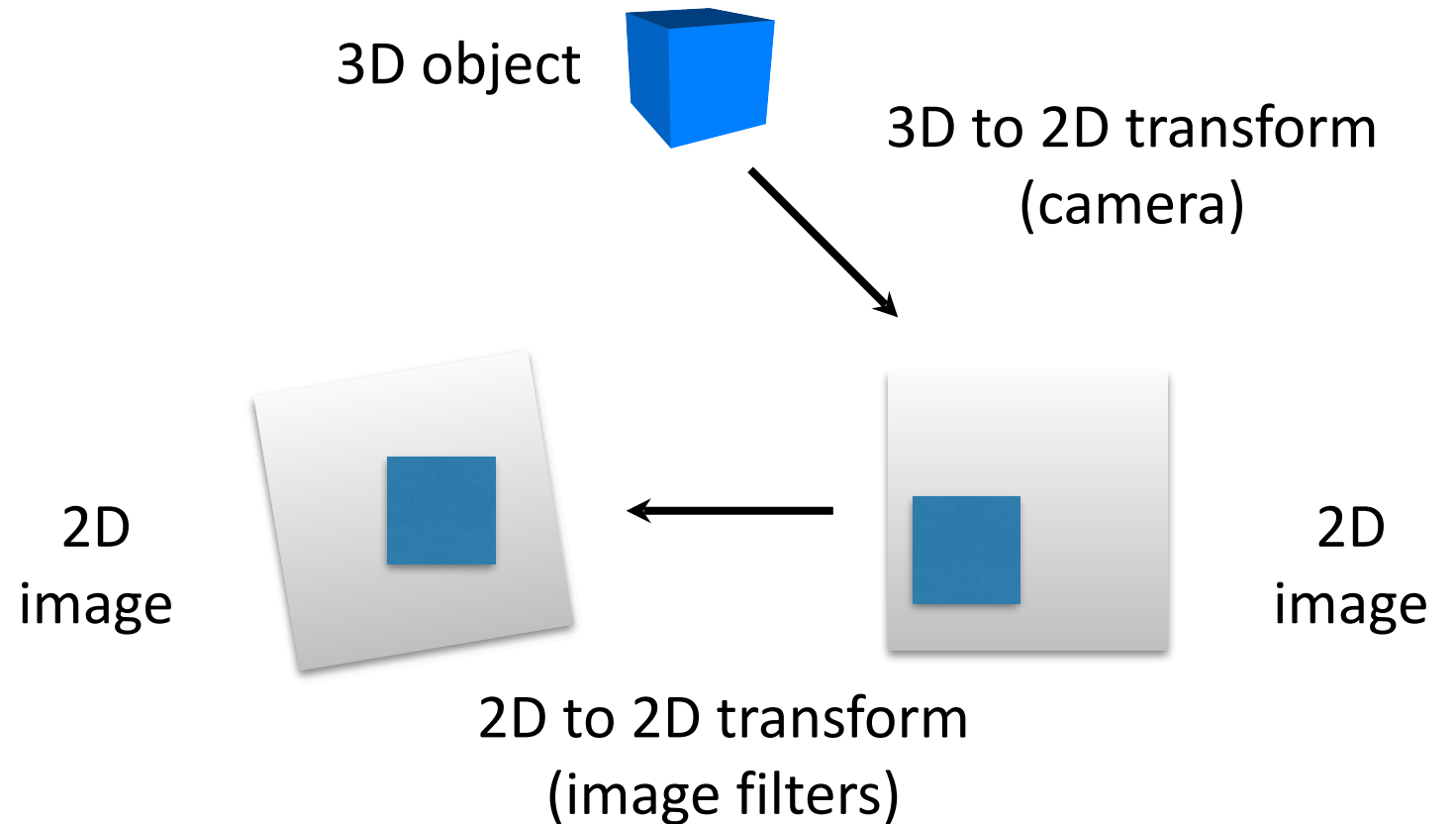


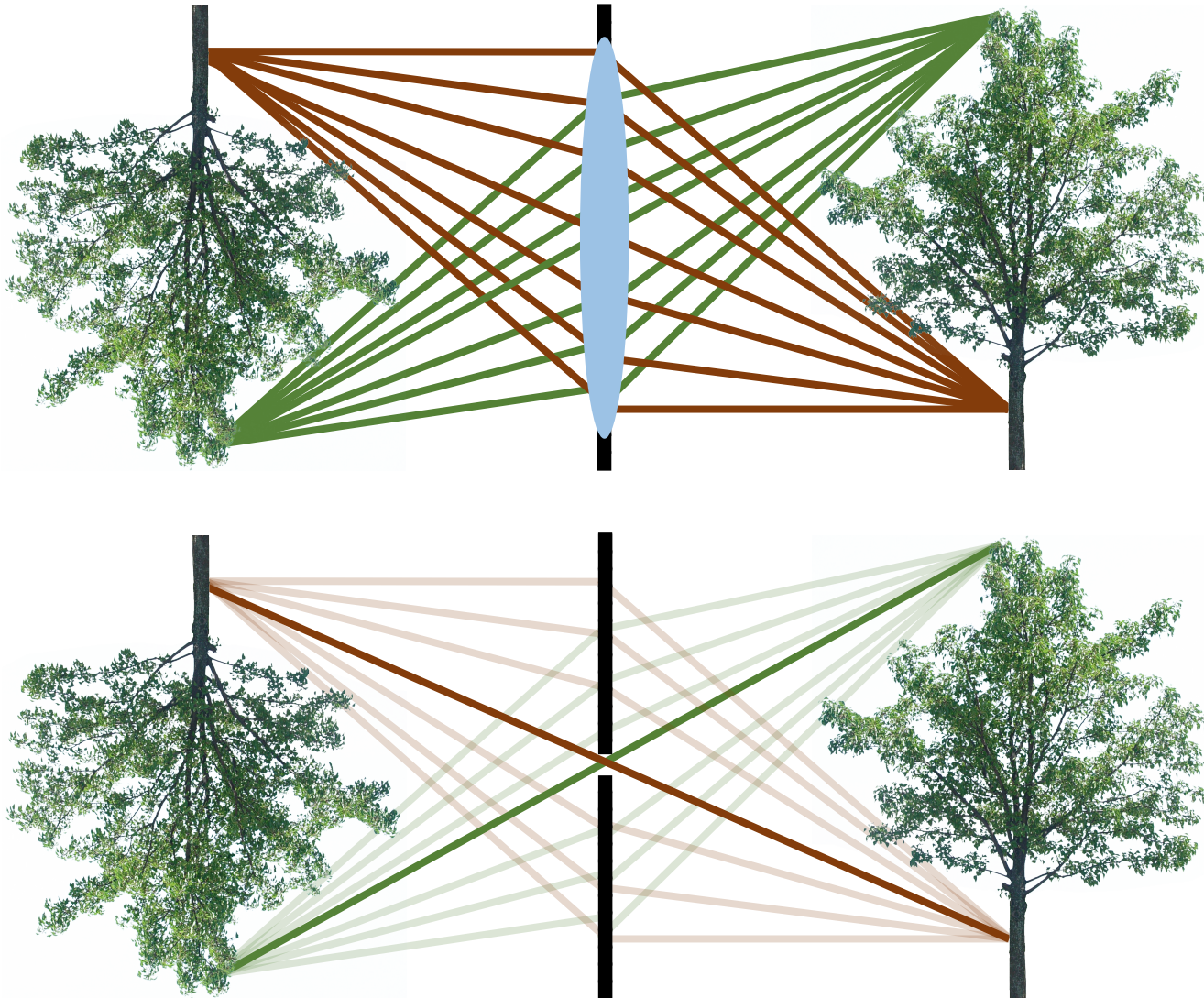
Lecture 12

Camera calibration

So far: 3D to 2D projections via a camera



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: 2D Transformations with homogeneous coordinates

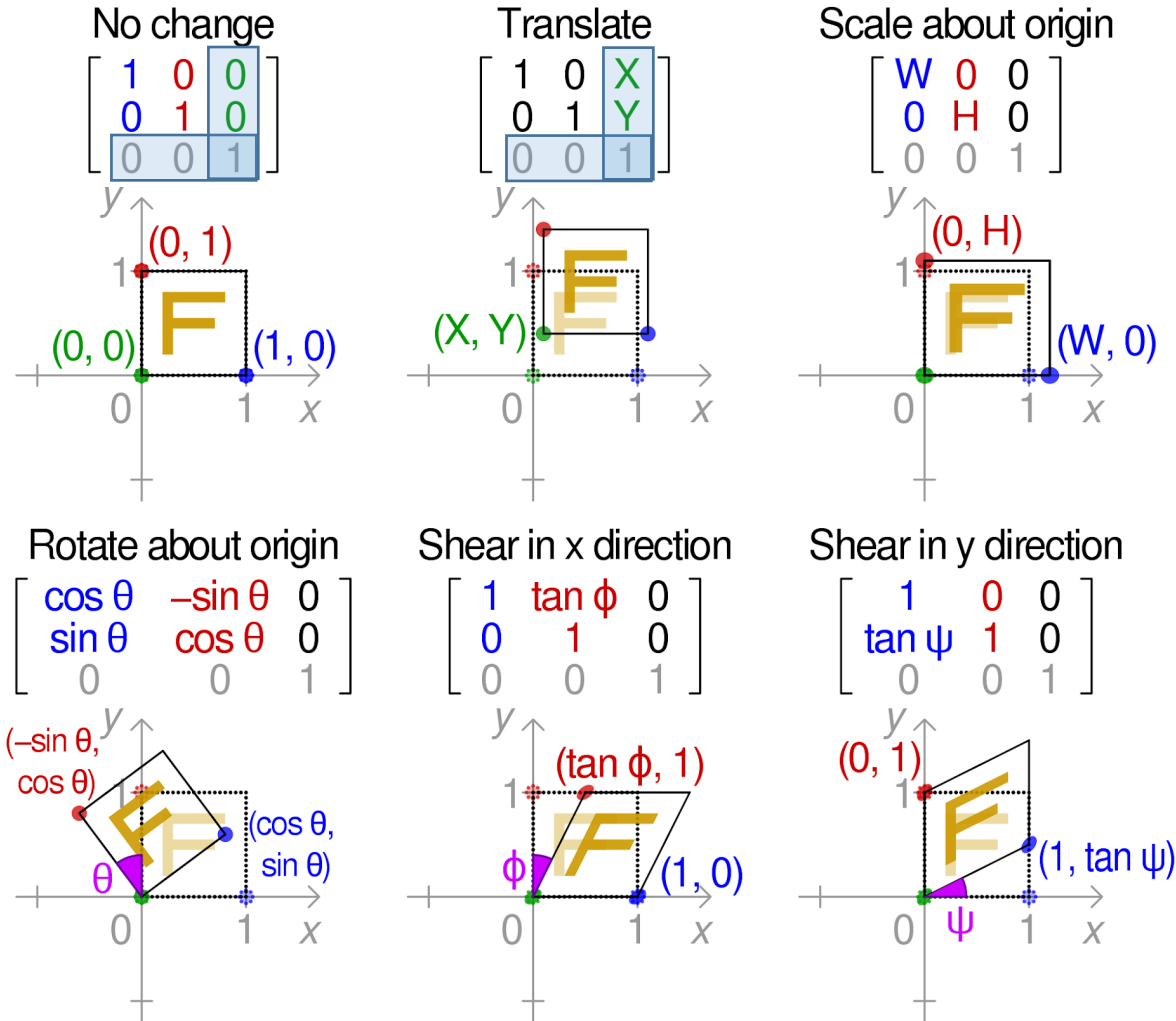


Figure: Wikipedia

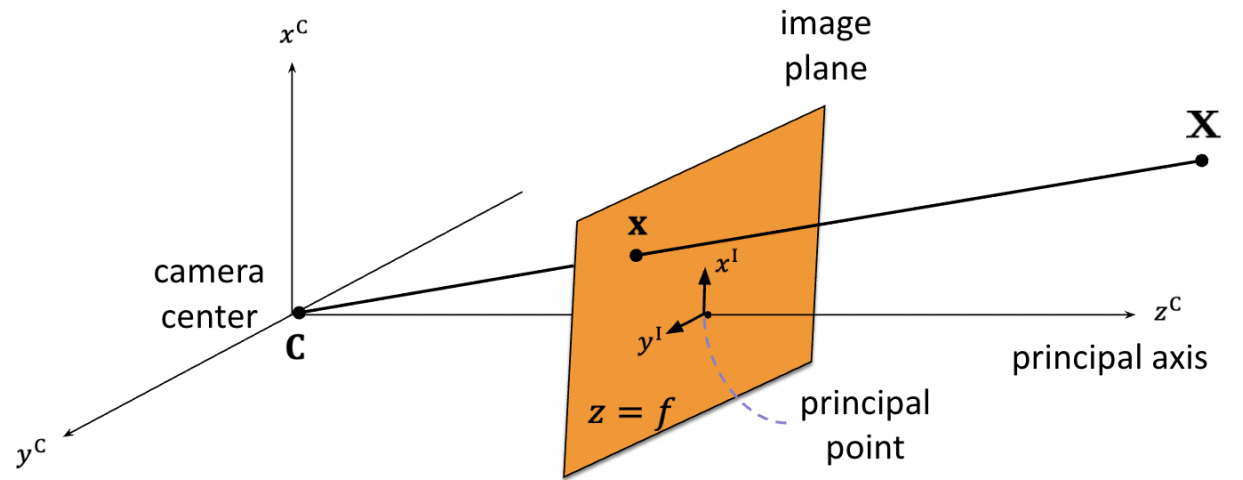
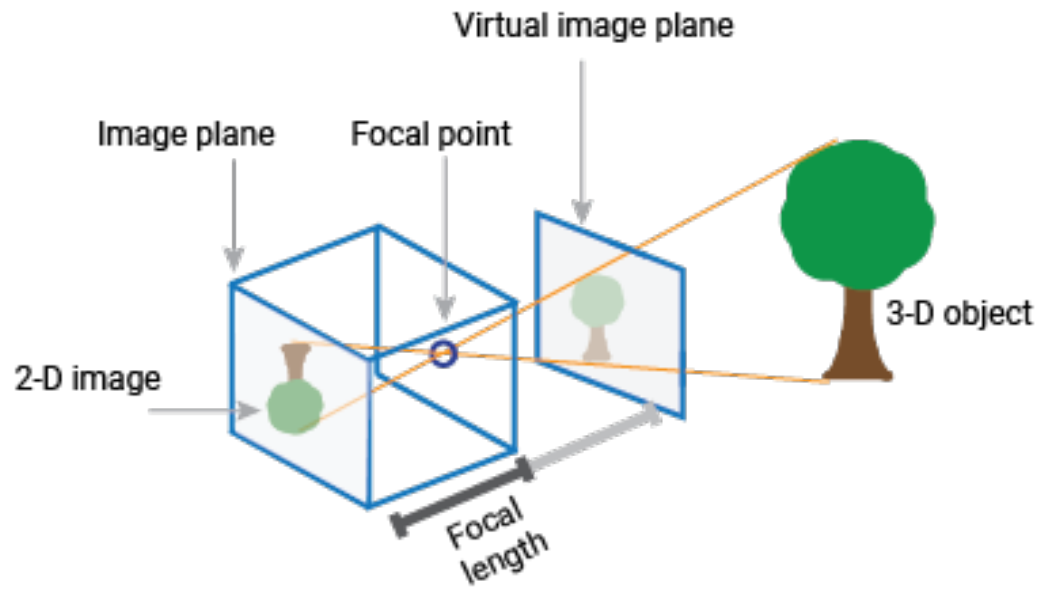
So far: 2D Transformations with homogeneous coordinates

2D Projective Space: $\mathcal{P}^2 = \mathcal{R}^3 - (0, 0, 0)$ (same story in 3D with \mathcal{P}^3)

- heterogeneous \rightarrow homogeneous $\begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
- homogeneous \rightarrow heterogeneous $\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow \begin{bmatrix} x/w \\ y/w \end{bmatrix}$
- points differing only by scale are *equivalent*: $(x, y, w) \sim \lambda (x, y, w)$

$$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(x, y, 1) = \tilde{w}\bar{\mathbf{x}}$$

So far: General pinhole camera matrix



What is the transformation $\mathbf{x} = \mathbf{P}\mathbf{X}$?

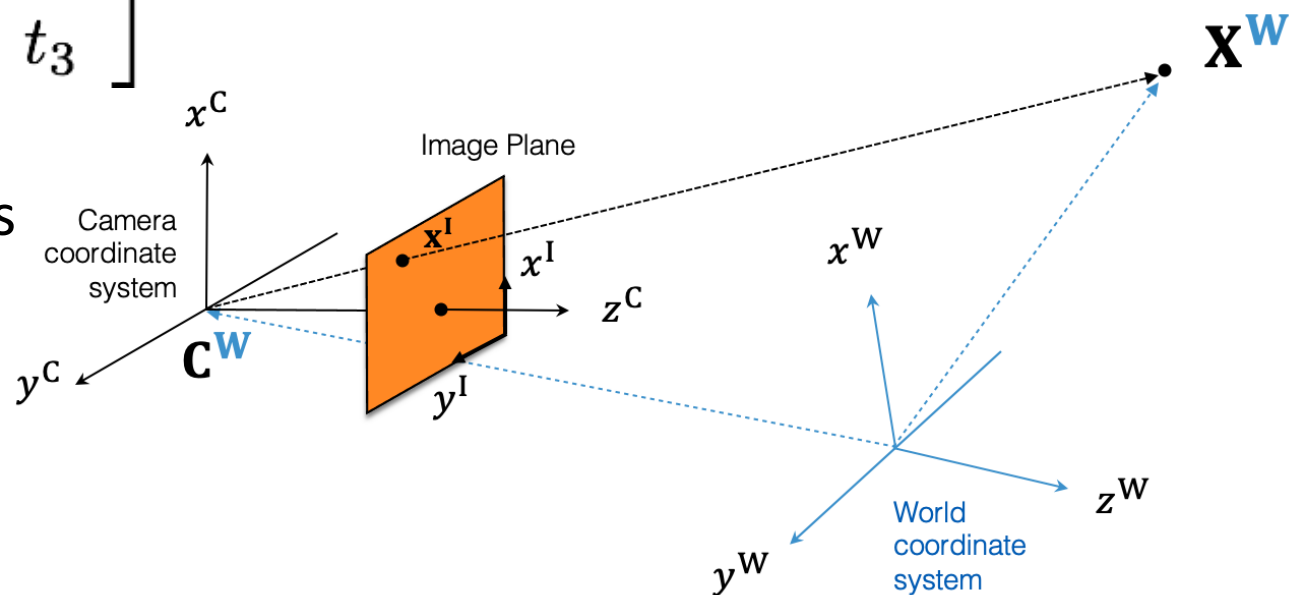
So far: General pinhole camera matrix

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

$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix}$$

intrinsic
parameters

extrinsic
parameters



Today's agenda

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

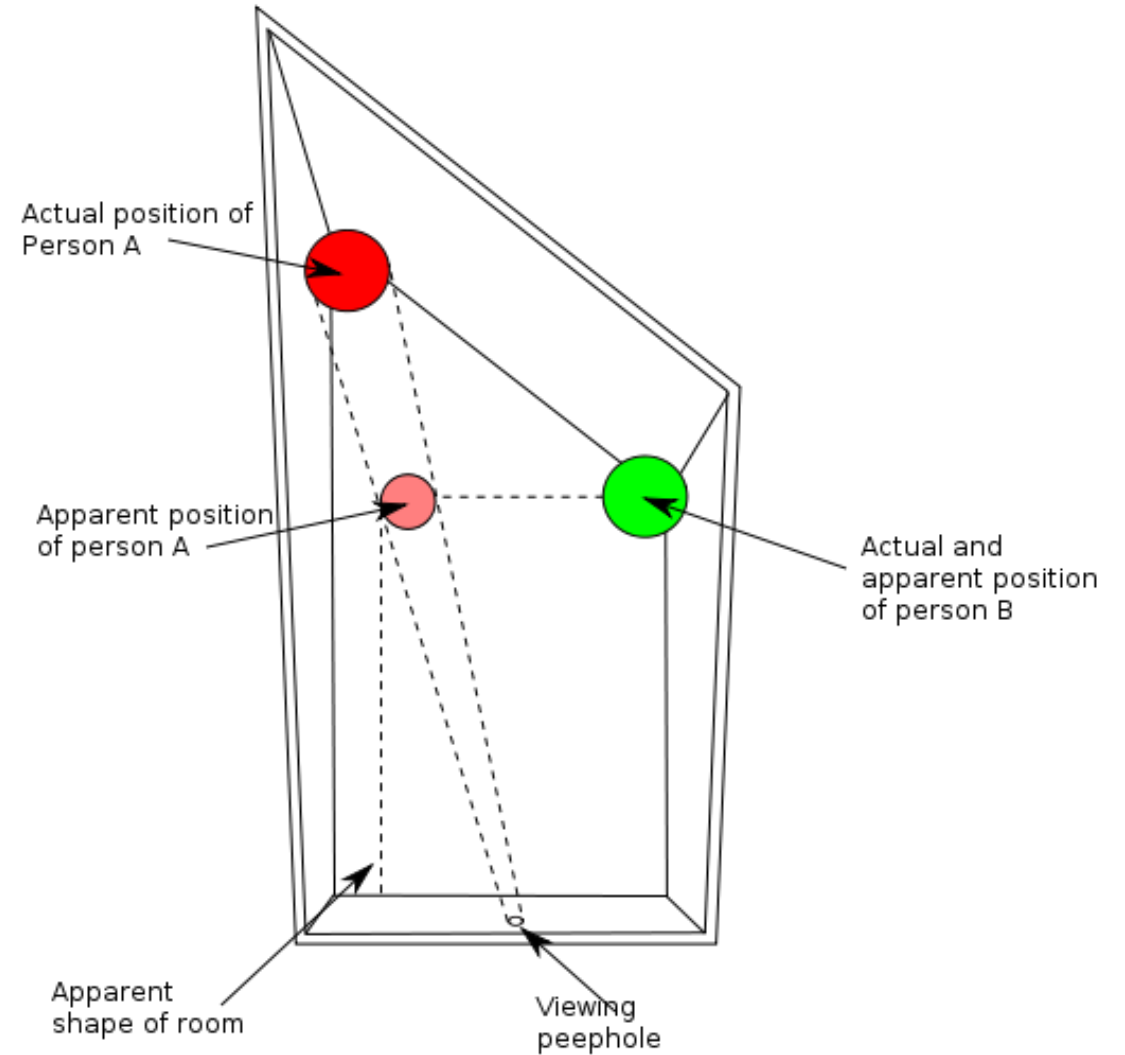
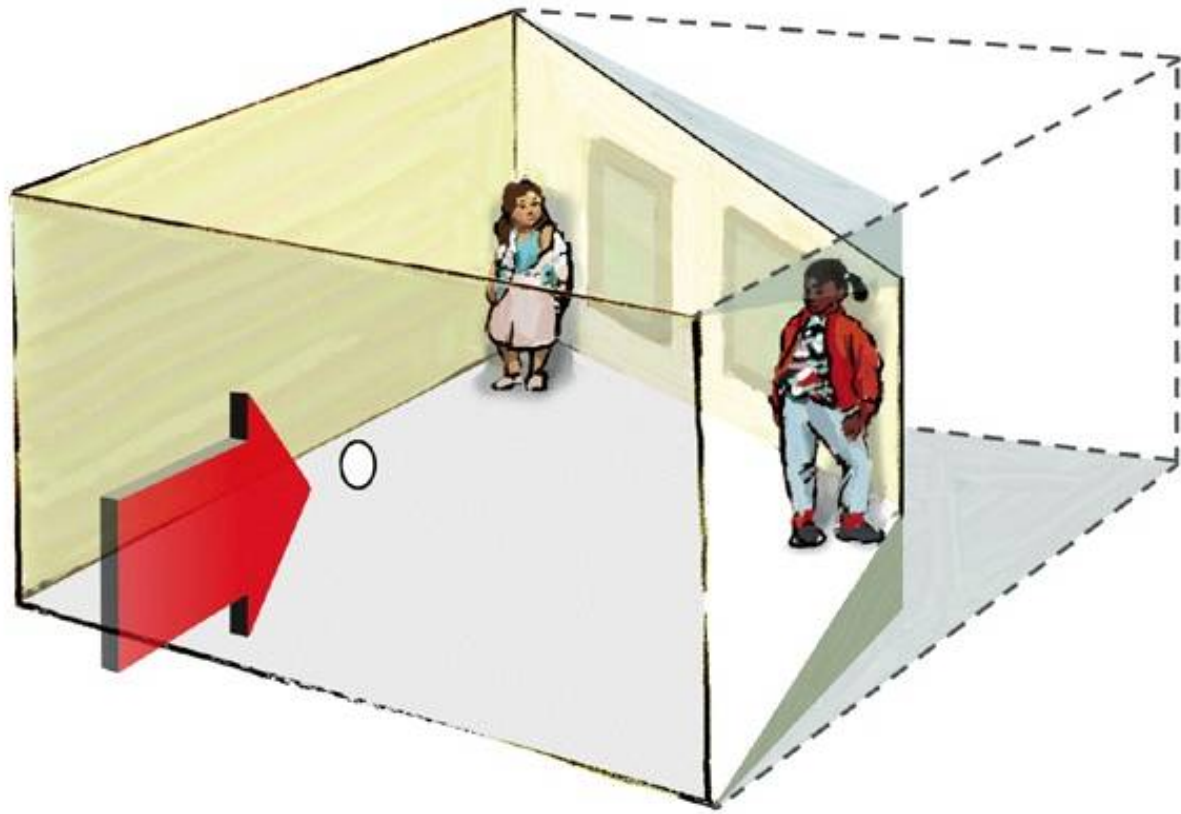
Today's agenda

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

Similar illusion as last lecture



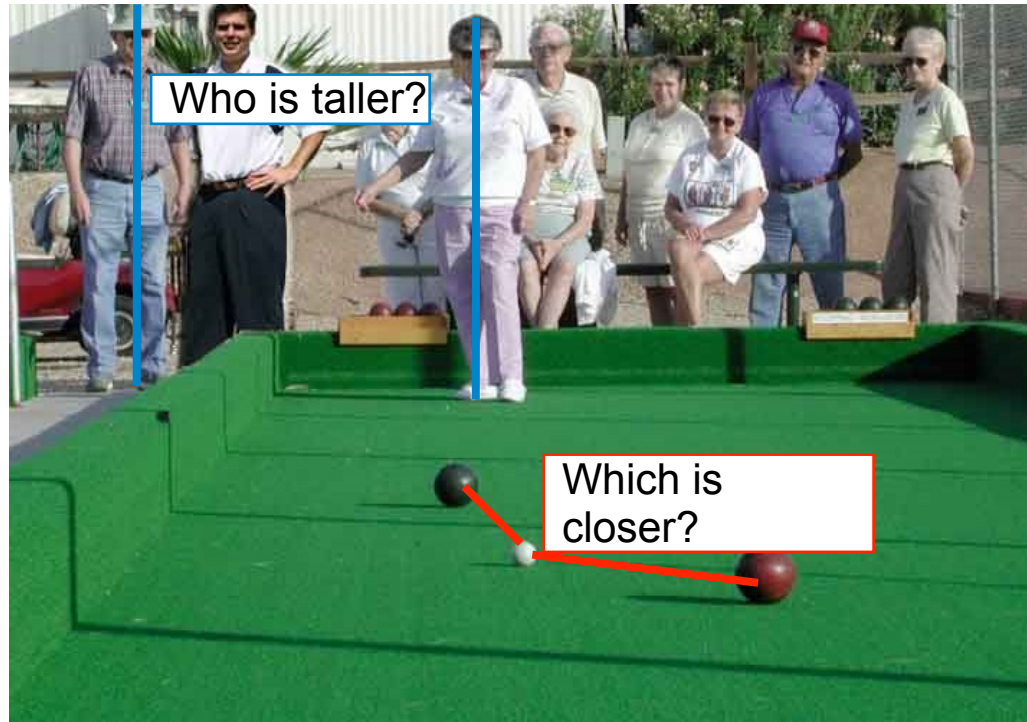
The Ames room illusion



Projective Geometry

Q. Who is taller?

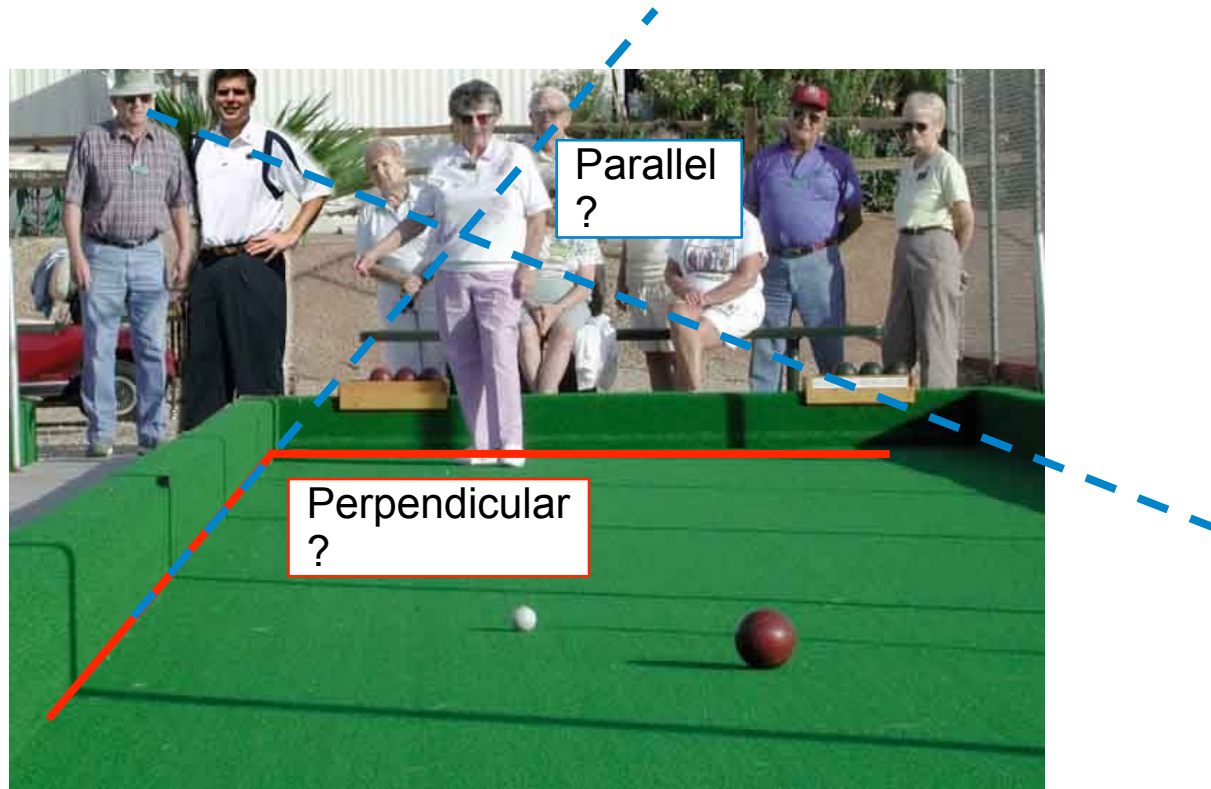
The two blue lines are the same length



Projective Geometry

What is **not** preserved?

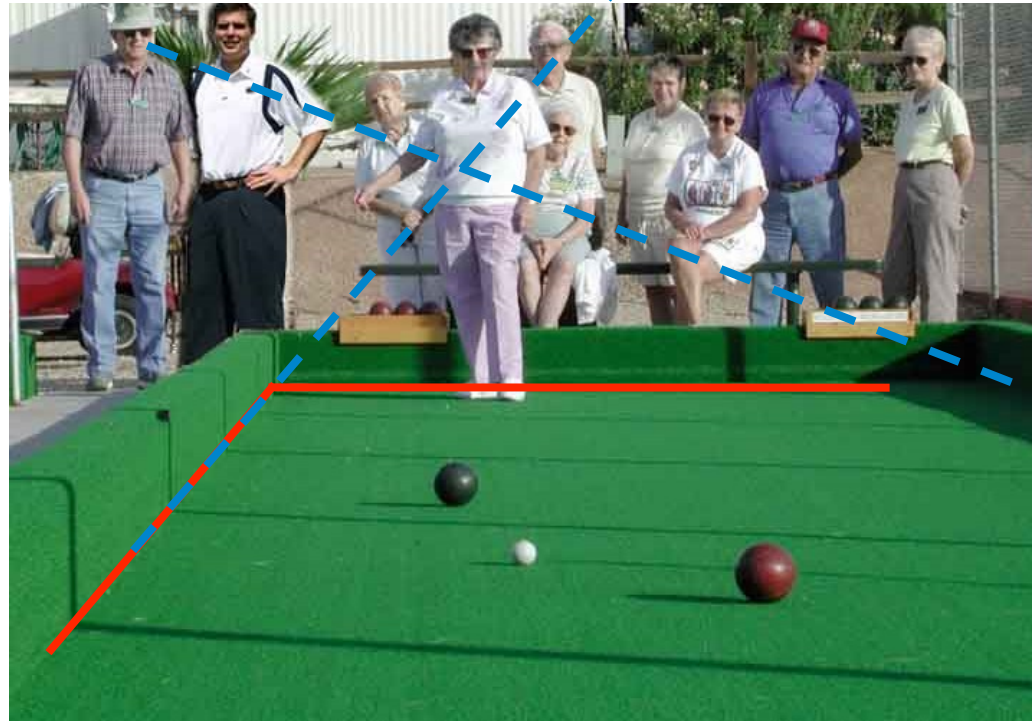
- Length
- Angles



Projective Geometry

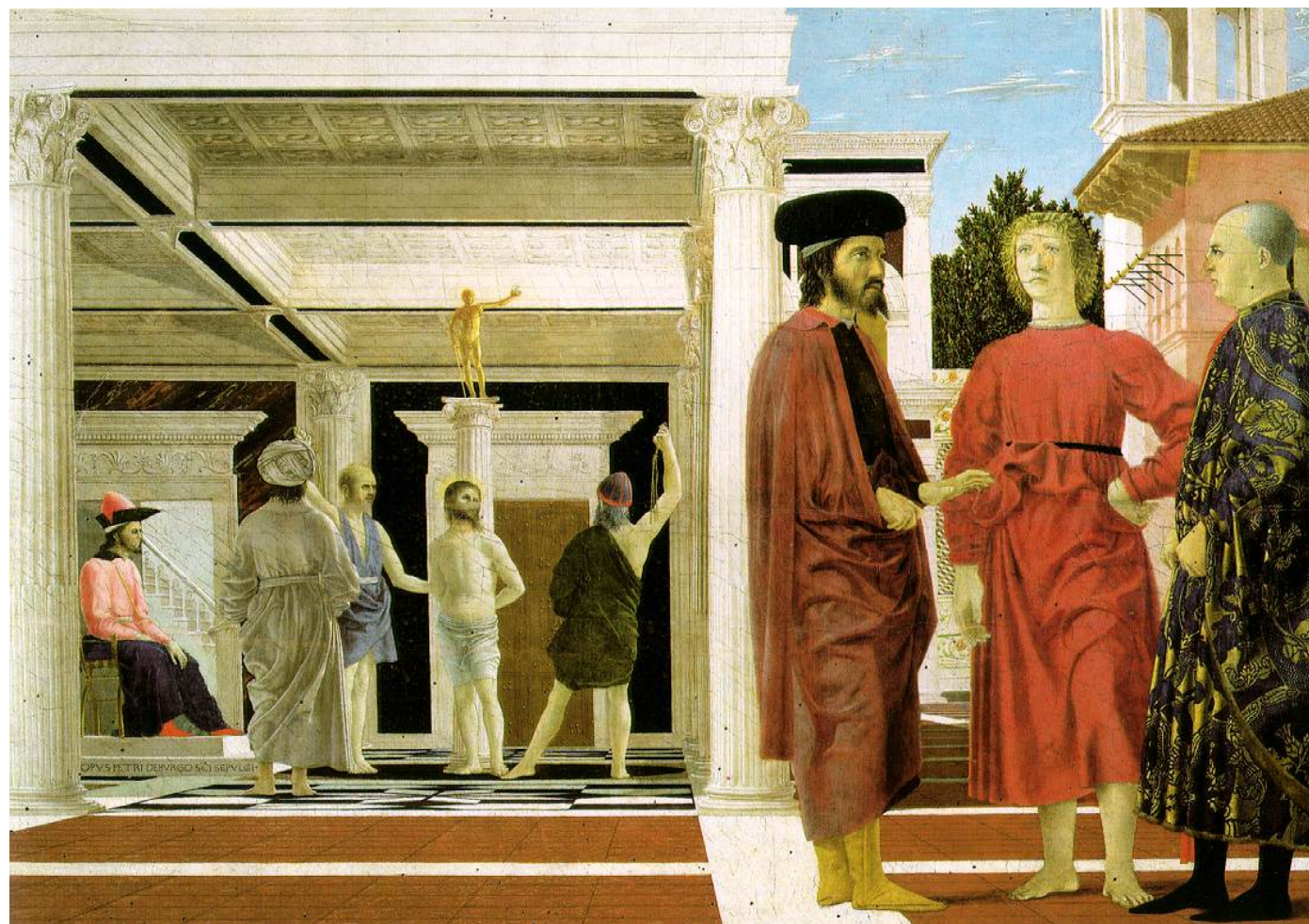
What is preserved?

- Straight lines are still straight



Projection of lines

Q. When is parallelism preserved?

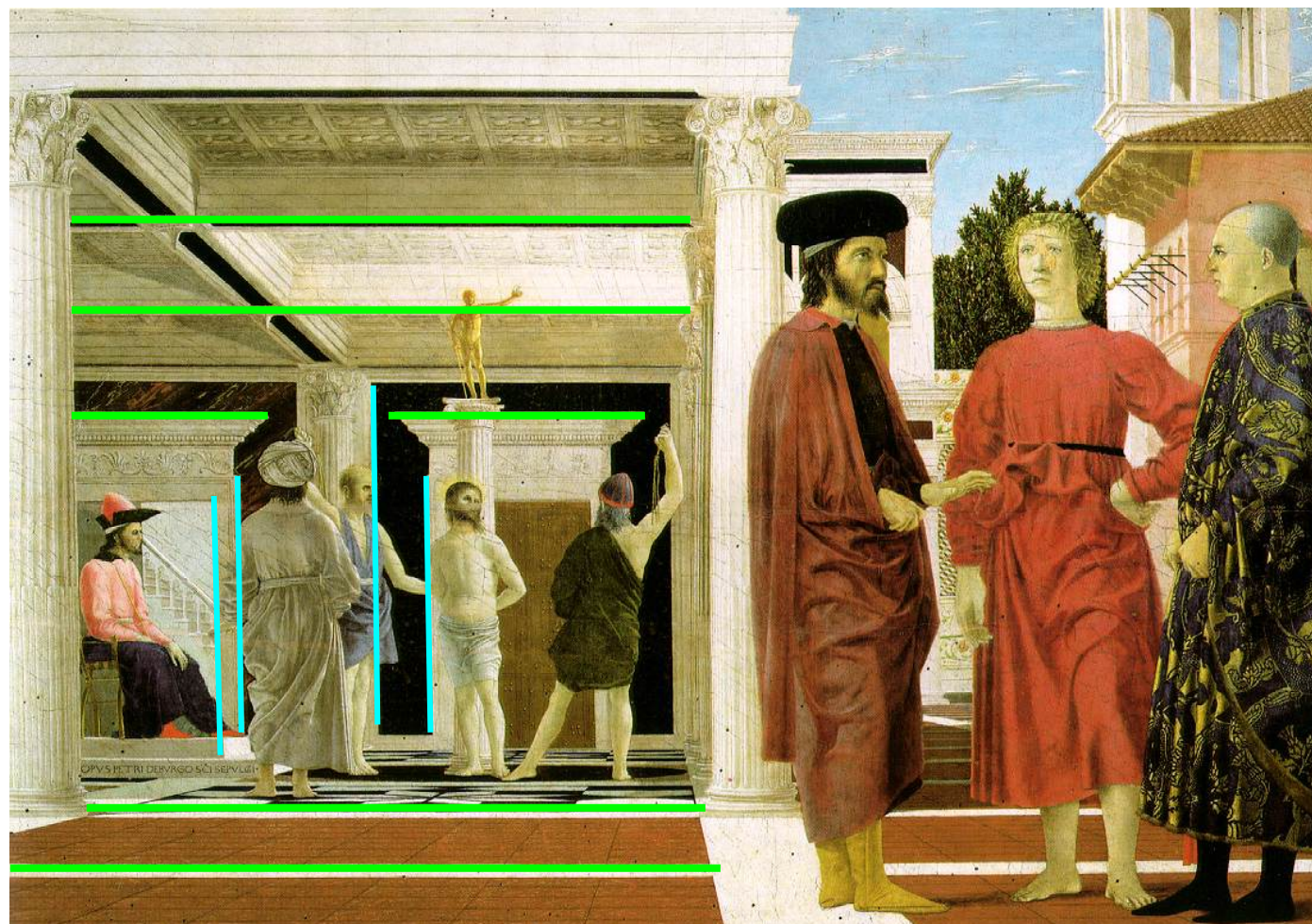


Piero della Francesca, *Flagellation of Christ*, 1455-1460

Projection of lines

Q. When is parallelism preserved?

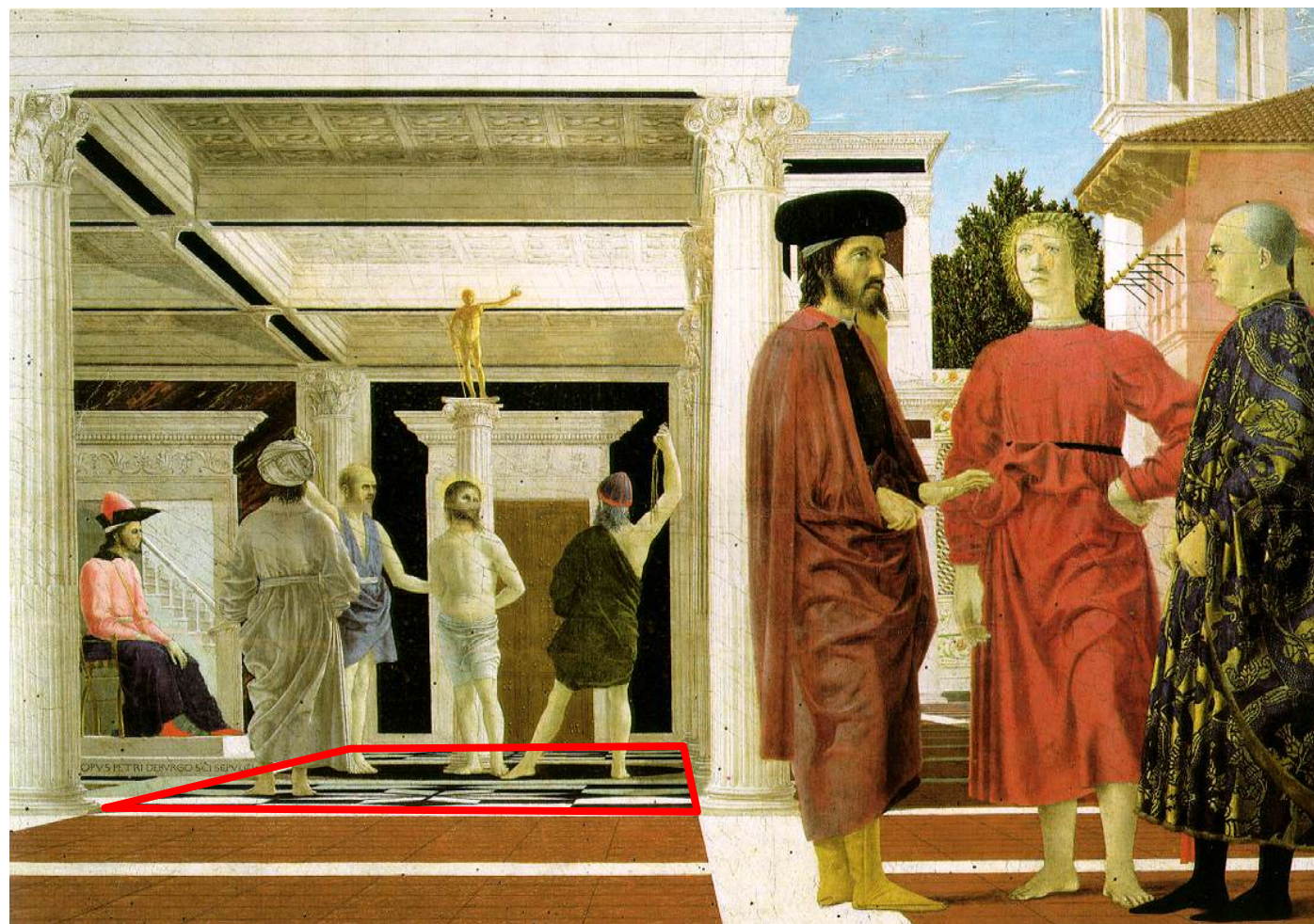
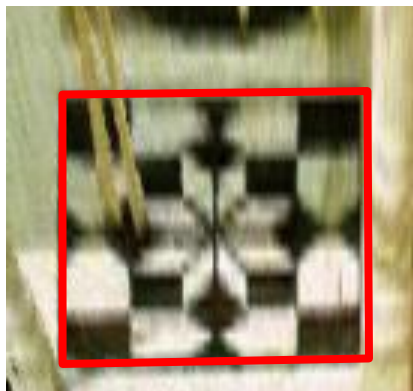
When the parallel lines are also parallel to the image plane



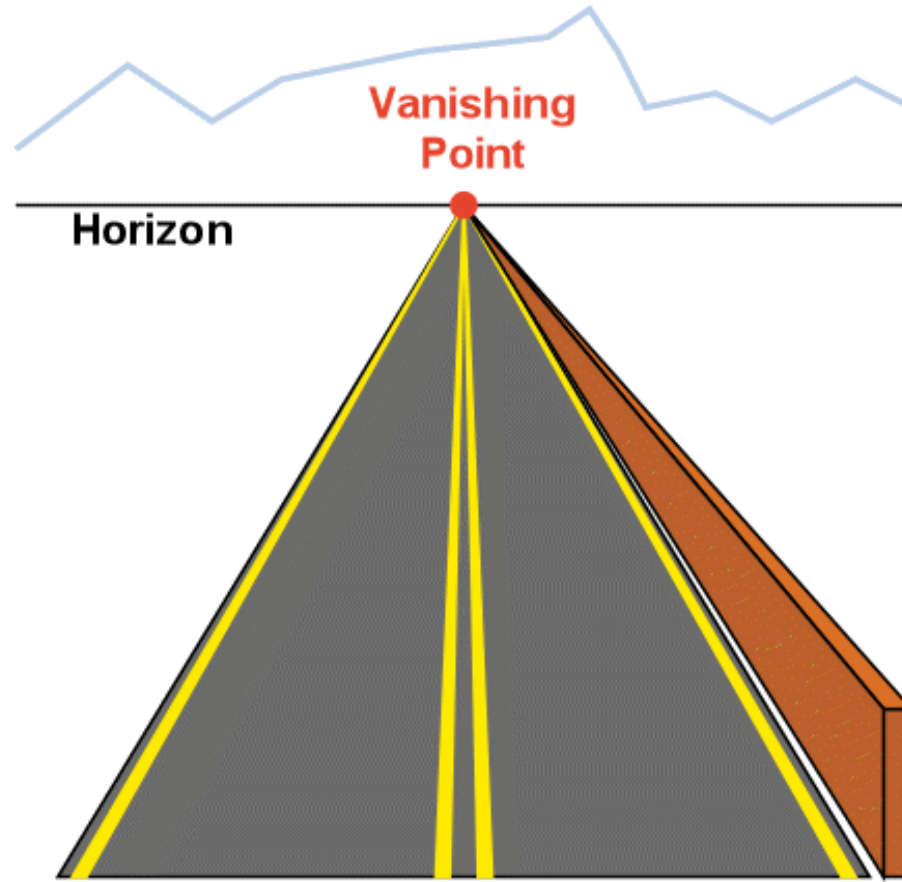
Piero della Francesca, *Flagellation of Christ*, 1455-1460

Projection of lines

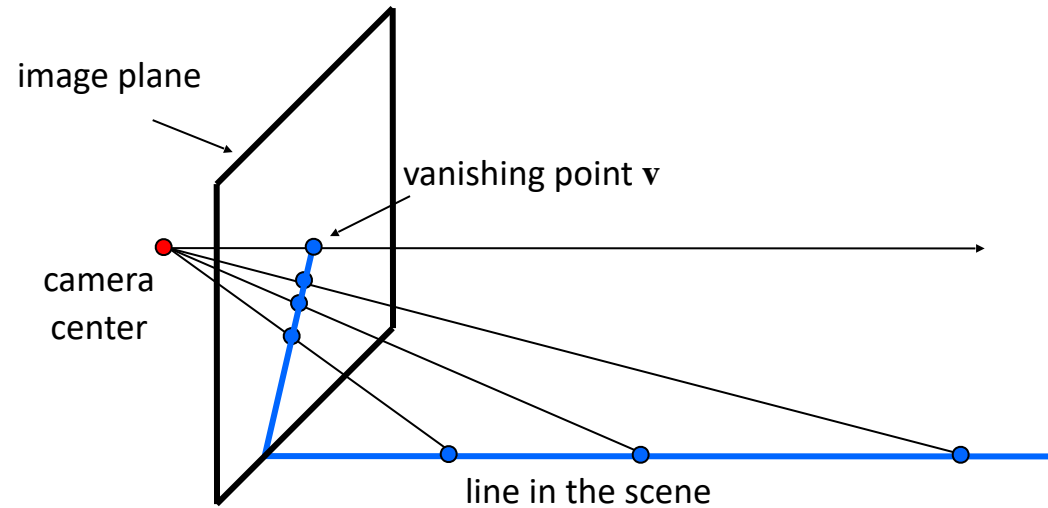
Patterns on *non-fronto-parallel* planes are distorted by a *homography*



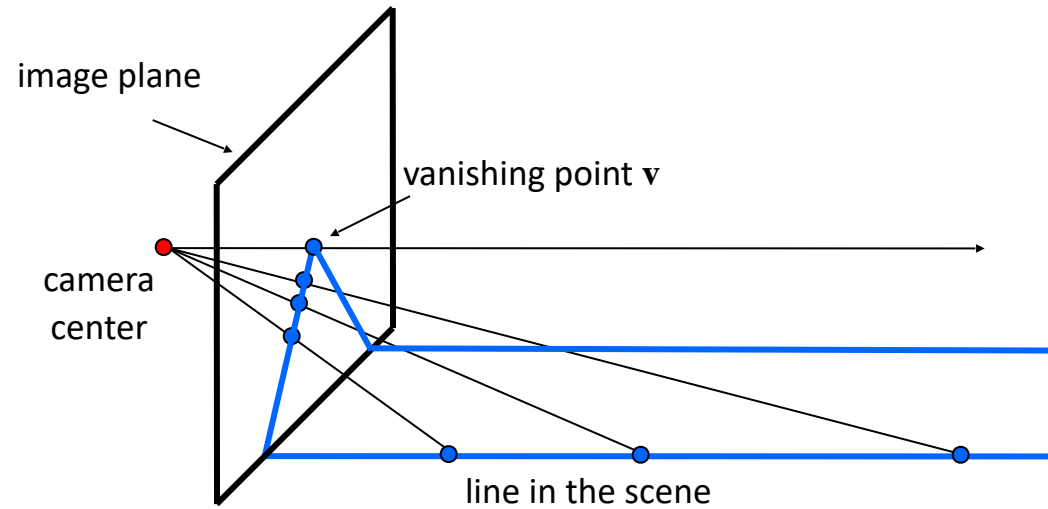
If parallel lines are no longer parallel, where do they intersect?



Vanishing Points & Lines



Vanishing Points & Lines



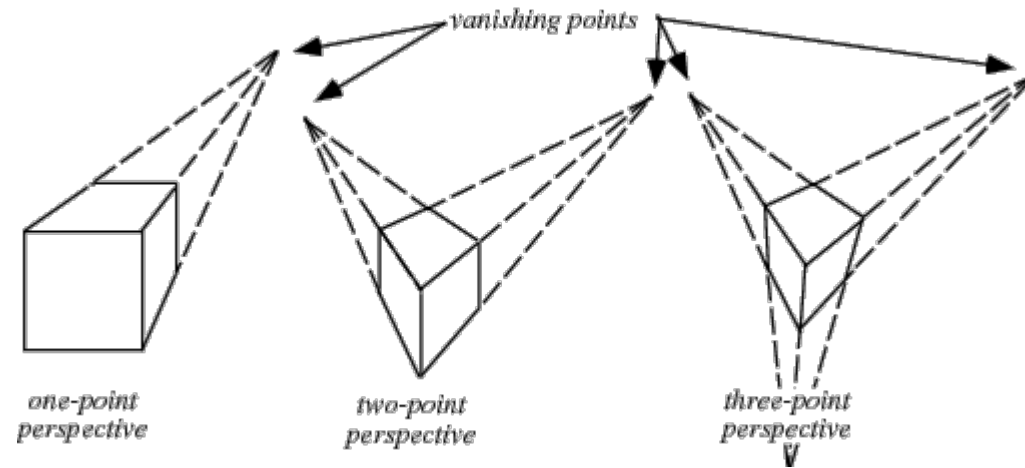
Vanishing Points & Lines



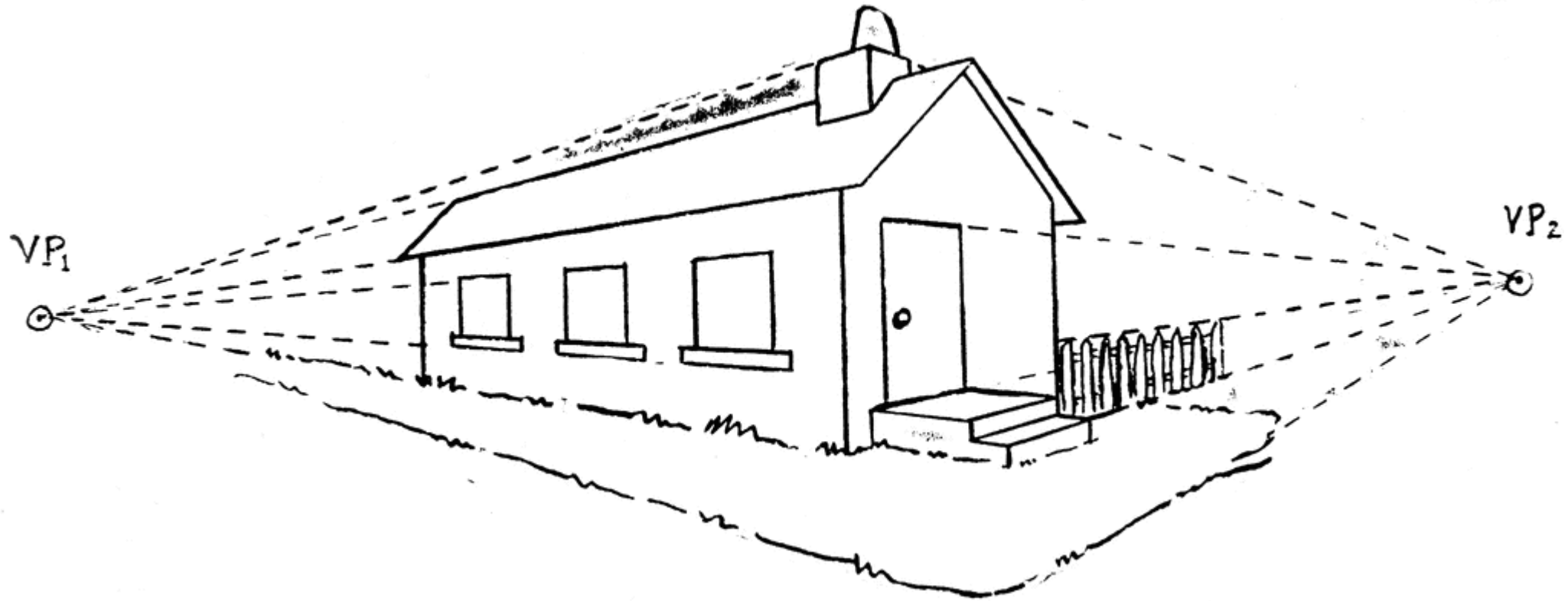
1-, 2-, 3-perspective

3D objects can have 1, 2, or 3 vanishing points depending on the camera location.

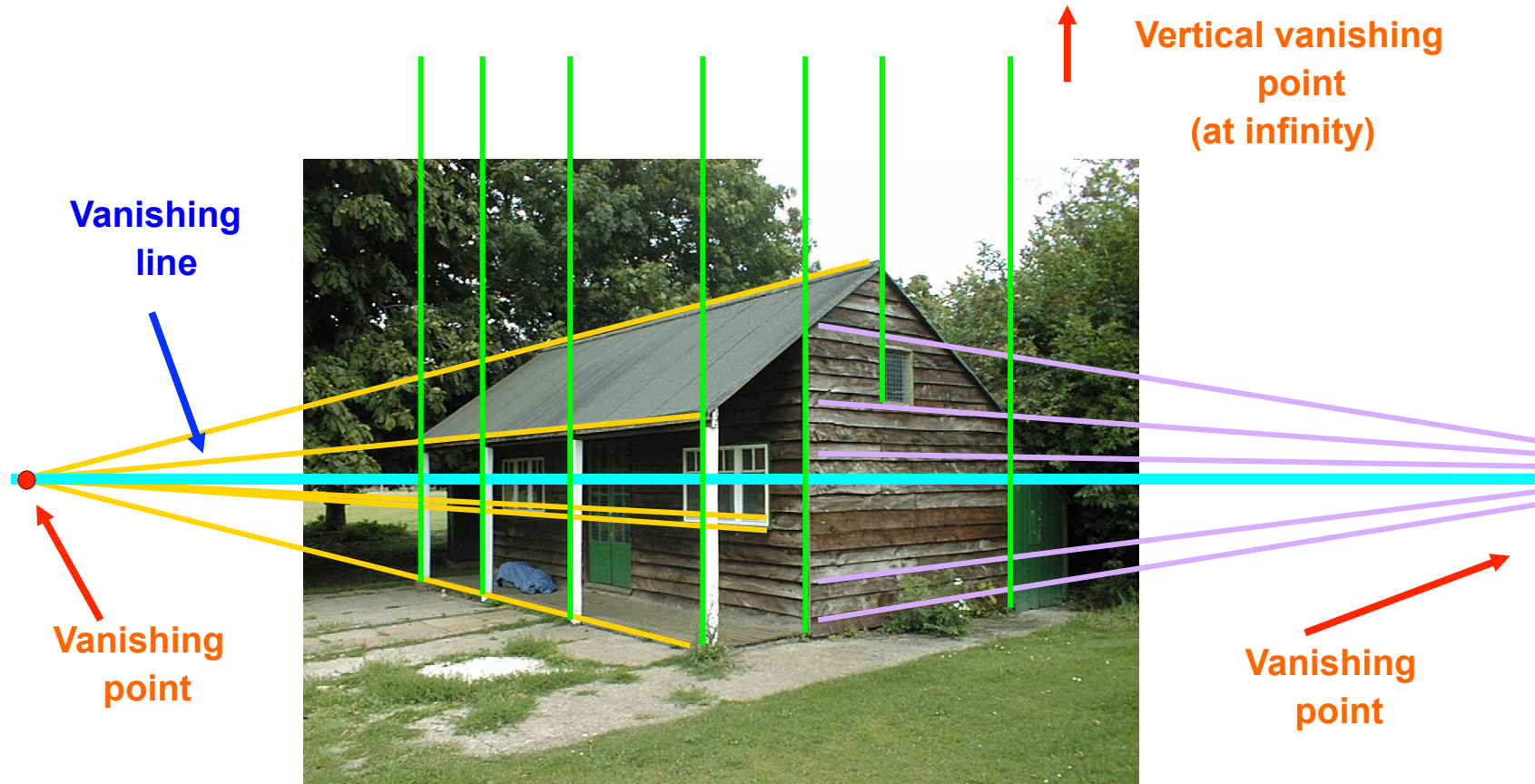
An image with multiple objects can have an arbitrary number of vanishing points.



Vanishing lines for a house

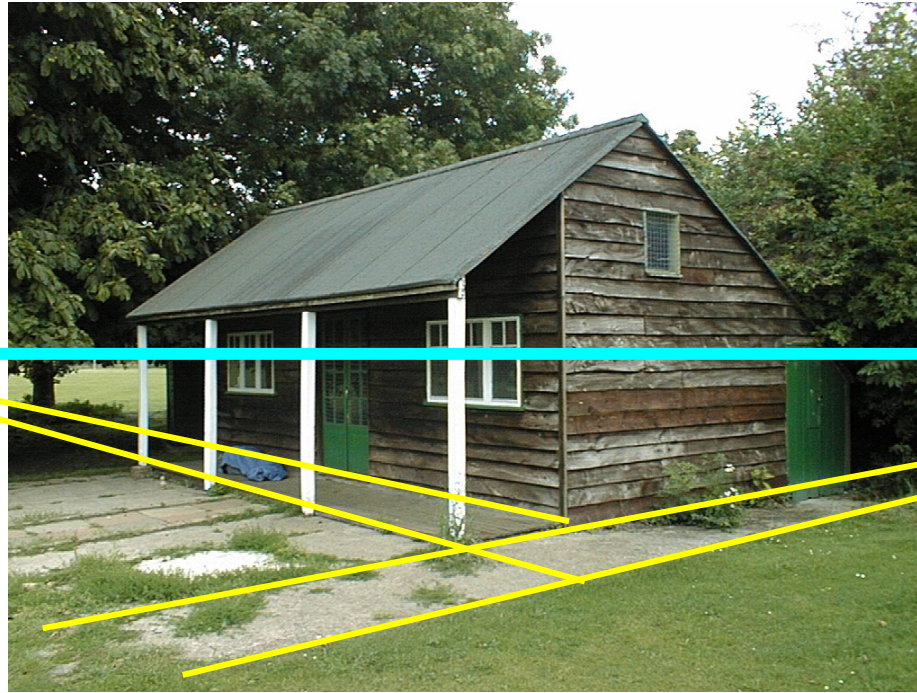


Parallel lines in the world intersect in the image at a vanishing point

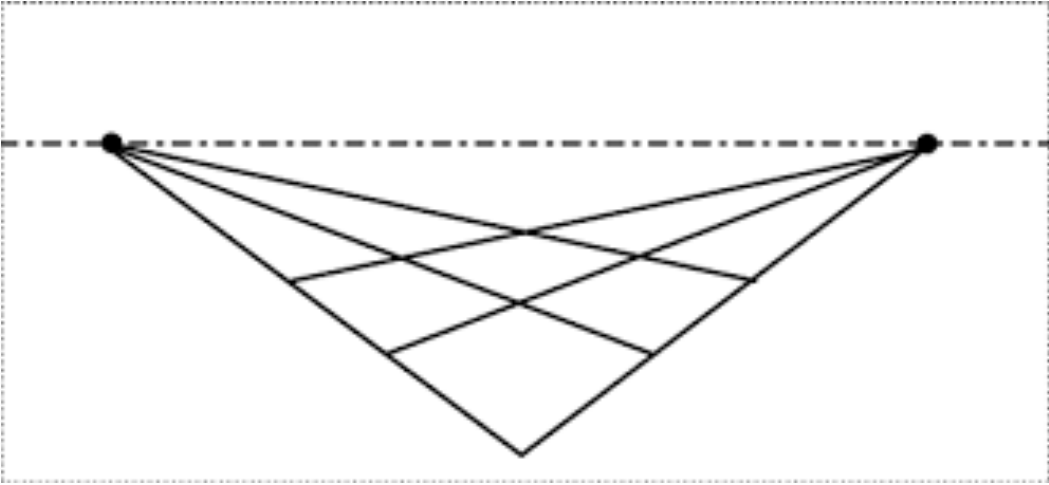


Not all lines that intersect in 2D are parallel in 3D!

Horizon: vanishing line of the ground plane (and planes parallel to it)



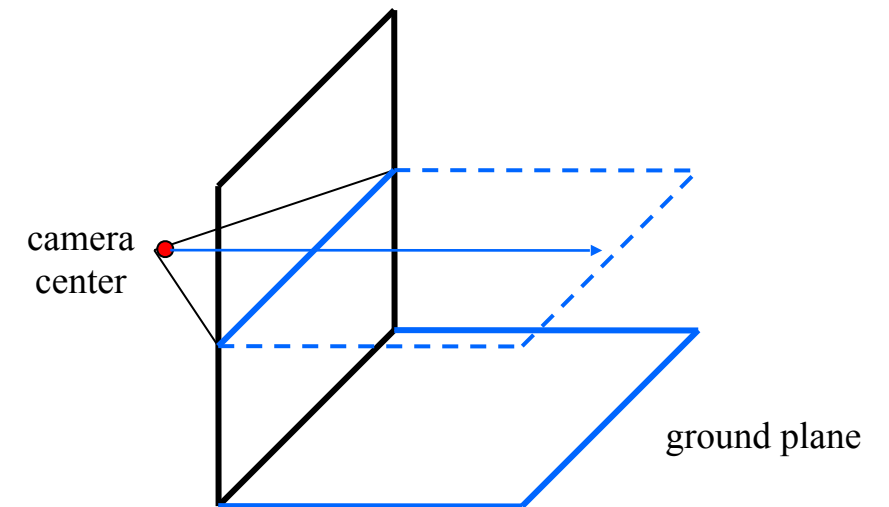
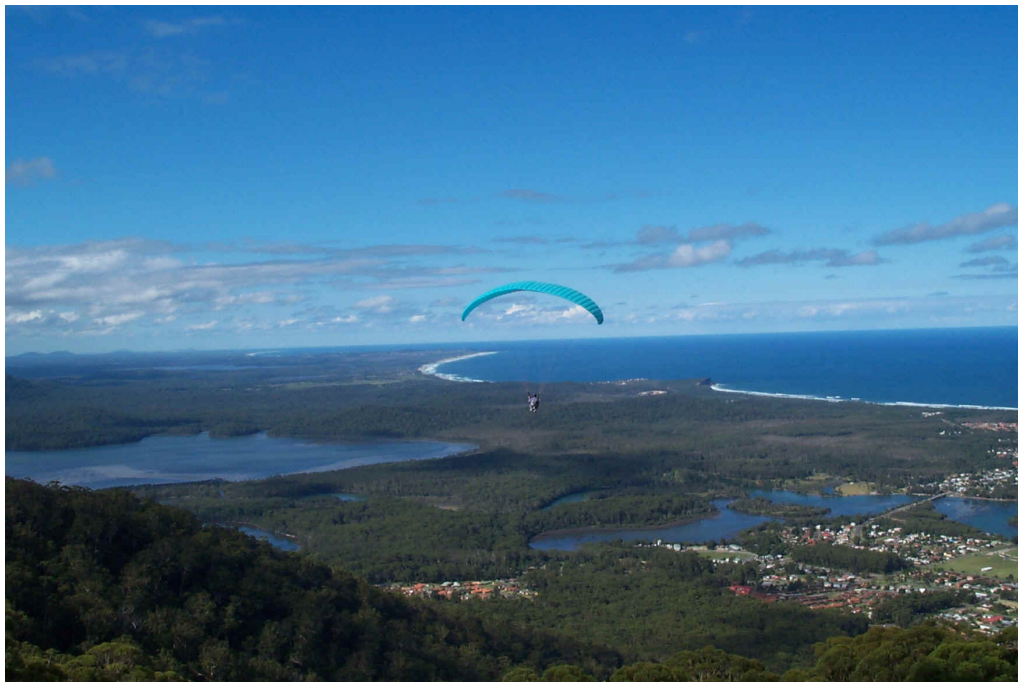
Parallel planes *in the world* intersect *in the image* at a **vanishing line**



Vanishing lines of planes

Horizon: vanishing line of the ground plane

Q. Is this parachutist **higher** or **lower** than the person taking this picture?

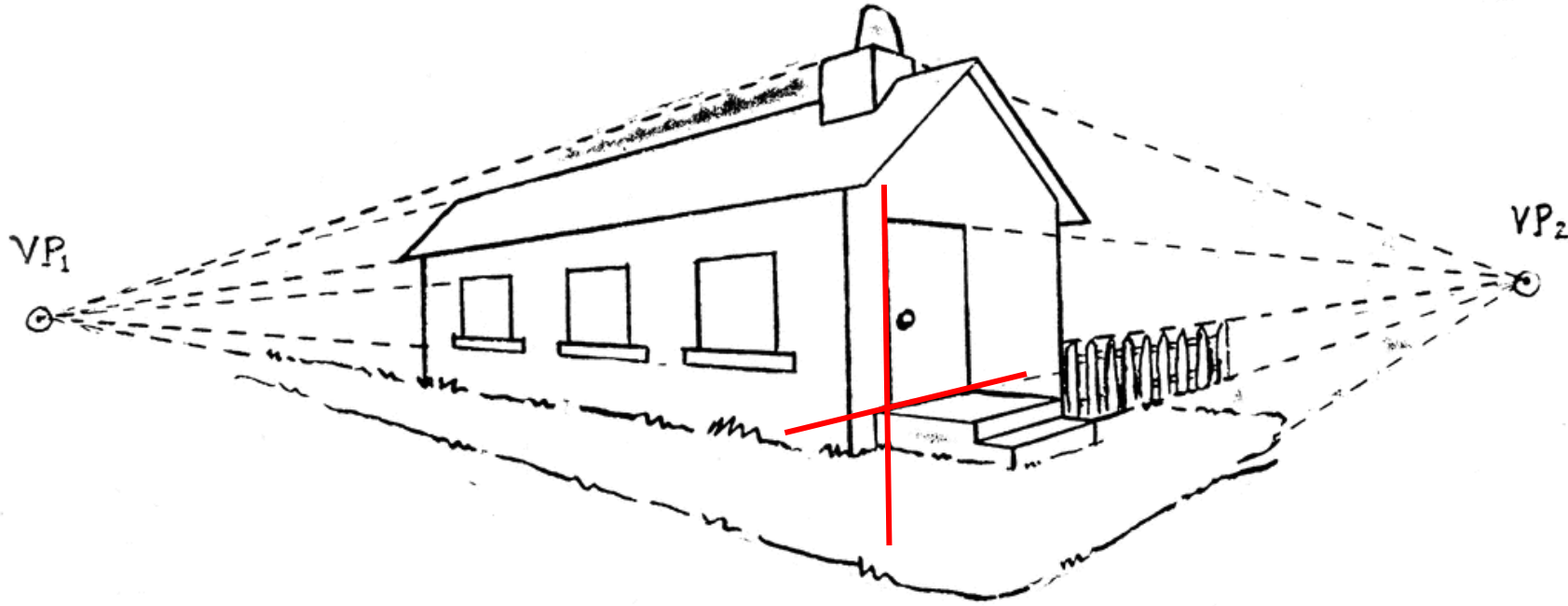


Q. Let's try and answer these together with a neighbor

1. All lines that intersect in 2D are parallel in 3D
2. Lines in 3D always intersect each other in 2D at a single point
3. All non-parallel 3D planes in the real world intersect each other
4. 3D lines in the real world always intersect each other
5. All 3D lines intersect each other in the 2D image
6. All parallel lines in 3D meet at the same vanishing point
7. 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

1. All lines that intersect in 2D are parallel in 3D

Not true.



2. Lines in 3D always intersect each other in 2D at a single point

Lines in 3D are still lines in 2D.

Any two distinct lines meet in exactly one point (which may lie 'at infinity' if the lines are parallel in the Euclidean sense).

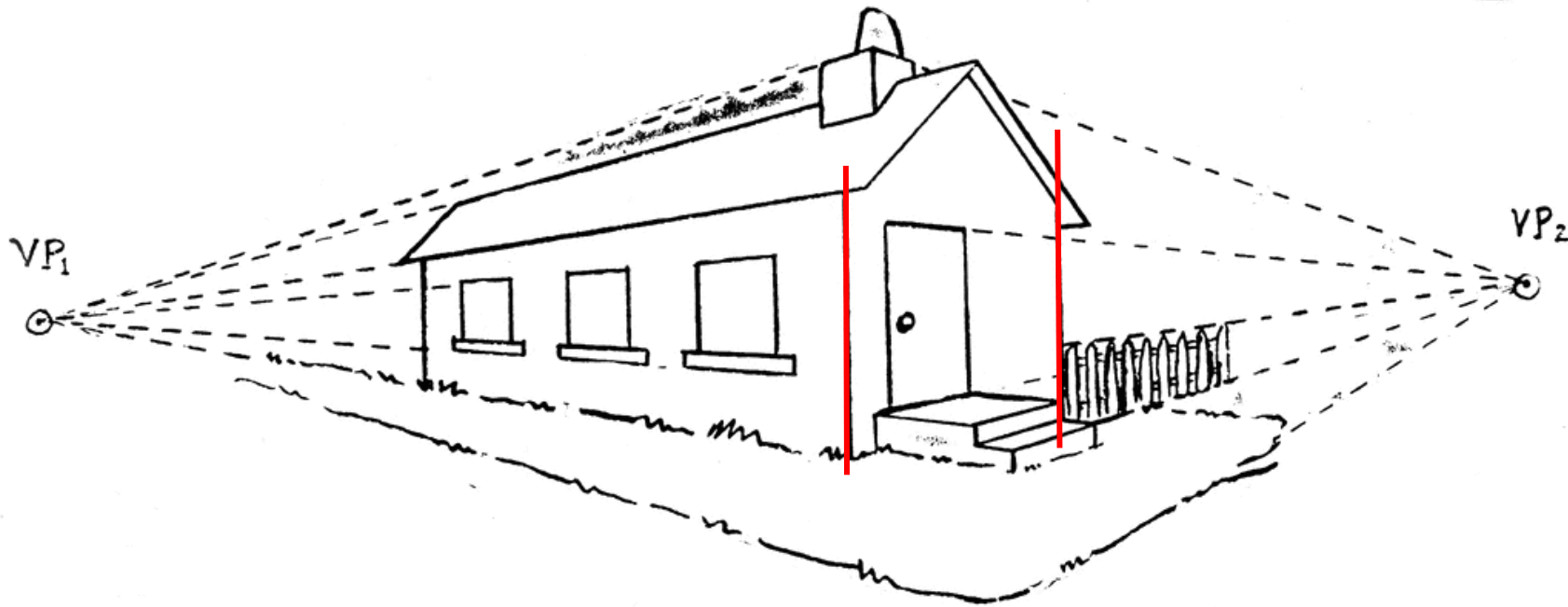
3. All non-parallel 3D planes in the real world intersect each other

True. They intersect at a line.

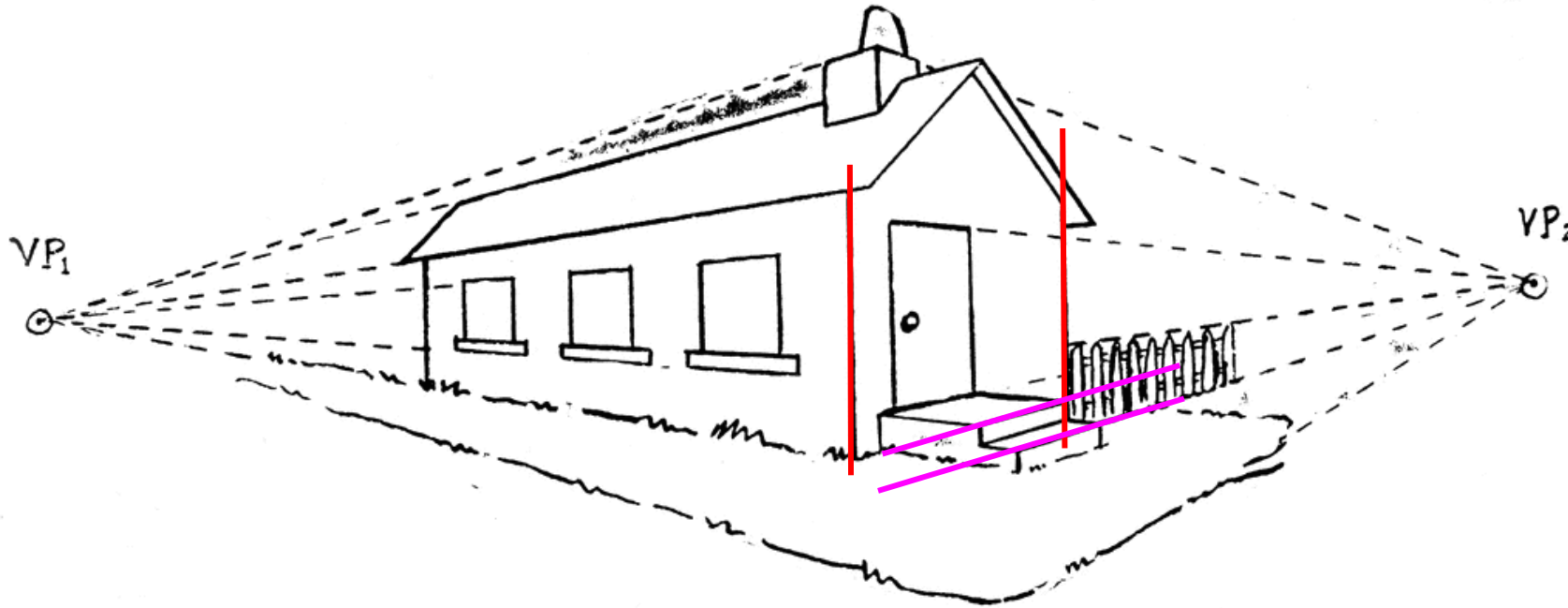
Parallel planes could potentially intersect at infinity (but don't have to)

The horizon line is where the ground planes intersect

4. 3D lines in the real world always intersect each other



5. All 3D lines intersect each other in the 2D image



6. All parallel lines in 3D meet at the same vanishing point

Q. how would you go about proving this?

6. All parallel lines in 3D meet at the same vanishing point

consider a simple camera with its camera z-axis aligned with the world z-axis:

For line $\ell_i(t) = (p_{ix} + t d_x, p_{iy} + t d_y, p_{iz} + t d_z)$,

its projected image in the 2D plane is:

$$(x_i(t), y_i(t)) = \pi(\ell_i(t)) = \left(\frac{p_{ix} + t d_x}{p_{iz} + t d_z}, \frac{p_{iy} + t d_y}{p_{iz} + t d_z} \right),$$

6. All parallel lines in 3D meet at the same vanishing point

consider a simple camera with its camera z-axis aligned with the world z-axis:

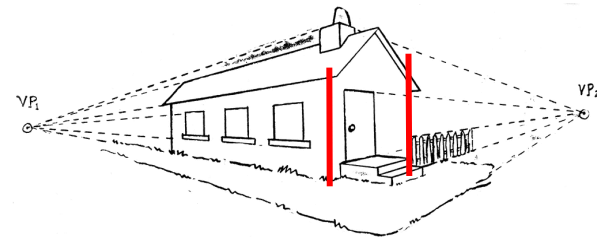
For line $\ell_i(t) = (p_{ix} + t d_x, p_{iy} + t d_y, p_{iz} + t d_z)$,

its projected image in the 2D plane is:

$$(x_i(t), y_i(t)) = \pi(\ell_i(t)) = \left(\frac{p_{ix} + t d_x}{p_{iz} + t d_z}, \frac{p_{iy} + t d_y}{p_{iz} + t d_z} \right),$$

as long as $p_{iz} + t d_z \neq 0$.

when $d_z = 0$, the lines are parallel in both 3D and 2D and never intersect



6. All parallel lines in 3D meet at the same vanishing point

Since the lines are parallel in 3D, they intersect

As $t \rightarrow \infty$, the ratio

$$x_i(t) = \frac{p_{ix} + t d_x}{p_{iz} + t d_z} \quad \text{and} \quad y_i(t) = \frac{p_{iy} + t d_y}{p_{iz} + t d_z}$$

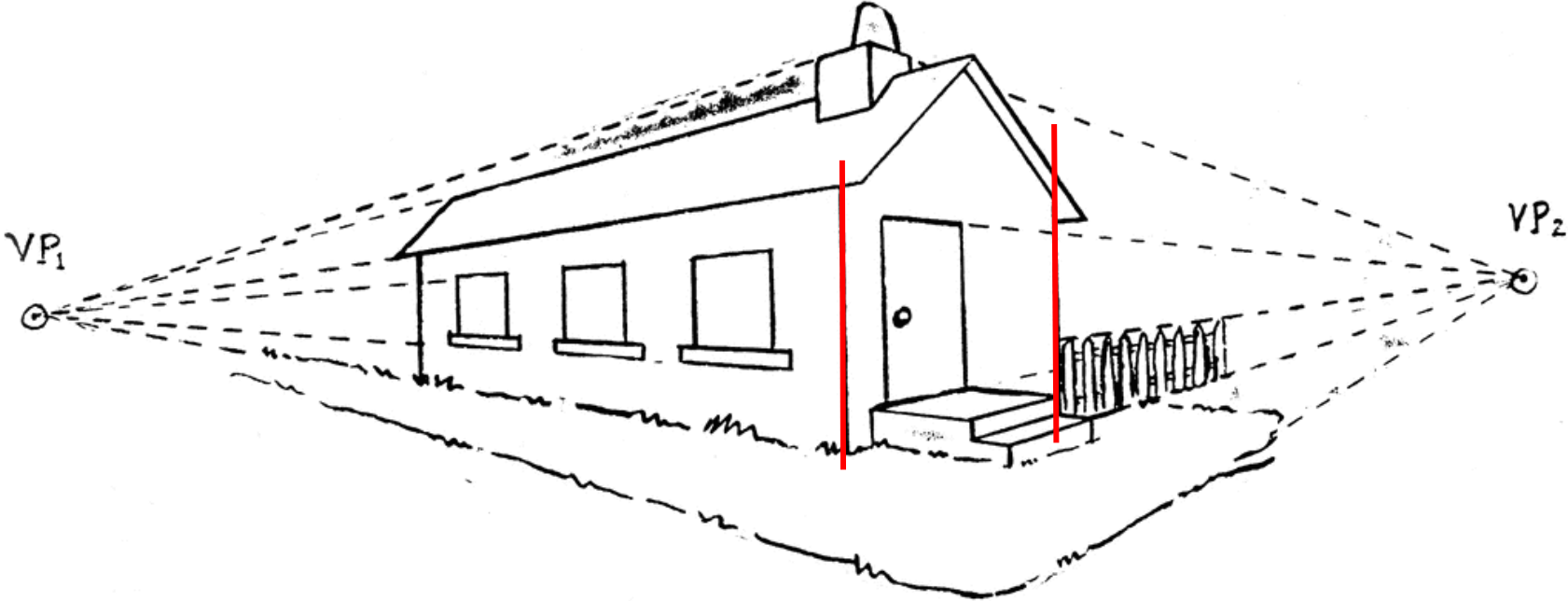
behaves like

$$x_i(t) \approx \frac{t d_x}{t d_z} = \frac{d_x}{d_z}, \quad y_i(t) \approx \frac{d_y}{d_z}.$$

In the limit $t \rightarrow \infty$, $(x_i(t), y_i(t))$ approaches the **same** coordinate $\left(\frac{d_x}{d_z}, \frac{d_y}{d_z}\right)$, regardless of

\mathbf{p}_i .

7. Non-intersecting lines in 3D meet at the same vanishing point

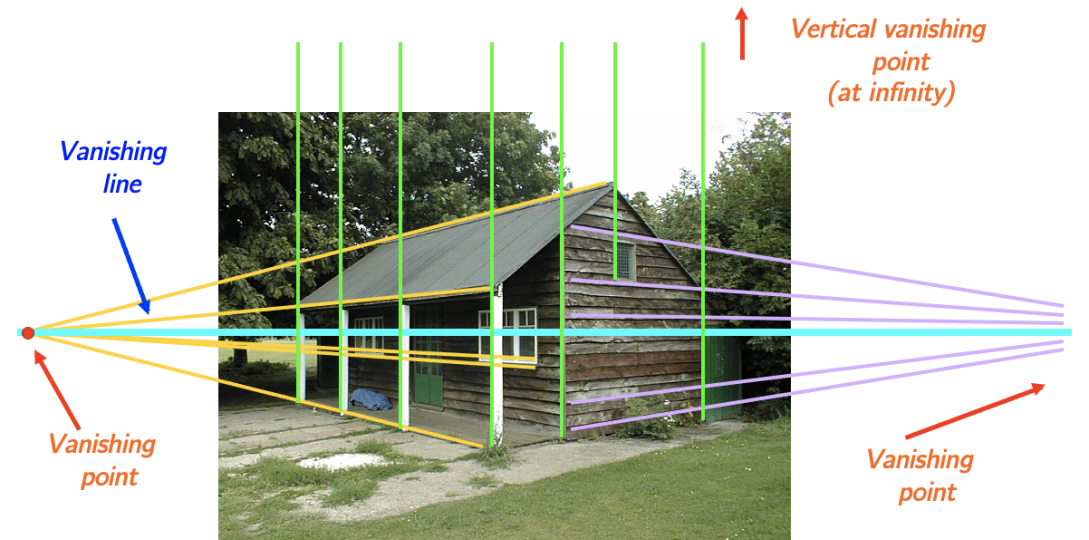


8. If a set of parallel 3D lines are also parallel to a particular plane, their vanishing point will lie on the vanishing line of the plane

Same proof as 6.

Properties of Vanishing Points & Lines

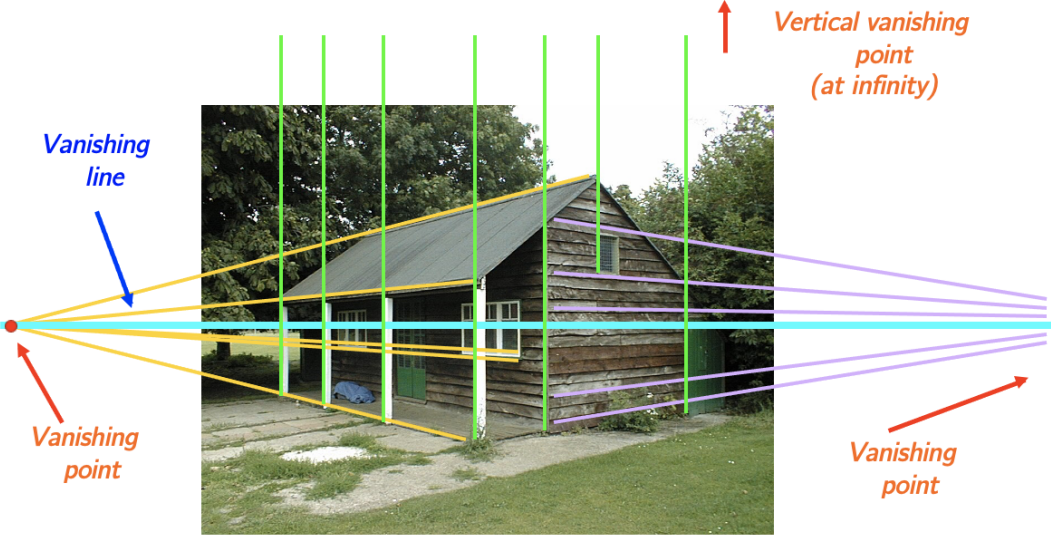
- The projections of parallel 3D **lines** intersect at a vanishing **point**
- The projection of parallel 3D **planes** intersect at a vanishing **line**
- Vanishing point \leftrightarrow 3D direction of a line
- Vanishing line \leftrightarrow 3D orientation of a surface



Vanishing Points are a key perspective tool:
from making realistic drawings, to measuring
3D from 2D, and even for camera calibration!

How do you calculate vanishing points?

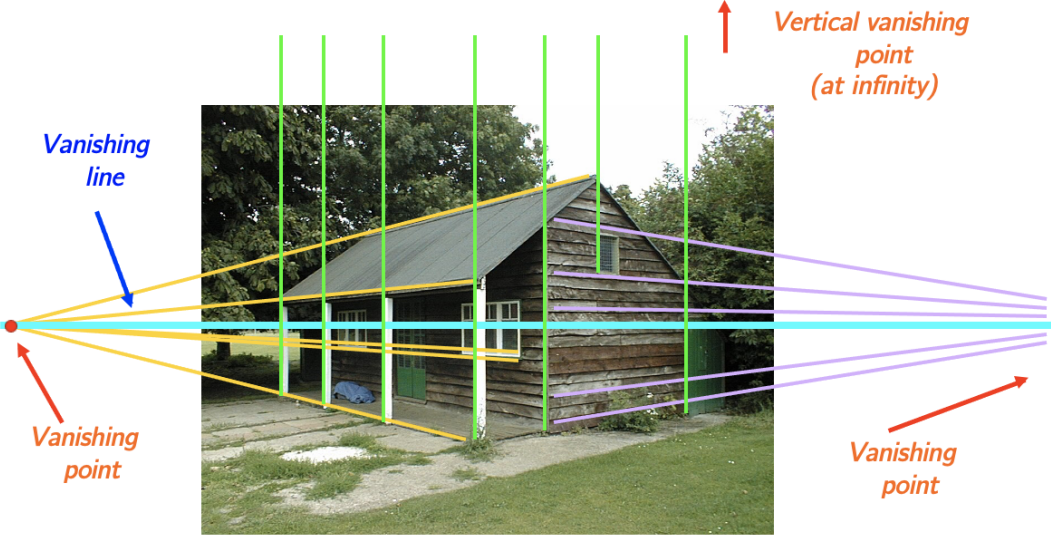
Q. What do you need first?



How do you calculate vanishing points?

Lines!

Q. How do you get the lines?



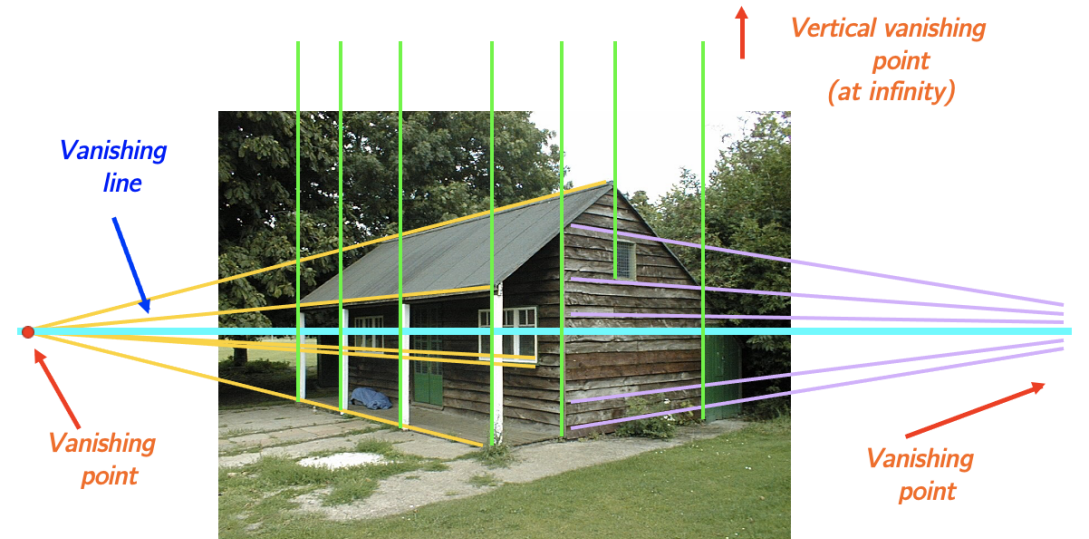
How do you calculate vanishing points?

Lines!

First calculate edges using Canny Edge Detector.

Then use RANSAC or Hough transforms to get the lines.

Q. Once you have the lines, how do you find the vanishing points?



How do you calculate vanishing points?

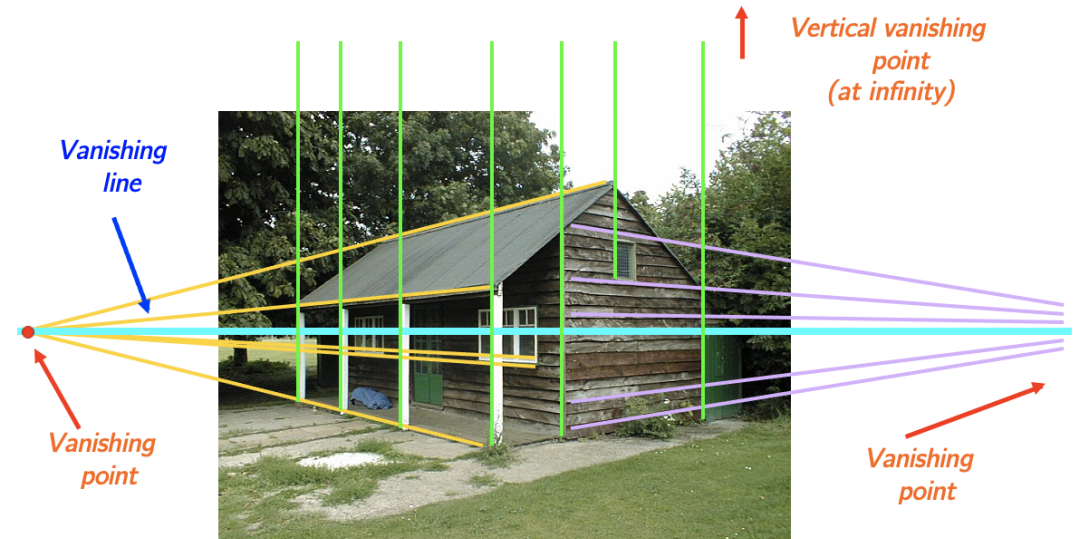
Lines!

First calculate edges using Canny Edge Detector.

Then use RANSAC or Hough transforms to get the lines.

Q. Once you have the lines, how do you find the vanishing points?

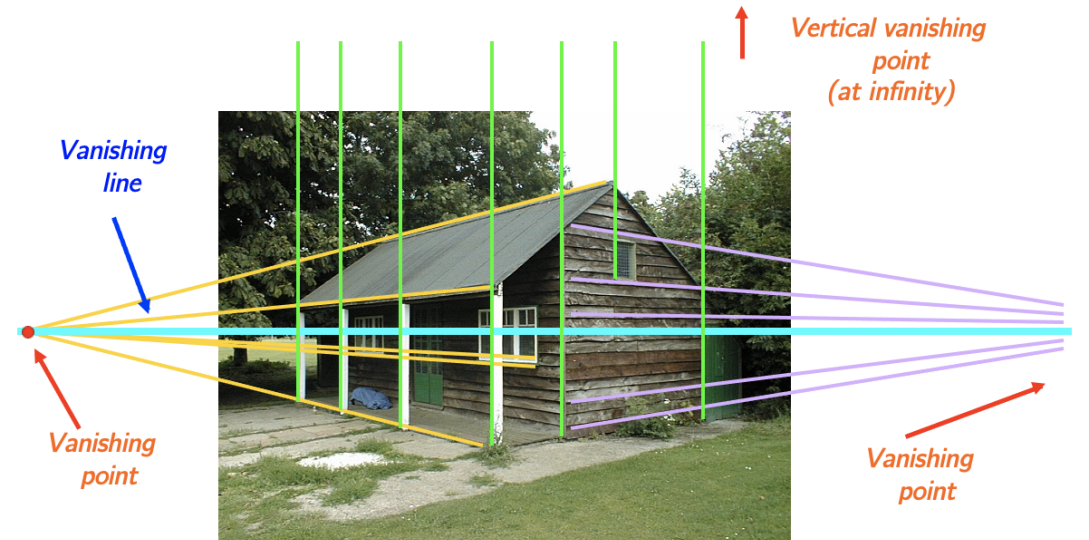
RANSAC again, but in pixel space to count the intersections!



What can you do with the vanishing points?

You can calculate camera intrinsics.

Let v_1, v_2 be the *homogeneous* coordinates two distinct vanishing points.



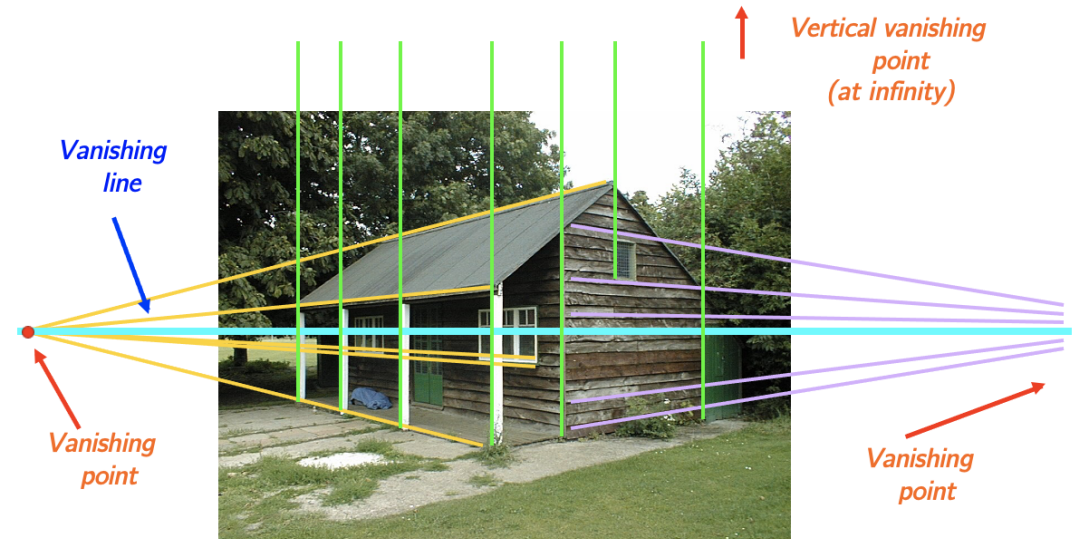
Calculating the lines from the vanishing points (assuming we already know intrinsics K)

You can calculate camera intrinsics.

Let $\mathbf{v}_1, \mathbf{v}_2$ be the *homogeneous* coordinates two distinct vanishing points.

Then the 3D directions of these lines—
after undoing the intrinsics—are:

$$\mathbf{d}_1 \propto K^{-1}\mathbf{v}_1, \quad \mathbf{d}_2 \propto K^{-1}\mathbf{v}_2.$$



Calculating the lines from the vanishing points (*assuming we already know intrinsics K*)

You can calculate camera intrinsics.

Let $\mathbf{v}_1, \mathbf{v}_2$ be the *homogeneous* coordinates two distinct vanishing points.

Then the 3D directions of these lines—after undoing the intrinsics—are:

$$\mathbf{d}_1 \propto K^{-1}\mathbf{v}_1, \quad \mathbf{d}_2 \propto K^{-1}\mathbf{v}_2.$$

Because these directions are orthogonal in 3D, we have: $\mathbf{d}_1^\top \mathbf{d}_2 = 0$.

Calculating the lines from the vanishing points (*assuming we already know intrinsics K*)

You can calculate camera intrinsics.

Let $\mathbf{v}_1, \mathbf{v}_2$ be the *homogeneous* coordinates two distinct vanishing points.

Then the 3D directions of these lines—after undoing the intrinsics—are:

$$\mathbf{d}_1 \propto K^{-1}\mathbf{v}_1, \quad \mathbf{d}_2 \propto K^{-1}\mathbf{v}_2.$$

Because these directions are orthogonal in 3D, we have: $\mathbf{d}_1^\top \mathbf{d}_2 = 0$.

Substituting $\mathbf{d}_i = K^{-1}\mathbf{v}_i$, we get:

$$(K^{-1}\mathbf{v}_1)^\top (K^{-1}\mathbf{v}_2) = \mathbf{v}_1^\top (K^{-1})^\top (K^{-1}) \mathbf{v}_2 = 0$$

Calculating the lines from the vanishing points (assuming we already know intrinsics K)

You can calculate camera intrinsics.

Let $\mathbf{v}_1, \mathbf{v}_2$ be the *homogeneous* coordinates two distinct vanishing points.

Then the 3D directions of these lines—after undoing the intrinsics—are:

$$\mathbf{d}_1 \propto K^{-1}\mathbf{v}_1, \quad \mathbf{d}_2 \propto K^{-1}\mathbf{v}_2.$$

Because these directions are orthogonal in 3D, we have: $\mathbf{d}_1^\top \mathbf{d}_2 = 0$.

Substituting $\mathbf{d}_i = K^{-1}\mathbf{v}_i$, we get:

$$(K^{-1}\mathbf{v}_1)^\top (K^{-1}\mathbf{v}_2) = \mathbf{v}_1^\top (K^{-1})^\top (K^{-1}) \mathbf{v}_2 = 0$$

K has only 3 values:
$$\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Each pair of vanishing points gives us 1 equation. We need at least 3 pairs

Calculating the lines from the vanishing points (*assuming we already know intrinsics K*)

You can calculate camera intrinsics.

Because these directions are orthogonal in 3D, we have $\mathbf{v}_1 \cdot \mathbf{v}_2 = 0$.

We will use this method in your assignments. We will explore another way in lecture to calculate camera intrinsics

after finding the intrinsics K are:

$$\mathbf{d}_1 \propto K^{-1}\mathbf{v}_1, \quad \mathbf{d}_2 \propto K^{-1}\mathbf{v}_2.$$

K has only 3 values:

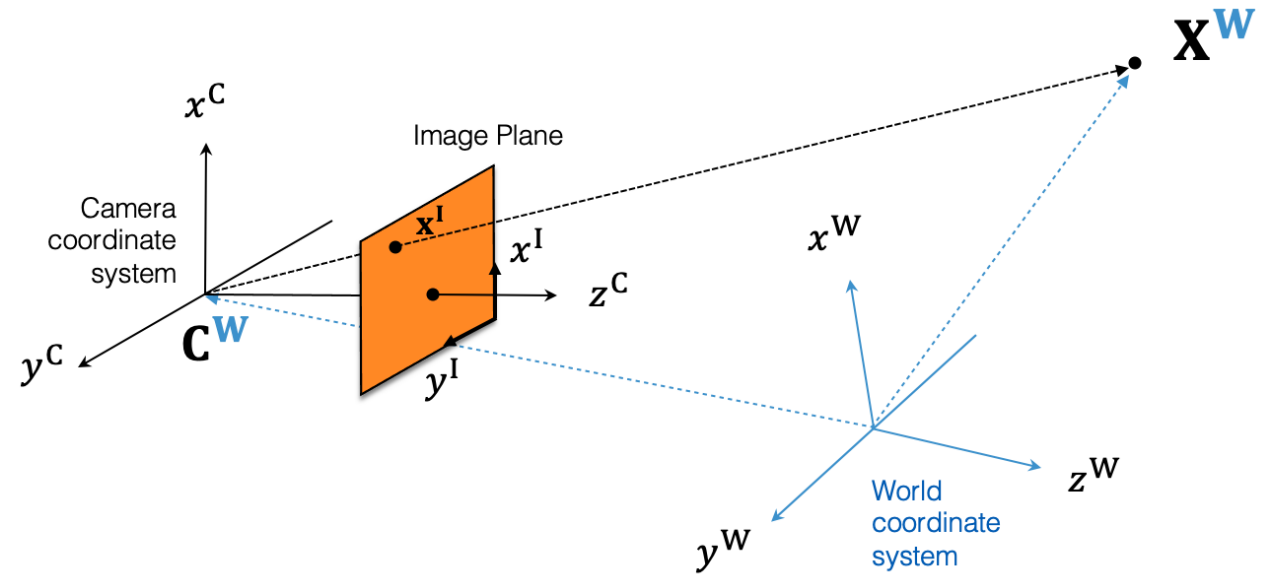
$$\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Each pair of vanishing points gives us 1 equation. We need at least 3 pairs

Today's agenda

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

Recap: The Pinhole Camera Model



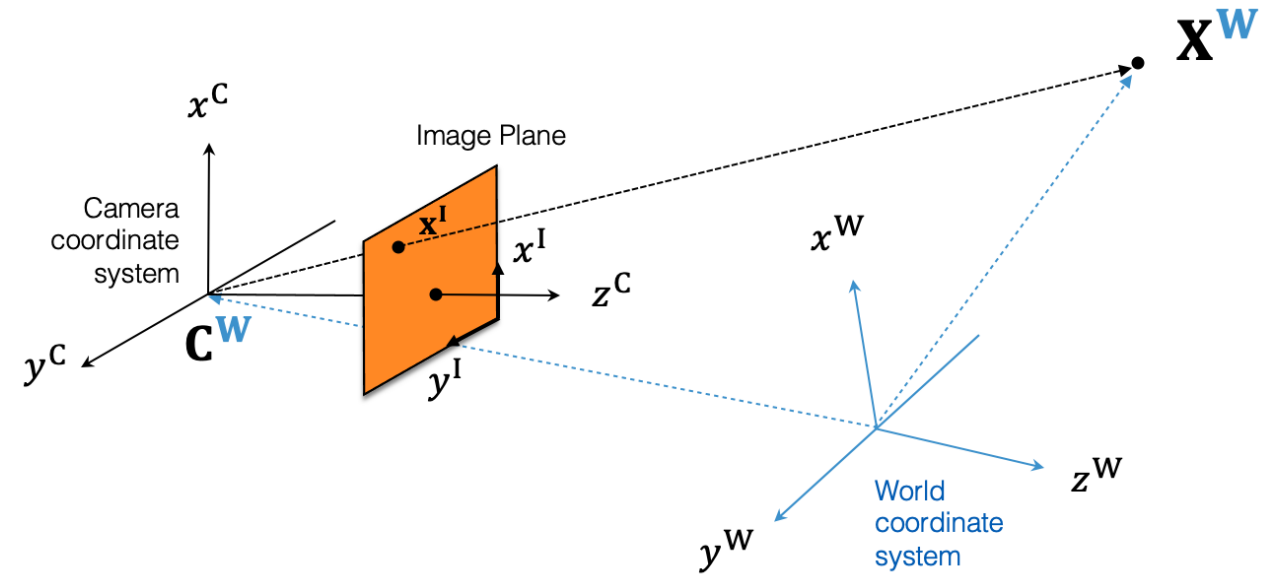
$$\tilde{\mathbf{x}}^I \sim \mathbf{P} \tilde{\mathbf{X}}^W \quad \mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & | & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{R} & -\mathbf{R}\mathbf{C} \\ \mathbf{0} & 1 \end{bmatrix} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$

intrinsic parameters \mathbf{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



$$\tilde{\mathbf{x}}^I \sim \mathbf{P} \tilde{\mathbf{X}}^W \quad \mathbf{P} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]$$

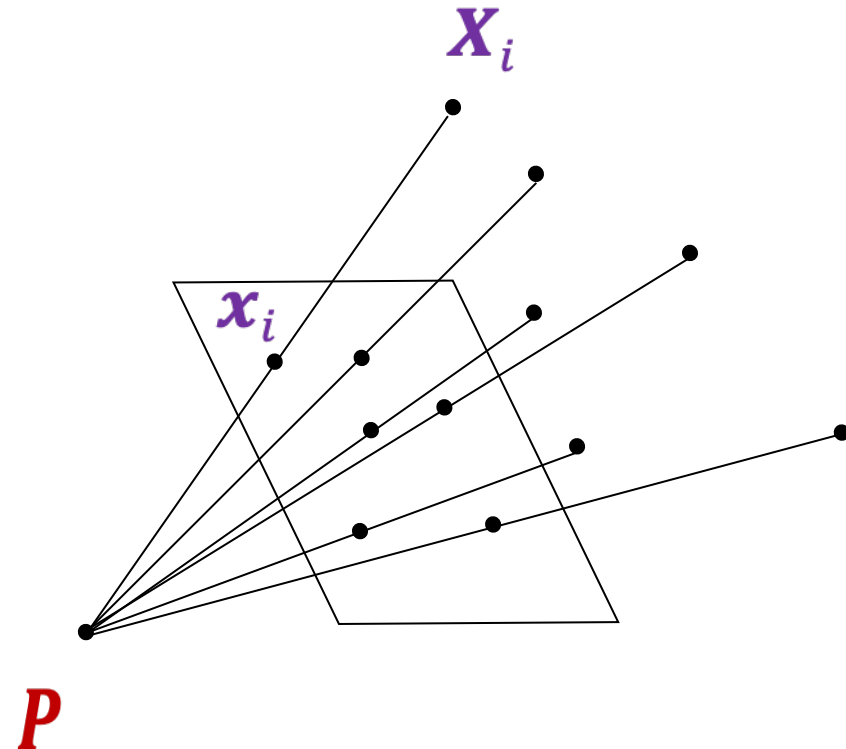
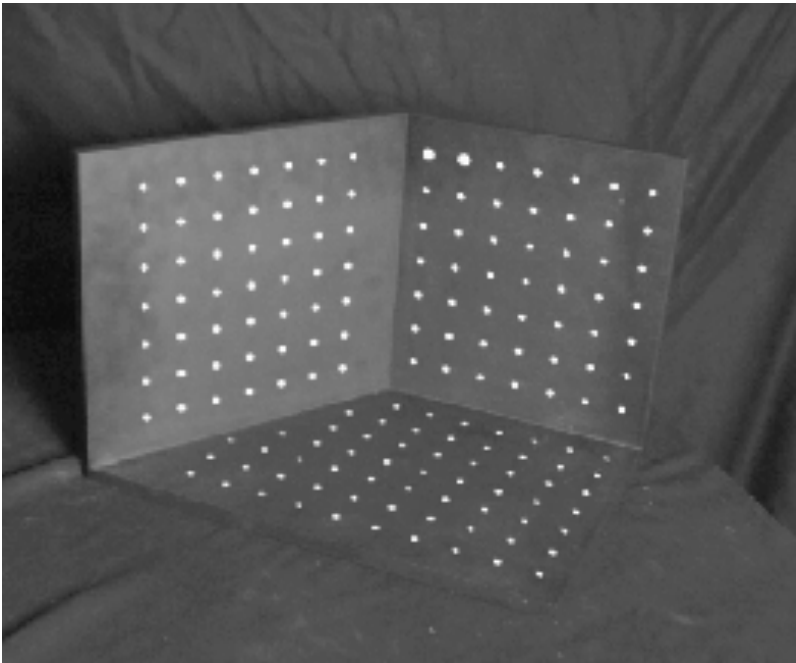
How can we estimate \mathbf{P} and its components?

Camera Calibration: estimating *intrinsics* \mathbf{K}

Pose Estimation: estimating *extrinsics* $[\mathbf{R} \mid \mathbf{t}]$

Camera calibration from 3D-2D correspondences

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



Camera calibration from 3D-2D correspondences

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



Known 2D image
coords

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

Known 3D locations

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

Camera calibration from 3D-2D correspondences

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

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

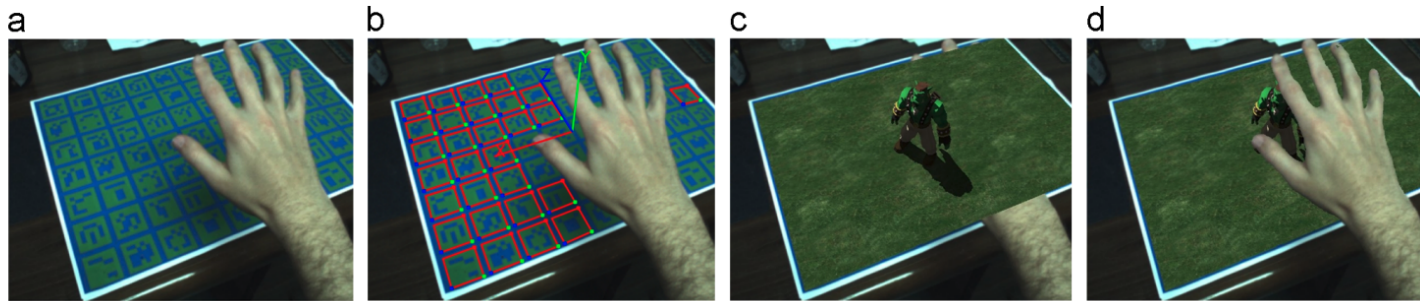


Fig. 1. Example of augmented reality scene. (a) Input image containing a set of fiducial markers. (b) Markers automatically detected and used for camera pose estimation. (c) Augmented scene without considering user's occlusion. (d) Augmented scene considering occlusion.

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

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

Heterogeneous coordinates

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

(non-linear relationship between coordinates)

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

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
- Depth estimation

How can we make these relations linear?

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

Make them linear with algebraic manipulation...

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

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

$$\mathbf{p}_1^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} x' = 0$$

$$\mathbf{p}_2^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} y' = 0$$

In matrix form ...

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

Q1. How many equations does each correspondence give us?

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

$$\mathbf{p}_1^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} x' = 0$$

$$\mathbf{p}_2^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} y' = 0$$

In matrix form for 1 corresponding point ...

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

For N points ...

$$\begin{bmatrix} \mathbf{X}_1^T & \mathbf{0} & -x'_1 \mathbf{X}_1^T \\ \mathbf{0} & \mathbf{X}_1^T & -y'_1 \mathbf{X}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{X}_N^T & \mathbf{0} & -x'_N \mathbf{X}_N^T \\ \mathbf{0} & \mathbf{X}_N^T & -y'_N \mathbf{X}_N^T \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} = \mathbf{0}$$

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 ...

$$\begin{bmatrix} \mathbf{X}_1^T & 0 & -x'_1 \mathbf{X}_1^T \\ \mathbf{0} & \mathbf{X}_1^T & -y'_1 \mathbf{X}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{X}_N^T & 0 & -x'_N \mathbf{X}_N^T \\ \mathbf{0} & \mathbf{X}_N^T & -y'_N \mathbf{X}_N^T \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} = 0$$

How do we solve this system?

Solve for camera matrix via total least squares

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

$$\mathbf{A} = \begin{bmatrix} \mathbf{X}_1^T & 0 & -x'_1 \mathbf{X}_1^T \\ \mathbf{0} & \mathbf{X}_1^T & -y'_1 \mathbf{X}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{X}_N^T & 0 & -x'_N \mathbf{X}_N^T \\ \mathbf{0} & \mathbf{X}_N^T & -y'_N \mathbf{X}_N^T \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Singular Value Decomposition!

Singular Value Decomposition (SVD)

- Represents any matrix **A** as a product of three matrices: **UΣV^T**
- Python command:
 - `[U,Σ,V]= numpy.linalg.svd(A)`

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

Singular Value Decomposition (SVD)

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

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

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

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

Solve for camera matrix via total least squares

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

$$\mathbf{A} = \begin{bmatrix} \mathbf{X}_1^T & 0 & -x'_1 \mathbf{X}_1^T \\ \mathbf{0} & \mathbf{X}_1^T & -y'_1 \mathbf{X}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{X}_N^T & 0 & -x'_N \mathbf{X}_N^T \\ \mathbf{0} & \mathbf{X}_N^T & -y'_N \mathbf{X}_N^T \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ Solution \mathbf{x} is the **column of \mathbf{V}** corresponding to the smallest singular value of \mathbf{A}

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

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

Is equivalent to:

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

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

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

Is equivalent to:

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

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

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

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

Is equivalent to:

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

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

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

So, $\mathbf{A}^T \mathbf{A} \mathbf{v}_k$ 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

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

$$\mathbf{A} = \begin{bmatrix} \mathbf{X}_1^T & 0 & -x'_1 \mathbf{X}_1^T \\ \mathbf{0} & \mathbf{X}_1^T & -y'_1 \mathbf{X}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{X}_N^T & 0 & -x'_N \mathbf{X}_N^T \\ \mathbf{0} & \mathbf{X}_N^T & -y'_N \mathbf{X}_N^T \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ Equivalently, solution \mathbf{x} is the Eigenvector corresponding to the smallest Eigenvalue of \mathbf{A}

We calculated the camera matrix!

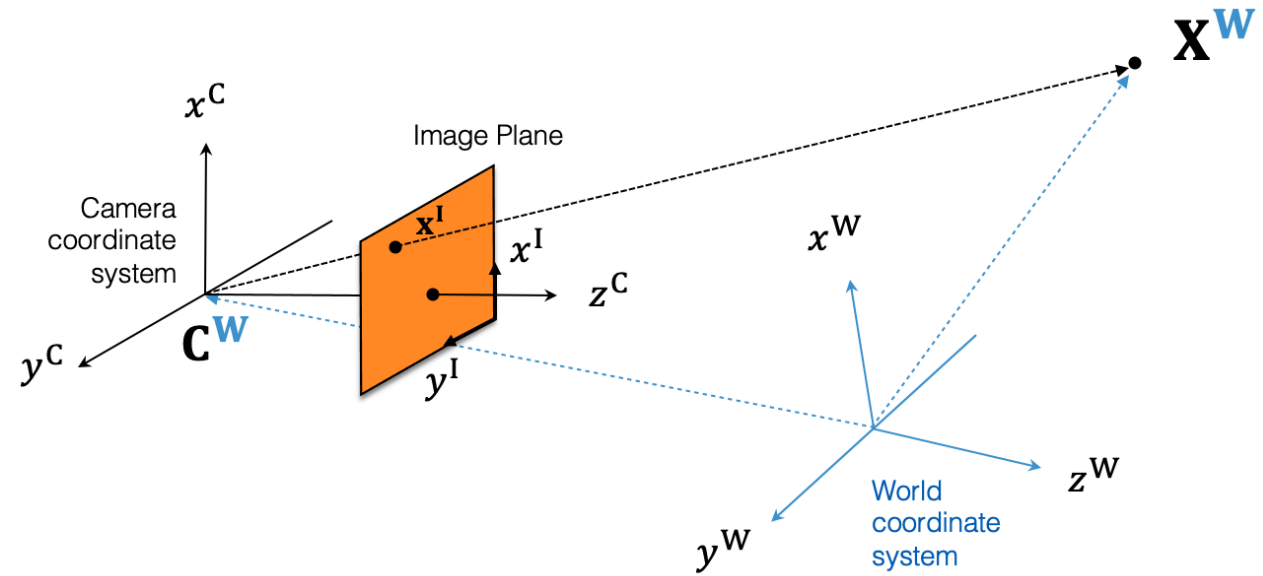
Now we have:

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

Today's agenda

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

Recap: The Pinhole Camera Model



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

intrinsic parameters \mathbf{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 \mathbf{C} ?

Decomposition of the Camera Matrix

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

$$\bar{\mathbf{P}} = \left[\begin{array}{ccc} p_1 & p_2 & p_3 \\ p_5 & p_6 & p_7 \\ p_9 & p_{10} & p_{11} \end{array} \right] \sim \mathbf{KR}$$

Q2. Is there a way we can get rid of \mathbf{R} ?

Decomposition of the Camera Matrix

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

$$\bar{\mathbf{P}} = \left[\begin{array}{ccc} p_1 & p_2 & p_3 \\ p_5 & p_6 & p_7 \\ p_9 & p_{10} & p_{11} \end{array} \right] \sim \mathbf{KR}$$

$$\bar{\mathbf{P}}\bar{\mathbf{P}}^T \sim \mathbf{KRR}^T\mathbf{K}^T$$

$$\bar{\mathbf{P}}\bar{\mathbf{P}}^T \sim \mathbf{KK}^T$$

$$(\mathbf{RR}^T = \mathbf{I})$$

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

Decomposition of the Camera Matrix

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

$$\bar{\mathbf{P}} = \left[\begin{array}{ccc} p_1 & p_2 & p_3 \\ p_5 & p_6 & p_7 \\ p_9 & p_{10} & p_{11} \end{array} \right] \sim \mathbf{KR}$$

$$\bar{\mathbf{P}}\bar{\mathbf{P}}^T \sim \mathbf{KRR}^T\mathbf{K}^T$$

$$\bar{\mathbf{P}}\bar{\mathbf{P}}^T \sim \mathbf{KK}^T$$

$$(\mathbf{RR}^T = \mathbf{I})$$

K is upper triangular and positive definite!

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

Decomposition of the Camera Matrix

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

$\bar{\mathbf{P}}^T \bar{\mathbf{P}} \sim \mathbf{K}^T \mathbf{K}$ with \mathbf{K} upper triangular p.d.

Obtain \mathbf{K} by [Cholesky decomposition](#) of $\bar{\mathbf{P}}^T \bar{\mathbf{P}} = \mathbf{L}\mathbf{L}^T$
 $\mathbf{K} \sim \mathbf{L}^T$

Scalar factor: fixed to $1/L_{3,3}$ ($K_{3,3} = 1$)

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

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

Decomposition of the Camera Matrix

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

$\bar{\mathbf{P}}^T \bar{\mathbf{P}} \sim \mathbf{K}^T \mathbf{K}$ with \mathbf{K} upper triangular p.d.

Obtain \mathbf{K} by [Cholesky decomposition](#) of $\bar{\mathbf{P}}^T \bar{\mathbf{P}} = \mathbf{L}\mathbf{L}^T$
 $\mathbf{K} \sim \mathbf{L}^T$

Scalar factor: fixed to $1/L_{3,3}$ ($K_{3,3} = 1$)

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

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

Scaling R

$$\text{Let } \mathbf{R} = \lambda \mathbf{K}^{-1} \bar{\mathbf{P}}$$

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

If A is an $n \times n$ matrix and λ is a scalar, then

$$\det(\lambda A) = \lambda^n \det(A).$$

Calculating C

$$\mathbf{P} = \left[\begin{array}{c|c} & \bar{\mathbf{P}} \\ \hline & \begin{bmatrix} p_4 \\ p_8 \\ p_{12} \end{bmatrix} \end{array} \right] \sim \mathbf{K} [\mathbf{R} \mid -\mathbf{R}\mathbf{C}]$$

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!

(out of scope for class) Non-Linear Camera Calibration

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

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

- 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!

Today's agenda

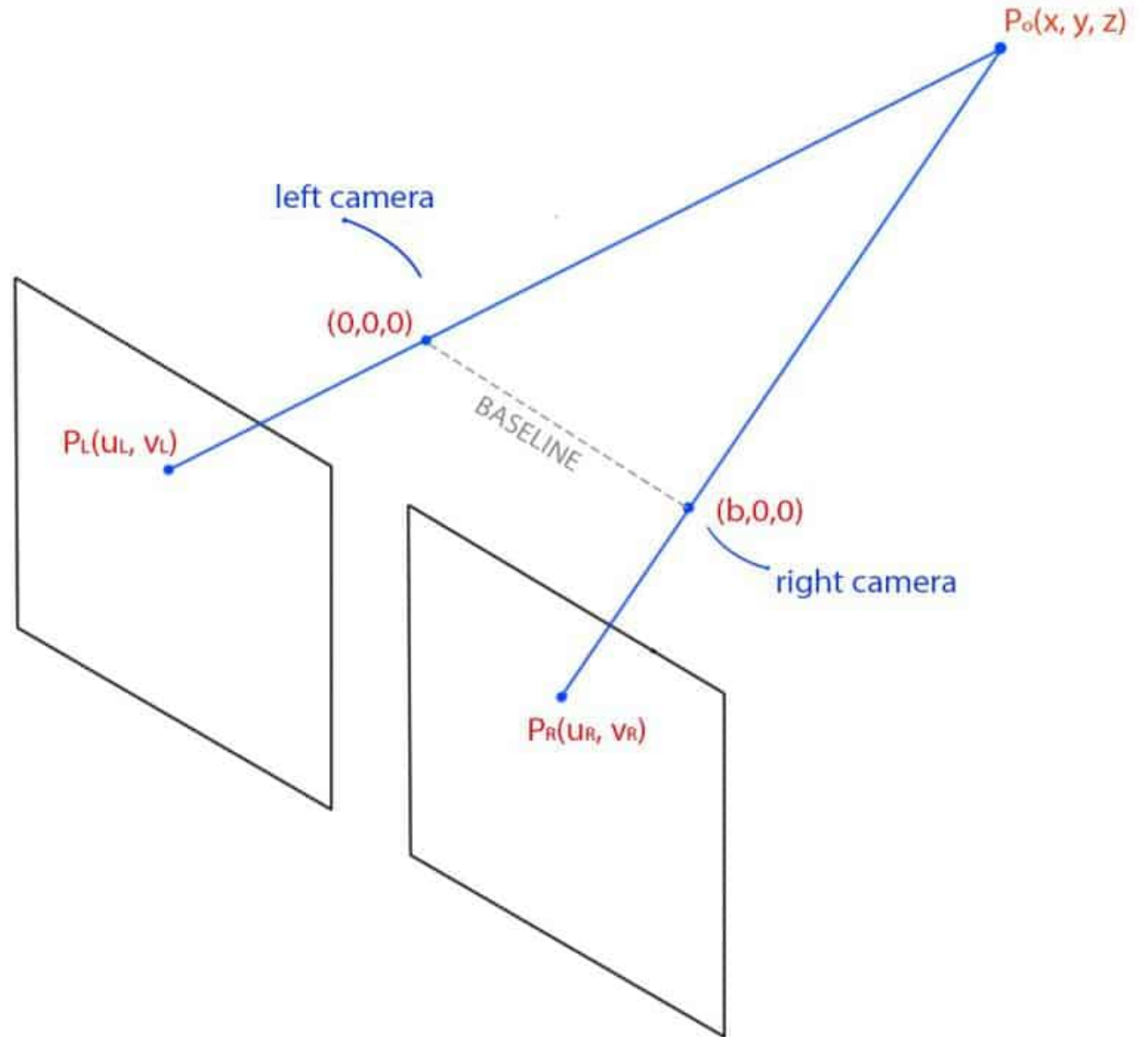
- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Depth estimation

Estimating depth

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

We want to estimate z

We know (u_L, v_L) and (u_R, v_R)



Estimating depth

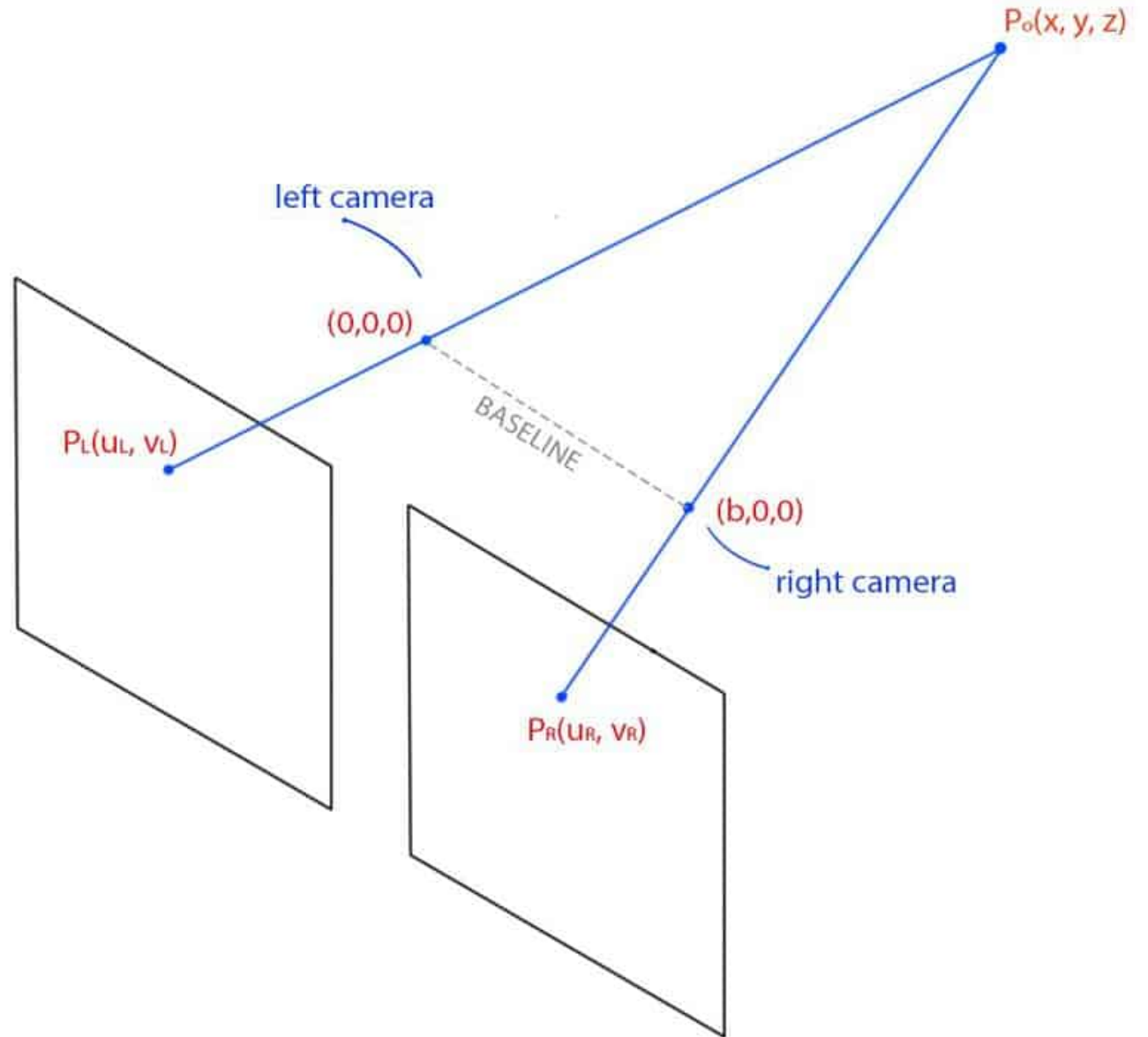
We can estimate the intrinsics of the camera from the previous sections

So, we know:

focal length: f_x, f_y

translation: p_x, p_y

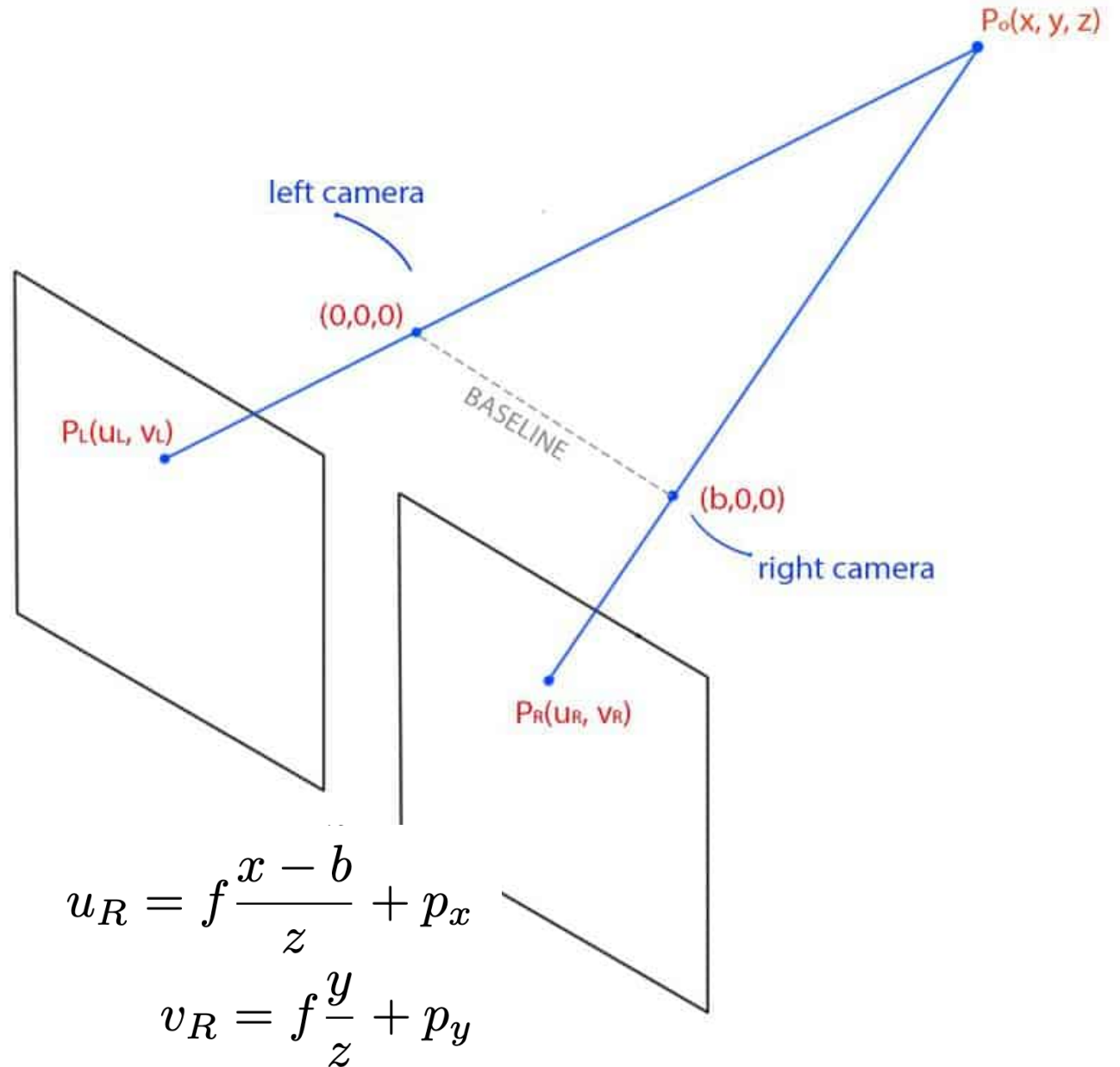
$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$



Estimating depth

$$u_L = f \frac{x}{z} + p_x$$
$$v_L = f \frac{y}{z} + p_y$$

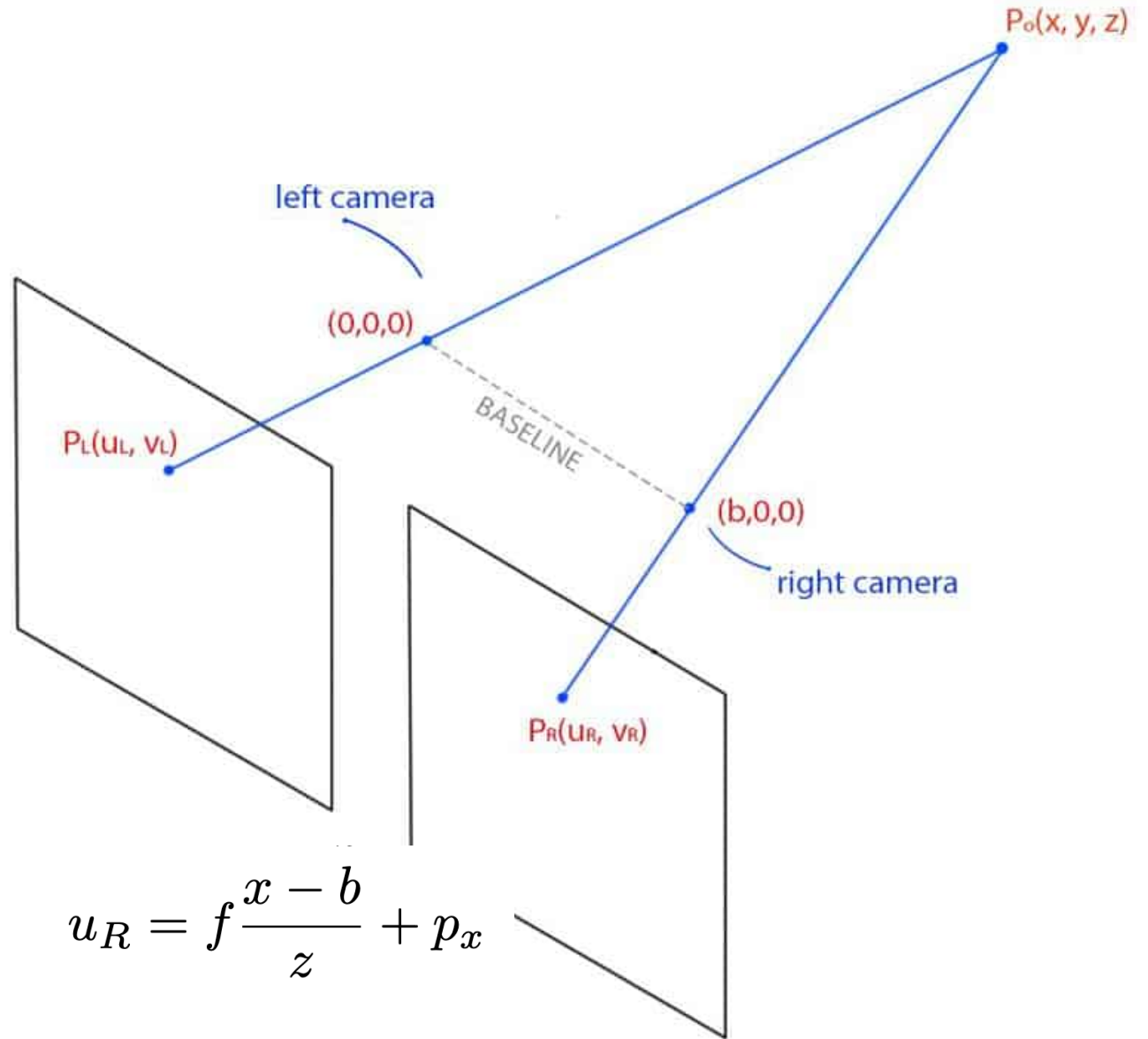
$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$



Estimating depth

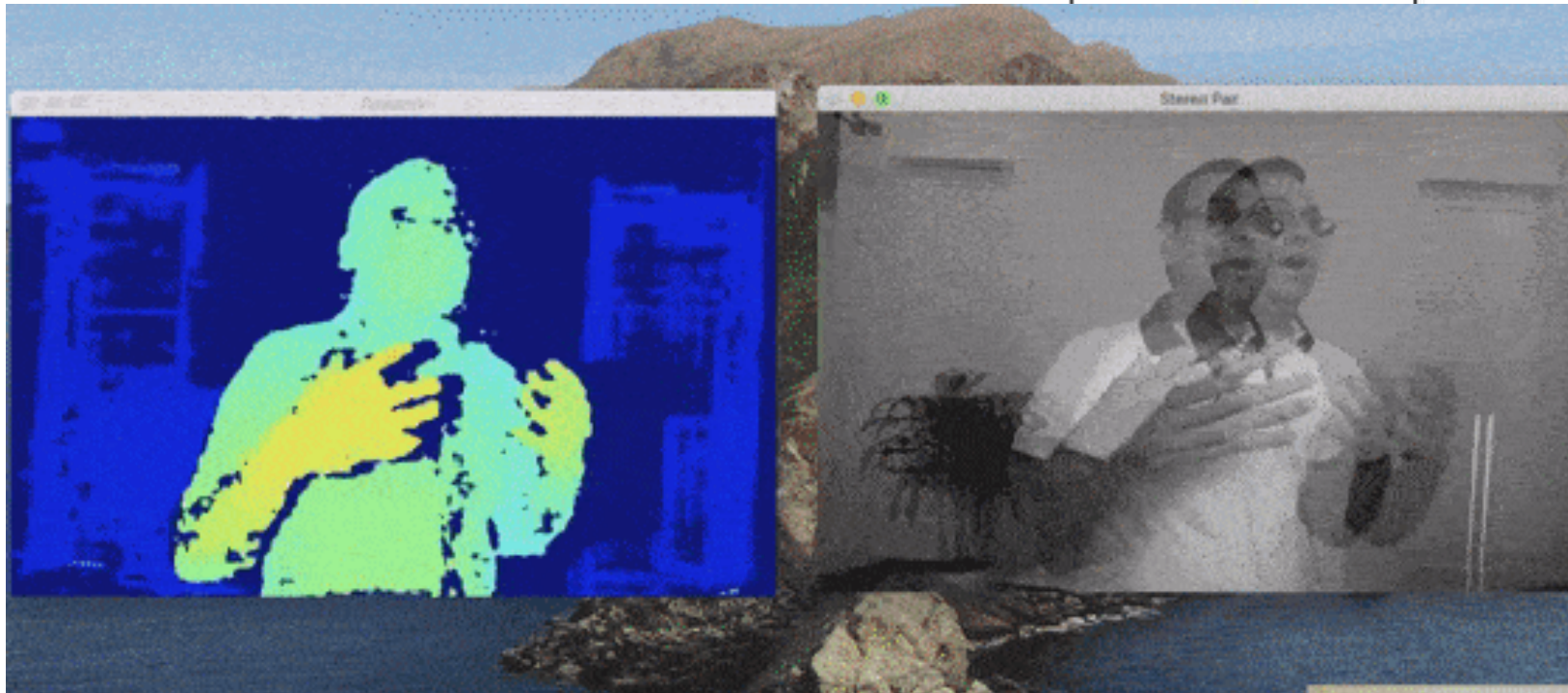
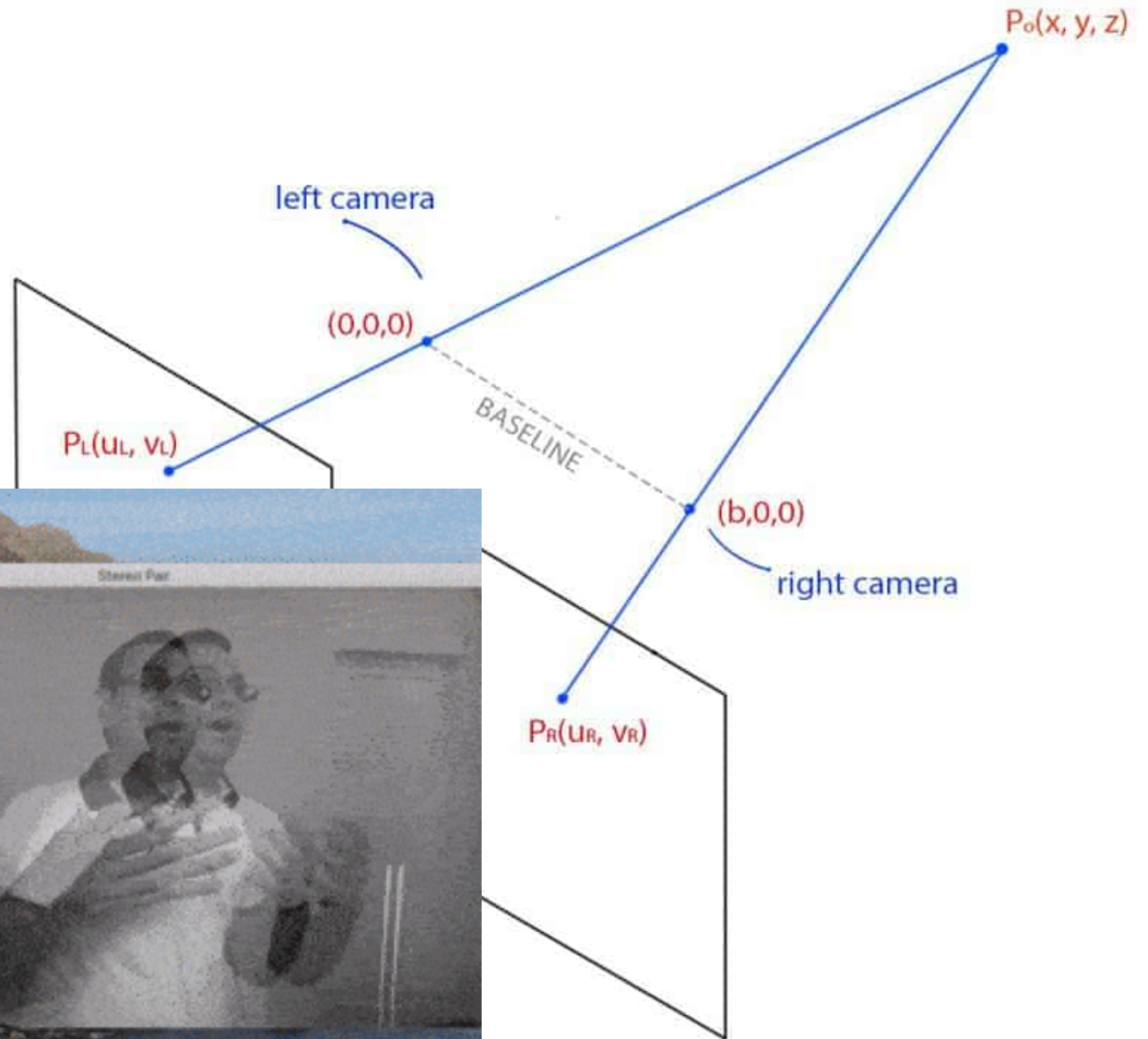
$$u_L = f \frac{x}{z} + p_x$$

$$z = \frac{fb}{u_L - u_R}$$



Estimating depth

$$z = \frac{fb}{u_L - u_R}$$



Today's agenda

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

Next lecture

Recognition