Robotic Mapping & Localization

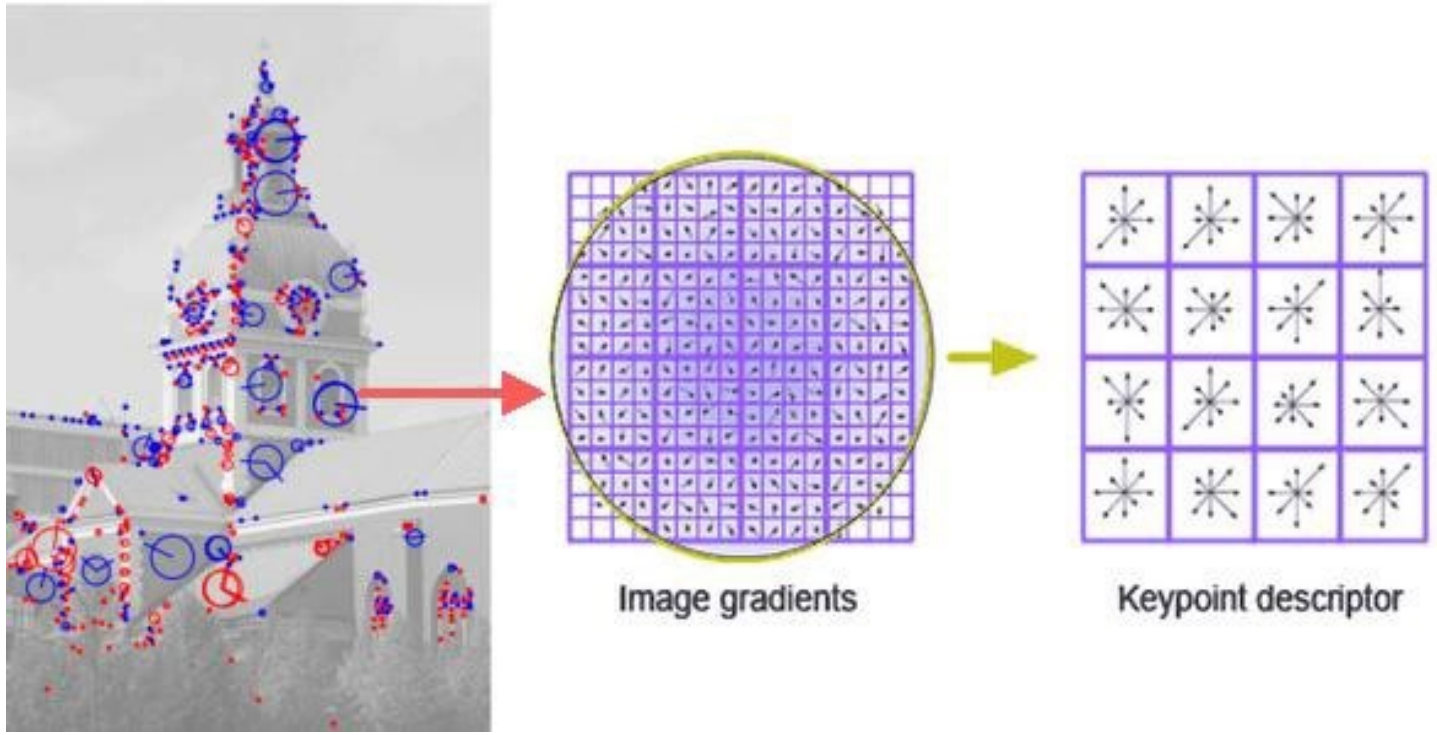
Kaveh Fathian
Assistant Professor
Computer Science Department
Colorado School of Mines

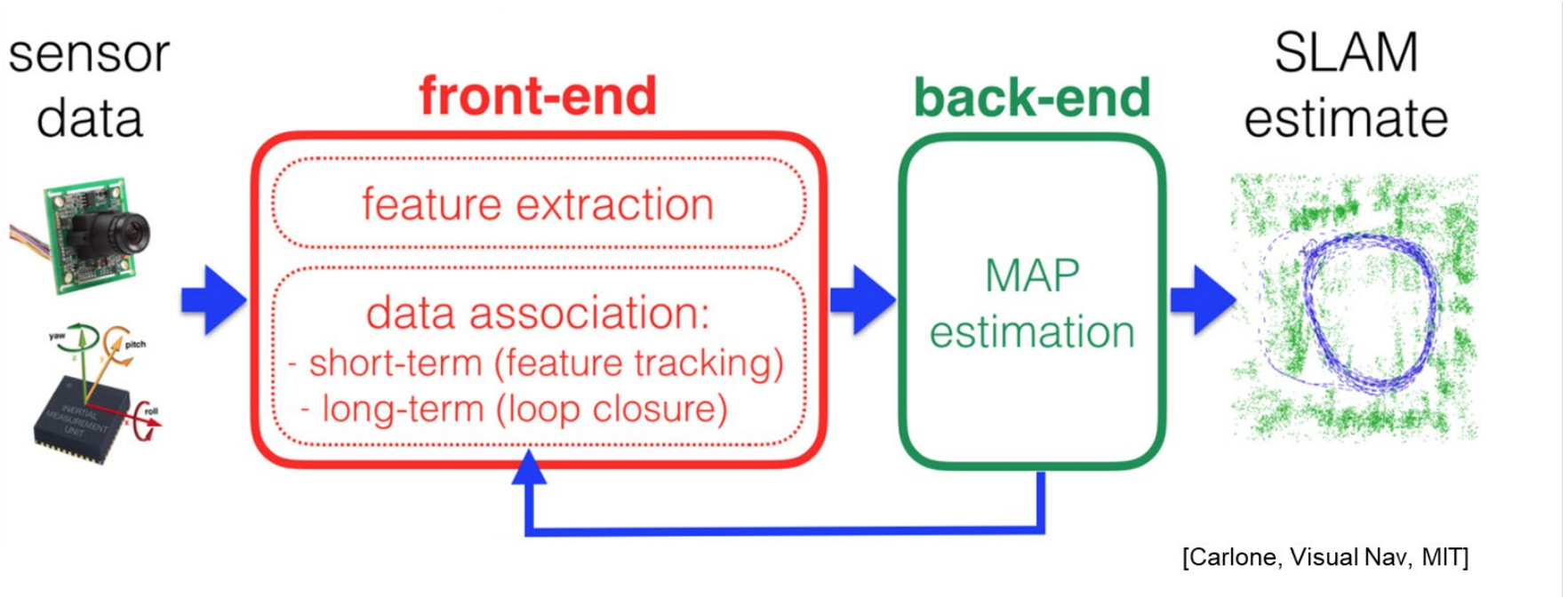
Lec04: Image Features

*Courtesy of Thomas Williams, Bill Freeman, Antonio Torralba, Cyrill Stachniss

Lecture Outline

- **Image Features**
  - Filters
  - Features
  - Descriptors
  - References

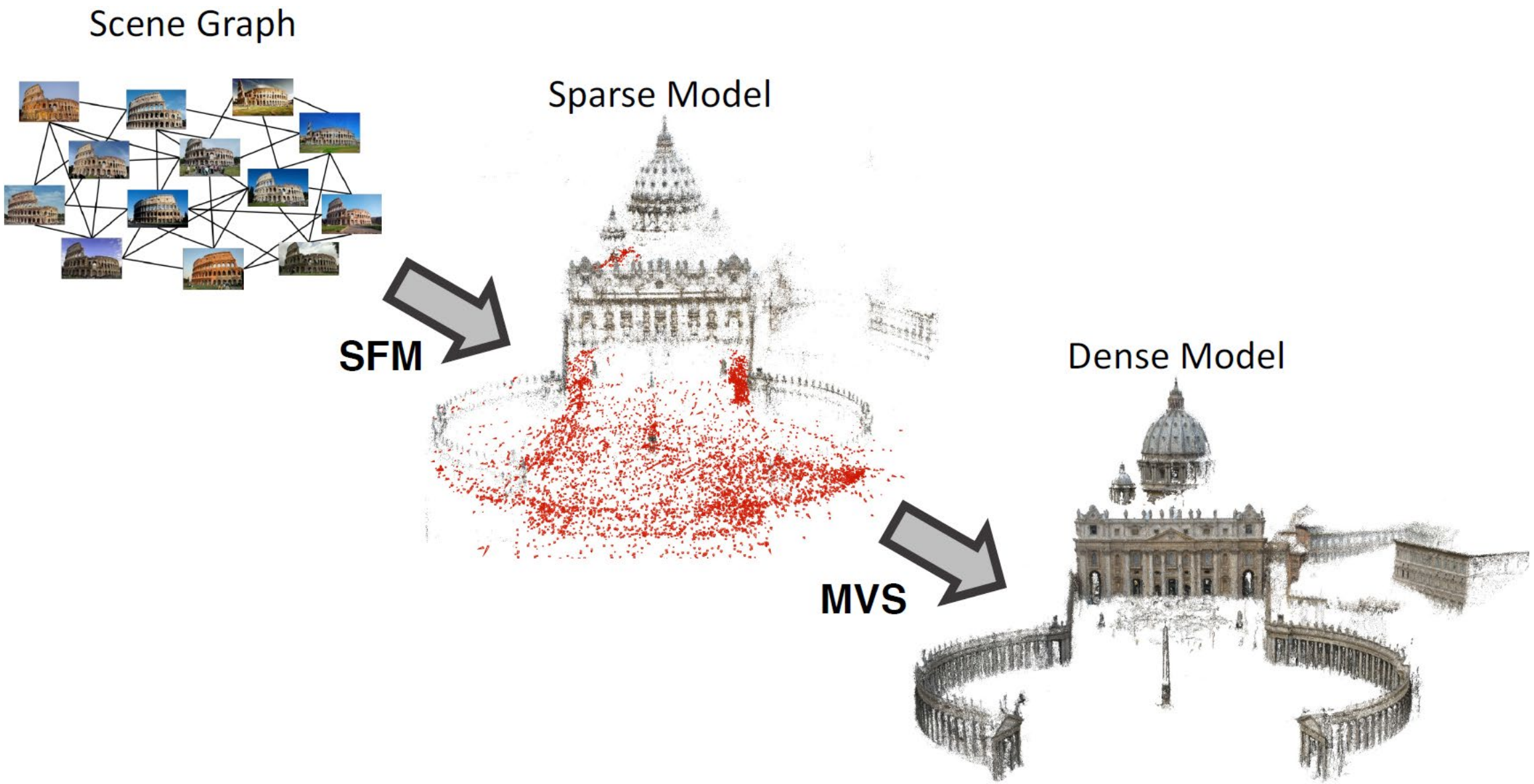




[Carlone, Visual Nav, MIT]

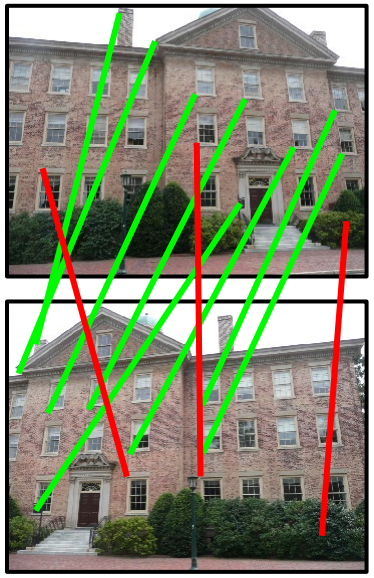
- A SLAM pipeline consists of
  - Sensors
  - Frontend algorithms
  - Backend algorithms

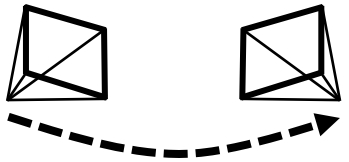
Structure from Motion (SfM)



Simple SfM/SLAM Pipeline

1. Feature Extraction → 2. Feature Matching → 3. Geometric Verification → 4. Triangulation





Robust Two-View Geometry Estimation

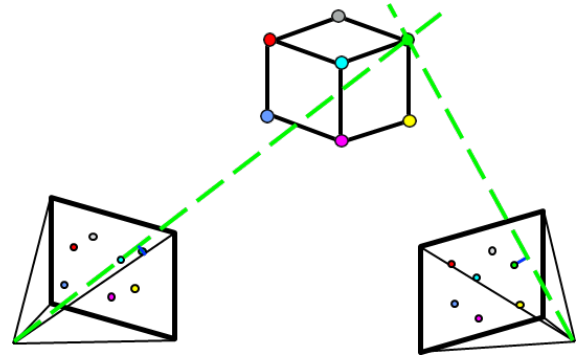
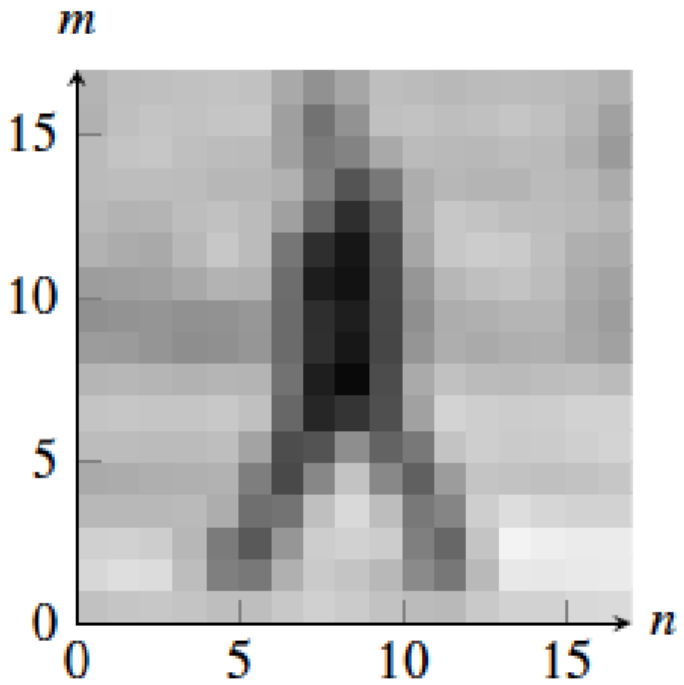


Image Filters

Image as a 2D discrete signal



A tiny person of 18 x 18 pixels

Image as a 2D discrete signal

$$\mathbf{I} = \begin{bmatrix}
160 & 175 & 171 & 168 & 168 & 172 & 164 & 158 & 167 & 173 & 167 & 163 & 162 & 164 & 160 & 159 & 163 & 162 \\
149 & 164 & 172 & 175 & 178 & 179 & 176 & 118 & 97 & 168 & 175 & 171 & 169 & 175 & 176 & 177 & 165 & 152 \\
161 & 166 & 182 & 171 & 170 & 177 & 175 & 116 & 109 & 169 & 177 & 173 & 168 & 175 & 175 & 159 & 153 & 123 \\
171 & 174 & 177 & 175 & 167 & 161 & 157 & 138 & 103 & 112 & 157 & 164 & 159 & 160 & 165 & 169 & 148 & 144 \\
163 & 163 & 162 & 165 & 167 & 164 & 178 & 167 & 77 & 55 & 134 & 170 & 167 & 162 & 164 & 175 & 168 & 160 \\
173 & 164 & 158 & 165 & 180 & 180 & 150 & 89 & 61 & 34 & 137 & 186 & 186 & 182 & 175 & 165 & 160 & 164 \\
152 & 155 & 146 & 147 & 169 & 180 & 163 & 51 & 24 & 32 & 119 & 163 & 175 & 182 & 181 & 162 & 148 & 153 \\
134 & 135 & 147 & 149 & 150 & 147 & 148 & 62 & 36 & 46 & 114 & 157 & 163 & 167 & 169 & 163 & 146 & 147 \\
135 & 132 & 131 & 125 & 115 & 129 & 132 & 74 & 54 & 41 & 104 & 156 & 152 & 156 & 164 & 156 & 141 & 144 \\
151 & 155 & 151 & 145 & 144 & 149 & 143 & 71 & 31 & 29 & 129 & 164 & 157 & 155 & 159 & 158 & 156 & 148 \\
172 & 174 & 178 & 177 & 177 & 181 & 174 & 54 & 21 & 29 & 136 & 190 & 180 & 179 & 176 & 184 & 187 & 182 \\
177 & 178 & 176 & 173 & 174 & 180 & 150 & 27 & 101 & 94 & 74 & 189 & 188 & 186 & 183 & 186 & 188 & 187 \\
160 & 160 & 163 & 163 & 161 & 167 & 100 & 45 & 169 & 166 & 59 & 136 & 184 & 176 & 175 & 177 & 185 & 186 \\
147 & 150 & 153 & 155 & 160 & 155 & 56 & 111 & 182 & 180 & 104 & 84 & 168 & 172 & 171 & 164 & 168 & 167 \\
184 & 182 & 178 & 175 & 179 & 133 & 86 & 191 & 201 & 204 & 191 & 79 & 172 & 220 & 217 & 205 & 209 & 200 \\
184 & 187 & 192 & 182 & 124 & 32 & 109 & 168 & 171 & 167 & 163 & 51 & 105 & 203 & 209 & 203 & 210 & 205 \\
191 & 198 & 203 & 197 & 175 & 149 & 169 & 189 & 190 & 173 & 160 & 145 & 156 & 202 & 199 & 201 & 205 & 202 \\
153 & 149 & 153 & 155 & 173 & 182 & 179 & 177 & 182 & 177 & 182 & 185 & 179 & 177 & 167 & 176 & 182 & 180
\end{bmatrix}$$

Image as a 2D discrete signal

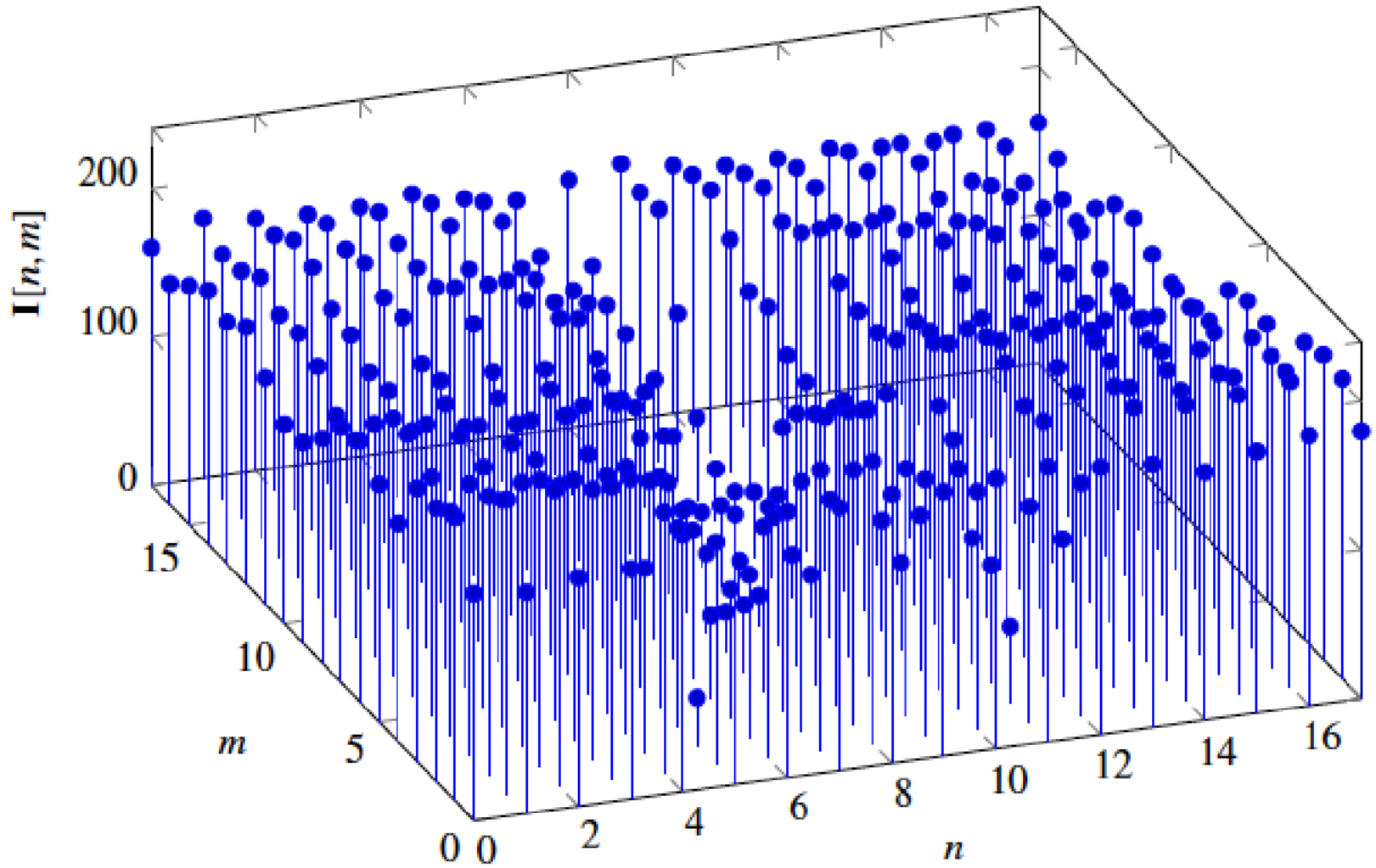
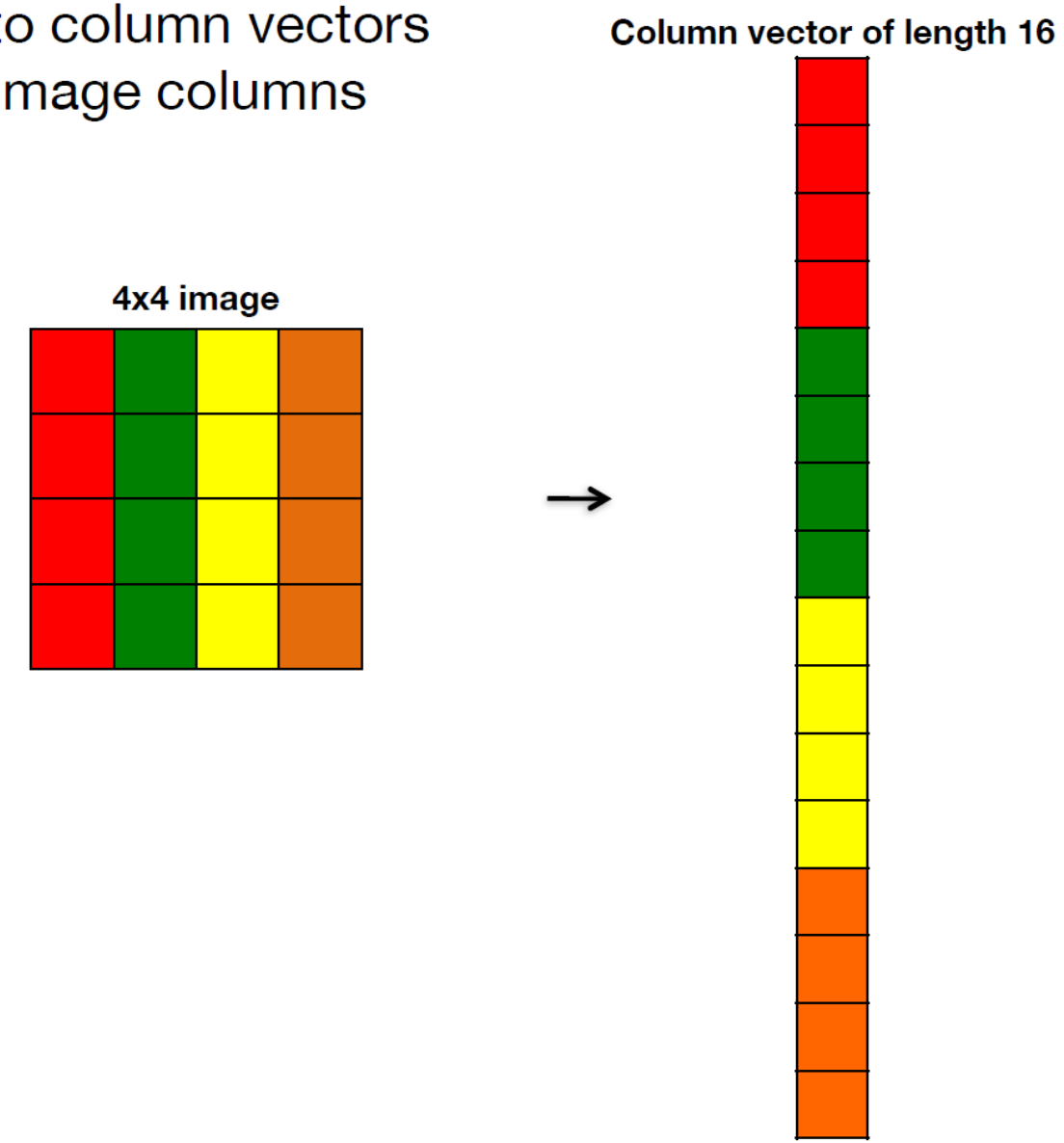
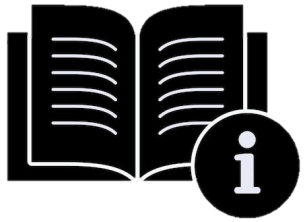


Image as a 2D discrete signal

Images are turned into column vectors by concatenating all image columns





Signal / Image space

Scalar product between two signals f, g :

$$\langle f, g \rangle = \sum_{n=0}^{N-1} f[n]\, g^*[n] = f^T g^*$$

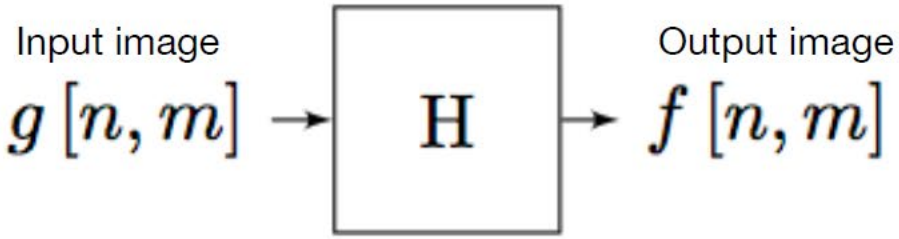
L2 norm of f :

$$E_f = \|f\|^2 = \langle f, f \rangle = \sum_{n=0}^{N-1} |f[n]|^2 = f^T f^*$$

Distance between two signals f, g :

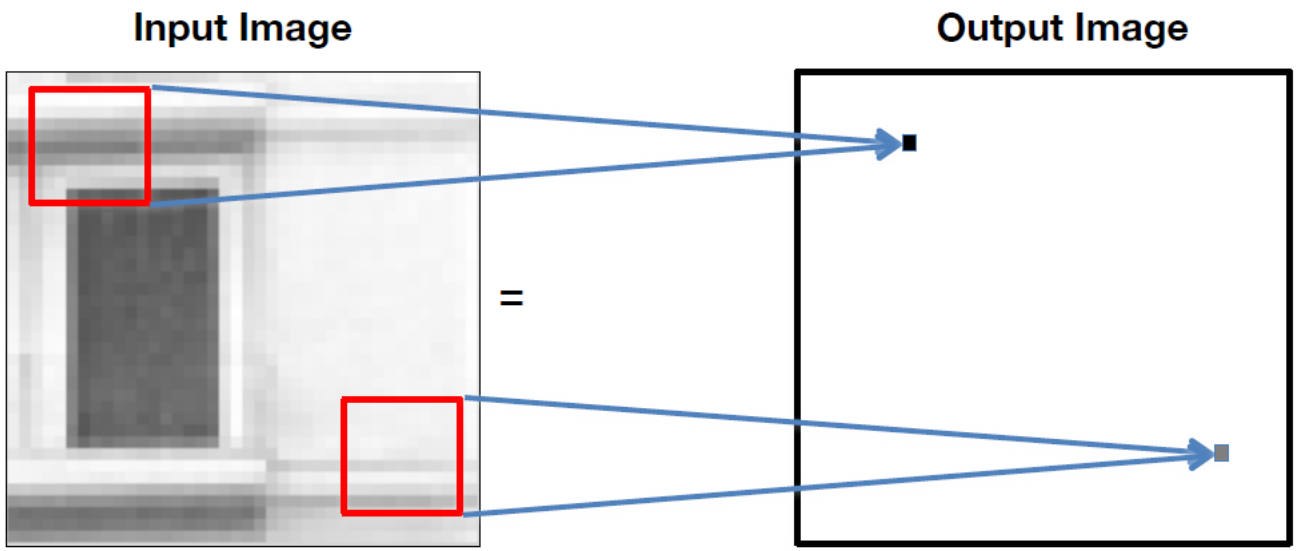
$$d_{f,g}^2 = \|f - g\|^2 = \sum_{n=0}^{N-1} |f[n] - g[n]|^2 = E_f + E_g - 2\langle f, g \rangle$$

Filtering



We want to remove unwanted sources of variation, and keep the information relevant for whatever task we need to solve

2D Convolution



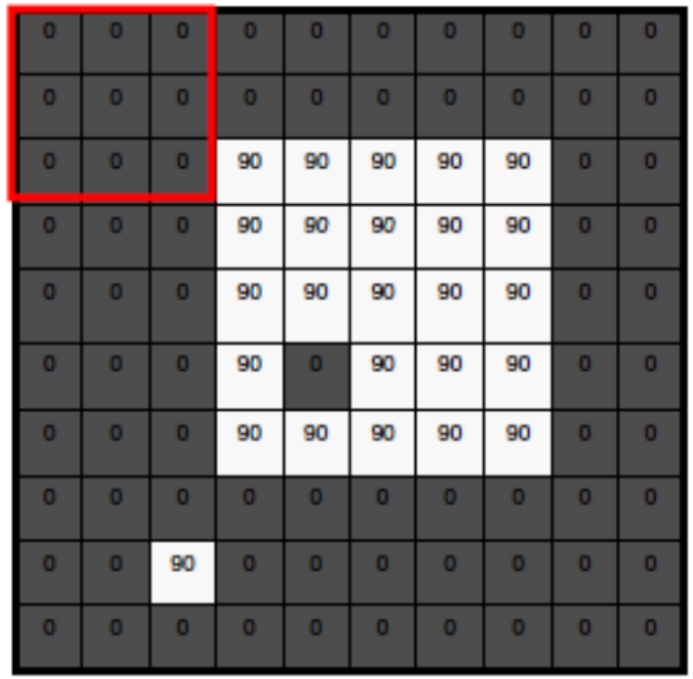
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Convolution weights

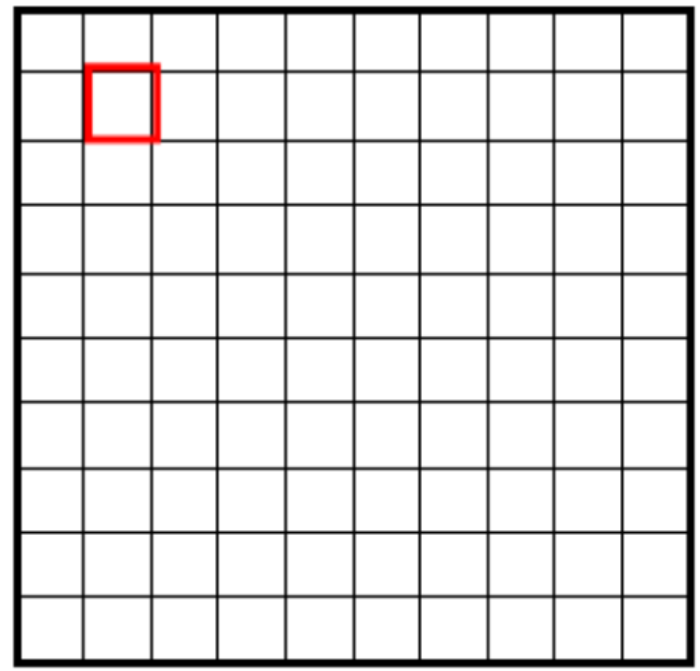
2D convolution

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1



Input image



Convolution output

2D convolution

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

	0								

Convolution output

2D convolution

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

	0	10							

Convolution output

2D convolution

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

	0	10	20						

Convolution output

2D convolution

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

	0	10	20	30					

Convolution output

2D convolution

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

	0	10	20	30	30				

Convolution output

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Convolution output

2D Convolution

g[m,n]

$\otimes$

h[m,n]

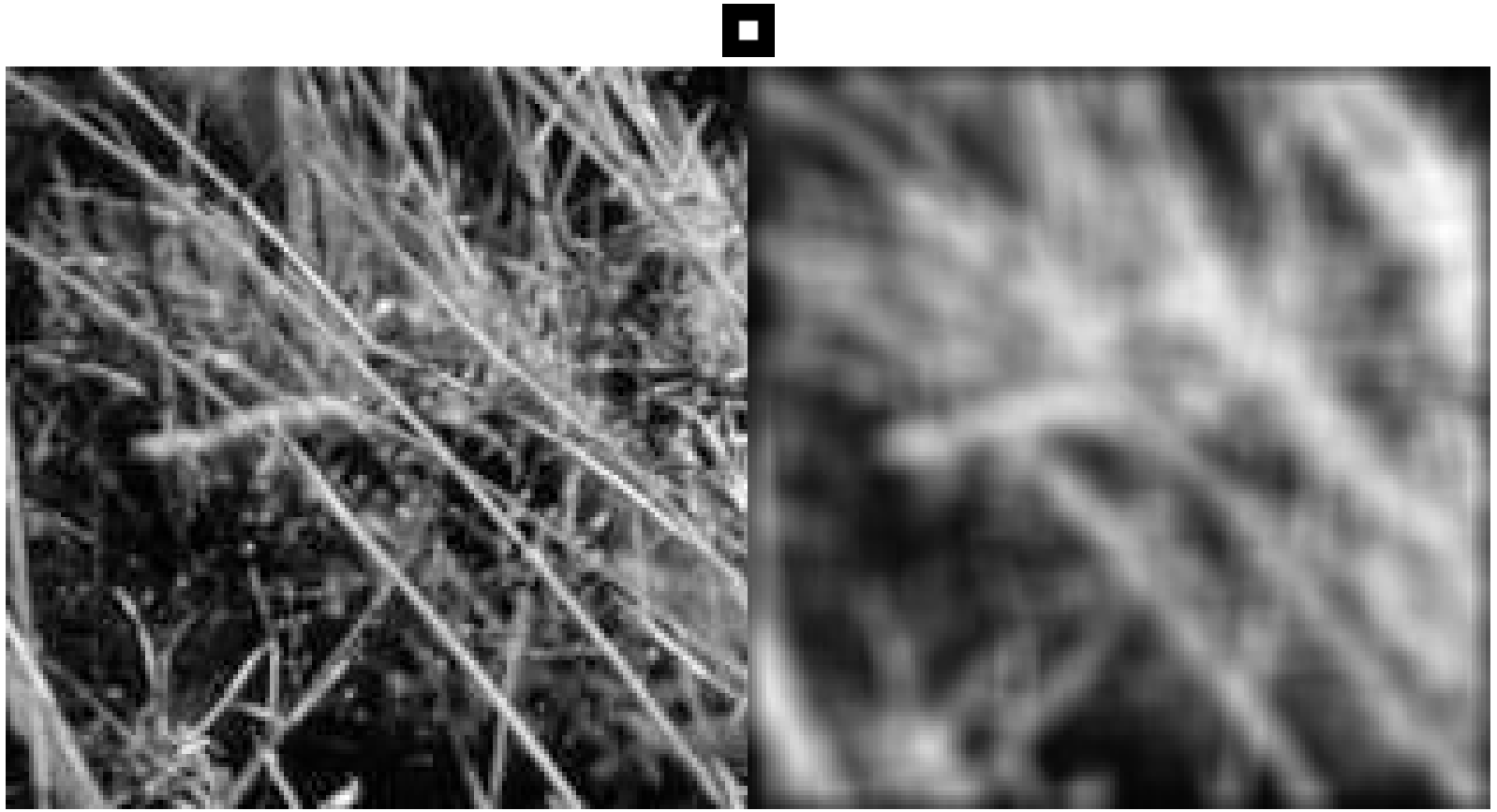
=

f[m,n]

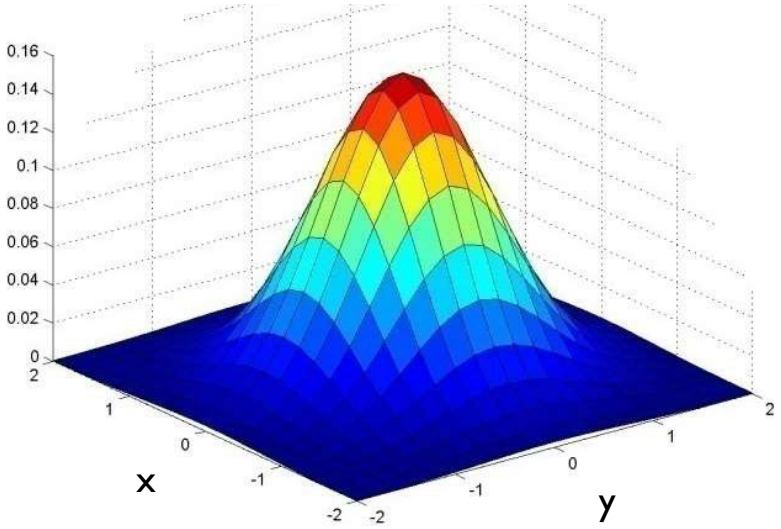
Smoothing with Box Filter

$f[\cdot,\cdot]\,\frac{1}{9}$

1	1	1
1	1	1
1	1	1

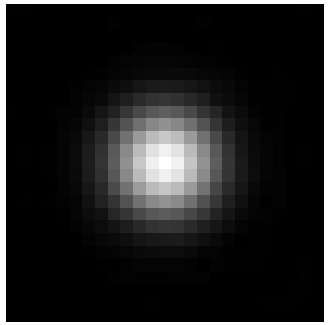


Gaussian Filter / Kernel



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

Viewed from top



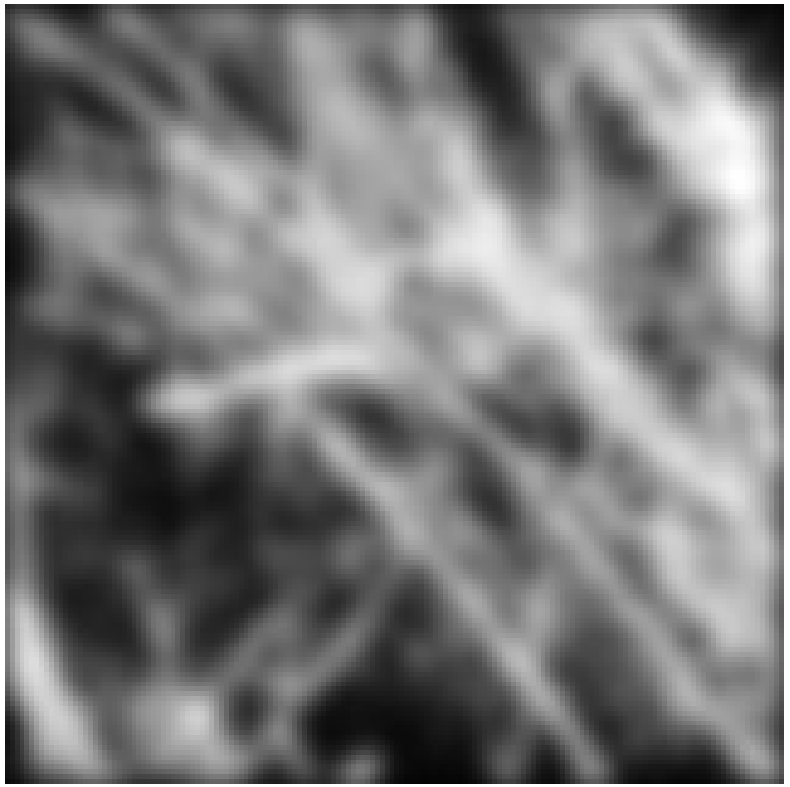
x

0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

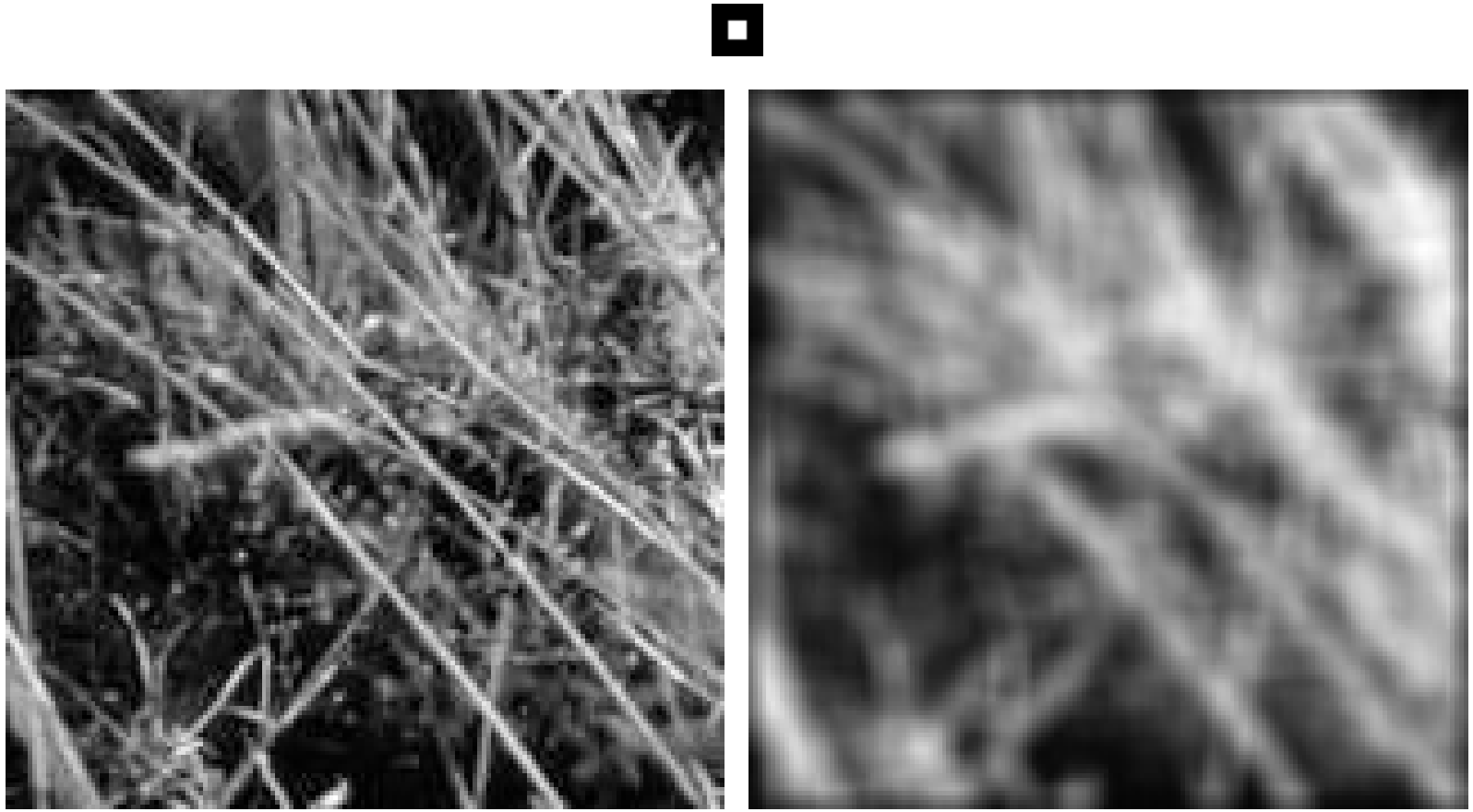
y

Filter/Kernel size 5 x 5,
Standard deviation σ = 1

Smoothing with Gaussian Filter

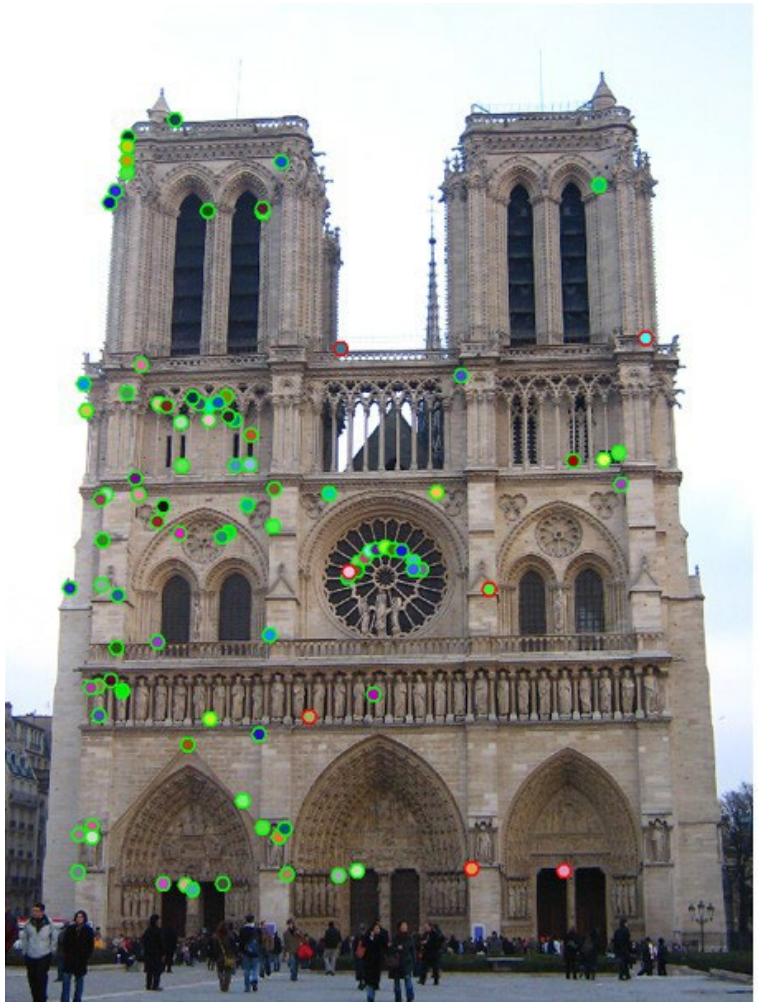


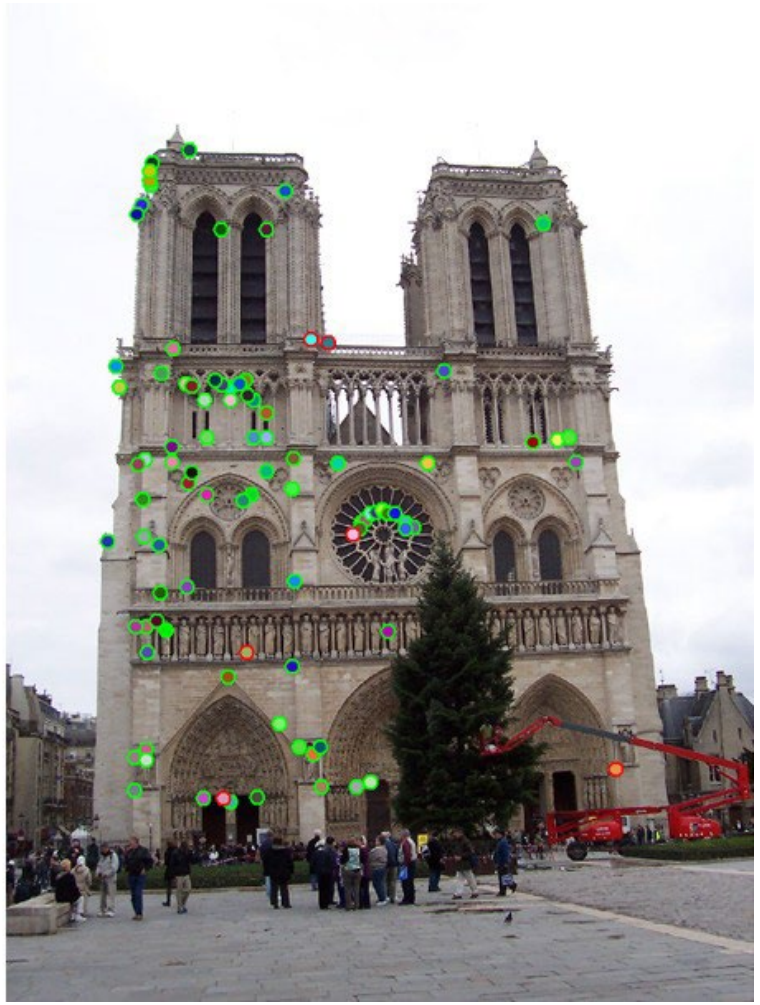
Smoothing with Box Filter



James Hays

Image Features

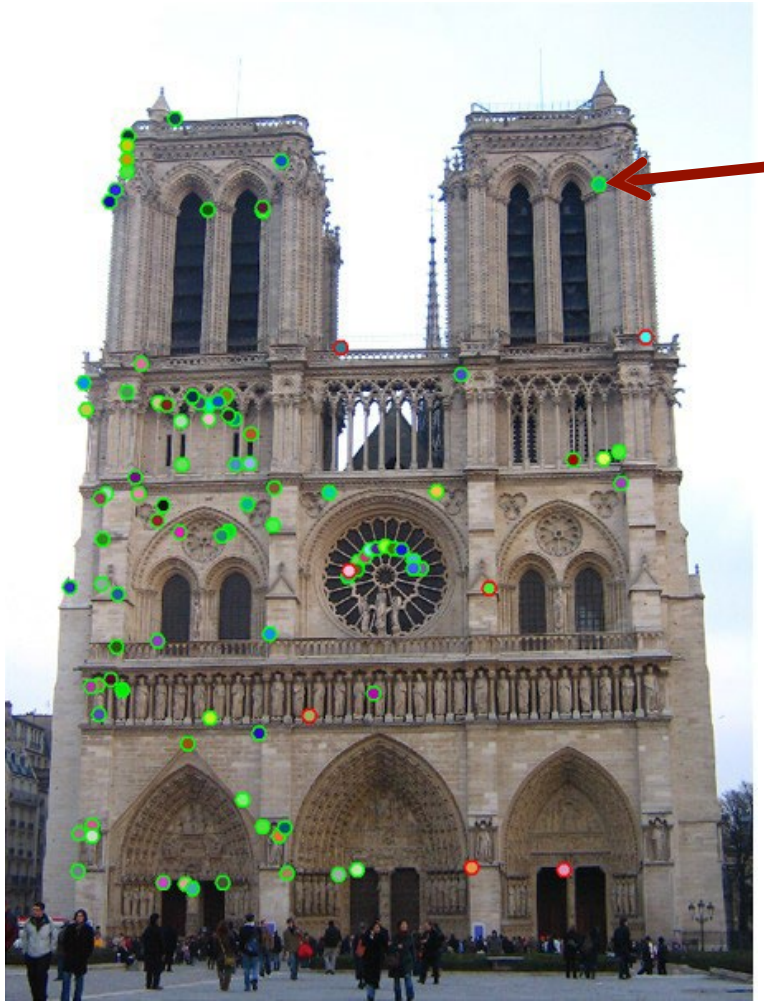




Visual Features: Keypoints and Descriptors

- **Keypoint** is a (locally) distinct location in an image
- The feature **descriptor** summarizes the local structure around the keypoint

Keypoint and Descriptor



keypoint

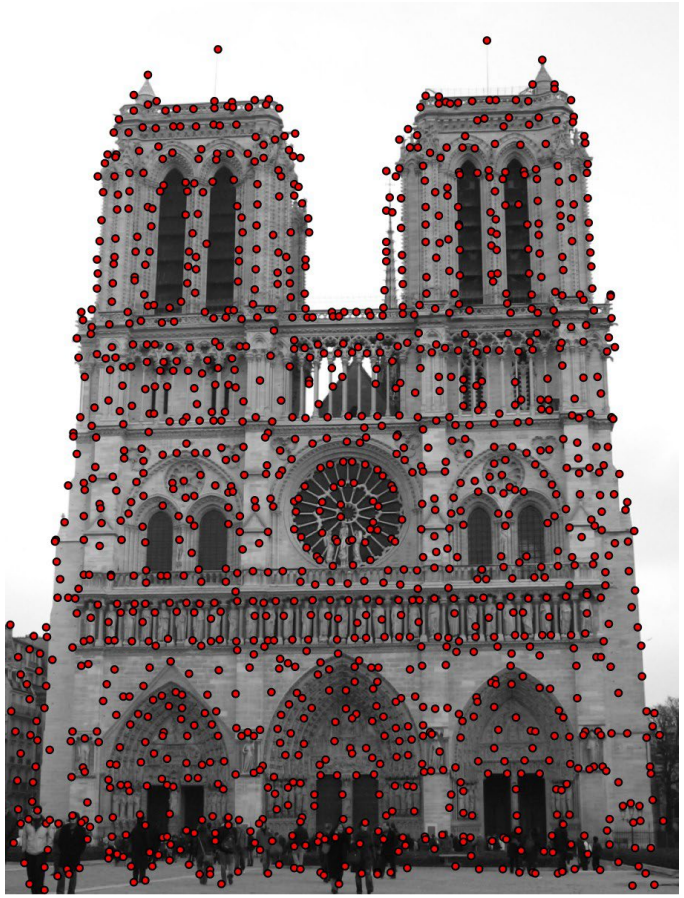
descriptor at the keypoint

$$f = \begin{bmatrix} 0.02 \\ 0.04 \\ 0.1 \\ 0.03 \\ 0 \\ \dots \end{bmatrix}$$

- **Keypoints: Finding distinct points**
  - **Harris** corners
  - **Shi-Tomasi** corner detector
  - **Difference of Gaussians**

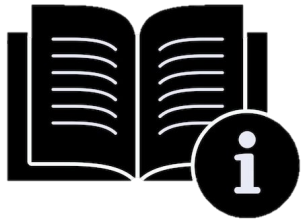
Corners

- Corners are often highly distinct points



Corners & Edges

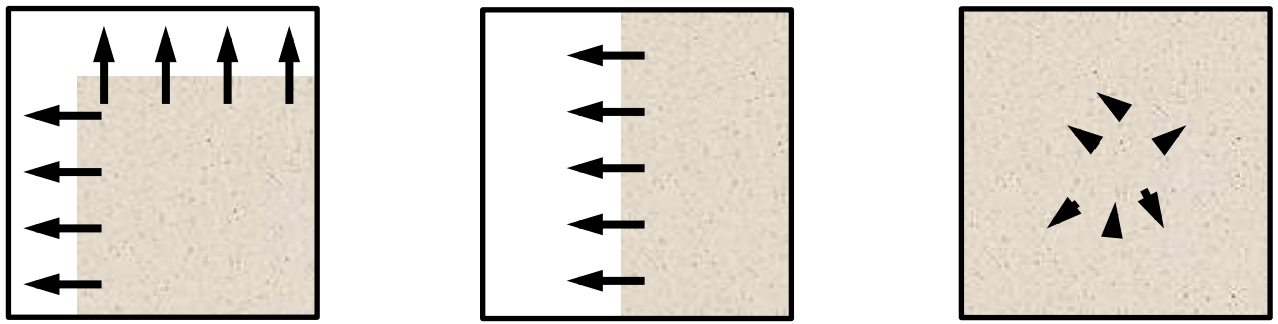
- Corners are often highly distinct points
- Corners are invariant to translation, rotation, and illumination
- **Corner** = **two edges** in roughly orthogonal directions
- **Edge** = a sudden **brightness change**

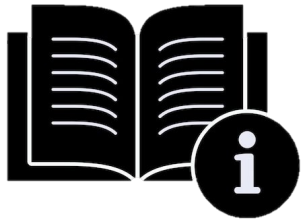


Finding Corners

- To find corners we need to **search for intensity changes** in two directions
- Compute the SSD of neighbor pixels around (x, y)

$$f(x,y) = \sum_{(u,v)\in W_{xy}} (I(u,v) - I(u+\delta u, v+\delta v))^2$$





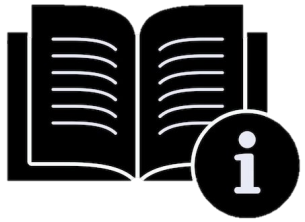
Finding Corners

- To find corners we need to **search for intensity changes** in two directions
- Compute the SSD of neighbor pixels around (x, y)

$$f(x,y) = \sum_{(u,v)\in W_{xy}} (I(u,v) - I(u+\delta u, v+\delta v))^2$$

local patch around (x,y)

sum of squared differences of image intensity values of pixels under a given shift (du, dv)



Finding Corners

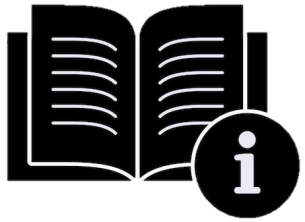
- To find corners we need to **search for intensity changes** in two directions
- Compute the SSD of neighbor pixels around (x, y)

$$f(x,y) = \sum_{(u,v)\in W_{xy}} (I(u,v) - I(u+\delta u, v+\delta v))^2$$

- Using Taylor expansion, we obtain

$$I(u+\delta u, v+\delta v) \approx I(u,v) + [J_x \; J_y] \begin{bmatrix} \delta u \\ \delta v \end{bmatrix}$$

Jacobian



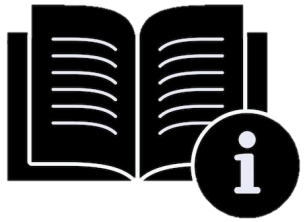
Finding Corners

- The Taylor approximation leads to

$$f(x,y) \approx \sum_{(u,v)\in W_{xy}} \left([J_x \; J_y] \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} \right)^2$$

- Written in matrix form as

$$f(x,y) \approx \sum_{(u,v)\in W_{xy}} \begin{bmatrix} \delta u \\ \delta v \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} J_x^2 & J_x J_y \\ J_x J_y & J_y^2 \end{bmatrix} \begin{bmatrix} \delta u \\ \delta v \end{bmatrix}$$



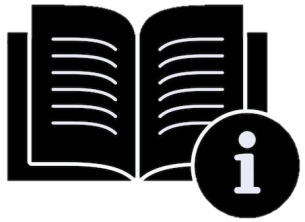
Finding Corners

- Given

$$f(x,y) \approx \sum_{(u,v)\in W_{xy}} \begin{bmatrix} \delta u \\ \delta v \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} J_x^2 & J_x J_y \\ J_x J_y & J_y^2 \end{bmatrix} \begin{bmatrix} \delta u \\ \delta v \end{bmatrix}$$

- Move the sums inside the matrix

$$f(x,y) \approx \begin{bmatrix} \delta u \\ \delta v \end{bmatrix}^{\mathsf{T}} \underbrace{\begin{bmatrix} \sum_W J_x^2 & \sum_W J_x J_y \\ \sum_W J_y J_x & \sum_W J_y^2 \end{bmatrix}}_{\textbf{structure matrix}} \begin{bmatrix} \delta u \\ \delta v \end{bmatrix}$$

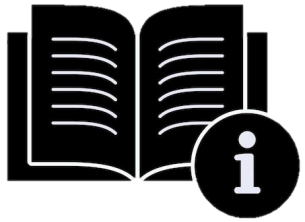


Structure Matrix

- The structure matrix is key to finding edges and corners
- It encodes the changes in image intensities in a local area

$$M = \begin{bmatrix} \sum_W J_x^2 & \sum_W J_x J_y \\ \sum_W J_y J_x & \sum_W J_y^2 \end{bmatrix}$$

- Built from the image gradients



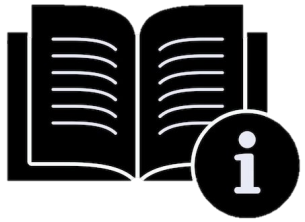
Computing the Structure Matrix

- Matrix build from the image gradients

$$M = \begin{bmatrix} \sum_W J_x^2 & \sum_W J_x J_y \\ \sum_W J_y J_x & \sum_W J_y^2 \end{bmatrix}$$

- Jacobians computed via a convolution with a gradient kernel such as Scharr or Sobel:

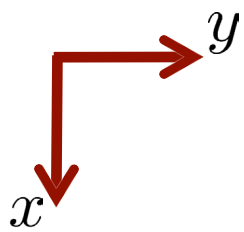
$$\begin{aligned} J_x^2 &= (D_x * I)^2 \\ J_x J_y &= (D_x * I)(D_y * I) \\ J_y^2 &= (D_y * I)^2 \end{aligned}$$



Computing the Structure Matrix

- Matrix build from the image gradients

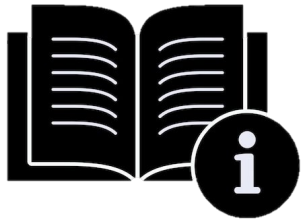
$$M = \begin{bmatrix} \sum_W J_x^2 & \sum_W J_x J_y \\ \sum_W J_y J_x & \sum_W J_y^2 \end{bmatrix}$$



- Jacobians via Scharr or Sobel Op:

$$D_x^{\text{Scharr}} = \frac{1}{32}\begin{bmatrix} 3 & 10 & 3 \\ 0 & \underline{0} & 0 \\ -3 & -10 & -3 \end{bmatrix} \qquad D_x^{\text{Sobel}} = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & \underline{0} & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

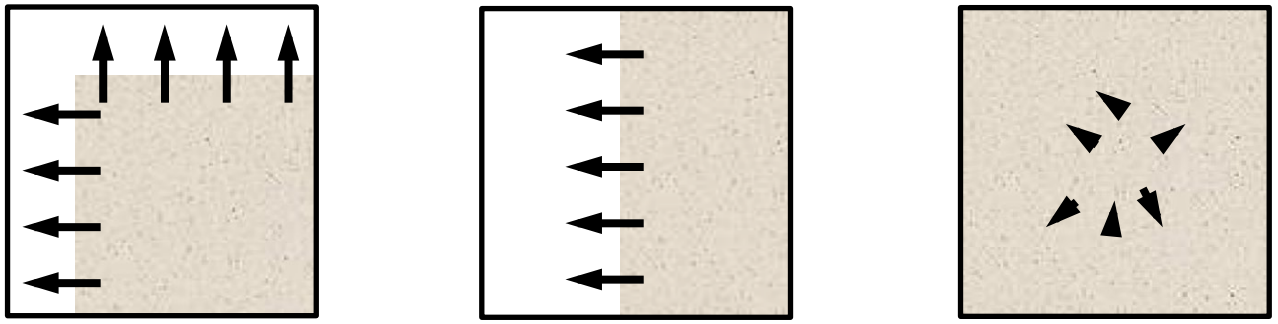
$$D_y^{\text{Scharr}} = \frac{1}{32}\begin{bmatrix} 3 & 0 & -3 \\ 10 & \underline{0} & -10 \\ 3 & 0 & -3 \end{bmatrix} \qquad D_y^{\text{Sobel}} = \frac{1}{8}\begin{bmatrix} 1 & 0 & -1 \\ 2 & \underline{0} & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

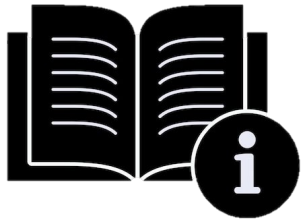


Structure Matrix

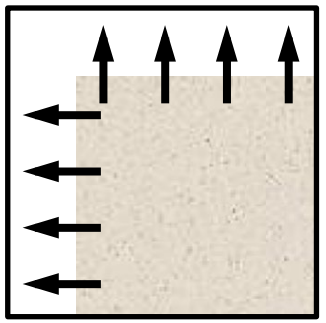
- Summarizes the dominant directions of the gradient around a point

$$M = \begin{bmatrix} \sum_W J_x^2 & \sum_W J_x J_y \\ \sum_W J_y J_x & \sum_W J_y^2 \end{bmatrix}$$

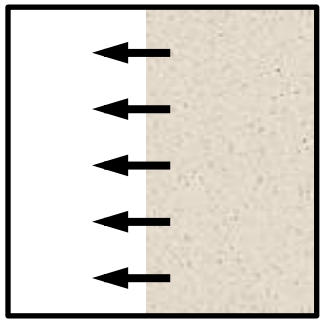




Structure Matrix Examples

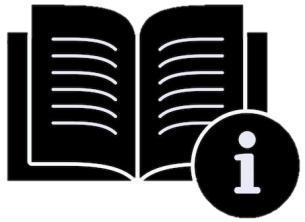


$$M = \begin{bmatrix} \gg 1 & \sim 0 \\ \sim 0 & \gg 1 \end{bmatrix}$$

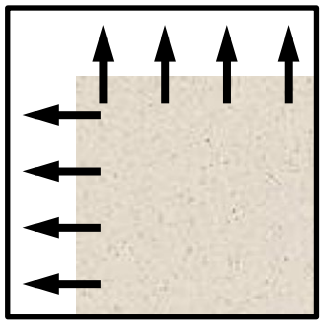


$$M = \begin{bmatrix} \sim 0 & \sim 0 \\ \sim 0 & \gg 1 \end{bmatrix}$$

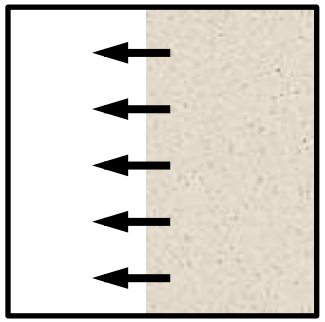
$$M = \begin{bmatrix} \sim 0 & \sim 0 \\ \sim 0 & \sim 0 \end{bmatrix}$$



Structure Matrix Examples

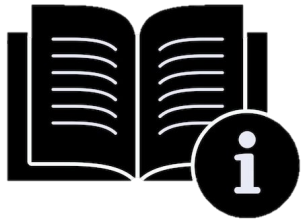


$$M = \begin{bmatrix} \gg 1 & \sim 0 \\ \sim 0 & \gg 1 \end{bmatrix}$$ **YES!**

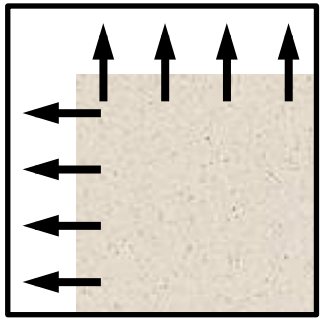


$$M = \begin{bmatrix} \sim 0 & \sim 0 \\ \sim 0 & \gg 1 \end{bmatrix}$$ **NO!**

$$M = \begin{bmatrix} \sim 0 & \sim 0 \\ \sim 0 & \sim 0 \end{bmatrix}$$ **NO!**



Corners from Structure Matrix



$$M = \begin{bmatrix} \gg 1 & \sim 0 \\ \sim 0 & \gg 1 \end{bmatrix}$$

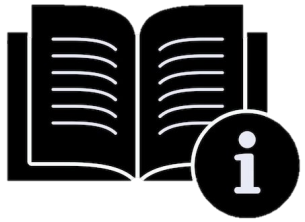
YES!

Key idea:

Considers points as corners if their structure matrix has two large Eigenvalues

Harris, Shi-Tomasi

- Similar approaches
- Proposed in
  - 1988 (Harris)
  - 1994 (Shi-Tomasi)
- All rely on the structure matrix
- Use different criterion for deciding of a point is a corner or not



Harris Corner Criterion

- Criterion

$$\begin{aligned} R &= \det(M) - k\,(\mathrm{trace}(M))^2 \\ &= \lambda_1\,\lambda_2 - k(\lambda_1 + \lambda_2)^2 \end{aligned}$$

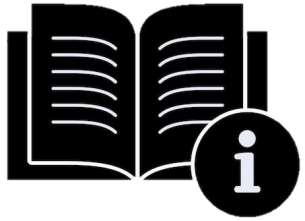
- with

$k \in [0.04, 0.06]$

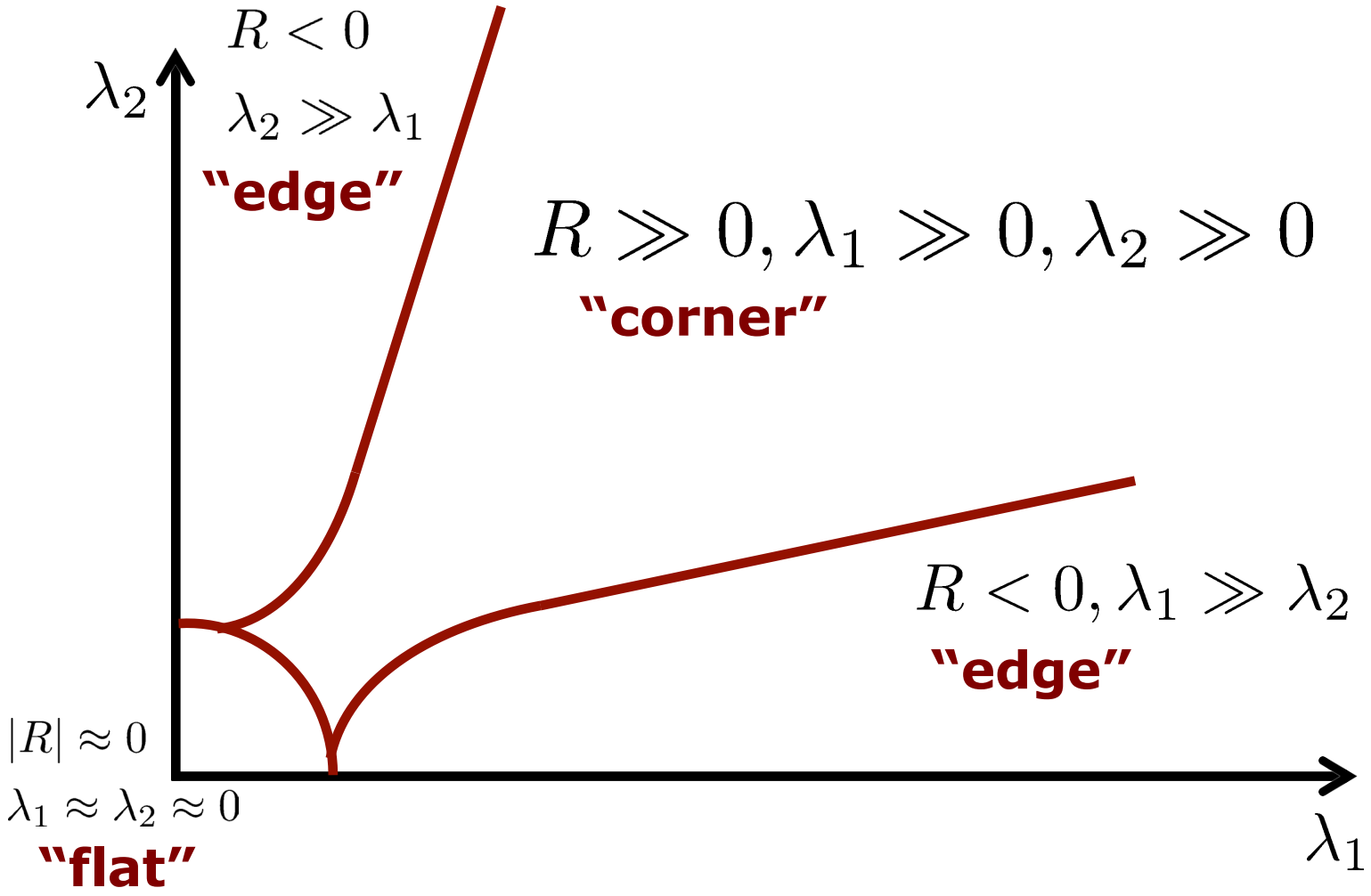
$|R| \approx 0 \Rightarrow \lambda_1 \approx \lambda_2 \approx 0$ **: flat region**

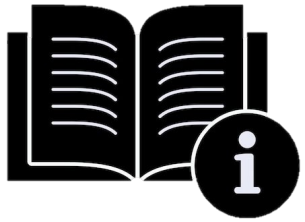
$R < 0 \Rightarrow \lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$ **: edge**

$R \gg 0 \Rightarrow \lambda_1 \approx \lambda_2 \gg 0$ **: corner**



Harris Criterion Illustrated



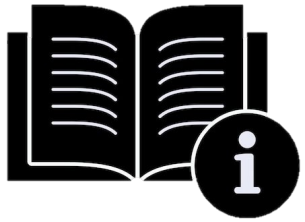


Shi-Tomasi Corner Detector

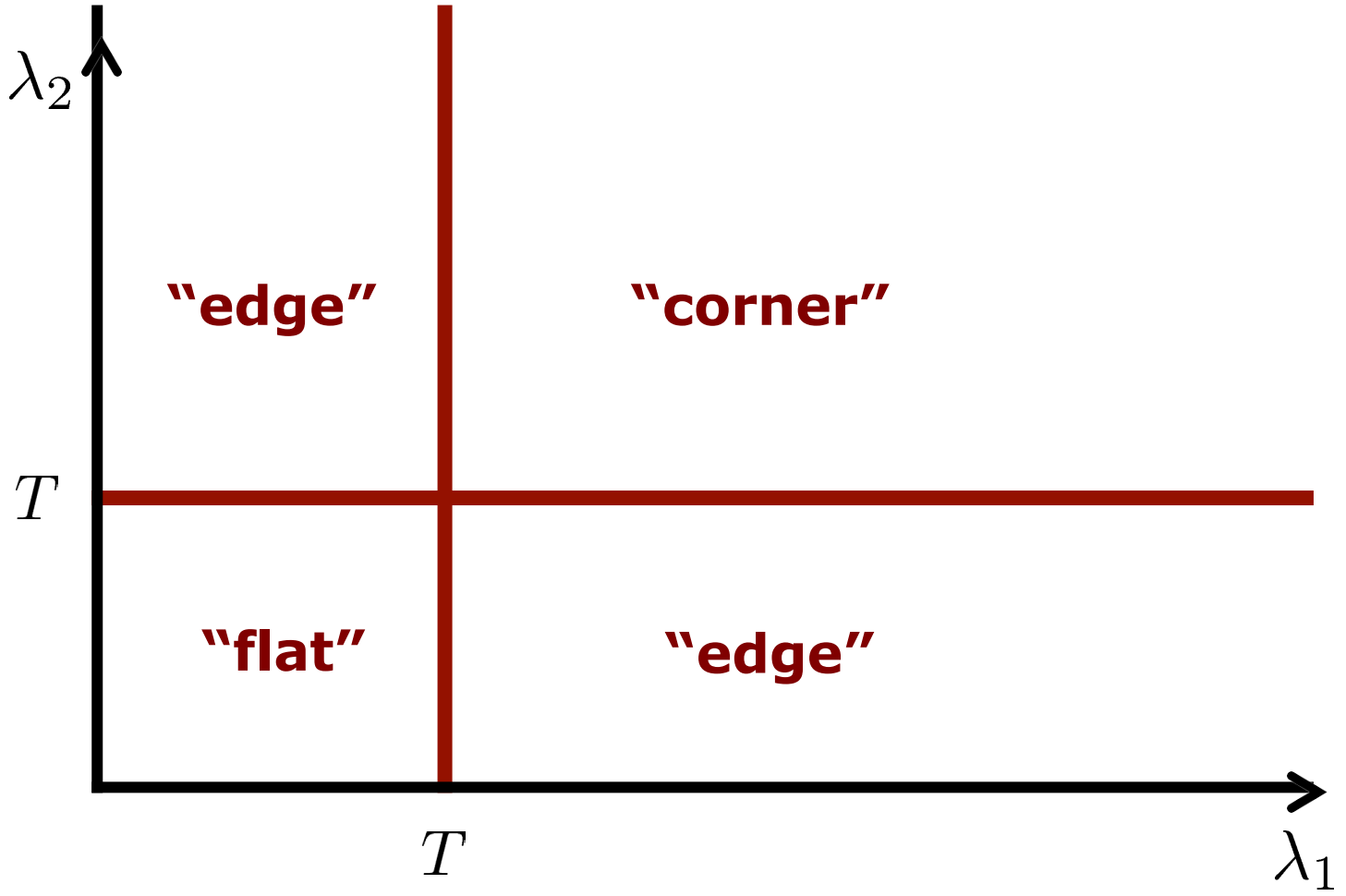
- Criterion: Threshold smallest Eigenvalue

$$\lambda_{\min}(M) = \frac{\text{trace}(M)}{2} - \frac{1}{2}\sqrt{(\text{trace}(M))^2 - 4\det(M)}$$

$$\lambda_{\min}(M) \geq T \quad \textbf{: corner}$$



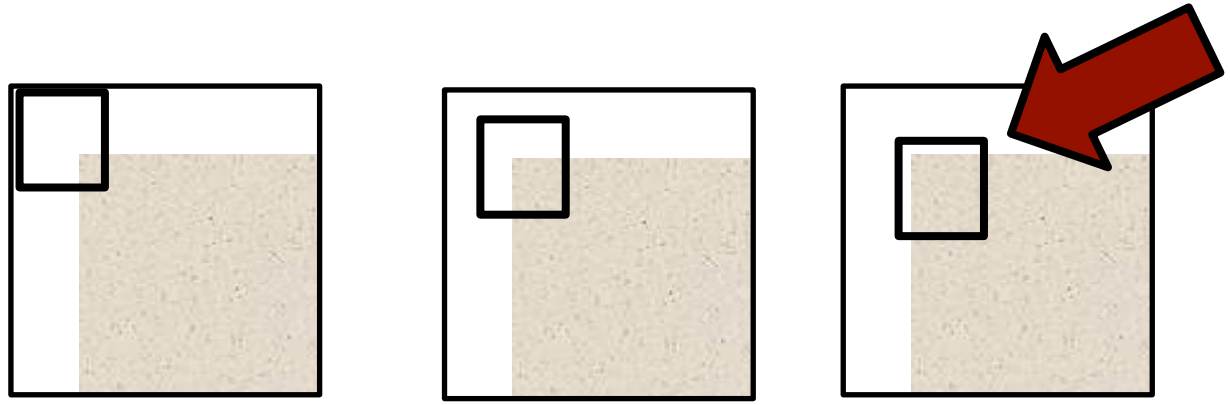
Shi-Tomasi Criterion Illustrated



Non-Maxima Supression

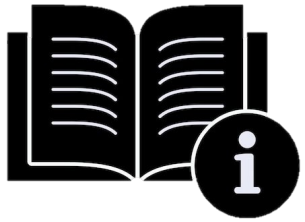
- Within a local region, looks for the position with the maximum value (R or $\lambda_{\min}$) and select this point

- Example for the Förstner operator

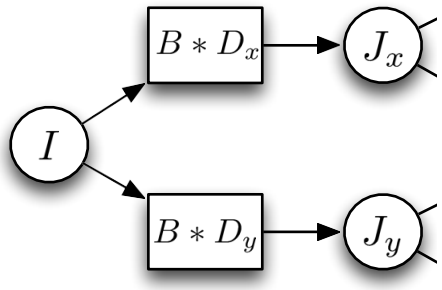


Implementation Remarks

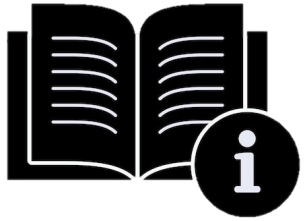
- RGB to gray-scale conversion first
- Real images are affected by noise, smoothing of the input is suggested



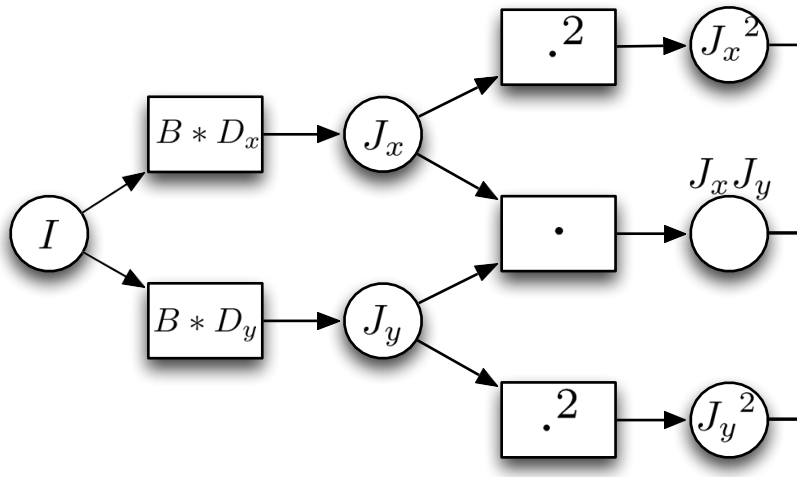
Summary Corner Detection



convolutions
(smoothing
& derivatives)

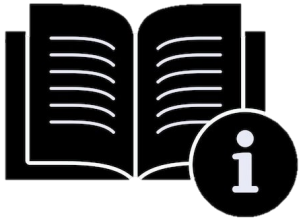


Summary Corner Detection

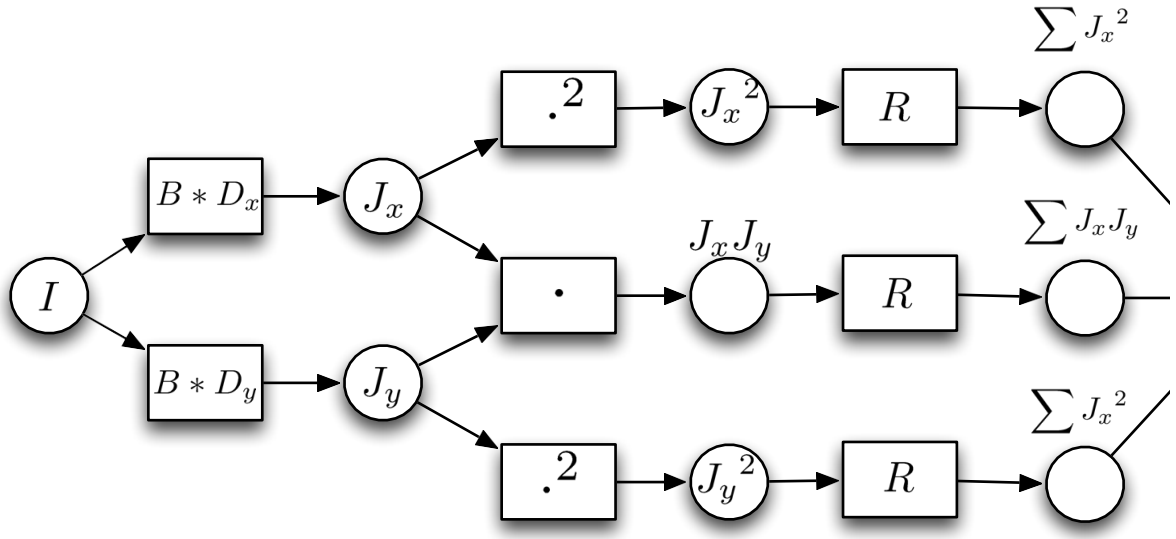


convolutions (smoothing & derivatives)

multiplications



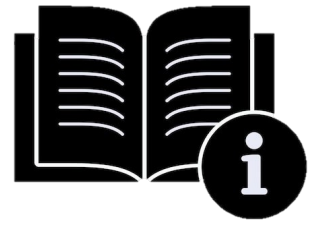
Summary Corner Detection



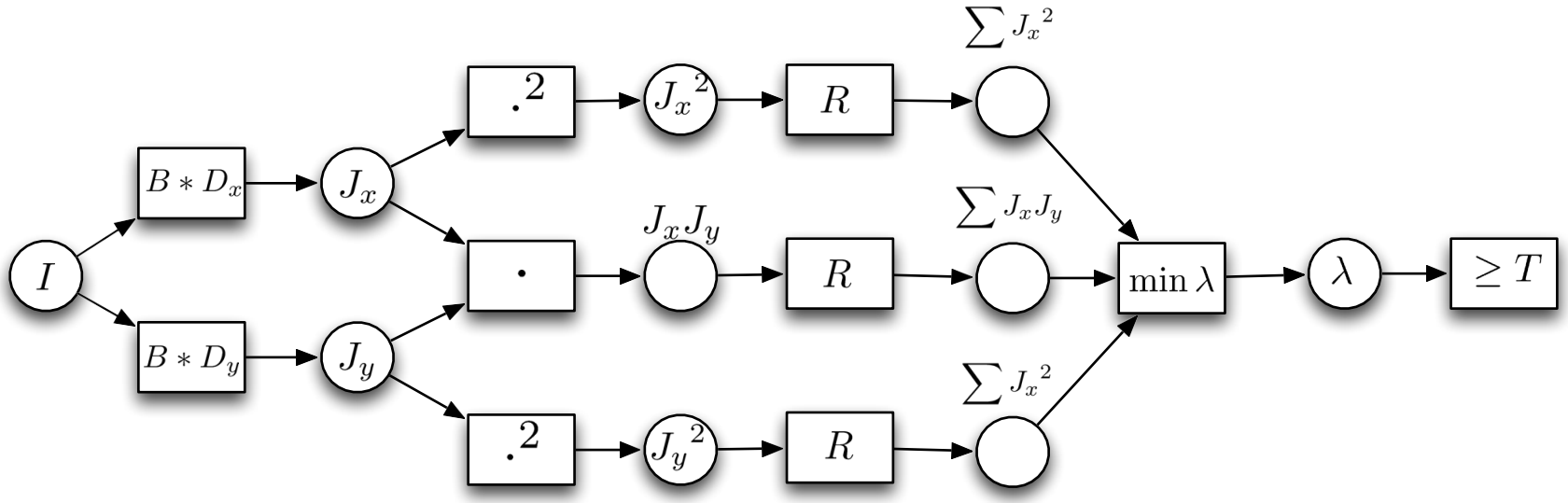
convolutions (smoothing & derivatives)

multiplications

convolutions (box-summing)



Summary Corner Detection



convolutions (smoothing & derivatives)

multiplications

convolutions (box-summing)

multiplications, sums, sqrt

thresholding non-max suppression

Harris Corners Example

Corner Detectors Comparison

- Harris became the most famous corner detector in the past
- Shi-Tomasi seems to slightly outperform Harris corners
- Most libraries use Shi-Tomasi as the default corner detector (e.g., openCV)

Difference of Gaussians Keypoints

- A variant of corner detection
- Provides responses at corners, edges, and blobs
- Blob = mainly constant region but different to its surroundings

Keypoints: Difference of Gaussians Over Scale-Space Pyramid

Procedure

Over different image pyramid levels

- Step 1: Gaussian smoothing
- Step 2: Difference-of-Gaussians: find extrema (over smoothing scales)
- Step 3: maxima suppression at edges

Illustration

1/16

1	2	1
2	4	2
1	2	1

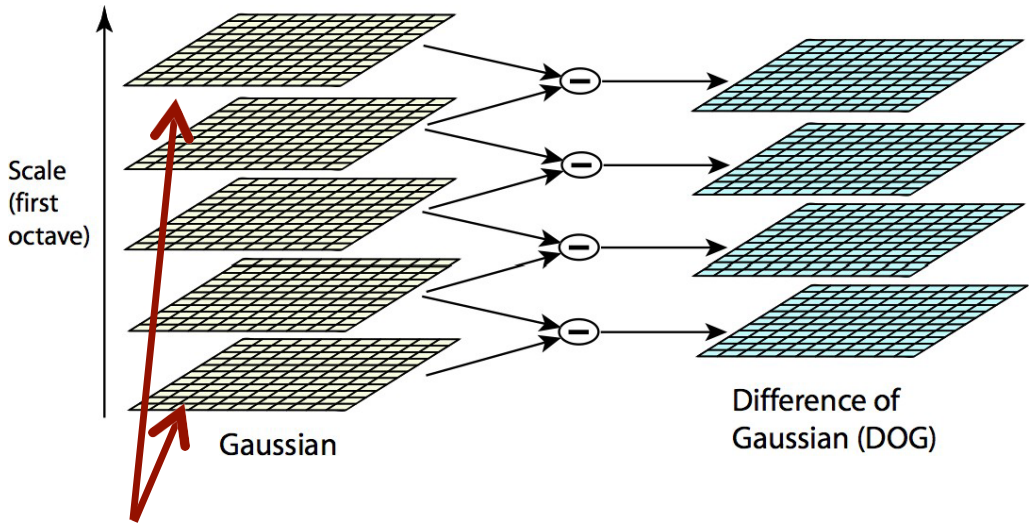
1/273

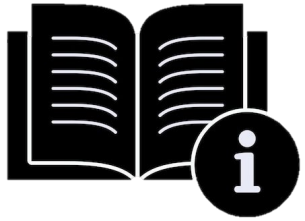
1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

1/1003

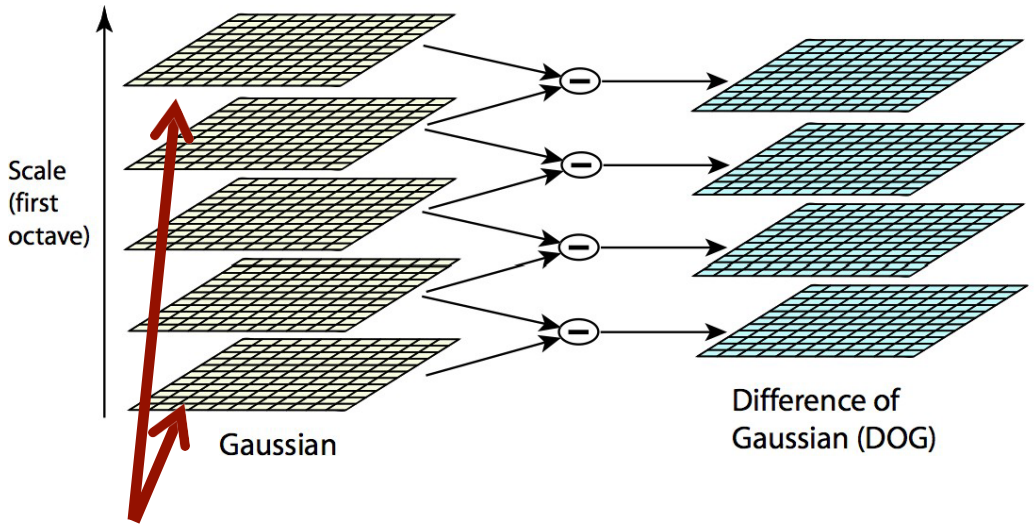
0	0	1	2	1	0	0
0	3	13	22	13	3	0
1	13	59	97	59	13	1
2	22	97	159	97	22	2
1	13	59	97	59	13	1
0	3	13	22	13	3	0
0	0	1	2	1	0	0

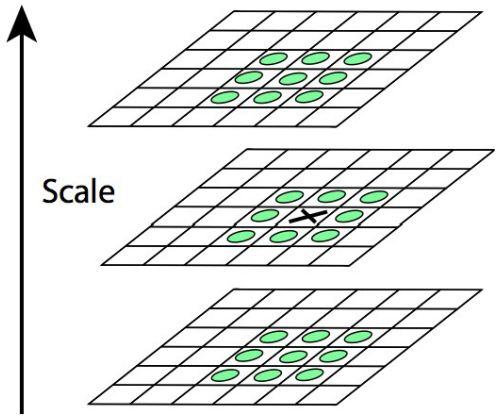
Discrete approximation of the Gaussian kernels 3x3, 5x5, 7x7

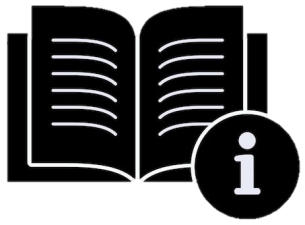




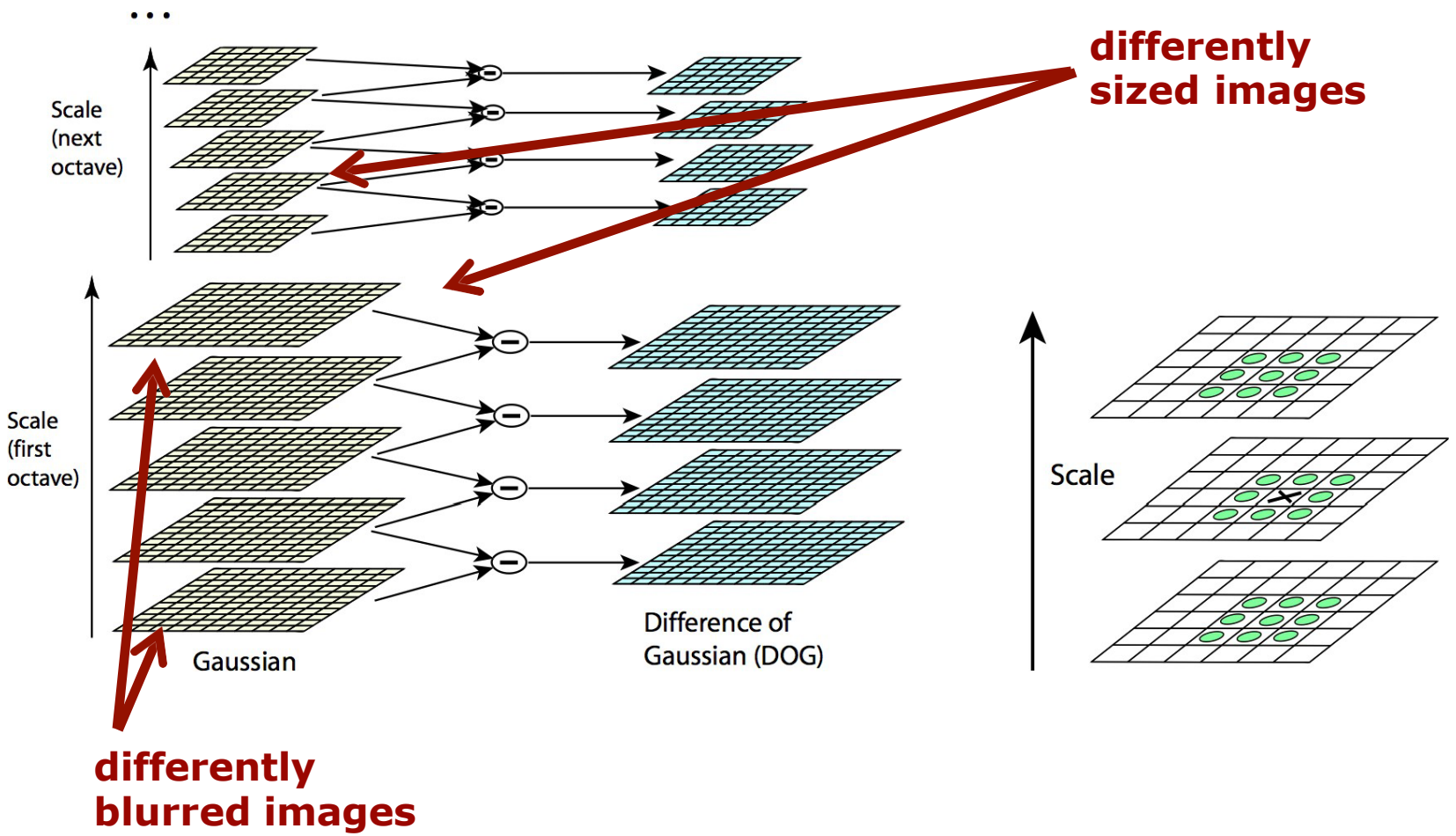
Illustration







Illustration



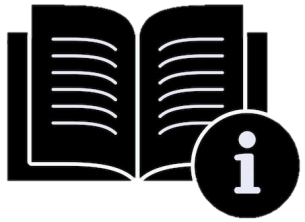
Difference of Gaussians

- Subtract differently blurred images from each other

- Increases visibility of corners, edges, and other detail present in the image

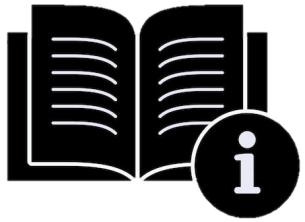
Difference of Gaussians

Keypoints are extrema in the DoG over different (smoothing) scales



Difference of Gaussians

- Blurring filters out high-frequencies (noise)
- Subtracting differently blurred images from each other only keeps the frequencies that lie between the blur level of both images
- DoG acts as a band-pass filter



Extrema Suppression

- The DoG finds blob-like and corner-like image structures but also leads to strong responses along edges
- Edges are bad for matching
- Eliminate edges via Eigenvalue test (similar to Harris corners)

Keypoints

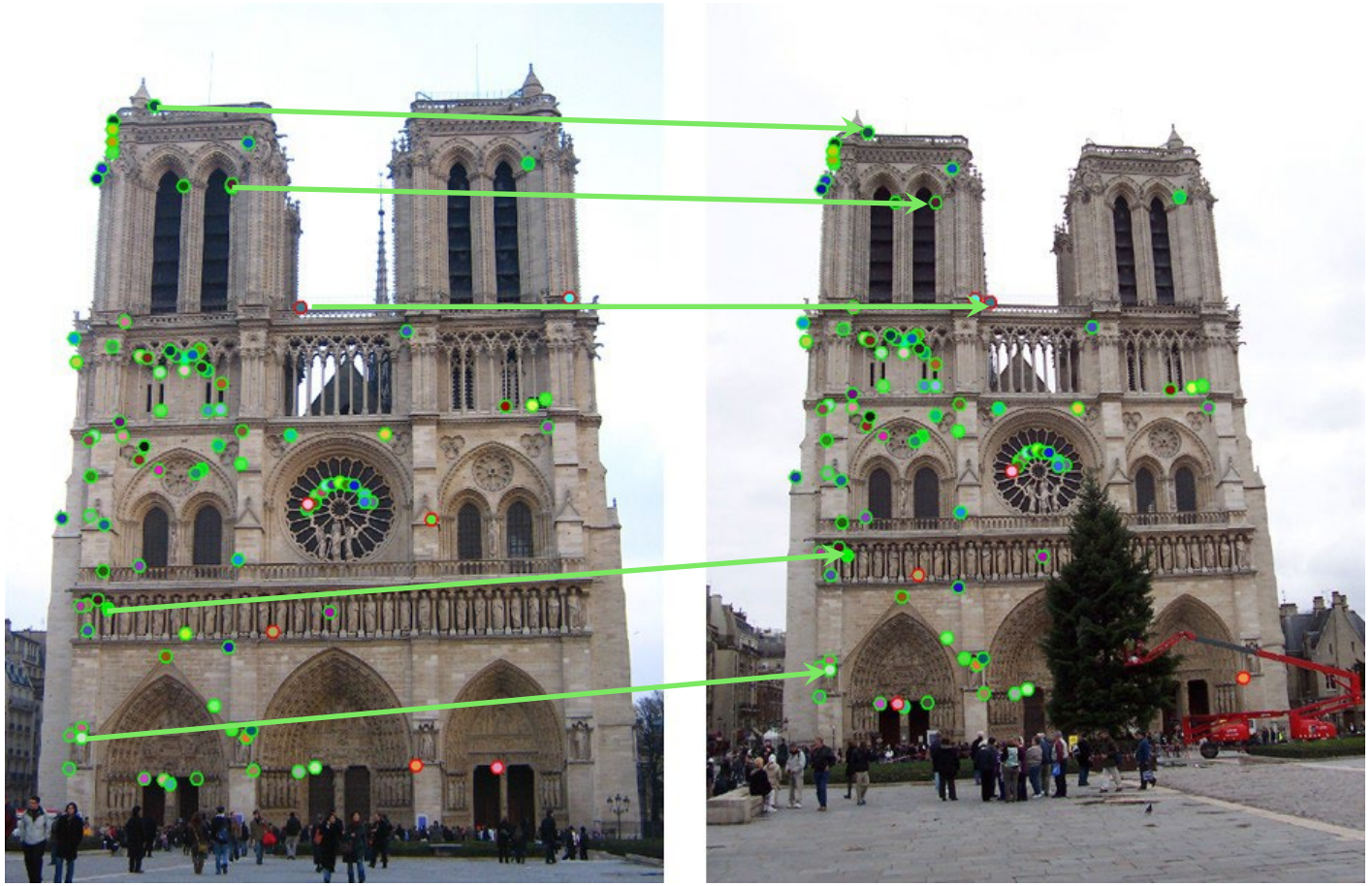
- Two groups of approaches for finding locally distinct points:
- 1. Corners via structure matrix
  - Harris, Shi-Tomasi
- 2. Difference of Gaussians
  - Iterates over scales and blur
  - Finds corners and blobs
- These approaches are key ingredients of most hand-designed features

Summary

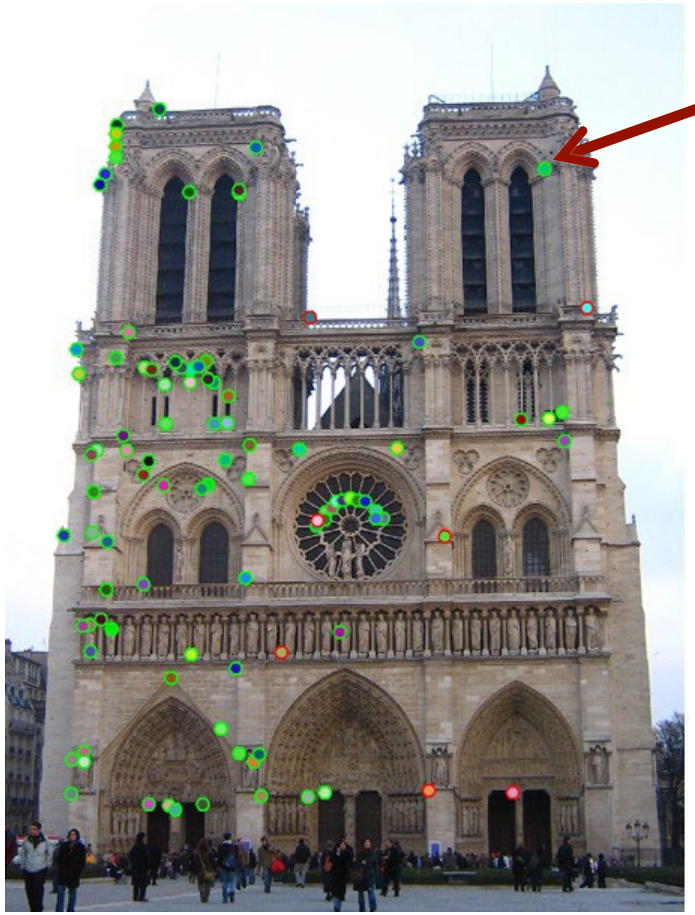
- Keypoints and descriptor together define common visual features
- Keypoint defines the location
- Most keypoints use image gradients
- Corners and blobs are good keypoints

Feature Descriptors

Can We Describe Keypoints to Enable Matching Across Images?



Is is All About the Vector f...



keypoint

descriptor at the keypoint

$$f = \begin{bmatrix} 0.02 \\ 0.04 \\ 0.1 \\ 0.03 \\ 0 \\ \dots \end{bmatrix}$$

- **Features: Describing a keypoint**
  - **SIFT** – Scale Invariant Feature Transform
  - **BRIEF** – Binary Robust Independent Elementary Features
  - **ORB** – Oriented FAST Rotated BRIEF

Popular Features Descriptors

- HOG: Histogram of Oriented Gradients
- SIFT: Scale Invariant Feature Transform
- SURF: Speeded-Up Robust Features
- GLOH: Gradient Location and Orientation Histogram
- BRIEF: Binary Robust Independent Elementary Features
- ORB: Oriented FAST and rotated BRIEF
- BRISK: Binary Robust Invariant Scalable Keypoints
- FREAK: Fast REtinA Keypoint
- ...

Popular Features Descriptors

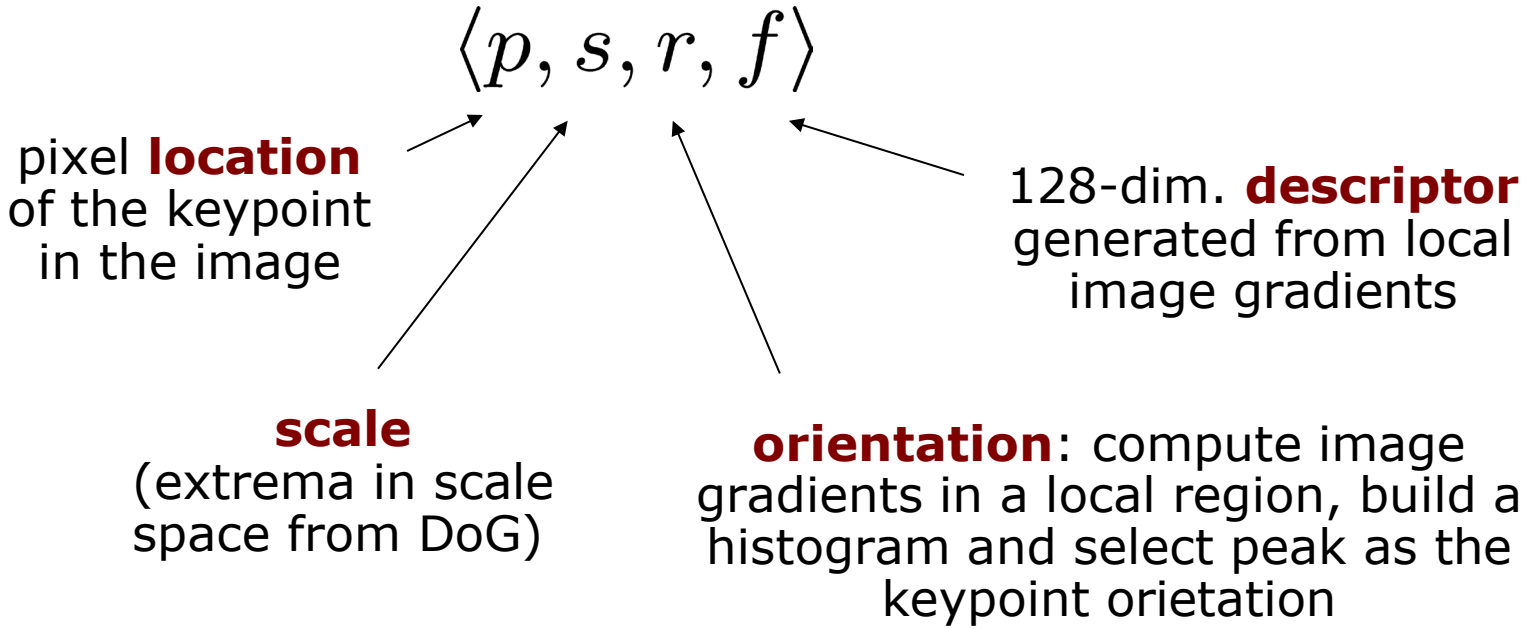
- HOG: Histogram of Oriented Gradients
- **SIFT: Scale Invariant Feature Transform**
- SURF: Speeded-Up Robust Features
- GLOH: Gradient Location and Orientation Histogram
- **BRIEF: Binary Robust Independent Elementary Features**
- **ORB: Oriented FAST and rotated BRIEF**
- BRISK: Binary Robust Invariant Scalable Keypoints
- FREAK: Fast REtinA Keypoint
- …

SIFT Descriptor

- Image content is transformed into features that are **invariant to**
  - image translation,
  - rotation, and
  - scale
- They are **partially invariant to**
  - illumination changes and
  - affine transformations and 3D projections
- Suitable for detecting visual landmarks
  - **from different angles and distances**
  - **with a different illumination**

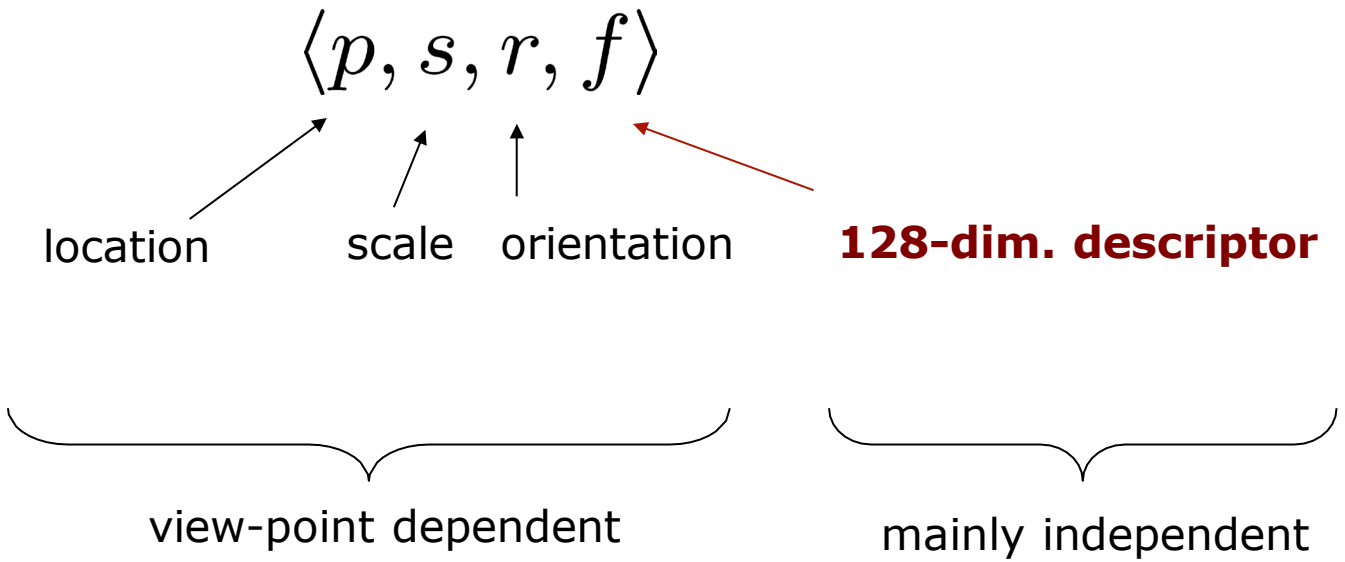
SIFT Features

A SIFT feature is given by a vector computed at a local extreme point in the scale space



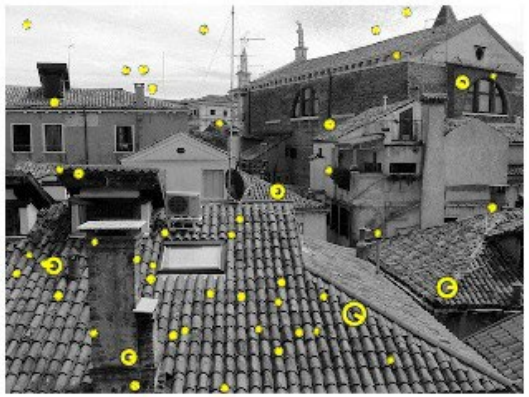
SIFT Features

A SIFT feature is given by a vector computed at a local extreme point in the scale space

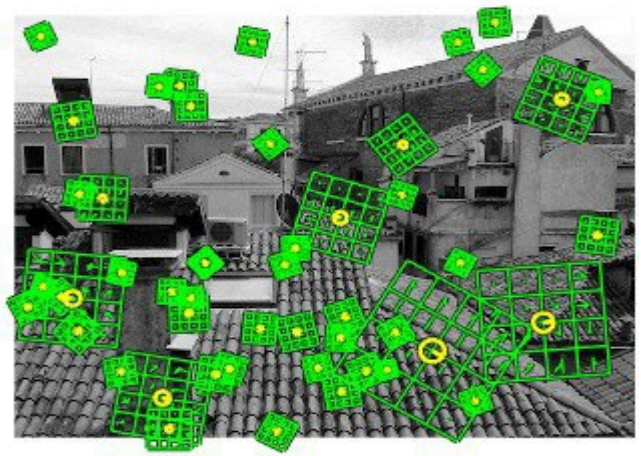


SIFT Considers the Distribution of Gradients Around Keypoints

input

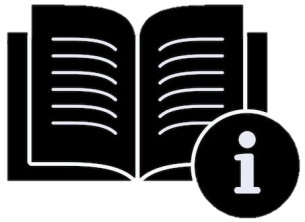


key points



descriptor (regions)

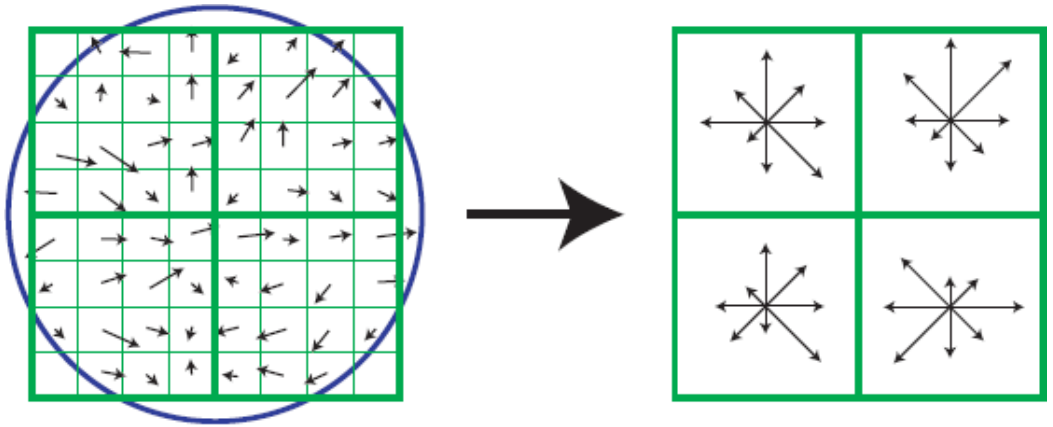
Image courtesy: Vedaldi and Fulkerson



SIFT Descriptor in Sum

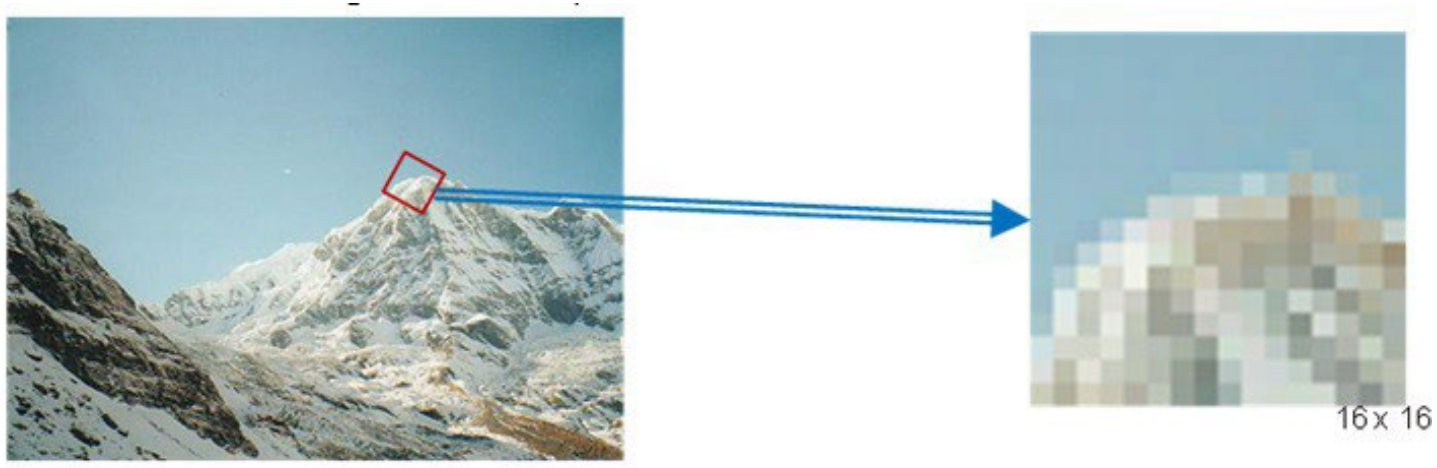
- Compute image gradients in local 16x16 area at the selected scale
- Create an array of orientation histograms
- 8 orientations x 4x4 histogram array = 128 dimensions (yields best results)

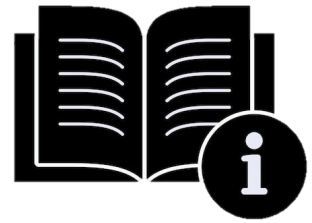
Example using a 8x8 area:



SIFT Illustration (1)

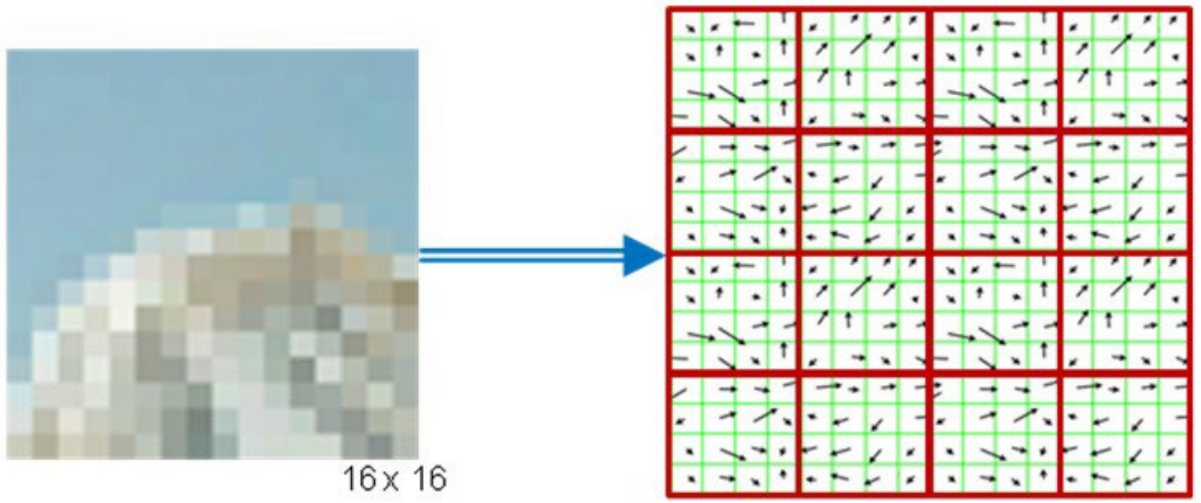
For a given keypoint, warp the region around it to orientation and scale and resize the region to 16X16 pixels.

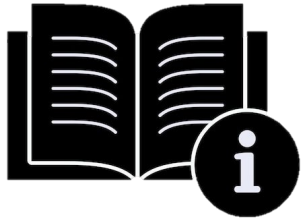




SIFT Illustration (2)

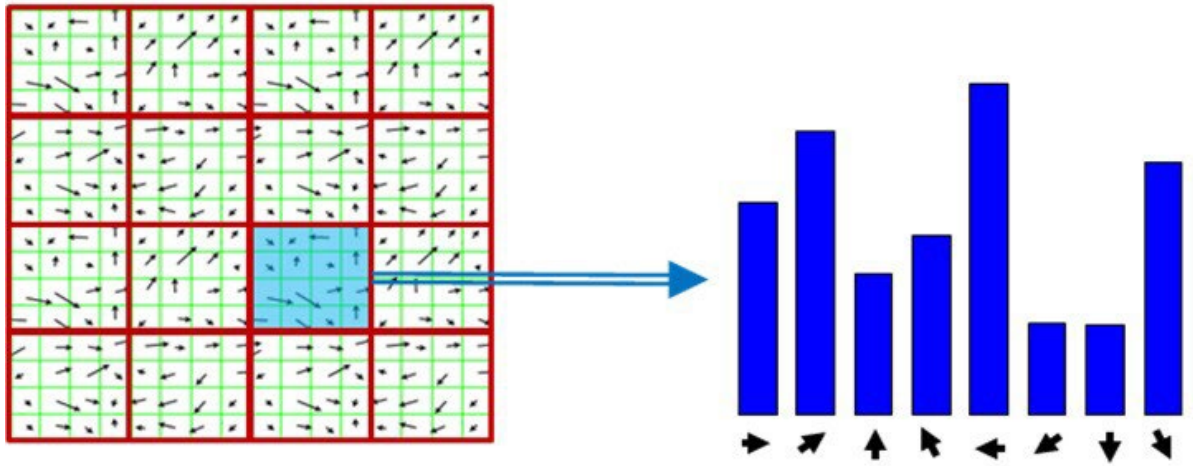
Compute the gradients for each pixels (orientation and magnitude) and divide the pixels into 16, 4X4 pixels squares

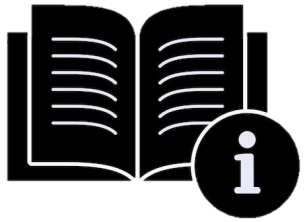




SIFT Illustration (3)

For each square, compute gradient direction histogram over 8 directions

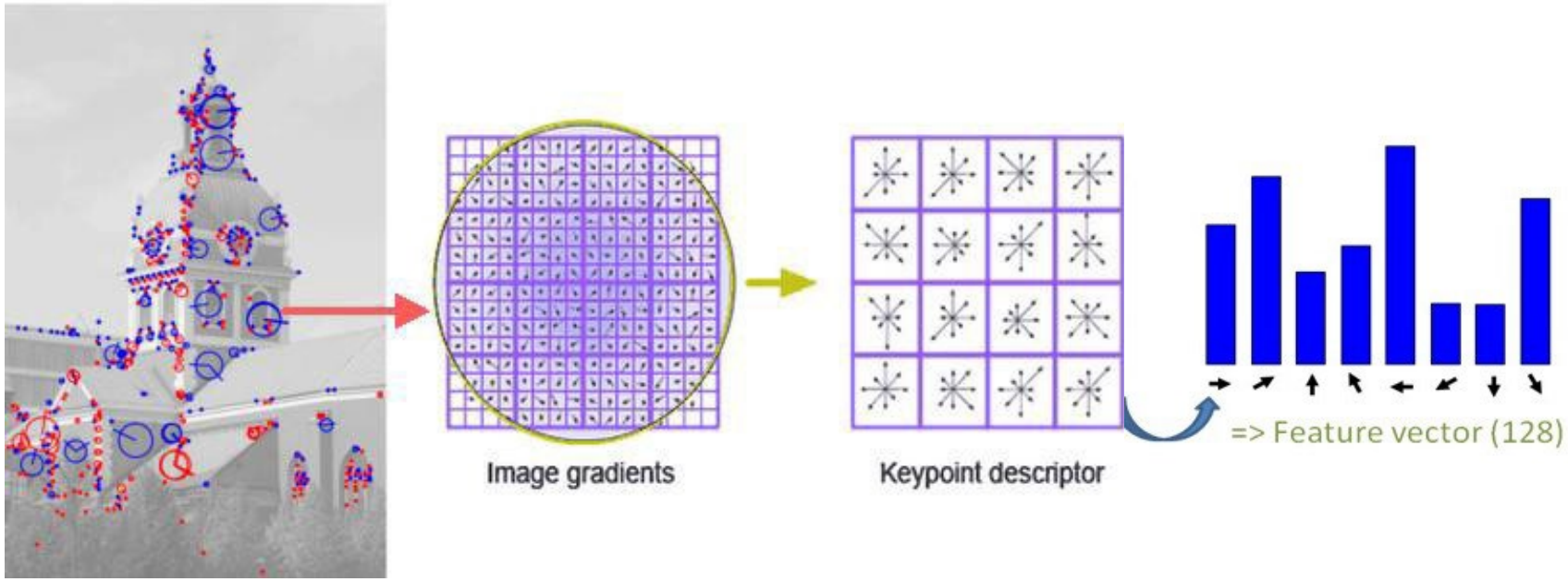




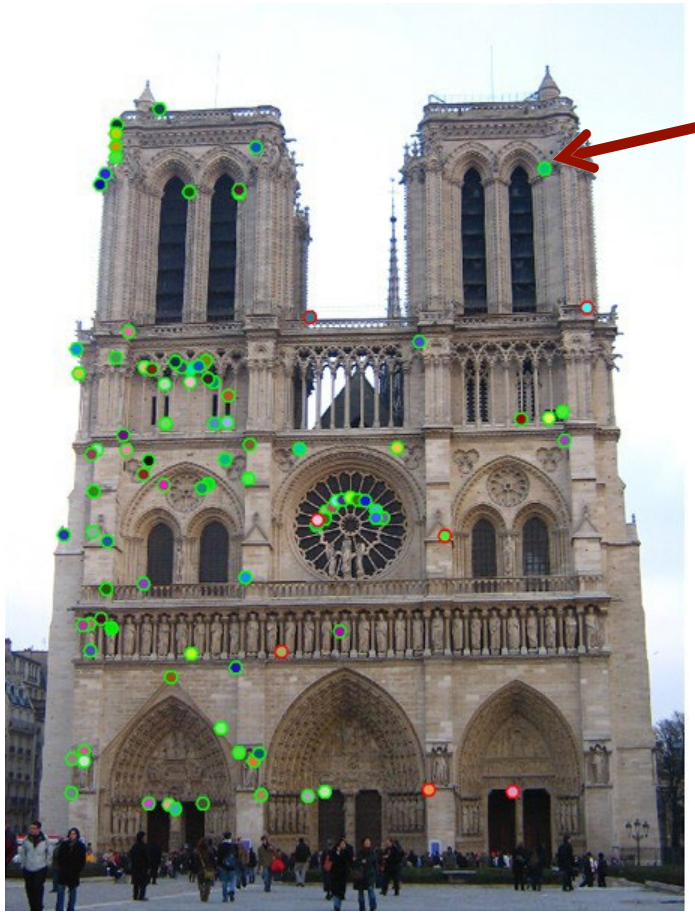
SIFT Illustration (4)

Concatenate the histograms to obtain a 128 (16*8) dimensional feature vector:

SIFT Descriptor Illustration



SIFT Approach Done!

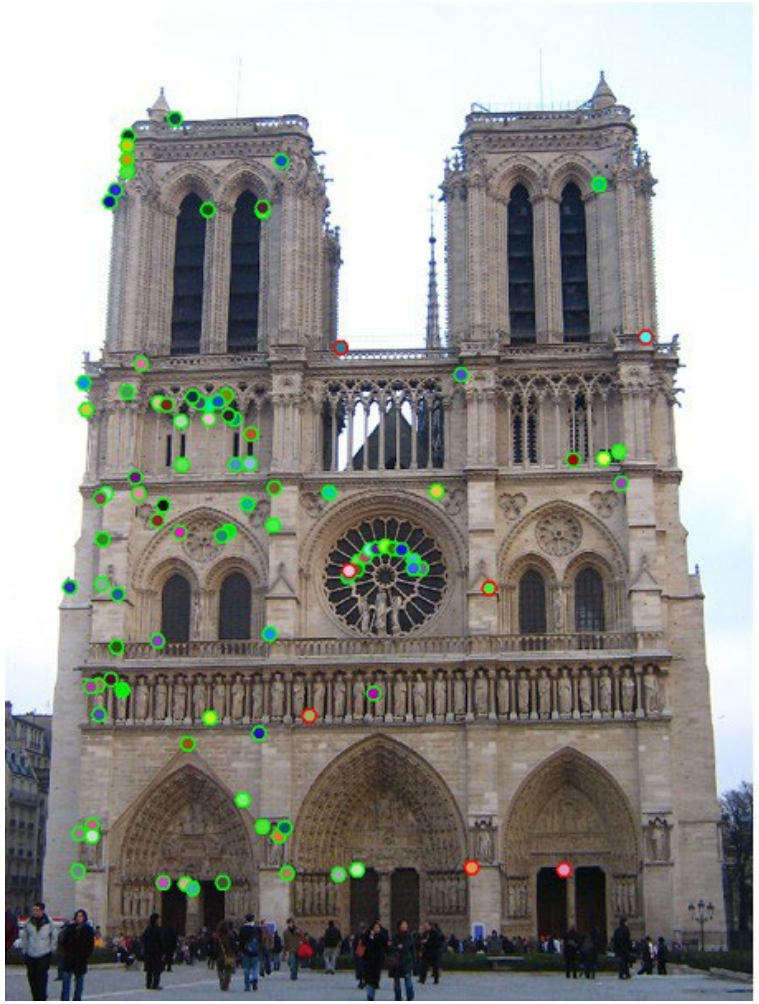


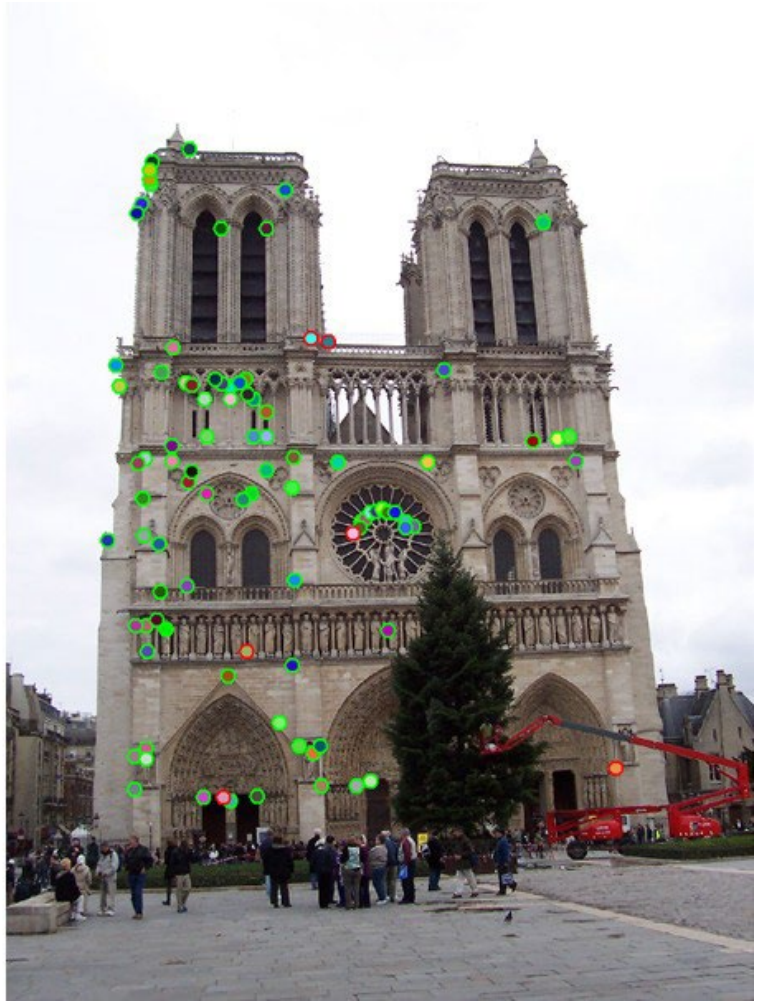
keypoint (via DoG)

descriptor (via gradient histogram)

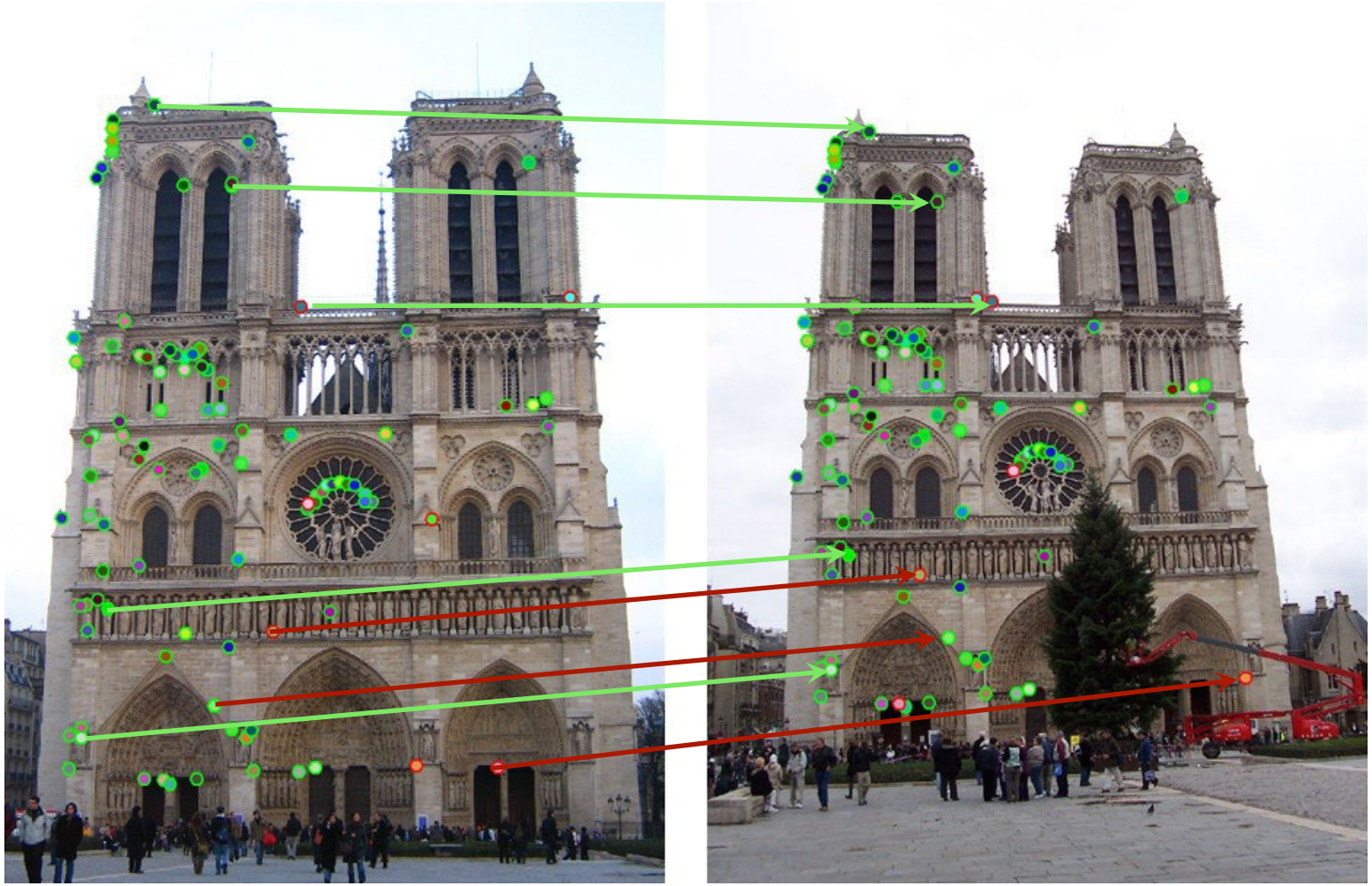
$$f = \begin{bmatrix} 0.02 \\ 0.04 \\ 0.1 \\ 0.03 \\ 0 \\ \dots \end{bmatrix}$$

How To Match Them?





Based on Descriptor Difference

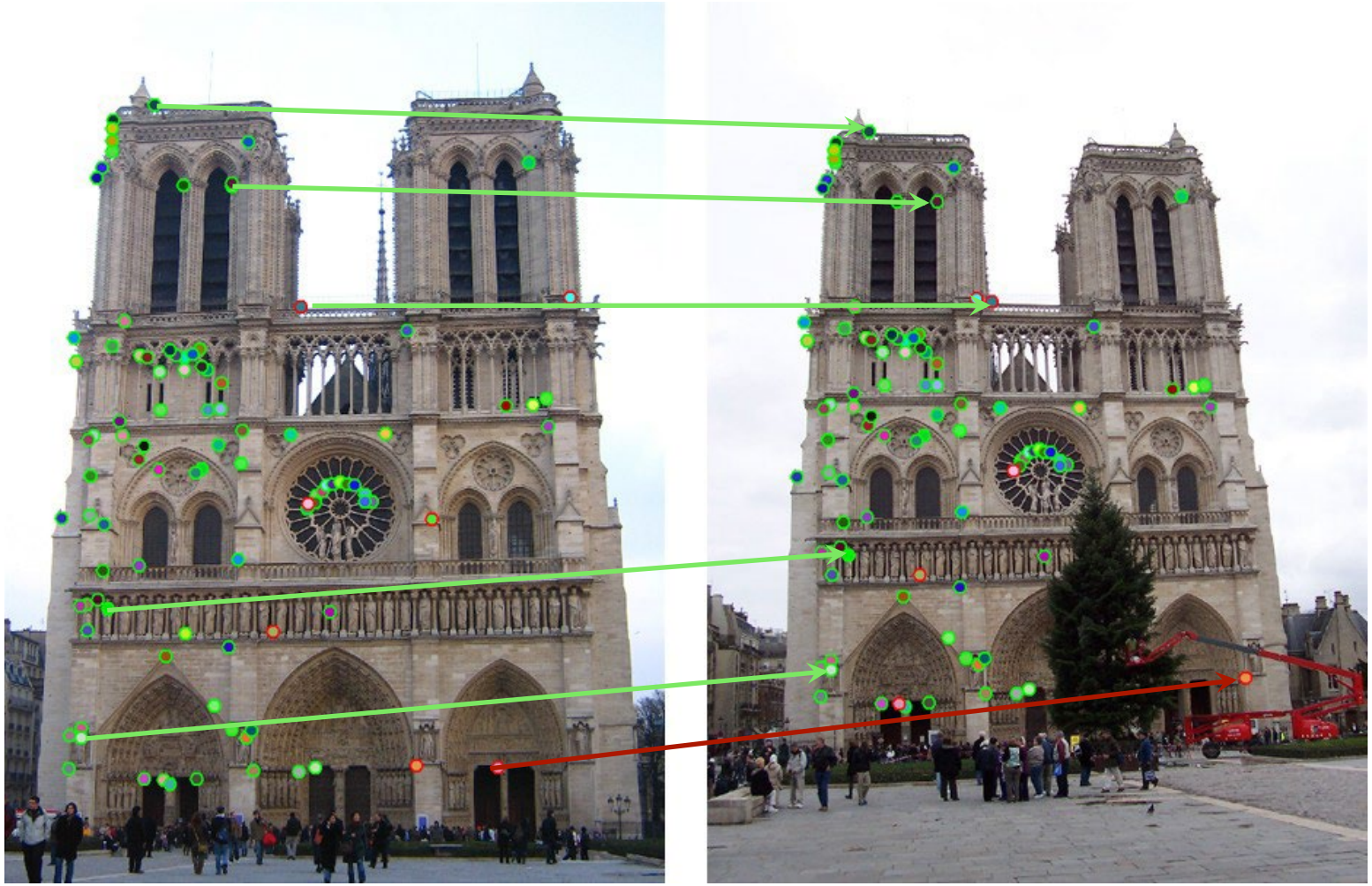


Lowe's Ratio Test

- 3 Step test to eliminate ambiguous matches for a query feature q
- 1. Step: Find closest two descriptors to q, called p_1 and p_2 based on the Euclidian distance d
- 2. Step: Test if distance to best match is smaller than a threshold: $d(q, p_1) < T$
- 3. Step: Accept match only if the best match is substantially better than second:

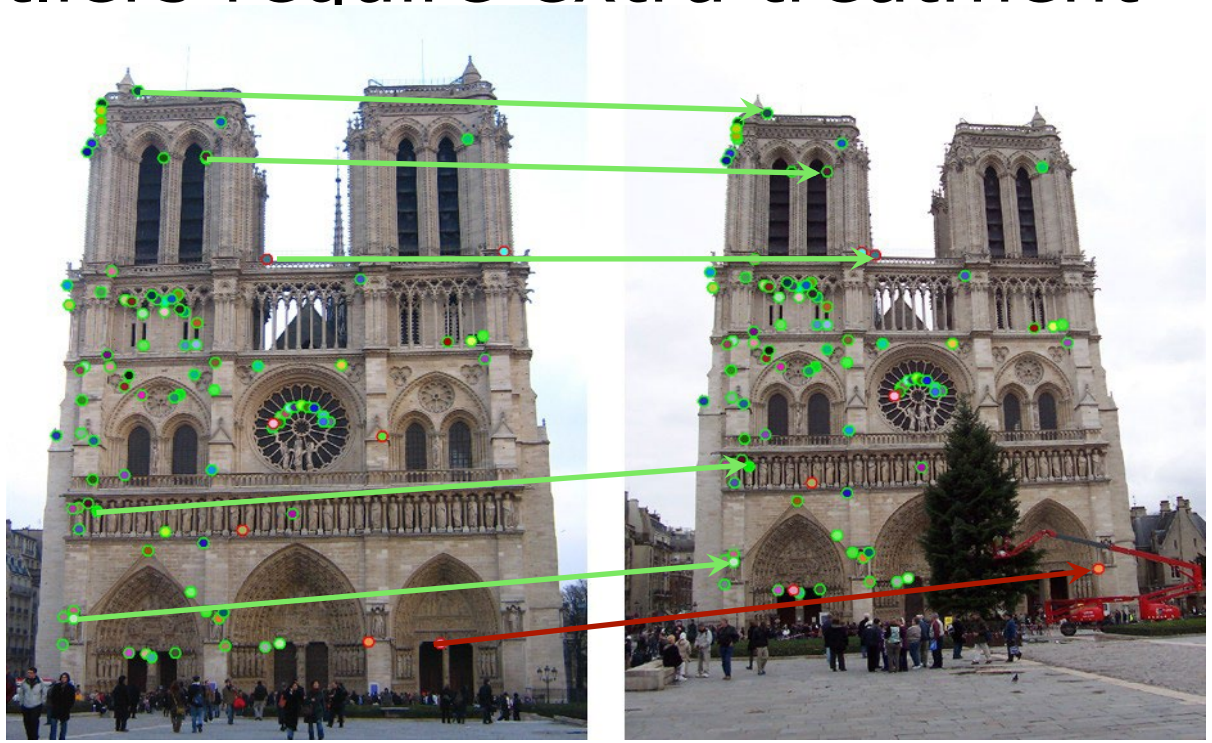
$$\frac{d(q, p_1)}{d(q, p_2)} < \frac{1}{2}$$

Based on Ratio Test



Outliers

- Lowe's Ratio test works well
- There will still remain few outliers
- Outliers require extra treatment



Binary Descriptors
or
“computing descriptor fast”

Why Binary Descriptors?

- Complex features such as SIFT work well and a a gold standard
- SIFT is expansive to compute
- Binary descriptors aim at generating **small binary strings** that are **easy to compute and compare**

Key Idea of Binary Descriptors

Fairly simple strategy

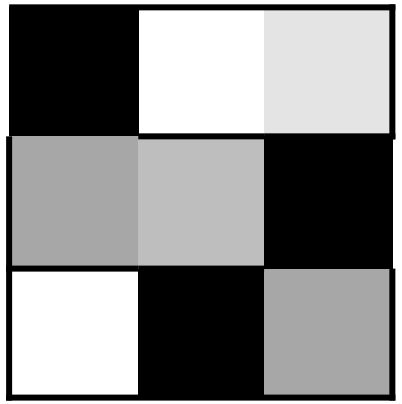
- Select a patch around a keypoint
- Select a set of pixel pairs in that patch
- For each pair, compare the intensities

$$b = \left\{ \begin{array}{ll} 1 & \text{if } I(s_1) < I(s_2) \\ 0 & \text{otherwise} \end{array} \right\}$$

- Concatenate all b's to a bit string

Example

image area



index (for pairs)

1	2	3
4	5	6
7	8	9

pairs: $\{(5,1), (5,9), (4,6), (8,2), (3,7)\}$

tests: $b=0 \quad b=0 \quad b=0 \quad b=1 \quad b=1$

result: $B = 00011$

Key Advantages of Binary Descriptors

- **Compact descriptor**
  The number of pairs gives the length in bits

- **Fast to compute**
  Simply intensity value comparisons

- **Trivial and fast to compare**
  Hamming distance

$$d_{\text{Hamming}}(B_1, B_2) = \text{sum}(\text{xor}(B_1, B_2))$$

Different Binary Descriptors Differ Mainly by the Strategy of Selecting the Pairs

Important Remark – Pairs

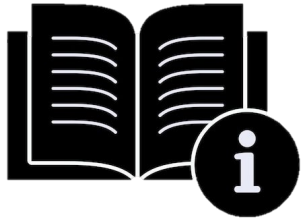
In order to compare descriptors among images, we must:

- Use the same pairs
- Maintain the same order in which the pairs are tested

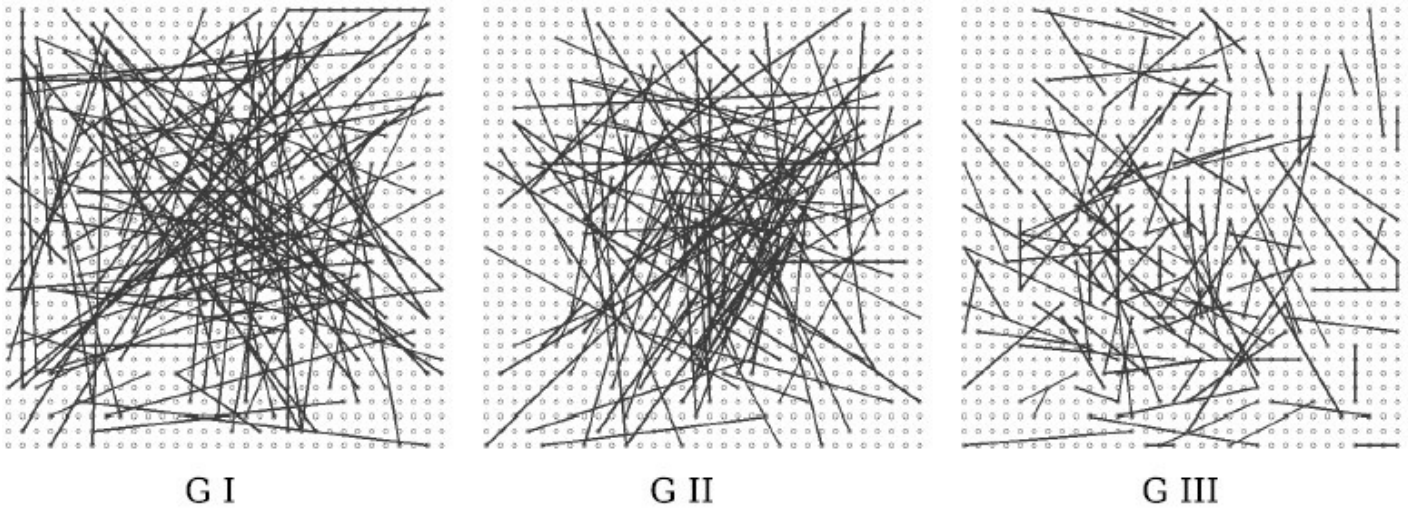
Different descriptors once determine the way the pairs are chosen and fix it!

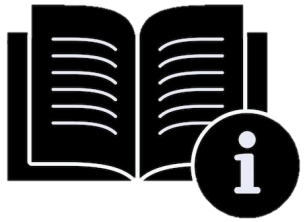
BRIEF : Binary robust independent elementary features

- First binary image descriptor
- Proposed in 2010
- 256 bit descriptor
- Provides five different geometries as sampling strategies
- Noise: operations performed on a **smoothed** image to deal with noise



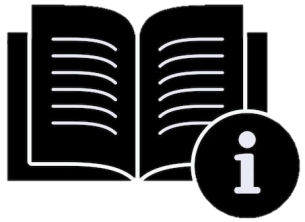
BRIEF Sampling Pairs



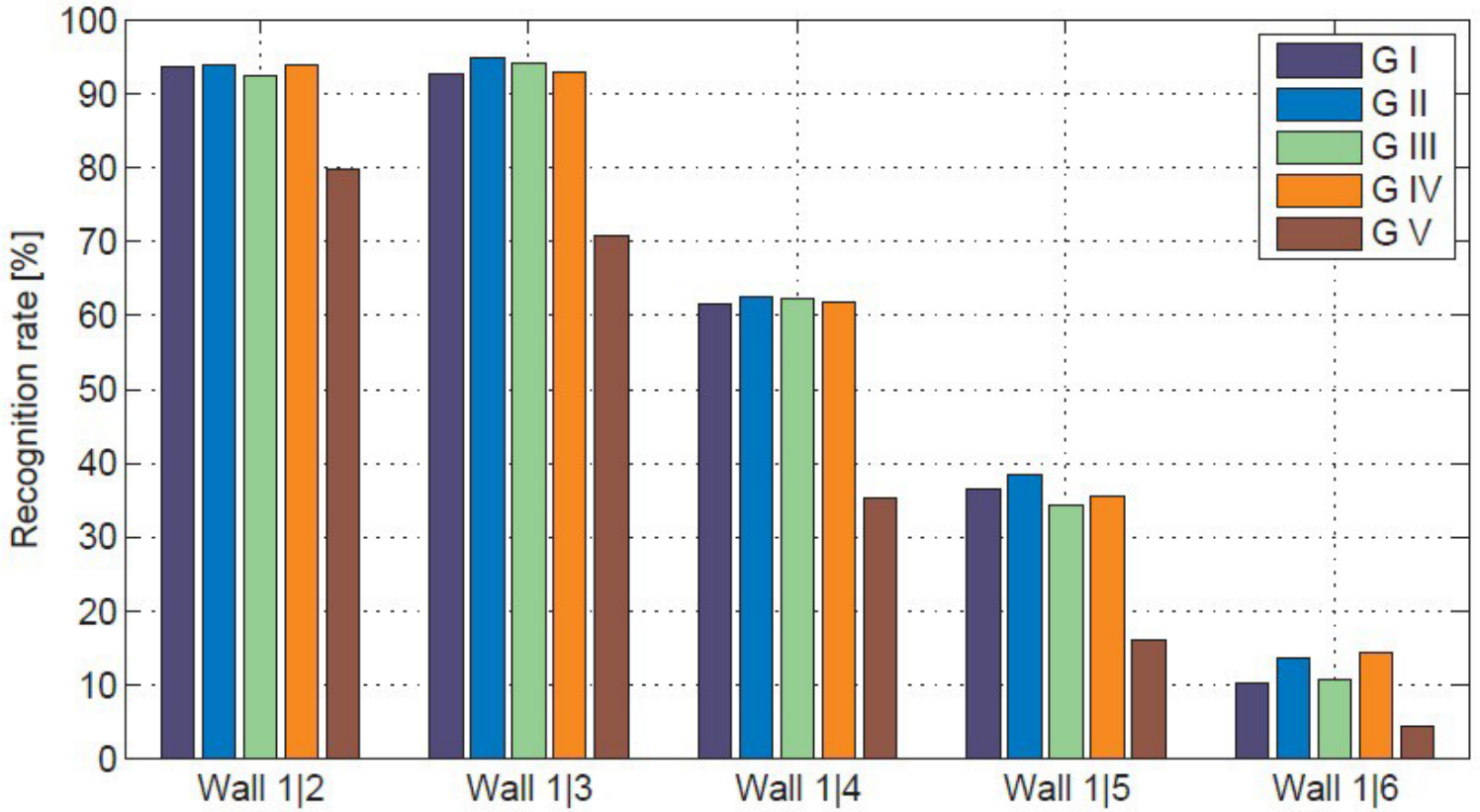


BRIEF Sampling Pairs

- G I: Uniform random sampling
- G II: Gaussian sampling
- G III: s_1 Gaussian; s_2 Gaussian centered around s_1
- G IV: Discrete location from a coarse polar gird
- G V: s_1=(0,0) ; s_2 are all location from a coarse polar gird



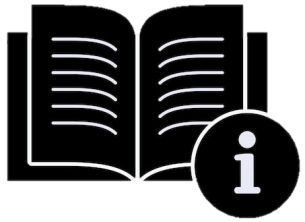
Performance: G I – G IV are all good, G V less useful



ORB:
Oriented FAST Rotated BRIEF

An extension to BRIEF that

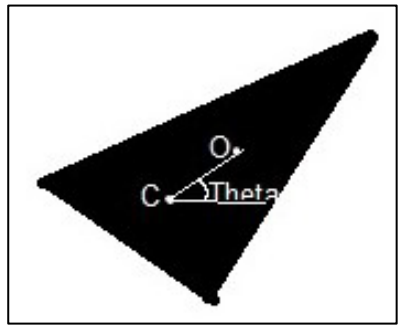
- Adds rotation compensation
- Learns the optimal sampling pairs



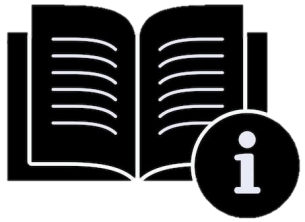
ORB: Rotation Compensation

- Estimates the center of mass and the main orientation of the area/patch
- Image moment

$$m_{pq} = \sum_{x,y} x^p \, y^q \, I(x,y)$$



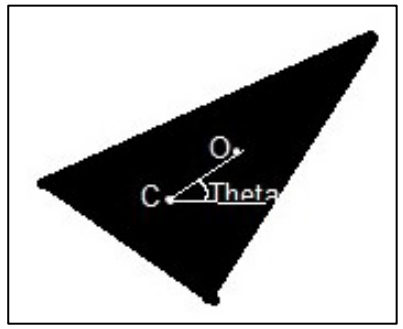
- Center of mass $C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$
- Orientation $\theta = \text{atan2}(m_{01}, m_{10})$



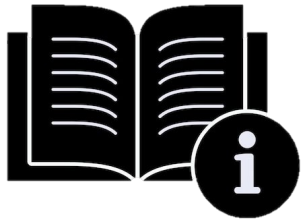
ORB: Rotation Compensation

- Given CoM and orientation C, θ, we can rotate the coordinates of all pairs by θ around C:

$$s' = T(C, \theta)\, s$$



- Use the transformed pixel coordinates for performing the test
- Invariance to rotation in the plane



ORB: Learning Sampling Pairs

Pairs should be / have

- **uncorrelated** – so that each new pair adds new information to the descriptor
- **high variance** – it makes a feature more discriminative

- ORB defines a strategy for selection 256 pairs optimizing for both properties using a training database

ORB vs. SIFT

- ORB is 100x faster than SIFT
- ORB: 256 bit vs. SIFT: 4096 bit
- ORB is not scale invariant (achievable via an image pyramid)
- ORB mainly in-plane rotation invariant
- ORB has a similar matching performance as SIFT (w/o scale)
- Several modern online systems (e.g. SLAM) use binary features

Summary

- Keypoints and descriptor together define common visual features
- Keypoint defines the location
- Descriptor describes the appearance
- Several descriptors operating on gradient histograms (SIFT, SURF, …)
- Binary descriptors for efficiency (BRIEF, ORB, ...)

References

- **Slides:** Photogrammetry I & II Course (https://www.ipb.uni-bonn.de/photo12-2021/index.html)
- **Slides:** Mines Computer Vision Course (by Thomas Williams)
- **Slides:** MIT's Advances in Computer Vision (http://6.869.csail.mit.edu/fa19/materials.html)
- **OpenCV camera calibration**: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html