Industrial Computer Vision

- Feature Detection

8th lecture, 2021.10.27

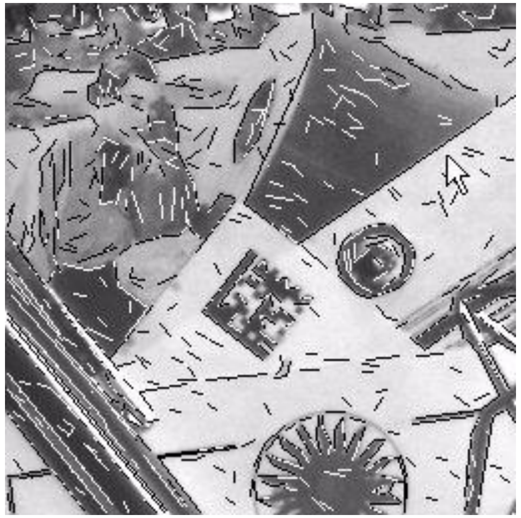
Lecturer: Youngbae Hwang

Contents

- Corner Detection
  - Harris Corner
  - FAST
  - Good Feature To Track
- SIFT (Scale Invariant Feature Transform)

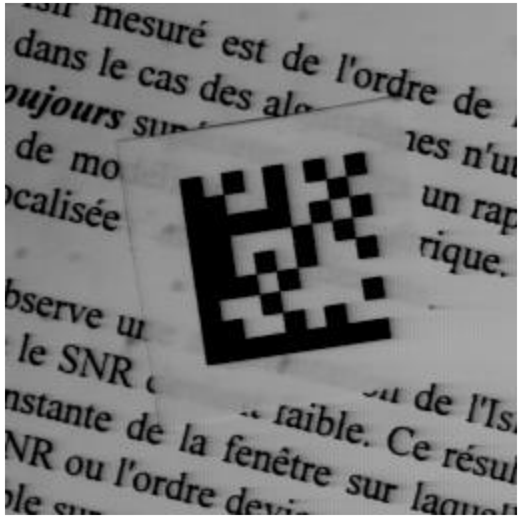
Line Detection

- Useful in remote sensing, document processing etc.
- Edges:
  - boundaries between regions with relatively distinct gray-levels
  - the most common type of discontinuity in an image
- Lines:
  - instances of thin lines in an image occur frequently enough
  - it is useful to have a separate mechanism for detecting them.



Line detection: How?

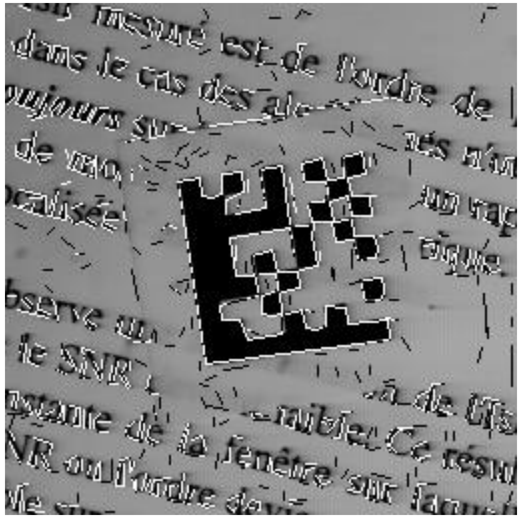
- Possible approaches: Hough transform (more global analysis and may not be considered as a local pre-processing technique)
- Convolve with line detection kernels



- $$L_h = \begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$$

- $$L_v = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

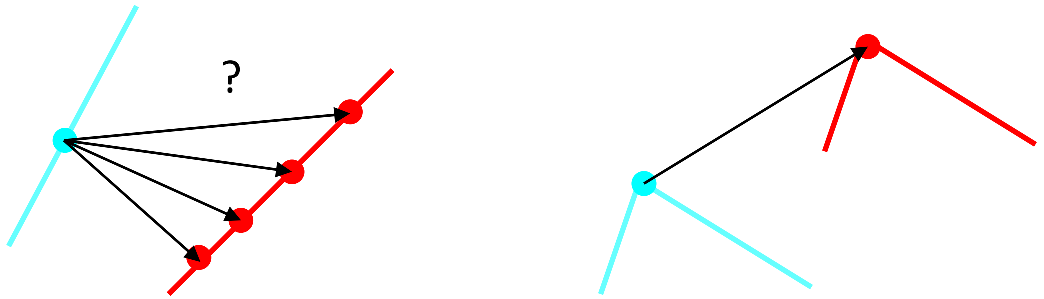
- $$L_o = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$



- How to detection lines along other directions?

Lines and corner for correspondence

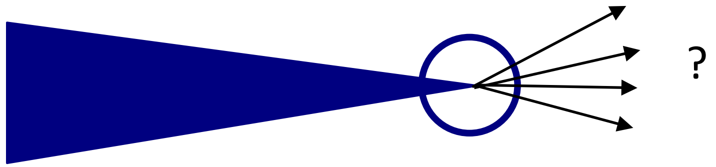
- **Interest points** for solving correspondence problems in time series data.
- Corners are better than lines in solving the above problem due to the **aperture problem**
  - Consider that we want to solve point matching in two images



  - A vertex or corner provides better correspondence

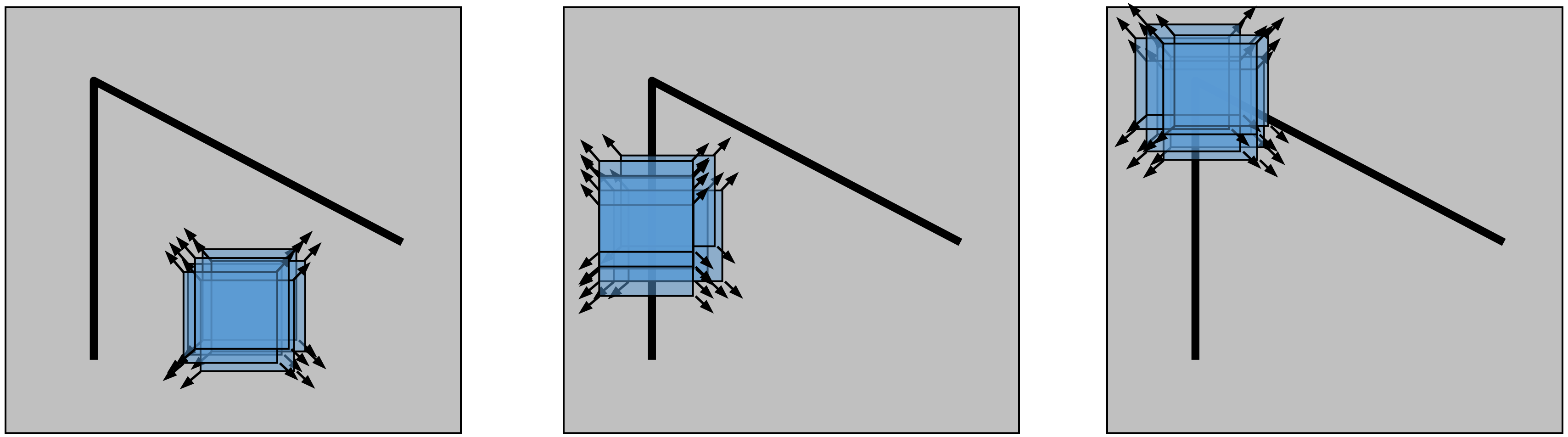
Corners

- **Challenges**
  - Gradient computation is less reliable near a corner due to ambiguity of edge orientation



  - Corner detector are usually not very robust.
  - This deficiency is overcome either by manual intervention or large redundancies.
  - The later approach leads to many more corners than needed to estimate transforms between two images.

Basic Idea



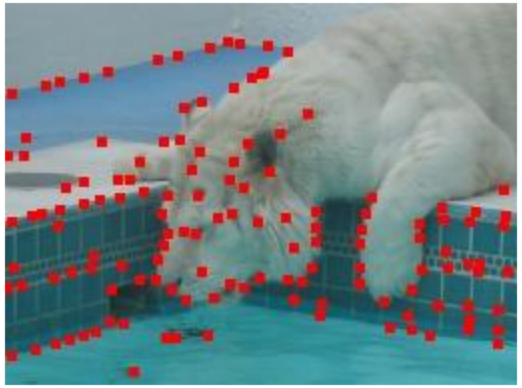
"flat" region:
no change in
all directions

"edge":
no change along
the edge direction

"corner":
significant change
in all directions

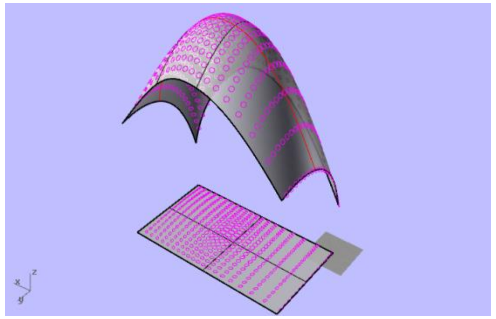
Corner detection

- **Moravec detector:** detects corners as the pixels with locally maximal contrast



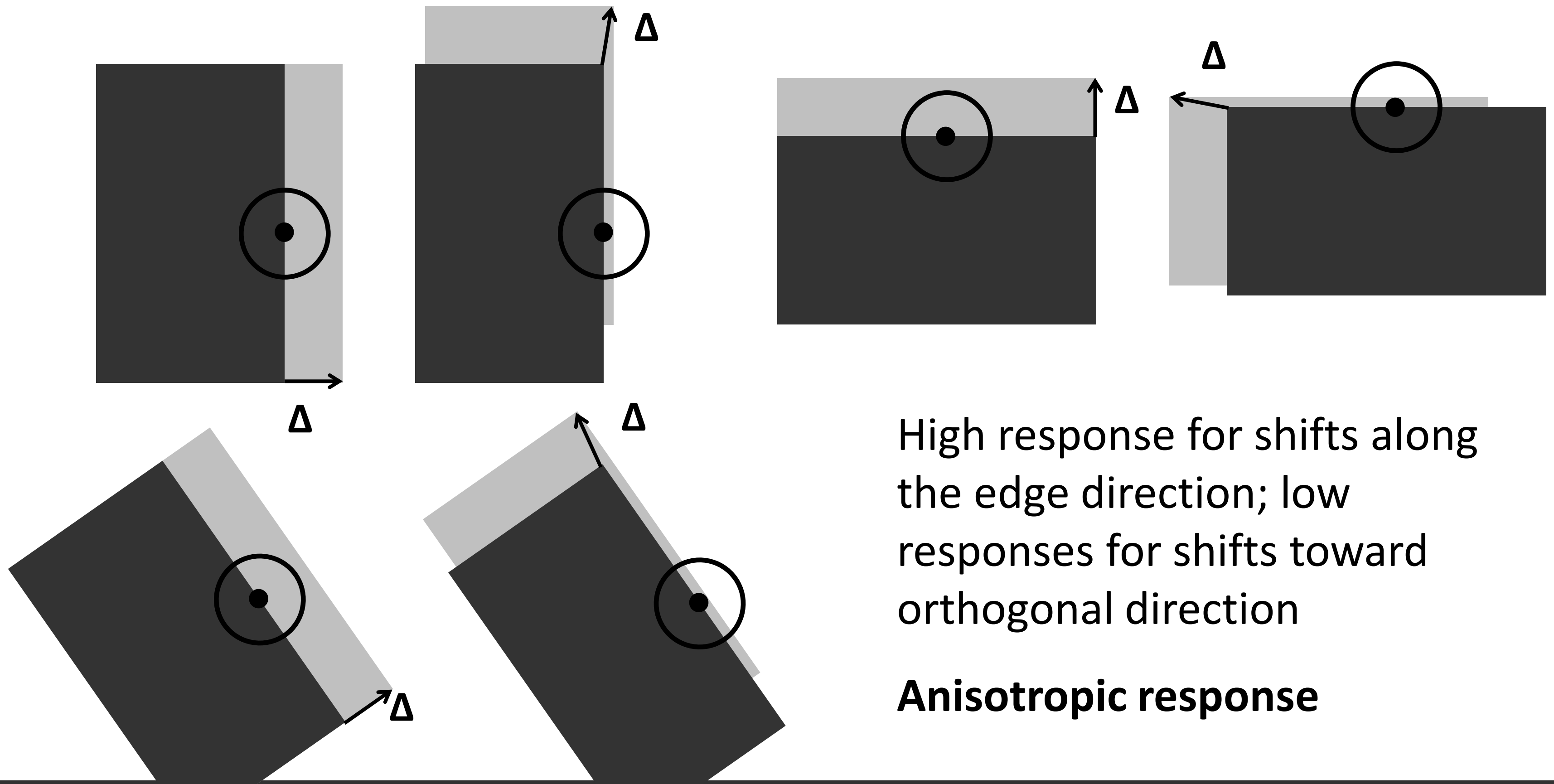
  - $MO(i,j) = \frac{1}{8}\sum_{\Delta i=-1}^{1}\sum_{\Delta j=-1}^{1}|f(i+\Delta i, j+\Delta j) - f(i,j)|$

- **Differential approaches:**
  - Beaudet's approach: Corners are measured as the determinant of the Hessian.
  - Note that the determinant of a Hessian is equivalent to the product of the min & max Gaussian curvatures
    - $H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$
    - Corner *measure* $DET(H) = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x \partial y}\right)^2$



Harris corner detector

- Key idea: Measure changes over a neighborhood due to a shift and then analyze its dependency on shift orientation
- Orientation dependency of the response for lines

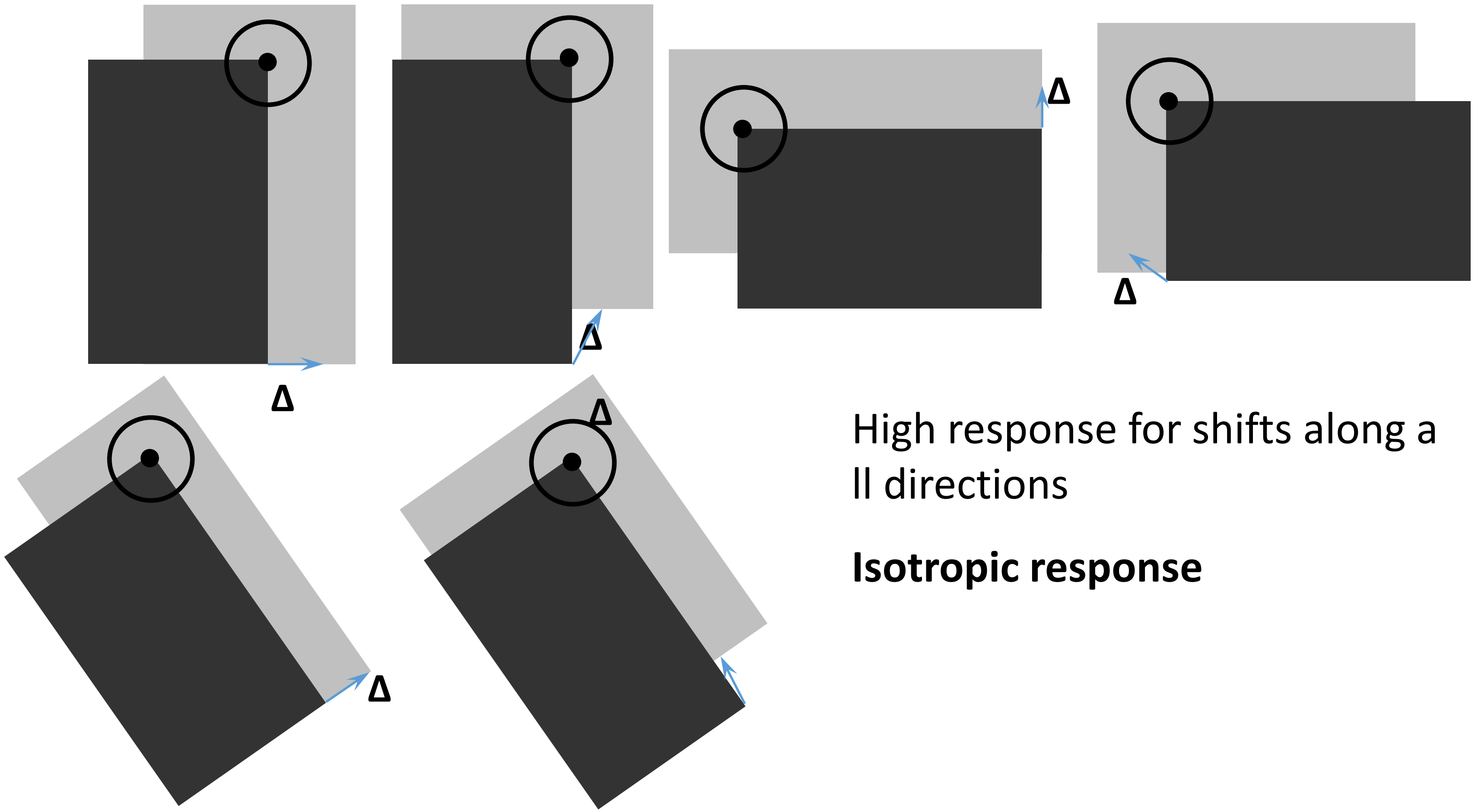


High response for shifts along the edge direction; low responses for shifts toward orthogonal direction

Anisotropic response

Key idea: continued …

- Orientation dependence of the shift response for corners



High response for shifts along a ll directions

Isotropic response

Harris Detector: Mathematics

Change of intensity for the shift [u,v]:

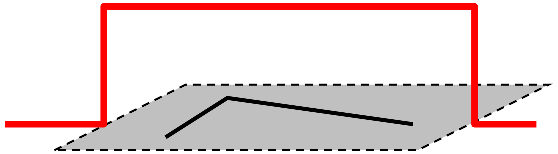
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

Window function

Shifted intensity

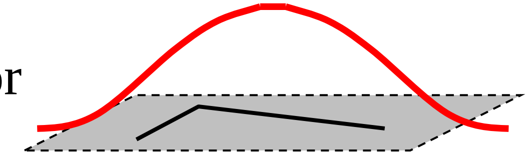
Intensity

Window function $w(x,y)$ =



1 in window, 0 outside

or



Gaussian

Harris corner: mathematical formulation

- An image patch or neighborhood W is shifted by a shift vector $\mathbf{\Delta} = [\Delta x, \Delta y]^T$
- A corner does not have the aperture problem and therefore should show high shift response for all orientation of **Δ**.
- The square intensity difference between the original and the shifted image over the neighborhood W is

- $S_W(\mathbf{\Delta}) = \sum_{(x_i,y_i)\in W} \left(f(x_i, y_i) - f(x_i + \Delta x, y_i + \Delta y)\right)^2$

- Apply first-order Taylor expansion

- $f(x_i + \Delta x, y_i + \Delta y) \approx f(x_i, y_i) + \left[\frac{\partial f(x_i,y_i)}{\partial x} \quad \frac{\partial f(x_i,y_i)}{\partial y}\right] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$

Continued …

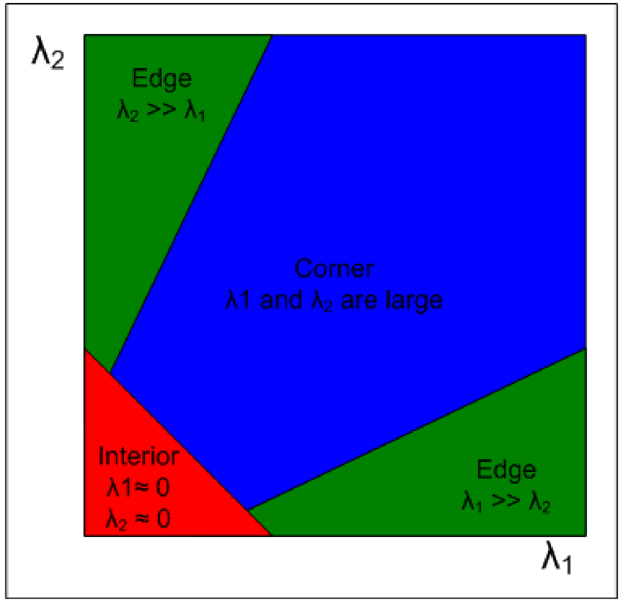
- $$S(x,y,\boldsymbol{\Delta}) = \sum_{(x_i,y_i)\in W} \left(f(x_i,y_i) - f(x_i,y_i) - \begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} & \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^2$$
- $$= \sum_{(x_i,y_i)\in W} \left(\begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} & \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^2$$
- $$= \sum_{(x_i,y_i)\in W} \left(\begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} & \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^{\mathrm{T}} \left(\begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} & \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)$$
- $$= \sum_{(x_i,y_i)\in W} \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} \\ \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} & \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$
- $$= \sum_{(x_i,y_i)\in W} \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \left(\begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} \\ \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} & \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \right) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

Continued …

- $$= \sum_{(x_i,y_i)\in W} [\Delta x \quad \Delta y] \left(\begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} \\ \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} & \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \right) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$
- $$= [\Delta x \quad \Delta y] \left(\sum_{(x_i,y_i)\in W} \begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} \\ \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial f(x_i,y_i)}{\partial x} & \frac{\partial f(x_i,y_i)}{\partial y} \end{bmatrix} \right) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$
- $$= [\Delta x \quad \Delta y] \begin{bmatrix} \sum_{(x_i,y_i)\in W} \left(\frac{\partial f(x_i,y_i)}{\partial x}\right)^2 & \sum_{(x_i,y_i)\in W} \frac{\partial f(x_i,y_i)}{\partial x}\frac{\partial f(x_i,y_i)}{\partial y} \\ \sum_{(x_i,y_i)\in W} \frac{\partial f(x_i,y_i)}{\partial x}\frac{\partial f(x_i,y_i)}{\partial y} & \sum_{(x_i,y_i)\in W} \left(\frac{\partial f(x_i,y_i)}{\partial y}\right)^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$
- $$= \mathbf{\Delta}^{\mathrm{T}} \mathrm{A}_w(x, y) \mathbf{\Delta}$$

Harris matrix

- The matrix $\mathbf{A}_W$ is called the Harris matrix and its symmetric and positive semi-definite. Eigen-value decomposition of of $\mathbf{A}_W$ gives eigenvectors and eigenvalues (λ_1, λ_2) of the response matrix.
- Three distinct situations:
  - Both λ_1 and λ_2 are small $\Rightarrow$ no edge or corner; a flat region
  - λ_i is large but $\lambda_{j\neq l}$ is small $\Rightarrow$ existence of an edge; no corner
  - Both λ_1 and λ_2 are large $\Rightarrow$ existence of a corner

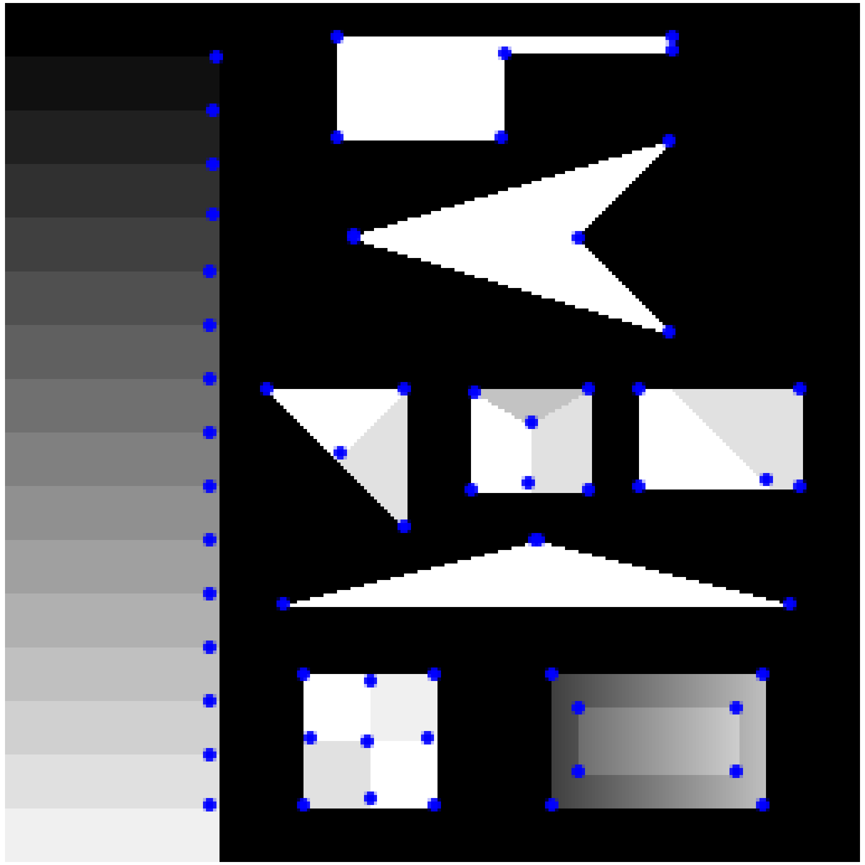


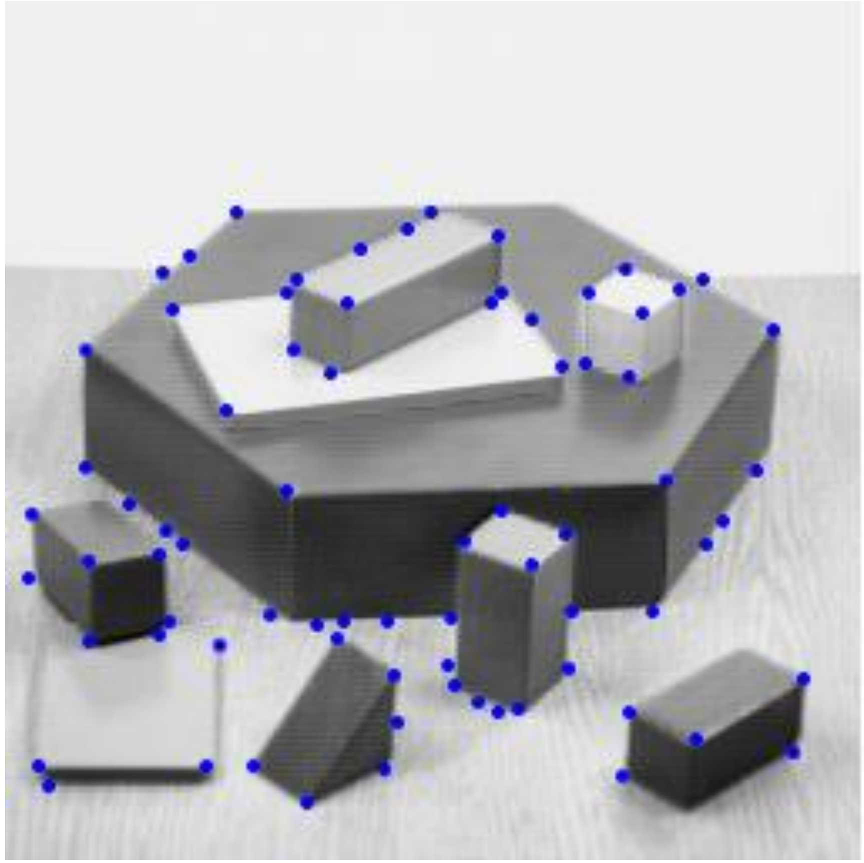
- Avoid eigenvalue decomposition and compute a single response measure
  - Harris response function
    - $R(A) = \det(A) - \kappa * trace^2(A)$
  - A value of κ between 0.04 and 0.15 has be used in literature.

Algorithm: Harris corner detection

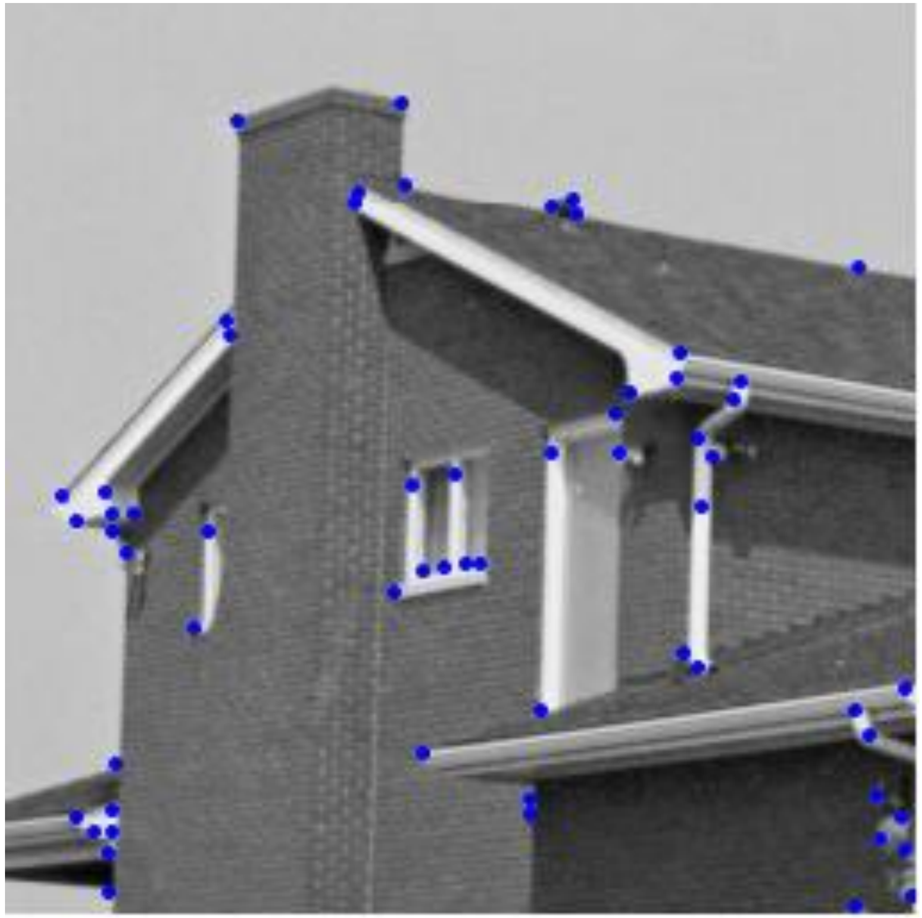
1. Filter the image with a Gaussian
2. Estimate intensity gradient in two coordinate directions
3. For each pixel c and a neighborhood window W
   a. Calculate the local Harris matrix A
   b. Compute the response function $R(A)$
4. Choose the best candidates for corners by selecting thresholds on the response function $R(A)$
5. Apply non-maximal suppression

Examples





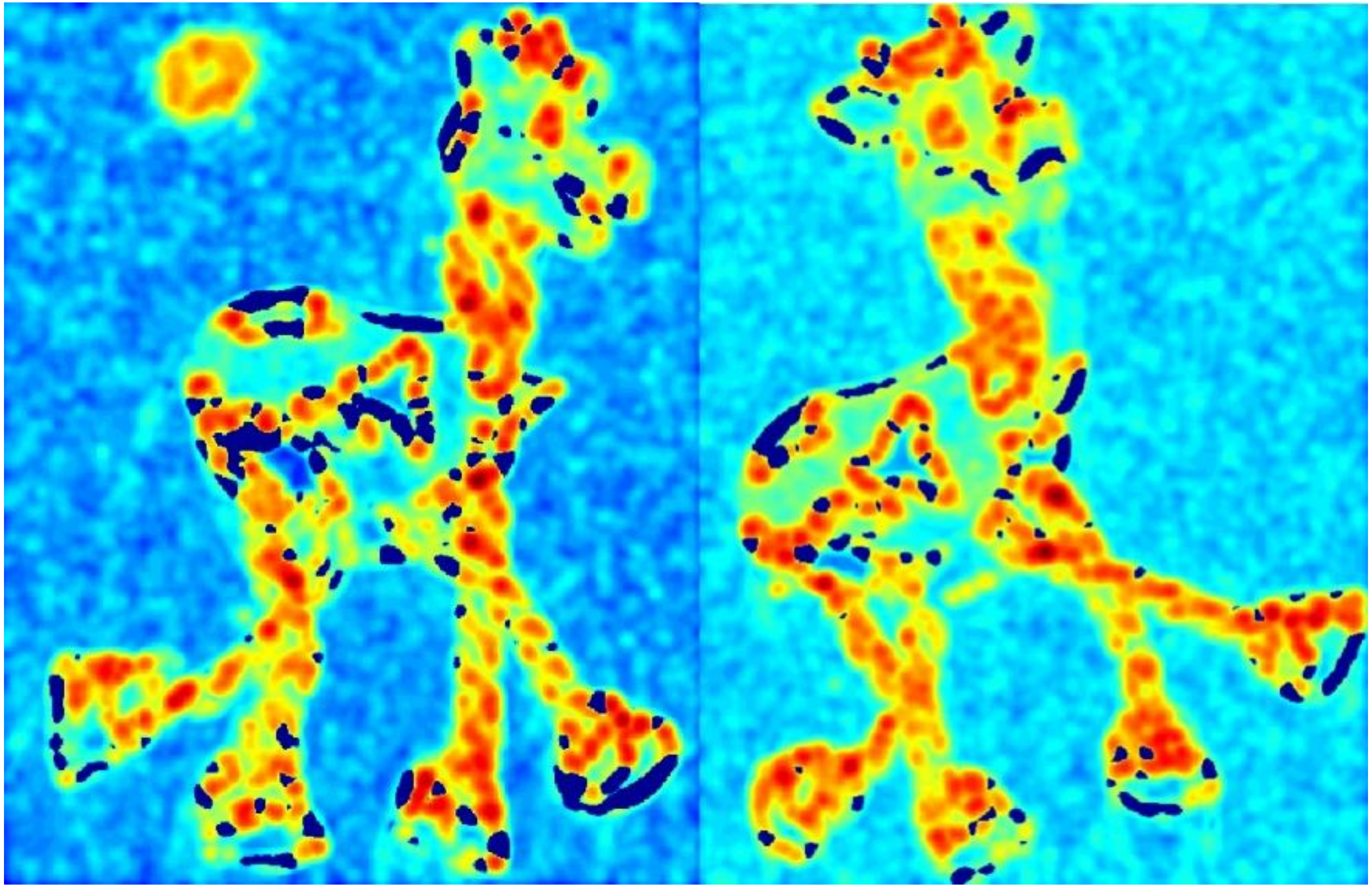
Examples



Harris Detector: Workflow

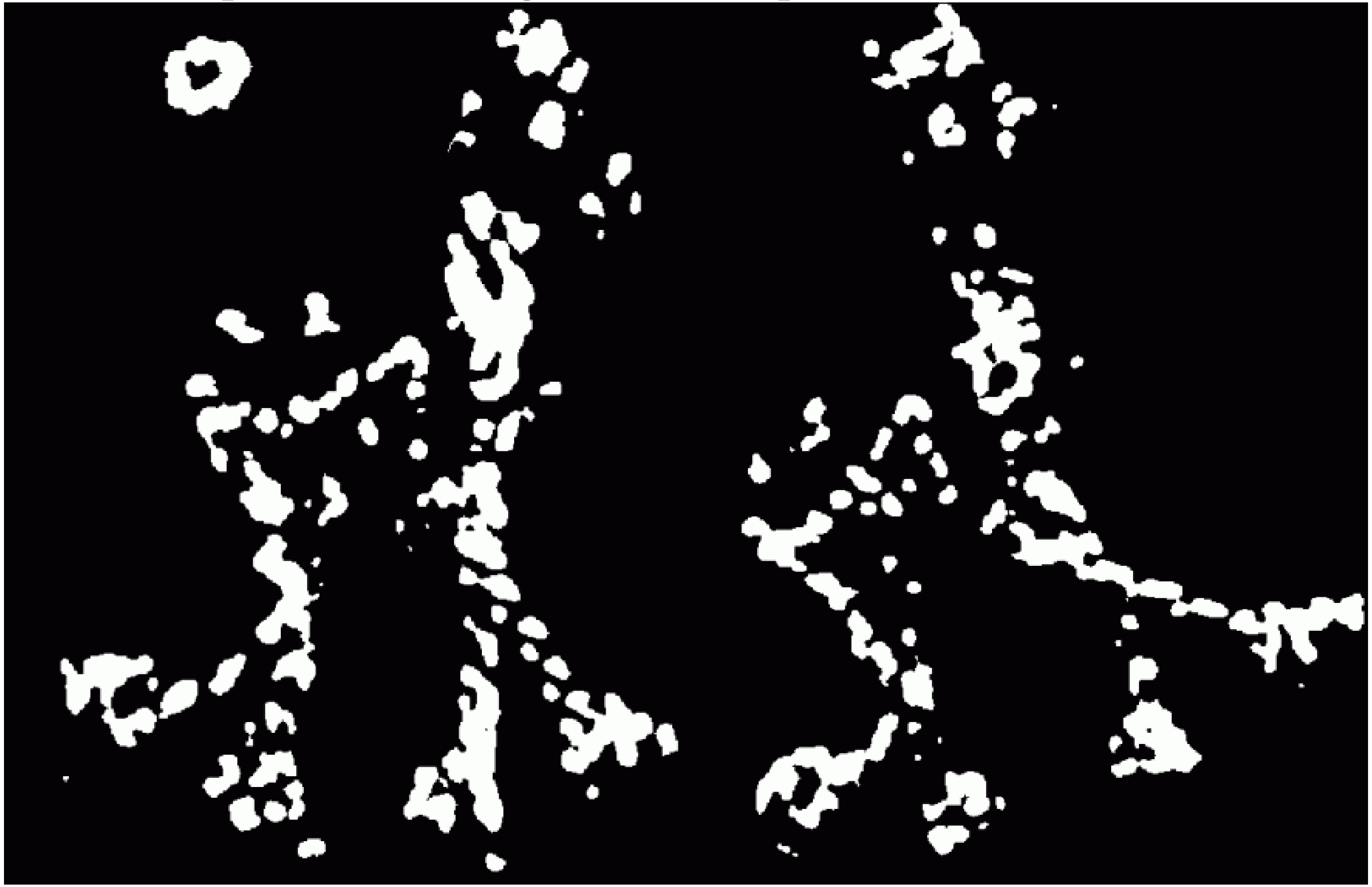
Harris Detector: Workflow

Compute corner response R



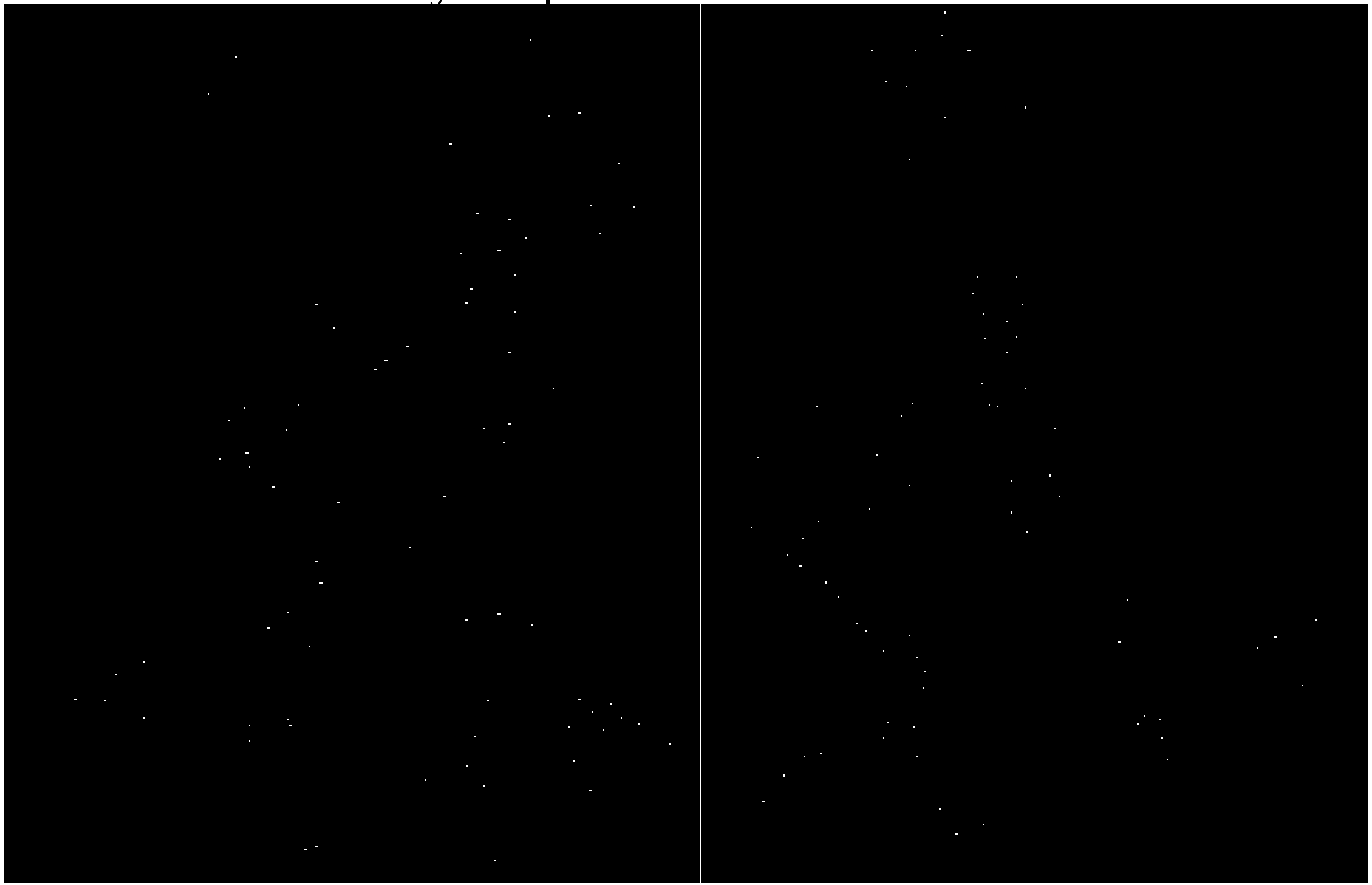
Harris Detector: Workflow

Find points with large corner response: $R>$threshold



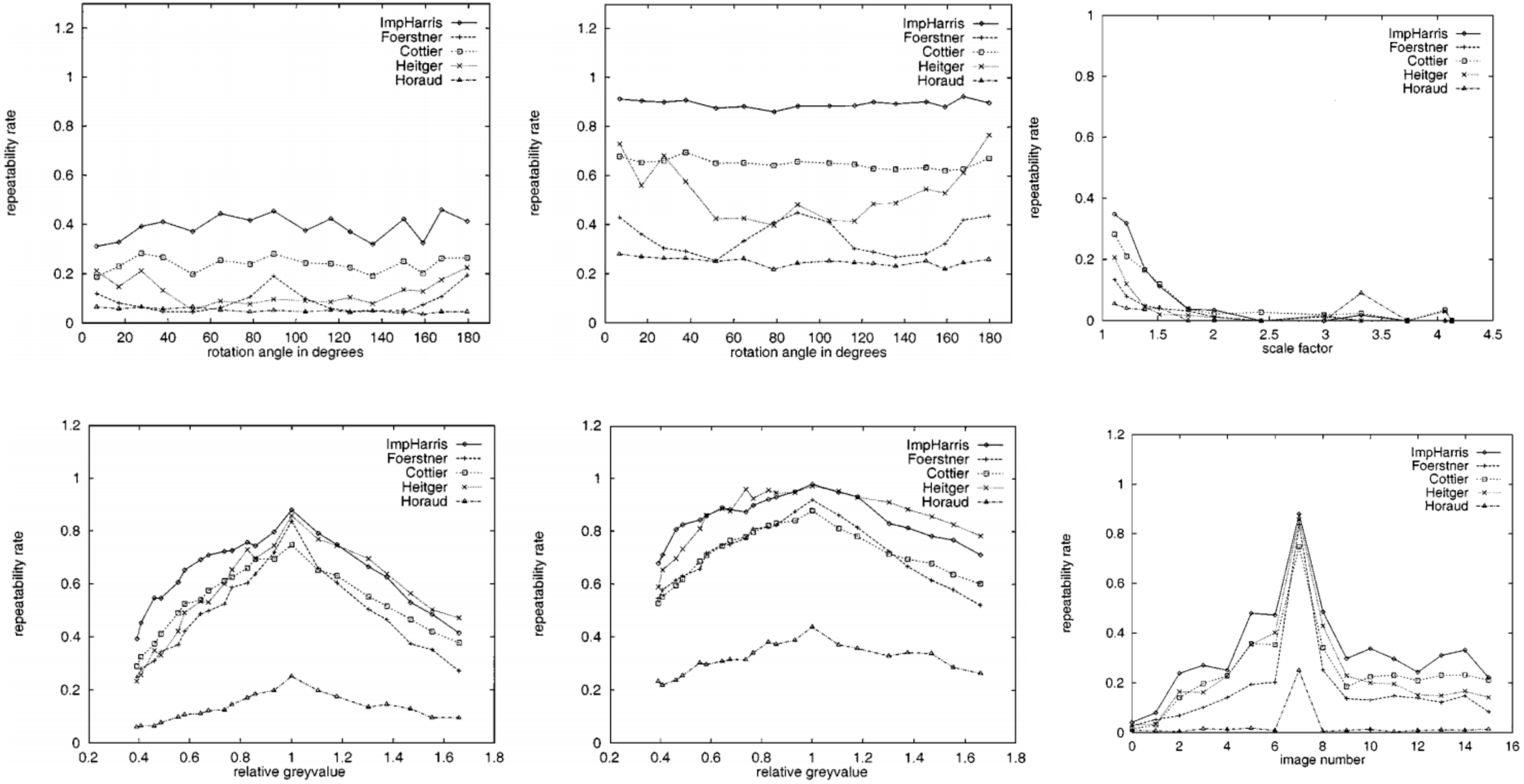
Harris Detector: Workflow

Take only the points of local maxima of R

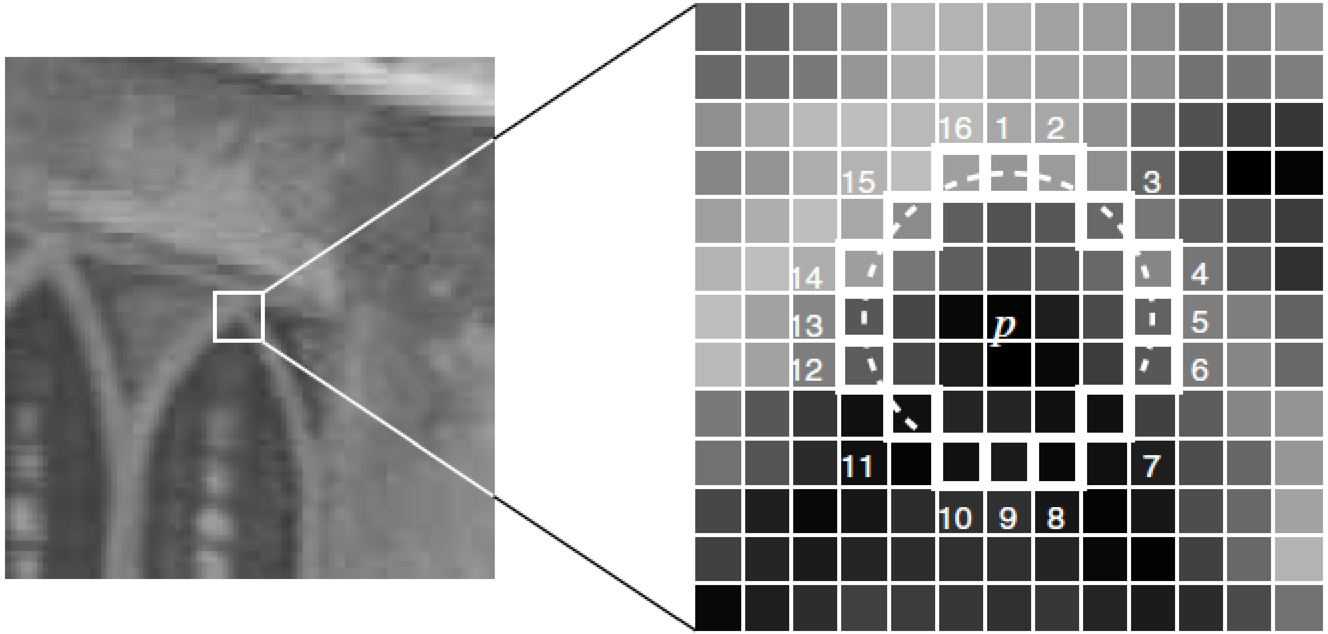


Harris Detector: Workflow

Repeatability



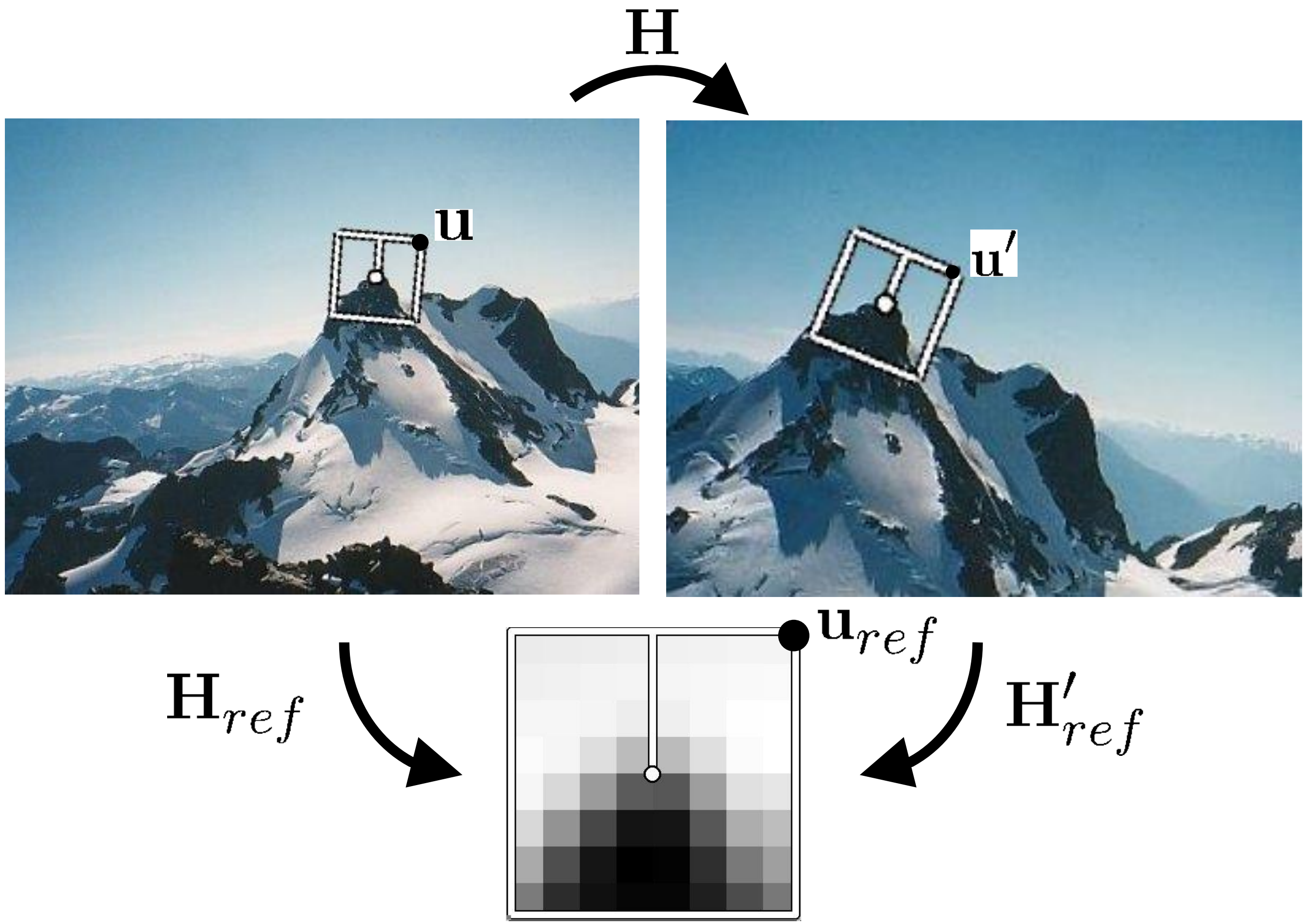
FAST (Features from Accelerated Segment Test)



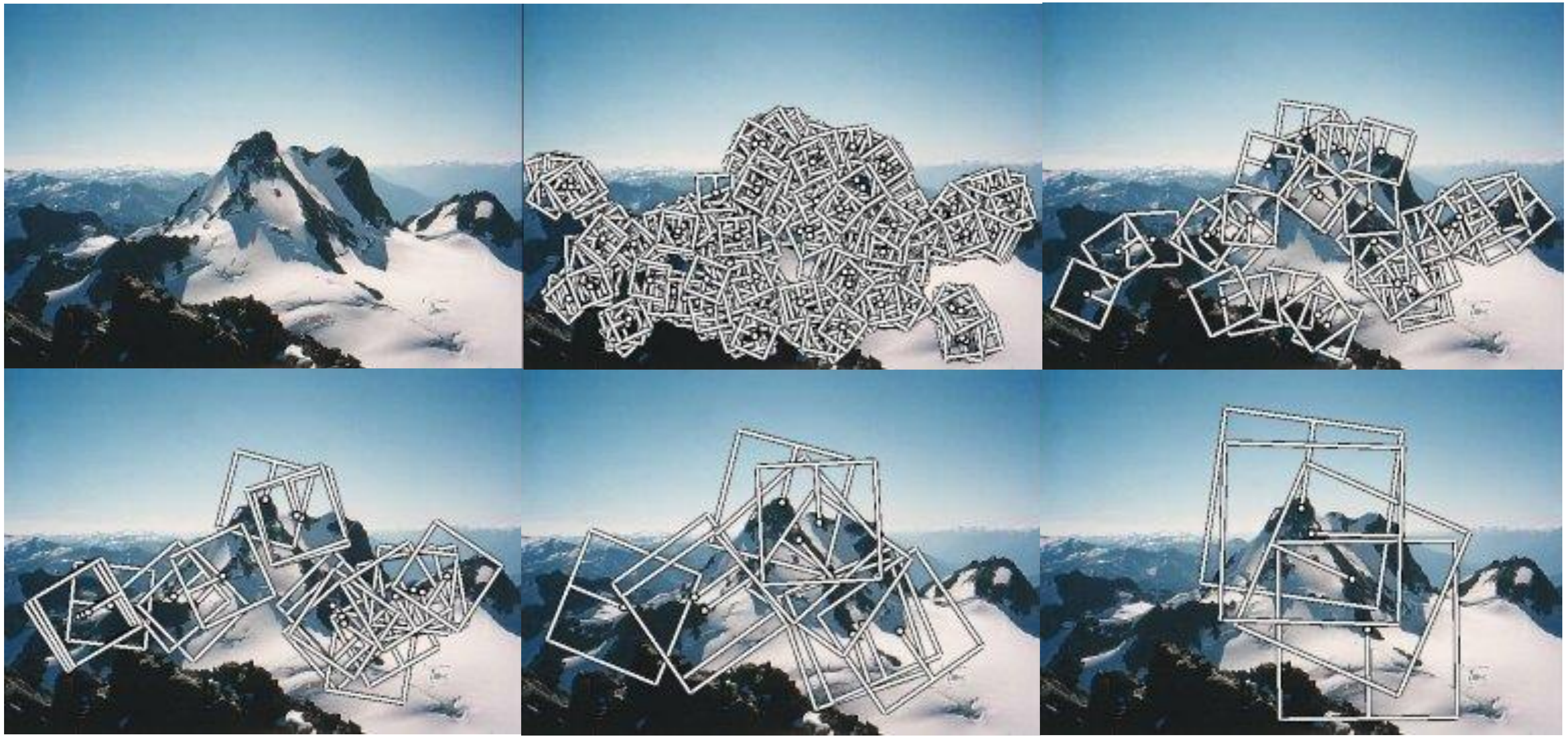
$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \leq I_p - t \quad \text{(darker)} \\ s, & I_p - t < I_{p\to x} < I_p + t \quad \text{(similar)} \\ b, & I_p + t \leq I_{p\to x} \quad \text{(brighter)} \end{cases}$$

Fig. 1. 12 point segment test corner detection in an image patch. The highlighted squares are the pixels used in the corner detection. The pixel at p is the centre of a candidate corner. The arc is indicated by the dashed line passes through 12 contiguous pixels which are brighter than p by more than the threshold.

Review: Matt Brown's Canonical Frames



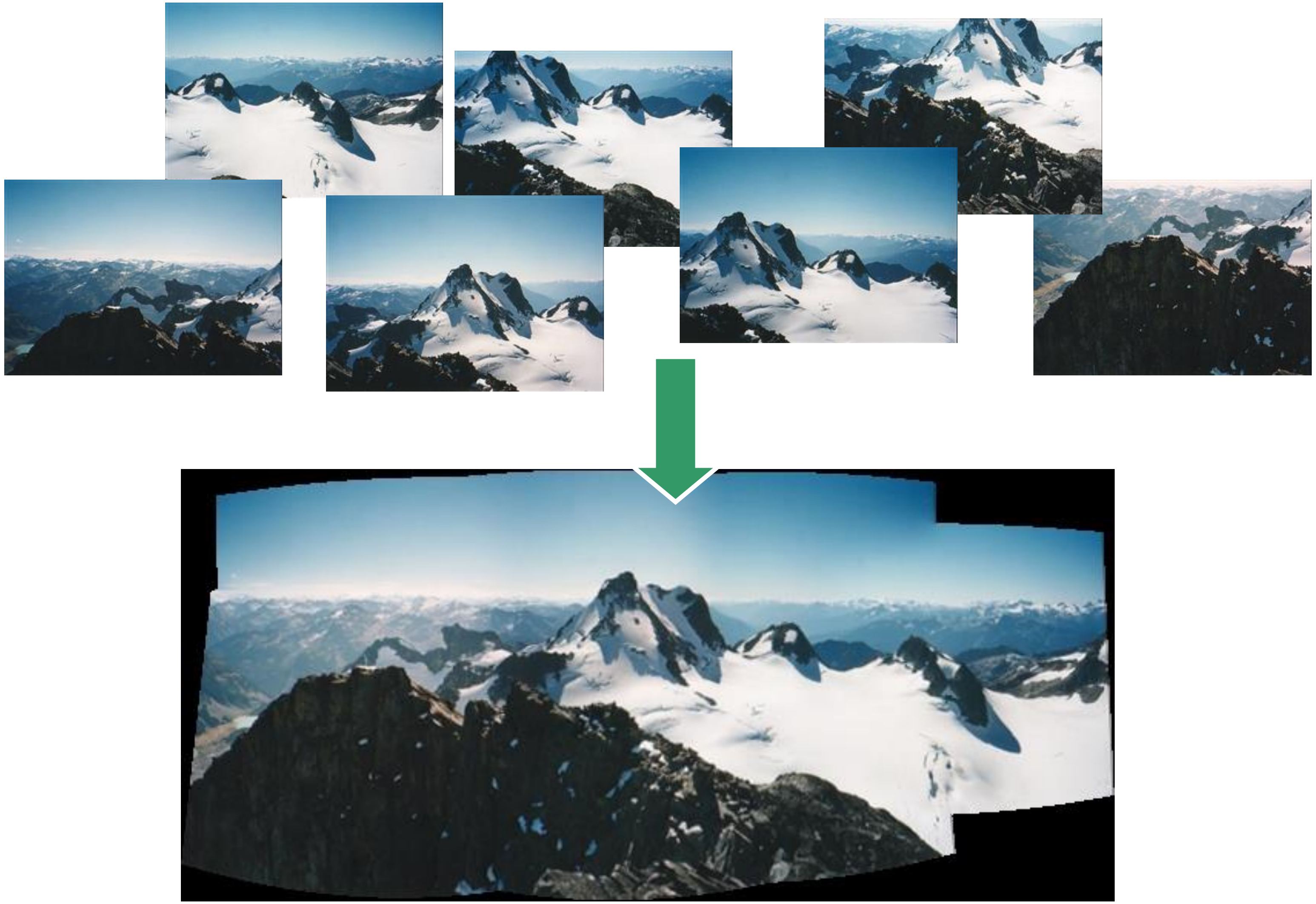
Multi-Scale Oriented Patches



- Extract oriented patches at multiple scales

[Brown, Szeliski, Winder CVPR 2005]

Application: Image Stitching



[Microsoft Digital Image Pro version 10]

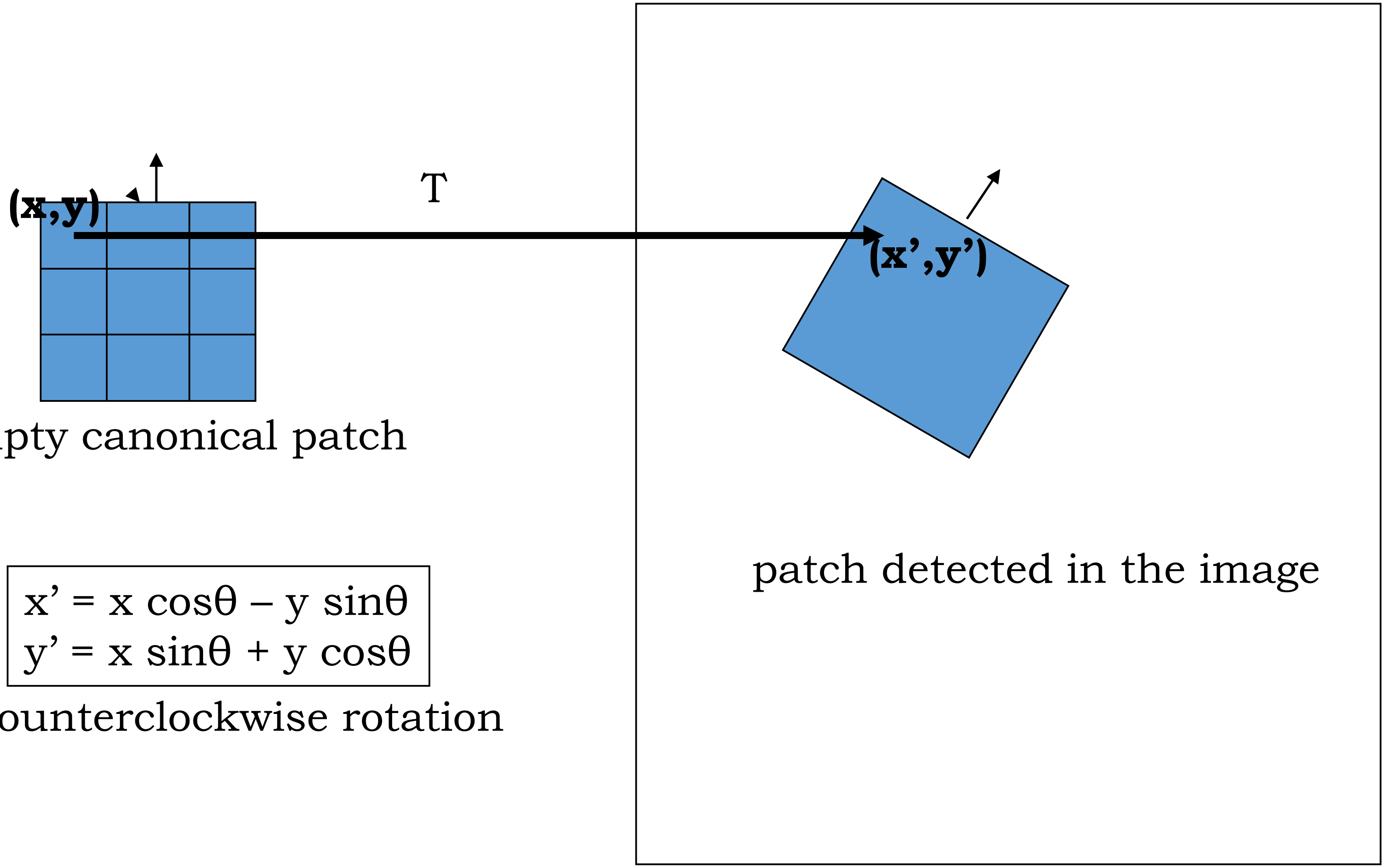
Ideas from Matt's Multi-Scale Oriented Patches

- 1. Detect an interesting patch with an interest operator. Patches are translation invariant.
- 2. Determine its dominant orientation.
- 3. Rotate the patch so that the dominant orientation points upward. This makes the patches rotation invariant.
- 4. Do this at multiple scales, converting them all to one scale through sampling.
- 5. Convert to illumination "invariant" form

Implementation Concern: How do you rotate a patch?

- Start with an “empty” patch whose dominant direction is “up”.
- For each pixel in your patch, compute the position in the detected image patch. It will be in floating point and will fall between the image pixels.
- Interpolate the values of the 4 closest pixels in the image, to get a value for the pixel in your patch.

Rotating a Patch



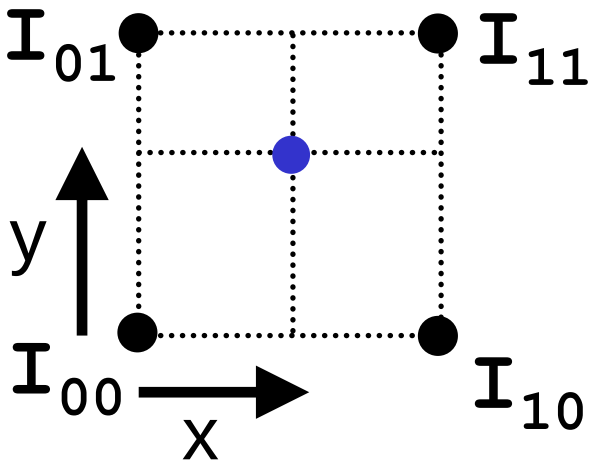
T

$$\boxed{\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned}}$$

counterclockwise rotation

Using Bilinear Interpolation

- Use all 4 adjacent samples



SIFT: Motivation

- The Harris operator is not invariant to scale and correlation is not invariant to rotation[1].

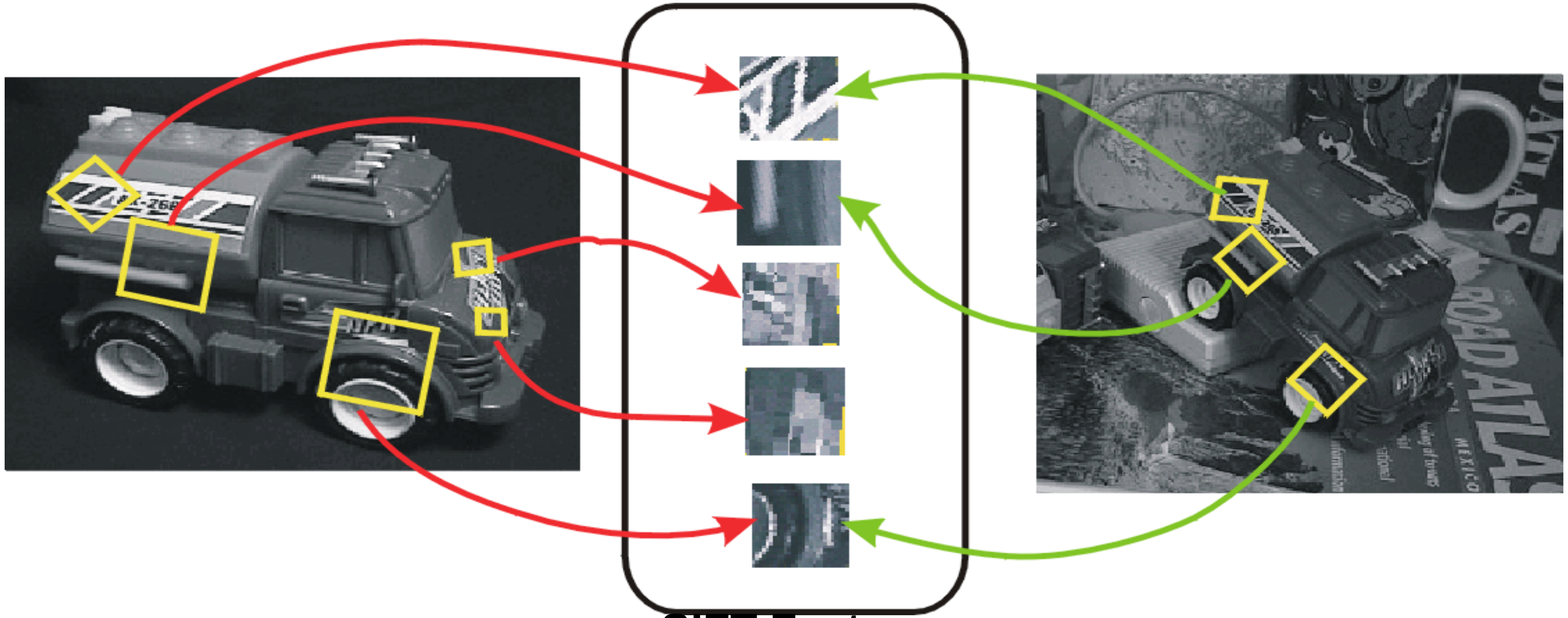
- For better image matching, Lowe's goal was to develop an interest operator that is invariant to scale and rotation.

- Also, Lowe aimed to create a descriptor that was robust to the variations corresponding to typical viewing conditions. The descriptor is the most-used part of SIFT.

[1]But Schmid and Mohr developed a rotation invariant descriptor for it in 1997.

Idea of SIFT

- Image content is transformed into local feature coordinates that are inv ariant to translation, rotation, scale, and other imaging parameters



SIFT Features

Claimed Advantages of SIFT

- Locality: features are local, so robust to occlusion and clutter (no prior segm entation)
- Distinctiveness: individual features can be matched to a large database of ob jects
- Quantity: many features can be generated for even small objects
- Efficiency: close to real-time performance
- Extensibility: can easily be extended to wide range of differing feature types, with each adding robustness

Overall Procedure at a High Level

1. Scale-space extrema detection

   Search over multiple scales and image locations.

2. Keypoint localization

   Fit a model to detrmine location and scale.
   Select keypoints based on a measure of stability.

3. Orientation assignment

   Compute best orientation(s) for each keypoint region.

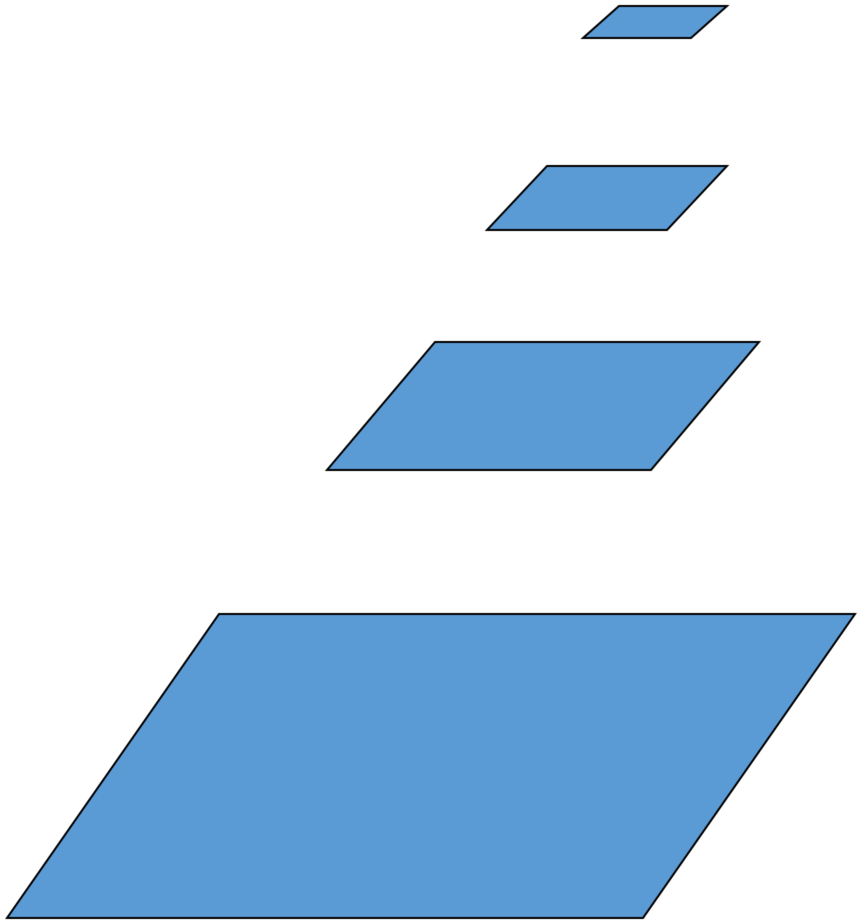
4. Keypoint description

   Use local image gradients at selected scale and rotation to describe each keypoint region.

1. Scale-space extrema detection

- Goal: Identify locations and scales that can be repeatably assigned under different views of the same scene or object.
- Method: search for stable features across multiple scales using a continuous function of scale.
- Prior work has shown that under a variety of assumption s, the best function is a Gaussian function.
- The scale space of an image is a function $L(x,y,\sigma)$ that is p roduced from the convolution of a Gaussian kernel (at dif ferent scales) with the input image.

Aside: Image Pyramids



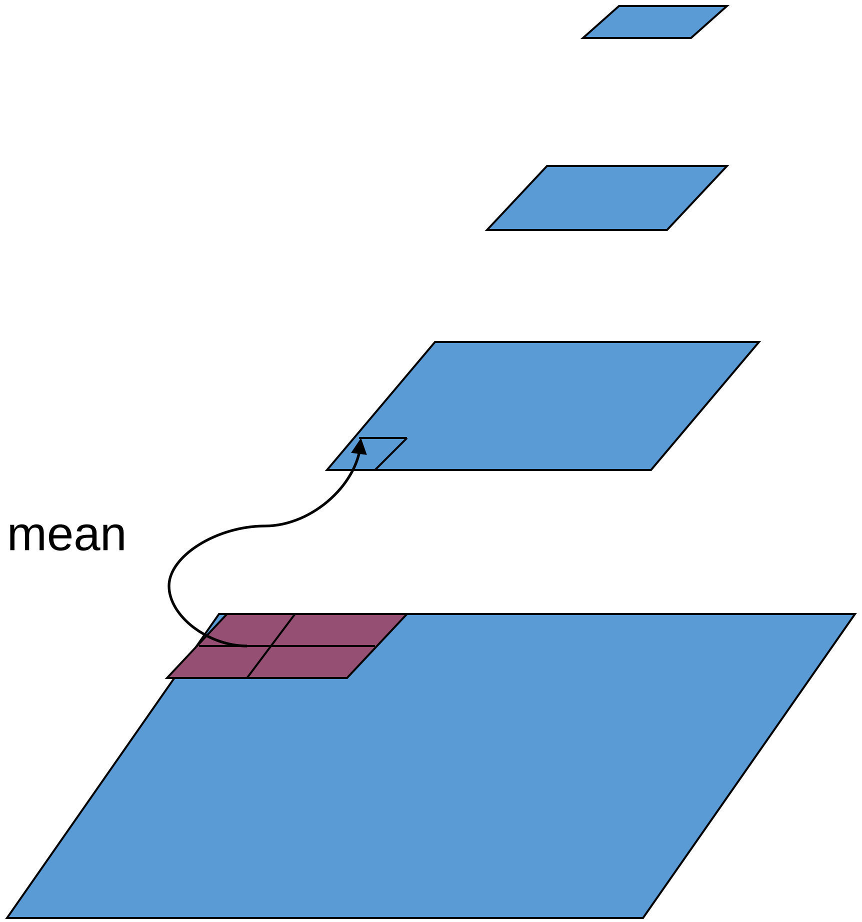
And so on.

3rd level is derived from the 2nd level according to the same funtion

2nd level is derived from the original image according to some function

Bottom level is the original image.

Aside: Mean Pyramid



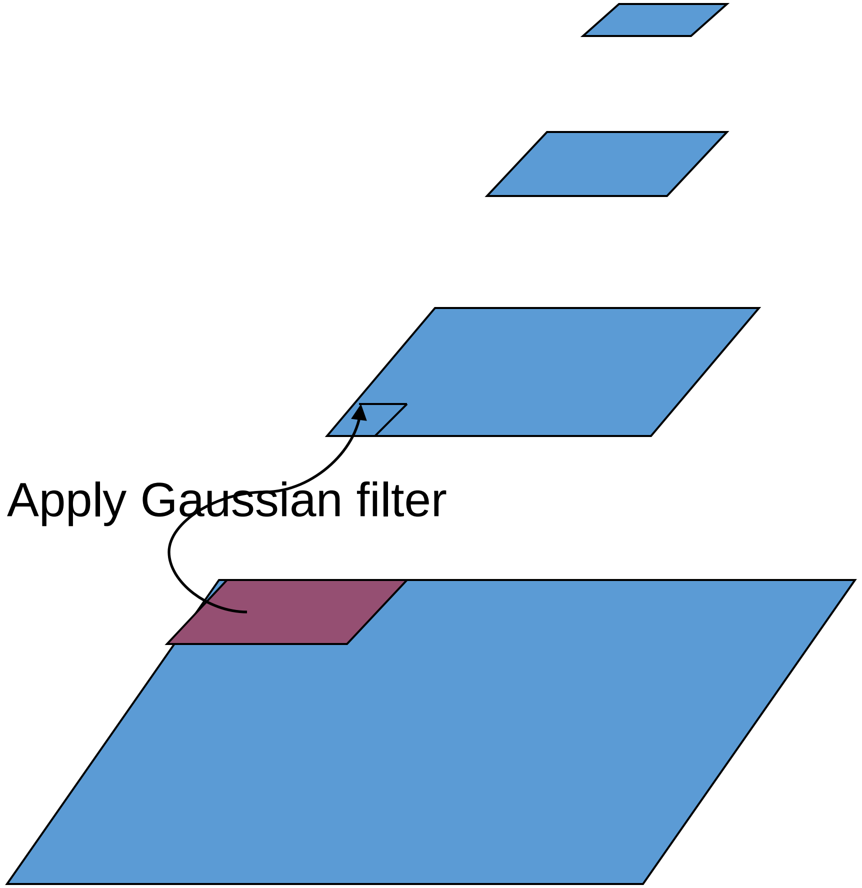
And so on.

At 3rd level, each pixel is the mean of 4 pixels in the 2nd level.

At 2nd level, each pixel is the mean of 4 pixels in the original image.

Bottom level is the original image.

Aside: Gaussian Pyramid
At each level, image is smoothed and reduced in size.

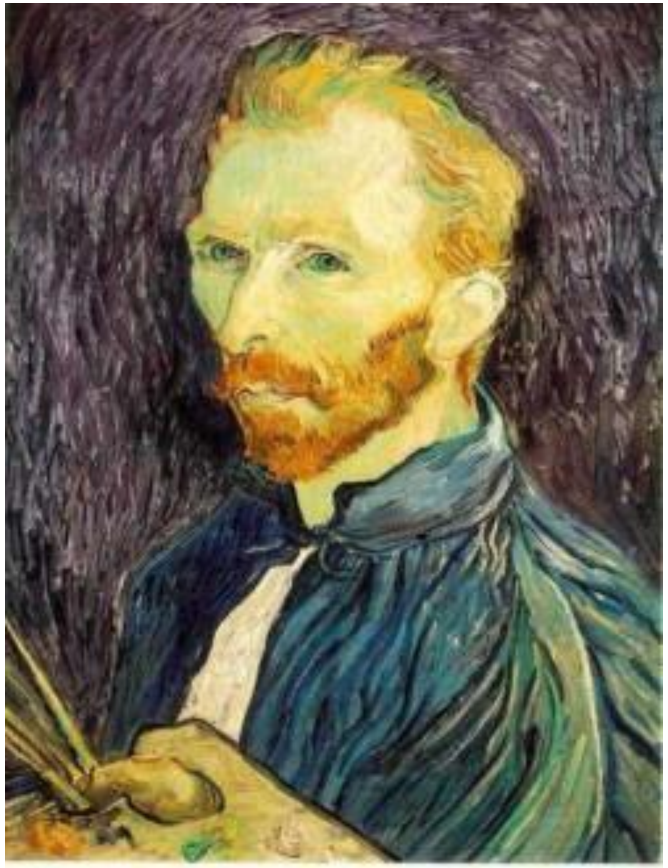


And so on.

At 2nd level, each pixel is the result of applying a Gaussian mask to the first level and then subsampling to reduce the size.

Bottom level is the original image.

Example: Subsampling with Gaussian pre-filtering



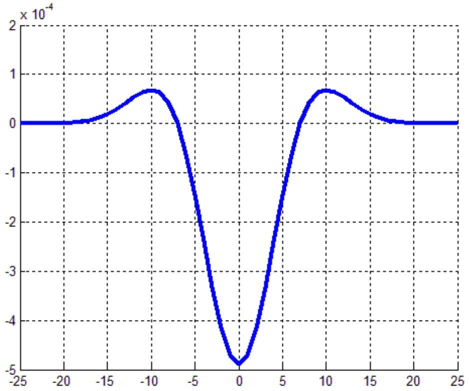
Gaussian 1/2

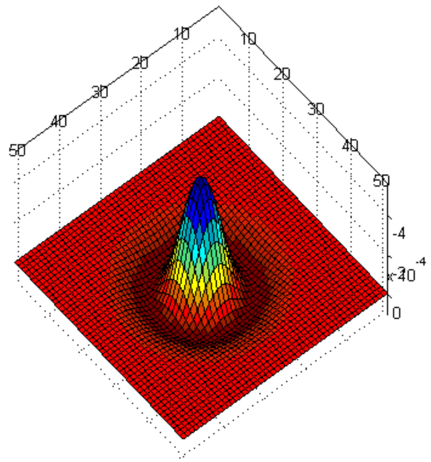
G 1/4

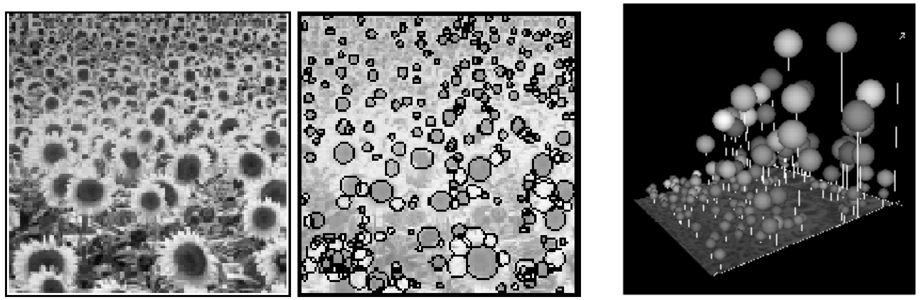
G 1/8

Lowe’s Scale-space Interest Points

- Laplacian of Gaussian kernel
  - Scale normalised (x by scale2)
  - Proposed by Lindeberg
- Scale-space detection
  - Find local maxima across scale/space
  - A good “blob” detector





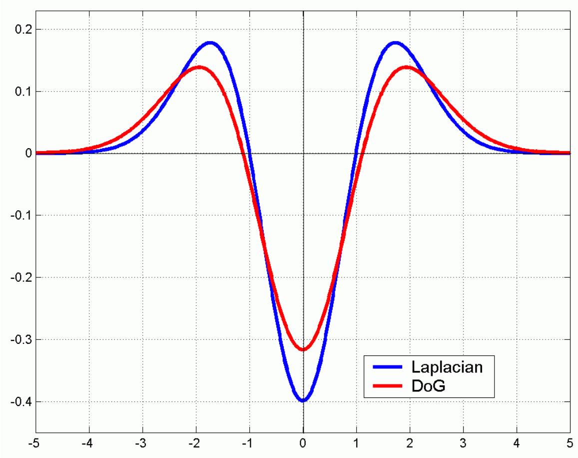


$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{x^2+y^2}{\sigma^2}}$$

$$\nabla^2 G(x, y, \sigma) = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$$

[T. Lindeberg IJCV 1998]

Lowe's Scale-space Interest Points: Difference of Gaussians



$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G.$$

- Gaussian is an ad hoc solution of heat diffusion equation
- Hence

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 \nabla^2 G.$$

- k is not necessarily very small in p ractice

Lowe’s Pyramid Scheme

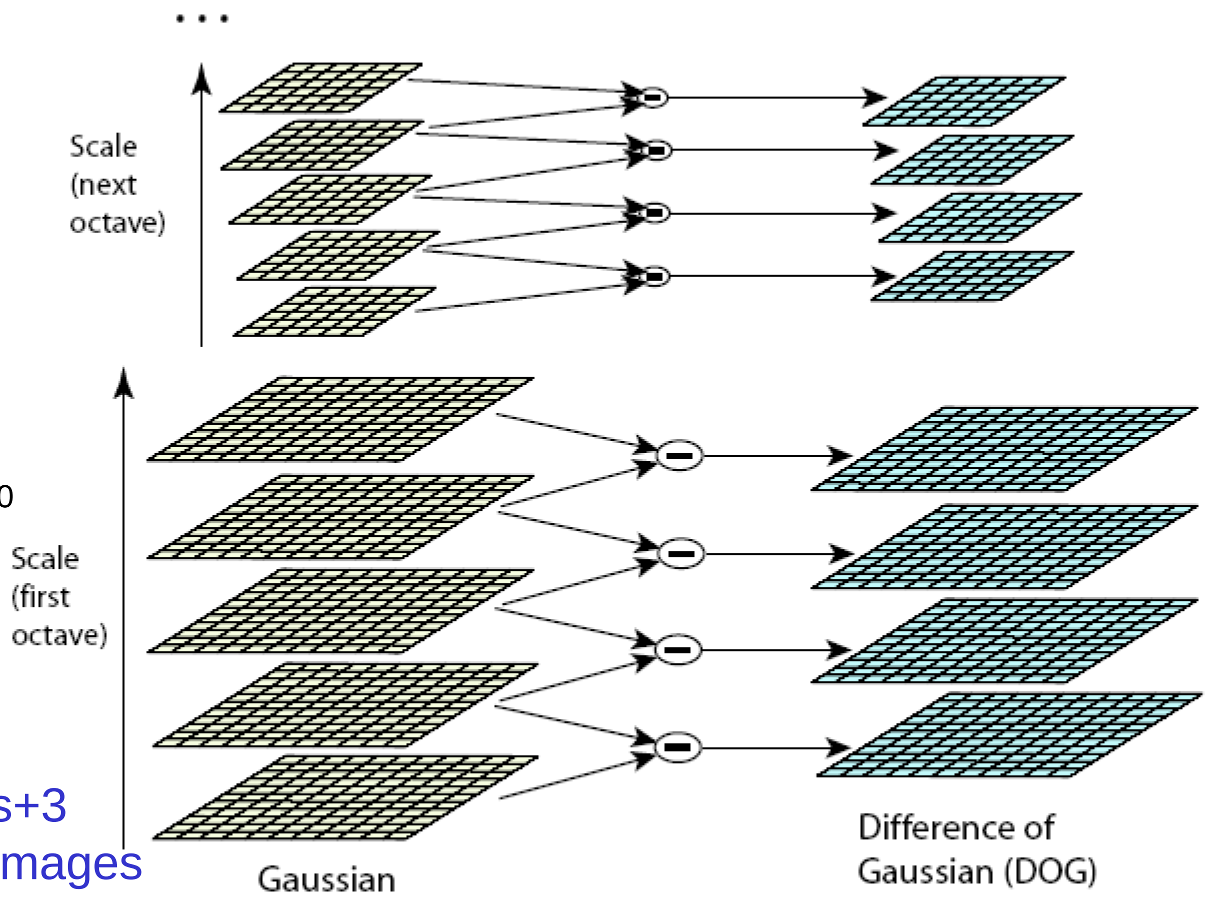
- Scale space is separated into octaves:
  - Octave 1 uses scale σ
  - Octave 2 uses scale 2σ
  - etc.

- In each octave, the initial image is repeatedly convolved with Gaussians to produce a set of scale space images.

- Adjacent Gaussians are subtracted to produce the DOG

- After each octave, the Gaussian image is down-sampled by a factor of 2 to produce an image ¼ the size to start the next level.

Lowe's Pyramid Scheme



s+2 filters

$\sigma_{s+1}=2^{(s+1)/s}\sigma_0$

.

.

$\sigma_i=2^{i/s}\sigma_0$

.

.

$\sigma_2=2^{2/s}\sigma_0$

$\sigma_1=2^{1/s}\sigma_0$

σ_0

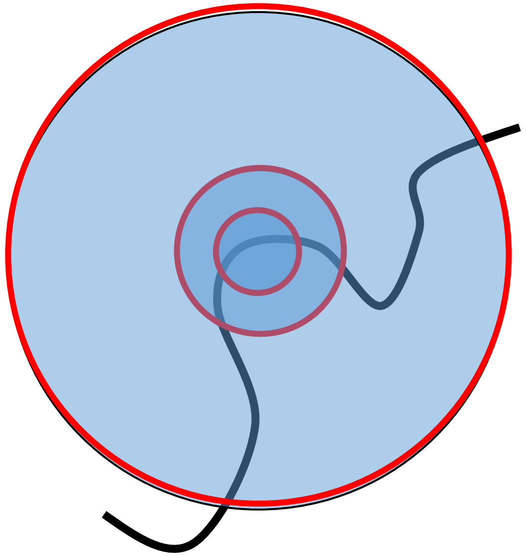
s+3 images including original

s+2 differ-ence images

The parameter **s** determines the number of images per octave.

Scale invariant detection

Suppose you're looking for corners

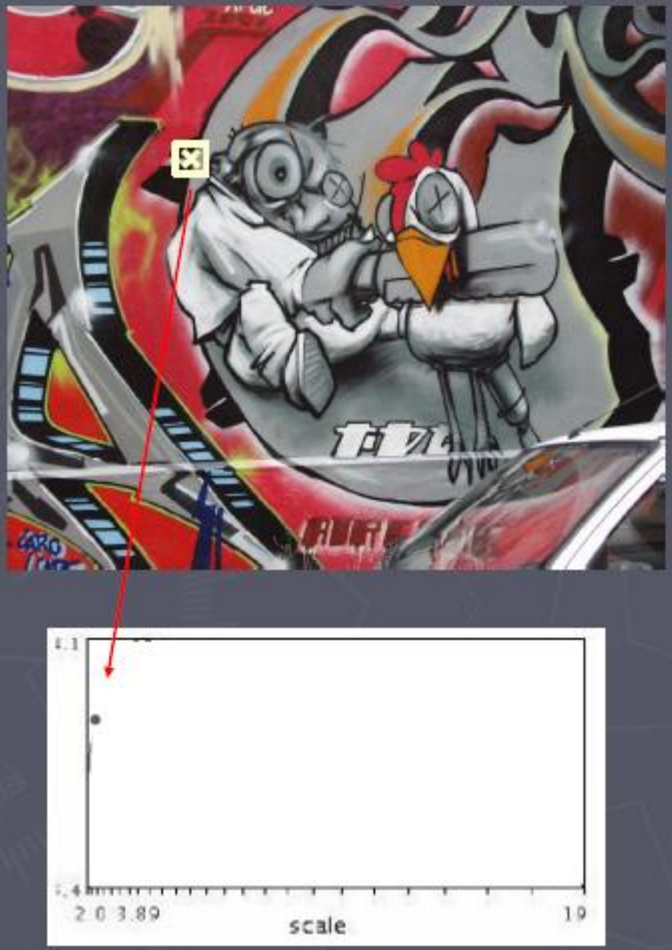


Key idea: find scale that gives local maximum of f

- f is a local maximum in both position and scale
- Common definition of f: Laplacian
  (or difference between two Gaussian filtered images with different sigmas)

Automatic scale selection

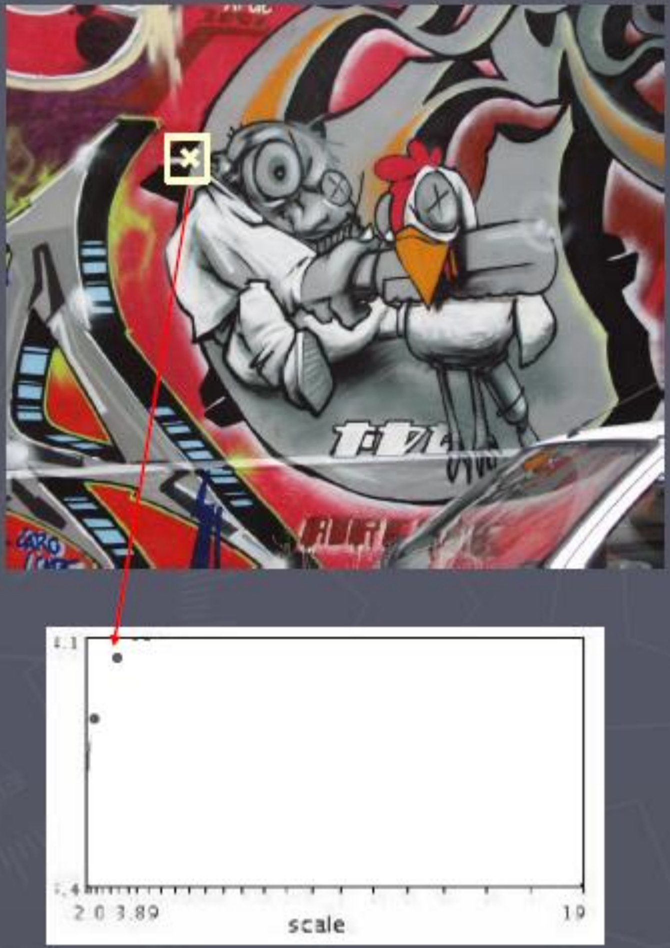
Lindeberg et al., 1996



$f(I_{i_1 \dots i_m}(x,\sigma))$

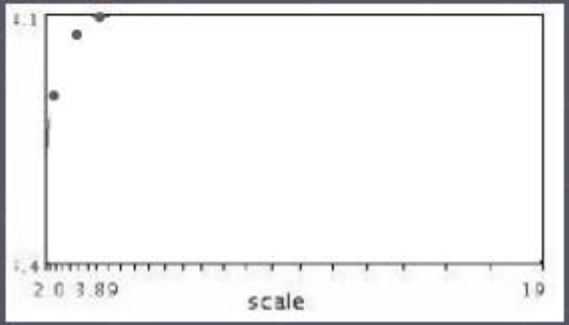
Slide from Tinne Tuytelaars

Automatic scale selection



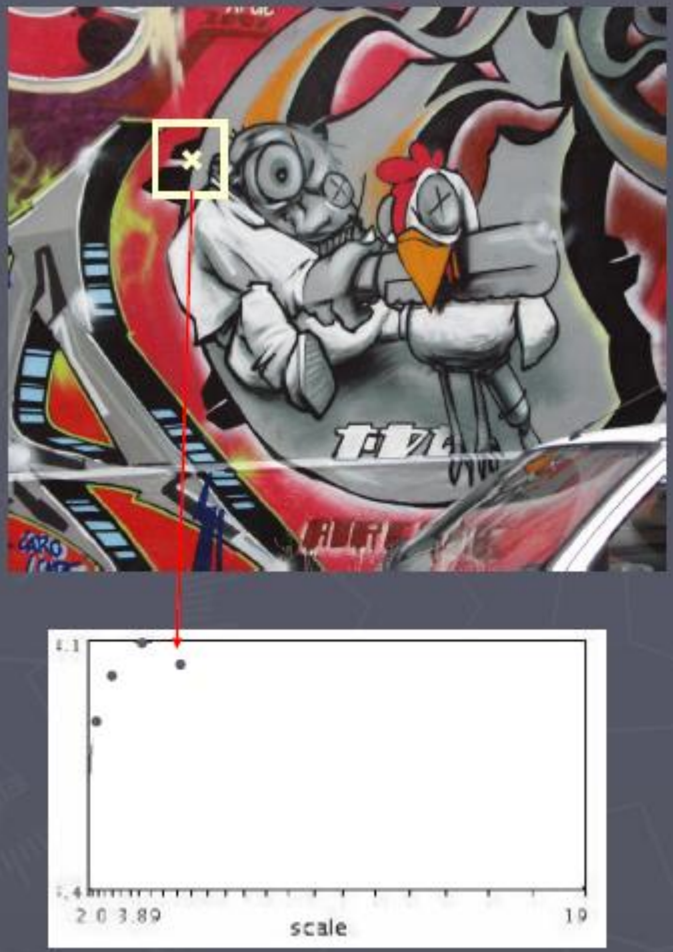
$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection



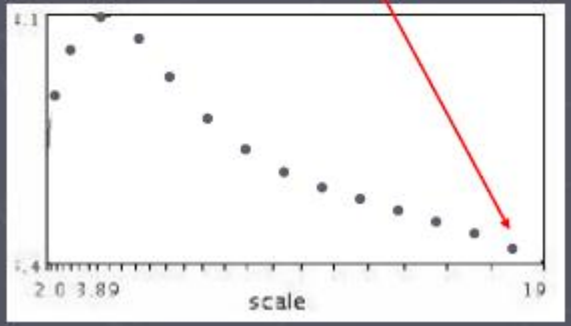
$f(I_{i_1 \dots i_m}(x,\sigma))$

Automatic scale selection



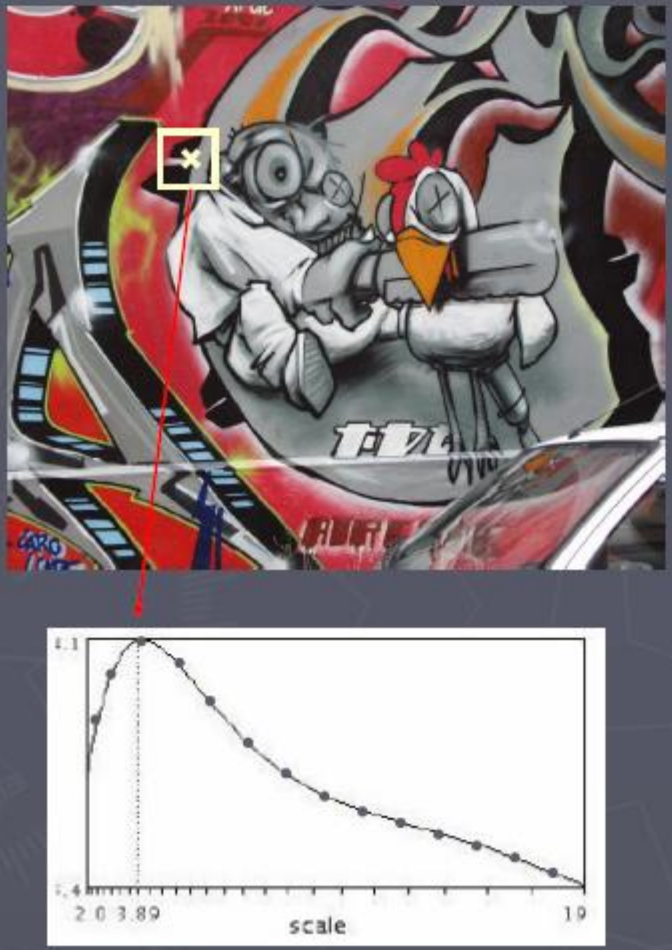
$$f(I_{i_1\ldots i_m}(x,\sigma))$$

Automatic scale selection



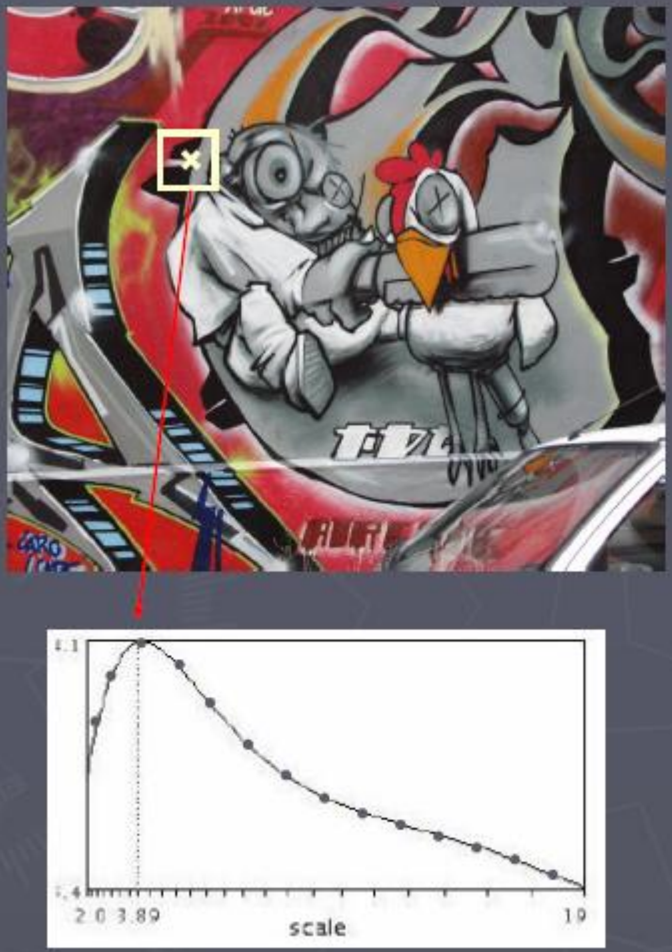
$f(I_{i_1 \ldots i_m}(x, \sigma))$

Automatic scale selection

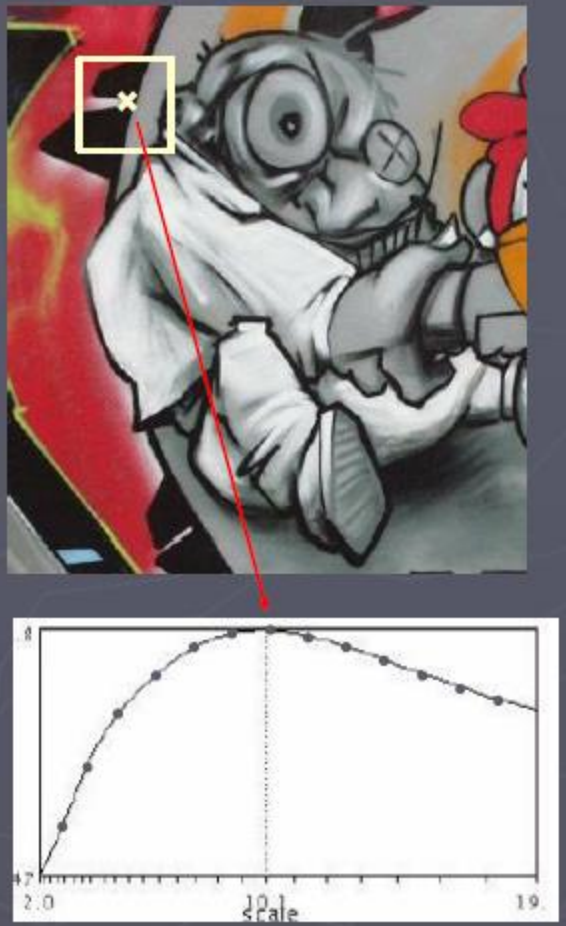


$f(I_{i_1 \ldots i_m}(x,\sigma))$

Automatic scale selection



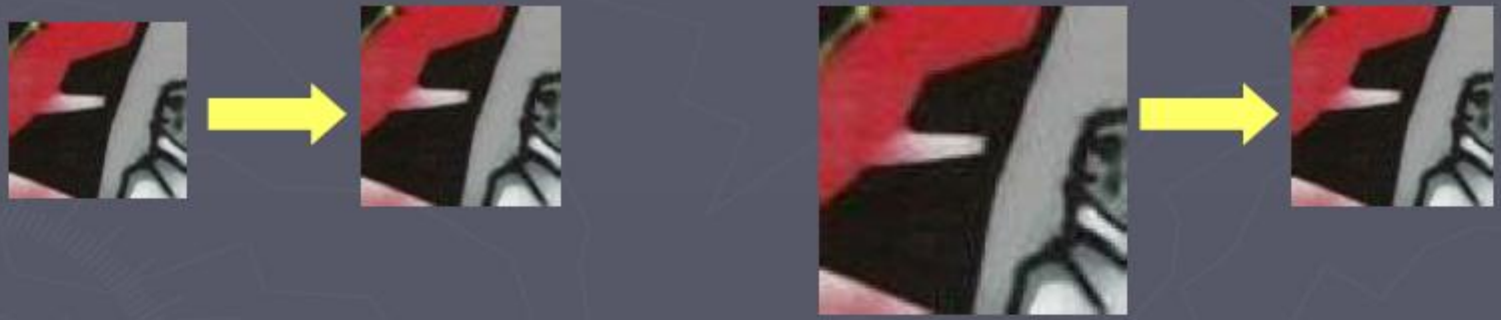
$f(I_{i_1 \dots i_m}(x, \sigma))$



$f(I_{i_1 \dots i_m}(x', \sigma'))$

Automatic scale selection

Normalize: rescale to fixed size

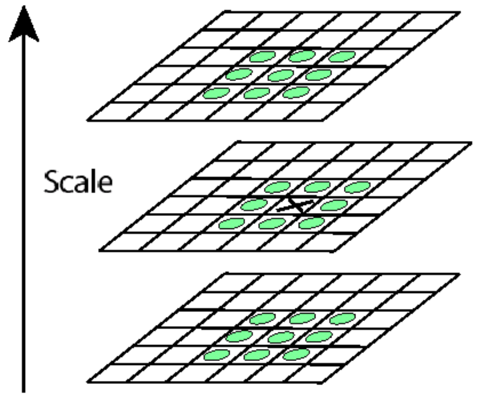


2. Key point localization

- Detect maxima and minima of difference-of-Gaussian in scale space

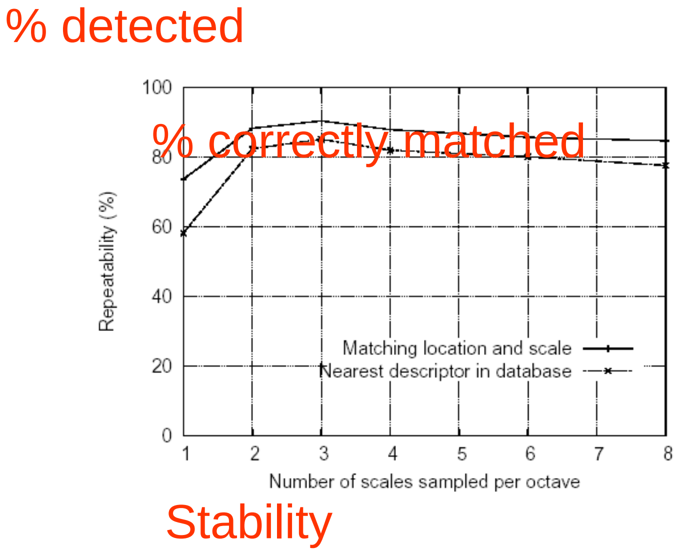
- Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below

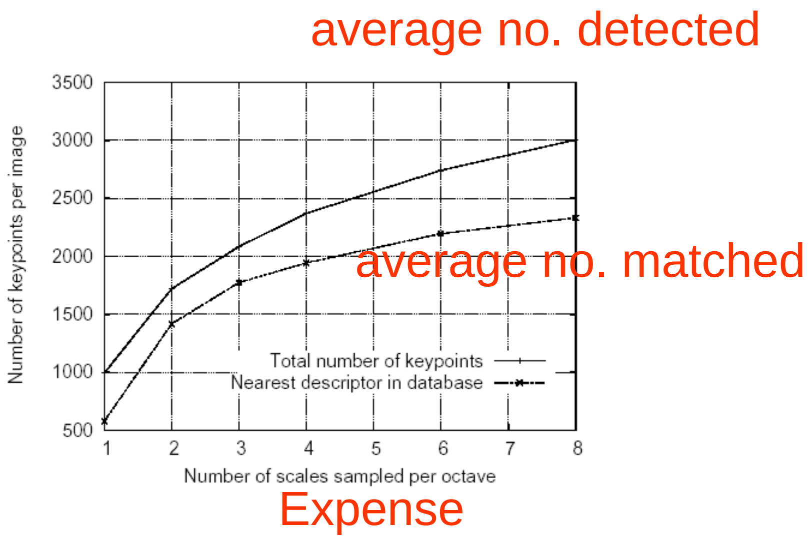
s+2 difference images.
top and bottom ignored.
s planes searched.



For each max or min found, output is the **location** and the **scale**.

Scale-space extrema detection: experimental results over 32 images that were synthetically transformed and noise added.





- Sampling in scale for efficiency
  - How many scales should be used per octave? S=?
    - More scales evaluated, more keypoints found
    - S < 3, stable keypoints increased too
    - S > 3, stable keypoints decreased
    - S = 3, maximum stable keypoints found

Keypoint localization

- Once a keypoint candidate is found, perform a detailed fit to nearby data to determi ne
  - location, scale, and ratio of principal curvatures
- In initial work, keypoints were found at location and scale of a central sample point.
- In newer work, they fit a 3D quadratic function to improve interpolation accuracy.
- The Hessian matrix was used to eliminate edge responses.

Eliminating the Edge Response

- Reject flats:
  - $|D(\hat{\mathbf{x}})|$ < 0.03
- Reject edges:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Let α be the eigenvalue with larger magnitude and β the smaller.

$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

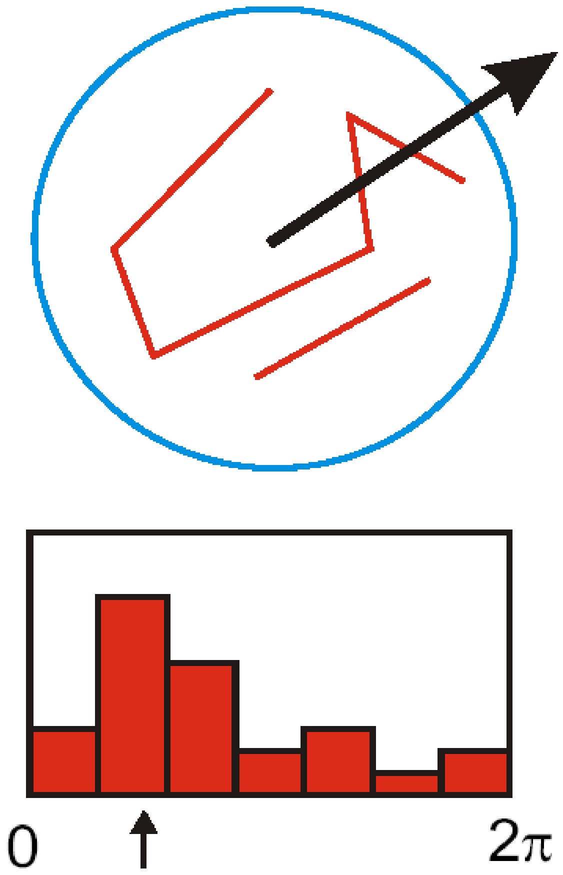
Let r = α/β.
So α = rβ

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

$(r+1)^2/r$ is at a min when the 2 eigenvalues are equal.

  - r < 10
- What does this look like?

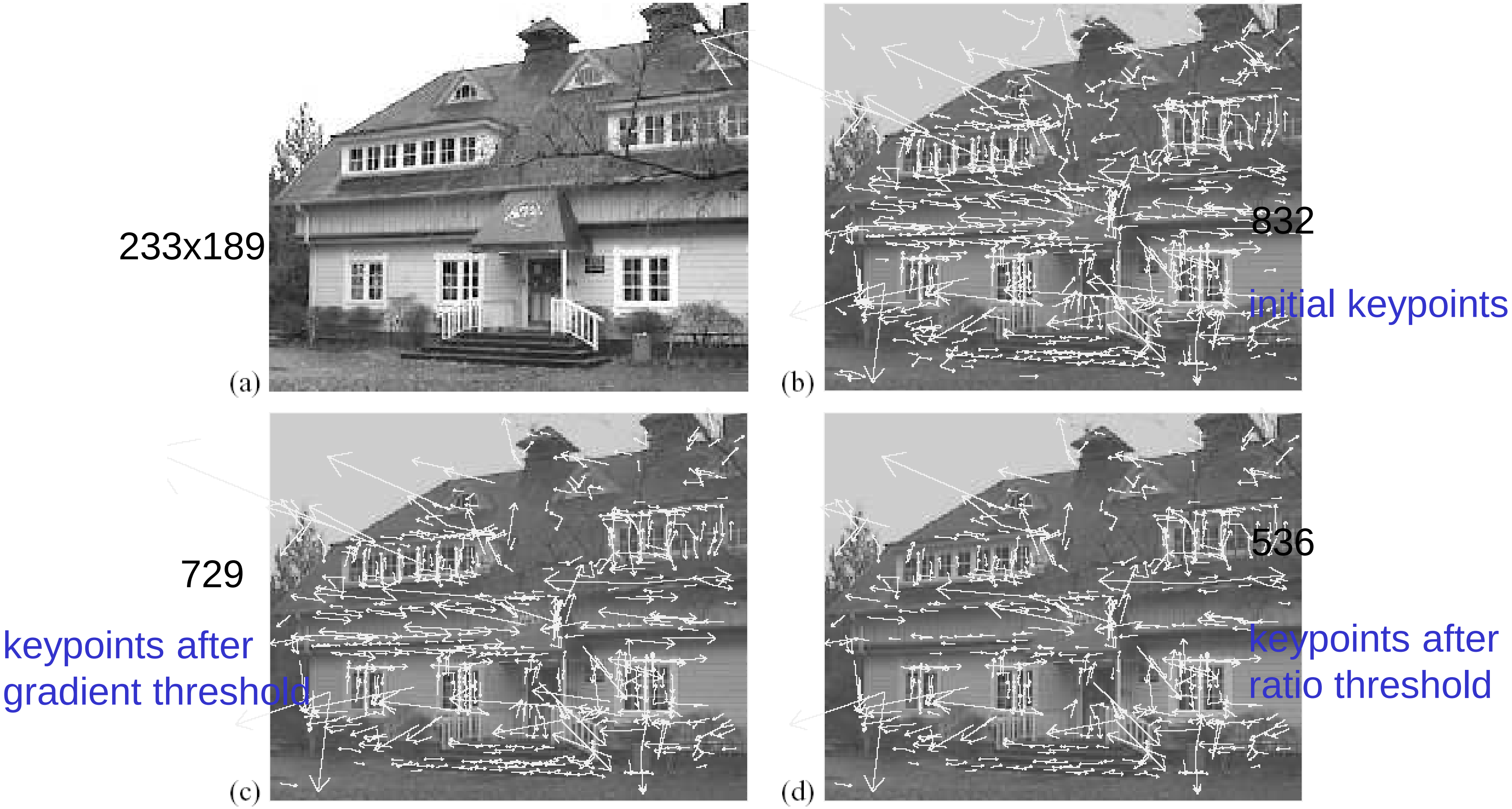
3. Orientation assignment



- Create histogram of local gradient directions at selected scale
- Assign canonical orientation at peak of smoothed histogram
- Each key specifies stable 2D coordinates (x, y, scale,orientation)

If 2 major orientations, use both.

Keypoint localization with orientation



4. Keypoint Descriptors

- At this point, each keypoint has
  - location
  - scale
  - orientation
- Next is to compute a descriptor for the local image region about each keypoint that is
  - highly distinctive
  - invariant as possible to variations such as changes in viewpoint and illumination

Normalization

- Rotate the window to standard orientation
- Scale the window size based on the scale at which the point was found.

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

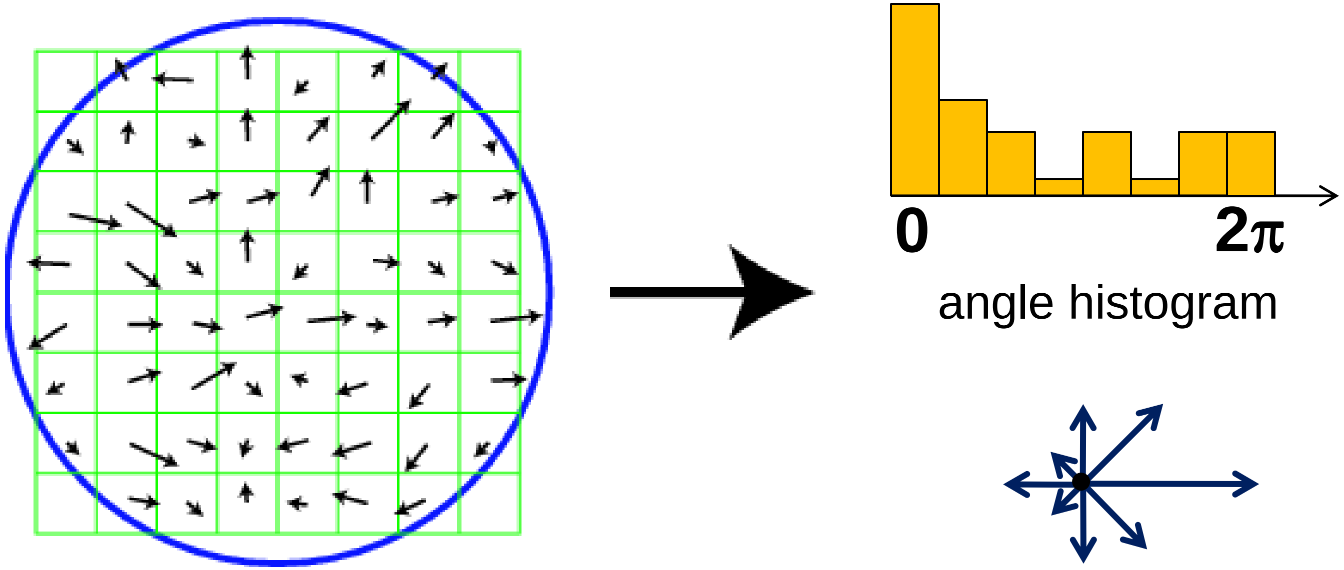
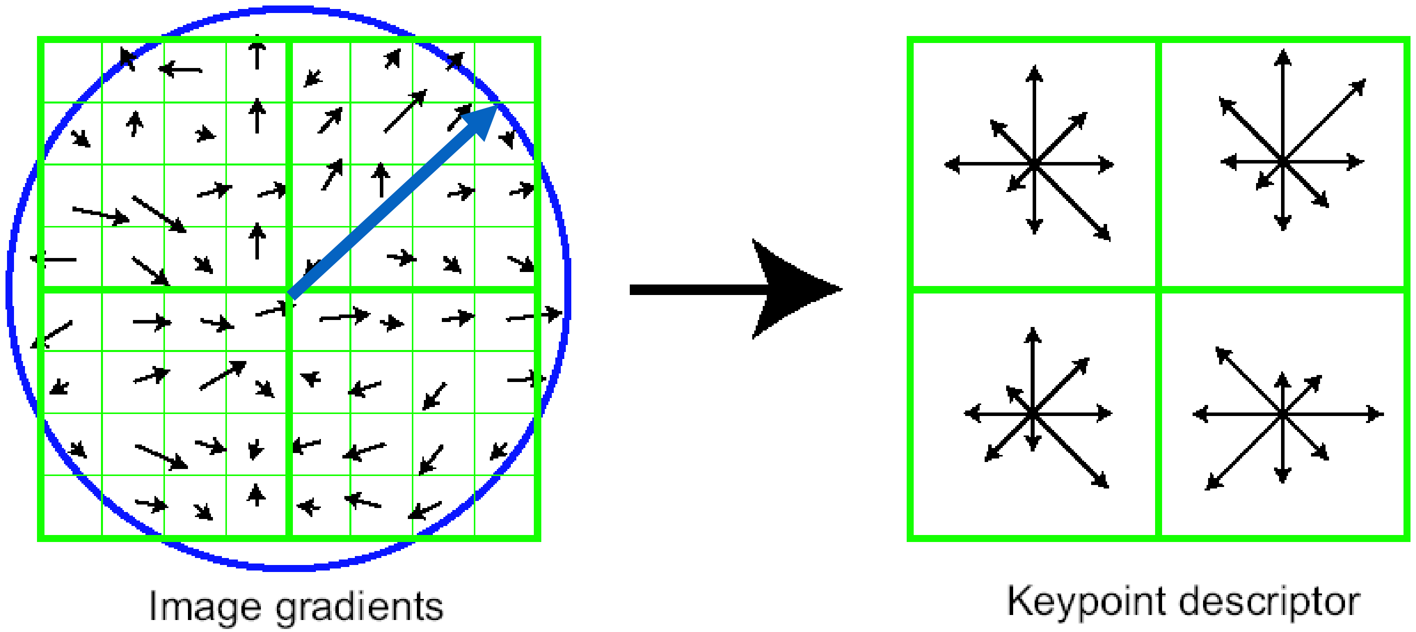


Image gradients

angle histogram

Adapted from slide by David Lowe

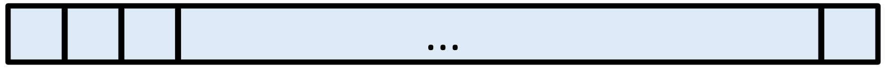
Lowe's Keypoint Descriptor
(shown with 2 X 2 descriptors over 8 X 8)



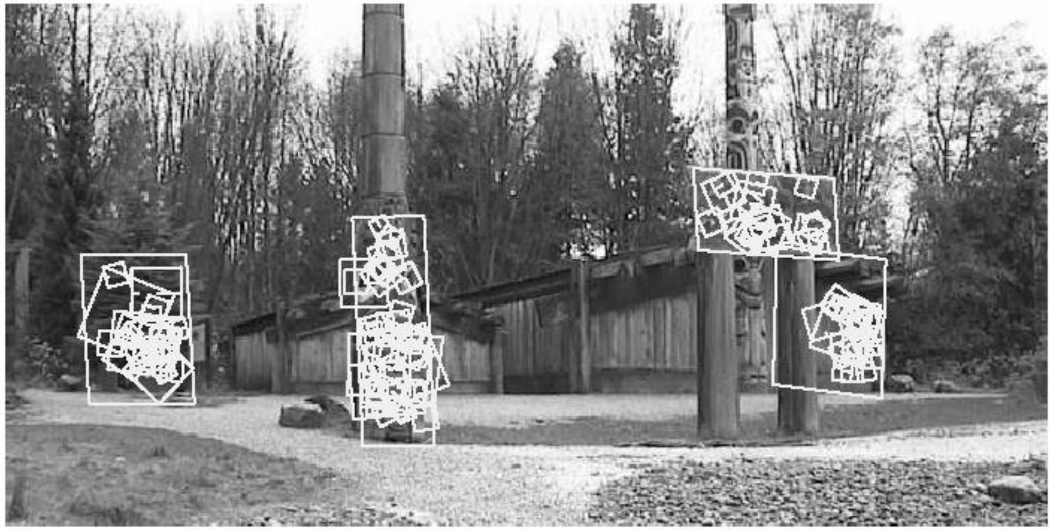
In experiments, 4x4 arrays of 8 bin histogram is used, a total of 128 features for one keypoint

Lowe's Keypoint Descriptor

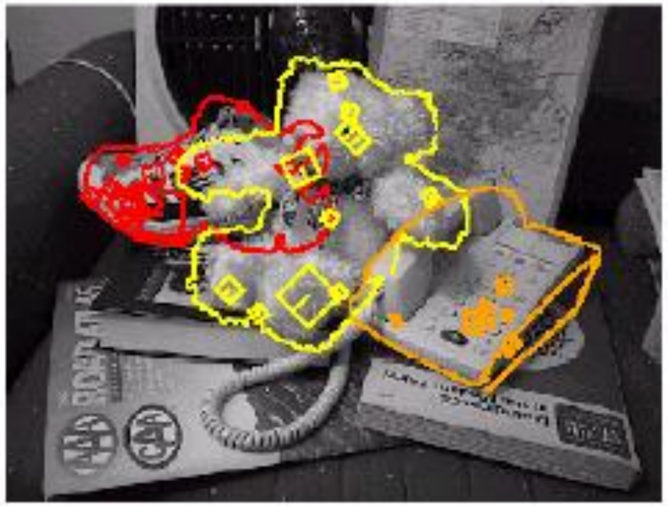
- use the normalized region about the keypoint
- compute gradient magnitude and orientation at each point in the region
- weight them by a Gaussian window overlaid on the circle
- create an orientation histogram over the 4 X 4 subregions of the window
- 4 X 4 descriptors over 16 X 16 sample array were used in practice. 4 X 4 times 8 directions gives a vector of 128 values.



Using SIFT for Matching “Objects”



Using SIFT for Matching “Objects”



Uses for SIFT

- Feature points are used also for:
  - Image alignment (homography, fundamental matrix)
  - 3D reconstruction (e.g. Photo Tourism)
  - Motion tracking
  - Object recognition
  - Indexing and database retrieval
  - Robot navigation
  - … many others

[Photo Tourism: Snavely et al. SIGGRAPH 2006]

Corner Detection (Harris corner, FAST)

```python
import cv2
import numpy as np

img = cv2.imread('../data/scenetext01.jpg', cv2.IMREAD_COLOR)
corners = cv2.cornerHarris(cv2.cvtColor(img, cv2.COLOR_BGR2GRAY), 2, 3, 0.04)

corners = cv2.dilate(corners, None)

show_img = np.copy(img)
show_img[corners>0.1*corners.max()]=[0,0,255]

corners = cv2.normalize(corners, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
show_img = np.hstack((show_img, cv2.cvtColor(corners, cv2.COLOR_GRAY2BGR)))

cv2.imshow('Harris corner detector', show_img)
if cv2.waitKey(0) == 27:
    cv2.destroyAllWindows()

fast = cv2.FastFeatureDetector_create(30, True, cv2.FAST_FEATURE_DETECTOR_TYPE_9_16)
kp = fast.detect(img)

show_img = np.copy(img)
for p in cv2.KeyPoint_convert(kp):
    cv2.circle(show_img, tuple(p), 2, (0, 255, 0), cv2.FILLED)

cv2.imshow('FAST corner detector', show_img)
if cv2.waitKey(0) == 27:
    cv2.destroyAllWindows()

fast.setNonmaxSuppression(False)
kp = fast.detect(img)

for p in cv2.KeyPoint_convert(kp):
    cv2.circle(show_img, tuple(p), 2, (0, 255, 0), cv2.FILLED)

cv2.imshow('FAST corner detector', show_img)
if cv2.waitKey(0) == 27:
    cv2.destroyAllWindows()
```

Corner Detection (Good Feature to Track)

$$R = min(\lambda_1, \lambda_2)$$

```python
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('../data/Lena.png', cv2.IMREAD_GRAYSCALE)

corners = cv2.goodFeaturesToTrack(img, 100, 0.05, 10)

for c in corners:
    x, y = c[0]
    cv2.circle(img, (x, y), 5, 255, -1)
plt.figure(figsize=(10, 10))
plt.imshow(img, cmap='gray')
plt.tight_layout()
plt.show()
```

Draw Keypoints, Descriptors, and Matches

```
import cv2
import numpy as np
import random

img = cv2.imread('../data/scenetext01.jpg', cv2.IMREAD_COLOR)

fast = cv2.FastFeatureDetector_create(160, True, cv2.FAST_FEATURE_DETECTOR_TYPE_9_16)
keyPoints = fast.detect(img)

for kp in keyPoints:
    kp.size = 100*random.random()
    kp.angle = 360*random.random()

matches = []
for i in range(len(keyPoints)):
    matches.append(cv2.DMatch(i, i, 1))

show_img = cv2.drawKeypoints(img, keyPoints, None, (255, 0, 255))

cv2.imshow('Keypoints', show_img)
cv2.waitKey()
cv2.destroyAllWindows()

show_img = cv2.drawKeypoints(img, keyPoints, None, (0, 255, 0),
                             cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imshow('Keypoints', show_img)
cv2.waitKey()
cv2.destroyAllWindows()

show_img = cv2.drawMatches(img, keyPoints, img, keyPoints, matches, None,
                           flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imshow('Matches', show_img)
cv2.waitKey()
cv2.destroyAllWindows()
```

Detecting scale invariant keypoints

```python
import cv2
import numpy as np

img0 = cv2.imread('../data/Lena.png', cv2.IMREAD_COLOR)
img1 = cv2.imread('../data/Lena_rotated.png', cv2.IMREAD_COLOR)
img1 = cv2.resize(img1, None, fx=0.75, fy=0.75)
img1 = np.pad(img1, ((64,)*2, (64,)*2, (0,)*2), 'constant', constant_values=0)
imgs_list = [img0, img1]

detector = cv2.xfeatures2d.SIFT_create(50)

for i in range(len(imgs_list)):
    keypoints, descriptors = detector.detectAndCompute(imgs_list[i], None)

    imgs_list[i] = cv2.drawKeypoints(imgs_list[i], keypoints, None, (0, 255, 0),
                                     flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imshow('SIFT keypoints', np.hstack(imgs_list))
cv2.waitKey()

cv2.destroyAllWindows()
```