CSCI 4830 / 5722
Computer Vision

CSCI 4830 / 5722
Computer Vision

Dr. Ioana Fleming
Spring 2019
Lecture 5

Reminders

Submissions:

- Homework 1: Sat 1/26 at 6 pm

Readings:

- Szeliski Ch. 3
- P&F Ch. 1 (or 1&2 in 1st edition)
- Camera Calibration paper

Office hours calendar - on Moodle

Moodle (password: '*vision*')

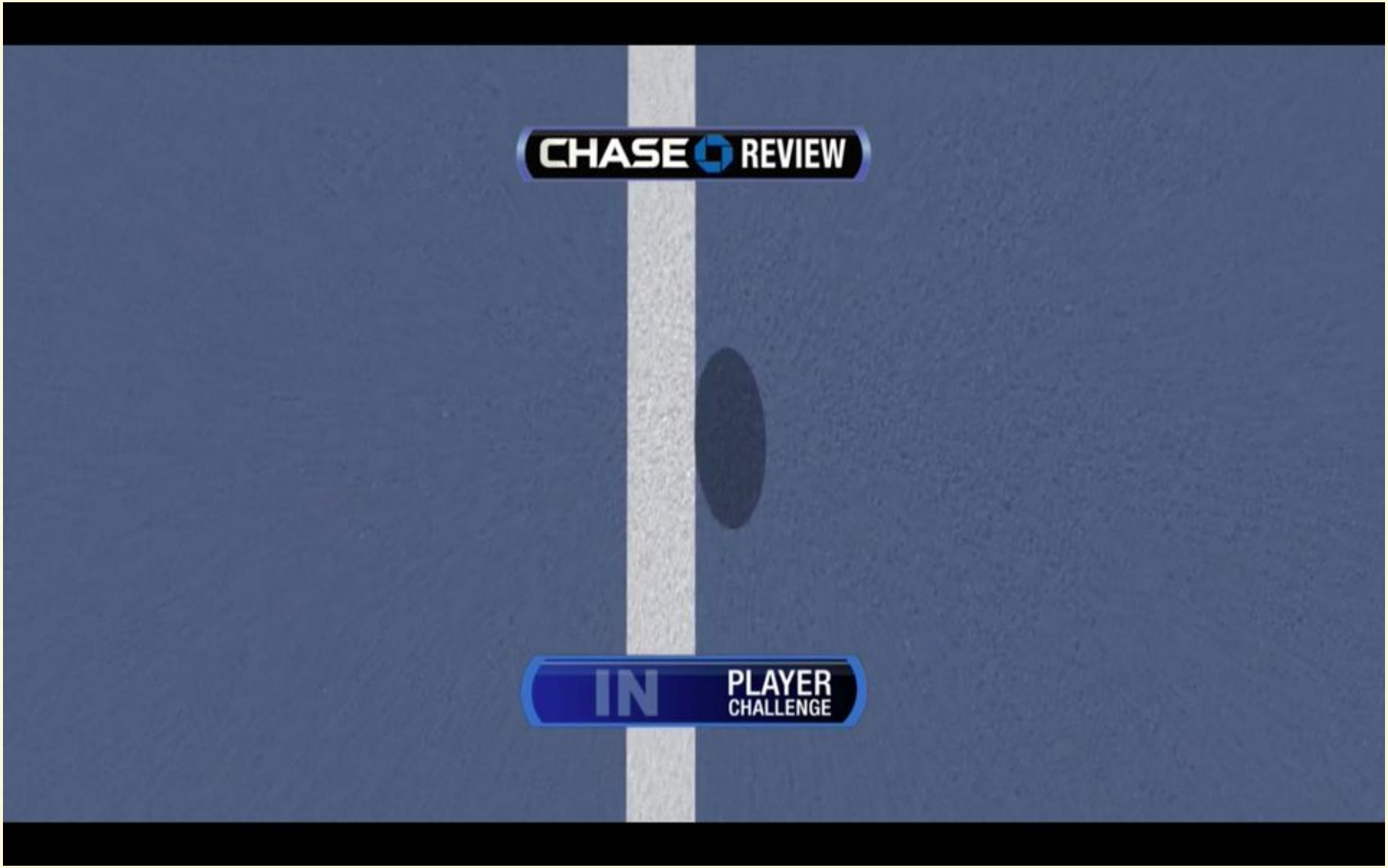
Today

- Mapping between image and world coordinates
  - Projective geometry
    - ***Vanishing points and lines***
  - Projection matrix

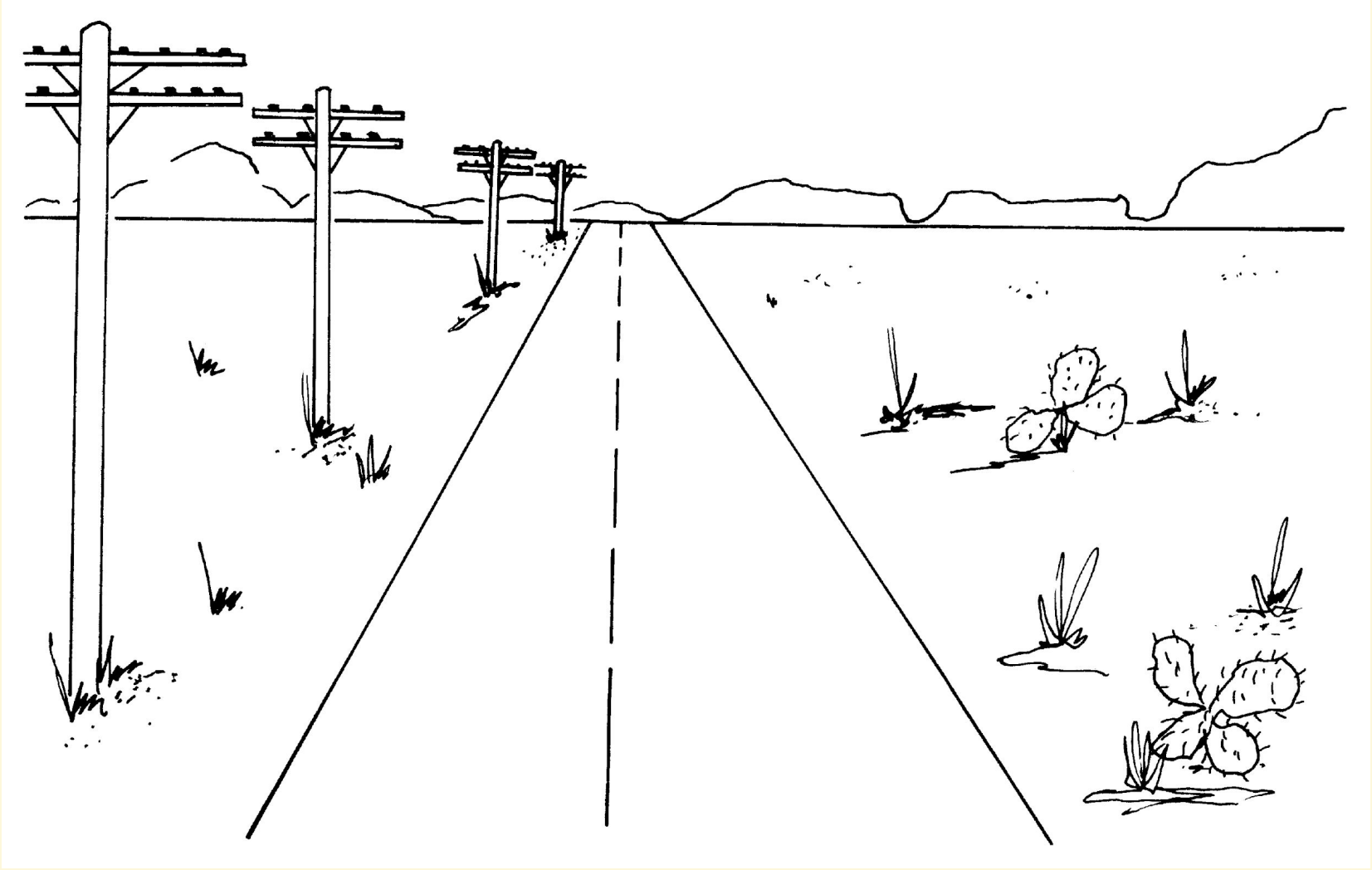
- Early Vision; one image
  - Linear Filters

Next Week

- Camera Calibration paper – posted on Moodle


CHASE REVIEW
IN
PLAYER
CHALLENGE

Some perspective phenomena…



Projective Geometry

What is lost?

Length is not preserved

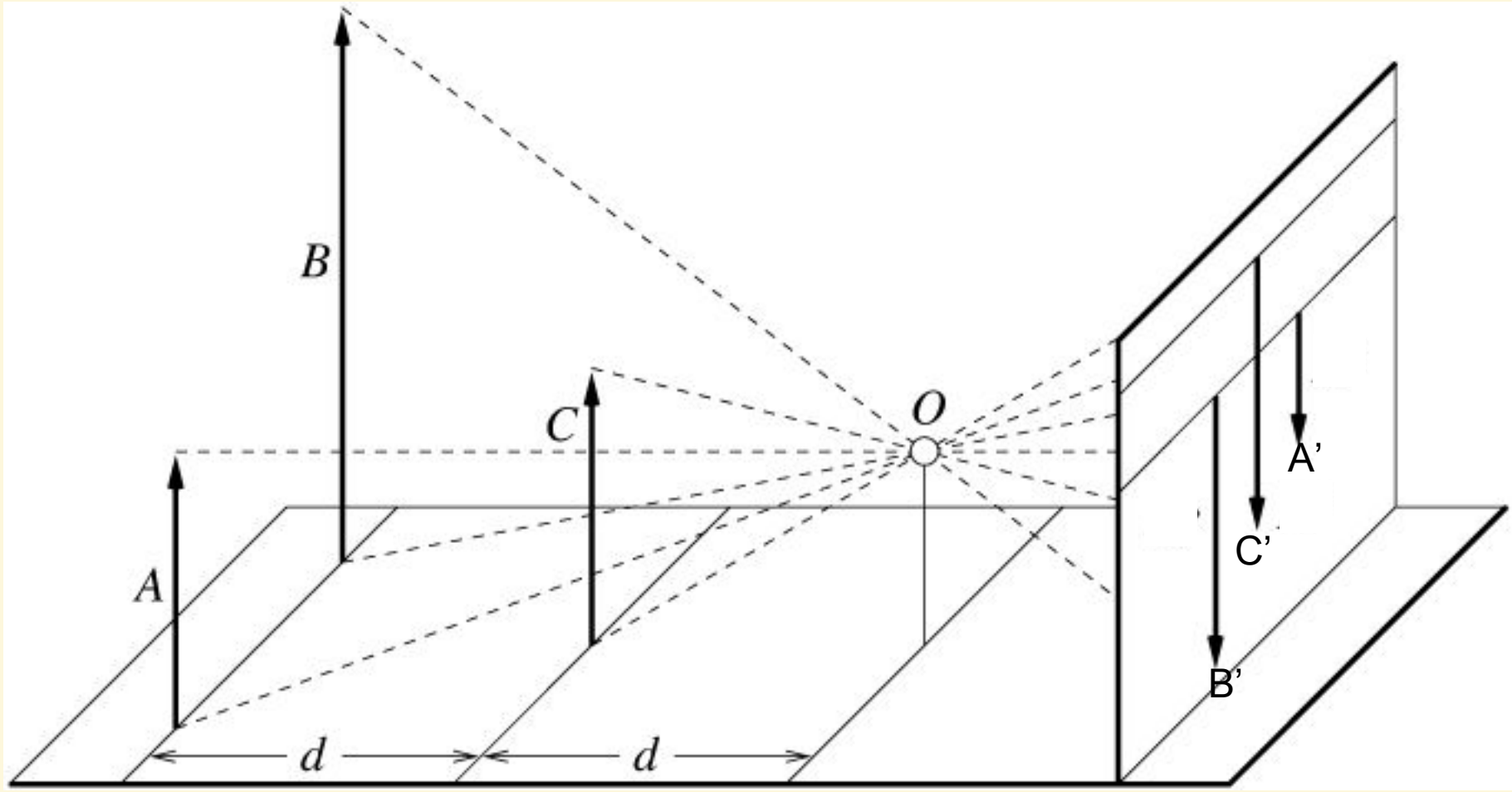


Figure by David Forsyth

Projective Geometry

What is lost?

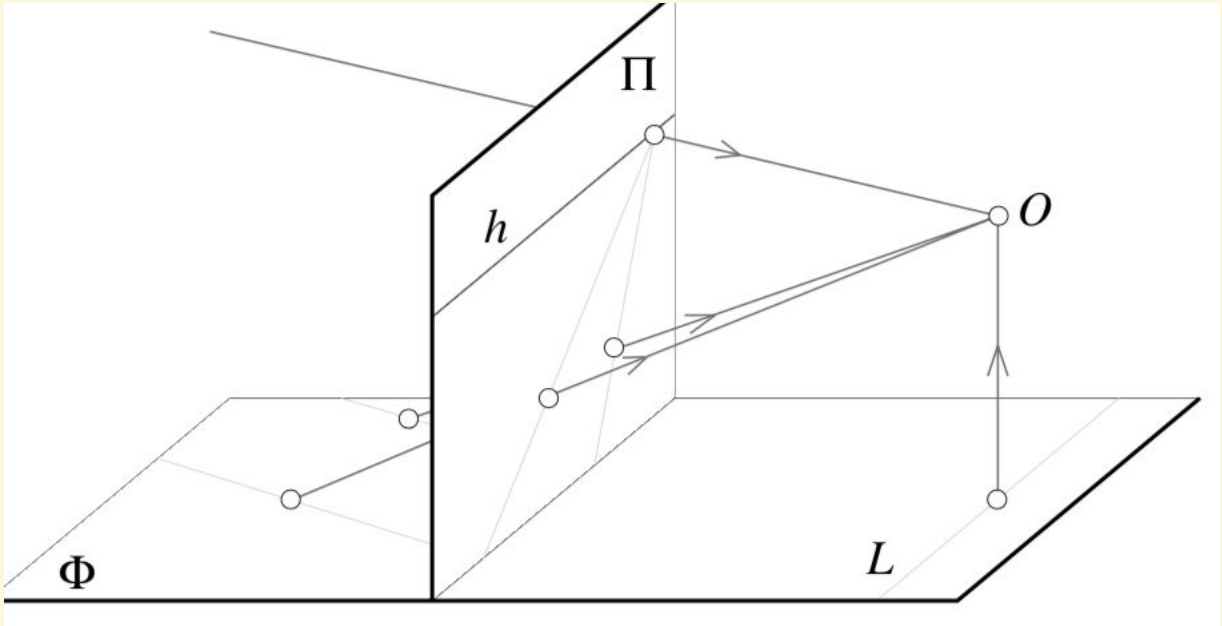
- Length
- Angles

Projective Geometry

What is preserved?

- Straight lines are still straight

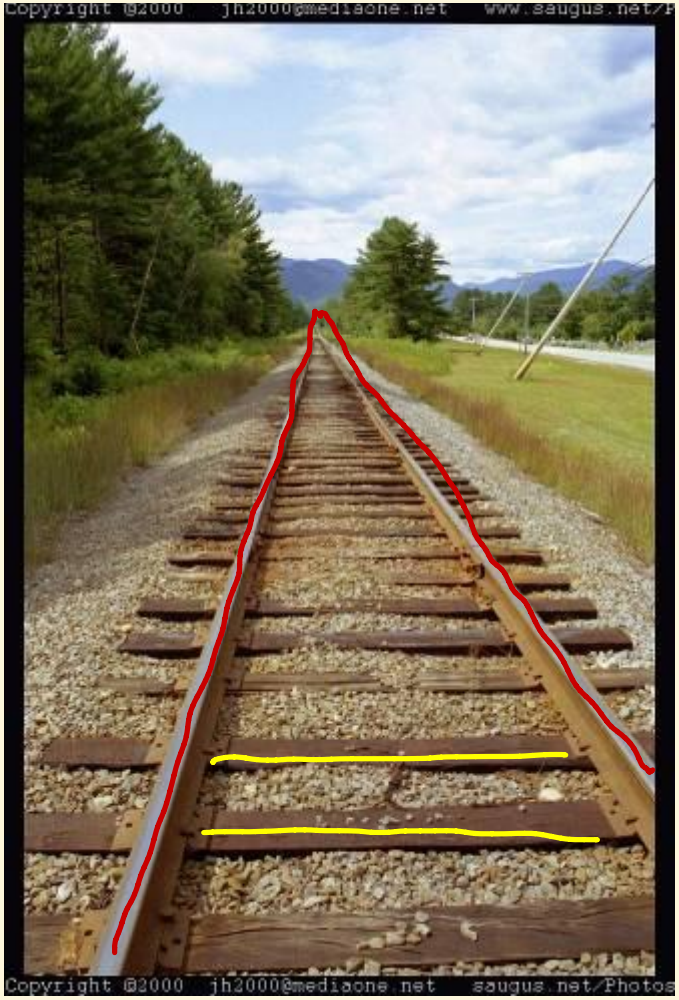
Parallel lines



The projections of two parallel lines lying in some plane Φ appear to converge on a horizon line h formed by the intersection of the image plane Π with the plane parallel to Φ and passing through the pinhole. Note that the line L parallel to Π in Φ has no image at all.

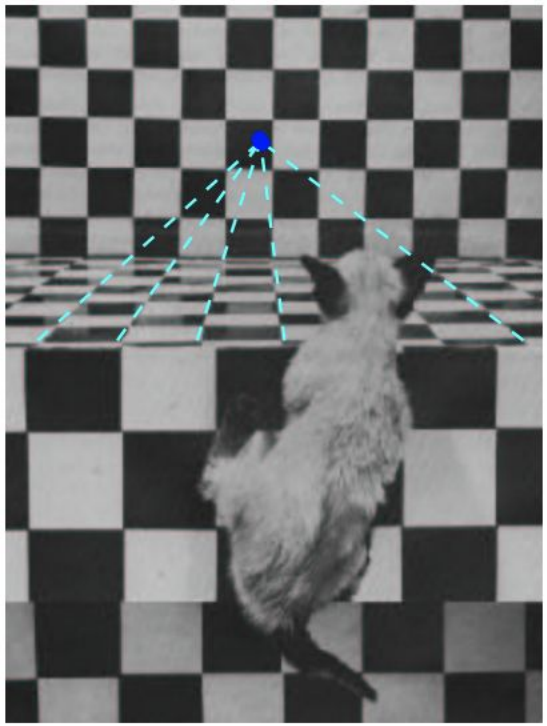
Vanishing points and lines

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

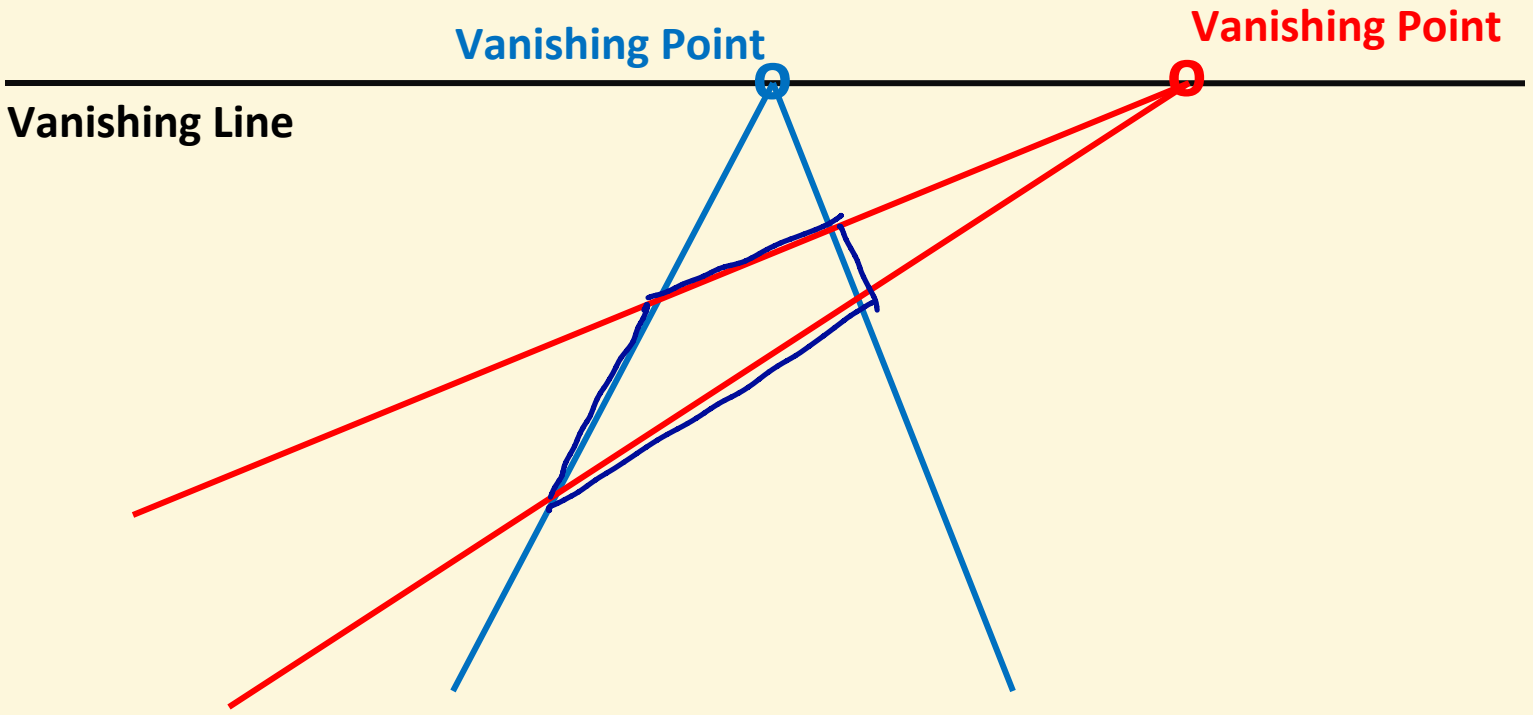


Vanishing points and lines

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



Vanishing points and lines

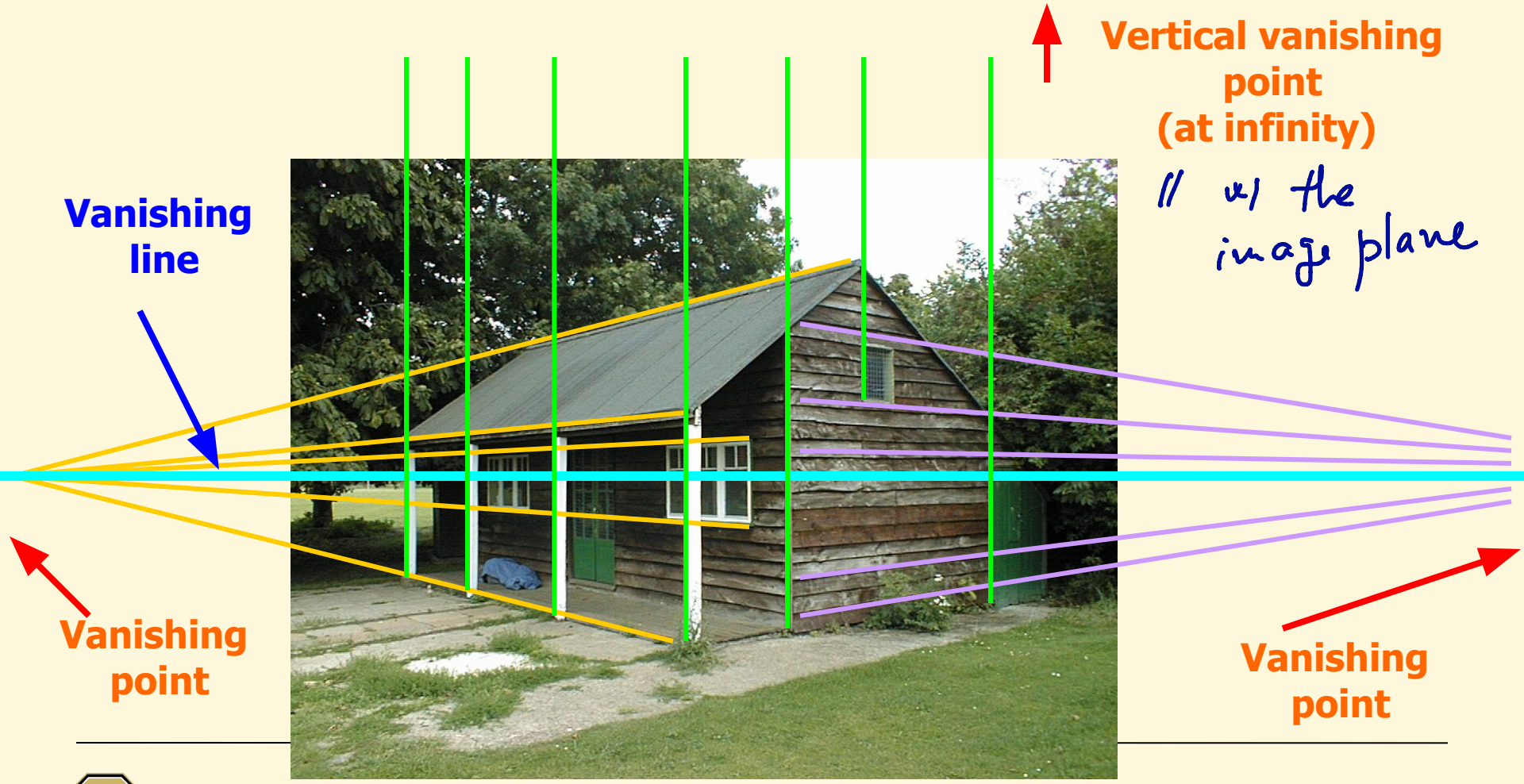


Vanishing points and lines

Properties:

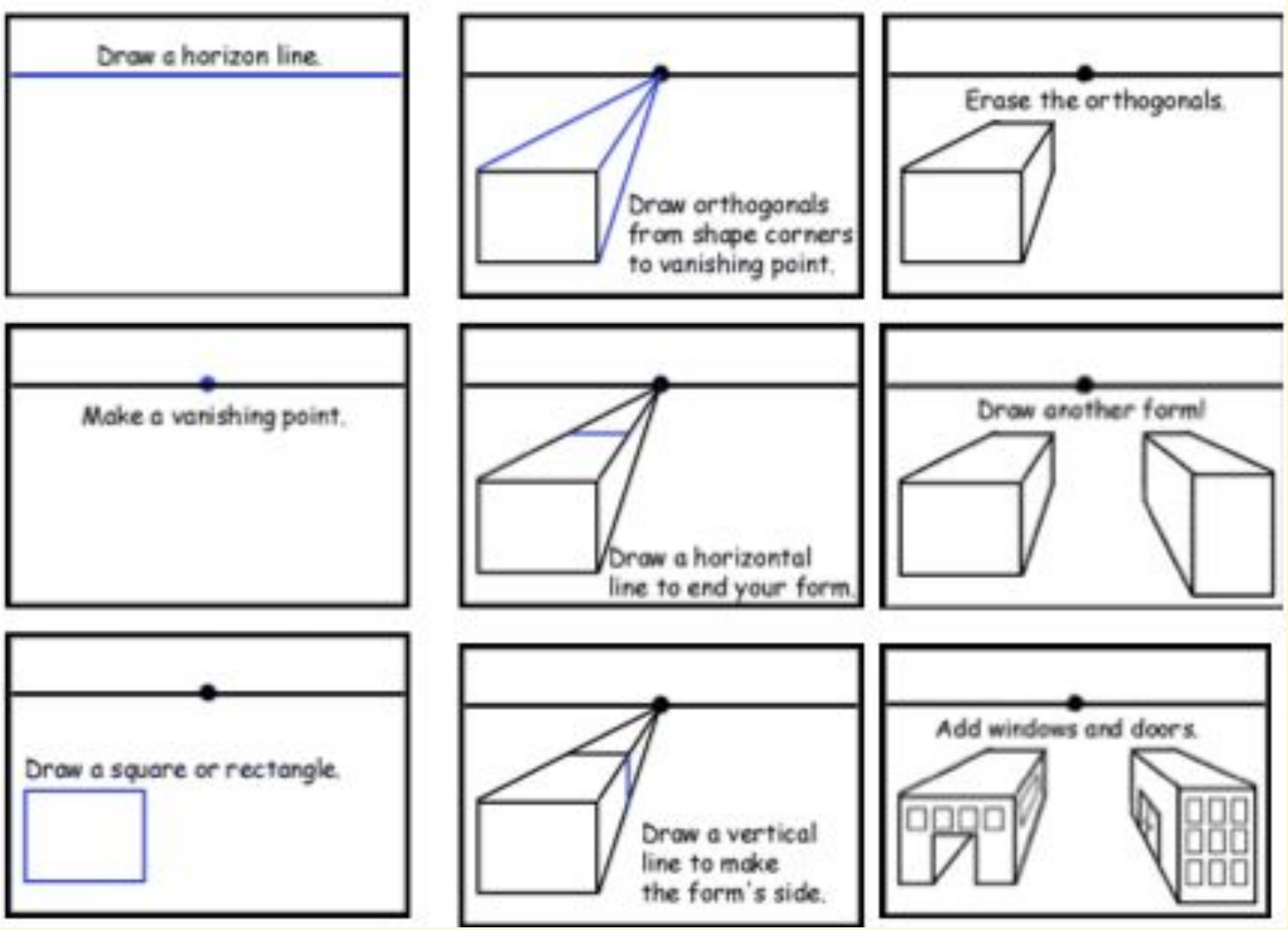
- Parallel lines intersect at a point
- The intersection of red lines and blue lines form a parallelogram
- Sets of parallel lines on the same plane form a vanishing line
- Sets of parallel planes form a vanishing line
- Not all lines that intersect are parallel

Vanishing points and lines



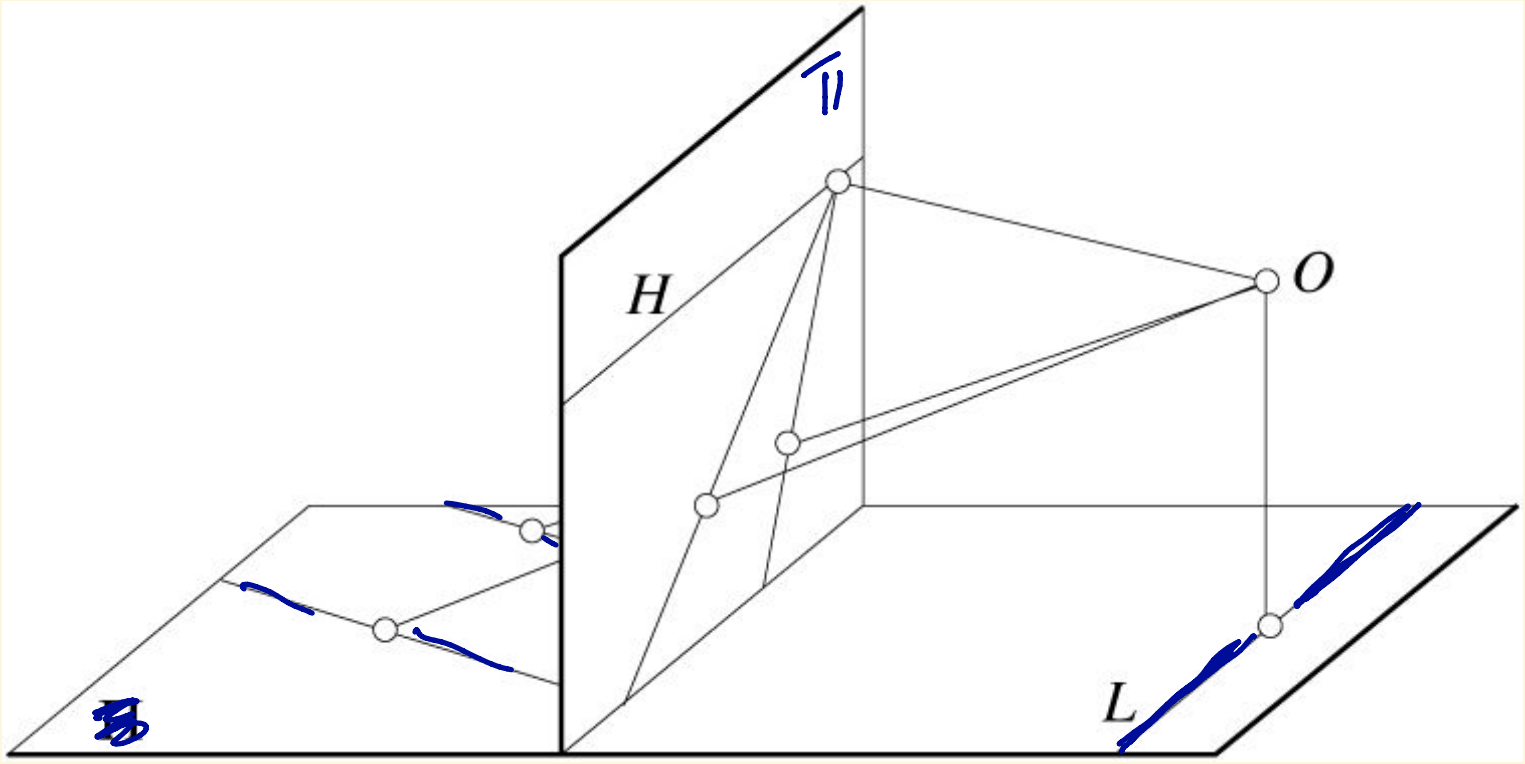
Slide from Efros, Photo from Criminisi

Perspective drawing



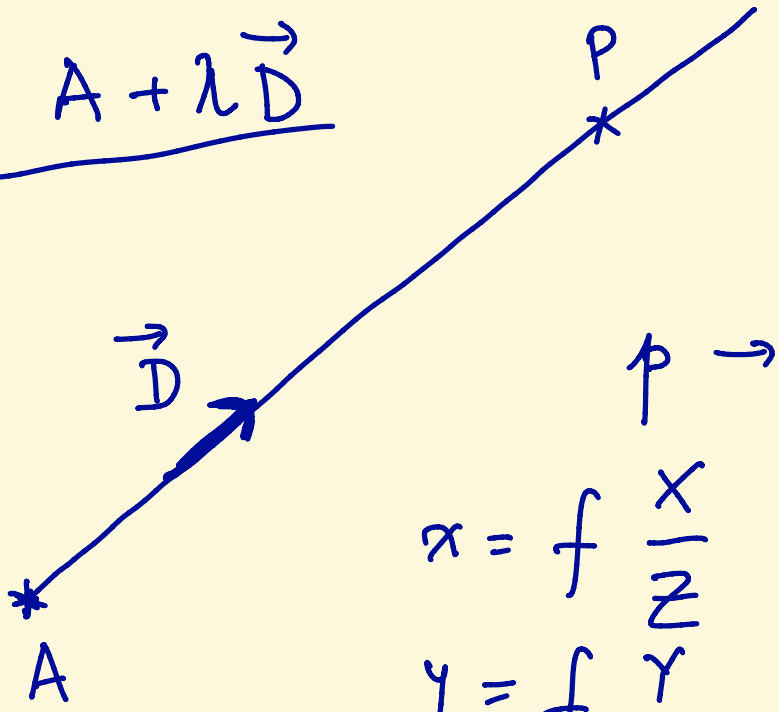
Parallel lines

Can the perspective image of two lines in a plane meet at a point?



Vanishing point in vector notation

$P = A + \lambda \vec{D}$



$\vec{D}$ - unit vector

$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$

$p \rightarrow$ image point $\begin{bmatrix} x \\ y \end{bmatrix}$ $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

$x = f \frac{X}{Z}$

$y = f \frac{Y}{Z}$

$p = f \frac{P}{Z}$

$= f \frac{A + \lambda \vec{D}}{A_z + \lambda D_z}$

Vanishing point in vector notation

$\lambda \to \infty$

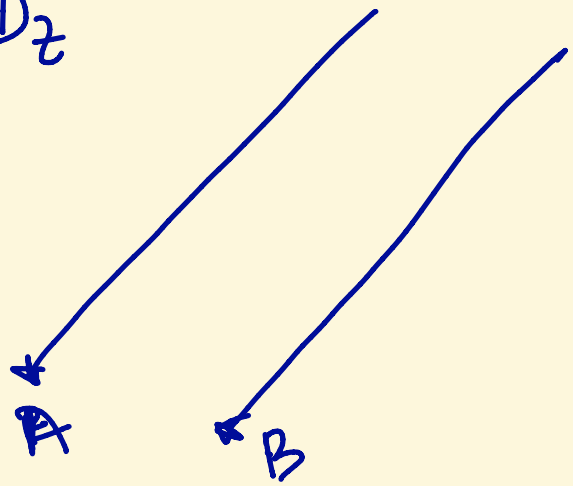
$$p = f \frac{A + \lambda \vec{D}}{A_z + \lambda D_z}$$

$$\lim_{\lambda \to \infty} p \approx f \frac{\lambda \vec{D}}{\lambda D_z}$$

$\vec{D}$ $\vec{s}$

$A \ll \lambda \vec{D}$

$A_z \ll \lambda D_z$

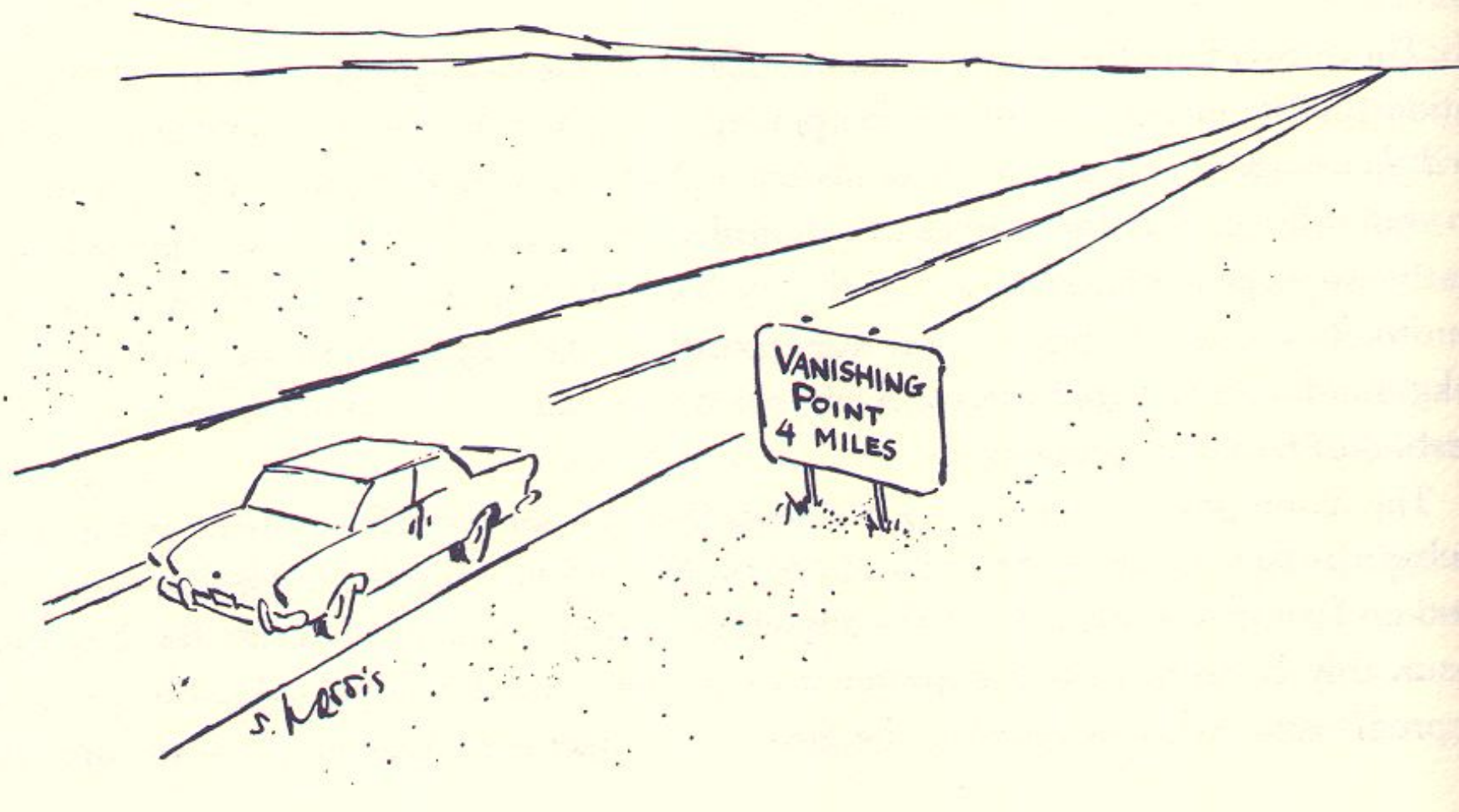


Vanishing point in vector notation

There are two advantages of using homogeneous notation to represent perspective projection:

1. Non-linear projections equations are turned into linear equations. The tools of linear algebra can then be used.
2. Vanishing points are treated naturally, and awkward limiting procedures are then avoided.

Cartoon. (Drawing by S. Harris; © 1975 The New Yorker Magazine, Inc.)



Vanishing point in vector notation

$$\mathbf{p} = f\frac{\mathbf{X}}{Z}$$

A line of points in 3D can be represented as $\mathbf{X} = \mathbf{A} + \lambda\mathbf{D}$, where $\mathbf{A}$ is a fixed point, $\mathbf{D}$ a unit vector parallel to the line, and λ a measure of distance along the line. As λ increases points are increasingly further away and in the limit:

$$\lim_{\lambda\to\infty} \mathbf{p} = f\frac{\mathbf{A} + \lambda\mathbf{D}}{A_Z + \lambda D_Z} = f\frac{\mathbf{D}}{D_Z}$$

i.e. the image of the line terminates in a *vanishing point* with coordinates $(fD_X/D_Z, fD_Y/D_Z)$, unless the line is parallel to the image plane ($D_Z = 0$). Note, the vanishing point is unaffected (invariant to) line position, $\mathbf{A}$, it only depends on line orientation, $\mathbf{D}$. Consequently, the family of lines parallel to $\mathbf{D}$ have the same vanishing point.

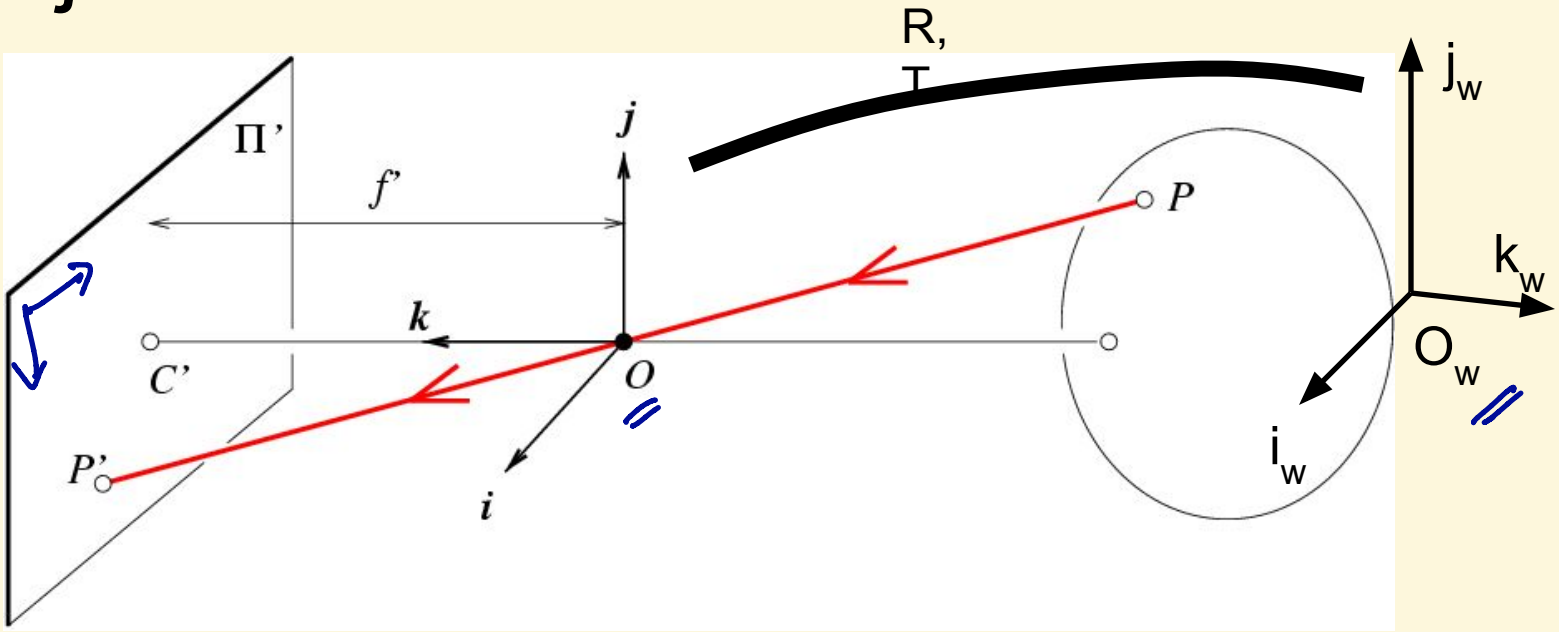
Camera Parameters

- Summary:
  - points expressed in external frame
  - points are converted to canonical camera coordinates
  - points are projected
  - points are converted to pixel units

$$\begin{pmatrix} U \\ V \\ W \end{pmatrix} = \begin{pmatrix} \text{Transformation} \\ \text{representing} \\ \text{intrinsic parameters} \end{pmatrix} \begin{pmatrix} \text{Transformation} \\ \text{representing} \\ \text{projection model} \end{pmatrix} \begin{pmatrix} \text{Transformation} \\ \text{representing} \\ \text{extrinsic parameters} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ T \end{pmatrix}$$

point in pixel coords.	point in metric image coords.	point in cam. coords.	point in world coords.

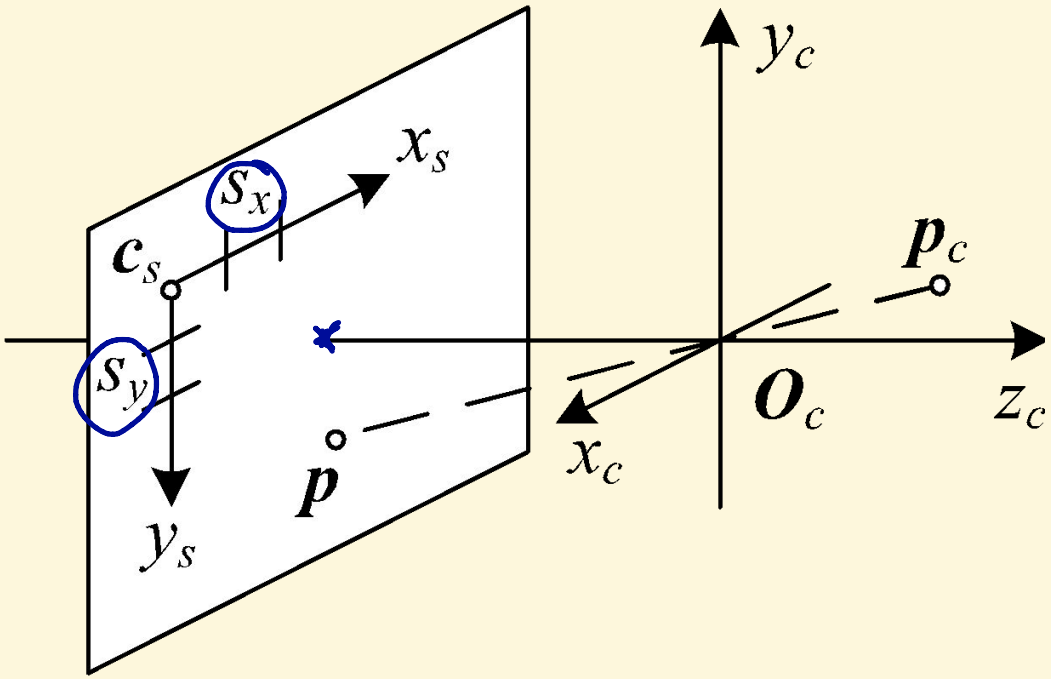
Projection matrix



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

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

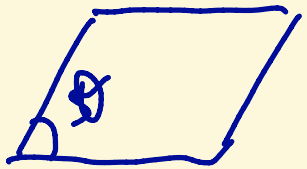
Estimating **K**: after projecting through the pinhole, we must convert metric coordinates to pixel coordinates, according to the pixel sensor spacing and the relative position of the sensor plane (image plane) to the origin.



Note: Image sensors return pixel values indexed by integer pixel coordinates (xs, ys), often with the coordinates starting at the upper-left corner of the image and moving down and to the right.

Intrinsic Parameters:

- focal length **f**
- **S_x and S_y** are pixel spacings (size) in the x and y direction (in microns)
- **c_x and c_y** are the coordinates of the optical center (in pixel units)
- **s** is the pixel skew (if pixel not rectangular) – same as shear transformation



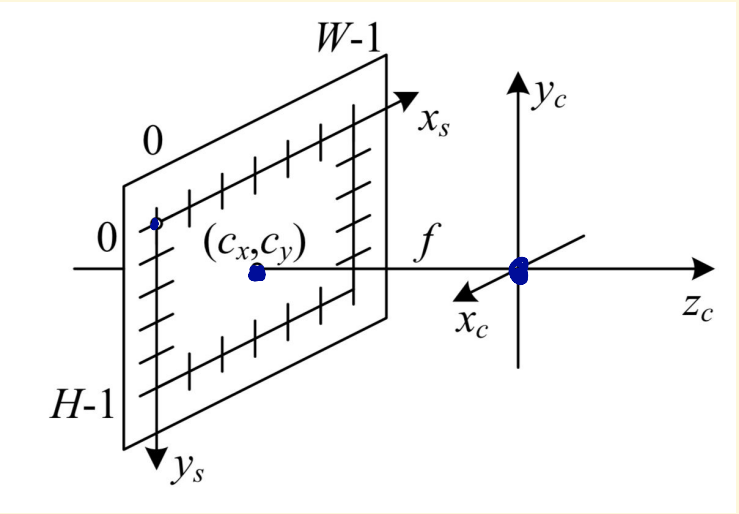
$s = \cos\theta = 0$

a = aspect ratio

$= \frac{s_x}{s_y} = 1$

Projection matrix

square



Intrinsic Assumptions

- Unit aspect ratio $a = 1$ $s_x = s_y$
- Optical center at (0,0)
- No skew $s = 0$

Extrinsic Assumptions

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

$$p = K P$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

3×4

$$x = f \frac{X}{Z}$$

$$y = f \frac{Y}{Z}$$

Slide Credit: Saverese

Remove assumption: non-skewed pixels

Intrinsic Assumptions

- Optical center at (0,0)

Extrinsic Assumptions

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

1) pixels are no longer square

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

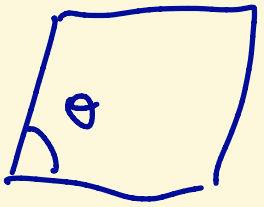
$\alpha = \frac{f}{s_x}$ $\beta = \frac{f}{s_y}$

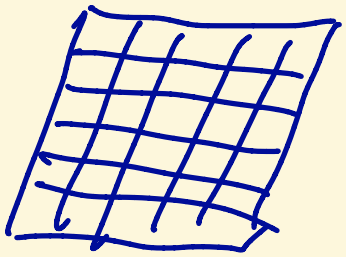
$a = \frac{s_x}{s_y} \neq 1$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha = f/s_x & 0 & 0 & 0 \\ 0 & \beta_{f/s_y} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow * s_x \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & a*f & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2) $s \neq 0$

$s = \cos\theta$



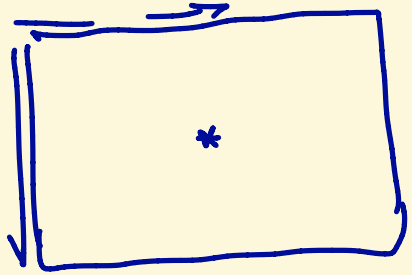


Note: different books use different notation for parameters

Remove assumption: known optical center

Intrinsic Assumptions

$(c_x, c_y) \neq (0, 0)$



$$c_x = \frac{w}{2}$$

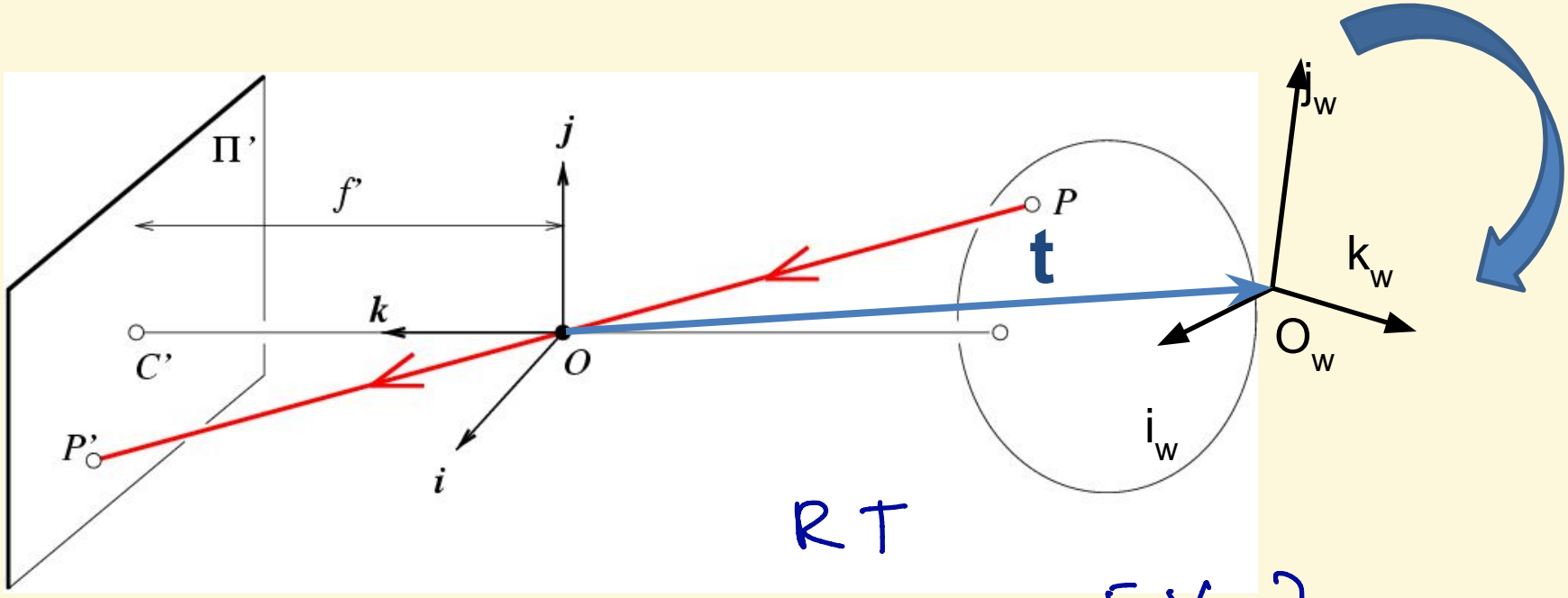
$$c_y = \frac{h}{2}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f & 0 & c_x & 0 \\ 0 & af & c_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{K} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Extrinsic Assumptions

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

Oriented and Translated Camera



$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = k \begin{bmatrix} \quad \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Allow camera translation

Intrinsic Assumptions Extrinsic Assumptions

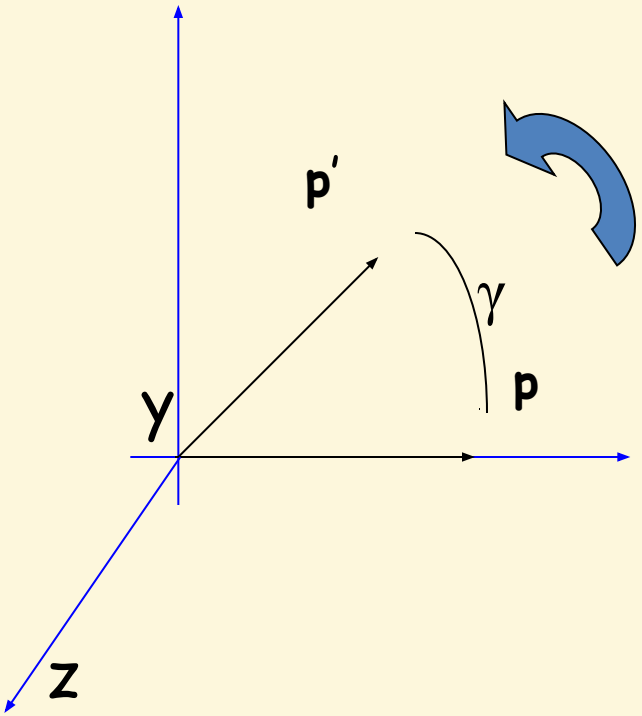
- No rotation

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \begin{bmatrix} 0 & 0 & 0 & t_x \\ 0 & 0 & 0 & t_y \\ 0 & 0 & 0 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

t_x
t_y
t_z

3D Rotation of Points

Rotation around the coordinate axes, counter-clockwise:



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

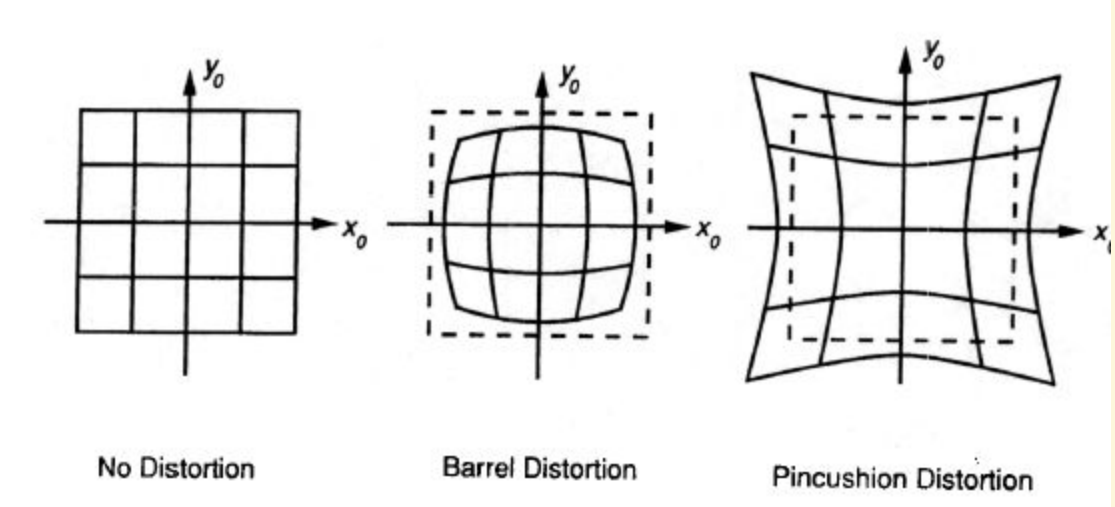
$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

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

Allow camera rotation

Beyond Pinholes: Radial Distortion

The radial distortion model says that coordinates in the observed images are displaced away(barrel distortion) or towards (pincushion distortion) the image center by an amount proportional to their radial distance



Corrected Barrel Distortion

Image from Martin Habbecke

Degrees of freedom

Q: How many known points are needed to estimate this?

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

5 6

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

Early vision: one image

- Linear filters and convolution
- Ponce & Forsyth chapter 4
- Szeliski chapter 3.2

Linear filters

Definition = an output pixel's value is determined as a weighted sum of input pixel values

Properties: Whatever the weights chosen, the output of this procedure is:

- *shift invariant*—meaning that the value of the output depends on the pattern in an image neighborhood, rather than the position of the neighborhood
- *linear*—meaning that the output for the sum of two images is the same as the sum of the outputs obtained for the images separately.

Convolution

- Each pixel in output image is
  - weighted average of window of pixels in input image
  - weights stay the same
  - window centered on pixel
  - window weights form the convolution **kernel**
- Linear filtering can also be used as a pre-processing stage to edge extraction (Section 4.2 Szeliski) and interest point detection (Section 4.1) algorithms.
- Important operations
  - Example: smoothing by averaging
  - Example: smoothing by weighted average

Kernel example

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

Convolution - Example I

- Example: replace each pixel with an average of neighbors (mean filter or box filter)
  - (2k+1) by (2k+1) window centered at pixel

$$\mathcal{R}_{ij} = \frac{1}{(2k+1)^2} \sum_{u=i-k}^{u=i+k} \sum_{v=j-k}^{v=j+k} \mathcal{F}_{uv}.$$

i, j'th pixel in output image

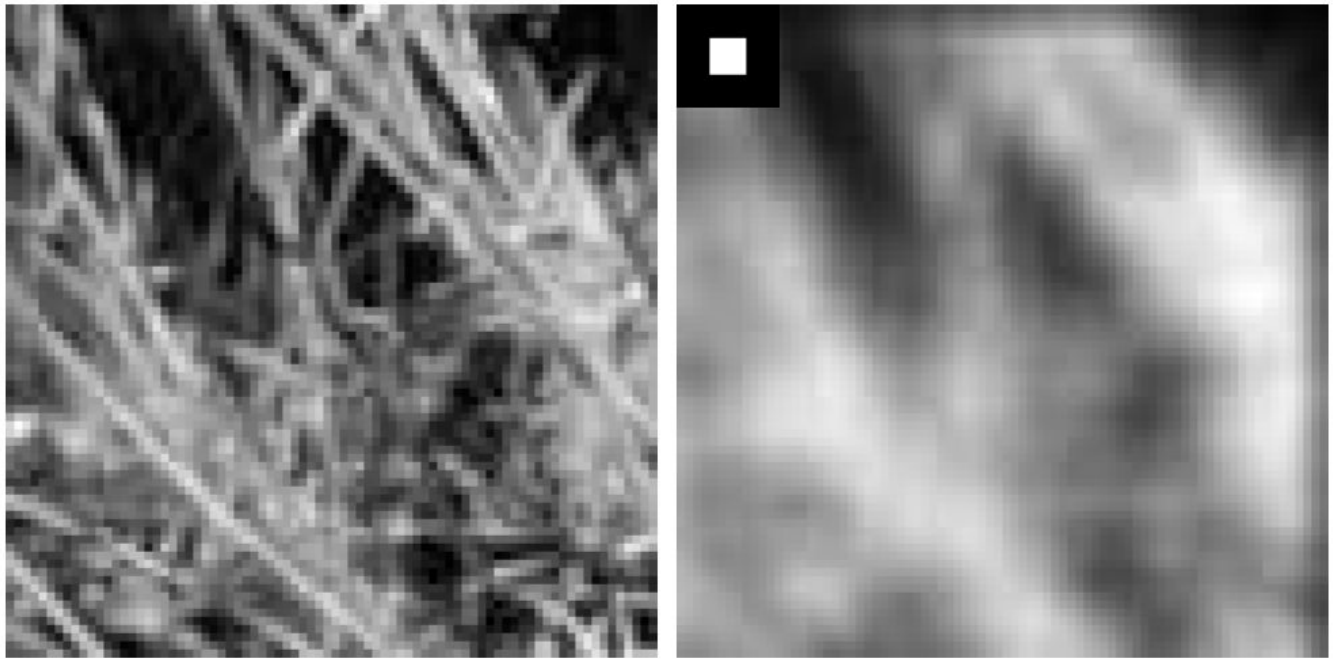
u, v'th pixel in input image

Kernel = ?

Smoothing by averaging

Inset: smoothing weights

Input image



Output image

Figure 7.1 Ponce & Forsyth

Forming a weighted average

- Place window over image at pixel, sum weighted pixels
  - Indexing convention: window is zero outside range

Weights

$$R_{ij} = \sum_{u,v} H_{i-u,j-v} F_{u,v}$$

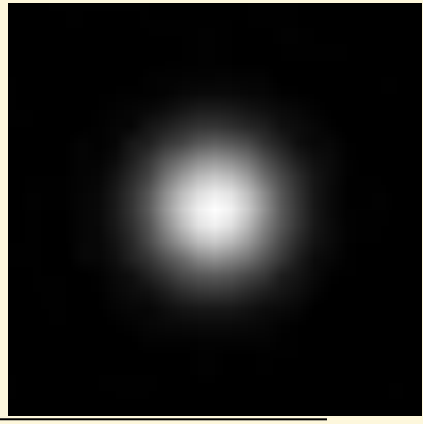
i, j'th pixel in output image

u, v'th pixel in input image

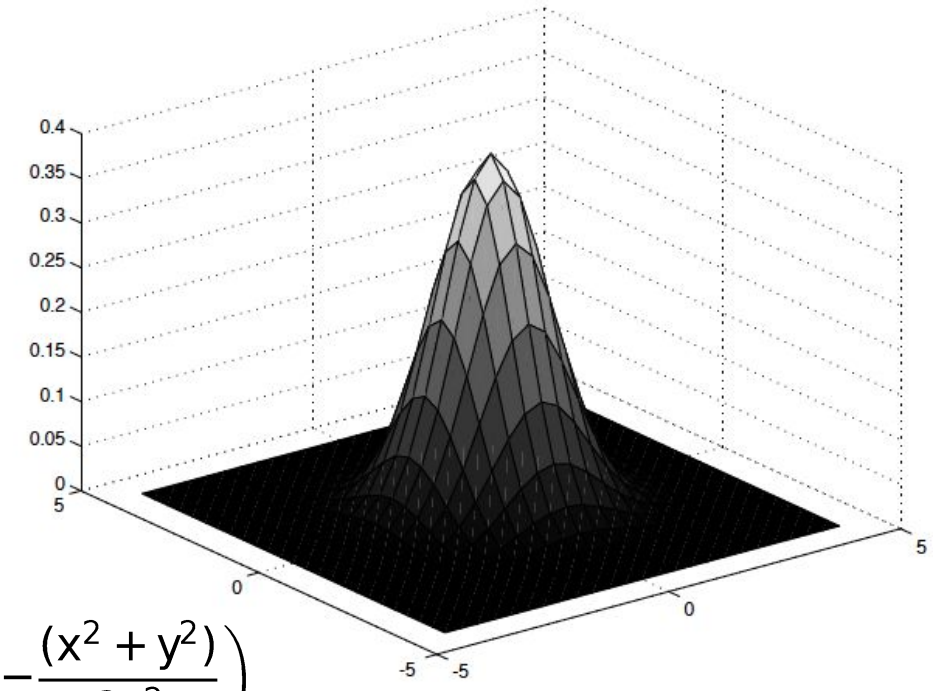
Convolution - Example II

- Averaging neighbors yields poor smoothing
  - look at picture - ringing effects
  - distant neighbors have the same effect as nearby neighbors
- Idea:
  - distant neighbors have small weights, nearby have large
  - weights from Gaussian

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x^2 + y^2)}{2\sigma^2}\right)$$



The Gaussian Smoothing Kernel

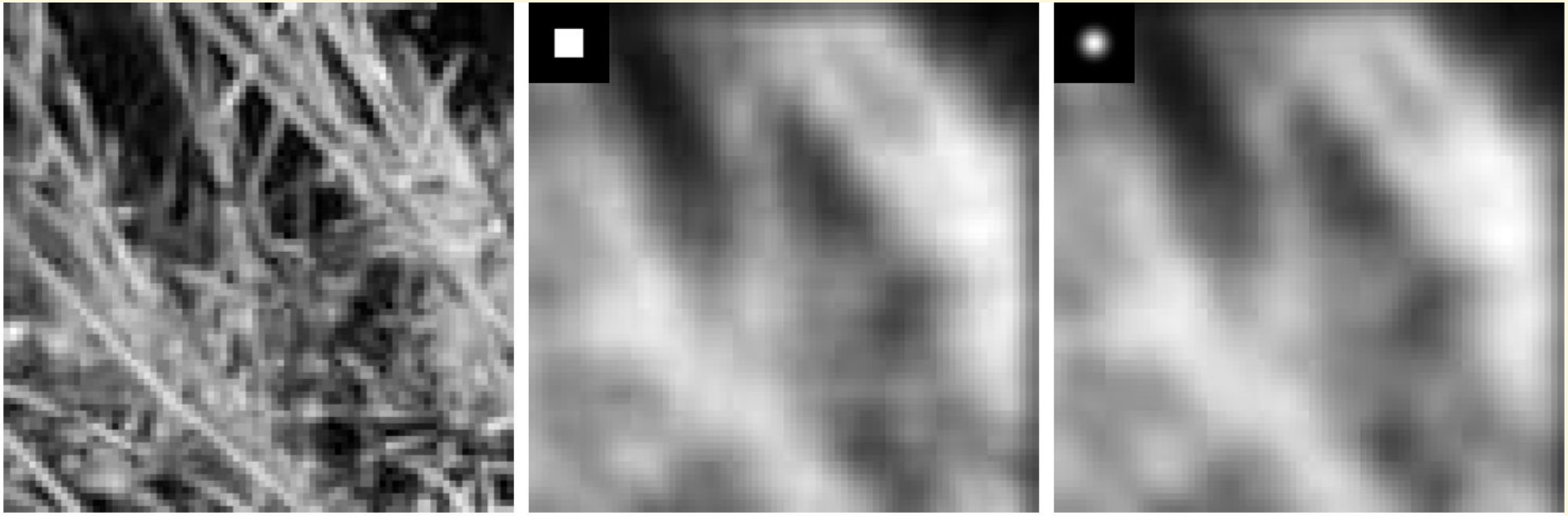


$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x^2 + y^2)}{2\sigma^2}\right)$$

FIGURE 4.2: The symmetric Gaussian kernel in 2D. This view shows a kernel scaled so that its sum is equal to one; this scaling is quite often omitted. The kernel shown has $\sigma = 1$. Convolution with this kernel forms a weighted average that stresses the point at the center of the convolution window and incorporates little contribution from those at the boundary. Notice how the Gaussian is qualitatively similar to our description of the point spread function of image blur: it is circularly symmetric, has strongest response in the center, and dies away near the boundaries.

Smoothing with a Gaussian

Inset: smoothing weights



Input image

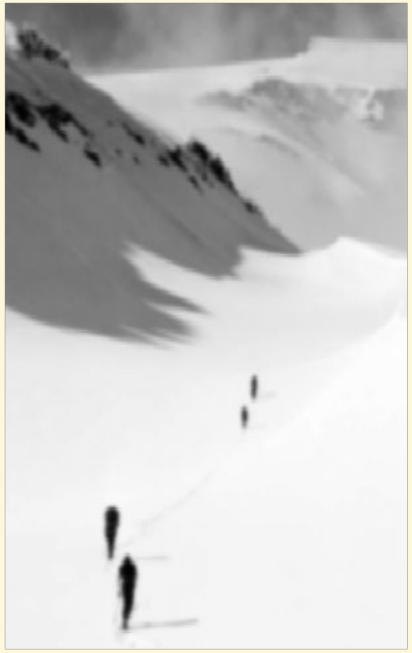
Output image, average

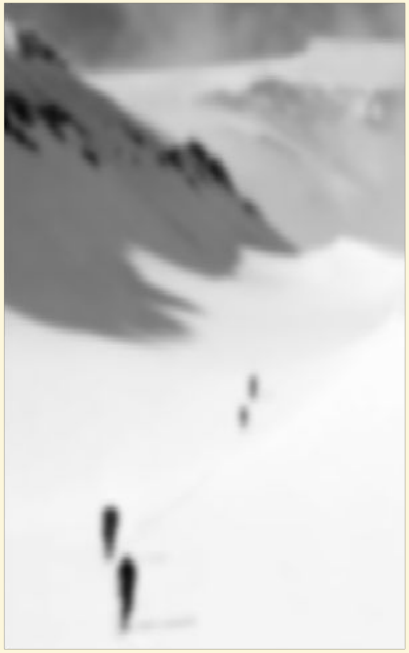
Output image, gaussian weights

Figure 7.1, Ponce & Forsyth

Properties and effects of Gaussian smoothing

- suppressing noise – distant neighbors factors less
- if standard deviation of the Gaussian is very small — say smaller than one pixel — the smoothing will have very little effect, because the weights for all pixels off the center will be very small;
- for a larger standard deviation, the neighboring pixels will have larger weights in the weighted average, which means in turn that the average will be strongly biased toward a consensus of the neighbors — this will be a good estimate of a pixel's value, and the noise will largely disappear, at the cost of some blurring;
- finally, a kernel that has very large standard deviation will cause much of the image detail to disappear along with the noise.





Original Smoothed at σ=1 Smoothed at σ=2