Transformations and Camera Geometry

CS 650: Computer Vision

Spatial Transformations

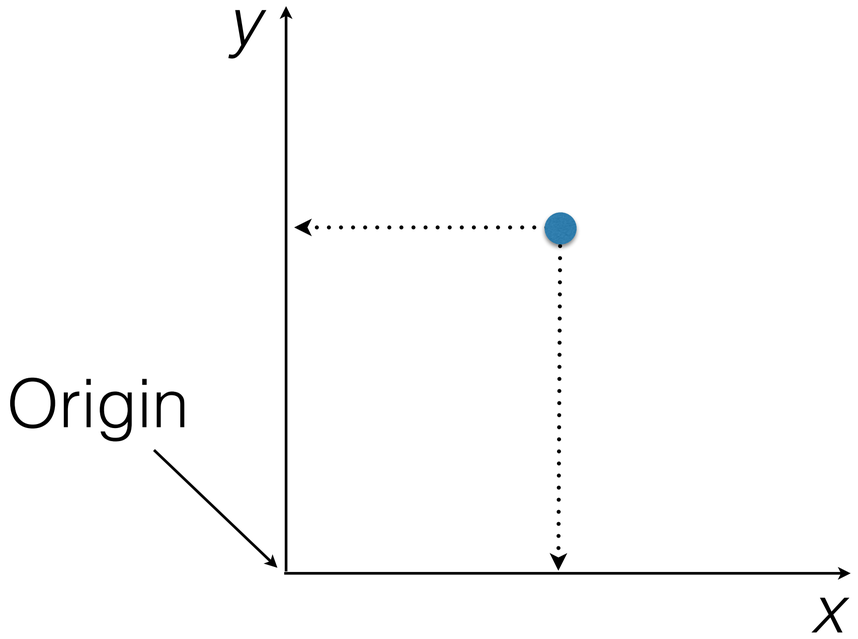
- Many things in computer vision involve analyzing things in different spatial configurations:
  - Different 2D positions (translation)
  - Different 2D orientations (rotation)
  - Different 2D sizes (scaling)
  - Other 2D deformations (e.g., affine)
  - Different 3-D to 2-D projections
  - 3-D to 3-D transformations

Quick detour:
Describing spatial things numerically…

Describing Points

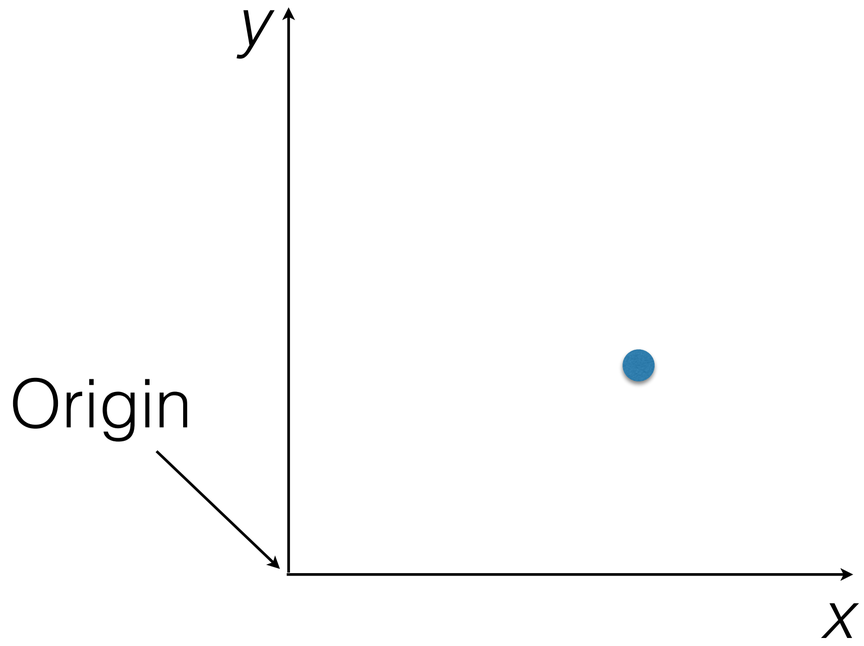
How do you describe this point numerically?

Coordinate Systems



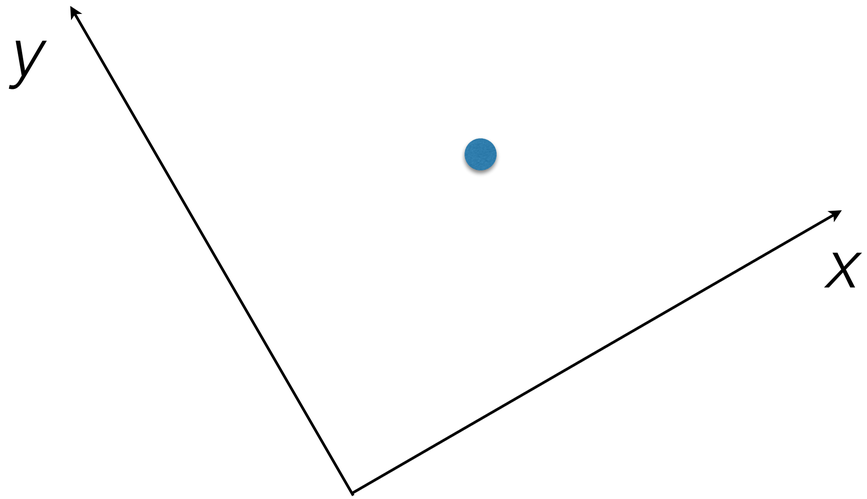
How do you describe this point numerically?

Coordinate Systems



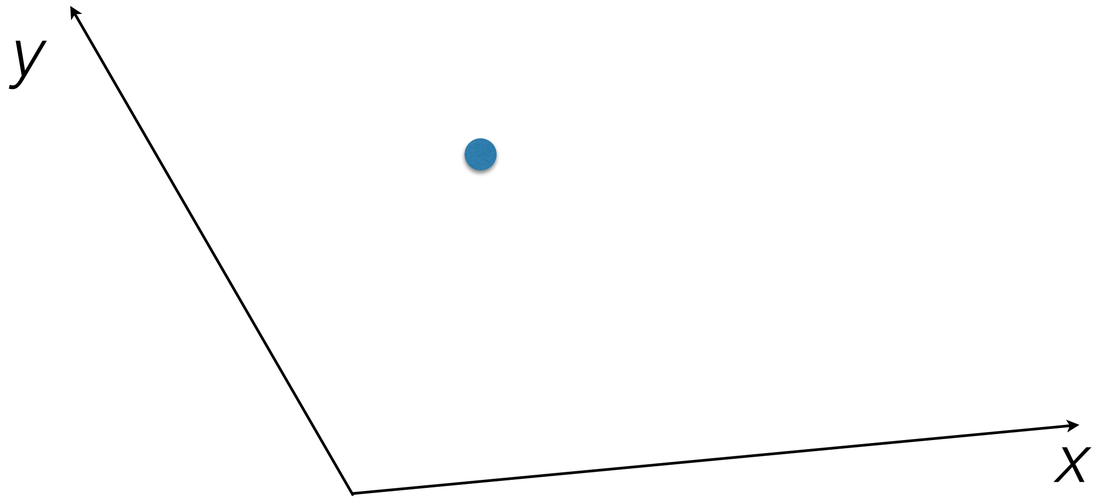
How about this coordinate system?

Coordinate Systems



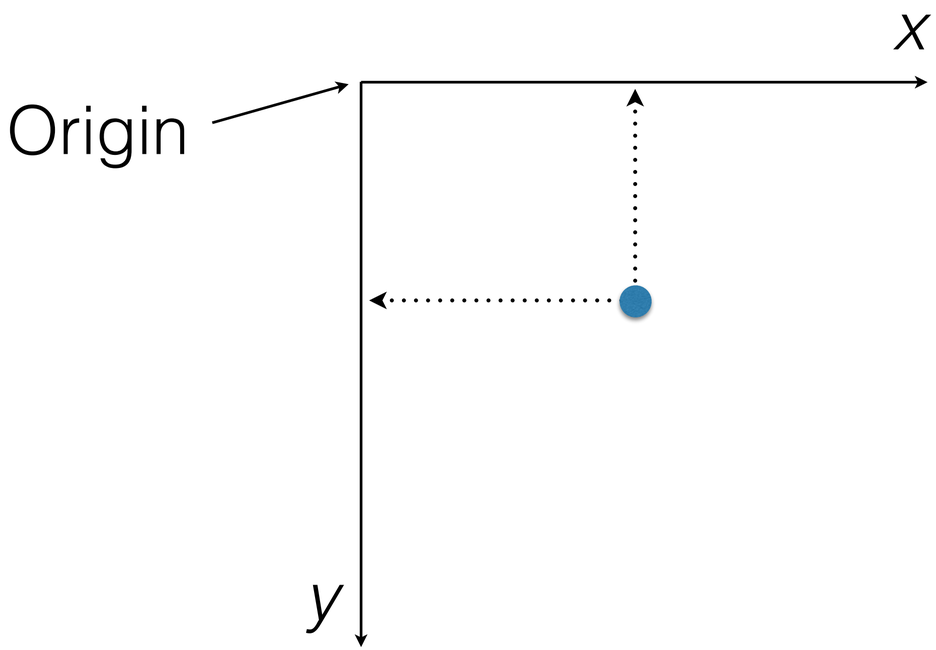
Or this one?

Coordinate Systems



What about this one?

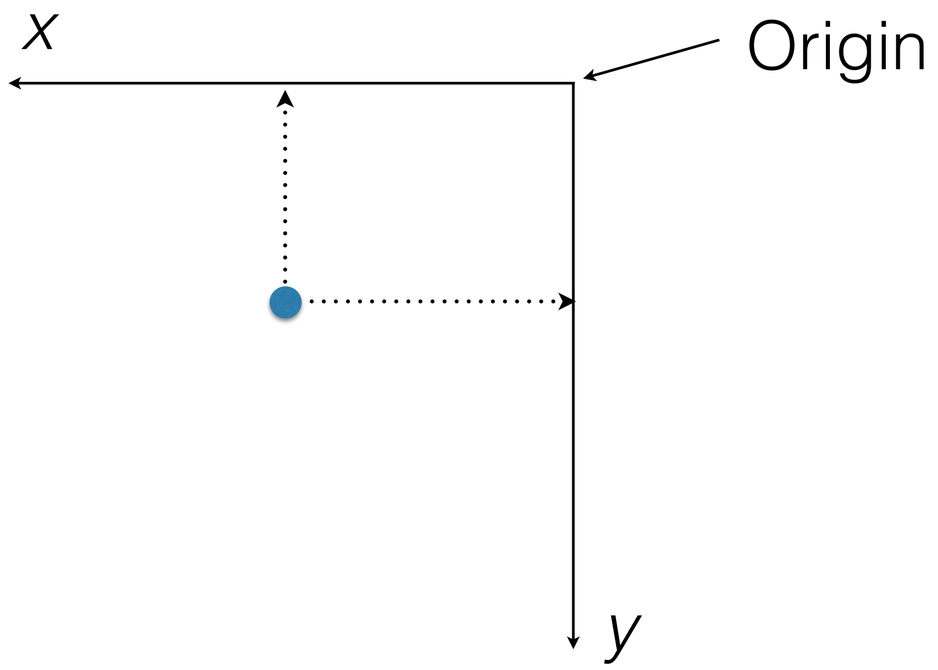
Coordinate Systems



(This is what most screens use)

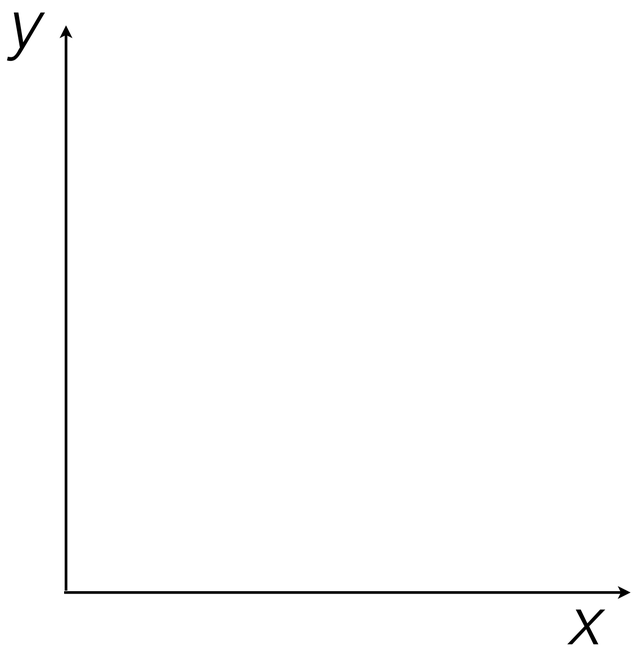
Why not this?

Coordinate Systems

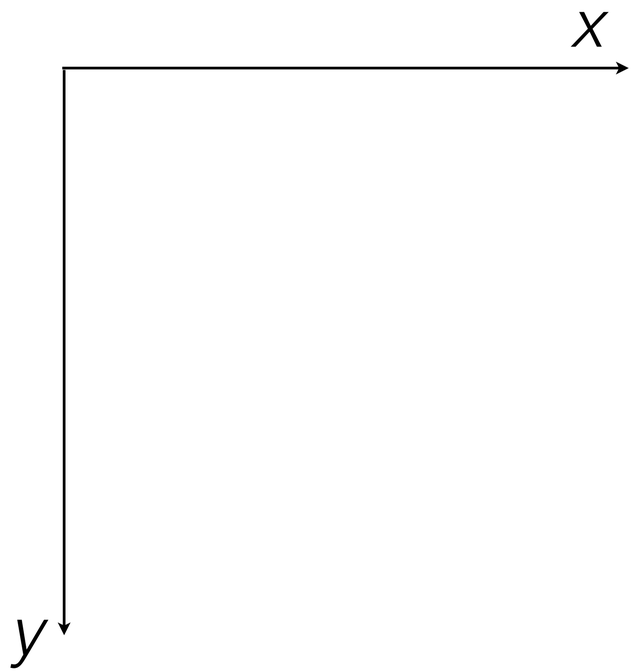


Or this?

Coordinate Systems

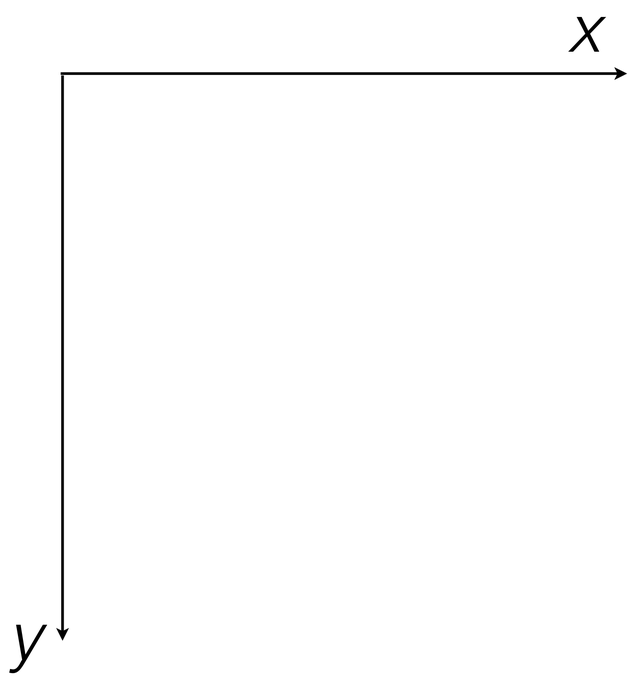


Math teachers
(“right handed”)

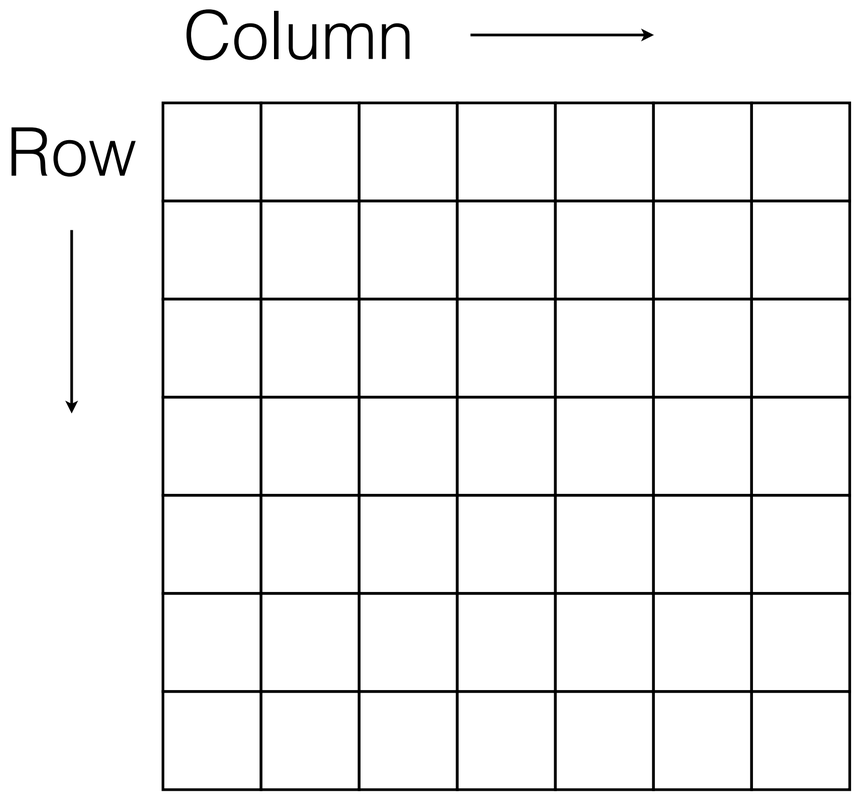


Computer screens
(“left handed”)

Math vs. Code



How we draw
(x,y)



How we store
(row,col)

What About 3D?

- Same ideas apply in 3D
- Right-handed vs. left-handed becomes more important (which way does *z* go?)

Linear Transformations

Translation

- *Translating* a coordinate system simply moves the origin
- Or can be thought of as moving the point
- Keeps the x and y directions the same
- Just add desired x, y offsets

$$(x', y') = (x + t_x, y + t_y)$$

OR

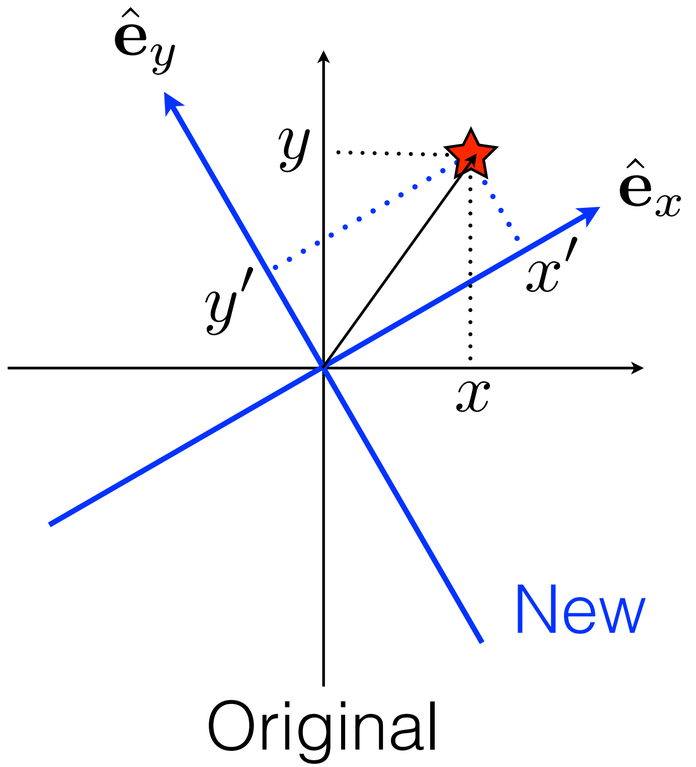
$$\mathbf{p}' = \mathbf{p} + \mathbf{t}$$

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Rotation

- Can again thing interchangeably about rotating a coordinate system and rotating points around the origin of that coordinate system
- To rotate a point's coordinates in a rotated system, project the point on to the new coordinate axes:

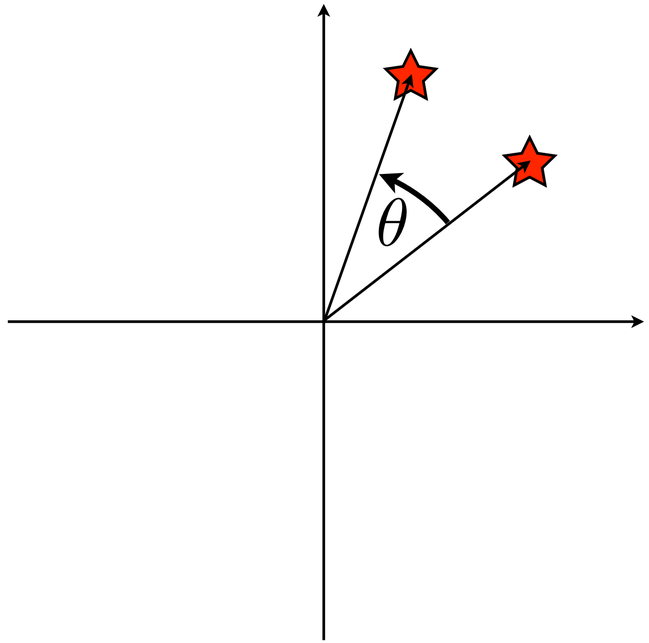
$$\mathbf{p}' = \begin{bmatrix} e_{x1} & e_{x2} \\ e_{y1} & e_{y2} \end{bmatrix} \mathbf{p}$$



Rotation

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{R}^{-1}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

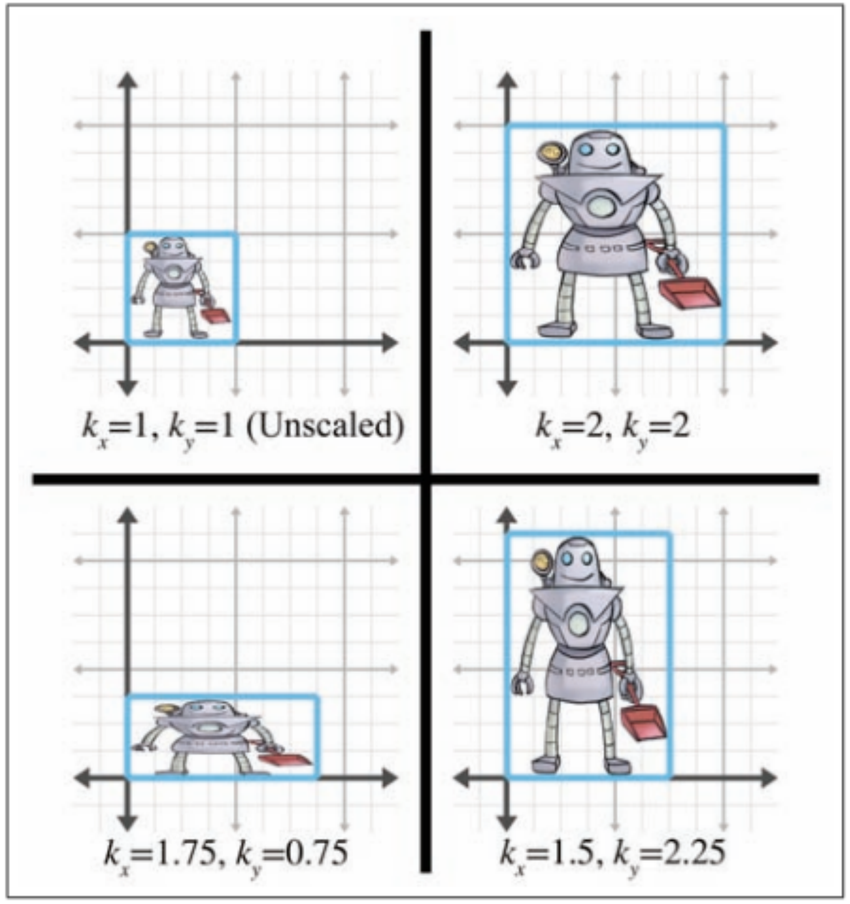


$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scaling

$$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ sy \end{bmatrix}$$

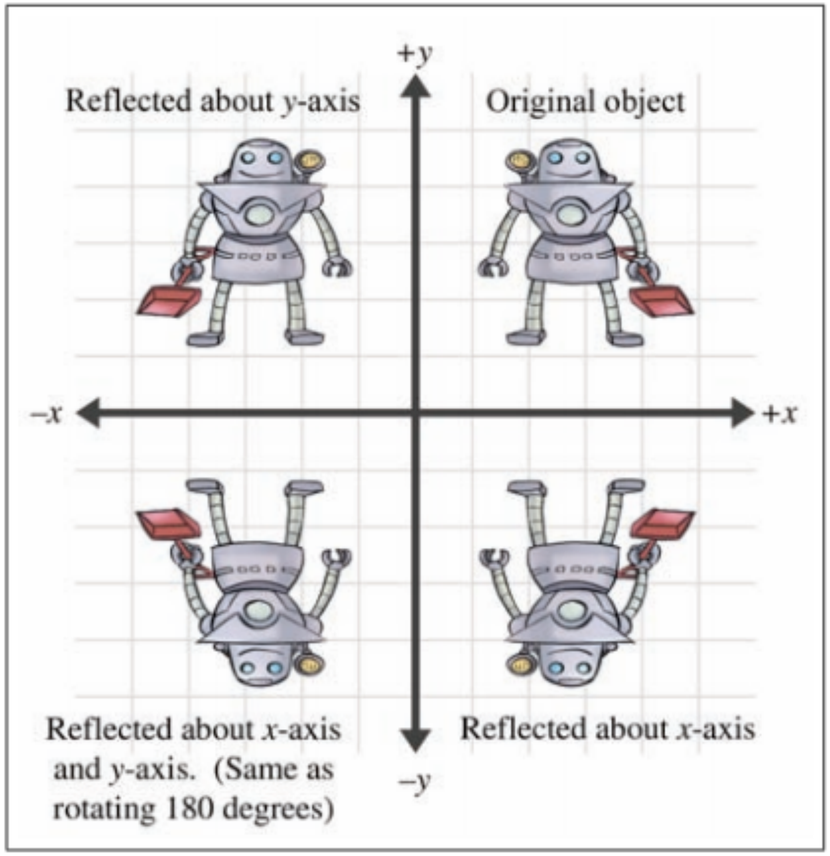
$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x\ x \\ s_y\ y \end{bmatrix}$$



Reflection

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -x \\ y \end{bmatrix}$$

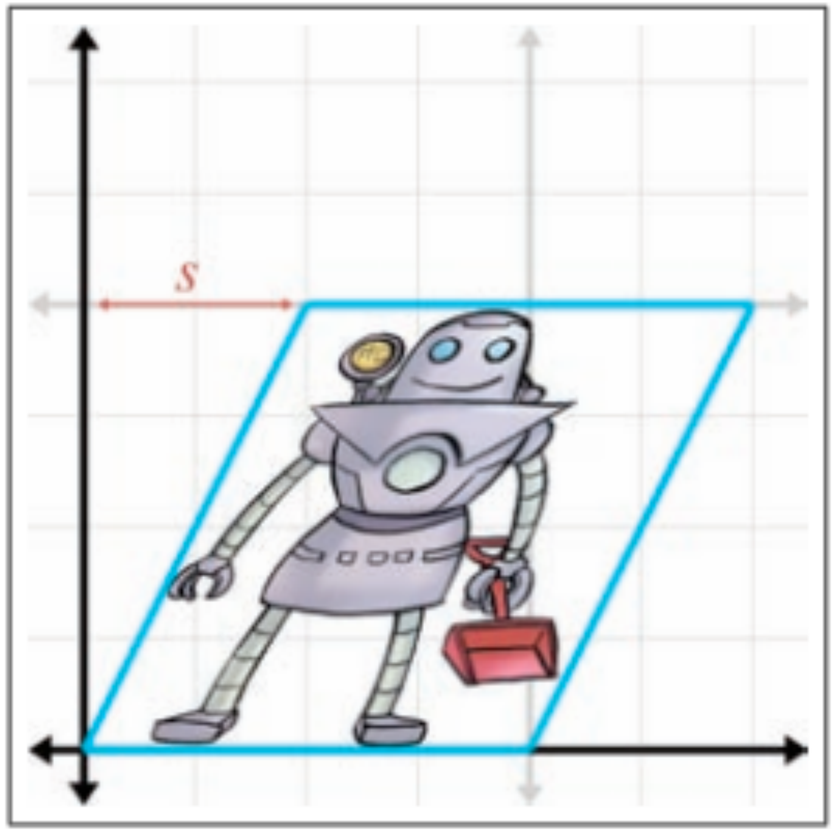
$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ -y \end{bmatrix}$$



Shearing

$$\begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + sy \\ y \end{bmatrix}$$

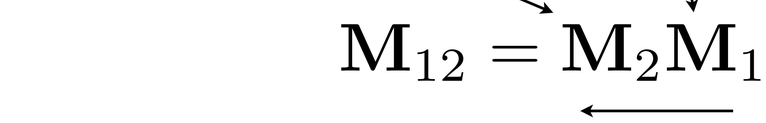
$$\begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ sx + y \end{bmatrix}$$



Sometimes called a skew transform

Composition

$$\mathbf{M}_2 = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \qquad \mathbf{M}_1 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$



$$\mathbf{M}_{12} = \mathbf{M}_2\mathbf{M}_1$$

$$\mathbf{M}_{12}\mathbf{p} = (\mathbf{M}_2\mathbf{M}_1)\mathbf{p} = \mathbf{M}_2(\mathbf{M}_1\mathbf{p})$$

But...

- Composition of transformations (matrix multiplication) is really useful
- But translation must be done as a separate operation
- *Can you include translation as a linear transformation*?

Homogeneous Coordinates

- Can include translation as a linear transformation using homogeneous coordinates
- Also useful for perspective (coming up in a bit)

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Add additional “homogeneous” element

Homogeneous Matrices

- To make a matrix work with homogeneous coordinates, add an extra row and column
- To do nothing else, make these [0 0 1]

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$$

$$\begin{bmatrix} m_{11} & m_{12} & 0 \\ m_{21} & m_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation

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

Points vs. Vectors

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{v} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

Remember: points have a position and *can* be translated,
but vectors are only a displacement and *cannot* be translated

Classes of Transformations

- Euclidean / Rigid body (preserves lengths):
  Rotation and translation

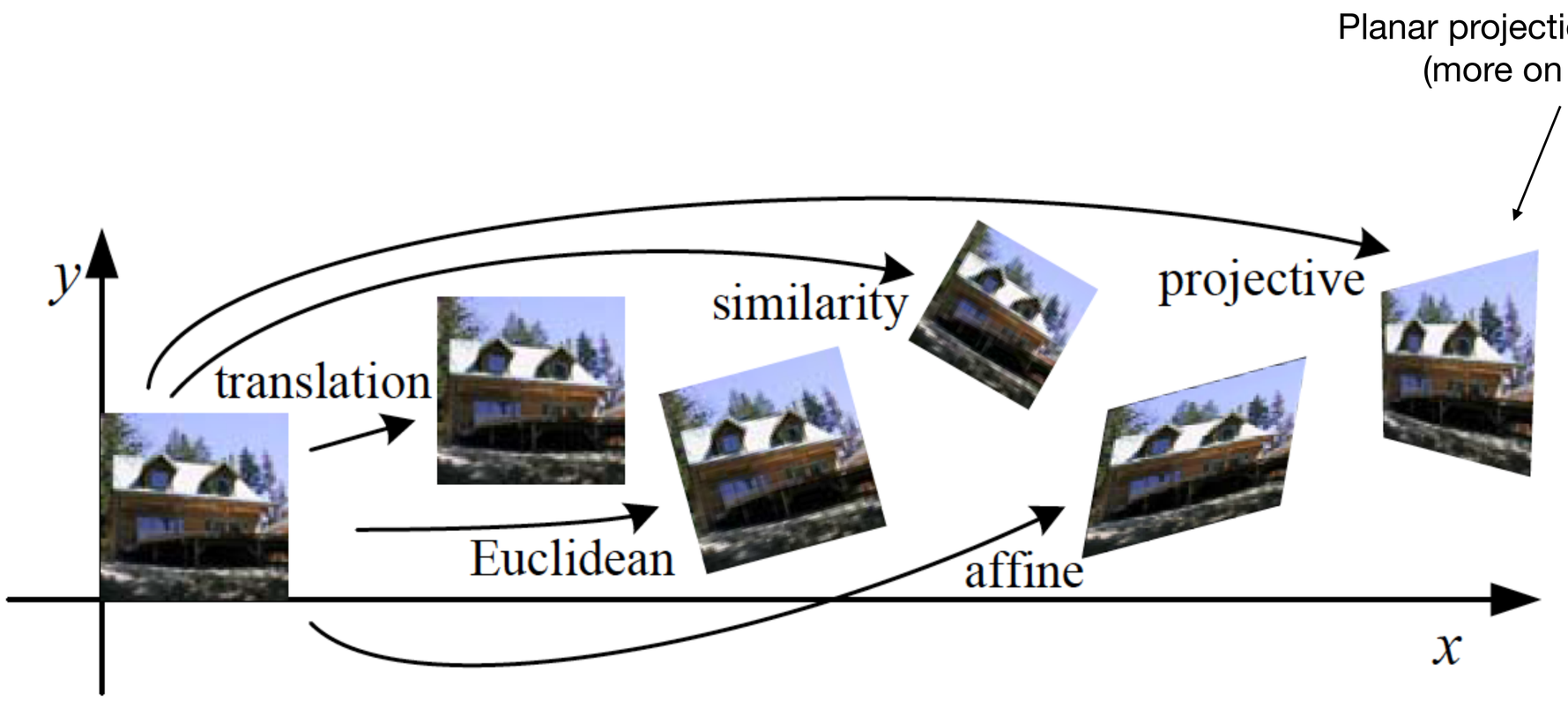
- Similarity (preserves angles):
  Rotation, translation, scale

- Affine (preserves straight lines):
  Arbitrary linear combinations

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

General form of an affine transformation

Classes of Transformations



3D Linear Transformations

- Scaling has the same form as in 2D
- Translation has the same form as in 2D
- Rotation has the same form as in 2D
  if you begin with unit vectors for the
  new coordinate axes
  - If using orthonormal coordinate systems,
    this is always an orthonormal matrix
  - Pro tip: any orthonormal matrix is doing a
    rotation or some form

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

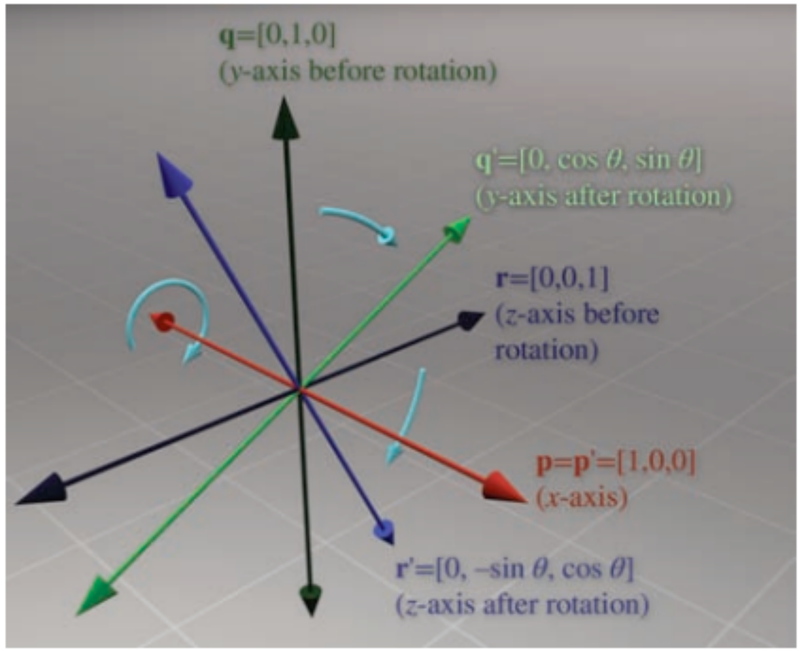
$$\mathbf{R} = \begin{bmatrix} e_{x1} & e_{x2} & e_{x3} & 0 \\ e_{y1} & e_{y2} & e_{y3} & 0 \\ e_{z1} & e_{z2} & e_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation in 3D

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

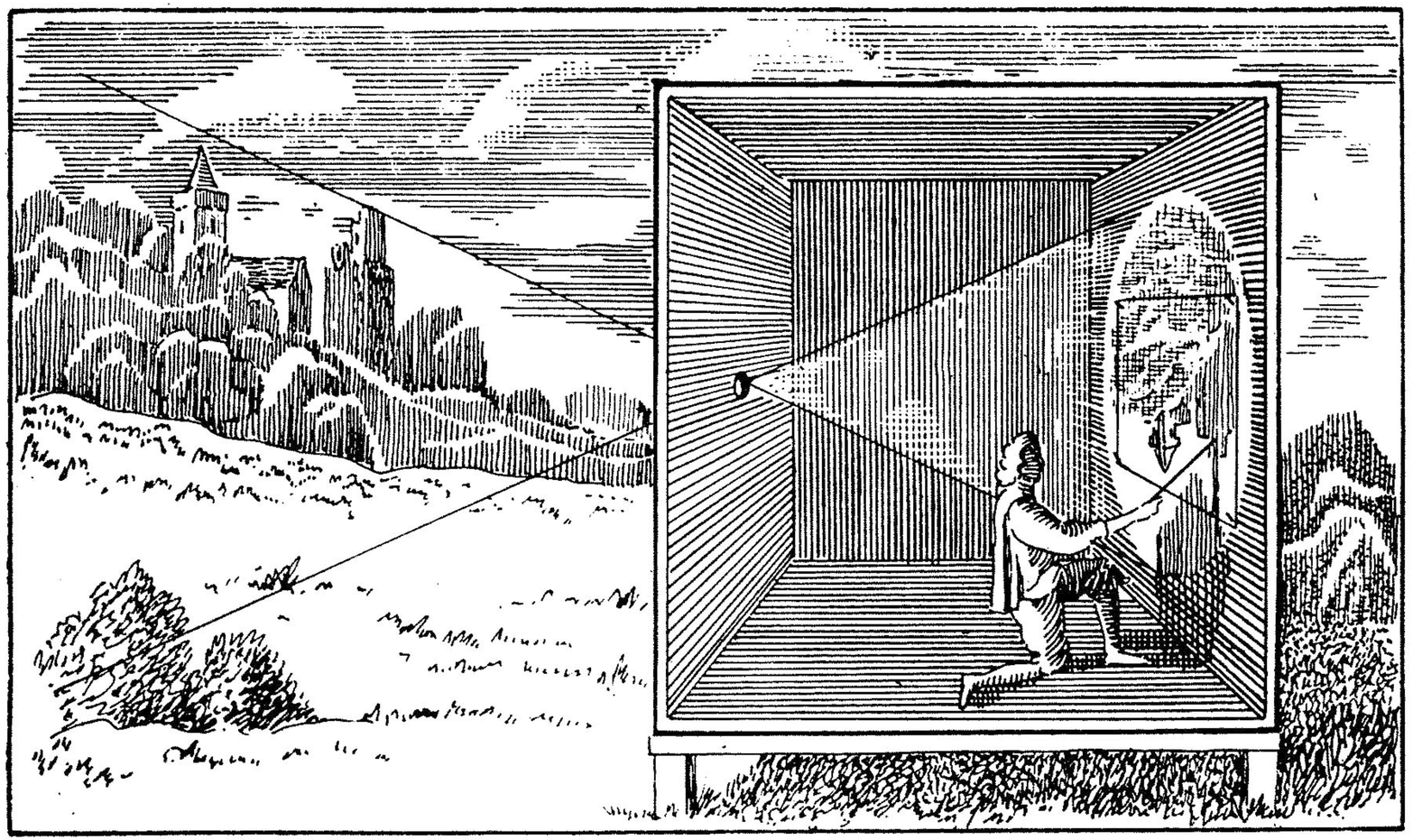


Perspective Projection

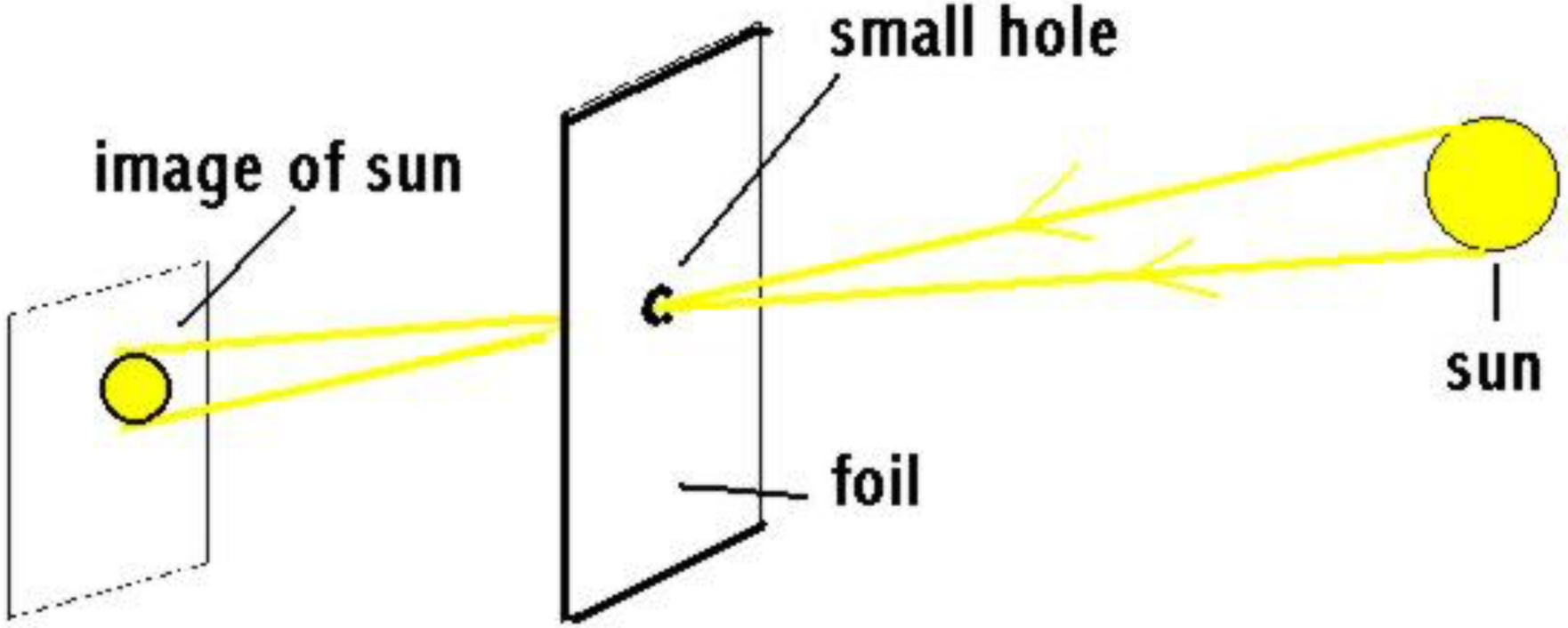
Perspective Projection

- What we're used to seeing
- Things farther away appear smaller
  (size inversely proportional to distance)
- Straight lines stay straight
- Parallel lines intersect in *vanishing points*

Camera Obscura

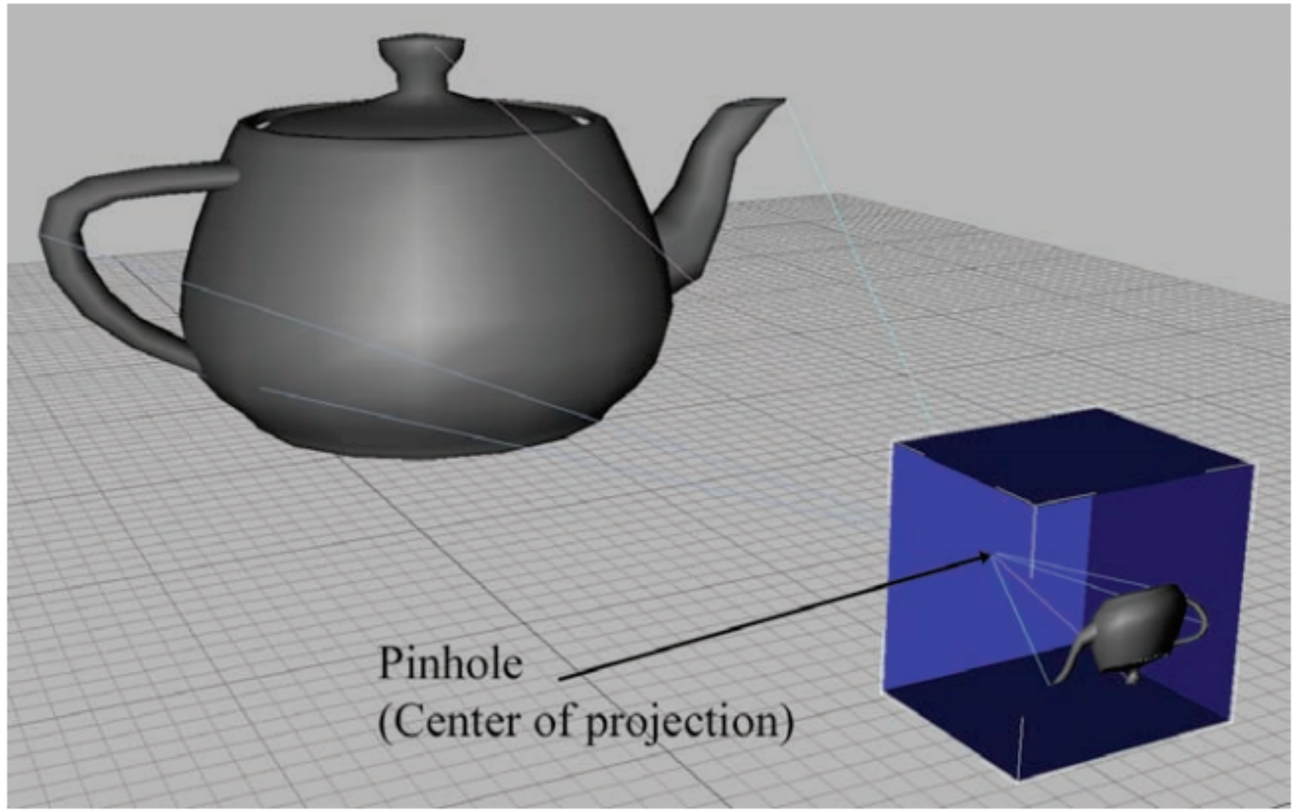


Eclipse Viewing



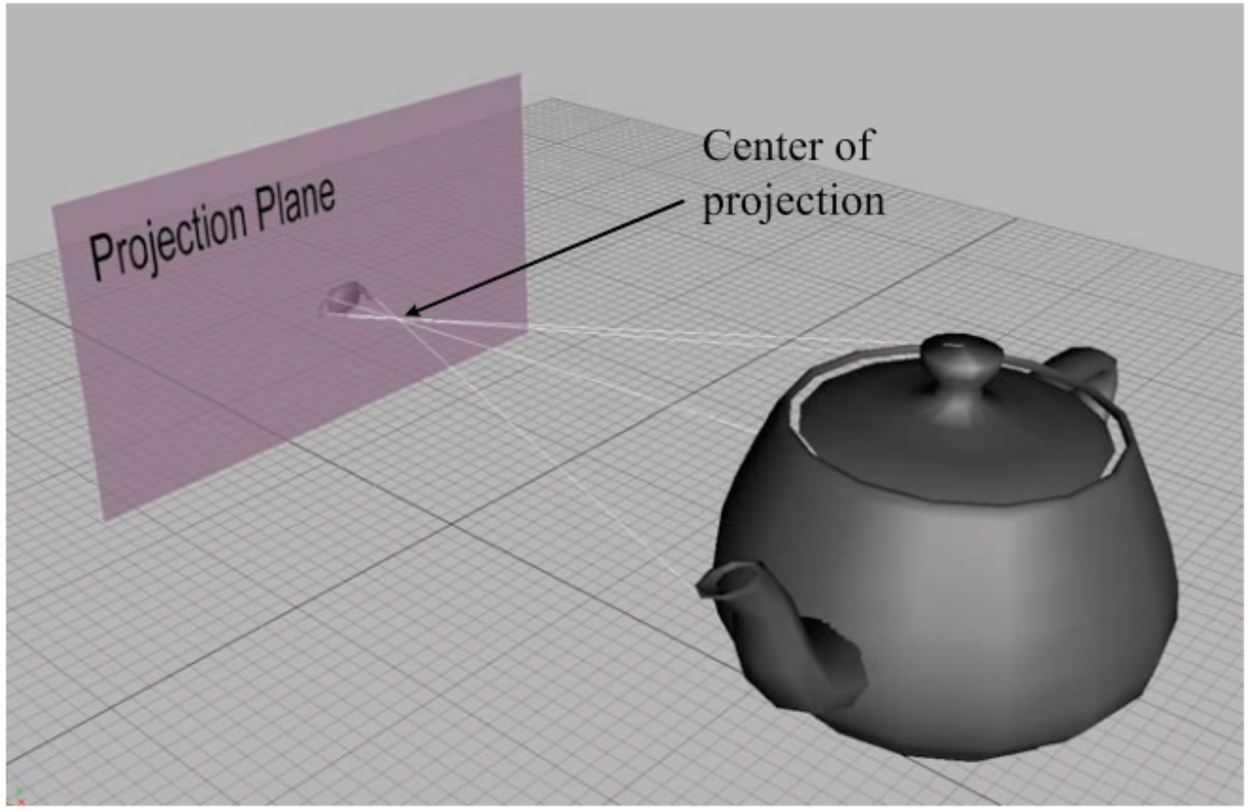
Eclipse Viewing

Perspective Projection



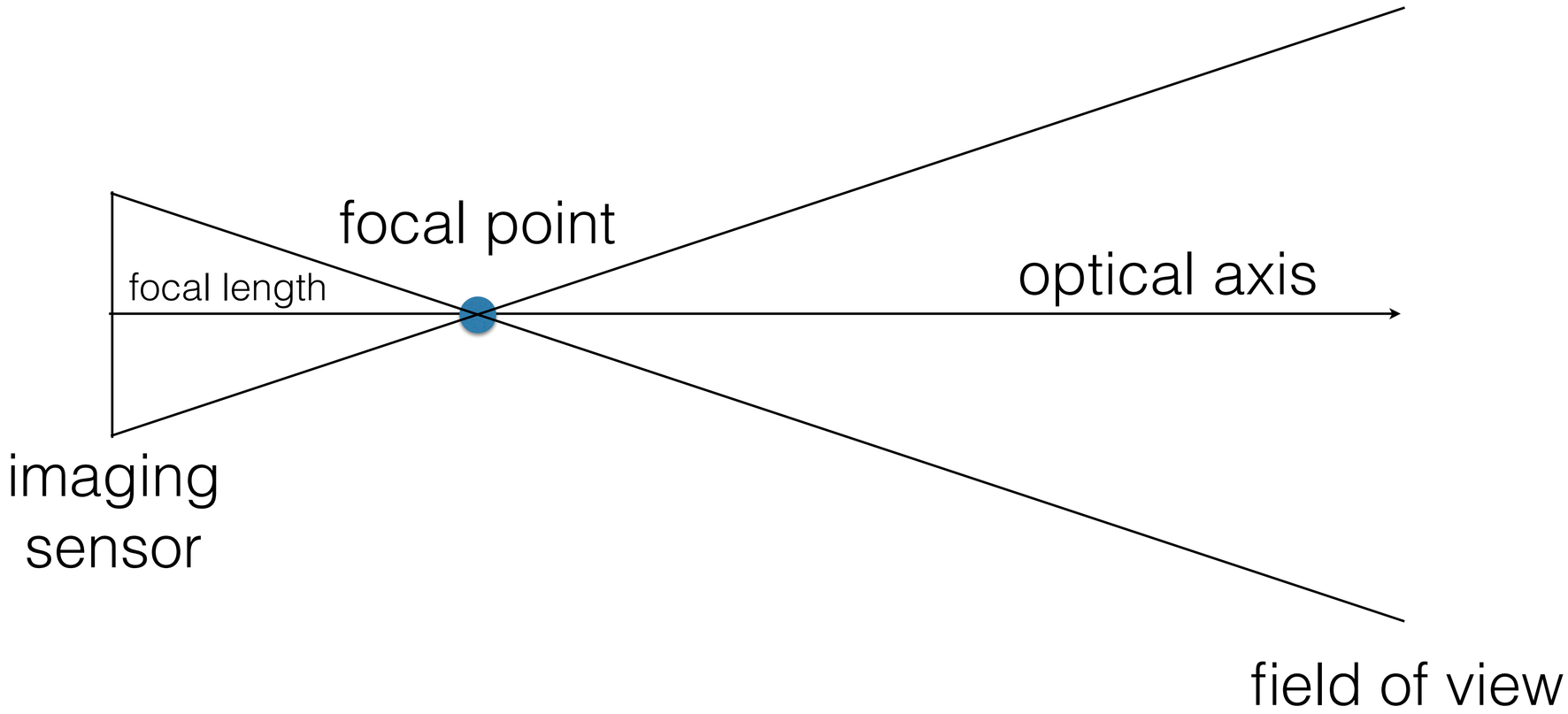
Many graphics/vision systems assume a simple pinhole camera model

Perspective Projection

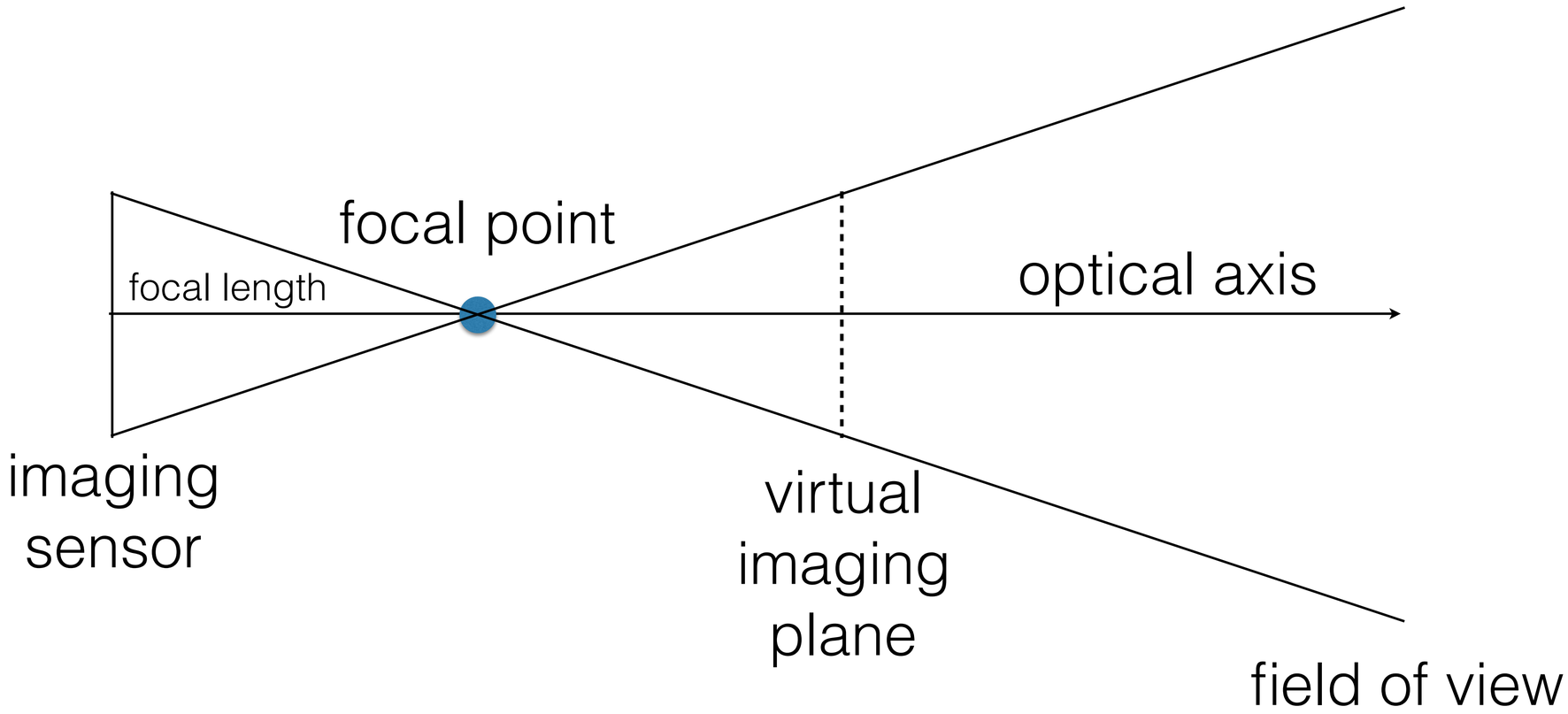


Pinhole camera model

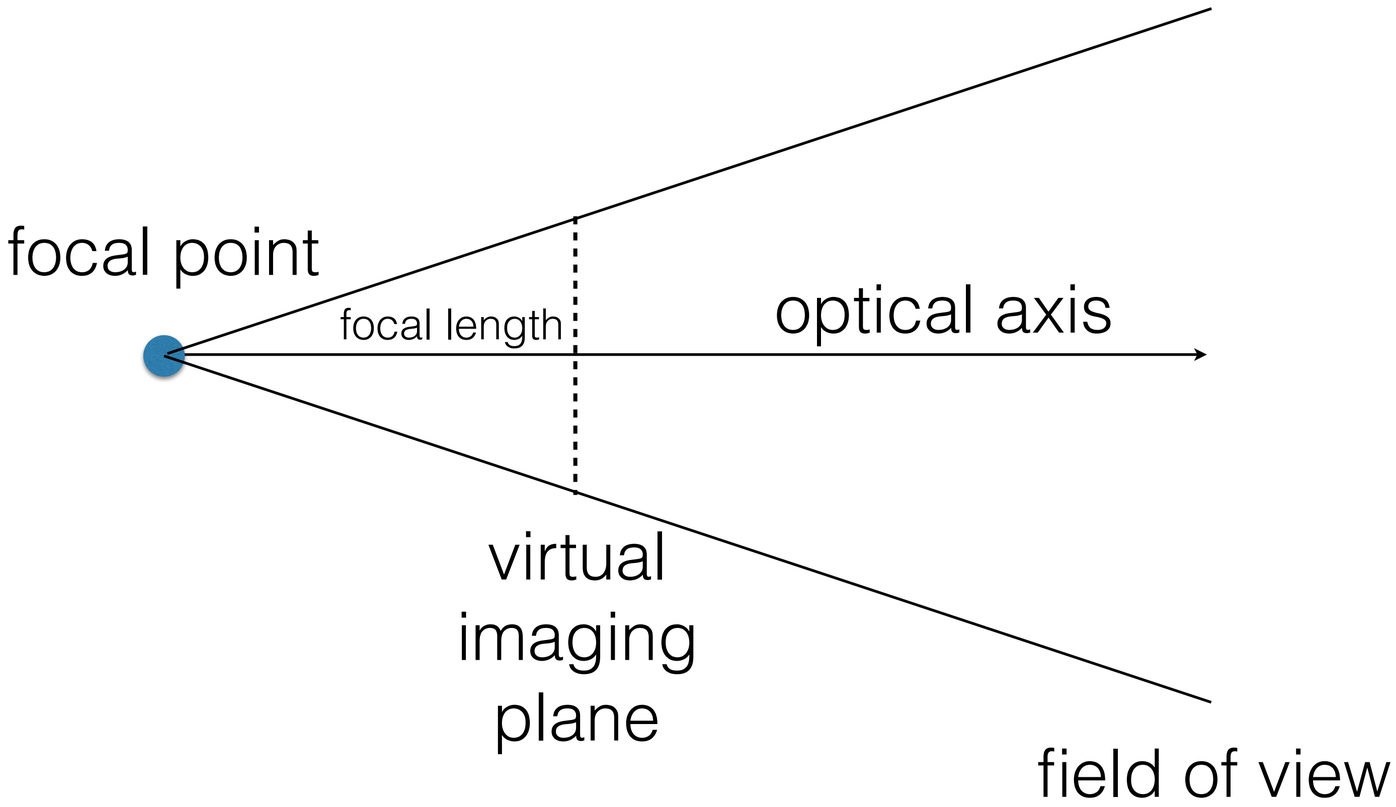
Pinhole Cameras



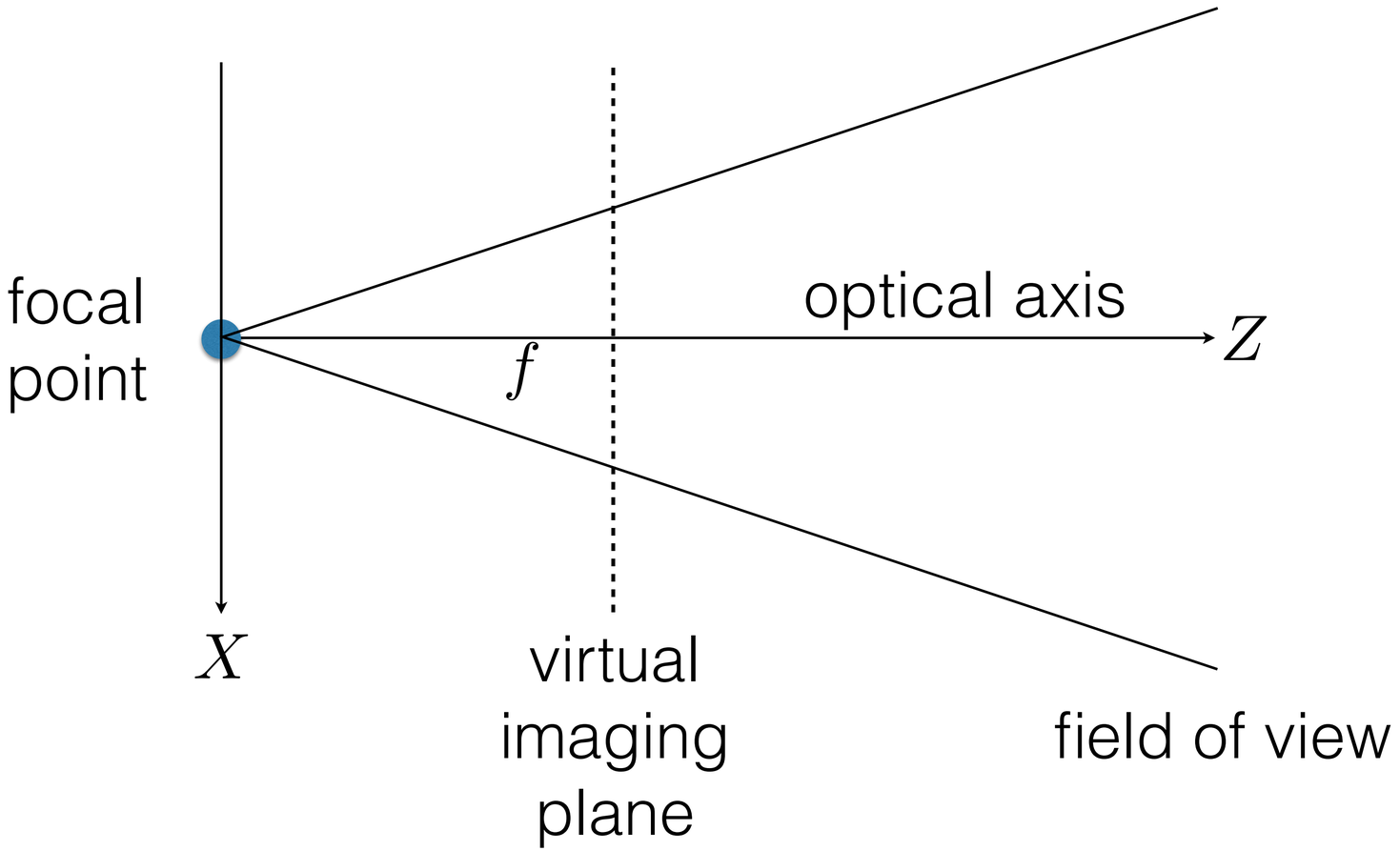
Geometric Model



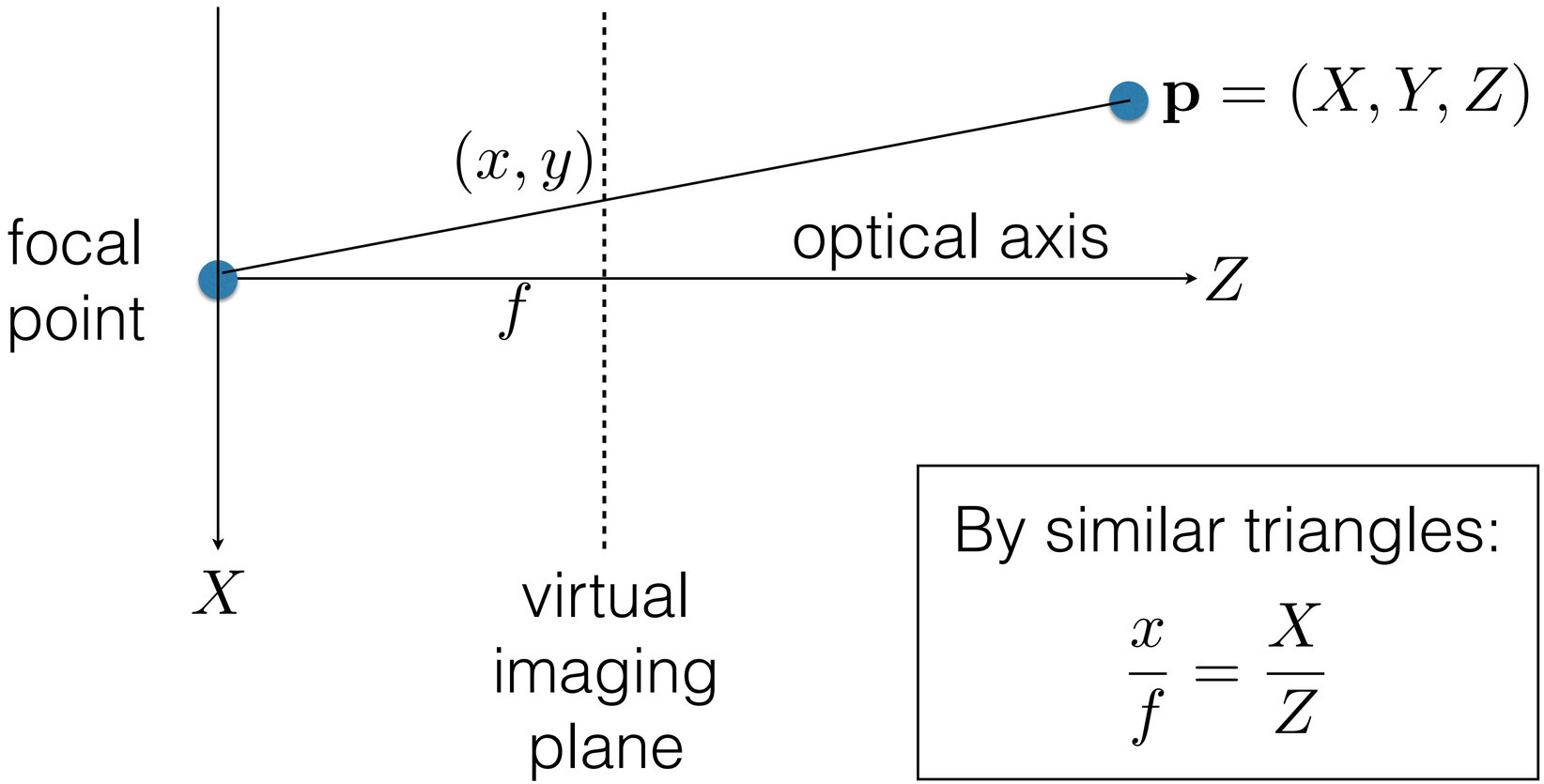
Geometric Model



Camera Coordinates



Projection



By similar triangles:

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

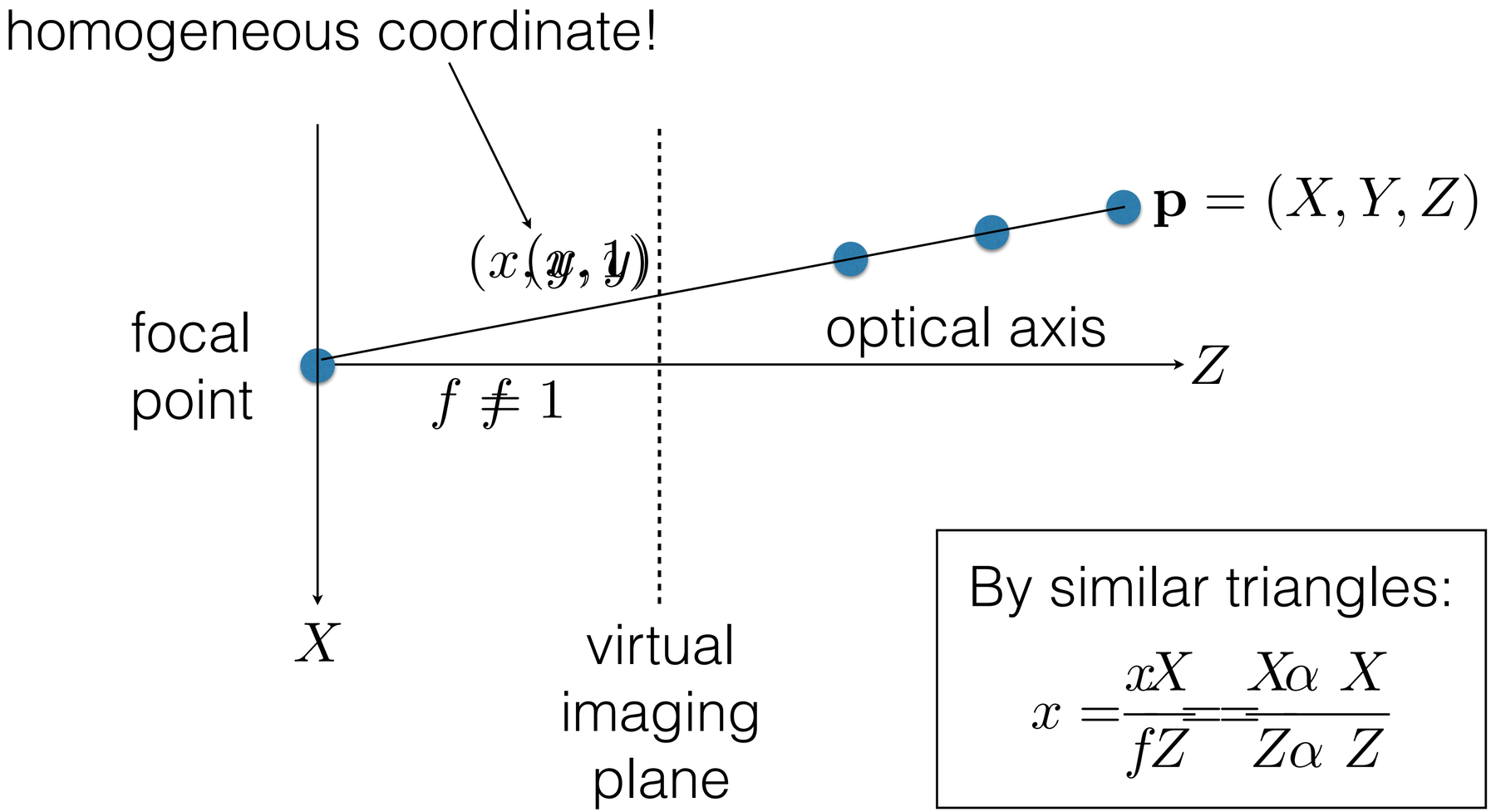
Projection

$$\frac{x}{f} = \frac{X}{Z} \qquad \frac{y}{f} = \frac{Y}{Z}$$

$$(x, y) = \left(\frac{fX}{Z}, \frac{fY}{Z} \right)$$

Note: this is the projected coordinate in real-world units.
To get actual pixel location, have to scale by pixel density and apply offset to image origin (more on this later...)

Projection



By similar triangles:

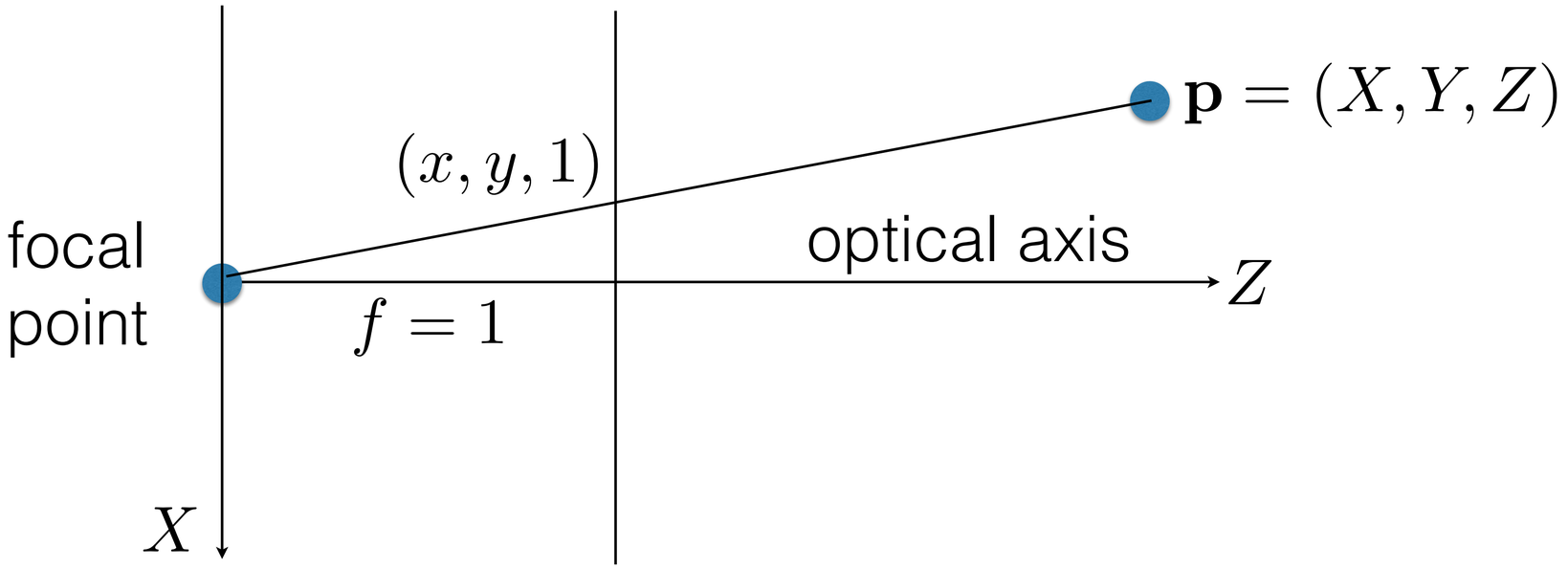
$$x = \frac{xX}{fZ} = \frac{X\alpha}{Z\alpha} = \frac{X}{Z}$$

Homogenous Coordinates

- Homogeneous coordinates are used to represent all 3D points along the ray that falls on the same 2D projection:

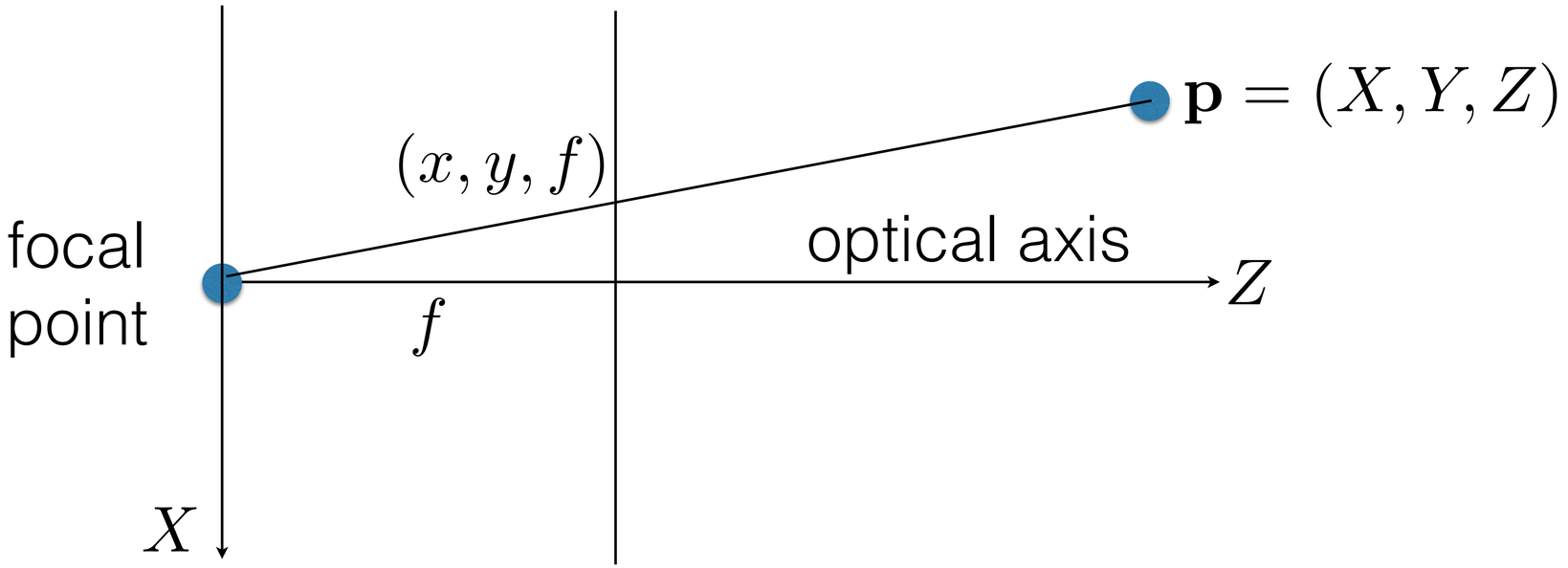
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} \alpha\ x \\ \alpha\ y \\ \alpha \end{bmatrix}$$

Perspective Projection



$$\begin{bmatrix} \boxed{\begin{matrix} x \\ y \end{matrix}} \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} X/Z \\ Y/Z \\ 1 \\ 1 \end{bmatrix} \sim \begin{bmatrix} X \\ Y \\ Z \\ Z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} X \\ Y \end{matrix}} \\ Z \\ 1 \end{bmatrix}$$

Perspective Projection



$$\begin{bmatrix} \boxed{x} \\ \boxed{y} \\ f \\ 1 \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \\ f \\ 1 \end{bmatrix} \sim \begin{bmatrix} X \\ Y \\ Z \\ Z/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \boxed{1/f} & 0 \end{bmatrix} \begin{bmatrix} \boxed{X} \\ \boxed{Y} \\ \boxed{Z} \\ \boxed{1} \end{bmatrix}$$

Alternative Form

One way (some implementation advantages):

More commonly used in graphics

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

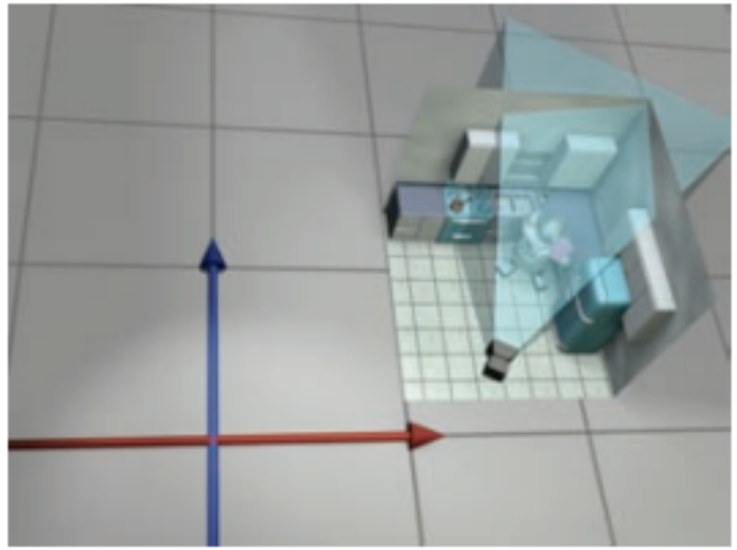
Another way (some conceptual advantages):

More commonly used in vision

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

3D Spatial Reference Frames

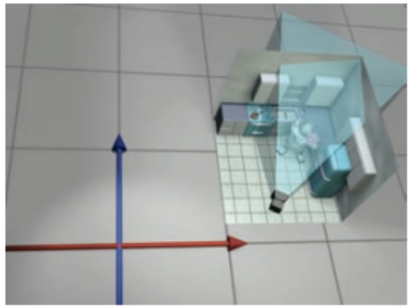
- When working with multiple cameras/views in a 3D environment, we often use multiple 3D coordinate systems (frames of reference)
  - A global “world” coordinate system
  - Each camera’s camera-centric coordinate system
- In graphics, we often use an arbitrarily chosen world coordinate system and place virtual objects in that space
- In vision, we don’t often have a global reference frame, so we often use one of the cameras as the global reference frame



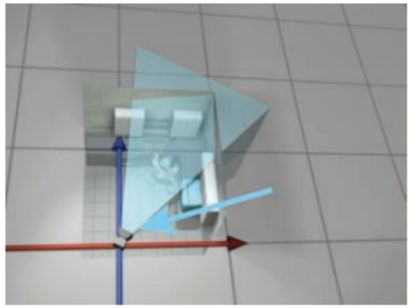
World to Camera

- Two steps:
  - **Translate**
    everything to be relative to the camera position
  - **Rotate**
    into the camera's viewing orientation

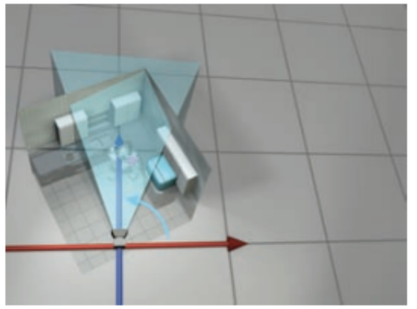
Original position



Step 1. Translate



Step 2. Rotate



World to Camera

- Two steps:
  - **Translate**
    everything to be relative to the camera position
  - **Rotate**
    into the camera's viewing orientation

$$\begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera's x
Camera's y
Camera's z

3D Perspective Projection

Let's factor this into a 3 x 4 "canonical projection matrix" and a camera-specific 2D scaling by focal length f

World-to-camera transformation

$$\begin{bmatrix} x \\ y \\ f \\ 1 \end{bmatrix} \sim \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ Z_c/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Normalize Project Rotate Translate

3D Perspective Projection

World-to-camera transformation

$$\begin{bmatrix} \boxed{\begin{matrix} x \\ y \end{matrix}} \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} X_w \\ Y_w \\ Z_w \end{matrix}} \\ 1 \end{bmatrix}$$

Camera Project Rotate Translate

K **P** **R** **T**

Intrinsic Camera Parameters

- What we've built so far works fine for modeling an ideal camera, but real ones aren't so simple:
- The optical axis through the center of the lens may not be aligned with the center of the imaging array
- The imaging sensors may not have a square aspect ratio
- The imaging array may not be mounted perpendicular to the optical axis

Intrinsic Camera Parameters

Can model these intrinsic parameters this way:

$$\mathbf{K} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} fs_x & fs & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

f	focal length (real-world units)
s_x	x scaling (pixels)
s_y	y scaling (pixels)
s	skew (pixels)
o_x	x translation (pixels)
o_y	y translation (pixels)

Can calibrate the camera to determine these parameters

3D Perspective Projection

World-to-camera transformation

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} fs_x & fs & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Camera Project Rotate Translate

K **P** **R** **T**

3D Perspective Projection

Putting it all together: $$\mathbf{p} = \mathbf{K}\ \mathbf{P}\ \mathbf{R}\ \mathbf{T}\ \mathbf{p}_w$$

Or sometimes written: $$\mathbf{p} = \mathbf{K}\ \mathbf{P} \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{p}_w$$

Or just: $$\mathbf{p} = \mathbf{M}\ \mathbf{p}_w$$

Lens Distortion

- One more geometric transformation:
  radial distortion caused by the lens:
  - Straight lines before curved
  - Usually radially symmetric
    (function of distance from the center)
  - More pronounced with larger FOV lenses
- How to fix it:
  - Model the spatial distortion
    (usually by imaging a grid of straight lines)
  - Pre-calibrate the camera/lens
    (usually with other calibration as well)
  - Warp the images to correct for the distortion
    before further processing

Camera Calibration

- Essential idea: take pictures of things with easy-to-find points/lines in known configurations

- Use these to solve for the unknowns in the pipeline

- Intrinsic calibration:
  Solving for the camera's internal parameters
  (**K** matrix, radial distortion)

- Extrinsic calibration:
  Solving for the camera's external parameters
  (position **T** and orientation **R**)