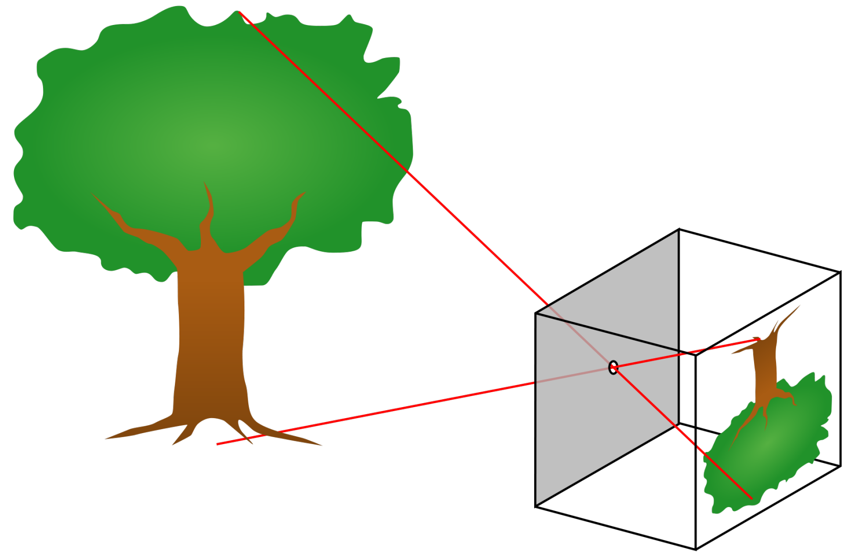
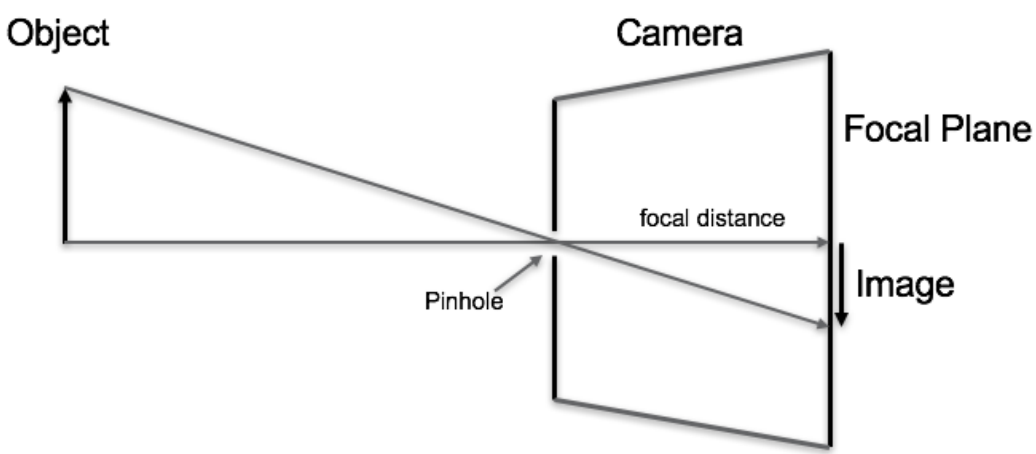
Perspective Correction
and
Motion Estimation

The Simplest Camera Model

- **The Pinhole Camera**
  - Image is inverted
  - Infinite depth of field

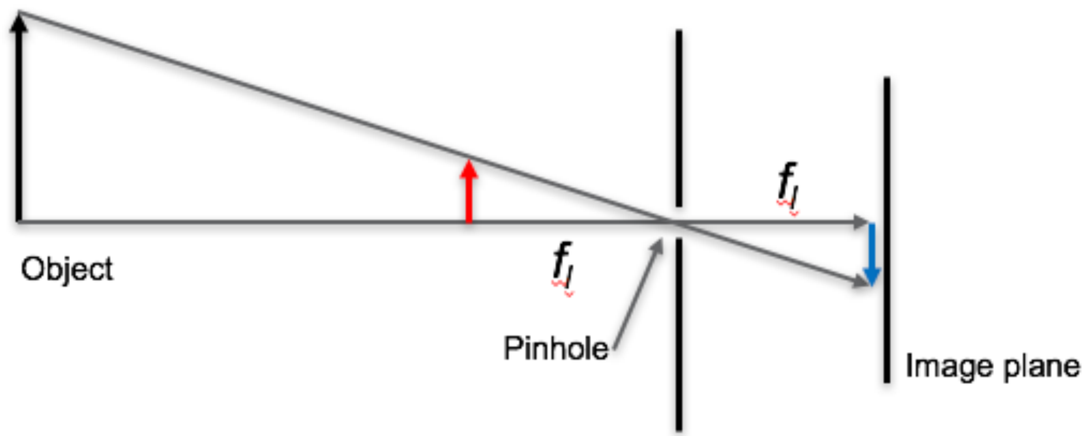


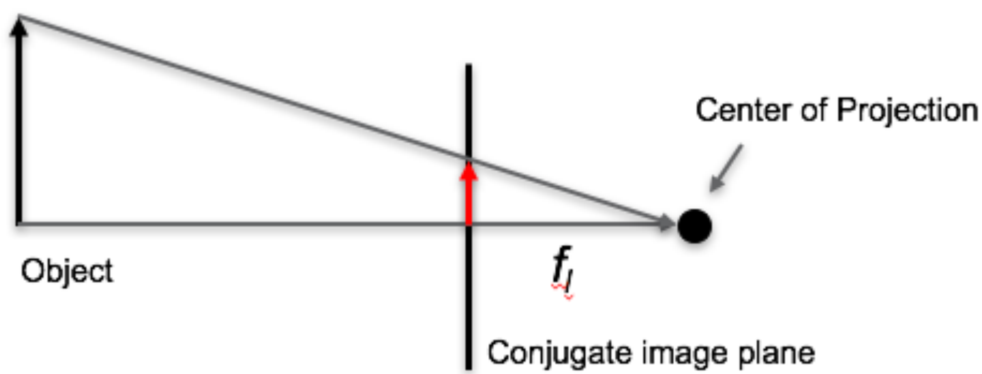
www.funsizephysics.com



Simplest Camera Model cont.

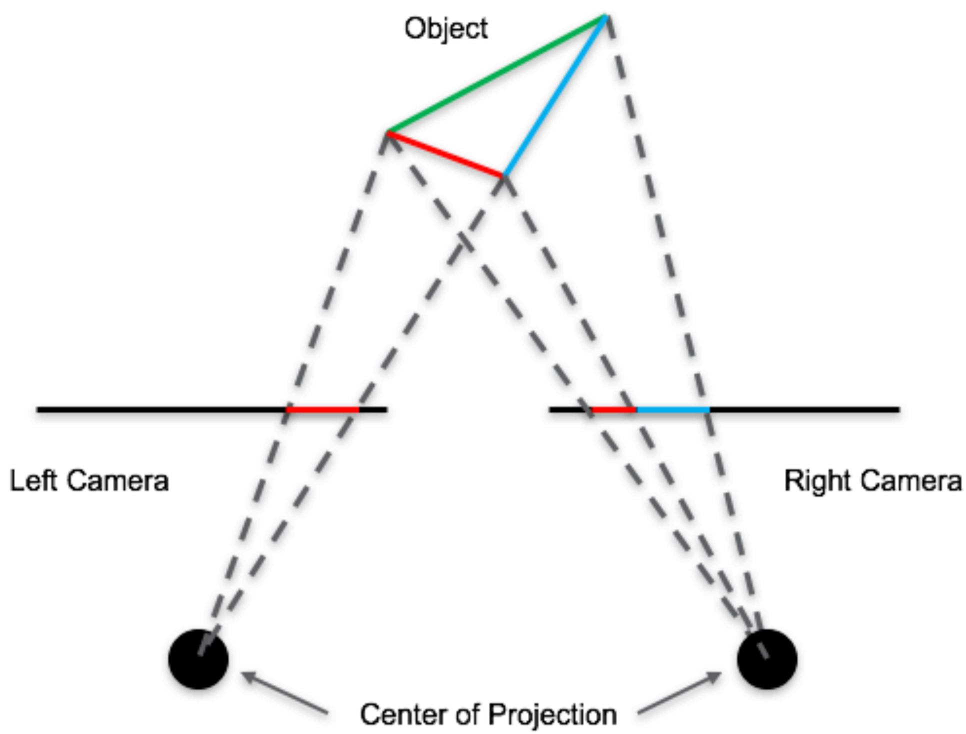
- **Image plane** ⇨ **Conjugate image plane**
- **Pinhole** ⇨ **Center of Projection**
- **Upside down** ⇨ **Upside up**





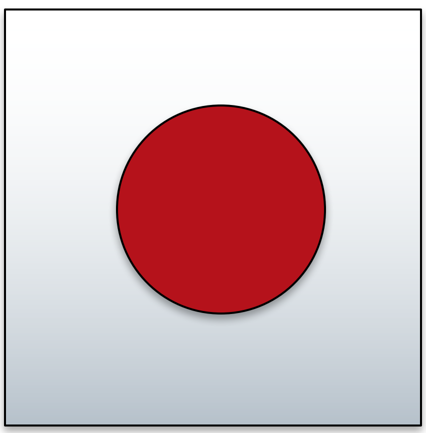
Stereo Camera Model

- **Camera diagrams like this are commonly encountered in computer vision and graphics**

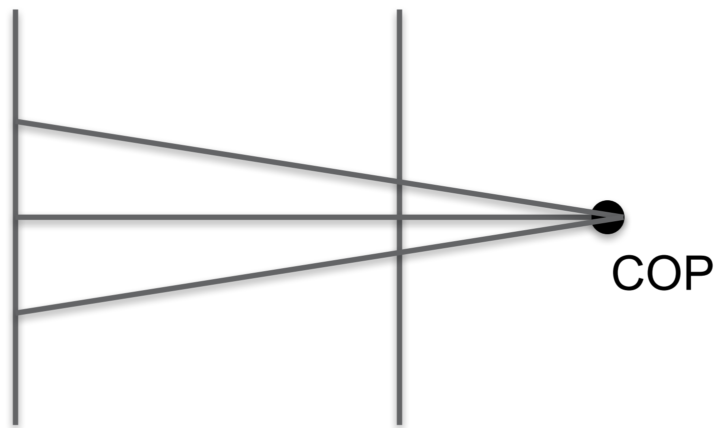


Projective Distortion

- **A circle in an object plane parallel to the image plane images as a circle**



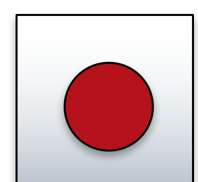
Front view of object plane



Side view of object plane

Image plane

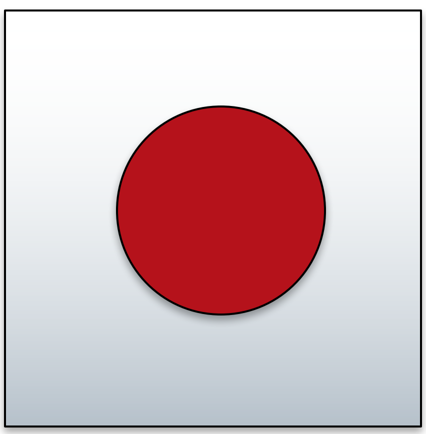
Front view of image plane picture



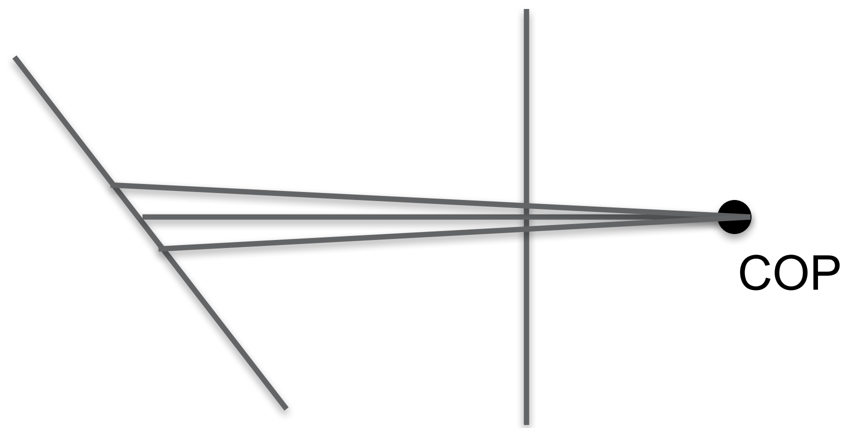
Perspective Distortion cont.

CU Boulder

- A circle in a plane *tilted* relative to the image plane images as an ellipse



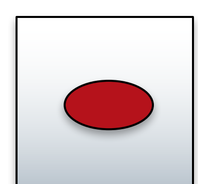
Front view of object plane



Side view of object plane

Image plane

Front view of image plane picture



Perspective Distortion cont.

- **Another example**
  - Parallel lines lying in a plane that is tilted relative to the image plane appear to converge at infinity
  - If the lines were lying in a plane parallel to the image plane, they would appear parallel

Perspective Distortion cont.

- **Distortions include**
  - Shape is not preserved
    - ✓e.g. cirlces become ellipses
  - Distance between two points is not preserved
    - ✓e.g. parallel lines converge
  - Angles are not preserved
    - ✓e.g. Imagine rotating a right triangle lying in a plane that starts parallel to the image plane and ends perpendicular to it

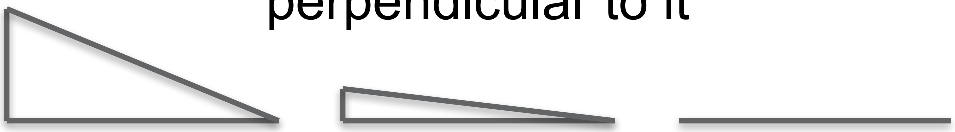
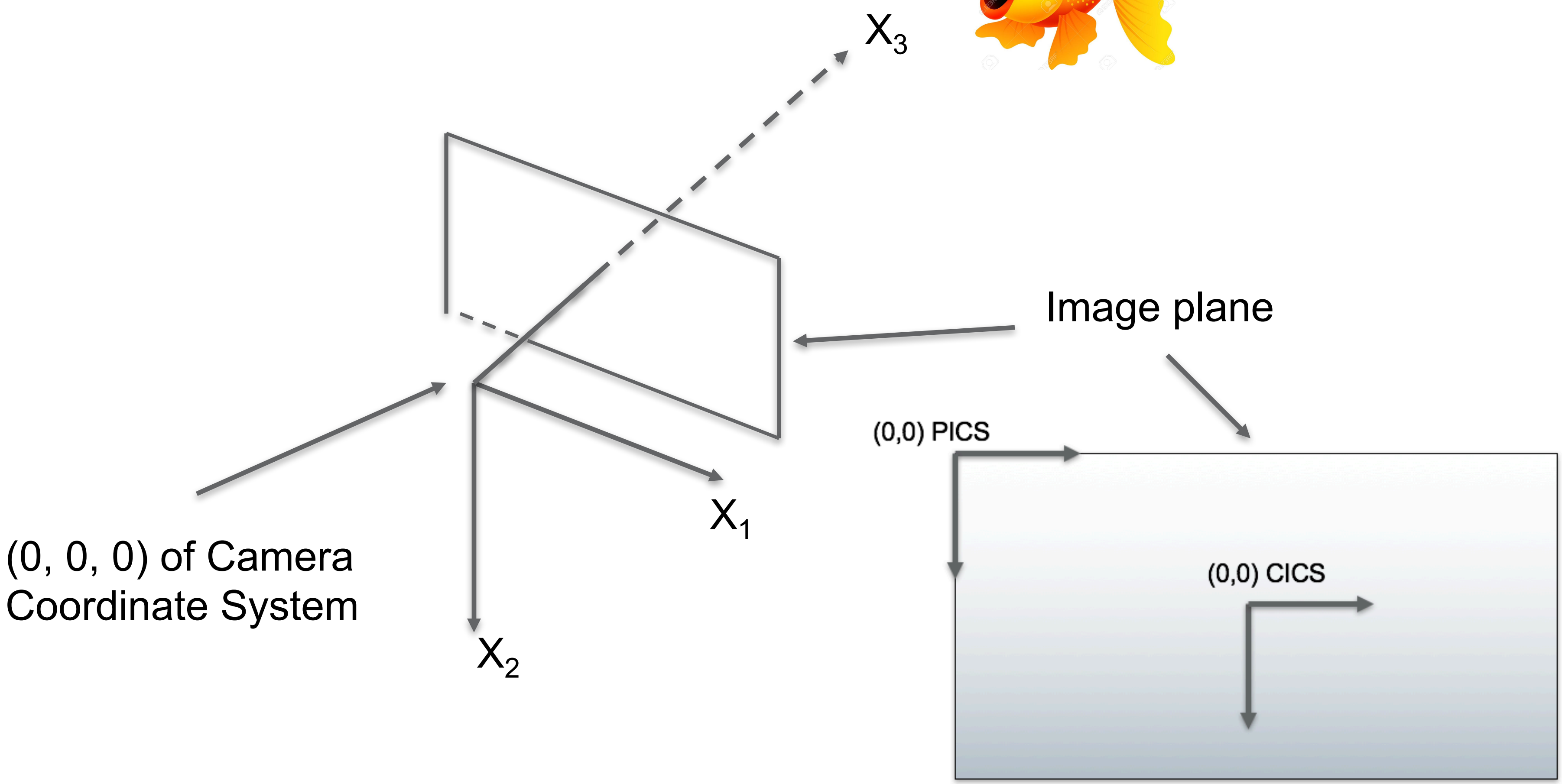


Image and triangle planes parallel

Triangle plane nearly perpendicular

Triangle plane perpendicular

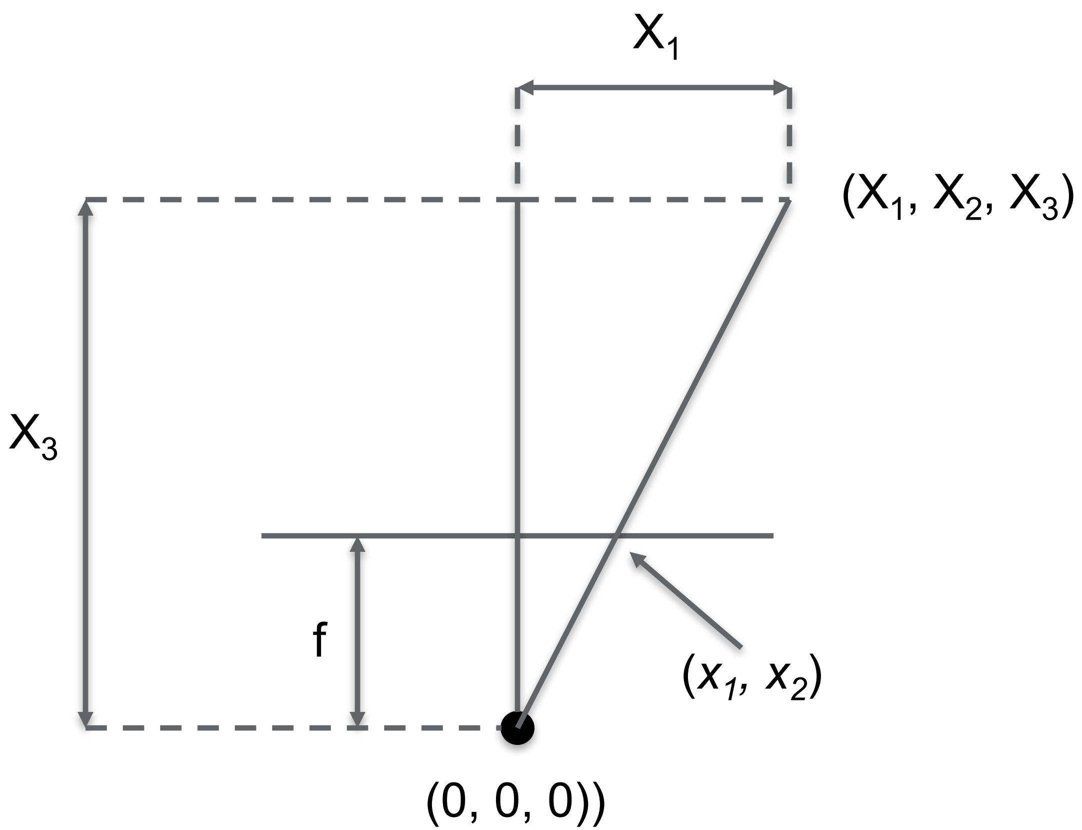
Image Coordinate Systems



PICS: Pixel Image Coordinate System   CICS: Canonical Image Coordinate System

Projection Diagram

CU Boulder



This diagram is in the X_1X_3 plane.

Visualize the point (X_1, X_2, X_3) as lying above the plane so that $X_2 < 0$

$$\frac{f}{X_3} = \frac{x_1}{X_1}$$

$$x_1 = f\frac{X_1}{X_3}$$

A similar argument yields $x_2 = f\, X_2 / X_3$

Non-linear Projection Formulas

- **Therefore, the Camera Coordinates (X_1, X_2, X_3) are related to the canonical image coordinates *(x_1, x_2)* via**

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} f & 0 \\ 0 & f \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \frac{1}{X_3}$$

This term is UGLY and makes the relationship between (X_1, X_2, X_3) and *(x_1, x_2)* non-linear.
There is a better way!

Linearizing the Projection Formulas

- Consider the following ***linear*** equation

$$\underline{\mathbf{x}} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ 1 \end{bmatrix}$$

- **We put a "1" under the (X_1, X_2, X_3) vector**
- **We can get the CIC coordinates of the projection point, $\underline{x}$, by computing x_1/x_3 and x_2/x_3**

This equation uses homogenous coordinates

Homogenous Coordinates

- **In Euclidean geometry, a point in N-D space is described by an N-D vector**
- **In Projective geometry, a point in N-D space is described by an (N+1)-D vector (homogenous coordinates)**
- **The last coordinate is a <u>multiplier</u> of the first N coordinates: let its value be *k***
- **A point now has <u>multiple representations</u>, one for each value of *k***
  - Example: the 2-D Euclidean point, (*rows, cols*), is the SAME POINT as <u>ANY</u> 3-D vector of the form

$$(k*rows,\ k*cols,\ k)$$

Homogenous Coordinates Rules

- **We refer to such a representation as a homogenous vector**

(k^*rows, k^*cols, k) ← homogenous vector

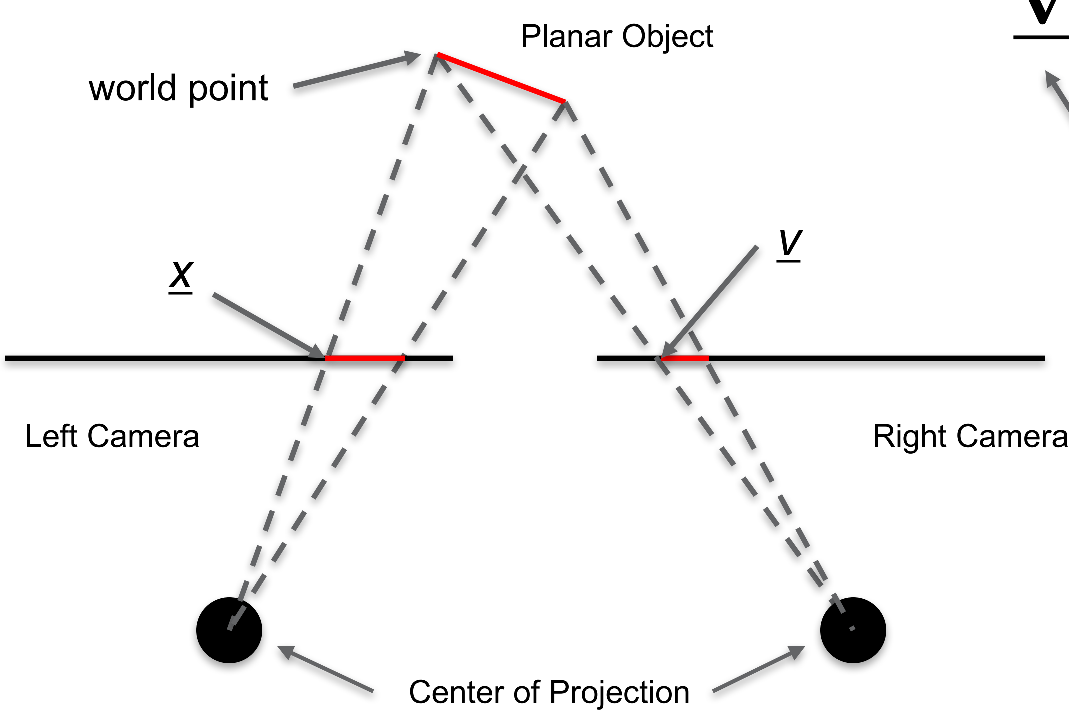
- **We say we are using <u>homogenous coordinates</u>**
- **We can find the Euclidean point corresponding to a homogenous vector by dividing the first N components by the last component**
  - 2-D Example: $(rows, cols) = (k^*rows/k, k^*cols/k)$

$$(12, 8, 2) \rightarrow (6, 4)$$

Projective Geometry Fact

- **Given a set of world points known to lie on a <u>plane</u>, there exists a matrix, *H*, that maps the (*row, col*) image points in one camera to the (*row, col*) image points in another**

$$\underline{\mathbf{v}} = H\underline{\mathbf{x}}$$



Homogenous vector of point in "left" image

Homogenous vector of same point in "right" image

Take a course in computer vision for the details!

Removing Perspective Distortion

- **We seek a matrix H that maps lines converging at infinity to parallel lines in the Euclidean plane**

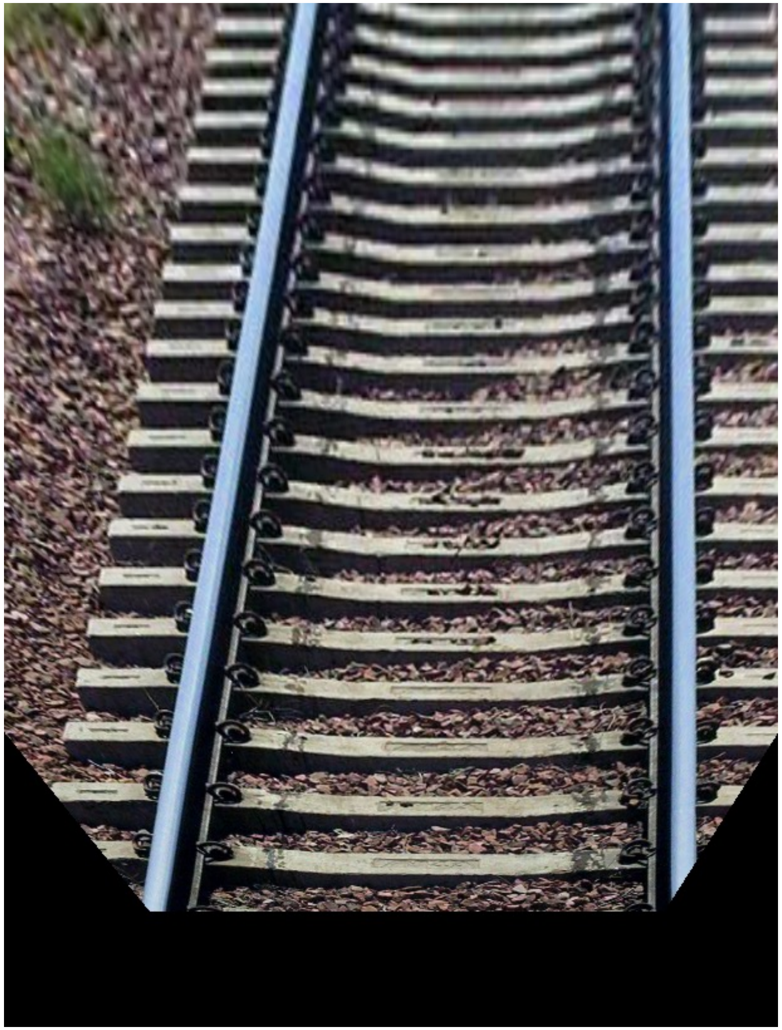
$$\underline{\mathbf{v}} = H\underline{\mathbf{x}}$$

Homogenous vector in processed image

Homogenous vector in "distorted" image

Note that the points on the wall lie in a plane

Perspective Distortion cont.



Perspective Distortion and its removal via image processing

Finding the Matrix *H*

- **Assume we have specified *N* corresponding points in both images**
  - We are free to choose points in one image and specify where they want them to be after mapping
  - For example, assume below are 4 of the *N* point correspondences we have specified

$$
\begin{aligned}
(c_1, d_1) &\rightarrow (a_1, b_1) \\
(c_2, d_2) &\rightarrow (a_2, b_2) \\
(c_3, d_3) &\rightarrow (a_3, b_3) \\
(c_4, d_4) &\rightarrow (a_4, b_4)
\end{aligned}
$$

(c, d) pairs are in the "distorted" image. (a, b) pairs are in the processed image. Coordinates are in PICS

Finding *H* cont.

- **The equations we need are these**

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} c \\ d \\ 1 \end{bmatrix} \text{ and } \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} v_1/v_3 \\ v_2/v_3 \end{bmatrix}$$

- **We need to solve for the matrix values h_{ij}**
  - There are 8 unknown variables in *H*

Note: you might wonder why we can specify the (3, 3) component of H to be one. The reason: since we are using homogenous coordinates, multiplying both sides of the equation by a constant has no net effect. So whatever h_{33} is, we assume we have multiplied the equation by $1/h_{33}$.

Finding *H* cont.

- **By solving for v_1/v_3 and v_2/v_3 the previous equations yield**

$$a = \frac{h_{11}c + h_{12}d + h_{13}}{h_{31}c + h_{32}d + 1}$$

$$b = \frac{h_{21}c + h_{22}d + h_{23}}{h_{31}c + h_{32}d + 1}$$

Each pair of corresponding points yields two equations, so we need a minimum of four corresponding points to solve for the eight variables in H. In practice, we want more.

Finding *H* cont.

- **These two equations can be rewritten in matrix form**

$$\begin{bmatrix} c & d & 1 & 0 & 0 & 0 & -ac & -ad \\ 0 & 0 & 0 & c & d & 1 & -bc & -bd \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

Finding H cont.

- **Each new pair of corresponding points adds 2 more rows to the matrix and 2 more rows to the right-hand side vector**

$$\begin{bmatrix} c & d & 1 & 0 & 0 & 0 & -ac & -ad \\ 0 & 0 & 0 & c & d & 1 & -bc & -bd \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

These grow downward with more corresponding points

Finding *H* cont.

- **This matrix equation can be solved for *h***
  - If more than 4 points are used, solving for H becomes a linear regression problem
    - ✓ Use the pseudo inverse
  - *h* can then be reshaped into the 3 x 3 matrix *H*

For this lab we will code up the pseudo-inverse calculation ourselves (you can check your result with the library function if you wish)

Removing Perspective Distortion

- **Constructing the corrected image, once you have *H*, proceeds as follows:**
  - Step through the pixels in the corrected image using *for( )* loops
    - ✓ Declare an array of all black pixels as the corrected image
  - Use H^{-1} to find corresponding pixels in the input image
    - ✓ The values you find will <u>not</u> be integers
    - ✓ If the values are outside the edge of the picture, skip to the next pixel
  - Bi-linearly interpolate values in the input image
    - ✓ Code to do this is provided in the lab writeup
  - Assign the interpolated result to the corrected image pixel

Measuring Points in an Image

- **You can do this with photoshop, GIMP, or other off-the-shelf image display tools**
- **I have also uploaded to the lab 3 module a script called `get_mouse_pos.py`**
  - It requires the module ”`pyuserinput`” to run
  - You can install it via: `pip3 install pyuserinput`
- **To use `get_mouse_pos.py`**
  - Display the picture
  - Run the script in a terminal window
  - Click the upper left-hand corner of the picture (make sure the picture is displayed at its full resolution)
  - Click on the image points whose coordinates you desire (right-click to exit)