About Assignments and Projects

- Any work related to image and vision computing is acceptable
- Presentation at the end of the semester
  - Pre-submission of demos, codes, and documents
  - PPT and DEMO at presentation
  - Each student has around 30 minutes
- Good work win exemption of final exam

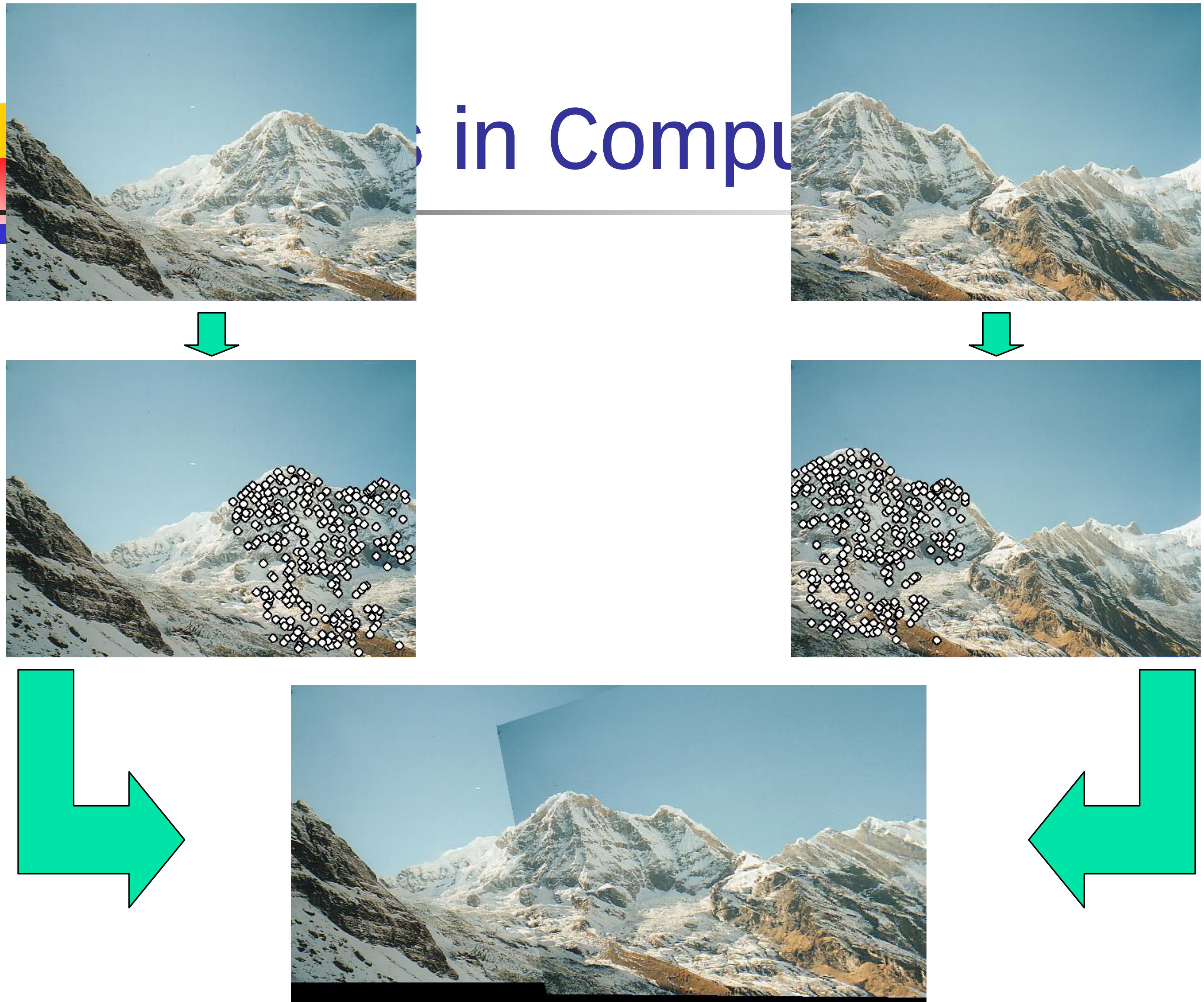
Image and Vision Computing
Image registration

Instructor: Chen Yisong
HCI & Multimedia Lab, Peking University

s in Compu

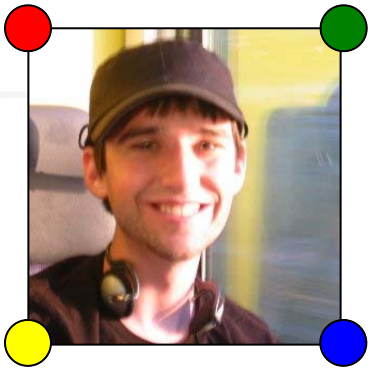
Image 1

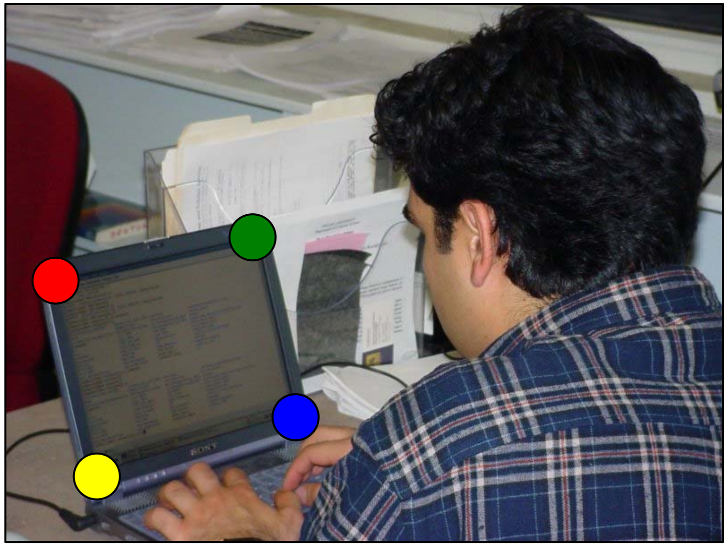
Image 2



Features in computer vision

- Compositing





This is your test image set

Linear Algebra Review

Matrices

$$A_{n\times m} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ a_{31} & a_{32} & \cdots & a_{3m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$$

Sum:

$$C_{n\times m} = A_{n\times m} + B_{n\times m}$$

$$c_{ij} = a_{ij} + b_{ij}$$

A and B must have the same dimensions

Example:

$$\begin{bmatrix} 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 8 & 7 \\ 4 & 6 \end{bmatrix}$$

Matrices

Product:

$$C_{n\times p} = A_{n\times m} B_{m\times p}$$

A and B must have compatible dimensions

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

$$A_{n\times n} B_{n\times n} \neq B_{n\times n} A_{n\times n}$$

Examples:

$$\begin{bmatrix} 2 & 5 \\ 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 17 & 29 \\ 19 & 11 \end{bmatrix}$$

$$\begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix} \cdot \begin{bmatrix} 2 & 5 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 18 & 32 \\ 17 & 10 \end{bmatrix}$$

Matrices

Transpose:

$$C_{m\times n} = A^{T}{}_{n\times m} \qquad (A+B)^T = A^T + B^T$$

$$c_{ij} = a_{ji} \qquad (AB)^T = B^T A^T$$

If $A^T = A$ A is symmetric

Examples:

$$\begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix}^T = \begin{bmatrix} 6 & 1 \\ 2 & 5 \end{bmatrix} \qquad \begin{bmatrix} 6 & 2 \\ 1 & 5 \\ 3 & 8 \end{bmatrix}^T = \begin{bmatrix} 6 & 1 & 3 \\ 2 & 5 & 8 \end{bmatrix}$$

Matrices

Determinant: *A must be square*

$$\det\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12}$$

$$\det\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

Example: $\det\begin{bmatrix} 2 & 5 \\ 3 & 1 \end{bmatrix} = 2 - 15 = -13$

Matrices

Inverse:

A must be square

$$A_{n\times n}A^{-1}{}_{n\times n} = A^{-1}{}_{n\times n}A_{n\times n} = I$$

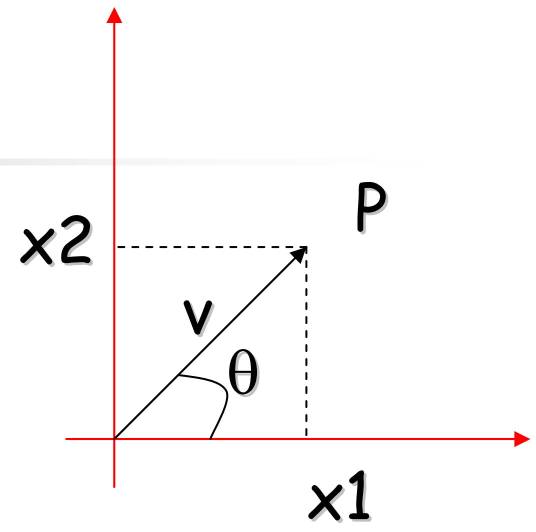
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} = \frac{1}{a_{11}a_{22} - a_{21}a_{12}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}$$

Example:

$$\begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix}^{-1} = \frac{1}{28} \begin{bmatrix} 5 & -2 \\ -1 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix}^{-1} . \begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix} = \frac{1}{28} \begin{bmatrix} 5 & -2 \\ -1 & 6 \end{bmatrix} . \begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix} = \frac{1}{28} \begin{bmatrix} 28 & 0 \\ 0 & 28 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

2D Vector



$$\mathbf{v} = (x_1, x_2)$$

Magnitude: $\|\mathbf{v}\| = \sqrt{x_1^{\,2} + x_2^{\,2}}$

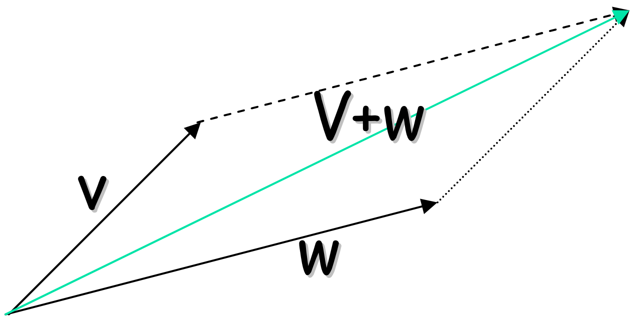
If $\|\mathbf{v}\| = 1$, $\mathbf{v}$ Is a UNIT vector

$$\frac{\mathbf{v}}{\|\mathbf{v}\|} = \left(\frac{x_1}{\|\mathbf{v}\|}, \frac{x_2}{\|\mathbf{v}\|} \right)$$ Is a unit vector

Orientation: $\theta = \tan^{-1}\left(\frac{x_2}{x_1} \right)$

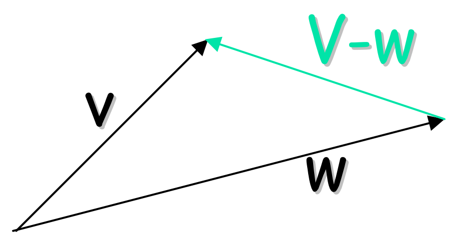
Vector Addition

$$\mathbf{v} + \mathbf{w} = (x_1, x_2) + (y_1, y_2) = (x_1 + y_1, x_2 + y_2)$$



Vector Subtraction

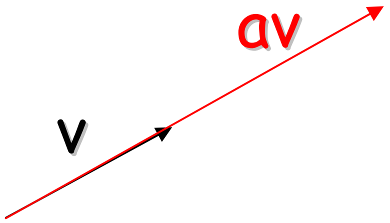
$$\mathbf{v}-\mathbf{w}=(x_1,x_2)-(y_1,y_2)=(x_1-y_1,x_2-y_2)$$



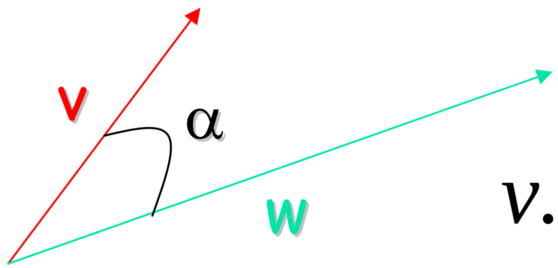
Note that: $\mathbf{w}+(\mathbf{v}-\mathbf{w})=\mathbf{v}$

Scaling (Product with a scalar)

$$a\mathbf{v} = a(x_1, x_2) = (ax_1, ax_2)$$



Inner (dot/scalar) Product



$$v.w = (x_1, x_2).(y_1, y_2) = x_1 y_1 + x_2 . y_2$$

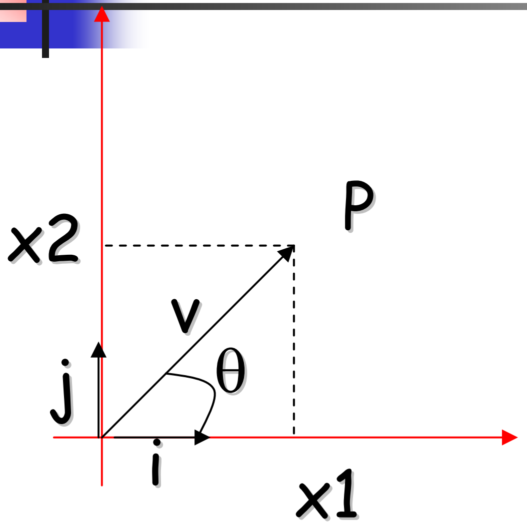
The inner product is a SCALAR!

$$v.w = (x_1, x_2).(y_1, y_2) = v^T w = w^T v = \| v \| \cdot \| w \| \cos \alpha$$

$$v.w = 0 \Leftrightarrow v \perp w$$

The inner product measures the similarity of two vectors

Orthonormal Basis (标准正交基)



$$\mathbf{i} = (1,0) \qquad \|\mathbf{i}\| = 1$$

$$\mathbf{j} = (0,1) \qquad \|\mathbf{j}\| = 1$$

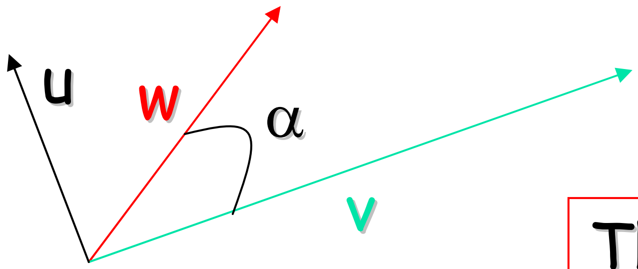
$$\mathbf{i} \cdot \mathbf{j} = 0$$

$$\mathbf{v} = (x_1, x_2) \qquad \mathbf{v} = x_1.\mathbf{i} + x_2.\mathbf{j}$$

$$\mathbf{v}.\mathbf{i} = (x_1.\mathbf{i} + x_2.\mathbf{j}).\mathbf{i} = x_1.1 + x_2.0 = x_1$$

$$\mathbf{v}.\mathbf{j} = (x_1.\mathbf{i} + x_2.\mathbf{j}).\mathbf{j} = x_1.0 + x_2.1 = x_2$$

Outer (cross/vector) Product



$$u = v \times w$$

The cross product is a VECTOR!

Magnitude: $\| \mathrm{u} \| = \| v.w \| = \| v \| \| w \| \sin \alpha$

Orientation:

$$u \perp v \Rightarrow u \cdot v = (v \times w) \cdot v = 0$$

$$u \perp w \Rightarrow u \cdot w = (v \times w) \cdot w = 0$$

Vector Product Computation

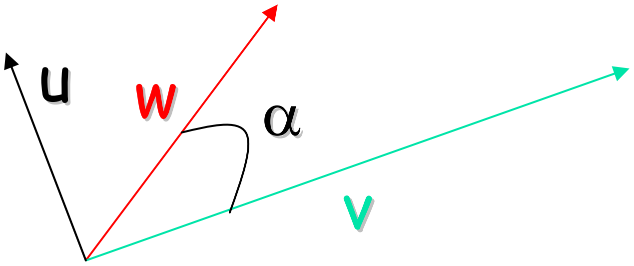
$\mathbf{i} = (1,0,0) \qquad \|\mathbf{i}\| = 1$

$\mathbf{j} = (0,1,0) \qquad \|\mathbf{j}\| = 1 \qquad \mathbf{i}\cdot\mathbf{j} = 0, \mathbf{i}\cdot\mathbf{k} = 0, \mathbf{j}\cdot\mathbf{k} = 0$

$\mathbf{k} = (0,0,1) \qquad \|\mathbf{k}\| = 1$

$$\mathbf{u} = \mathbf{v} \times \mathbf{w} = (x_1, x_2, x_3) \times (y_1, y_2, y_3)$$

$$\mathbf{u} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}$$



$$= (x_2 y_3 - x_3 y_2)\mathbf{i} + (x_3 y_1 - x_1 y_3)\mathbf{j} + (x_1 y_2 - x_2 y_1)\mathbf{k}$$

Cross Product

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - y_1 x_2 \end{pmatrix}$$

Every entry is a determinant of the two other entries

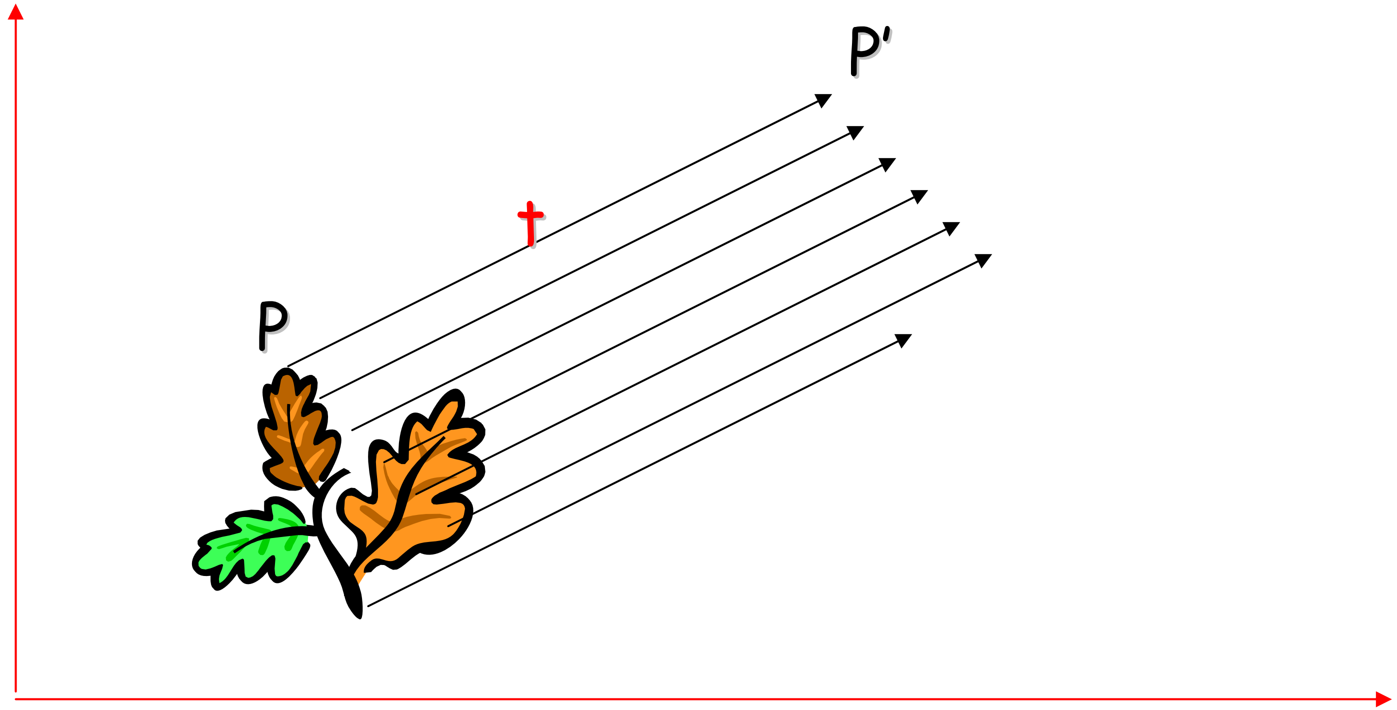
$$\vec{w} = \vec{u} \times \vec{v} \quad \Rightarrow \quad w^T u = w^T v = 0$$

Magnitude: $\|\mathrm{u}\| = \|v.w\| = \|v\|\|w\| \sin\alpha$

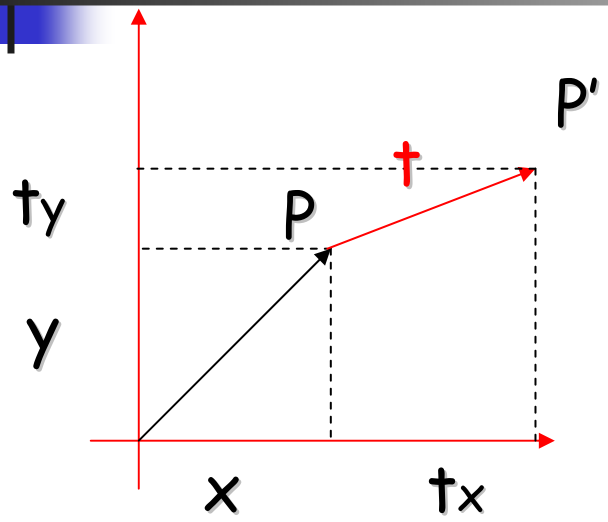
$\|\vec{w}\| =$ Area of parallelogram bounded by *u* and *v*

2D Geometrical Transformations

2D Translation



2D Translation Equation

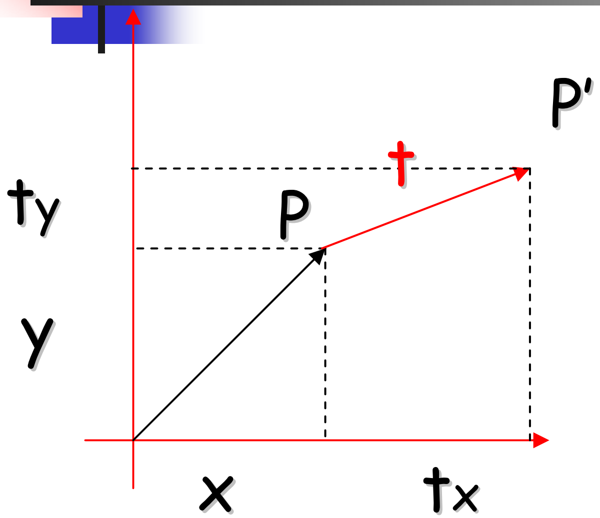


$$\mathbf{P} = (x, y)$$

$$\mathbf{t} = (t_x, t_y)$$

$$\mathbf{P}' = (x + t_x, y + t_y) = \mathbf{P} + \mathbf{t}$$

2D Translation using Matrices



$$\mathbf{P} = (x, y)$$

$$\mathbf{t} = (t_x, t_y)$$

$$\begin{array}{cc} & \begin{array}{ccc} & & \mathbf{t} \quad\quad \mathbf{P} \end{array} \\ \mathbf{P}' \rightarrow \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix} = & \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \end{array}$$

Homogeneous Coordinates (齐次坐标)

- Multiply the coordinates by a non-zero scalar and add an extra coordinate equal to that scalar. For example,

$$(x, y) \rightarrow (x \cdot z, y \cdot z, z) \quad z \neq 0$$

$$(x, y, z) \rightarrow (x \cdot w, y \cdot w, z \cdot w, w) \quad w \neq 0$$

- NOTE: If the scalar is 1, there is no need for the multiplication!

Example:

$$(2,3) \rightarrow (2,3,1) \sim (4,6,2) \sim (-4,-6,-2)\ldots$$

$$(3,-1,2) \rightarrow (3,-1,2,1) \sim (6,-2,4,2) \sim (-6,2,-4,-2)\ldots$$

Back to Cartesian Coordinates:

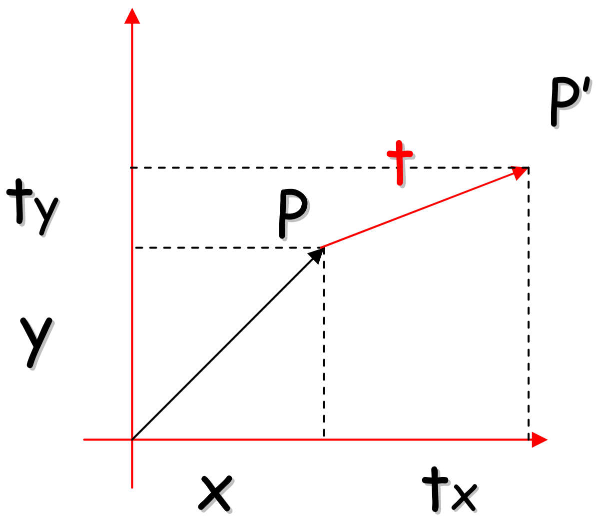
- Divide by the last coordinate and eliminate it. For example,

$$(x, y, z) \quad z \neq 0 \rightarrow (x/z, y/z)$$

$$(x, y, z, w) \quad w \neq 0 \rightarrow (x/w, y/w, z/w)$$

Question: What if *z=0* ?

2D Translation using Homogeneous Coordinates



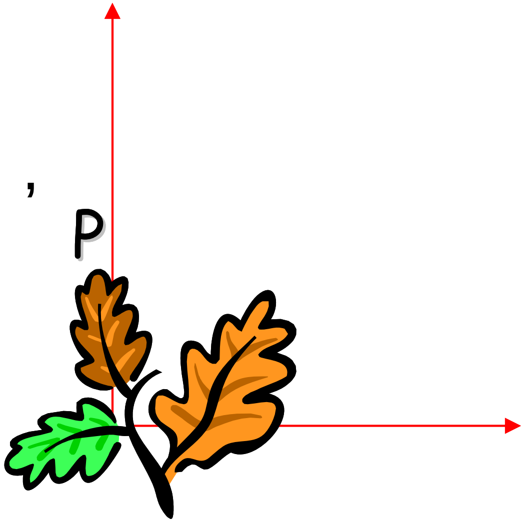
$$\mathbf{P} = (x, y) \rightarrow (x, y, 1)$$

$$\mathbf{t} = (t_x, t_y) \rightarrow (t_x, t_y, 1)$$

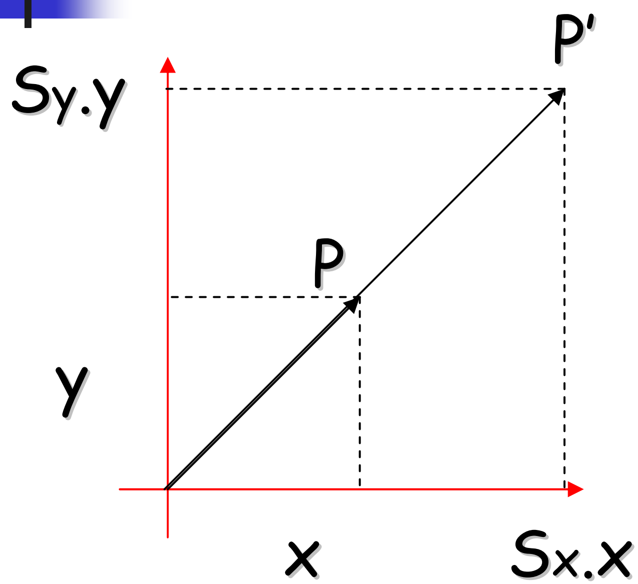
$$\mathbf{P}' \rightarrow \begin{bmatrix} x+t_x \\ y+t_y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & \overset{t}{t_x} \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}} \cdot \overset{P}{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}$$

$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{P}$$

Scaling



Scaling Equation



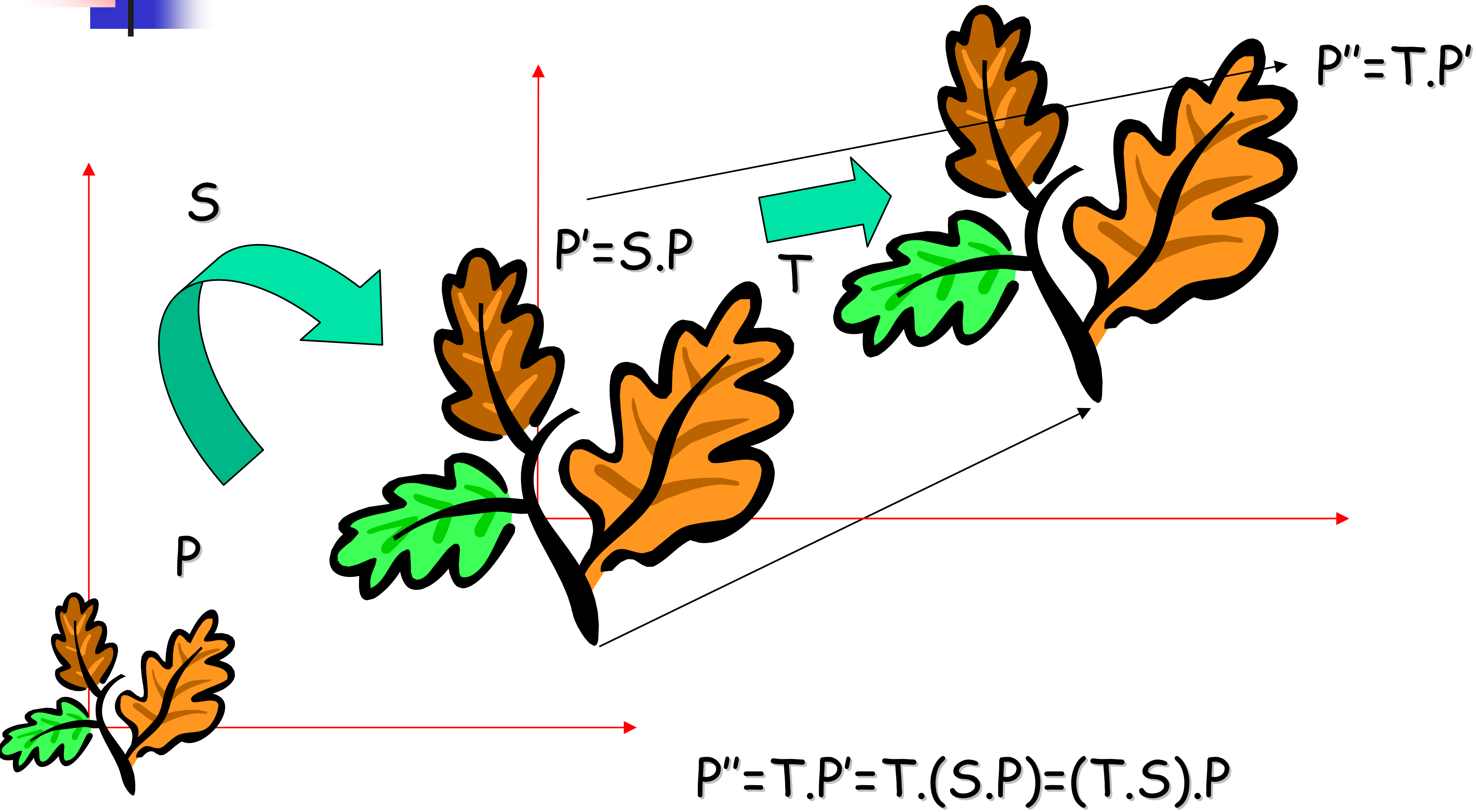
$$\mathbf{P} = (x, y) \rightarrow (x, y, 1)$$

$$\mathbf{P}' = (s_x x, s_y y) \rightarrow (s_x x, s_y y, 1)$$

$$\mathbf{P}' \rightarrow \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{S}} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

Scaling & Translating



Scaling & Translating

P''=T.P'=T.(S.P)=(T.S).P   Matrix product is associative

$$\mathbf{P''} = \mathbf{T} \cdot \mathbf{S} \cdot \mathbf{P} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \\ 1 \end{bmatrix}$$

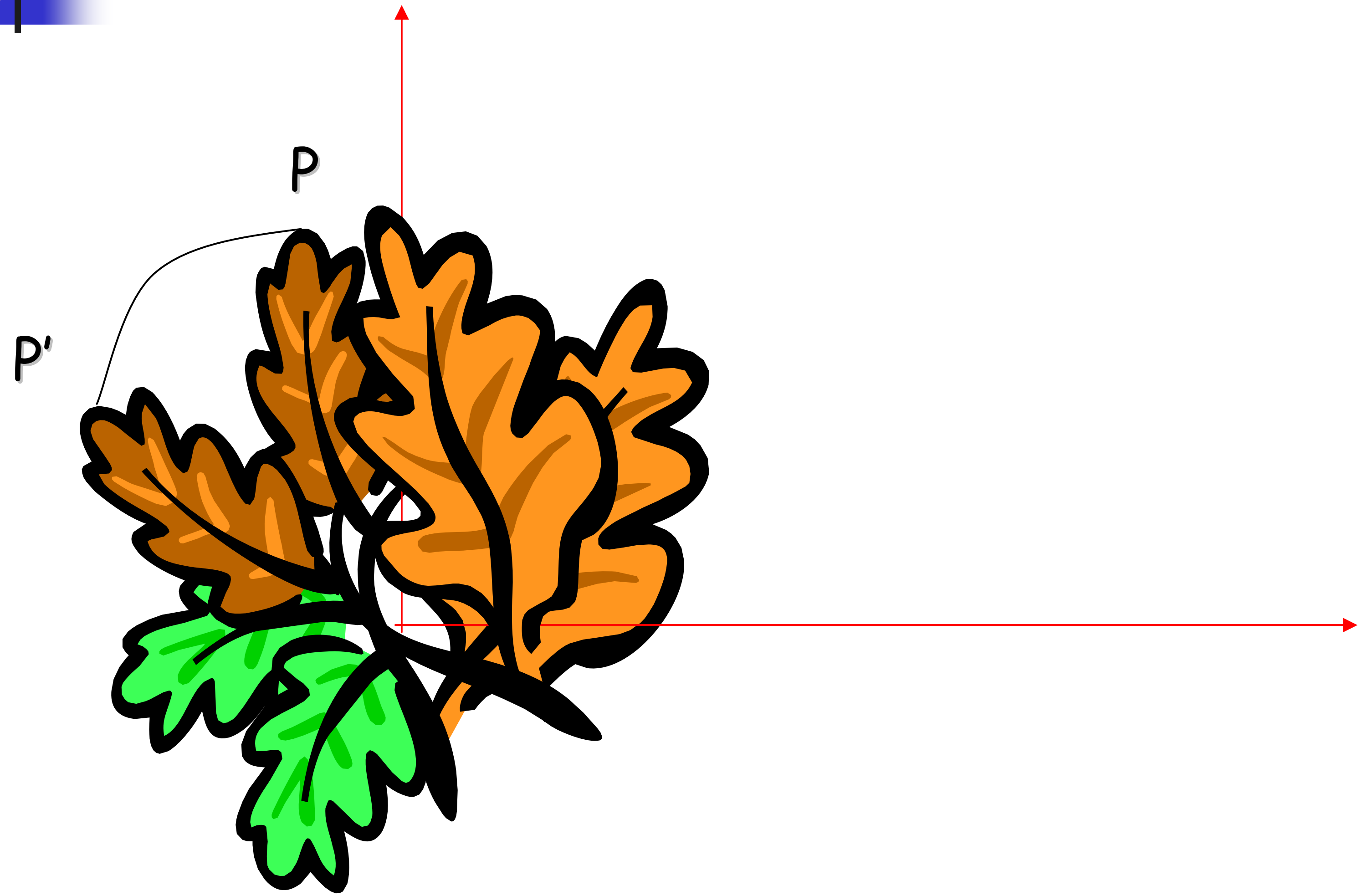
Translating & Scaling $\neq$ Scaling & Translating

S.(T.P) $\neq$(T.S).P Matrix product is NOT commutative

$$\mathbf{P}'' = \mathbf{S} \cdot \mathbf{T} \cdot \mathbf{P} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

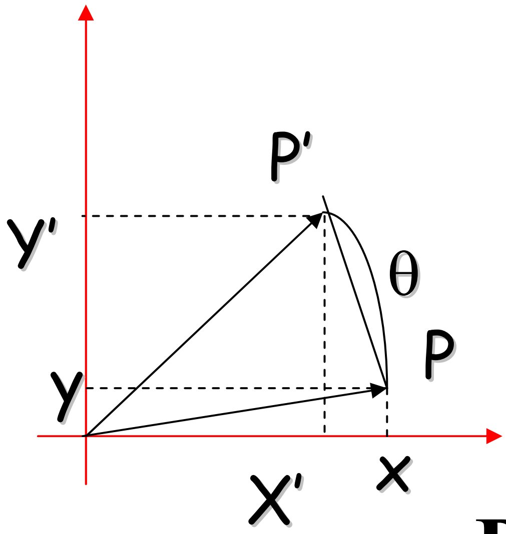
$$= \begin{bmatrix} s_x & 0 & s_x t_x \\ 0 & s_y & s_y t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + s_x t_x \\ s_y y + s_y t_y \\ 1 \end{bmatrix}$$

Rotation



Rotation Equations

Counter-clockwise rotation by an angle θ



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{P}' \rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{R.P}$$

Degrees of Freedom

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

R is 2x2 $\longrightarrow$ 4 elements

BUT! There is only 1 degree of freedom: θ

The 4 elements must satisfy the following constraints:

$$\mathbf{R} \cdot \mathbf{R}^{\mathrm{T}} = \mathbf{R}^{\mathrm{T}} \cdot \mathbf{R} = \mathbf{I}$$

$$\det(\mathbf{R}) = 1$$

Scaling, Translating & Rotating

Order matters!

$$P' = S.P$$
$$P'' = T.P' = (T.S).P$$
$$P''' = R.P'' = R.(T.S).P = (R.T.S).P$$

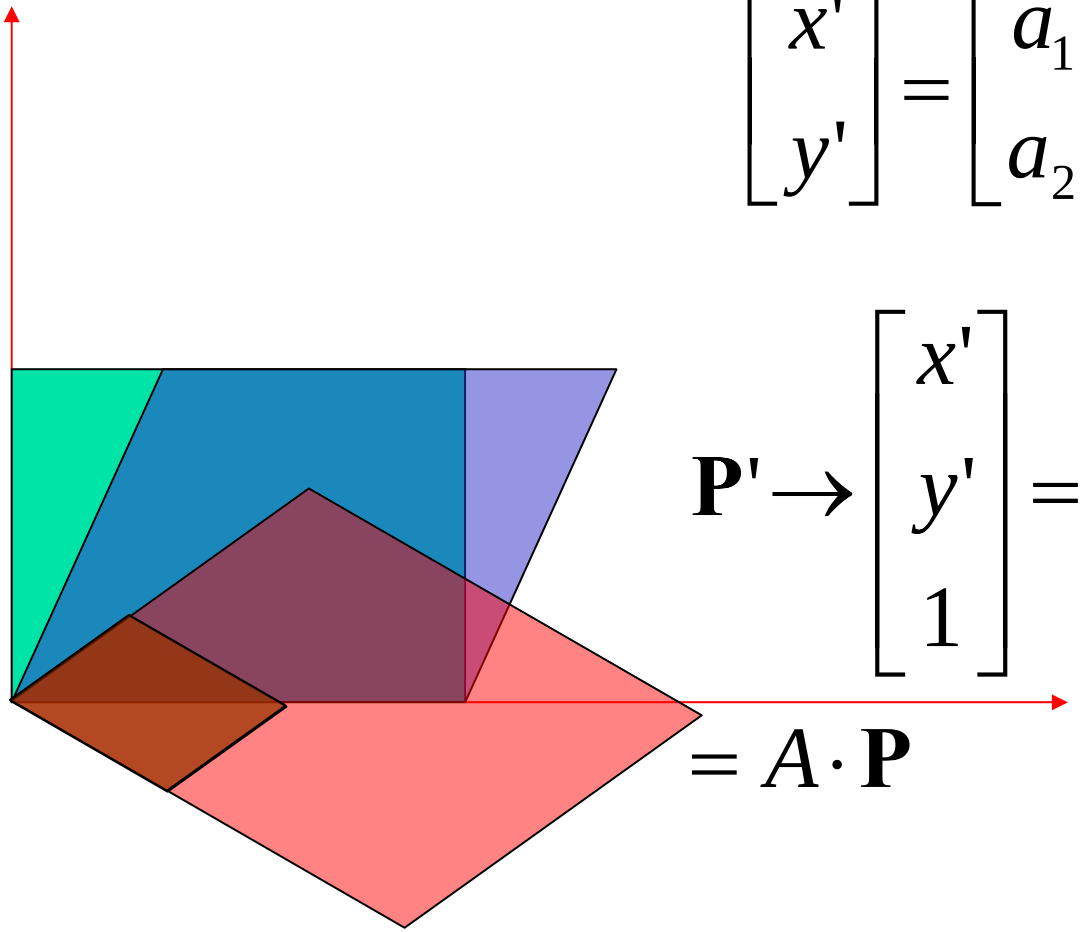
$$R.T.S \neq R.S.T \neq T.S.R \ldots$$

Affine Transformation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

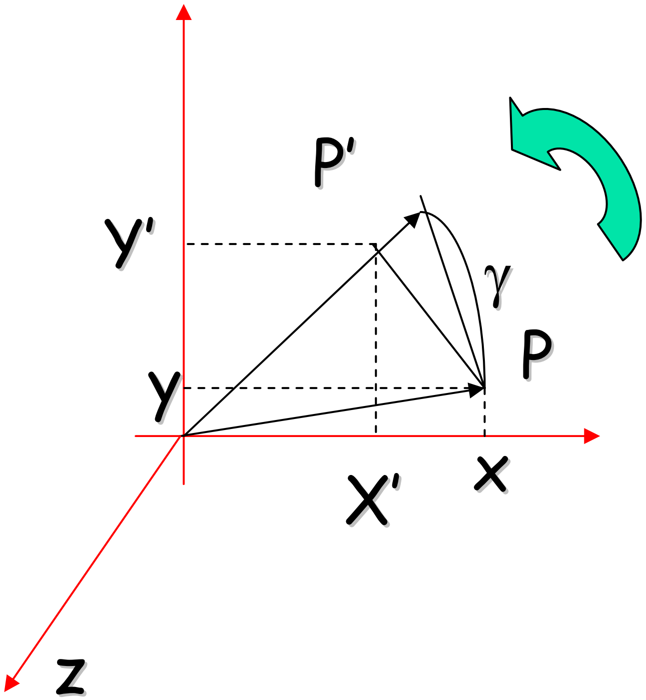
$$\mathbf{P}' \rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= A \cdot \mathbf{P}$$



3D Rotation of Points

Rotation around the coordinate axes, counter-clockwise:



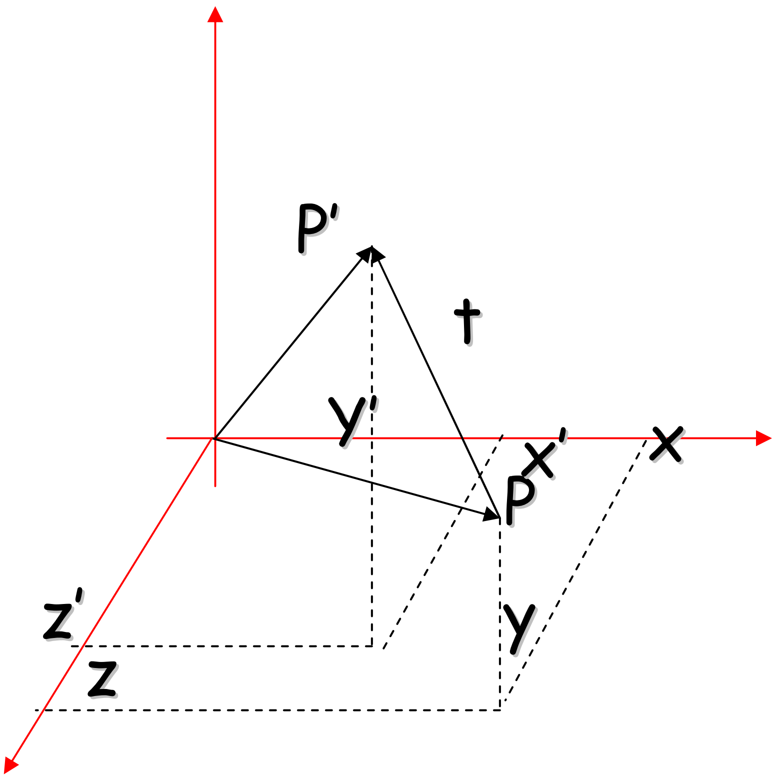
$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3D Translation of Points

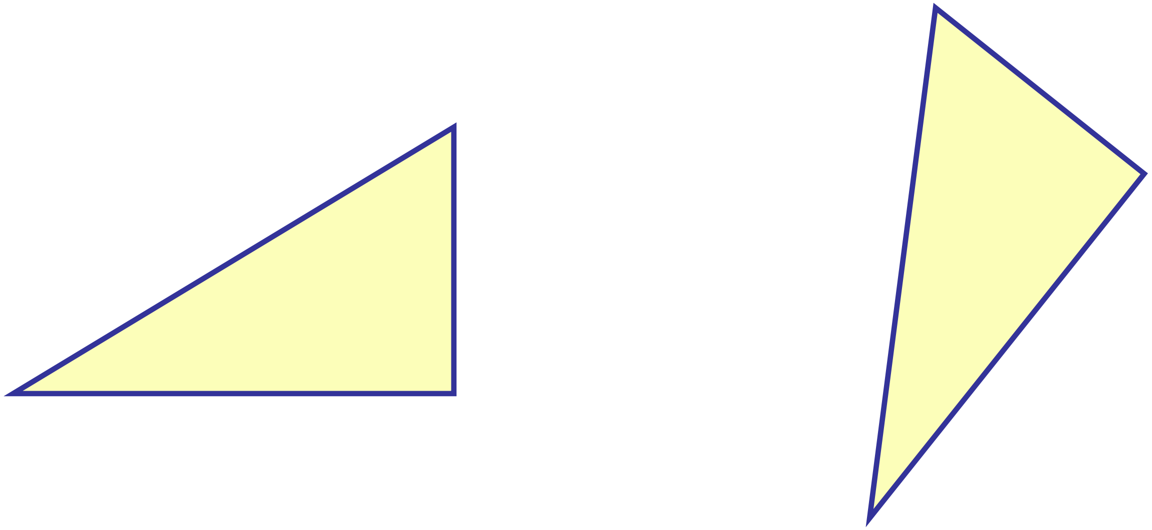
Translate by a vector $t=(t_x,t_y,t_x)^T$:



$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Euclidean Geometry

- Answers the question what objects have the same shape (= congruent)



Same shapes are related by rotation and translation

Euclidean Transformations (Isometries)

$$q = Rp + t$$

<u>Rotation</u>:

$$R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \quad R^T R = I, \quad \det R = 1$$

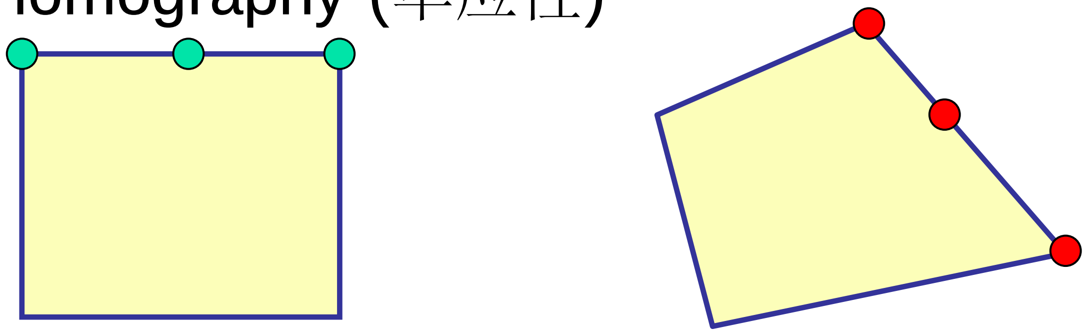
$$R = \begin{pmatrix} a & -b \\ b & a \end{pmatrix}, \quad a^2 + b^2 = 1, \quad R \in SO(2)$$

<u>Translation</u>:

$$\vec{t} = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Projective Transformations in a Plane (射影变换/透视变换)

- Projectivity (直射)
  - Mapping from points in plane to points in plane
  - 3 aligned points are mapped to 3 aligned points
- Also called
  - Collineation (共线，直射变换)
  - Homography (单应性)



Same shapes are related by a projective transformation

Projective Geometry

- Answers the question what appearances (projections) represent the same shape

Same shapes are related by a projective transformation

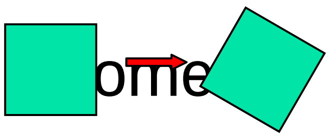
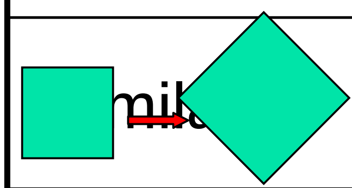
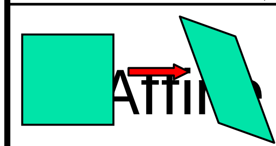
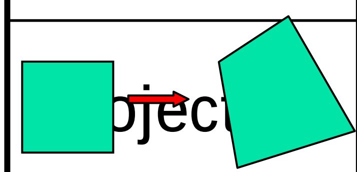
Hierarchy of Transformations

- Isometry (Euclidean), $\begin{pmatrix} R & \vec{t} \\ 0 & 1 \end{pmatrix}$
- Similarity, $\begin{pmatrix} sR & \vec{t} \\ 0 & 1 \end{pmatrix}, \quad sR = \begin{pmatrix} a & -b \\ b & a \end{pmatrix}$
- Affine, $\begin{pmatrix} A & \vec{t} \\ 0 & 1 \end{pmatrix}, \quad A \in GL(2)$ general linear
- Projective, $H \in GL(3): \quad \alpha q = Hp, \quad \alpha \neq 0$

Special Projectivities

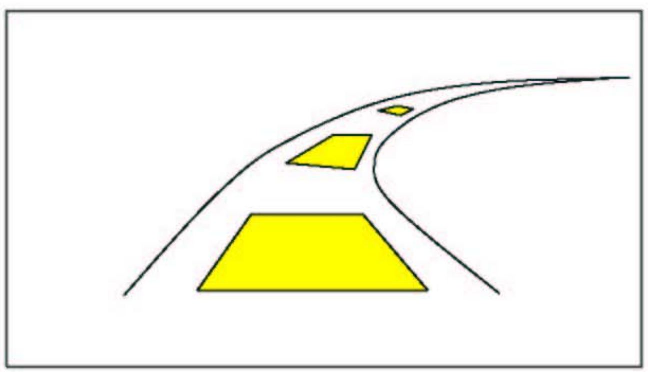
Invariants

Invariants(不变量)

	Length Area	Angle Shape	Parallelism Area ratio	Collinearity Cross-ratio
 ome				
 mil				
 Affi				
 ojec				

Example of Application

- Robot going down the road
- Large squares painted on the road to make it easier
- Find road shape without perspective distortion from image
  - Use corners of squares: coordinates of 4 points allow us to compute matrix **H**
  - Then use matrix **H** to compute 3D road shape



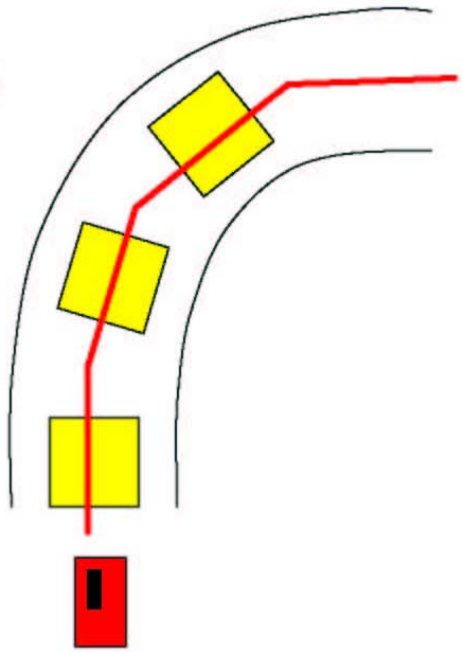


Image Alignment and Stitching

[Szeliski & Shum, SIGGRAPH'97]

[Szeliski, MSR-TR-2004-92]

Wide-angle Imaging

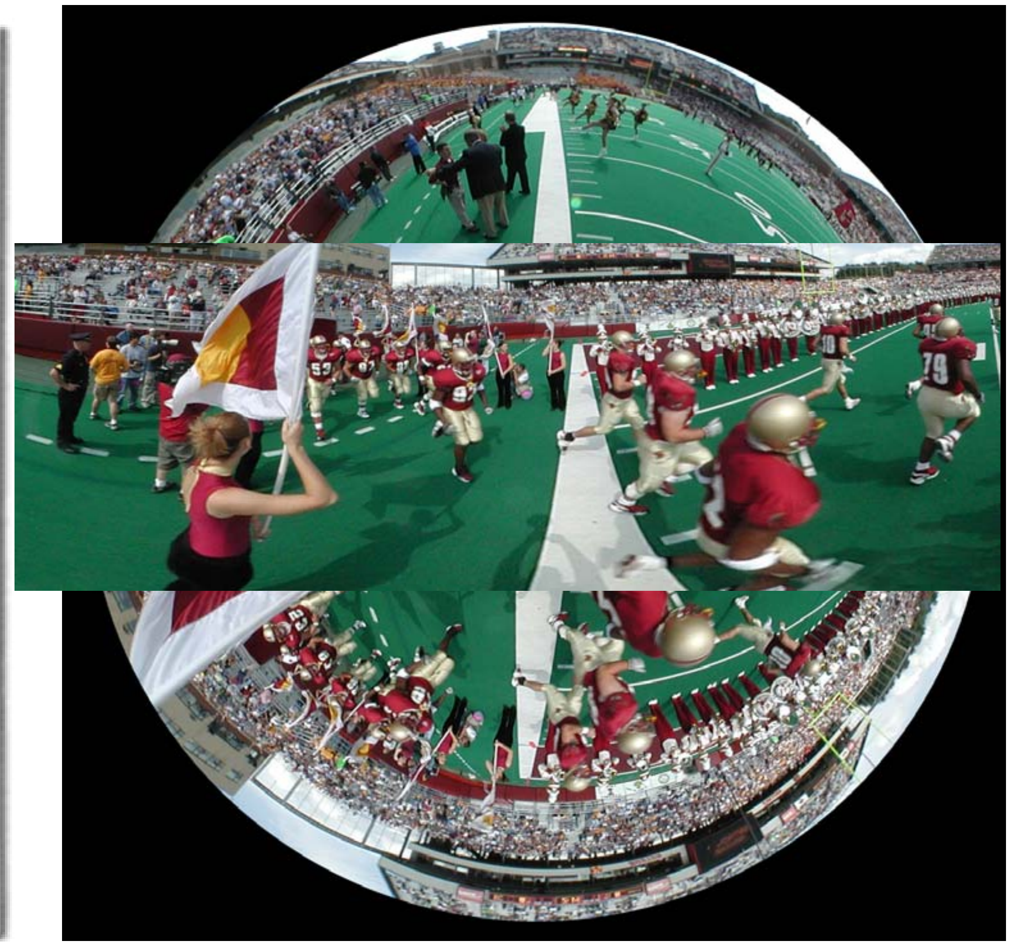
- How do you increase the field of view?

Wide-angle Imaging
Fisheye cameras

f=18mm

f=10mm

Wide-angle Imaging
Catadioptric sensor



Remote Reality

Contents

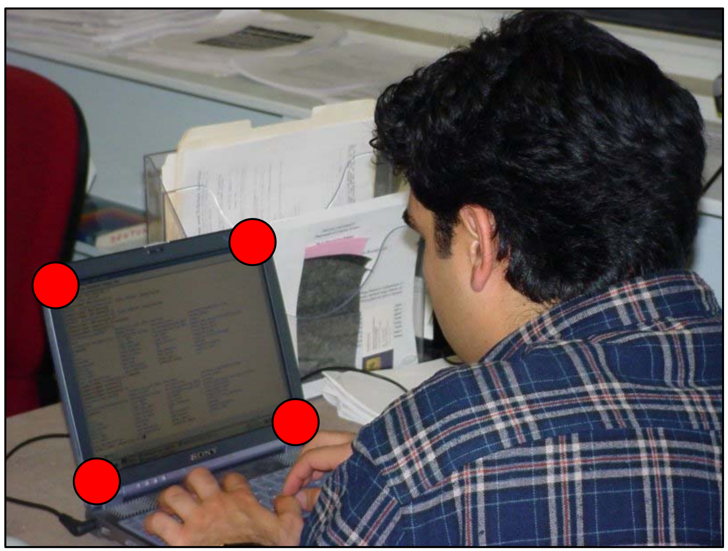
- Image alignment and stitching
- motion models
- direct alignment
- point-based alignment
- complete mosaics (global alignment)
- ghost and parallax removal
- compositing and blending

Readings

- Szeliski & Shum, SIGGRAPH'97
- Szeliski, Image Alignment and Stitching, MSR-TR-2004-92
- Bergen *et al*, Hierarchical model-based motion estimation, ECCV'92
- Shi & Tomasi, Good Features to Track, CVPR'94
- Recognizing Panoramas, Brown & Lowe, ICCV'2003
- Multi-image matching using multi-scale oriented patches, Brown, Szeliski, and Winder, CVPR'2005

Example

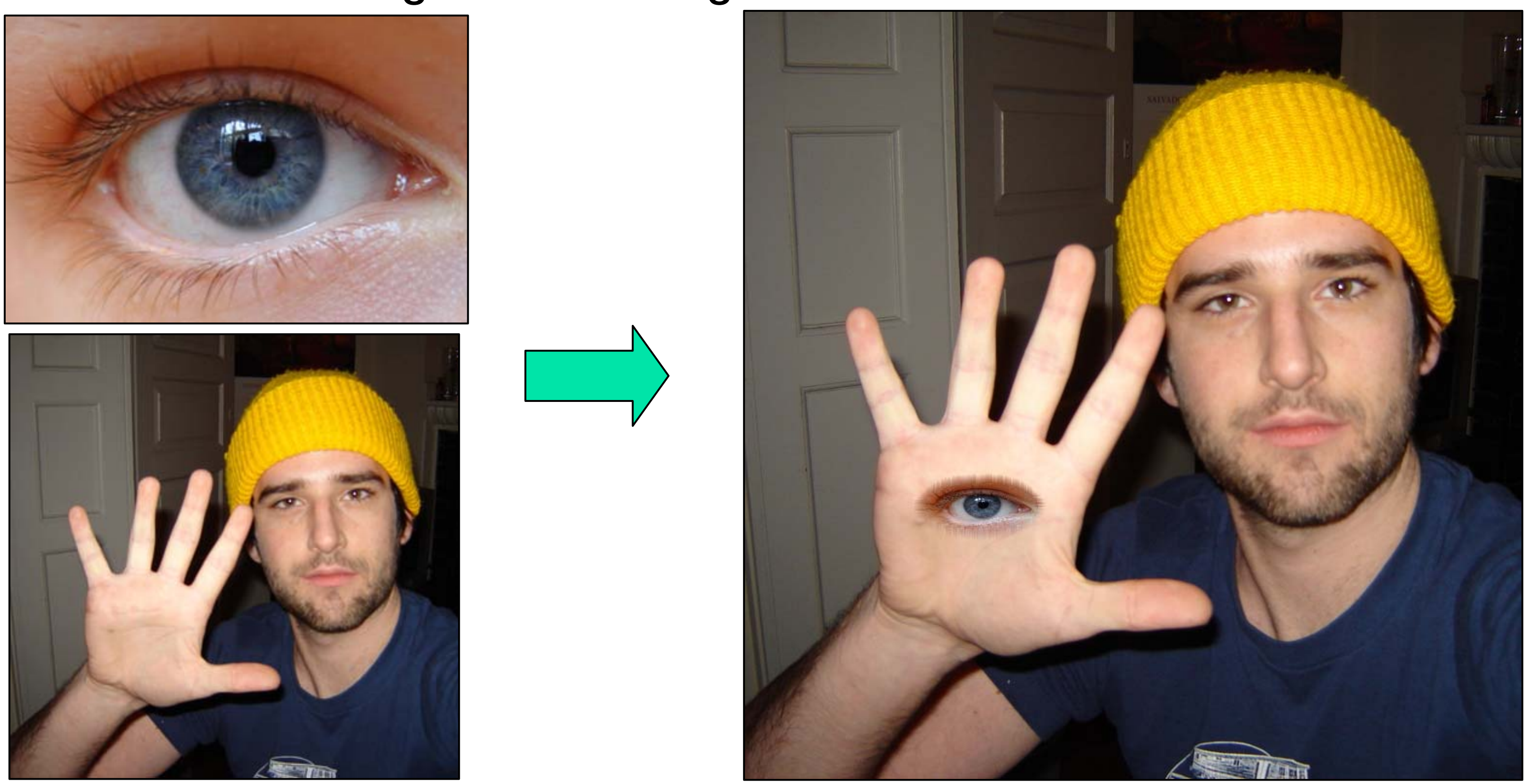
- Compositing



This is your test image set

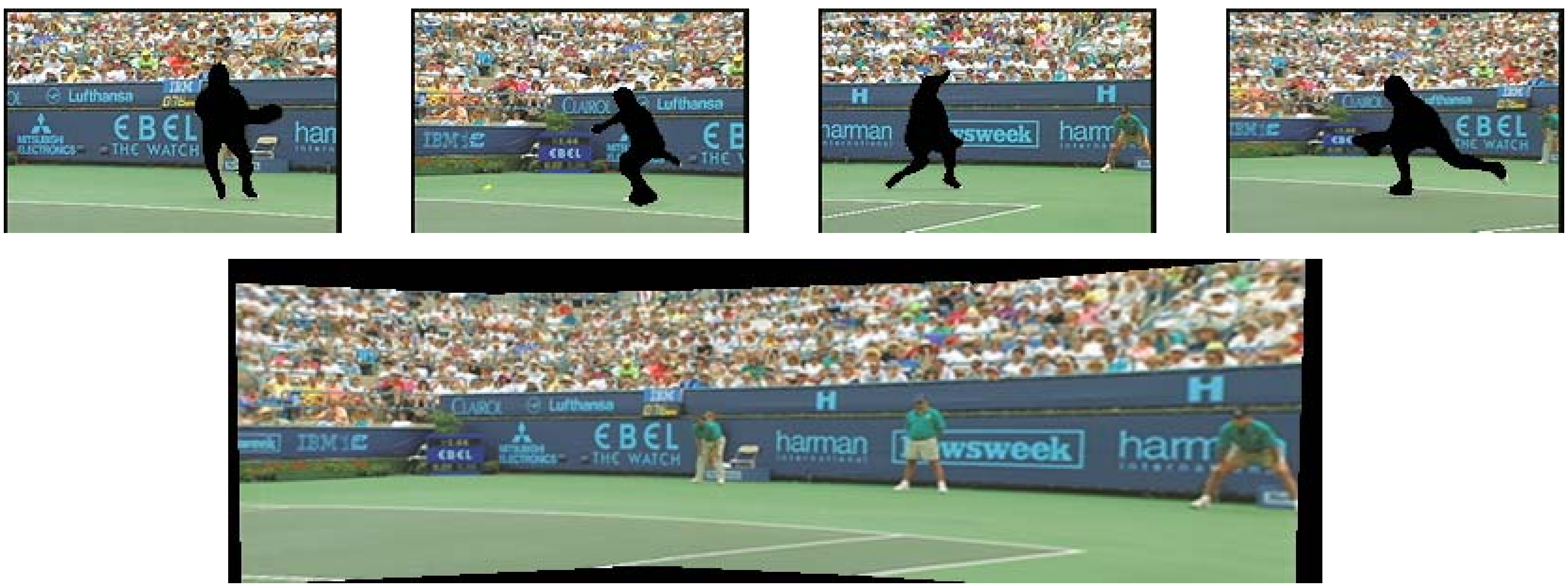
Example

- Composite
  - Need not be rectangular
  - Masking and Blending



Mosaics for Video Coding

- Convert masked images into a background sprite for content-based coding



Mosaic Examples

- http://www.panoramas.dk/

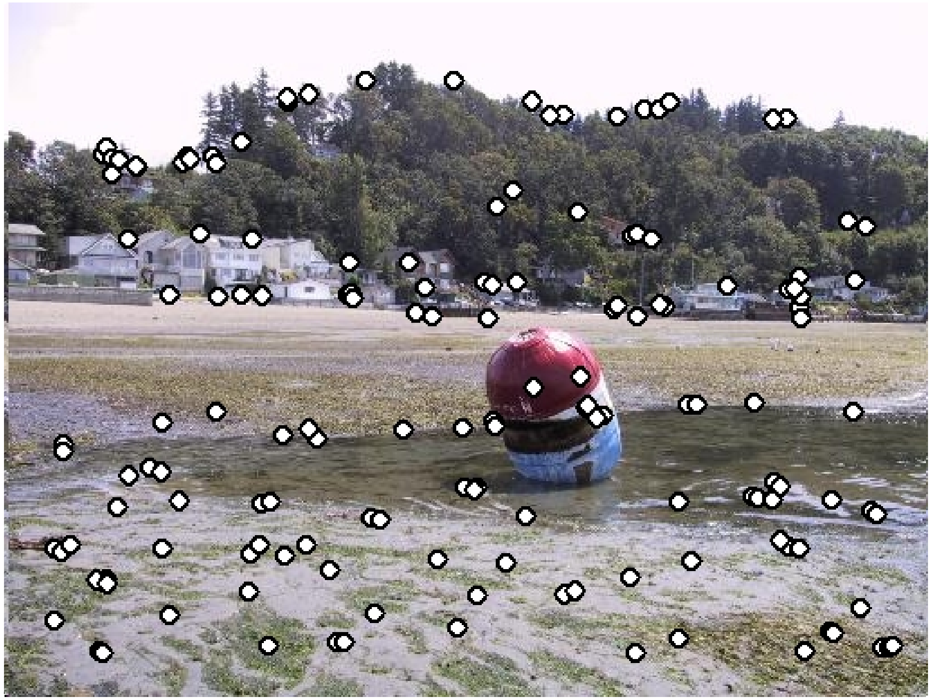
pixel-based image alignment

Establishing correspondences

1. Direct method:
   - Use generalization of affine motion model [Szeliski & Shum '97]
2. Feature-based method
   - Use Shi-Tomasi tracker after initial rough alignment
     [Lowe ICCV'99; Schmid ICCV'98, Brown&Lowe ICCV'2003]
   - Compute $\boldsymbol{R}$ from correspondences (absolute orientation)

Feature irregularities

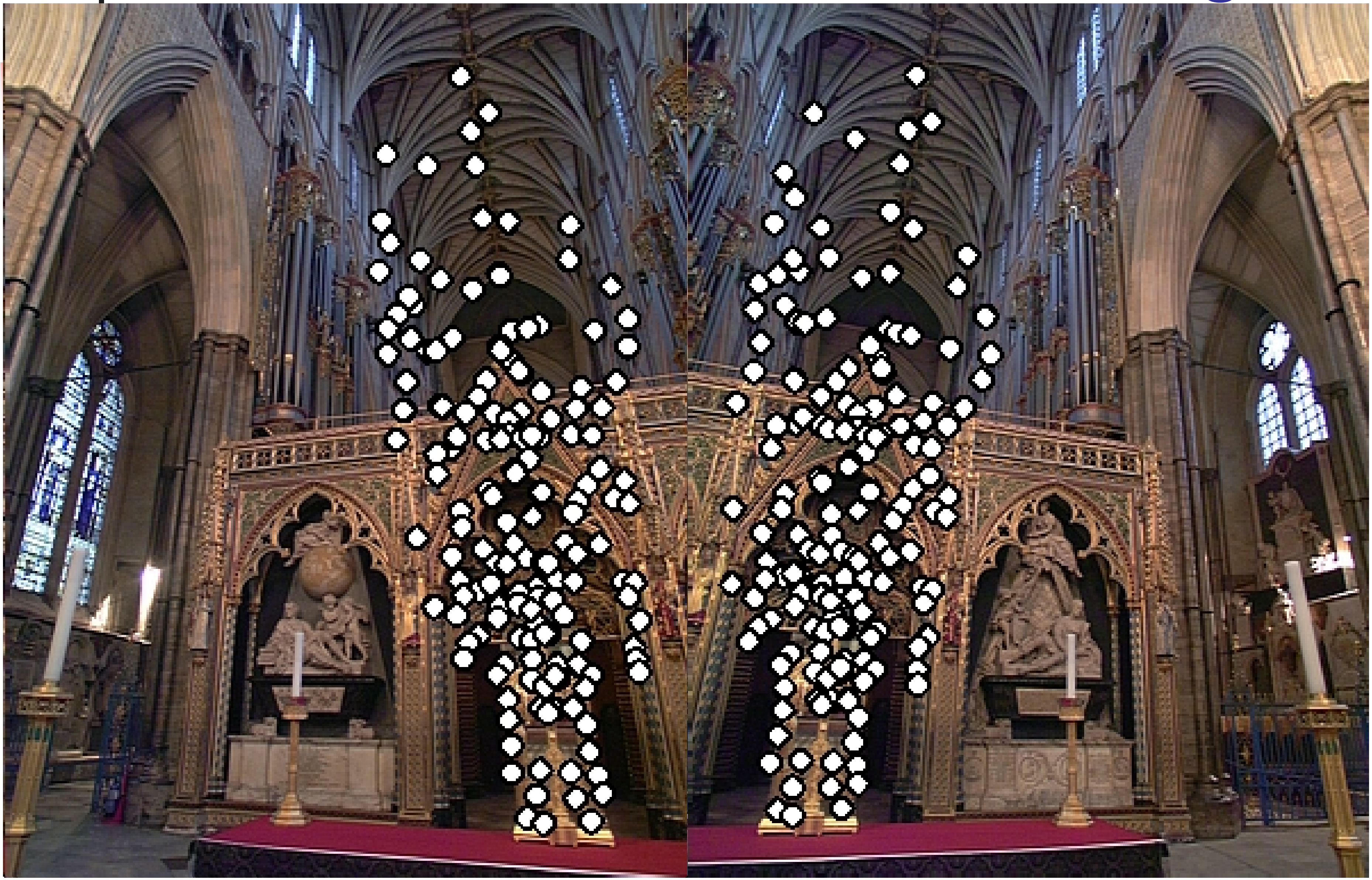
- Distribute points evenly over the image



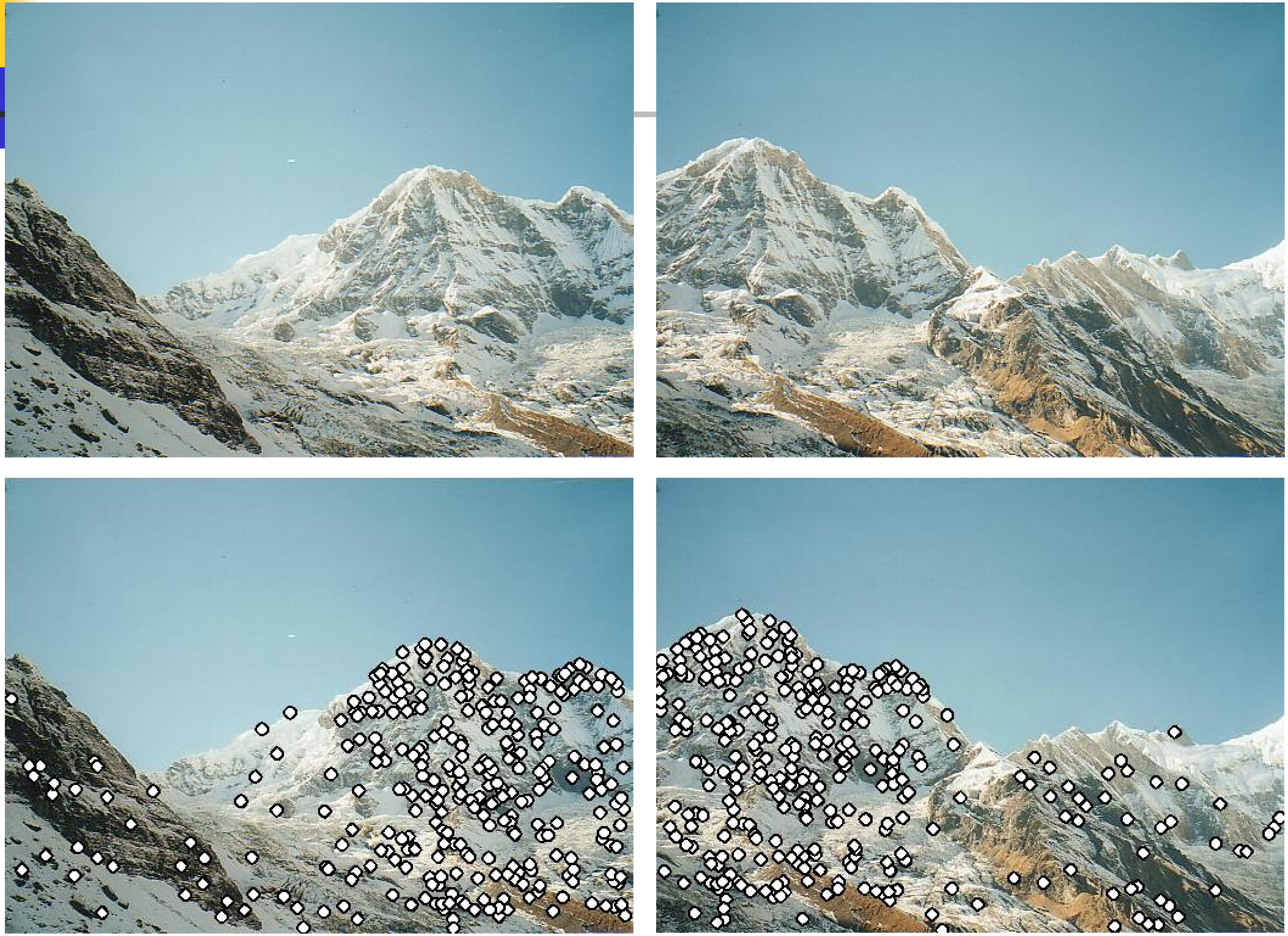
Descriptor Vector(SIFT)

- Orientation = blurred gradient
- Similarity Invariant Frame
  - Scale-space position (x, y, s) + orientation (θ)

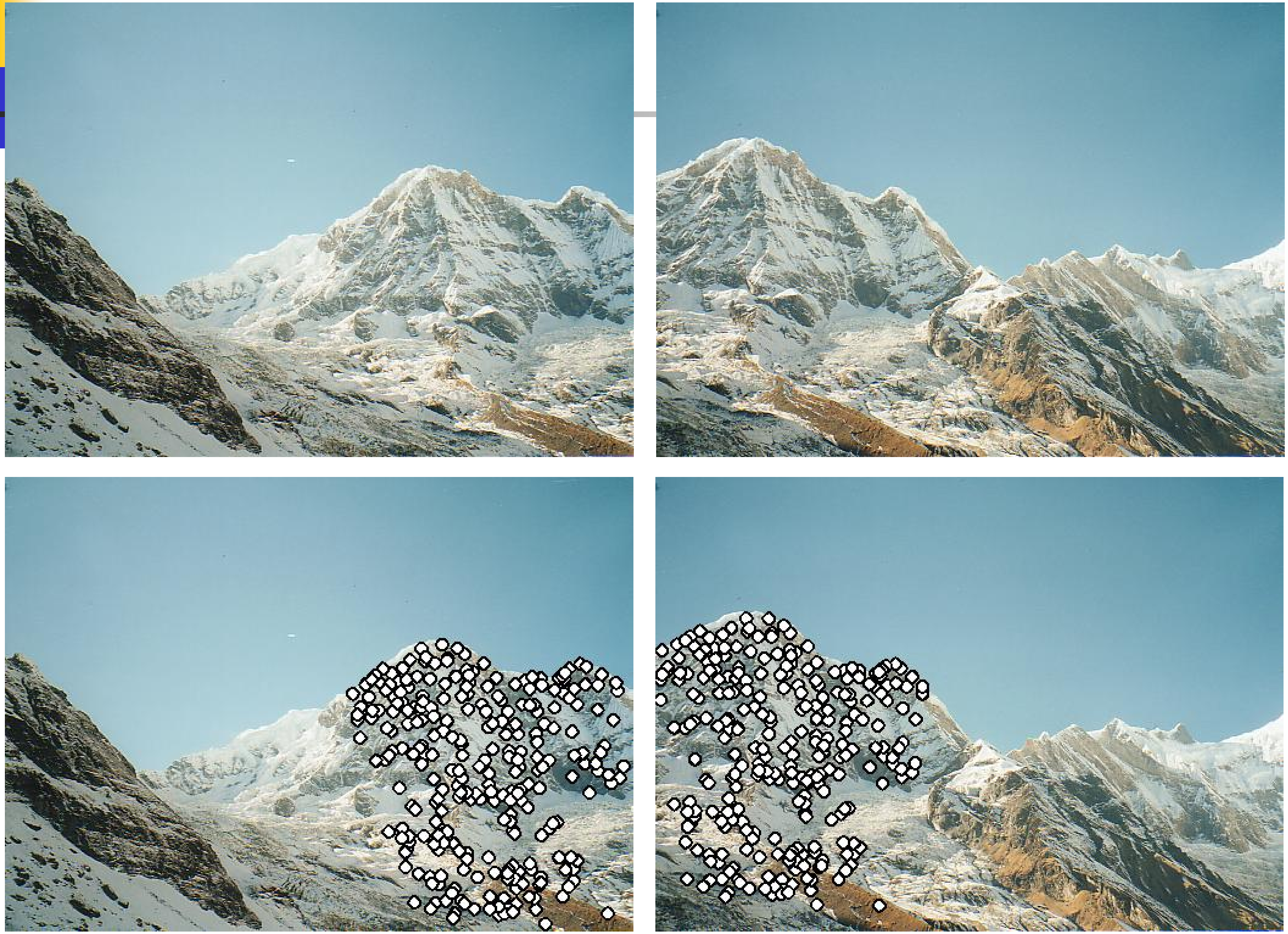
Probabilistic Feature Matching



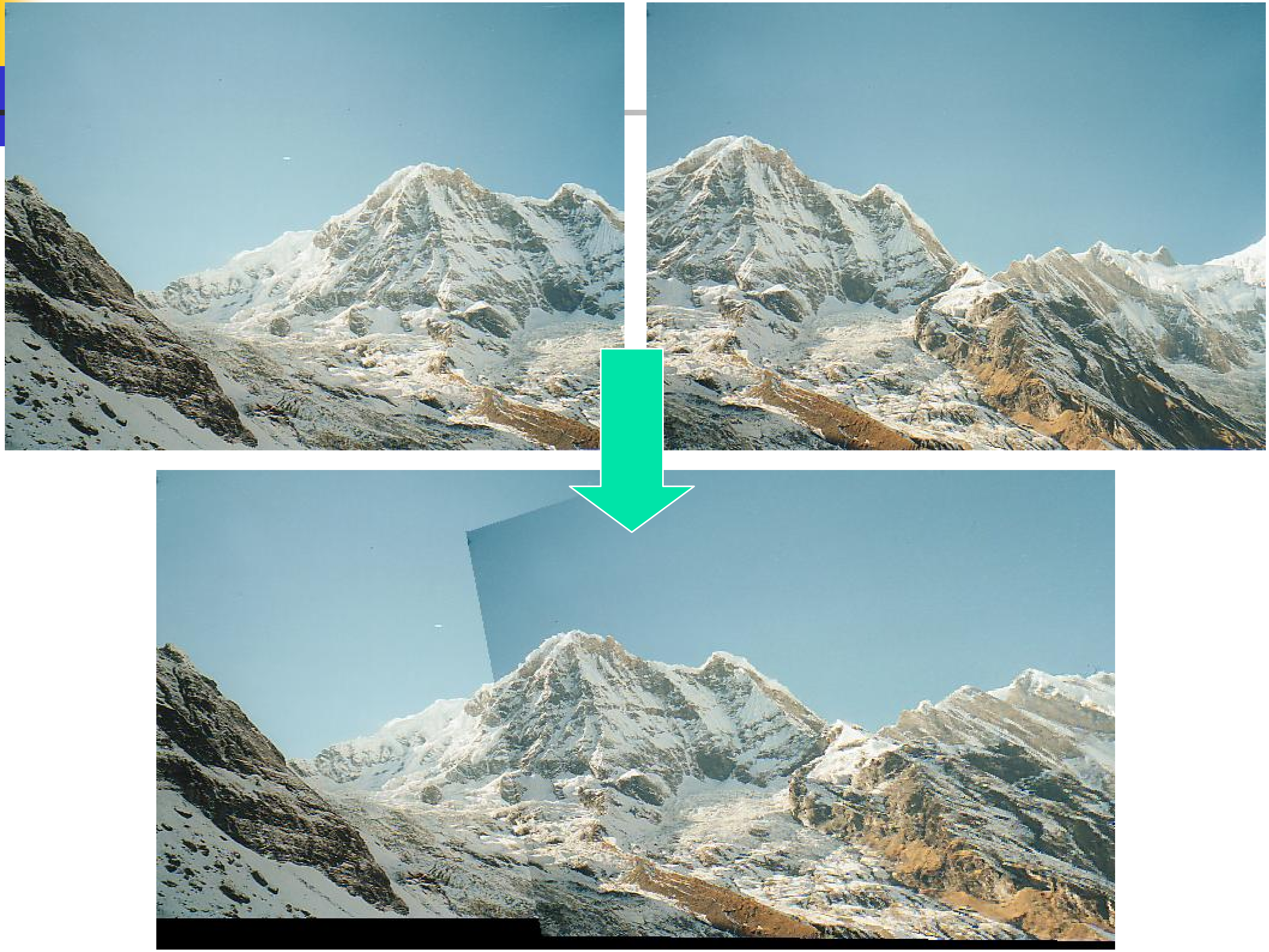
RANSAC motion model



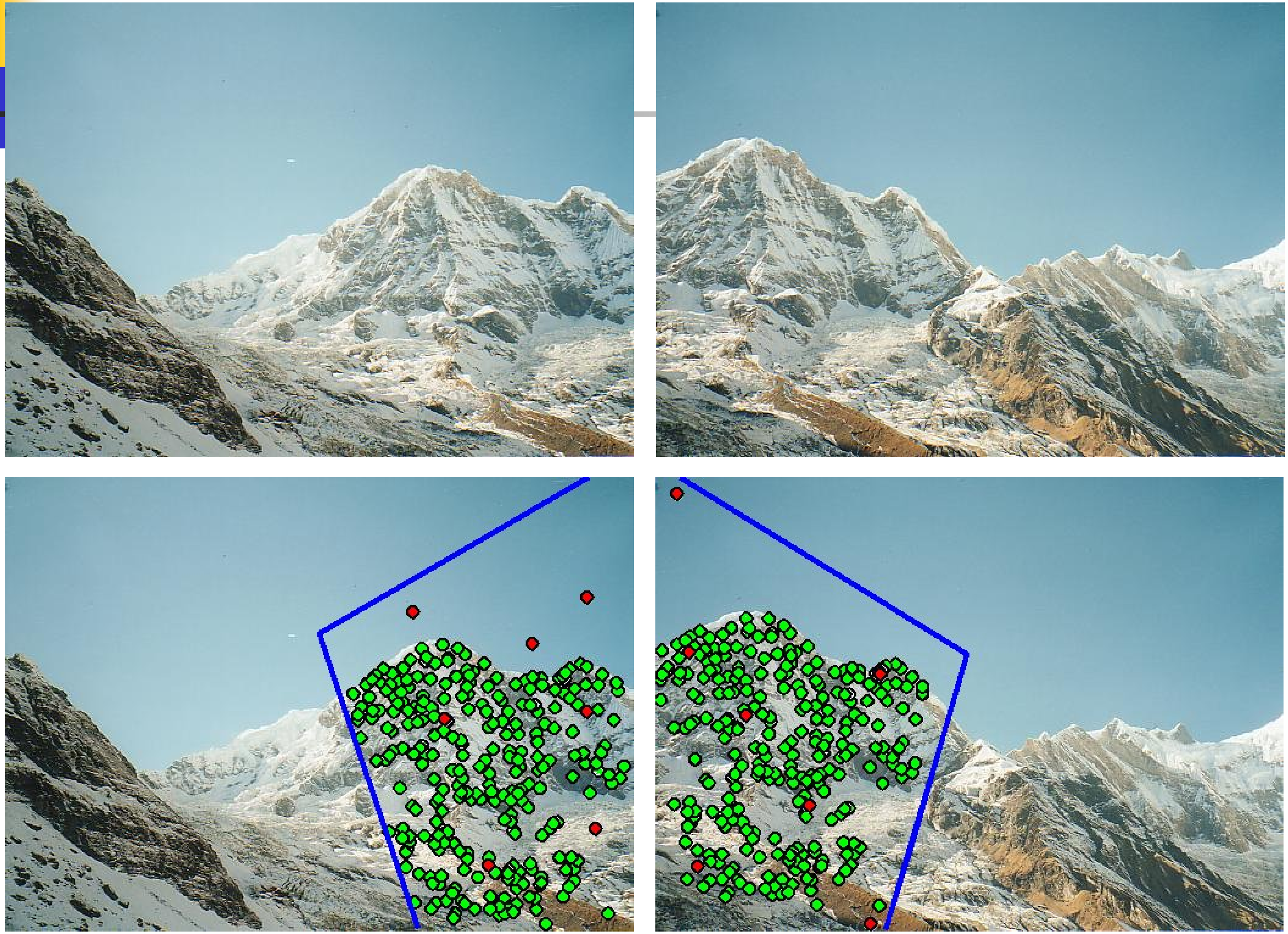
RANSAC motion model



RANSAC motion model



Probabilistic model for verification



How well does this work?

Test on 100s of examples…

…still too many failures (5-10%) for <u>consumer</u> application

Matching Mistakes: False Positive

Matching Mistakes: False Positive

Matching Mistake: False Negative

- Moving objects: large areas of disagreement

Matching Mistakes

- Accidental alignment
  - repeated / similar regions
- Failed alignments
  - moving objects / parallax
  - low overlap
  - “feature-less” regions (more variety?)
- 100% reliable algorithm?

How can we fix these?

- Tune the feature detector (reliable feature)
- Tune the feature matcher (reliable match)
- Tune the RANSAC stage (motion model)
- Use “higher-level” knowledge
  - e.g., typical camera motions
- → Sounds like a big “learning” problem
  - Need a large training/test data set (panoramas)

Global motion

- Common motion observed in the frame
  - Motion of all points in the scene
  - Motion of most of the points in the scene
- Reasons
  - Motion of sensor (Ego Motion)
  - Motion of a rigid scene
- Parametric flow describes optical flow for each pixel
  - Affine
  - Projective
- Global motion can be used to
  - Visual mosaics
  - Image registration
  - Removing camera jitter
  - Object tracking
  - Video segmentation

Aligning images

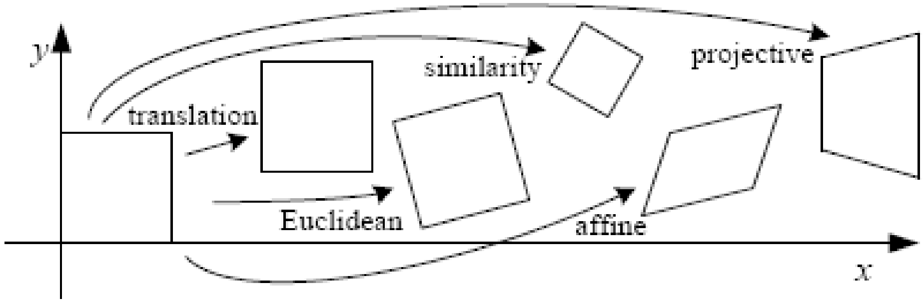
- How to account for warping?
  - Translations are not enough to align the images

Motion models

Motion models

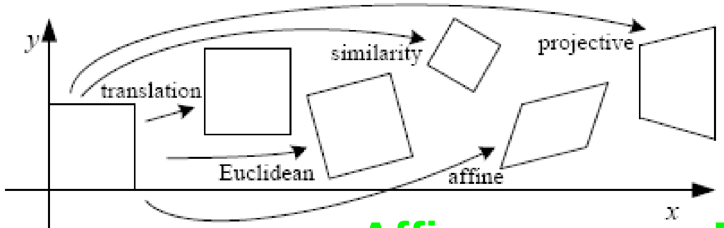
- What happens when we take two images with a camera and try to align them?
- translation?
- rotation?
- scale?
- affine?
- perspective?

Motion models

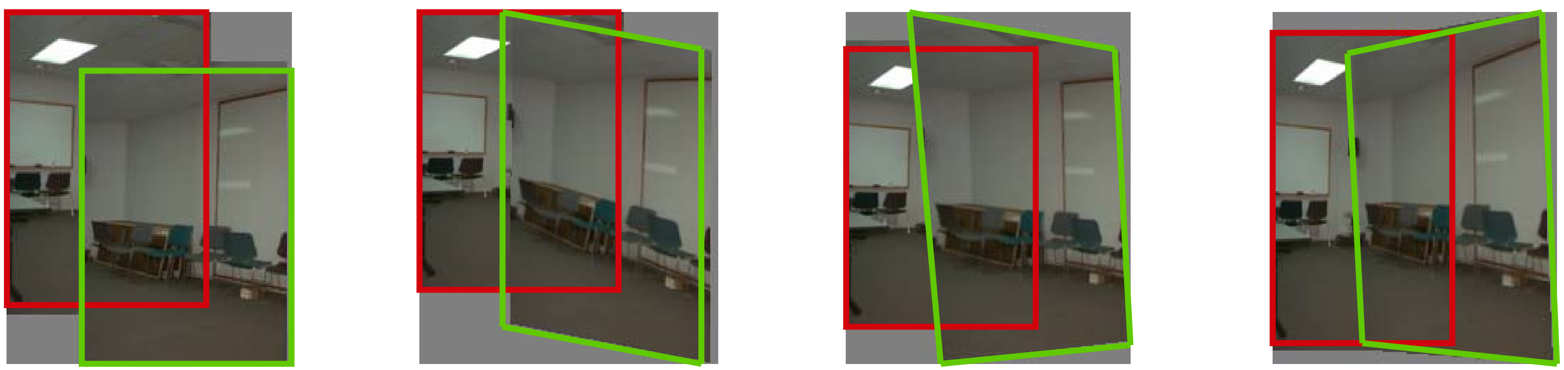


Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\begin{array}{c\|c} I & t \end{array}\right]_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\left[\begin{array}{c\|c} R & t \end{array}\right]_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\left[\begin{array}{c\|c} sR & t \end{array}\right]_{2\times 3}$	4	angles $+\cdots$	
affine	$\left[\begin{array}{c} A \end{array}\right]_{2\times 3}$	6	parallelism $+\cdots$	
projective	$\left[\begin{array}{c} \tilde{H} \end{array}\right]_{3\times 3}$	8	straight lines	

Motion models



Translation **Affine** **Perspective** **3D rotation**



2 unknowns **6 unknowns** **8 unknowns** **3 unknowns**

Affine Motion

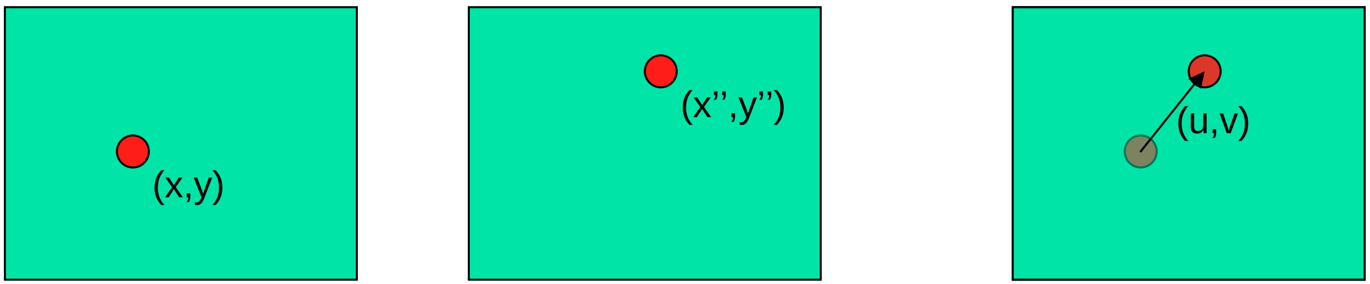


image at time t

image at time t+1

$$u = x''$$
$$v = y''$$

$$u = x'' - x$$
$$v = y'' - y$$

Affine motion:

$$u(x, y) = a_1 x + a_2 y + b_1$$
$$v(x, y) = a_3 x + a_4 y + b_2$$

$$\boxed{a_1, a_2, b_1, a_3, a_4, b_2}$$

Affine motion parameters

Global Affine Motion

$$u(x, y) = a_1 x + a_2 y + b_1$$

$$v(x, y) = a_3 x + a_4 y + b_2$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Solving for affine transformation

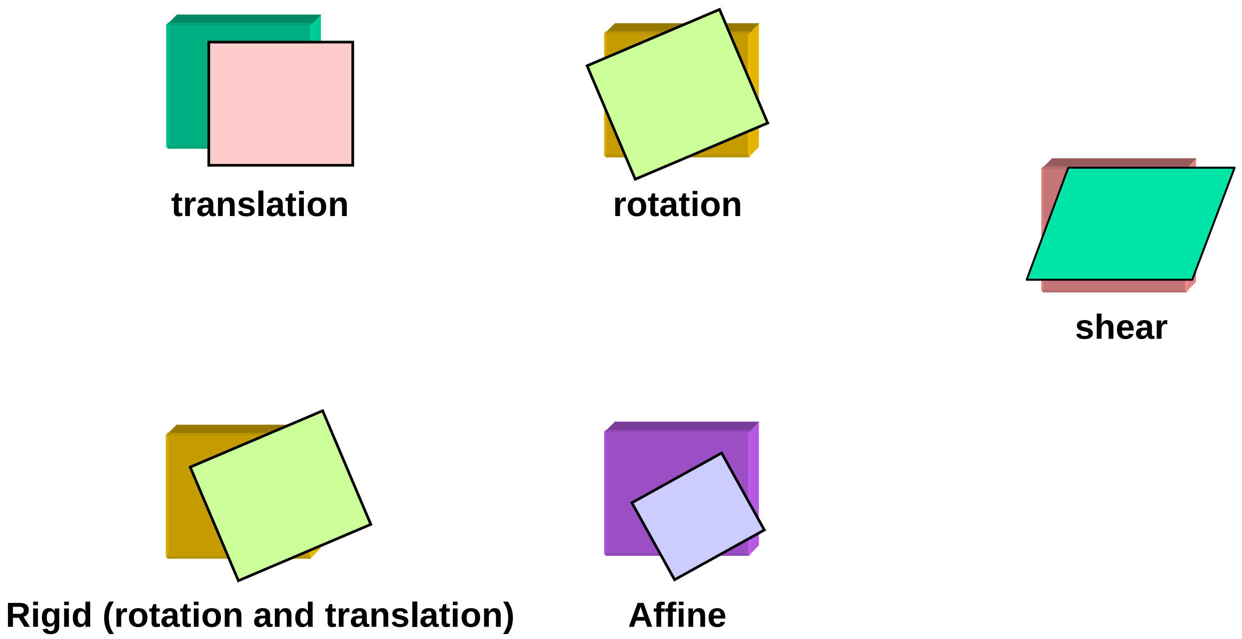
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_i & y_i & 1 & & & \\ & & & x_i & y_i & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{32} \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

- This is a general linear equation set
  - How many point correspondences are necessary?

Spatial Transformations

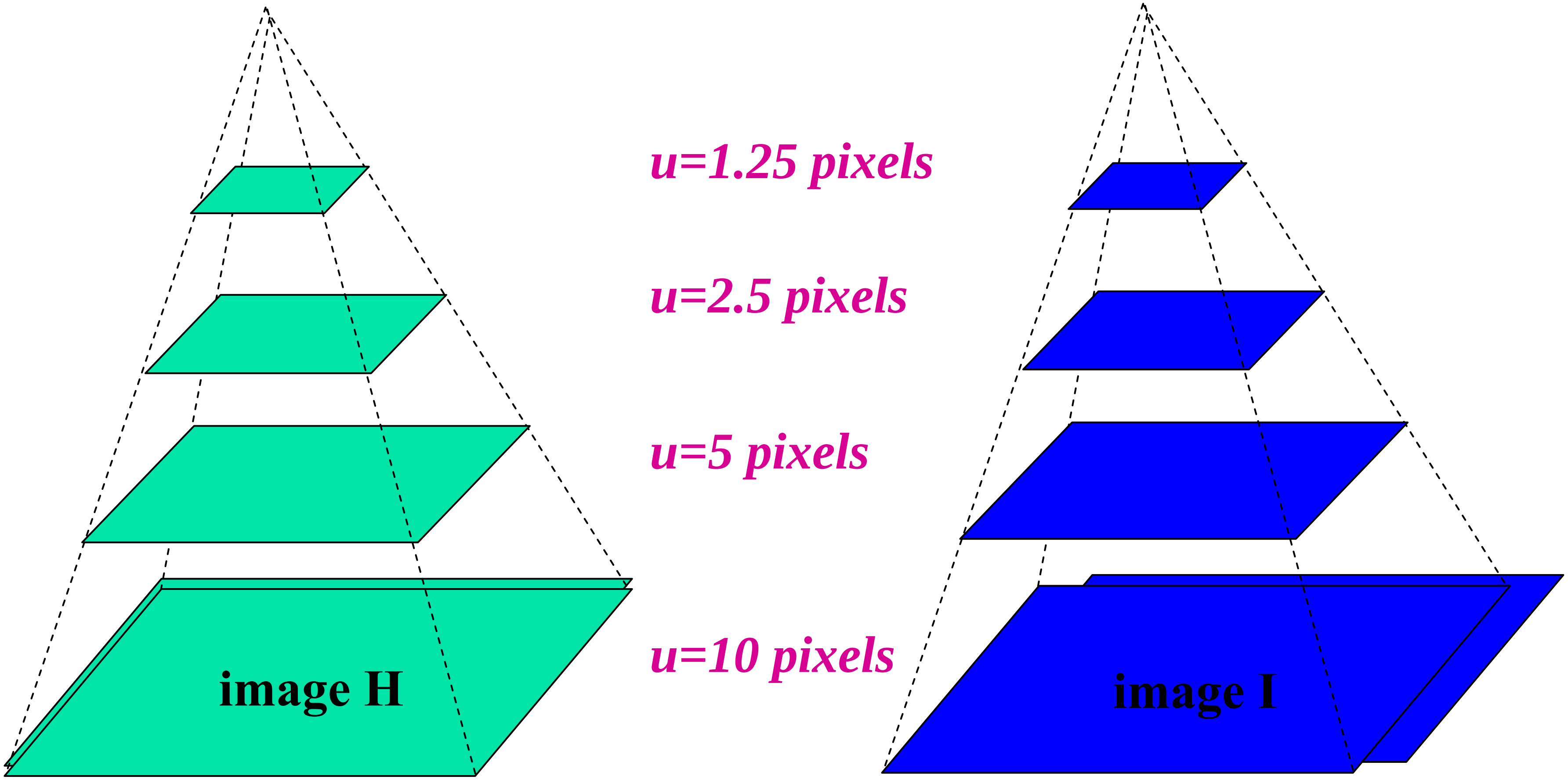
- Transformations in image space



Affine transform based Algorithm

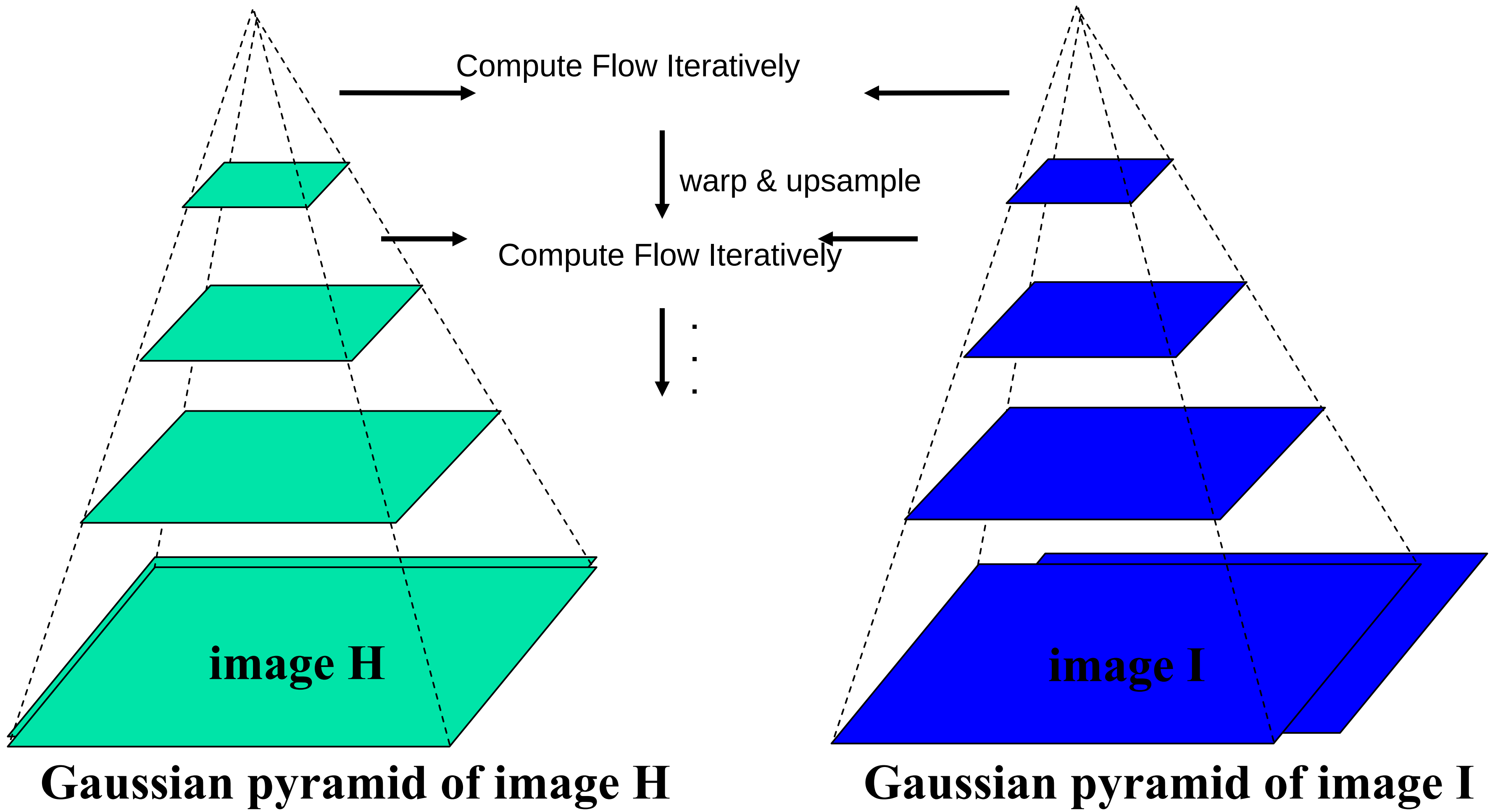
- Initialize affine parameters (local match)
- Compute affine parameters iteratively
  - Compute new affine parameters (global match)
  - At each iteration update the global affine solution based on matching error
- Stop when affine parameters do not update (global minimum achieved)
- If motion in between frames is high, construct pyramid representation.

Using Pyramids



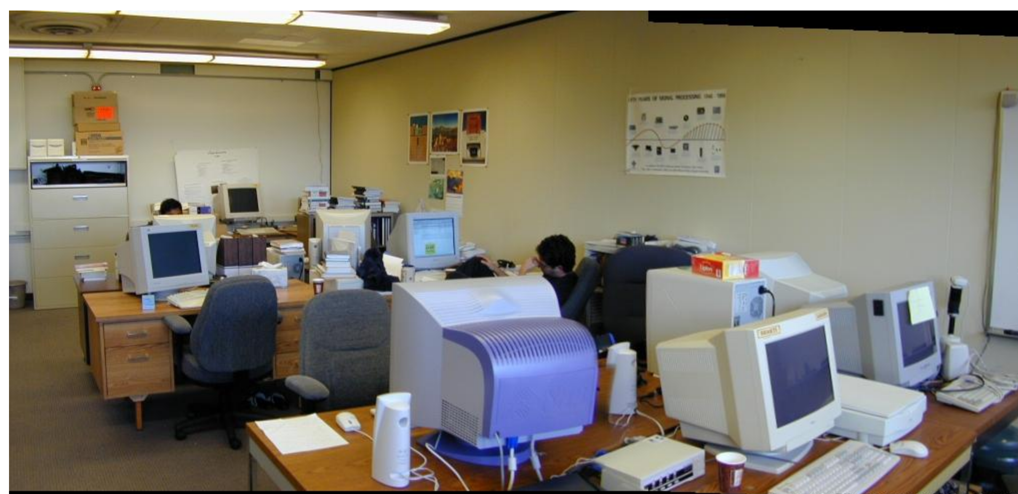
Gaussian pyramid of image H **Gaussian pyramid of image I**

Using Pyramids

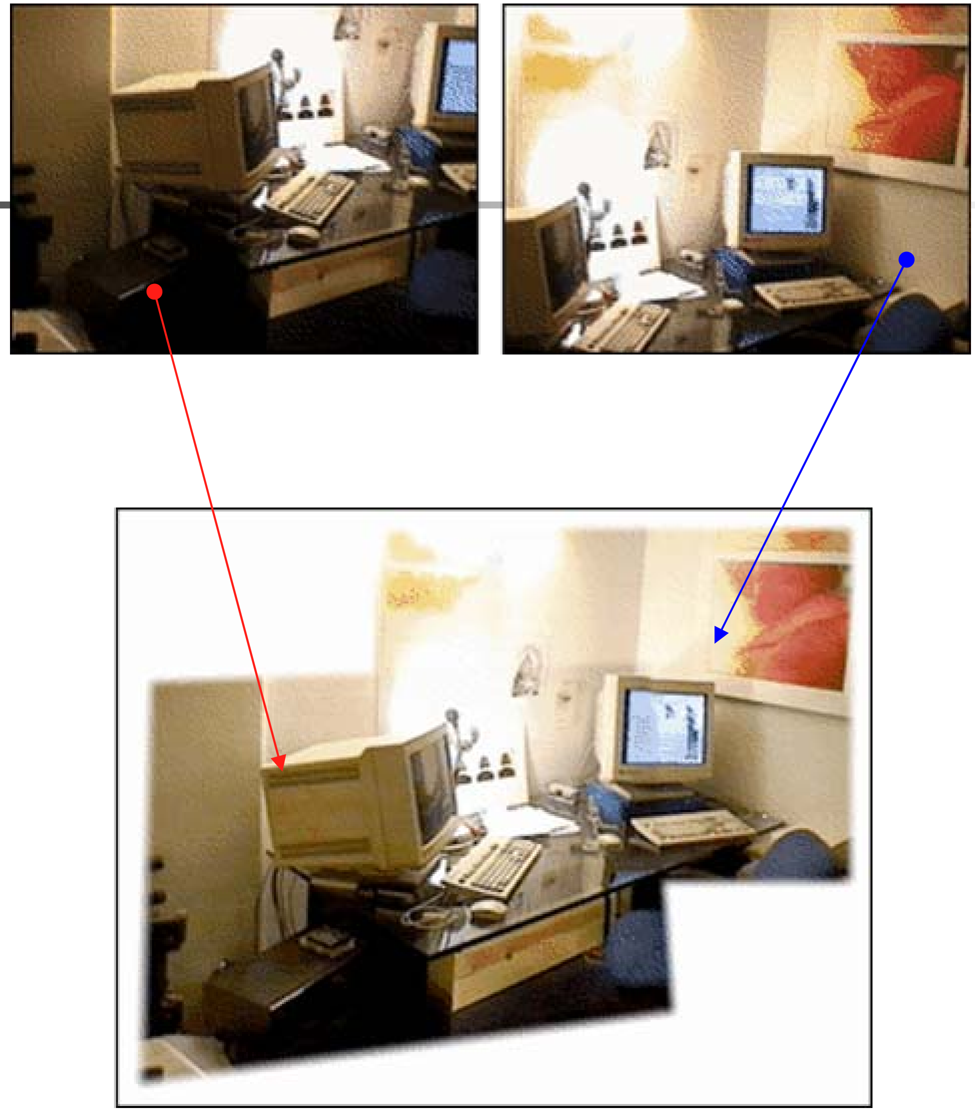


An Example

- Mosaic



Examples

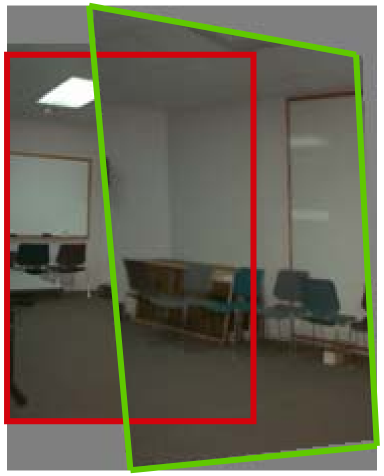


Examples

Examples

From affine to perspective

- 8-parameter generalization of affine motion
  - works for pure rotation or planar surfaces
- Limitations:
  - local minima
  - slow convergence
  - difficult to control interactively



Special Projectivities

Invariants

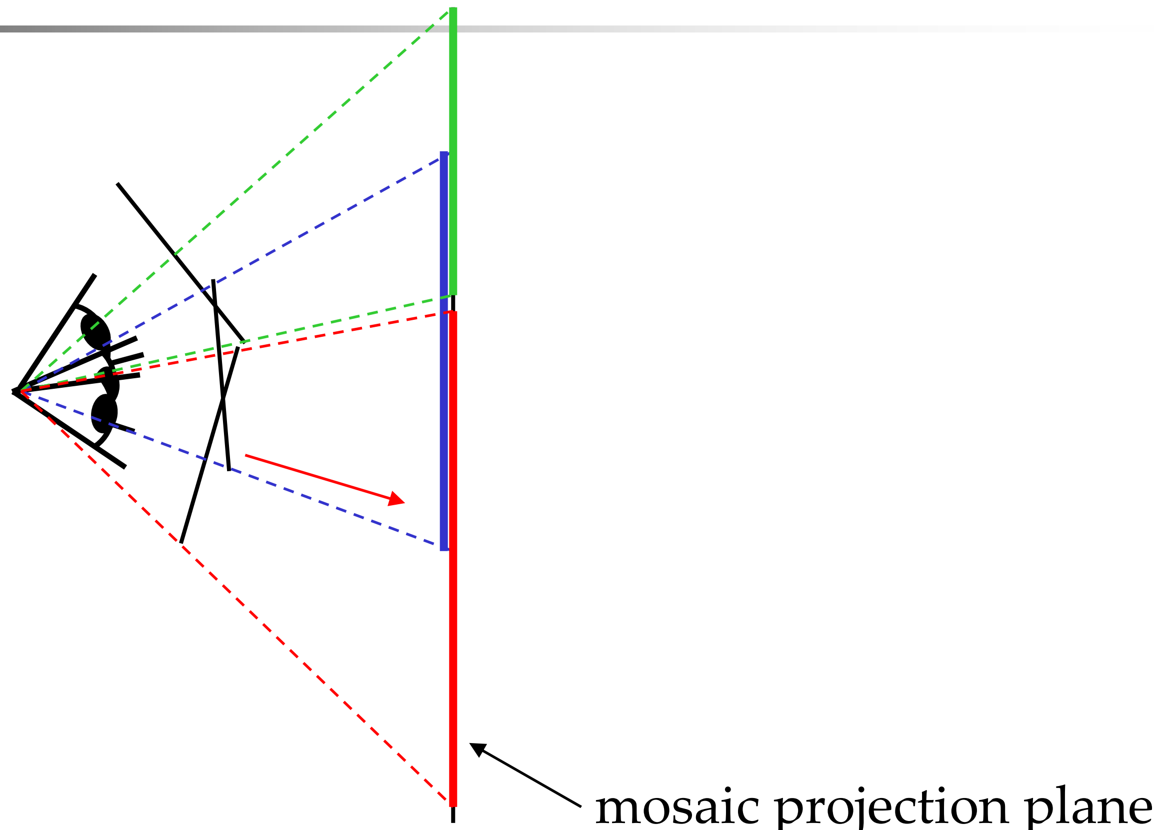
Projectivity
8 dof

Affine transform
6 dof

Similarity
4 dof

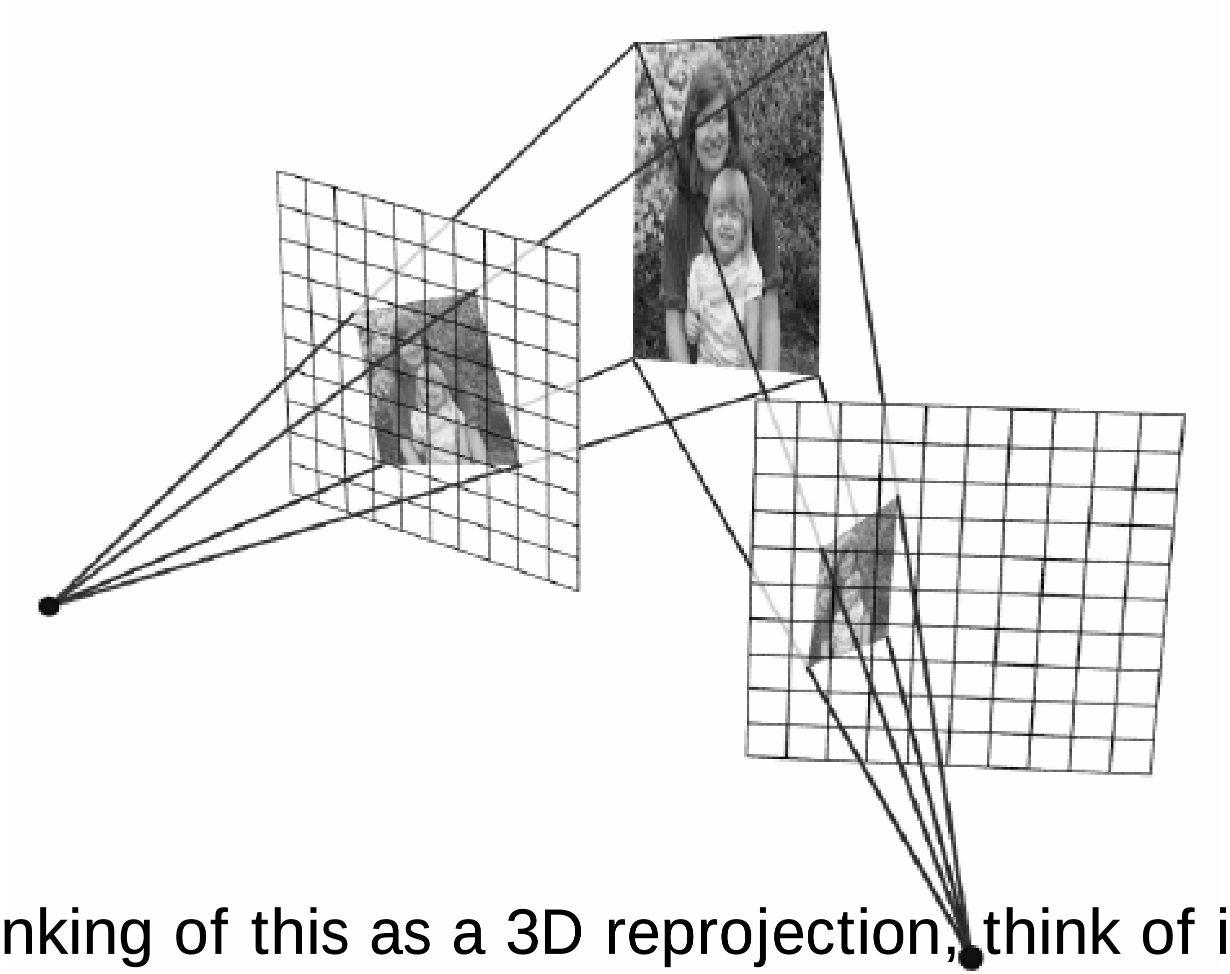
Euclidean transform
3 dof

Image Reprojection



- The mosaic has a natural interpretation in 3D
  - The images are reprojected onto a common plane
  - The mosaic is formed on this plane

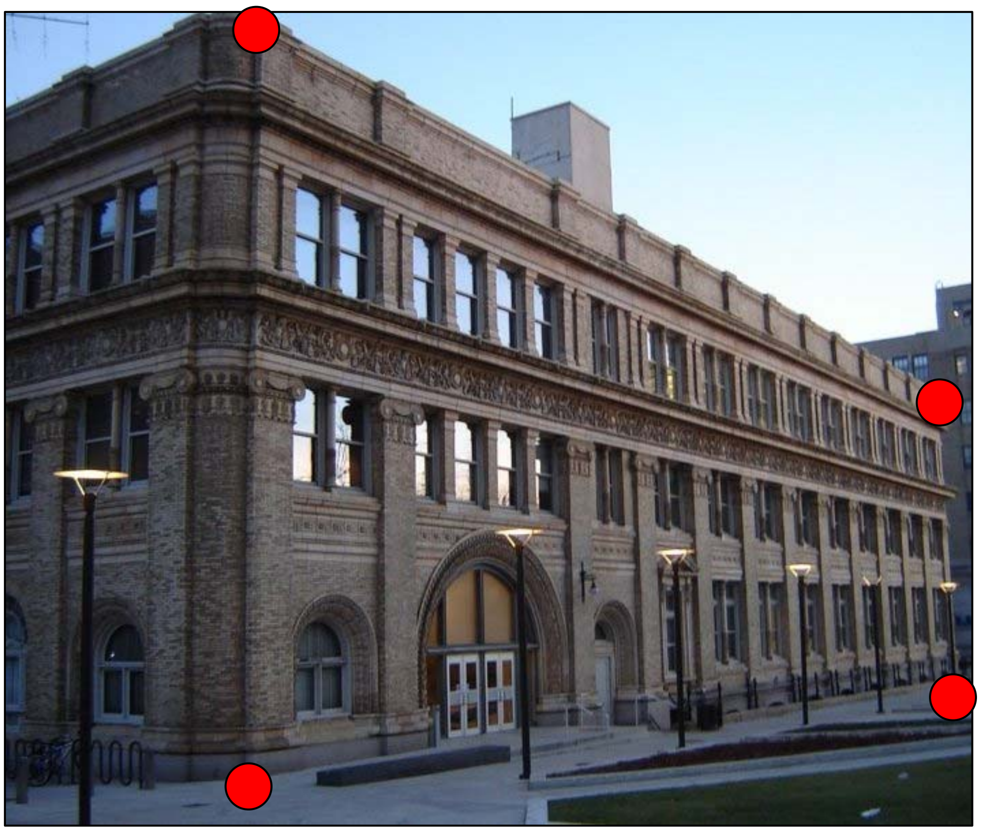
Image Reprojection



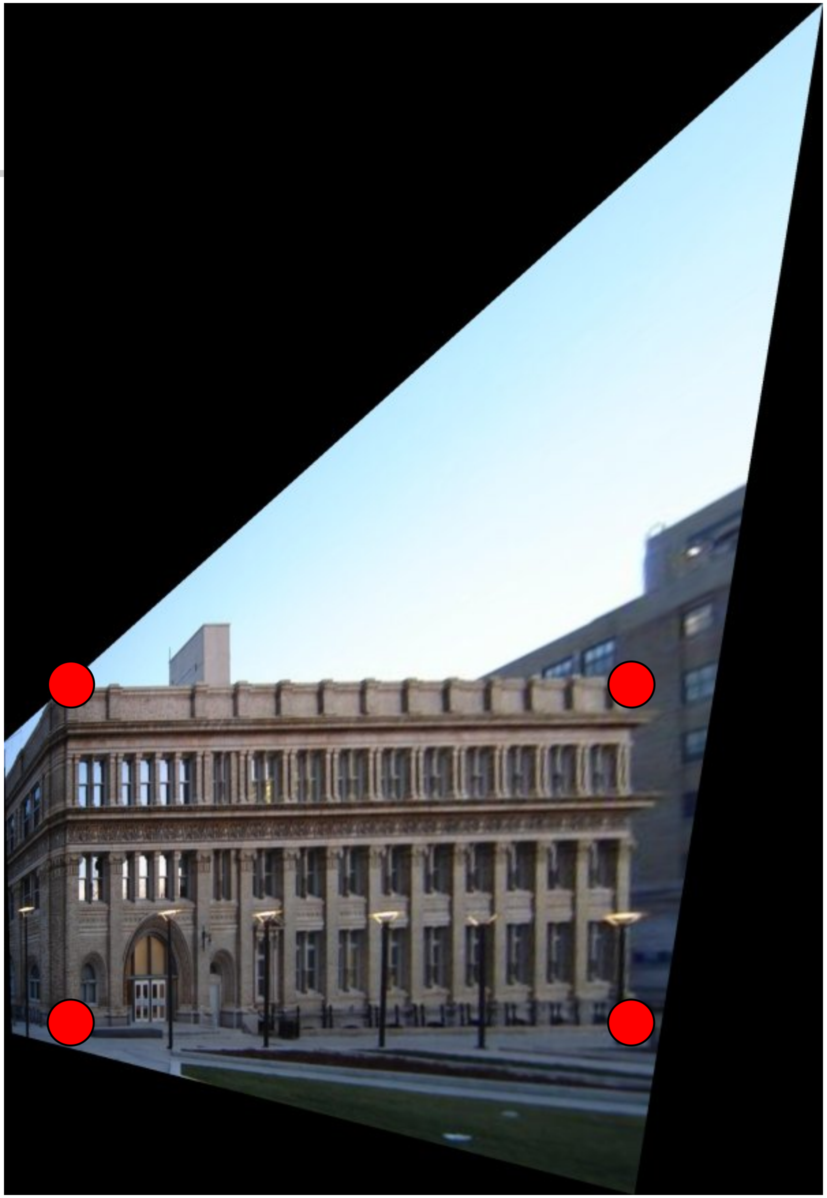
- Rather than thinking of this as a 3D reprojection, think of it as a 2D image warp from one image to another

Example

- Rectification

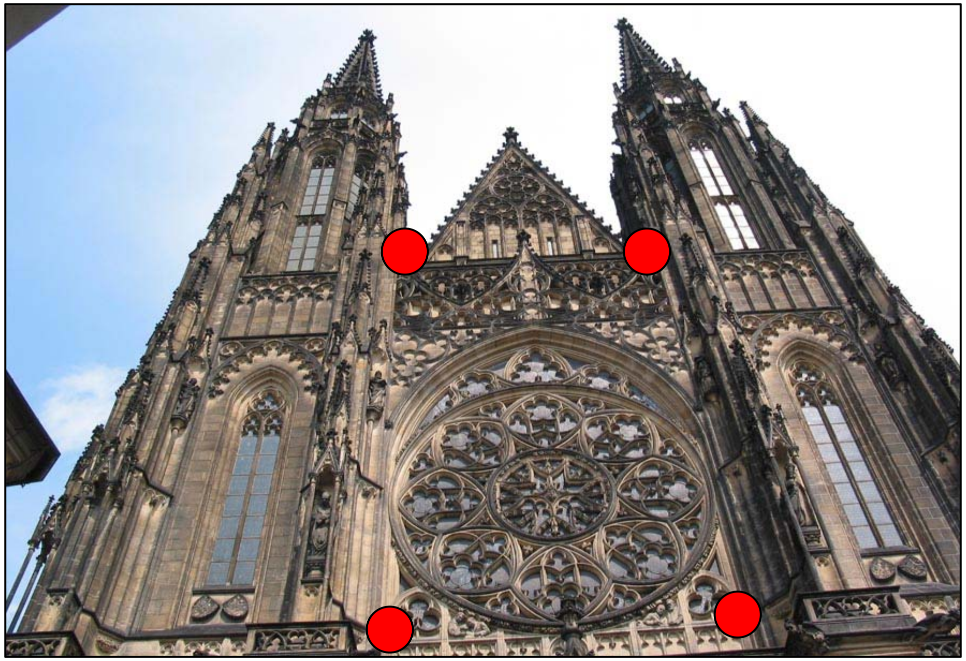


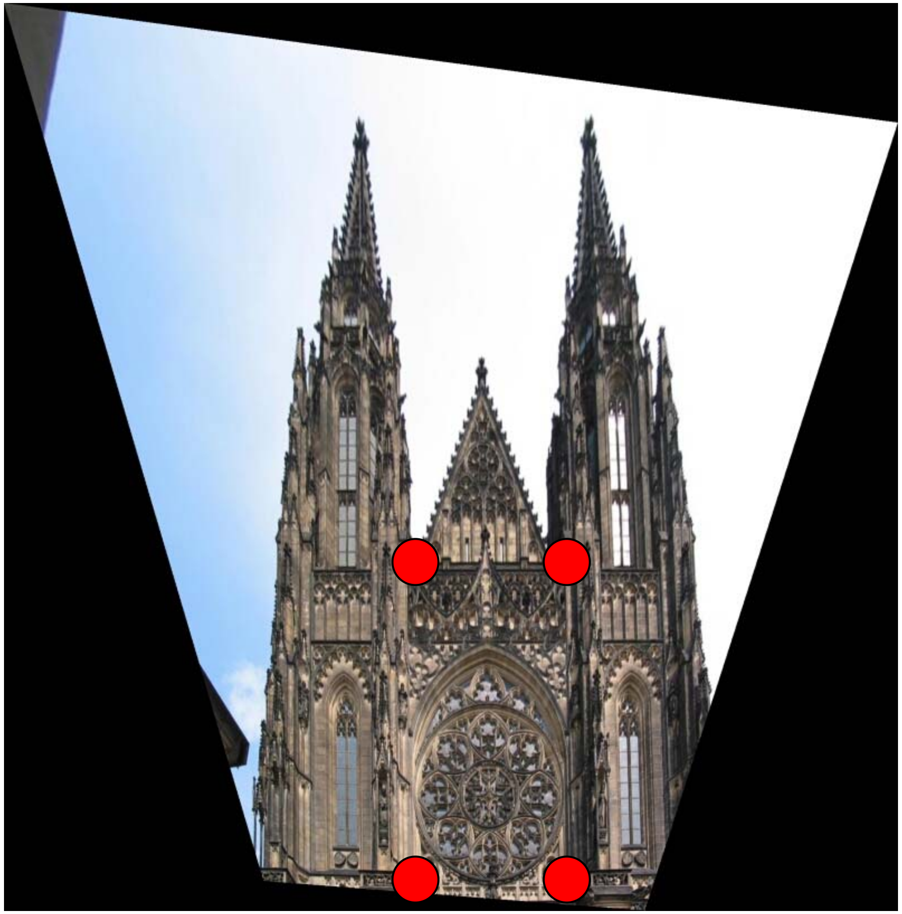
This is your test image



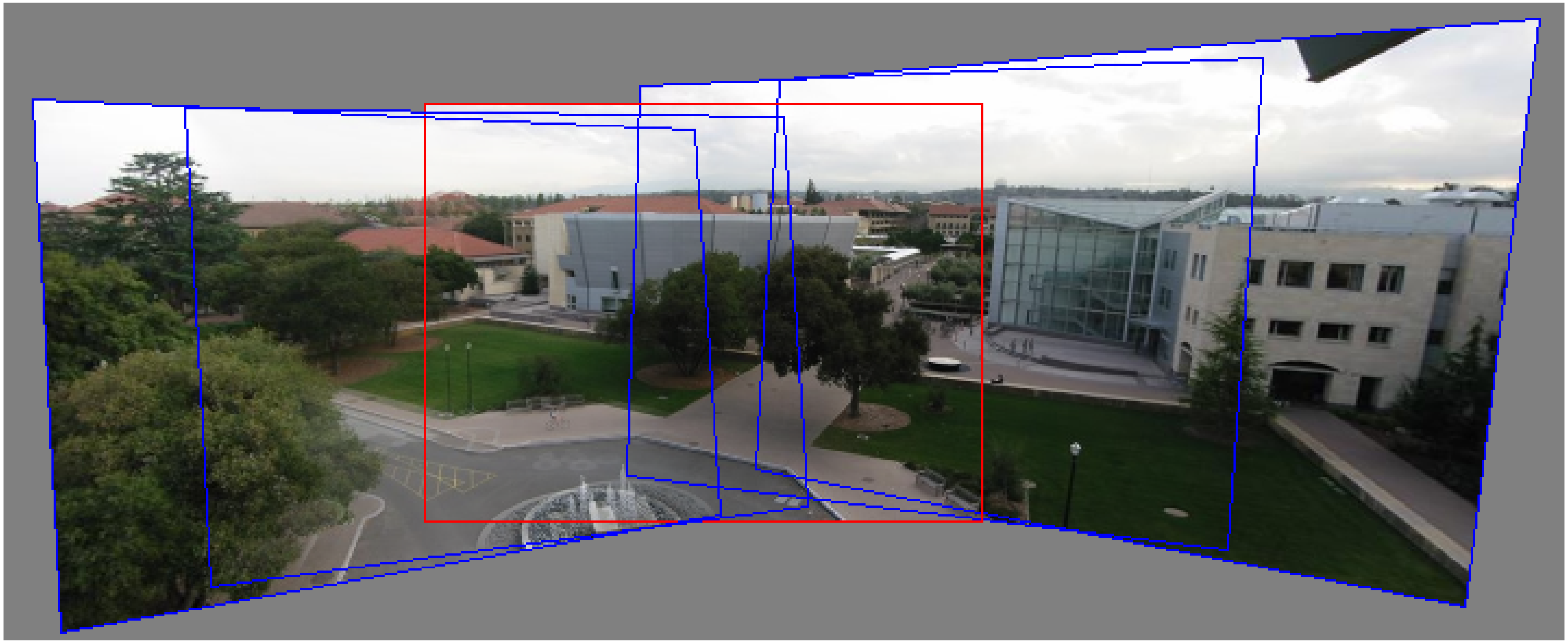
Example

- Rectification





Stitching demo




CARLTON

Ingredients

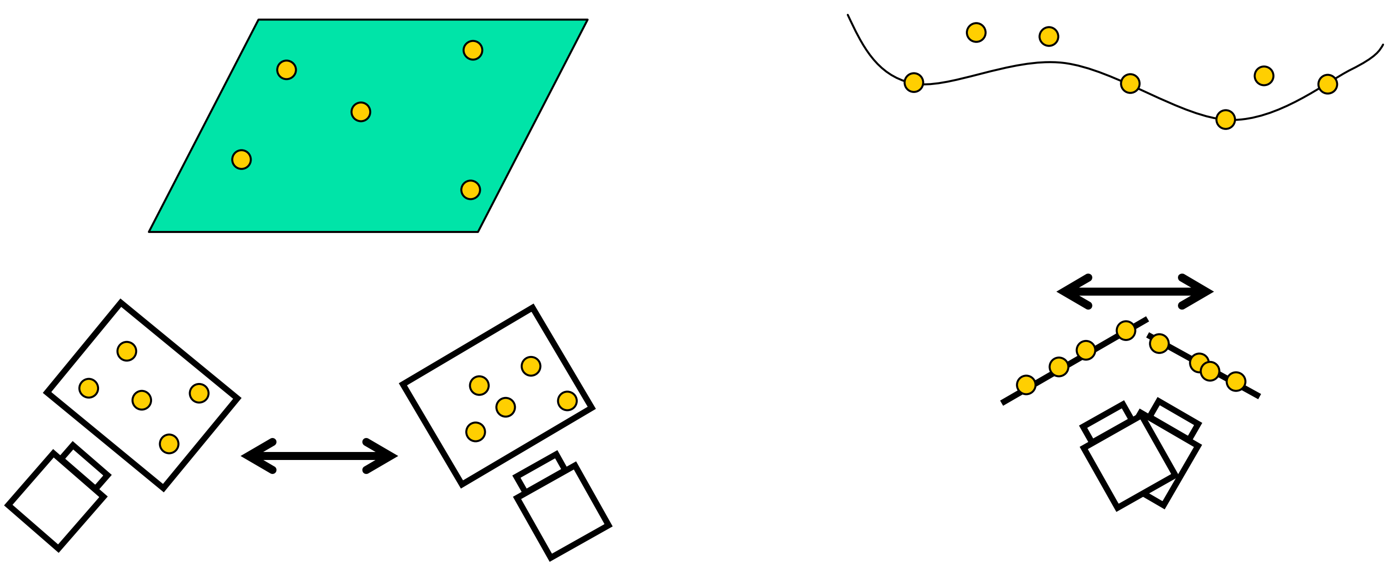
- Take good images
- Specify correspondences (manual)
- Compute homography
  - Solve with eigen decomposition
- Apply homography
  - Warping
  - Interpolation
  - Masking
  - Blending

How to do it?

- Basic Procedure
  - Take a sequence of images from the same position
    - Rotate the camera about its optical center
  - Compute transformation between the second image and the first
  - Shift the second image to overlap with the first
  - Blend the two together to create a mosaic
  - If there are more images, repeat

Homography

- Homography is a singular case of the Fundamental Matrix(基本矩阵)
  - Two views of **coplanar points**
  - Two views that **share the same center of projection**



Homographies

- Perspective projection of a plane
  - Lots of names for this:
    - **homography**, collineation, planar projective map
  - Modeled as a 2D warp using homogeneous coordinates

$$\underset{\mathbf{p'}}{\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}} = \underset{\mathbf{H}}{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}} \underset{\mathbf{p}}{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}$$

To apply a homography **H**

- Compute $\mathbf{p'} = \mathbf{Hp}$ (regular matrix multiplication)
- Convert $\mathbf{p'}$ from homogeneous to image coordinates
  - divide by w (third) coordinate

Solving for homographies

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x'_i = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$y'_i = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$\begin{aligned} x'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned}$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x'_i x_i & -x'_i y_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_i x_i & -y'_i y_i & -y'_i \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Solving for homographies

$$
\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n \end{bmatrix}}_{\substack{\mathbf{A} \\ 2n \times 9}} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\substack{\mathbf{h} \\ 9}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{\substack{\mathbf{0} \\ 2n}}
$$

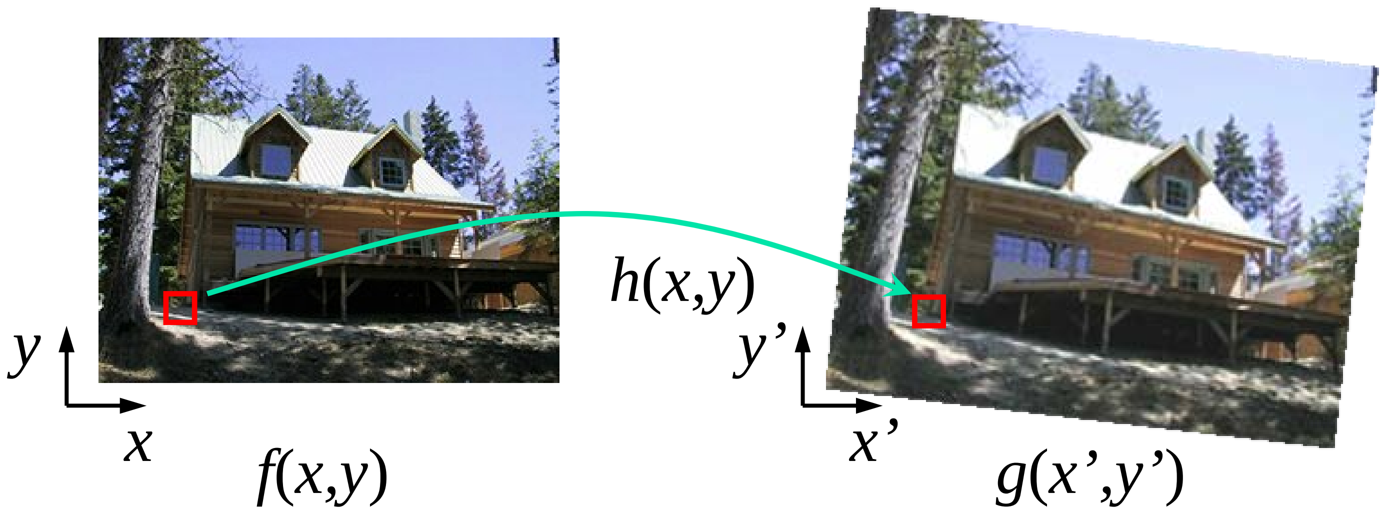
- Defines a least squares problem: $\text{minimize } \|\mathbf{A}\mathbf{h} - \mathbf{0}\|^2$
  - Since **h** is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
  - Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue
  - Works with 4 or more points

Radial distortion

- Correct for “bending” in wide field of view lenses

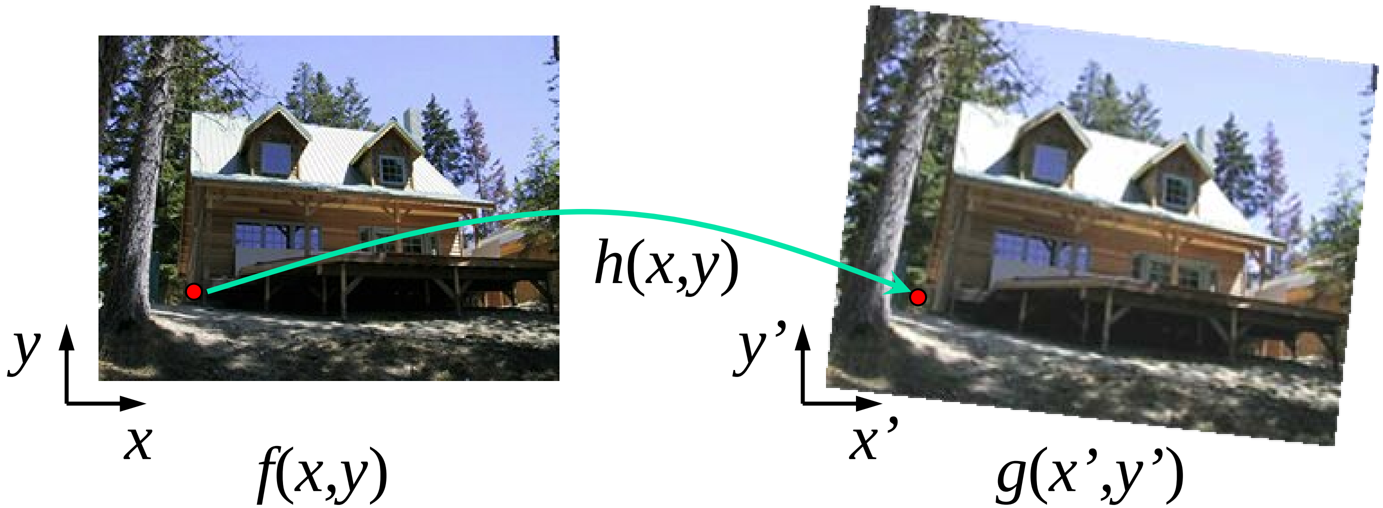
$$
\begin{aligned}
\hat{r}^2 &= \hat{x}^2 + \hat{y}^2 \\
\hat{x}' &= \hat{x}/(1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4) \\
\hat{y}' &= \hat{y}/(1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4) \\
x &= f\hat{x}'/\hat{z} + x_c \\
y &= f\hat{y}'/\hat{z} + y_c
\end{aligned}
$$

Image Warping



- Given a coordinate transform $(x',y') = h(x,y)$ and a source image $f(x,y)$, how do we compute a transformed image $g(x',y') = f(h(x,y))$?

Forward Warping

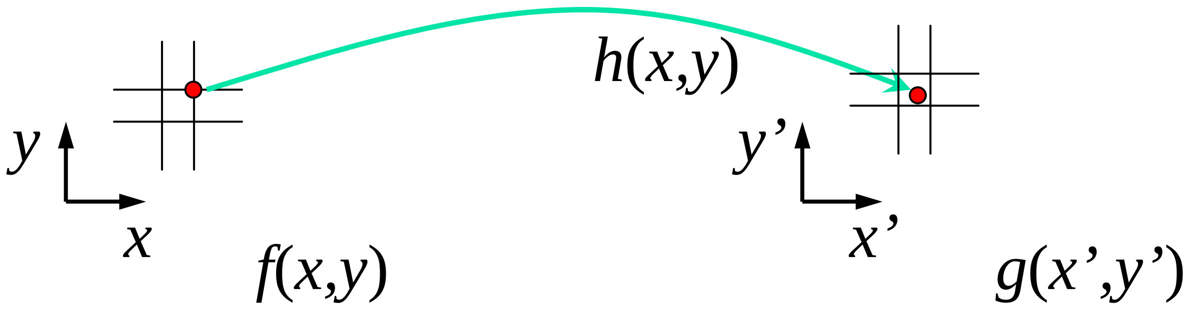


- Send each pixel $f(x,y)$ to its corresponding location

  $(x',y') = h(x,y)$ in the second image

Q: what if pixel lands "between" two pixels?

Forward Warping



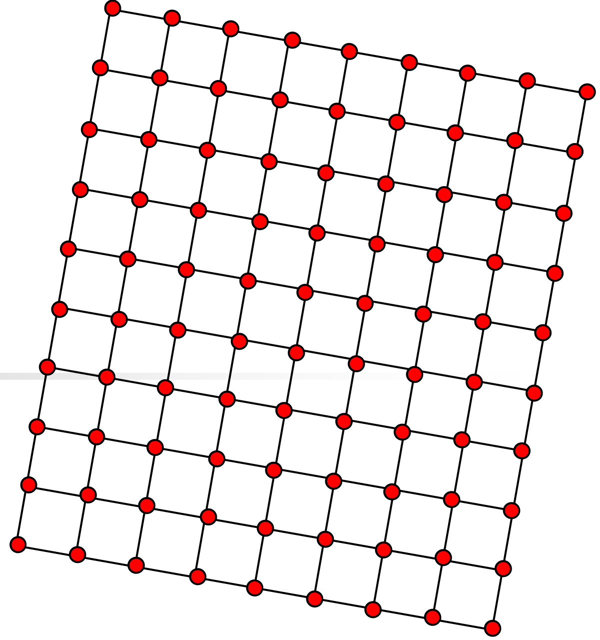
- Send each pixel $f(x,y)$ to its corresponding location $(x',y') = h(x,y)$ in the second image

Q: what if pixel lands "between" two pixels?

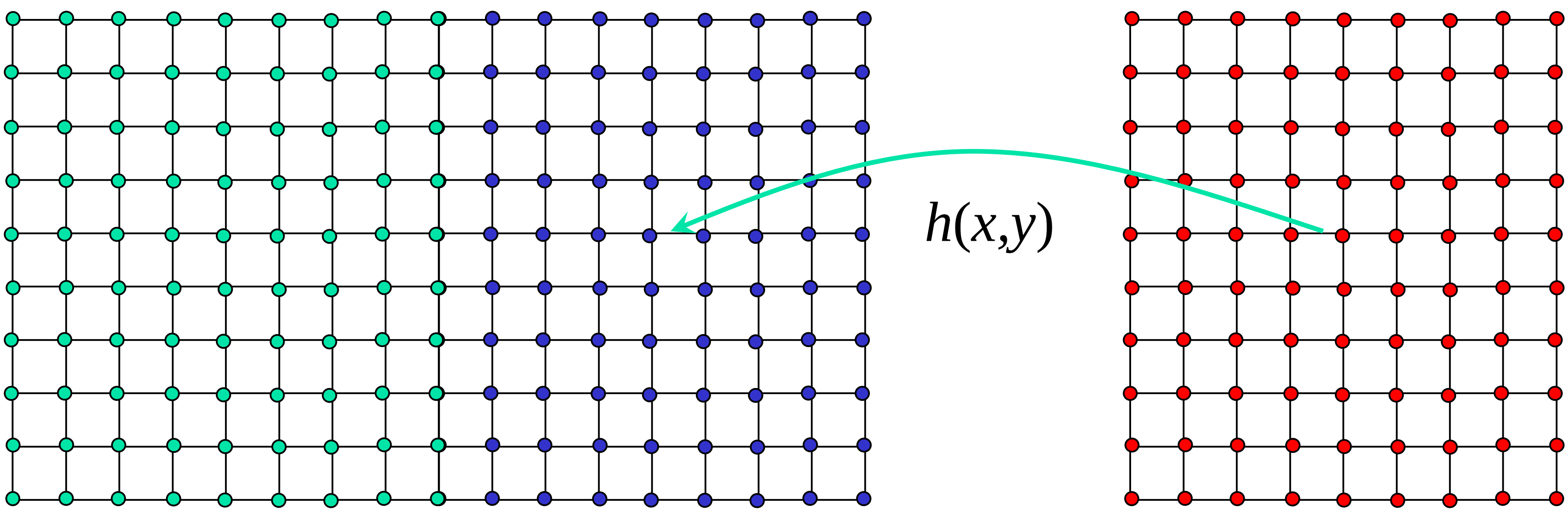
A: distribute color among neighboring pixels (x',y')

– Known as "splatting"

Forward Warping

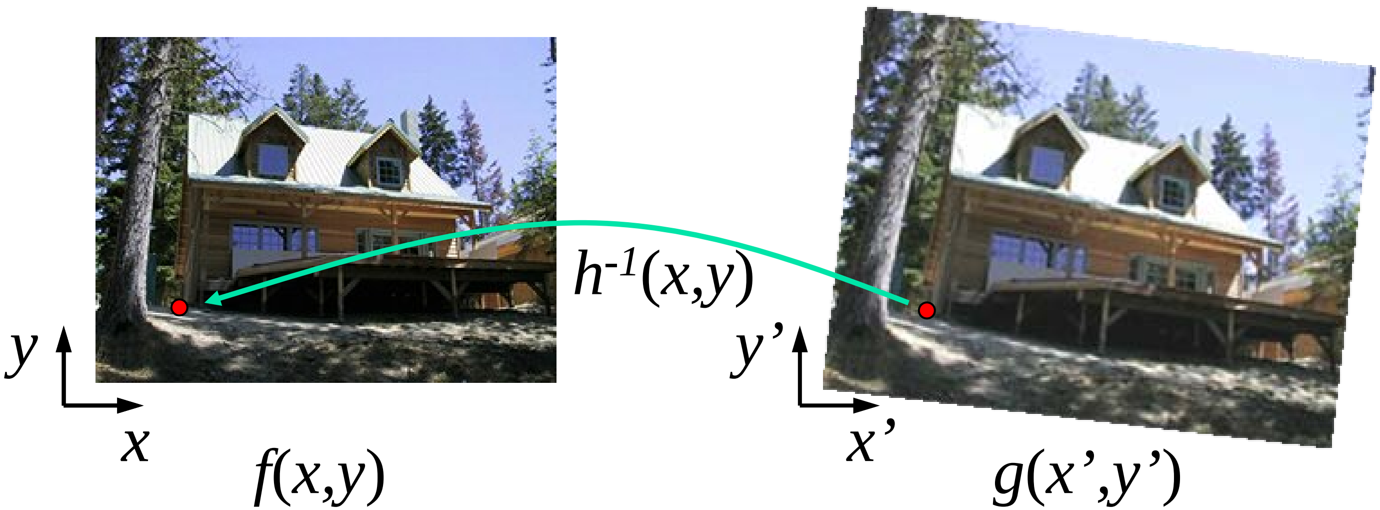


Forward warped image



Reference image Extended region Target image

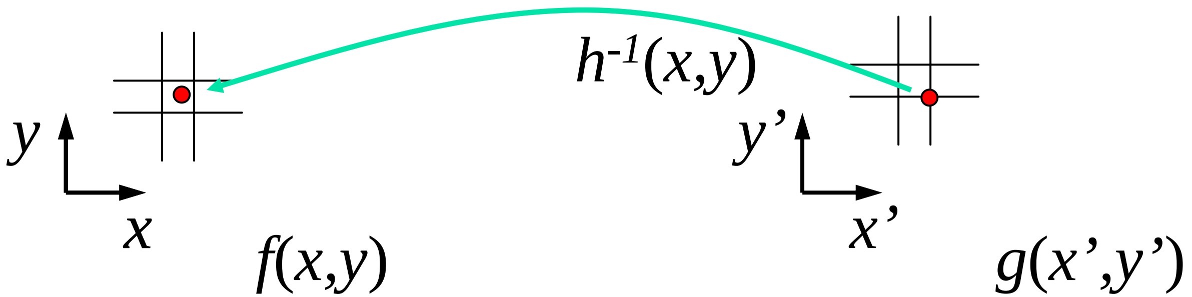
Inverse Warping



- Get each pixel $g(x',y')$ from its corresponding location
- $(x,y) = h^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?

Inverse Warping

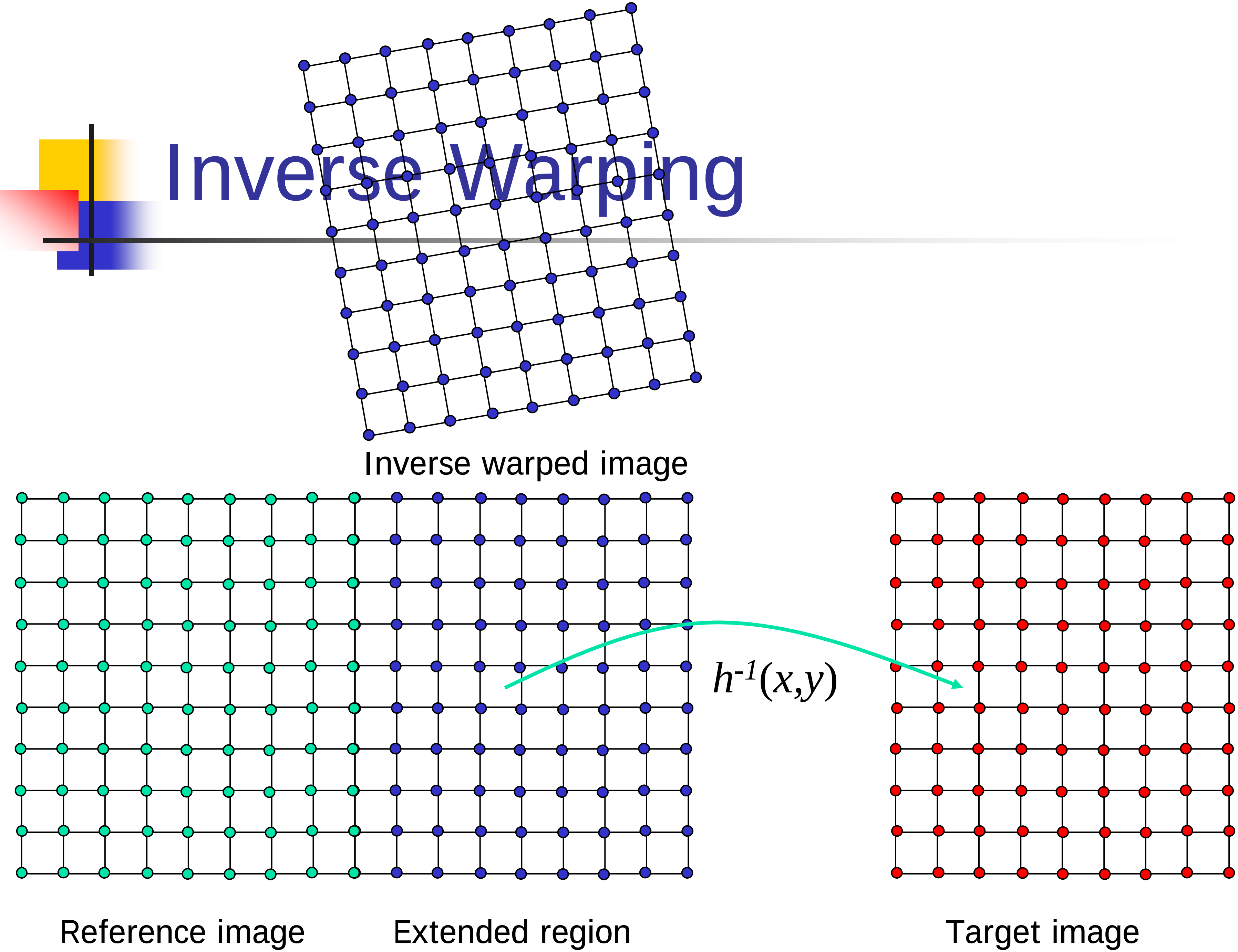


- Get each pixel $g(x',y')$ from its corresponding location $(x,y) = h^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?

A: *resample* color value

Inverse Warping

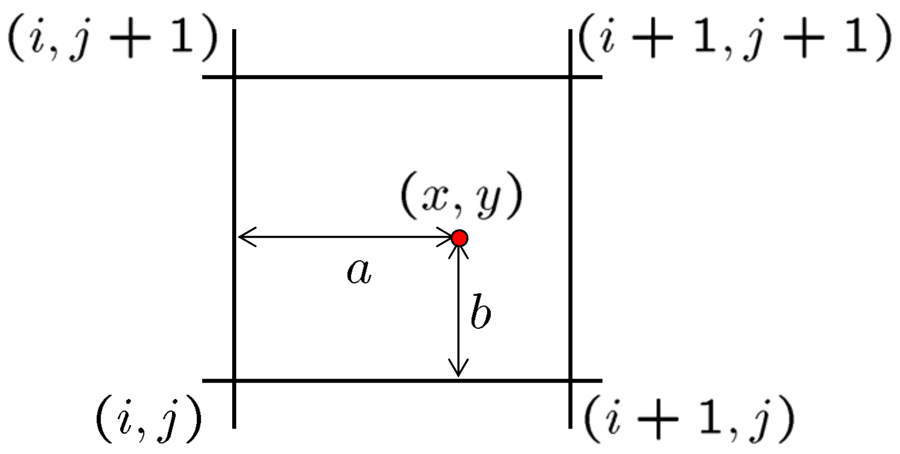


Forward vs. Inverse Warping

- Q: which is better?
- A: usually inverse—eliminates holes
  - however, it requires an invertible warp function—not always possible...

Bilinear Interpolation

- A simple method for resampling images



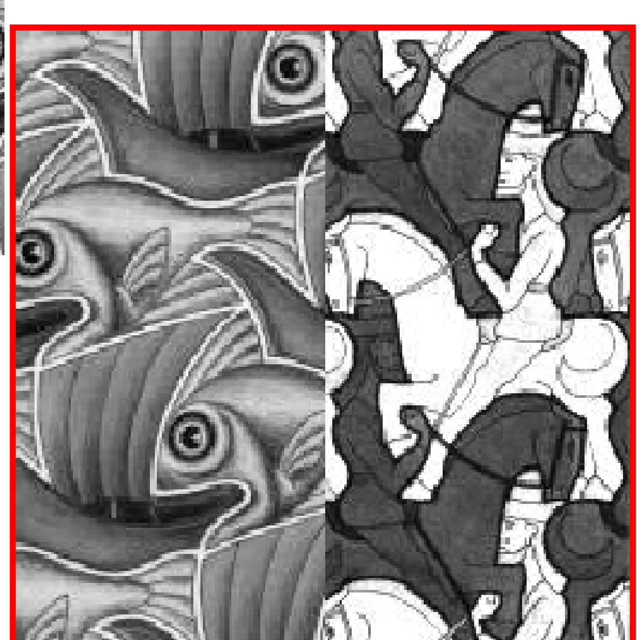
$$
\begin{aligned}
f(x,y) = \; & (1-a)(1-b) && f[i,j] \\
& +a(1-b) && f[i+1,j] \\
& +ab && f[i+1,j+1] \\
& +(1-a)b && f[i,j+1]
\end{aligned}
$$

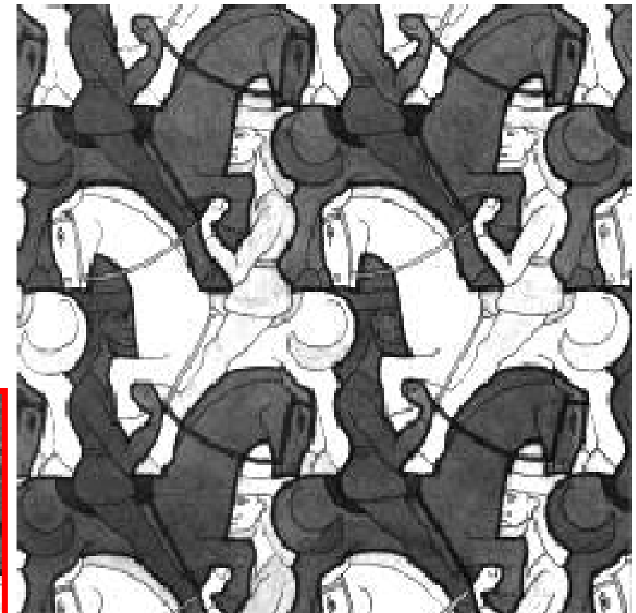
Postprocessing

- Planar Mosaic

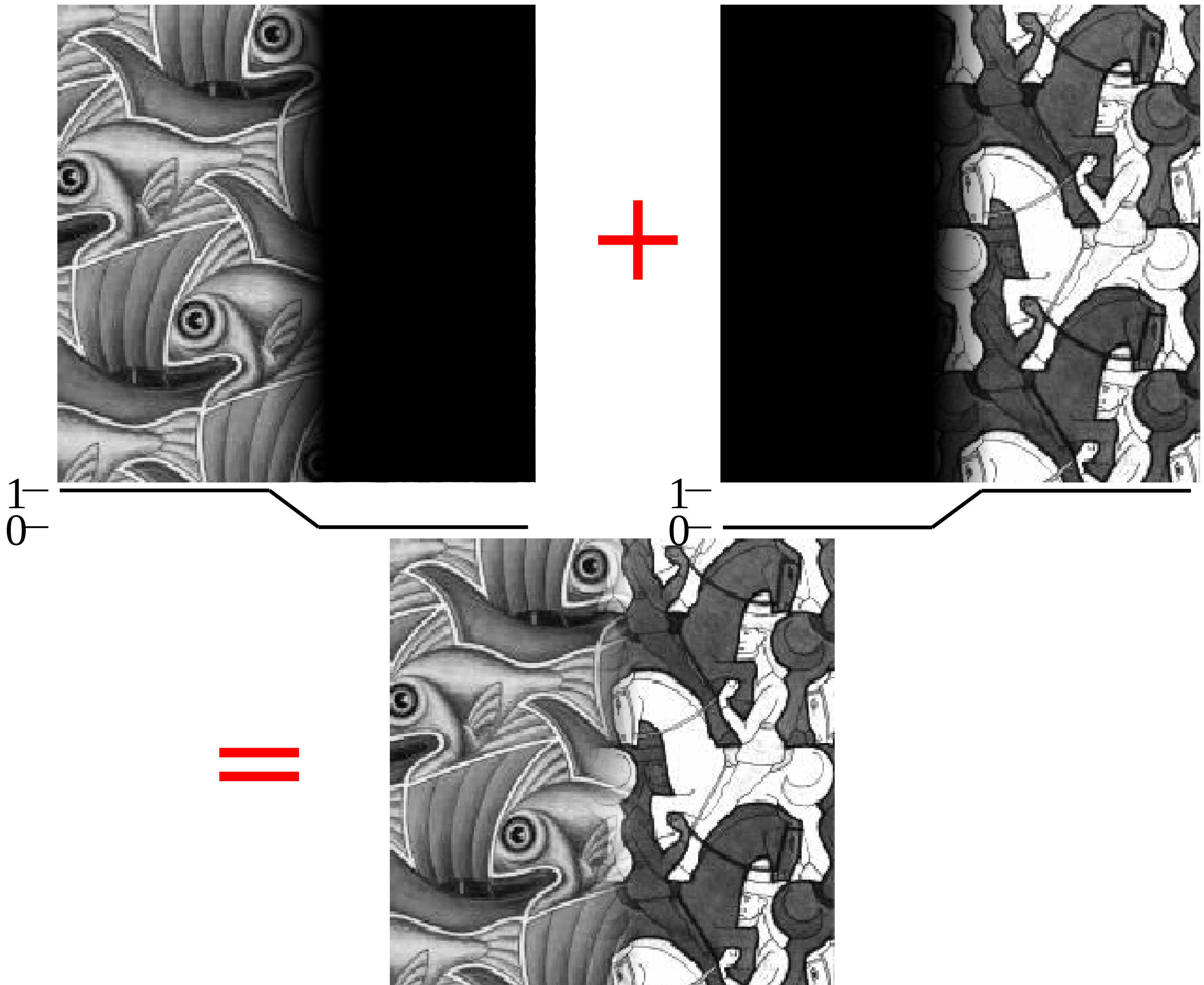
Note that the COP is the same

Image Blending

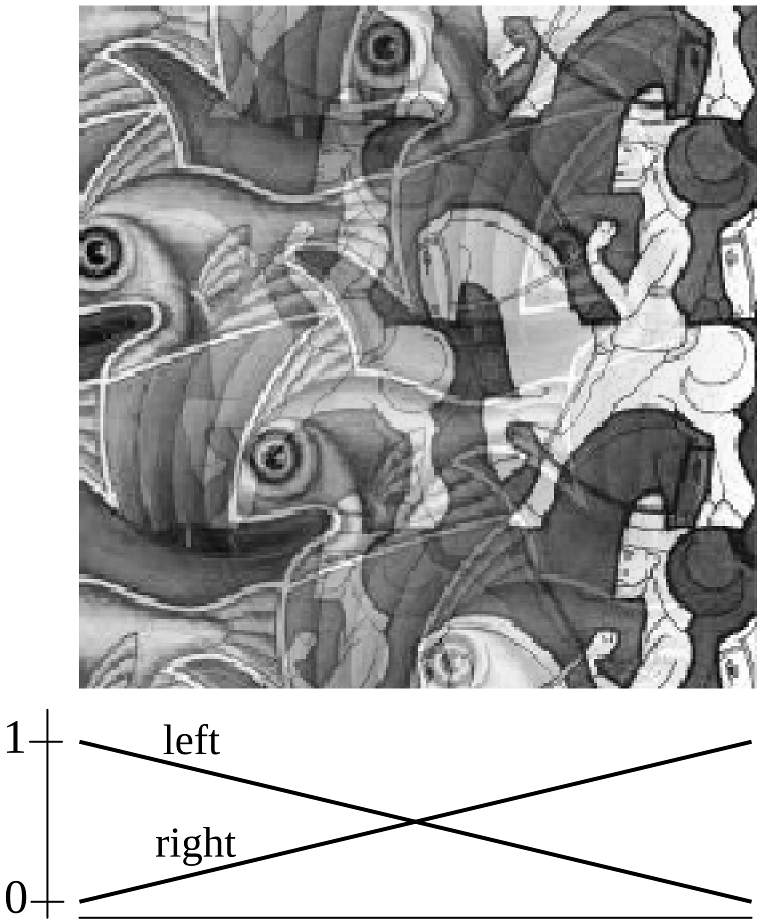


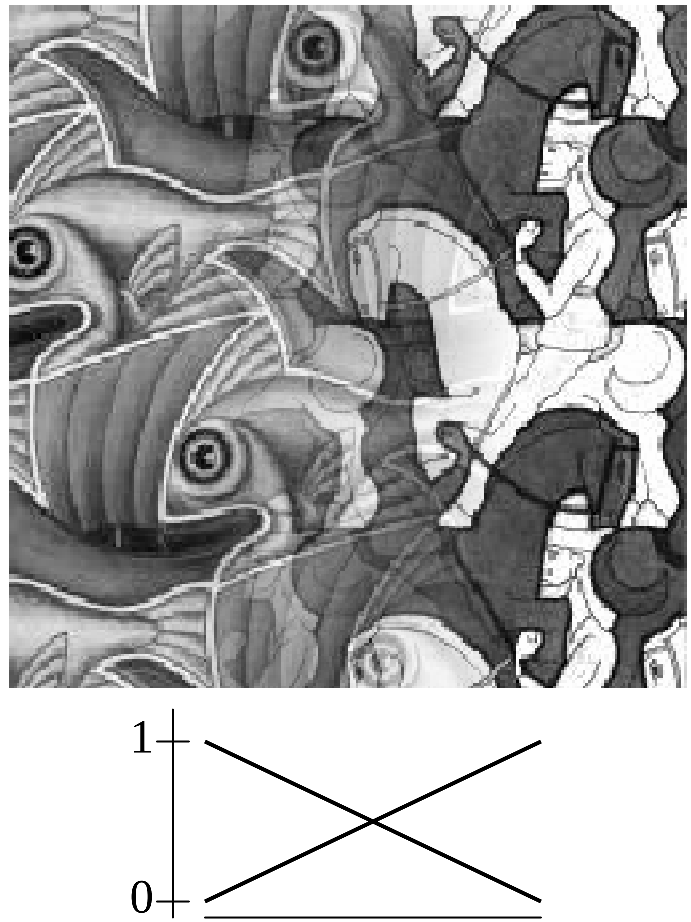


Feathering

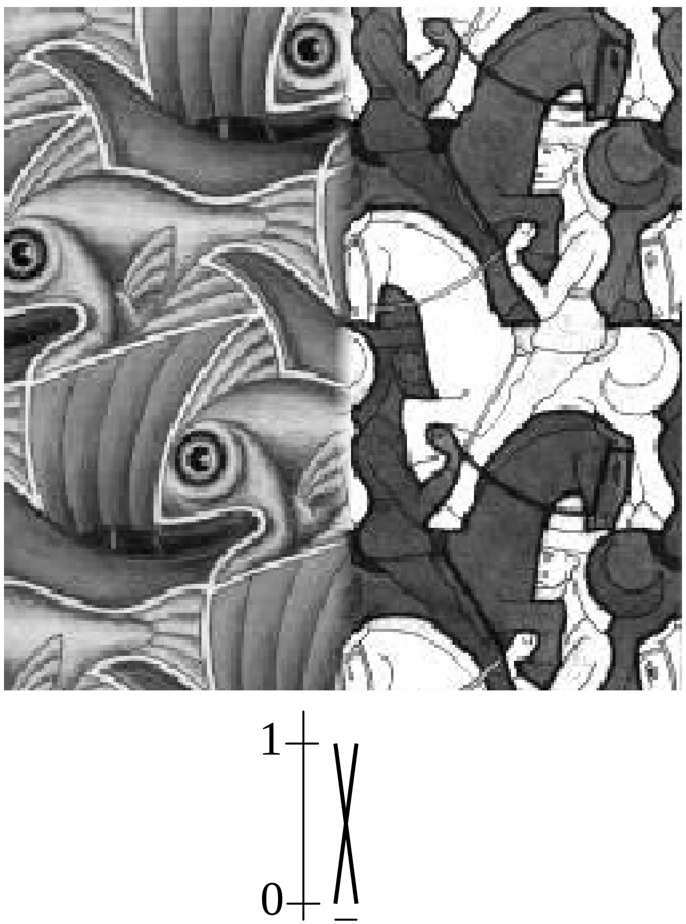


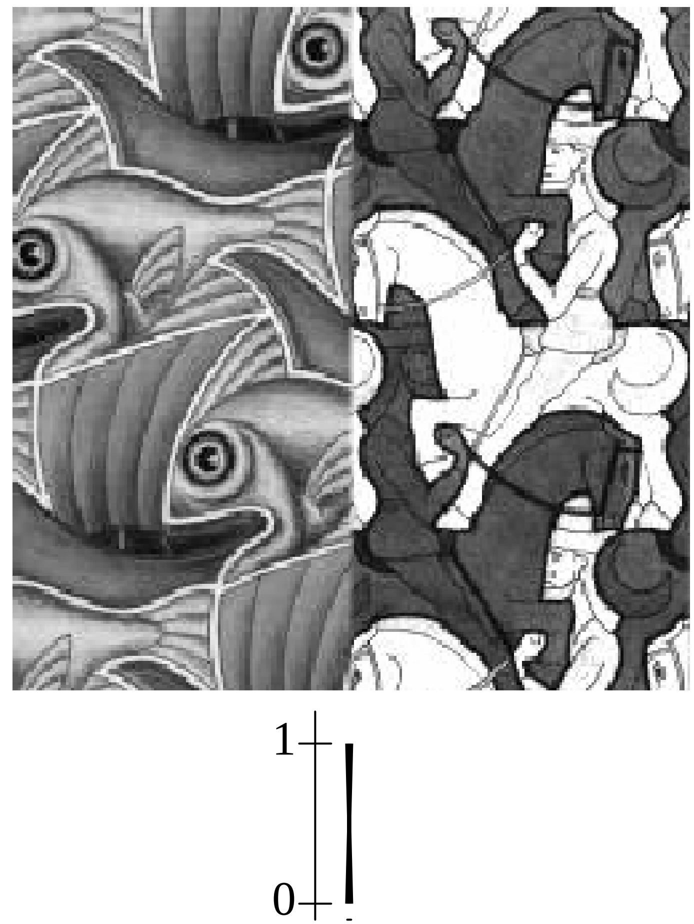
Effect of Window Size





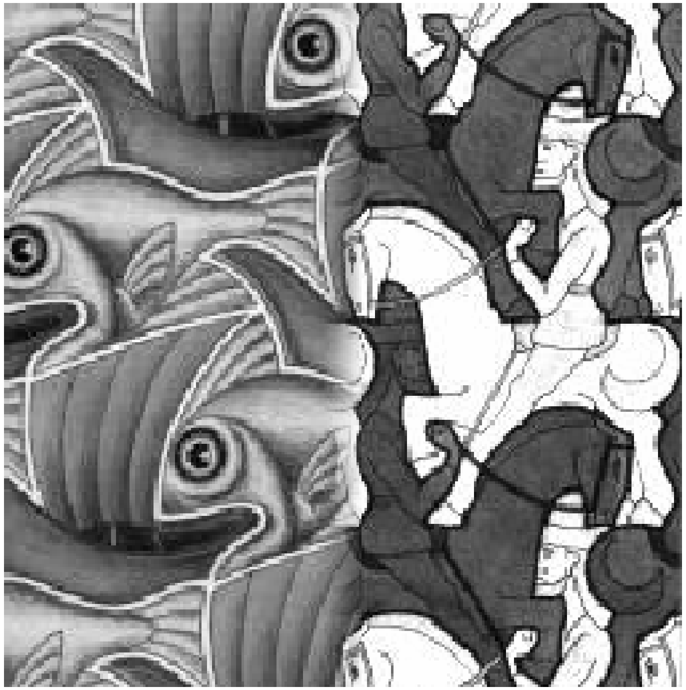
Effect of Window Size

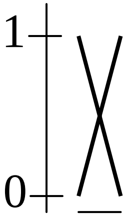




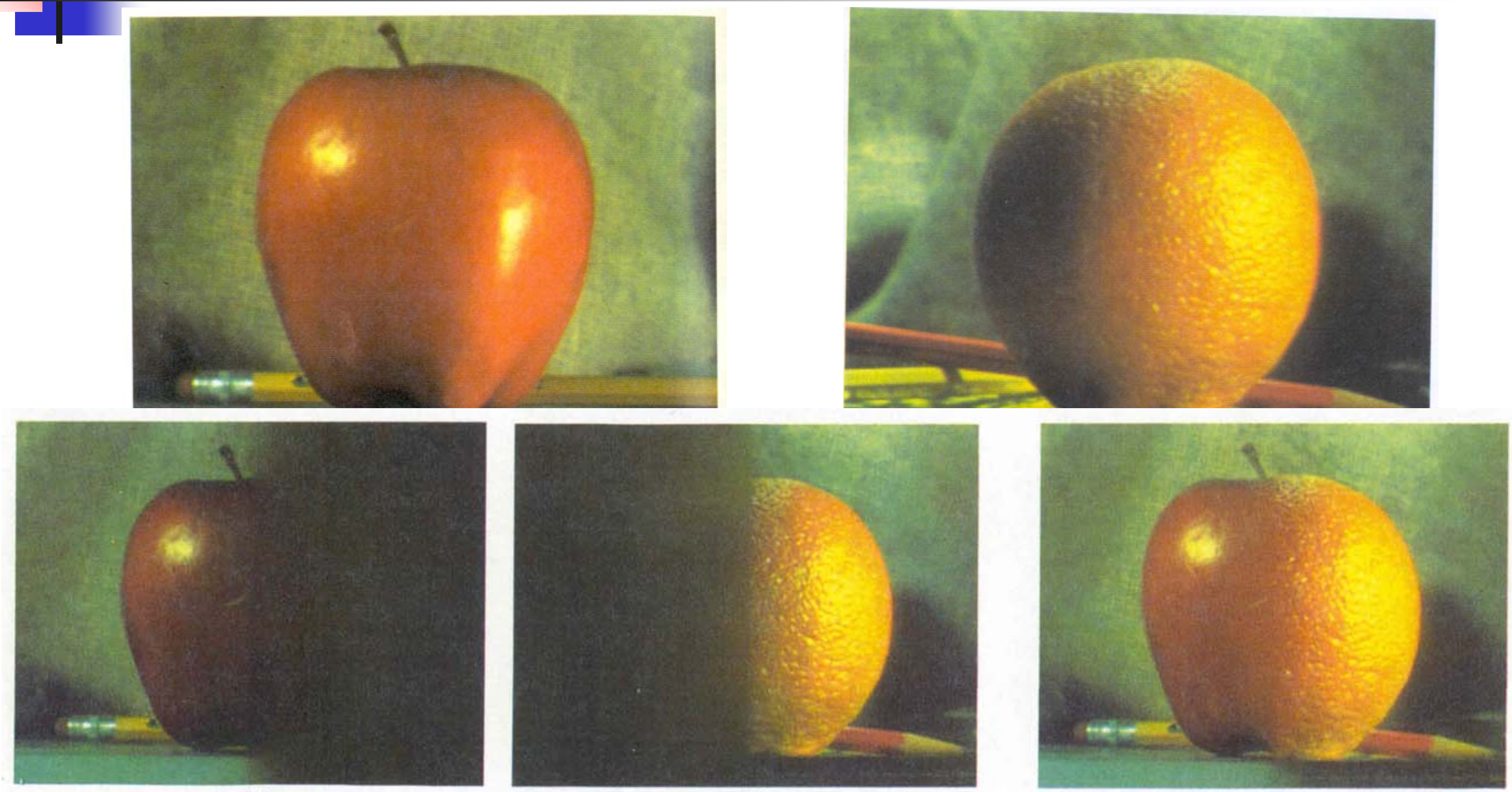
Good Window Size

- "Optimal" window: smooth but not ghosted
  - Doesn't always work...





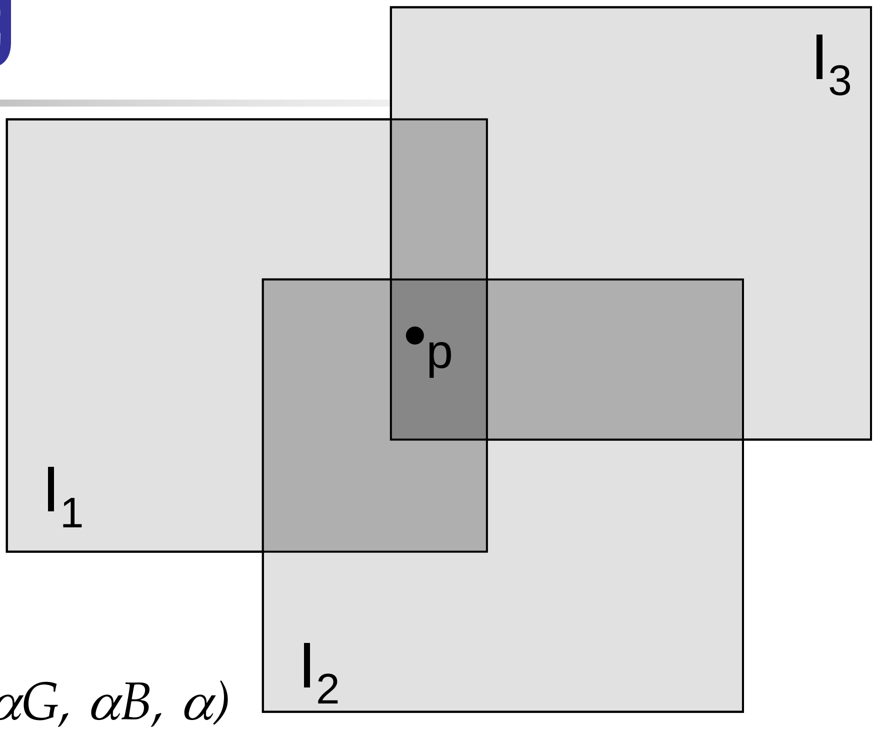
Pyramid Blending



- Create a Laplacian pyramid, blend each level

Burt, P. J. and Adelson, E. H., A multiresolution spline with applications to image mosaics, ACM Transactions on Graphics, 42(4), October 1983, 217-236.

Alpha Blending



Encoding blend weights: $I(x,y) = (\alpha R, \alpha G, \alpha B, \alpha)$

color at p = $$\frac{(\alpha_1 R_1,\ \alpha_1 G_1,\ \alpha_1 B_1) + (\alpha_2 R_2,\ \alpha_2 G_2,\ \alpha_2 B_2) + (\alpha_3 R_3,\ \alpha_3 G_3,\ \alpha_3 B_3)}{\alpha_1 + \alpha_2 + \alpha_3}$$

Implement this in two steps:

1. accumulate: add up the (α premultiplied) RGBα values at each pixel
2. normalize: divide each pixel's accumulated RGB by its α value

Example

- For more info: Perez et al, SIGGRAPH 2003
  - http://research.microsoft.com/vision/cambridge/papers/perez_siggraph03.pdf

Global alignment

- Register *all* pairwise overlapping images
- Use Optical center rotation model (one R per image)
- Use direct alignment (patch centers) or feature based
- *Infer* overlaps based on previous matches (incremental)
- Optionally *discover* which images overlap other images using feature selection (RANSAC)

Local alignment (deghosting)

- Use local optic flow to compensate for small motions [Shum & Szeliski, ICCV'98]

Figure 3: Deghosting a mosaic with motion parallax: (a) with parallax; (b) after single deghosting step (patch size 32); (c) multiple steps (sizes 32, 16 and 8).

Local alignment (deghosting)

- Use local optic flow to compensate for radial distortion [Shum & Szeliski, ICCV'98]

Figure 4: Deghosting a mosaic with optical distortion: (a) with distortion; (b) after multiple steps.

Image feathering

- Weight each image proportional to its distance from the edge (distance map)
- Cut out the appropriate region from each image and then blend together
- Problem: non-static background

Region-based de-ghosting

- Select only one image in *regions-of-difference* using weighted vertex cover [Uyttendaele *et al.*, CVPR'01]

(A) (B)

Figure 5 – (A) Ghosted mosaic. (B) Result of de-ghosting algorithm.

Region-based de-ghosting

- Select only one image in *regions-of-difference* using weighted vertex cover [Uyttendaele *et al.*, CVPR'01]

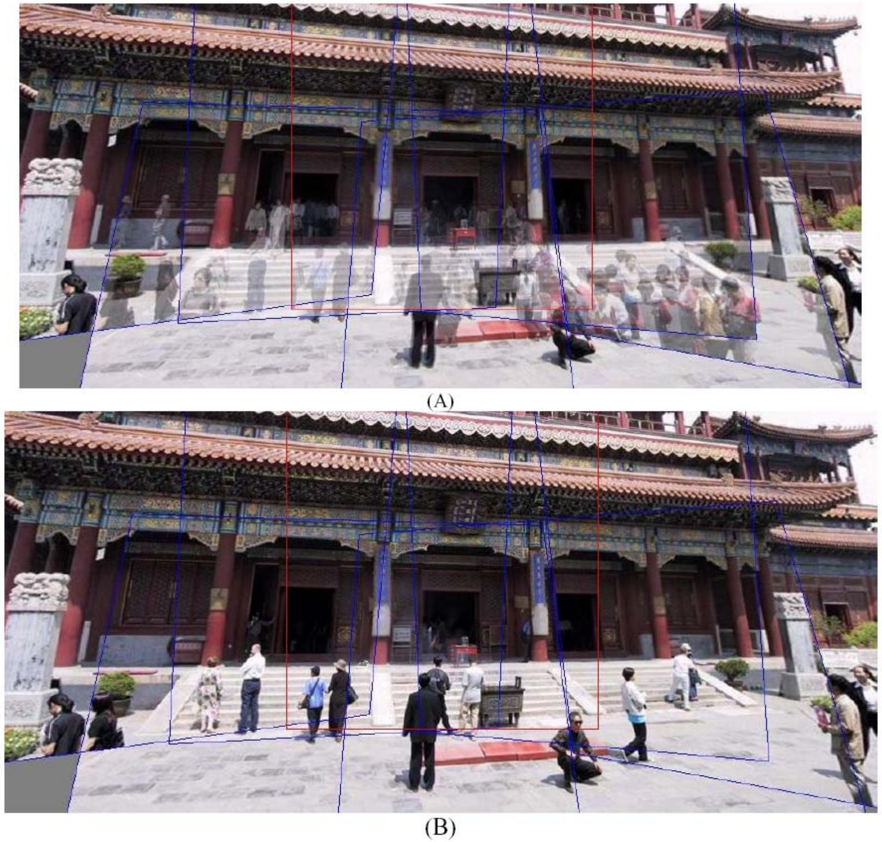
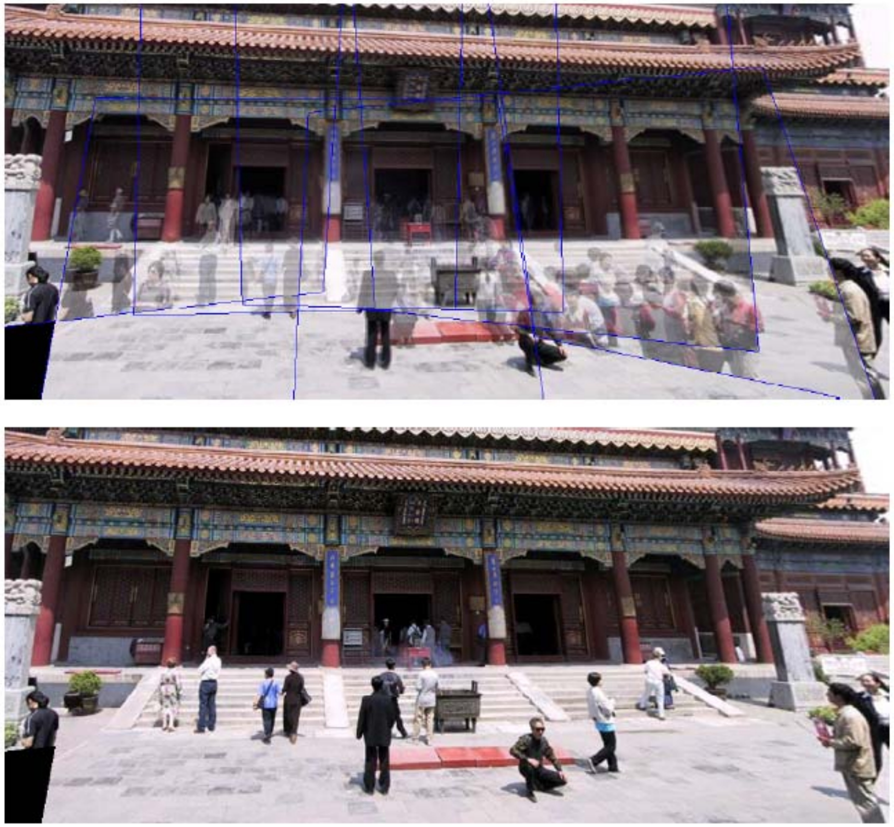


Figure 6 – (A) Ghosted mosaic. (B) Result of de-ghosting algorithm.

Cutout-based de-ghosting

- Select only one image per output pixel, using spatial continuity
- Blend across seams using gradient continuity (“Poisson blending”)
  [Agarwala *et al.*, SG’2004]



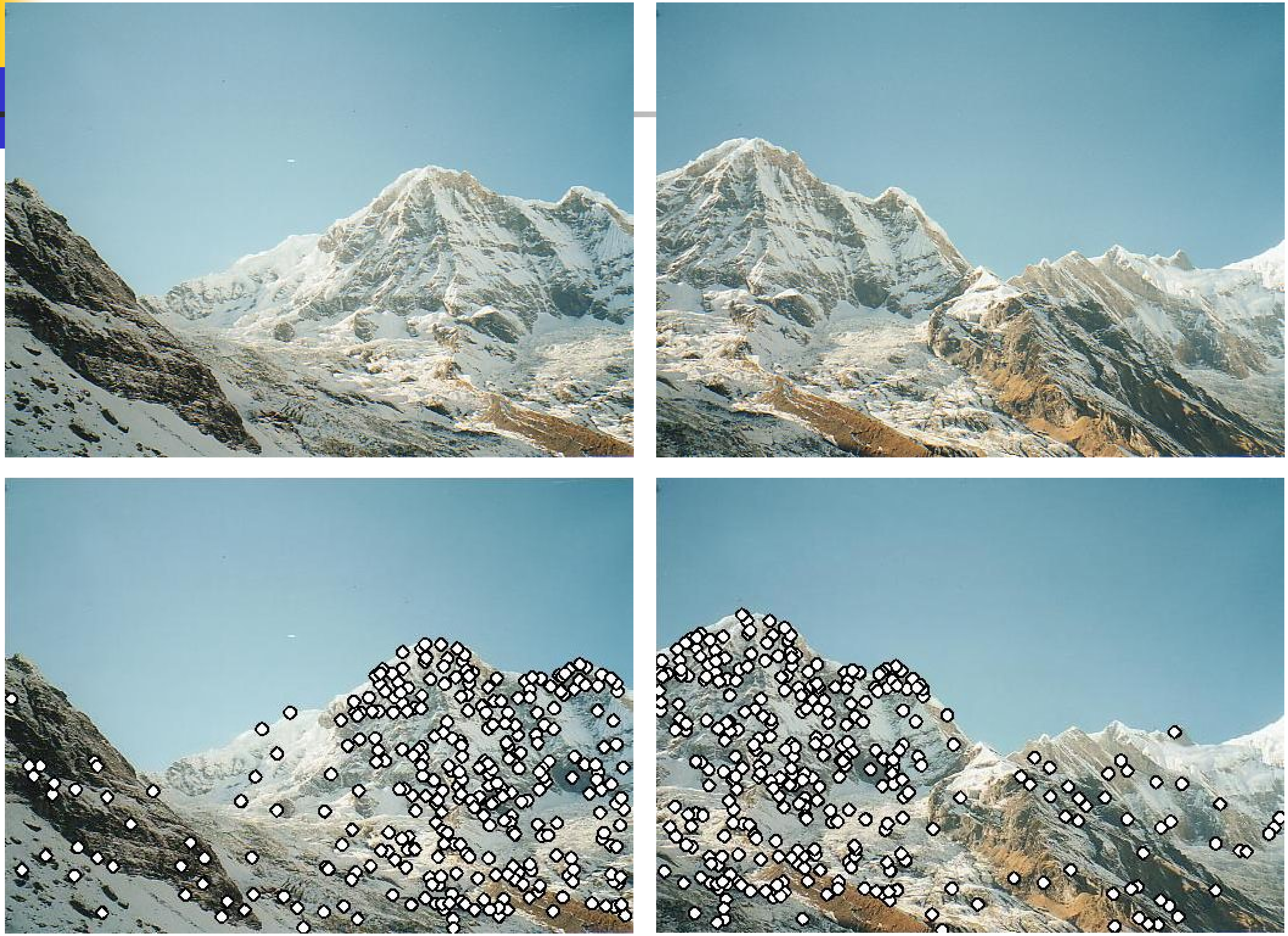
A general pipeline

Feature detection → Feature matching (Tracking) → Initial alignment → Refinement (Optimization) → Stitching (Warping) → Postprocessing (Deghosting)

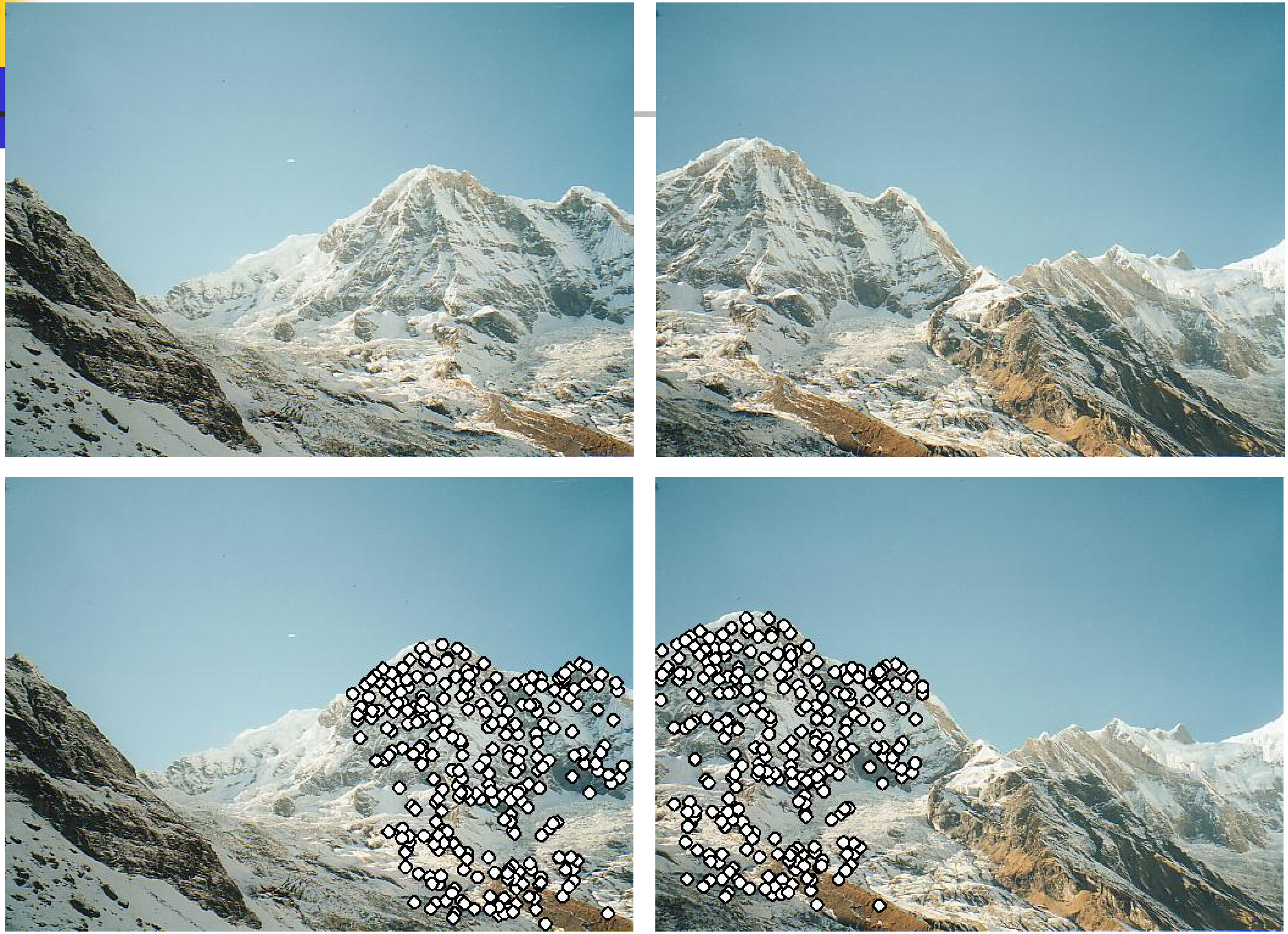
Which module is most difficult ?

MultiView/Extraction/matching/tracking/morphing/optimization /blending/inpainting/editing/…

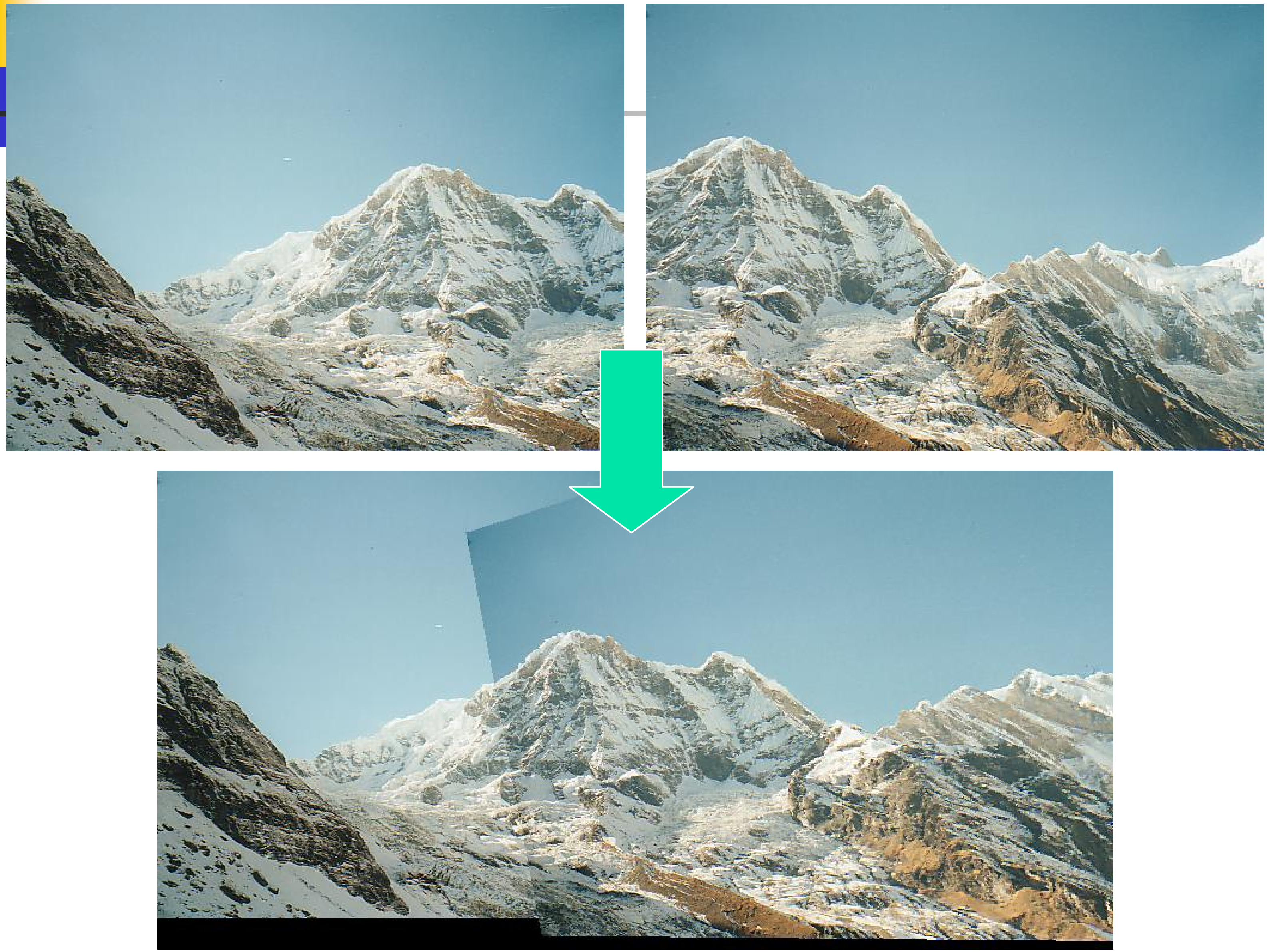
RANSAC motion model



RANSAC motion model

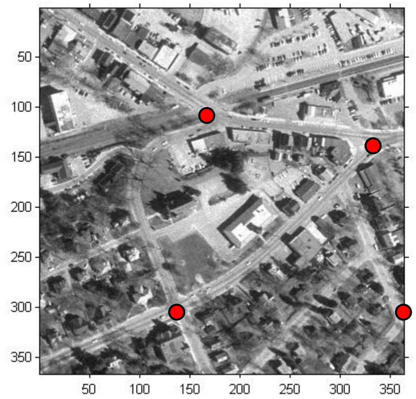


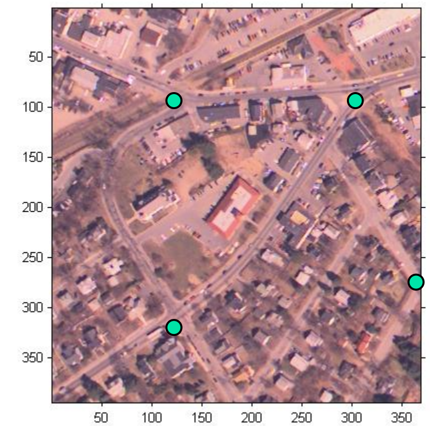
RANSAC motion model

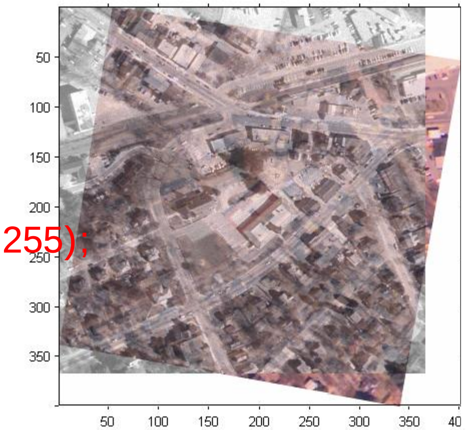


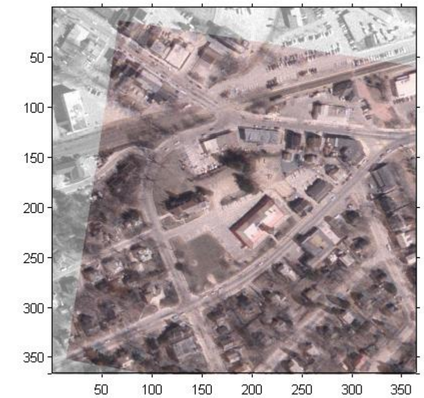
Matlab Demo (Break)

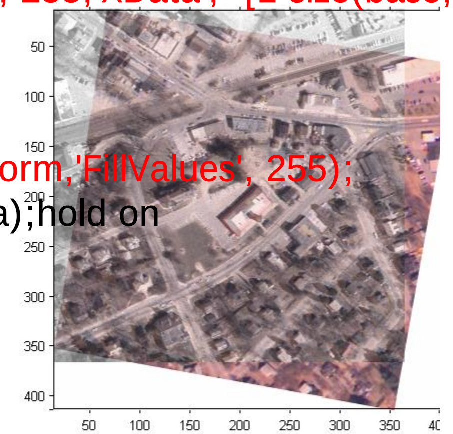
```
base = imread('westconcordorthophoto.png');
unregistered = imread('westconcordaerial.png');
iptsetpref('ImshowAxesVisible','on')
figure;imshow(base);
figure;imshow(unregistered);
load westconcordpoints;
tform = cp2tform(input_points, base_points, 'projective');
registered = imtransform(unregistered, tform,'FillValues', 255);
figure; imshow(registered);hold on
h = imshow(base, gray(256));
set(h, 'AlphaData', 0.6);
%appear misregistered
registered1 = imtransform(unregistered,tform,'FillValues', 255,'XData',  [1 size(base,2)], 'YData', [1 size(base,1)]);
figure; imshow(registered1);hold on
h = imshow(base, gray(256));
set(h, 'AlphaData', 0.6)
[registered2 xdata ydata] = imtransform(unregistered, tform,'FillValues', 255);
figure; imshow(registered2, 'XData', xdata, 'YData', ydata);hold on
h = imshow(base, gray(256));
set(h, 'AlphaData', 0.6);
ylim = get(gca, 'YLim');
set(gca, 'YLim', [0.5 ylim(2)]);
```

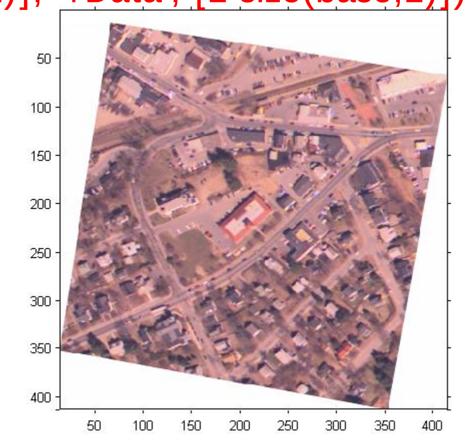




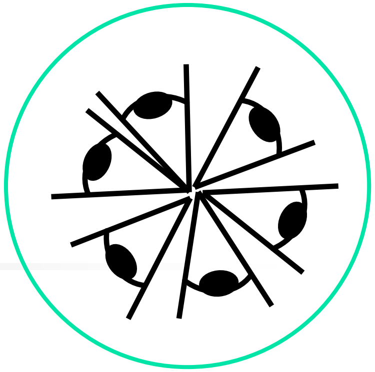








Wide-angle Imaging

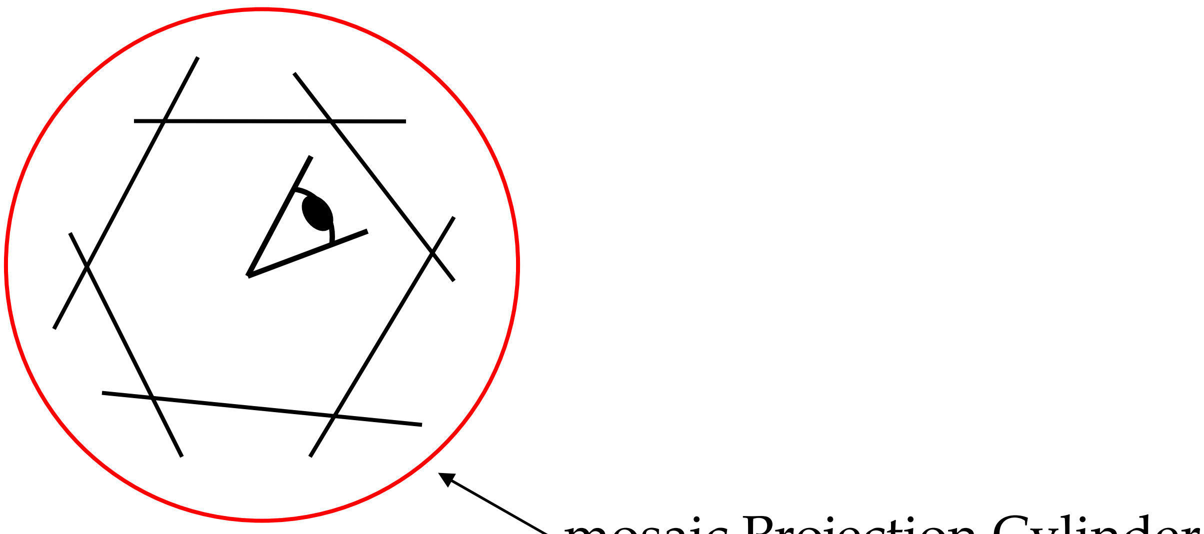


- Goal
  - Stitch together several images into a seamless composite

 +  + … +  =

Panoramas

- What if you want a 360° field of view?



Radial Distortion

- Wide-angle view input is better
- Suffers from radial distortion

(x, y): ideal image coordinates

(in normalized coordinates; focal length=1)

(x', y'): distorted image coordinates

$$x' = x\left(1 + \kappa_1 r^2 + \kappa_2 r^4\right)$$

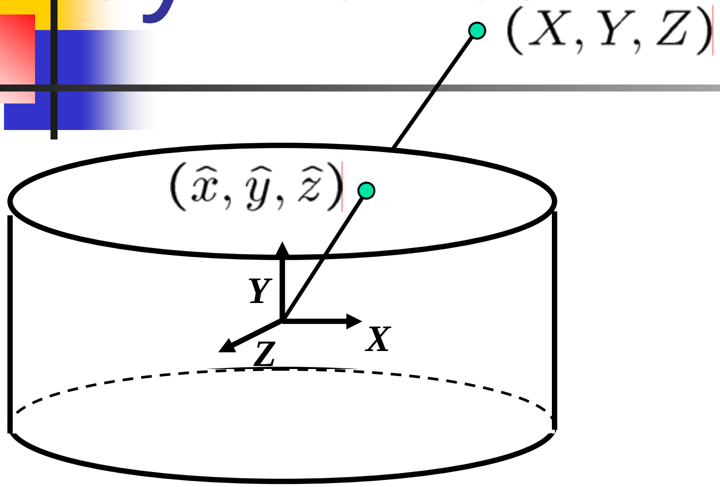
$$y' = y\left(1 + \kappa_1 r^2 + \kappa_2 r^4\right)$$

$$r^2 = x^2 + y^2$$

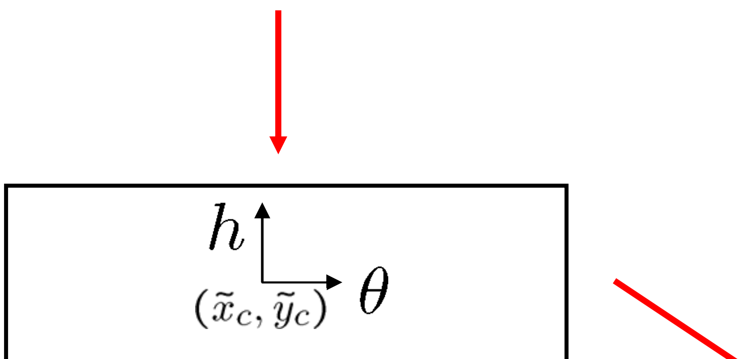
Get the radial distortion parameters as well as the focal length through camera calibration

* Optical center is not necessarily the image center, too!

Cylindrical Projection



unit cylinder



unwrapped cylinder

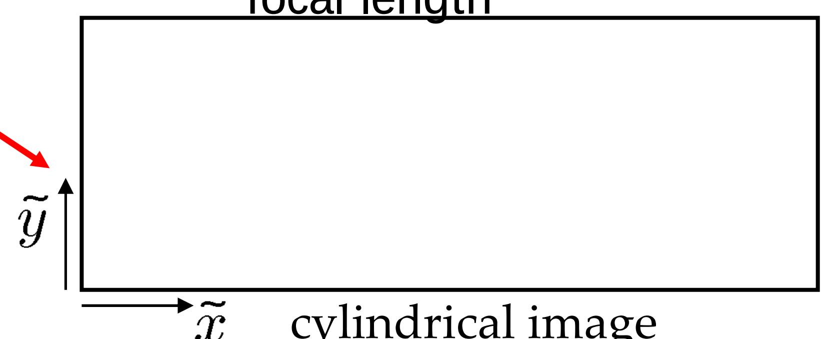
- Map 3D point (X,Y,Z) onto cylinder

$$(\hat{x},\hat{y},\hat{z}) = \frac{1}{\sqrt{X^2+Z^2}}(X,Y,Z)$$

- Convert to cylindrical coordinates

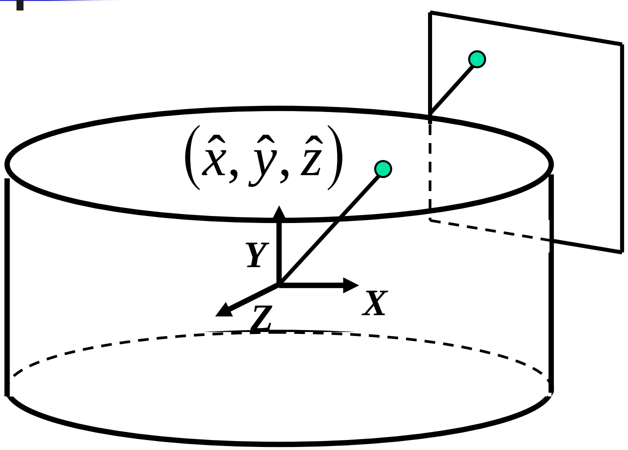
$$(sin\theta, h, cos\theta) = (\hat{x},\hat{y},\hat{z})$$

- Convert to cylindrical image coordinates $(\tilde{x},\tilde{y}) = (s\theta, sh) + (\tilde{x}_c,\tilde{y}_c)$
  - *s* defines size of the final image
    - often convenient to set *s* = camera focal length

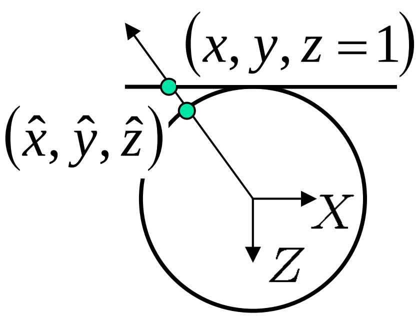


cylindrical image

Cylindrical Reprojection



side view



top-down view

- Normalize the image coordinates

$$x = \frac{x' - \frac{W}{2}}{f'}, \quad y = \frac{y' - \frac{H}{2}}{f'}, \quad z = f = \frac{f'}{f'} = 1$$

- Forward warping

$$(\hat{x}, \hat{y}, \hat{z}) = \frac{1}{\sqrt{x^2 + 1^2}}(x, y, 1)$$

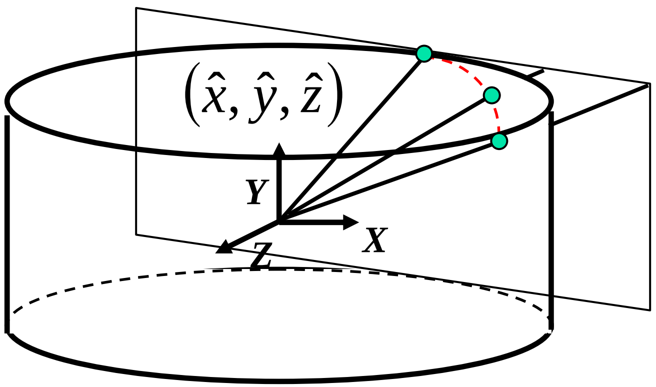
$$= (\sin\theta, h, \cos\theta) \Rightarrow (\tilde{x}, \tilde{y})$$

- Inverse warping

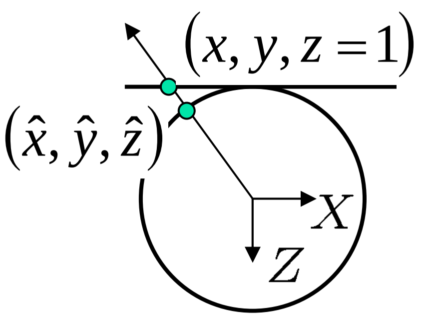
Derive $(\tilde{x}, \tilde{y}) \Rightarrow (\hat{x}, \hat{y}, \hat{z}) \Rightarrow (x, y)$

Inverse warping + interpolation!
s=*f* minimizes the scaling near the center of image

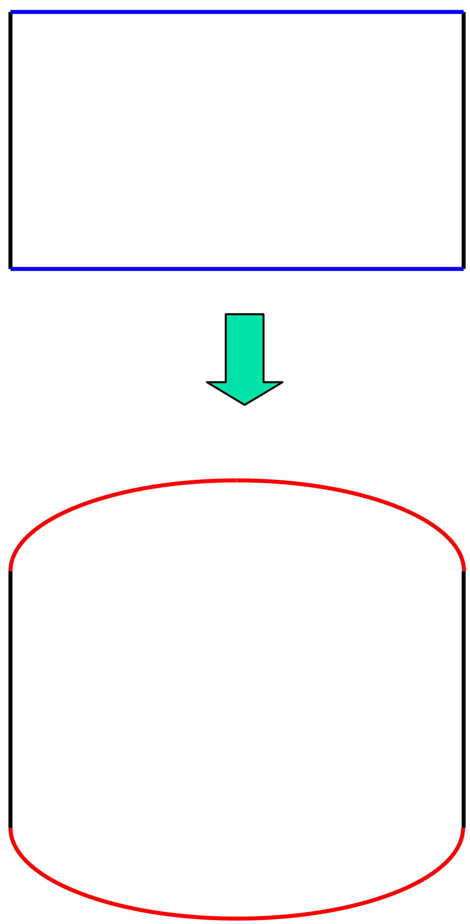
Cylindrical Reprojection



side view



top-down view



Cylindrical Panoramas

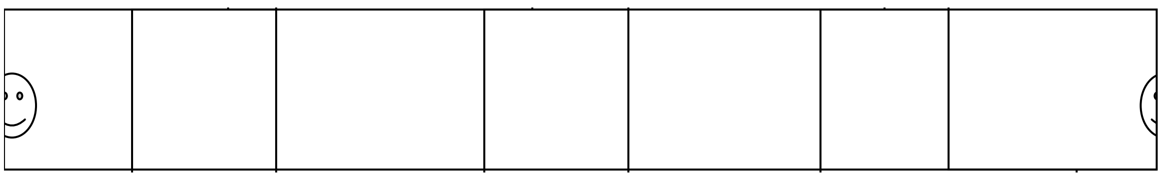
- Map image to cylindrical or spherical coordinates
  - need *known* focal length

Image 384x300 **f = 180 (pixels)** **f = 280** **f = 380**

Image Stitching

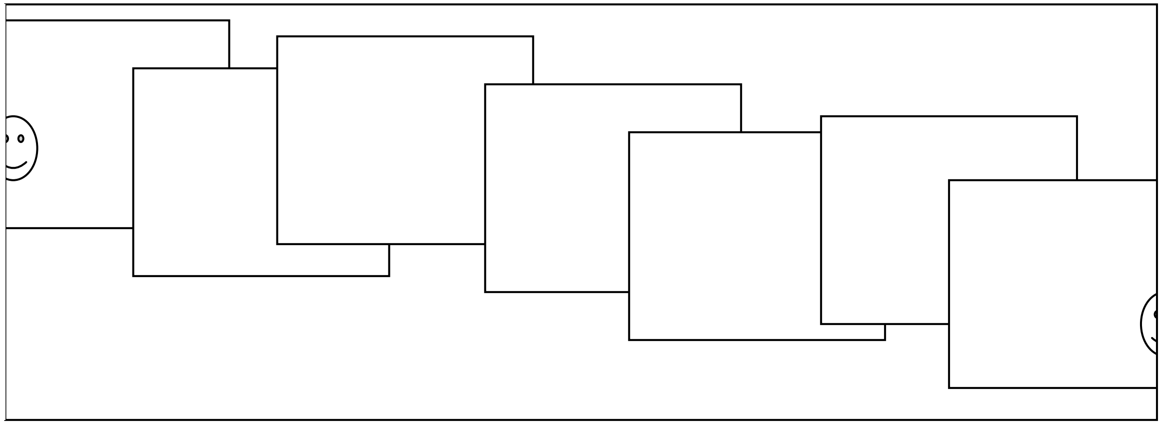
1. Align and paste the images on a cylinder
2. Blend the images together

Assembling the Panorama



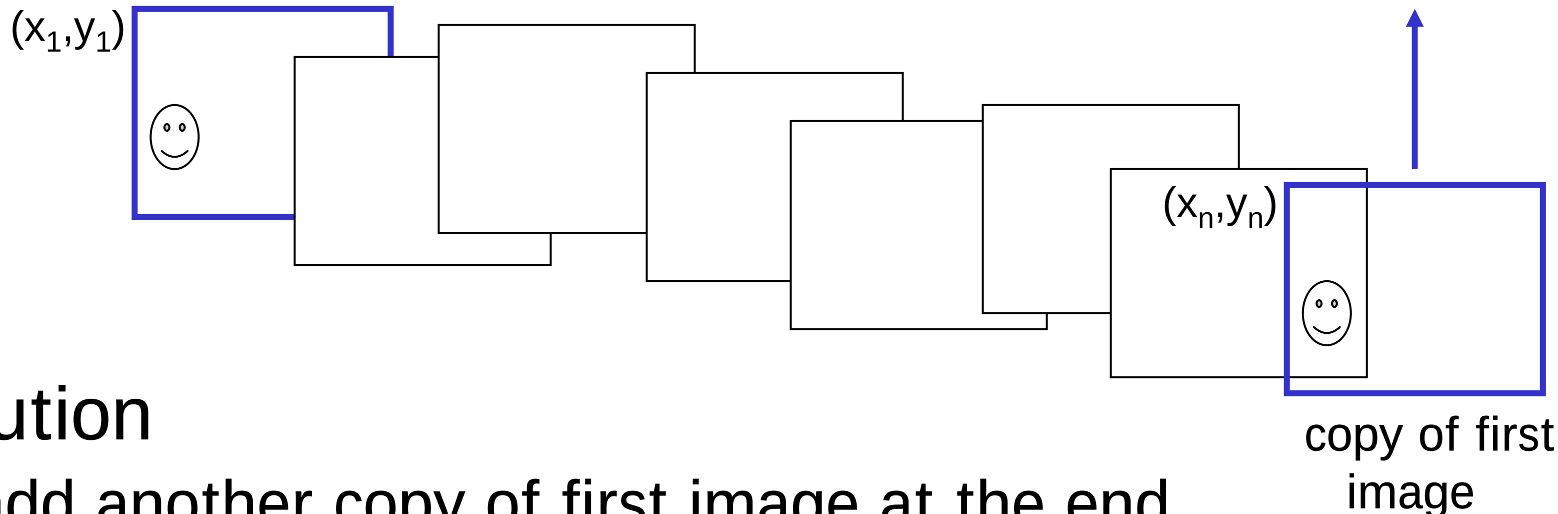
- Stitch pairs together, blend, then crop

Problem: Drift



- Error accumulation
  - small errors accumulate over time

Problem: Drift



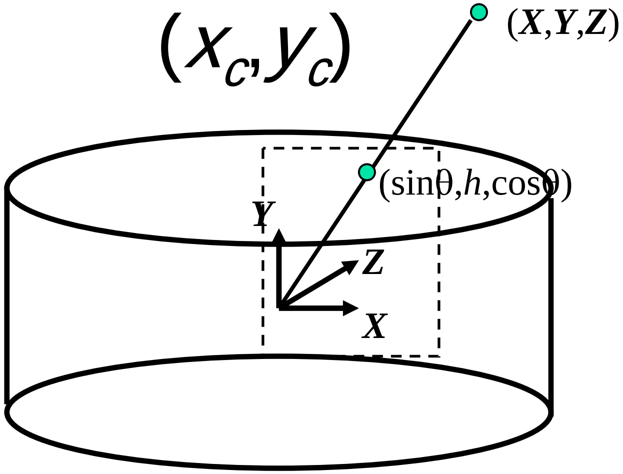
- Solution
  - add another copy of first image at the end
  - this gives a constraint: $y_n = y_1$
  - there are a bunch of ways to solve this problem
    - add displacement $(y_1 - y_n)/(n-1)$ to each image after the first
    - **compute a global warp: $y' = y + ax$**
    - run a big optimization problem, incorporating this constraint
      - best solution, but more complicated (bundle adjustment)

Full-view Panorama

Different Projections are Possible

Cylindrical warping

- Given focal length f and image center (x_c, y_c)



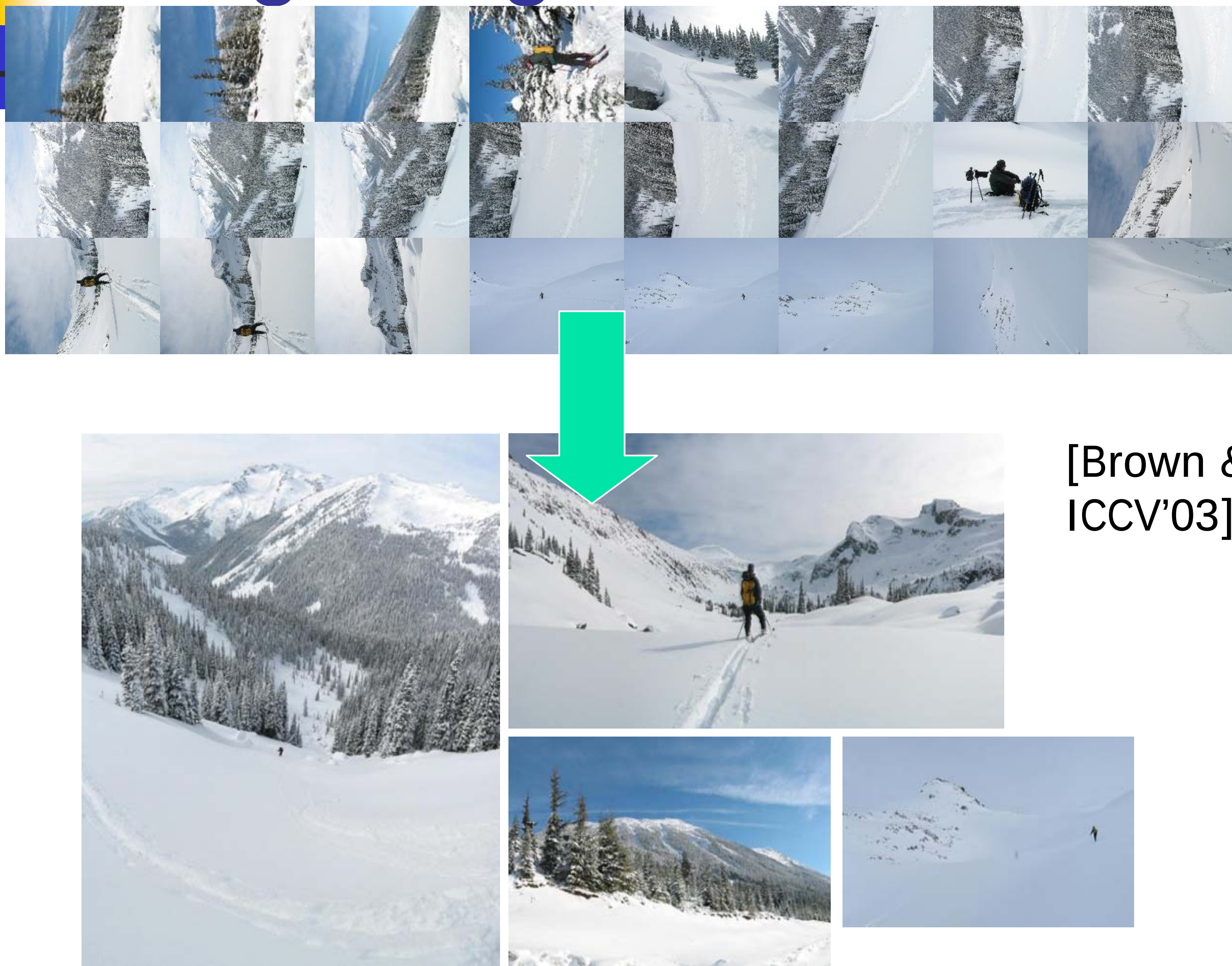
$$
\begin{aligned}
\theta &= (x_{cyl} - x_c)/f \\
h &= (y_{cyl} - y_c)/f \\
\hat{x} &= \sin\theta \\
\hat{y} &= h \\
\hat{z} &= \cos\theta \\
x &= f\hat{x}/\hat{z} + x_c \\
y &= f\hat{y}/\hat{z} + y_c
\end{aligned}
$$

Recognizing Panoramas

Matthew Brown & David Lowe

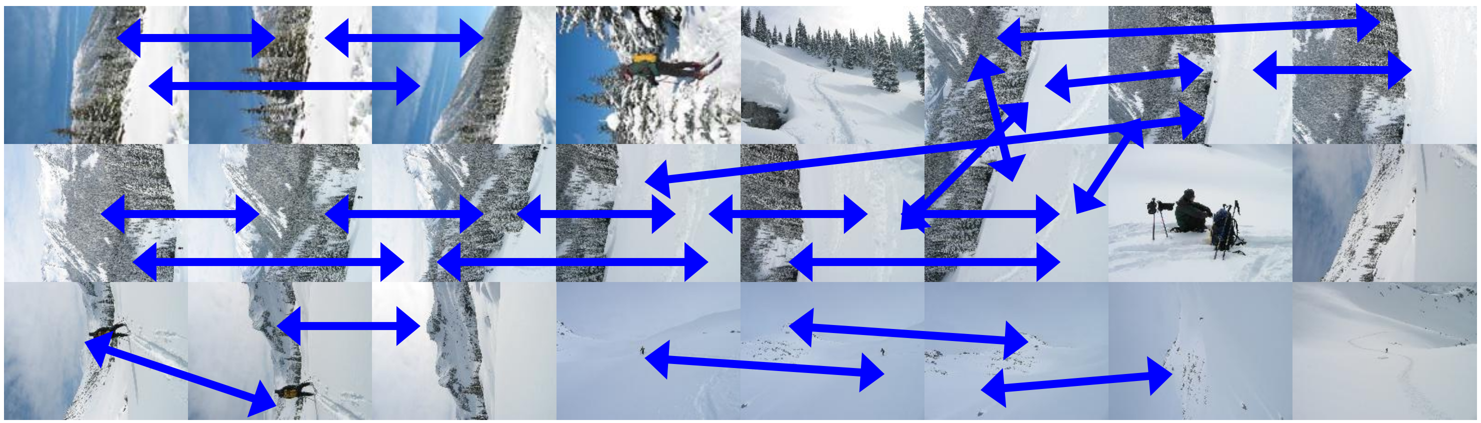
ICCV'2003

Recognizing Panoramas

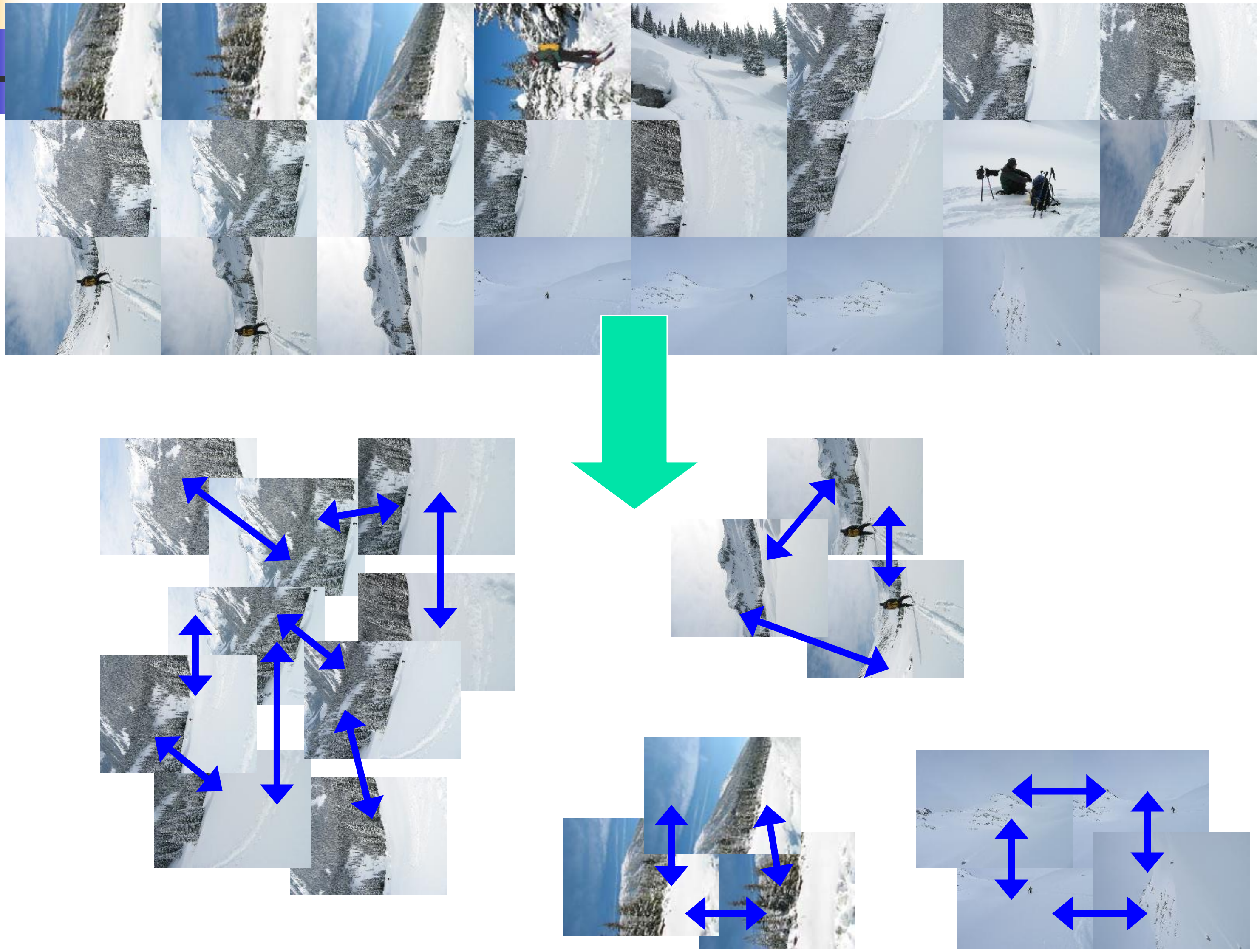


[Brown & Lowe, ICCV’03]

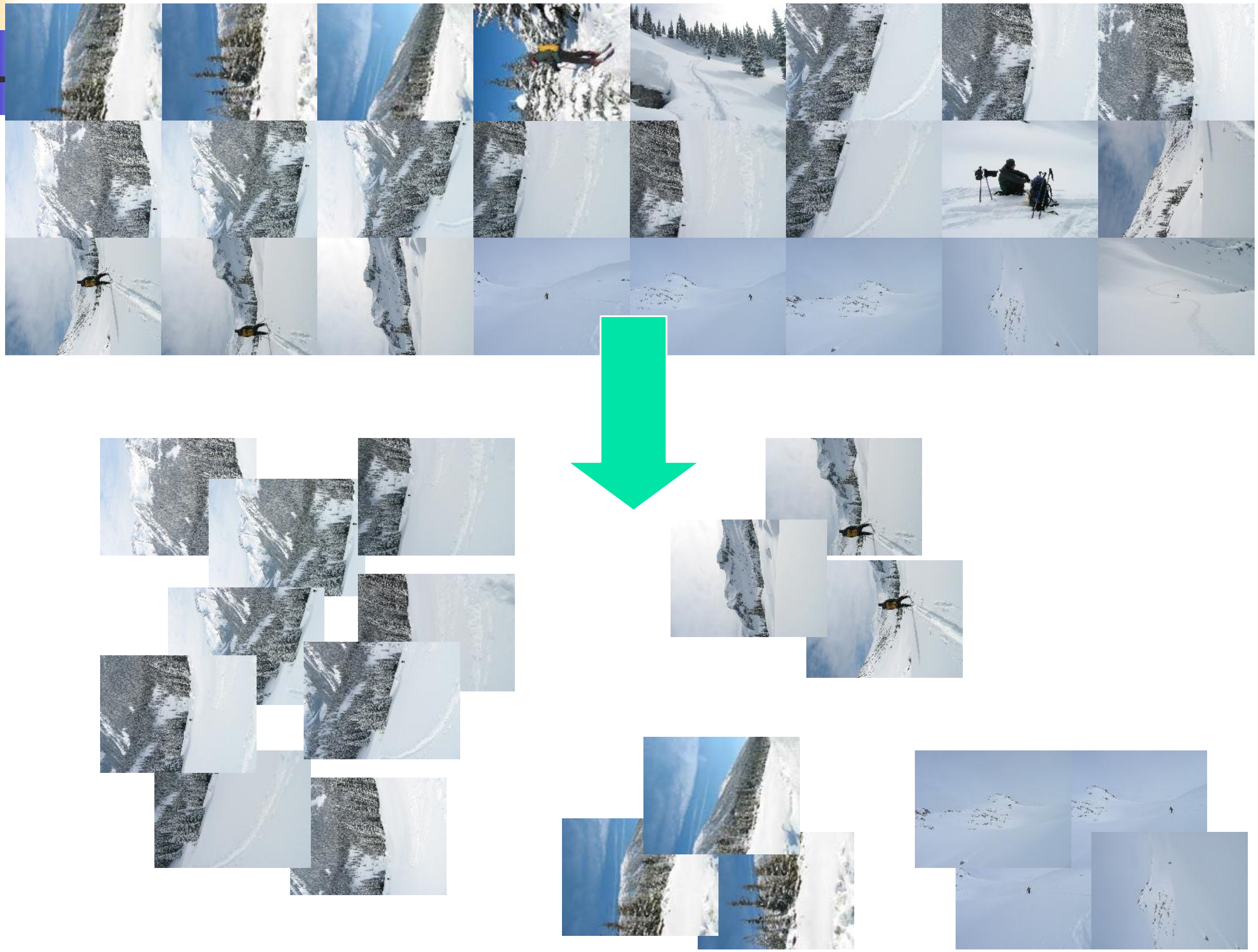
Finding the panoramas



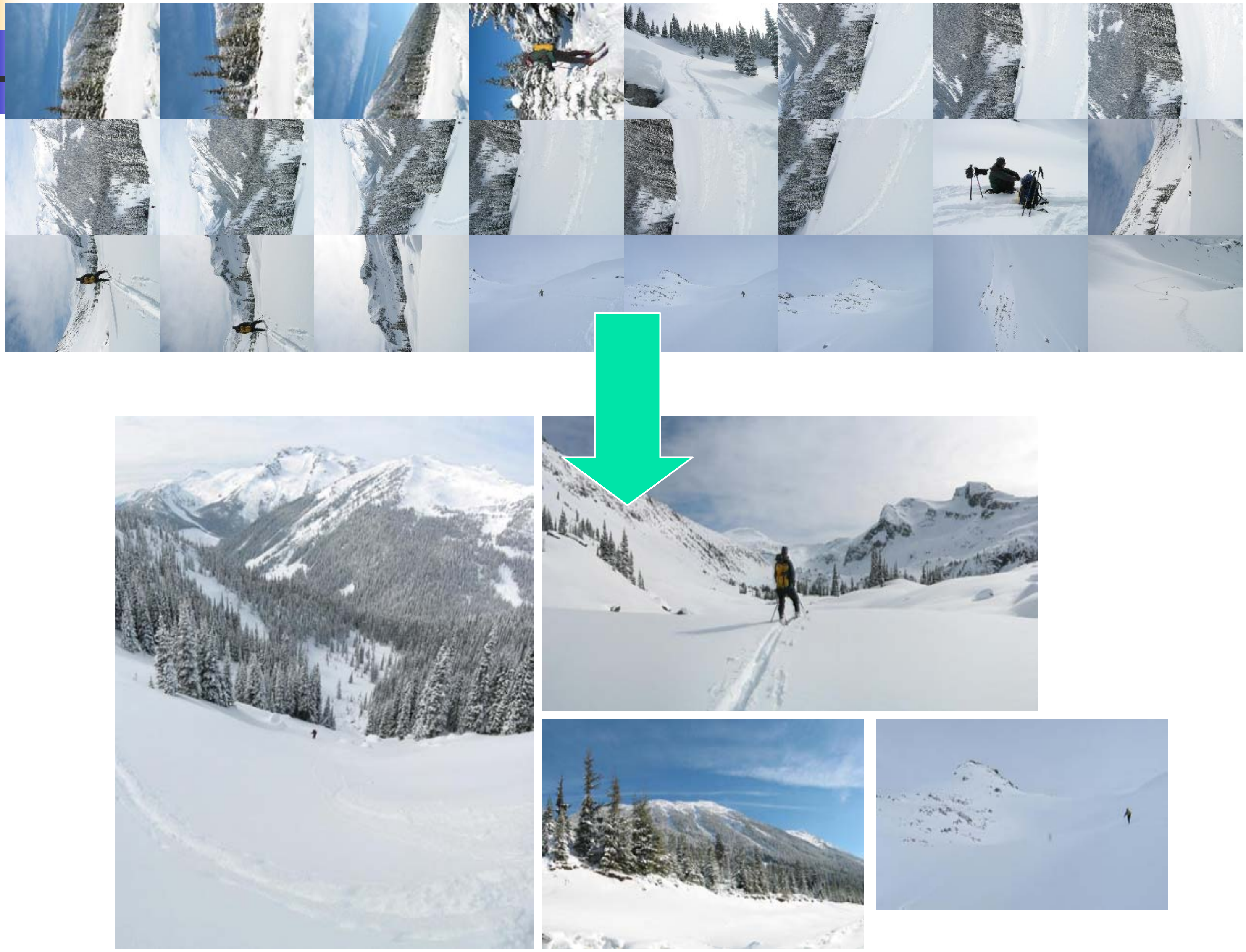
Finding the panoramas



Finding the panoramas



Finding the panoramas



System components

- Feature detection and description
  - more uniform point density
- Fast matching (hash table)
- RANSAC filtering of matches
- Intensity-based verification
- Incremental bundle adjustment

- [M. Brown, R. Szeliski, and S. Winder. Multi-image matching using multi-scale oriented patches, CVPR'2005]

Multi-Scale Oriented Patches

- Interest points
  - Multi-scale Harris corners
  - Orientation from blurred gradient
  - Geometrically invariant to similarity transforms
- Descriptor vector
  - Bias/gain normalized sampling of local patch (8x8)
  - Photometrically invariant to affine changes in intensity

Cutout-based compositing

- Photomontage [Agarwala *et al.*, SG'2004]
- Interactively blend *different* images: group portraits

Figure 1 From a set of five source images (of which four are shown on the left), we quickly create a composite family portrait in which everyone is smiling and looking at the camera (right). We simply flip through the stack and coarsely draw strokes using the *designated source* image objective over the people we wish to add to the composite. The user-applied strokes and computed regions are color-coded by the borders of the source images on the left (middle).

Cutout-based compositing

- Photomontage [Agarwala *et al.*, SG'2004]
- Interactively blend *different* images:
  focus settings

Figure 2 A set of macro photographs of an ant (three of eleven used shown on the left) taken at different focal lengths. We use a global *maximum contrast* image objective to compute the graph-cut composite automatically (top left, with an inset to show detail, and the labeling shown directly below). A small number of remaining artifacts disappear after gradient-domain fusion (top, middle). For comparison we show composites made by Auto-Montage (top, right), by Haeberli's method (bottom, middle), and by Laplacian pyramids (bottom, right). All of these other approaches have artifacts; Haeberli's method creates excessive noise, Auto-Montage fails to attach some hairs to the body, and Laplacian pyramids create halos around some of the hairs.

Cutout-based compositing

- Photomontage [Agarwala *et al.*, SG'2004]
- Interactively blend *different* images:
  people's faces

Figure 6 We use a set of portraits (first row) to mix and match facial features, to either improve a portrait, or create entirely new people. The faces are first hand-aligned, for example, to place all the noses in the same location. In the first two images in the second row, we replace the closed eyes of a portrait with the open eyes of another. The user paints strokes with the *designated source* objective to specify desired features. Next, we create a fictional person by combining three source portraits. Gradient-domain fusion is used to smooth out skin tone differences. Finally, we show two additional mixed portraits.

Final thought: What is a “panorama”?

- Tracking a subject

- Repeated (best) shots
- Multiple exposures

- “Infer” what photographer wants?

Optional Assignments: Image registration

- Goal:
  - affine registration
  - Perspective registration
  - Panorama creation
- Techniques:
  - Feature selection and matching (Ransac)
  - Solving and making transformation
  - Post processing (Blending…)
  - ……