CMPSCI 370: Intro. to Computer Vision

Image alignment

University of Massachusetts, Amherst
March 22/24, 2016

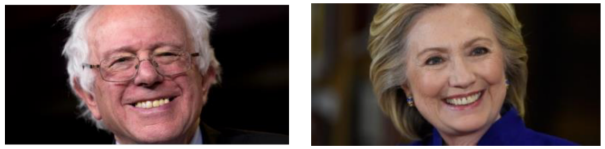
Instructor: Subhransu Maji

Administrivia

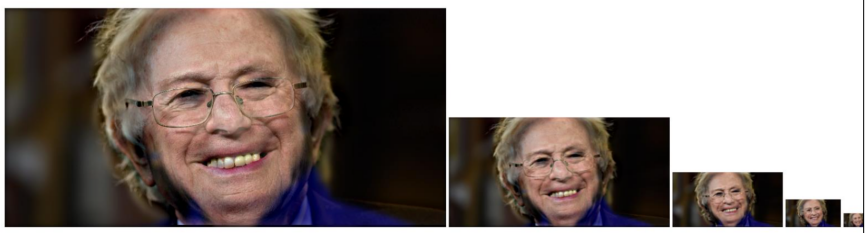
- And now the winners of homework #3 hybrid images
  - as decided by the graders …

Winner (#3): John Williams

Bernie Sanders + Hillary Clinton

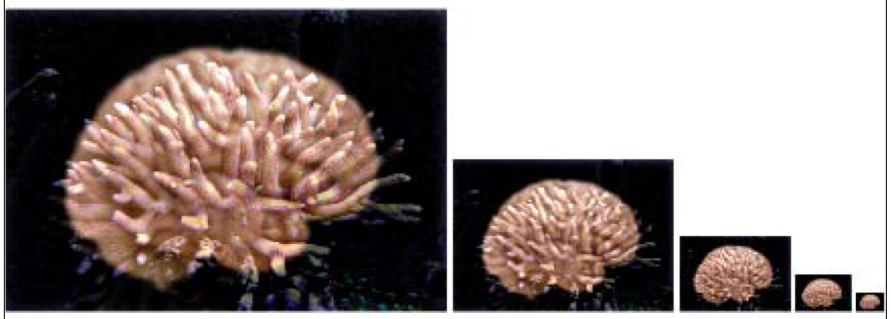


Bernie Clinton

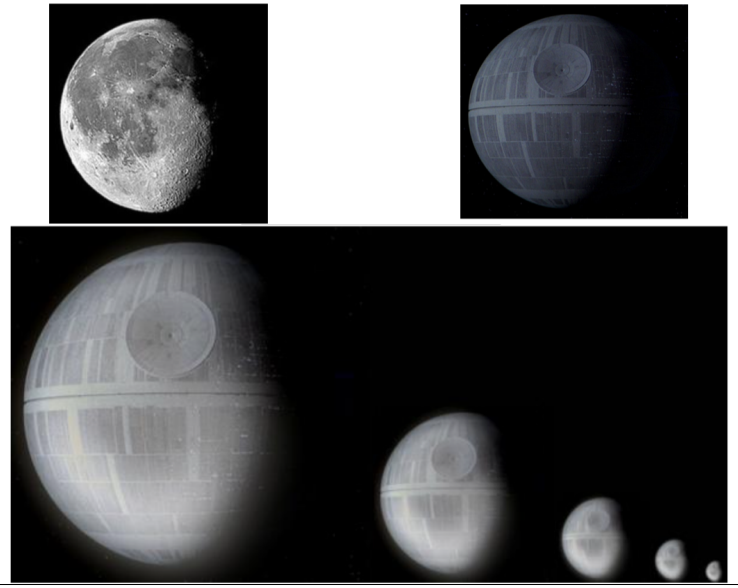


Winner (#2): Joshua Espinosa

Brain & Coral



Winner (#1): Alan Rusell



Honorable mention: Matthew Lydigsen

lighthouse + dalek

Honorable mention: Makenzie Schwartz

hillary + trump

Honorable mention: Nathan Greenberg

?? + simon pegg

Honorable mention: David Carlson

snoop dogg + dog

Image alignment

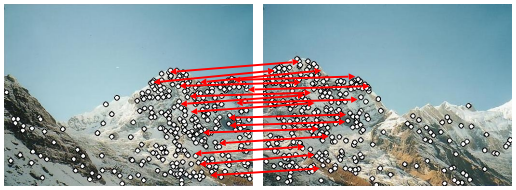
- But first are there any questions?

A framework for alignment

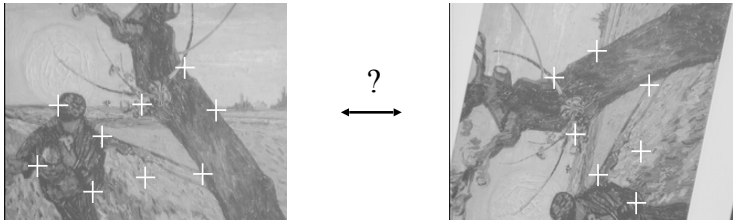
- Matching local features
  - Local information used, can contain outliers
  - But hopefully enough of these matches are good

A framework for alignment

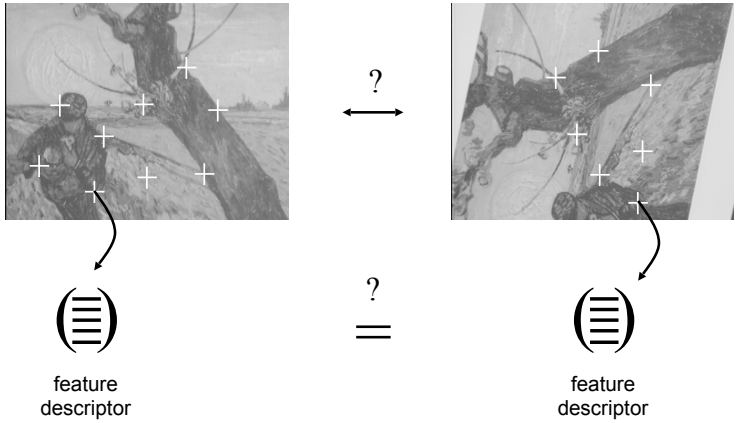
- Matching local features
  - Local information used, can contain outliers
  - But hopefully enough of these matches are good
- Consensus building
  - Aggregate the good matches and find a transformation that explains these matches



Generating putative correspondences



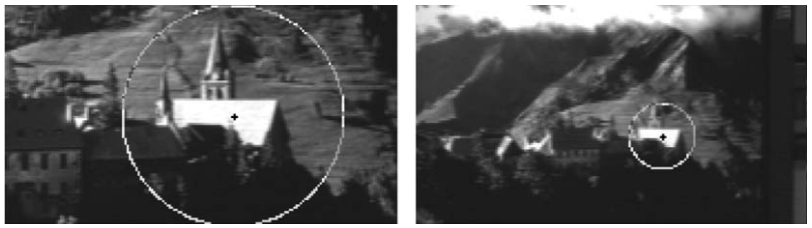
Generating putative correspondences



- Need to find **regions** and compare their **feature descriptors**

Feature detection with scale selection

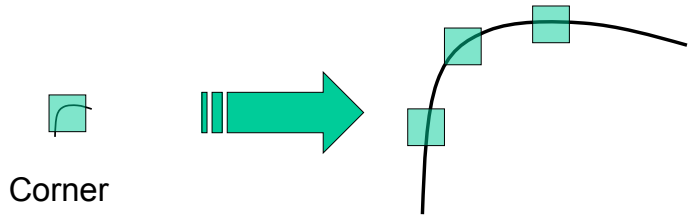
- We want to extract features with characteristic scale that matches the image transformation such as scaling and translation (a.k.a. covariance)



Matching regions across scales

Source: L. Lazebnik

Scaling

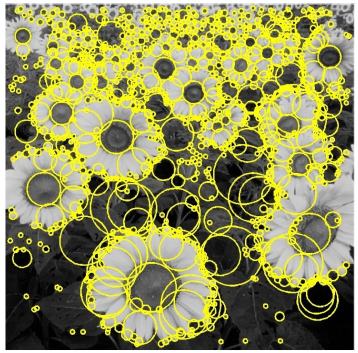


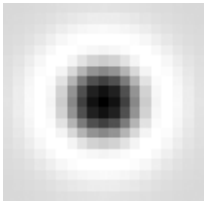
Corner detection is sensitive to the image scale!

Source: L. Lazebnik

Blob detection: basic idea

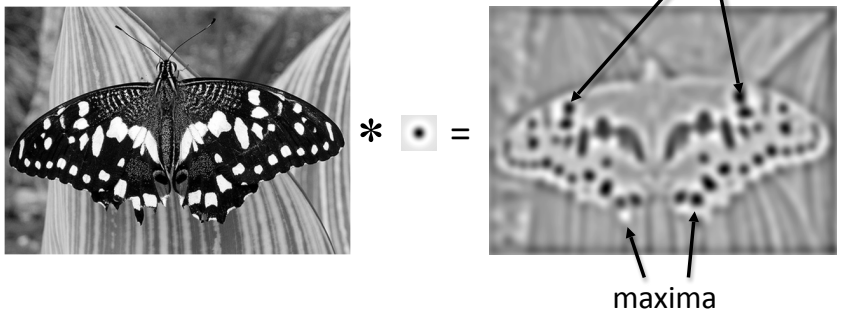
- Convolve the image with a “blob filter” at multiple scales
- Look for extrema (maxima or minima) of filter response in the resulting *scale space*
- This will give us a scale and space covariant detector





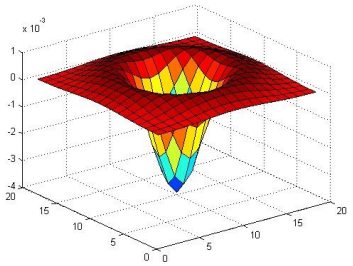
Blob detection: basic idea

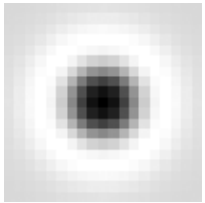
Find maxima ***and minima*** of blob filter response in space ***and scale***



Blob detection in 2D

Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D





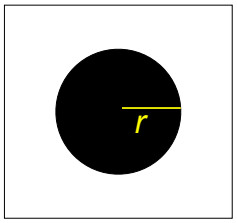
Scale-normalized: $$\nabla^2_{\text{norm}} g = \sigma^2 \left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \right)$$

Scale selection

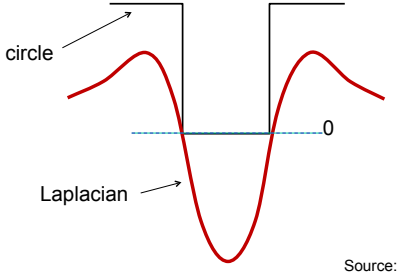
- At what scale does the Laplacian achieve a maximum response to a binary circle of radius r?
- To get maximum response, the zeros of the Laplacian have to be aligned with the circle
- The Laplacian is given by (up to scale):

$$(x^2 + y^2 - 2\sigma^2)\, e^{-(x^2+y^2)/2\sigma^2}$$

- Therefore, the maximum response occurs at $\sigma = r/\sqrt{2}$.

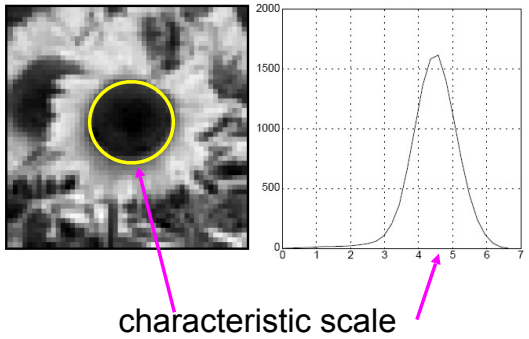


image



Characteristic scale

- We define the characteristic scale of a blob as the scale that produces peak of Laplacian response in the blob center



T. Lindeberg (1998). **"Feature detection with automatic scale selection."** *International Journal of Computer Vision* **30** (2): pp 77--116.

Scale-space blob detector

1. Convolve image with scale-normalized Laplacian at several scales

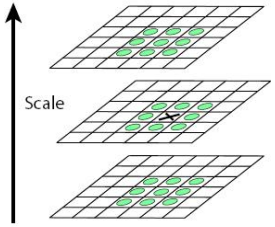
Scale-space blob detector: Example

Scale-space blob detector: Example

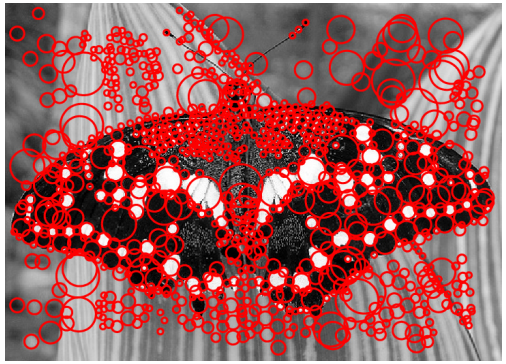
sigma = 11.9912

Scale-space blob detector

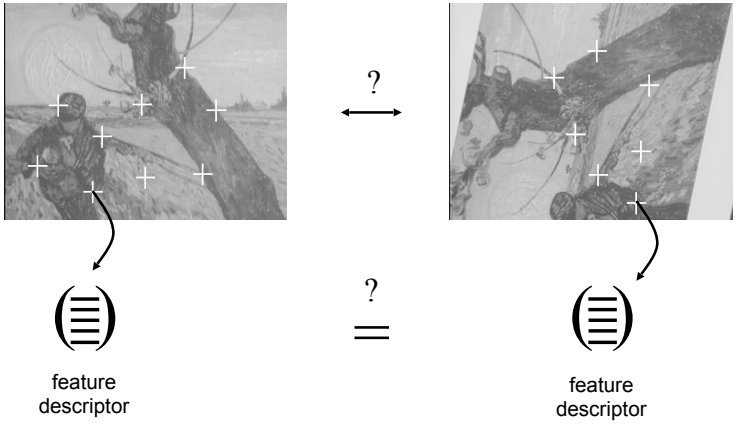
1. Convolve image with scale-normalized Laplacian at several scales
2. Find maxima of squared Laplacian response in scale-space



Scale-space blob detector: Example



Generating putative correspondences



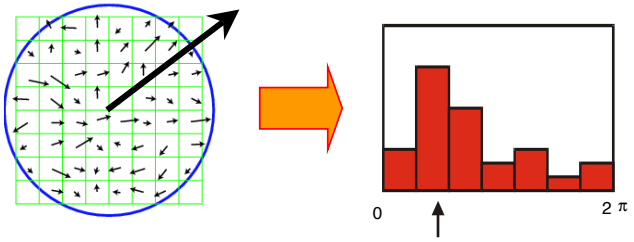
- Need to find **regions** and compare their **feature descriptors**

From feature detection to description

- Scaled and rotated versions of the same neighborhood will give rise to blobs that are related by the same transformation
- What to do if we want to compare the appearance of these image regions?
  - **Normalization**: transform these regions into same-size circles
  - **Problem**: rotational ambiguity

Eliminating rotation ambiguity

- To assign a unique orientation to circular image windows:
  - Create histogram of local gradient directions in the patch
  - Assign canonical orientation at peak of smoothed histogram

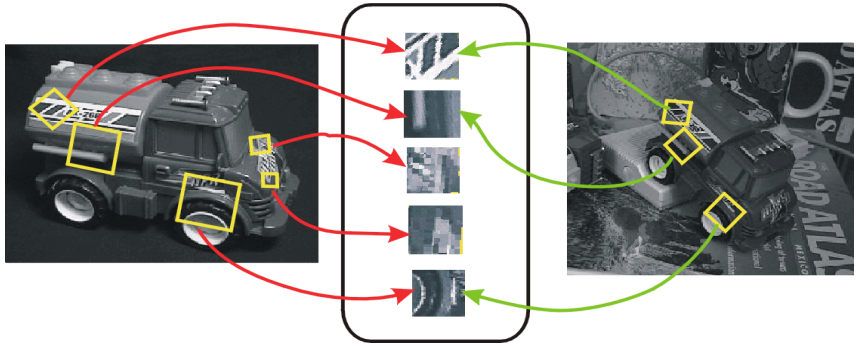


SIFT features

- Detected features with characteristic scales and orientations:

David G. Lowe. **"Distinctive image features from scale-invariant keypoints."** *IJCV* 60 (2), pp. 91-110, 2004.

From feature detection to description



how should we represent the patches?

Feature descriptors

- Simplest descriptor: vector of raw intensity values
- How to compare two such vectors?
  - Sum of squared differences (SSD) — this is a distance measure

$$\mathrm{SSD}(\mathbf{u},\mathbf{v}) = \sum_i (u_i - v_i)^2$$

    - Not invariant to intensity change

  - Normalized correlation — this is a similarity measure

$$\rho(\mathbf{u},\mathbf{v}) = \frac{(\mathbf{u}-\bar{\mathbf{u}})}{\|\mathbf{u}-\bar{\mathbf{u}}\|} \cdot \frac{(\mathbf{v}-\bar{\mathbf{v}})}{\|\mathbf{v}-\bar{\mathbf{v}}\|} = \frac{\sum_i (u_i-\bar{\mathbf{u}})(v_i-\bar{\mathbf{v}})}{\sqrt{\left(\sum_j (u_j-\bar{\mathbf{u}})^2\right)\left(\sum_j (v_j-\bar{\mathbf{v}})^2\right)}}$$

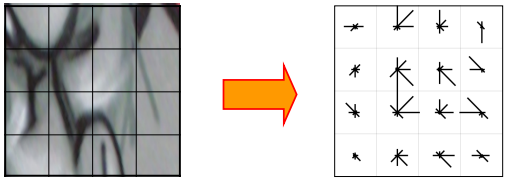
    - Invariant to affine (translation + scaling) intensity change

Problem with intensity vectors as descriptors

- Small deformations can affect the matching score a lot

Feature descriptors: SIFT

- Descriptor computation:
  - Divide patch into 4x4 sub-patches
  - Compute histogram of gradient orientations (8 reference angles) inside each sub-patch
  - Resulting descriptor: 4x4x8 = 128 dimensions



David G. Lowe. "Distinctive image features from scale-invariant keypoints." *IJCV* 60 (2), pp. 91-110, 2004.

Feature descriptors: SIFT

- Descriptor computation:
  - Divide patch into 4x4 sub-patches
  - Compute histogram of gradient orientations (8 reference angles) inside each sub-patch
  - Resulting descriptor: 4x4x8 = 128 dimensions
- Advantage over raw vectors of pixel values
  - Gradients less sensitive to illumination change
  - Pooling of gradients over the sub-patches achieves robustness to small shifts, but still preserves some spatial information

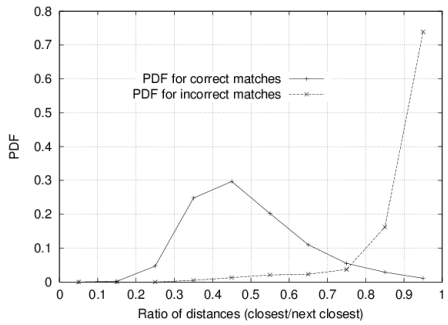
David G. Lowe. "Distinctive image features from scale-invariant keypoints." *IJCV* 60 (2), pp. 91-110, 2004.

Problem: Ambiguous putative matches

Source: Y. Furukawa

Rejection of unreliable matches

- How can we tell which putative matches are more reliable?
- Heuristic: compare distance of **nearest** neighbor to that of **second** nearest neighbor
  - Ratio of closest distance to second-closest distance will be *high* for features that are *not* distinctive



Threshold of 0.8 provides good separation

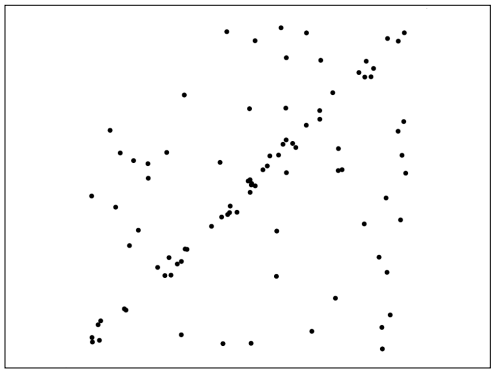
David G. Lowe. "Distinctive image features from scale-invariant keypoints." *IJCV* 60 (2), pp. 91-110, 2004.

RANSAC

- **Ran**dom **Sa**mple **C**onsensus
  - Choose a small subset of points uniformly at random
  - Fit a model to that subset
  - Find all remaining points that are "close" to the model and reject the rest as outliers
  - Do this many times and choose the best model
- For rigid transformation we can estimate the parameters of the **transformation**, e.g., rotation angle, scaling, translation, etc, from **putative** correspondence matches
- Lets see how RANSAC works for a simple example.

M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981.

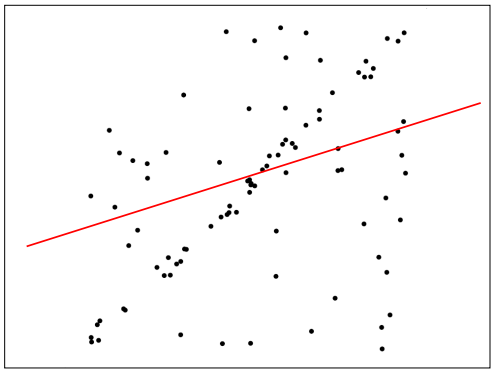
RANSAC for line fitting example



Source: R. Raguram

RANSAC for line fitting example

$$\min_{a,b} \sum_i (ax_i + b - y_i)^2$$

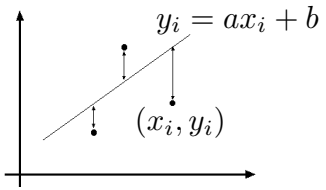


Least-squares fit

Source: R. Raguram

Least-squares line fitting

- Data: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Line equation: $y_i = ax_i + b$



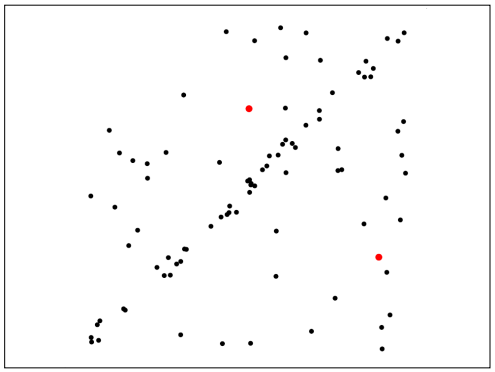
$$E = \sum_{i=1}^{n} (ax_i + b - y_i)^2$$

$$\frac{dE}{da} = \sum_{i=1}^{n} 2\,(ax_i + b - y_i)\,x_i = 0 \qquad \frac{dE}{db} = \sum_{i=1}^{n} 2\,(ax_i + b - y_i) = 0$$

$$a = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

$$b = \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

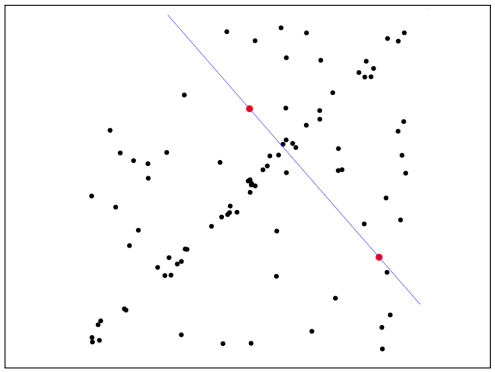
RANSAC for line fitting example



1. Randomly select minimal subset of points

Source: R. Raguram

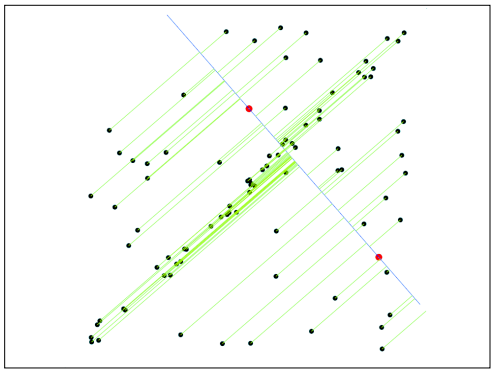
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model

Source: R. Raguram

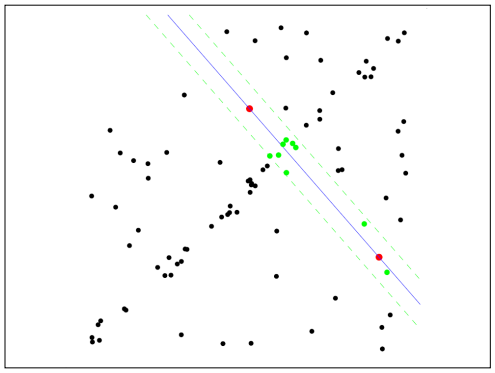
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function

Source: R. Raguram

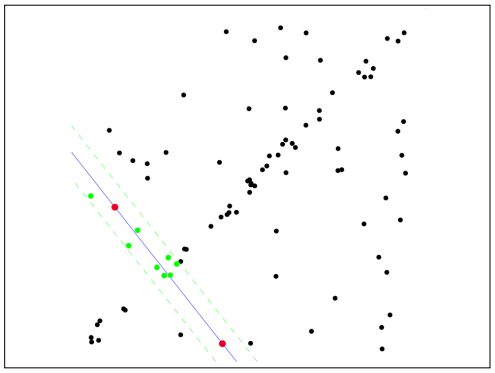
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model

Source: R. Raguram

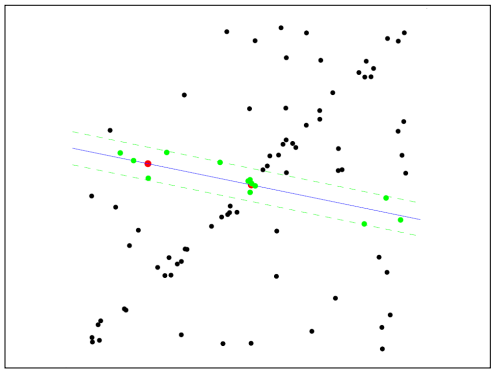
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example

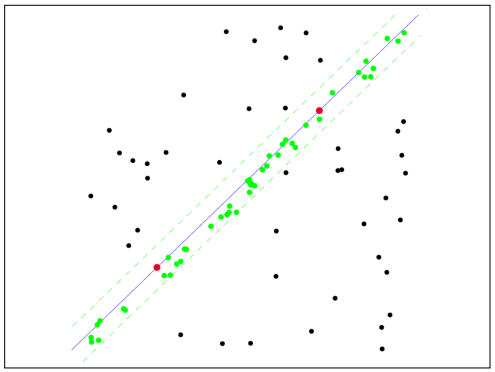


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example

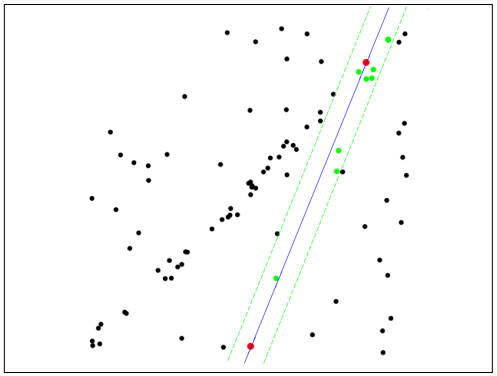
Uncontaminated sample



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

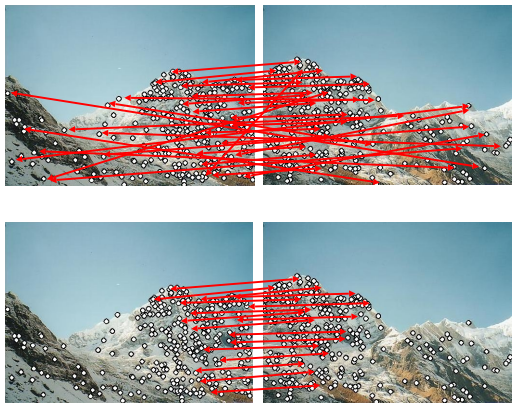
RANSAC for line fitting

Repeat ***N*** times:

- Draw ***s*** points uniformly at random
- Fit line to these ***s*** points
- Find ***inliers*** to this line among the remaining points (i.e., points whose distance from the line is less than ***t***)
- If there are ***d*** or more inliers and the number of inliers is higher than the *previous best*, accept the line and refit using all inliers.

RANSAC for image matching

- Given a number of putative matches, select **inliers** based on their agreement with an underlying **transformation**



- How do we represent image transformations?

Families of transformation

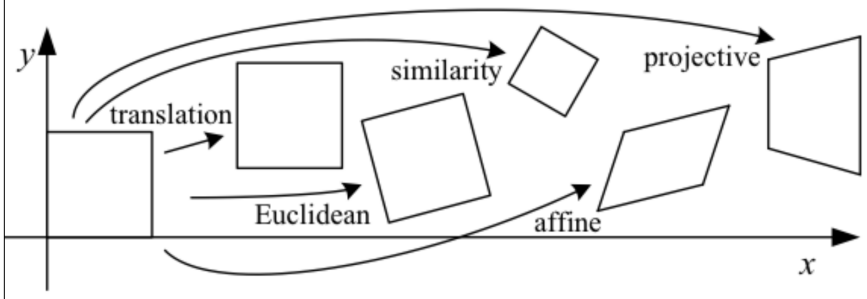


Figure 2.4 Basic set of 2D planar transformations.

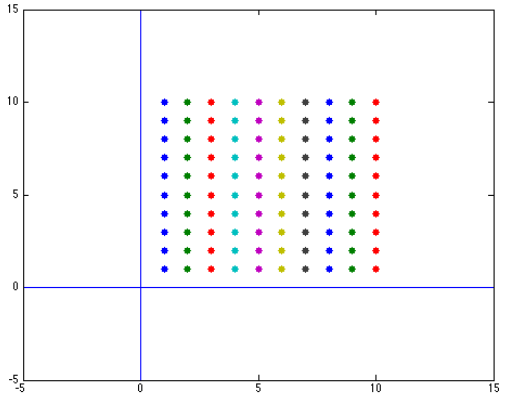
From Computer Vision: Algorithms and Applications, by Rick Szeliski

How do we move pixels?

- Think about moving coordinates, not pixels.

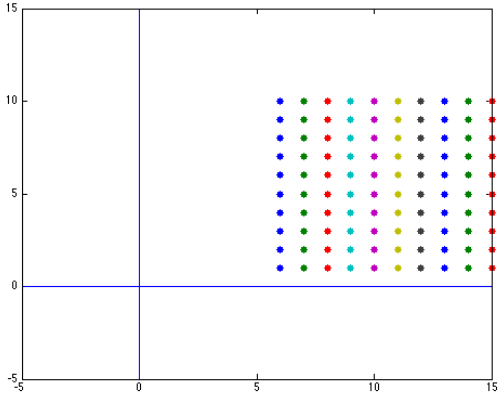
Points in 2D

```
figure(1);
[ox oy]=meshgrid(1:10,1:10);
plot(ox,oy,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



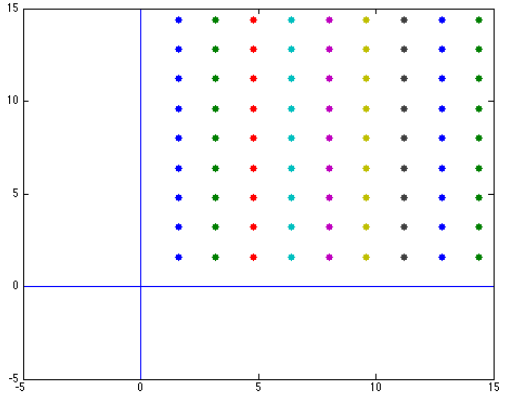
Translation

```
figure(2);
nx = ox + 5;
ny = oy;

plot(nx,ny,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



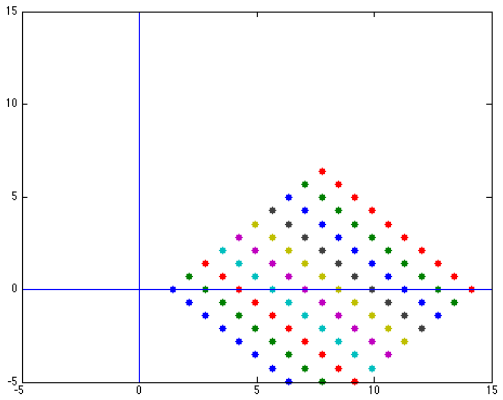
Scaling

```
figure(3);
oxy = [ox(:)'; oy(:)'];
A = [1.6 0; 0 1.6];
nxy = A*oxy;

nx = nxy(1,:);
ny = nxy(2,:);
nx = reshape(nx, [10 10]);
ny = reshape(ny, [10 10]);
plot(nx,ny,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



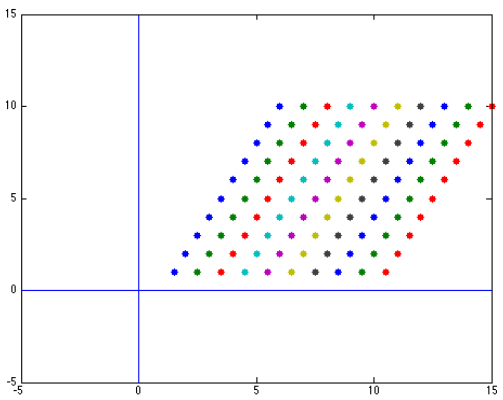
Rotation

```
figure(4);
oxy = [ox(:)'; oy(:)'];
A = [.707 .707;.707 -.707];
nxy = A*oxy;

nx = nxy(1,:);
ny = nxy(2,:);
nx = reshape(nx,[10 10]);
ny = reshape(ny,[10 10]);
plot(nx,ny,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



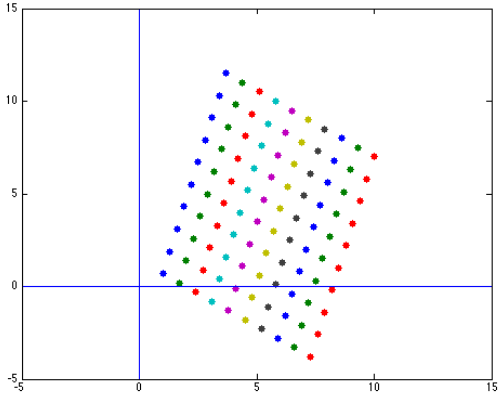
Shear

```
figure(5);
oxy = [ox(:)'; oy(:)'];
A = [1 0.5; 0 1];
nxy = A*oxy;

nx = nxy(1,:);
ny = nxy(2,:);
nx = reshape(nx,[10 10]);
ny = reshape(ny,[10 10]);
plot(nx,ny,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



Arbitrary linear transformation

```
figure(6);
oxy = [ox(:)'; oy(:)'];
A = [0.7 0.3; -0.5 1.2];
nxy = A*oxy;

nx = nxy(1,:);
ny = nxy(2,:);
nx = reshape(nx,[10 10]);
ny = reshape(ny,[10 10]);
plot(nx,ny,'.');
line([0 0],[-5 15]);
line([-5 15],[0 0]);
axis([-5 15 -5 15]);
```



Families of linear transforms

Identity:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Uniform scaling:

$$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

Scaling in x:

$$\begin{bmatrix} s_x & 0 \\ 0 & 1 \end{bmatrix}$$

Rotation by θ radians:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Shearing in x:

$$\begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix}$$

Arbitrary linear transformation:

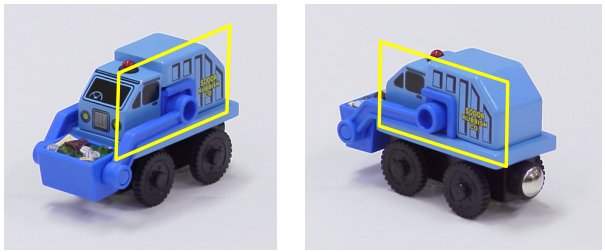
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Affine transformations

- Affine = linear + translation

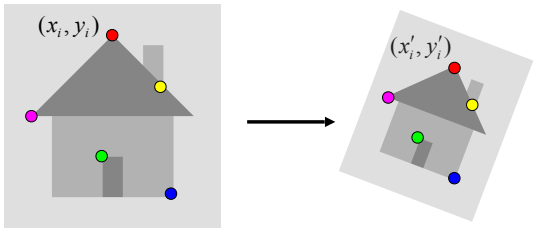
$$\mathbf{x} \rightarrow \mathbf{M}\mathbf{x} + \mathbf{t}$$

- Approximates viewpoint changes for roughly planar objects and roughly orthographic cameras
- Can be used to initialize fitting for more complex models



Fitting an affine transformation

- Assume we know the correspondences, how do we get the transformation?



$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

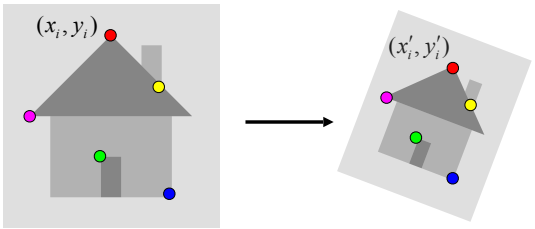
$$\mathbf{x}_i' = \mathbf{M}\mathbf{x}_i + \mathbf{t}$$

Want to find **M**, **t** to minimize

$$\sum_{i=1}^{n} || \mathbf{x}_i' - \mathbf{M}\mathbf{x}_i - \mathbf{t} ||^2$$

Fitting an affine transformation

- Assume we know the correspondences, how do we get the transformation?



$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

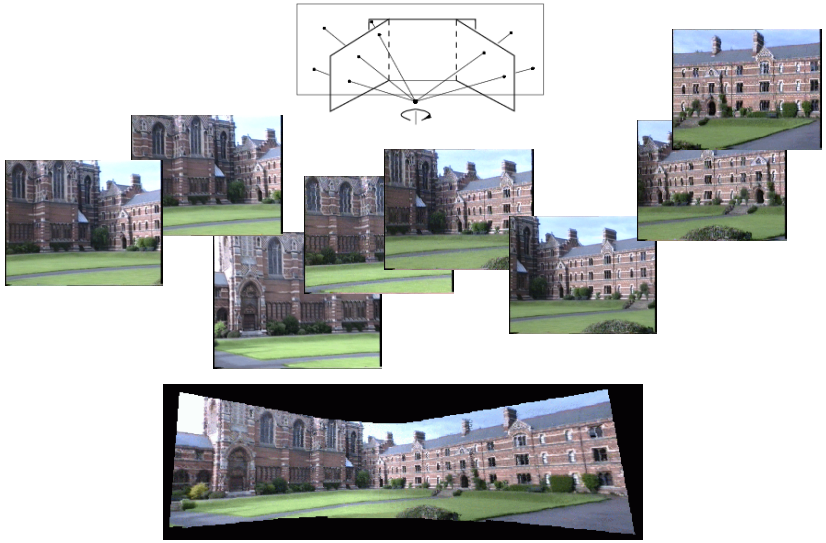
$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

Fitting an affine transformation

$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

- Linear system with six unknowns
- Each match gives us two linearly independent equations: need at least three to solve for the transformation parameters

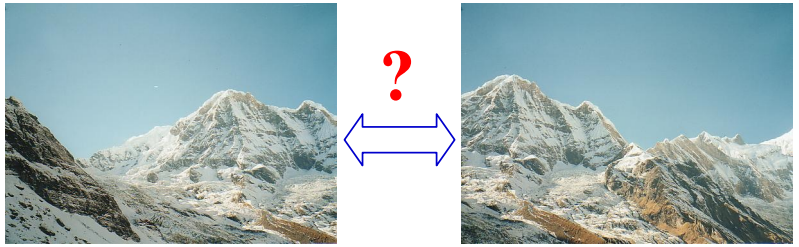
Application: Panorama stitching



Source: Hartley & Zisserman

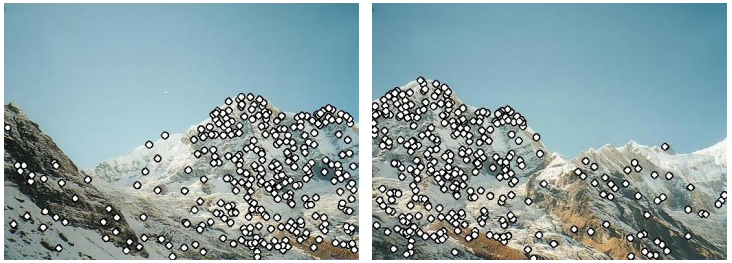
Panoramic stitching

- **Approach**
  - Local feature matching
  - RANSAC for alignment



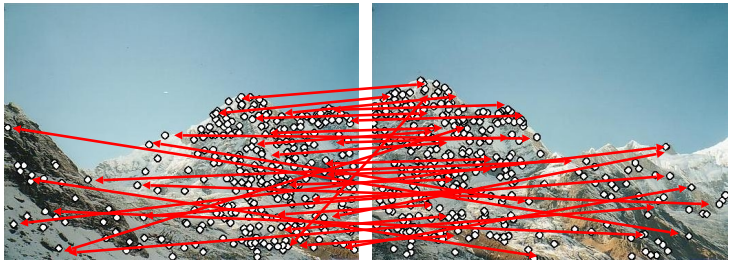
Panoramic stitching

Panoramic stitching



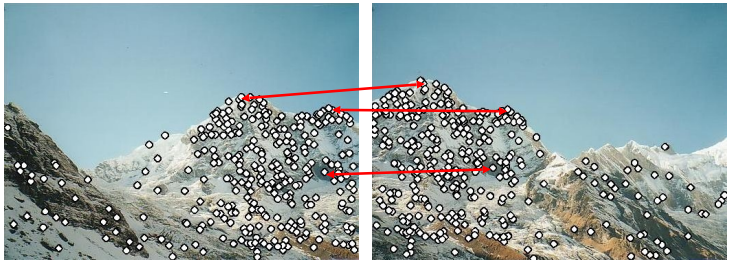
- Extract features
  - corner/blob detector

Panoramic stitching



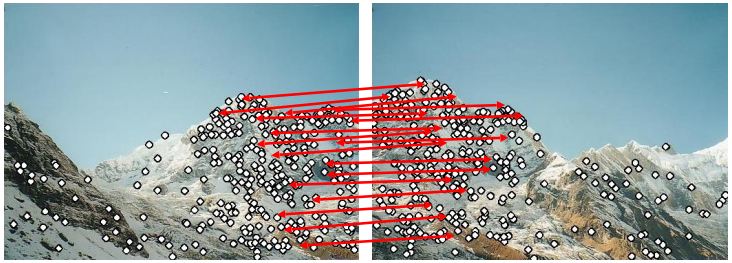
- Extract features
- Compute ***putative matches***

Panoramic stitching



- Extract features
- Compute ***putative matches***
- Loop:
  - ***Hypothesize*** transformation ***T***

Panoramic stitching



- Extract features
- Compute ***putative matches***
- Loop:
  - ***Hypothesize*** transformation ***T***
  - ***Verify*** transformation (search for other matches consistent with ***T***)

Warping images

To shift an image (dx, dy)

- Let **(ox, oy)** be the coordinates of a pixel in the original image with a particular appearance
- Let **(nx, ny)** be the new coordinates, i.e., where we want that pixel
- For each pixel
  - nx = ox + dx
  - ny = oy + dy
  - newIm(ny, nx) = im(oy, ox);

Mechanics of transformation

procedure *forwardWarp*$(f, \boldsymbol{h}, \textbf{out}\ g)$:

For every pixel $\boldsymbol{x}$ in $f(\boldsymbol{x})$

1. Compute the destination location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$.
2. Copy the pixel $f(\boldsymbol{x})$ to $g(\boldsymbol{x}')$.

From Computer Vision: Algorithms and Applications, by Rick Szeliski

Mechanics of transformation

procedure *forwardWarp*$(f, \boldsymbol{h}, \textbf{out}\ g)$:

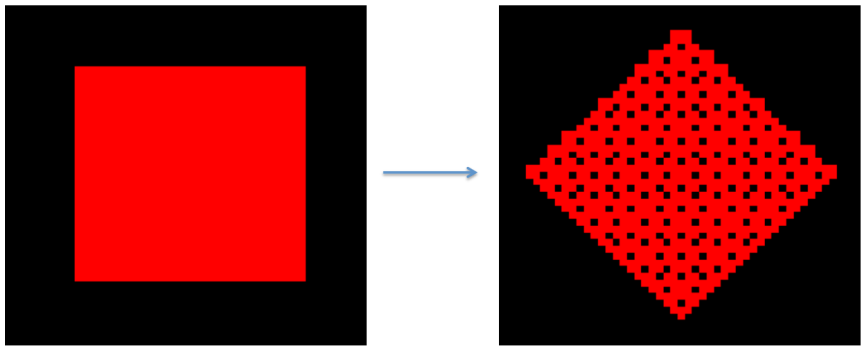
For every pixel $\boldsymbol{x}$ in $f(\boldsymbol{x})$

1. Compute the destination location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$.
2. Copy the pixel $f(\boldsymbol{x})$ to $g(\boldsymbol{x}')$.

- Problems with this transformation
  - Leaves gaps in the target image
  - Interpolation less intuitive

Example of forward warp

- Rotation by 45 degrees



Mechanics of transformation

procedure *inverseWarp*$(f, \boldsymbol{h}, \textbf{out}\ g)$:

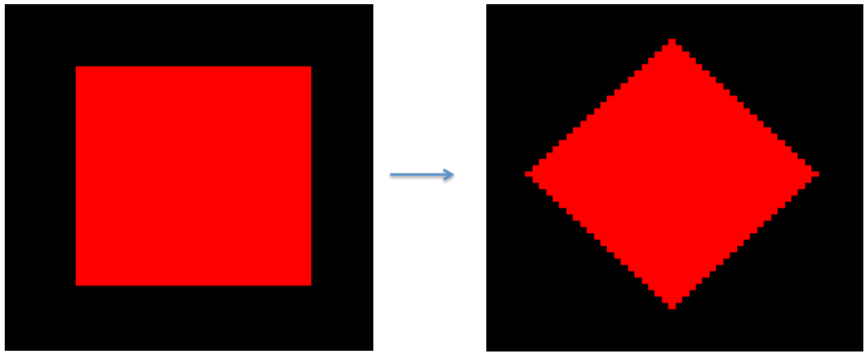
For every pixel $\boldsymbol{x}'$ in $g(\boldsymbol{x}')$

1. Compute the source location $\boldsymbol{x} = \hat{\boldsymbol{h}}(\boldsymbol{x}')$
2. Resample $f(\boldsymbol{x})$ at location $\boldsymbol{x}$ and copy to $g(\boldsymbol{x}')$

From Computer Vision: Algorithms and Applications, by Rick Szeliski

Example of inverse warp

- Rotation by 45 degrees



Panoramic stitching warping