Lecture 13

Camera calibration

Administrative

A3 is out
- Due May 12th

A4 is out
- Due May 25th

Administrative

Recitation this friday

- Ontologies
- Zihan Wang

So far: 2D Transformations with homogeneous coordinates

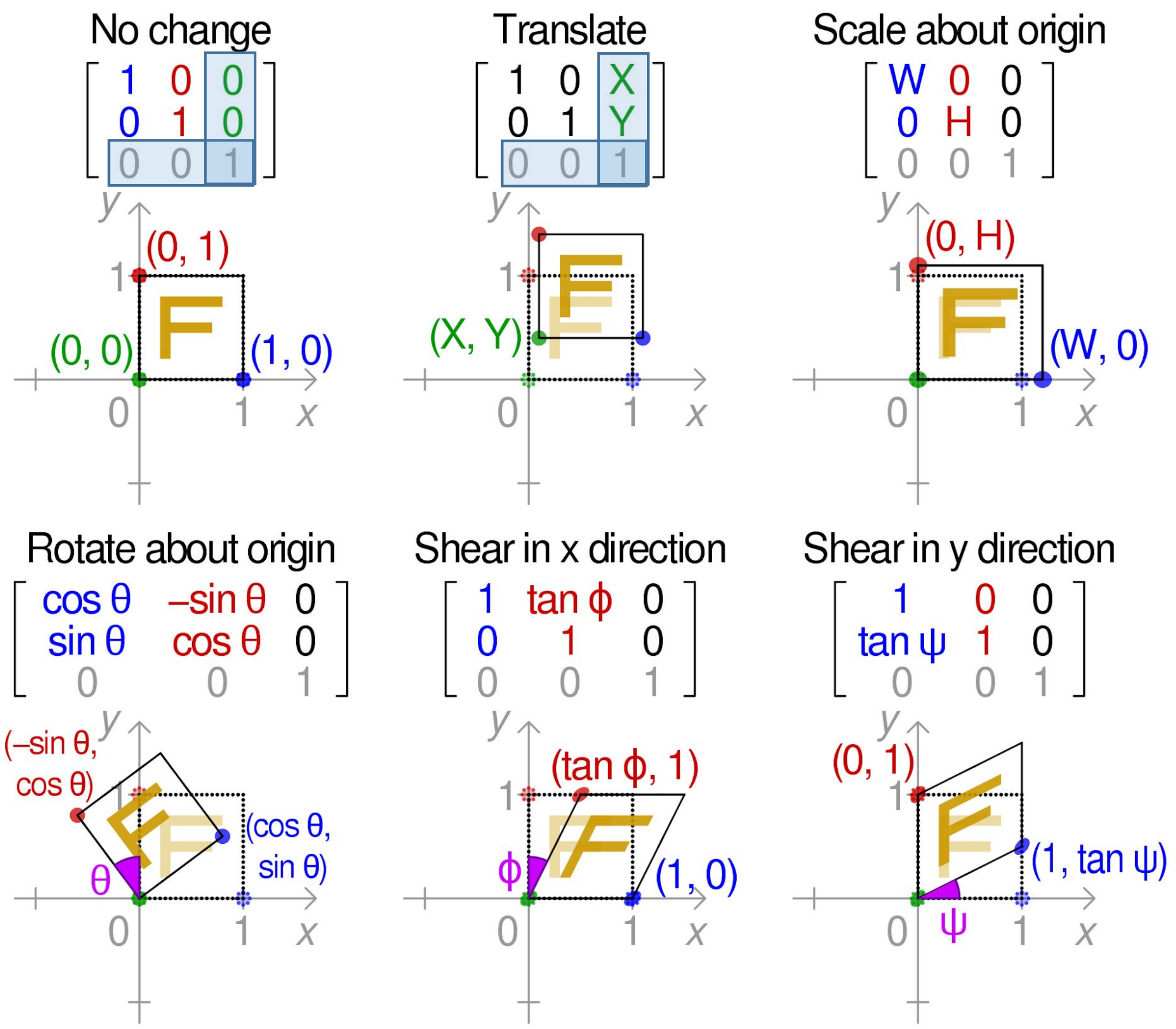
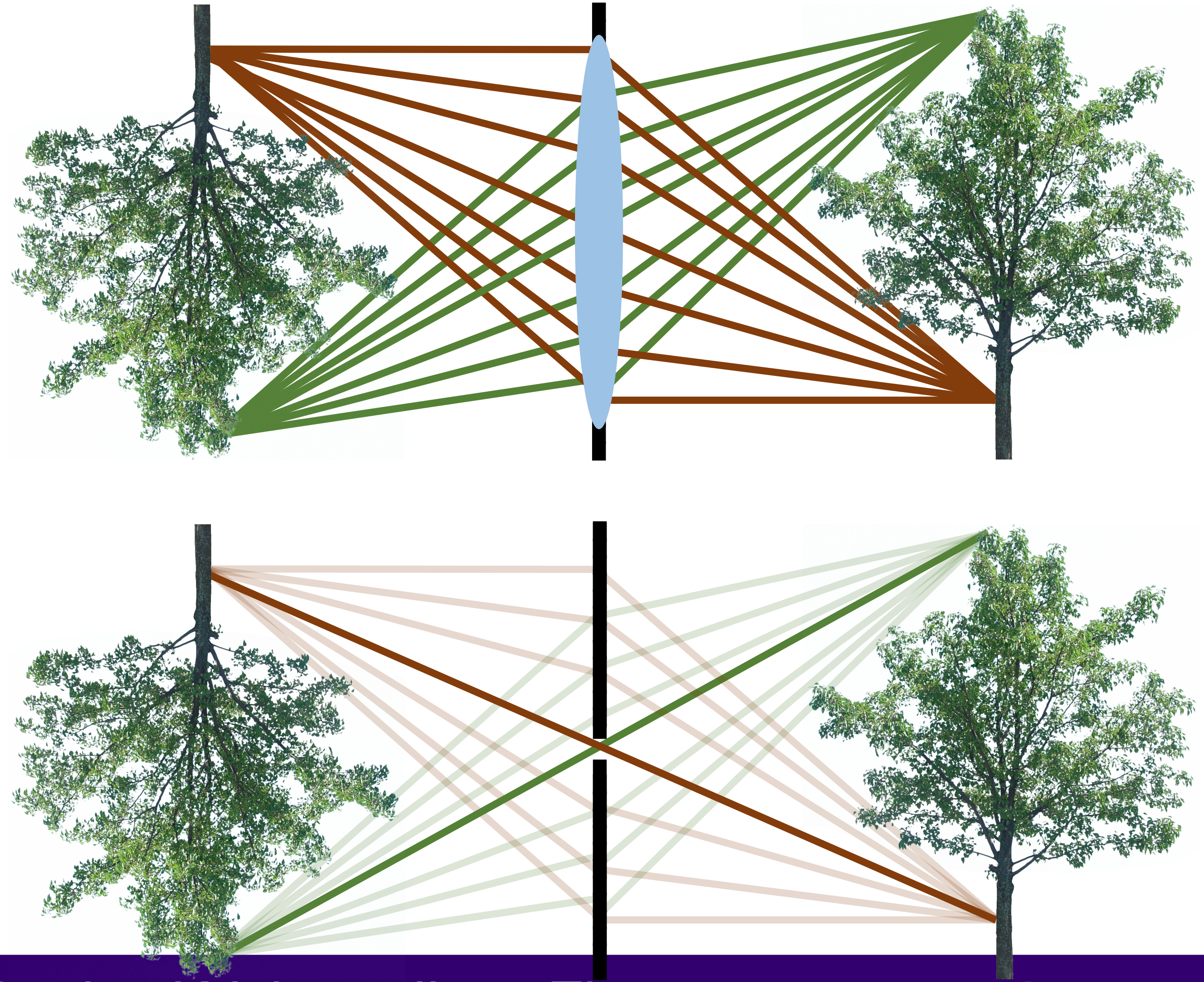


Figure: Wikipedia

So far: The pinhole camera



For this course, we focus on the pinhole model.

- Similar to thin lens model in Physics: central rays are not deviated.
- Assumes lens camera in focus.
- Useful approximation but ignores important lens distortions.

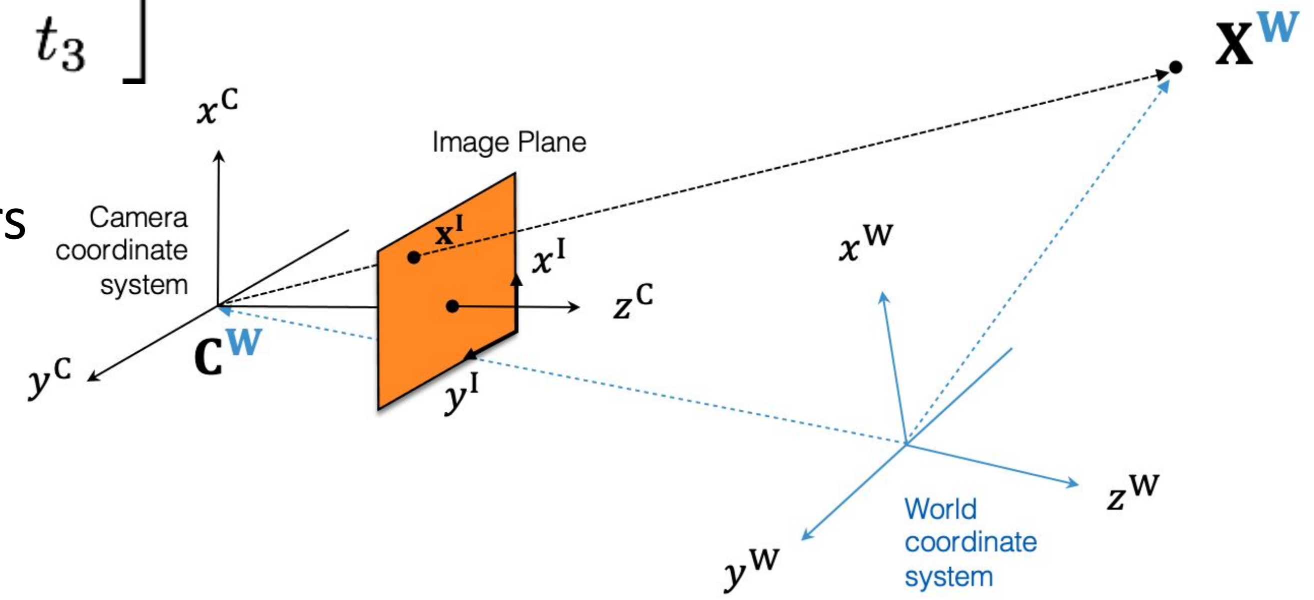
So far: General pinhole camera matrix

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \qquad \text{where} \qquad \mathbf{t} = -\mathbf{RC}$$

$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{ccc:c} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{array}\right]$$

intrinsic parameters

extrinsic parameters



Today’s agenda

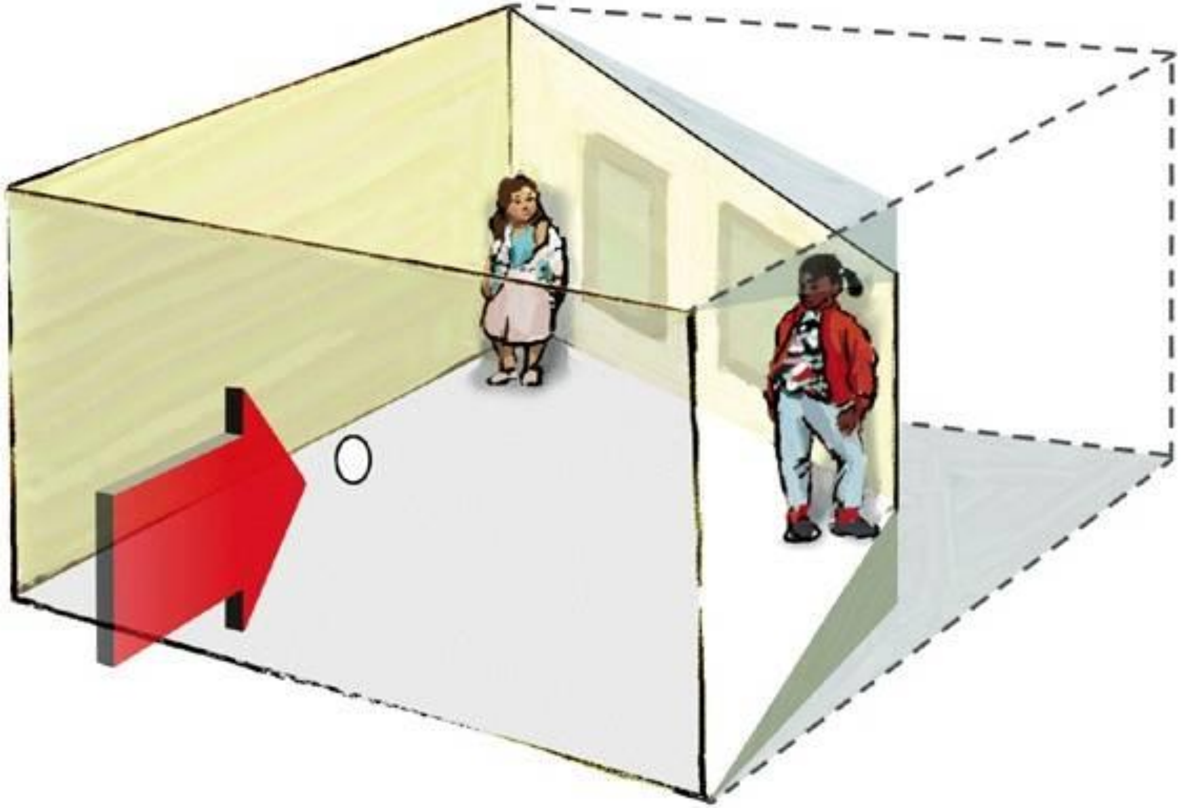
- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods

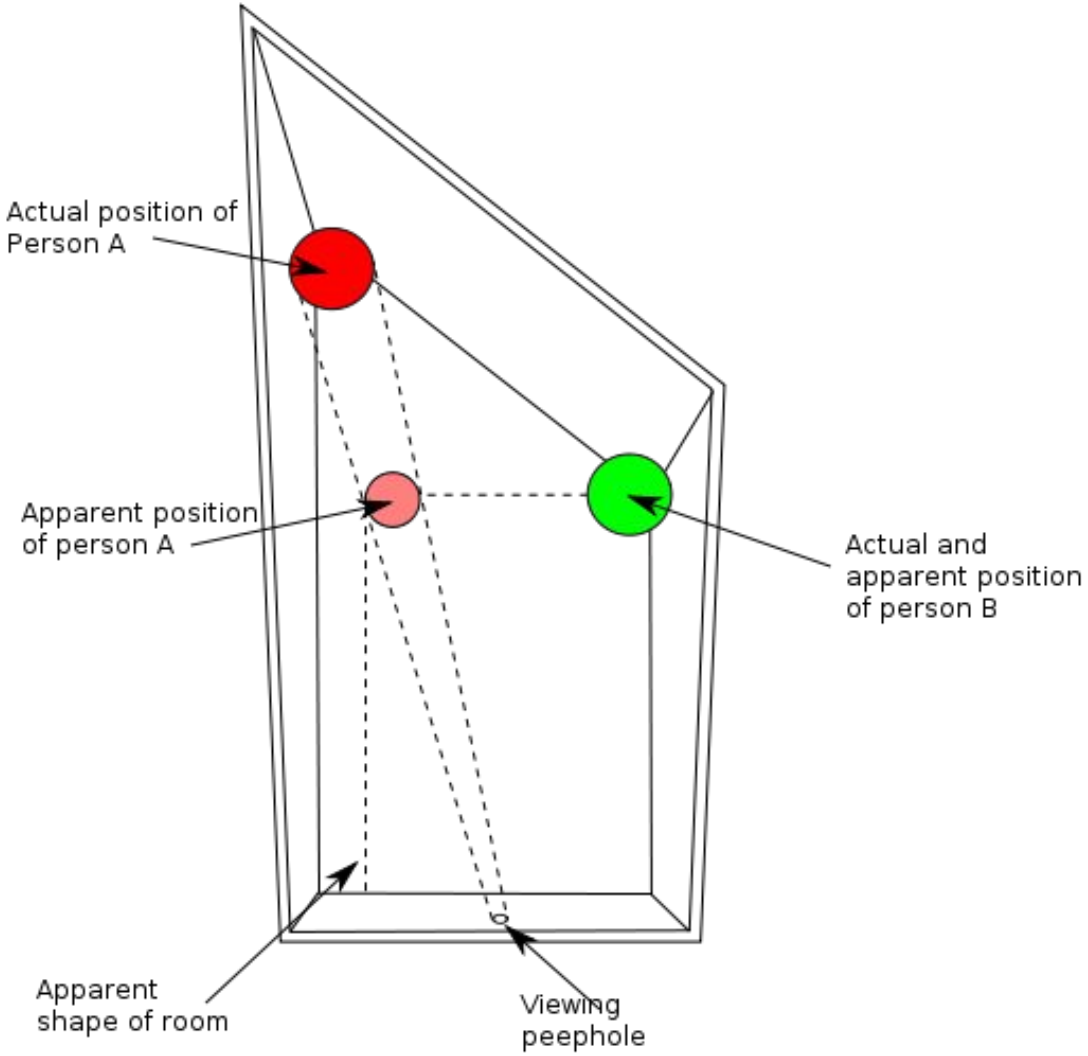
Today's agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods

Similar illusion as last lecture

The Ames room illusion

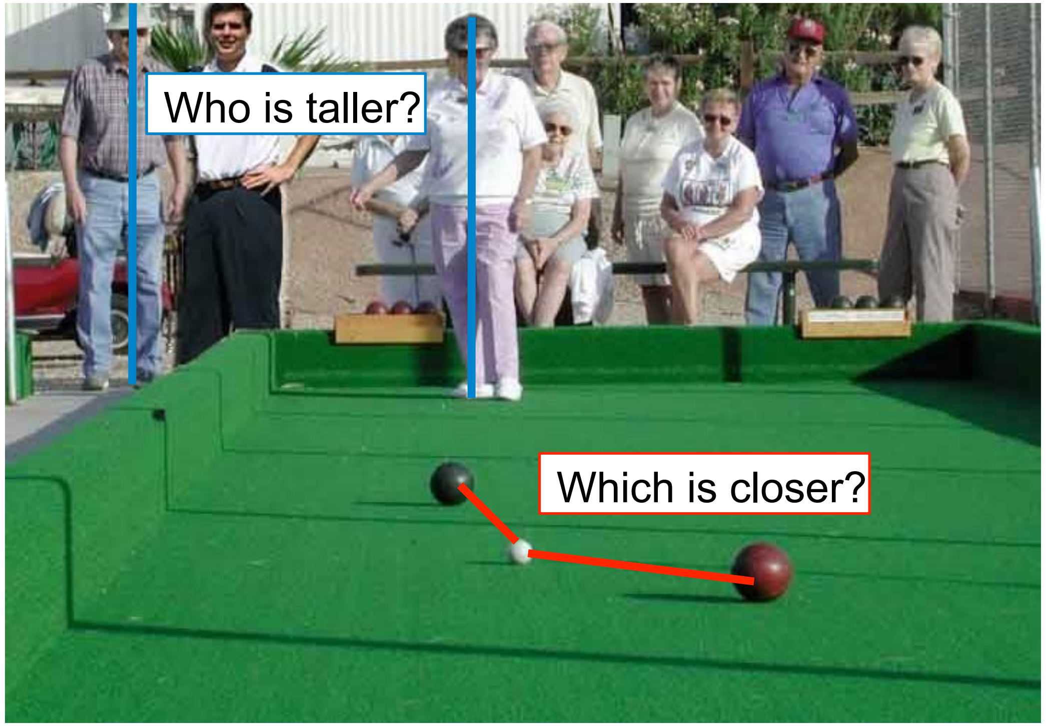




Projective Geometry

Q. Who is taller?

The two blue lines are the same length

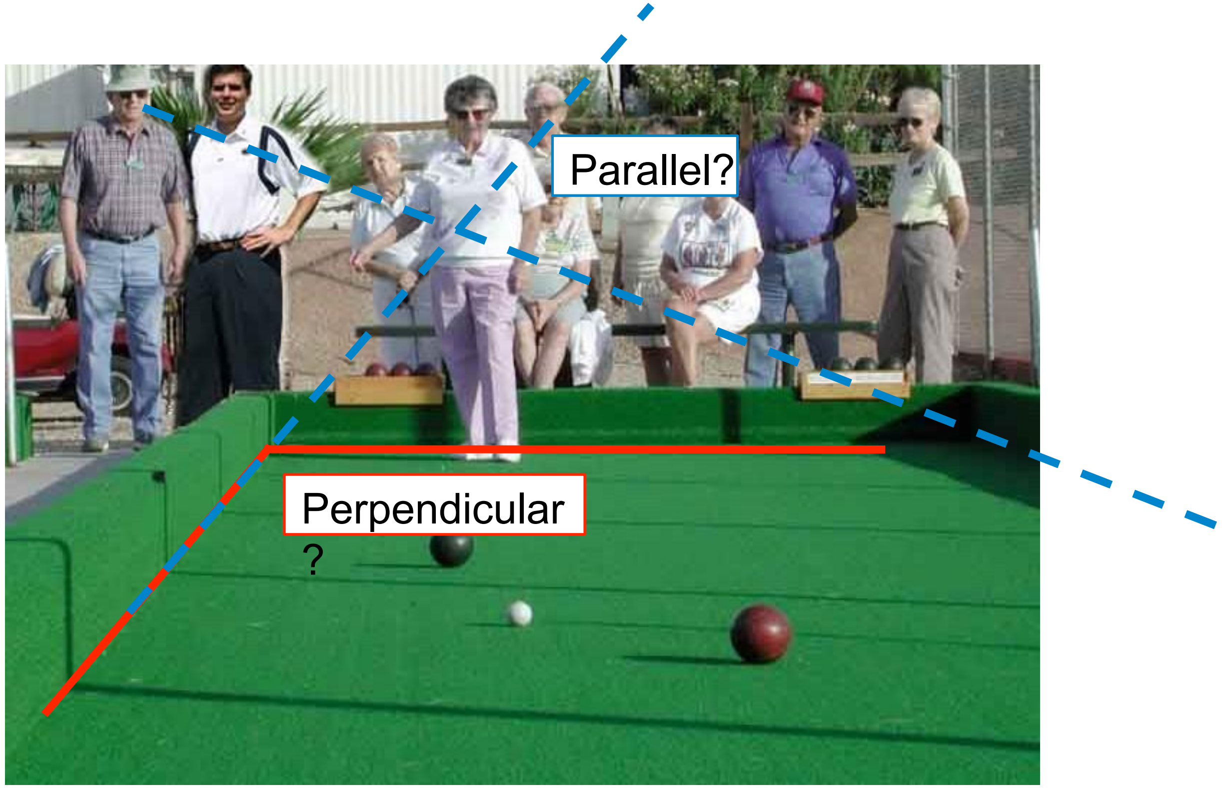


Slide credit: J. Hays

Projective Geometry

What is not preserved?

- Length
- Angles

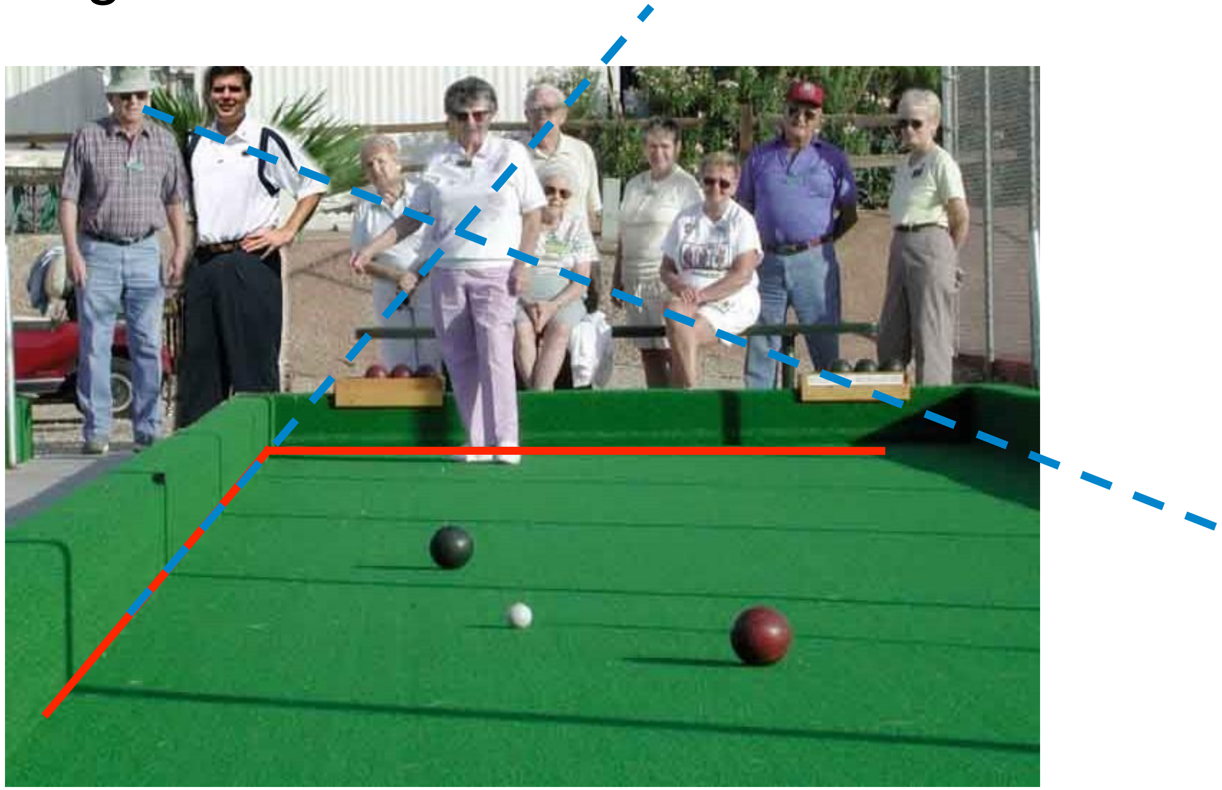


Slide credit: J. Hays

Projective Geometry

What is preserved?

- Straight lines are still straight



Slide credit: J. Hays

Projection of lines

Q. When is parallelism preserved?

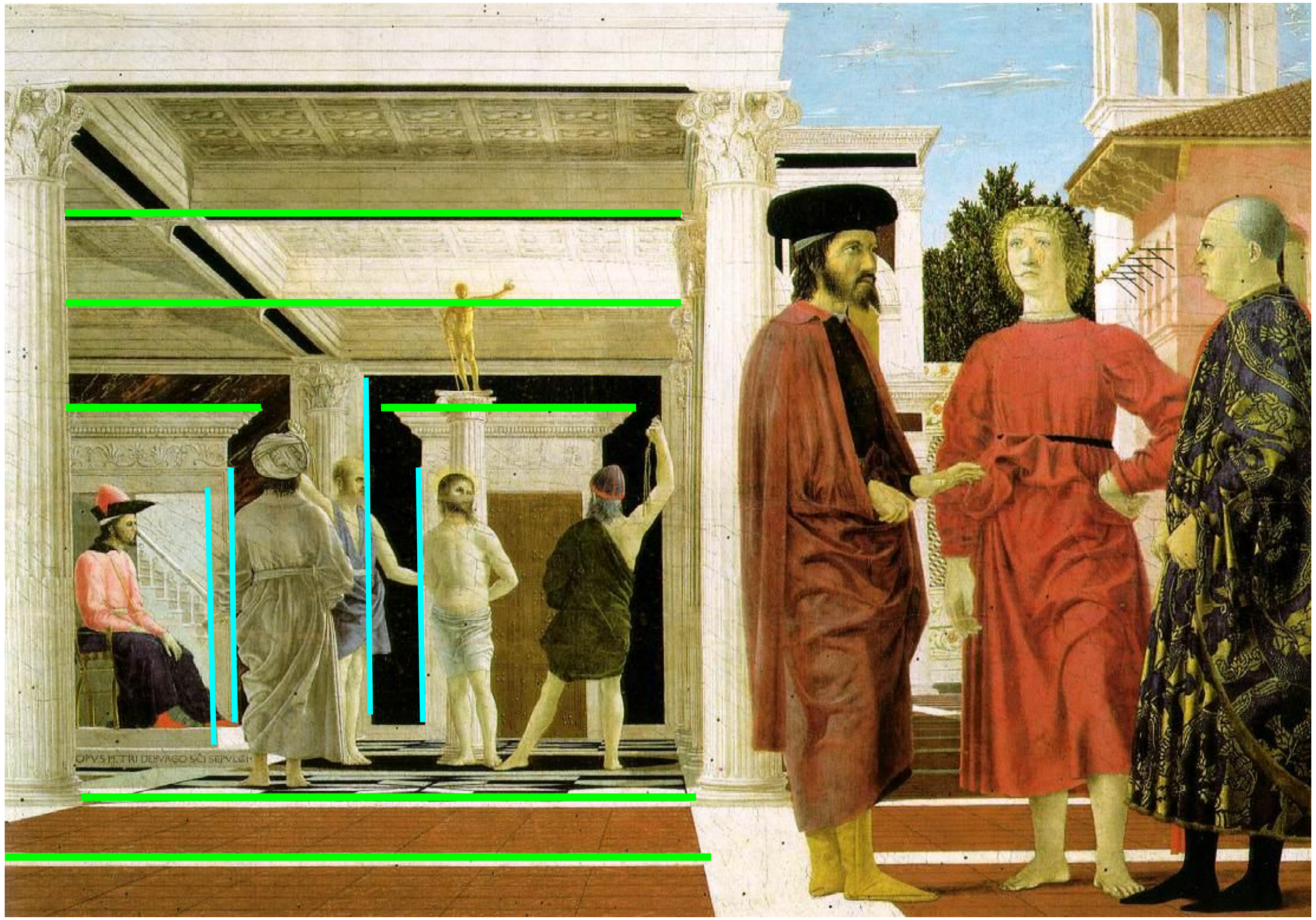
Piero della Francesca, *Flagellation of Christ*, 1455-1460

Slide: S. Lazebnik

Projection of lines

Q. When is parallelism preserved?

When the parallel lines are also parallel to the image plane

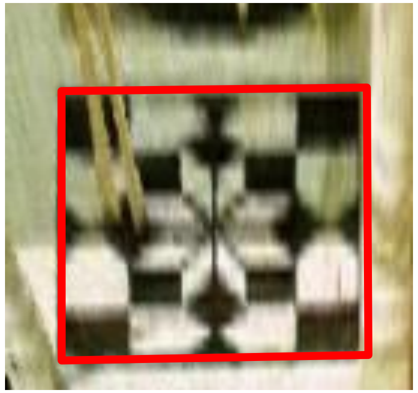


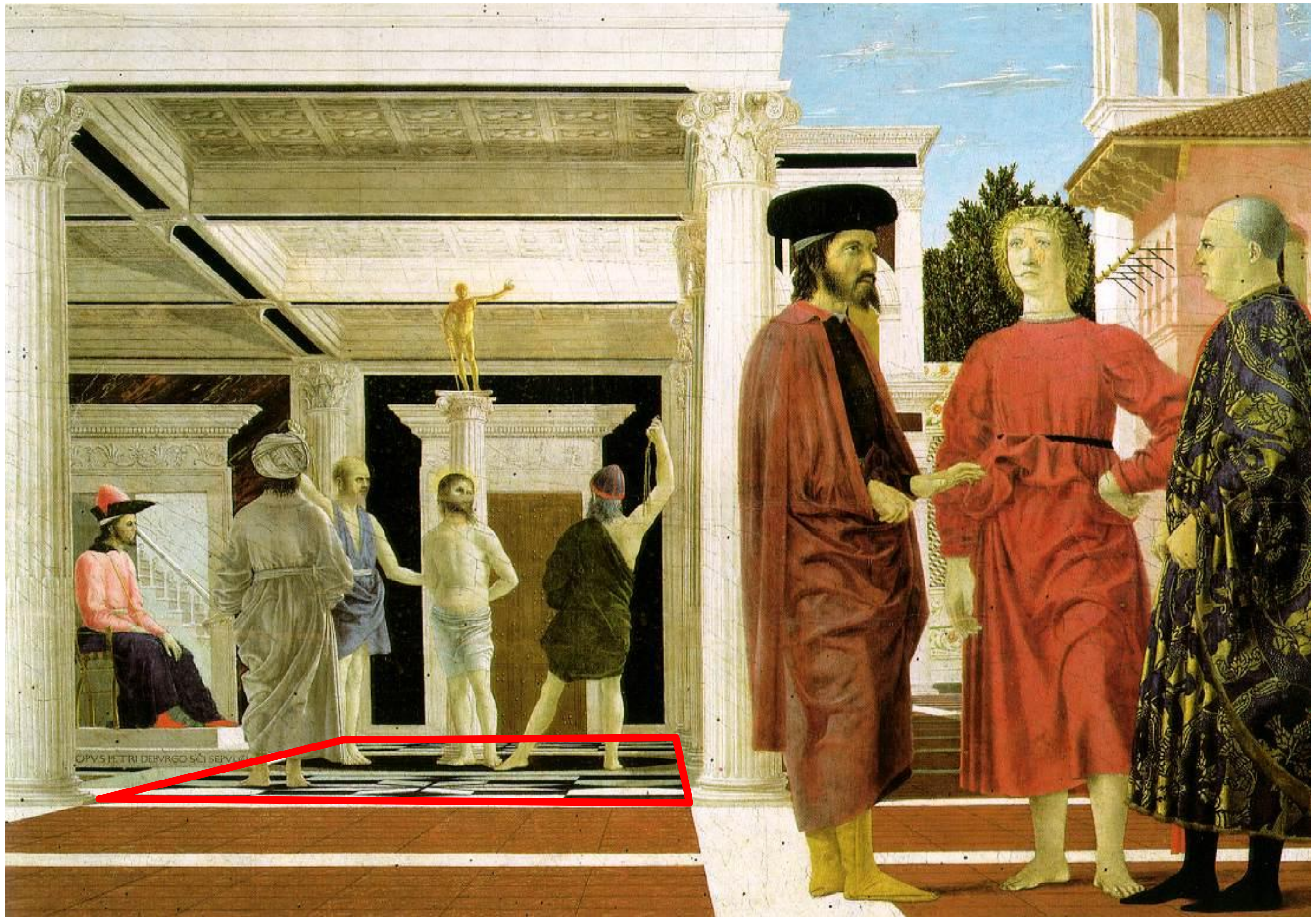
Piero della Francesca, *Flagellation of Christ*, 1455-1460

Slide: S. Lazebnik

Projection of lines

Patterns on *non-fronto-parallel* planes are distorted by a *homography*

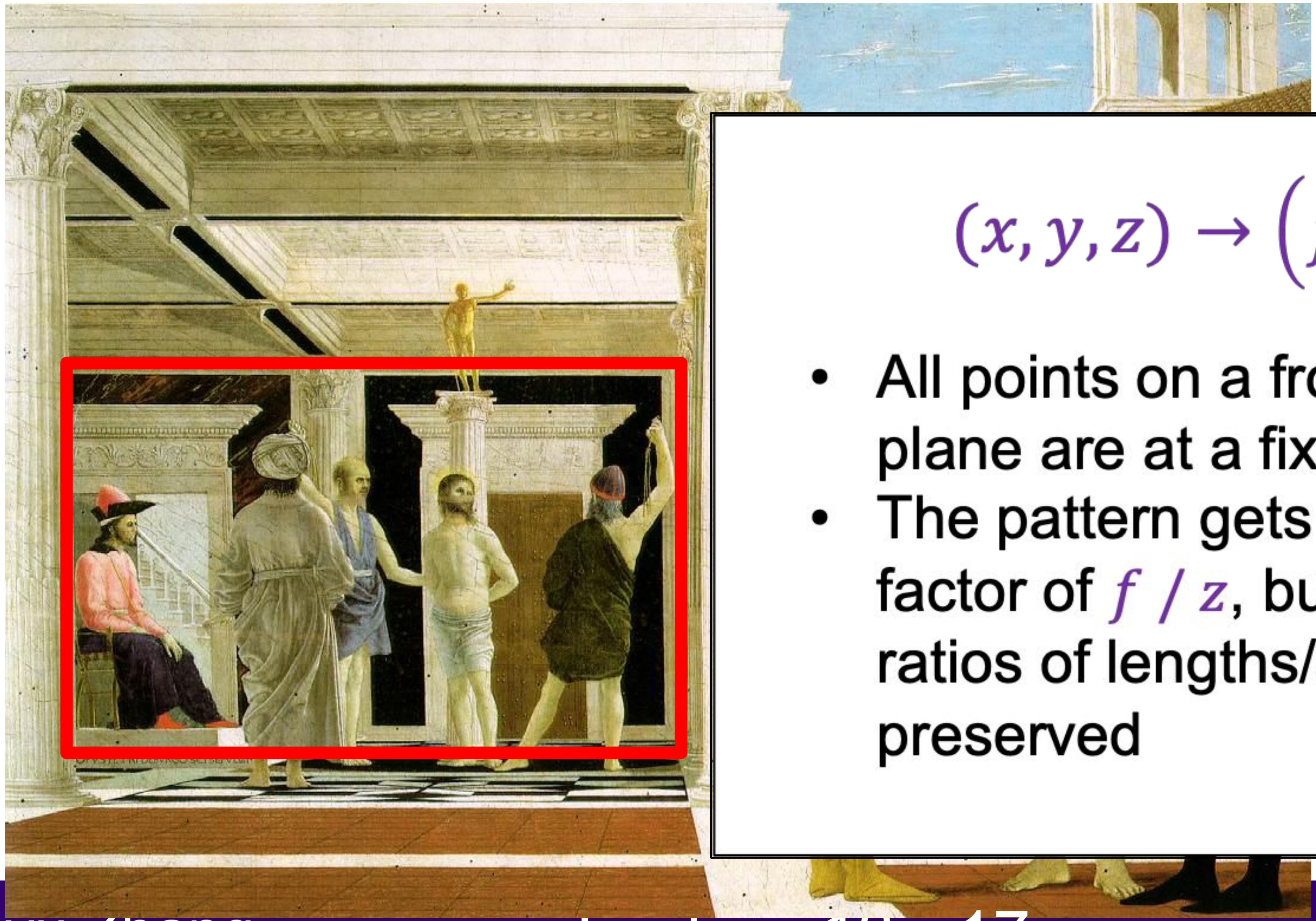




Slide: S. Lazebnik

Projection of planes

What about patterns on *fronto-parallel planes*?



Piero della Francesca, *Flagellation of Christ*, 1455-1460

$$(x, y, z) \rightarrow \left(f\frac{x}{z}, f\frac{y}{z}\right)$$

- All points on a fronto-parallel plane are at a fixed depth z
- The pattern gets scaled by a factor of $f\,/\,z$, but angles and ratios of lengths/areas are preserved

Slide: S. Lazebnik

Vanishing Points & Lines

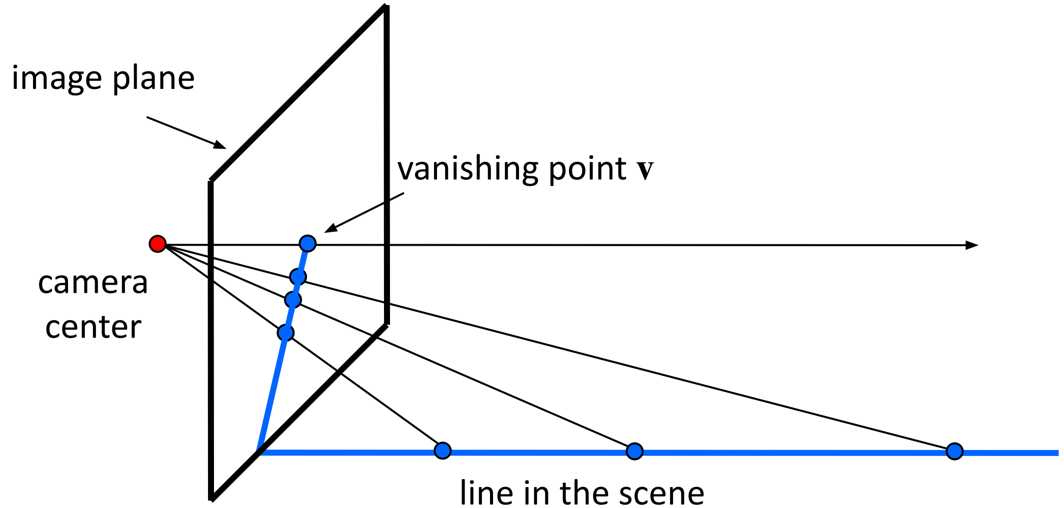


Figure: S. Lazebnik

Vanishing Points & Lines

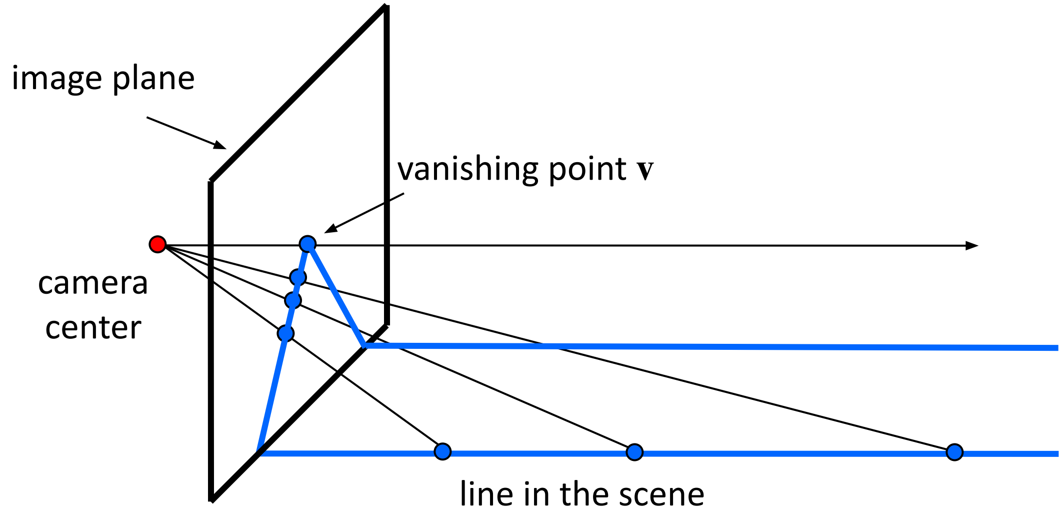
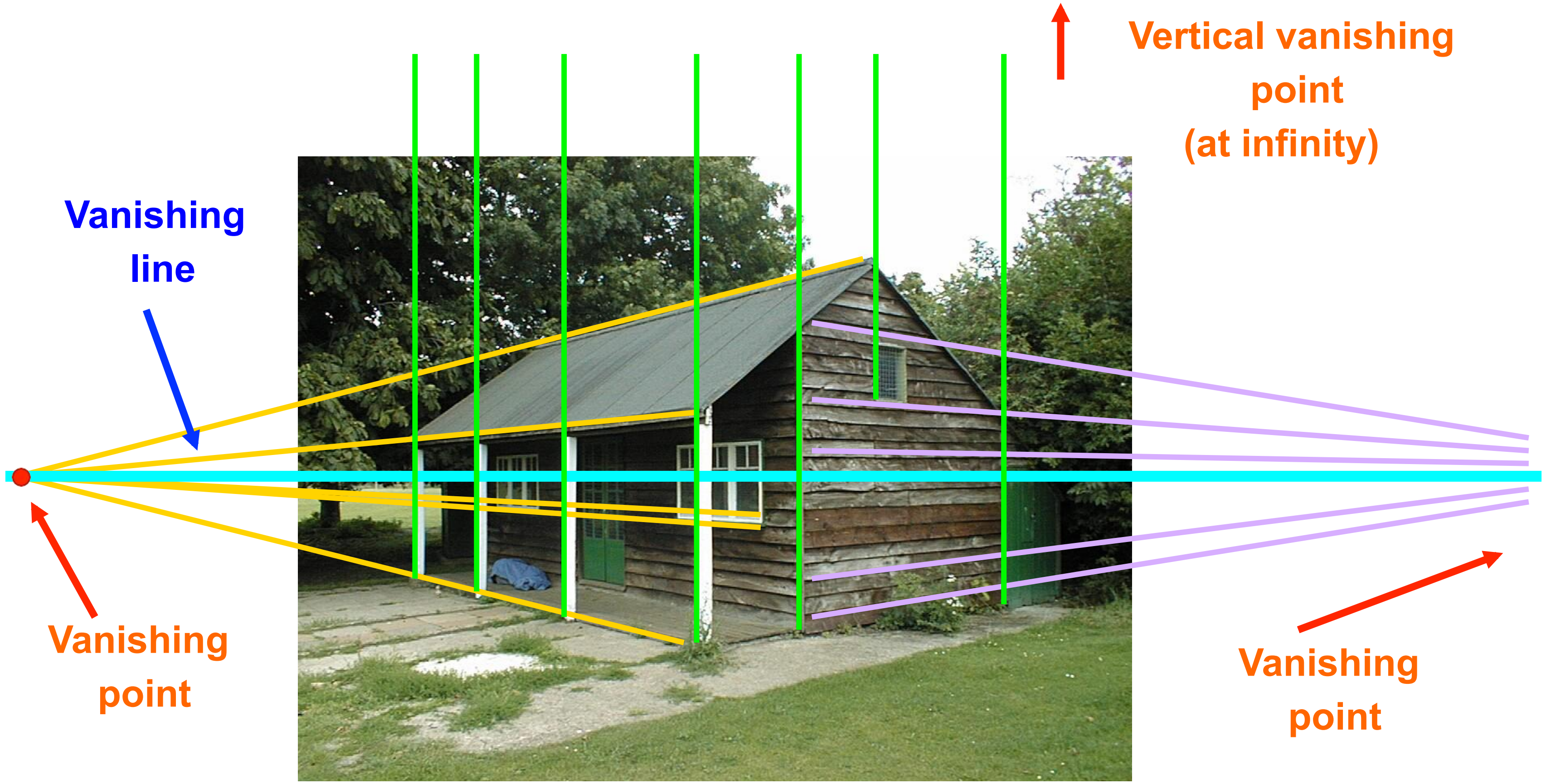


Figure: S. Lazebnik

Vanishing Points & Lines

Image: Wikipedia

Vanishing Points & Lines



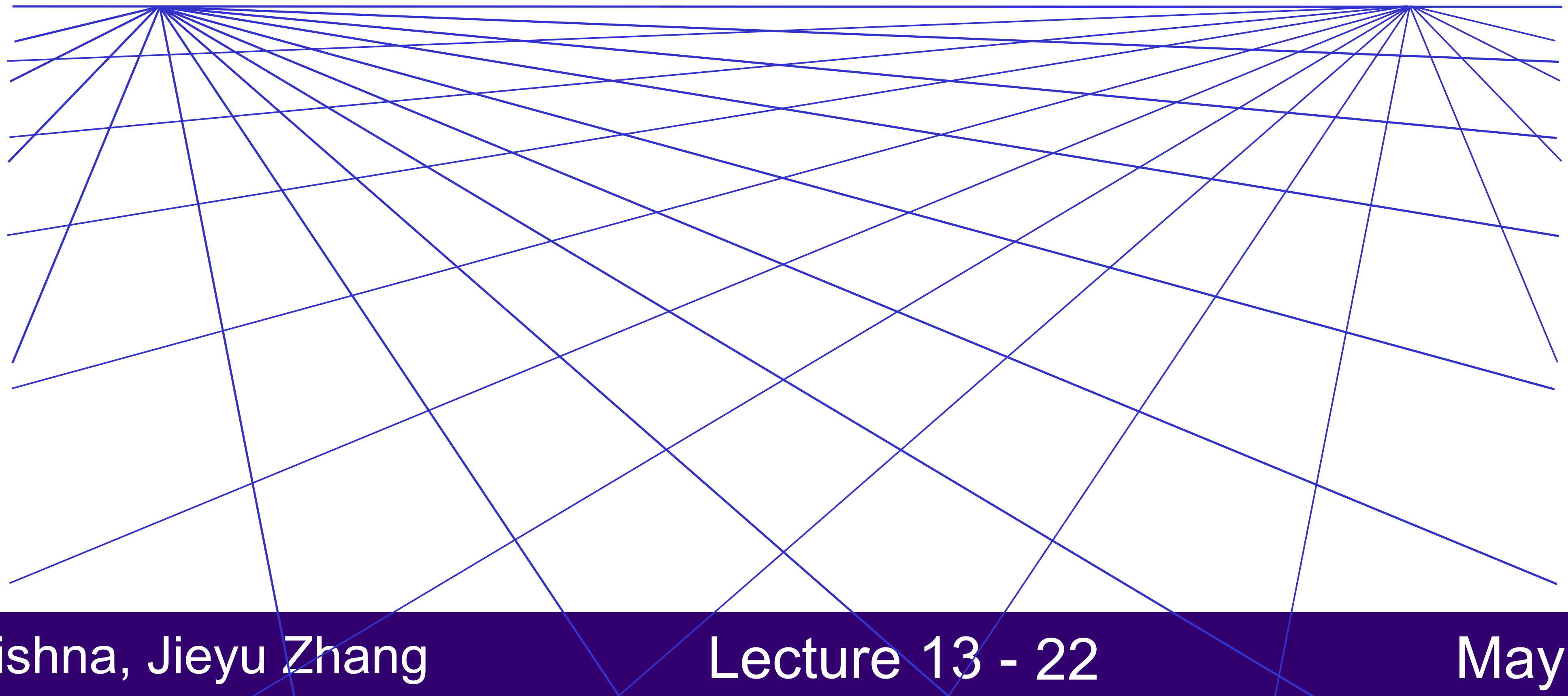
Parallel lines *in the world* intersect *in the image* at a **vanishing point**

Parallel planes *in the world* intersect *in the image* at a **vanishing line**

Slide from Efros, Photo from Criminisi

Vanishing lines of planes

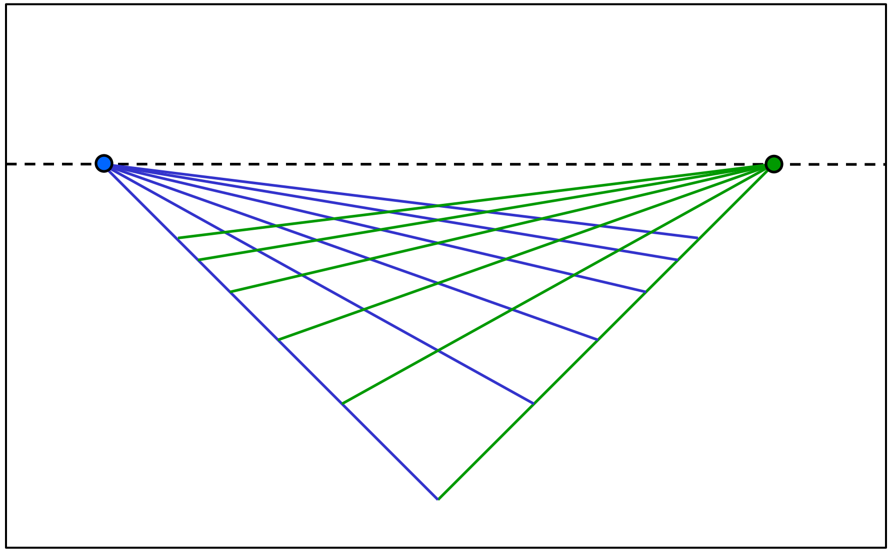
- The projection of parallel 3D planes intersect at a vanishing line
- How can we construct the vanishing line of a plane?



Vanishing lines of planes

- The projection of parallel 3D planes intersect at a vanishing line

- How can we construct the vanishing line of a plane?



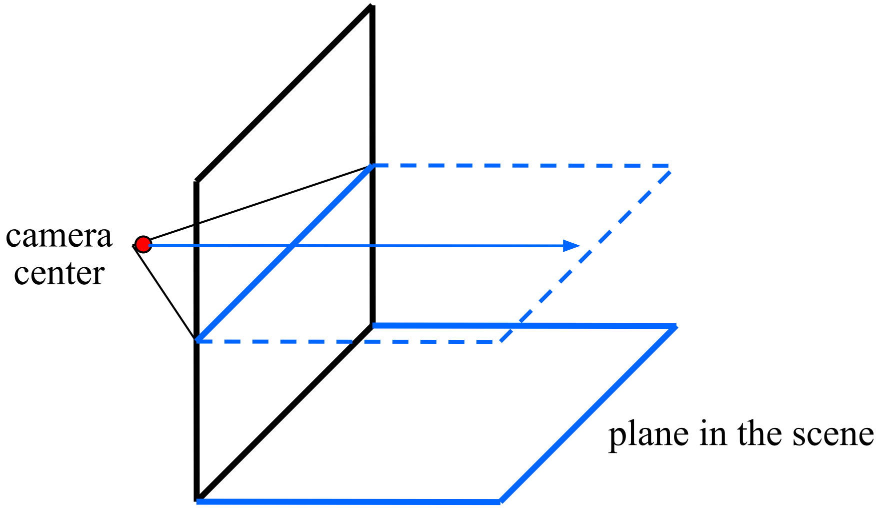


Figure source: S. Seitz

Vanishing lines of planes

Horizon: vanishing line of the ground plane

Q. What can the horizon tell us about the relative height pixels with respect to the camera?

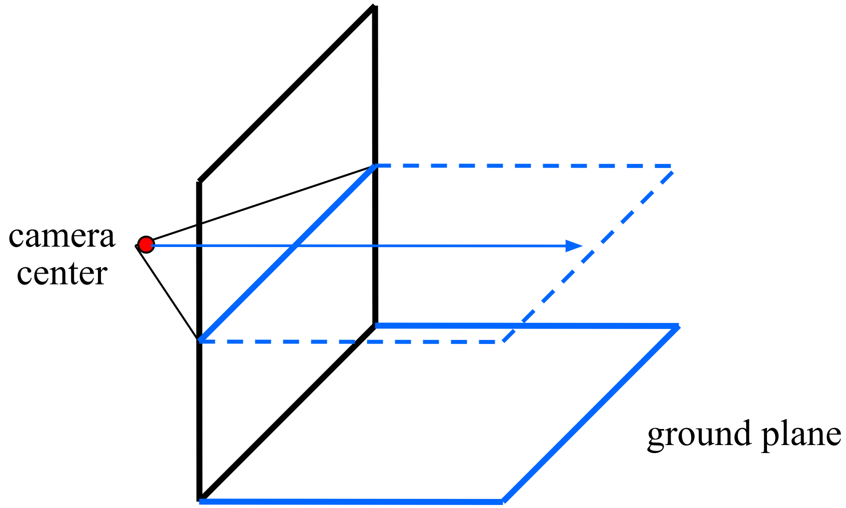


Image source: S. Seitz

Q. Are these assertions true or false?

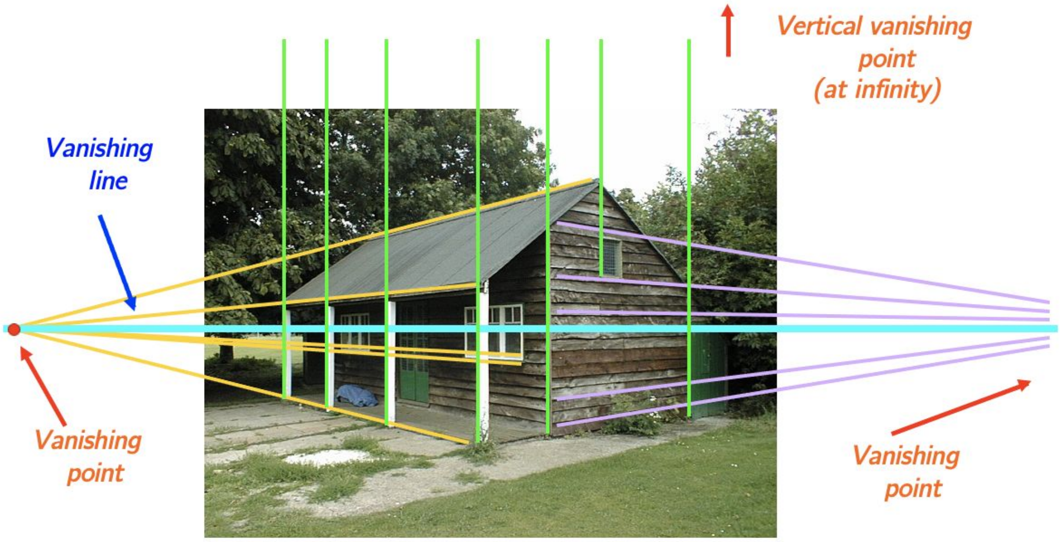
1. All lines that intersect in 2D are parallel in 3D
2. 2D lines always intersect each other in P^2
3. 3D planes always intersect each other in P^3
4. 3D lines always intersect each other in P^3
5. 3D lines always intersect each other in P^2
6. The projection of parallel lines in 3D meet at the same vanishing point
7. The projection of non-intersecting lines in 3D meet at the same vanishing point
8. If a set of parallel 3D lines are also parallel to a particular plane, their vanishing point will lie on the vanishing line of the plane

Q. Are these assertions true or false?

1. All lines that intersect in 2D are parallel in 3D → NO!
2. 2D lines always intersect each other in P^2 → YES!
3. 3D planes always intersect each other in P^3 → YES!
4. 3D lines always intersect each other in P^3 → NO!
5. 3D lines always intersect each other in P^2 → YES!
6. The projection of parallel lines in 3D meet at the same vanishing point → YES!
7. The projection of non-intersecting lines in 3D meet at the same vanishing point → NO!
8. If a set of parallel 3D lines are also parallel to a particular plane, their vanishing point will lie on the vanishing line of the plane → YES!

Properties of Vanishing Points & Lines

- The projections of parallel 3D lines intersect at a vanishing point
- The projection of parallel 3D planes intersect at a vanishing line
- Vanishing point <-> 3D direction of a line
- Vanishing line <-> 3D orientation of a surface



Vanishing Points are a key perspective tool:

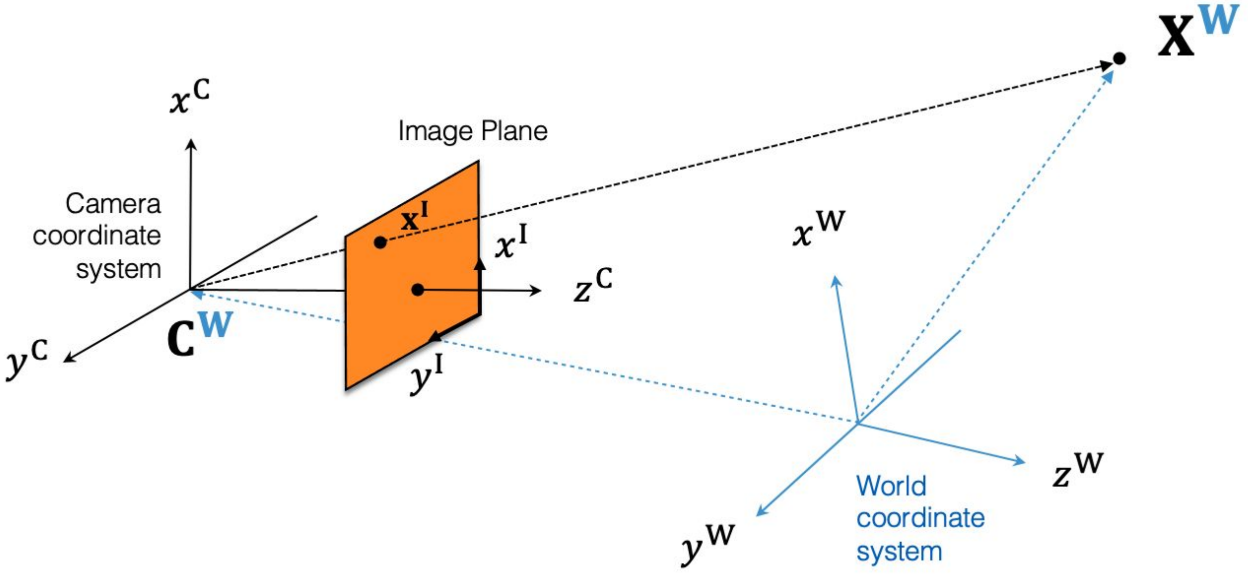
from making realistic drawings, to measuring

3D from 2D, and even for camera calibration!

Today's agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods

Recap: The Pinhole Camera Model



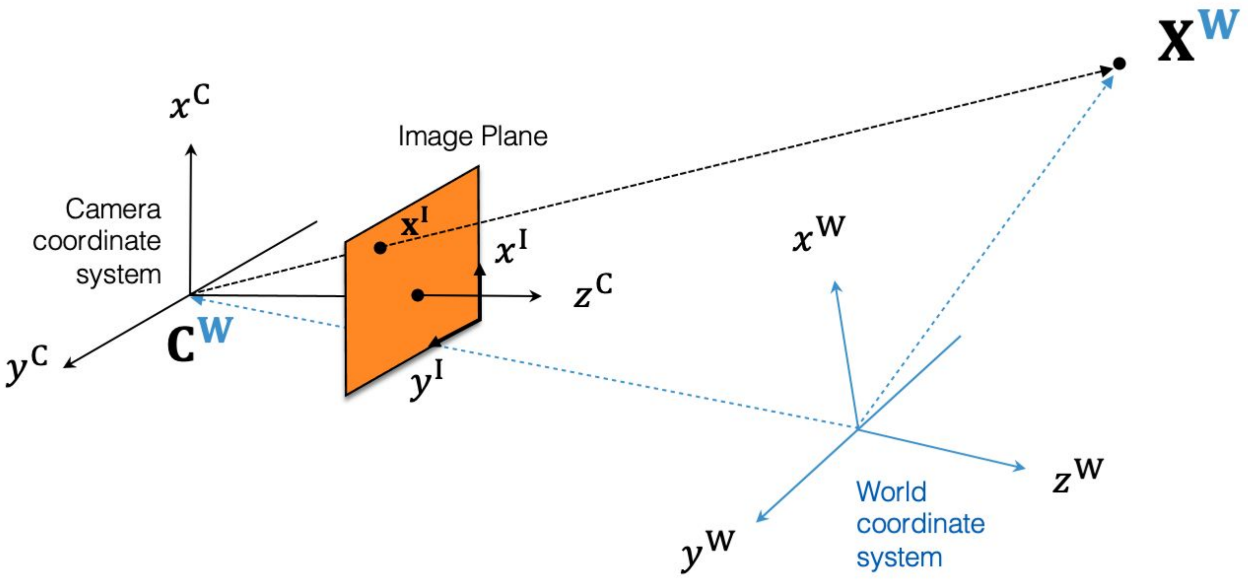
$$\tilde{\mathbf{x}}^{\mathbf{I}} \sim \mathbf{P}\tilde{\mathbf{X}}^{\mathbf{W}} \qquad \mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{I} \mid \mathbf{0}] \begin{bmatrix} \mathbf{R} & -\mathbf{RC} \\ \mathbf{0} & 1 \end{bmatrix} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$

intrinsic parameters **K** (3 x 3): correspond to camera internals (image-to-image transformation)

perspective projection (3 x 4): maps 3D to 2D points (camera-to-image transformation)

extrinsic parameters (4 x 4): correspond to camera externals (world-to-camera transformation)

Camera Calibration & Pose Estimation



$$\tilde{\mathbf{x}}^{\mathrm{I}} \sim \mathbf{P}\widetilde{\mathbf{X}}^{\mathrm{W}} \qquad \mathbf{P} = \mathbf{K}[\mathbf{R}\,|\mathrm{t}]$$

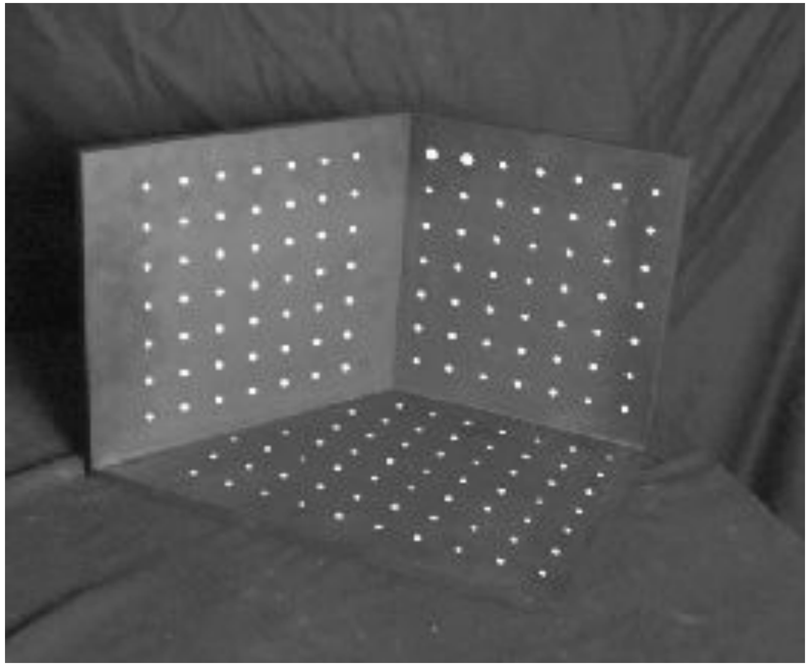
How can we estimate $\mathbf{P}$ and its components?

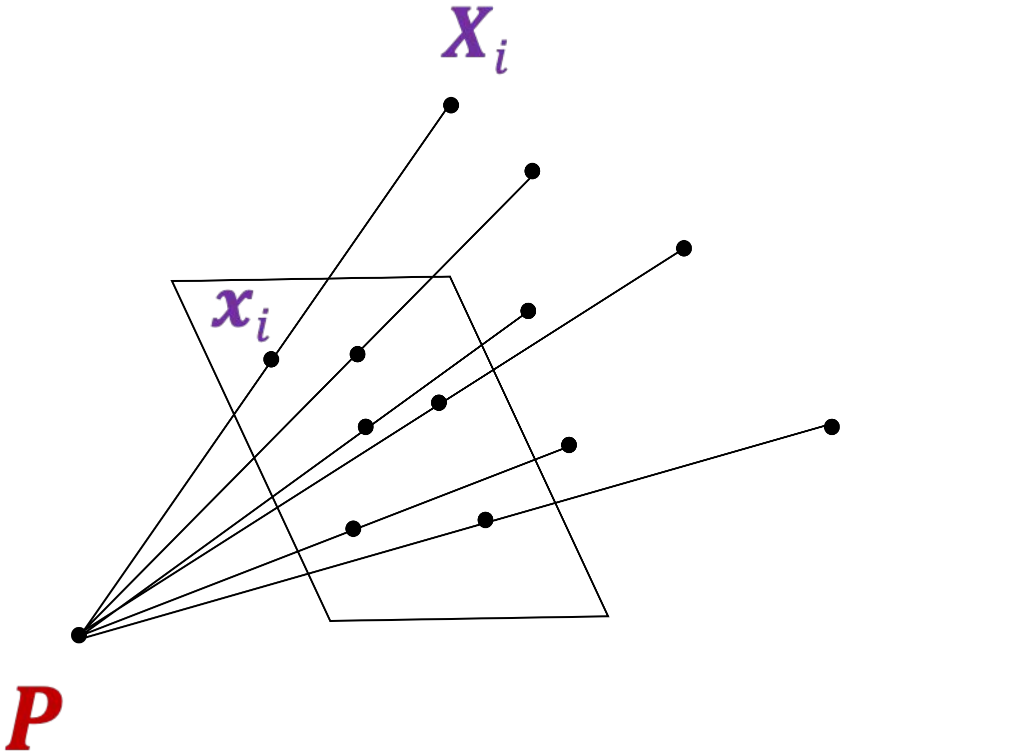
Camera Calibration: estimating *intrinsics* **K**

Pose Estimation: estimating *extrinsics* **[R | t]**

Camera calibration from 3D-2D correspondences

Given n points with ***known*** 3D coordinates $\boldsymbol{X}_i$ and known image projections $\boldsymbol{x}_i$, estimate the camera parameters $\mathbf{P}$ such that $\tilde{\mathbf{x}}_i^{\mathrm{I}} \sim \mathbf{P}\tilde{\mathbf{X}}_i^{\mathrm{W}}$





Credit: S. Lazebnik

Camera calibration from 3D-2D correspondences

Given n points with ***known*** 3D coordinates $\boldsymbol{X}_i$ and known image projections $\boldsymbol{x}_i$, estimate the camera parameters $\mathbf{P}$ such that $\tilde{\mathbf{x}}_i^{\mathrm{I}} \sim \mathbf{P}\tilde{\mathbf{X}}_i^{\mathrm{W}}$

Known 2D image coords	Known 3D locations
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
783 521	308.253 306.300 28.881
235 427	306.650 309.301 28.905
665 429	308.069 306.831 29.189
655 362	309.671 308.834 29.029
427 333	308.255 309.955 29.267
412 415	307.546 308.613 28.963
746 351	311.036 309.206 28.913
434 415	307.518 308.175 29.069
525 234	309.950 311.262 29.990
716 308	312.160 310.772 29.080
602 187	311.988 312.709 30.514

Camera calibration from 3D-2D correspondences

Given n points with ***known*** 3D coordinates $\boldsymbol{X}_i$ and known image projections $\boldsymbol{x}_i$, estimate the camera parameters $\mathbf{P}$ such that $\tilde{\mathbf{x}}_i^{\mathrm{I}} \sim \mathbf{P}\tilde{\mathbf{X}}_i^{\mathrm{W}}$

Many good solutions for accurate 3D position from good fiducial markers: ArUco, AprilTags, …

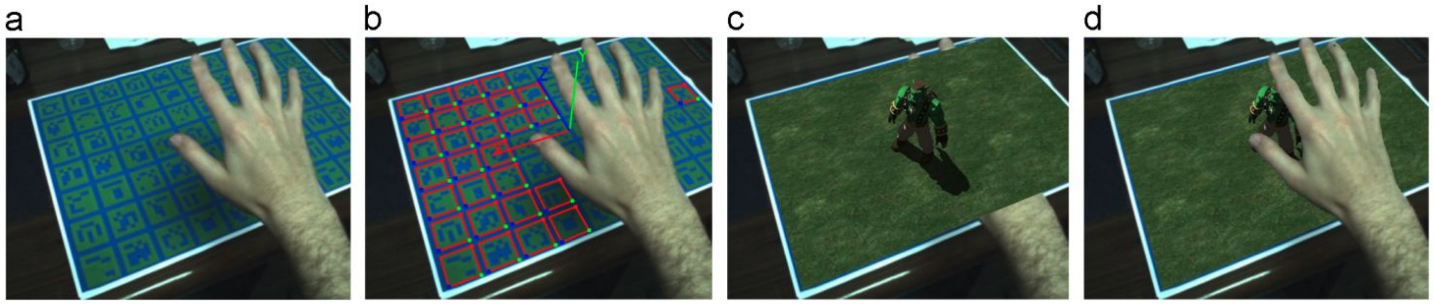


Fig. 1. Example of augmented reality scene. (a) Input image containing a set of fiducial markers. (b) Markers automatically detected and used for camera pose estimation. (c) Augmented scene without considering user's occlusion. (d) Augmented scene considering occlusion.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., & Marín-Jiménez, M. J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*

https://april.eecs.umich.edu/software/apriltag

Mapping between 3D point and image points

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

What are the knowns and unknowns?

Mapping between 3D point and image points

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} — & \boldsymbol{p}_1^\top & — \\ — & \boldsymbol{p}_2^\top & — \\ — & \boldsymbol{p}_3^\top & — \end{bmatrix} \begin{bmatrix} | \\ \boldsymbol{X} \\ | \end{bmatrix}$$

Heterogeneous coordinates

$$x' = \frac{\boldsymbol{p}_1^\top \boldsymbol{X}}{\boldsymbol{p}_3^\top \boldsymbol{X}} \qquad y' = \frac{\boldsymbol{p}_2^\top \boldsymbol{X}}{\boldsymbol{p}_3^\top \boldsymbol{X}}$$

(non-linear relationship between coordinates)

How can we make these relations linear?

Today's agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods

How can we make these relations linear?

$$x' = \frac{\boldsymbol{p}_1^\top \boldsymbol{X}}{\boldsymbol{p}_3^\top \boldsymbol{X}} \qquad y' = \frac{\boldsymbol{p}_2^\top \boldsymbol{X}}{\boldsymbol{p}_3^\top \boldsymbol{X}}$$

Make them linear with algebraic manipulation…

$$\boldsymbol{p}_1^\top \boldsymbol{X} - \boldsymbol{p}_3^\top \boldsymbol{X} x' = 0 \qquad \boldsymbol{p}_2^\top \boldsymbol{X} - \boldsymbol{p}_3^\top \boldsymbol{X} y' = 0$$

Now we can setup a system of linear equations with multiple corresponding points

Camera Calibration: Linear Method

$$\boldsymbol{p}_i \equiv \boldsymbol{M}\boldsymbol{X}_i$$

Remember (from geometry): this implies $\mathbf{MX_i}$ **&** $\mathbf{p_i}$ are proportional/scaled copies of each other

$$\boldsymbol{p}_i = \lambda \boldsymbol{M}\boldsymbol{X}_i, \lambda \neq 0$$

Remember (from homography fitting): this implies their cross product is **0**

$$\boldsymbol{p}_i \times \boldsymbol{M}\boldsymbol{X}_i = \boldsymbol{0}$$

Slide credit: D. Fouhey & J. Johnson

$$\boldsymbol{p}_1^\top \boldsymbol{X} - \boldsymbol{p}_3^\top \boldsymbol{X} x' = 0 \qquad \boldsymbol{p}_2^\top \boldsymbol{X} - \boldsymbol{p}_3^\top \boldsymbol{X} y' = 0$$

In matrix form …
$$\begin{bmatrix} \boldsymbol{X}^\top & \boldsymbol{0} & -x'\boldsymbol{X}^\top \\ \boldsymbol{0} & \boldsymbol{X}^\top & -y'\boldsymbol{X}^\top \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{bmatrix} = \boldsymbol{0}$$

Q1. How many equations does each correspondence give us?

Q2. How many correspondences do we need to solve for P?

$$\boldsymbol{p}_1^\top \boldsymbol{X} - \boldsymbol{p}_3^\top \boldsymbol{X} x' = 0 \qquad \boldsymbol{p}_2^\top \boldsymbol{X} - \boldsymbol{p}_3^\top \boldsymbol{X} y' = 0$$

In matrix form for 1 corresponding point …

$$\begin{bmatrix} \boldsymbol{X}^\top & \boldsymbol{0} & -x'\boldsymbol{X}^\top \\ \boldsymbol{0} & \boldsymbol{X}^\top & -y'\boldsymbol{X}^\top \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{bmatrix} = \boldsymbol{0}$$

For N points …

$$\begin{bmatrix} \boldsymbol{X}_1^T & 0 & -x_1'\boldsymbol{X}_1^T \\ \boldsymbol{0} & \boldsymbol{X}_1^T & -y_1'\boldsymbol{X}_1^T \\ \vdots & \vdots & \vdots \\ \boldsymbol{X}_N^T & 0 & -x_N'\boldsymbol{X}_N^T \\ \boldsymbol{0} & \boldsymbol{X}_N^T & -y_N'\boldsymbol{X}_N^T \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{bmatrix} = 0$$

How do we solve this system?

A few things to look out for

- N points should not be co-planar. Otherwise, the rows will not be independent
- Usually any measurements you make in 3D space will be noisy. So you need more than the minimum number of points!
- P has 12 values but we only need to find 11 since everything is scaled

For N points …

$$\begin{bmatrix} \boldsymbol{X}_1^T & 0 & -x_1'\boldsymbol{X}_1^T \\ \boldsymbol{0} & \boldsymbol{X}_1^T & -y_1'\boldsymbol{X}_1^T \\ \vdots & \vdots & \vdots \\ \boldsymbol{X}_N^T & 0 & -x_N'\boldsymbol{X}_N^T \\ \boldsymbol{0} & \boldsymbol{X}_N^T & -y_N'\boldsymbol{X}_N^T \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{bmatrix} = 0$$

How do we solve this system?

Solve for camera matrix via total least squares

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} ||\mathbf{A}\boldsymbol{x}||^2 \text{ subject to } ||\boldsymbol{x}||^2 = 1$$

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{X}_1^T & 0 & -x_1'\boldsymbol{X}_1^T \\ \mathbf{0} & \boldsymbol{X}_1^T & -y_1'\boldsymbol{X}_1^T \\ \vdots & \vdots & \vdots \\ \boldsymbol{X}_N^T & 0 & -x_N'\boldsymbol{X}_N^T \\ \mathbf{0} & \boldsymbol{X}_N^T & -y_N'\boldsymbol{X}_N^T \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{bmatrix}$$

Singular Value Decomposition!

Singular Value Decomposition (SVD)

- Represents any matrix **A** as a product of three matrices: $\mathbf{U\Sigma V^T}$
- Python command:
  - [U,**Σ**,V]= numpy.linalg.svd(A)

$$\overset{U}{\begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix}} \times \overset{\Sigma}{\begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix}} = \overset{A}{\begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}}$$

Singular Value Decomposition (SVD)

- Beyond 2x2 matrices:
  - In general, if **A** is *m* x *n*, then **U** will be *m* x *m*, **Σ** will be *m* x *n*, and **V**^{T} will be *n* x *n*.
  - (Note the dimensions work out to produce *m* x *n* after multiplication)

$$\underset{U}{\begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix}} \times \underset{\Sigma}{\begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix}} = \underset{A}{\begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}}$$

Singular Value Decomposition (SVD)

- **U** and **V** are always **rotation** matrices.
  - Each column is a unit vector.
- **Σ** is a diagonal matrix
  - The number of nonzero entries = rank of **A**
  - The algorithm always sorts the entries high to low

$$\overset{U}{\begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix}} \times \overset{\Sigma}{\begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix}} = \overset{A}{\begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}}$$

Solve for camera matrix via total least squares

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} ||\mathbf{A}\boldsymbol{x}||^2 \text{ subject to } ||\boldsymbol{x}||^2 = 1$$

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{X}_1^T & 0 & -x_1'\boldsymbol{X}_1^T \\ \mathbf{0} & \boldsymbol{X}_1^T & -y_1'\boldsymbol{X}_1^T \\ \vdots & \vdots & \vdots \\ \boldsymbol{X}_N^T & 0 & -x_N'\boldsymbol{X}_N^T \\ \mathbf{0} & \boldsymbol{X}_N^T & -y_N'\boldsymbol{X}_N^T \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{bmatrix}$$

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

Solution **x** is the column of **V** corresponding to the smallest singular value of A

Why is it the column of V with the smallest eigenvalue?

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} ||\mathbf{A}\boldsymbol{x}||^2 \text{ subject to } ||\boldsymbol{x}||^2 = 1$$

Is equivalent to:

$$\mathbf{x} = \arg\min_{\mathbf{x}} \mathbf{x}^T(\mathbf{A}^T\mathbf{A})\mathbf{x} \qquad \text{such that} \qquad \|\mathbf{x}\|^2 = 1.$$

Why is it the column of V with the smallest eigenvalue?

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} ||\mathbf{A}\boldsymbol{x}||^2 \text{ subject to } ||\boldsymbol{x}||^2 = 1$$

Is equivalent to:

$$\mathbf{x} = \arg\min_{\mathbf{x}} \mathbf{x}^T(\mathbf{A}^T\mathbf{A})\mathbf{x} \qquad \text{such that} \qquad \|\mathbf{x}\|^2 = 1.$$

$$\begin{aligned} A^TA &= (U\Sigma V)^T(U\Sigma V) \\ &= (V^T\Sigma U^T)(U\Sigma V) \\ &= V^T\Sigma(U^TU)\Sigma V \\ &= V^T\Sigma I\Sigma V \\ &= V^T\Sigma^2V \end{aligned}$$

Why is it the column of V with the smallest eigenvalue?

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} ||\mathbf{A}\boldsymbol{x}||^2 \text{ subject to } ||\boldsymbol{x}||^2 = 1$$

Is equivalent to:

$$\mathbf{x} = \arg\min_{\mathbf{x}} \mathbf{x}^T(\mathbf{A}^T\mathbf{A})\mathbf{x} \qquad \text{such that} \qquad \|\mathbf{x}\|^2 = 1.$$

$A^TA = (U\Sigma V)^T(U\Sigma V)$
$= (V^T\Sigma U^T)(U\Sigma V)$
$= V^T\Sigma(U^TU)\Sigma V$
$= V^T\Sigma I\Sigma V$
$= V^T\Sigma^2 V$

$$\mathbf{A}^T\mathbf{A}\mathbf{v}_k = \sigma_k^2\mathbf{v}_k,$$

So, A^TAv_k is proportional to the vector's eigenvalue. So the vector corresponding to the smallest eigenvalue is what we want

Solve for camera matrix via total least squares

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} ||\mathbf{A}\boldsymbol{x}||^2 \text{ subject to } ||\boldsymbol{x}||^2 = 1$$

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{X}_1^T & 0 & -x_1'\boldsymbol{X}_1^T \\ \mathbf{0} & \boldsymbol{X}_1^T & -y_1'\boldsymbol{X}_1^T \\ \vdots & \vdots & \vdots \\ \boldsymbol{X}_N^T & 0 & -x_N'\boldsymbol{X}_N^T \\ \mathbf{0} & \boldsymbol{X}_N^T & -y_N'\boldsymbol{X}_N^T \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{bmatrix}$$

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$$

Equivalently, solution ***x*** is the Eigenvector corresponding to the smallest Eigenvalue of **A**

A.2.1 Total least squares

In some problems, e.g., when performing geometric line fitting in 2D images or 3D plane fitting to point cloud data, instead of having measurement error along one particular axis, the measured points have uncertainty in all directions, which is known as the *errors-in-variables* model (Van Huffel and Lemmerling 2002; Matei and Meer 2006). In this case, it makes more sense to minimize a set of homogeneous squared errors of the form

$$E_{\mathrm{TLS}} = \sum_i (\mathbf{a}_i \mathbf{x})^2 = \|\mathbf{A}\mathbf{x}\|^2, \tag{A.35}$$

which is known as *total least squares* (TLS) (Van Huffel and Vandewalle 1991; Björck 1996; Golub and Van Loan 1996; Van Huffel and Lemmerling 2002).

The above error metric has a trivial minimum solution at $\mathbf{x} = 0$ and is, in fact, homogeneous in $\mathbf{x}$. For this reason, we augment this minimization problem with the requirement that $\|\mathbf{x}\|^2 = 1$. which results in the eigenvalue problem

$$\mathbf{x} = \arg\min_{\mathbf{x}} \mathbf{x}^T (\mathbf{A}^T \mathbf{A}) \mathbf{x} \qquad \text{such that} \qquad \|\mathbf{x}\|^2 = 1. \tag{A.36}$$

The value of $\mathbf{x}$ that minimizes this constrained problem is the eigenvector associated with the smallest eigenvalue of $\mathbf{A}^T\mathbf{A}$. This is the same as the last right singular vector of $\mathbf{A}$, because

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T, \tag{A.37}$$

$$\mathbf{A}^T\mathbf{A} = \mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}^T, \tag{A.38}$$

$$\mathbf{A}^T\mathbf{A}\mathbf{v}_k = \sigma_k^2 \mathbf{v}_k, \tag{A.39}$$

which is minimized by selecting the smallest σ_k value.

Szeliski A.2.1

We calculated the camera matrix!

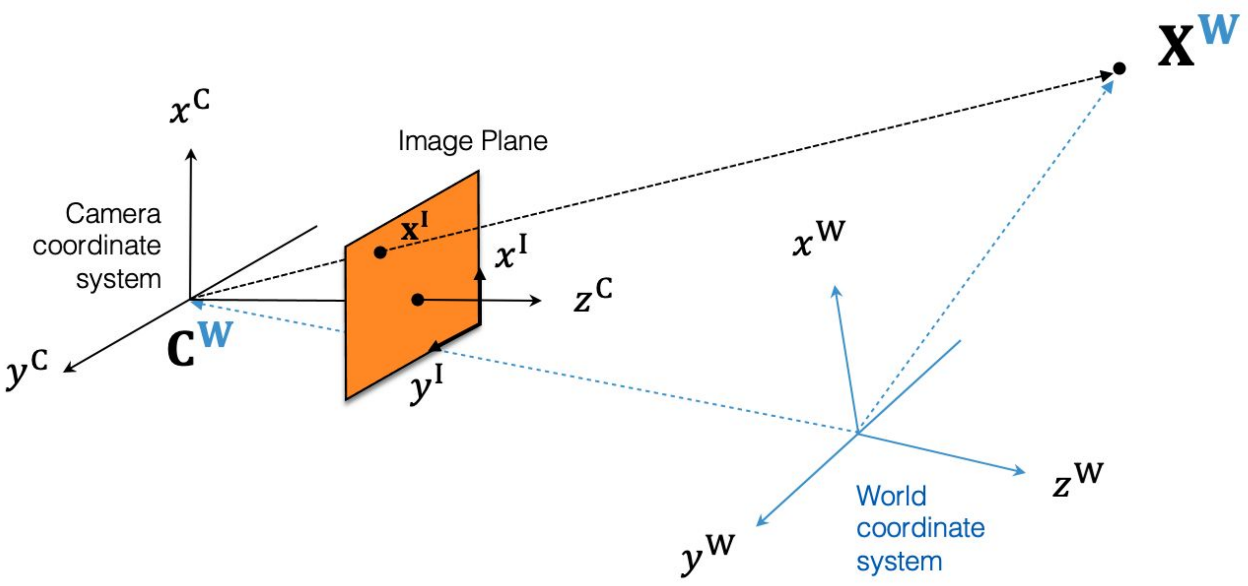
Now we have:

$$\mathbf{P} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix}$$

Today’s agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods

Recap: The Pinhole Camera Model



$$\tilde{\mathbf{x}}^{\mathrm{I}} \sim \mathbf{P}\tilde{\mathbf{X}}^{\mathrm{W}} \qquad \mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{I} \mid \mathbf{0}] \begin{bmatrix} \mathbf{R} & -\mathbf{RC} \\ \mathbf{0} & 1 \end{bmatrix} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$

intrinsic parameters **K** (3 x 3): correspond to camera internals (image-to-image transformation)

perspective projection (3 x 4): maps 3D to 2D points (camera-to-image transformation)

extrinsic parameters (4 x 4): correspond to camera externals (world-to-camera transformation)

Almost there … $$\mathbf{P} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix}$$

Want to get $$\mathbf{P} = \mathbf{K}\,[\mathbf{R} \mid -\mathbf{RC}]$$

How can we calculate the intrinsic and extrinsic parameters from the projection matrix?

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array}\right] \sim \mathbf{K}\,[\mathbf{R} \mid -\mathbf{RC}]$$

Q1. Is there a way we can get rid of **C**?

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array}\right] \sim \mathbf{K}\,[\mathbf{R} \mid -\mathbf{RC}]$$

$$\overline{\mathbf{P}} = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_5 & p_6 & p_7 \\ p_9 & p_{10} & p_{11} \end{bmatrix} \sim \mathbf{KR}$$

Q2. Is there a way we can get rid of **R**?

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array}\right] \sim \mathbf{K}\,[\mathbf{R} \mid -\mathbf{RC}]$$

$$\overline{\mathbf{P}} = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_5 & p_6 & p_7 \\ p_9 & p_{10} & p_{11} \end{bmatrix} \sim \mathbf{KR}$$

$$\overline{\mathbf{P}}\,\overline{\mathbf{P}}^\top \sim \mathbf{KRR}^\top\mathbf{K}^\top$$

$$\overline{\mathbf{P}}\,\overline{\mathbf{P}}^\top \sim \mathbf{KK}^\top \qquad (\mathbf{RR}^\top = \mathbf{I})$$

Q. Do you remember a **property of K** that could help us solve this?

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array}\right] \sim \mathbf{K}\,[\mathbf{R} \mid -\mathbf{RC}]$$

$$\overline{\mathbf{P}} = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_5 & p_6 & p_7 \\ p_9 & p_{10} & p_{11} \end{bmatrix} \sim \mathbf{KR}$$

$$\overline{\mathbf{P}}\overline{\mathbf{P}}^\top \sim \mathbf{KRR}^\top\mathbf{K}^\top$$

$$\overline{\mathbf{P}}\overline{\mathbf{P}}^\top \sim \mathbf{KK}^\top \qquad (\mathbf{RR}^\top = \mathbf{I})$$

K is upper triangular and positive definite!

$$\boxed{\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}}$$

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{c|c} \overline{\mathbf{P}} & \begin{matrix} p_4 \\ p_8 \\ p_{12} \end{matrix} \end{array} \right] \sim \mathbf{K}\,[\mathbf{R} \mid -\mathbf{RC}]$$

$\overline{\mathbf{P}}^\top \overline{\mathbf{P}} \sim \mathbf{K}^\top \mathbf{K}$ with $\mathbf{K}$ upper triangular p.d.

Obtain $\mathbf{K}$ by Cholesky decomposition of $\overline{\mathbf{P}}^\top \overline{\mathbf{P}} = \mathbf{L}\mathbf{L}^\top$

$$\mathbf{K} \sim \mathbf{L}^\top$$

Scalar factor: fixed to $1/\mathrm{L}_{3,3}$ $(\mathrm{K}_{3,3} = 1)$

Once $\mathbf{K}$ is known, we can compute $\mathbf{R} \sim \mathbf{K}^{-1} \overline{\mathbf{P}}$

Everything is calculated up to a scaling factor. So we need to rescale

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{c|c} \quad \overline{\mathbf{P}} \quad & \begin{array}{c} p_4 \\ p_8 \\ p_{12} \end{array} \end{array} \right] \sim \mathbf{K}\,[\mathbf{R} \mid -\mathbf{RC}]$$

$\overline{\mathbf{P}}^\top \overline{\mathbf{P}} \sim \mathbf{K}^\top \mathbf{K}$ with $\mathbf{K}$ upper triangular p.d.

Obtain $\mathbf{K}$ by Cholesky decomposition of $\overline{\mathbf{P}}^\top \overline{\mathbf{P}} = \mathbf{L}\mathbf{L}^\top$

$$\mathbf{K} \sim \mathbf{L}^\top$$

Scalar factor: fixed to $1/\mathrm{L}_{3,3}$ $(\mathrm{K}_{3,3} = 1)$

Once $\mathbf{K}$ is known, we can compute $\mathbf{R} \sim \mathbf{K}^{-1}\overline{\mathbf{P}}$

$\mathbf{R}$ is a rotation matrix: $|\mathbf{R}| = 1$!

Scaling R and calculating C

$$|\mathbf{R}| = 1 \ \Rightarrow \lambda = |\mathbf{K}^{-1}\overline{\mathbf{P}}|^{-1/3}$$

$$\mathbf{R} = \lambda \mathbf{K}^{-1}\overline{\mathbf{P}}$$

Finally, easy to know the camera center: $\mathbf{C} = -\lambda^{-1}\ \mathbf{R}^{\top}\mathbf{K}^{-1}[p_4\ \ p_8\ \ p_{12}]^{\top}$

Linear Camera Calibration

- Advantages:
  - Simple to formulate
  - Analytical solution
- Disadvantages:
  - Doesn’t model radial distortion (non-linear!)
  - Hard to impose constraints (e.g., known *f*)
  - Doesn’t minimize the correct error function: the reprojection error in 2D is what we truly care about!
- Hence why ***non-linear*** methods are preferred in practice.
- They can reuse the linear method we just saw!

Today’s agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods

Non-Linear Camera Calibration

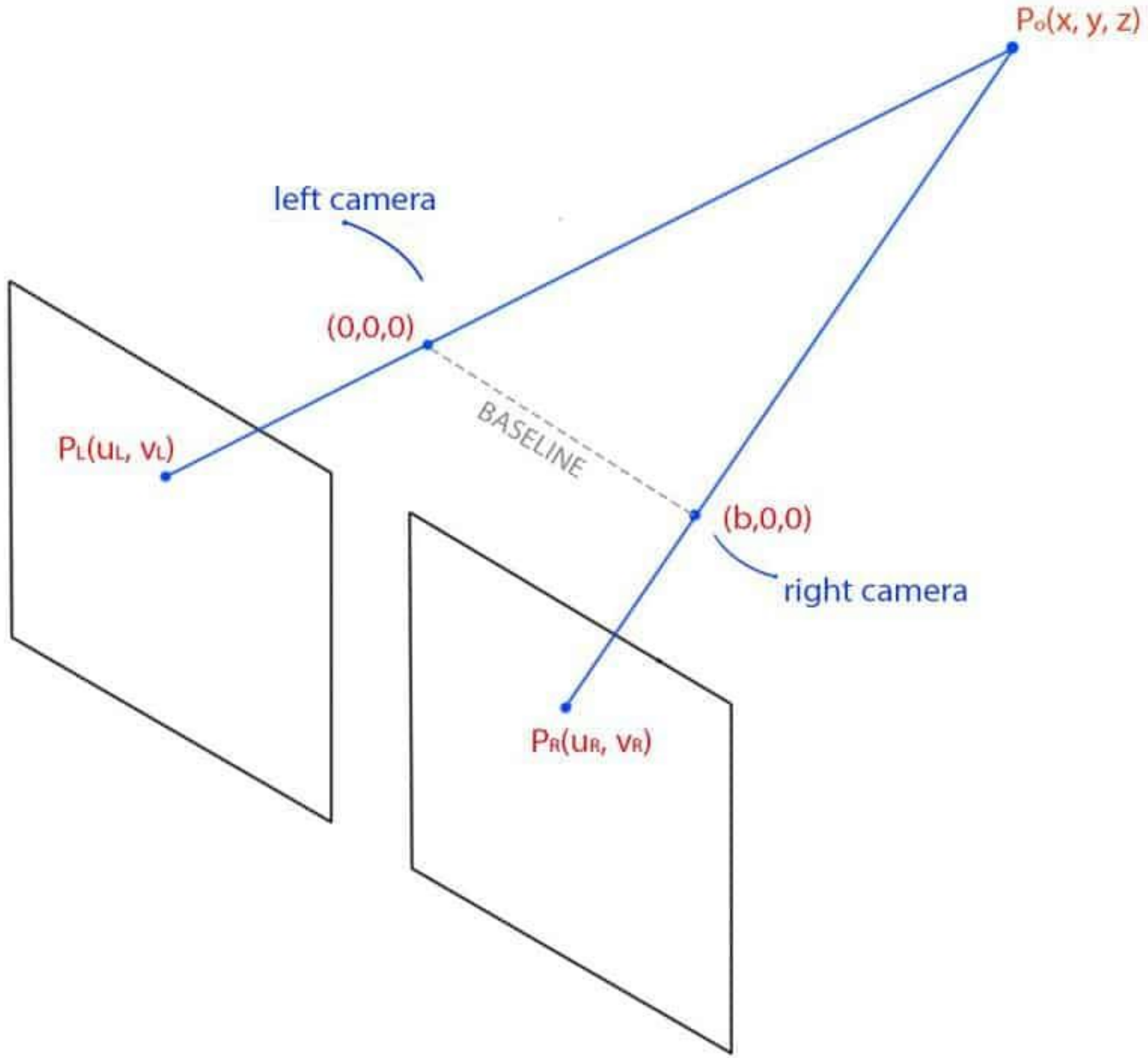
- Write down objective function in terms of intrinsic and extrinsic parameters, as sum of squared distances between ***measured*** 2D points $\boldsymbol{x}_i$ and ***estimated*** projections of corresponding 3D points:

$$\sum_i \|\mathrm{proj}(\mathbf{K}[\mathbf{R} \mid \mathbf{t}]\mathbf{X}_\mathrm{i}; \boldsymbol{\kappa}) - \boldsymbol{x}_i\|_2^2$$

- Can include radial distortion (cf. Szeliski 2.1.5 & 11.1.4) or other parameters $\boldsymbol{\kappa}$ in the projection model (non-linear in the parameters!)
- Can include constraints such as known focal length, orthogonality, visibility of points, or even known $\mathbf{K}$ ("extrinsics calibration")
- Minimize error using standard non-linear optimization techniques (traditionally Levenberg-Marquardt, cf. Szeliski A.3, 8.1.3, 11.1.4)
- Iterative non-linear optimization is sensitive to initialization: use the output of the linear method we just saw!

Estimating depth

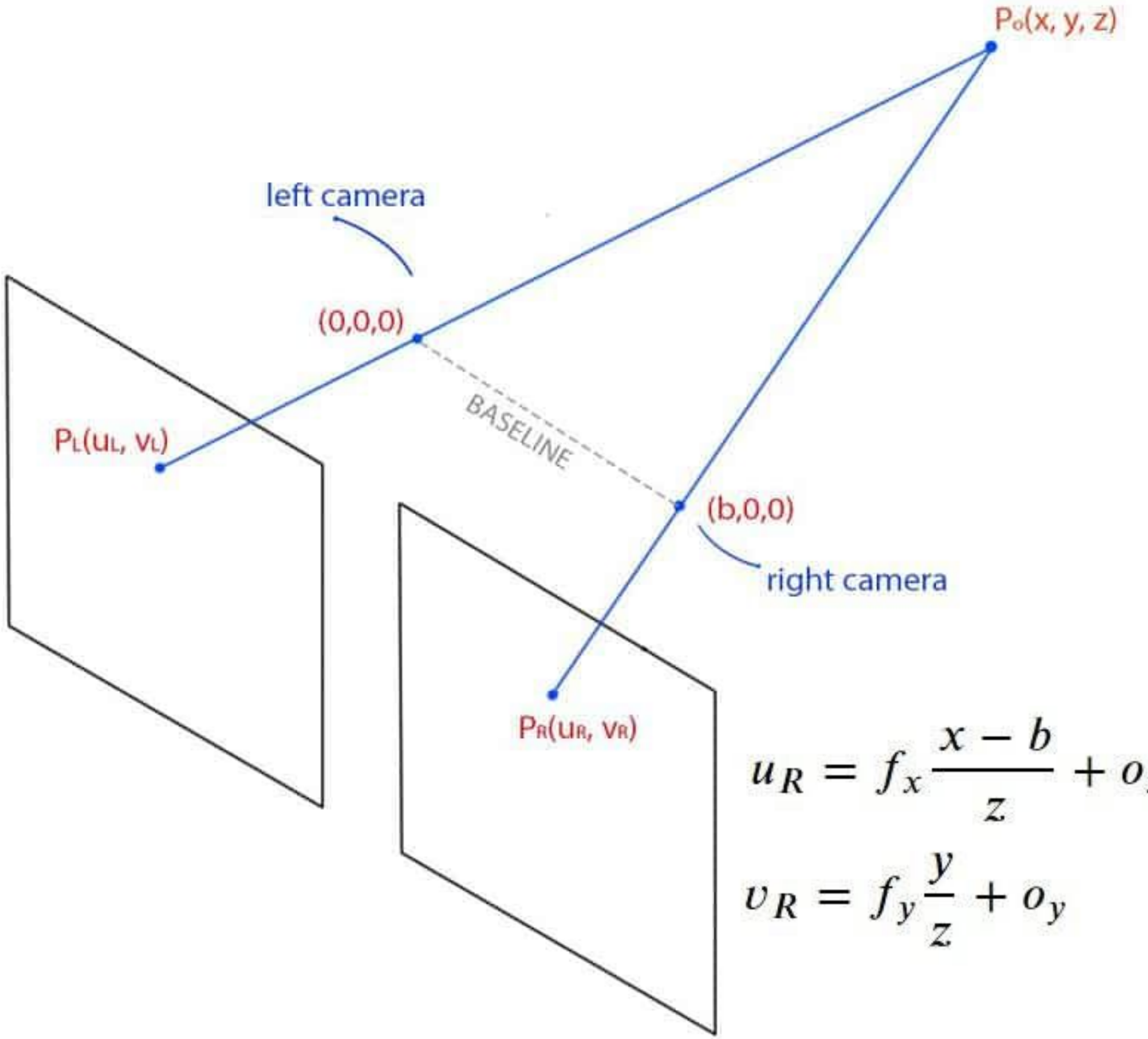
b is the translation from camera at location L and camera at R



Estimating depth

$$u_L = f_x \frac{x}{z} + o_x$$

$$v_L = f_y \frac{y}{z} + o_y$$



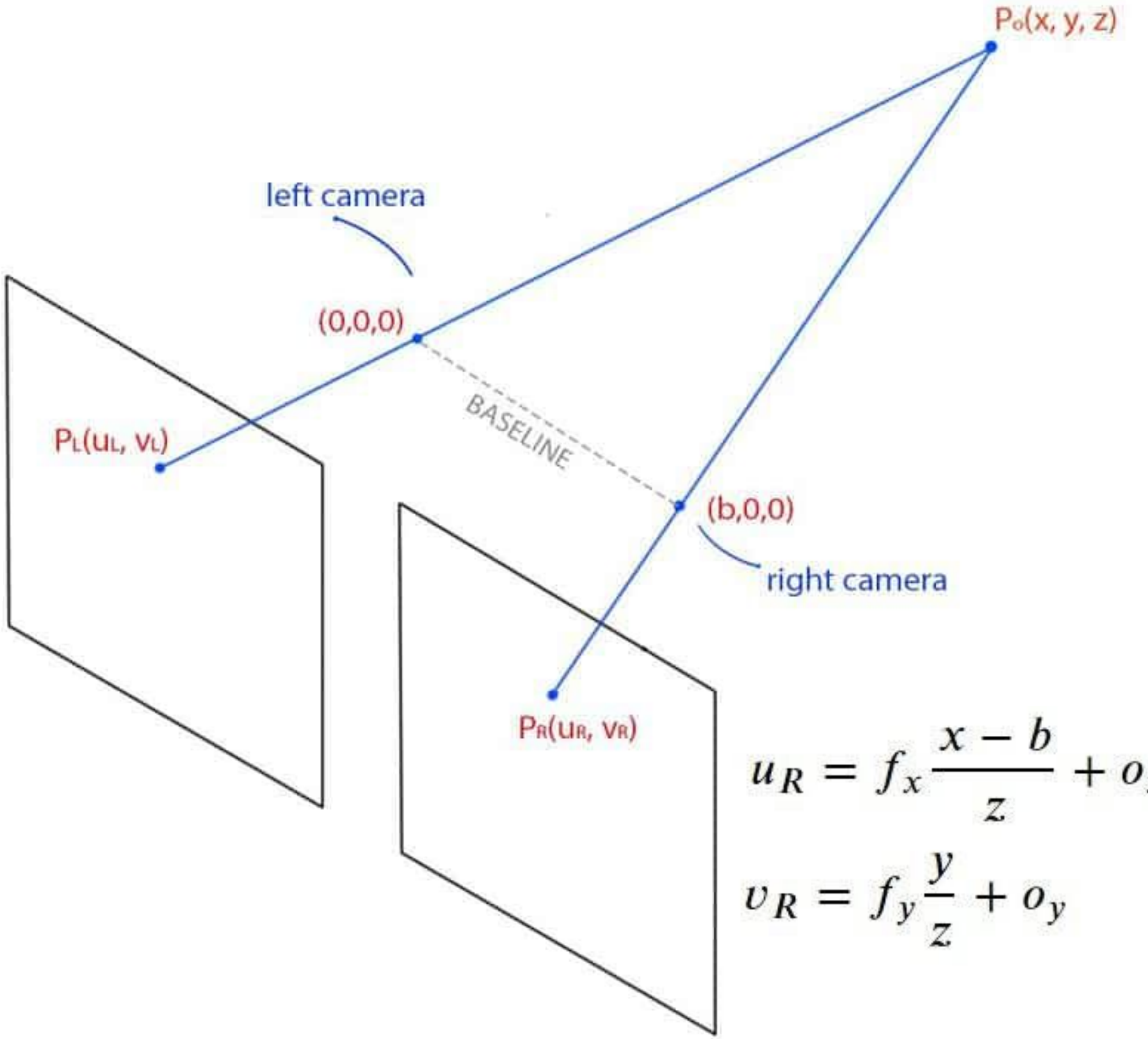
$$u_R = f_x \frac{x - b}{z} + o_x$$

$$v_R = f_y \frac{y}{z} + o_y$$

Estimating depth

$$u_L = f_x \frac{x}{z} + o_x$$

$$v_L = f_y \frac{y}{z} + o_y$$



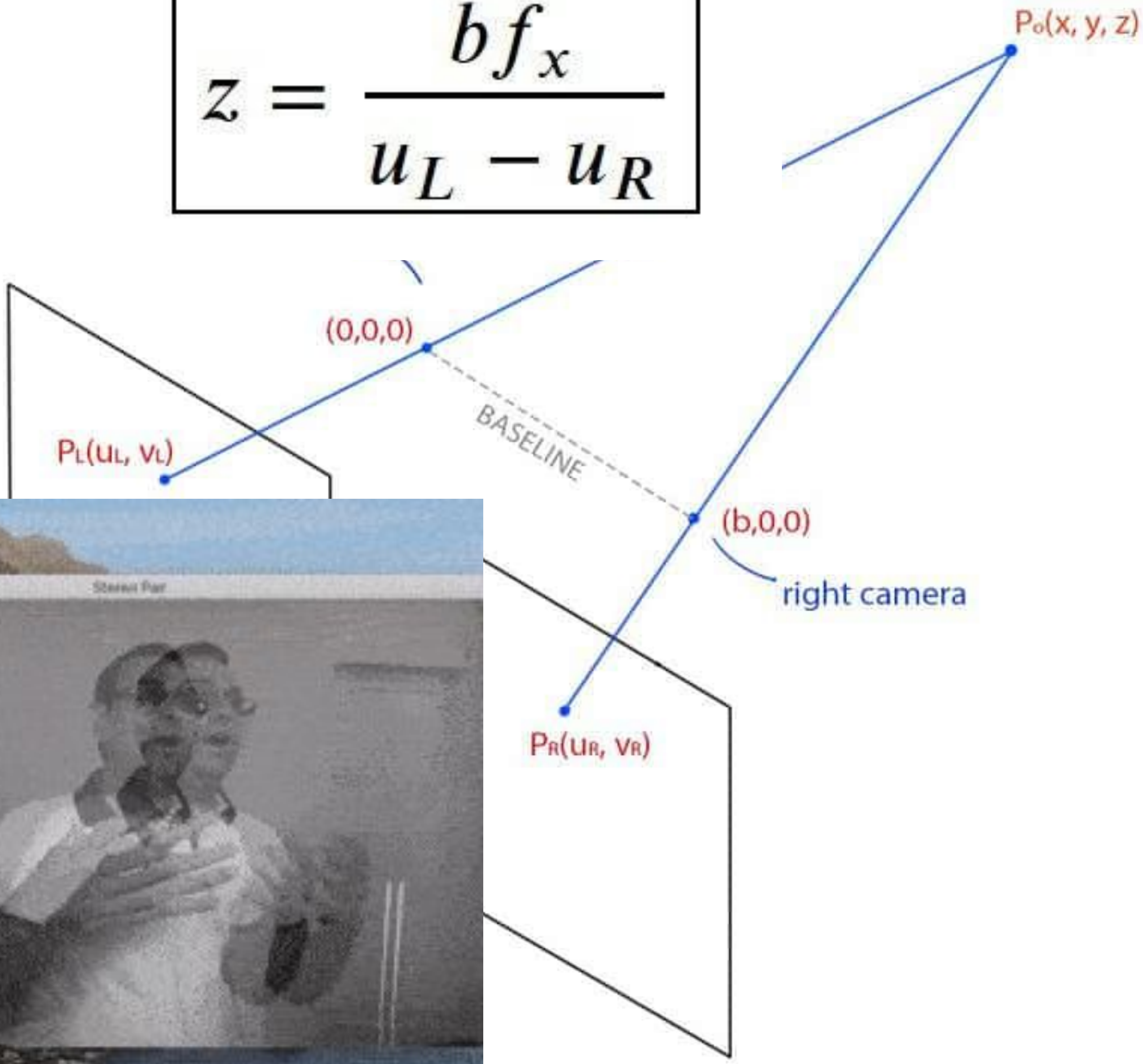
$$u_R = f_x \frac{x - b}{z} + o_x$$

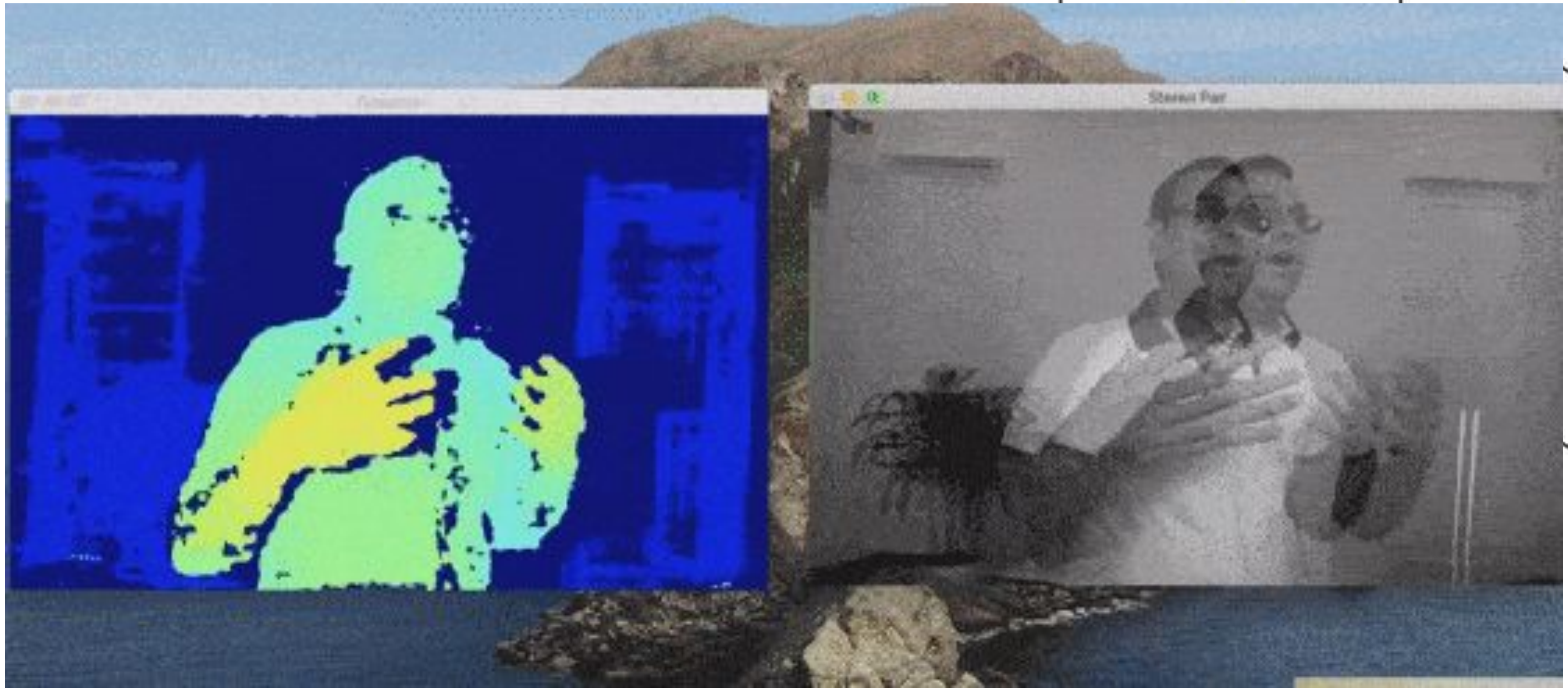
$$v_R = f_y \frac{y}{z} + o_y$$

$$\boxed{z = \frac{b f_x}{u_L - u_R}}$$

Estimating depth

$$z = \frac{b f_x}{u_L - u_R}$$



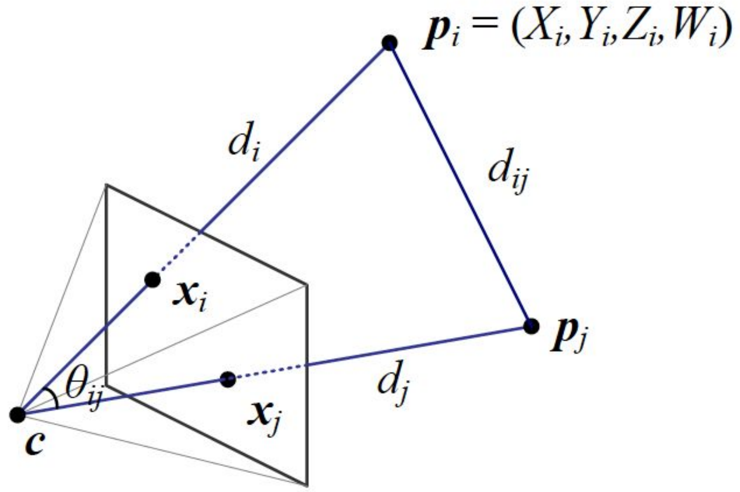


Estimating pose [R|t] (extrinsics)

We have already done it together with estimating K!

What if we know K already?

Estimating [R|t] only = pose estimation, a.k.a. extrinsincs calibration (and previous linear method is called the "Direct Linear Transform")



Estimating pose [R|t] (extrinsics)

Other linear algorithm: PnP (Perspective-n-Point)

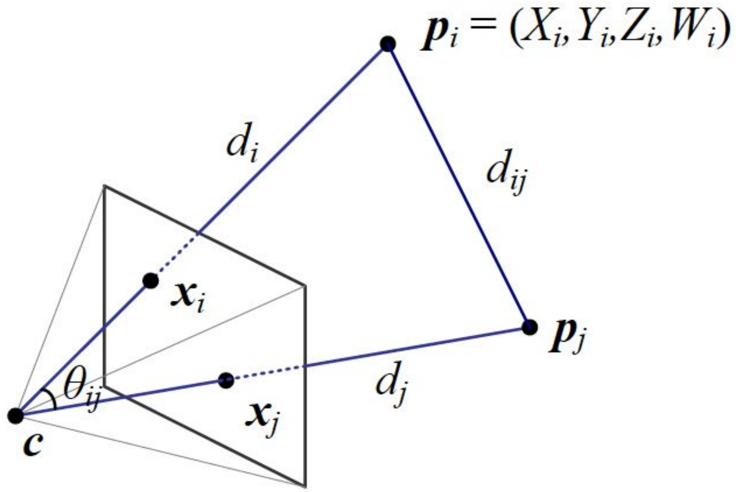
Commonly used solution available in standard libraries like OpenCV

Minimal form: P3P (3 noise-free non-colinear correspondences)

Main idea: same angle between rays of 2 2D points and 2 3D points

In practice: use $n \geq 4$ correspondences + RANSAC

More details:



Quan, Long; Lan, Zhong-Dan (1999). "Linear N-Point Camera Pose Determination" (PDF). *IEEE TPAMI*.
Lepetit, V.; Moreno-Noguer, M.; Fua, P. (2009). "EPnP: An Accurate O(n) Solution to the PnP Problem". *IJCV*

Figure: R. Szeliski

Estimating pose [R|t] (extrinsics)

Non-linear method:

- Minimize reprojection error as function of [R|t]
- More accurate and flexible (e.g., using constraints)
- Can be robustified and easy to implement via transformation decomposition and backpropagation (yes, like in Deep Learning!)

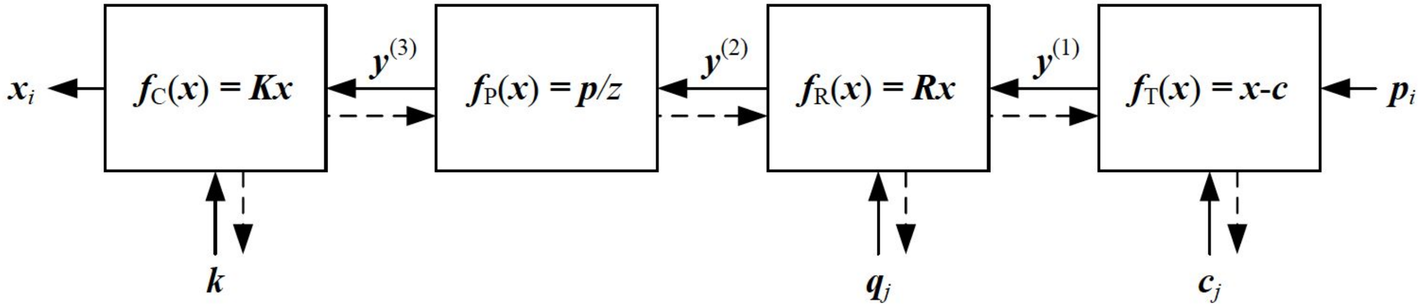


Figure: R. Szeliski

cf. Szeliski 11.2.2 for more details

Today’s agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods

Next lecture

Recognition