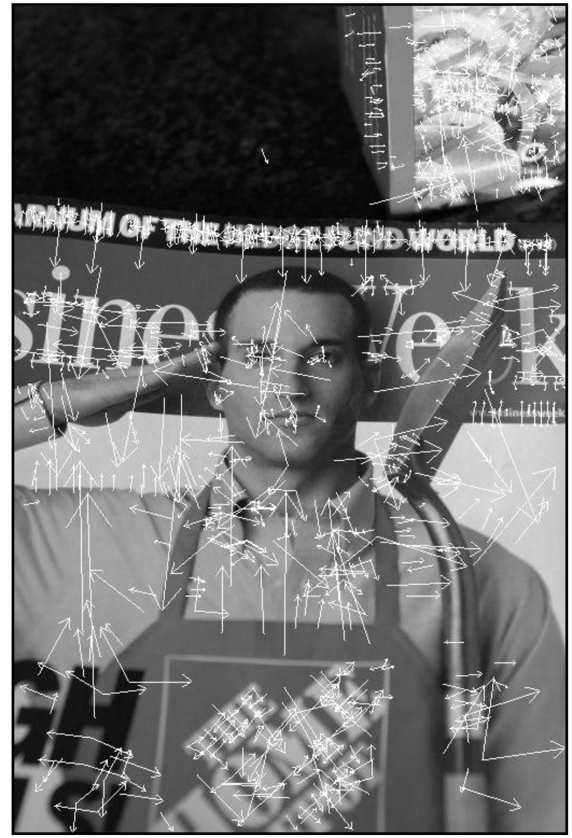
Topic 10:

Feature Detection & Image Matching

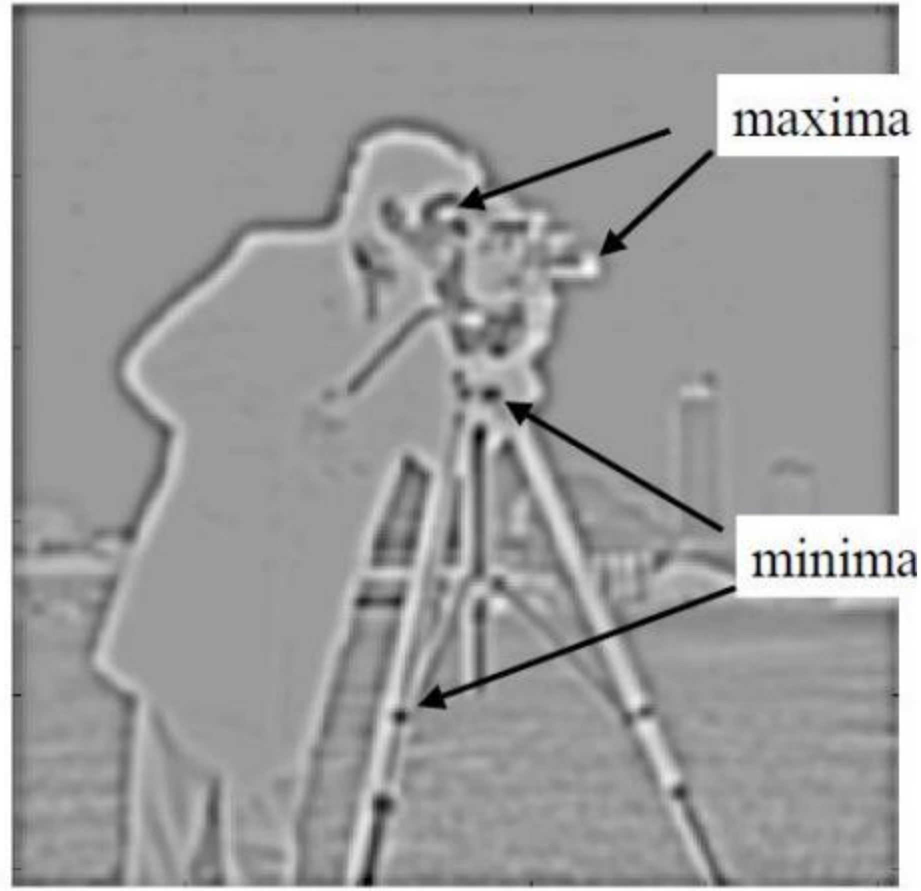
- Introduction to the image matching problem
- Image matching using SIFT features
- The SIFT feature detector
- The SIFT descriptor

Goal

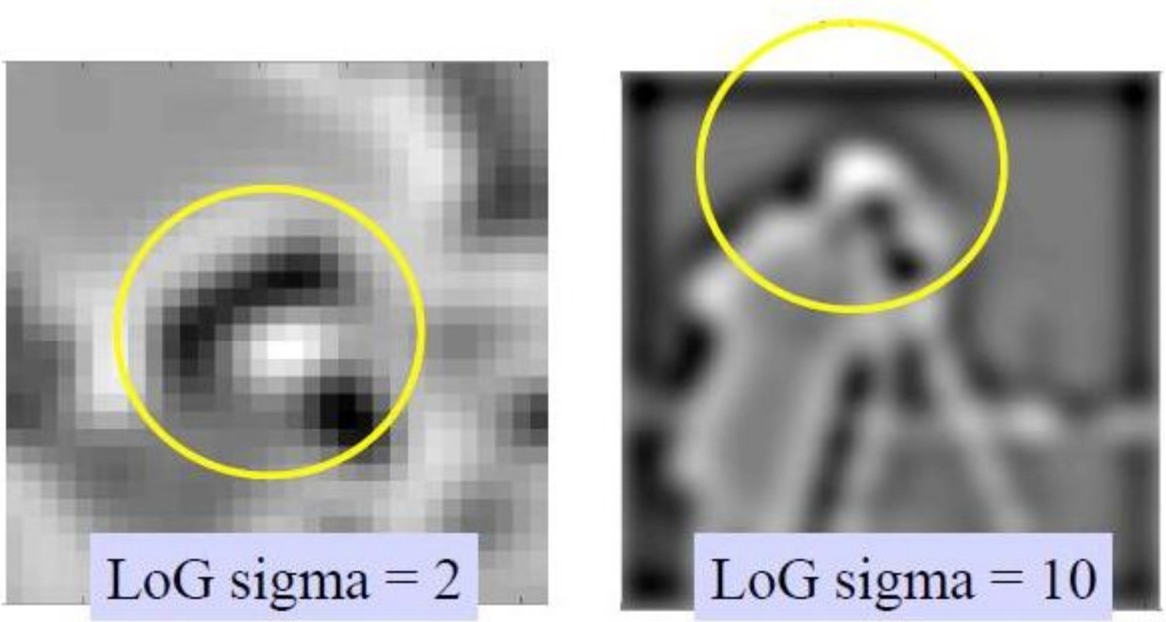


- To identify distinctive image locations (keypoints).
- Assign a scale and orientation associated to each keypoint.
- Make keypoints invariant to magnification, brightness, etc.

Keypoints: Contrasting Blobs at Different Scales

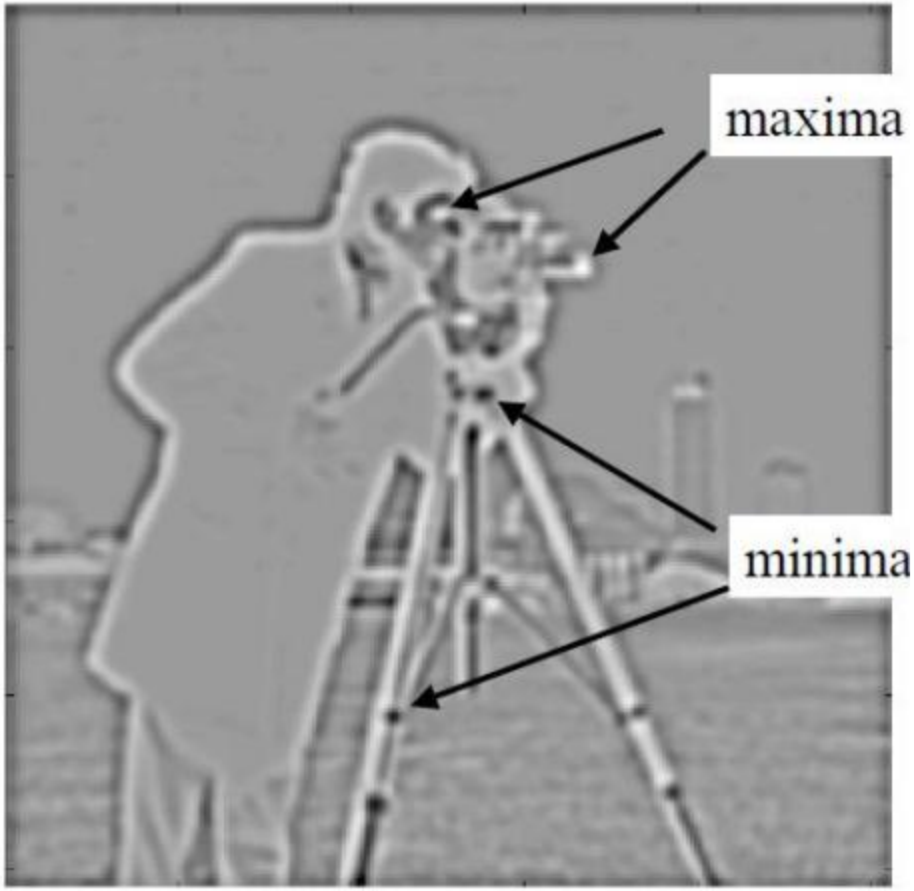


Keypoints: Contrasting Blobs at Different Scales



Why are these features good?

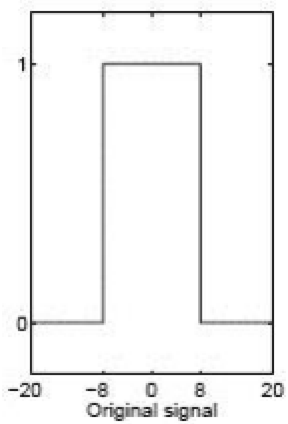
Keypoints: Contrasting Blobs at Different Scales



SIFT

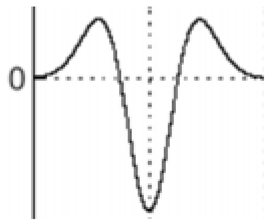
SIFT is an algorithm that finds contrasting blobs at different scales

Original signal



(radius=8)

Difference of Gaussians Template

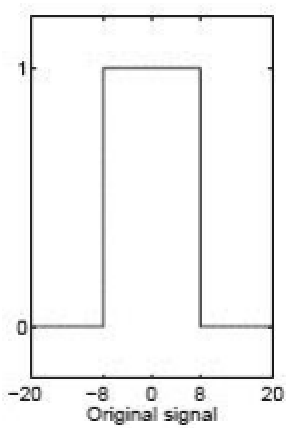


This will find a blob

SIFT: Scale Detection Example

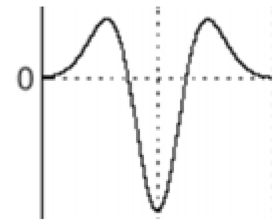
SIFT is an algorithm that finds contrasting blobs at different scales

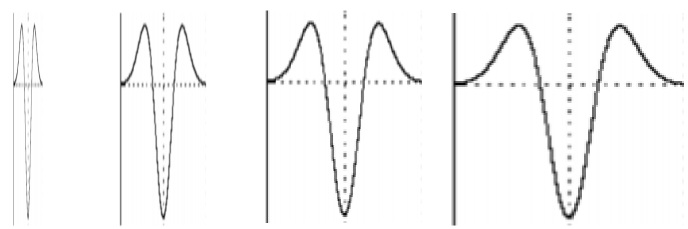
Original signal



(radius=8)

Difference of Gaussians Template

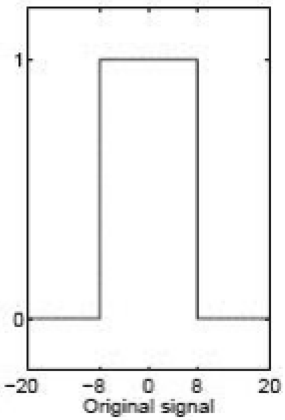




And these will do so at different scales

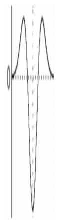
Blobs At Different Scales

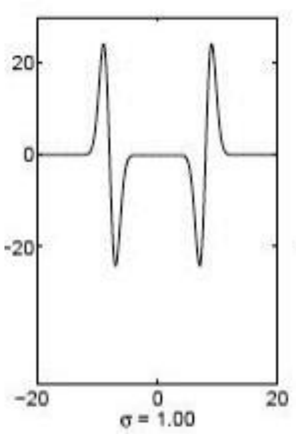
Original signal



(radius=8)

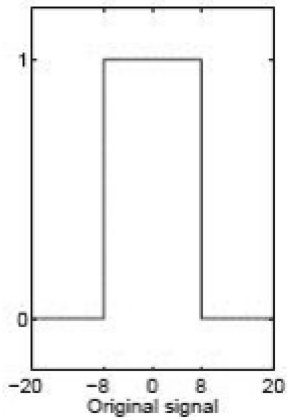
Difference of Gaussians Template





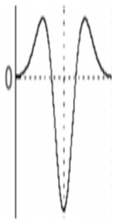
Blobs At Different Scales

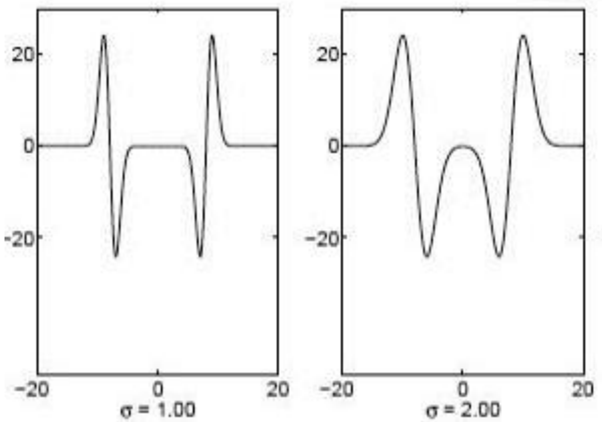
Original signal



(radius=8)

Difference of Gaussians Template

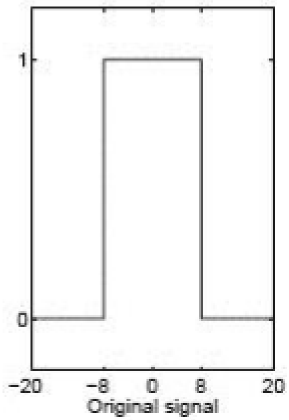




increasing σ

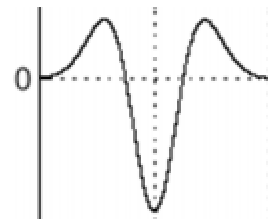
Blobs At Different Scales

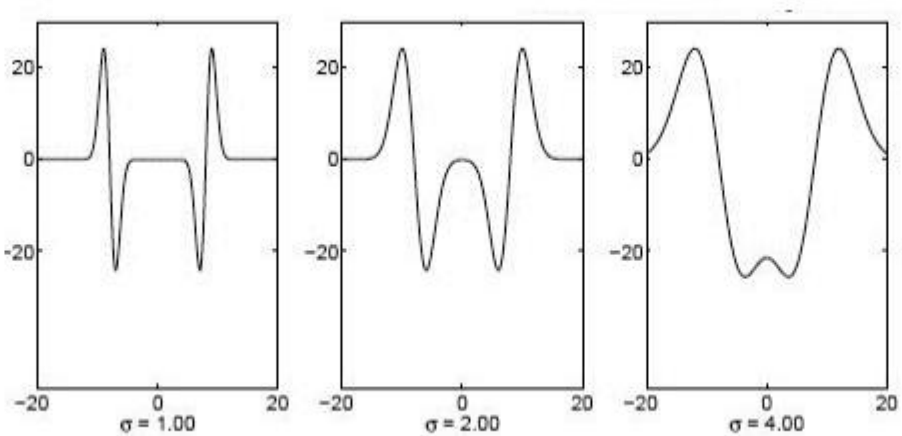
Original signal



(radius=8)

Difference of Gaussians Template

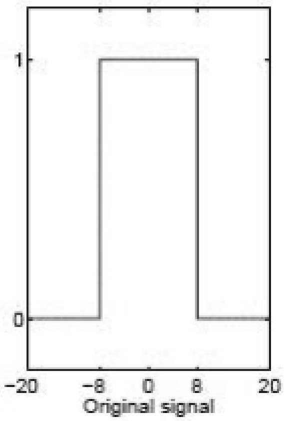




increasing σ

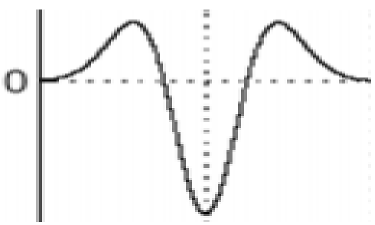
Blobs At Different Scales

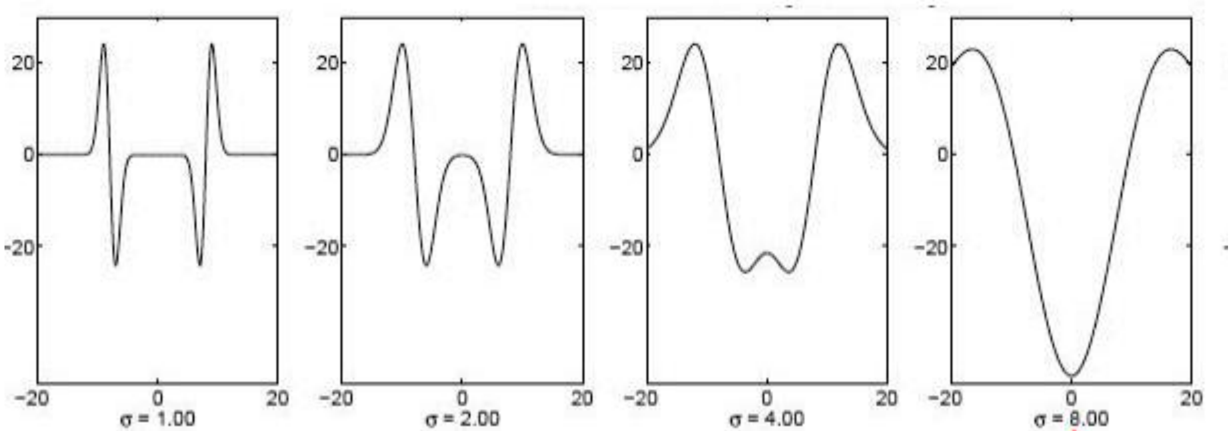
Original signal



(radius=8)

Difference of Gaussians Template

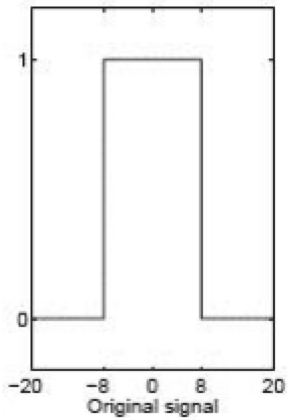




increasing σ

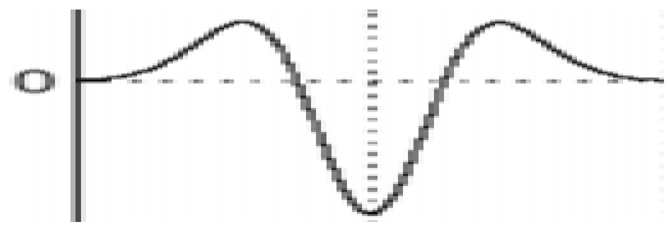
Blobs At Different Scales

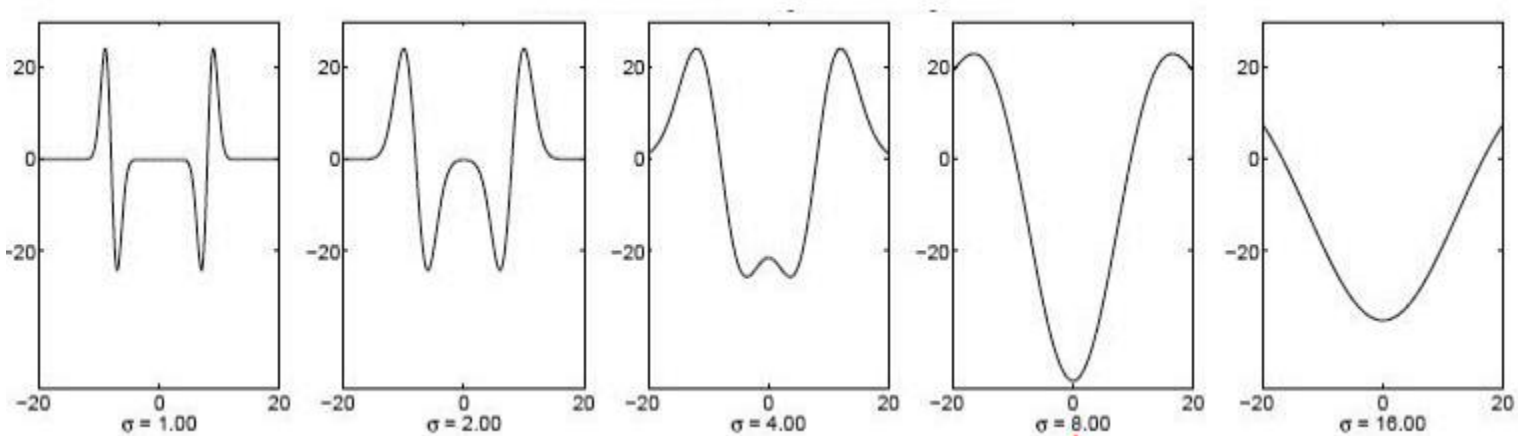
Original signal



(radius=8)

Difference of Gaussians Template





increasing σ

maximum

Computing SIFT Keypoints: Basic Steps

Source image I

⇒

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

⇒

DOG pyramid

Step 1b

Buid DOG pyramid

⇒

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Location refinement

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \to (x_i', y_i', \rho_i')$

⇐

Extremum pruning

Step 1e

Prune set of extrema

Keypoints = { all remaining (x_i', y_i', ρ_i') }

⇐

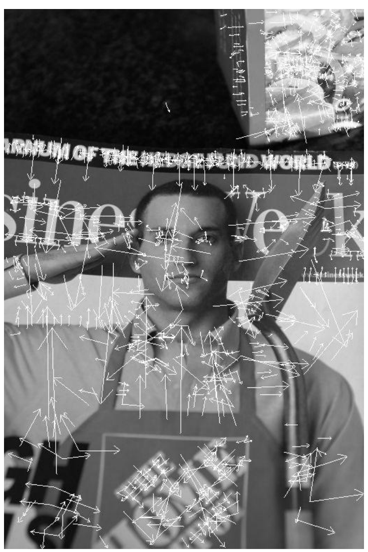
Orientation assign

Step 1f

Assign orientation to eatrema

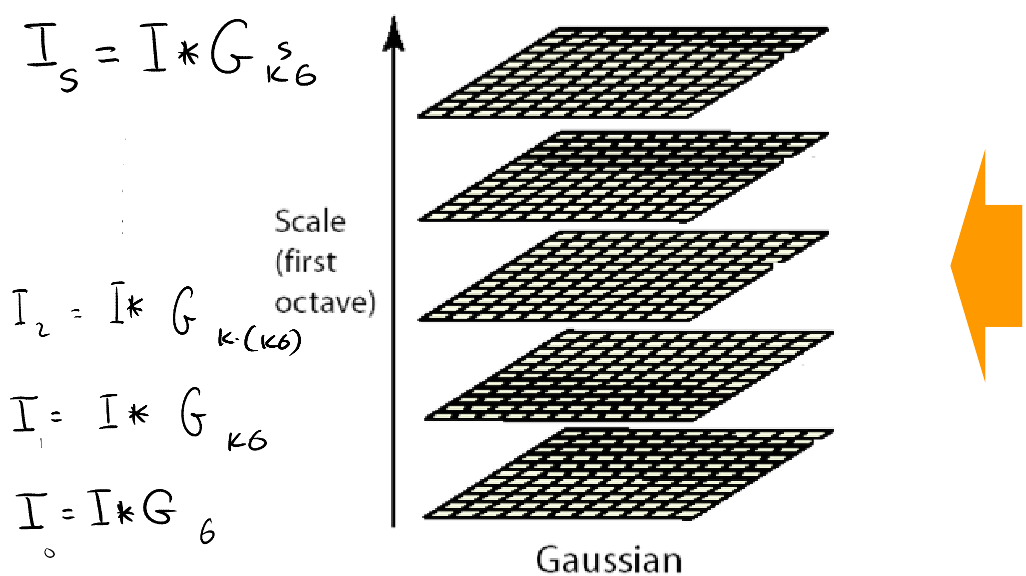
$P_i = (x_i', y_i', \rho_i', \theta_i)$

=



Step 1a: Construct a Gauss-like Pyramid

Step 1a: Compute a pyramid of Gauss-filtered images, organized into octaves of s+1 images



Each image is smoothed by a factor of k more than the image below

Step 1a: Construct a Gauss-like Pyramid

SIFT terminology – An octave is a set of Gauss-convolved images, $I_1, \ldots, I_s$ representing a doubling of the scale parameter σ between I_1 and I_s

each octave contains s+1 images

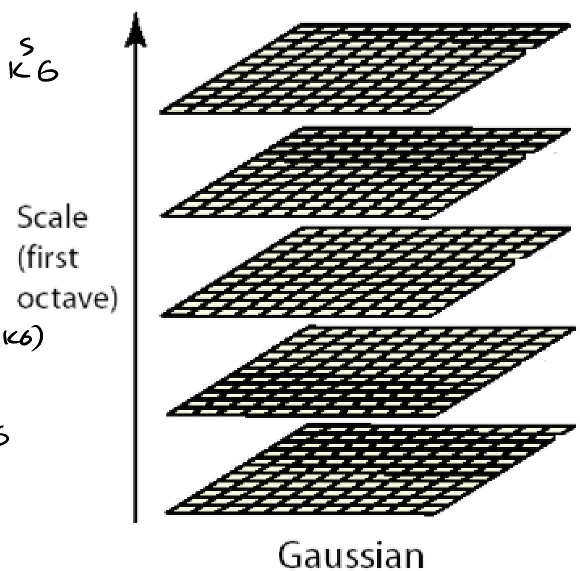
$I_s = I * G_{k^s \sigma}$

$\vdots$

$I_2 = I * G_{k \cdot (k\sigma)}$

$I_1 = I * G_{k\sigma}$

$I_0 = I * G_{\sigma}$



Each image is smoothed by a factor of k more than the image below

Step 1a: Construct a Gauss-like Pyramid

Images in the next octave are subsampled and stored at ½ the resolution of the previous octave.

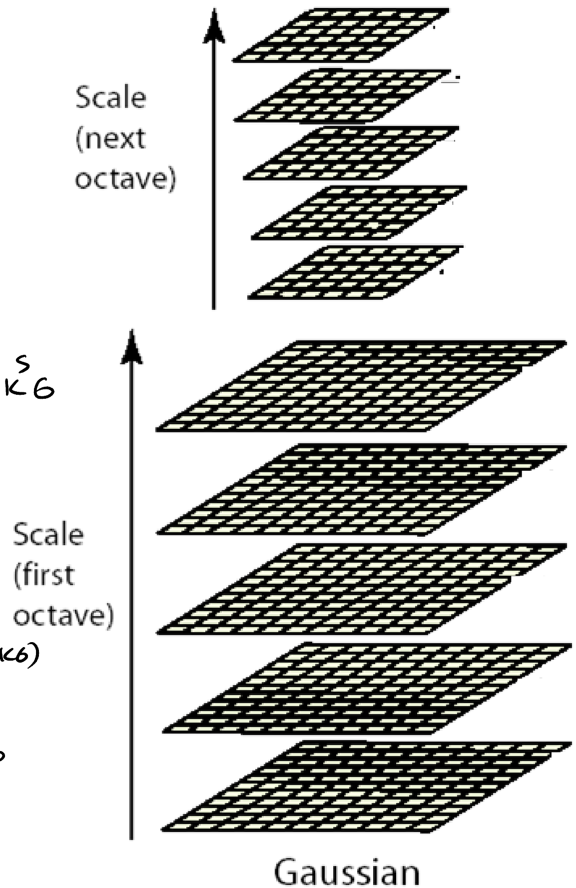
$I_s = I * G_{k^s \sigma}$

$I_2 = I * G_{k\cdot(k\sigma)}$

$I_1 = I * G_{k\sigma}$

$I_0 = I * G_{\sigma}$

each octave contains $s+1$ images



In practice:

$s = 3$ (3 images/octave)

$\sigma = 1.6$ (for 1st octave)

Computing SIFT Keypoints: Basic Steps

Source image I

⇒

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

⇒

DOG pyramid

Step 1b

Build DOG pyramid

⇒

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Location refinement

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$

⇐

Extremum pruning

Step 1e

Prune set of extrema

Keypoints = { all remaining (x_i', y_i', ρ_i') }

⇐

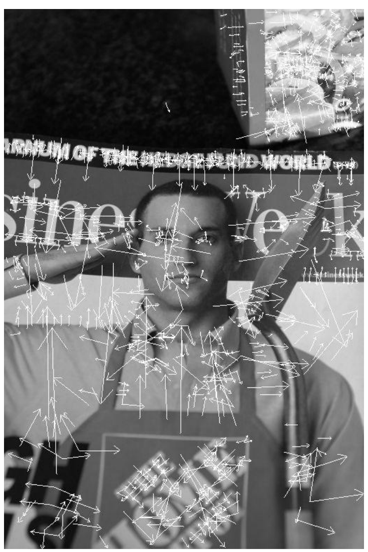
Orientation assign

Step 1f

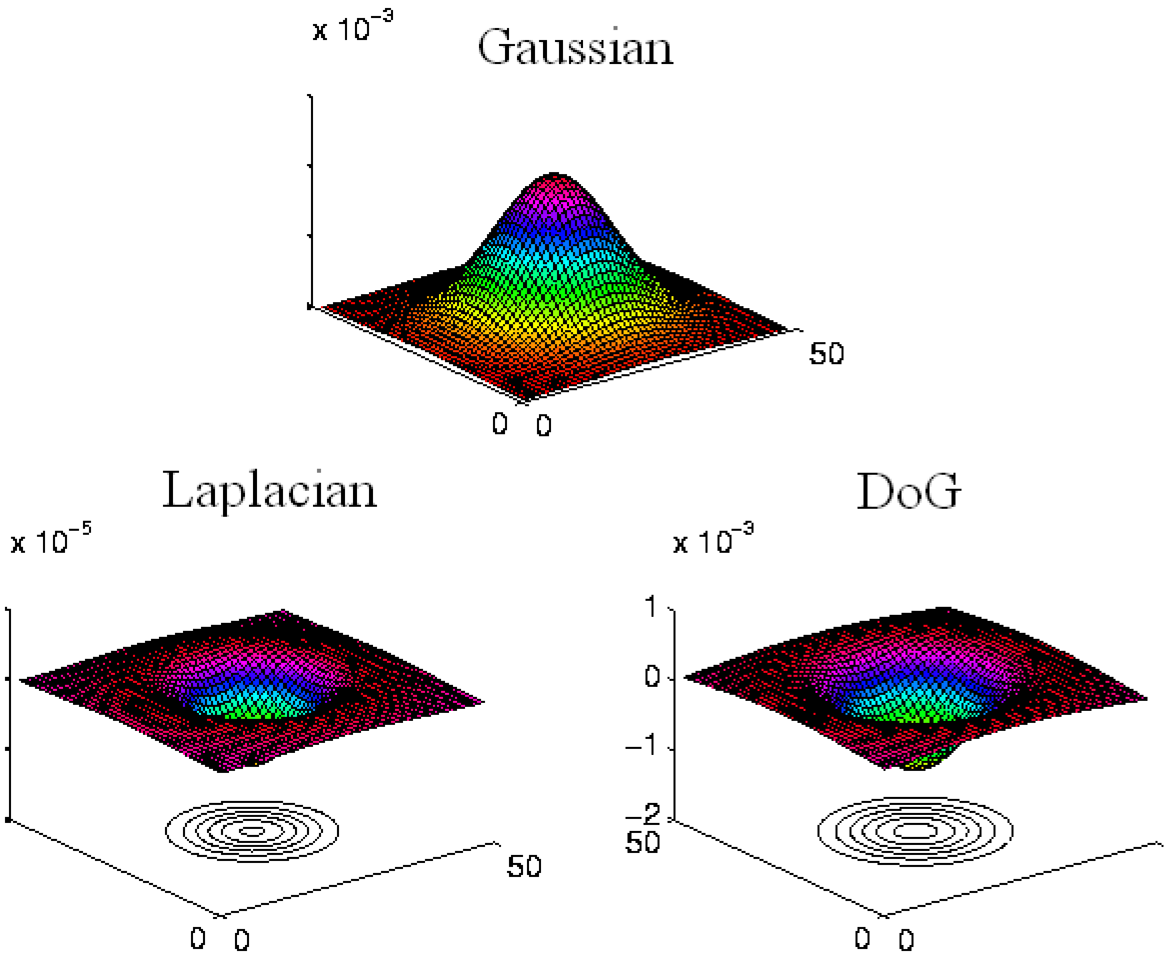
Assign orientation to extrema

$P_i = (x_i', y_i', \rho_i', \theta_i)$

=

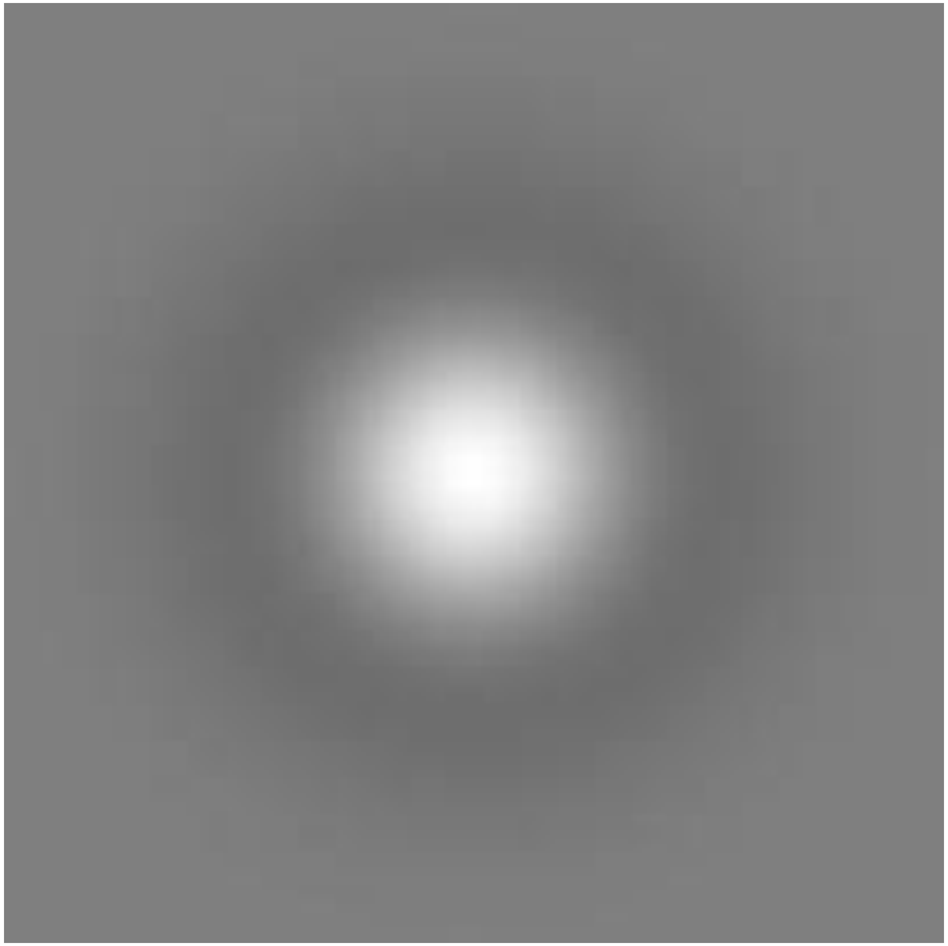


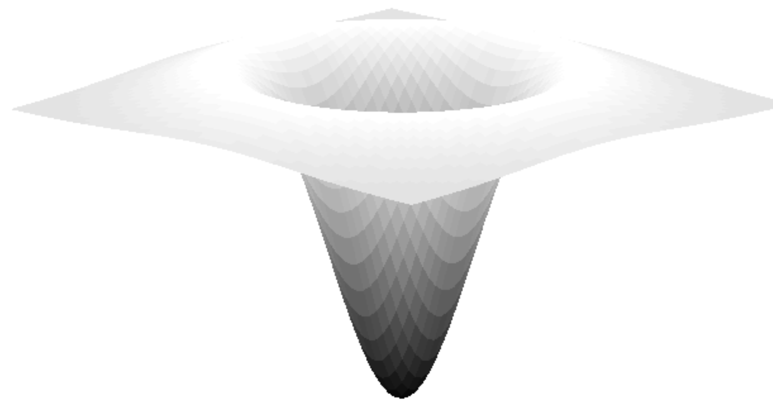
Sure, because it approximates the Laplacian.



but what is interesting about the Laplacian?

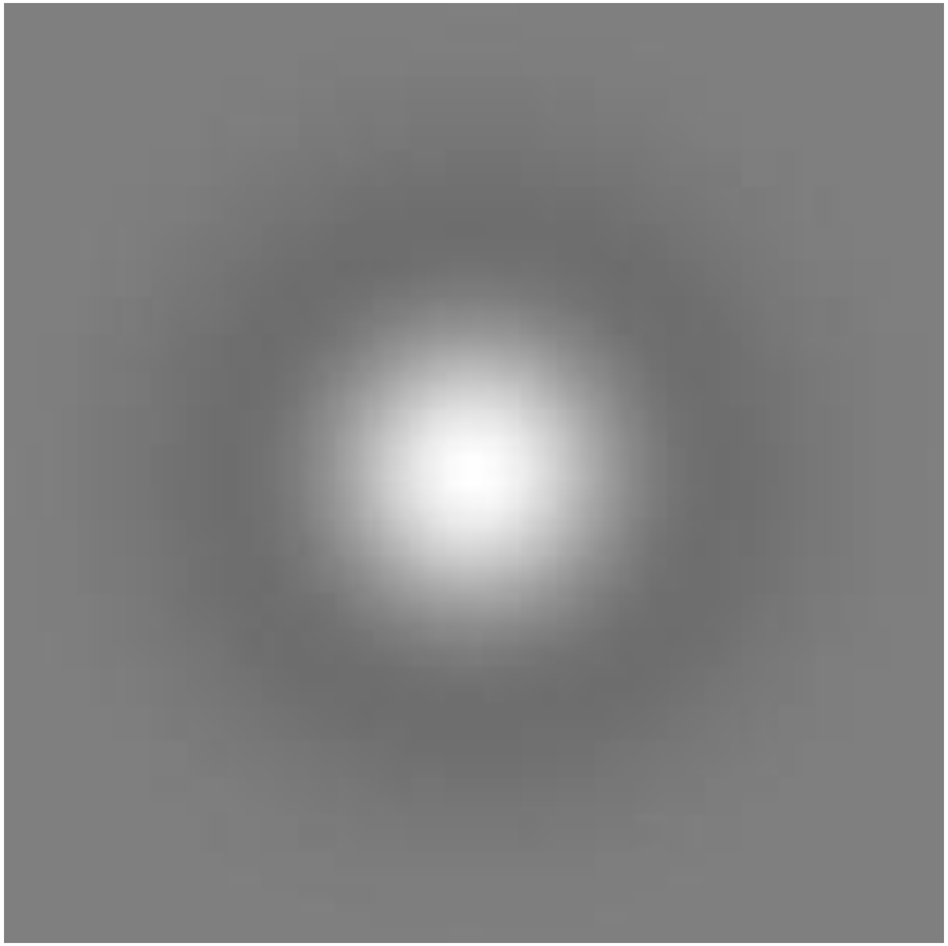
Sure, because it approximates the Laplacian.

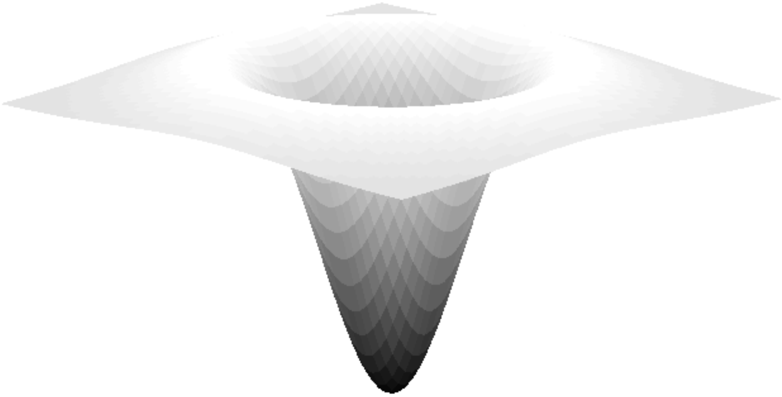




but what is interesting about the Laplacian?

Sure, because it approximates the Laplacian.





The Laplacian looks for coherent contrast variations in all directions

Step 1b: Compute Pyramid of DOG Images

How do we do this efficiently at multiple scales

$$D(x,y,\rho) = I(x,y) * (G(x,y,k\rho) - G(x,y,\rho))$$

$$\text{for } \rho = \{\sigma,\, k\sigma,\, k^2\sigma, \ldots, k^{s-1}\sigma\}$$

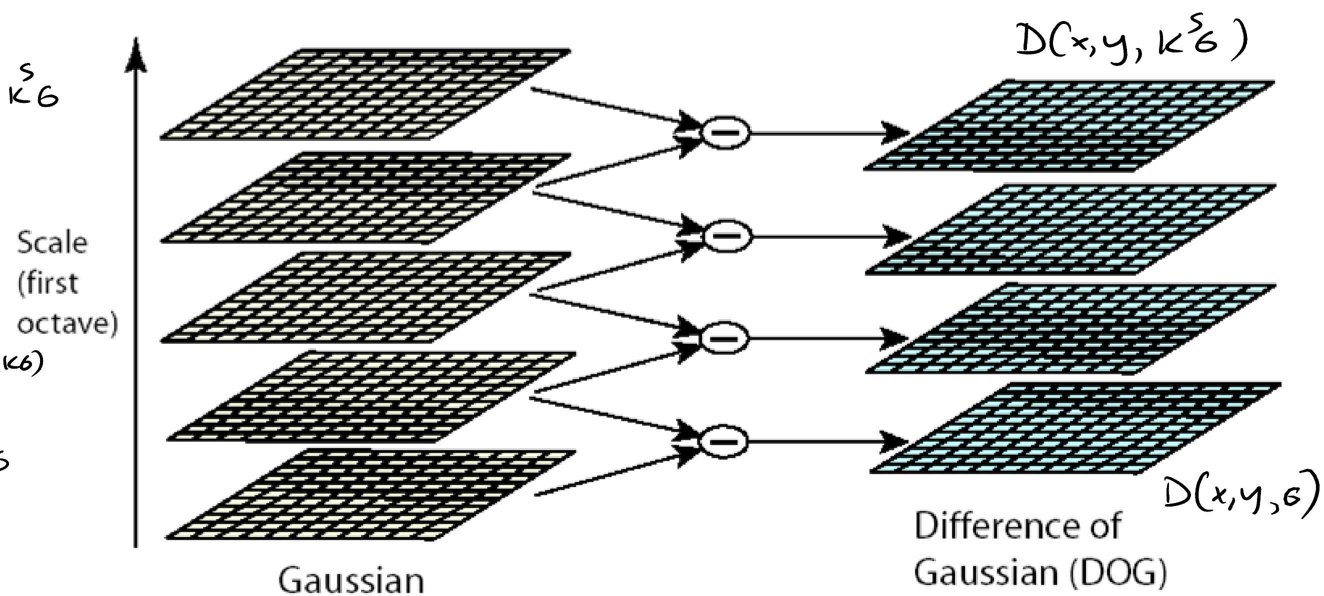
$I_s = I * G_{k^s\sigma}$

$\vdots$

$I_2 = I * G_{k\cdot(k\sigma)}$

$I_1 = I * G_{k\sigma}$

$I_0 = I * G_{\sigma}$



Step 1b: Compute Pyramid of DOG Images

And do so for all octaves

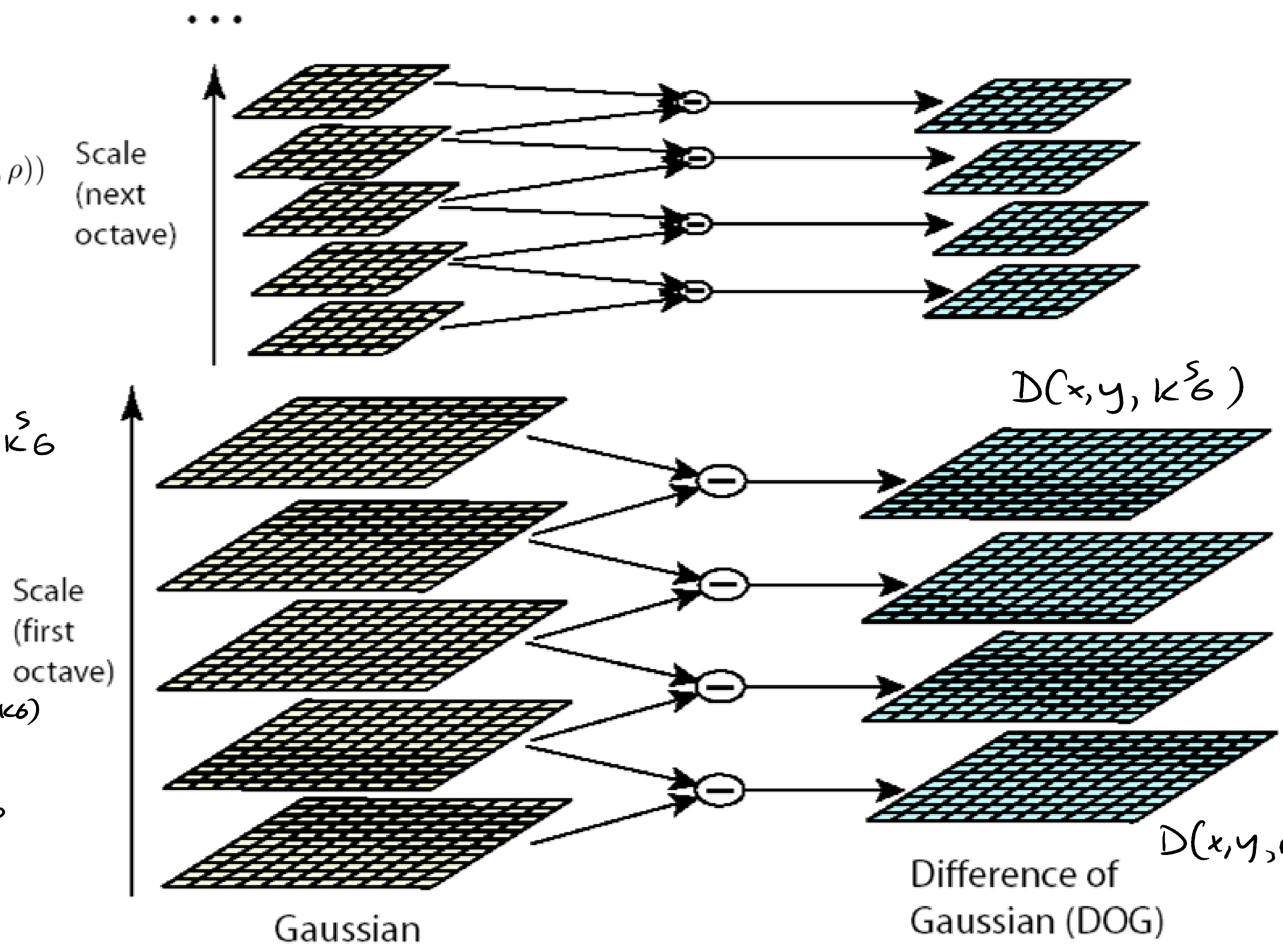
$$D(x,y,\rho) = I(x,y) * (G(x,y,k\rho) - G(x,y,\rho))$$
$$\text{for } \rho = \{\sigma, k\sigma, k^2\sigma, \ldots, k^{s-1}\sigma\}$$

$I_s = I * G_{k^s \sigma}$

$I_2 = I * G_{k\cdot(k\sigma)}$

$I_1 = I * G_{k\sigma}$

$I_0 = I * G_{\sigma}$

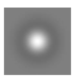


$D(x,y,k^s\sigma)$

$D(x,y,\sigma)$

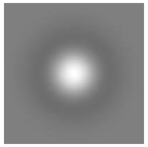
Reminder: Difference-of-Gaussian Filtering

$I * G_{\rho}$



Reminder: Difference-of-Gaussian Filtering

$I * G_{k\rho}$



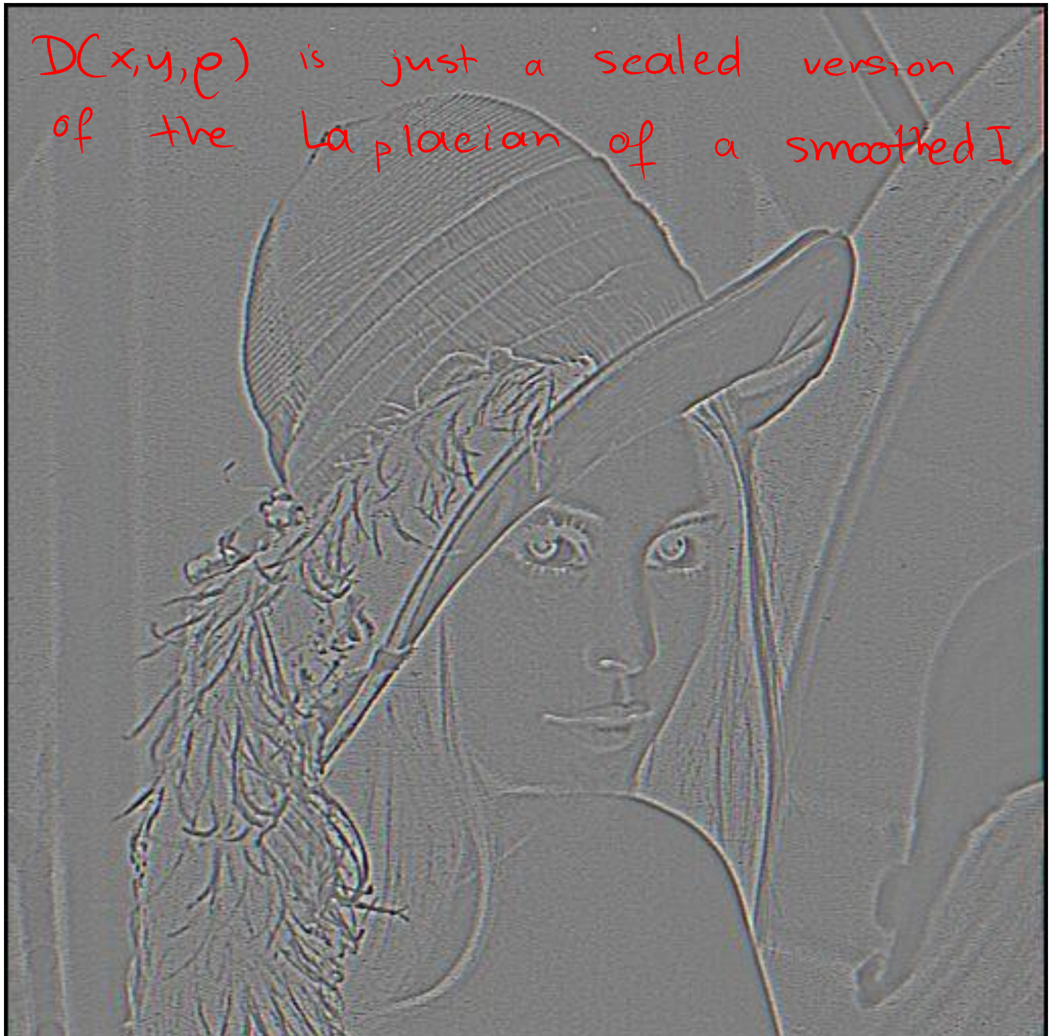
Reminder: Difference-of-Gaussian Filtering

Difference of two Gaussian-smoothed versions of I:

$$I * G_{k\rho} - I * G_{\rho} = I * (G_{k\rho} - G_{\rho})$$

(just the difference between two Gaussian masks)

$D(x,y,\rho)$ is just a scaled version of the Laplacian of a smoothed I



But we know that

$$G_{k\rho} - G_{\rho} = k\rho(k\rho-\rho)\nabla^2 G_{\rho}$$

$$\Downarrow$$

$$\Longrightarrow \quad D(x,y,\rho) = \left[\nabla^2 (I * G_{\rho})\right] \cdot \frac{1}{\rho^2 k(k-1)}$$

Computing SIFT Keypoints: Basic Steps

Source image I

⇒

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

⇒

DOG pyramid

Step 1b

Build DOG pyramid

⇒

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Location refinement

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$

⇐

Extremum pruning

Step 1e

Prune set of extrema

Keypoints = { all remaining (x_i', y_i', ρ_i') }

⇐

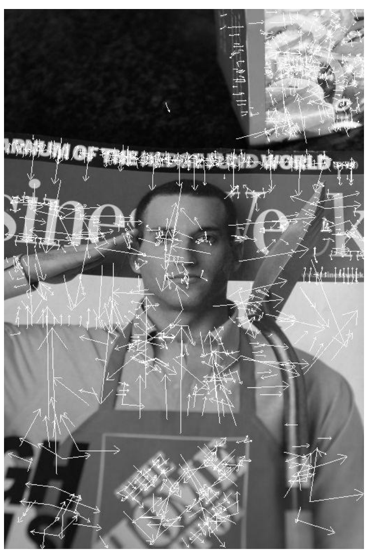
Orientation assign

Step 1f

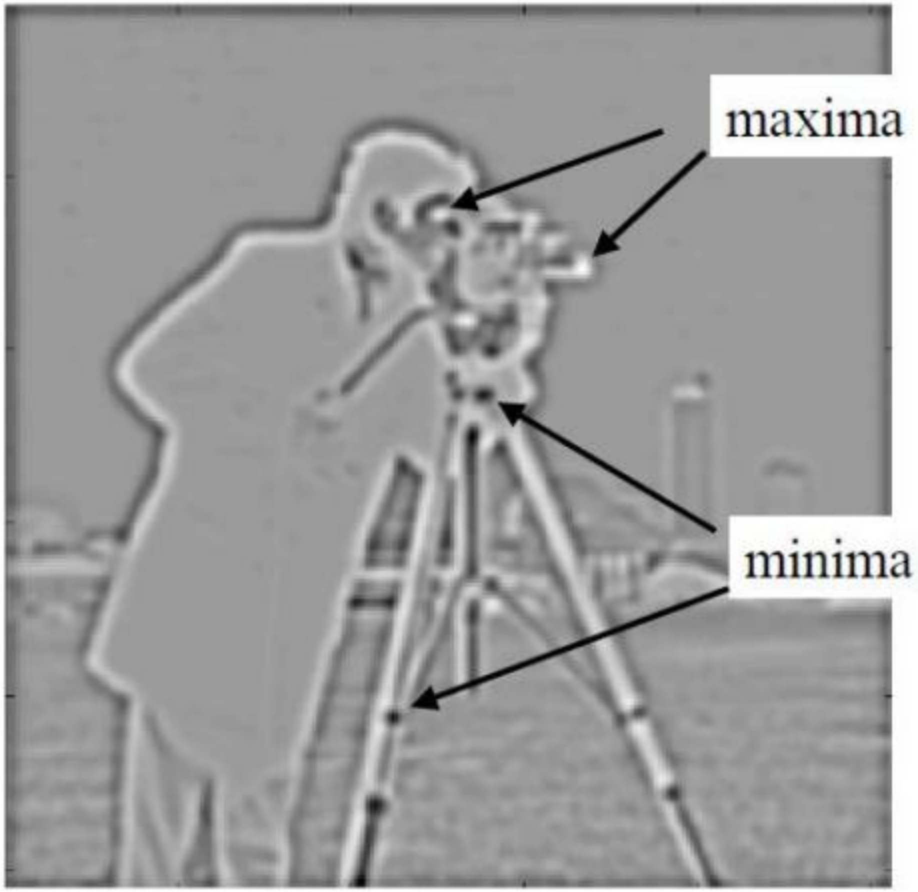
Assign orientation to extrema

$P_i = (x_i', y_i', \rho_i', \theta_i)$

=



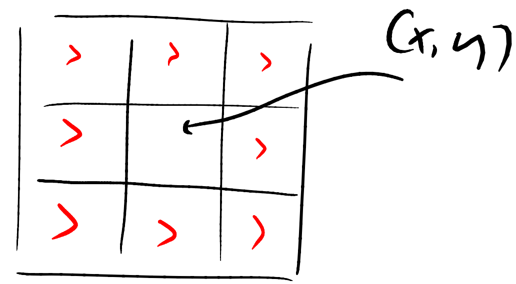
Finding Extrema In A Single Image



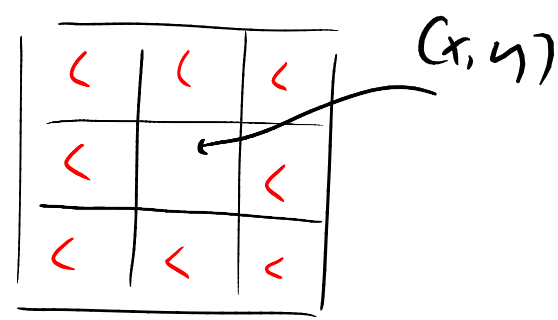
σ=2

Finding Extrema In A Single Image

Minimum at (x,y) if
D(x,y,r) > all neighbors

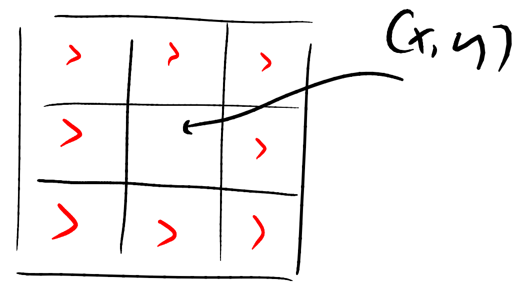


Minimum at (x,y) if
D(x,y,ρ) > all neighbors

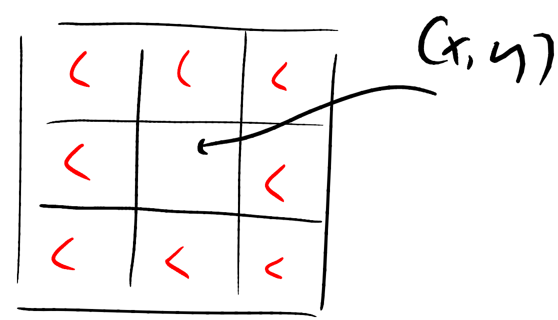


Finding Extrema In A Single Image

Minimum at (x,y) if
D(x,y,r) > all neighbors



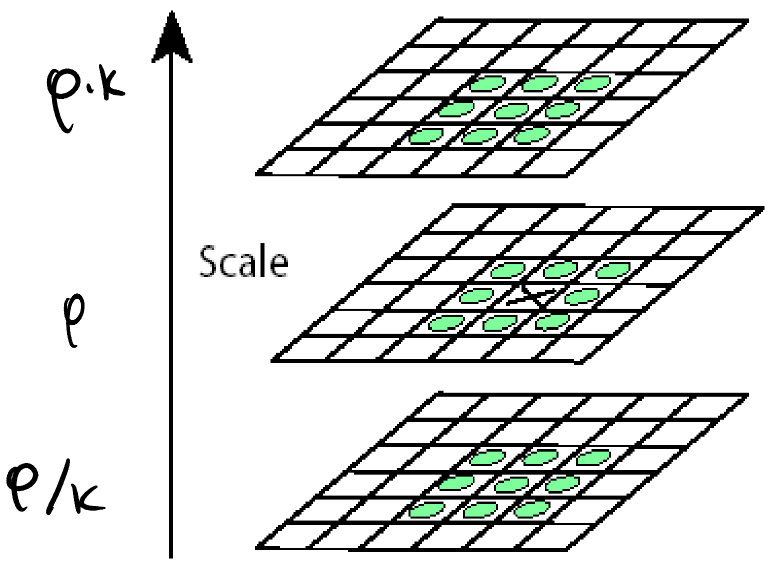
Minimum at (x,y) if
D(x,y,ρ) > all neighbors



It is a local operation

Step 1c: Detecting DOG Extrema

But because we want the extrema in the three dimensions x, y and ρ we look at the values of the adjacent scales too



x must also be bigger (or smaller) than all the neighbors in the adjacent scales

Still a local operation

There is usually a few (thousand) points that satisfy this in an image pyramid

Step 1c: Detecting DOG Extrema: Algorithm

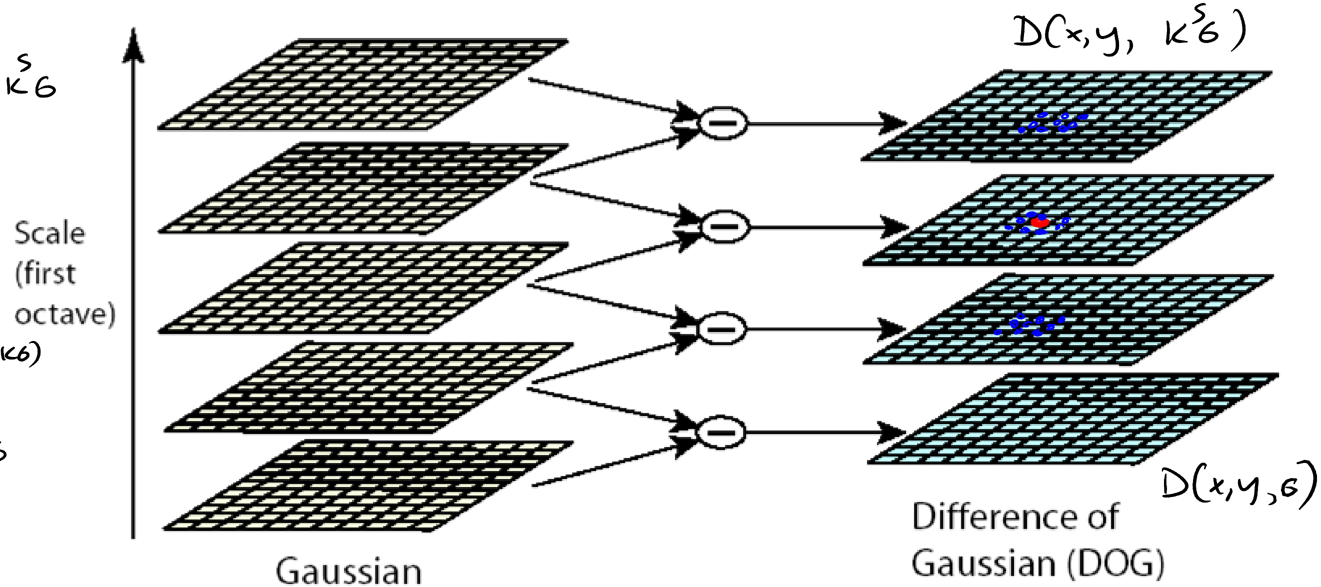
Algorithm:
For each (x,y,ρ), check whether D(x,y,ρ) is greater (or smaller than) all of its neighbours in the current scale and in the adjacent scales above and below.

$I_s = I * G_{k^s \sigma}$

$I_2 = I * G_{k \cdot (k\sigma)}$

$I_1 = I * G_{k\sigma}$

$I_0 = I * G_{\sigma}$



Step 1c: SIFT Keypoints = DOG Extrema

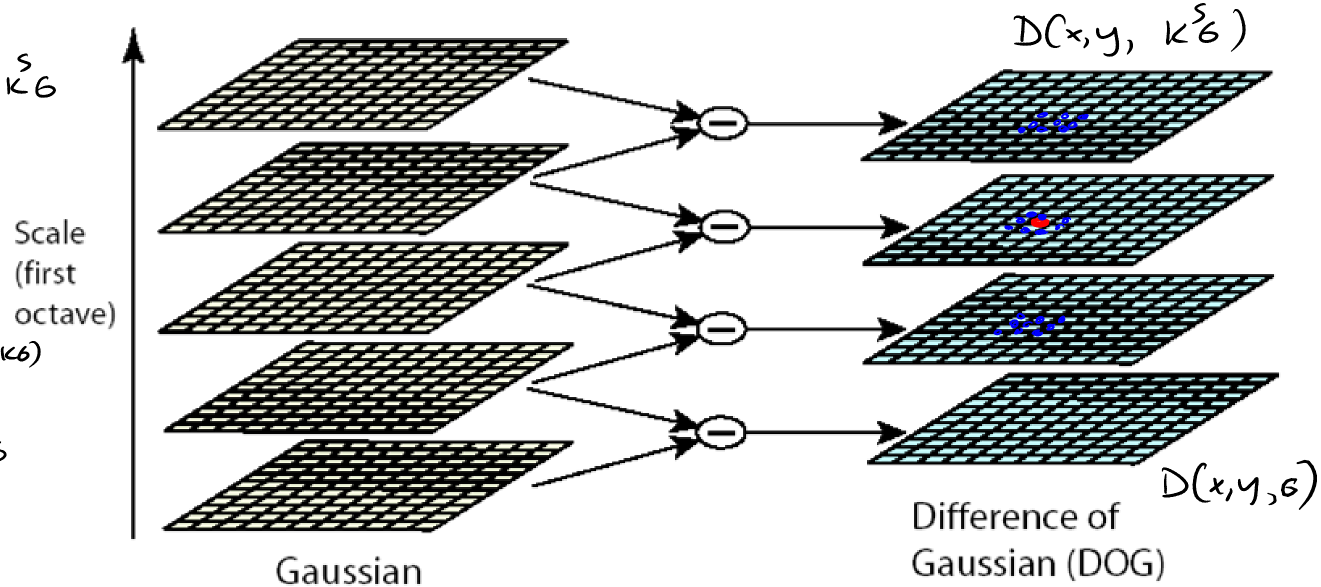
An extremum that is detected at D(x,y,ρ) defines the keypoint (x, y, ρ).

$I_s = I * G_{k^s \sigma}$

$I_2 = I * G_{k \cdot (k\sigma)}$

$I_1 = I * G_{k\sigma}$

$I_0 = I * G_{\sigma}$



Computing SIFT Keypoints: Basic Steps

Source image I

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

DOG pyramid

Step 1b

Buid DOG pyramid

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$

Location refinement

Step 1e

Prune set of extrema

Keypoints = { all remaining (x_i', y_i', ρ_i') }

Extremum pruning

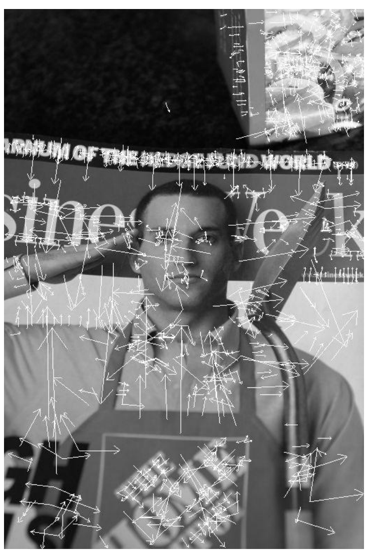
Step 1f

Assign orientation to eatrema

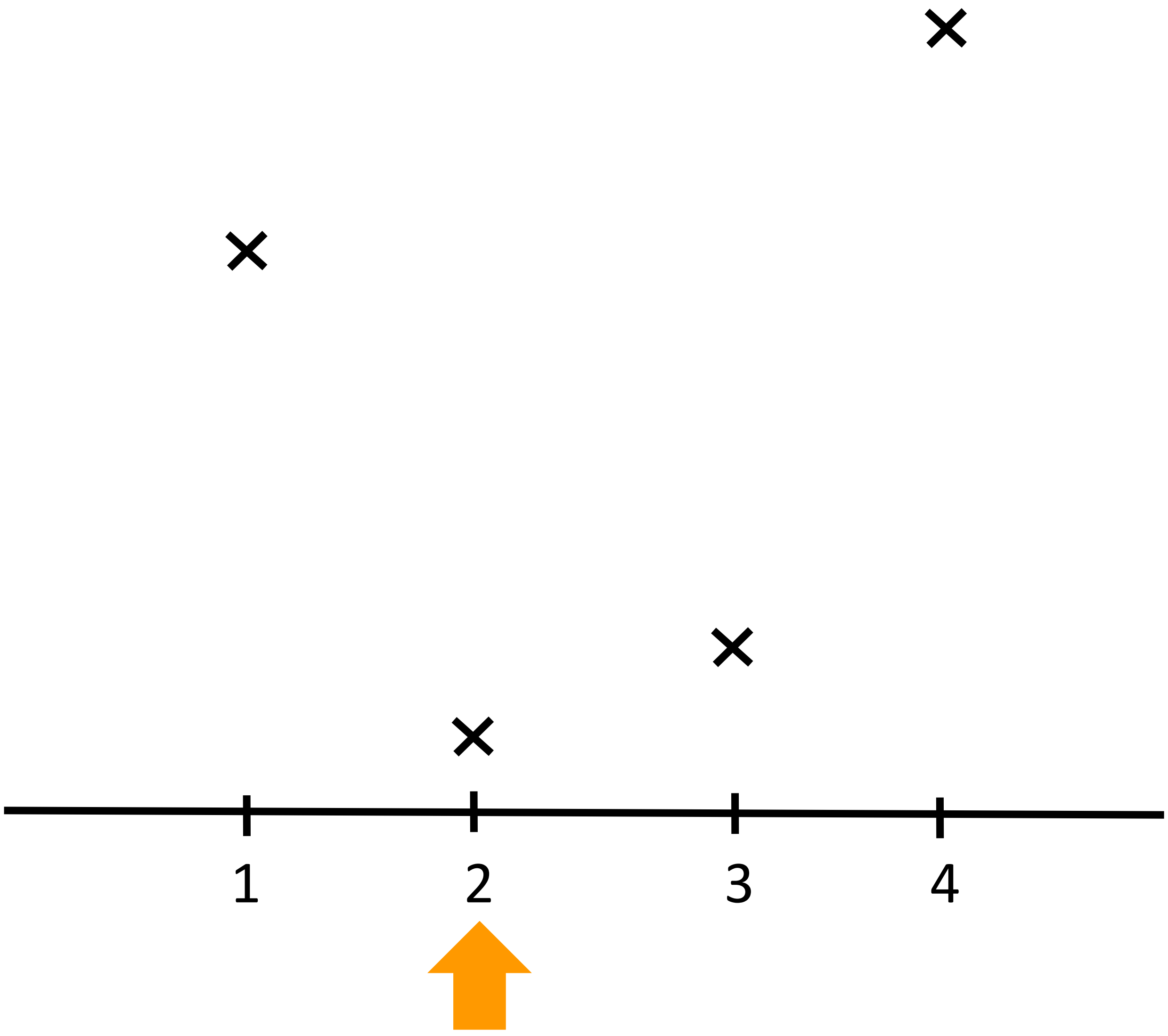
$P_i = (x_i', y_i', \rho_i', \theta_i)$

Orientation assign

=

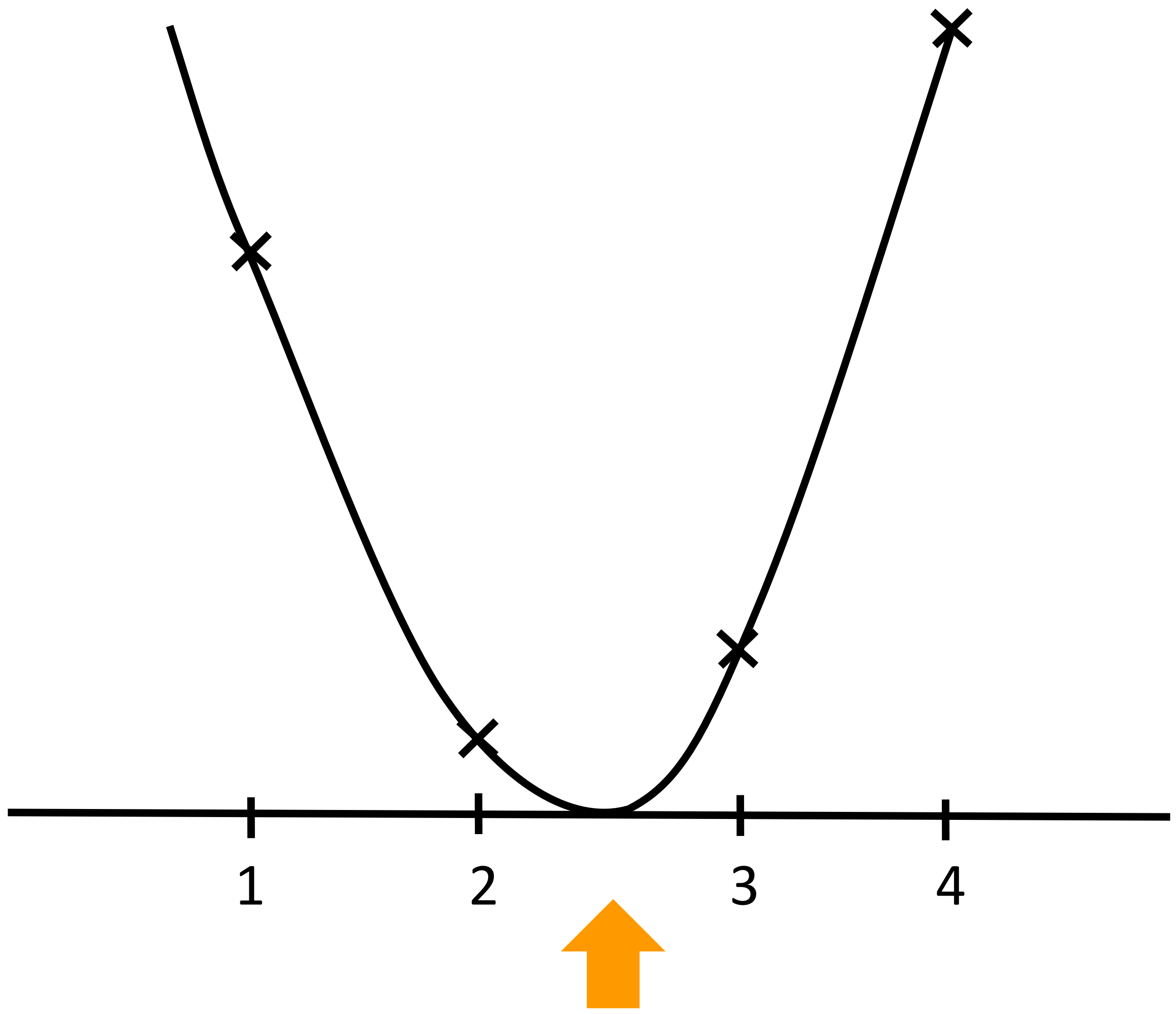


Pixels and Scales Are at Discrete Locations



Discrete approximation to the minimum: x = 2

Pixels and Scales Are at Discrete Locations



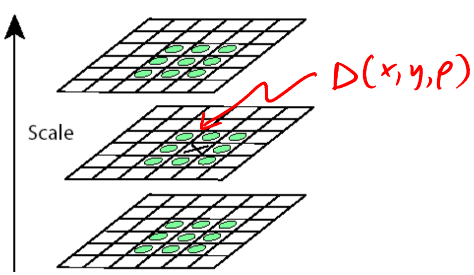
Real minimum: x = 2.48

Step 1d: Refining Location of Extrema

Use a 2nd order Taylor series approximation:

$$D(\Delta x, \Delta y, \Delta \rho) = D(x, y, \rho) + \frac{\partial D}{\partial x} \cdot \Delta x + \frac{\partial D}{\partial y} \cdot \Delta y + \frac{\partial D}{\partial \rho} \cdot \Delta \rho +$$

$$\frac{1}{2} \begin{bmatrix} \Delta x & \Delta y & \Delta \rho \end{bmatrix} \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial D^2}{\partial x \partial \rho} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial D^2}{\partial y \partial \rho} \\ \frac{\partial^2 D}{\partial x \partial \rho} & \frac{\partial^2 D}{\partial y \partial \rho} & \frac{\partial D^2}{\partial \rho^2} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \rho \end{bmatrix}$$



Step 1d: Refining Location of Extrema

Use a 2nd order Taylor series approximation:

$$D(\Delta x, \Delta y, \Delta \rho) = D(x,y,\rho) + \frac{\partial D}{\partial x}\cdot\Delta x + \frac{\partial D}{\partial y}\cdot\Delta y + \frac{\partial D}{\partial \rho}\cdot\Delta\rho +$$

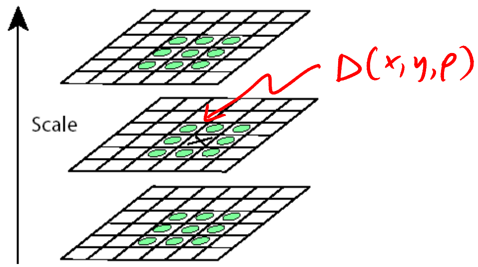
$$\frac{1}{2}\begin{bmatrix}\Delta x & \Delta y & \Delta\rho\end{bmatrix}\begin{bmatrix}\frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x\partial y} & \frac{\partial D^2}{\partial x\partial\rho}\\ \frac{\partial^2 D}{\partial x\partial y} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial D^2}{\partial y\partial\rho}\\ \frac{\partial^2 D}{\partial x\partial\rho} & \frac{\partial^2 D}{\partial y\partial\rho} & \frac{\partial D^2}{\partial \rho^2}\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\rho\end{bmatrix}$$

Derivatives are estimated using finite differences

$$\frac{\partial D}{\partial x} \simeq D(x+1, y, \rho) - D(x,y,\rho)$$

Step 1d: Refining Location of Extrema

①



In vector notation

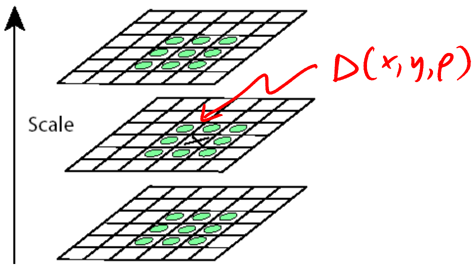
$$\vec{x} = \begin{bmatrix} x \\ y \\ \rho \end{bmatrix} \qquad \Delta\vec{x} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \rho \end{bmatrix}$$

$$D(\Delta\vec{x}) = D(\vec{x}) + \left(\frac{\delta D}{\delta \vec{x}}\right)^{\top} \Delta\vec{x} + \frac{1}{2}(\Delta\vec{x})^{\top}\left(\frac{\delta^2 D}{\delta \vec{x}^2}\right)\Delta\vec{x}$$

And knowing that the function is an extrema when…

Step 1d: Refining Location of Extrema

1



In vector notation

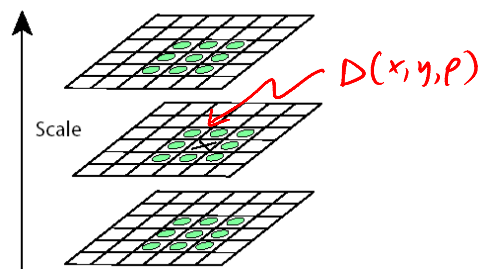
$$\vec{x} = \begin{bmatrix} x \\ y \\ \rho \end{bmatrix} \qquad \Delta\vec{x} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \rho \end{bmatrix}$$

$$D(\Delta\vec{x}) = D(\vec{x}) + \left(\frac{\delta D}{\delta \vec{x}}\right)^{\top} \Delta\vec{x} + \frac{1}{2}(\Delta\vec{x})^{\top} \left(\frac{\delta^2 D}{\delta \vec{x}^2}\right) \Delta\vec{x}$$

$D(\Delta\vec{x})$ is a function of $\Delta\vec{x}$, and we know it will have an extrema when…

Step 1d: Refining Location of Extrema

1



In vector notation

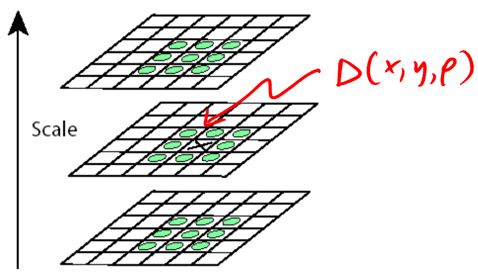
$$\vec{x} = \begin{bmatrix} x \\ y \\ \rho \end{bmatrix} \qquad \Delta\vec{x} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \rho \end{bmatrix}$$

$$D(\Delta\vec{x}) = D(\vec{x}) + \left(\frac{\delta D}{\delta\vec{x}}\right)^{\top} \Delta\vec{x} + \frac{1}{2}(\Delta\vec{x})^{\top}\left(\frac{\delta^2 D}{\delta\vec{x}^2}\right)\Delta\vec{x}$$

$D(\Delta\vec{x})$ is a function of $\Delta\vec{x}$, and we know it will have an extrema when the first derivative with respect to $\Delta\vec{x}$ is zero!

Step 1d: Refining Location of Extrema

1



In vector notation

$$\vec{x} = \begin{bmatrix} x \\ y \\ \rho \end{bmatrix} \qquad \Delta\vec{x} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \rho \end{bmatrix}$$

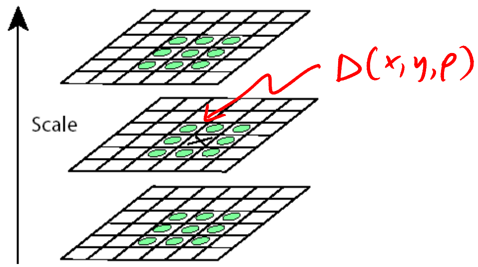
$$D(\Delta\vec{x}) = D(\vec{x}) + \left(\frac{\delta D}{\delta \vec{x}}\right)^{\top} \Delta\vec{x} + \frac{1}{2}(\Delta\vec{x})^{\top}\left(\frac{\delta^2 D}{\delta \vec{x}^2}\right)\Delta\vec{x}$$

To find the extrema, take the derivative of $D(\Delta\vec{x})$ with respect to a displacement $\Delta\vec{x}$ and solve for

$$\frac{\delta D}{\delta(\Delta\vec{x})} = 0$$

Step 1d: Refining Location of Extrema

①



The 2nd degree Taylor Expansion is:

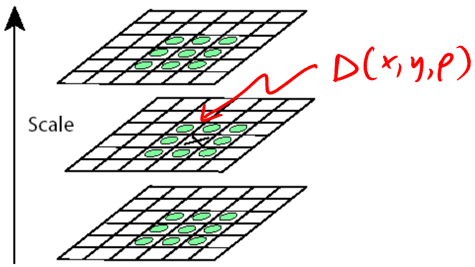
$$D(\Delta\vec{x}) = D(\vec{x}) + \left(\frac{\delta D}{\delta \vec{x}}\right)^{\top} \Delta\vec{x} + \frac{1}{2}(\Delta\vec{x})^{\top} \left(\frac{\delta^2 D}{\delta \vec{x}^2}\right) \Delta\vec{x}$$

And the derivative wrt $\Delta\vec{x}$ is:

$$\frac{\delta D}{\delta(\Delta\vec{x})} = \left(\frac{\delta D}{\delta \vec{x}}\right)^{\top} + \left(\frac{\delta^2 D}{\delta \vec{x}^2}\right)(\Delta\vec{x})$$

Step 1d: Refining Location of Extrema

1

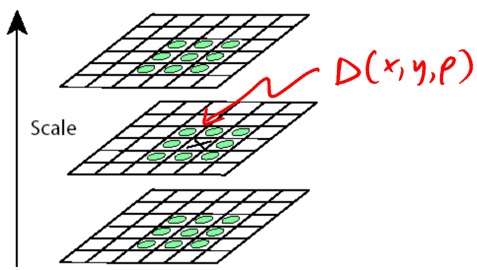


The extremum of D is when the derivative is equal to zero:

$$\Delta\vec{x} = \left(\frac{\delta^2 D}{\delta\vec{x}^2}\right)^{-1}\left(\frac{\delta D}{\delta\vec{x}}\right)$$

Step 1d: Refining Location of Extrema

The refinement is complete when the displacement is added to the initial estimate of the extremum.



Extremum at:

$$(x + \Delta x,\ y + \Delta y,\ \rho + \Delta\rho)$$

Where $\Delta\vec{x} = \left(\frac{\delta^2 D}{\delta \vec{x}^2}\right)^{-1} \left(\frac{\delta D}{\delta \vec{x}}\right)$ and:

$$\Delta\vec{x} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\rho \end{bmatrix} \qquad \frac{\partial D}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial D}{\partial x} \\ \frac{\partial D}{\partial y} \\ \frac{\partial D}{\partial \rho} \end{bmatrix} \qquad \frac{\partial D^2}{\partial \vec{x}^2} = \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial D^2}{\partial x \partial \rho} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial D^2}{\partial y \partial \rho} \\ \frac{\partial^2 D}{\partial x \partial \rho} & \frac{\partial^2 D}{\partial y \partial \rho} & \frac{\partial D^2}{\partial \rho^2} \end{bmatrix}$$

Computing SIFT Keypoints: Basic Steps

Source image I

⇒

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

⇒

DOG pyramid

Step 1b

Buid DOG pyramid

⇒

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$

Location refinement

⇐

Step 1e

Prune set of extrema

Keypoints = { all remaining (x_i', y_i', ρ_i') }

Extremum pruning

⇐

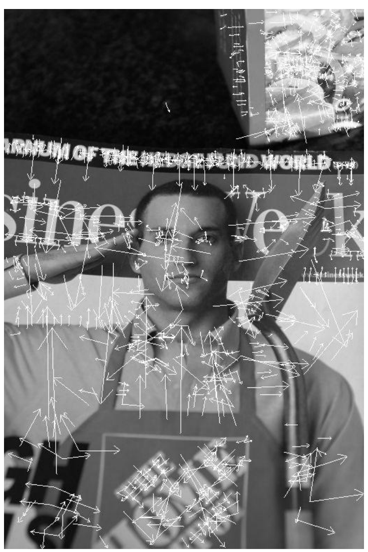
Step 1f

Assign orientation to extrema

$P_i = (x_i', y_i', \rho_i', \theta_i)$

Orientation assign

=



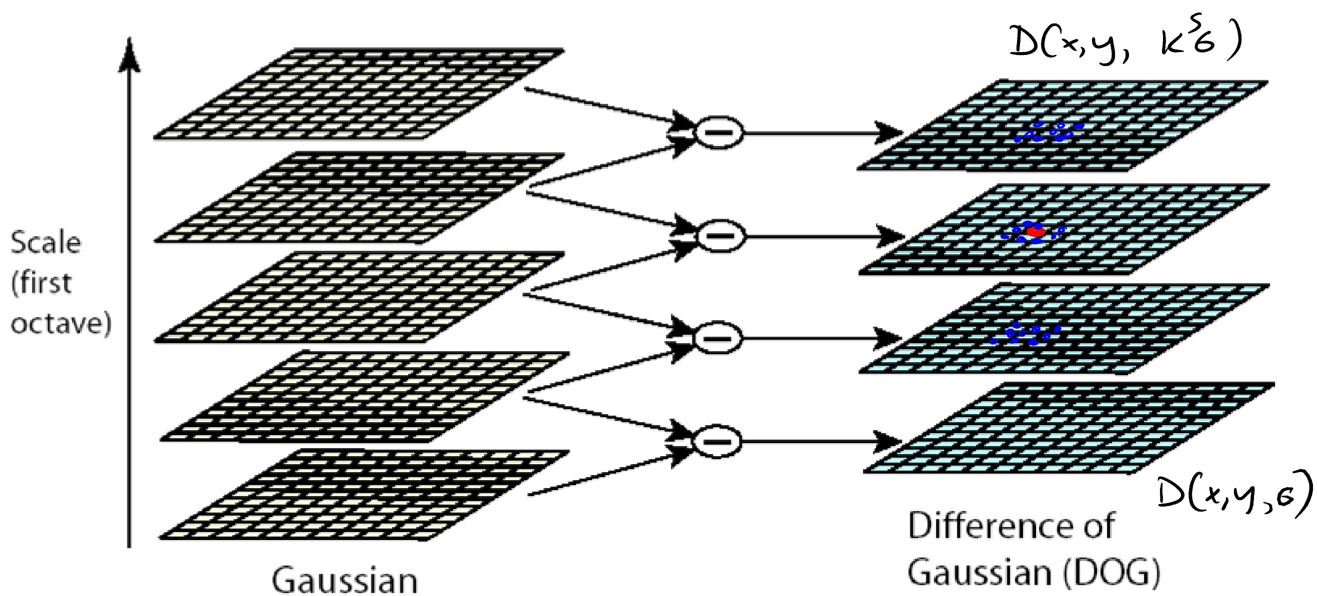
Step 1e: Pruning "Insignificant" Extrema

Discard extrema that are weak or that correspond to edges:

Strength is simply measured by the (absolute) magnitude of the Difference of Gaussians at the interest point

$$|D(x_i', y_i', \sigma_i')| = \lambda$$

In practice λ=0.03, when images are in [0, 1].

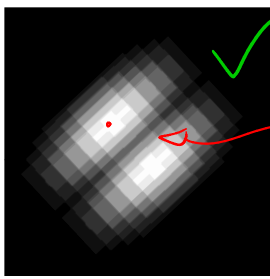


Step 1e: Pruning "Insignificant" Extrema

Also prune extrema that correspond to edges

corner-like extremum

position is well-constrained

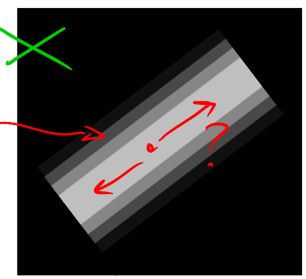


keep

$D(x_i', y_i', p_i')$

edge-like extremum

prune



Position along edge is not constrained

Reminder: Single-scale Lowe Feature Detector

Also prune extrema that correspond to edges

Compute $\frac{\partial^2 S}{\partial x^2}, \frac{\partial^2 S}{\partial y^2}, \frac{\partial^2 S}{\partial x \partial y}$

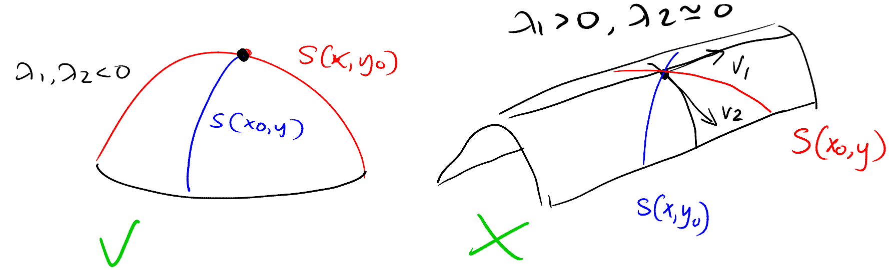
Compute $Det(H)$ at each pixel

Compute $Tr(H)$

Require $\frac{\lambda_1}{\lambda_2} = r$ = small

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

for SIFT: $r = 10$



λ_1, λ_2 = eigenvalues / principal curvatures of $S(x,y)$

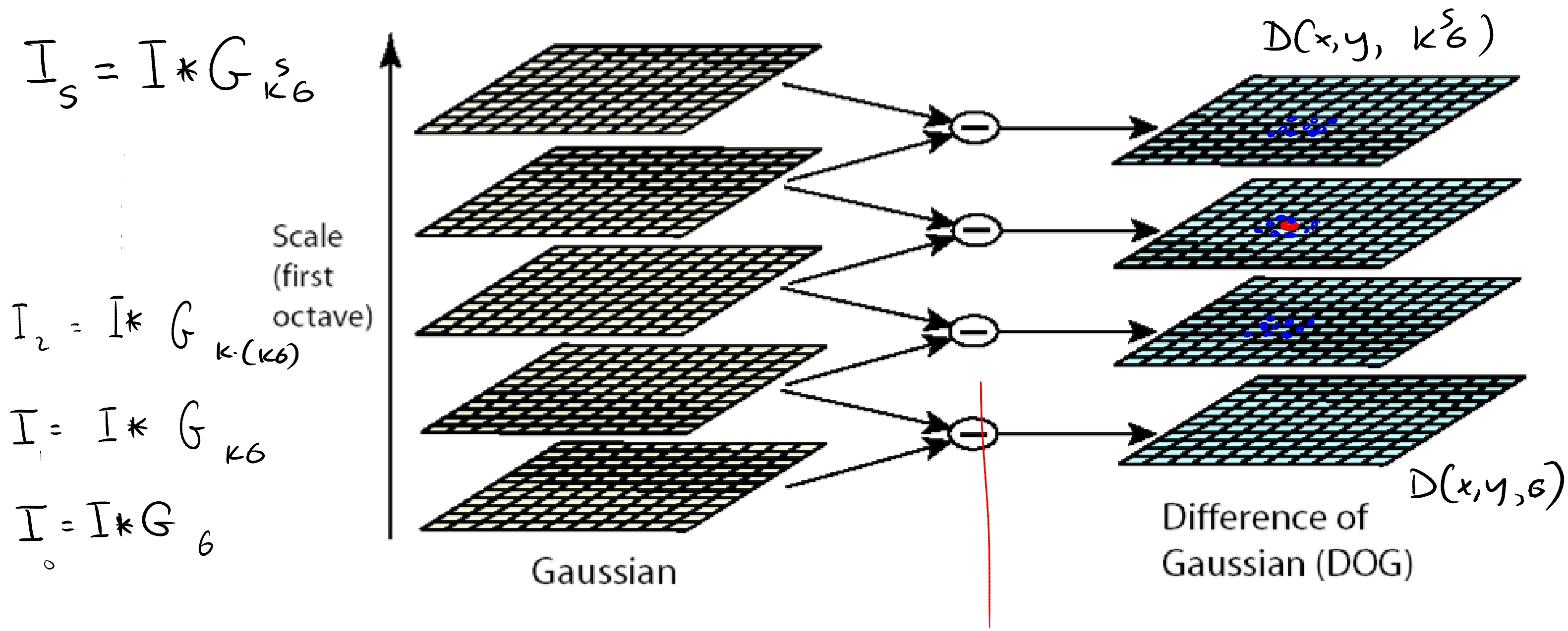
See lecture 6 for details!

Step 1e: Pruning "Insignificant" Extrema

Algorithm:

Compute the Hessian H of S(x,y) = D(x,y,ρ'), at (x,y) = (x',y'), and prune if:

$$\frac{Tr^2(H)}{Det(H)} > \left(\frac{11}{10}\right)^2$$



Computing SIFT Keypoints: Basic Steps

Source image I

⇒

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

⇒

DOG pyramid

Step 1b

Buid DOG pyramid

⇒

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Location refinement

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$

⇐

Extremum pruning

Step 1e

Prune set of extrema

Keypoints = { all remaining (x_i', y_i', ρ_i') }

⇐

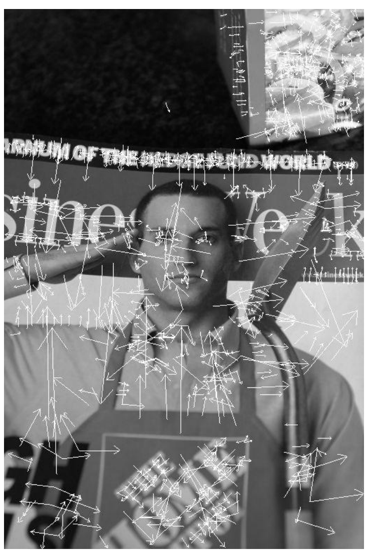
Orientation assign

Step 1f

Assign orientation to eatrema

$P_i = (x_i', y_i', \rho_i', \theta_i)$

=



Step 1f: Keypoint Orientation Assignment

Compute a Gaussian image with the relevant scale ρ'

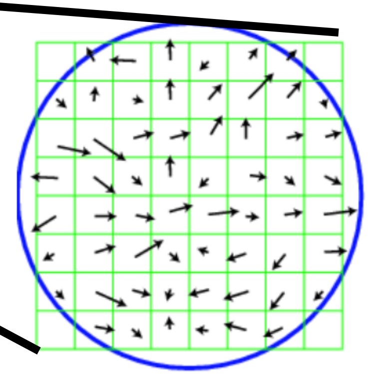
$I * G_{\rho_i'}$

Step 1f: Keypoint Orientation Assignment

Compute a Gaussian image with the relevant scale ρ’

$I * G_{\rho_i'}$

Compute the gradient magnitudes and orientations in the neighborhood of the detected feature point

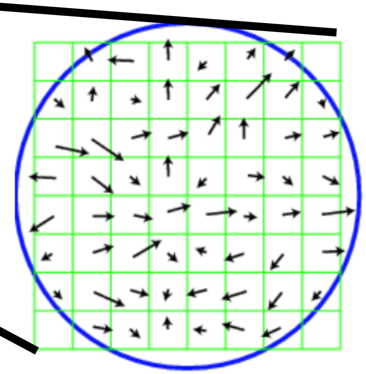


Step 1f: Keypoint Orientation Assignment

Compute a Gaussian image with the relevant scale ρ’

$I * G_{\rho_i'}$

Compute the gradient magnitudes and orientations in the neighborhood of the detected feature point

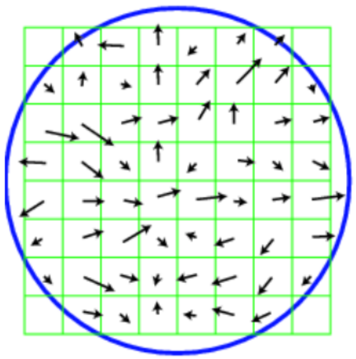


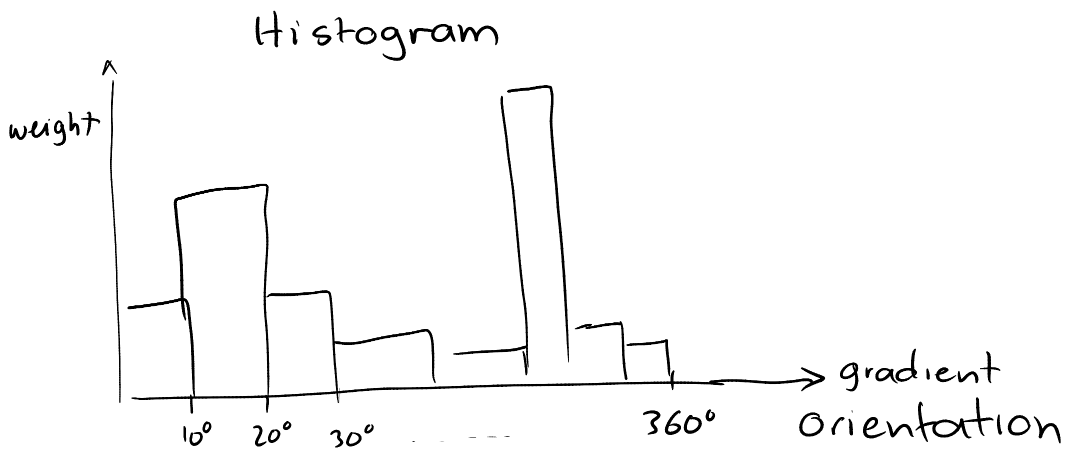
Step 1f: Keypoint Orientation Assignment

Compute a Gaussian image with the relevant scale ρ'

Compute the gradient magnitudes and orientations in the neighborhood of the detected feature point

Compute a histogram of orientations:





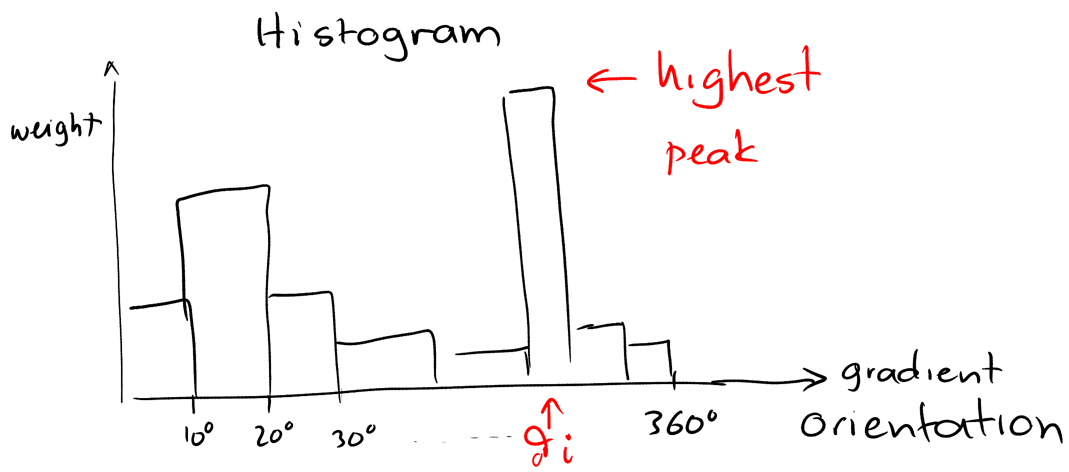
Step 1f: Keypoint Orientation Assignment

Compute a Gaussian image with the relevant scale ρ'

Compute the gradient magnitudes and orientations in the neighborhood of the detected feature point

Compute a histogram of orientations

Take the highest peak as the canonical orientation



Step 1f: Keypoint Orientation Assignment

Computing Histogram of Orientations

$I * G_{\rho_i'}$

Orientations are divided into 36 bins (one every 10 degrees). Pixel (x,y) contributes to the bin corresponding to the gradient orietation θ at (x,y).

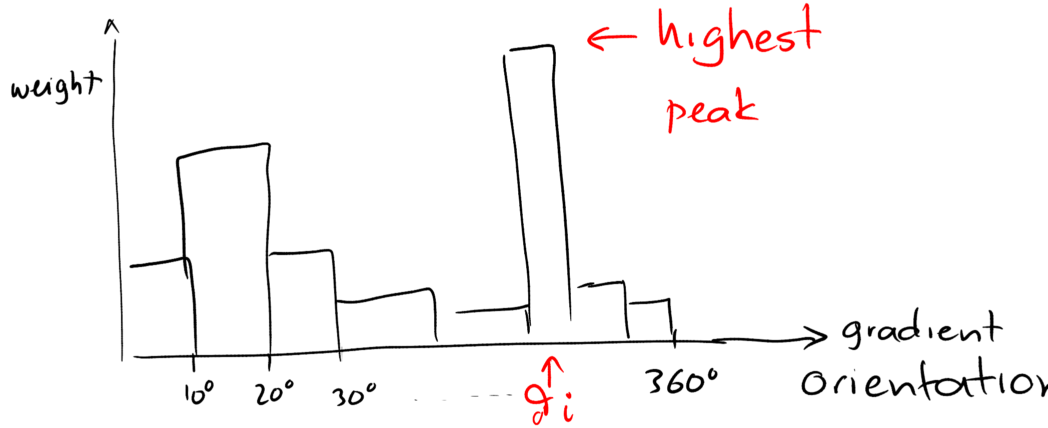
The contribution to that bin is equal to $|\nabla I(x,y)| \cdot G_{1.5\rho'}(d)$

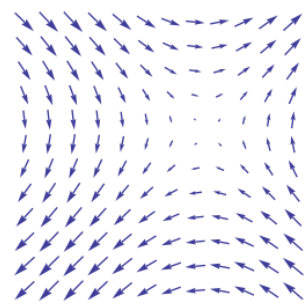
where $G_{1.5\rho'}$ is a circular Gaussian weighting function and *d* is the distance to the feature point

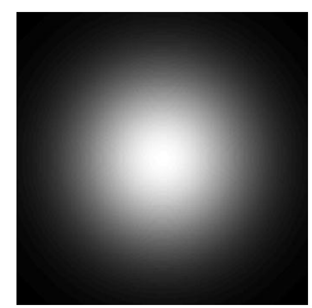
Step 1f: Keypoint Orientation Assignment

Computing Histogram of Orientations

$I * G_{\rho_i'}$







$$|\nabla I(x, y)| \cdot G_{1.5\rho'}(d)$$

Computing SIFT Keypoints: Basic Steps

Source image I

⇒

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

⇒

DOG pyramid

Step 1b

Buid DOG pyramid

⇒

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$

Location refinement

⇐

Step 1e

Prune set of extrema

Keypoints = { all remaining (x_i', y_i', ρ_i') }

Extremum pruning

⇐

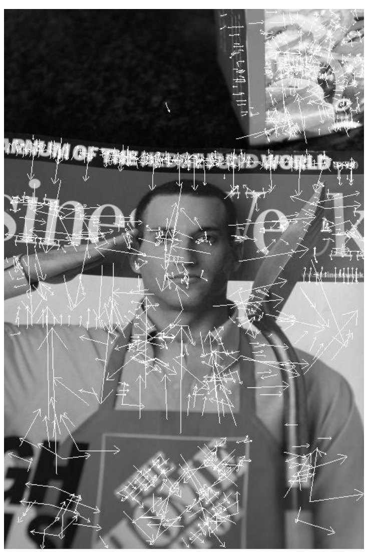
Step 1f

Assign orientation to eatrema

$P_i = (x_i', y_i', \rho_i', \theta_i)$

Orientation assign

=



Topic 10:

Feature Detection & Image Matching

- Introduction to the image matching problem
- Image matching using SIFT features
- The SIFT feature detector
- The SIFT descriptor

The SIFT Keypoints

We defined key-points that are “visually distinct” from its surroundings.

Input:

Image I

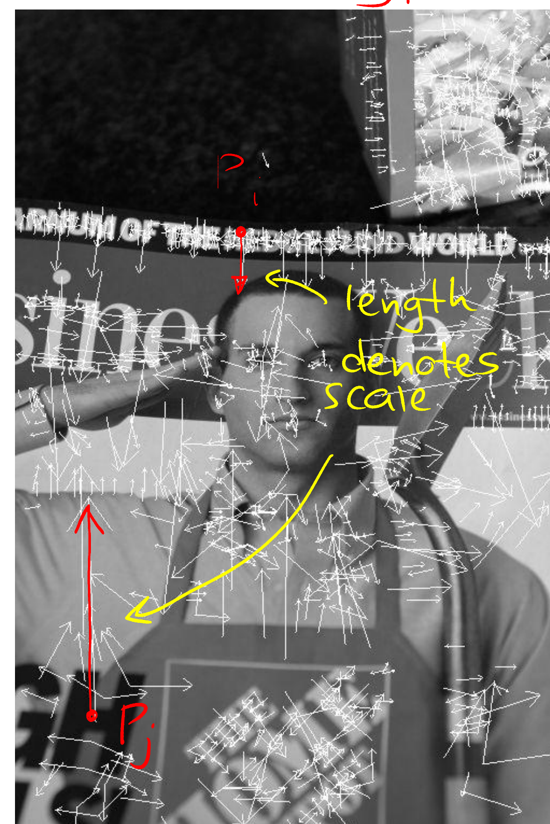
Output:

A set of k SIFT keypoints

$$P_i = (\underbrace{x_i, y_i}_{\text{location}}, \underbrace{\sigma_i}_{\text{scale}}, \underbrace{\theta_i}_{\text{orientation}})$$

“Source” Image I

Detected keypoints



The SIFT Feature Vectors

Feature vector (of a keypoint p_i): A vector of fixed length that represents the image patch centered at p_i

Input:

Image I

Output:

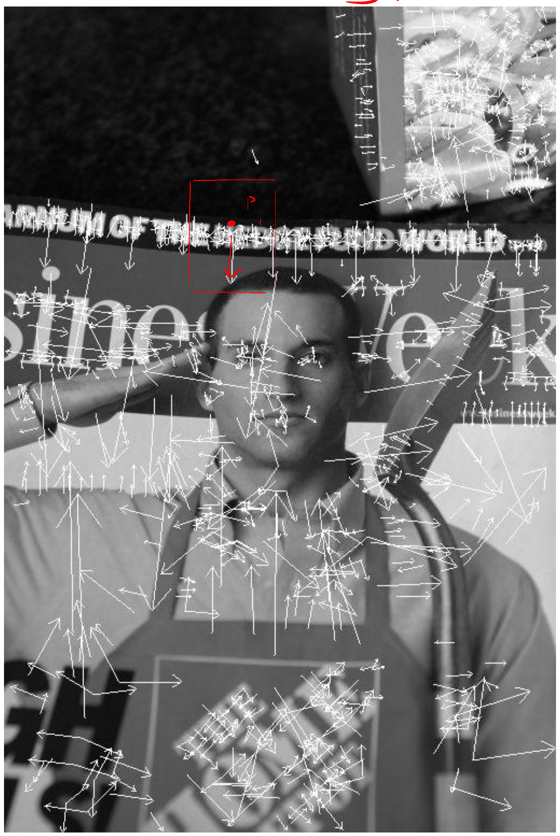
A set of K keypoints

A set of K feature vectors

$$p_i = (\underbrace{x_i, y_i}_{\text{location}}, \underbrace{\sigma_i}_{\text{scale}}, \underbrace{\theta_i}_{\text{orientation}})$$

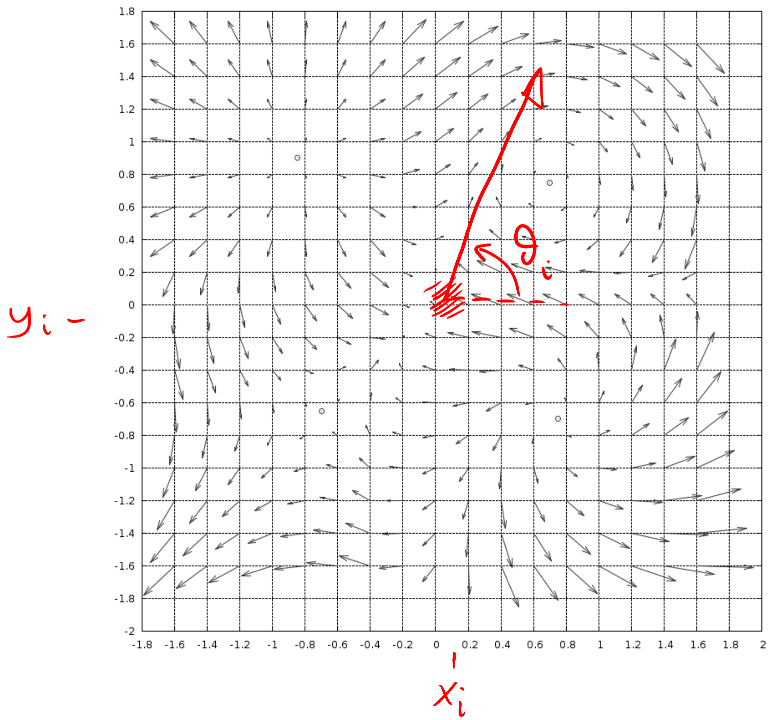
$$f_i = \underbrace{\boxed{}\boxed{}\boxed{\;\cdots\;}\boxed{}\boxed{}}_{\text{describes the patch centered at } p_i}$$

Detected keypoints

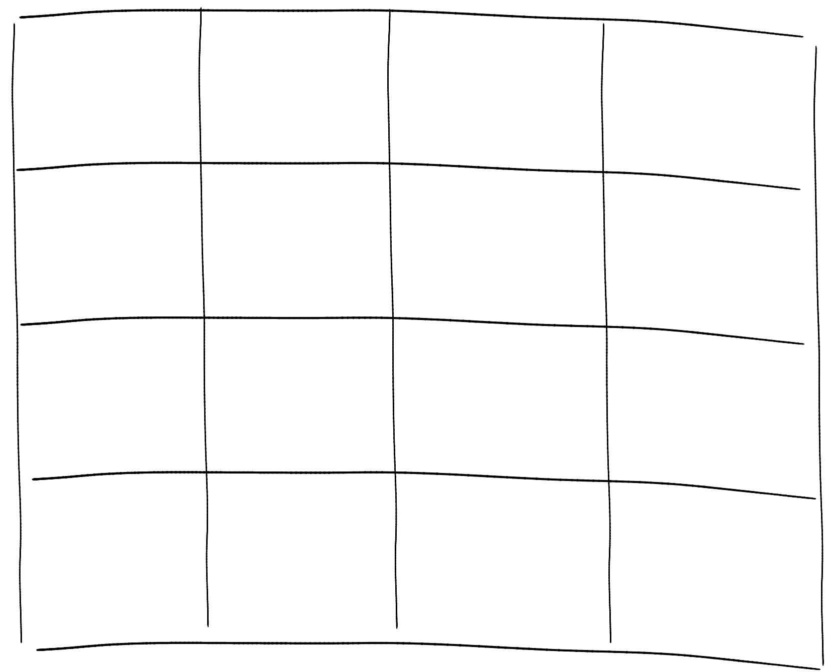


Building the SIFT Descriptor

Image patch centered at (x_i, y_i)



SIFT Descriptor

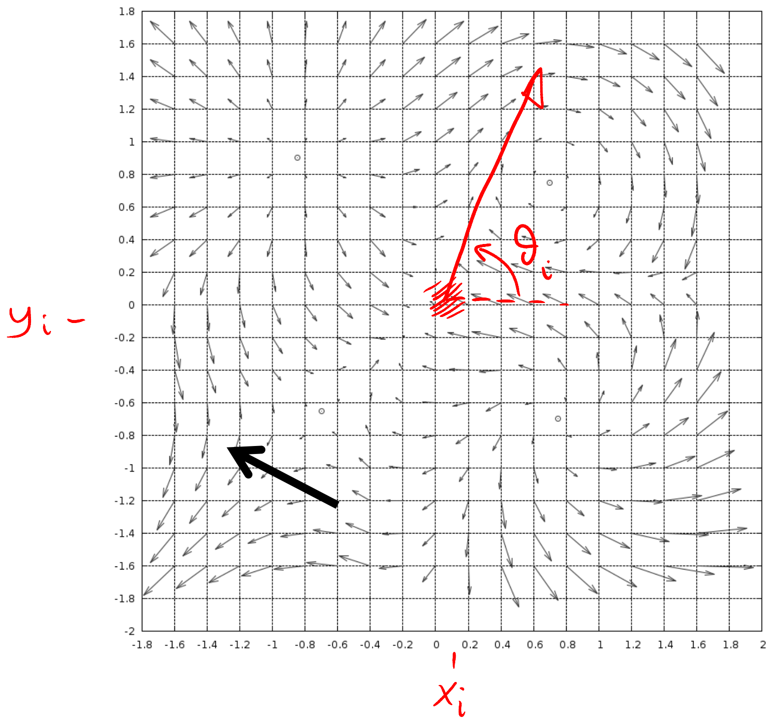


1. Compute gradients in 16×16 pixel patch of image $I * G_{\sigma_i}$ ← Gaussian-smoothed image at scale of the keypoint

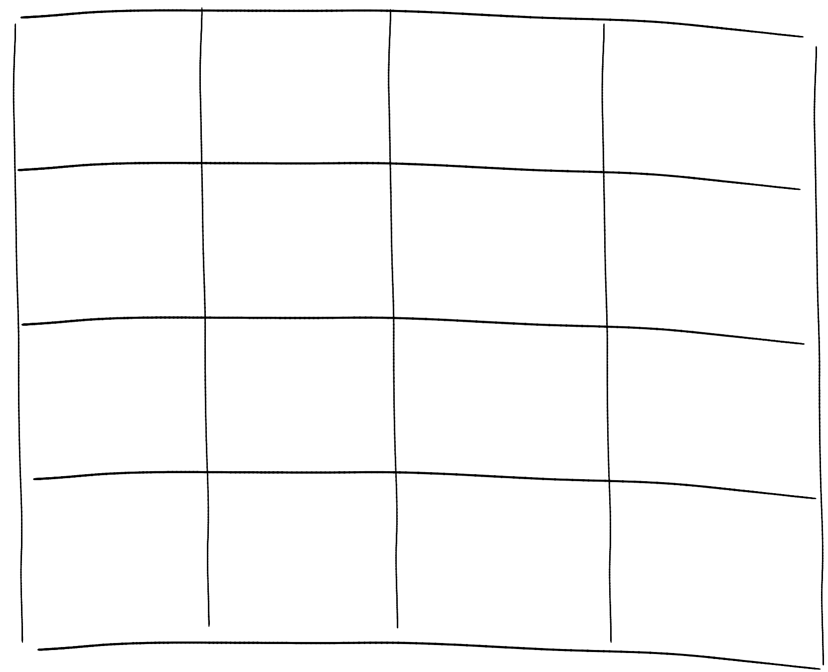
centered at (x_i, y_i)

Building the SIFT Descriptor

Image patch centered at (x_i, y_i)



SIFT Descriptor



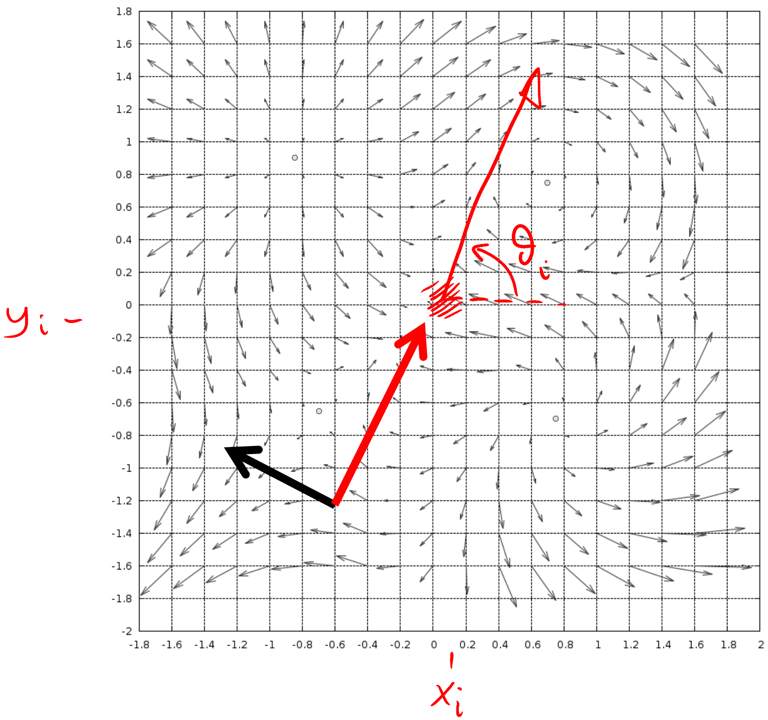
1. Compute gradients in 16×16 pixel patch of image $I * G_{\sigma_i}$ centered at (x_i, y_i)

2. Compute gradient orientation relative to keypoint orientation

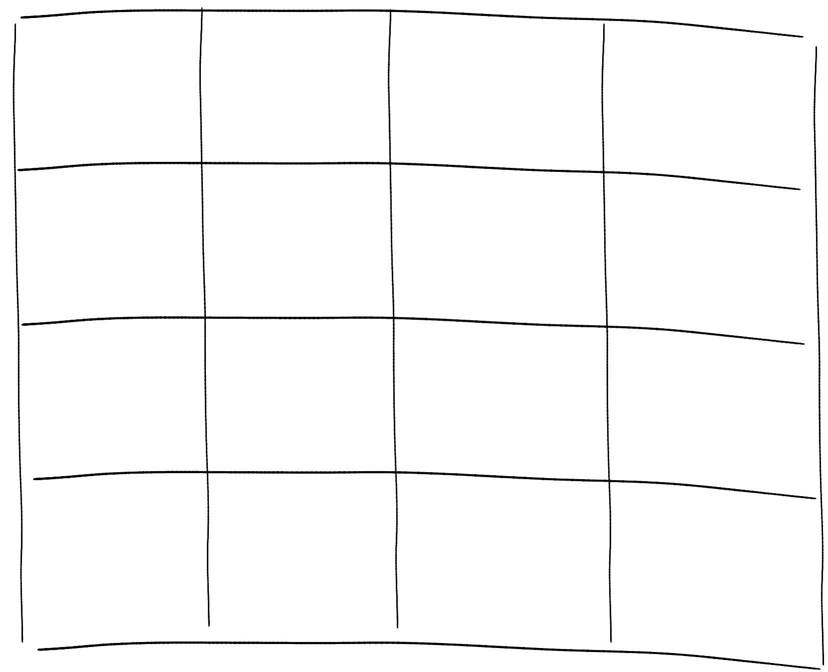
$$\boxed{\vartheta(x,y) = \tan^{-1}\left[\frac{\partial(I * G_{\sigma_i})}{\partial y} \Big/ \frac{\partial(I * G_{\sigma_i})}{\partial x}\right] - \vartheta_i}$$

Building the SIFT Descriptor

Image patch centered at (x_i, y_i)



SIFT Descriptor



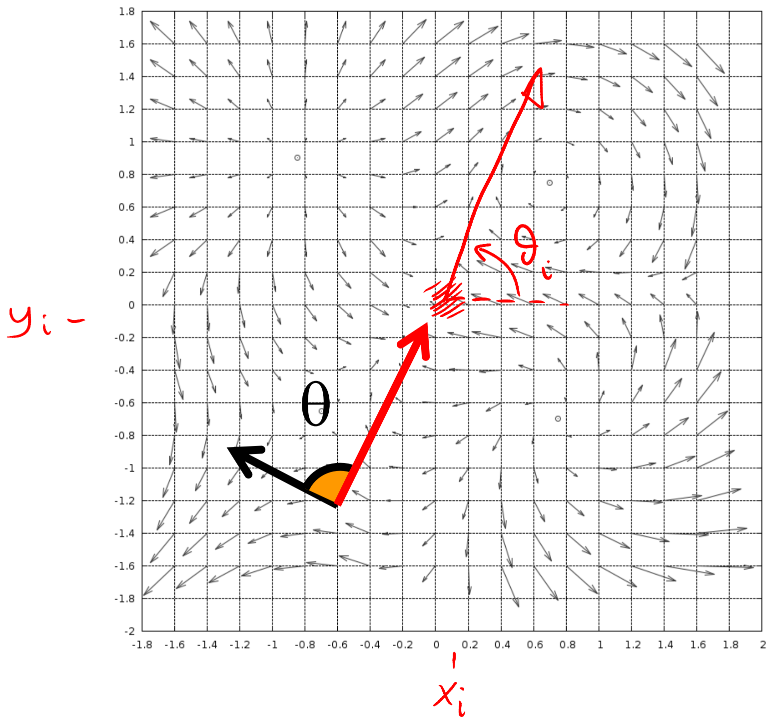
1. Compute gradients in 16×16 pixel patch of image $I * G_{\sigma_i}$ centered at (x_i, y_i)

2. Compute gradient orientation relative to keypoint orientation

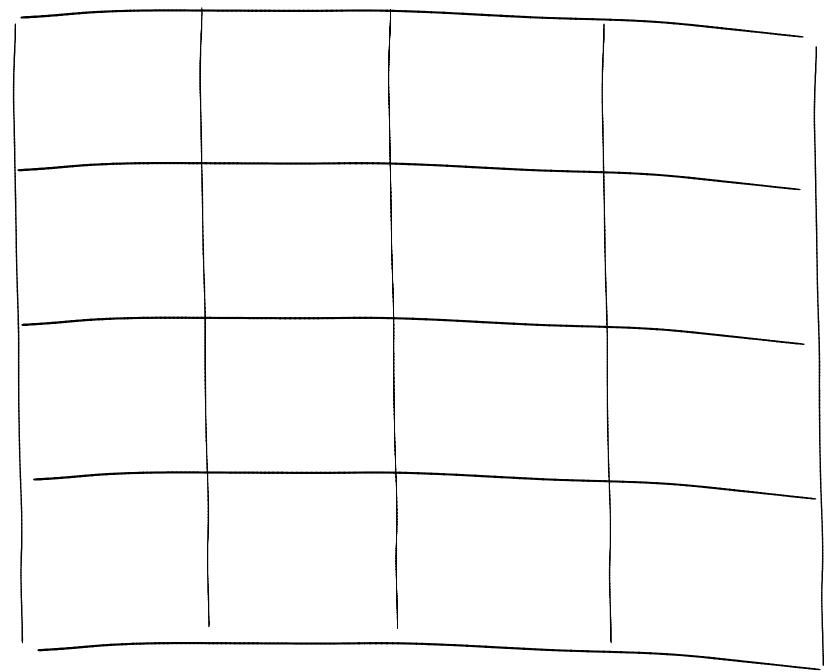
$$\boxed{\vartheta(x,y) = \tan^{-1}\left[\frac{\partial(I * G_{\sigma_i})}{\partial y} \Big/ \frac{\partial(I * G_{\sigma_i})}{\partial x}\right] - \vartheta_i}$$

Building the SIFT Descriptor

Image patch centered at (x_i, y_i)



SIFT Descriptor



1. Compute gradients in 16×16 pixel patch of image $I * G_{\sigma_i}$ centered at (x_i, y_i)

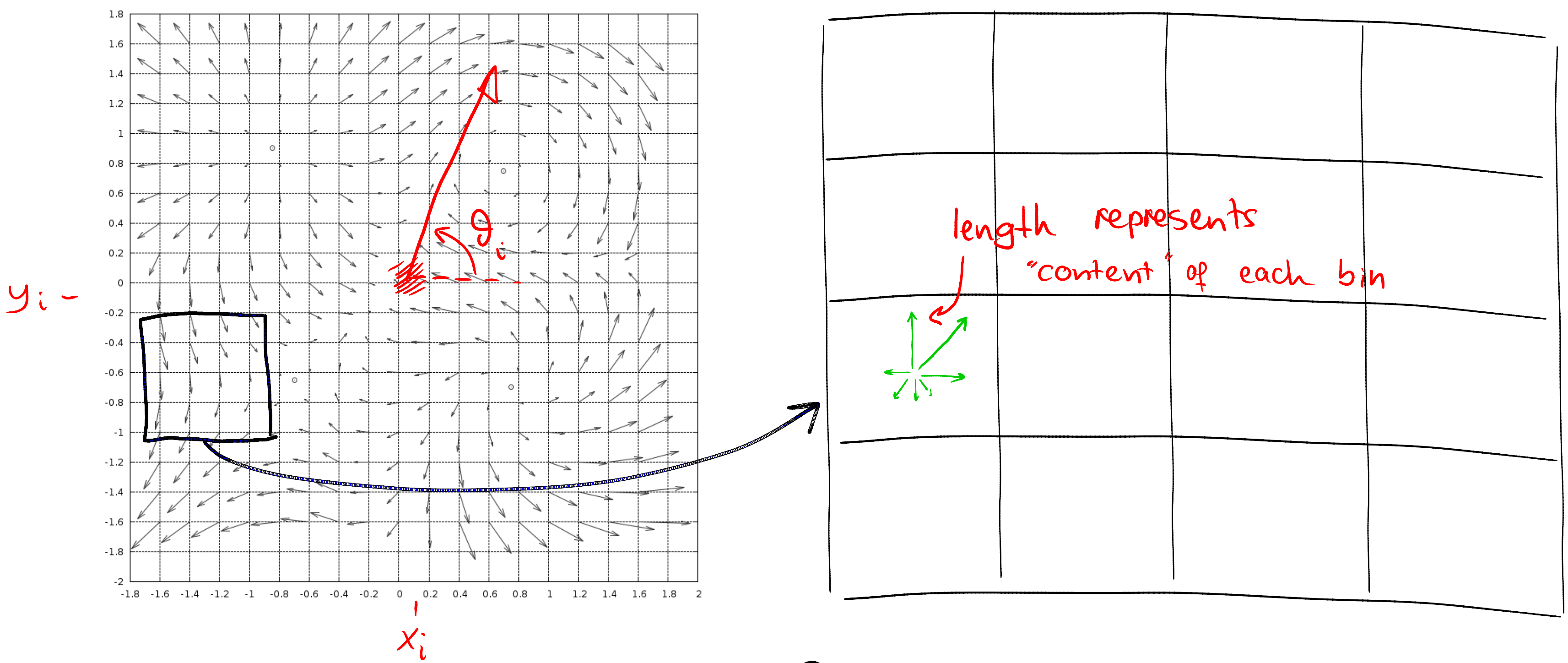
2. Compute gradient orientation relative to keypoint orientation

$$\boxed{\vartheta(x,y) = \tan^{-1}\left[\frac{\partial(I * G_{\sigma_i})}{\partial y} \Big/ \frac{\partial(I * G_{\sigma_i})}{\partial x}\right] - \vartheta_i}$$

Building the SIFT Descriptor

Image patch centered at (x_i, y_i)

SIFT Descriptor



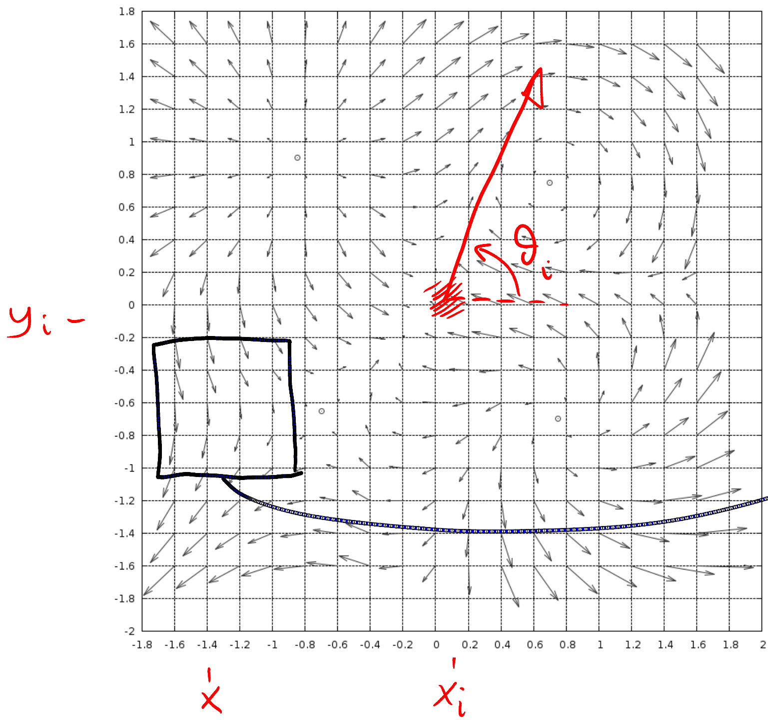
1. Compute gradients
2. Compute relative gradient orientations
3. Compute orientation histogram of each 4x4 pixel block

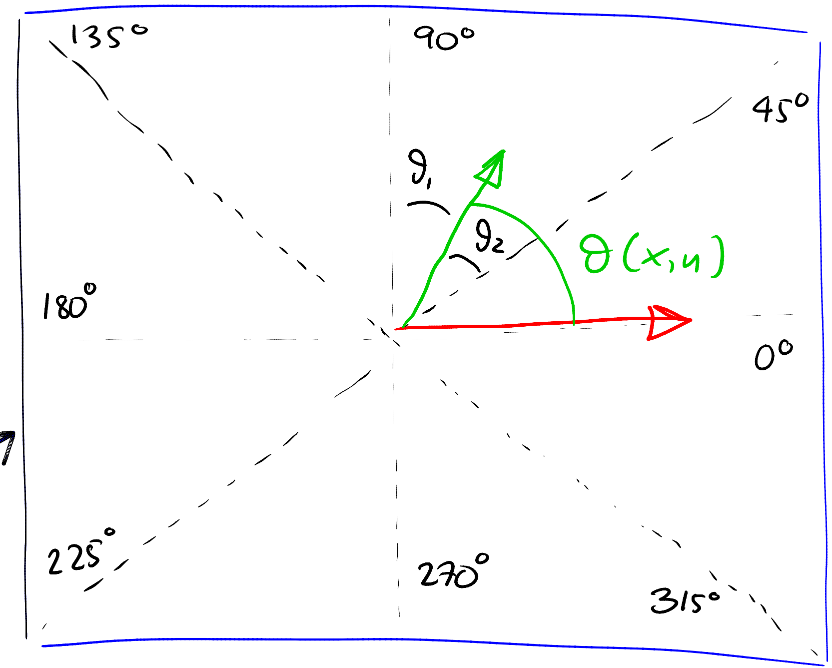
histogram contains 8 bins, each covering 45°

The 4x4 Orientation Histogram

Image patch centered at (x_i, y_i)

The Orientation Histogram of a 4x4 pixel block





Weight of (x,y):

$$w = G_{\frac{\sigma_i}{2}}\left(\|(x-x_i, y-y_i)\|\right)$$

⇒ pixels closer to keypoint center have higher weight

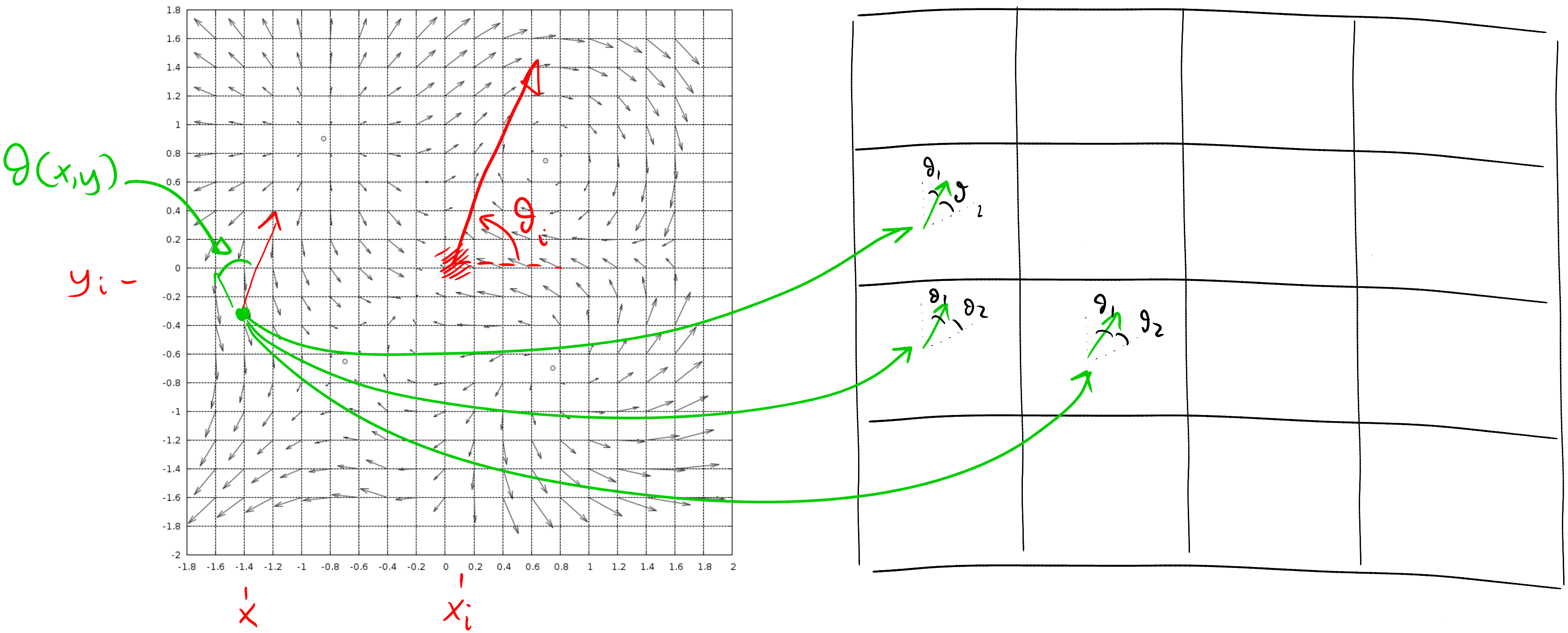
Total contribution of (x,y) to the orientation histograms:

$$c(x,y) = w \cdot \underbrace{\| \nabla I * G_{\sigma_i}(x,y) \|}_{\text{gradient magnitude of smoothed image}}$$

Building the SIFT Descriptor

Image patch centered at (x_i, y_i)

SIFT Descriptor



- Contribution spread across 2 closest orientations & 3 closest histograms

- Total contribution of (x,y) to the orientation histograms:

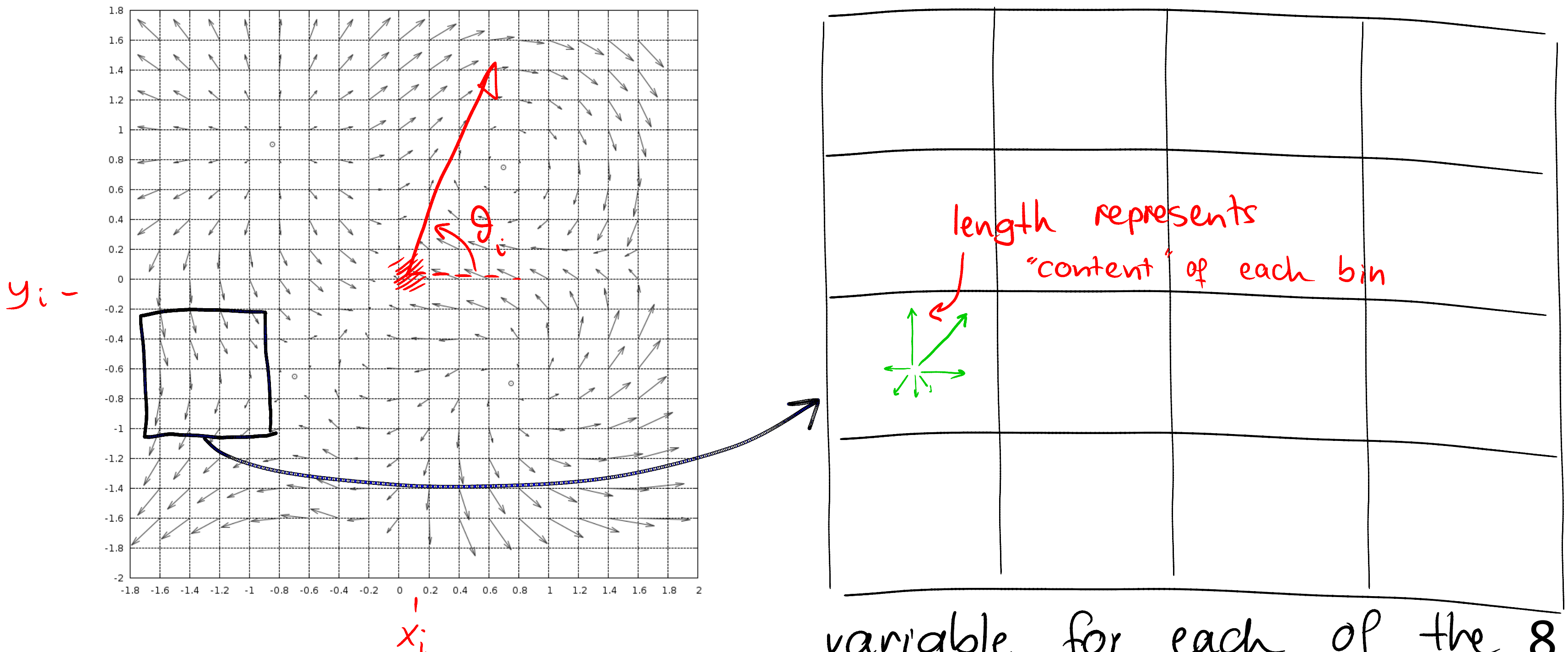
$$c(x,y) = w \cdot \| \nabla I * G_{\sigma_i}(x,y) \|$$

⇒ no abrupt changes in histogram if keypoint center displaced by 3-4 pixels

Building the SIFT Descriptor: Complete Algorithm

Image patch centered at (x_i, y_i)

SIFT Descriptor



1. Compute gradients
2. Compute relative gradient orientations
3. Define an "accumulator" variable for each of the 8 orientations in each of the 16 histograms (128 total)
4. For each pixel, calculate the pixel's contribution to each accumulator variable

Converting SIFT Descriptors to 128-dim Vectors

Post processing:

① Normalize f_i:

$$f_i \rightarrow \frac{f_i}{\|f_i\|}$$

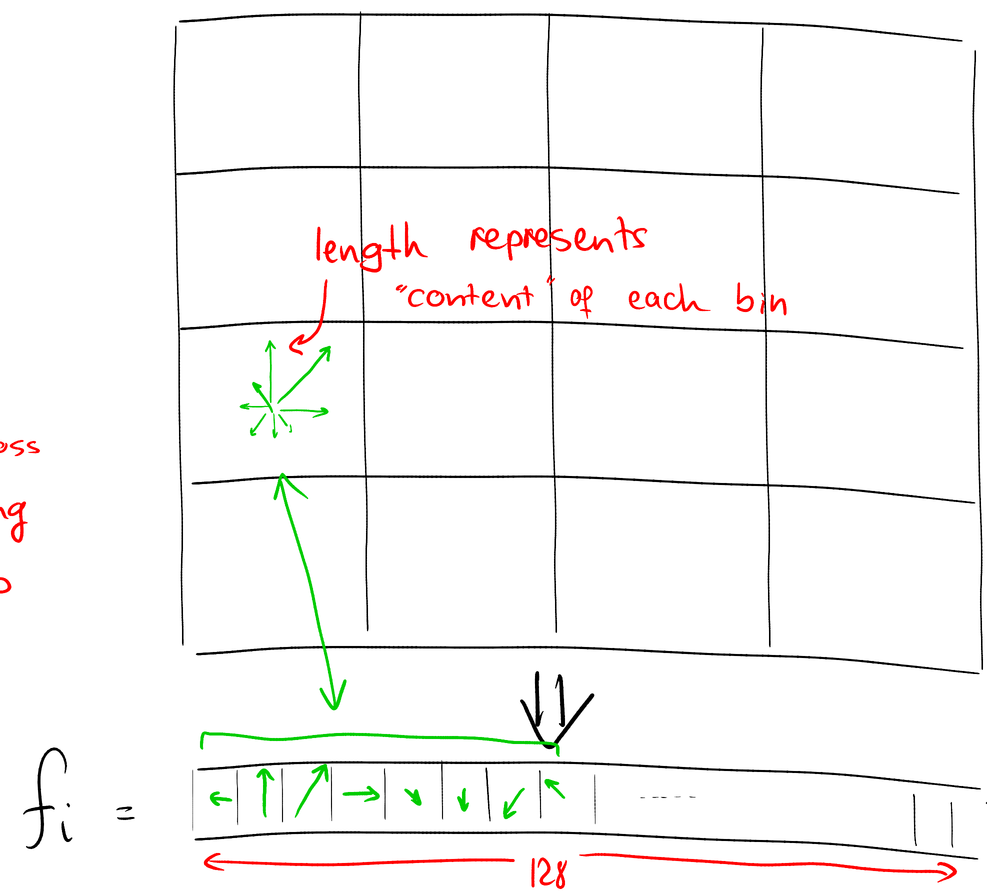
⇒ gives invariance to linear lighting variations across images when matching images, i.e. image I and image $a \cdot I + b$ (because f_i will be the same in both images)

② Clamp f_i

Clamp all elements of f_i at 0.2

⇒ gives less weight to very large gradient magnitudes

SIFT Descriptor

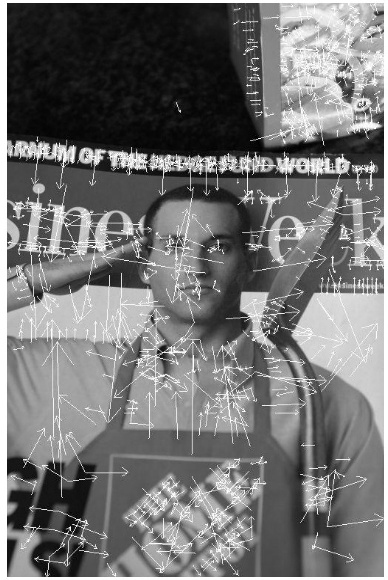


③ Re-normalize

Matching 2 Images Using SIFT Features

Image I

⇒



⇒

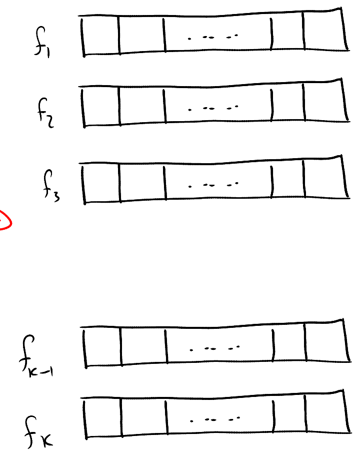
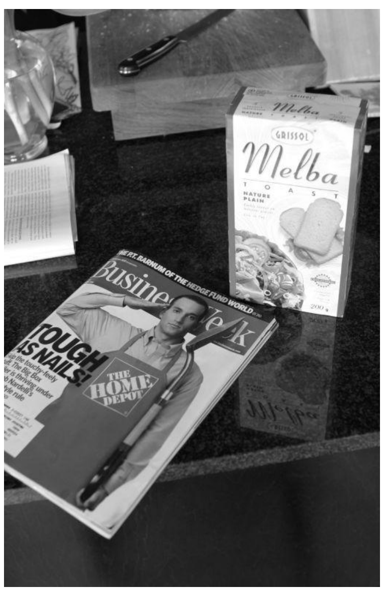
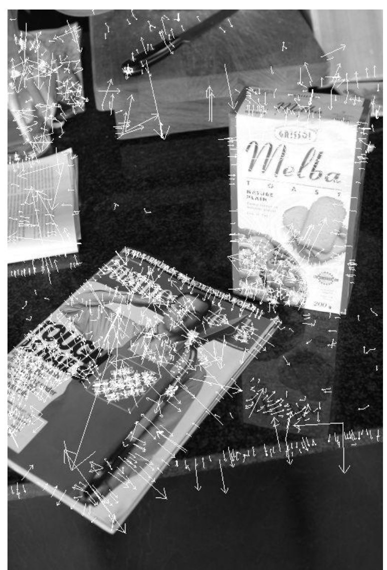


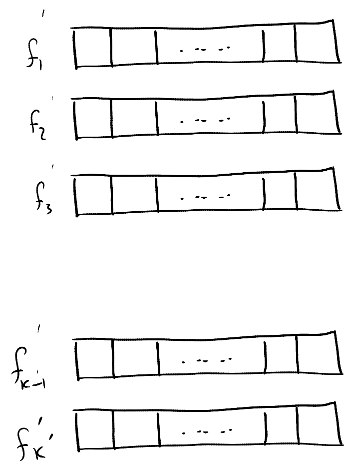
Image I'



⇒



⇒



① Identify keypoints

② Build feature vectors

③ Match feature vectors in the two sets

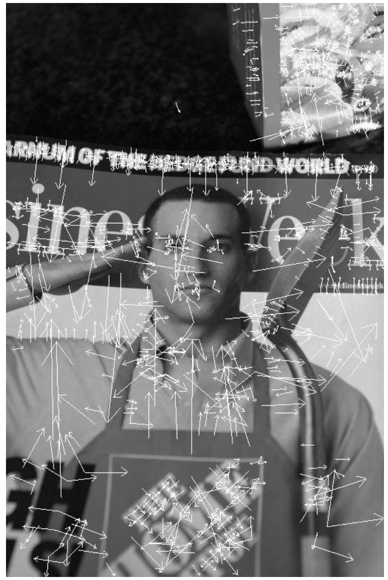
$\{f_1, f_2, \ldots, f_K\}$

and

$\{f'_1, f'_2, \ldots, f'_{K'}\}$

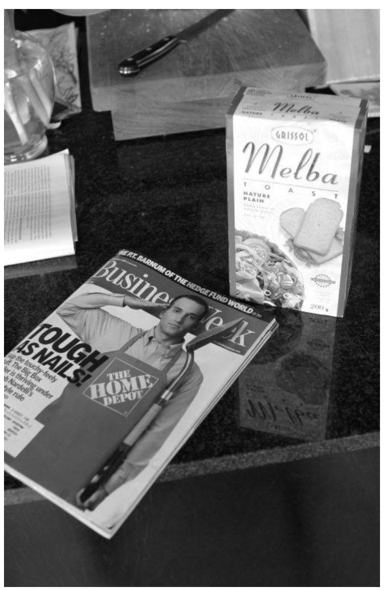
SIFT Feature Matching Algorithm

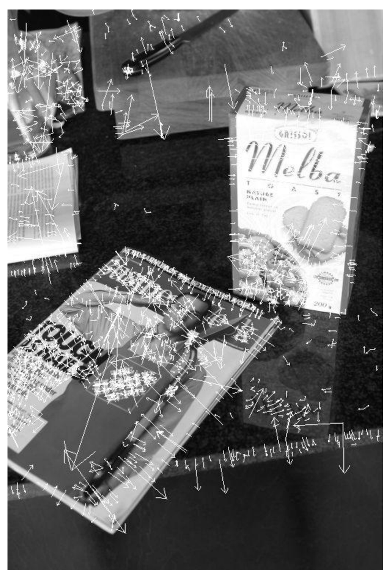
Image I



f_1 | | | - - - | |
f_2 | | | - - - | |
f_i | | | - - - | |
f_{k-1} | | | - - - | |
f_k | | | - - - | |

Image I'





f'_1 | | | - - - | |
$f'_{j^{**}}$ | | | - - - | |
f'_3 | | | - - - | |
f'_{j^*} | | | - - - | |
$f'_{k'}$ | | | - - - | |

① Identify keypoints

② Build feature vectors

③ Match f_i

a. Compute $\|f_i - f'_j\|$ for all j

b. Compute fraction

$$\phi = \frac{\|f_i - f'_{j^*}\|}{\|f_i - f'_{j^{**}}\|}$$

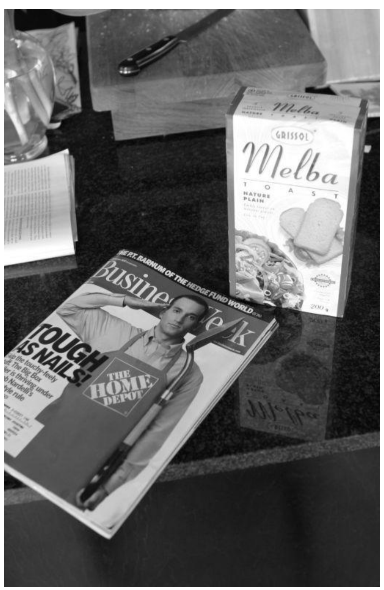
where f'_{j^*} is closest descriptor in I' and $f'_{j^{**}}$ is 2nd-closest

c. Match f_i to f'_{j^*} if $\phi < 0.8$

Matching 2 Images Using SIFT Features

Image I

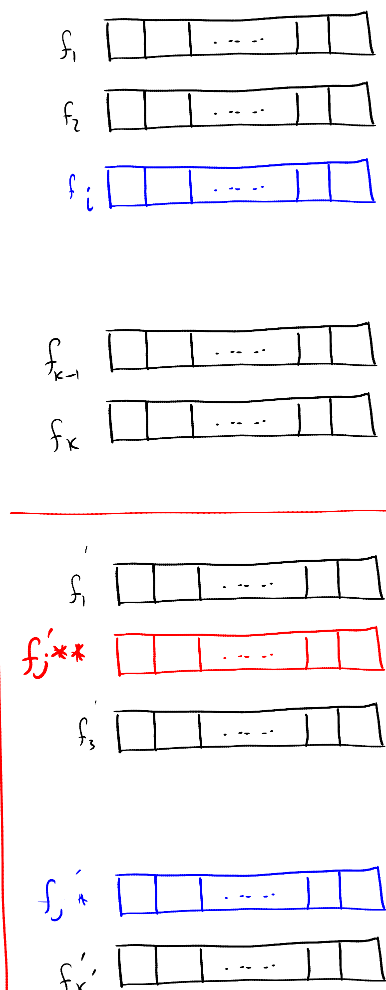
Image I′



① Identify keypoints

Intuition for matching algorithm:

match established only if it is deemed reliable, i.e. if there is only one very similar feature in image I′



② Build feature vectors

③ Match f_i

a. Compute $\|f_i - f_j'\|$ for all j

b. Compute fraction

$$\phi = \frac{\|f_i - f_j'^{*}\|}{\|f_i - f_j'^{**}\|}$$

where $f_j'^{*}$ is closest descriptor in I′ and $f_j'^{**}$ is 2nd-closest

c. Match f_i to $f_j'^{*}$ if $\phi < 0.8$

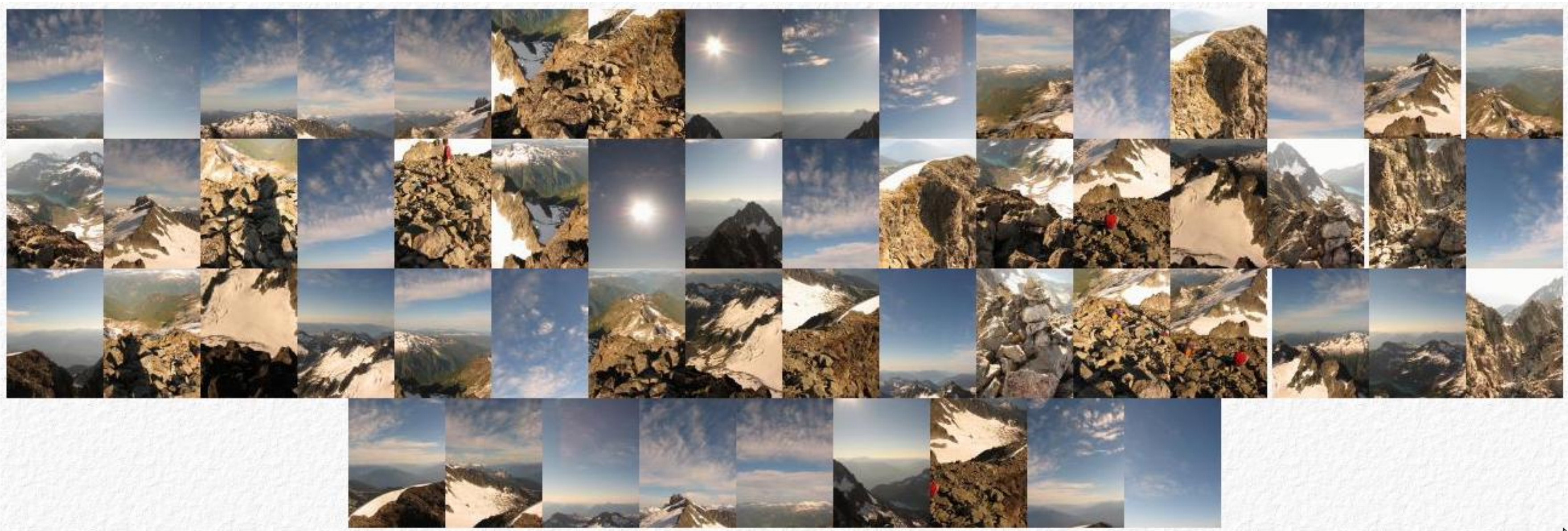
Topic 11:

Homographies & Image Mosaics

- Introduction to image mosaicing
- Homogeneous coordinates for points & lines
- Image homographies
- Estimating homographies from point correspondences
- The autostitch algorithm

Building Panoramic Image Mosaics

Input images

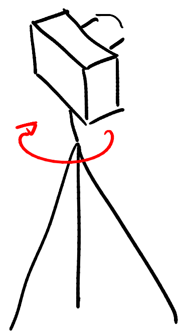


⇓ automatically created mosaic

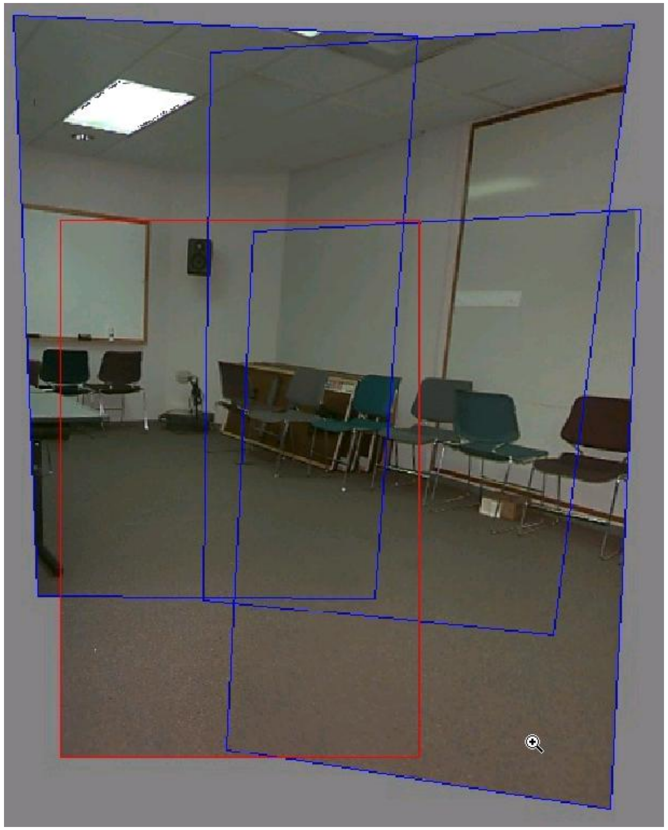
Image Mosaicing

Technique:

1. Take multiple photos while rotating camera on a tripod (or by hand)

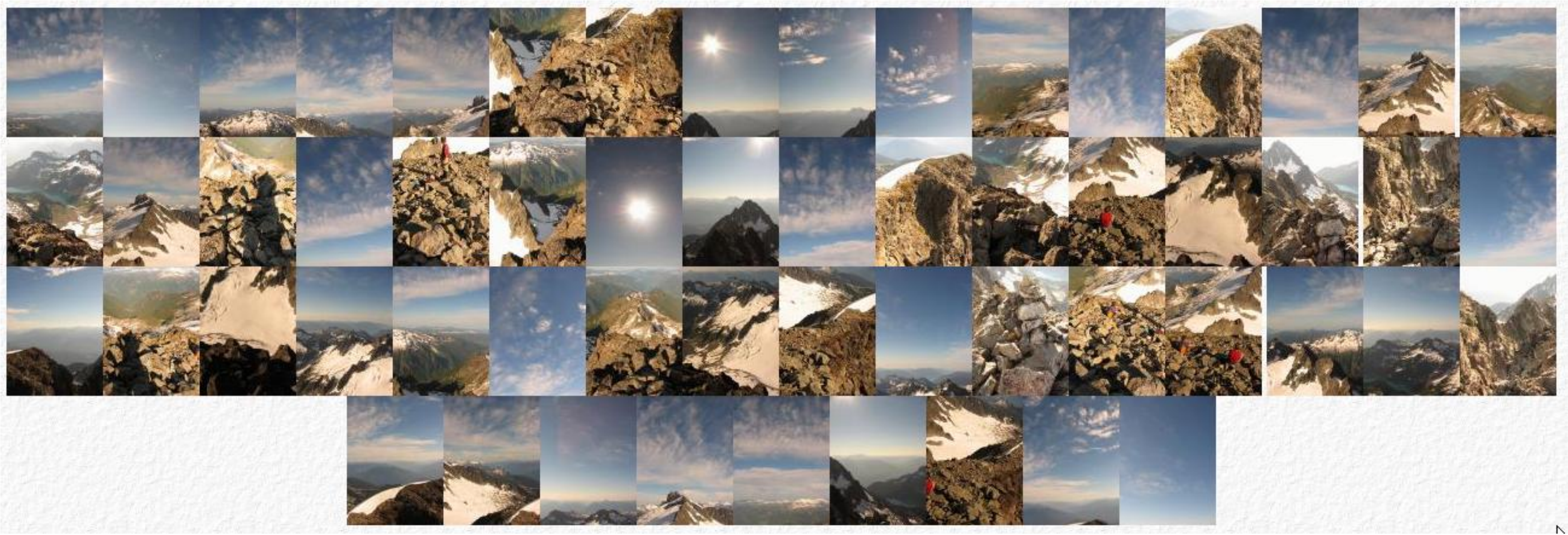


2. Warp & align the photos
3. Blend photos to compute final mosaic



* In general, photos must be <u>warped</u> to align their contents!

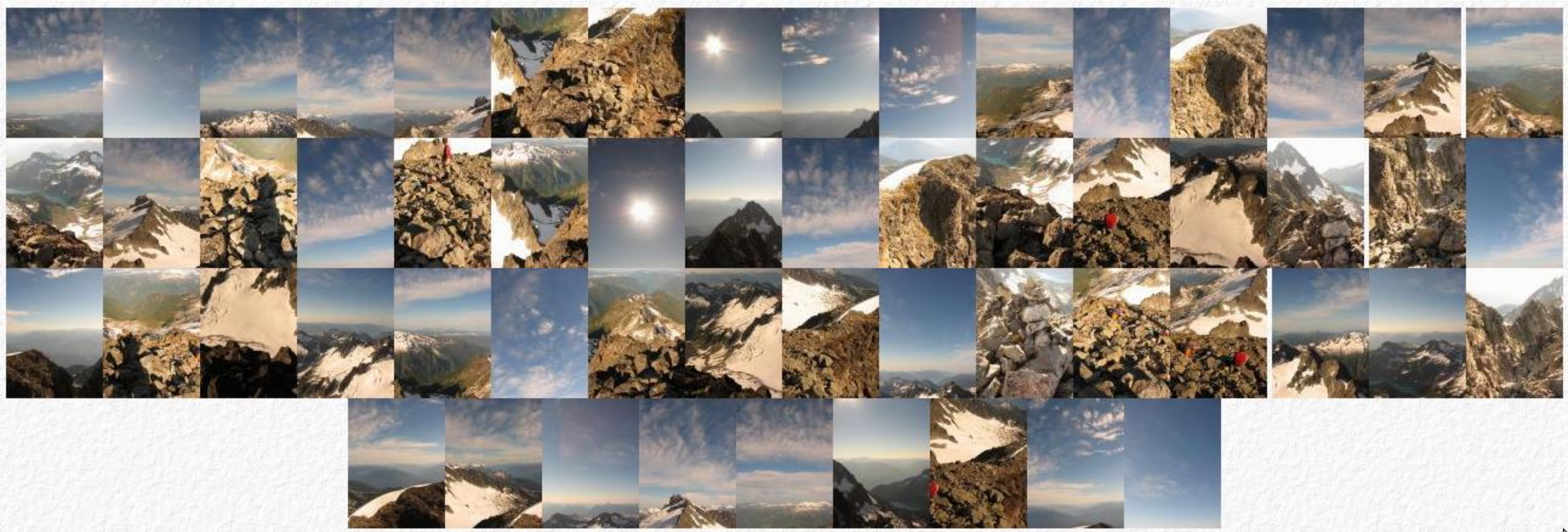
Step 1: Capture



Important:

- Camera should change orientation, not position
- Keep camera settings (gain, focus, speed, aperture) fixed if possible

Step 2: Warp & Align



⇓ 28/57 images aligned

Step 2: Warp & Align (Continued)

⇑ 57 / 57 images aligned

Step 3: Blend

Laplacian Pyramid Blending ⇓ seams not visible anymore

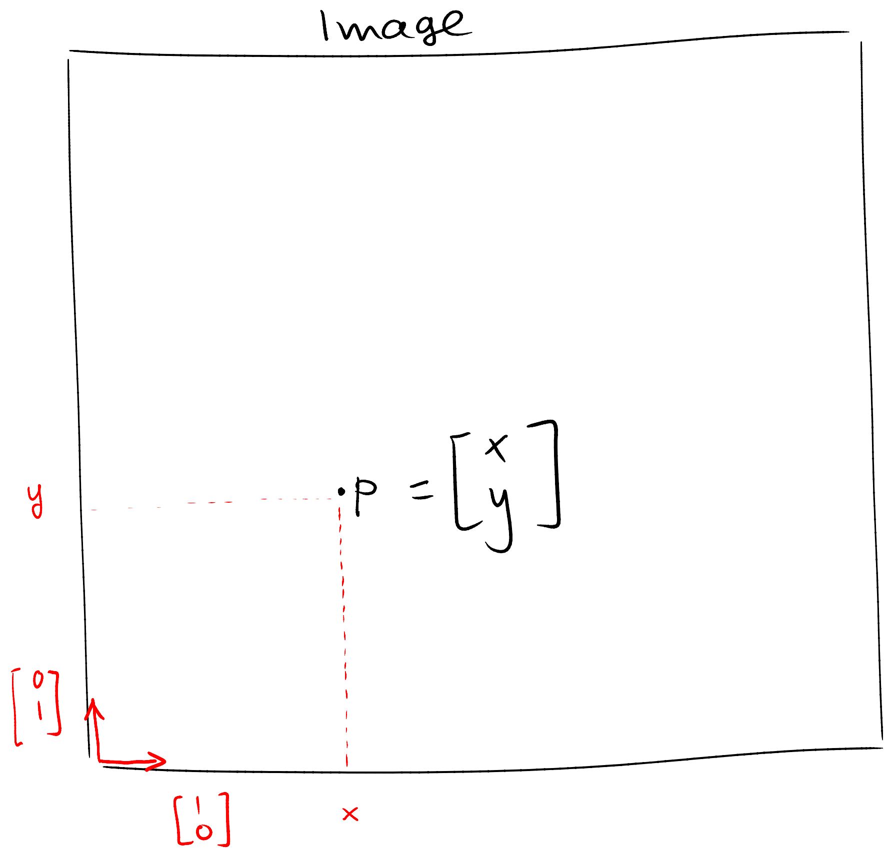
(Brown & Lowe ; ICCV 2003) google " Lowe Brown Autostitch "

Topic 11:

Homographies & Image Mosaics

- Introduction to image mosaicing
- Homogeneous coordinates for points & lines
- Image homographies
- Estimating homographies from point correspondences
- The autostitch algorithm

Representing Pixels by Euclidean 2D Coordinates



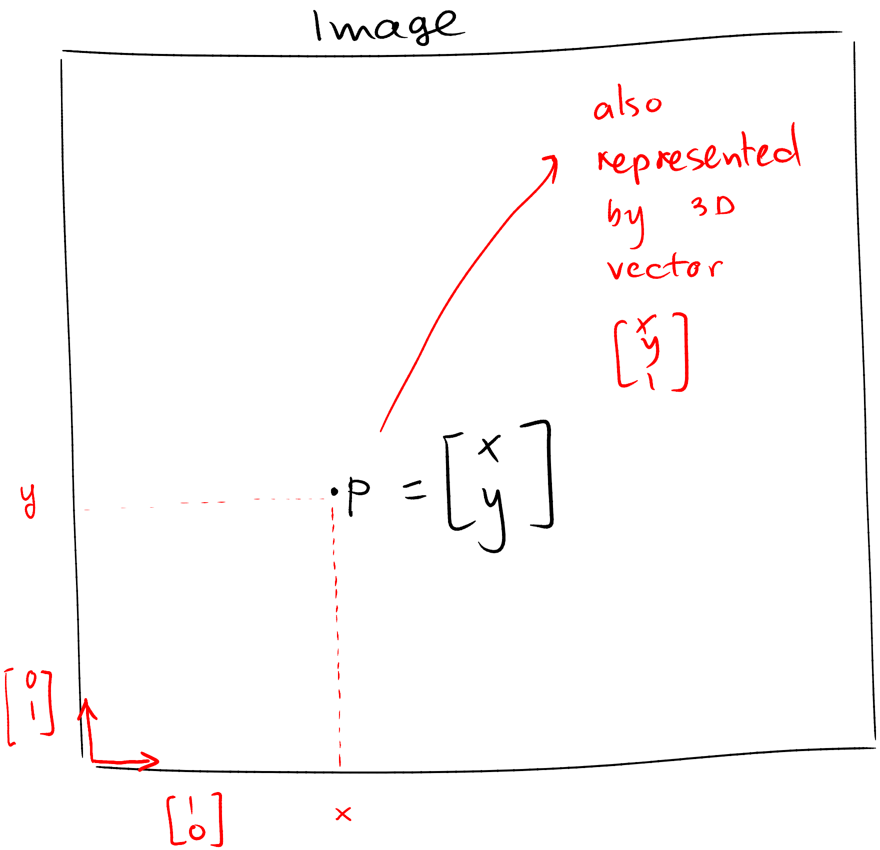
The "standard" (or Euclidean) representation of an image point is p is:

$$p = x \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

basis vectors

Euclidean coordinates

Euclidean Coordinates $\Rightarrow$ Homogeneous Coordinates



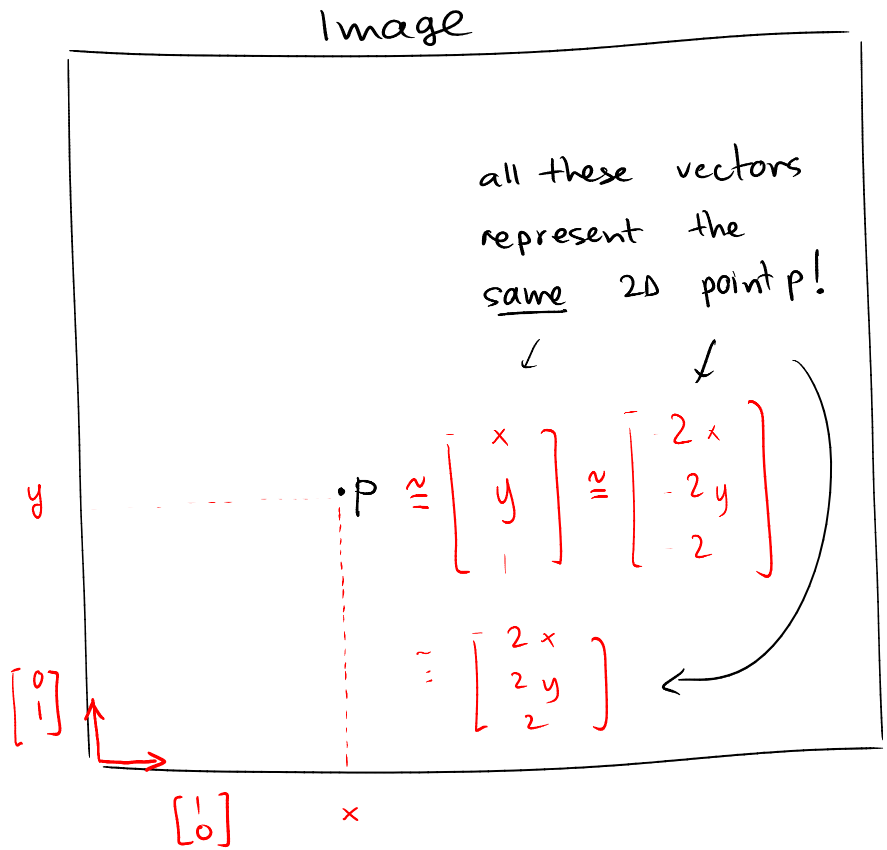
The “standard” (or Euclidean) representation of an image point is p is:

$$p = x \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

The Homogeneous (or Projective) representation of the same point is:

image coordinates $\begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ homogeneous 2D coordinates

2D Homogeneous Coordinates: Definition

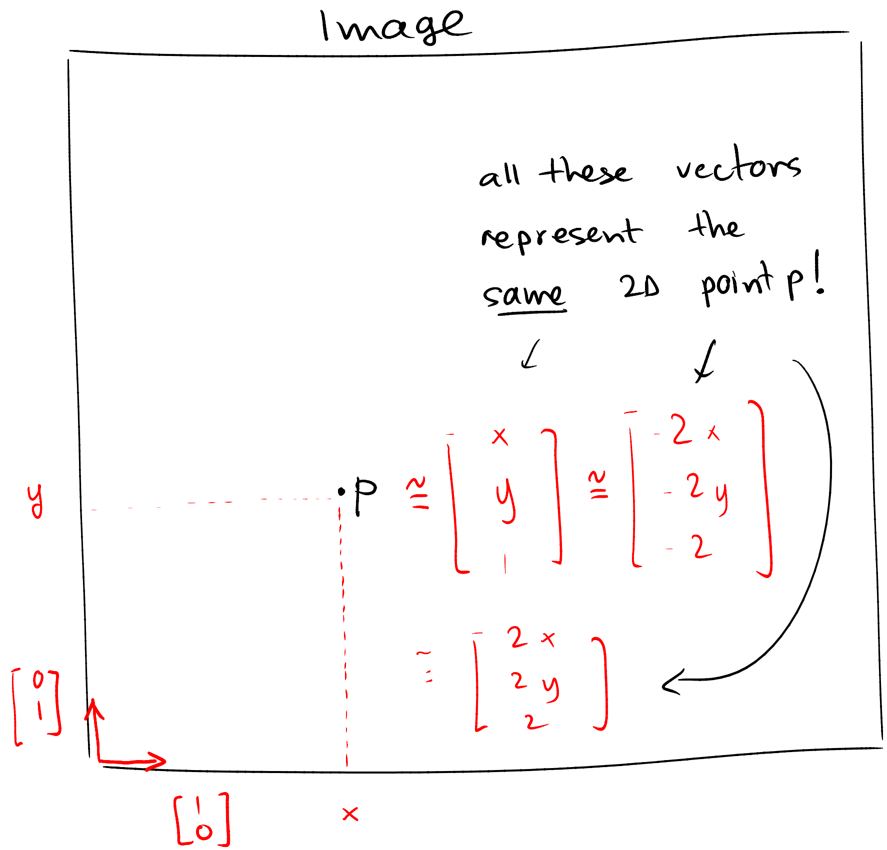


Definition:

Homogeneous representation of p

$\boxed{\text{p represented by any 3D vector } \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} \text{ with } \lambda \neq 0}$

2D Homogeneous Coordinates: Definition



Definition:

Homogeneous representation of p

$$\boxed{\text{p represented by any 3D vector } \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} \text{ with } \lambda \neq 0}$$

Homogeneous coordinates

2D Homogeneous Coordinates: Definition

Note that the transformation is simply:

image coordinates

homogeneous 2D coordinates

$$\begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow \begin{bmatrix} \lambda \cdot x \\ \lambda \cdot y \\ \lambda \cdot 1 \end{bmatrix} \qquad \lambda \neq 0$$

2D Homogeneous Coordinates: Equality

Definition (Homogeneous Equality)

Two vectors of homogeneous coords $v_1 = \begin{bmatrix} x \\ y \\ w \end{bmatrix}$ and $v_2 = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix}$ are called equal if they represent the same 2D point:

$v_1 \cong v_2$ denotes homog. equality $\iff$ there is a $\lambda \neq 0$ such that

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \lambda \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix}$$

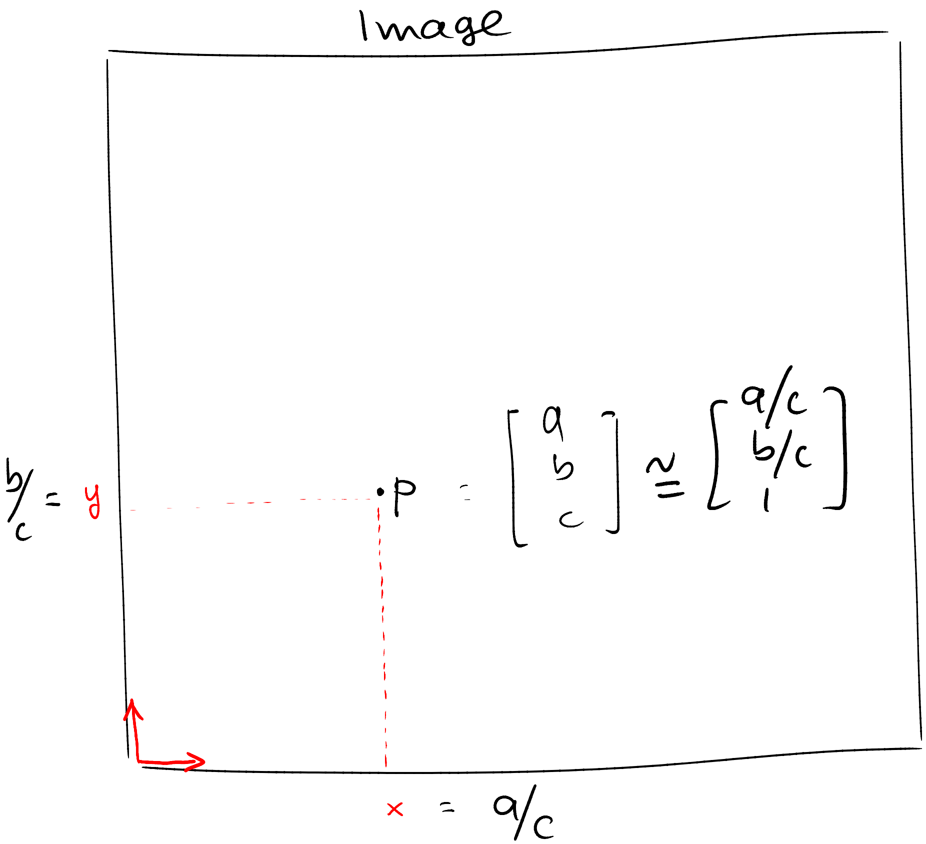
Examples:

Is $\begin{bmatrix} 3 \\ 4 \\ 6 \end{bmatrix} \cong \begin{bmatrix} 6 \\ 8 \\ 12 \end{bmatrix}$? yes (take $\lambda = 2$)

Is $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cong \begin{bmatrix} 0 \\ 0 \\ 30 \end{bmatrix}$? yes (take $\lambda = 30$)

Is $\begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \cong \begin{bmatrix} -2 \\ 0 \\ 4 \end{bmatrix}$? no!

Homogeneous Coordinates ⇒ Euclidean Coordinates

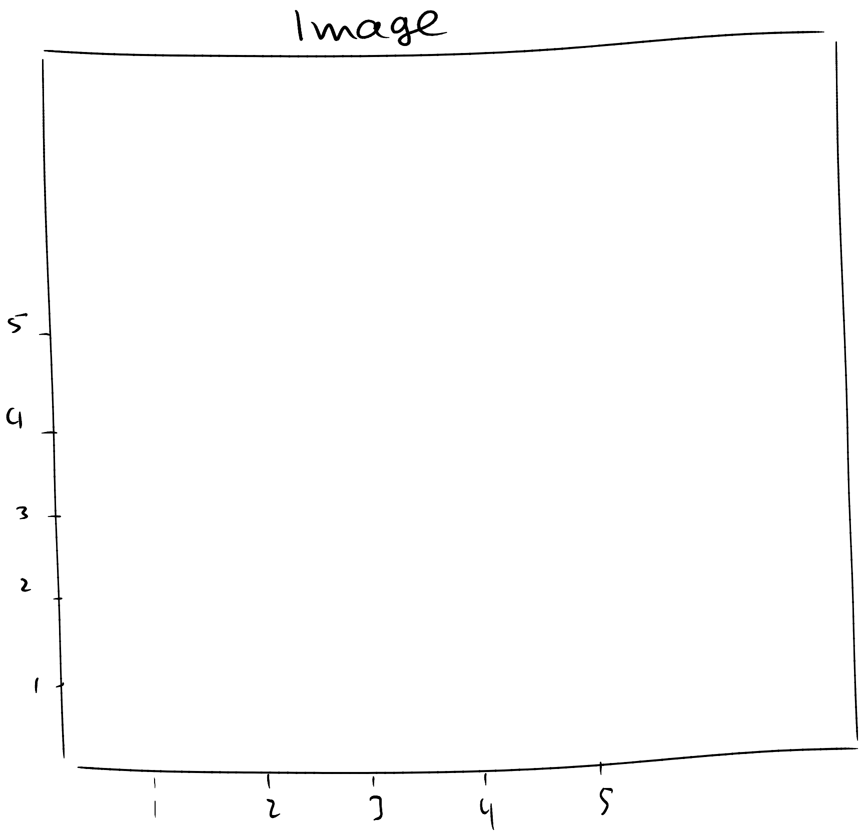


Converting from homogeneous to Euclidean coordinates:

$\begin{bmatrix} a \\ b \\ c \end{bmatrix}, \begin{bmatrix} a/c \\ b/c \\ 1 \end{bmatrix}$ represent the same 2D point

$\iff$ 2D coordinates are $\begin{bmatrix} a/c \\ b/c \end{bmatrix}$

Homogeneous Coordinates ⇒ Euclidean Coordinates



Converting from homogeneous Euclidean coordinates:

$\begin{bmatrix} a \\ b \\ c \end{bmatrix}$, $\begin{bmatrix} a/c \\ b/c \\ 1 \end{bmatrix}$ represent the same 2D point

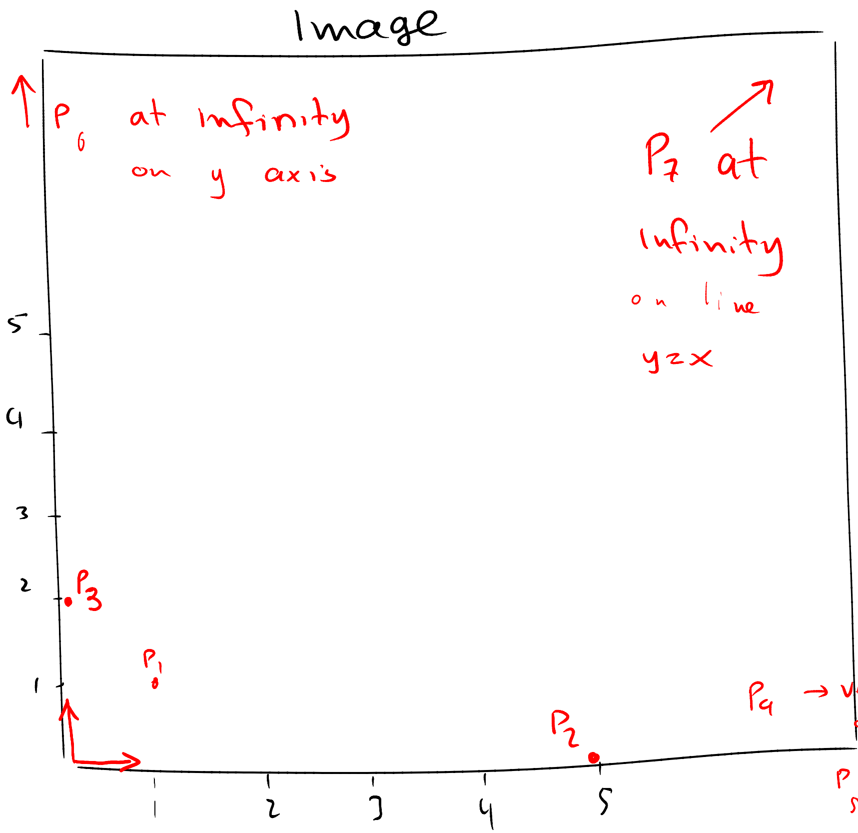
$\Leftrightarrow$ 2D coordinates are $\begin{bmatrix} a/c \\ b/c \end{bmatrix}$

Practice exercise: Plot positions of the following points

$P_6 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ $P_7 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$

$P_1 = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$ $P_2 = \begin{bmatrix} 10 \\ 0 \\ 2 \end{bmatrix}$ $P_3 = \begin{bmatrix} 0 \\ 8 \\ 4 \end{bmatrix}$ $P_4 = \begin{bmatrix} 1 \\ 0 \\ 0.0001 \end{bmatrix}$ $P_5 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

Homogeneous Coordinates ⇒ Euclidean Coordinates



Converting from homogeneous to Euclidean coordinates:

$\begin{bmatrix} a \\ b \\ c \end{bmatrix}$, $\begin{bmatrix} a/c \\ b/c \\ 1 \end{bmatrix}$ represent the same 2D point

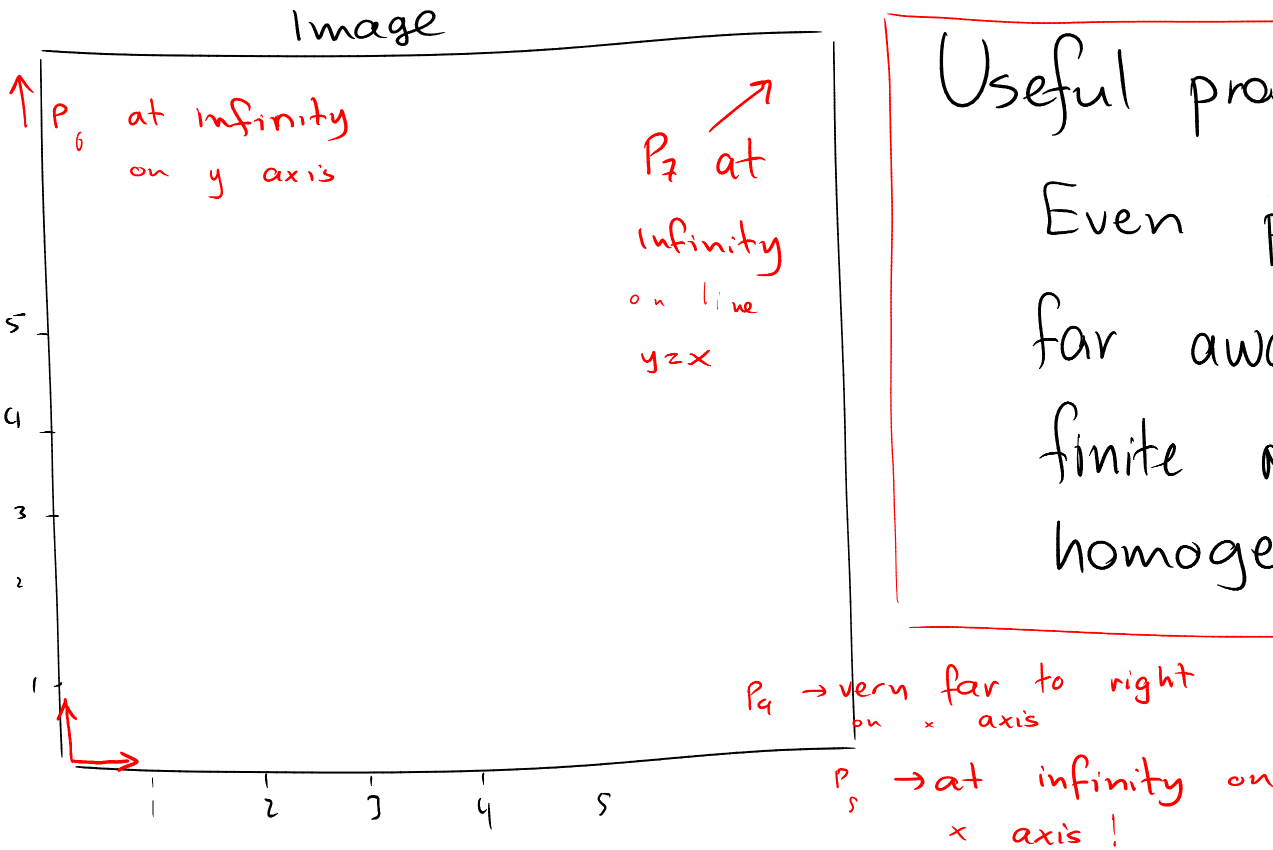
$\Longleftrightarrow$ 2D coordinates are $\begin{bmatrix} a/c \\ b/c \end{bmatrix}$

Practice exercise: Plot positions of the following points

$P_6 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ $P_7 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$

$P_1 = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$ $P_2 = \begin{bmatrix} 10 \\ 0 \\ 2 \end{bmatrix}$ $P_3 = \begin{bmatrix} 0 \\ 8 \\ 4 \end{bmatrix}$ $P_4 = \begin{bmatrix} 1 \\ 0 \\ 0.0001 \end{bmatrix}$ $P_5 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

Points at ∞ in Homogeneous Coordinates



Useful property # 1:

Even points infinitely far away have a finite representation in homogeneous coords!

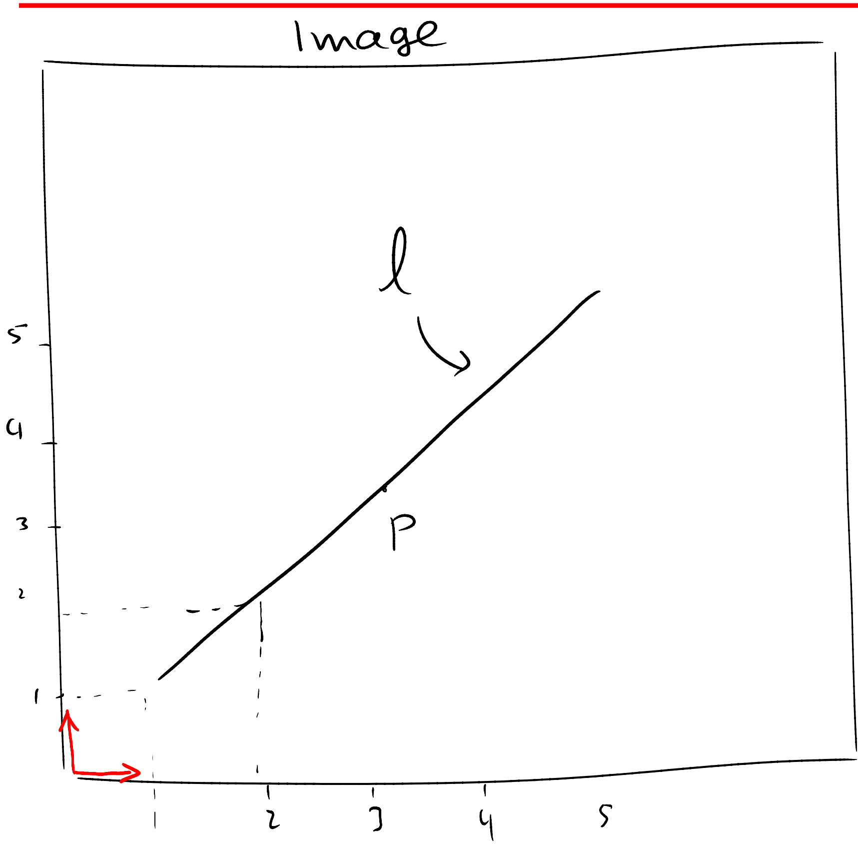
leads to very <u>stable</u> geometric computations

Points at infinity have their last coordinate equal to zero

$$P_6 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad P_7 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

$$P_4 = \begin{bmatrix} 1 \\ 0 \\ 0.0001 \end{bmatrix} \quad P_5 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Line Equations in Homogeneous Coordinates



- The equation of a line

$$ax + by + c = 0$$

line parameters

- In homogeneous coordinates

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

or $\ell^T \cdot p = 0$

vector holding line parameters

vector holding homogeneous coordinates of a point

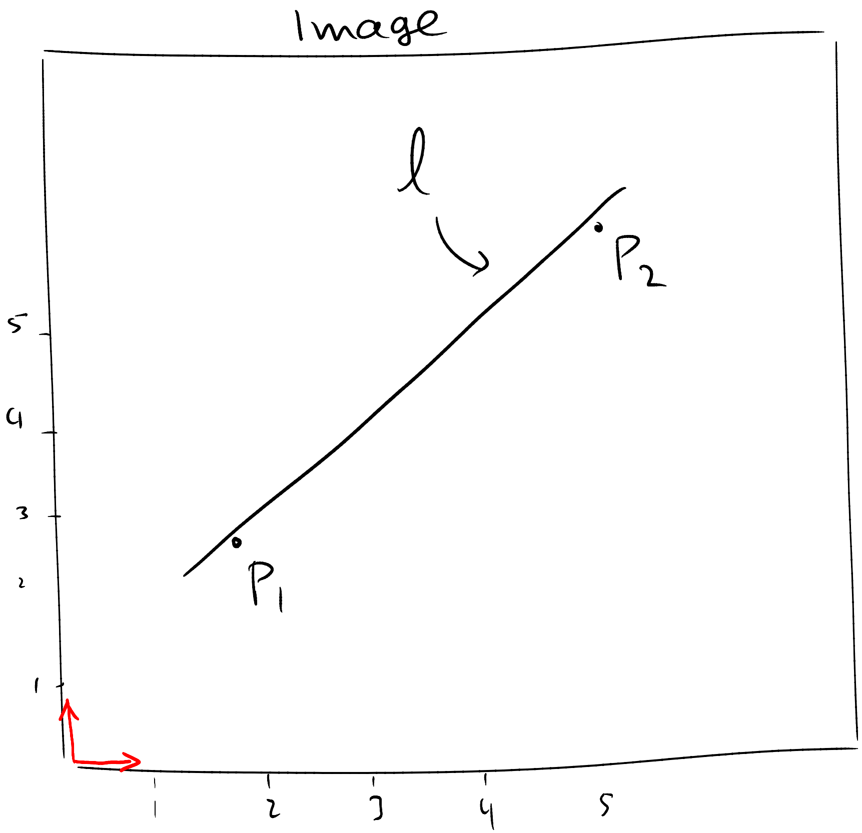
Example: line $y=x$ in homogeneous coords:

$$1.x - 1.y + 0.1 = 0$$

line parameters of ℓ

$$\begin{bmatrix} 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

The Line Passing Through 2 Points



Calculating the parameters of a line through two points with homogeneous coordinates p_1, p_2

$$l = p_1 \times p_2$$

cross product of two 3D vectors

- l must satisfy $l^T \cdot p_1 = 0$, $l^T \cdot p_2 = 0$
- taken as 3D vectors, l is perpendicular to both p_1 and p_2

$\Rightarrow$ it is along the cross product, $p_1 \times p_2$

- In homogeneous coordinates

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

or $l^T \cdot p = 0$

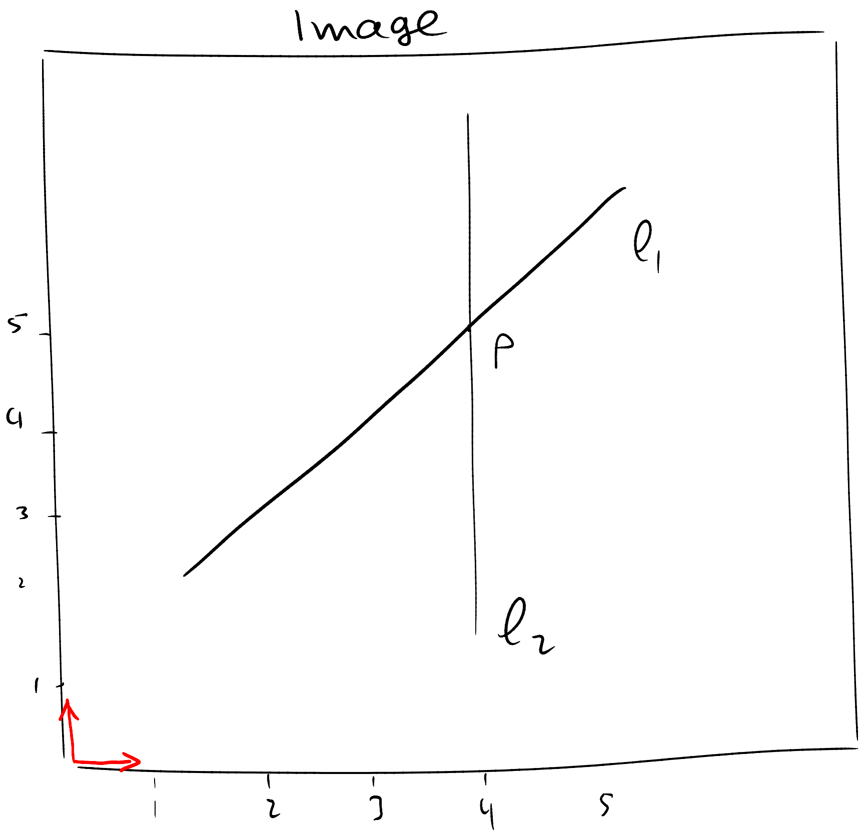
Cross product in matrix form

To compute the cross product using matrix multiplication do:

$$l_1 \times l_2 = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix} l_2$$

where $l_1=[a, b, c]^T$

The Point of Intersection of Two Lines



Calculating the homogeneous coordinates of the intersection of two lines ℓ_1, ℓ_2

$$p = \ell_1 \times \ell_2$$

↑ cross product of two 3D vectors

- p must satisfy $\ell_1^T p = 0$, $\ell_2^T p = 0$
- taken as 3D vectors, p is perpendicular to both ℓ_1 and ℓ_2

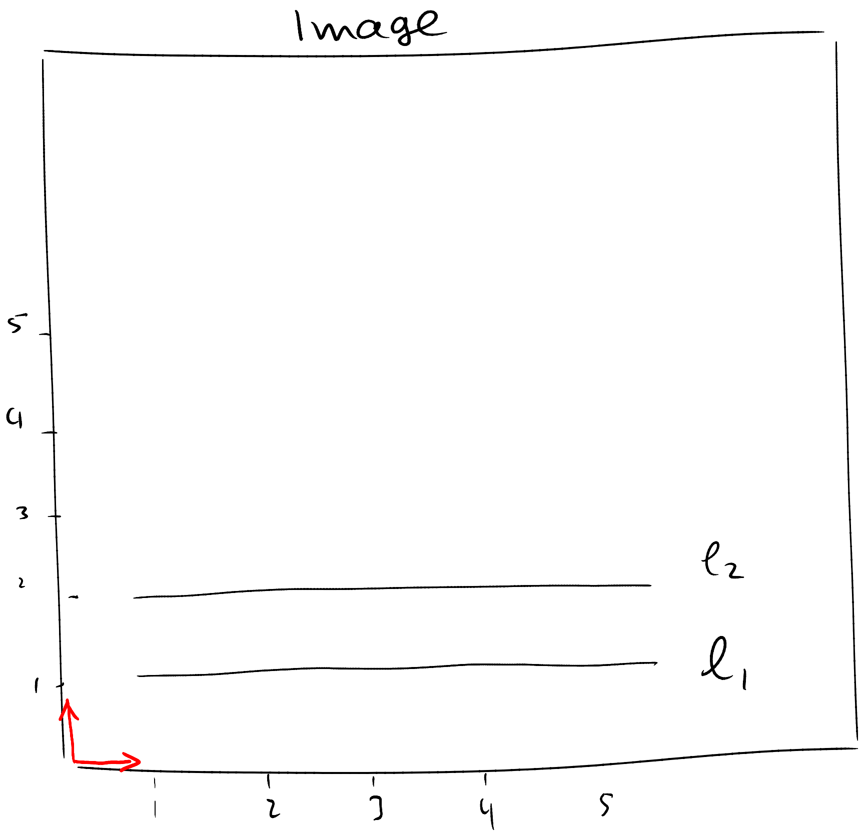
$\Rightarrow$ it is along the cross product, $\ell_1 \times \ell_2$

- In homogeneous coordinates

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

or $\ell^T p = 0$

Computing the Intersection of Parallel Lines



Calculating the homogeneous coordinates of the intersection of two lines ℓ_1, ℓ_2

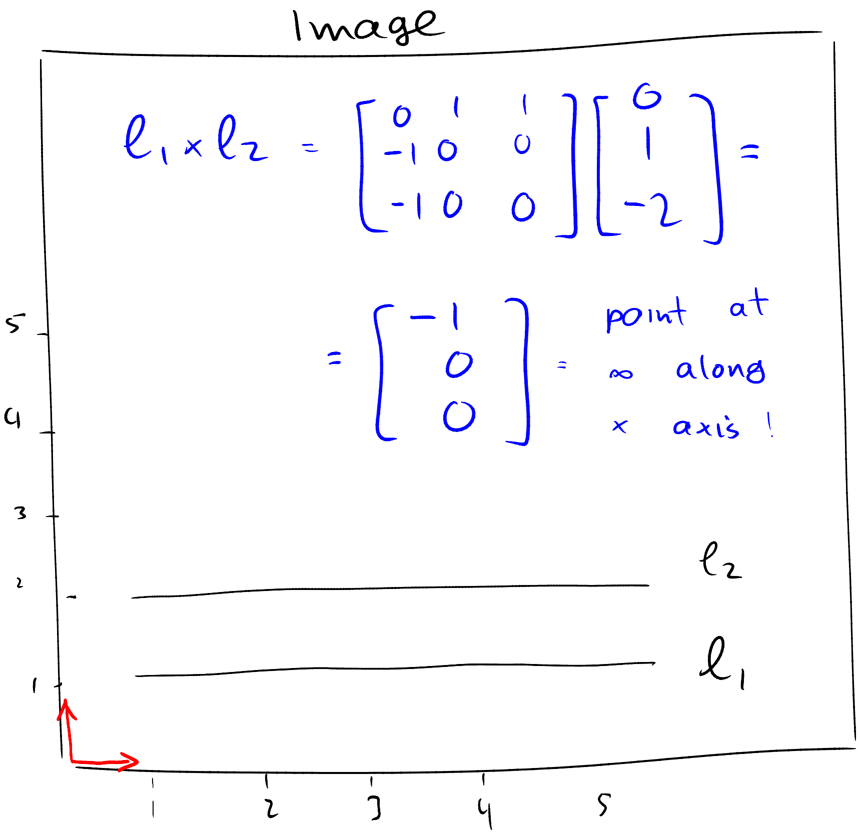
$$\boxed{p = \ell_1 \times \ell_2}$$

cross product of two 3D vectors

This calculation works even when ℓ_1, ℓ_2 are parallel !

(no floating point exceptions or divide-by-zero errors !)

Computing the Intersection of Parallel Lines



$$\ell_1 \times \ell_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix} =$$

$$= \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} =$$ point at ∞ along x axis !

Calculating the homogeneous coordinates of the intersection of two lines ℓ_1, ℓ_2

$$p = \ell_1 \times \ell_2$$

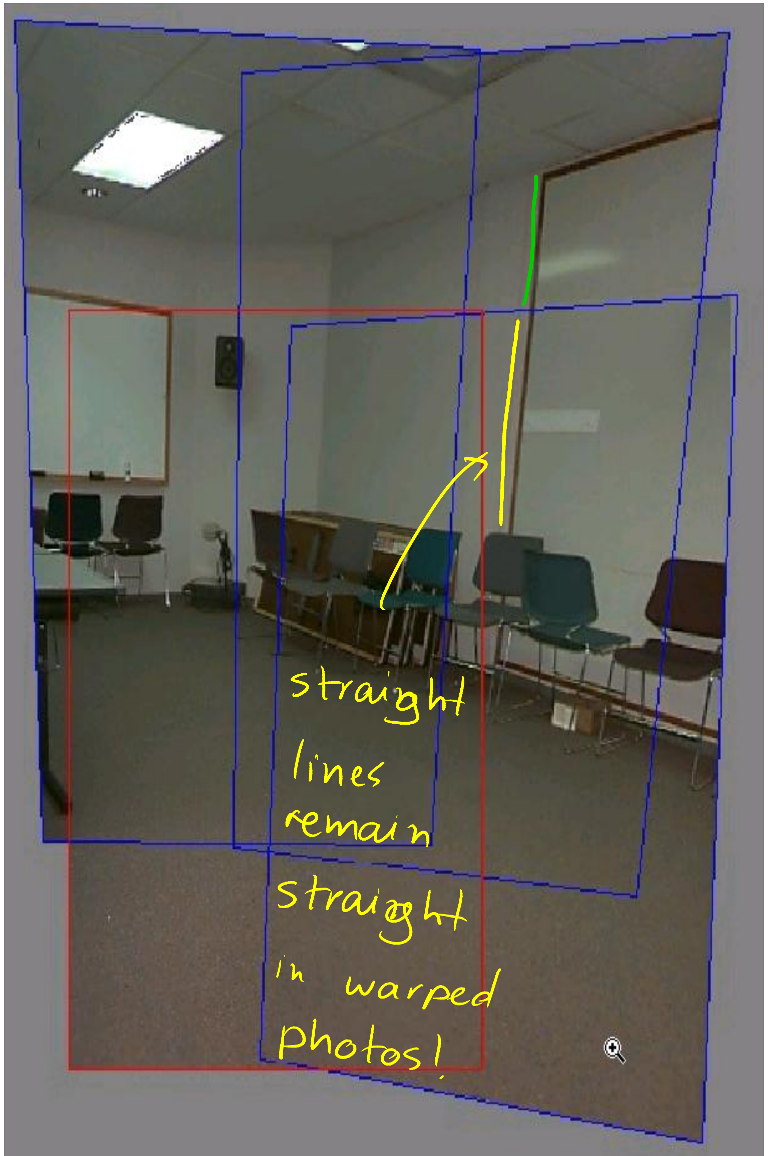
cross product of two 3D vectors

Topic 11:

Homographies & Image Mosaics

- Introduction to image mosaicing
- Homogeneous coordinates for points & lines
- Image homographies
- Estimating homographies from point correspondences
- The autostitch algorithm

Linear Image Warps

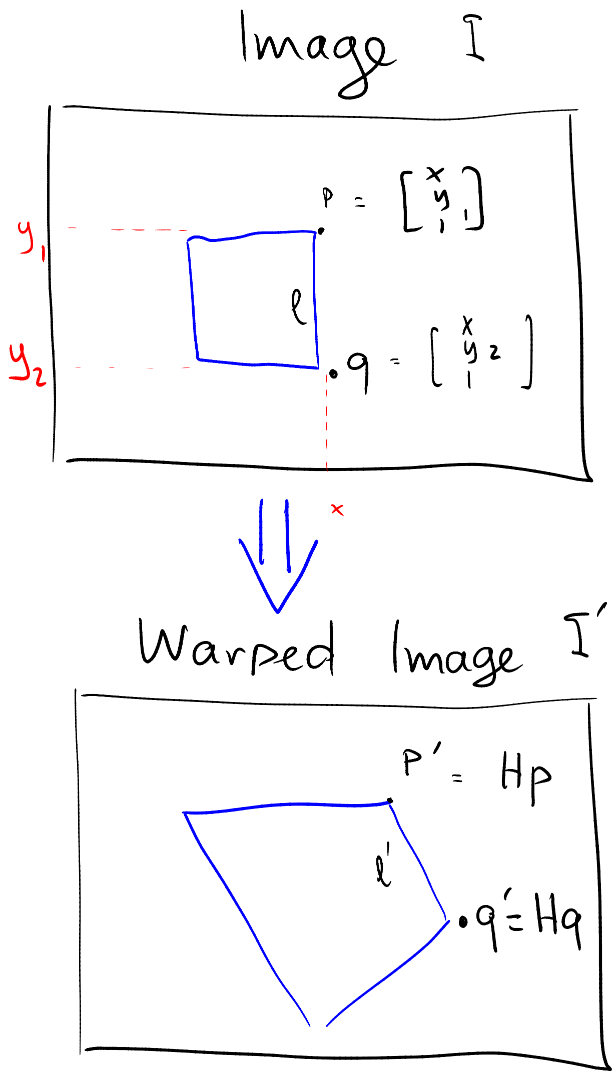


Basic insight:

To align multiple photos for mosaicing we must warp them in a way that preserves all lines

(i.e. lines before warping remain lines after warping)

Linear Image Warps & Homographies



The matrix H is called a **Homography**

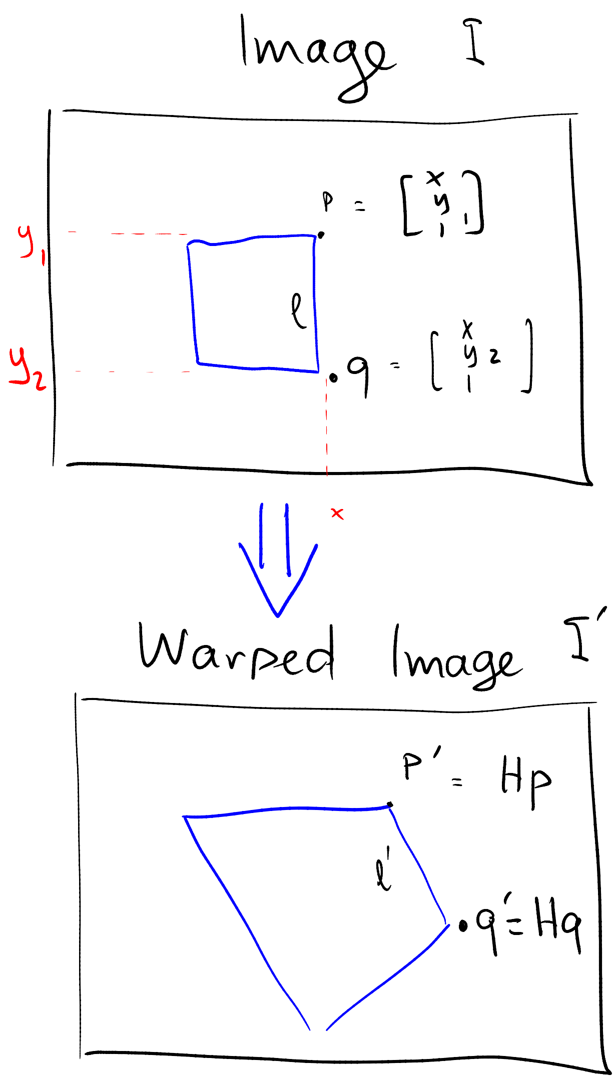
· **Definition (Linear Image Warps)**

An image warp is called linear if every 2D line l in the original image is transformed into a line l' in the warped image (i.e. the warp preserves all lines in the original photo)

· **Property (w/out proof)**

Every linear warp can be expressed as a 3×3 matrix H that transforms homogeneous image coordinates

Warping Images Using a Homography

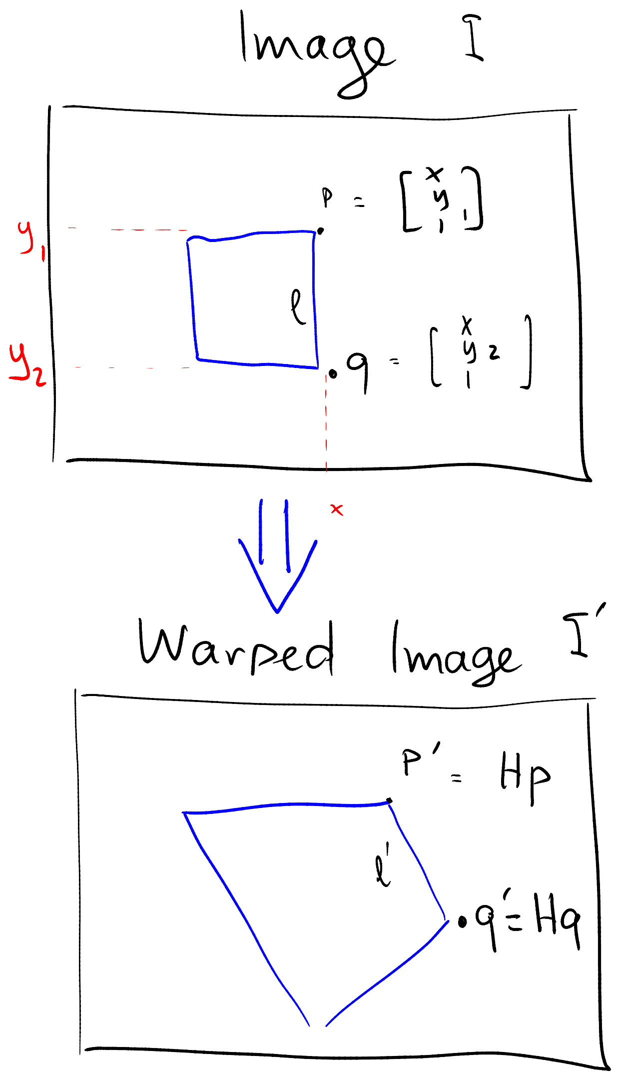


Linear warping equation

$$\underbrace{I(p)}_{\text{intensity at pixel in source image with homogeneous coordinates } p} = \underbrace{I'(Hp)}_{\text{intensity at pixel in warped image with homogeneous coordinates } p' = Hp}$$

The matrix H is called a Homography

Warping Images Using a Homography



The matrix H is called a **Homography**

- Linear warping equation

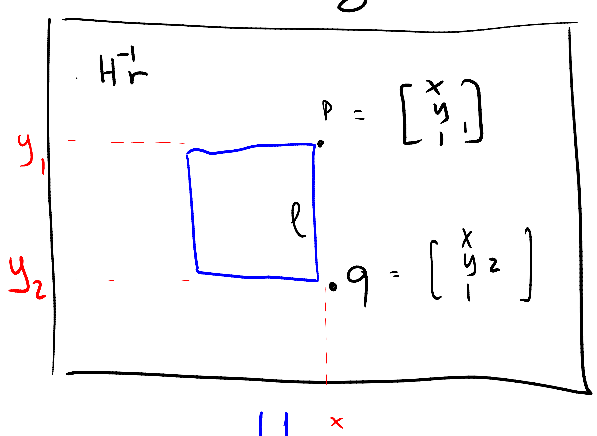
$$\underbrace{I(p)}_{\text{intensity at pixel in source image with homogeneous coordinates } p} = \underbrace{I'(Hp)}_{\text{intensity at pixel in warped image with homogeneous coordinates } p' = Hp}$$

- Note: Scaling H by a factor $\lambda \neq 0$ does not change the homography:

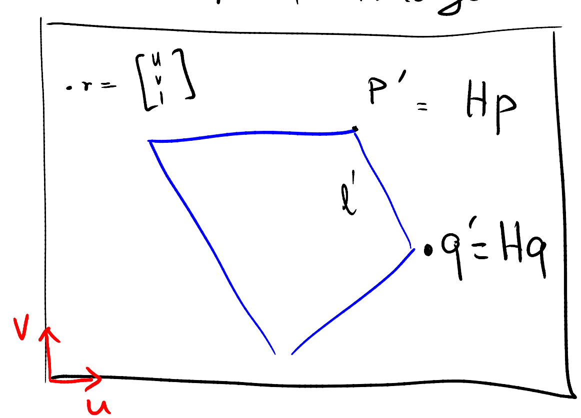
$$(\lambda \cdot H)p = H \cdot (\lambda p) \cong Hp$$

Warping Images Using a Homography

Image I



Warped Image I′



The matrix H is called a **Homography**

- Linear warping equation

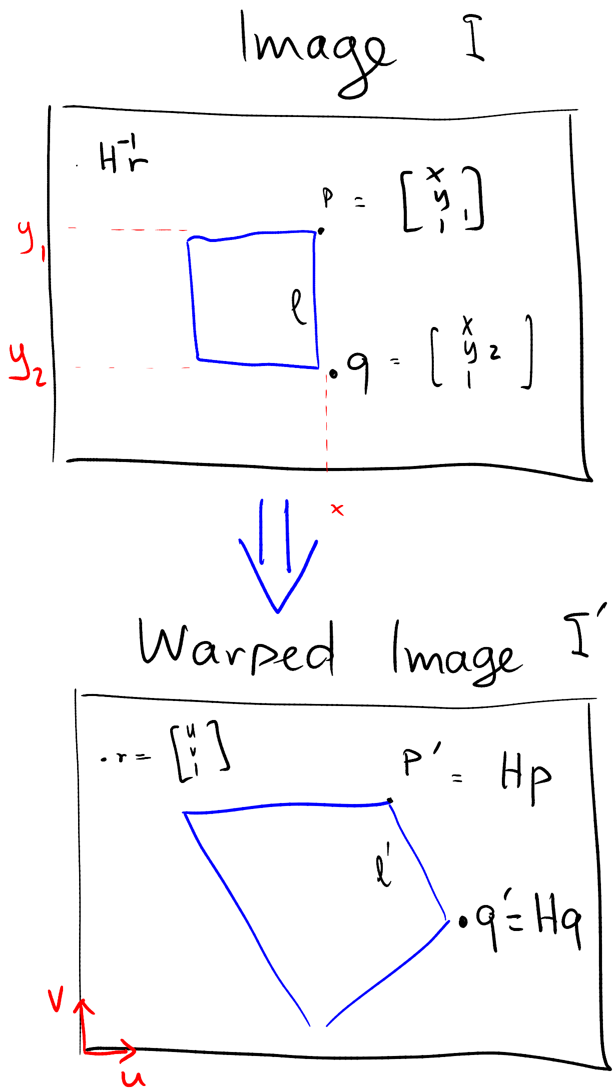
$$I(p) = I'(Hp)$$

$$I(H^{-1}r) = I'(r)$$

- Property (w/out proof)

Every linear warp can be expressed as a 3x3 matrix H that transforms homogeneous image coordinates

Warping Images Using a Homography



The matrix H is called a **Homography**

- Linear warping equation

$$I'\left(\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}\right) = I\left(H^{-1}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}\right)$$

$$I(H^{-1}r) = I'(r)$$

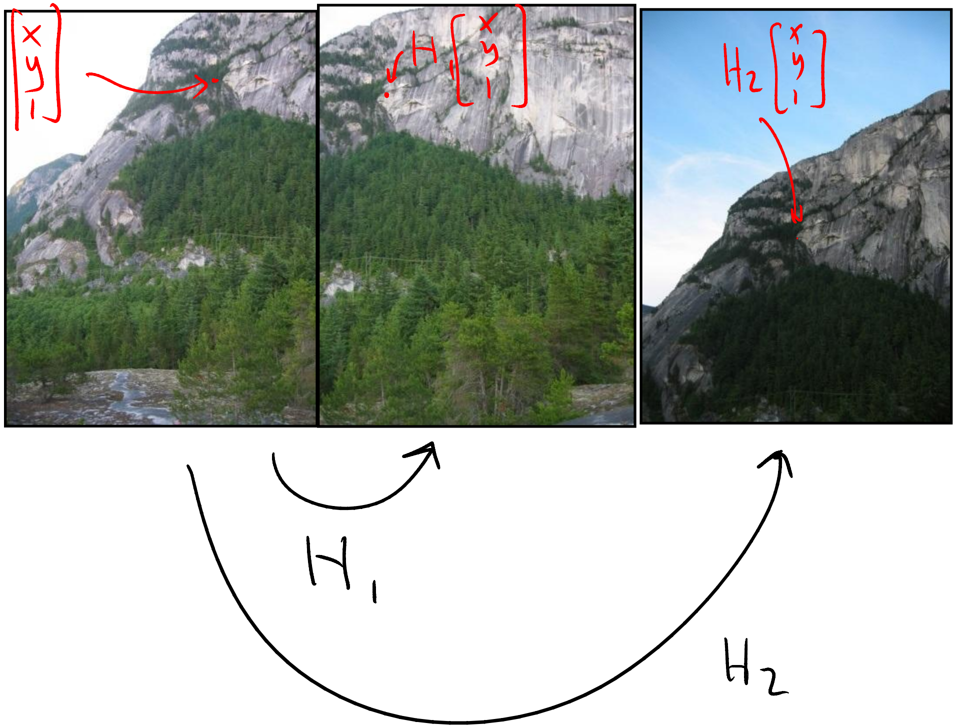
- Computing warp I' from I and H

① Compute H^{-1}

② To compute color of pixel (u,v) in warped image:

- compute $\begin{bmatrix} a \\ b \\ c \end{bmatrix} = H^{-1}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$
- copy color from $I(a/c, b/c)$

Homographies & Image Mosaicing

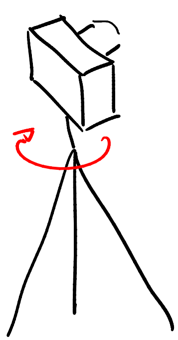


Useful property #3

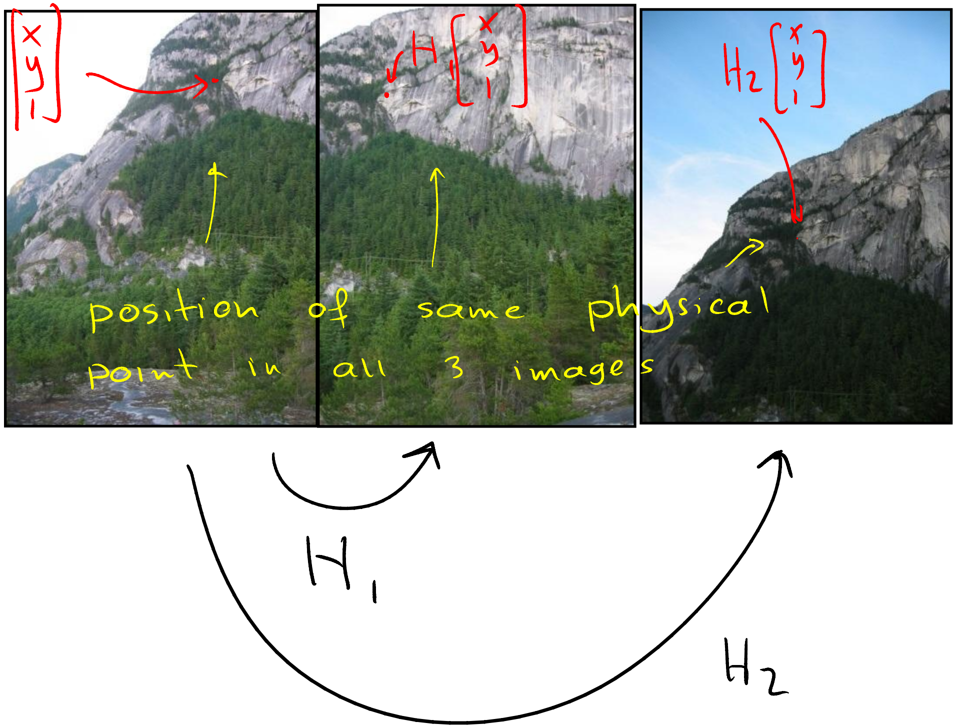
- Every photo taken from a tripod-mounted camera is related by a homography

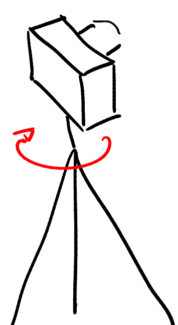
Assumptions:

- No lens distortions
- Camera's center of projection does not move while camera is mounted on tripod



Homographies & Image Mosaicing





Useful property # 3

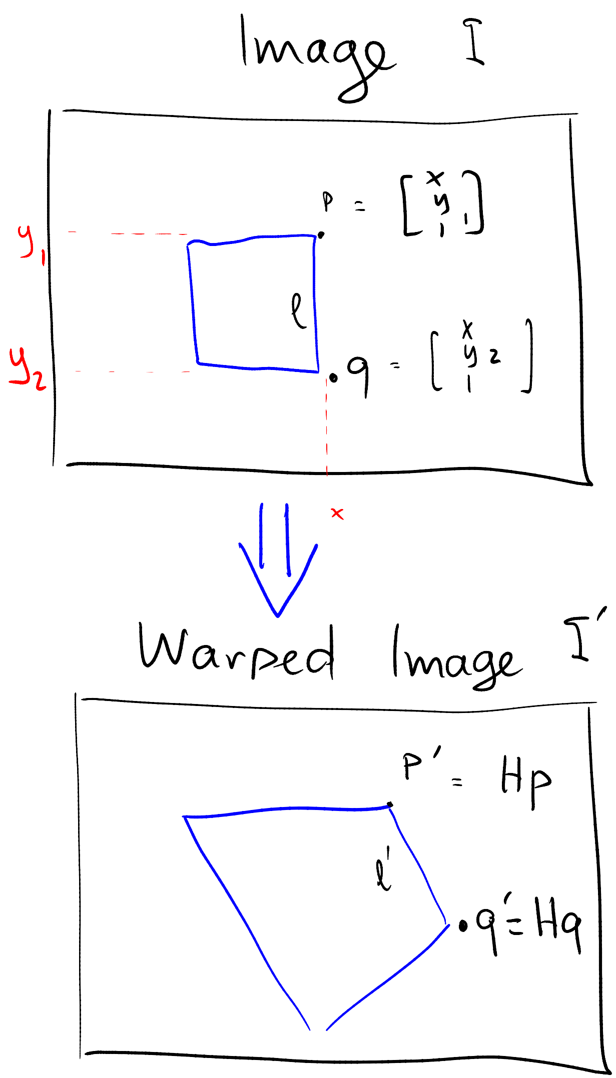
- Every photo taken from a tripod-mounted camera is related by a homography
- These homographies are <u>unknown</u>
- To align these photos for mosaicing we must estimate $H_1, H_2, \ldots$ etc

Topic 11:

Homographies & Image Mosaics

- Introduction to image mosaicing
- Homogeneous coordinates for points & lines
- Image homographies
- Estimating homographies from point correspondences
- The autostitch algorithm

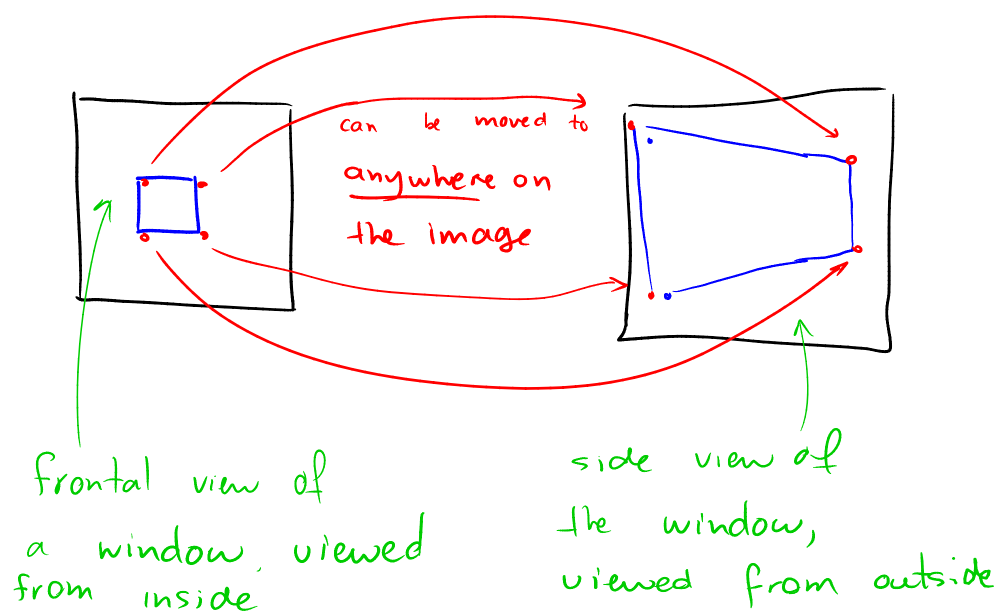
Homography Estimation: Basic Intuition



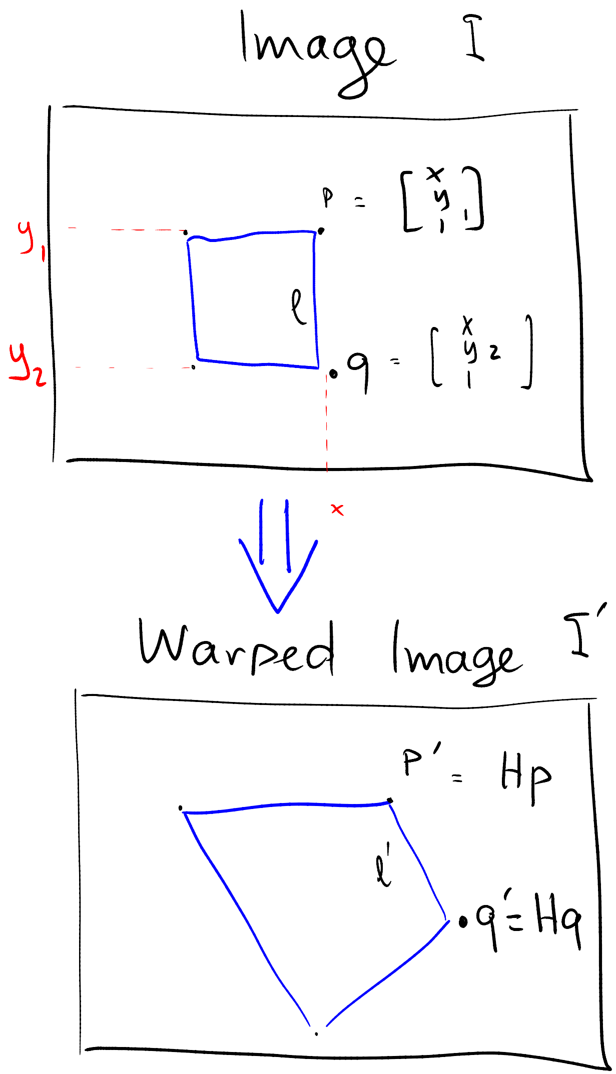
The matrix H is called a **Homography**

Intuition:

Linear warps correspond to every possible distortion of a square created by moving its vertices to arbitrary locations

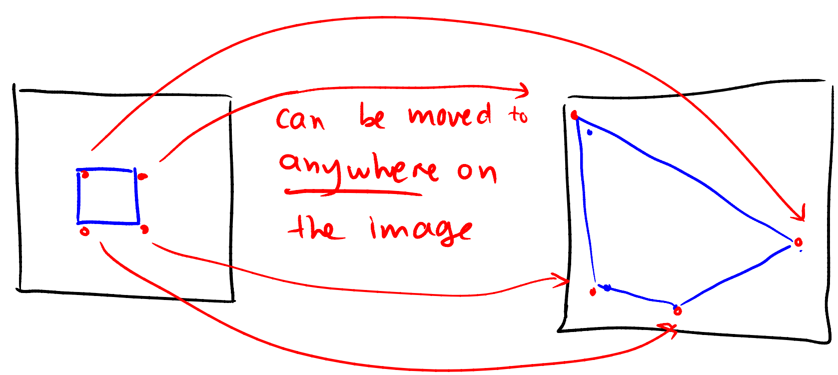


Estimating Homographies from Point Correspondences



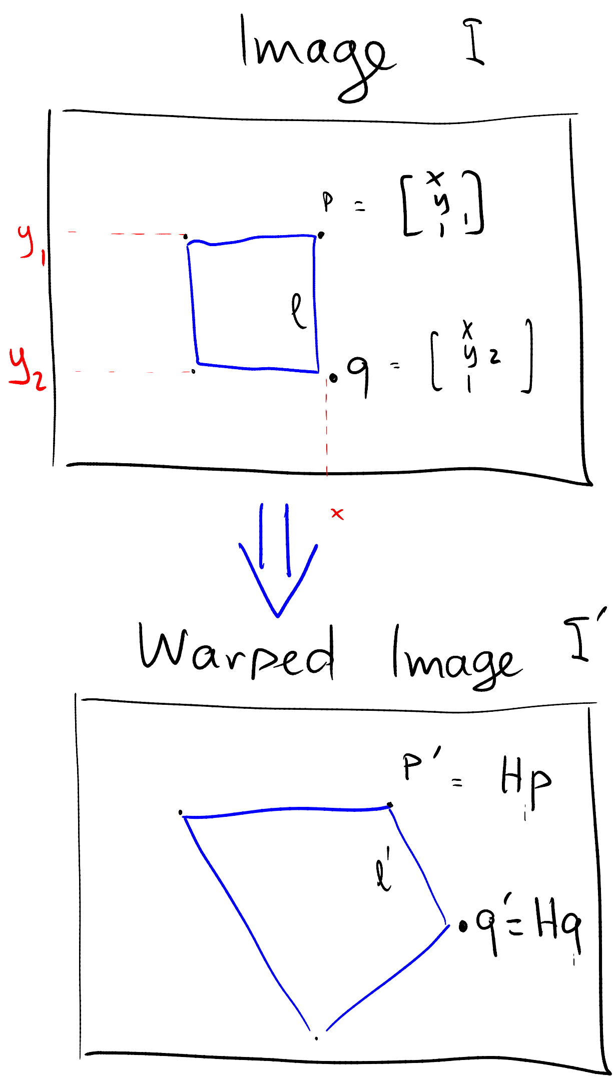
Intuition:

If we have a correspondence between 4 points in the two images, we can compute H



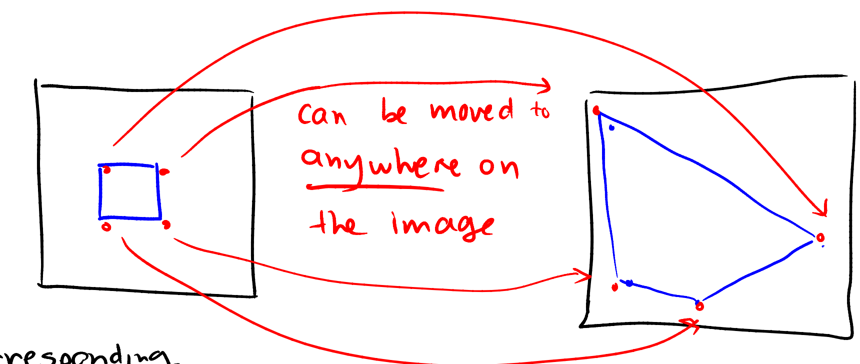
The matrix H is called a Homography

Estimating Homographies from Point Correspondences



Intuition:

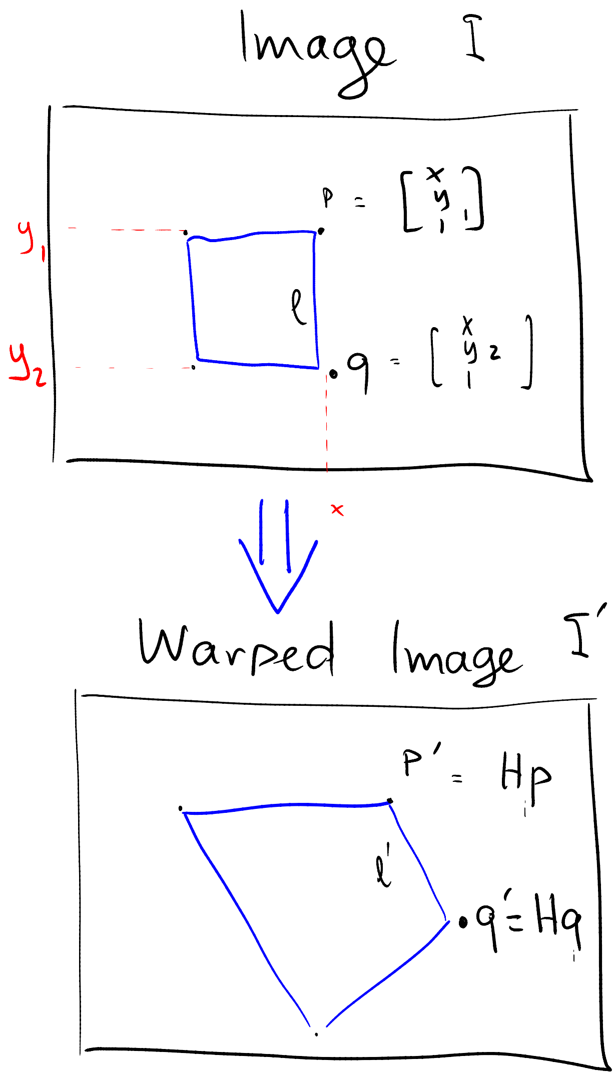
If we have a correspondence between 4 points in the two images, we can compute H_i



corresponding pixel in I' → H ← pixel in I

$$\underbrace{\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix}}_{\text{known}} \simeq \underbrace{\begin{bmatrix} a & b & c \\ d & e & f \\ h & i & j \end{bmatrix}}_{\text{unknown}} \underbrace{\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}}_{\text{known}}$$

Homography Estimation by Solving Linear System



Given point correspondances X1 and X2 and writing the homography H as:

$$
\begin{aligned}
\mathbf{h} &= (H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33})^T \\
\mathbf{a}_x &= (-x_1, -y_1, -1, 0, 0, 0, x_2'x_1, x_2'y_1, x_2')^T \\
\mathbf{a}_y &= (0, 0, 0, -x_1, -y_1, -1, y_2'x_1, y_2'y_1, y_2')^T .
\end{aligned}
$$

the homography can be estimated using least squares as Ah=0, where:

$$
A = \begin{pmatrix} \mathbf{a}_{x1}^T \\ \mathbf{a}_{y1}^T \\ \vdots \\ \mathbf{a}_{xN}^T \\ \mathbf{a}_{yN}^T \end{pmatrix} .
$$

$$x_i' = (a x_i + b y_i + c)/(h x_i + k y_i + l) \Leftrightarrow$$

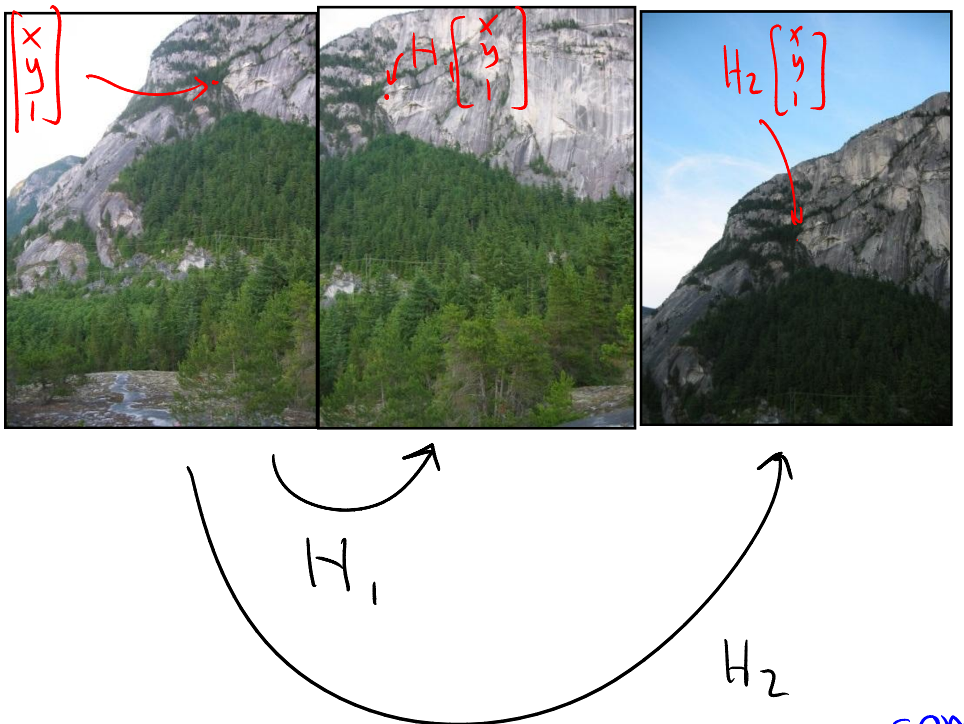
$$x_i'(h x_i + k y_i + l) - (a x_i + b y_i + c) = 0$$

Topic 11:

Homographies & Image Mosaics

- Introduction to image mosaicing
- Homogeneous coordinates for points & lines
- Image homographies
- Estimating homographies from point correspondences
- The autostitch algorithm

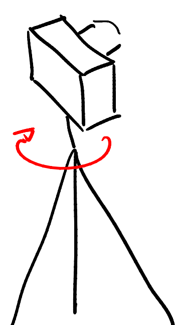
Feature-Based Image Matching



The Autostitch Algorithm

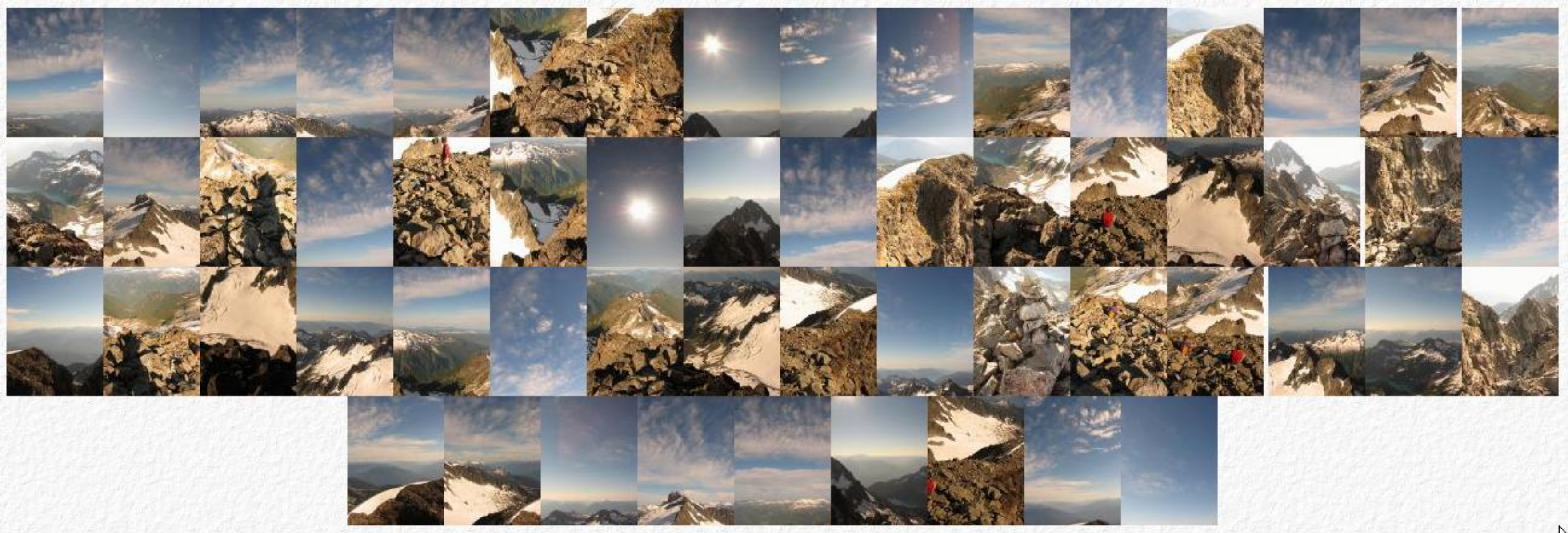
1. Run SIFT on each image
2. Match SIFT features across all photos
3. For every pair of photos:
   - Choose matches & compute H
4. Warp photos according to H
5. Blend photos using pyramid blending

because some matches maybe incorrect (i.e. outliers) RANSAC is used



Building Panoramic Image Mosaics

Input images



⇓ automatically created mosaic

What You Will Take Away …

#1: Yes, math IS useful in CS !!

#2: How to turn math into pictures

#3: Basics of image analysis & manipulation

#4: How to code interactive tools

#5: How to read research papers

Visual Computing Principles

Imaging essentials

Understanding pixel intensity & color

Image representation & transformation

Image ⇔ 2D array of pixels

camera response functions
pixel representations & matting

Image ⇔ continuous 2D function

polynomial fitting, WLS, RANSAC
derivative estimation
curves, curvature, gradients,
Laplacian, Hessian, edge & corner
detection

Image ⇔ n-dimensional vector

correlation, convolution, PCA
filtering ⇔ derivative computations
smoothing

Hierarchical image representations

pyramids, wavelets,
scale-space representations, SIFT

Homogeneous point representations

+ applications: Inpainting, scissoring, texture synthesis, alpha matting
morphing, mosaicing, recognition, feature matching, ...

What Comes Next...

Computer Vision Courses		Other Courses	
CS420	Into to Image Understanding	CS4 18	Computer Graphics
		CS321	Neural Networks
		CS411	Machine Learning & Data Mining
CS 2530	Visual Modeling	CSC 2521	Machine Learning for Computer Graphics
CS 2523	Visual Recognition	CSC 2522	Advanced Image Synthesis
CS 2539	Visual Motion Analysis	CSC 2529	Computer Animation