SIFT features
Deblurring

Fredo Durand
MIT EECS 6.815/6.865
SIFT
Goals for handling scale/rotation

• Ensure invariance
  – Same scene points must be detected in all images

• Prepare terrain for descriptor
  – Descriptors are based on a local neighborhood
  – Find canonical scale and angle for neighborhood (independently in all images)

Later do the matching between descriptors
Goal

• Compute stable features in space and scale

• For each feature magically compute some local characteristic scale and angle
  – consistent on a scaled or rotated version of the image

• Descriptors from window given by x, y, scale and angle
  – Ideally, the descriptor will be the same if we run it on a scaled or rotated version of the image
  – The descriptor will be invarient to rotation, on the result...
Select canonical orientation for descriptor

- One option:

- histogram of local gradient directions
- canonical orientation: peak of smoothed histogram
Center-surround across scale

- We want to analyze a “star” image
Center-surround across scale

• We want to analyze a “star” image
Center-surround across scale

- We want to analyze a “star” image

small scale:
zero response,
center has same
color as surround
Center-surround across scale

- We want to analyze a “star” image

small scale: zero response, center has same color as surround
Center-surround across scale

- We want to analyze a “star” image

small scale:
zero response,
center has same
color as surround

scale 2:
small response,
surround has
some white
Center-surround across scale

- We want to analyze a “star” image

small scale:
zero response,
center has same
color as surround

scale 2:
small response,
surround has
some white
Center-surround across scale

- We want to analyze a “star” image

small scale: zero response, center has same color as surround

scale 2: small response, surround has some white

\( f \) vs. region size
Center-surround across scale

- We want to analyze a “star” image

small scale: zero response, center has same color as surround

scale 2: small response, surround has some white

scale 3: big response, center is green, surround is white

\[ f \]

region size
Center-surround across scale

- We want to analyze a “star” image

small scale: zero response, center has same color as surround

scale 2: small response, surround has some white

scale 3: big response, center is green, surround is white
Center-surround across scale

- We want to analyze a “star” image

**small scale:**
- zero response,
- center has same color as surround

**scale 2:**
- small response,
- surround has some white

**scale 3:**
- big response,
- center is green,
- surround is white

**scale 4:**
- medium response,
- center is less green,
- surround is white

![Graph](graph.png)
Center-surround

- Small scale: zero response, center has same color as surround
- Scale 2: small response, surround has some white
- Scale 3: big response, center is green, surround is white
- Scale 4: medium response, center is less green, surround is white

• Note: the filter must be evaluated across space as well: Extrema over space & scale
Center-surround in practice

• Functions for determining scale

Kernels:

\[ L = \sigma^2 \left( G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right) \]

(Laplacian: 2nd derivative of Gaussian)

\[ DoG = G(x, y, k\sigma) - G(x, y, \sigma) \]

(Difference of Gaussians)

where Gaussian

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

Note: both kernels are invariant to scale and rotation
Three 1D bumps of different sizes

Let’s see how our difference of Gaussian works
ideally, it would find a scale proportional to each bump’s size
Bumps, filtered by difference-of-Gaussian

white means high positive response
grey means zero
Bumps, filtered by difference-of-Gaussian

Consider response across scale at center of bumps
Bumps, filtered by difference-of-Gaussian

Scales of peak responses are proportional to bump width

\[
\frac{[1.7, 3, 5.2]}{[5, 9, 15]} = 0.3400 \quad 0.3333 \quad 0.3467
\]
Visualize difference of Gaussian that achieves the extremum together with bumps

Note how the positive bump exactly matches the bump
Recap scale invariance

- Compute scale-invariant function over window
  - center surround, difference of Gaussian
- Extract extrema along space & scale
Scale discretization

- We need to find extrema over scale
- Brute force search: compute image at a discrete set of scales and apply center-surround
  - Trick: re-use a given Gaussian for center and surround of two adjacent scales
- Question: how finely do we subdivide scale?
Repeatability vs number of scales sampled per octave

Octave: factor of 2 in image scale
Scale Invariant Detectors

• **Harris-Laplacian**¹

  *Find local maximum of:*
  
  – Harris corner detector in space (image coordinates)
  – Laplacian in scale

Scale Invariant Detectors Zoo

• **Harris-Laplacian**¹
  
  *Find local maximum of:*
  
  – Harris corner detector in space (image coordinates)
  
  – Laplacian in scale


• **SIFT (Lowe)**²
  
  *Find local maximum of:*
  
  – Difference of Gaussians in space and scale

² D. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. Accepted to IJCV 2004
Scale Invariant Detectors

- Experimental evaluation of detectors w.r.t. scale change

Repeatability rate:

\[
\frac{\text{# correspondences}}{\text{# possible correspondences}}
\]

Advanced Key point localization

- Detect maxima and minima of difference-of-Gaussian in scale space
- Fit a quadratic to surrounding values for sub-pixel and sub-scale interpolation (Brown & Lowe, 2002)
- Taylor expansion around point:

\[ D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \]

- Offset of extremum (use finite differences for derivatives):

\[ \hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x} \]
Scale and Rotation Invariant Detection: Summary

• Given: two images of the same scene with a large scale difference and/or rotation between them
• Goal: find the same interest points independently in each image
• Solution: search for maxima of suitable functions in scale and in space (over the image). Also, find characteristic orientation.

Methods:
Harris-Laplacian [Mikolajczyk, Schmid]: maximize Laplacian over scale, Harris’ measure of corner response over the image
SIFT [Lowe]: maximize Difference of Gaussians over scale and space
Example of keypoint detection

Threshold on value at DOG peak and on ratio of principle curvatures (Harris approach)
Recap

• We have scale & rotation invariant distinctive feature
• We have a characteristic scale
• We have a characteristic orientation

• Now we need to find a descriptor of this point
  – For matching with other images
SIFT descriptor
Recall: Matching with Features

• Problem 2:
  – For each point correctly recognize the corresponding one
Recall: Matching with Features

- Problem 2:
  - For each point correctly recognize the corresponding one

We need a reliable and distinctive descriptor
CVPR 2003 Tutorial

Recognition and Matching Based on Local Invariant Features

David Lowe

Computer Science Department

University of British Columbia

Invariant Local Features

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters.
Motivation for using gradients:
gradient orientations somewhat invariant to lighting variations

SIFT vector formation

- Computed on rotated and scaled version of window according to computed orientation & scale
  - resample a 16x16 version of the window
- Based on gradients weighted by a Gaussian of variance half the window (for smooth falloff)
SIFT vector formation

• 4x4 array of gradient orientation histograms
  – not really histogram, weighted by magnitude
• 8 orientations x 4x4 array = 128 dimensions
• Motivation: some sensitivity to spatial layout, but not too much.
Reduce effect of illumination

• 128-dim vector normalized to 1

• Threshold gradient magnitudes to avoid excessive influence of high gradients
  – after normalization, clamp gradients >0.2
  – renormalize
Figure 8: This graph shows the percent of keypoints giving the correct match to a database of 40,000 keypoints as a function of width of the $n \times n$ keypoint descriptor and the number of orientations in each histogram. The graph is computed for images with affine viewpoint change of 50 degrees and addition of 4% noise.
Feature stability to noise

- Match features after random change in image scale & orientation, with differing levels of image noise
- Find nearest neighbor in database of 30,000 features
Feature stability to affine change

• Match features after random change in image scale & orientation, with 2% image noise, and affine distortion
• Find nearest neighbor in database of 30,000 features
Distinctiveness of features

- Vary size of database of features, with 30 degree affine change, 2% image noise
- Measure % correct for single nearest neighbor match
Figure 12: The training images for two objects are shown on the left. These can be recognized in a cluttered image with extensive occlusion, shown in the middle. The results of recognition are shown on the right. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affine transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition.
Figure 13: This example shows location recognition within a complex scene. The training images for locations are shown at the upper left and the 640x315 pixel test image taken from a different viewpoint is on the upper right. The recognized regions are shown on the lower image, with keypoints shown as squares and an outer parallelogram showing the boundaries of the training images under the affine transform used for recognition.
A good SIFT features tutorial


By Estrada, Jepson, and Fleet.

The original paper:
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.8899
Recap

- Stable (repeatable) feature points can be detected regardless of image changes
  - Scale: search for correct scale as *maximum* of appropriate function
- Invariant and distinctive descriptors can be computed
Autopano demo

- Uses sift
Deblurring
Sources of blur

• Spherical aberrations
• Axial chromatic aberrations
• Diffraction
• Camera shake
• Object movement
• Defocus
• Camera antialiasing filter
• Atmospheric turbulence
Sources of blur

- Spherical aberrations
- Axial chromatic aberrations
- Diffraction
- Camera shake
- Object movement
- Defocus
- Camera antialiasing filter
- Atmospheric turbulence
Simple model of blur: convolution

- Limited
  - Blur sometimes varies spatially
  - or depends on depth, object, etc.
- but will often be true locally
  - and insights usually generalize
Can we remove blur?
Problem statement

• We are given a blurry image $y$
• We know it came from the convolution of a sharp image $x$ with a known kernel $k$:
  \[
  y = x \ast k
  \]
• We want to retrieve $x$

• More general (harder problem): blind convolution
  – We don’t know $k$
Convolution is linear!

- We can write it as a big matrix $y = Mx$
- Deconvolution: just solve $Mx = y$!

Three important assumptions:
- we know the blur kernel $k$ (or $M$) perfectly
- blur is perfect, there is no sensor noise
- we assume the image encoding is linear
Solver?

- Deconvolution: just solve $Mx = y$!
Roadmap

• Our system is sparse
  – will allow us to speed things up
• Our system might be ill-conditioned
  – need to regularize
Linear least squares
Images and linear algebra

• We want to use powerful tools in linear algebra
  – System $Mx = y$
• Our unknown $x$ is an image
• We could solve deconvolution by stacking all our pixels into a big numpy 1D vector $x$, build a big matrix $M$, solve for $Mx = b$, then copy the pixels back into the image
• Instead, we’ll keep images as numpy arrays, but we’ll use abstract derivations in terms of $M$ and $x$ to know what to do
• The implementation will use simple image operations such as convolution, multiplication
Least squares

- Turn equation into a least square

$$\min \|Mx-y\|^2$$

- deals with over ill-conditioned systems
- enables derivation of iterative methods in terms of gradient descent
- enables the addition of other energy terms to regularize the system
Turn it into a least square

• \( \min \|Mx-y\|^2 \)
Turn it into a least square

- min $\|Mx-y\|^2$
- min $(Mx-y)^T (Mx-y)$
- min $x^T M^T Mx - 2x^T M^T y + y^T y$

- derive wrt $x$, set to 0
- $2M^T Mx - 2M^T y$

- Normal equation:
- $M^T Mx - M^T y = 0$
Keep in mind

• $\min ||Mx-y||^2$
• $\min x^T M^T M x - 2x^T M^T y + y^T y$
• $M^T M x - M^T y = 0$

• In our case:
  - $x$ is the unknown sharp image
  - $y$ is the blurry input
  - $M$ is the convolution by the known blurring kernel
Notation

- $\min x^T M^T M x - 2x^T M^T y + y^T y$
- $M^T M x - M^T y = 0$
Notation

\begin{itemize}
  \item \( \min x^T M^T M x - 2 x^T M^T y + y^T y \)
  \item \( M^T M x - M^T y = 0 \)
  \item call \( A = M^T M ; b = M^T y ; c = y^T y \)
\end{itemize}

add a 1/2 factor to get a cleaner derivative:

\begin{itemize}
  \item minimize \( f(x) = \frac{1}{2} x^T A x - b^T x + c \)
  \item derivative: \( f'(x) = A x - b \)
\end{itemize}
Gradient descent

with many illustrations from Jonathan Schewchuck
Linear least squares

- Minimize

\[ f(x) = \frac{1}{2} x^T A x - b^T x + c \]

- \( A = M^T M \) is nicely positive semi definite which means we have a nice parabola with a minimum (not a maximum)

Simple 2-variable example:

\[
\begin{bmatrix}
3 & 2 \\
2 & 6
\end{bmatrix} x = \begin{bmatrix}
2 \\
-8
\end{bmatrix}
\]

Graph of \( f(x) \). Our solution is the minimum

Isocontours of \( f(x) \). Our solution is the dot
Gradient of the quadratic form

\[
f'(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix}
\]

– Multidimensional gradient
(as many dim as rows in \(x\))

Careful:
not an image gradient
but gradient of quadratic

We saw that

\[
f'(x) = Ax - b
\]
New term: Residual

- How different is the value of an equation from the desired value
- At iteration $i$, we are at a point $x(i)$
- Residual of normal form $r(i) = b - Ax(i)$
- Cool property of quadratic form: residual = - gradient
Steepest descent/ascent

- Pick residual (negative gradient) direction
  \[ Ax(i) - b \]
Steepest descent/ascent

- **Pick residual (negative gradient) direction**
  - $A x(i) - b$

- **Find optimum in this direction**

Energy along the gradient direction
Optimum along gradient direction

- minimize $f(x_1)$
- $x_1 = x_0 + \alpha r_0$ unknown?
Optimum along gradient direction

- minimize $f(x_1)$
- $x_{(1)} = x_{(0)} + \alpha r_{(0)}$
- make derivative of $f$ wrt $\alpha$ along direction zero: unknown: $\alpha$
Optimum along gradient direction

- minimize \( f(x_1) \)
- \( x(1) = x(0) + \alpha r(0) \) \( \alpha \) unknown
- make derivative of \( f \) wrt \( \alpha \) along direction zero:

\[
\frac{d}{d\alpha} f(x_1) = f'(x_1) \frac{dx_1}{d\alpha}
\]

\[
b - Ax_1 \sim r(0)
\]

\[
(b - A(x_0 + \alpha r_0))^{T} r_0 = 0
\]

\[
\alpha = \frac{r_0^{T} r_0}{r_0^{T} A r_0}
\]
Recap: Gradient Descent

- Residual = - gradient : r
- Iteratively walk along residual: x
- Find optimal along residual direction:

\[ \alpha = \frac{r_T(i) r(i)}{r_T(i) A r(i)} \]

- note that we only need to apply matrix A to vectors
  - Not too expensive because A is sparse
- Not a very efficient method but we’ll improve it
Applying it to images
**Inputs:**

- input: blurry image $y$, known kernel $A$

$M$: Convolution by $A$

(careful, $A$ is not the kernel itself but the application of the kernel)

$b = M^T y$
Recall gradient descent

\[ r_i = b - Ax_i \]
\[ \alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i} \]
\[ x_{i+1} = x_i + \alpha r_i \]
Recall gradient descent

\[ r_i = b - Ax_i \]
\[ \alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i} \]
\[ x_{i+1} = x_i + \alpha r_i \]

- \( x \) is an unknown image
- \( A = M^T M \) where \( M \) is the blur operator
  - Multiplying by \( M \) just means computing a convolution
  - \( Mx \) is the convolved image
  - \( M^T \) blurs by the flipped kernel
- \( b = M^T y \) is the input blurry image, blurred by the flipped kernel
- \( r_i \) is also an image
Image dot product

\[ r_i = b - Ax_i \]
\[ \alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i} \]
\[ x_{i+1} = x_i + \alpha r_i \]
Image dot product

\[
\begin{align*}
    r_i &= b - Ax_i \\
    \alpha &= \frac{r_i \cdot r_i}{r_i \cdot Ar_i} \\
    x_{i+1} &= x_i + \alpha r_i
\end{align*}
\]

- What does \( r_i \). \( r_i \) mean?
- Dot product: take pairwise product of all the components and sum them, return a scalar
- For images: take the product of all pixel values for all channels and add them
- Easy with numpy operations
Image terms

\[ r_i = b - Ax_i \]
\[ \alpha = \frac{r_i \cdot r_i}{r_i \cdot Ar_i} \]
\[ x_{i+1} = x_i + \alpha r_i \]
Results
Bottomline

- We solve \( Ax = b \)
- But we never form or store matrix \( A \)
- We just do repeated convolutions and a few image dot products.

\[
\begin{align*}
    r_i &= b - Ax_i & \text{convolution, subtraction} \\
    \alpha &= \frac{r_i \cdot r_i}{r_i \cdot Ar_i} & \text{Dot product, convolution, dot product, division of two scalars} \\
    x_{i+1} &= x_i + \alpha r_i & \text{multiplication by scalar, addition}
\end{align*}
\]
Testing the limits
Recall gamma

- Digital images are usually gamma encoded
- Don’t forget to decode before performing linear processing

![Gamma Encoding Diagram]
Non-linear encoding

- Images are usually gamma-encoded
- What if you forget to linearize them?
- Yikes!
Sensor noise

• We usually don’t observe
\[ y = x \otimes k \]

• Noise \( n \) gets added:
\[ y = x \otimes k + n \]
Effect of noise

Blury image
Gaussian, $\text{sigma}=1$

Noise*20

Deblurred
Effect of noise

\[ y = Mx + n \]

\[ \hat{x} = M^{-1}y \]

\[ \hat{x} = M^{-1}(Mx + n) \]

\[ \hat{x} = x + M^{-1}n \]

• Deconvolved noise gets added!
Deconvolving the noise

Noise*20

Deconvolved noise
What’s next?

• Faster convergence with conjugate gradient
• “regularize” the inversion to mitigate the effect of noise
• Let’s understand the properties of $M$ and $M^{-1}$ better
  – to better fight noise
• In some cases, we’ll modify the physical blur (e.g. by changing the lens design) to make sure that deblurring doesn’t introduce as much noise