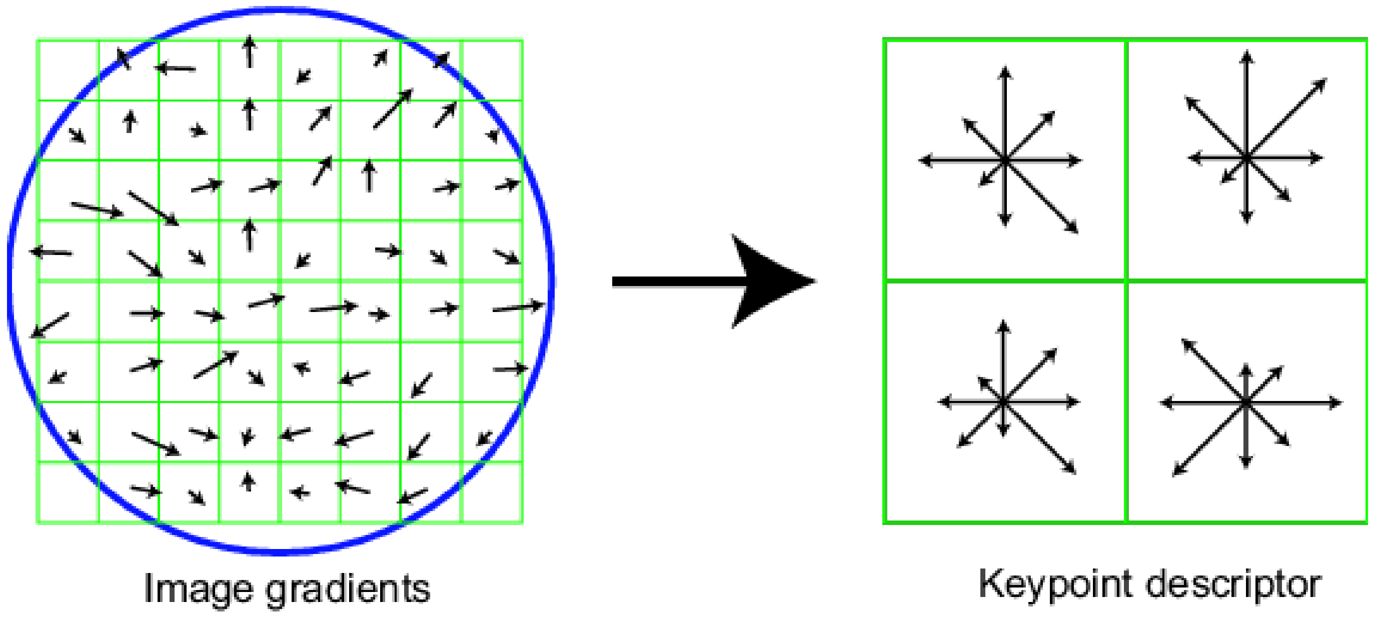


Adapted from slide by David Lowe

SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor

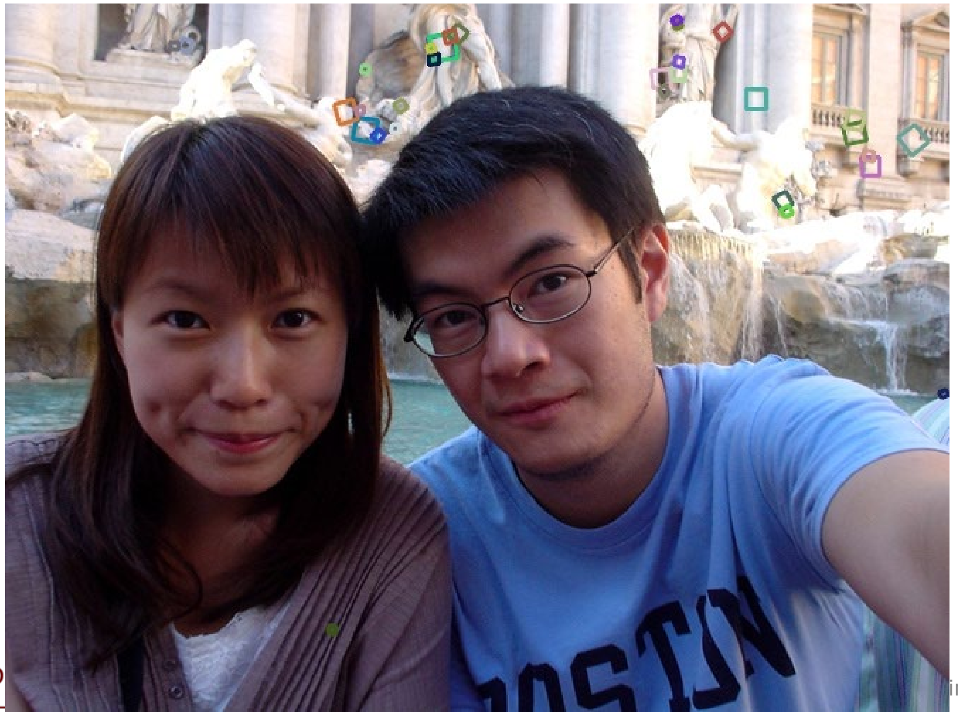
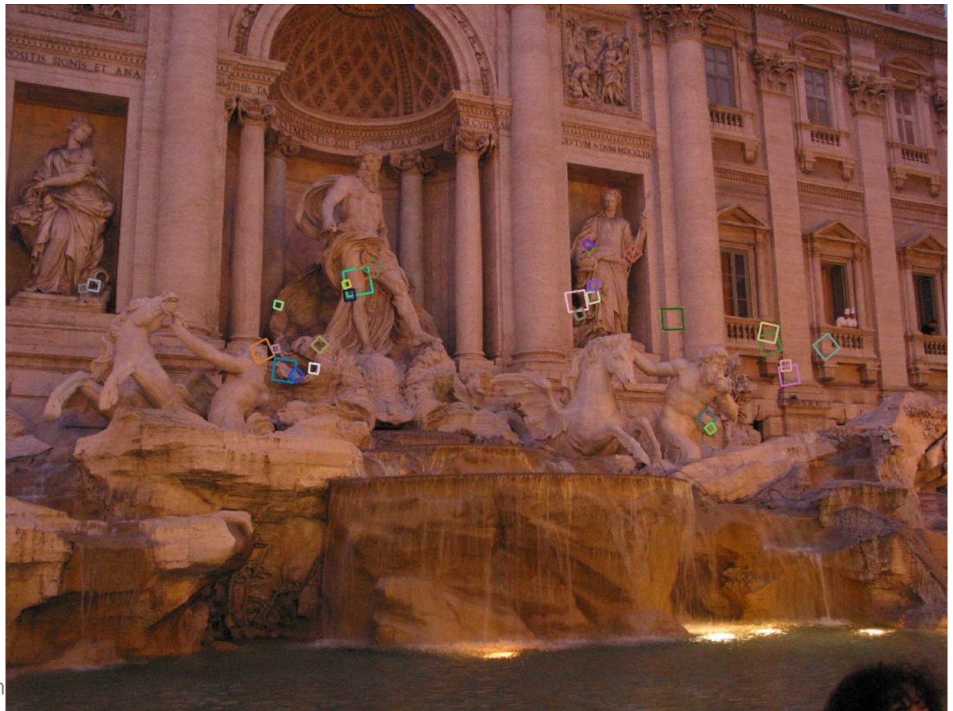


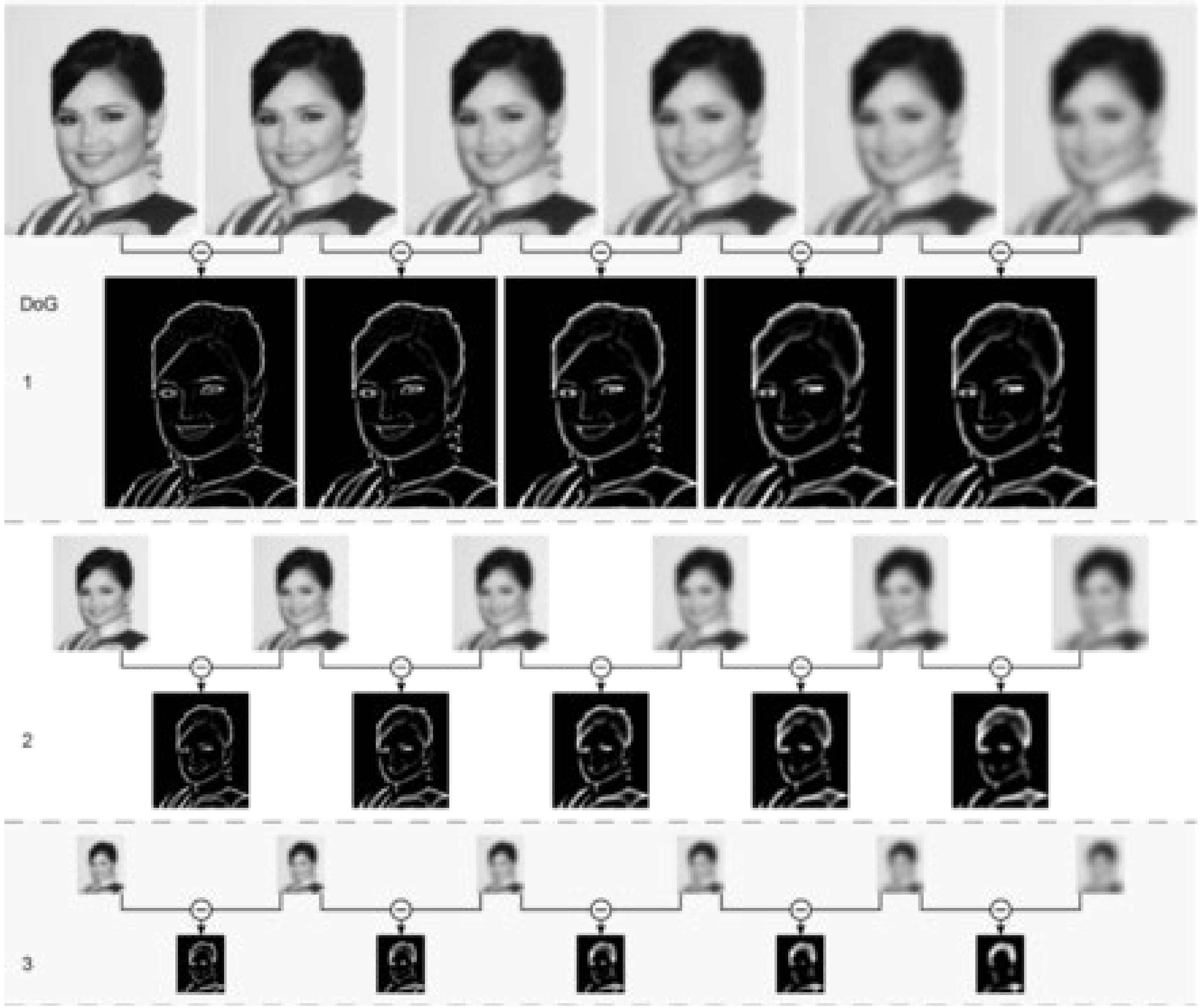
Adapted from slide by David Lowe

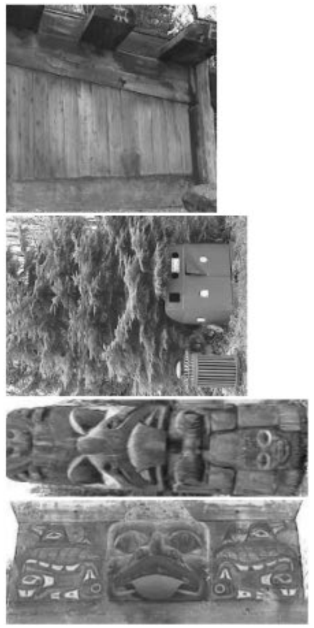
Properties of SIFT

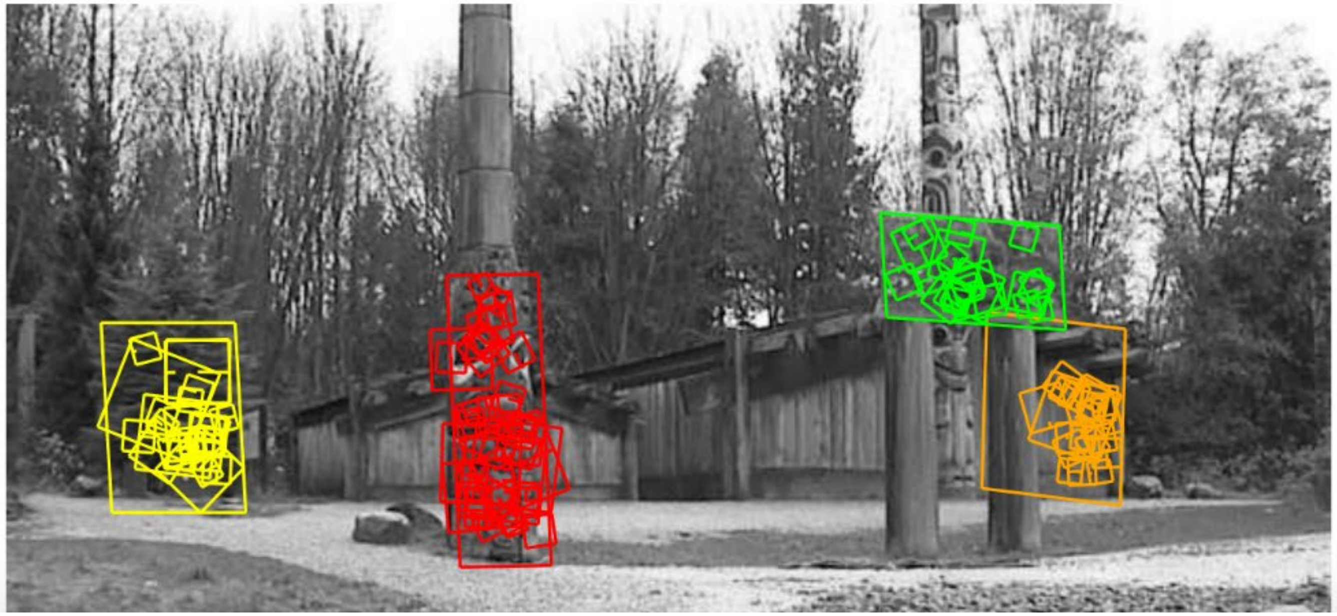
Extraordinarily robust matching technique

- Can handle changes in viewpoint
  - Up to about 60 degree out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available
  - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT


DoG
1
2
3





SIFT (and the related SURF) are covered by US patents

- The patent allows for free use in research but <u>not</u> in any commercial product
- Although the SIFT patent expired in March 2020, most packages (including OpenCV) don’t allow you to use the related functions without changing some parameters and rebuilding the libraries from the source code
- There are similar algorithms that are freely available; we will talk about some in future lectures
  - ORB
  - FAST
  - BRIEF

Today’s Objectives

Corner detection

- Concept
- Harris corner detection
- Shi-Tomasi corner detection
- Förstner corner detection

Keypoint Detection

- Invariant descriptors
- SIFT method