Lecture 5:

3D Rotations and Projection

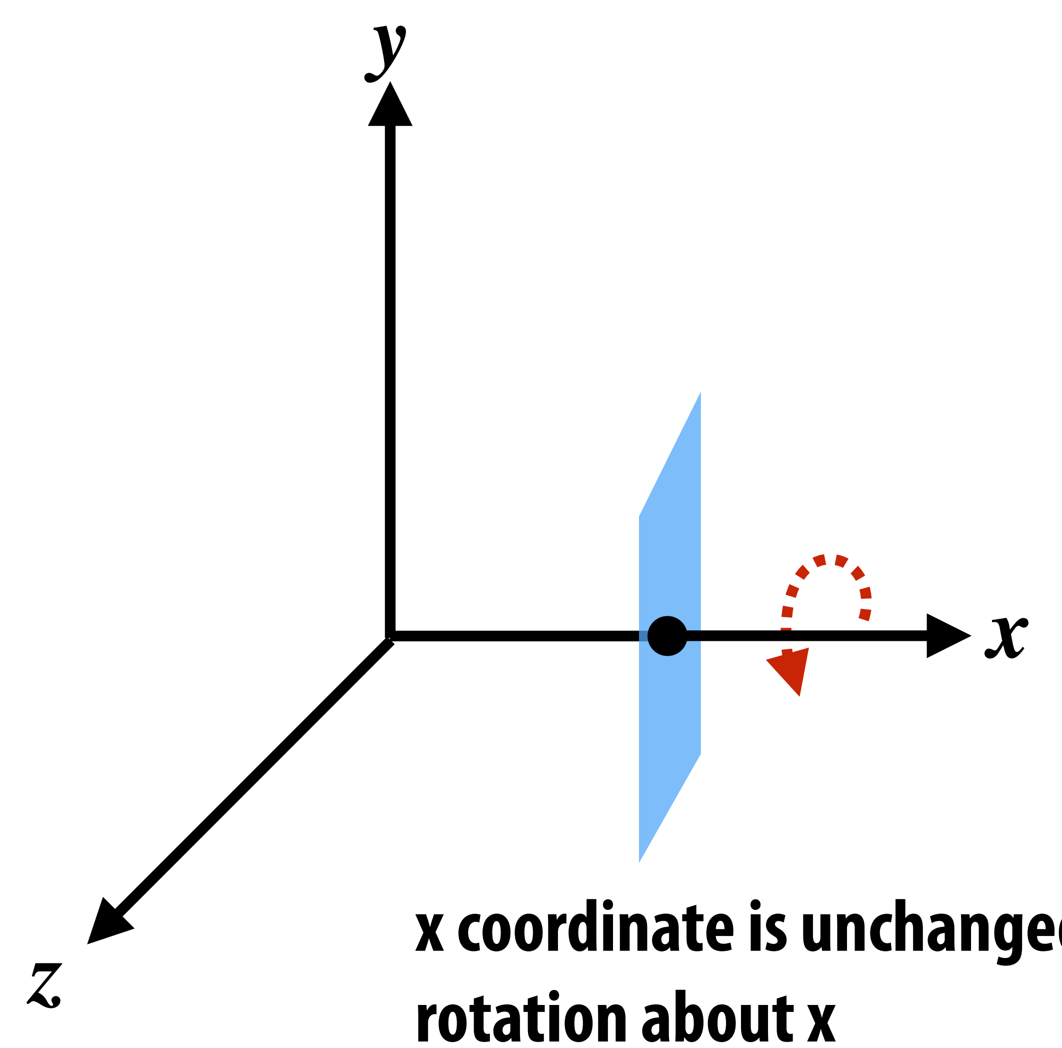
Computer Graphics

CMU 15-462/15-662, Spring 2016

Rotations in 3D

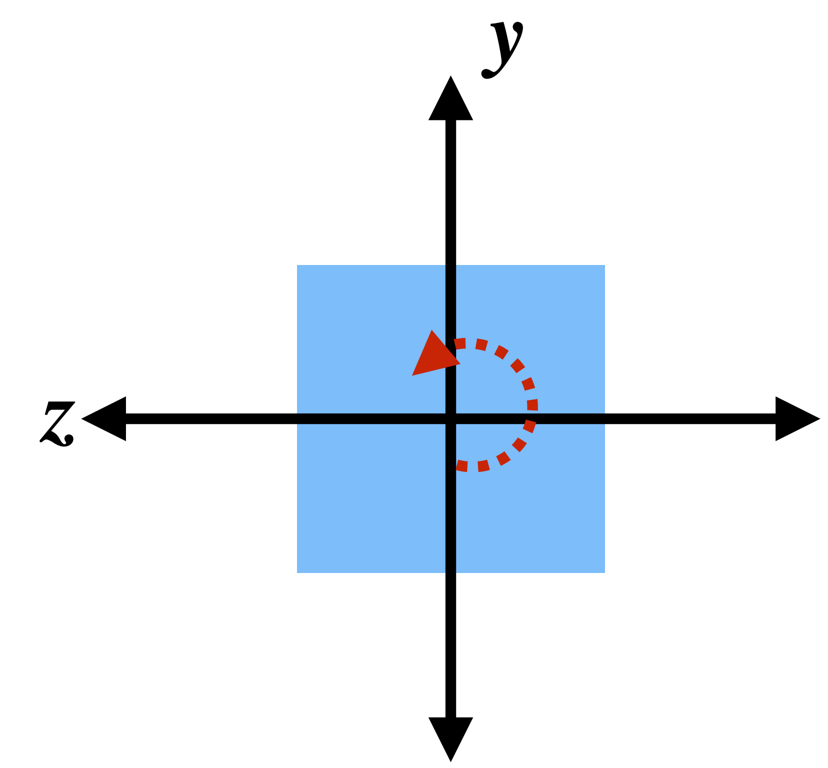
Rotation about x axis:

$$\mathbf{R}_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$



x coordinate is unchanged by rotation about x

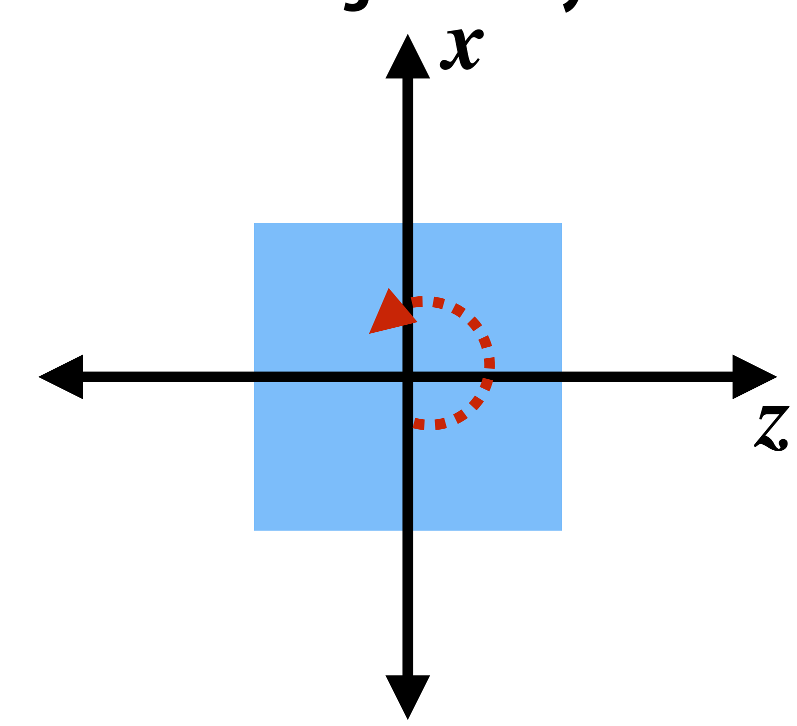
View looking down -x axis:



Rotation about y axis:

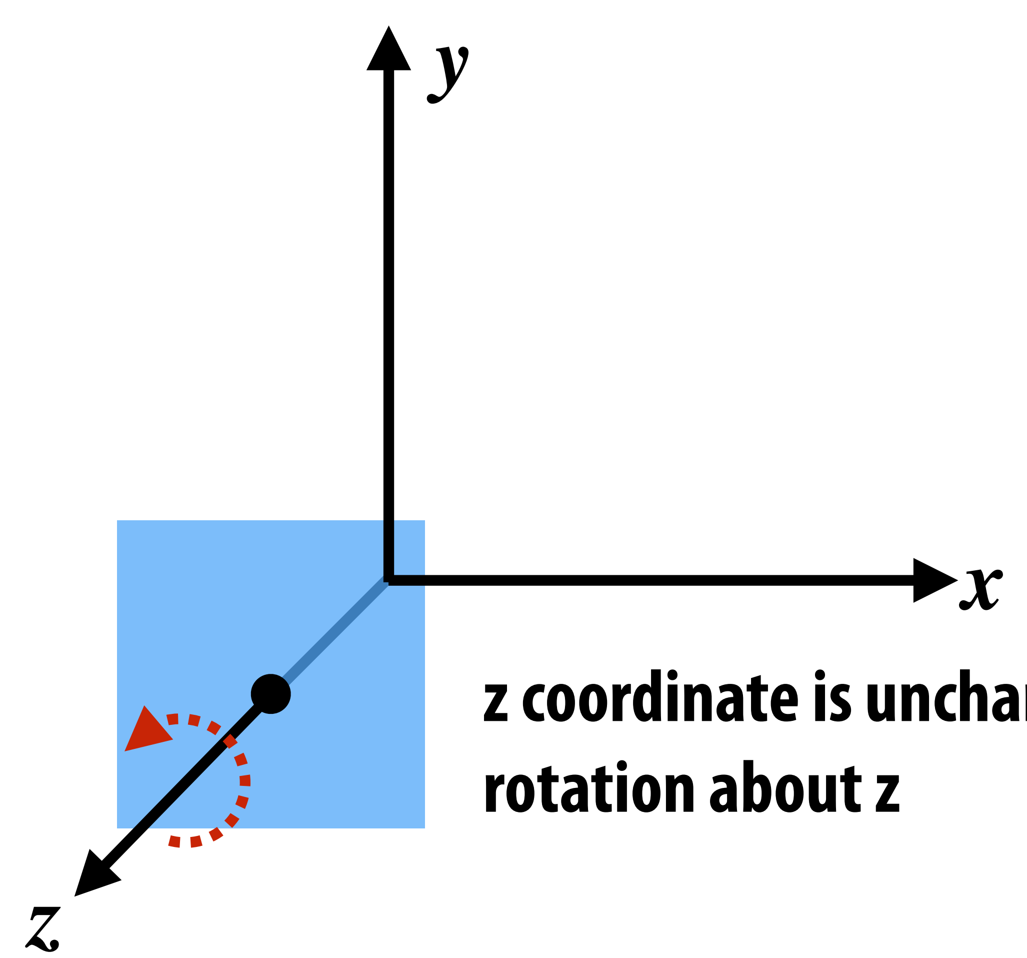
$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

View looking down -y axis:



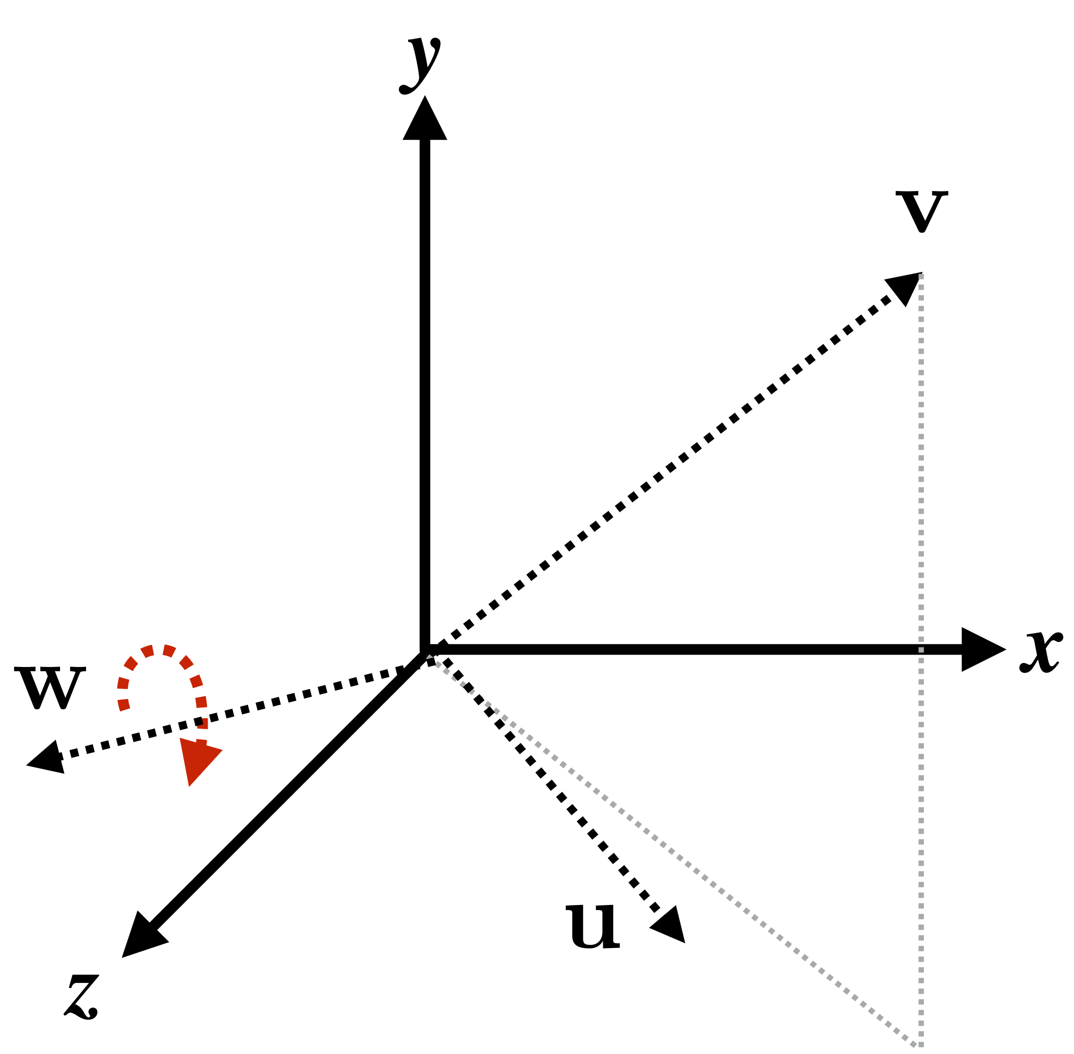
Rotation about z axis:

$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



z coordinate is unchanged by rotation about z

Rotation about an arbitrary axis



To rotate by θ about w:

1. Form orthonormal basis around w (see u and v in figure)

2. Rotate to map w to [0 0 1] (change in coordinate space)

$$\mathbf{R}_{uvw} = \begin{bmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z \\ \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z \\ \mathbf{w}_x & \mathbf{w}_y & \mathbf{w}_z \end{bmatrix}$$

$$\mathbf{R}_{uvw}\mathbf{u} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{R}_{uvw}\mathbf{v} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{R}_{uvw}\mathbf{w} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

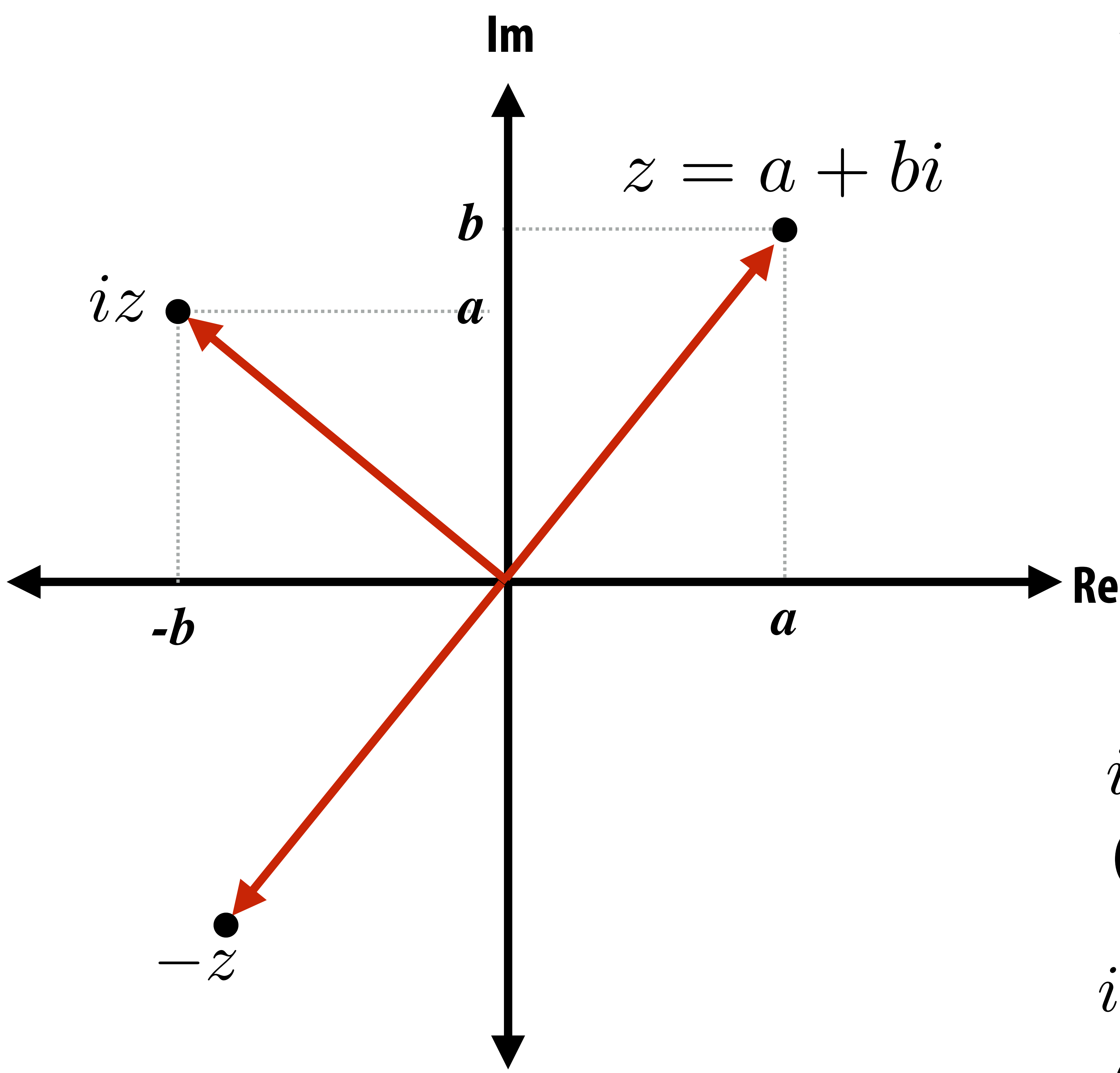
3. Perform rotation about z: $\mathrm{R}_{z,\theta}$

4. Rotate back to original coordinate space: $\mathrm{R}^{\mathbf{T}}_{\mathbf{uvw}}$

$$\mathbf{R}^{-1}_{uvw} = \mathbf{R}^{T}_{uvw} = \begin{bmatrix} \mathbf{u}_x & \mathbf{v}_x & \mathbf{w}_x \\ \mathbf{u}_y & \mathbf{v}_y & \mathbf{w}_y \\ \mathbf{u}_z & \mathbf{v}_x & \mathbf{w}_z \end{bmatrix}$$

$$\mathbf{R}_{\mathbf{w},\theta} = \mathbf{R}^{\mathbf{T}}_{\mathbf{uvw}}\mathbf{R}_{z,\theta}\mathbf{R}_{\mathbf{uvw}}$$

Alternative representation for rotations: complex numbers



$$i^2 = -1$$

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i$$

$$iz = i(a + bi) = -b + ai$$

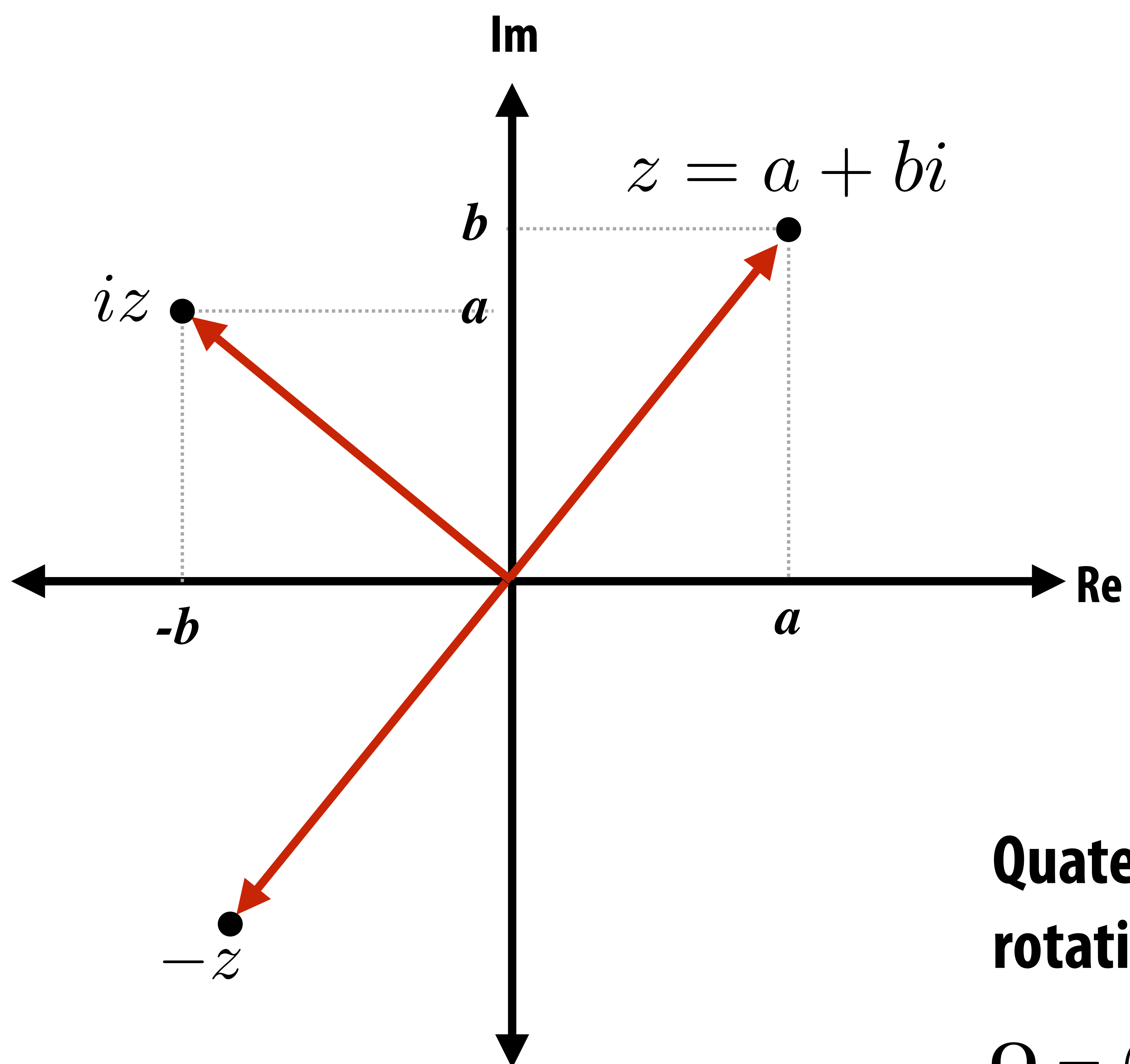
(multiplication by i → **rotation by** $\pi/2$ **)**

$$i(iz) = -a - bi = -z$$

(multiplication by i^2 → **rotation by** π **)**

$$\mathbf{R}_\theta = e^{i\theta} = \cos\theta + i\sin\theta$$

Alternative representation for rotations: complex numbers



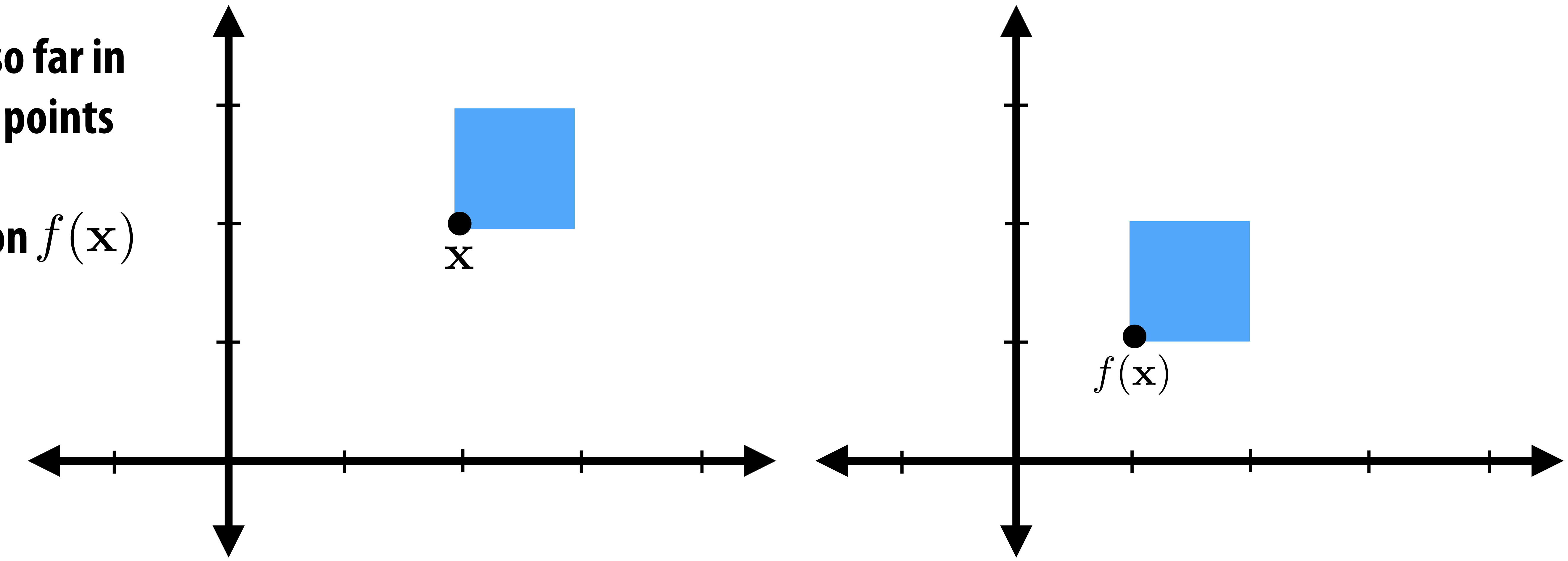
Quaternions are a representation of 3D rotations based on complex numbers

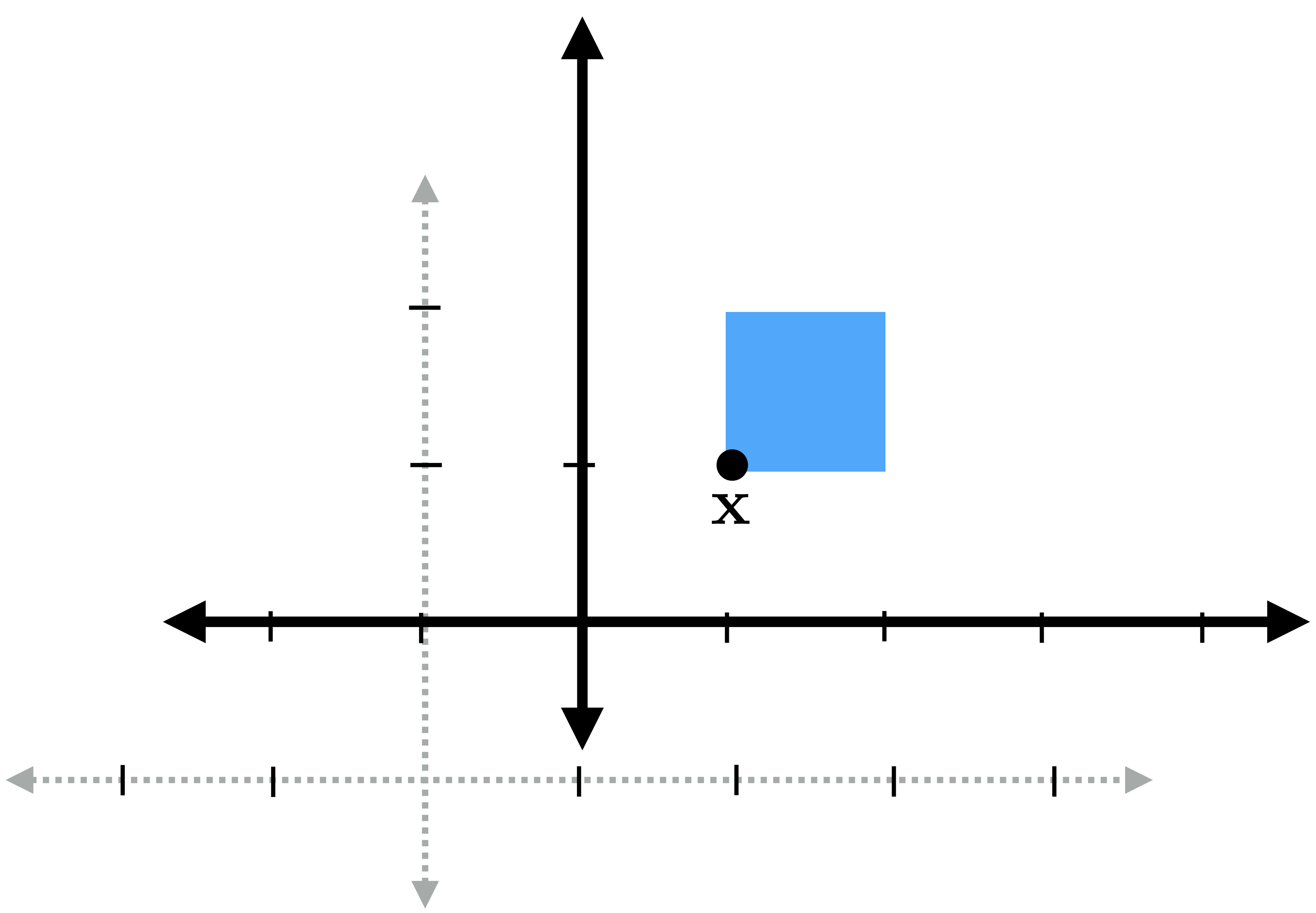
$$\mathbf{Q} = (\mathbf{q}_v, \mathbf{q}_w) = i\mathbf{q}_x + j\mathbf{q}_y + k\mathbf{q}_z + \mathbf{q}_w$$

Another way to think about transformations: change in coordinate space

Interpretation of transforms so far in this lecture: transforms move points

Point $\mathbf{x}$ moved to new position $f(\mathbf{x})$





Alternative interpretation:

Transformations induce of change of coordinate space: Representation of $\mathbf{x}$ changes since point is now described in a new coordinate space

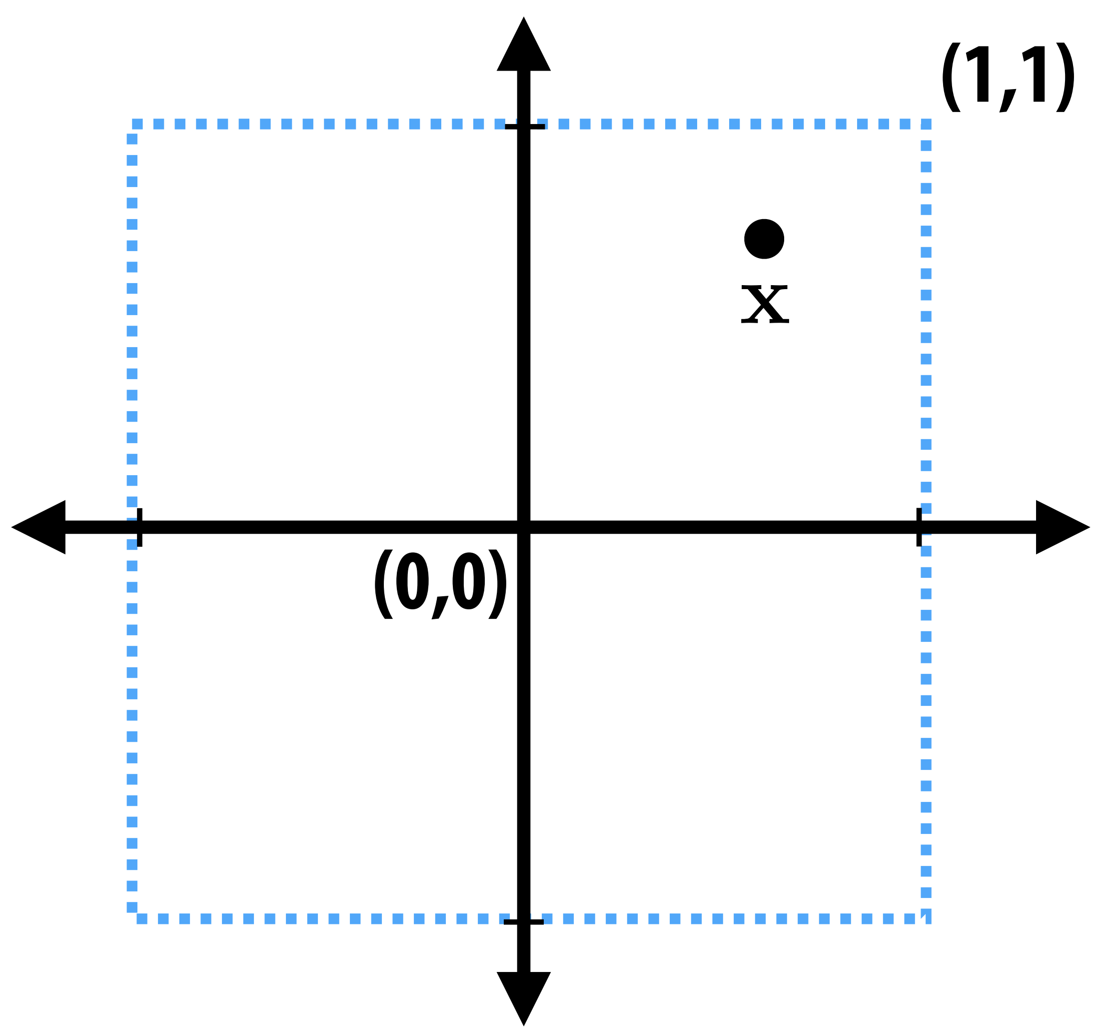
Review from last time: screen transform *

Convert points in normalized coordinate space to screen pixel coordinates

Example:

- **All points within (-1,1) to (1,1) region are on screen**
- **(1,1) in normalized space maps to (W,0) in screen**

Normalized coordinate space:

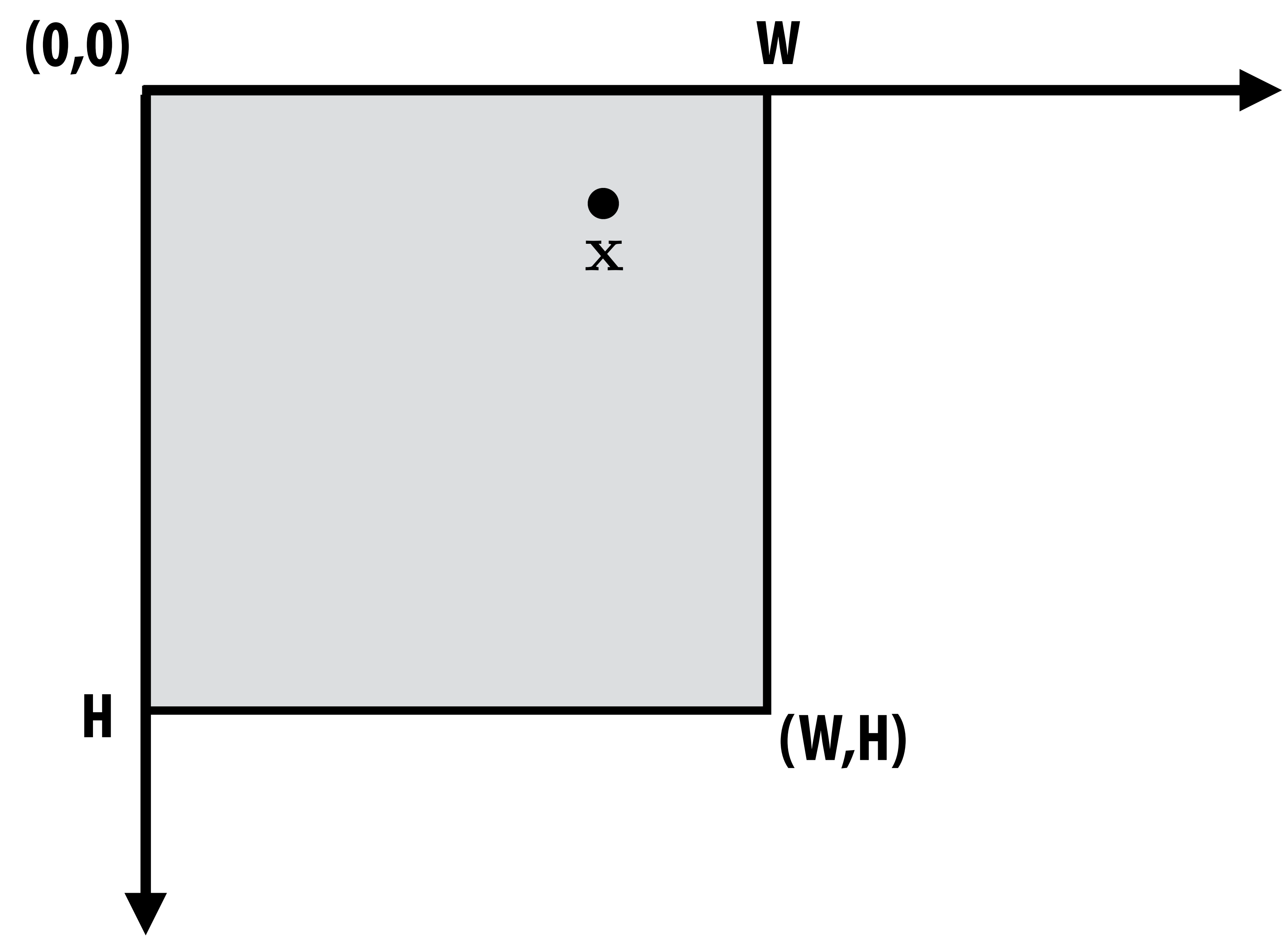


Step 1: reflect about x

Step 2: translate by (1,1)

Step 3: scale by (W/2,H/2)

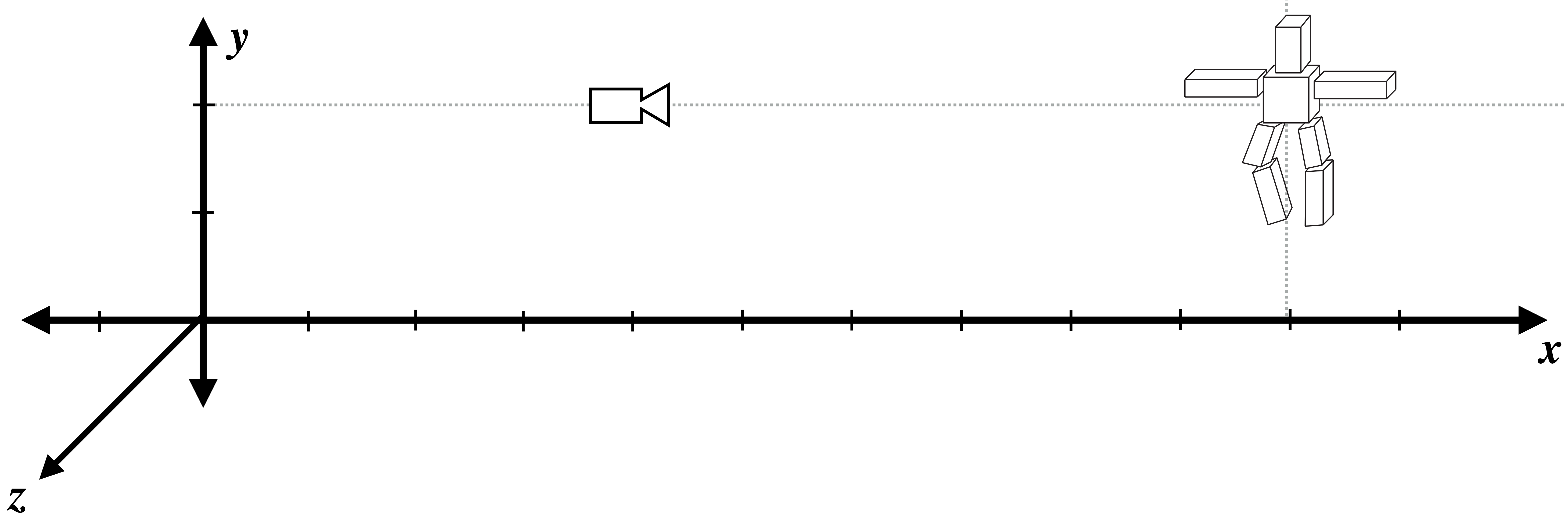
Screen (W x H output image) coordinate space:



* Adopting convention that top-left of screen is (0,0) to match SVG convention in Assignment 1.
Many 3D graphics systems like OpenGL place (0,0) in bottom-left. In this case what would the transform be?

Example: simple camera transform

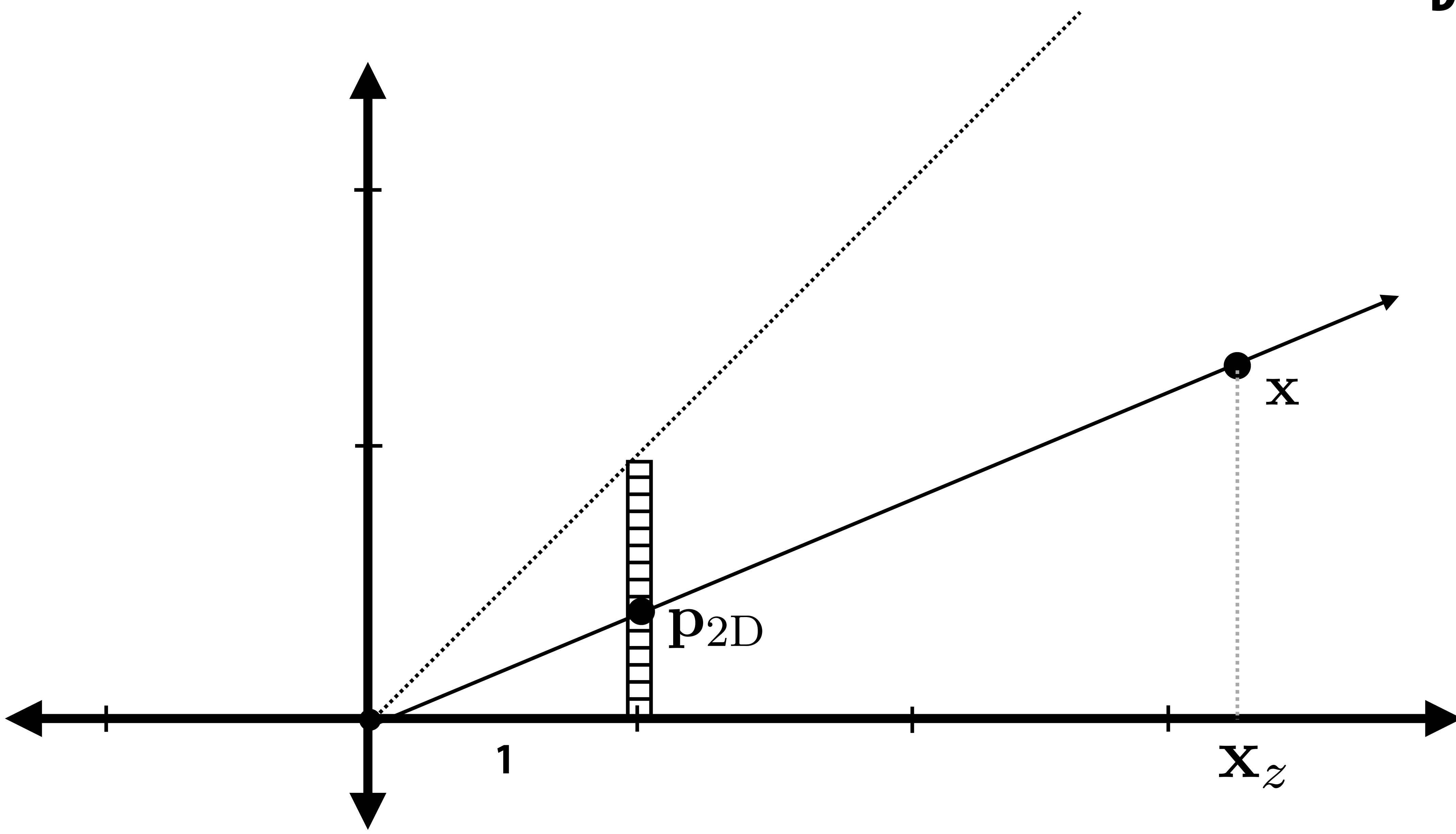
- Consider object in world at (10, 2, 0)
- Consider camera at (4, 2, 0), looking down x axis



- Translating object vertex positions by (-4, -2, 0) yields position relative to camera.
- Rotation about y by $-\pi/2$ gives position of object in coordinate system where camera's view direction is aligned with the z axis *

* The convenience of such a coordinate system will become clear on the next slide!

Basic perspective projection



Desired perspective projected result (2D point):

$$\mathbf{p}_{\text{2D}} = \begin{bmatrix} \mathbf{x}_x / \mathbf{x}_z & \mathbf{x}_y / \mathbf{x}_z \end{bmatrix}^T$$

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Input: point in 3D-H $\quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z & 1 \end{bmatrix}$

After applying $\mathbf{P}$: point in 3D-H $\quad \mathbf{P}\mathbf{x} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z & \mathbf{x}_z \end{bmatrix}^T$

Point in 2D-H (drop z coord) $\quad \mathbf{p}_{\text{2D-H}} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z \end{bmatrix}^T$

Point in 2D (homogeneous divide) $\quad \mathbf{p}_{\text{2D}} = \begin{bmatrix} \mathbf{x}_x / \mathbf{x}_z & \mathbf{x}_y / \mathbf{x}_z \end{bmatrix}^T$

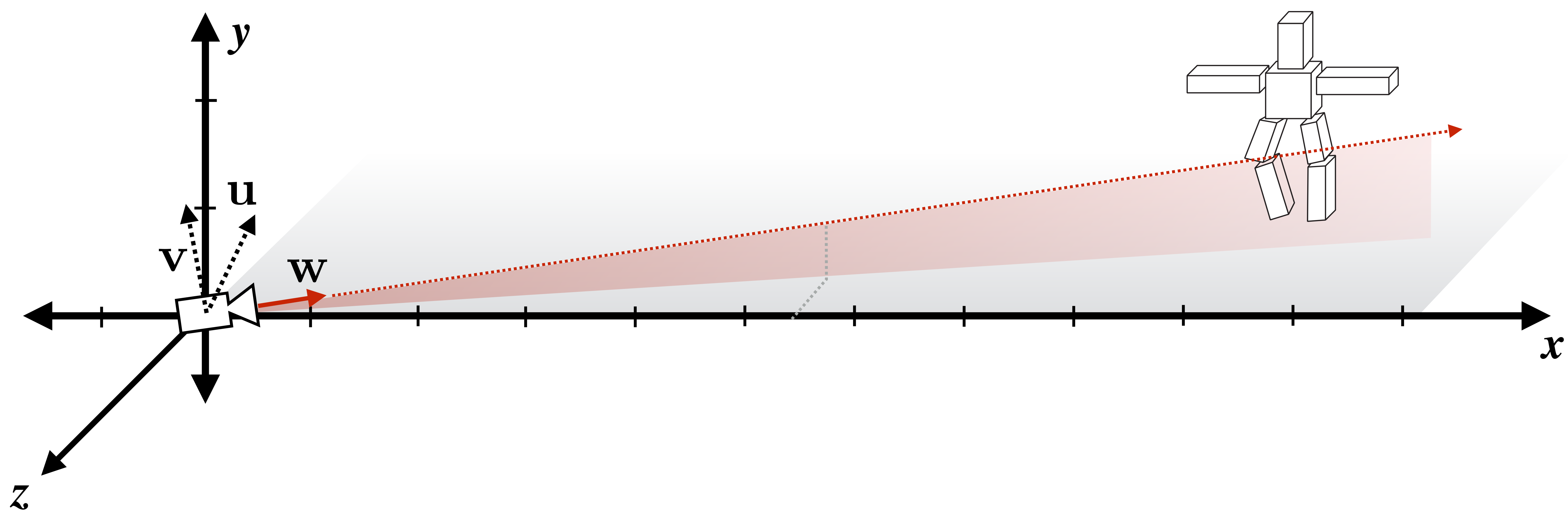
Assumption:

Pinhole camera at (0,0) looking down z

Camera with arbitrary orientation

Consider camera looking in direction $\mathbf{w}$

What transform places in the object in a coordinate space where the camera is at the origin and the camera is looking directly down the -z axis?



Form orthonormal basis around $\mathbf{w}$: (see $\mathbf{u}$ and $\mathbf{v}$)

Consider rotation matrix: $\mathbf{R}$

$$\mathbf{R} = \begin{bmatrix} \mathbf{u}_x & \mathbf{v}_x & -\mathbf{w}_x \\ \mathbf{u}_y & \mathbf{v}_y & -\mathbf{w}_y \\ \mathbf{u}_z & \mathbf{v}_z & -\mathbf{w}_z \end{bmatrix}$$

$\mathbf{R}$ maps x-axis to $\mathbf{u}$, y-axis to $\mathbf{v}$, z axis to -$\mathbf{w}$

$$\mathbf{R}^{-1} = \mathbf{R}^T = \begin{bmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z \\ \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z \\ -\mathbf{w}_x & -\mathbf{w}_y & -\mathbf{w}_z \end{bmatrix}$$

$$\mathbf{R}^T\mathbf{u} = \begin{bmatrix} \mathbf{u}\cdot\mathbf{u} & \mathbf{v}\cdot\mathbf{u} & -\mathbf{w}\cdot\mathbf{u} \end{bmatrix}^T = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$$

$$\mathbf{R}^T\mathbf{v} = \begin{bmatrix} \mathbf{u}\cdot\mathbf{v} & \mathbf{v}\cdot\mathbf{v} & -\mathbf{w}\cdot\mathbf{v} \end{bmatrix}^T = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$$

$$\mathbf{R}^T\mathbf{w} = \begin{bmatrix} \mathbf{u}\cdot\mathbf{w} & \mathbf{v}\cdot\mathbf{w} & -\mathbf{w}\cdot\mathbf{w} \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T$$

Perspective projection

Early painting: incorrect perspective

8-9th century painting

Geometrically correct perspective in art

Ambrogio Lorenzetti
Annunciation, 1344

Brunelleschi, elevation of Santo Spirito,
1434-83, Florence

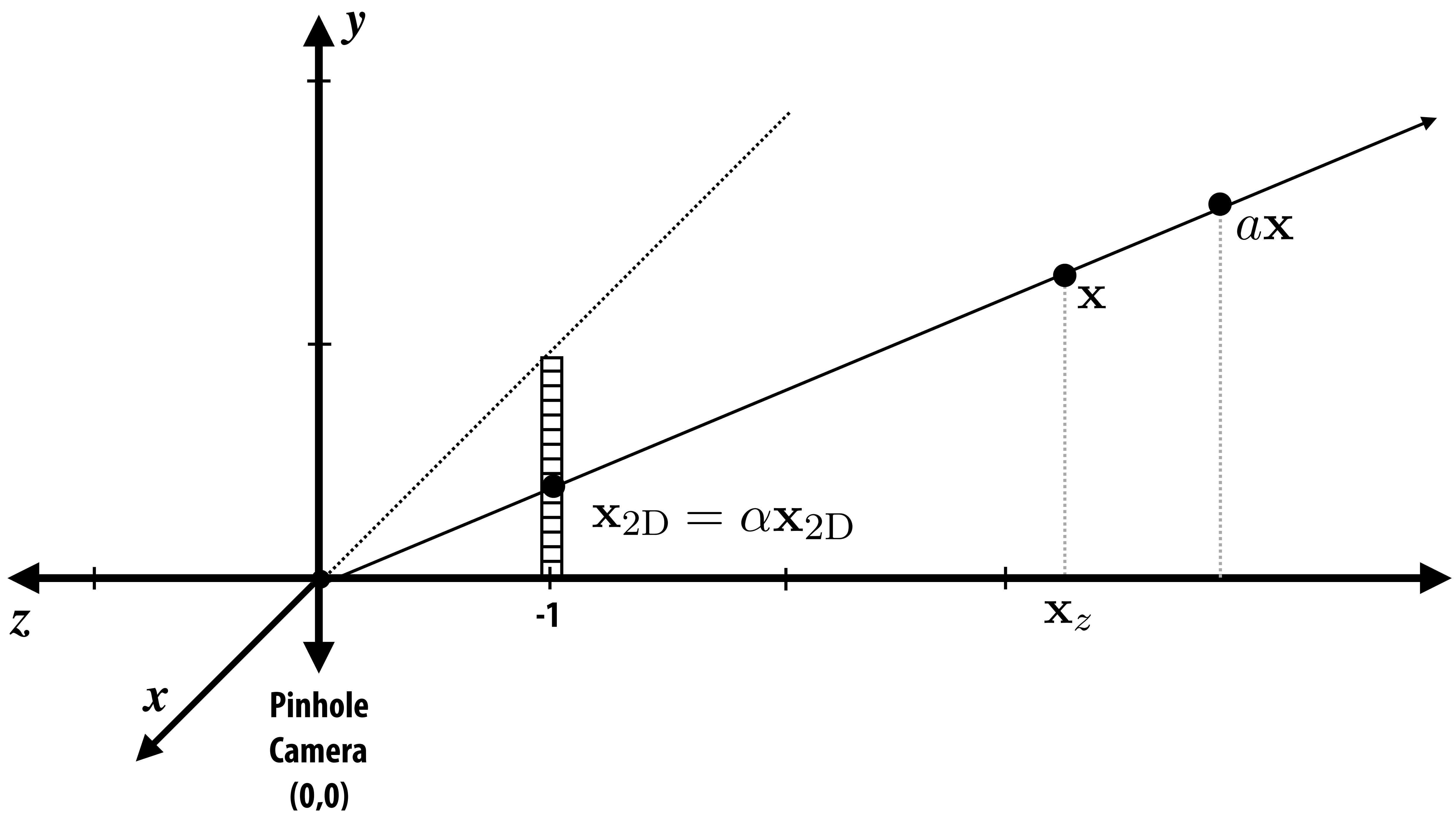
Masaccio – The Tribute Money c.1426-27
Fresco, The Brancacci Chapel, Florence

Later… rejection of proper perspective projection

Basic perspective projection

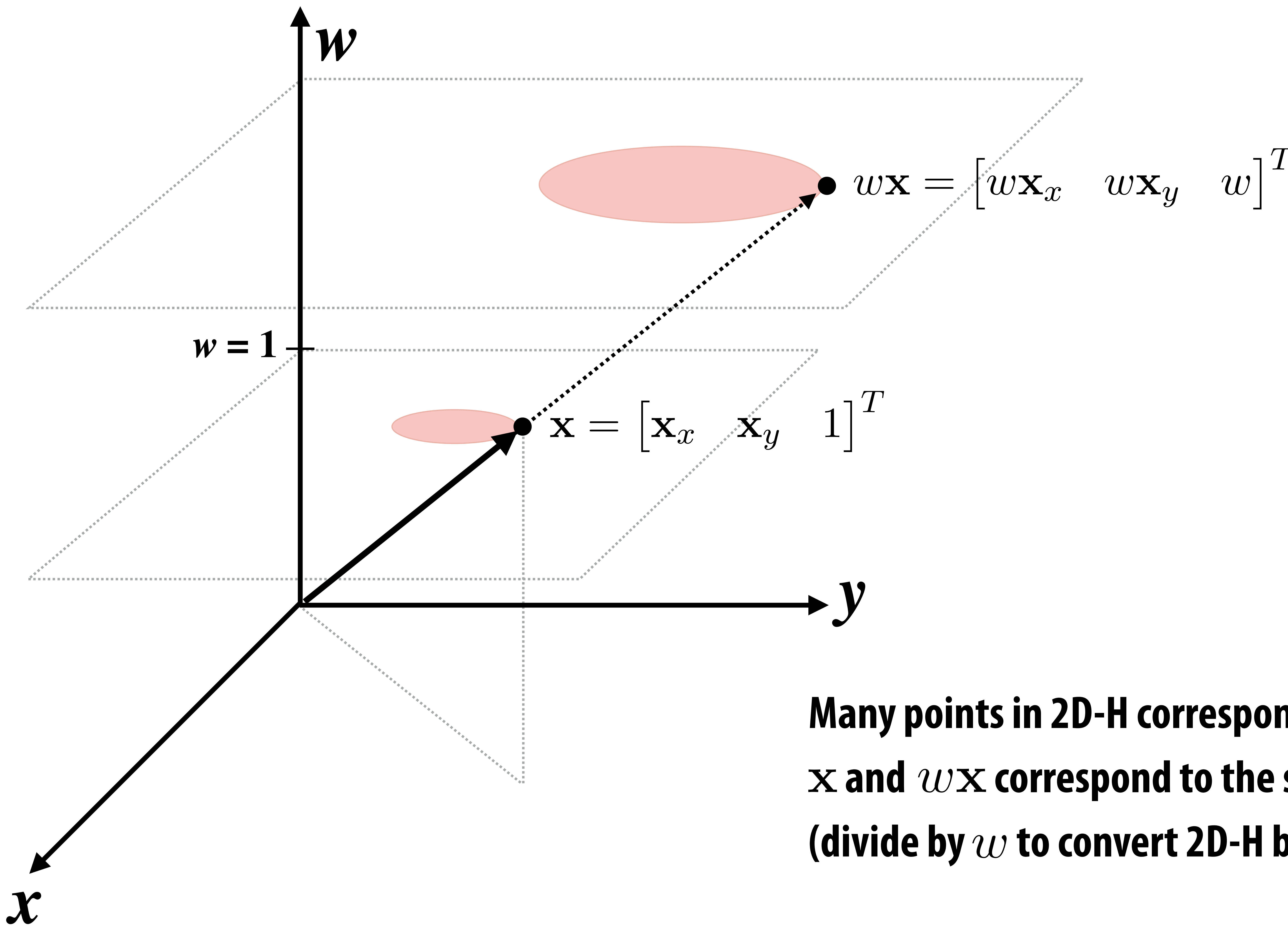
Input point in 3D-H: $\mathbf{x} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z & 1 \end{bmatrix}^T$

Perspective projected result (2D point): $\mathbf{x}_{2\mathrm{D}} = \begin{bmatrix} \mathbf{x}_x/-\mathbf{x}_z & \mathbf{x}_y/-\mathbf{x}_z \end{bmatrix}^T$



Assumption:
Pinhole camera at (0,0) looking down *-z*

Review: homogeneous coordinates



Many points in 2D-H correspond to same point in 2D

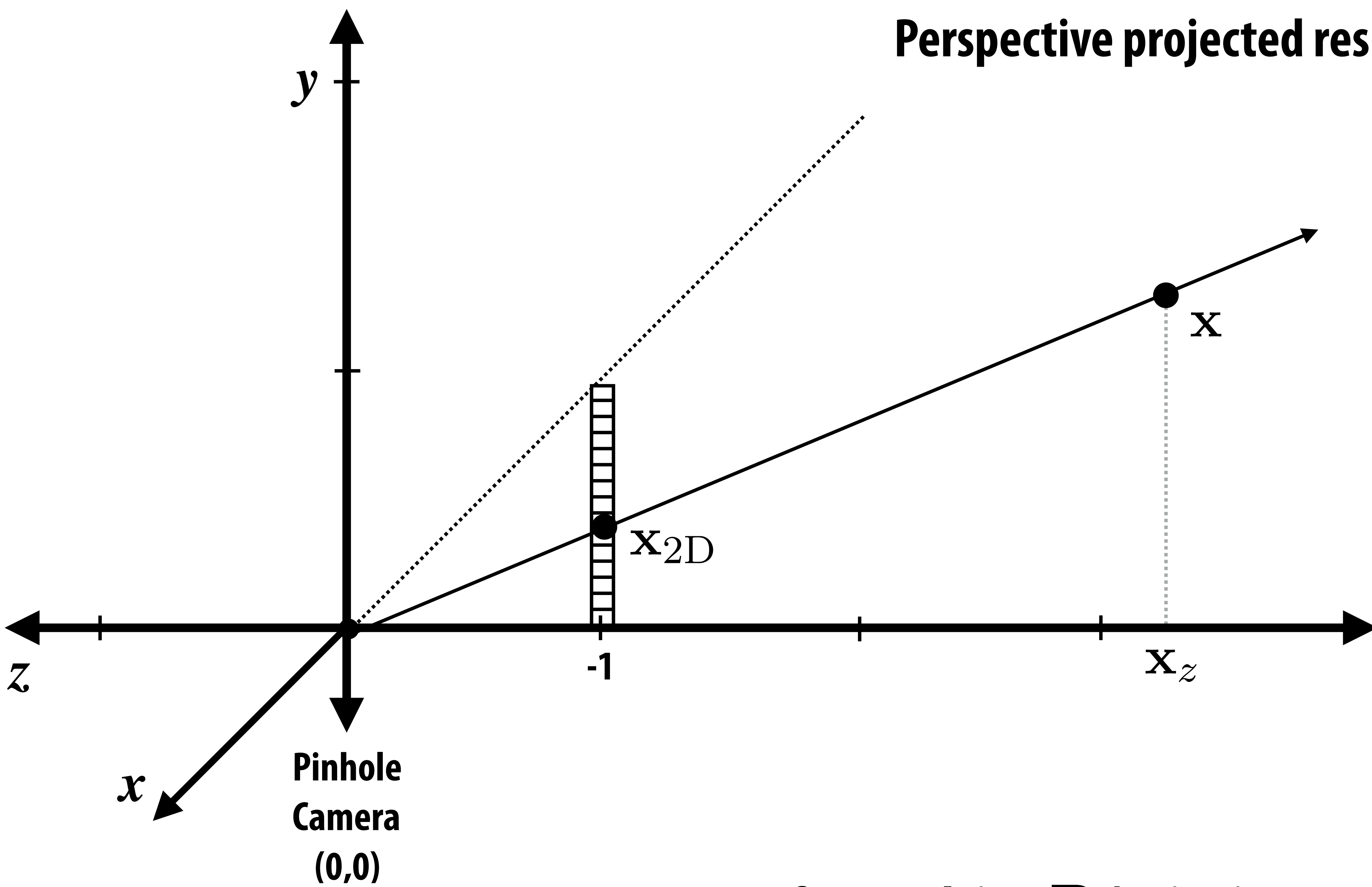
$\mathbf{x}$ and $w\mathbf{x}$ correspond to the same 2D point

(divide by w to convert 2D-H back to 2D)

Basic perspective projection

Input point in 3D-H: $\mathbf{x} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z & 1 \end{bmatrix}^T$

Perspective projected result (2D point): $\mathbf{x}_{\text{2D}} = \begin{bmatrix} \mathbf{x}_x/-\mathbf{x}_z & \mathbf{x}_y/-\mathbf{x}_z \end{bmatrix}^T$



$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

After applying $\mathbf{P}$ (point in 3D-H): $\mathbf{P}\mathbf{x} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z & -\mathbf{x}_z \end{bmatrix}^T$

Point in 2D-H (drop z coord): $\mathbf{x}_{\text{2D-H}} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & -\mathbf{x}_z \end{bmatrix}^T$

Point in 2D (after homogeneous divide): $\mathbf{x}_{\text{2D}} = \begin{bmatrix} \mathbf{x}_x/-\mathbf{x}_z & \mathbf{x}_y/-\mathbf{x}_z \end{bmatrix}^T$

Assumption:

Pinhole camera at (0,0) looking down *-z*

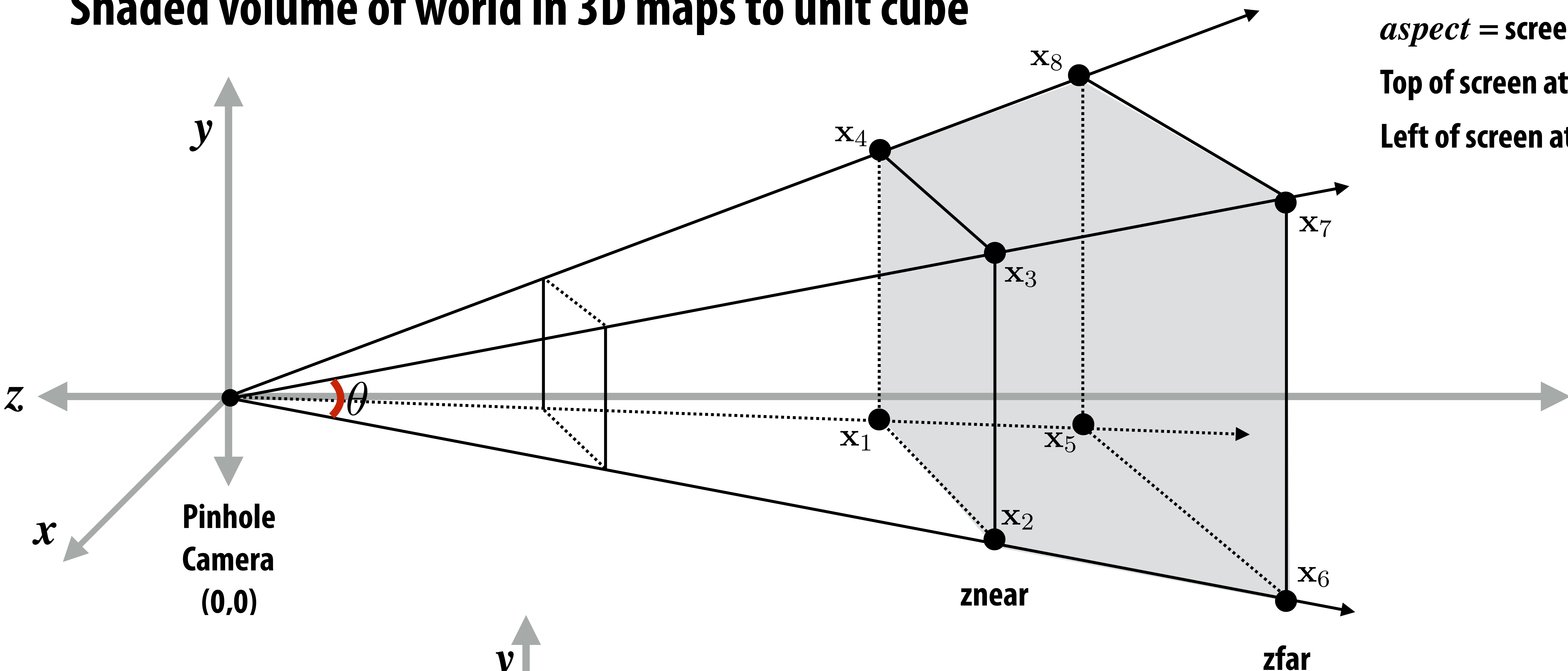
Projecting view frustum to unit cube

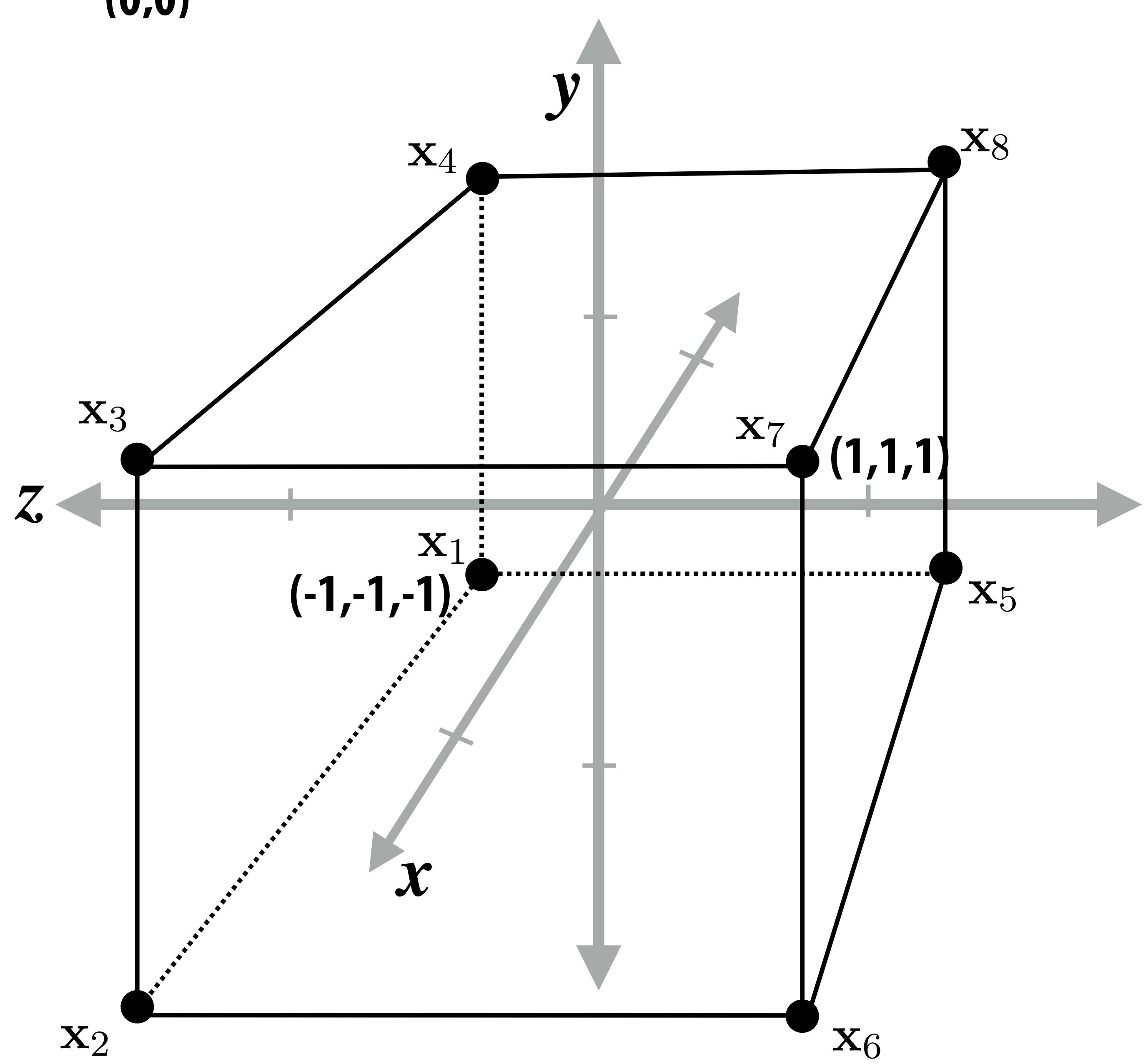
Shaded volume of world in 3D maps to unit cube

aspect = screen width / screen height

Top of screen at projection plane (z=-1): $\tan(\theta/2)$

Left of screen at projection plane: $aspect \times \tan(\theta/2)$





$$\mathbf{P} = \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{zfar+znear}{znear-zfar} & \frac{2\times zfar \times znear}{znear-zfar} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where: $f = \cot(\theta/2)$

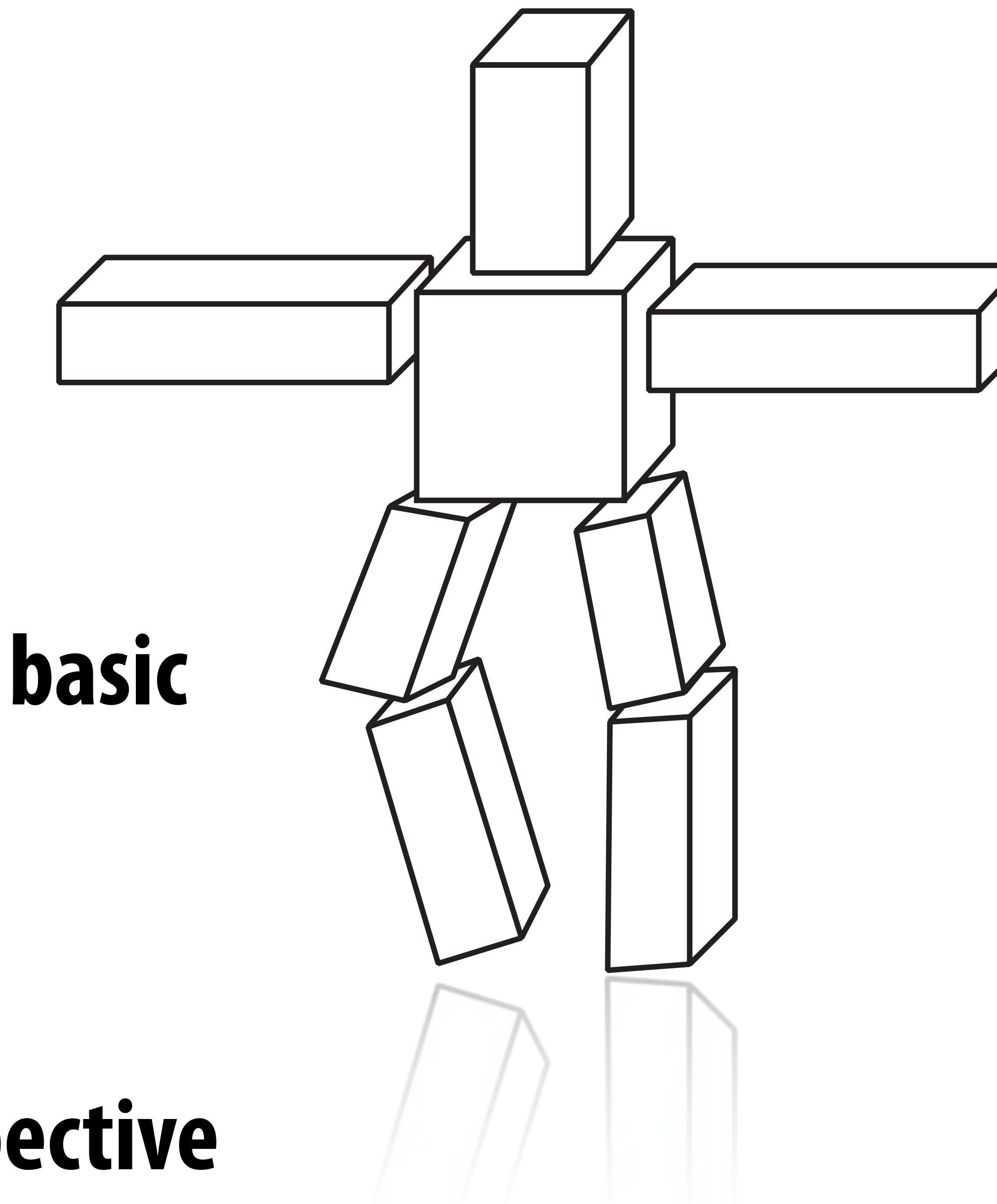
Note treatment of *z* by $\mathbf{P}$:

z = ***-znear*** **maps to -*z***

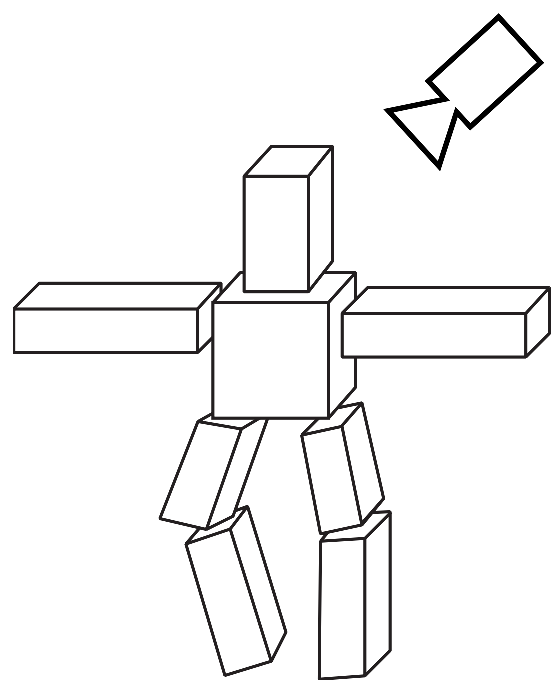
z = ***-zfar*** **maps to *z***

Tranformations summary

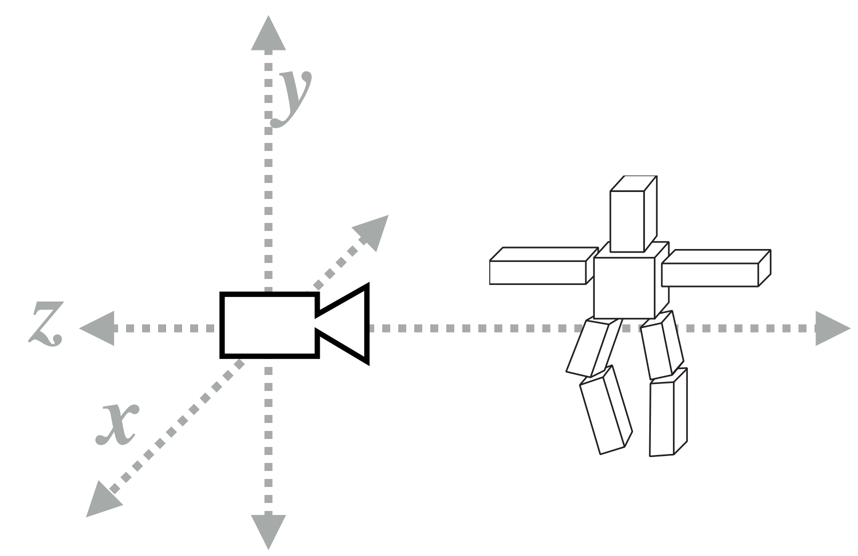
- Transformations can be interpreted as operations that move points in space
  - e.g., for modeling, animation
- Or as a change of coordinate system
  - e.g., screen and view transforms
- Construct complex transformations as compositions of basic transforms
- Homogeneous coordinate representation allows for expression of non-linear transforms (e.g., affine, perspective projection) as matrix operations (linear transforms) in higher-dimensional space
  - Matrix representation affords simple implementation and efficient composition



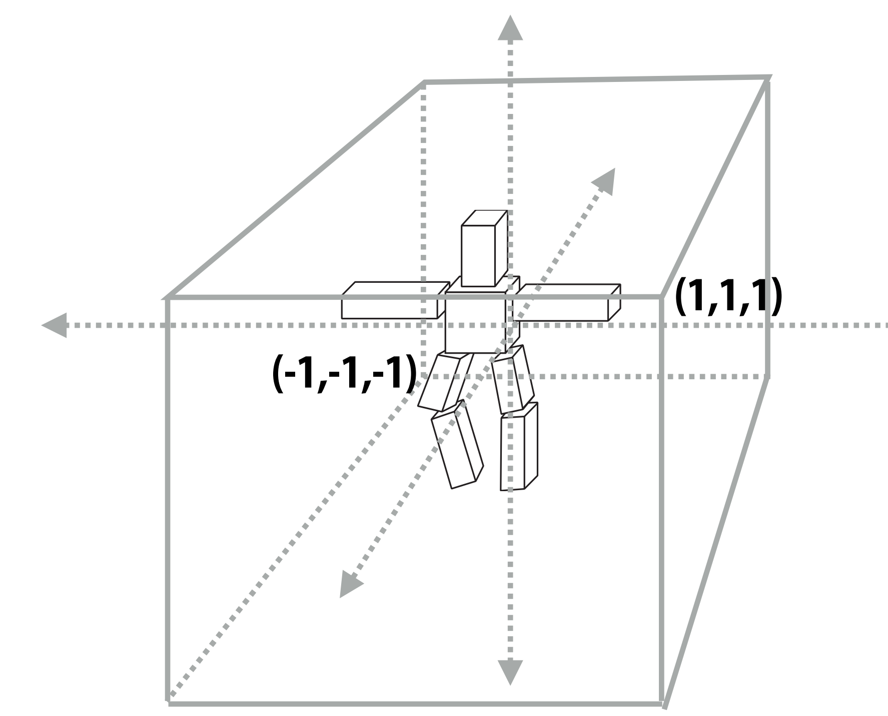
Transformations recap



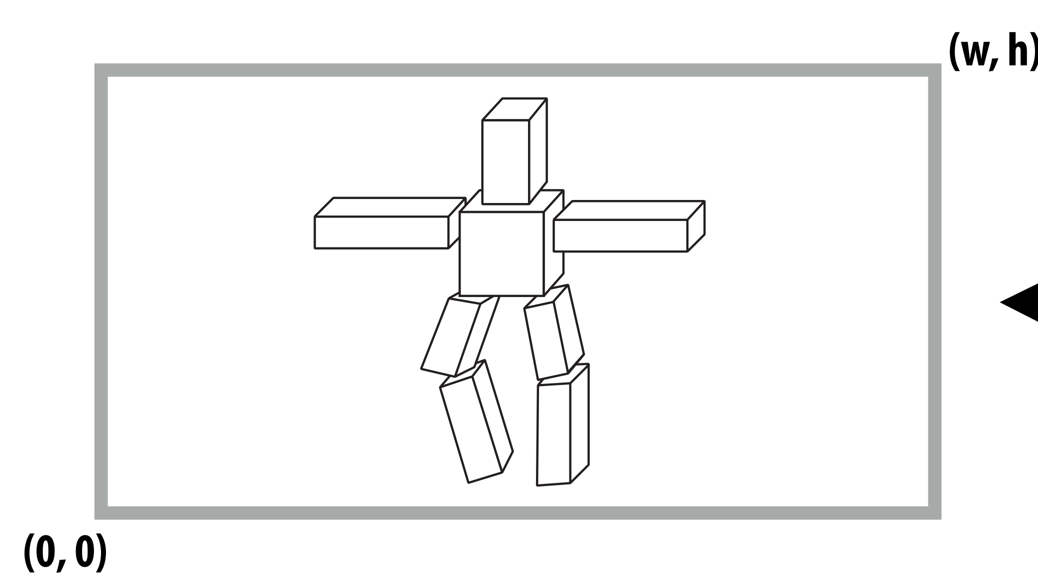
Modeling transforms:
Position object in scene



Viewing (camera) transform:
positions objects in coordinate space relative to camera
Canonical form: camera at origin looking down -z

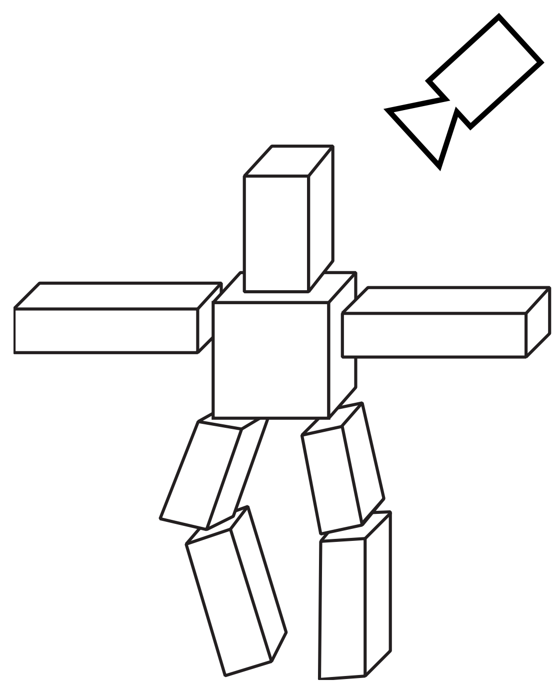


Projection transform + homogeneous divide:
Performs perspective projection
Canonical form: visible region of scene contained within unit cube

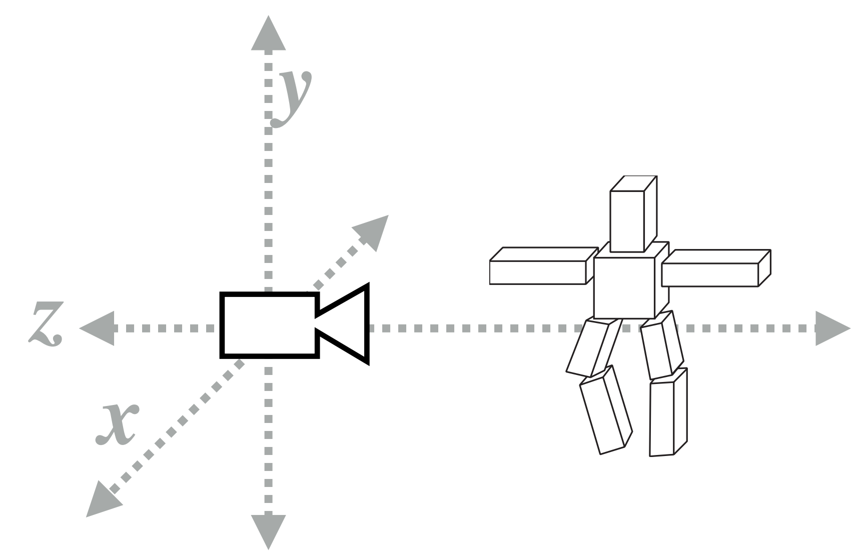


Screen transform:
objects now in 2D screen coordinates

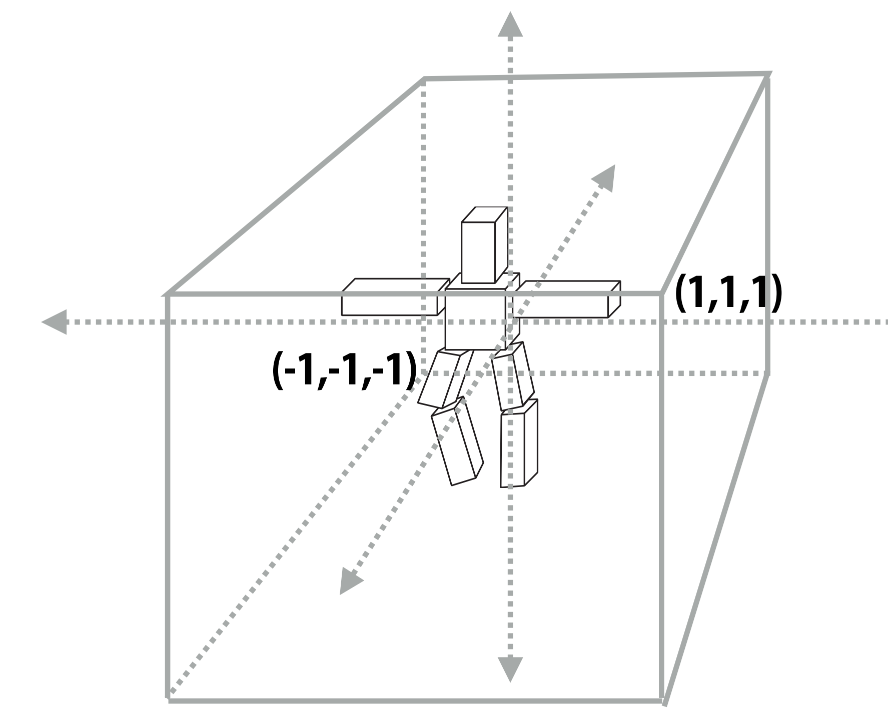
Transformations recap



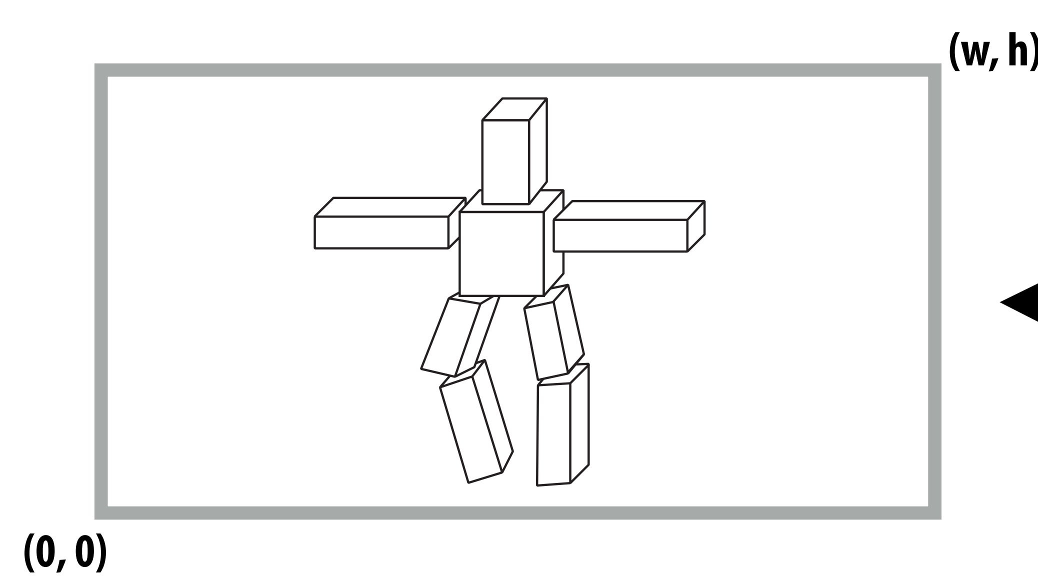
Modeling transforms:
Position object in scene



Viewing (camera) transform:
positions objects in coordinate space relative to camera
Canonical form: camera at origin looking down -z



Projection transform + homogeneous divide:
Performs perspective projection
Canonical form: visible region of scene contained within unit cube



Screen transform:
objects now in 2D screen coordinates

Compute screen coverage from 2D object position

Further Reading

- Basic transforms and quaternions are nicely covered here (Real Time Rendering -- Chapter 4. by T. Akenine Moller, E. Haines, N. Hoffman)

- For a clean description of Euler Angles, the gimbal lock problem, and quaternions and how to use them, check out this textbook chapter (excerpt from Ch. 15 of Advanced Animation and Rendering Techniques. by A. Watt, M. Watt)

What you should know:

- Form an orthonormal basis
- Create a rotation matrix to rotate any coordinate frame to xyz
- Create the rotation matrix to rotate the xyz coordinate frame to any other frame
- Rotate about axis w by amount theta
- Know basic facts about rotation matrices / how to recognize a rotation matrix
    - Rows (also columns) are unit vectors
    - Rows (also columns) are orthogonal to one another
    - If our rows (or columns) are u, v, and w, then $u \times v = w$
    - The inverse of a rotation matrix is its transpose
- Create a projection matrix that projects all points onto an image plane at $z=1$
- Propose a projection matrix that maintains some depth information
- Understand the motivation behind the projection matrix that projects the view frustum to a unit cube
- Be able to draw / discuss the details of the view frustum