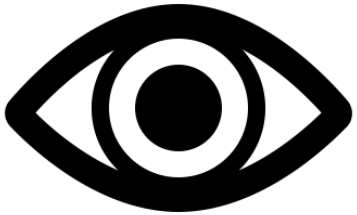


Advanced Image Processing

Cameras, multiple view and motion

SYDE 671

Photo Tourism

Exploring photo collections in 3D

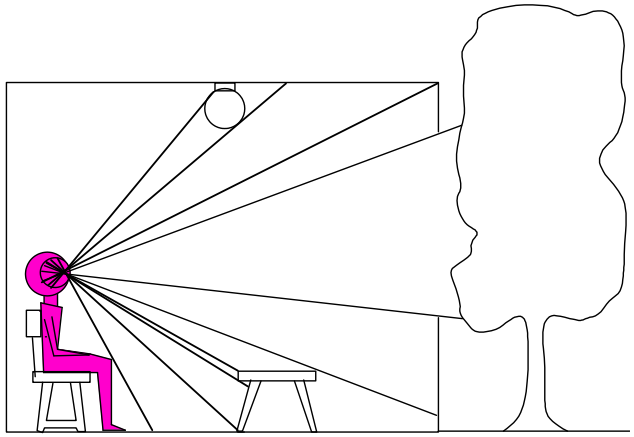
Noah Snavely Steven M. Seitz Richard Szeliski
University of Washington *Microsoft Research*

SIGGRAPH 2006

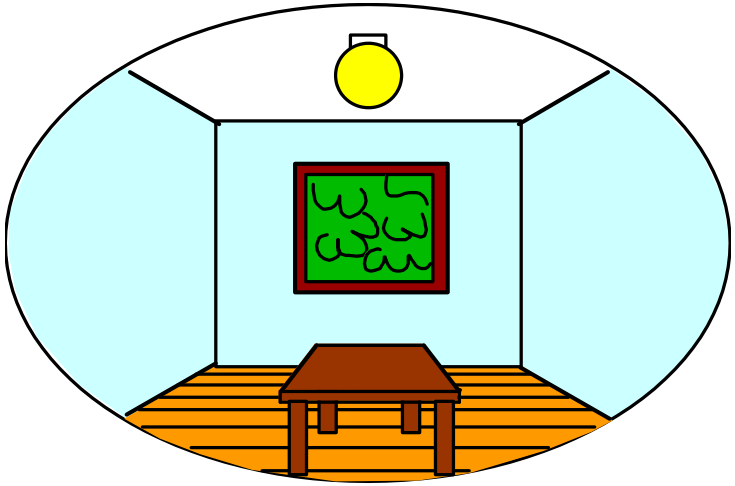
Dimensionality Reduction Machine (3D to 2D)

3D world

2D image



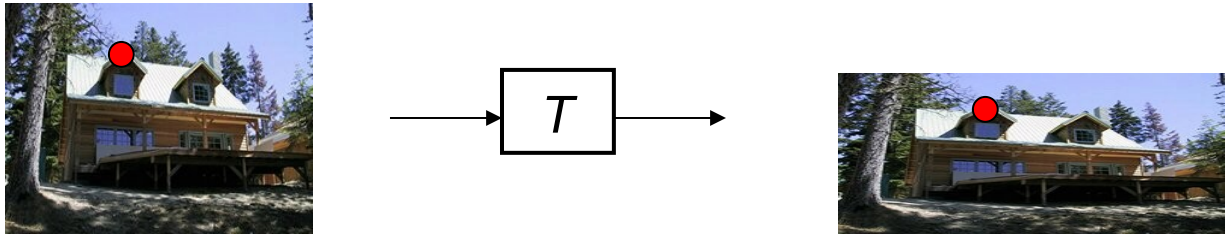
Point of observation



Holbein’s The Ambassadors - 1533

Holbein’s The Ambassadors – Memento Mori

Parametric (global) transformations



p = (x,y)

p’ = (x’,y’)

Transformation T is a coordinate-changing machine:

p’ = *T*(p)

What does it mean that *T* is global?

- *T* is the same for any point p

T can be described by just a few numbers (parameters)

For linear transformations, we can represent T as a matrix

p’ = **T**p

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

Common transformations

Original

Transformed

Translation

Rotation

Scaling

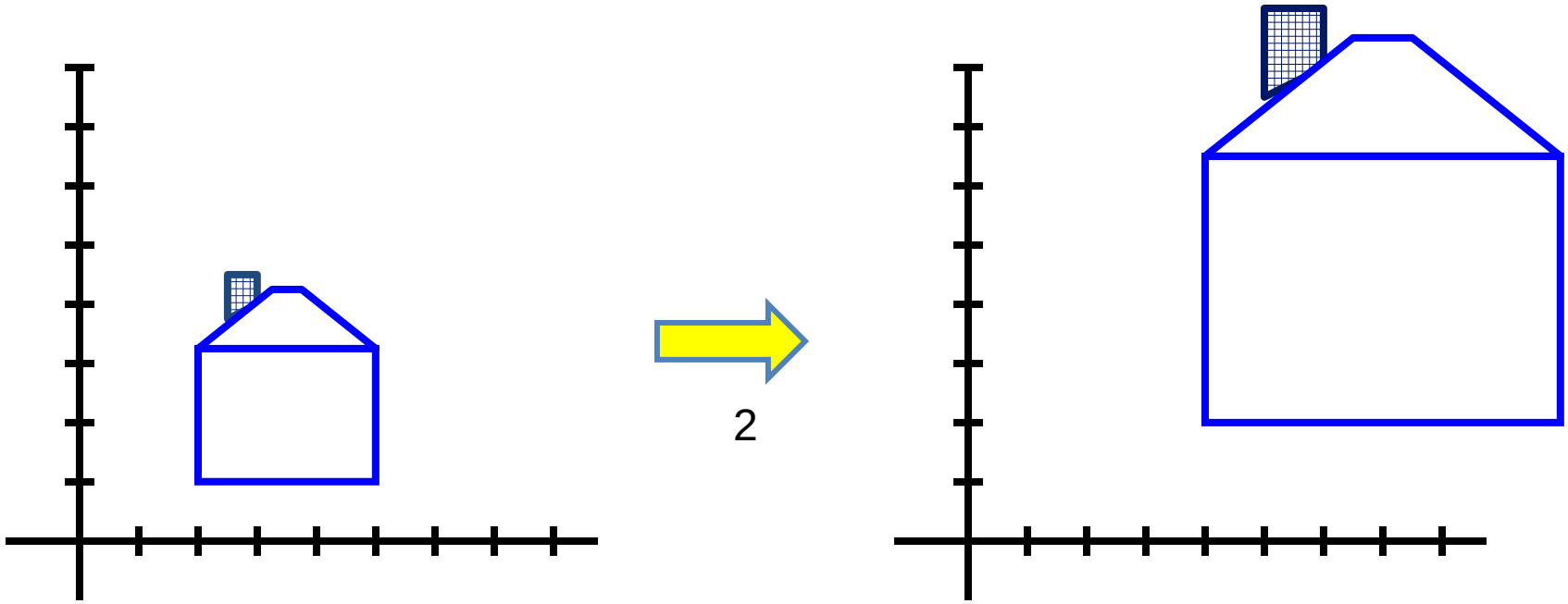
Affine

Perspective

Slide credit (next few slides):
A. Efros and/or S. Seitz

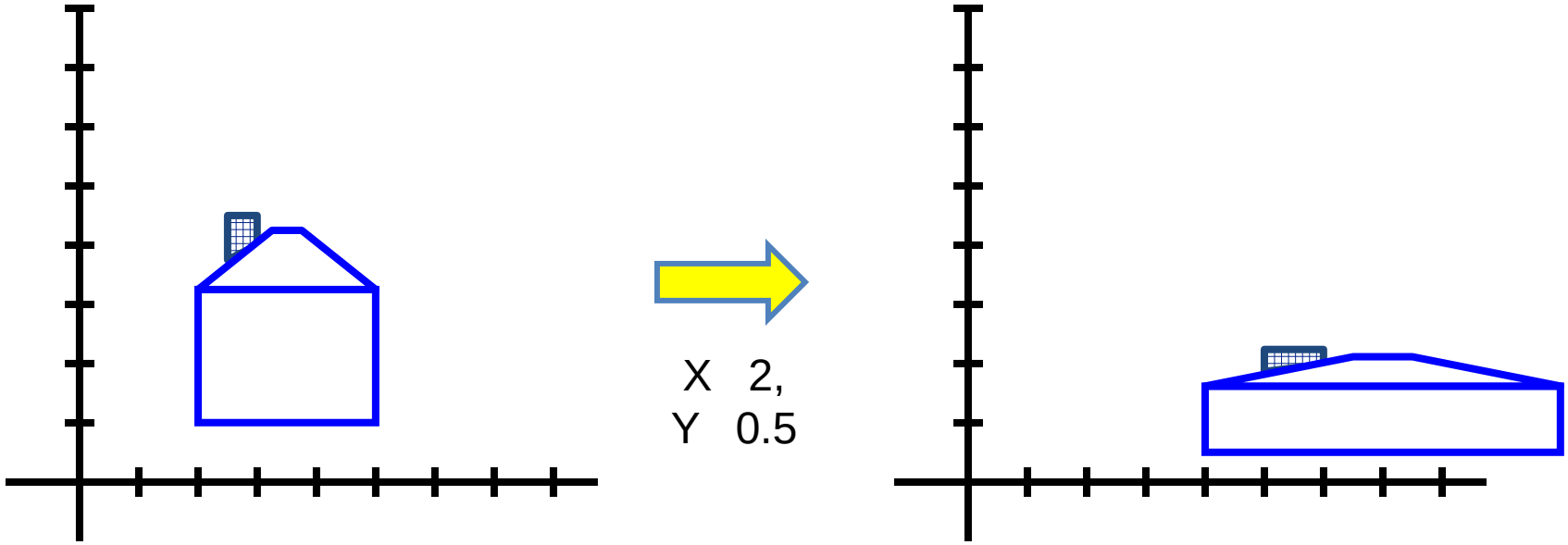
Scaling

- *Scaling* a coordinate means multiplying each of its components by a scalar
- *Uniform scaling* means this scalar is the same for all components:



Scaling

- *Non-uniform scaling*: different scalars per component:



Scaling

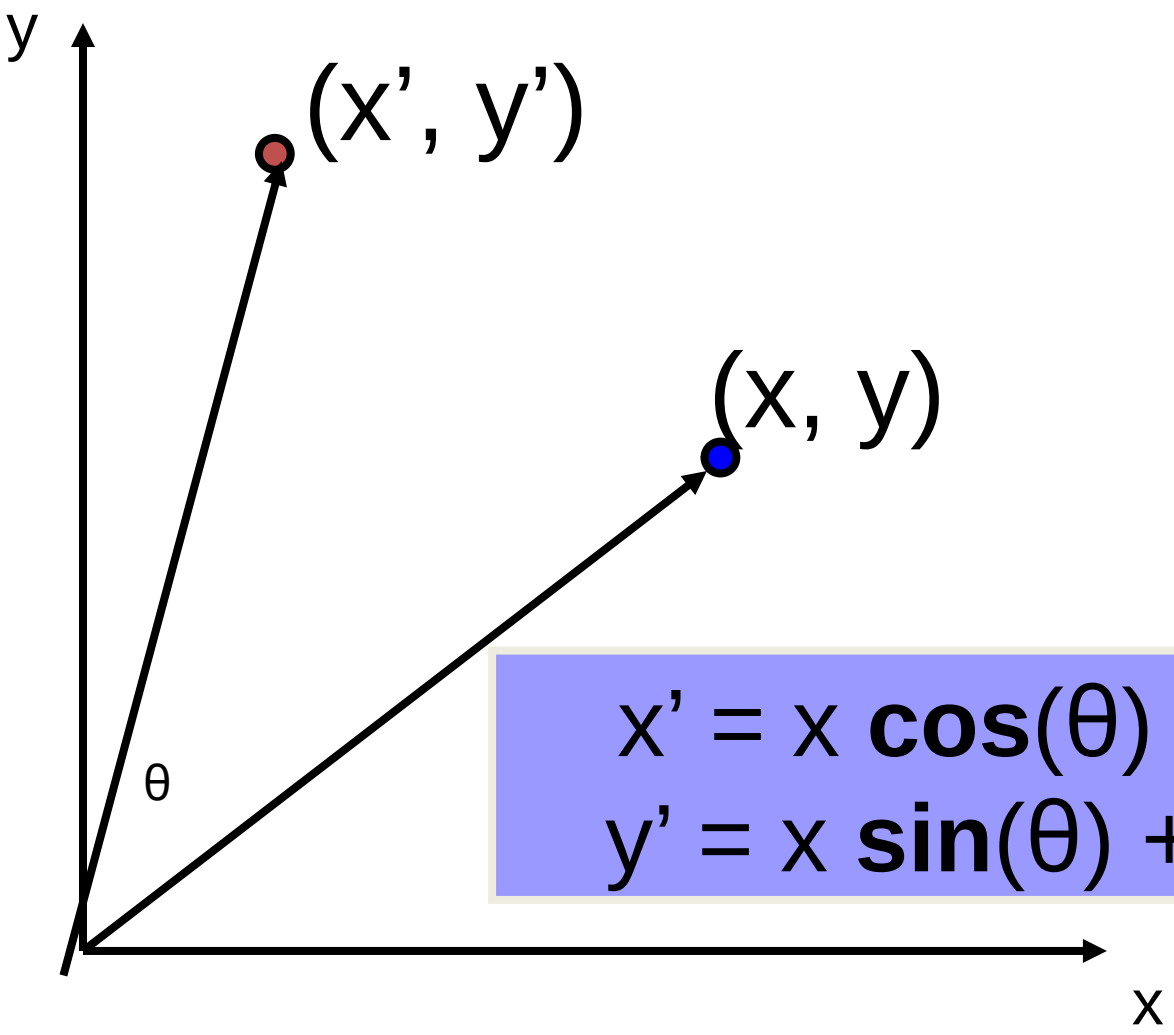
- Scaling operation: $x' = ax$

  $y' = by$

- Or, in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

2-D Rotation



$$x' = x\ \mathbf{cos}(\theta) - y\ \mathbf{sin}(\theta)$$
$$y' = x\ \mathbf{sin}(\theta) + y\ \mathbf{cos}(\theta)$$

2-D Rotation

This is easy to capture in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix}$$

The determinant of this R matrix would be 1 means it didn't scale anything.
But the determinant of the S matrix in page 12 may or may not be 1.

Even though sin(θ) and cos(θ) are nonlinear functions of θ,

- ***x' is a linear combination of x and y***
- ***y' is a linear combination of x and y***

What is the inverse transformation?

- Rotation by –θ
- For rotation matrices $\mathbf{R}^{-1} = \mathbf{R}^{T}$

Basic 2D transformations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & \alpha_x \\ \alpha_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Shear

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

Rotate

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

Translate and Affine are NOT linear transformation. Moreover, every linear transformation is an Affine transformation, but NOT every Affine transformation is a linear transformation.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine

Affine is any combination of translation, scale, rotation, and shear

Affine Transformations

Affine transformations are combinations of

- Linear transformations, and
- Translations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

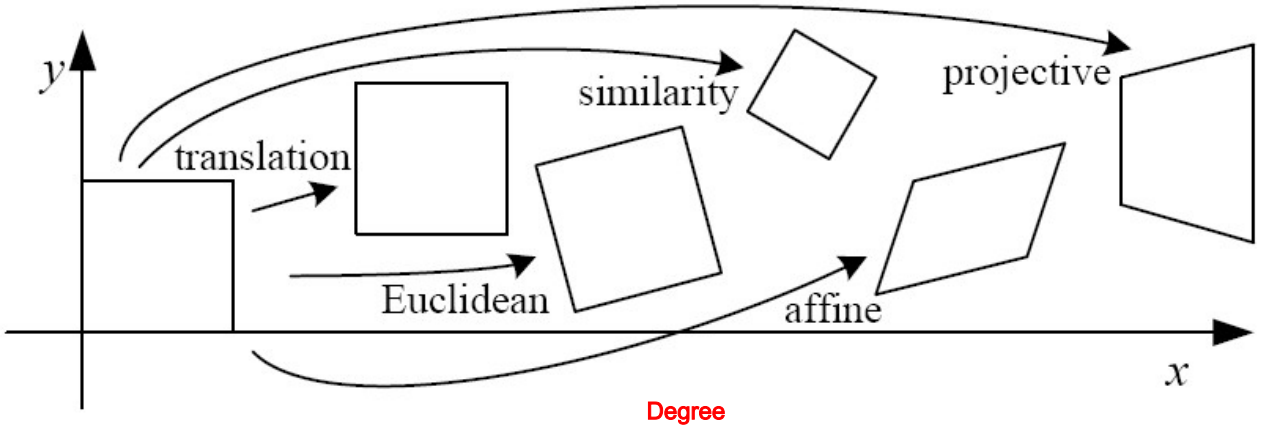
or

Properties of affine transformations:

- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition

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

2D image transformations (reference table)



Degree Of Freedom

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation + ···	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3(2 + 1)	lengths + ···	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4(3 + 1)	angles + ···	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism + ···	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

Projective Transformation is basically the one that takes us from a 3D world to a 2D world.

Szeliski 2.1

Projective Transformations

2D to 2D Projective transformation

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Projective transformations are combos of

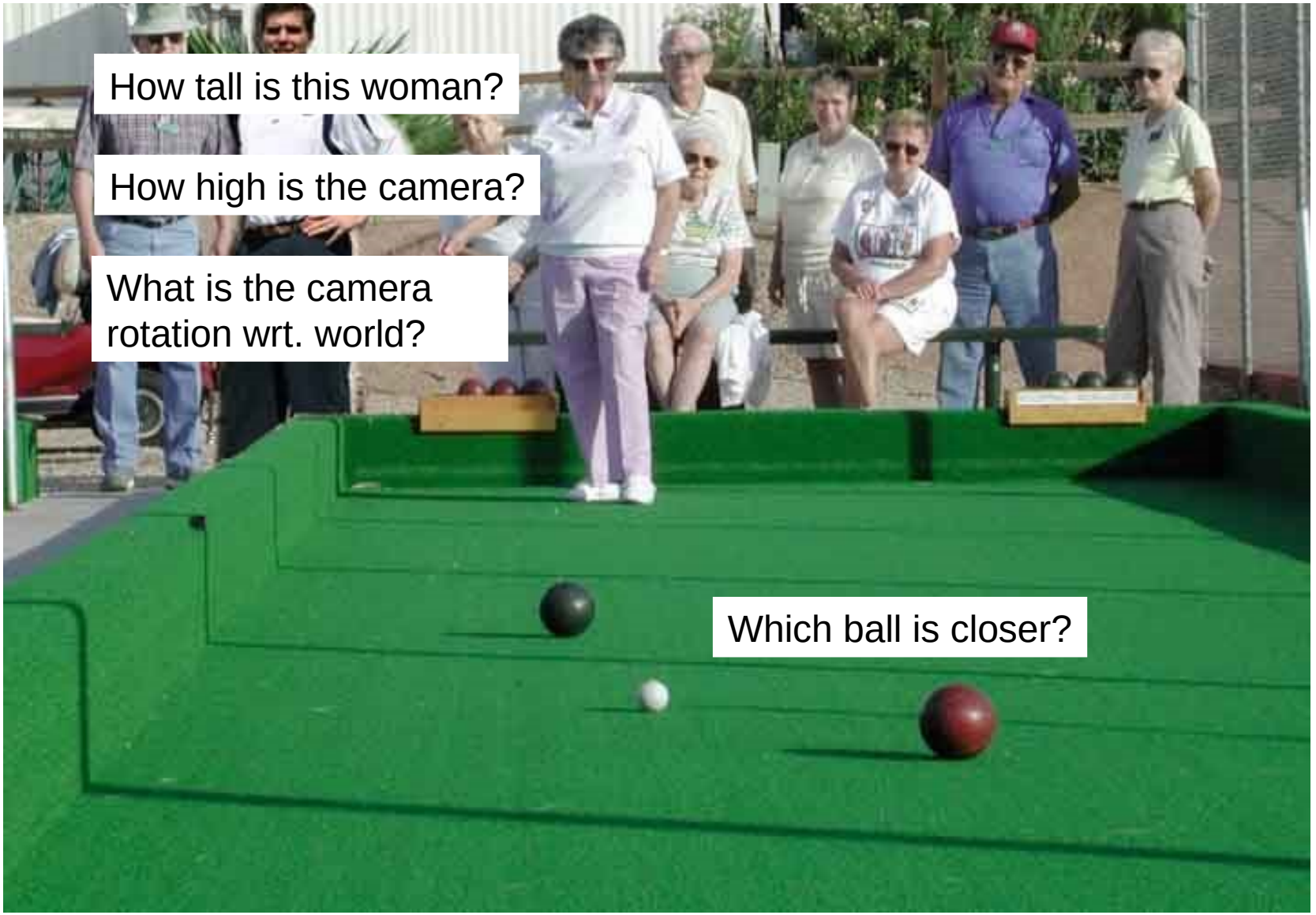
- Affine transformations, and
- Projective warps

Properties of projective transformations:

- Lines map to lines
- Angles are lost
- Parallel lines do not necessarily remain parallel
- length (consequently area) is lost
- Ratios are not preserved
- Closed under composition
- Models change of basis
- Projective matrix is defined up to a scale (8 DOF)

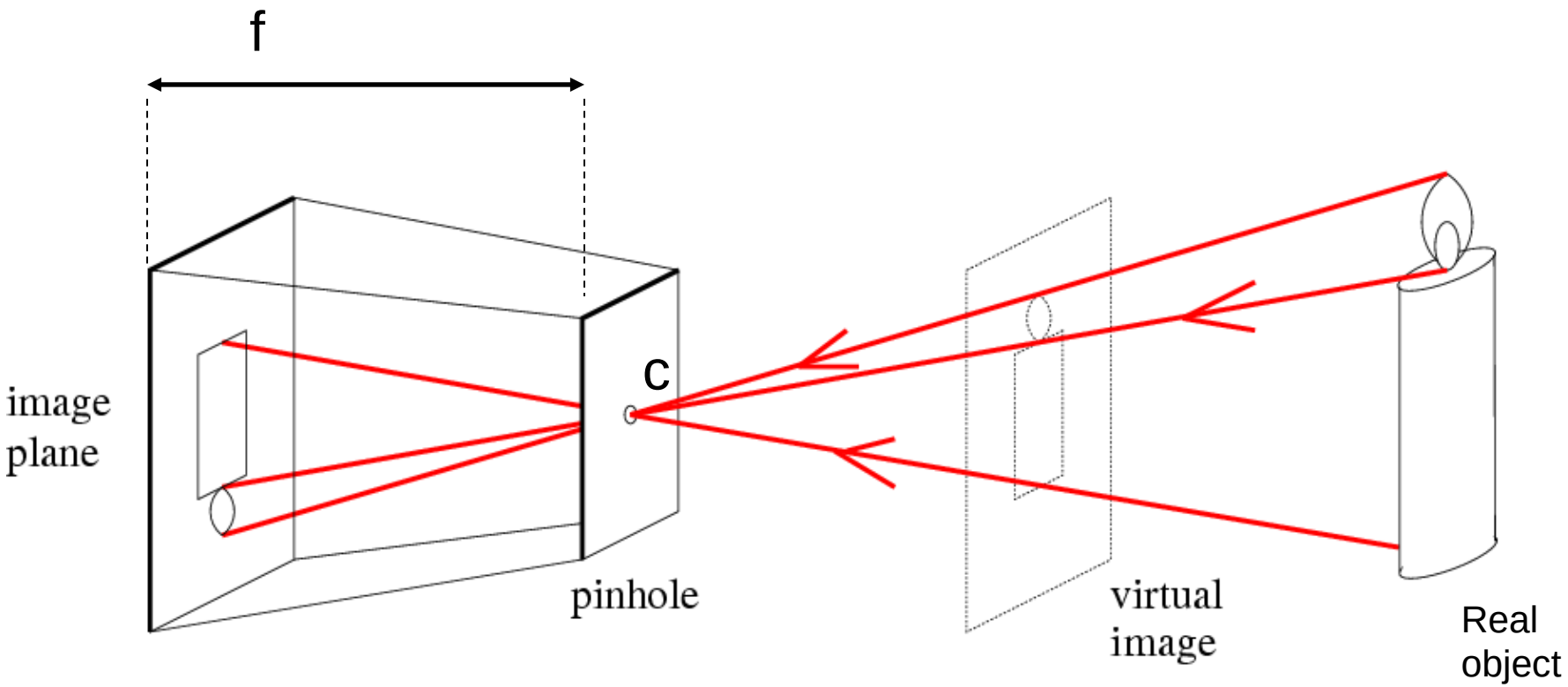
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \frac{k_{p1}}{k_{p2}} \begin{bmatrix} a_{11}/a_{33} & a_{12}/a_{33} & a_{13}/a_{33} \\ a_{21}/a_{33} & a_{22}/a_{33} & a_{23}/a_{33} \\ a_{31}/a_{33} & a_{32}/a_{33} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Cameras and World Geometry



James Hays

Pinhole camera model

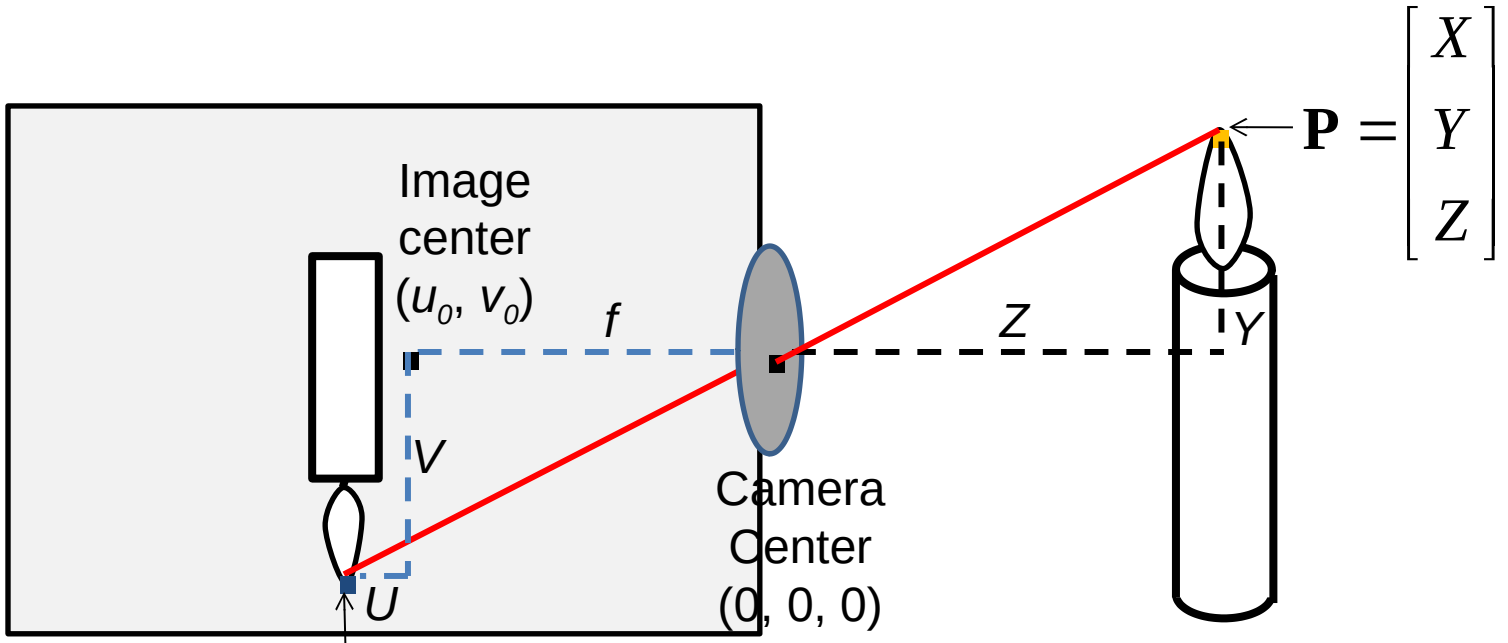


f = Focal length

c = Optical center of the camera

Figure from Forsyth

Projection: world coordinates to image coordinates



$$\mathbf{p} = \begin{bmatrix} U \\ V \end{bmatrix}$$

p = distance from image center

$$U = - X * \frac{f}{Z} \qquad V = - Y * \frac{f}{Z}$$

What is the effect if f and Z are equal?

Projective Geometry

Length (and so area) is lost.

Length and area are not preserved

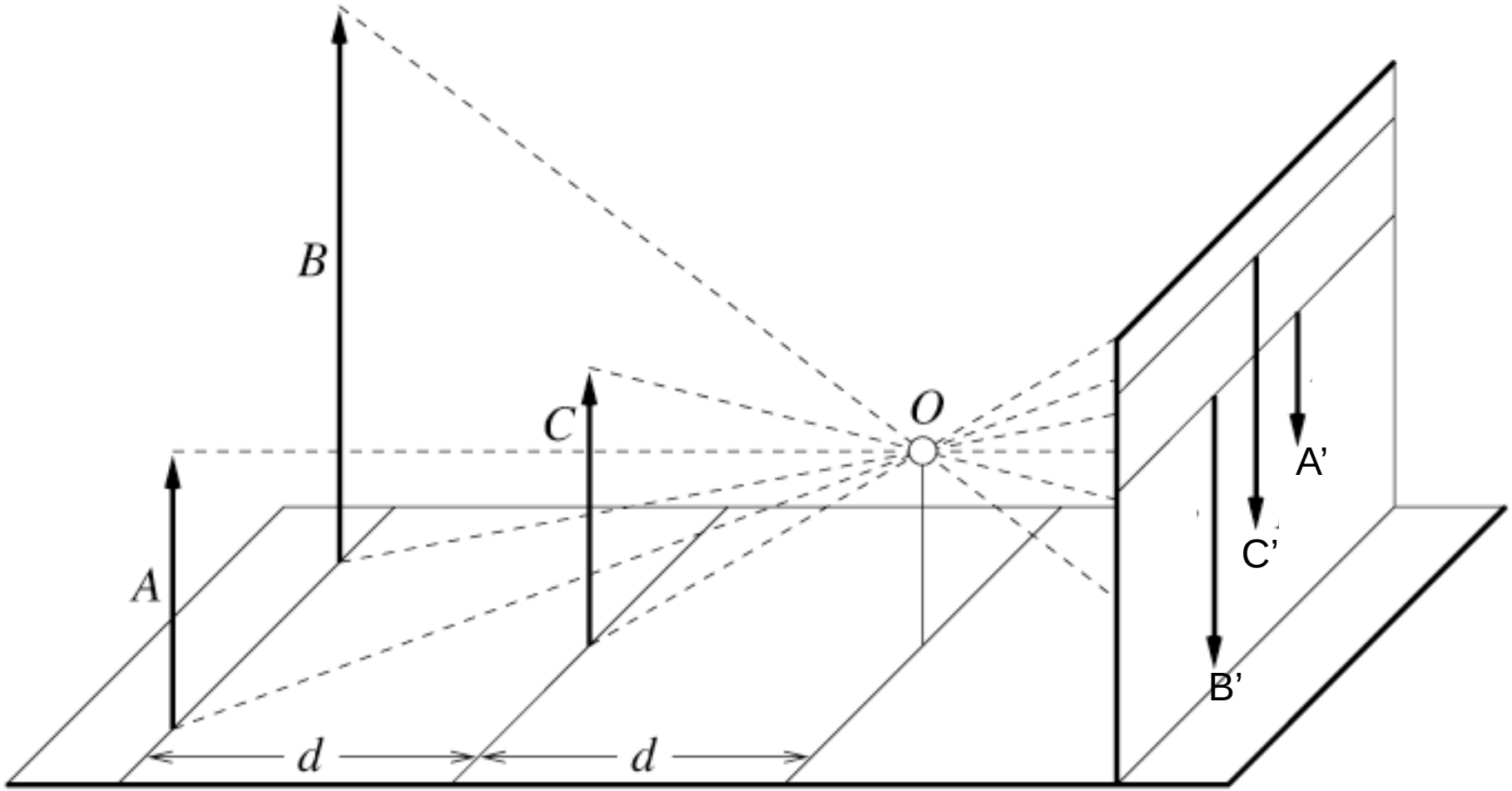
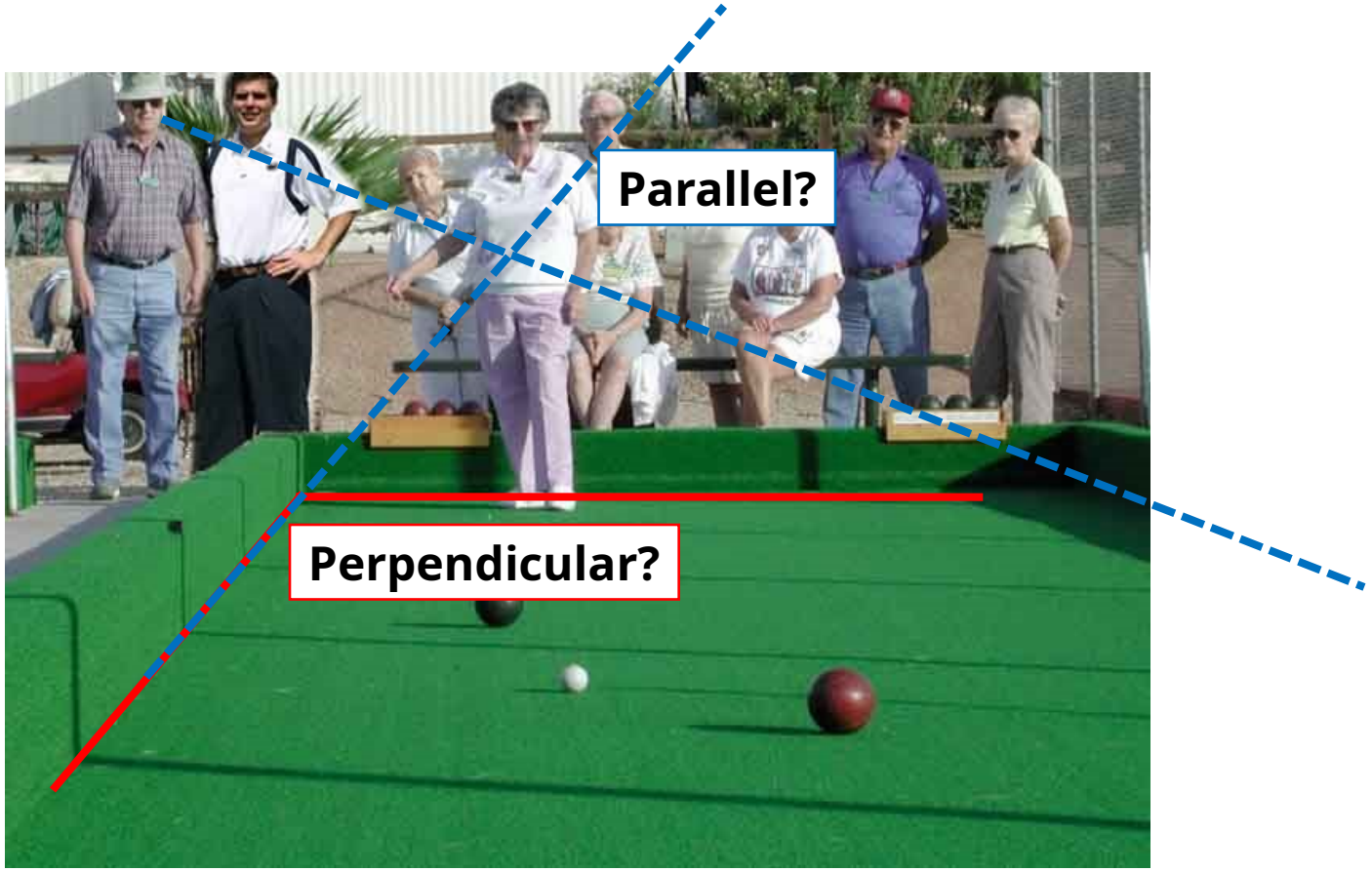


Figure by David Forsyth

Projective Geometry

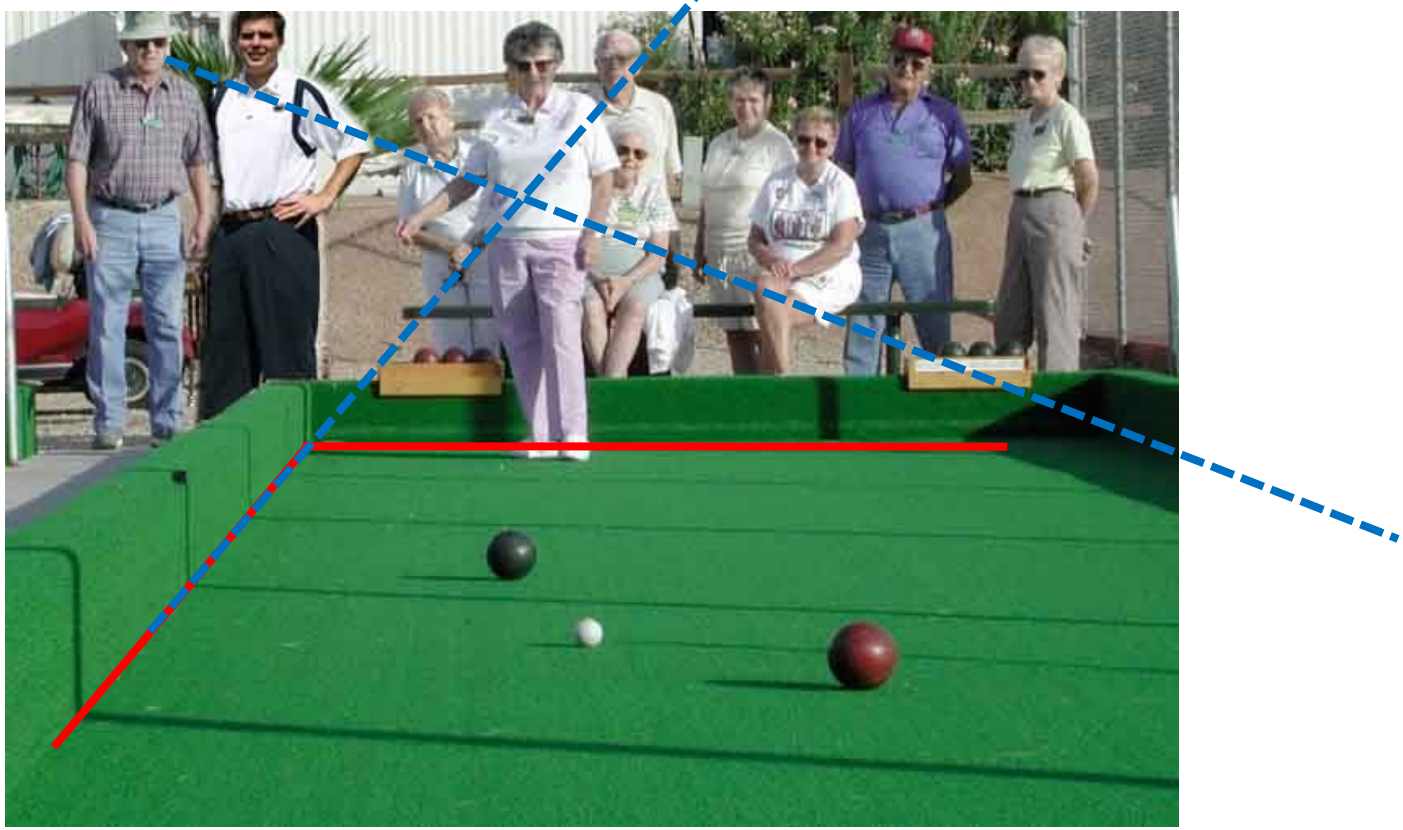
Angles are lost.



Projective Geometry

What is preserved?

- Straight lines are still straight (not really - lens distortion - problem for later).

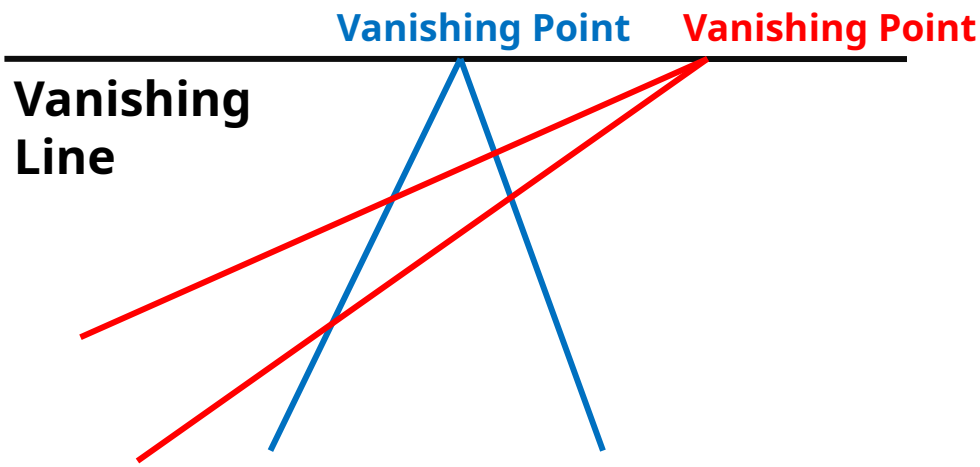


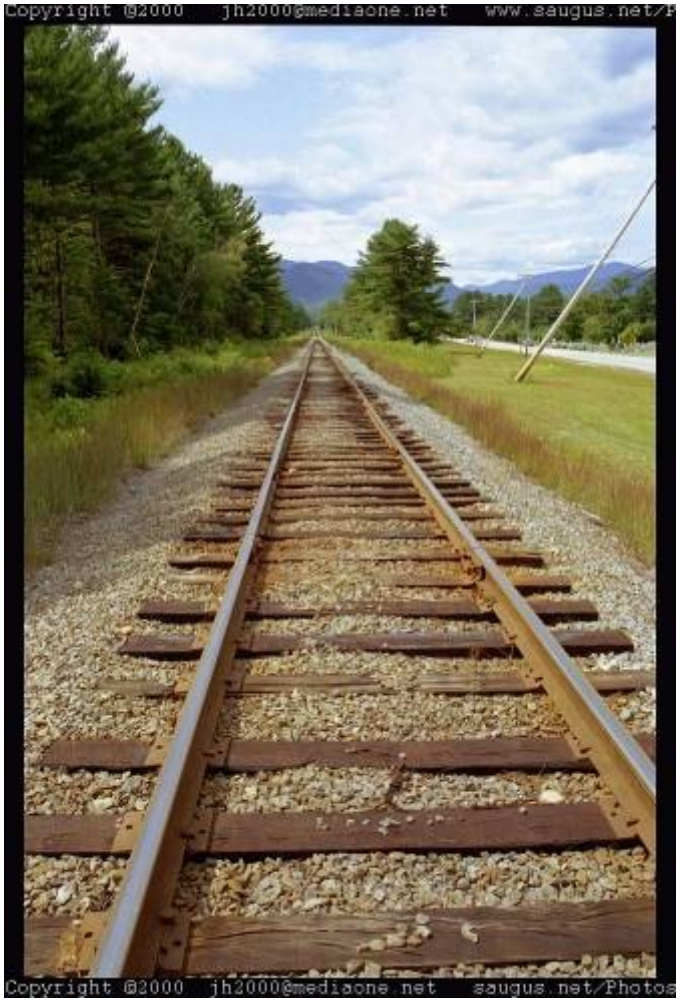
Vanishing points and lines

Parallel lines in the world intersect in the projected image at a “vanishing point”.

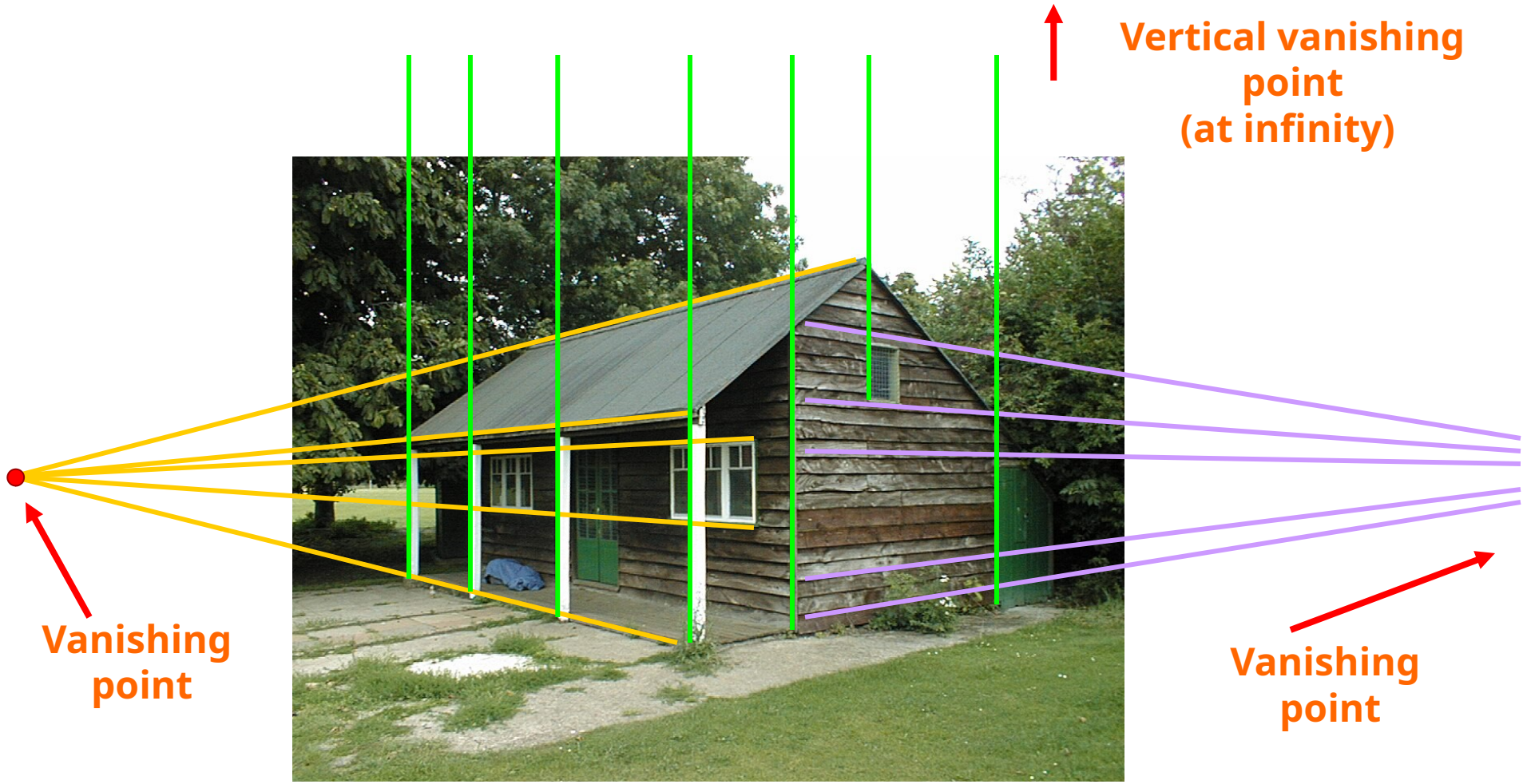
Parallel lines on the same plane in the world converge to vanishing points on a “vanishing line”.

E.G., the horizon.





Vanishing points and lines



Slide from Efros, Photo from Criminisi

Homogeneous coordinates

Converting *to* homogeneous coordinates

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D (image) coordinates

$$(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D (scene) coordinates

Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

2D (image) coordinates

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)$$

3D (scene) coordinates

With the w being the scale factor.

Homogeneous coordinates

Scale invariance in projection space

$$k\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} kx \\ ky \\ kw \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{kx}{kw} \\ \frac{ky}{kw} \end{bmatrix} = \begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \end{bmatrix}$$

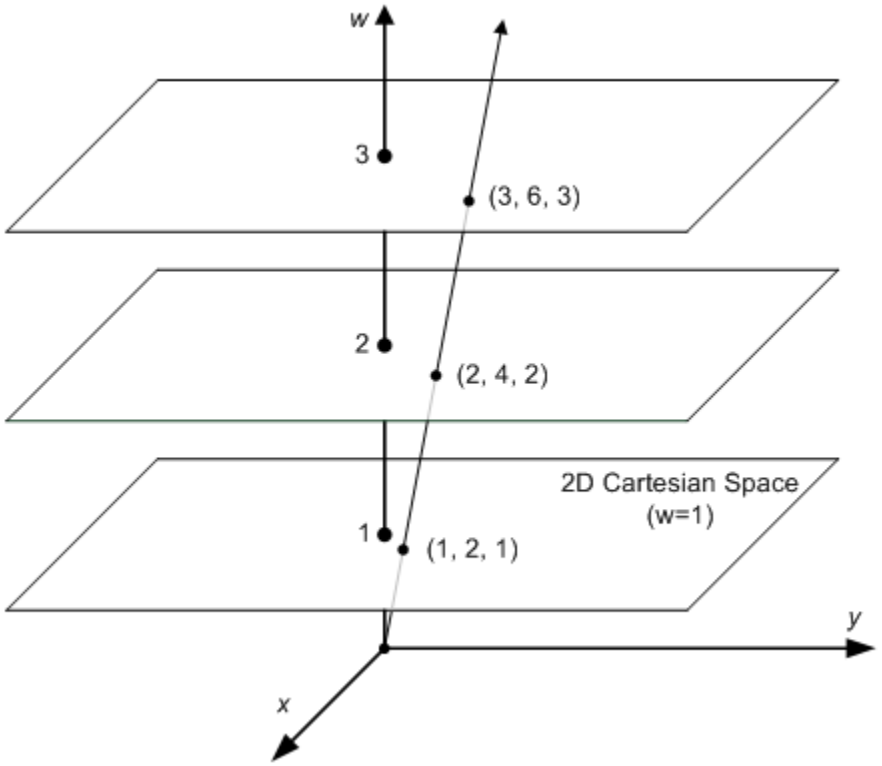
Homogeneous Coordinates

Cartesian Coordinates

E.G., we can uniformly scale the projective space, and it will still produce the same image -> *scale ambiguity*

Homogeneous coordinates

- Projective
- Point becomes a line



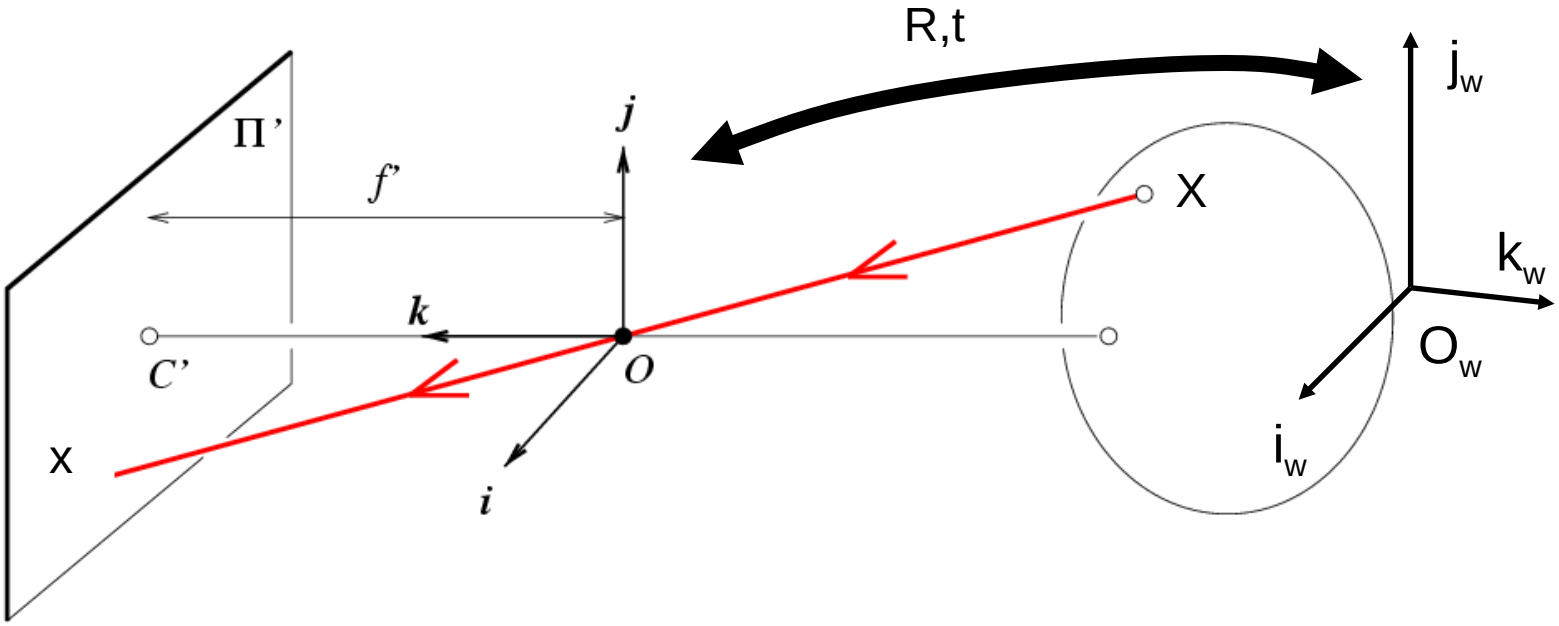
To homogeneous

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

From homogeneous

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

Camera (projection) matrix

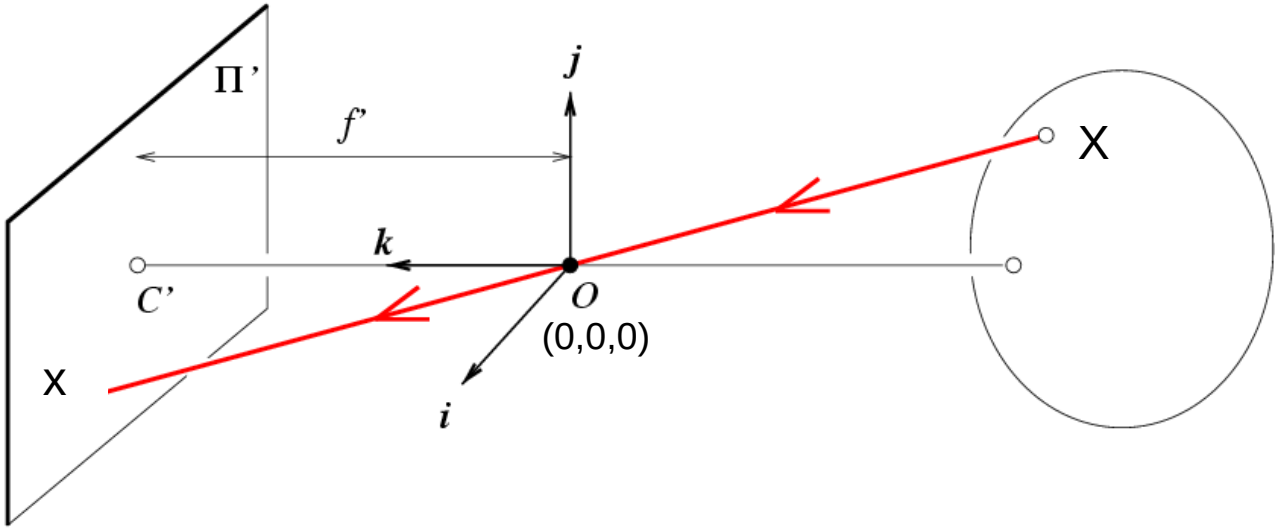


$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

Extrinsic Matrix

x: Image Coordinates: (u,v,1)
K: Intrinsic Matrix (3x3)
R: Rotation (3x3)
t: Translation (3x1)
X: World Coordinates: (X,Y,Z,1)

Projection matrix



Intrinsic Assumptions

- Unit aspect ratio
- Optical center at (0,0)
- No skew

Extrinsic Assumptions

- No rotation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{X} \Rightarrow w\begin{bmatrix}u\\v\\1\end{bmatrix} = \overbrace{\begin{bmatrix}f & 0 & 0\\0 & f & 0\\0 & 0 & 1\end{bmatrix}}^{\mathbf{K}}\begin{bmatrix}0\\0\\0\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

Slide Credit: Savarese

Remove assumption: known optical center

Intrinsic Assumptions
- Unit aspect ratio
- No skew

Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{X} \quad \Rightarrow \quad w\begin{bmatrix}u\\ v\\ 1\end{bmatrix} = \overbrace{\begin{bmatrix}f & 0 & u_0\\ 0 & f & v_0\\ 0 & 0 & 1\end{bmatrix}}^{\mathbf{K}}\begin{bmatrix}0\\ 0\\ 0\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

Remove assumption: equal aspect ratio

Intrinsic Assumptions
- No skew

Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{X} \Rightarrow w\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & 0 & u_0 & 0\\0 & f_y & v_0 & 0\\0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

Remove assumption: non-skewed pixels

Intrinsic Assumptions

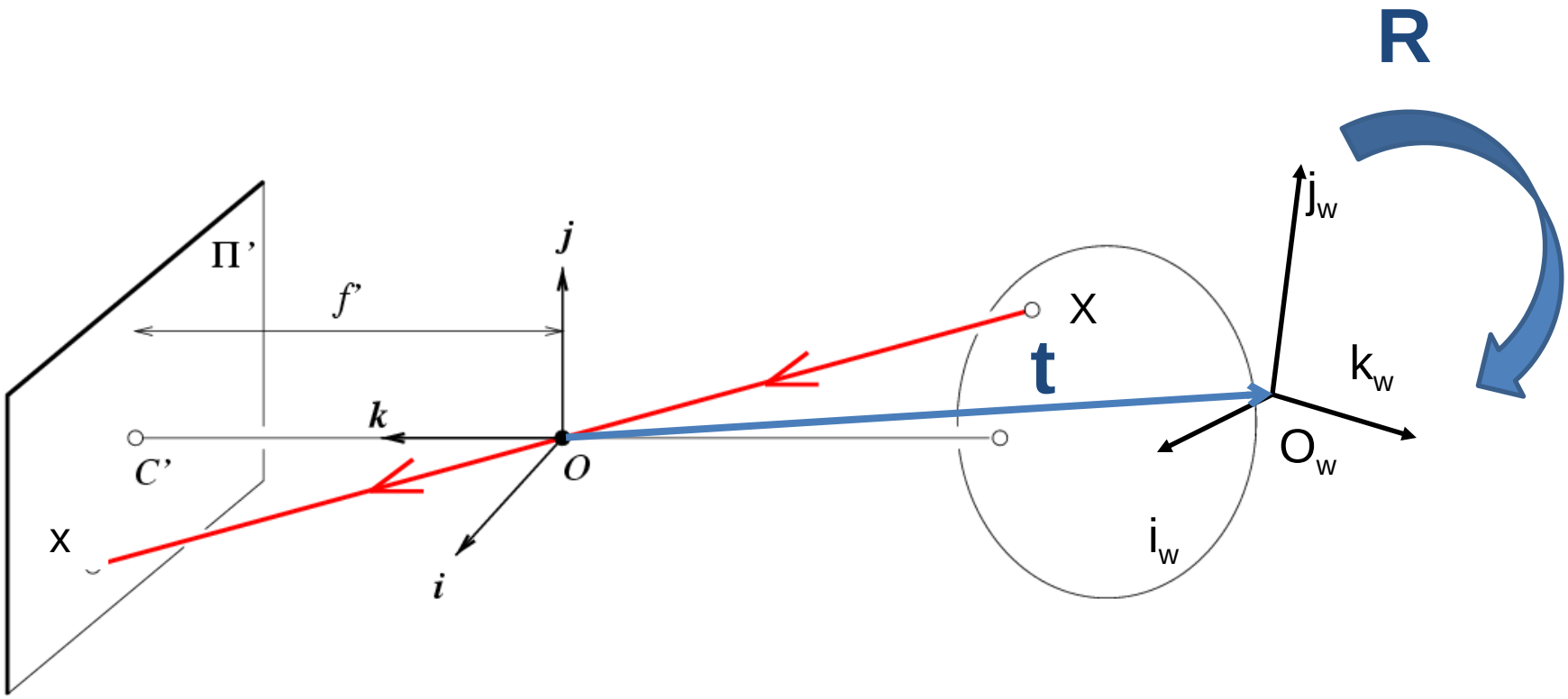
Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{X} \Rightarrow w\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & s & u_0 & 0\\0 & f_y & v_0 & 0\\0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

Remember that fx being the focal length of x, fy being the focal length of y.
s being the skew parameter,
u0, v0 being the optical centers

Note: different books use different notation for parameters

Oriented and Translated Camera



Allow camera translation

Intrinsic Assumptions

Extrinsic Assumptions

- No rotation

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{t}\end{bmatrix}\mathbf{X} \Rightarrow w\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & s & u_0\\0 & f_y & v_0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix} & \begin{bmatrix}t_x\\t_y\\t_z\end{bmatrix}\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

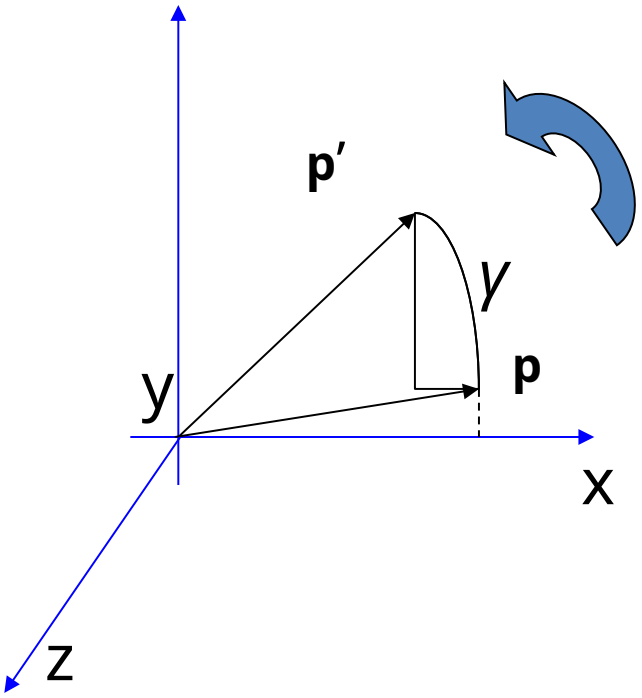
I t

Here I is the identity matrix as there's no ratation. And t is the translation.

3D Rotation of Points

There are many rotation representations (x,y,z) or (euler), etc. that are beyond the scope of this course.

Rotation around the coordinate axes, counter-clockwise:



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

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

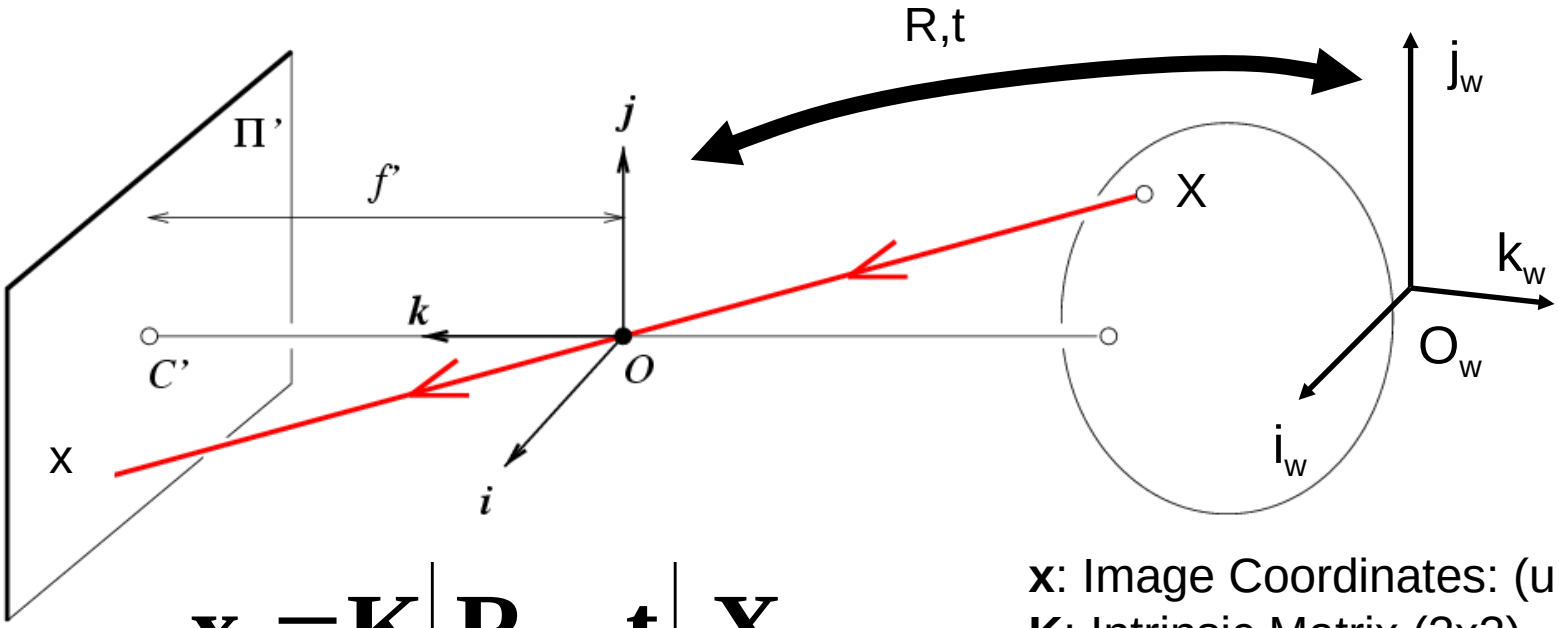
$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Allow camera rotation

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Camera (projection) matrix



$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

Extrinsic Matrix

x: Image Coordinates: (u,v,1)
K: Intrinsic Matrix (3x3)
R: Rotation (3x3)
t: Translation (3x1)
X: World Coordinates: (X,Y,Z,1)

$$w\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & s & u_0\\0 & f_y & v_0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_x\\r_{21} & r_{22} & r_{23} & t_y\\r_{31} & r_{32} & r_{33} & t_z\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

Demo – Kyle Simek

“Dissecting the Camera Matrix”

Three-part blog series

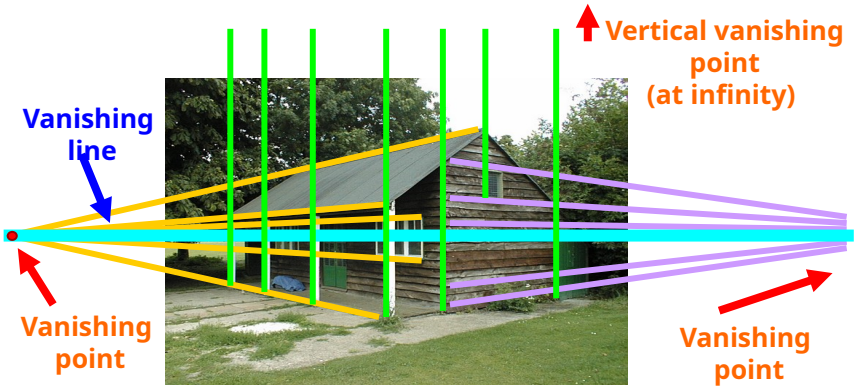
- http://ksimek.github.io/2012/08/14/decompose/
- http://ksimek.github.io/2012/08/22/extrinsic/
- http://ksimek.github.io/2013/08/13/intrinsic/

“Perspective toy”

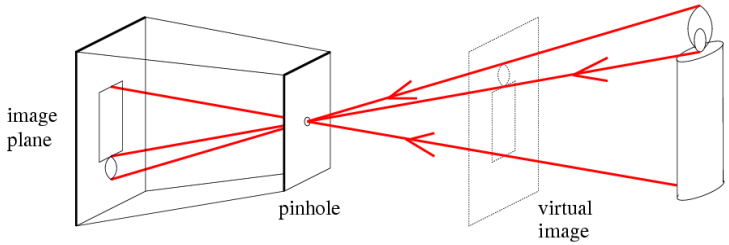
- http://ksimek.github.io/perspective_camera_toy.html

Things to remember

Vanishing points and vanishing lines



Pinhole camera model and camera projection matrix



$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

Homogeneous coordinates

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Camera Calibration

How to calibrate the camera? (also called “camera resectioning”)

Intrinsic calibration

extrinsic calibration

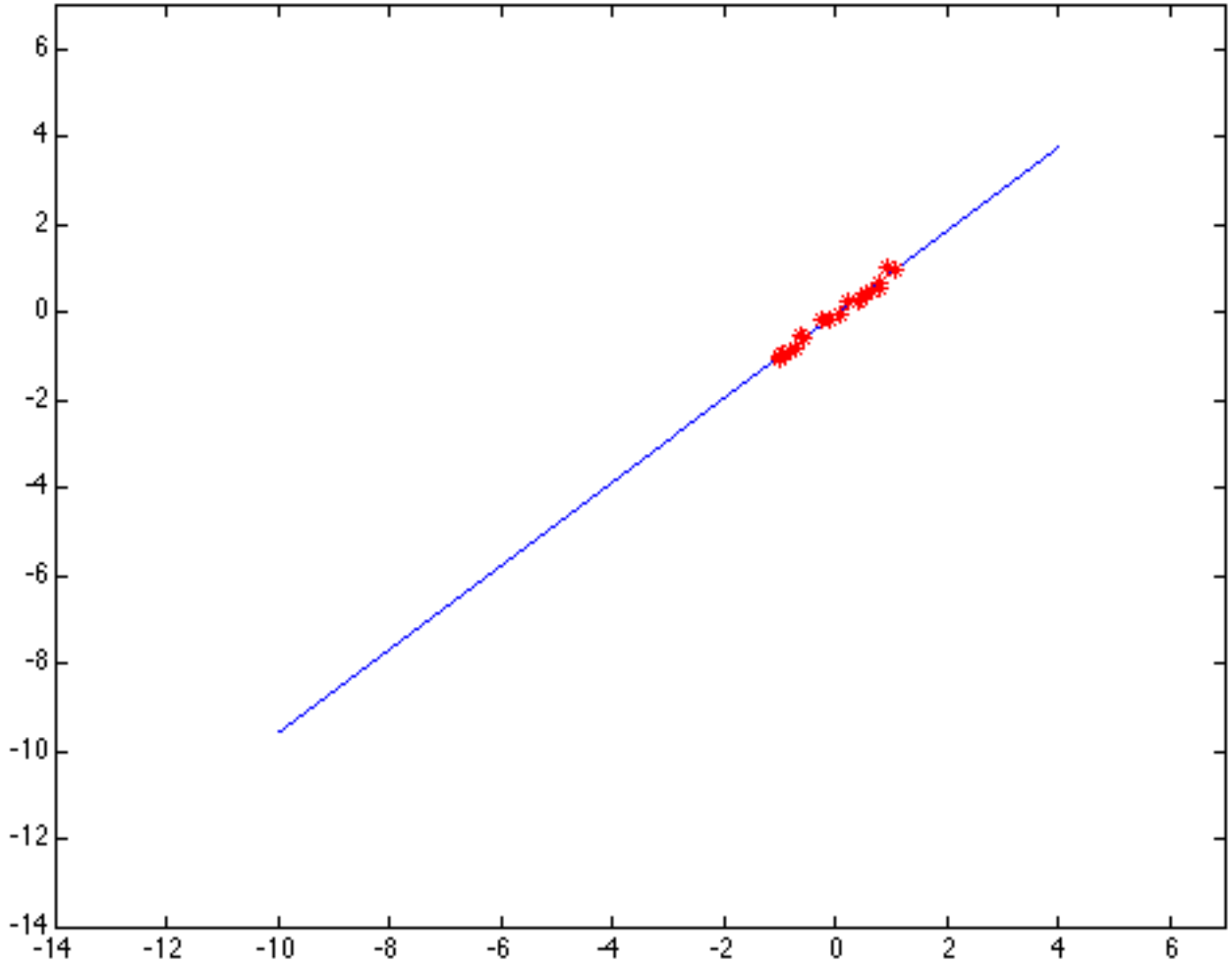
$$\mathbf{x} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \mathbf{X}$$

$$\mathbf{x} = \mathbf{M}\mathbf{X}$$

$$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

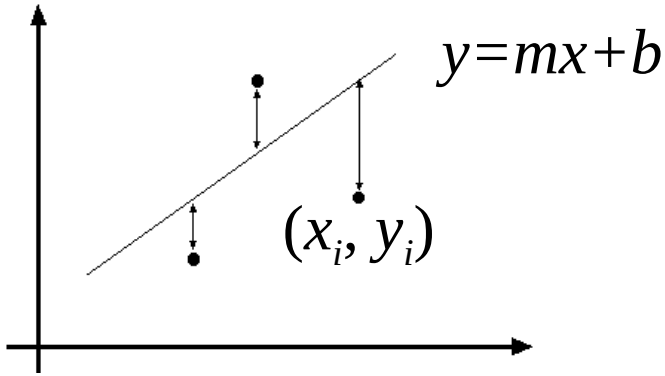
Linear least-squares regression!

Simple example of LLS: Fitting a line



Least squares line fitting

- Data: $(x_1, y_1), \ldots, (x_n, y_n)$
- Line equation: $y_i = m x_i + b$
- Find (m, b) to minimize



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

$$E = \sum_{i=1}^{n} \left(\begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - y_i \right)^2 = \left\| \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 = \|\mathbf{Ap} - \mathbf{y}\|^2$$

$$= \mathbf{y}^T\mathbf{y} - 2(\mathbf{Ap})^T\mathbf{y} + (\mathbf{Ap})^T(\mathbf{Ap})$$

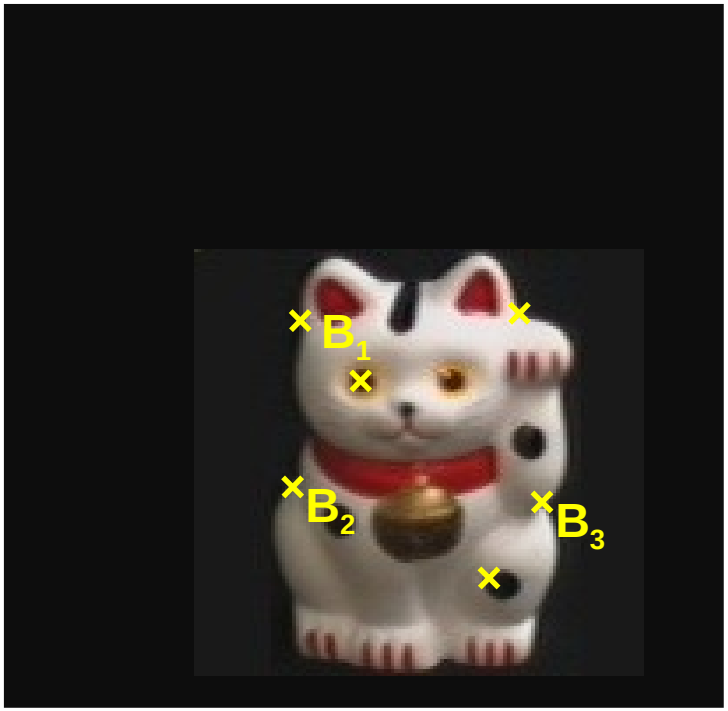
$$\frac{dE}{dp} = 2\mathbf{A}^T\mathbf{Ap} - 2\mathbf{A}^T\mathbf{y} = 0$$

$$\mathbf{A}^T\mathbf{Ap} = \mathbf{A}^T\mathbf{y} \Rightarrow \mathbf{p} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{y}$$

(Closed form solution)

```
Matlab: p = A \ y;
Python:
p = np.linalg.lstsq(A,y)
[0]
```

Example: solving for translation

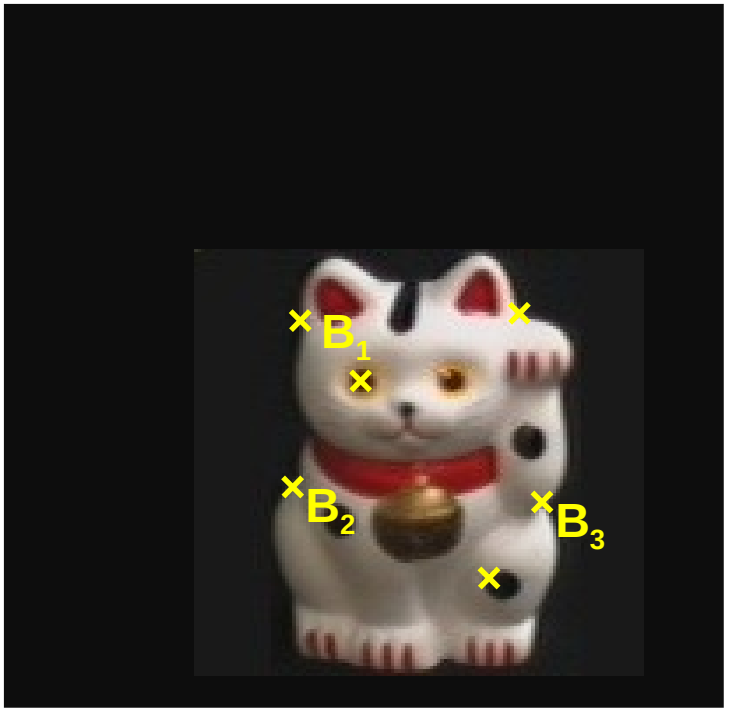


Given matched points in {A} and {B}, estimate the translation of the object

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Example: solving for translation

(t_x, t_y)



$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Least squares setup

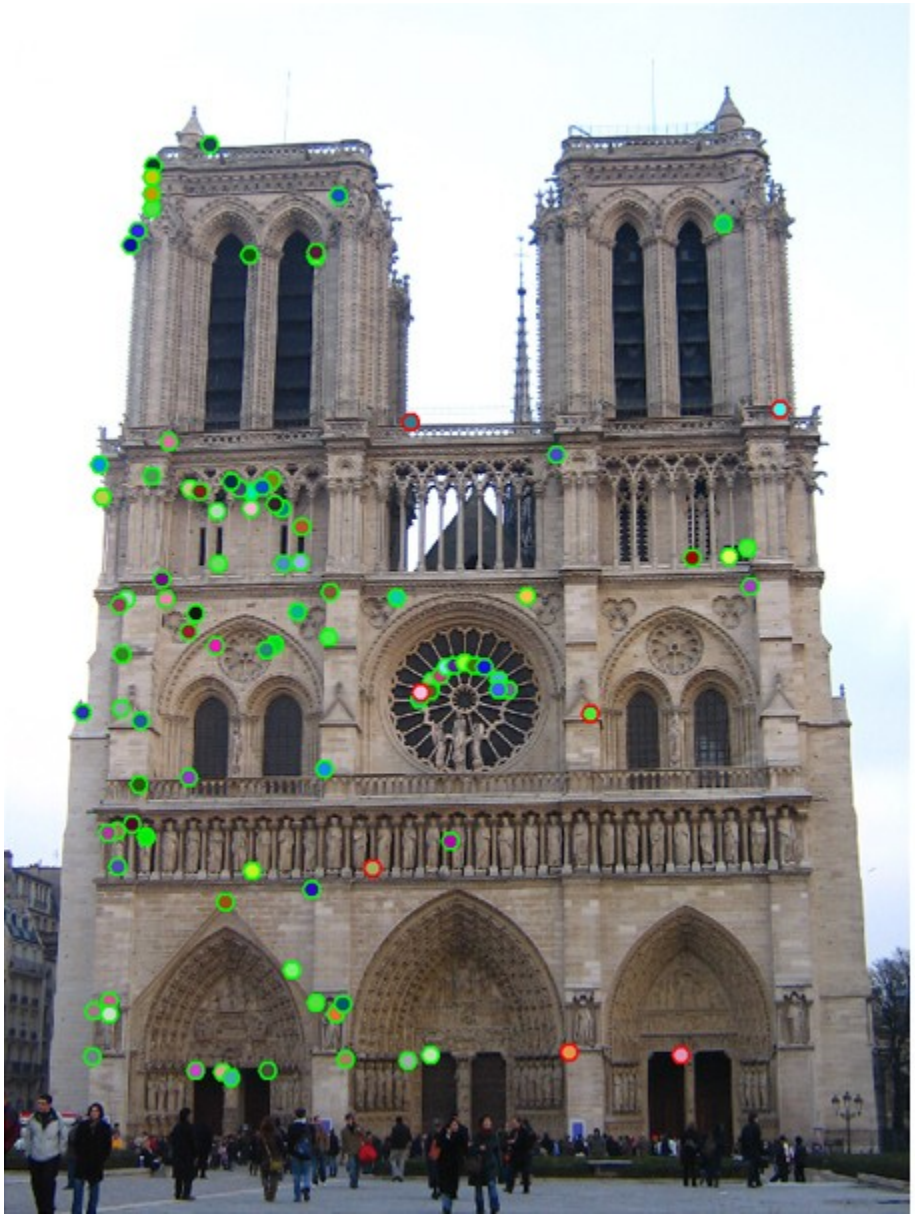
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x_1^B - x_1^A \\ y_1^B - y_1^A \\ \vdots \\ x_n^B - x_n^A \\ y_n^B - y_n^A \end{bmatrix}$$

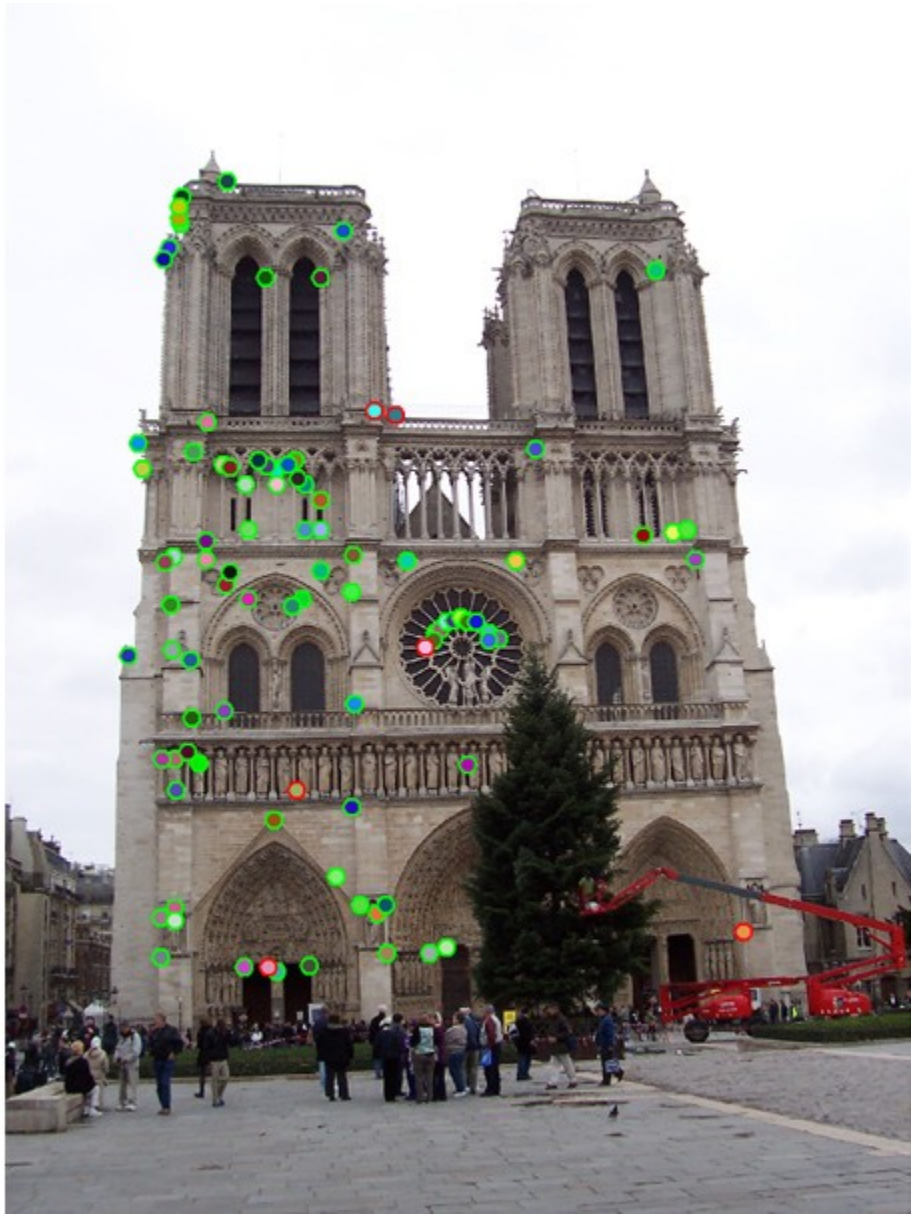
Example: discovering rot/trans/scale

Given matched points in {A} and {B}, estimate the transformation matrix

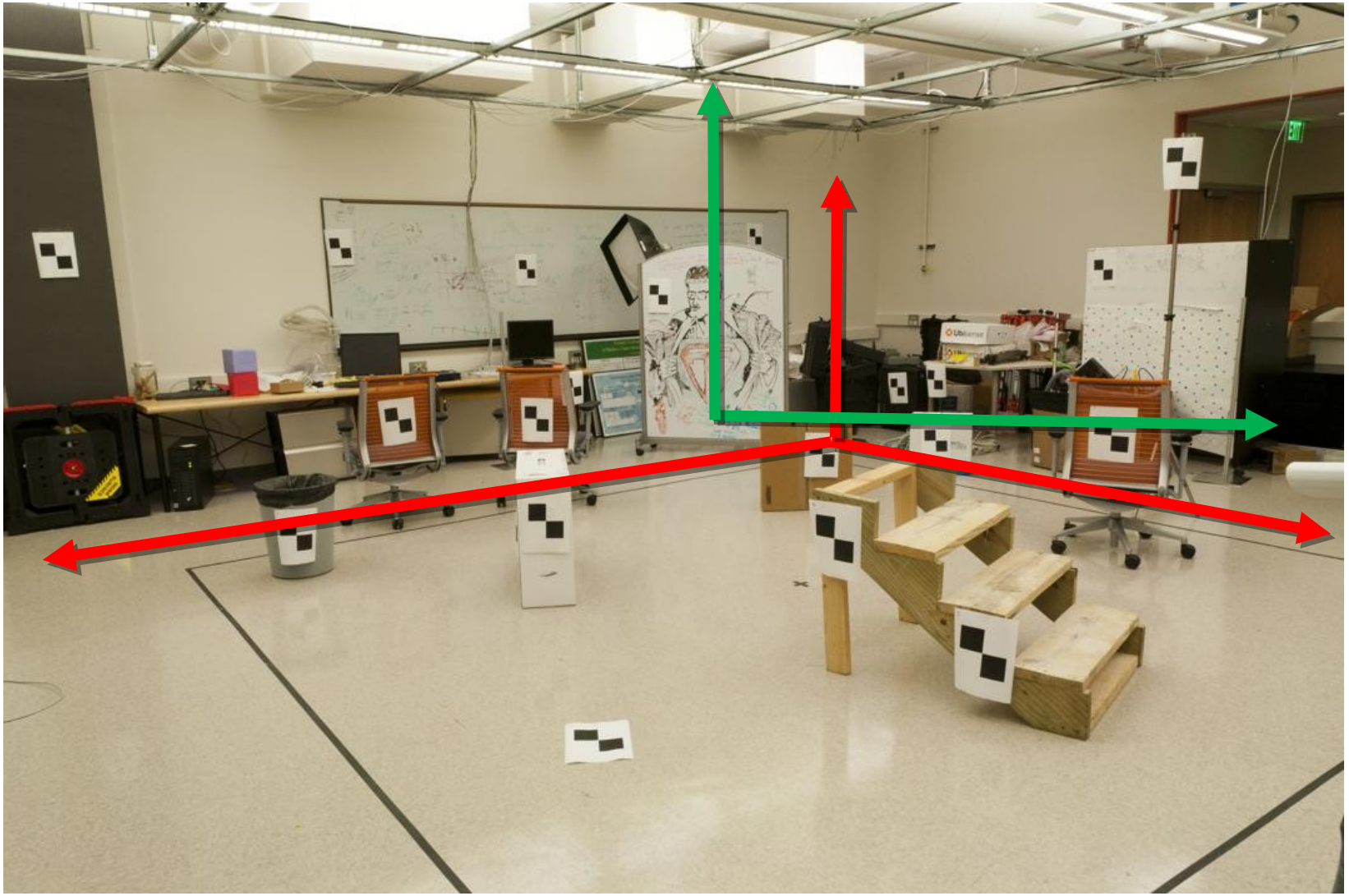
$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \mathrm{T} \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \qquad \mathrm{T} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Are these transformations enough?





World vs Camera coordinates

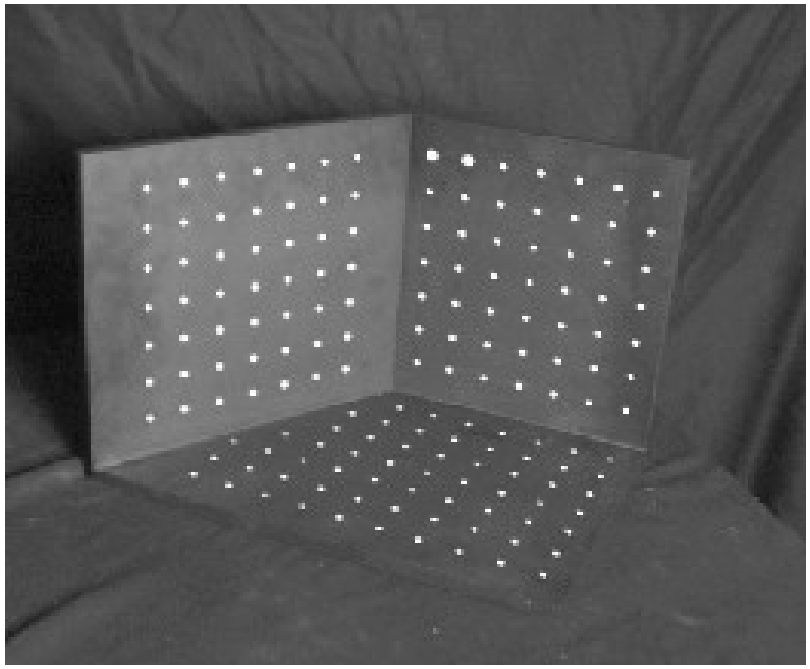


Calibrating the Camera

There are many ways to calibrate a camera.

Easiest way is to use a scene with **known** geometry (DLT - Direct Linear Transform)

- Correspond image points to 3d points
- Get least squares solution (or non-linear solution)



Known 2d image coords

Known 3d world locations

M

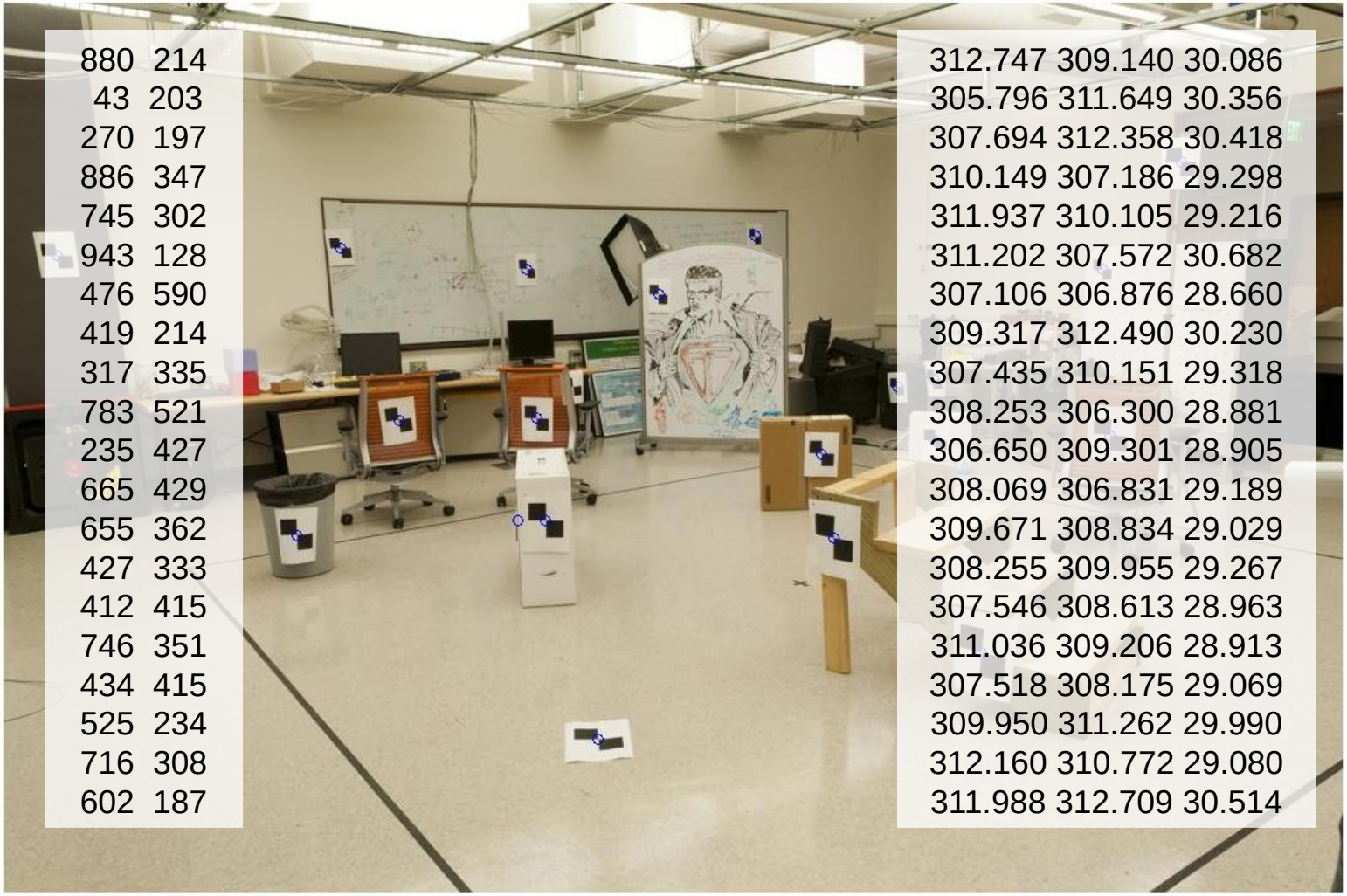
$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Unknown Camera Parameters

How do we calibrate a camera?

Known 2d image coords

Known 3d world locations

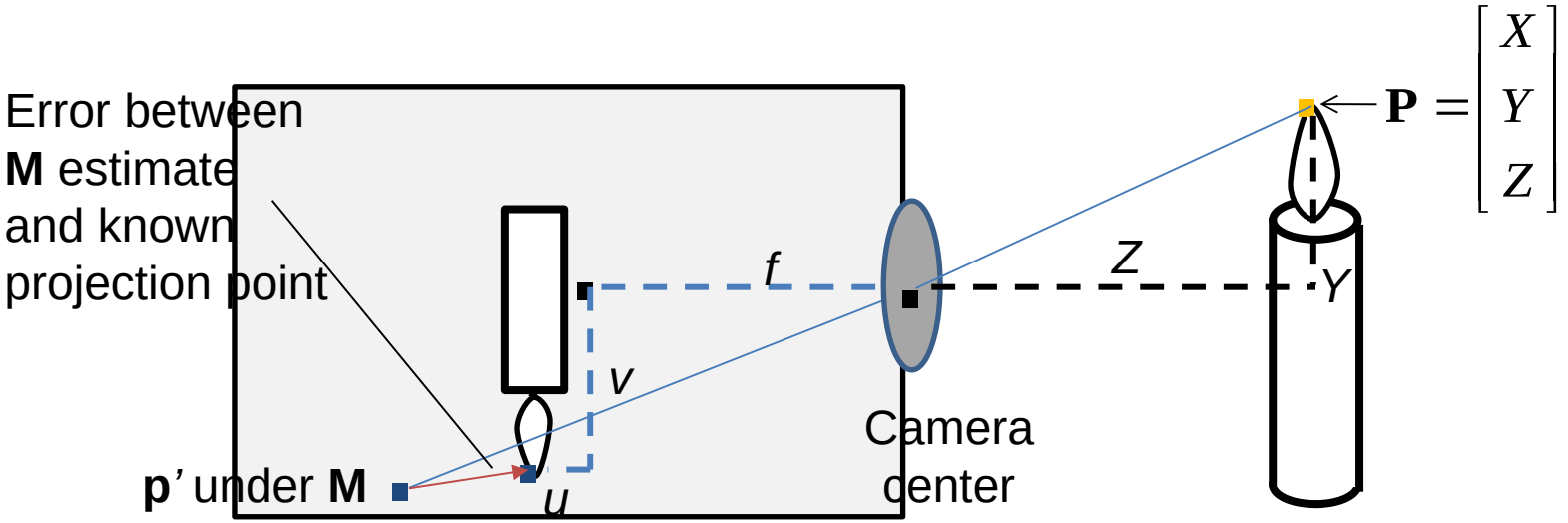


880	214
43	203
270	197
886	347
745	302
943	128
476	590
419	214
317	335
783	521
235	427
665	429
655	362
427	333
412	415
746	351
434	415
525	234
716	308
602	187

312.747	309.140	30.086
305.796	311.649	30.356
307.694	312.358	30.418
310.149	307.186	29.298
311.937	310.105	29.216
311.202	307.572	30.682
307.106	306.876	28.660
309.317	312.490	30.230
307.435	310.151	29.318
308.253	306.300	28.881
306.650	309.301	28.905
308.069	306.831	29.189
309.671	308.834	29.029
308.255	309.955	29.267
307.546	308.613	28.963
311.036	309.206	28.913
307.518	308.175	29.069
309.950	311.262	29.990
312.160	310.772	29.080
311.988	312.709	30.514

What is least squares doing?

- Given 3D point evidence, find best **M** which minimizes error between estimate (**p'**) and known corresponding 2D points (**p**).

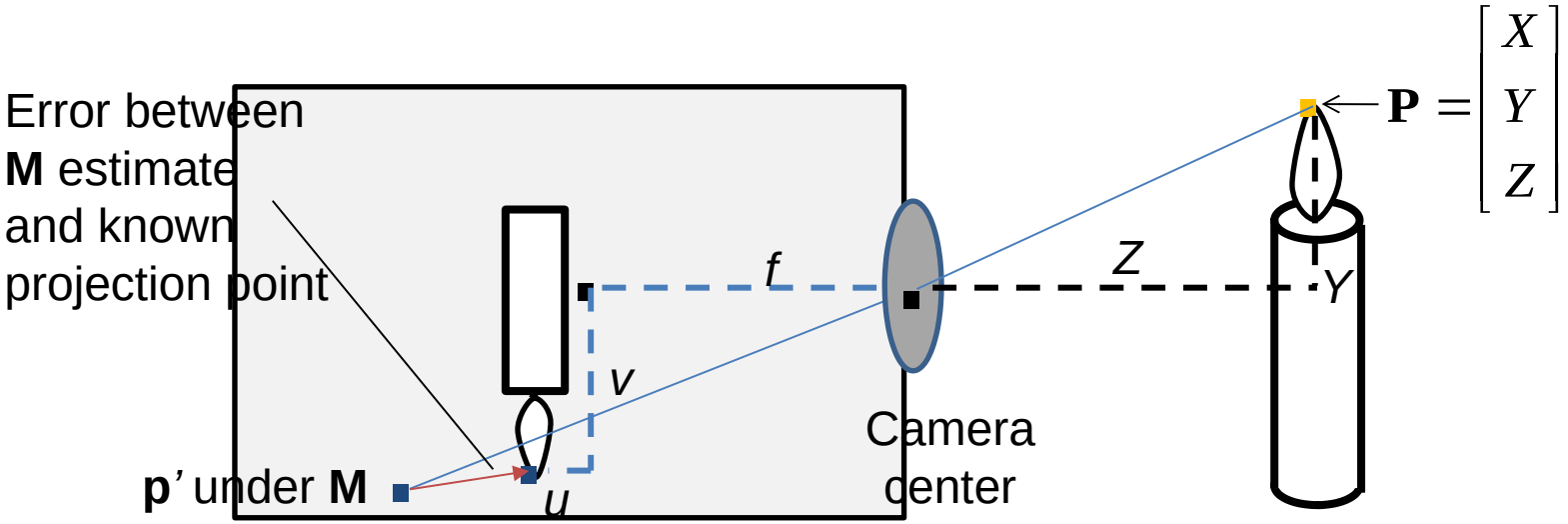


$$\mathbf{p} = \begin{bmatrix} u \\ v \end{bmatrix}$$

p = distance from image center

What is least squares doing?

- Best **M** occurs when **p’** = **p**, or when **p’** – **p** = 0
- Form these equations from all point evidence
- Solve for model via closed-form regression



$$\mathbf{p} = \begin{bmatrix} u \\ v \end{bmatrix}$$

p = distance from image center

Unknown Camera Parameters

Known 2d image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations

First, work out where X,Y,Z projects to under candidate **M.**

$$su = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$sv = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

$$s = m_{31}X + m_{32}Y + m_{33}Z + m_{34}$$

Two equations per 3D point correspondence

$$u = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$v = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

Unknown Camera Parameters

Known 2d image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations

Next, rearrange into form where all **M** coefficients are individually stated in terms of X,Y,Z,u,v.
-> Allows us to form lsq matrix.

$$u = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$v = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$(m_{31}X + m_{32}Y + m_{33}Z + m_{34})u = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$(m_{31}X + m_{32}Y + m_{33}Z + m_{34})v = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

$$m_{31}uX + m_{32}uY + m_{33}uZ + m_{34}u = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$m_{31}vX + m_{32}vY + m_{33}vZ + m_{34}v = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

Unknown Camera Parameters

Known 2d image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations

Next, rearrange into form where all **M** coefficients are individually stated in terms of X,Y,Z,u,v.
-> Allows us to form lsq matrix.

$$m_{31}uX + m_{32}uY + m_{33}uZ + m_{34}u = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$
$$m_{31}vX + m_{32}vY + m_{33}vZ + m_{34}v = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

$$0 = m_{11}X + m_{12}Y + m_{13}Z + m_{14} - m_{31}uX - m_{32}uY - m_{33}uZ - m_{34}u$$
$$0 = m_{21}X + m_{22}Y + m_{23}Z + m_{24} - m_{31}vX - m_{32}vY - m_{33}vZ - m_{34}v$$

Unknown Camera Parameters

Known 2d image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations

$$0 = m_{11}X + m_{12}Y + m_{13}Z + m_{14} - m_{31}uX - m_{32}uY - m_{33}uZ - m_{34}u$$
$$0 = m_{21}X + m_{22}Y + m_{23}Z + m_{24} - m_{31}vX - m_{32}vY - m_{33}vZ - m_{34}v$$

- Finally, solve for m's entries using linear least squares
- Method 1 : **Ax**=**b** form

$$\underbrace{\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 \\ & & & & & & \vdots & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n \end{bmatrix}}_{\text{A}} \underbrace{\begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix}}_{\text{x}} = \underbrace{\begin{bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{bmatrix}}_{\text{b}}$$

```
MATLAB:
M = A\b;
M = [M;1];
M = reshape(M,[],3)';

Python Numpy:
M = np.linalg.lstsq(A,b)
[0];
M = np.append(M,1)
M = np.reshape(M, (3,4))
```

Note: Must reshape M afterwards!

Unknown Camera Parameters

Known 2d image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations

- Or, solve for m's entries using total linear least-squares
- Method 2 – **Ax=0** form
  - Find non-trivial solution (not A=0)

$$\underbrace{\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix}}_{\text{A}} \underbrace{\begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix}}_{\text{x}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

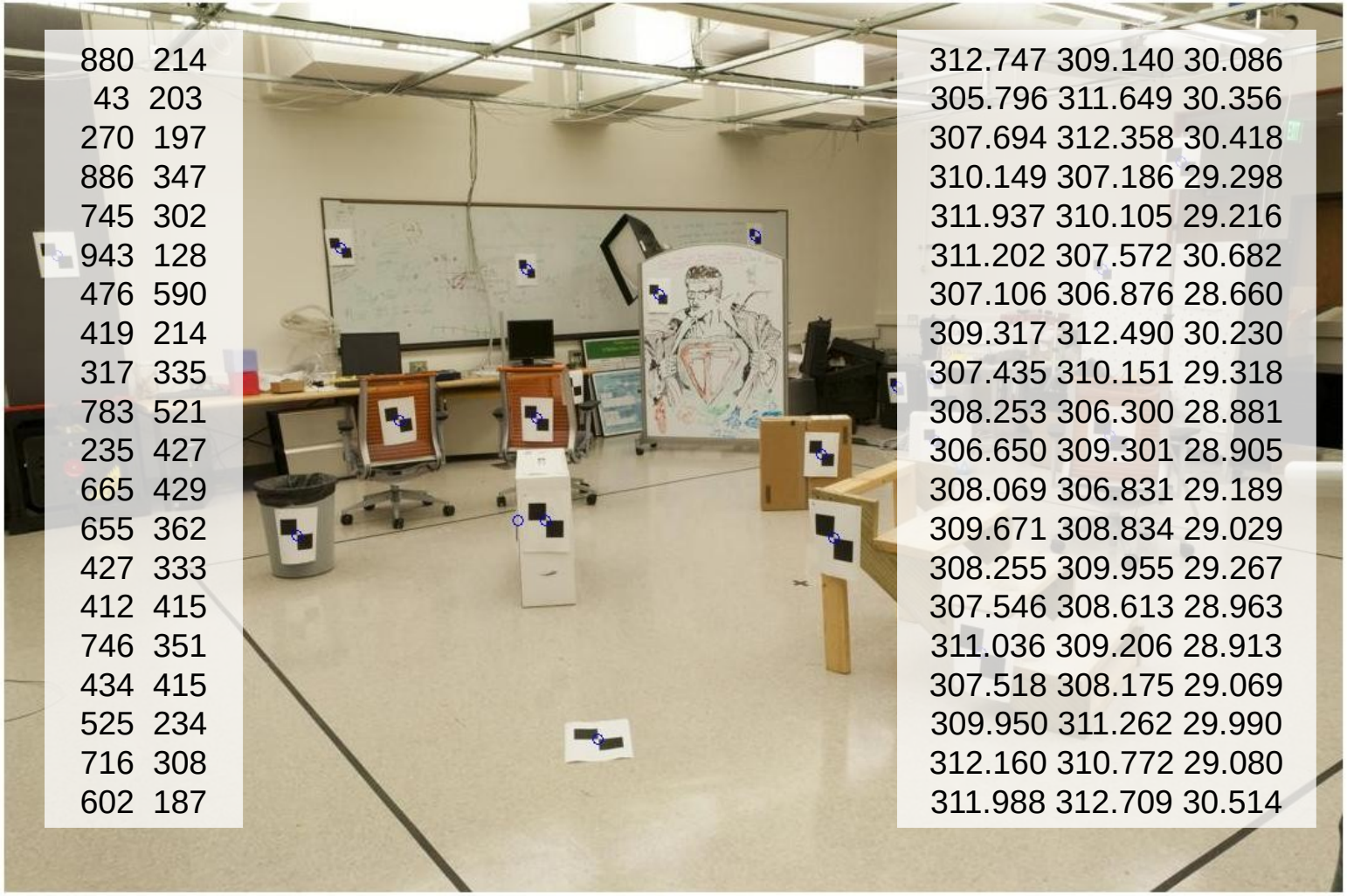
```
MATLAB:
[U, S, V] = svd(A);
M = V(:,end);
M = reshape(M,[],3)';

Python Numpy:
U, S, Vh =
np.linalg.svd(a)
# V = Vh.T
M = Vh[-1,:]
M = np.reshape(M, (3,4))
```

last row instead of last column

How do we calibrate a camera?

Known 2d image coords

Known 3d world locations



880	214
43	203
270	197
886	347
745	302
943	128
476	590
419	214
317	335
783	521
235	427
665	429
655	362
427	333
412	415
746	351
434	415
525	234
716	308
602	187

312.747	309.140	30.086
305.796	311.649	30.356
307.694	312.358	30.418
310.149	307.186	29.298
311.937	310.105	29.216
311.202	307.572	30.682
307.106	306.876	28.660
309.317	312.490	30.230
307.435	310.151	29.318
308.253	306.300	28.881
306.650	309.301	28.905
308.069	306.831	29.189
309.671	308.834	29.029
308.255	309.955	29.267
307.546	308.613	28.963
311.036	309.206	28.913
307.518	308.175	29.069
309.950	311.262	29.990
312.160	310.772	29.080
311.988	312.709	30.514

Known 2d image coords

Known 3d world locations

1st point

$(u1, v1)$	$(X1, Y1, Z1)$
880 214	312.747 309.140 30.086
43 203	305.796 311.649 30.356
270 197	307.694 312.358 30.418
886 347	310.149 307.186 29.298
745 302	311.937 310.105 29.216
943 128	311.202 307.572 30.682
476 590	307.106 306.876 28.660
419 214	309.317 312.490 30.230
317 335	307.435 310.151 29.318
…	…..

Projection error defined by two equations – one for *u* and one for *v*

$$\begin{bmatrix} 312.747 & 309.140 & 30.086 & 1 & 0 & 0 & 0 & 0 & -880\times312.747 & -880\times309.140 & -880\times30.086 & -880 \\ 0 & 0 & 0 & 0 & 312.747 & 309.140 & 30.086 & 1 & -214\times312.747 & -214\times309.140 & -214\times30.086 & -214 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Known 2d image coords

Known 3d world locations

2nd point

…

…..

Projection error defined by two equations – one for *u* and one for *v*

$$\begin{bmatrix} 312.747 & 309.140 & 30.086 & 1 & 0 & 0 & 0 & 0 & -880\times312.747 & -880\times309.140 & -880\times30.086 & -880 \\ 0 & 0 & 0 & 0 & 312.747 & 309.140 & 30.086 & 1 & -214\times312.747 & -214\times309.140 & -214\times30.086 & -214 \\ 305.796 & 311.649 & 30.356 & 1 & 0 & 0 & 0 & 0 & -43\times305.796 & -43\times311.649 & -43\times30.356 & -43 \\ 0 & 0 & 0 & 0 & 305.796 & 311.649 & 30.356 & 1 & -203\times305.796 & -203\times311.649 & -43\times30.356 & -203 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1_n & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

How many points do I need to fit the model?

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

Degrees of freedom?

5 dof

6 dof (3 rotation + 3 translation)

$$w\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}\alpha & s & u_0\\0 & \beta & v_0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_x\\r_{21} & r_{22} & r_{23} & t_y\\r_{31} & r_{32} & r_{33} & t_z\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

Think 3:

- Rotation around x
- Rotation around y
- Rotation around z

This matrix projects a 4D space to a 3D space:
One of the dimensions is lost – intuitive, dimensionality reduction machine.

How many points do I need to fit the model?

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

Degrees of freedom? 5 6

$$w\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}\alpha & s & u_0\\0 & \beta & v_0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_x\\r_{21} & r_{22} & r_{23} & t_y\\r_{31} & r_{32} & r_{33} & t_z\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

M is 3x4, so 12 unknowns, but projective scale ambiguity – 11 deg. freedom.
One equation per unknown -> 5 1/2 point correspondences determines a solution (e.g., either u or v).

More than 5 1/2 point correspondences -> overdetermined, many solutions to **M**.
Least squares is finding the solution that best satisfies the overdetermined system.

Why use more than 6? Robustness to error in feature points.

Calibration with linear method

- Advantages
  - Easy to formulate and solve
  - Provides initialization for non-linear methods

- Disadvantages
  - Doesn’t directly give you camera parameters
  - Doesn’t model radial distortion
  - Can’t impose constraints, such as known focal length

- Non-linear methods are preferred
  - Define error as difference between projected points and measured points
  - Minimize error using Newton’s method or other non-linear optimization

Can we factorize M back to K [R | T]?

- Yes!
- We can directly solve for the individual entries of K [R | T].

$\mathbf{a}_n$ = nth column of *A*

Extracting camera parameters

$$\frac{M}{\rho} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha \cot\theta \boldsymbol{r}_2^T + u_0 \boldsymbol{r}_3^T & \alpha t_x - \alpha \cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta} \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T & \frac{\beta}{\sin\theta} t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix} = K\begin{bmatrix} R & T \end{bmatrix}$$

A $\quad$ $\mathbf{b}$

$$K = \begin{bmatrix} \alpha & -\alpha\cot\theta & u_o \\ 0 & \frac{\beta}{\sin\theta} & v_o \\ 0 & 0 & 1 \end{bmatrix}$$

Box 1

$$A = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \mathbf{a}_3^T \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Estimated values

Intrinsic

$$\rho = \frac{\pm 1}{|\mathbf{a}_3|} \qquad u_o = \rho^2(\mathbf{a}_1 \cdot \mathbf{a}_3) \qquad v_o = \rho^2(\mathbf{a}_2 \cdot \mathbf{a}_3)$$

$$\cos\theta = \frac{(\mathbf{a}_1 \times \mathbf{a}_3) \cdot (\mathbf{a}_2 \times \mathbf{a}_3)}{|\mathbf{a}_1 \times \mathbf{a}_3| \cdot |\mathbf{a}_2 \times \mathbf{a}_3|}$$

Extracting camera parameters

$$\frac{\mathcal{M}}{\rho} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha \cot\theta \boldsymbol{r}_2^T + u_0 \boldsymbol{r}_3^T & \alpha t_x - \alpha \cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta} \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T & \frac{\beta}{\sin\theta} t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix} = \mathrm{K}\begin{bmatrix}\mathrm{R} & \mathrm{T}\end{bmatrix}$$

A **b**

$$\mathrm{A} = \begin{bmatrix} \mathbf{a}_1^{\mathrm{T}} \\ \mathbf{a}_2^{\mathrm{T}} \\ \mathbf{a}_3^{\mathrm{T}} \end{bmatrix} \qquad \mathrm{b} = \begin{bmatrix} \mathrm{b}_1 \\ \mathrm{b}_2 \\ \mathrm{b}_3 \end{bmatrix}$$

Estimated values

Intrinsic

$$\alpha = \rho^2 |\mathbf{a}_1 \times \mathbf{a}_3| \sin\theta$$

$$\beta = \rho^2 |\mathbf{a}_2 \times \mathbf{a}_3| \sin\theta$$

Extracting camera parameters

$$\frac{\mathcal{M}}{\rho} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha \cot\theta \boldsymbol{r}_2^T + u_0 \boldsymbol{r}_3^T & \alpha t_x - \alpha \cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta} \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T & \frac{\beta}{\sin\theta} t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix} = \mathrm{K}\begin{bmatrix}\mathrm{R} & \mathrm{T}\end{bmatrix}$$

$\mathbf{A}$ $\mathbf{b}$

$$\mathrm{A} = \begin{bmatrix} \mathbf{a}_1^{\mathrm{T}} \\ \mathbf{a}_2^{\mathrm{T}} \\ \mathbf{a}_3^{\mathrm{T}} \end{bmatrix} \qquad \mathrm{b} = \begin{bmatrix} \mathrm{b}_1 \\ \mathrm{b}_2 \\ \mathrm{b}_3 \end{bmatrix}$$

Estimated values

Extrinsic

$$\mathbf{r}_1 = \frac{(\mathbf{a}_2 \times \mathbf{a}_3)}{|\mathbf{a}_2 \times \mathbf{a}_3|} \qquad \mathbf{r}_3 = \frac{\pm \mathbf{a}_3}{|\mathbf{a}_3|}$$

$$\mathbf{r}_2 = \mathbf{r}_3 \times \mathbf{r}_1 \qquad \mathrm{T} = \rho\, \mathrm{K}^{-1}\mathbf{b}$$

Can we factorize M back to K [R | T]?

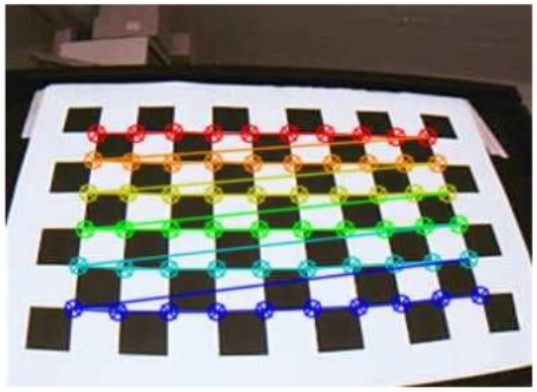
- Yes!
- We can also use *RQ* factorization (not QR)
  - R in RQ is not rotation matrix R; crossed names!
- *R* (right diagonal) is K
- *Q* (orthogonal basis) is R.
- T, the last column of [R | T], is inv(K) * last column of M.
  - But you need to do a bit of post-processing to make sure that the matrices are valid. See for full implementation: http://ksimek.github.io/2012/08/14/decompose/

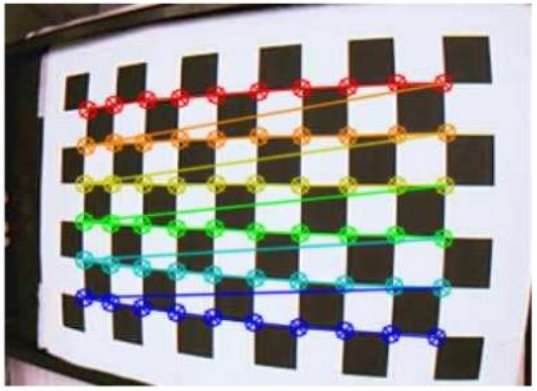
Many methods to calibrate a camera!

- Calibration using 3D calibration object
- Calibration using 2D planar pattern (check this)
- Calibration using 1D object (line-based calibration)
- Self Calibration: no calibration objects using:
  - vanishing points
  - Really modern methods:
    - Absolute conics
    - Absolute Quadrics
  - etc.

Camera Calibration with 2D Pattern

- For more detail: check this
- Use a known 2d planar object with many photos of it
- Main ‘tricks’:
  - Your ‘world origin’ is one corner of the checkerboard (usually top left)
  - Your Z coordinate is 0, so your M matrix changes a bit





There are also many camera models!!

- Pinhole
- atan: pinhole + distortion (one parameter - paper)
- fisheye model: pinhole + distortion (4-5 parameters)
- Omnidirectional cameras (toolbox)
- and many many more!

Commonly used

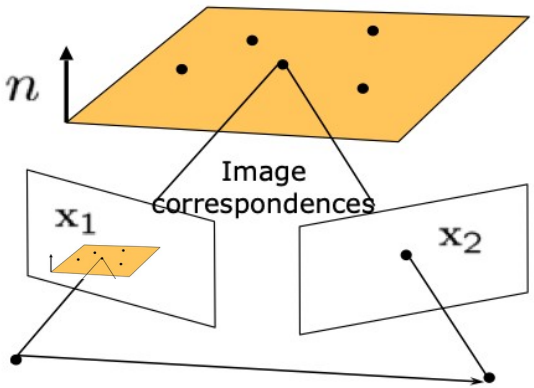
default opencv calibration model

What happens when you violate the laws of geometric image formation

Erik Johansson – The Architect

Homography

HOMOGRAPHY



A homography is a transformation from a projective plane to another projective plane

$$\lambda_1 \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H_1 \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$\lambda_2 \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = H_2 \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$\lambda \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = H_2 H_1^{-1} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

[LHS and RHS are related by a scale factor]

[Aside: H usually invertible]

Computing homography projections

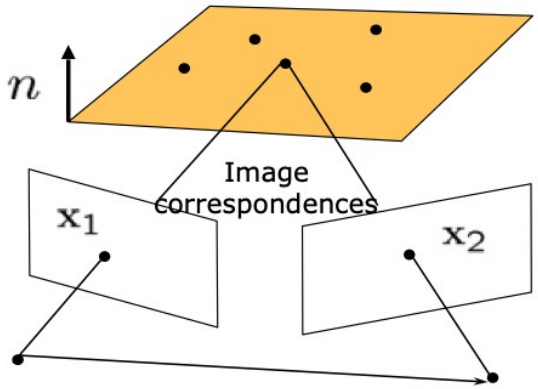
Given (x_1,y_1) and H, how do we compute (x_2,y_2)?

$$\lambda \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$x_2 = \frac{\lambda x_2}{\lambda} = \frac{ax_1 + by_1 + c}{gx_1 + hy_1 + i}$$

Estimating homographies

Given corresponding 2D points in left and right image, estimate H



$$x_2(gx_1 + hy_1 + i) = ax_1 + by_1 + c$$

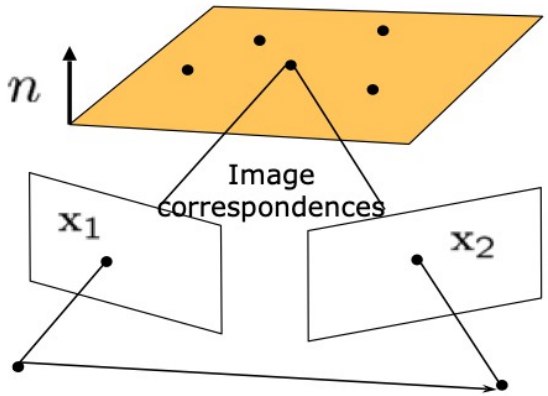
$$\vdots$$

$$AH(:) = \begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}$$ Homogenous linear system

How many corresponding points needed?
How many degrees of freedom in H?

Estimating homographies

Given corresponding 2D points in left and right image, estimate H

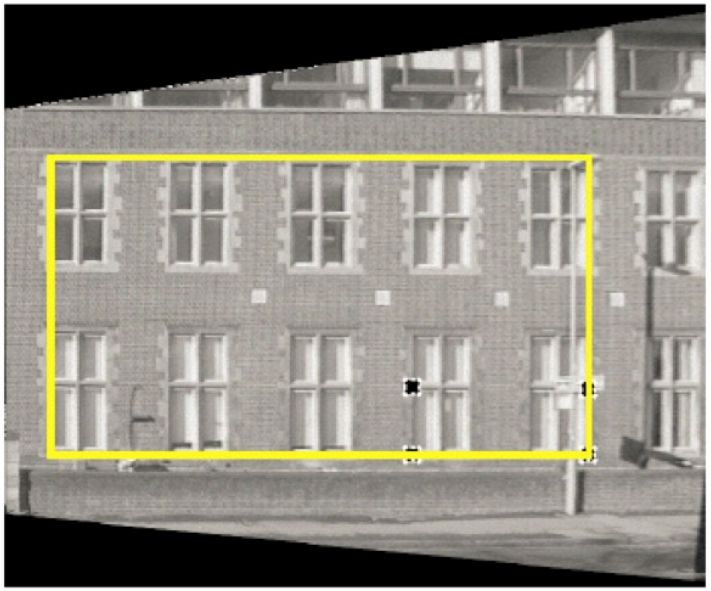


$$AH(:) = \begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}$$

H is determined only up to scale factor (8 DOFs)
Need 4 points minimum. How to handle more points?

$$\min_{||H(:)||^2=1} ||AH(:)||^2$$

“Frontalizing” planes using homographies

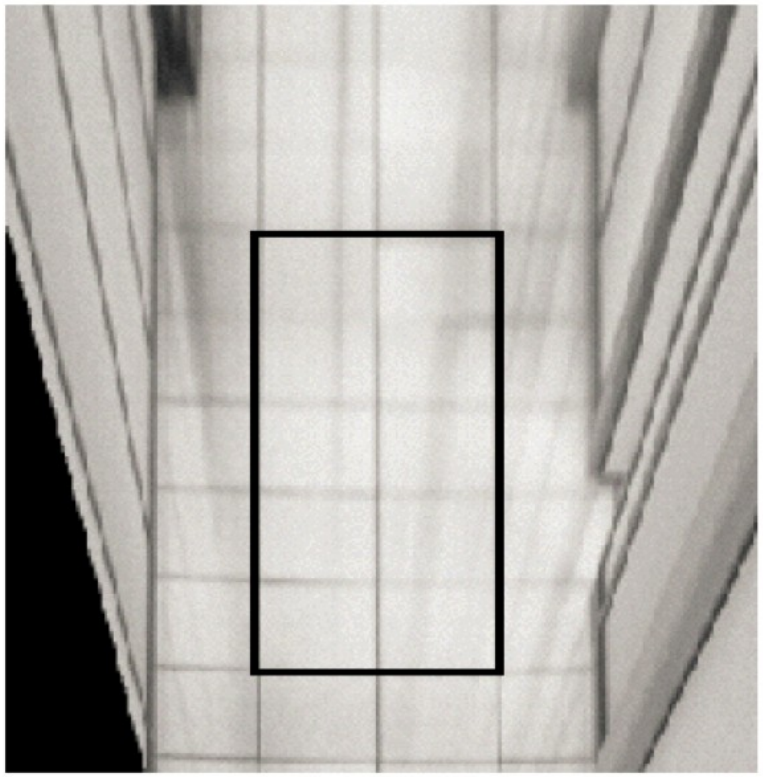


from Hartley & Zisserman

Estimate homography on (at least) 4 pairs of corresponding points (e.g., corners of quad/rect)

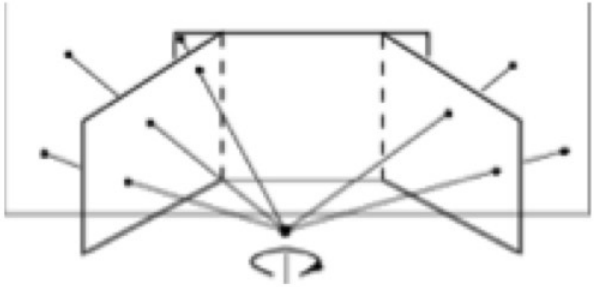
Apply homography on all (x,y) coordinates inside target rectangle to compute source pixel location

“Frontalizing” planes using homographies

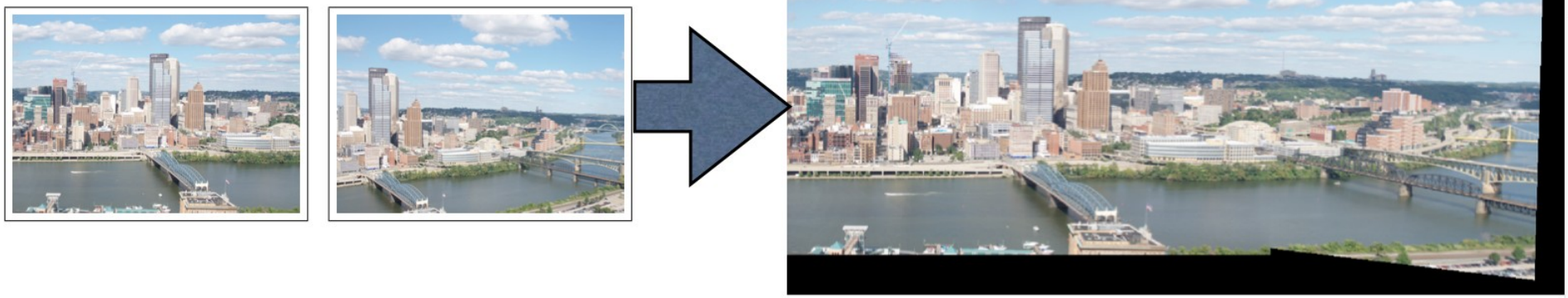


from Hartley & Zisserman

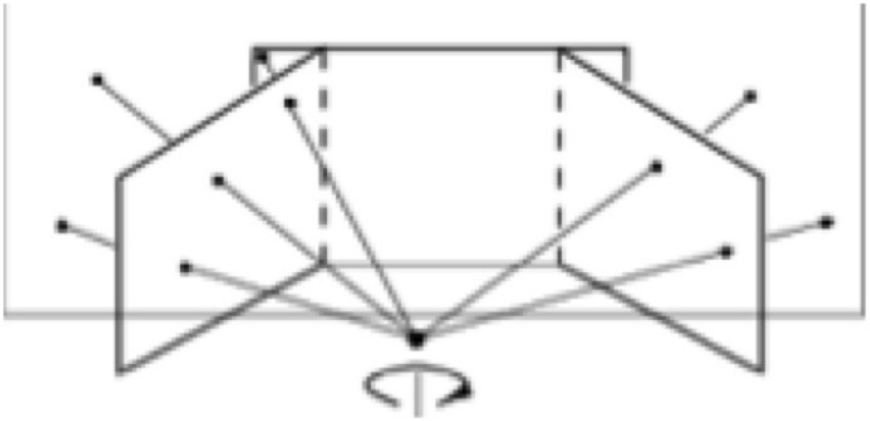
Special case of 2 views: rotations about camera center



Can be modeled as planar transformations, regardless of scene geometry!



Derivation



$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} = R \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix}$$

K_2

$$\lambda_2 \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} f_2 & 0 & 0 \\ 0 & f_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix}$$

$$\cdots$$

$$\lambda \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = K_2 R K_1^{-1} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Take-home points for homographies

$$\lambda \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

- If camera rotates about its center, then the images are related by a homography irrespective of scene depth.
- If the scene is planar, then images from any two cameras are related by a homography.
- Homography mapping is a 3x3 matrix with 8 degrees of freedom.

Camera calibration standard methods – Zhang ,

Checkerboard calibration:
http://www.vision.caltech.edu/bouguetj/calib_doc/ (matlab)

http://robots.stanford.edu/cs223b04/JeanYvesCalib/htmls/links.html

http://www.robots.ox.ac.uk/~cmei/Toolbox.html (matlab)

OpenCV calibration

https://github.com/RViMLab/ICCV2017-plenoptic-camera-calibration

https://calib.io/blogs/knowledge-base/camera-calibration

Some useful notes
https://kushalvyas.github.io/calib.html