

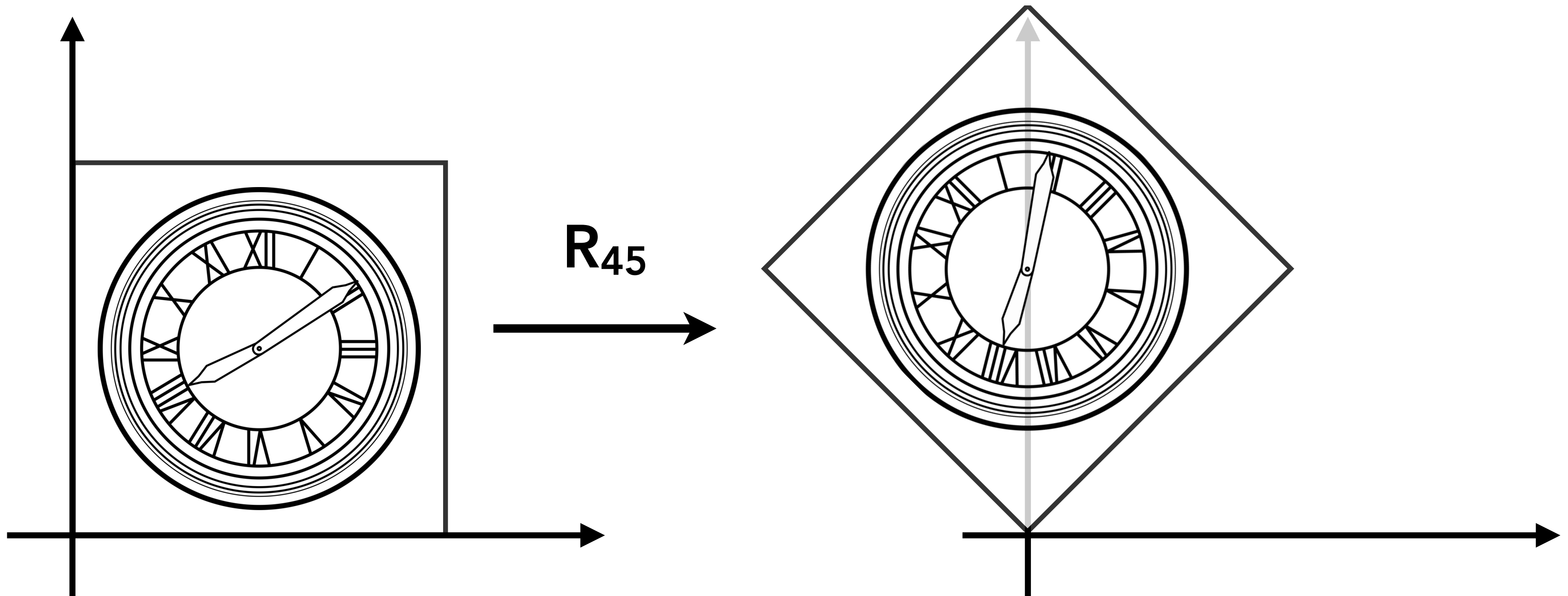
Lecture 4:

Transforms

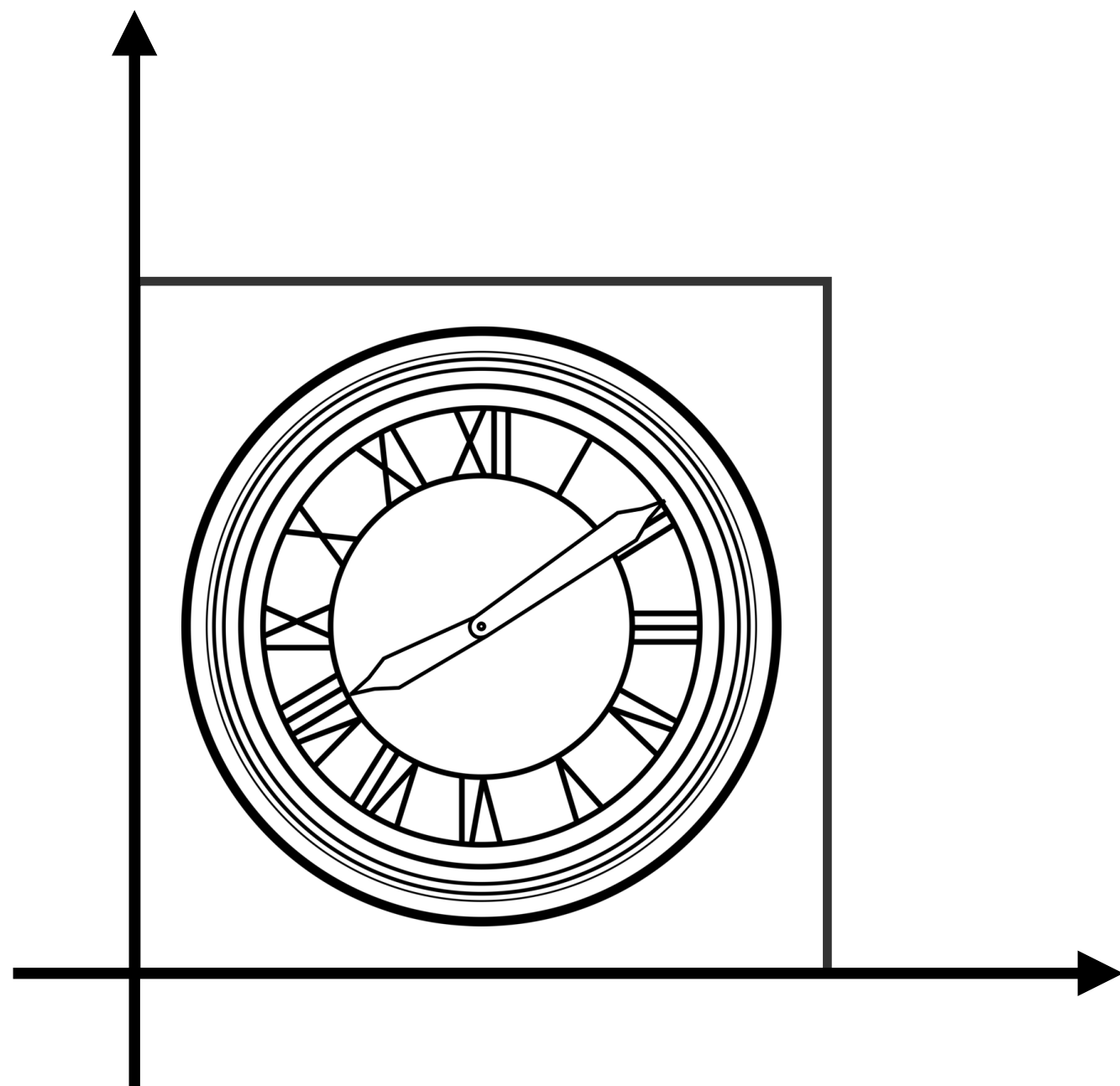
Computer Graphics and Imaging
UC Berkeley CS184/284A

Basic Transforms

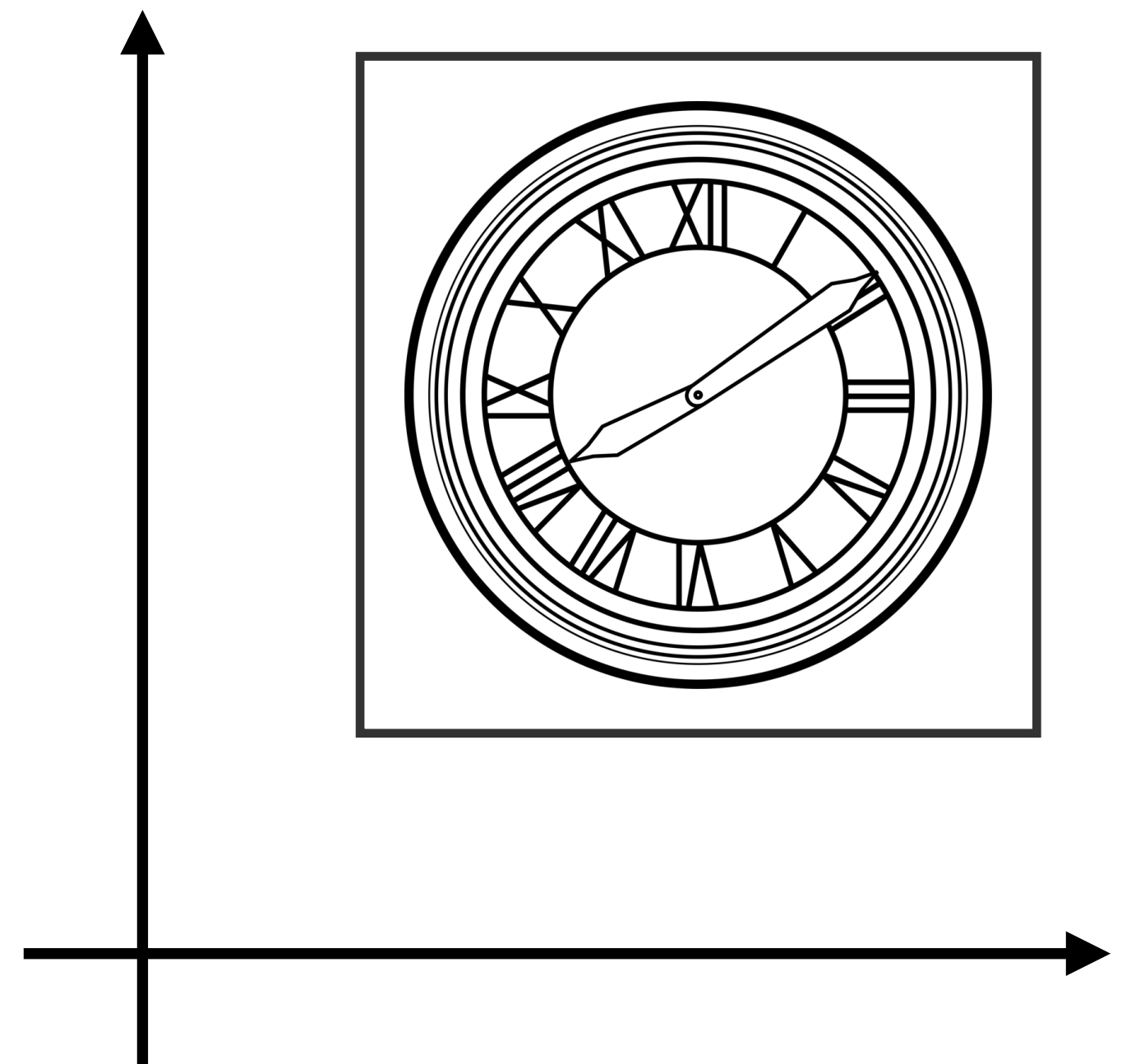
Rotate



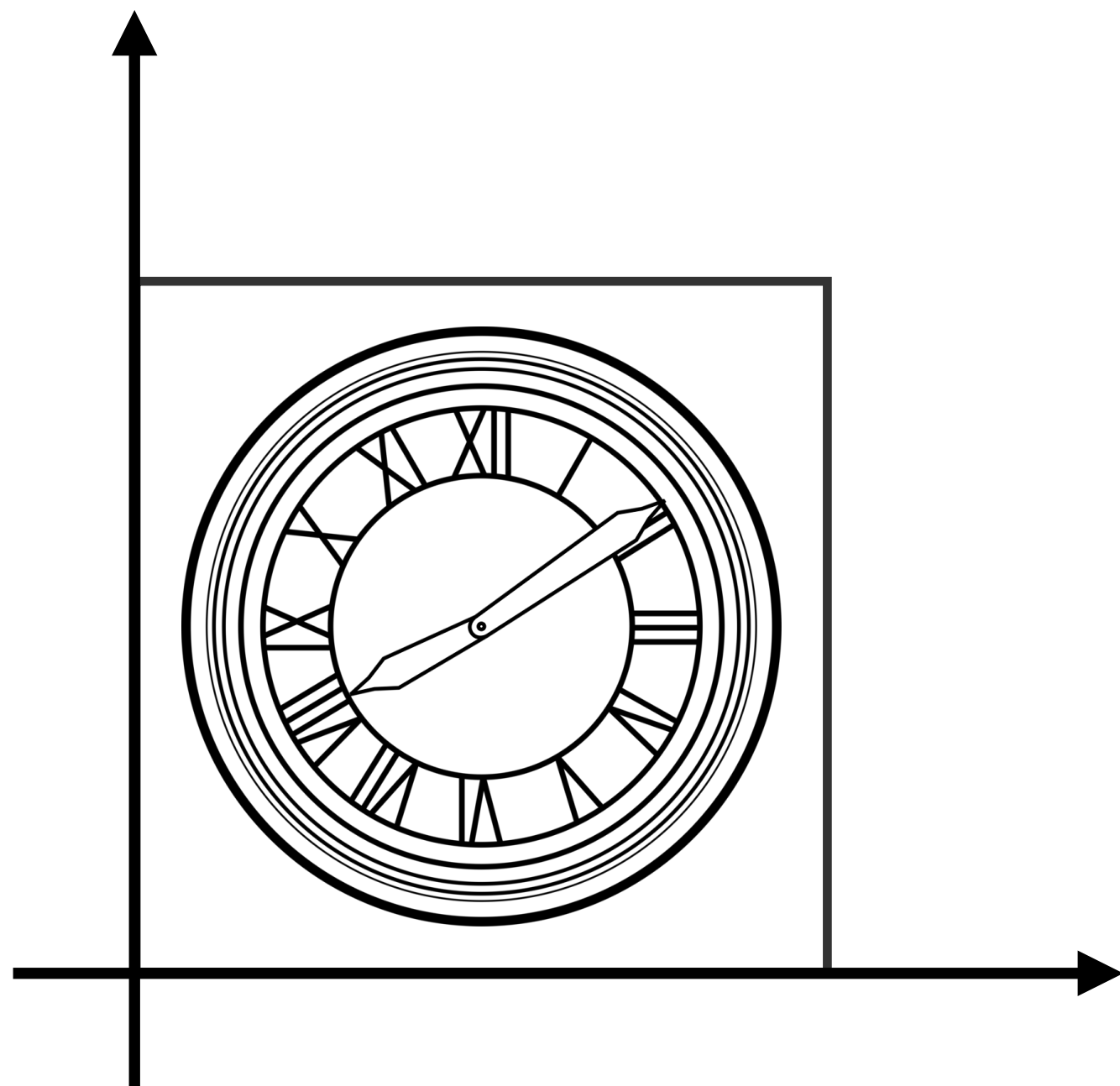
Translate



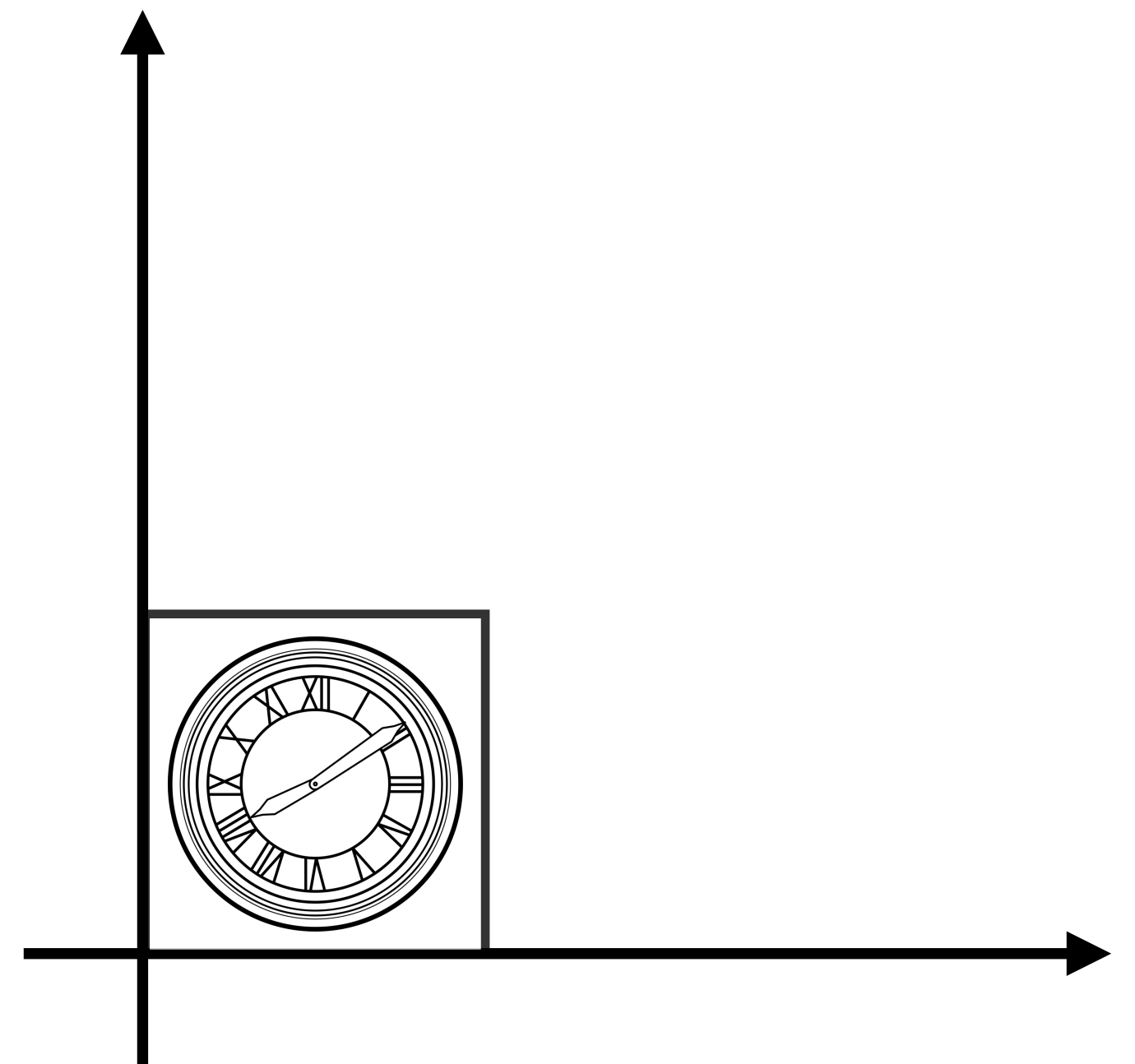
$T_{1,1}$



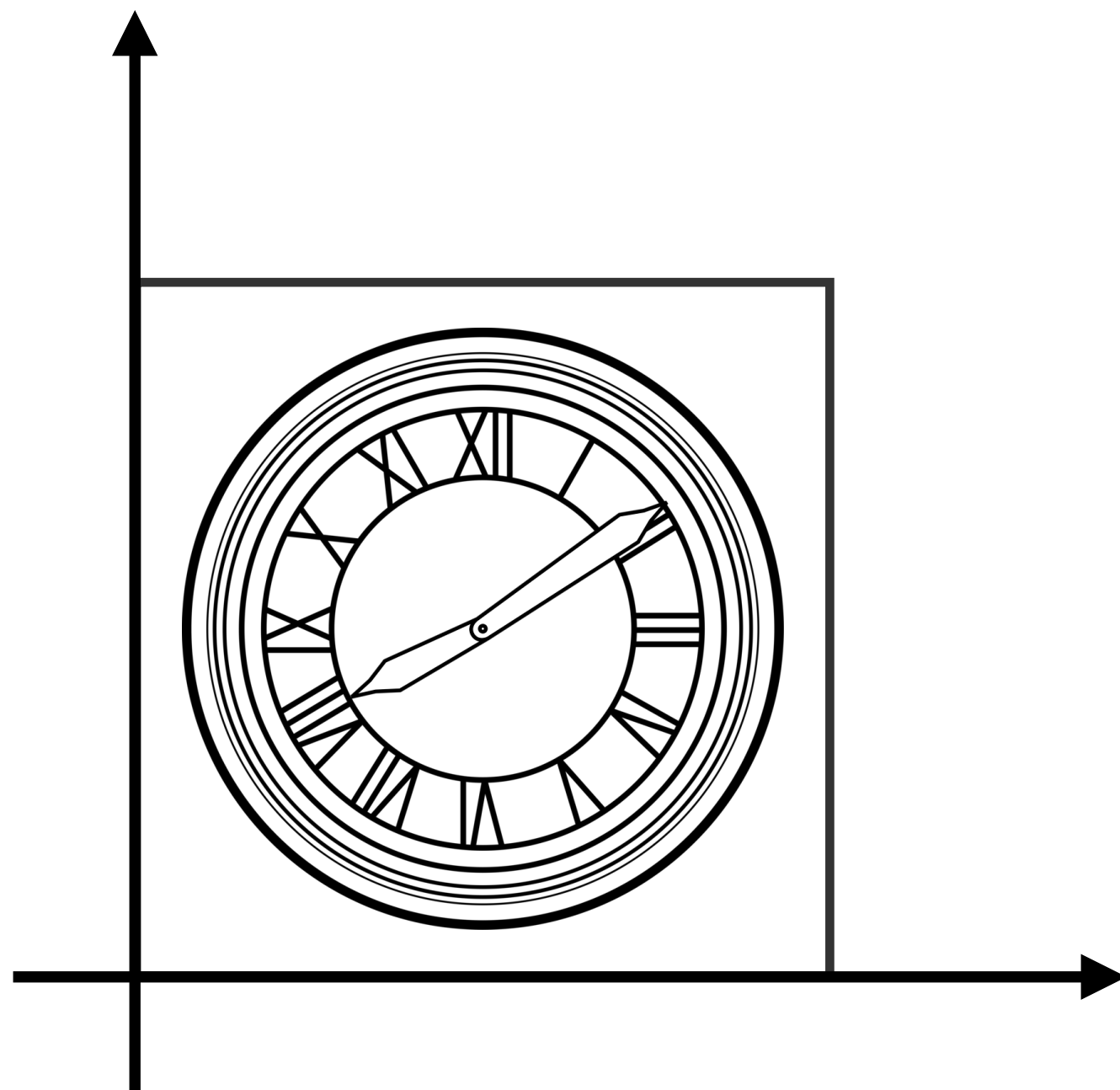
Scale



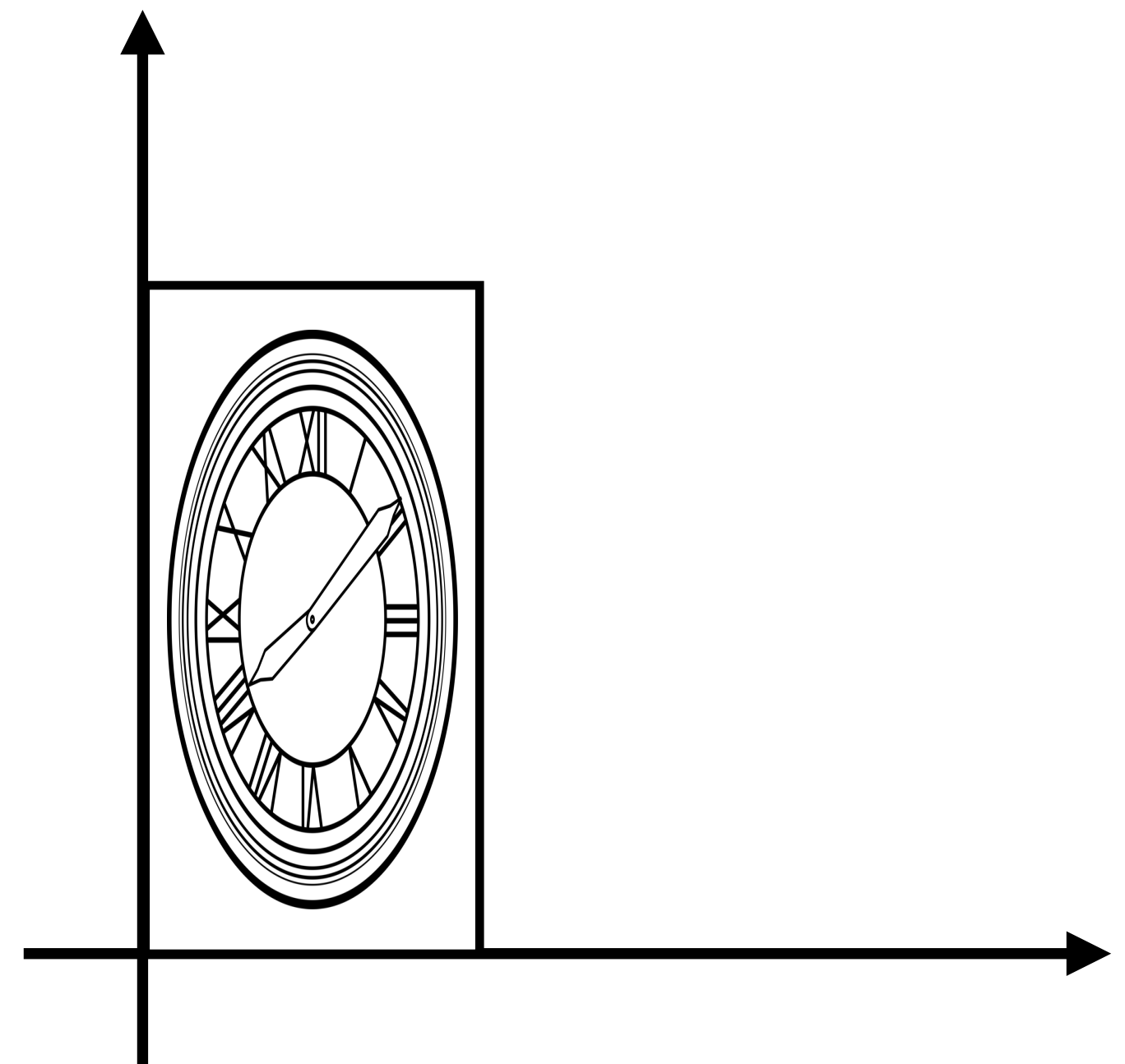
$S_{0.5}$



Scale (Non-Uniform)



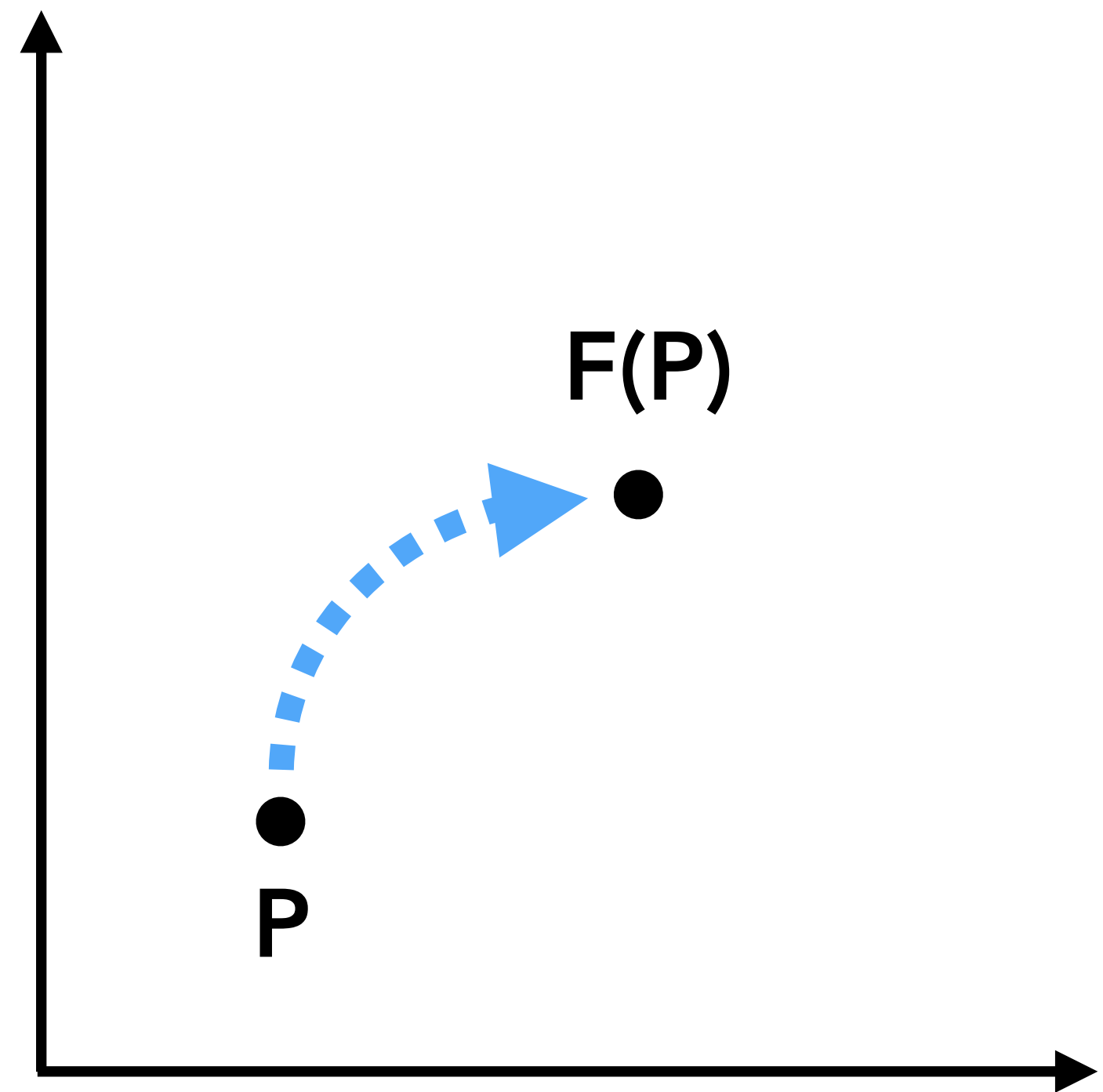
$S_{0.5,1.0}$



What Are Transforms?

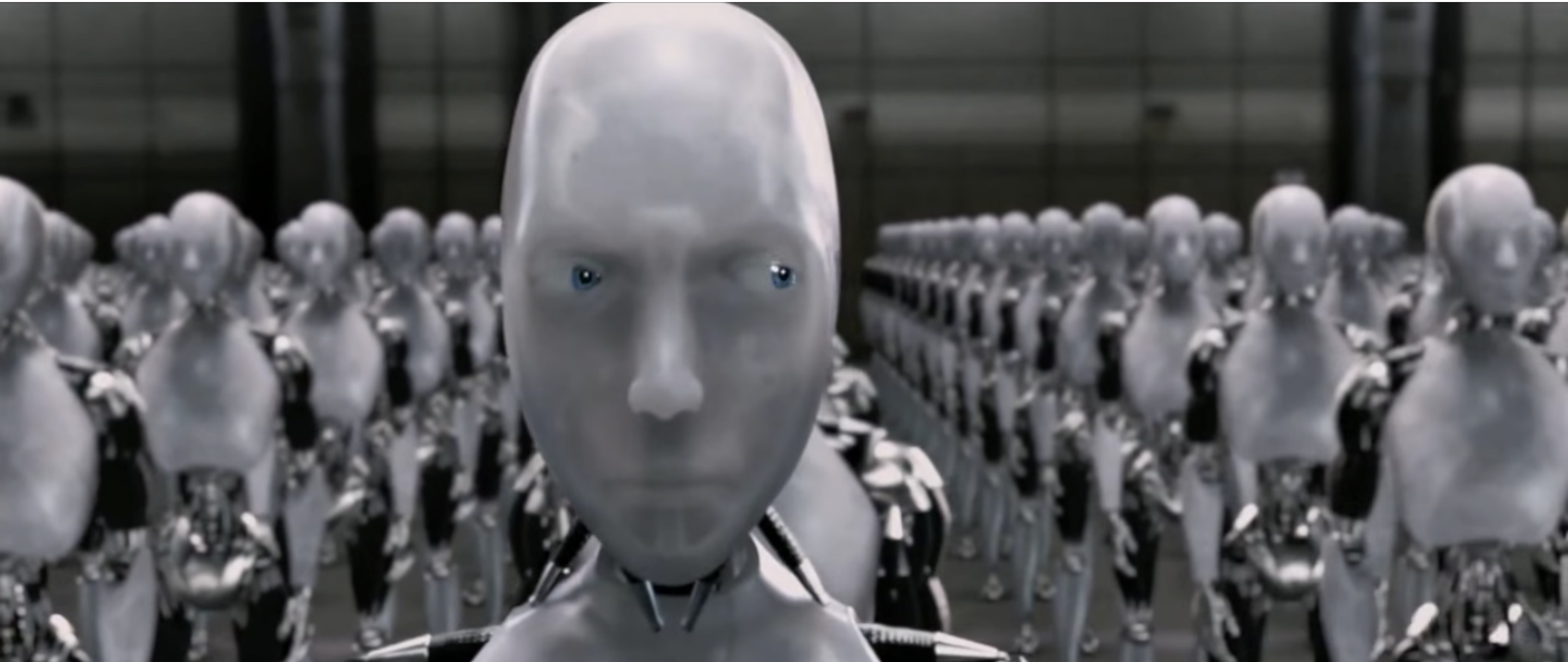
Just functions acting on points

- $(x', y', z') = F(x, y, z)$
- $P' = F(P)$



Why Study Transforms?

Modeling A Robot Army



iRobot movie

Transforms can describe position of object instances

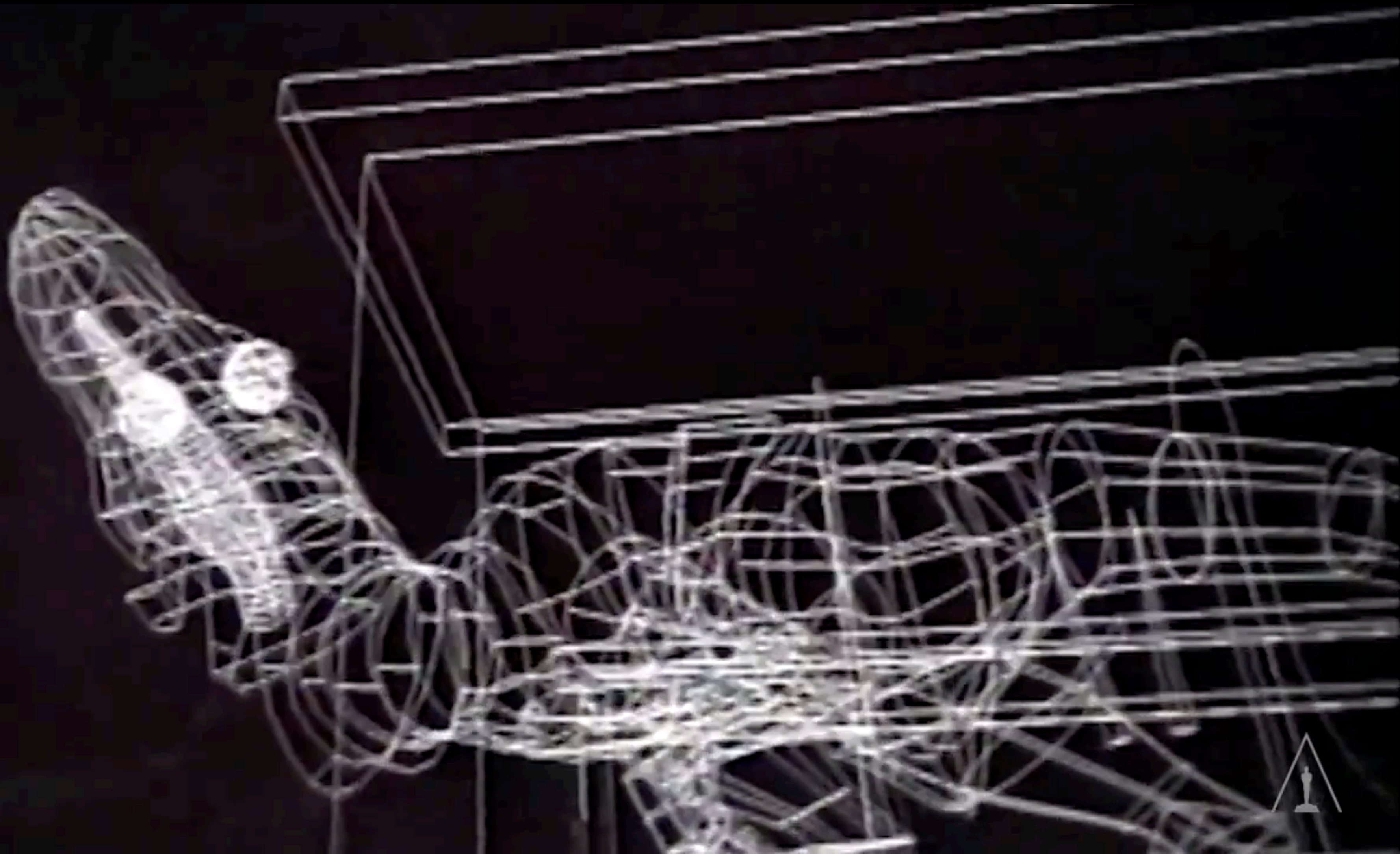
Posing a Character's Skeleton



iRobot movie

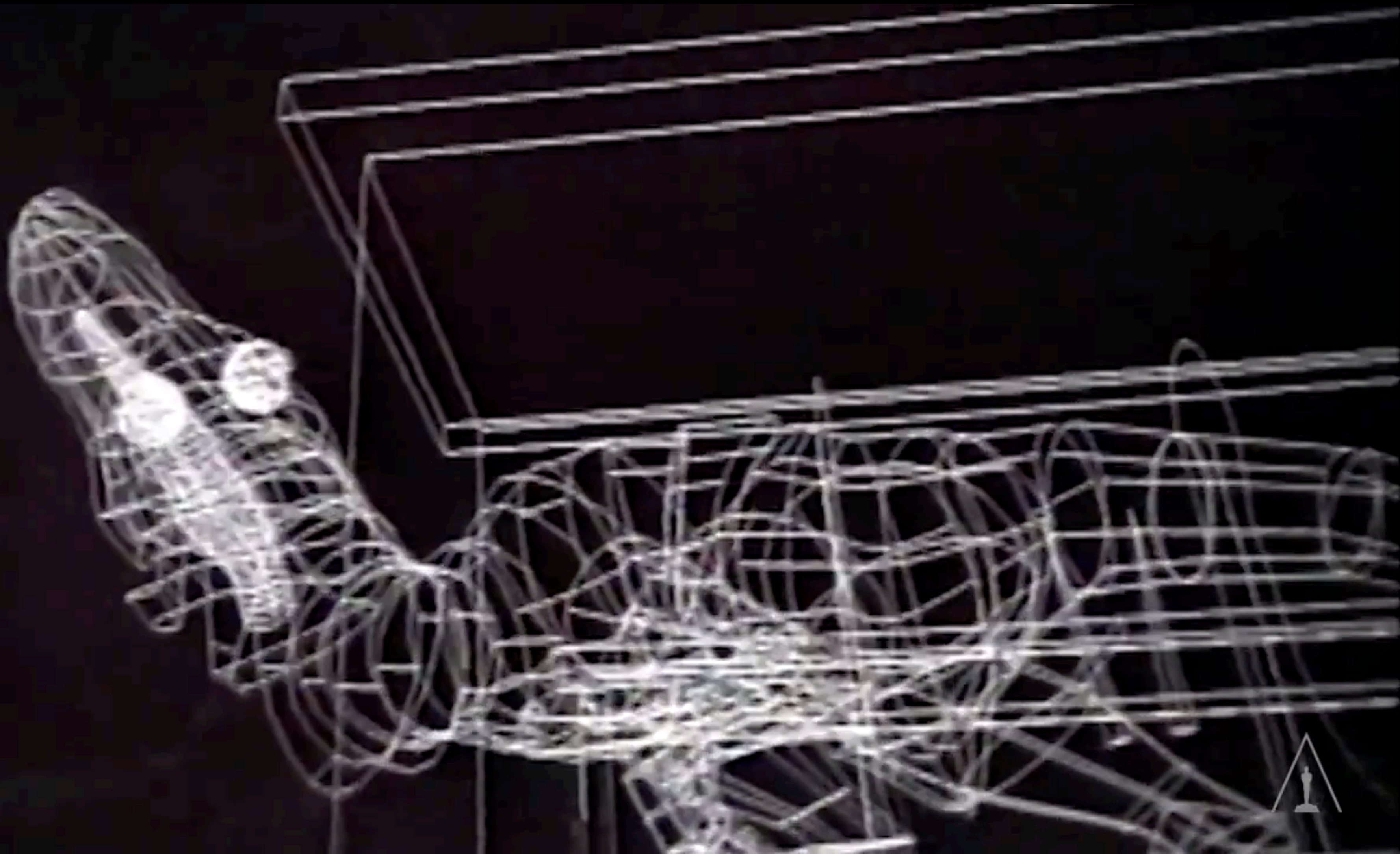
Transforms can describe relative position of connected body parts

Project Polygons in 3D to 2D Screen



Moments That Changed The Movies: Jurassic Park
<https://www.youtube.com/watch?v=KWsbcbvYqN8>

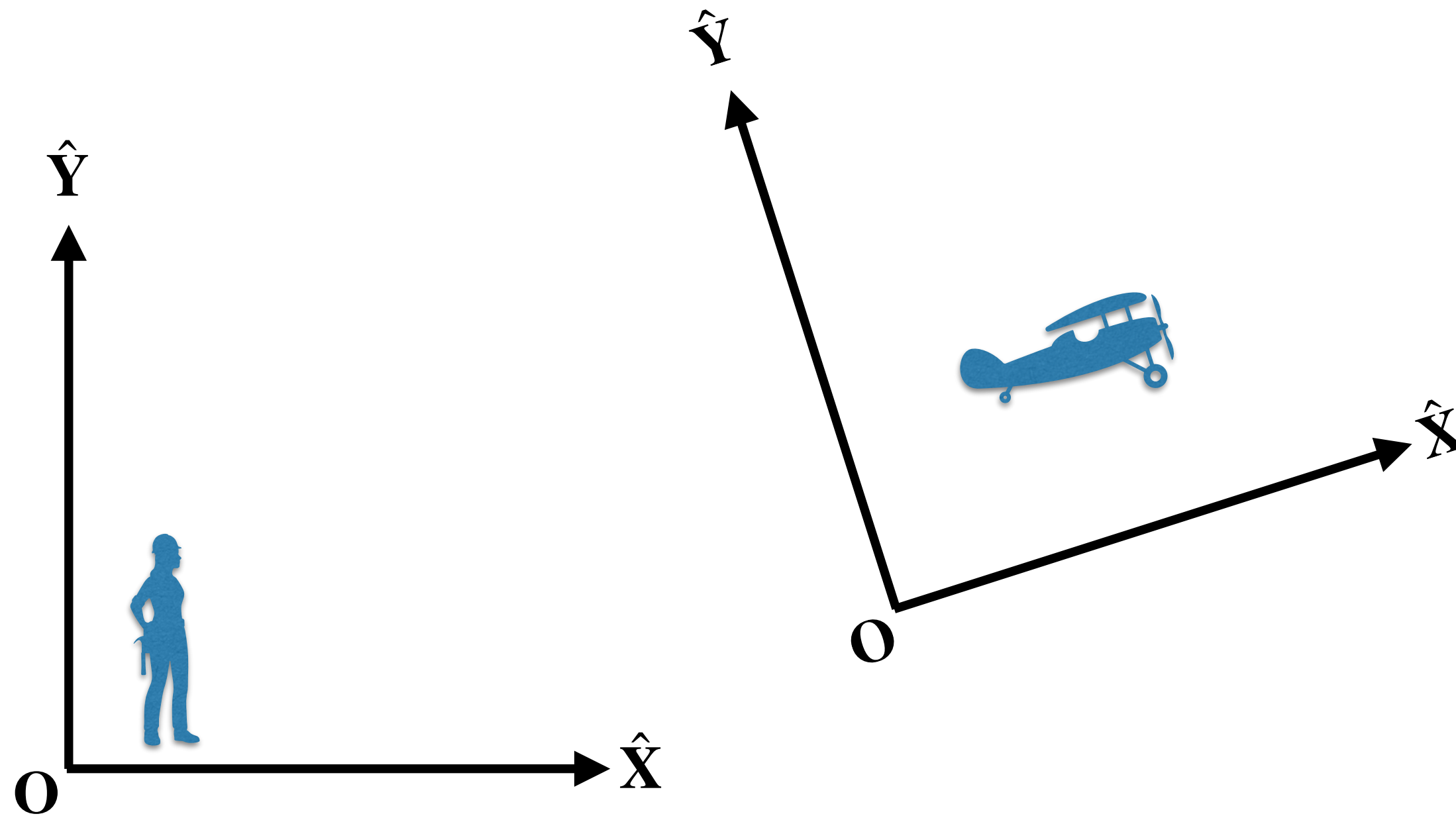
Project Polygons in 3D to 2D Screen



Moments That Changed The Movies: Jurassic Park
<https://www.youtube.com/watch?v=KWsbcbvYqN8>

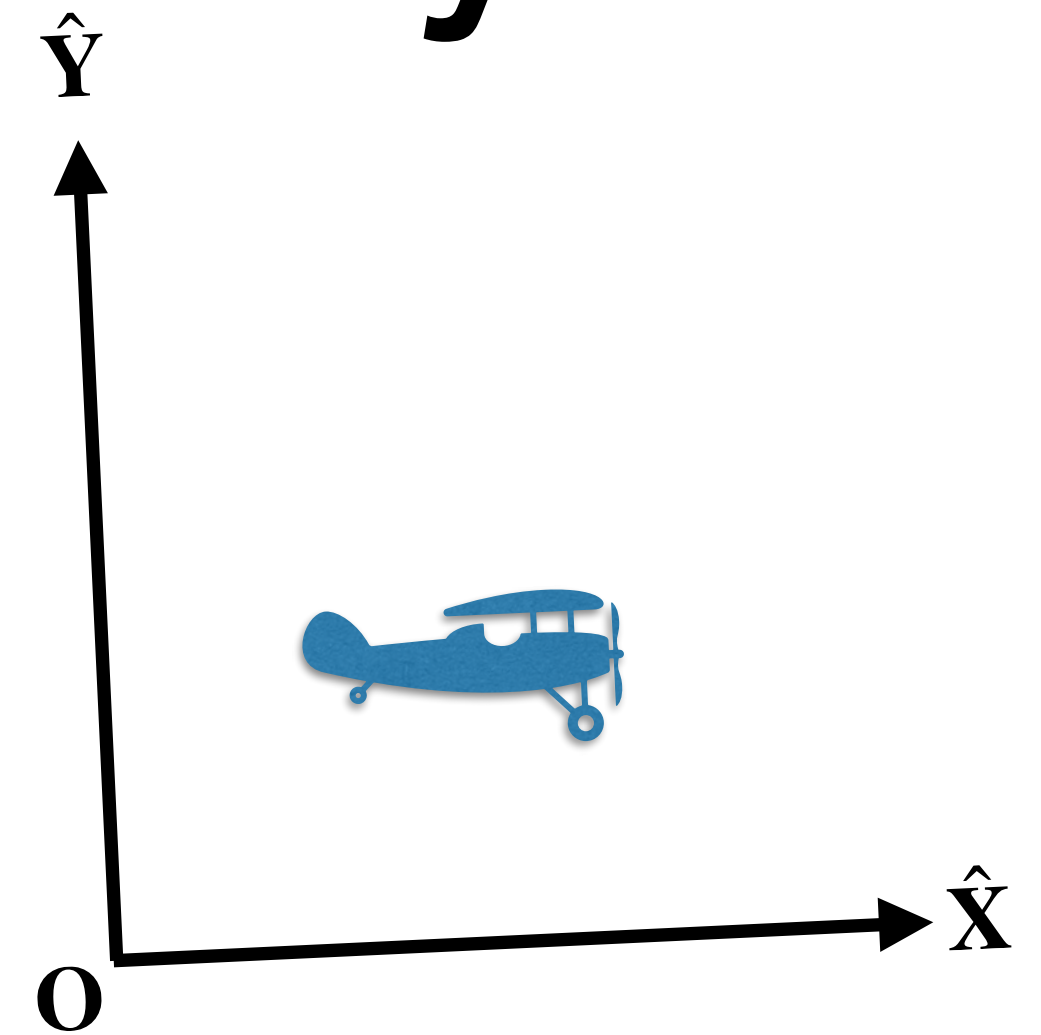
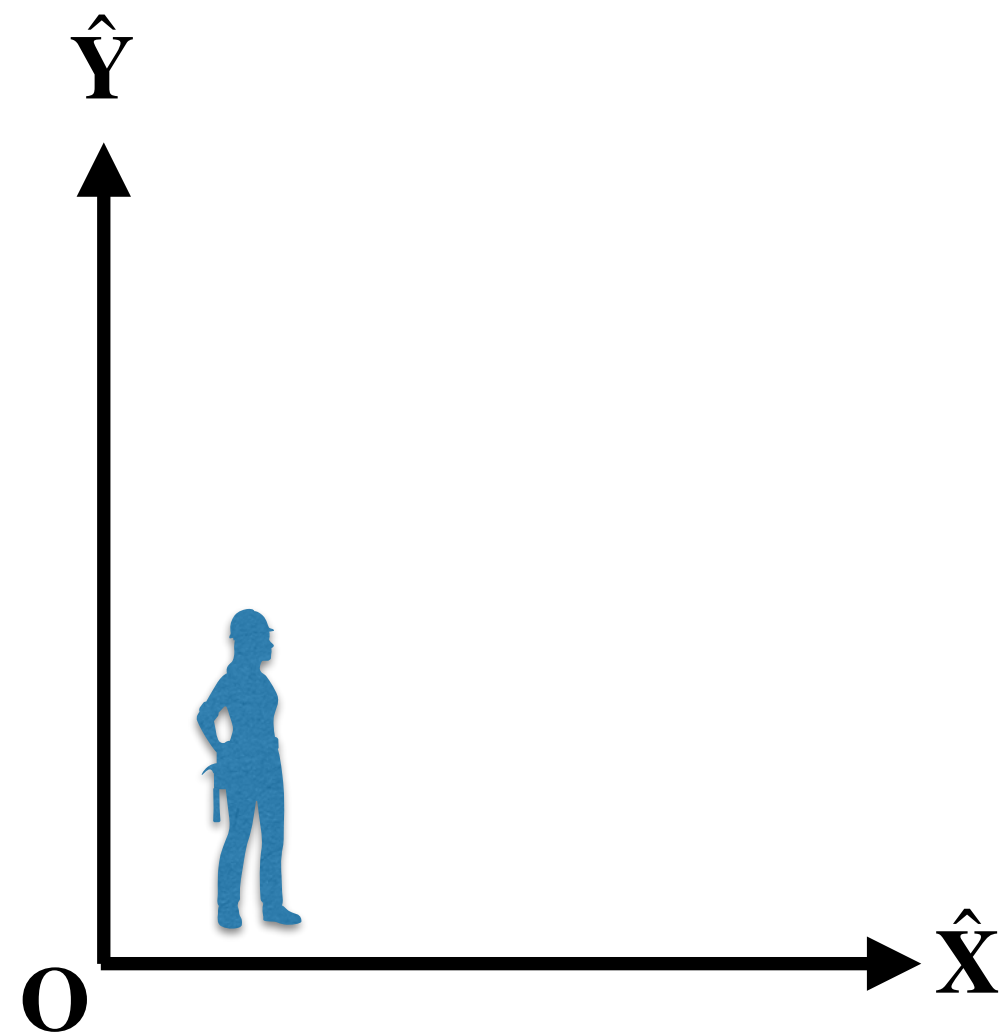
Coordinate Systems are Arbitrary

So feel free to pick one you like!



Coordinate Systems are Arbitrary

So feel free to pick one you like!



Why Study Transforms?

Modeling

- Define shapes in convenient coordinates
- Enable multiple copies of the same object
- Efficiently represent hierarchical scenes

Viewing

- World coordinates to camera coordinates
- Parallel / perspective projections from 3D to 2D

Other

- Switching between coordinate systems

Lecture Outline

How to think about and use transformations

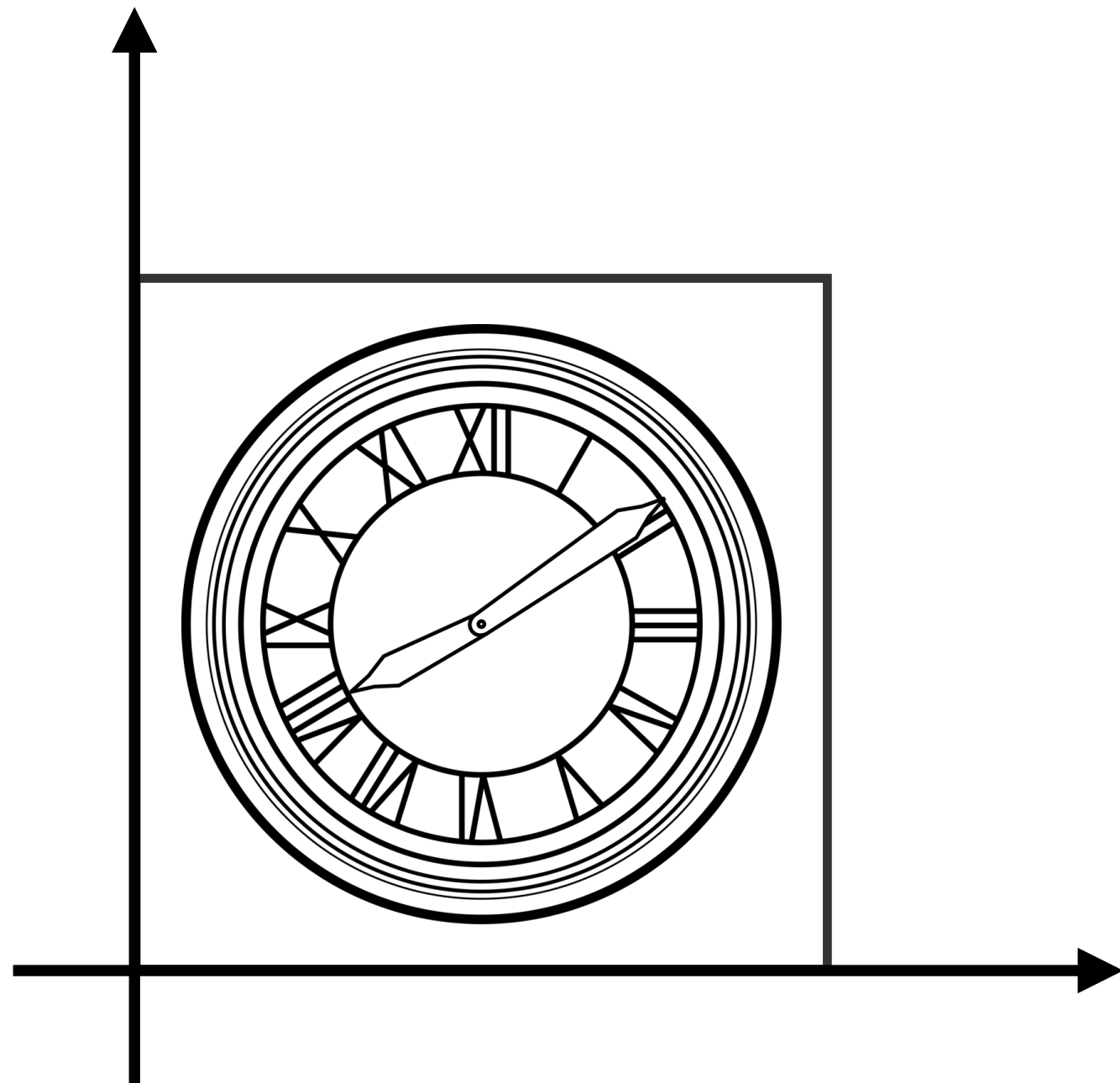
- Types: rotate, translate, scale, ...
- Coordinate frames
- Composing multiple transformations
- Hierarchical transforms
- Perspective projection

How to implement?

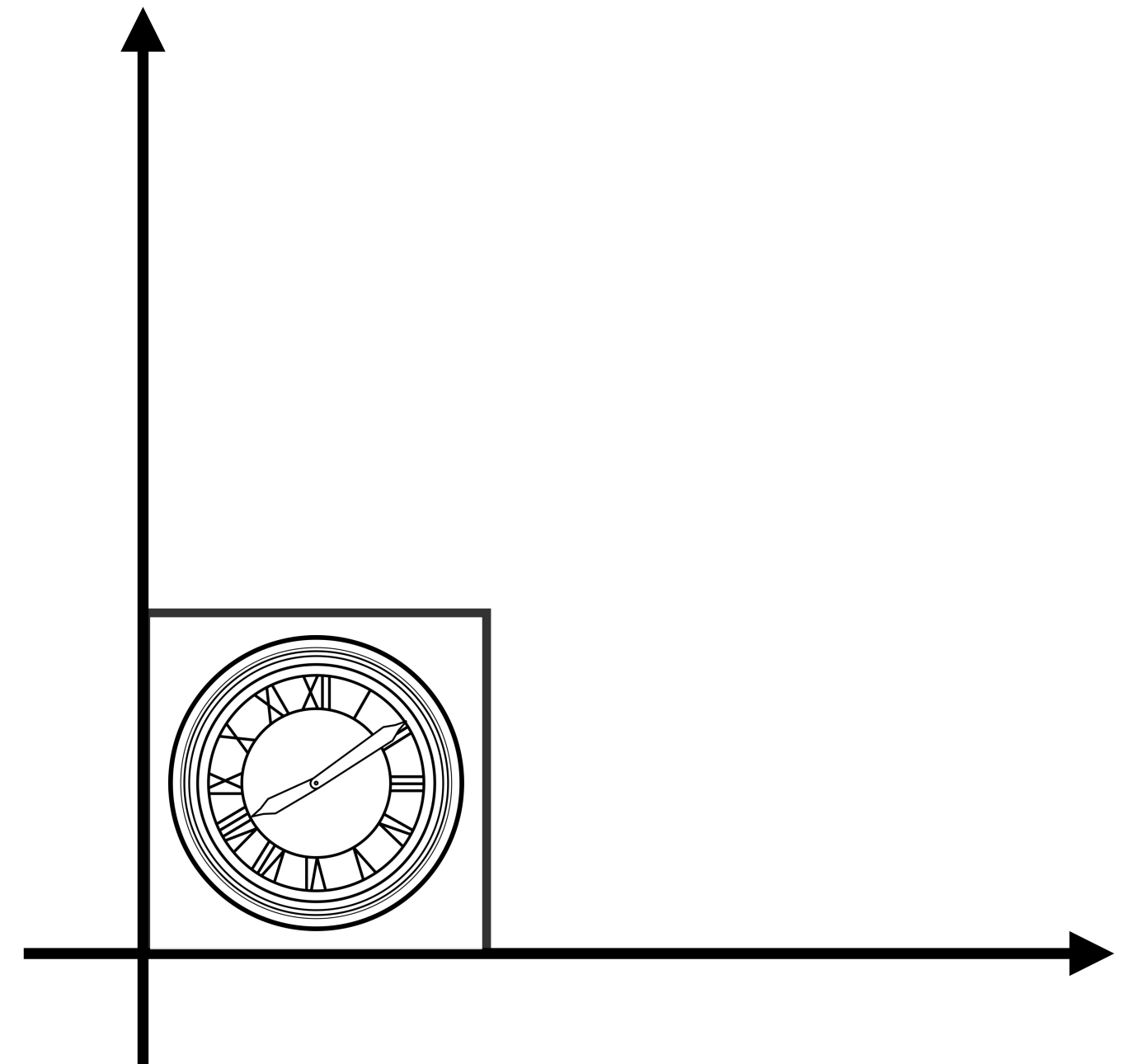
- Represent transforms as matrices
- Homogeneous coordinates

Linear Transforms = Matrices

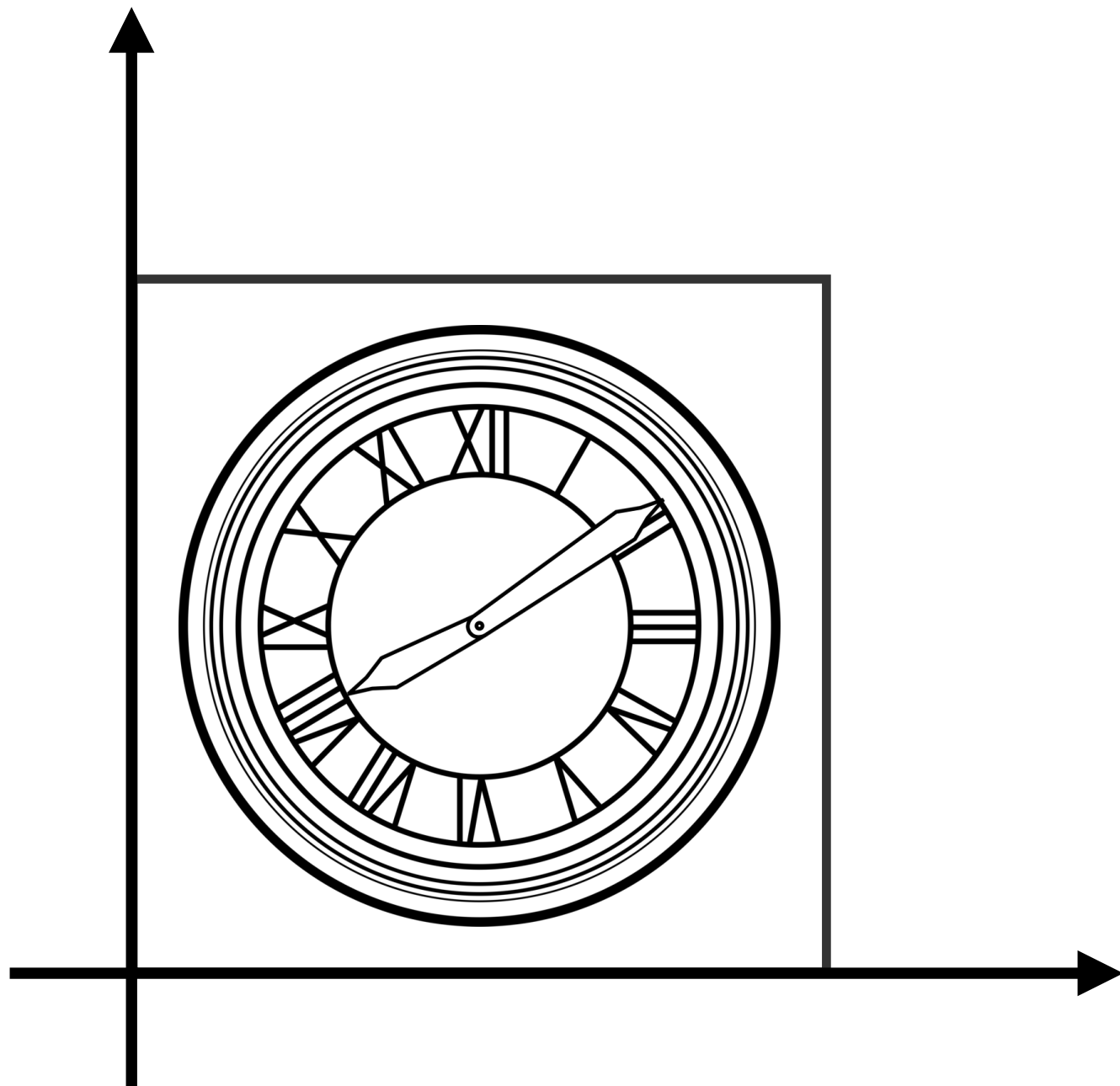
Scale Transform



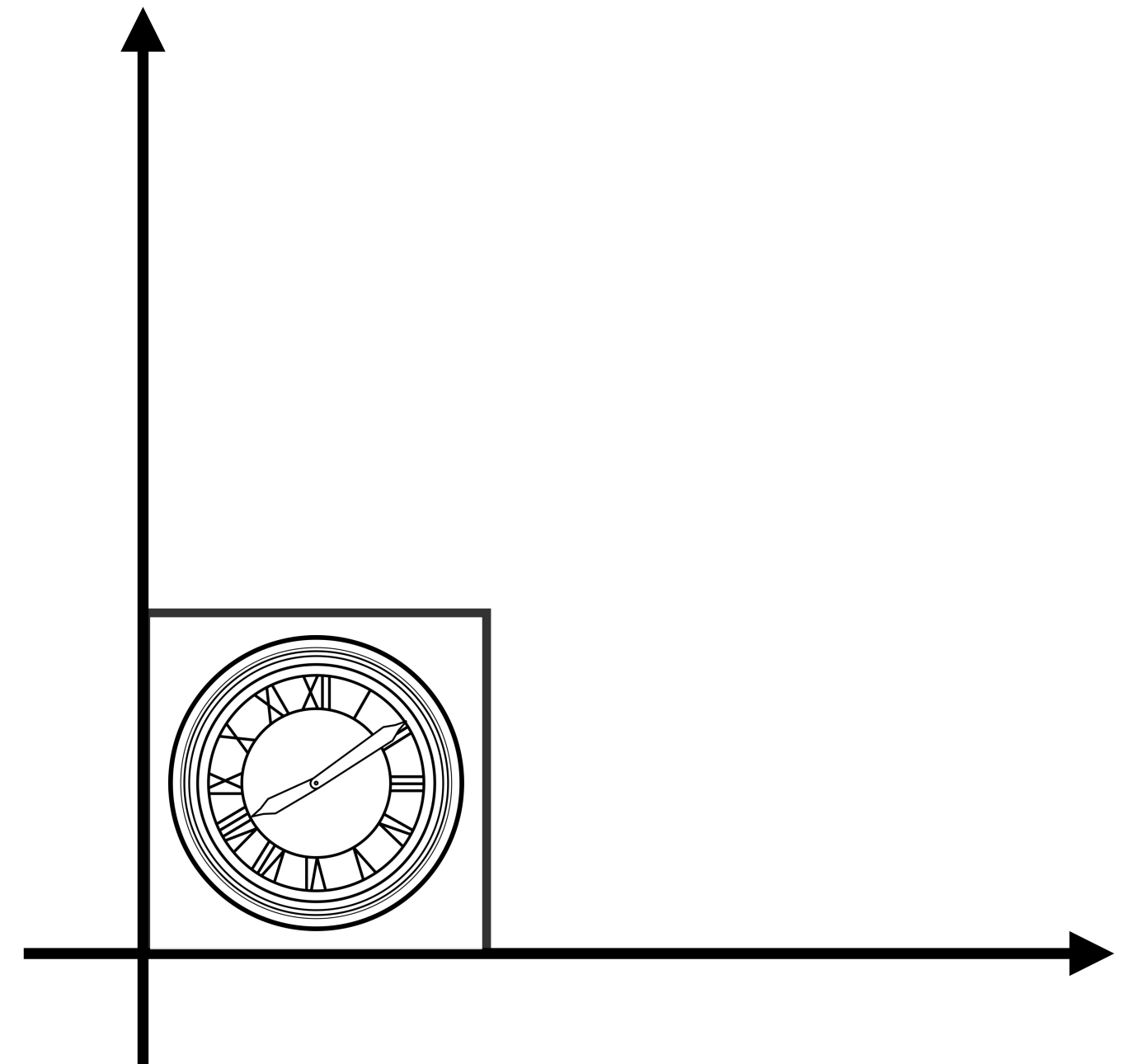
$S_{0.5}$



Scale Transform



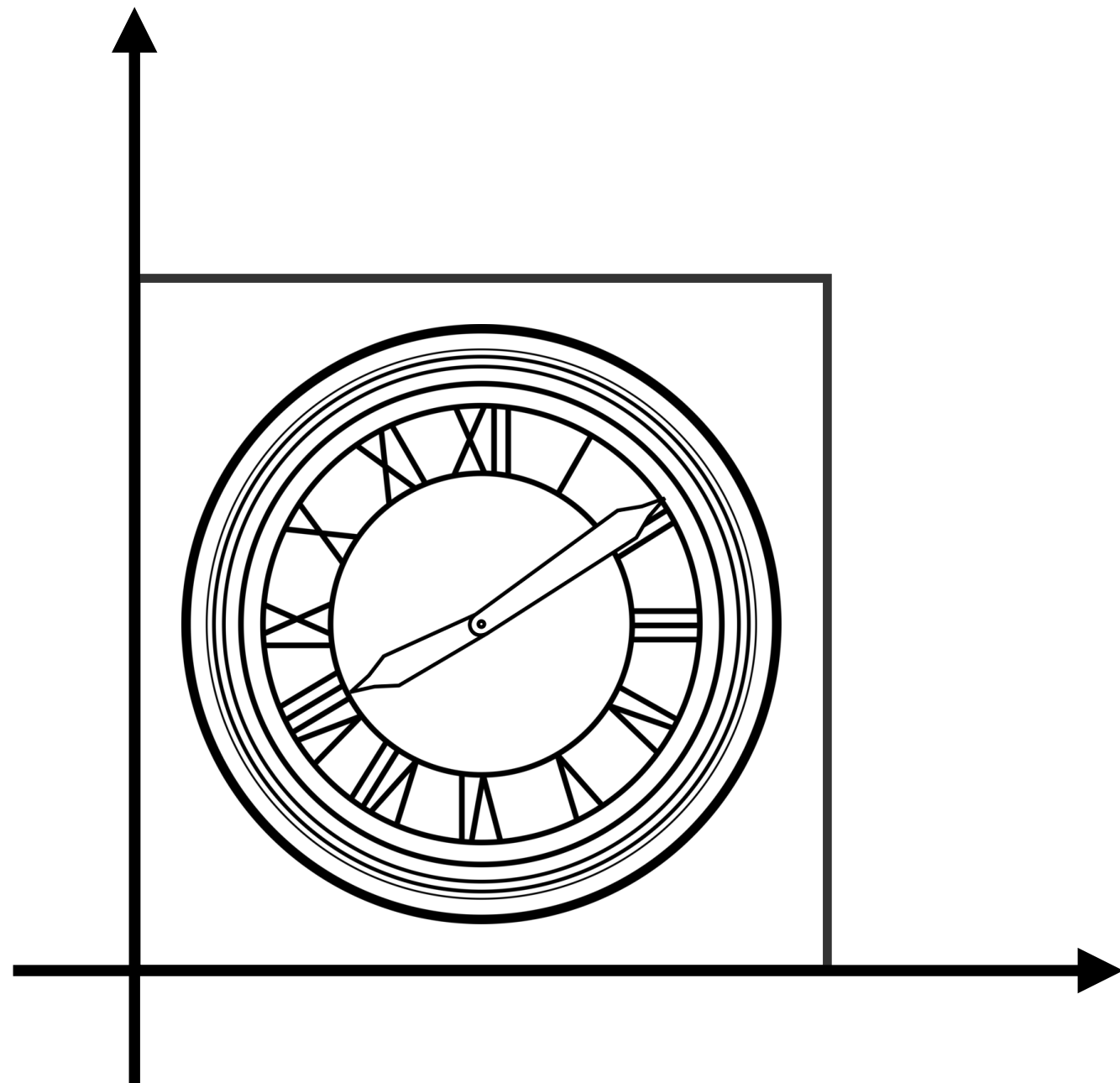
$S_{0.5}$



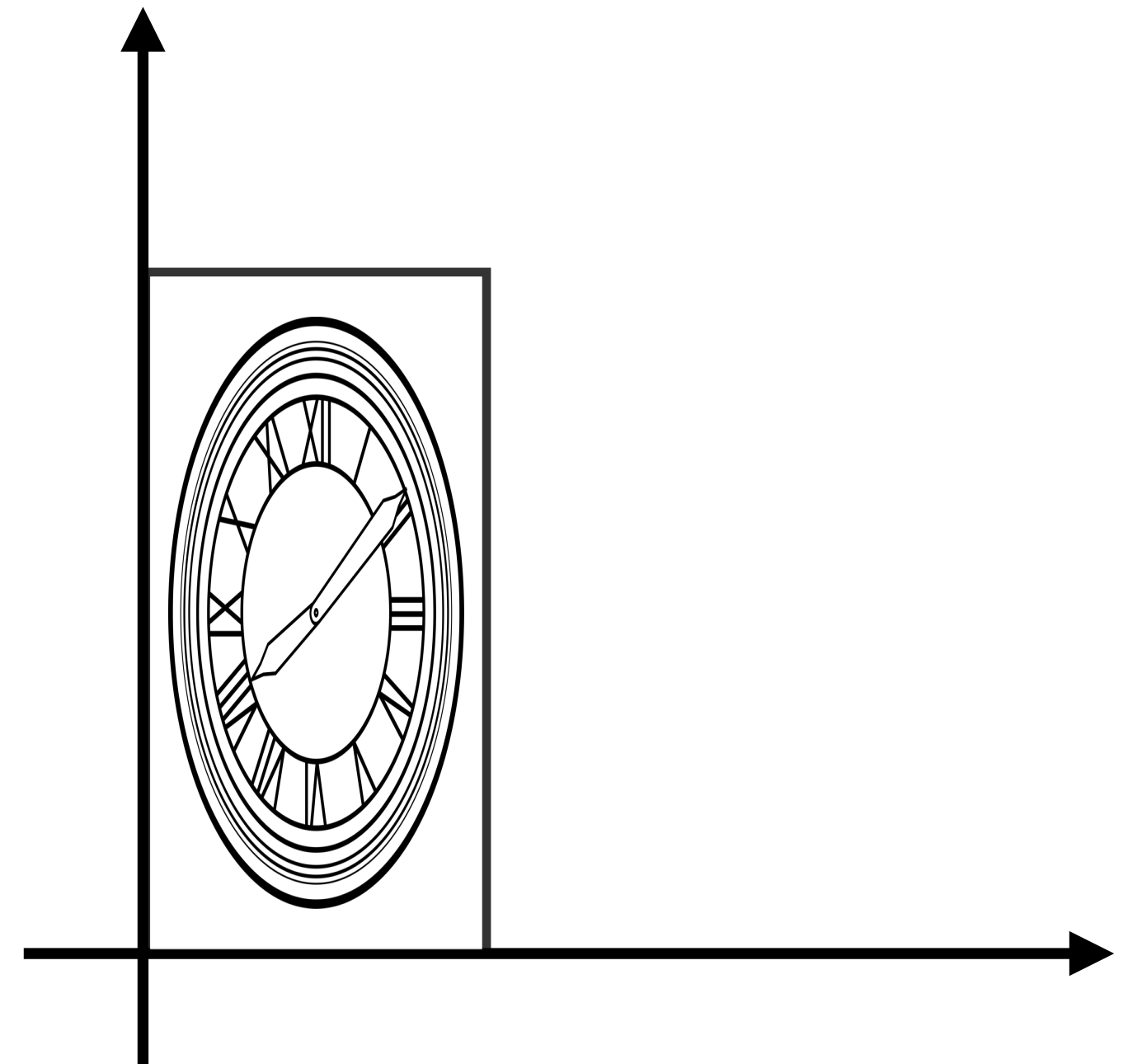
$$x' = 0.5x$$

$$y' = 0.5y$$

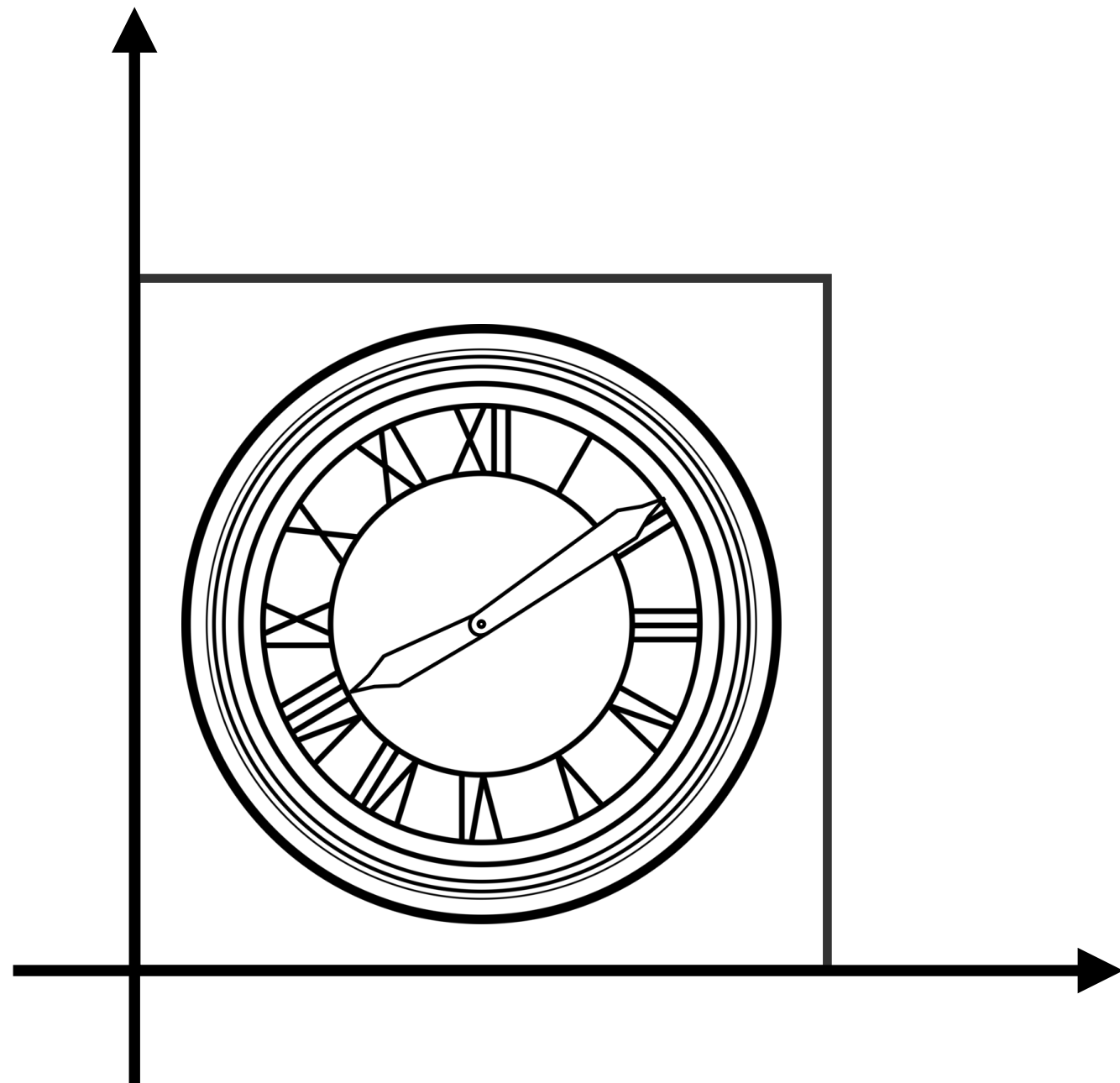
Scale Transform (non-uniform a.k.a anisotropic)



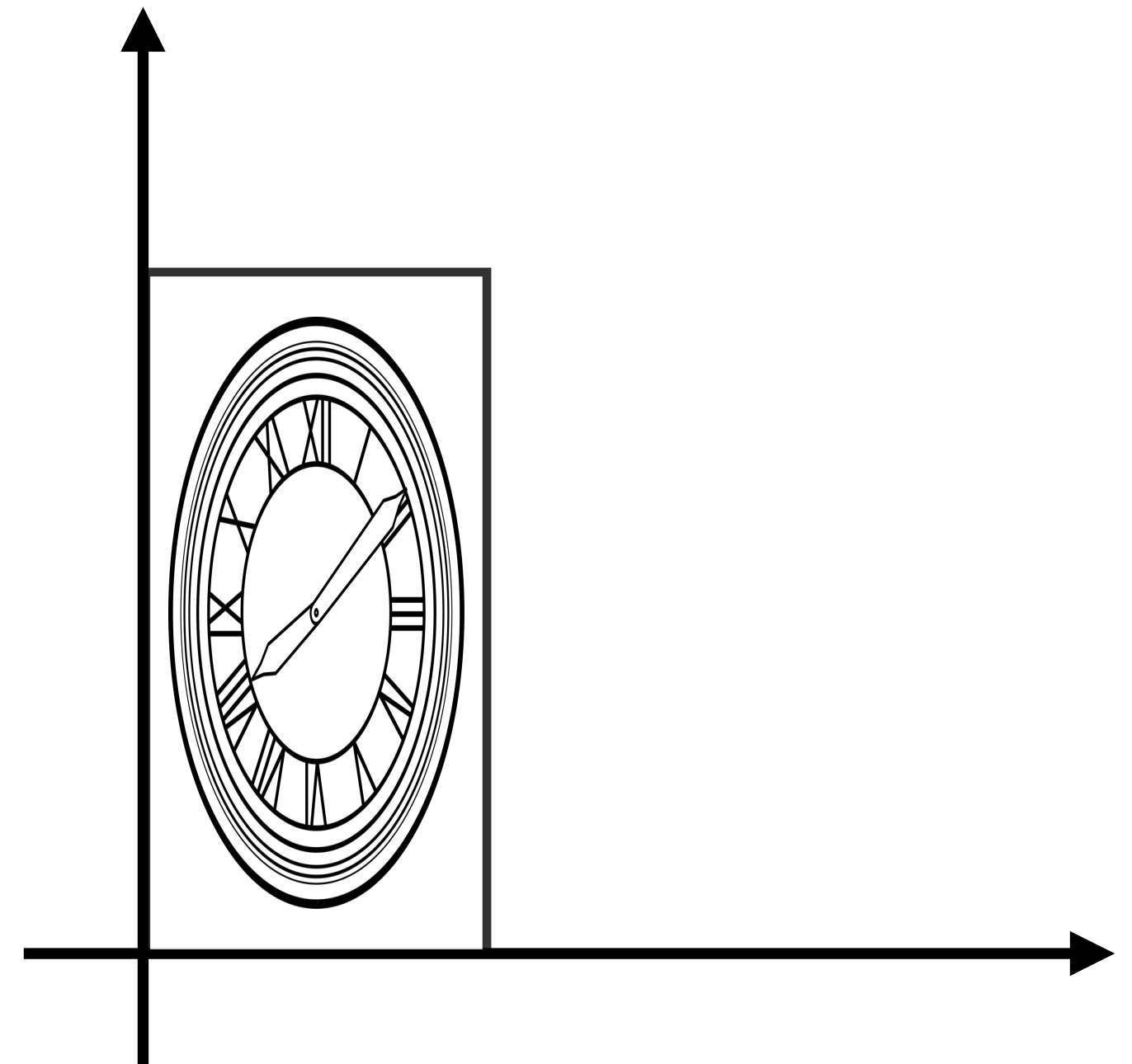
$S_{0.5, 1.0}$



Scale Transform (non-uniform a.k.a anisotropic)



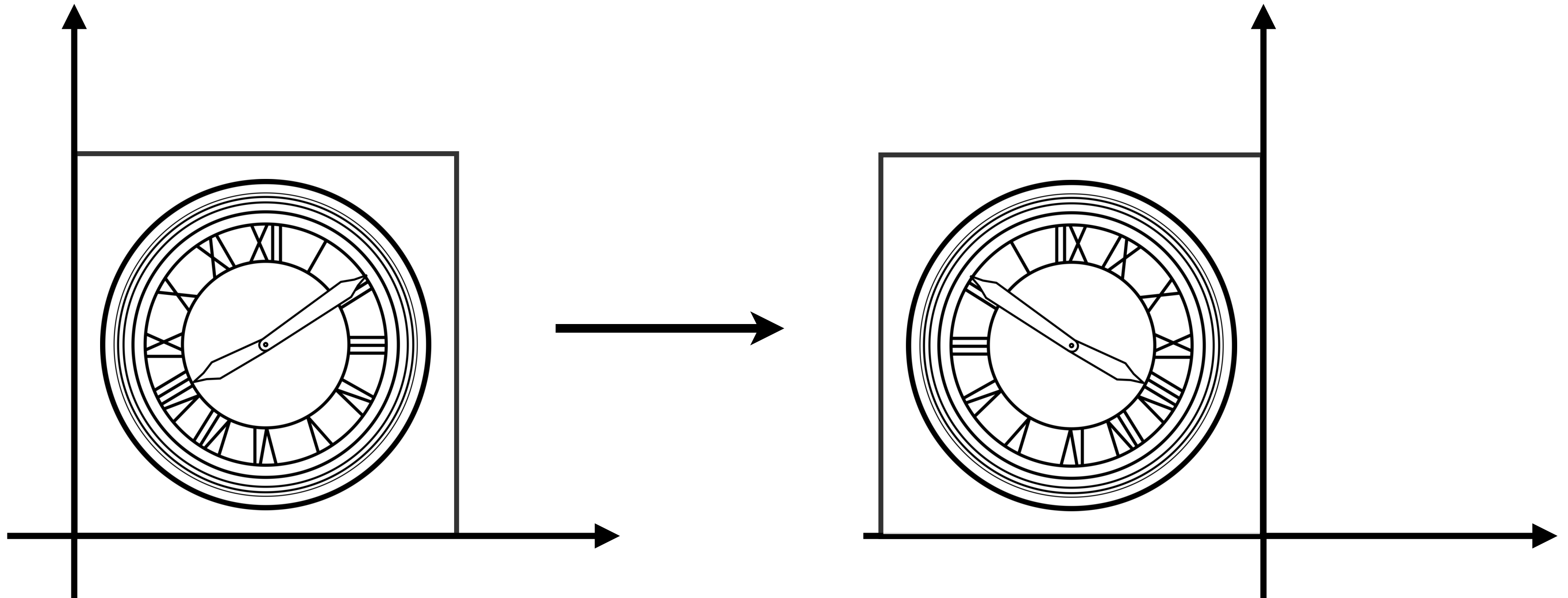
$$S_{0.5, 1.0}$$

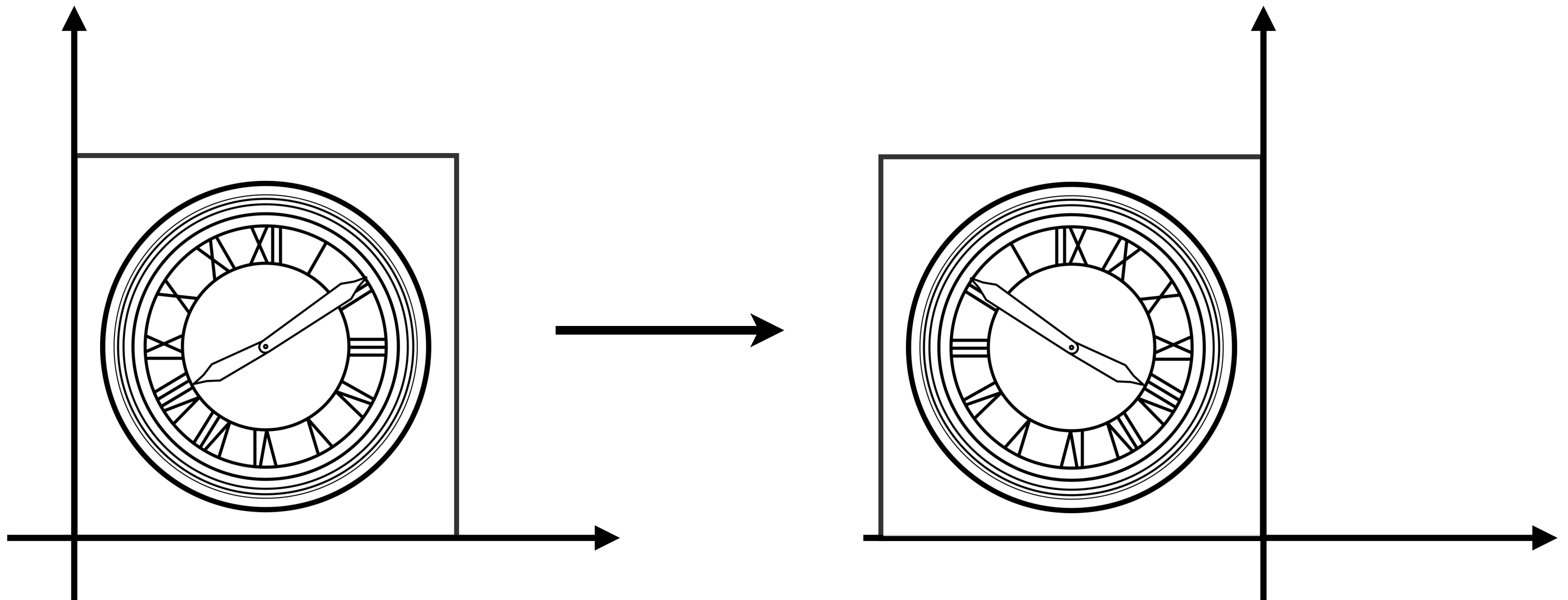
$$x' = 0.5x$$

$$y' = y$$

Reflection Transform



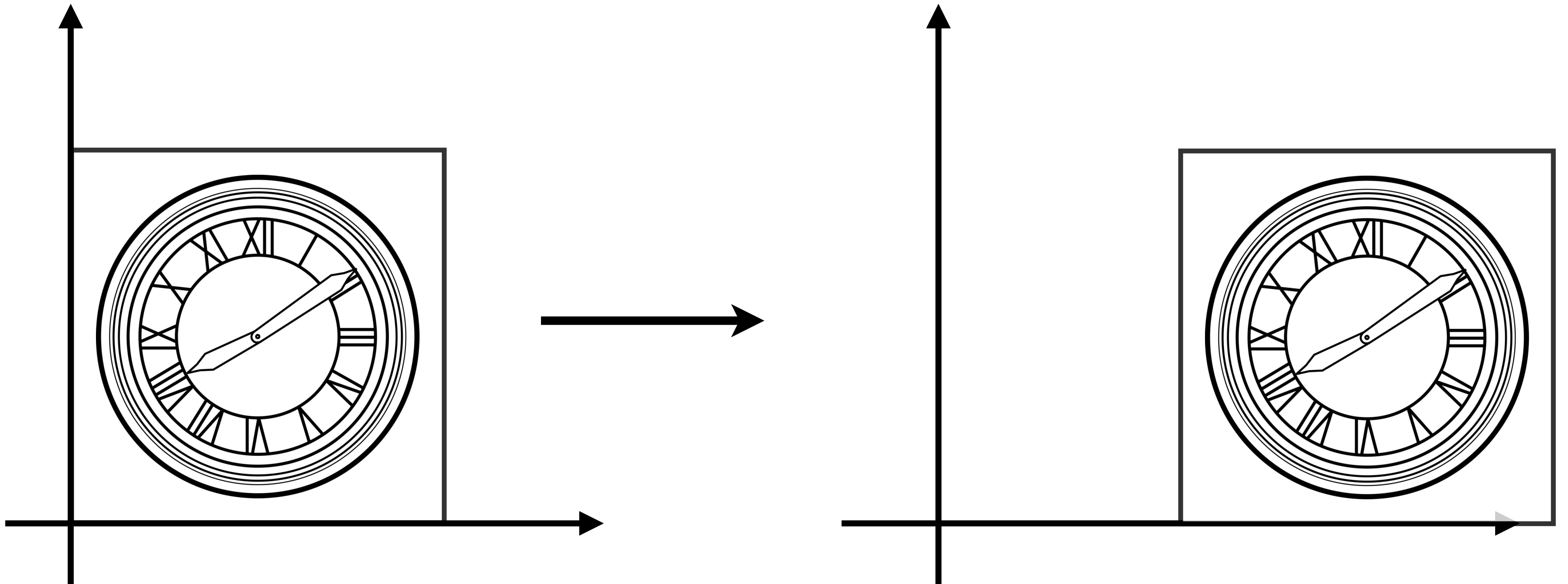
Reflection Transform



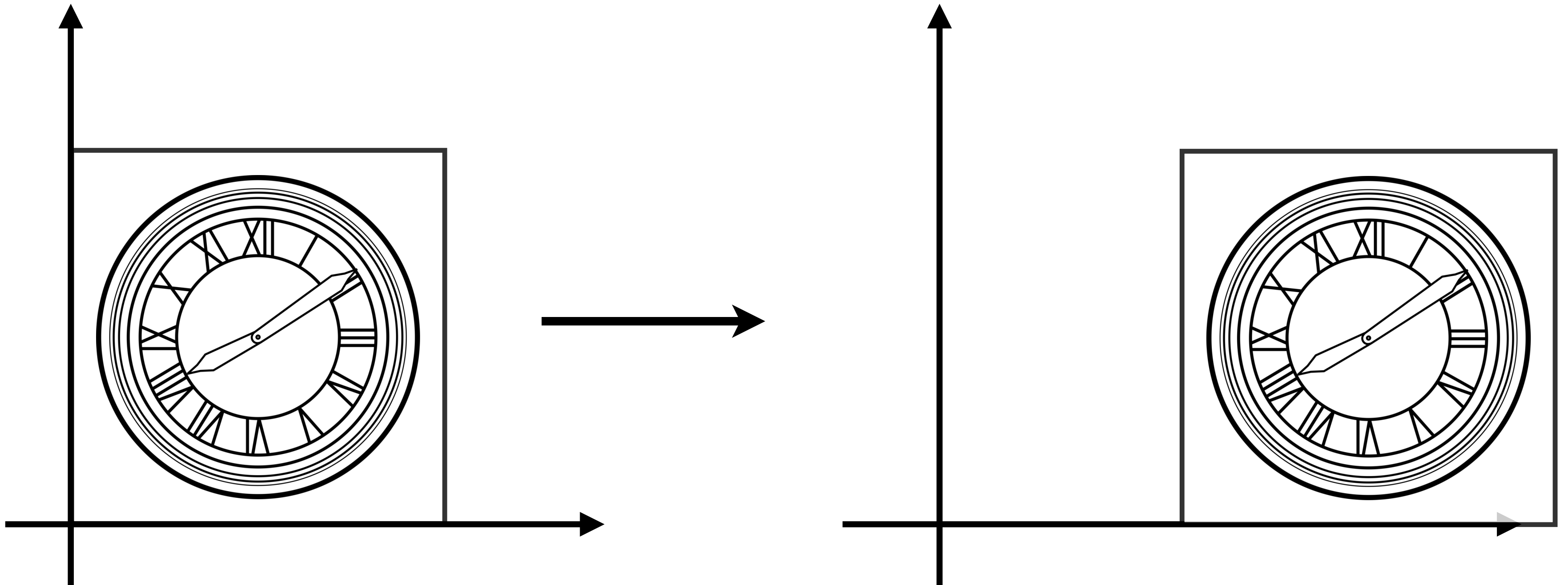
$$x' = -1.0x$$

$$y' = y$$

Translation Transform



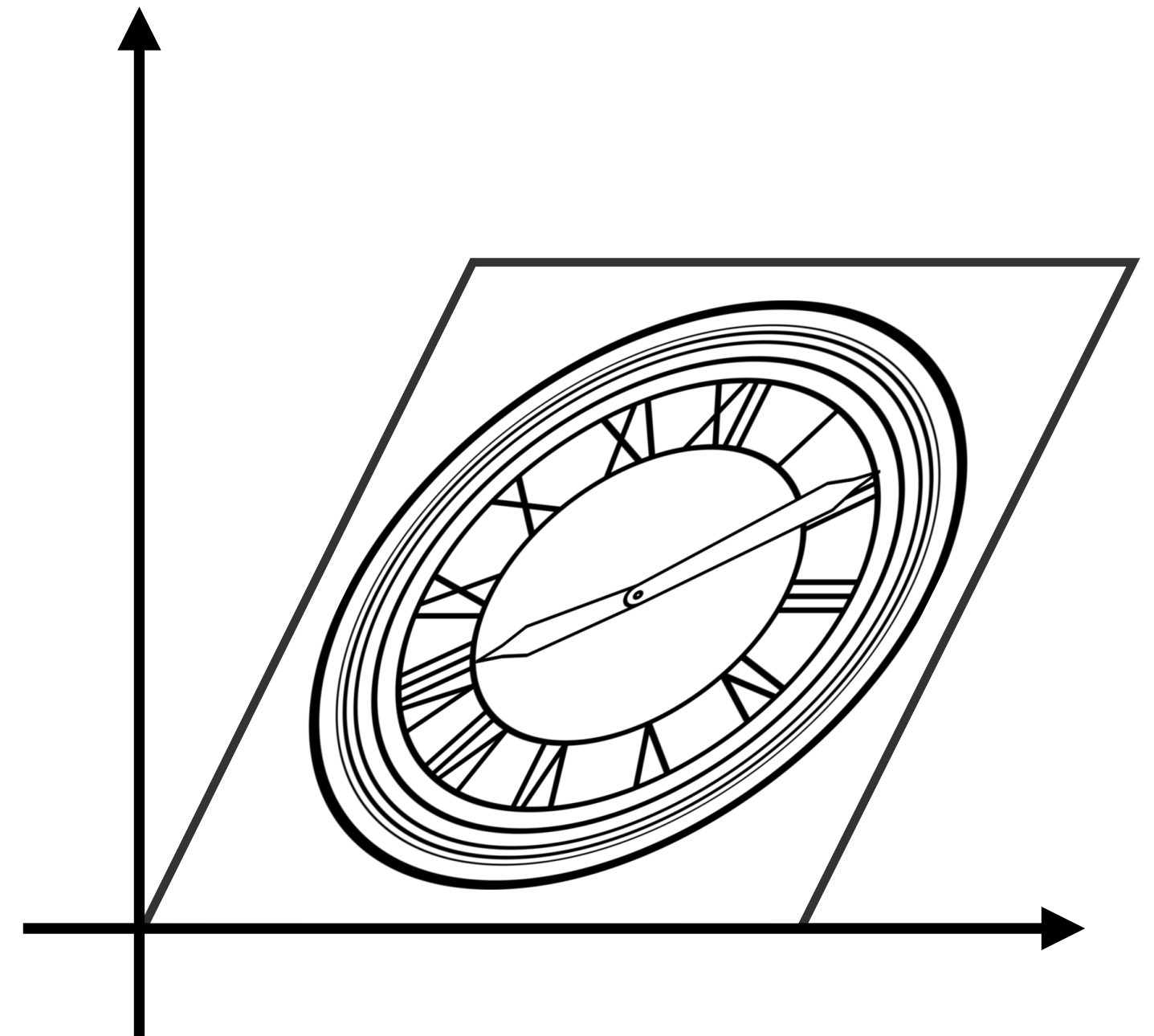
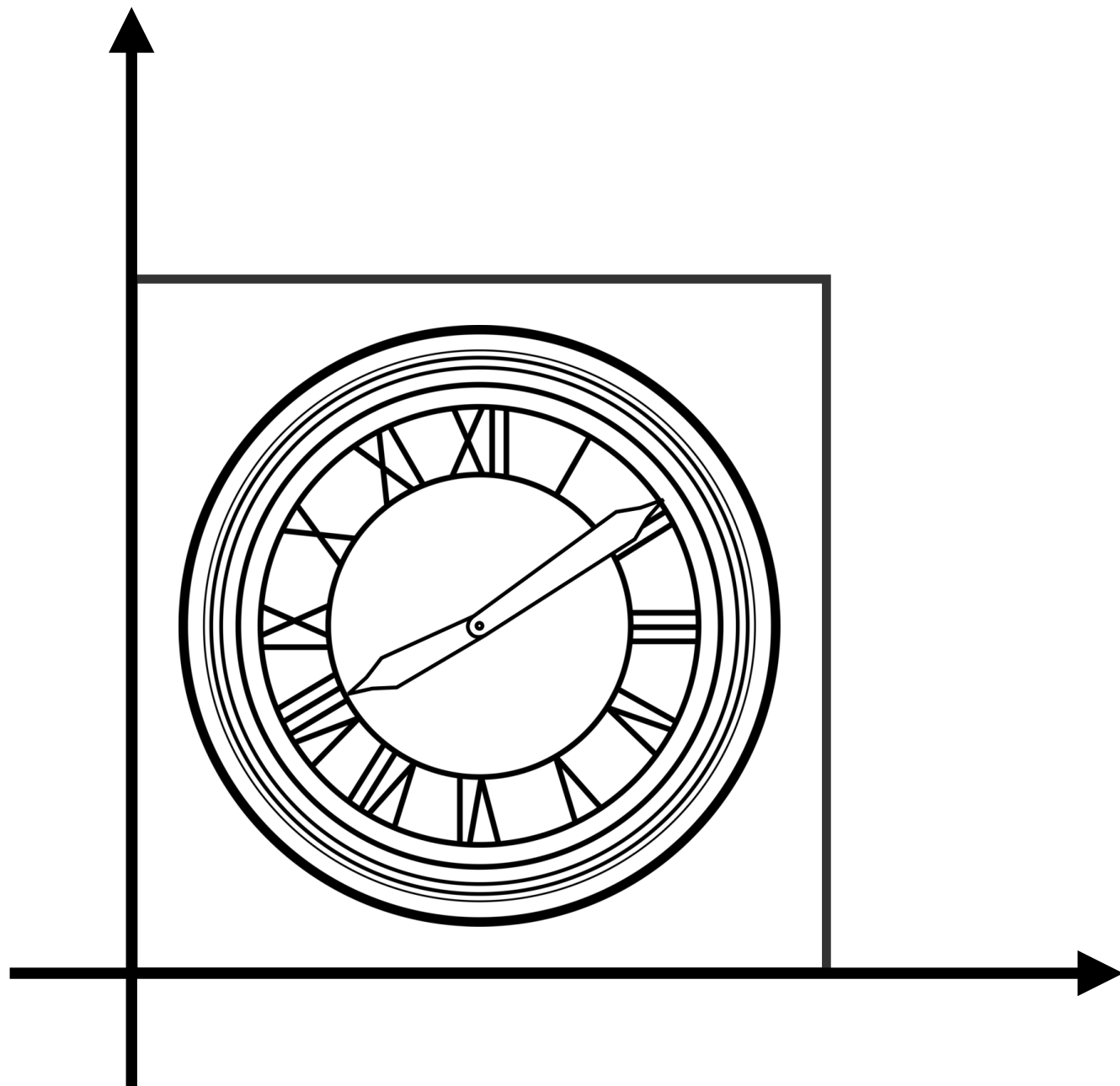
Translation Transform



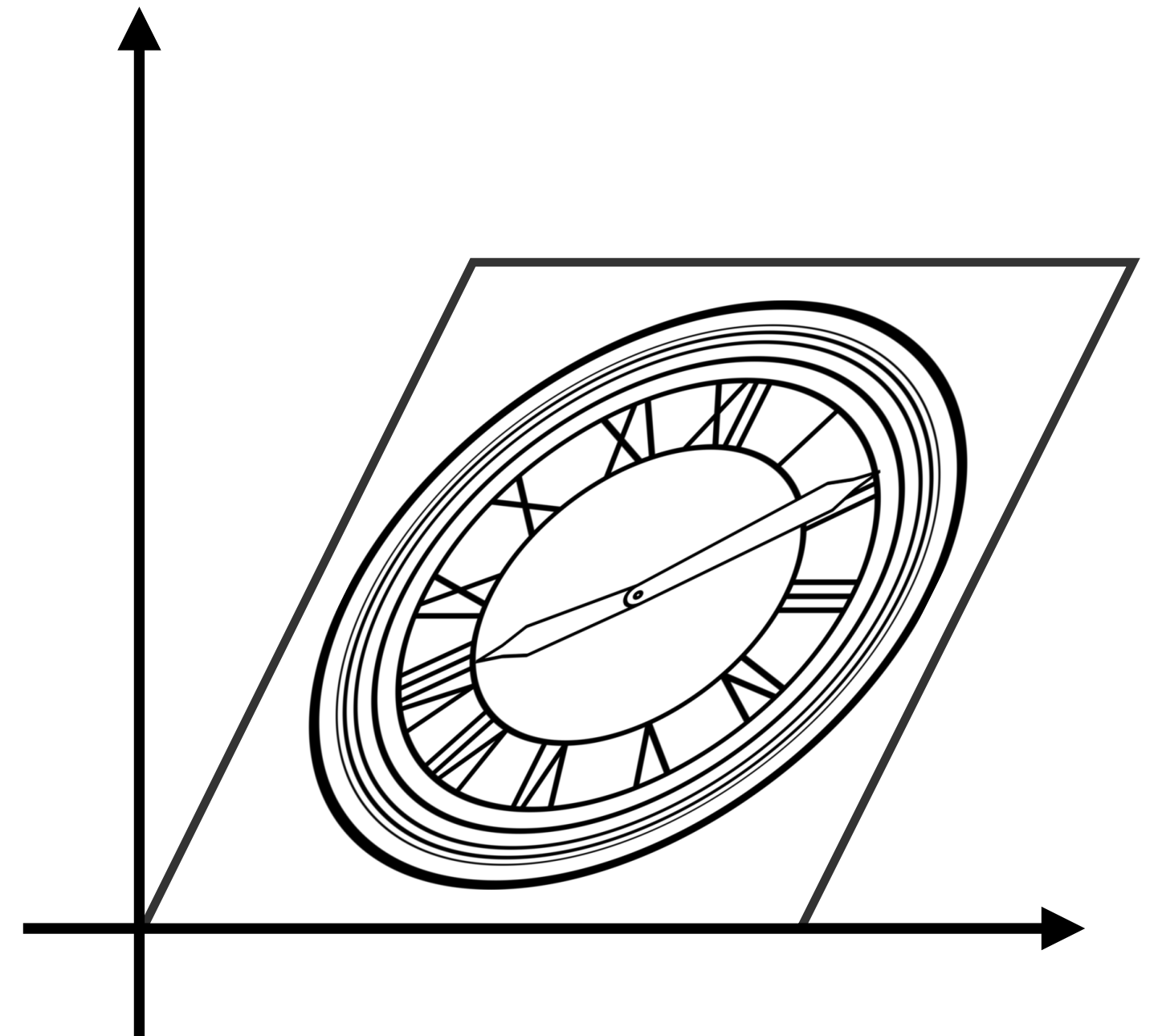
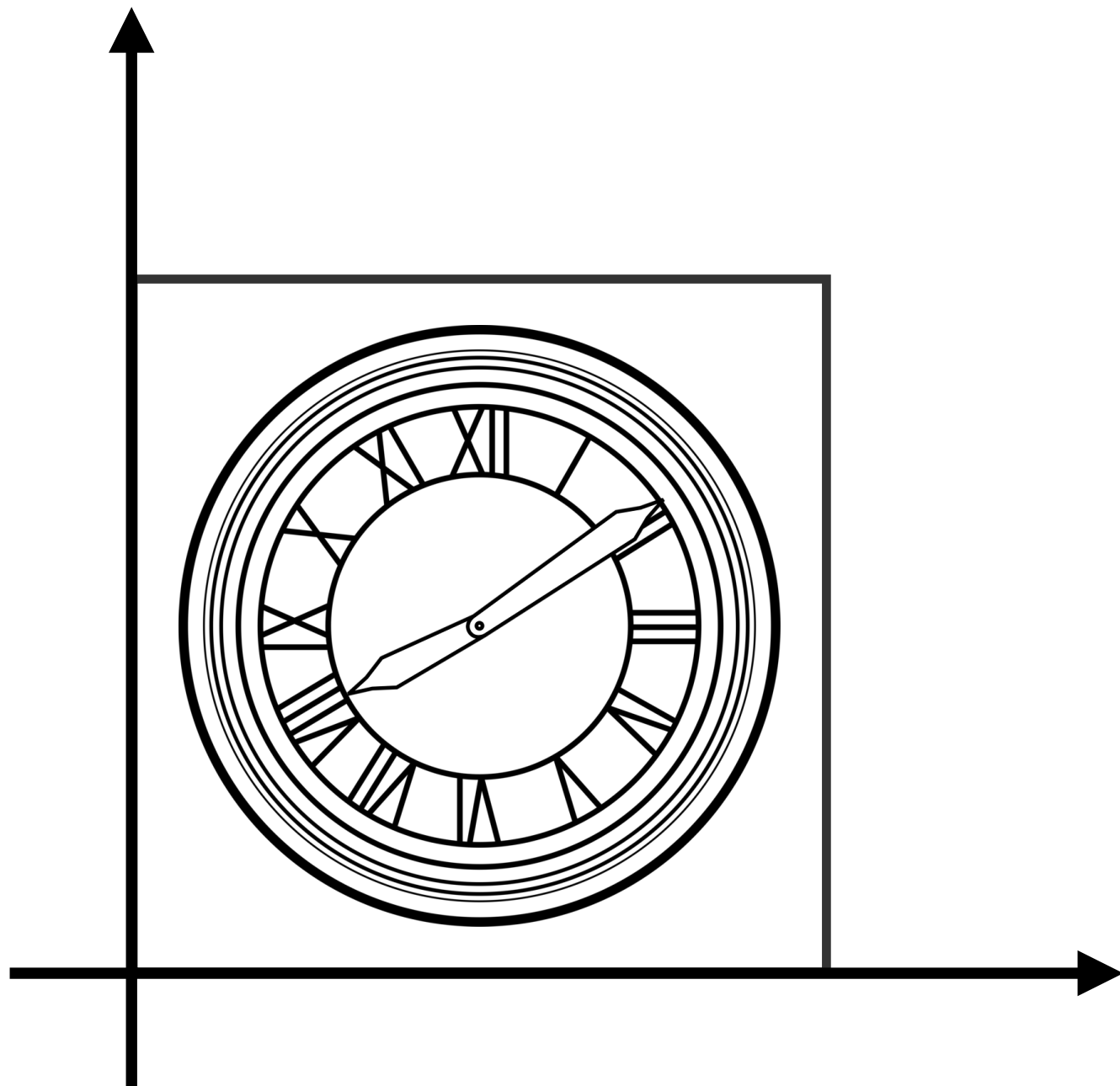
$$x' = x + 5$$

$$y' = y$$

Shear Transform



Shear Transform



$$x' = x + 0.5y$$

$$y' = y$$

Linear Transforms = Matrices

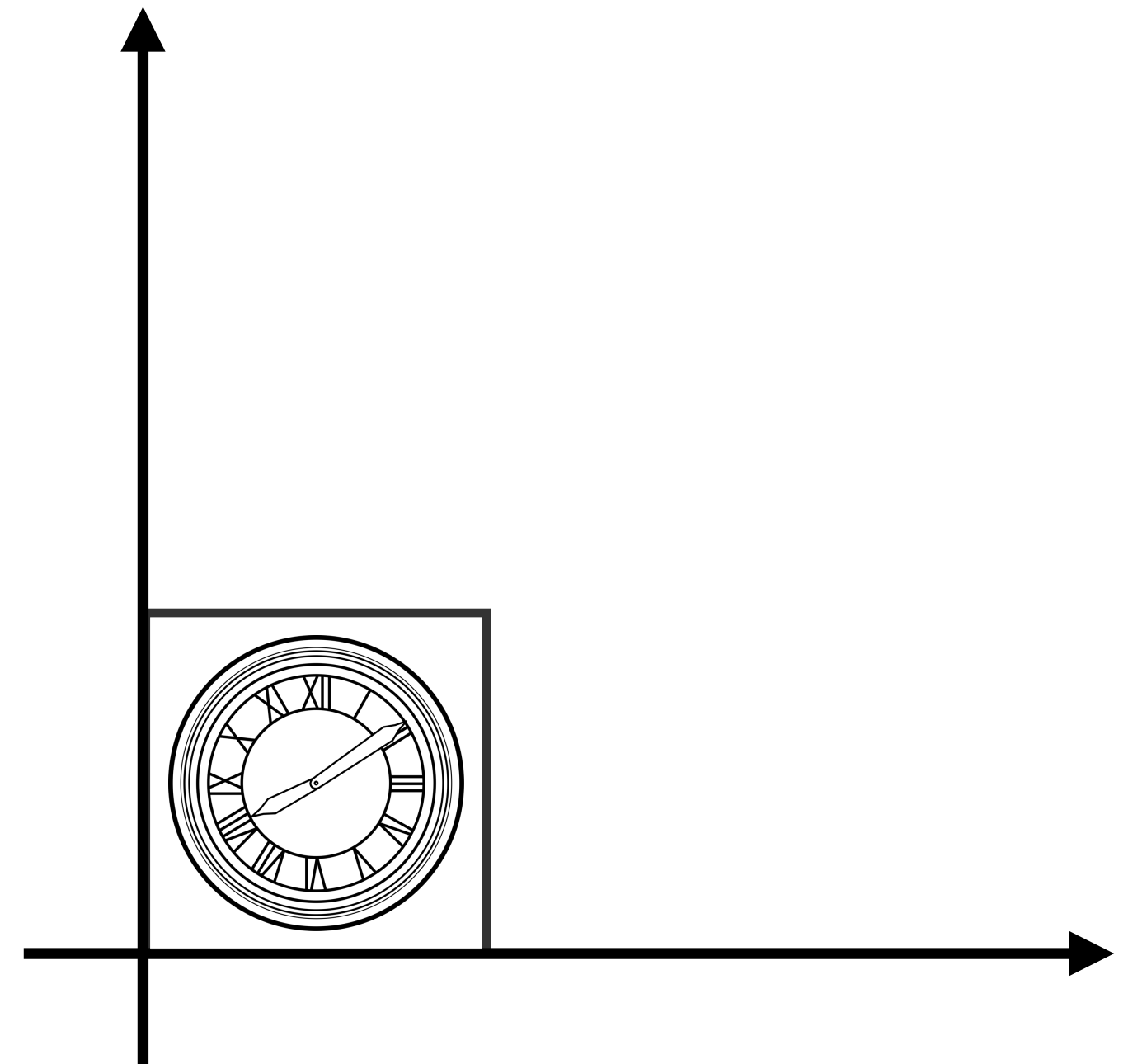
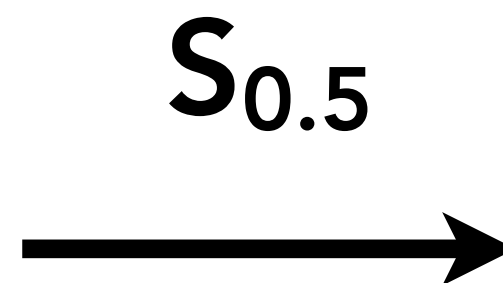
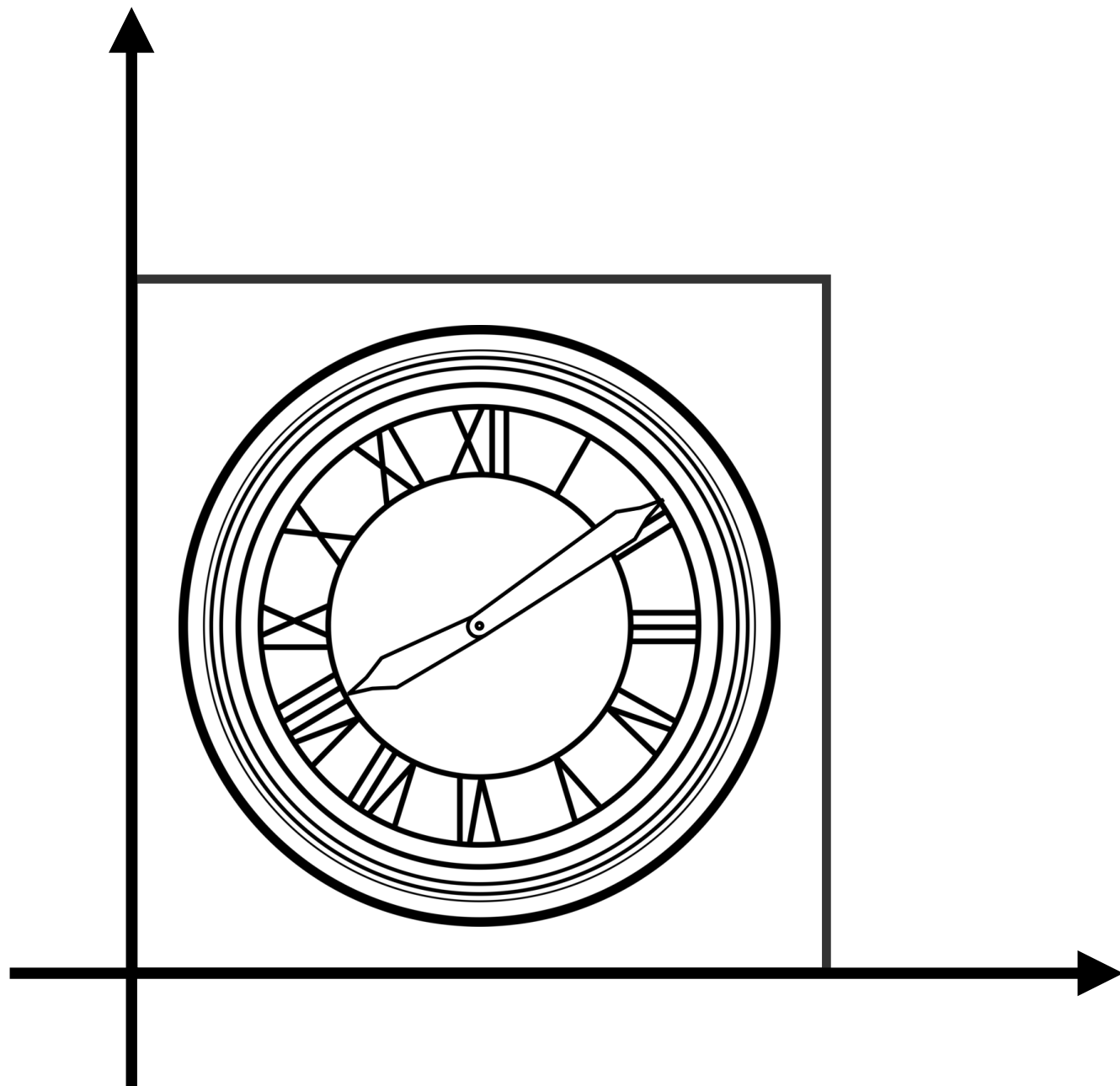
$$x' = a x + b y$$

$$y' = c x + d y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

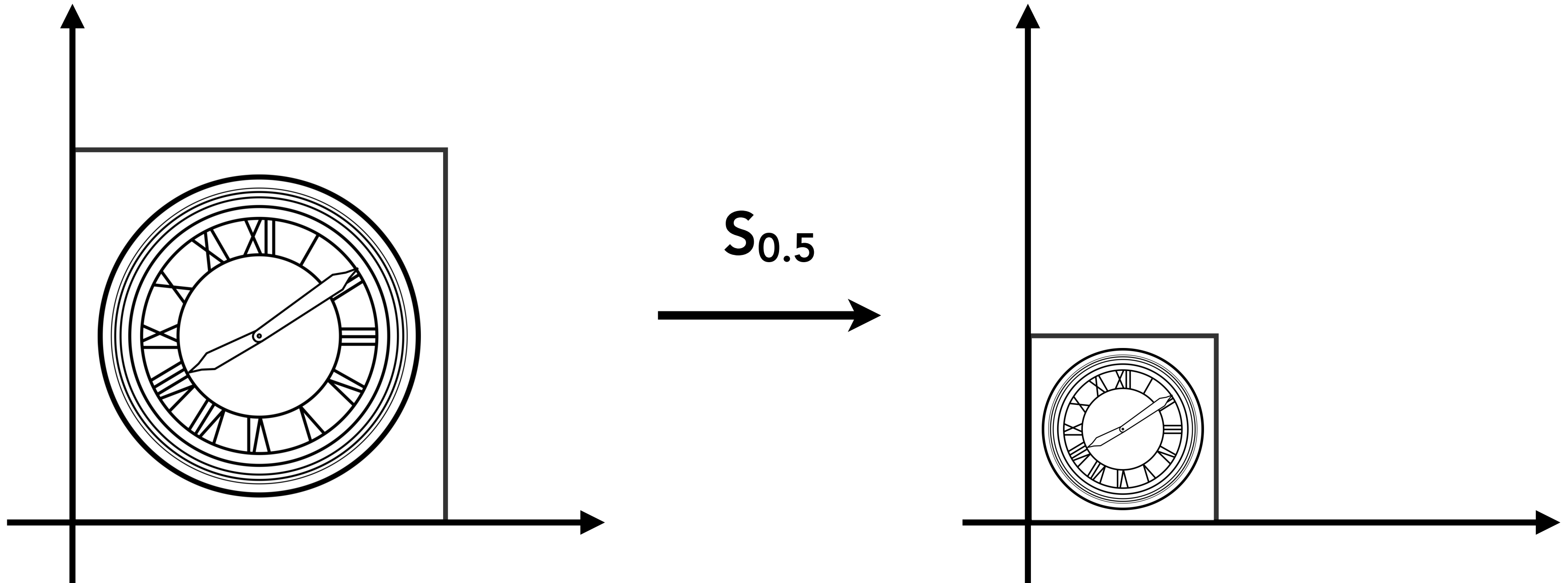
$$\mathbf{x}' = \mathbf{M} \mathbf{x}$$

Scale Matrix



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scale Matrix



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Linear Transforms = Matrices

$$x' = a x + b y$$

$$y' = c x + d y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{x}' = \mathbf{M} \mathbf{x}$$

$$\mathbf{b} = \mathbf{M}_1 \mathbf{a}$$

$$\mathbf{c} = \mathbf{M}_2 \mathbf{b}$$

$$\mathbf{d} = \mathbf{M}_3 \mathbf{c}$$

$$\mathbf{d} = \mathbf{M}_3 \mathbf{M}_2 \mathbf{M}_1 \mathbf{a}$$

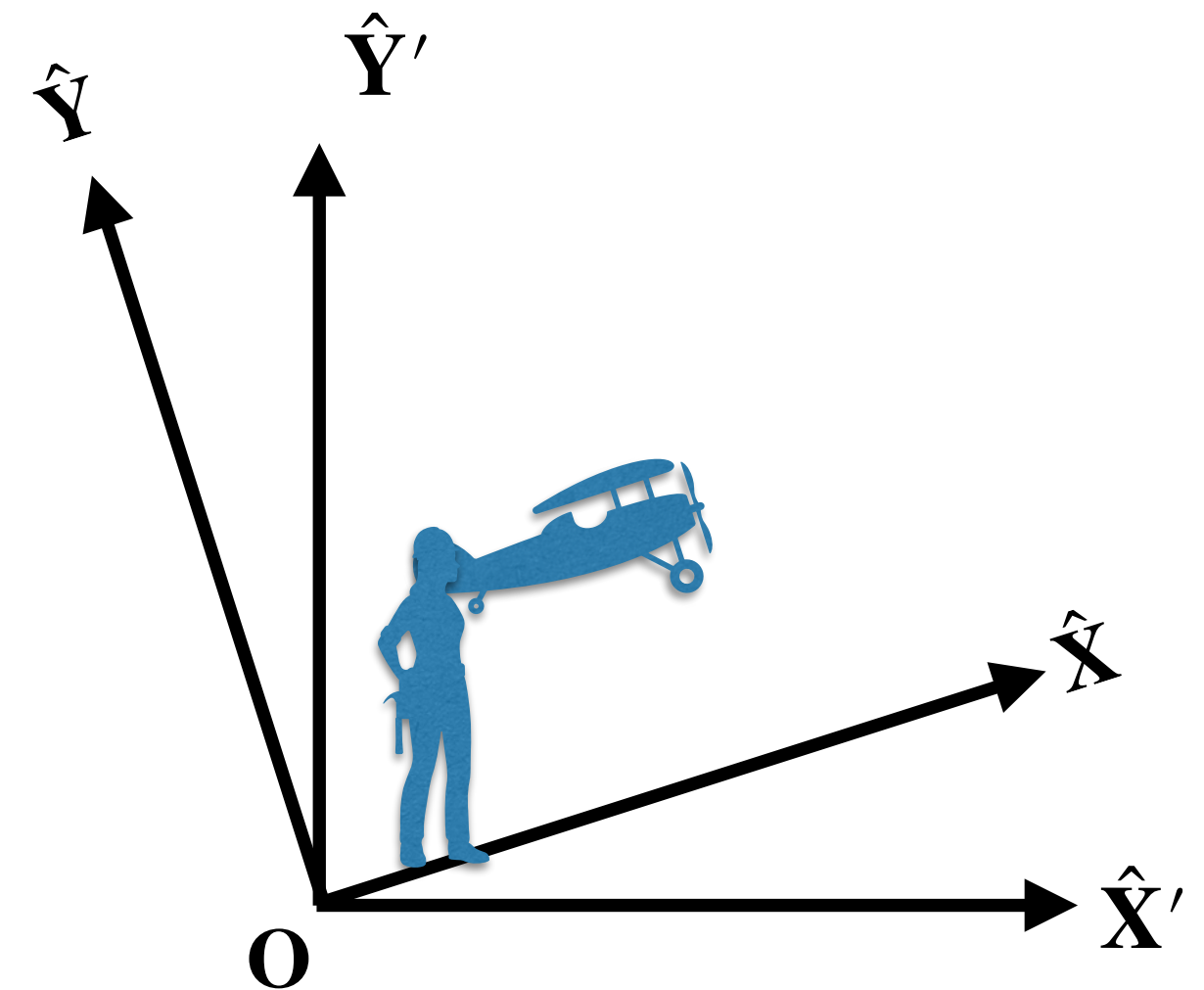
$$\mathbf{d} = (\mathbf{M}_3 \mathbf{M}_2 \mathbf{M}_1) \mathbf{a}$$

$$\mathbf{d} = \mathbf{M} \mathbf{a}$$

Note: no translation (addition) yet. Will come later...

2D Coordinate Systems

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



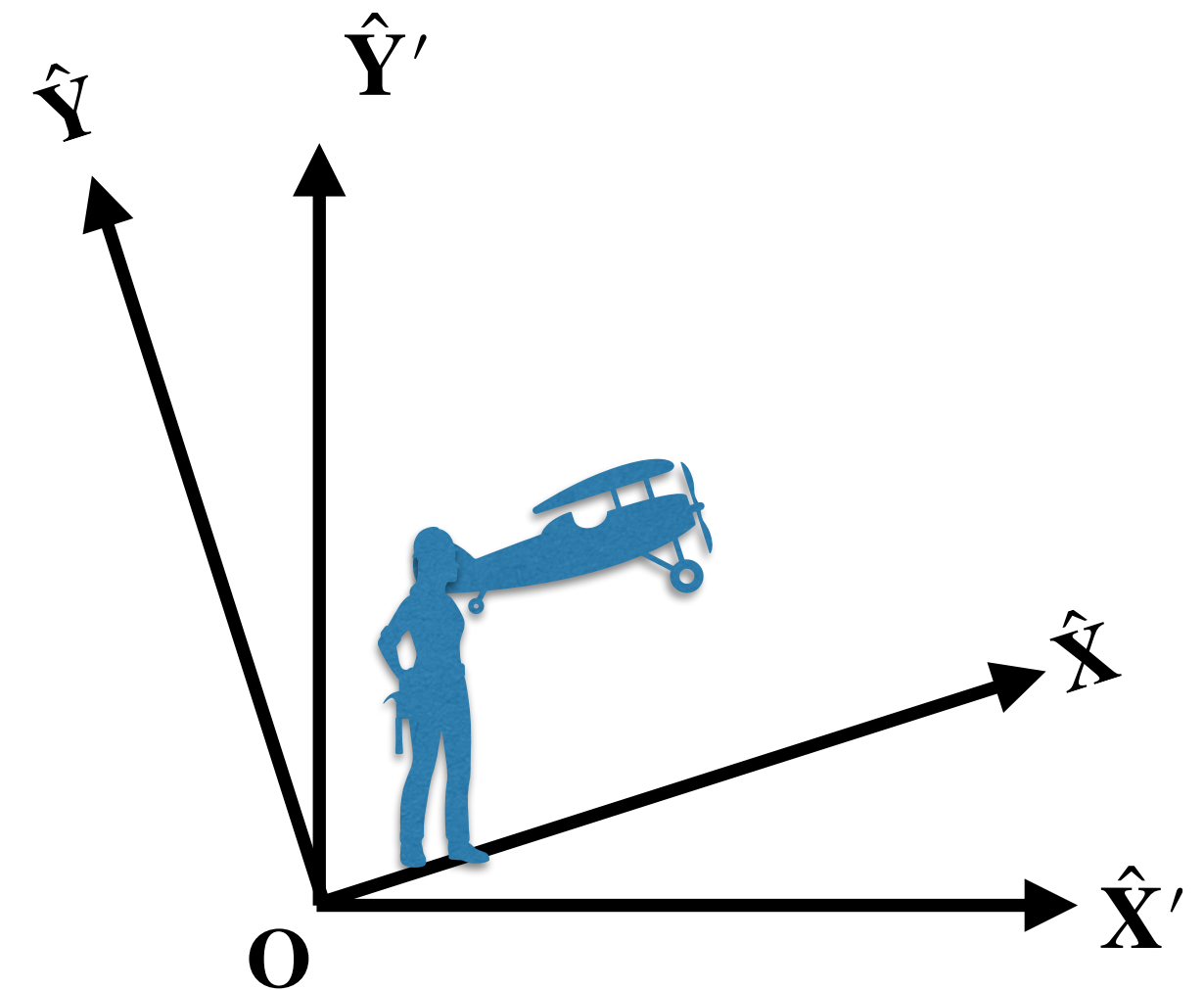
"The x-axis of the plane's coordinate system expressed in the person's coordinate system." $\begin{bmatrix} a \\ c \end{bmatrix}$

Can interpret the columns of the matrix as the x and y axes of the coordinate frame

2D Coordinate Systems

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} a \\ c \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$



"The x-axis of the plane's coordinate system expressed in the person's coordinate system." $\begin{bmatrix} a \\ c \end{bmatrix}$

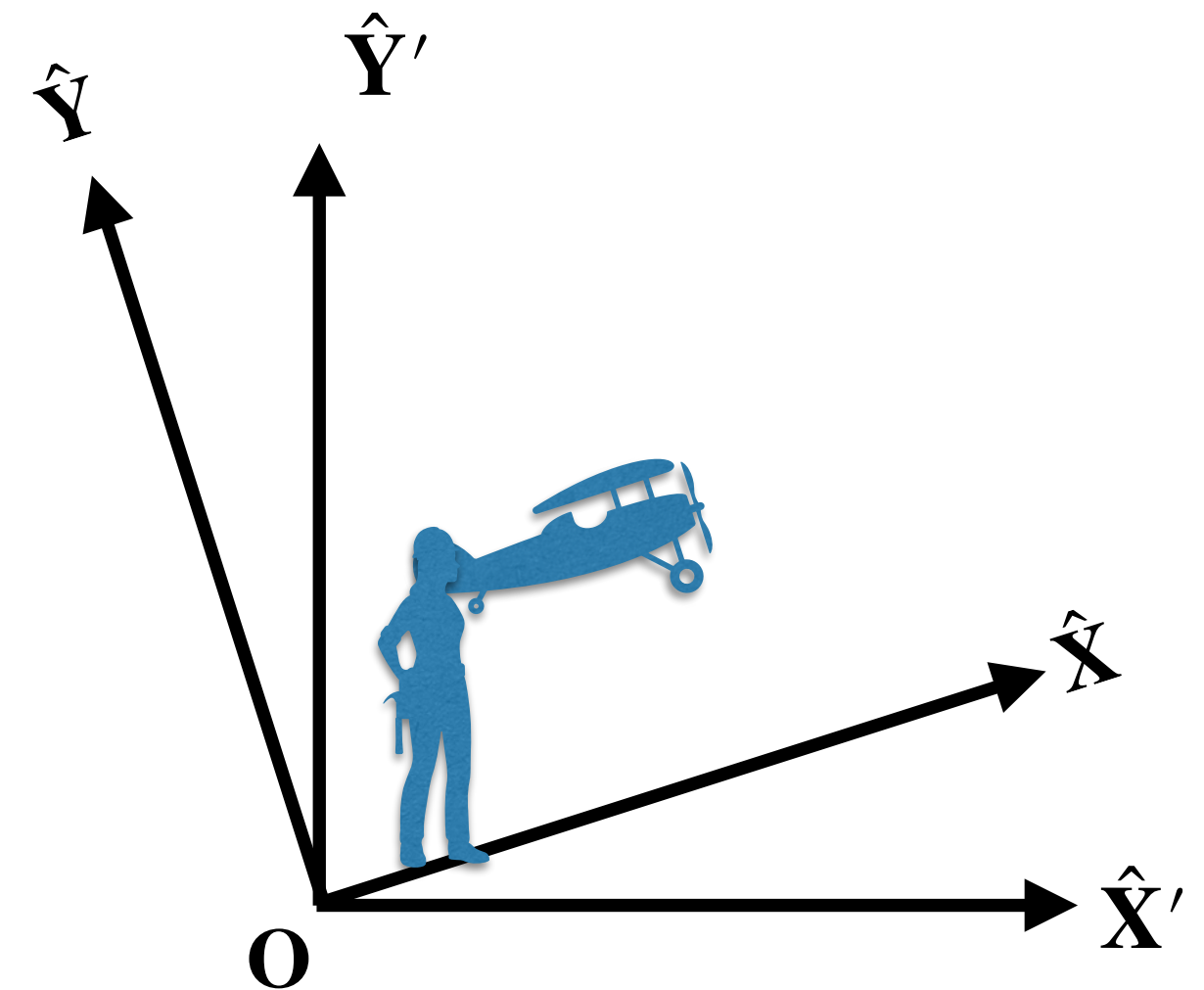
Can interpret the columns of the matrix as the x and y axes of the coordinate frame

2D Coordinate Systems

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} a \\ c \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$



"The x-axis of the plane's coordinate system expressed in the person's coordinate system." $\begin{bmatrix} a \\ c \end{bmatrix}$

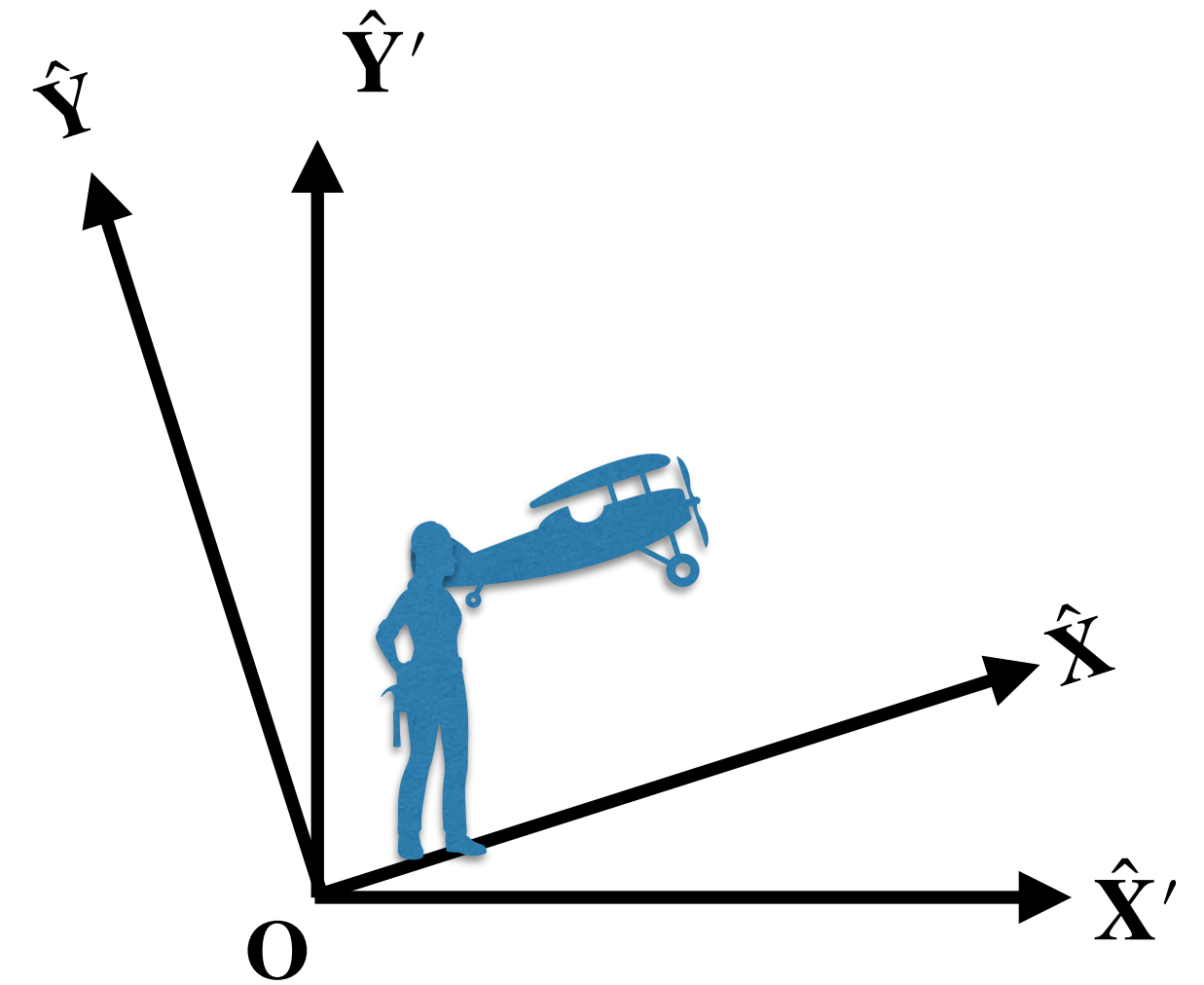
Can interpret the columns of the matrix as the x and y axes of the coordinate frame

2D Coordinate Systems

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

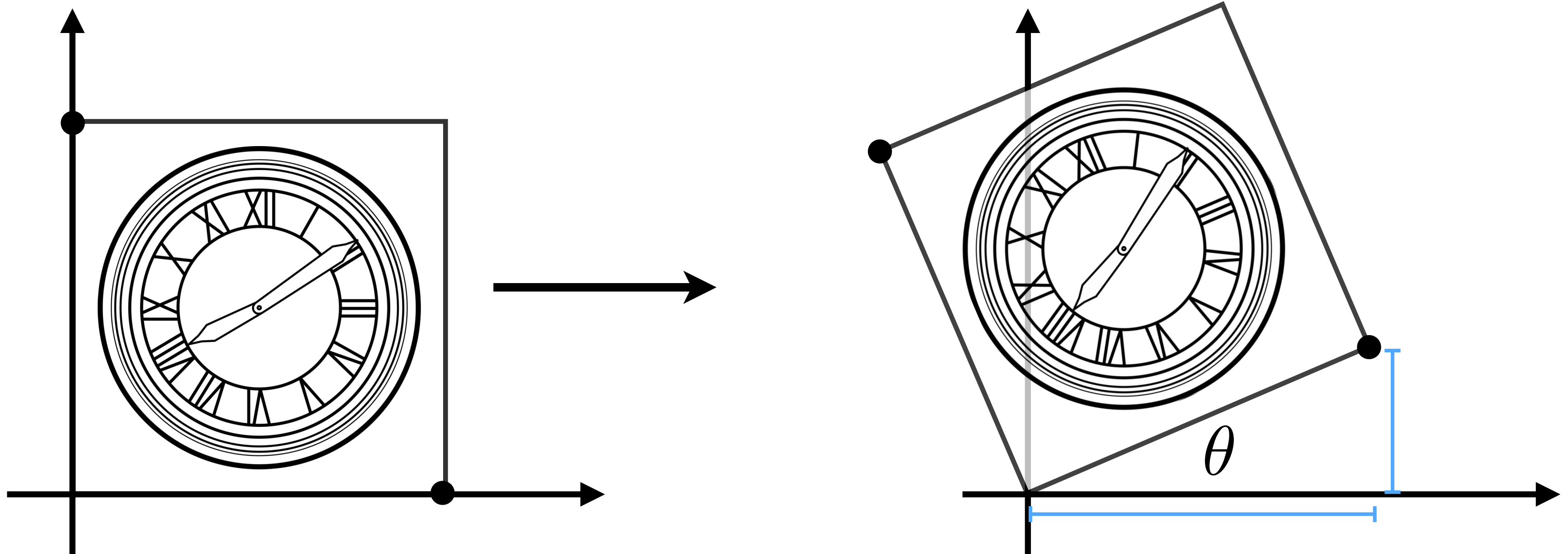
$$\begin{bmatrix} a \\ c \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

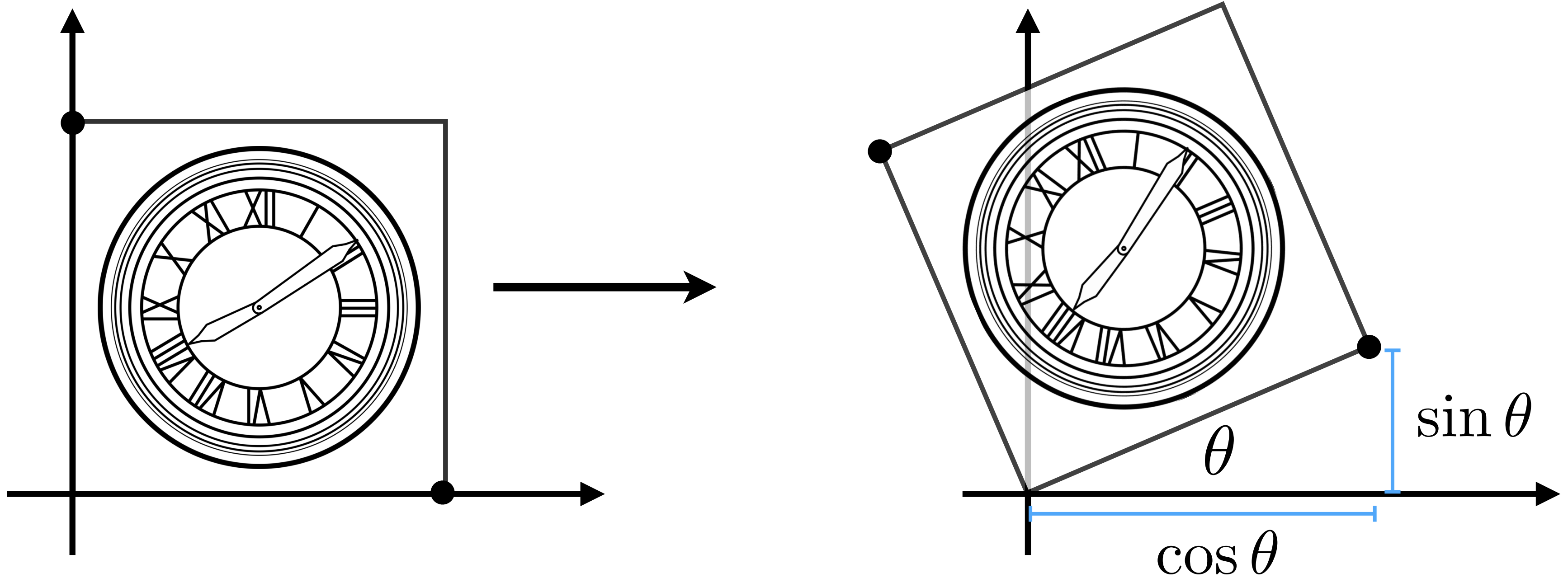


Consider the inverse matrix...

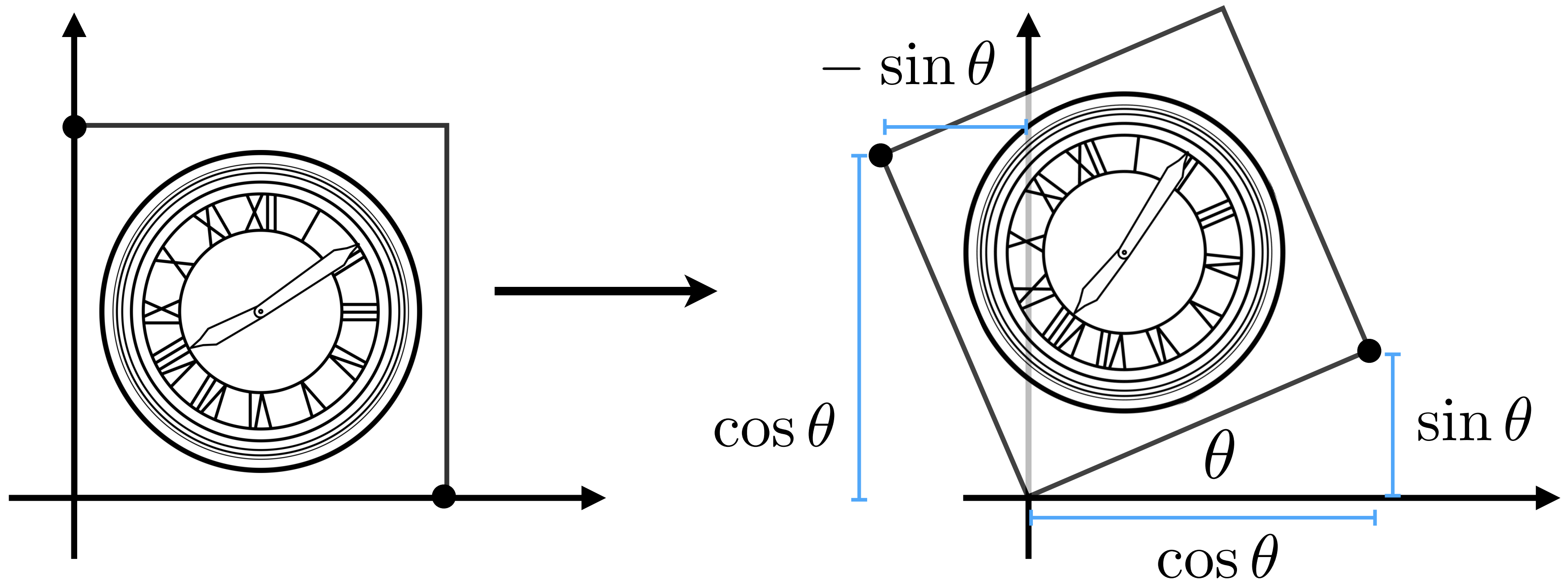
Rotation Matrix



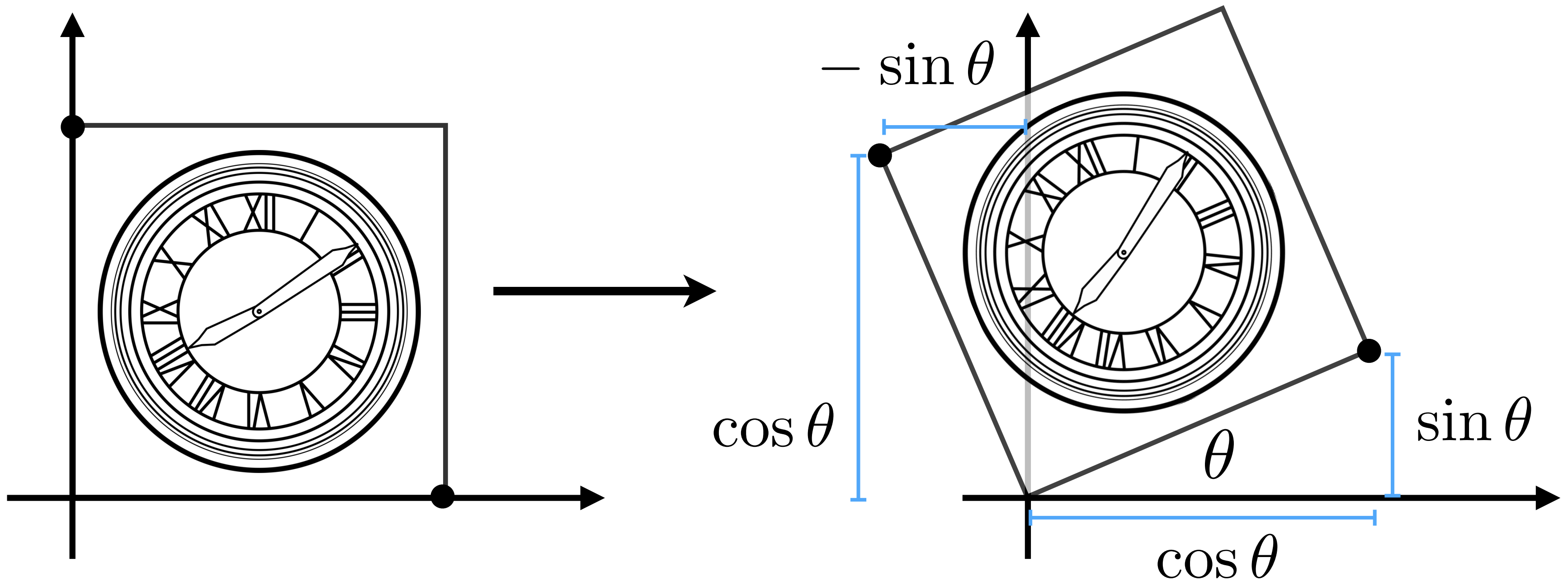
Rotation Matrix



Rotation Matrix

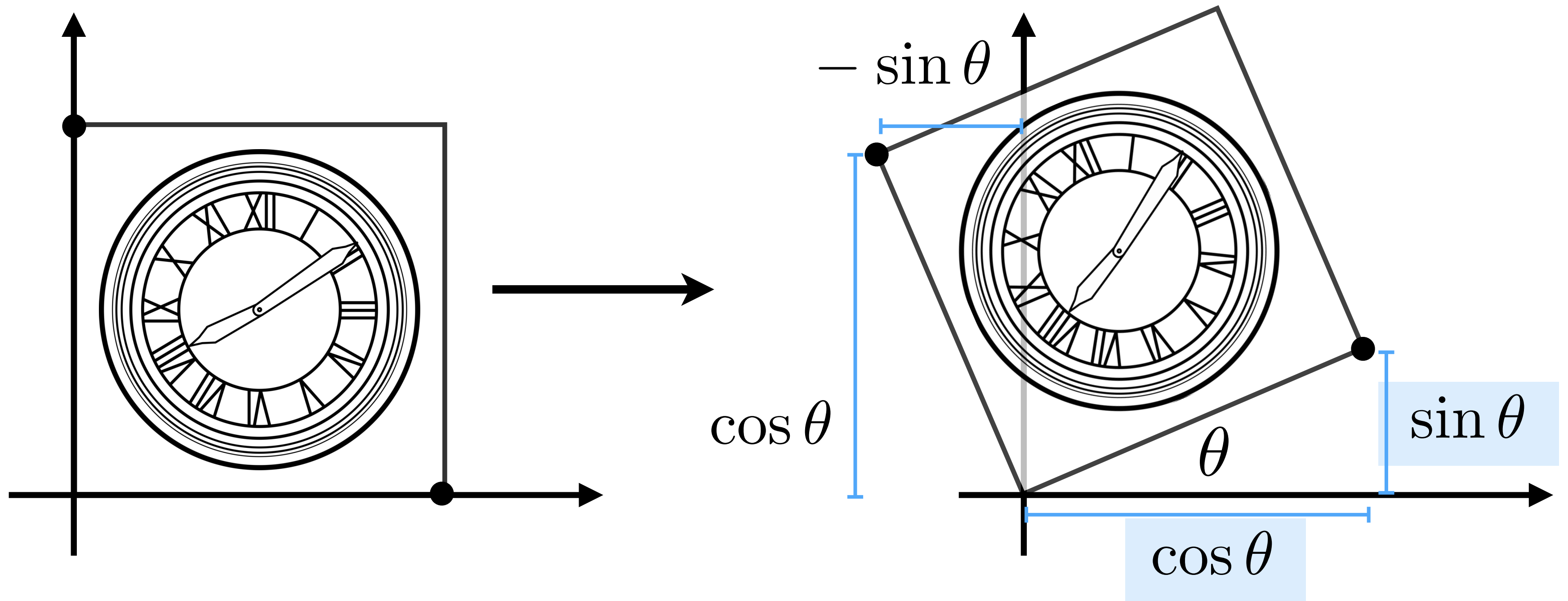


Rotation Matrix



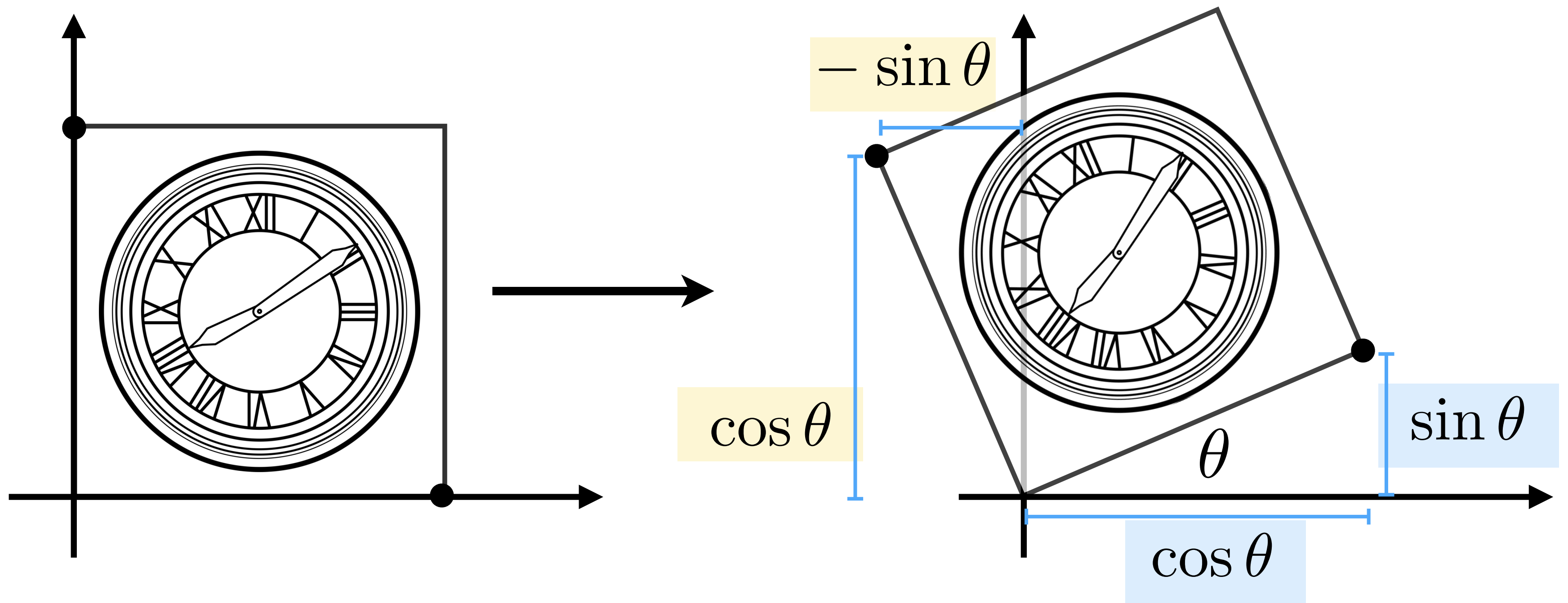
$$\mathbf{R}_{\theta} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Rotation Matrix



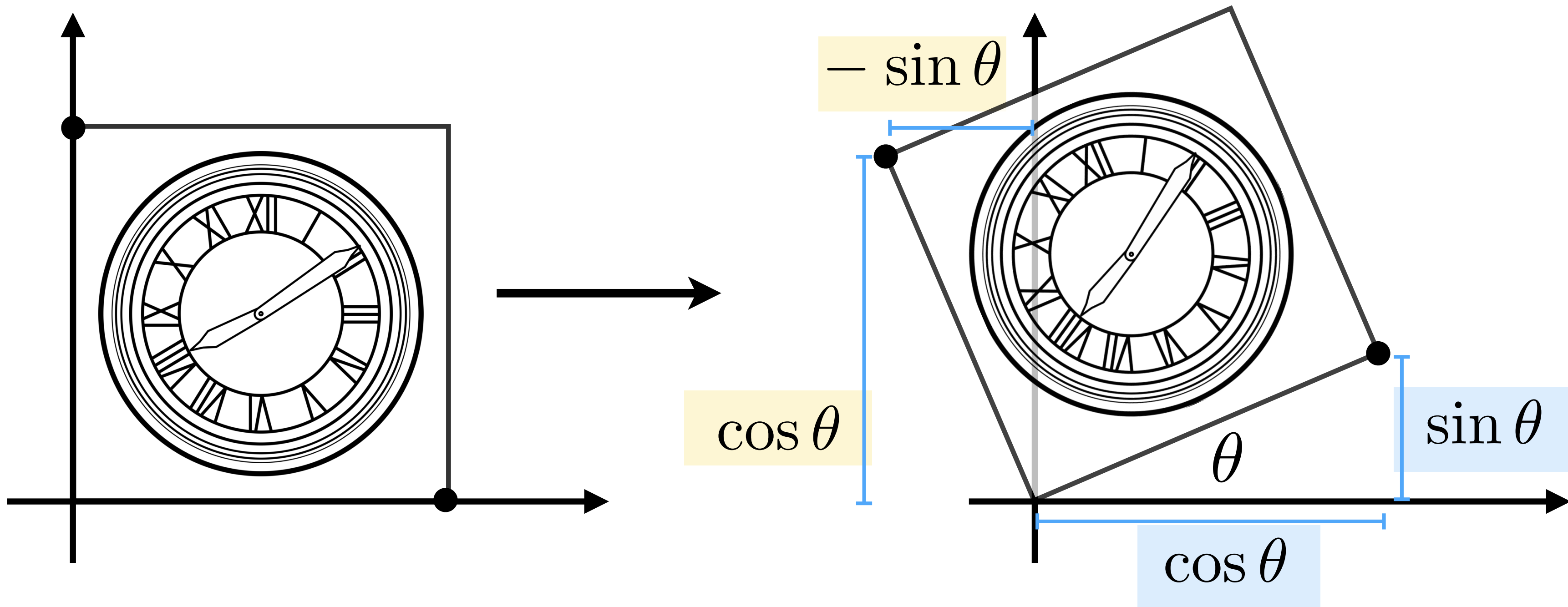
$$\mathbf{R}_{\theta} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Rotation Matrix



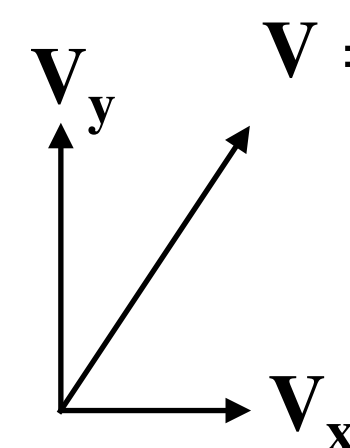
$$\mathbf{R}_{\theta} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Rotation Matrix



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

Where did this come from?



If you can figure out horizontal and vertical then you can do any vector!

2D Rotation Matrix: Another Way



A *WEBCOMIC OF ROMANCE,
SARCASM, MATH, AND LANGUAGE.*

$$\begin{bmatrix} \cos 90^\circ & \sin 90^\circ \\ -\sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} =$$

<http://xkcd.com/184/>

2D Rotation Matrix: Another Way



A *WEBCOMIC OF ROMANCE,
SARCASM, MATH, AND LANGUAGE.*

$$\begin{bmatrix} \cos 90^\circ & \sin 90^\circ \\ -\sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ a_1 \end{bmatrix}$$

<http://xkcd.com/184/>

2D Rotation Matrix: Another Way



A *WEBCOMIC OF ROMANCE,
SARCASM, MATH, AND LANGUAGE.*

$$\begin{bmatrix} \cos 90^\circ & \sin 90^\circ \\ -\sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ a_1 \end{bmatrix}$$

<http://xkcd.com/184/>

2D Rotation Matrix: Another Way

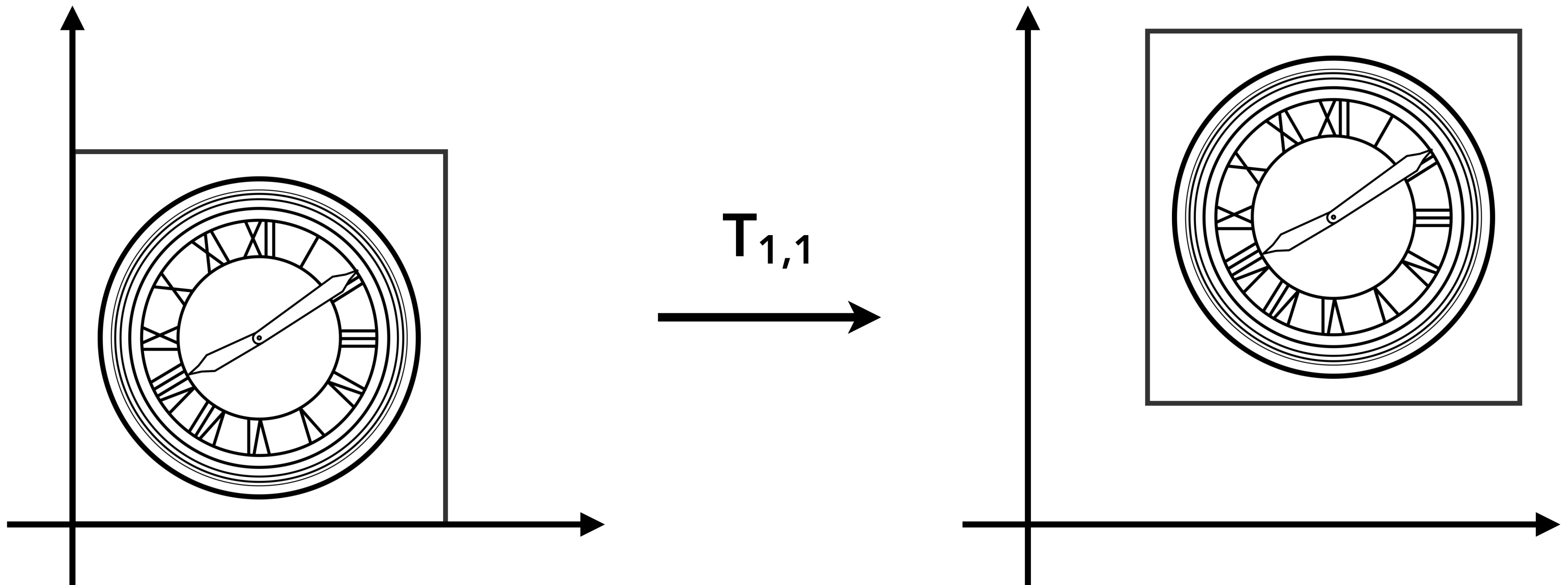


A *WEBCOMIC OF ROMANCE,
SARCASM, MATH, AND LANGUAGE.*

$$\begin{bmatrix} \cos 90^\circ & \sin 90^\circ \\ -\sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ a_1 \end{bmatrix}$$

<http://xkcd.com/184/>

Translation??



$$x' = x + t_x$$

$$y' = y + t_y$$

Solution: Homogenous Coordinates

Add a third coordinate (*w-coordinate*)

- 2D point = $(x, y, 1)^T$
- 2D vector = $(x, y, 0)^T$

Solution: Homogenous Coordinates

Add a third coordinate (*w-coordinate*)

- 2D point = $(x, y, 1)^T$
- 2D vector = $(x, y, 0)^T$

Now you can express translation as a matrix!!

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

$$\mathbf{M} \frac{\begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \frac{\mathbf{M} \begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \mathbf{M} \begin{bmatrix} a/3 \\ b/3 \\ c/3 \\ 1 \end{bmatrix}$$

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Valid operation if w -coordinate of result is 1 or 0

$$\mathbf{M} \frac{\begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \frac{\mathbf{M} \begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \mathbf{M} \begin{bmatrix} a/3 \\ b/3 \\ c/3 \\ 1 \end{bmatrix}$$

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Valid operation if w -coordinate of result is 1 or 0

- vector + vector = vector

$$\mathbf{M} \frac{\begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \frac{\mathbf{M} \begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \mathbf{M} \begin{bmatrix} a/3 \\ b/3 \\ c/3 \\ 1 \end{bmatrix}$$

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Valid operation if w -coordinate of result is 1 or 0

- vector + vector = vector
- point - point = vector

$$\mathbf{M} \frac{\begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \frac{\mathbf{M} \begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \mathbf{M} \begin{bmatrix} a/3 \\ b/3 \\ c/3 \\ 1 \end{bmatrix}$$

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Valid operation if w -coordinate of result is 1 or 0

- vector + vector = vector
- point - point = vector
- point + vector = point

$$\mathbf{M} \frac{\begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \frac{\mathbf{M} \begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \mathbf{M} \begin{bmatrix} a/3 \\ b/3 \\ c/3 \\ 1 \end{bmatrix}$$

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Valid operation if w -coordinate of result is 1 or 0

- vector + vector = vector
- point - point = vector
- point + vector = point
- point + point = ??

$$\mathbf{M} \frac{\begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \frac{\mathbf{M} \begin{bmatrix} a \\ b \\ c \\ 3 \end{bmatrix}}{3} = \mathbf{M} \begin{bmatrix} a/3 \\ b/3 \\ c/3 \\ 1 \end{bmatrix}$$

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Valid operation if w -coordinate of result is 1 or 0

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Valid operation if w -coordinate of result is 1 or 0

- vector + vector = vector

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Valid operation if w -coordinate of result is 1 or 0

- vector + vector = vector
- point - point = vector

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Valid operation if w -coordinate of result is 1 or 0

- vector + vector = vector
- point - point = vector
- point + vector = point

Homogenous Coordinates

If w is not zero but not one either

- Divide entire vector by w
- Generally you can do it later

Equivalent

$$\tilde{\mathbf{x}} = \begin{bmatrix} 64 \\ 12 \\ 30 \\ 2 \end{bmatrix} \rightarrow \tilde{\mathbf{x}} = \begin{bmatrix} 32 \\ 6 \\ 15 \\ 1 \end{bmatrix}$$

Valid operation if w -coordinate of result is 1 or 0

- vector + vector = vector
- point - point = vector
- point + vector = point
- point + point = ??

Affine Transformations

Affine map = linear map + translation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Using homogenous coordinates:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

2D Transformations

(homogenous coordinates)

Scale

$$\mathbf{S}(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation

$$\mathbf{R}(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Translation

$$\mathbf{T}(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

2D Transformations

(homogenous coordinates)

Scale

$$\mathbf{S}(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation

$$\mathbf{R}(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Translation

$$\mathbf{T}(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

“Rigid Transform”

2D Transformations

(homogenous coordinates)

"Projective Transform"
(any matrix)

Scale

$$\mathbf{S}(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

"Affine Transform"

**"Similarity Transform
if $s_x=s_y$ "**

Rotation

$$\mathbf{R}(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Translation

$$\mathbf{T}(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

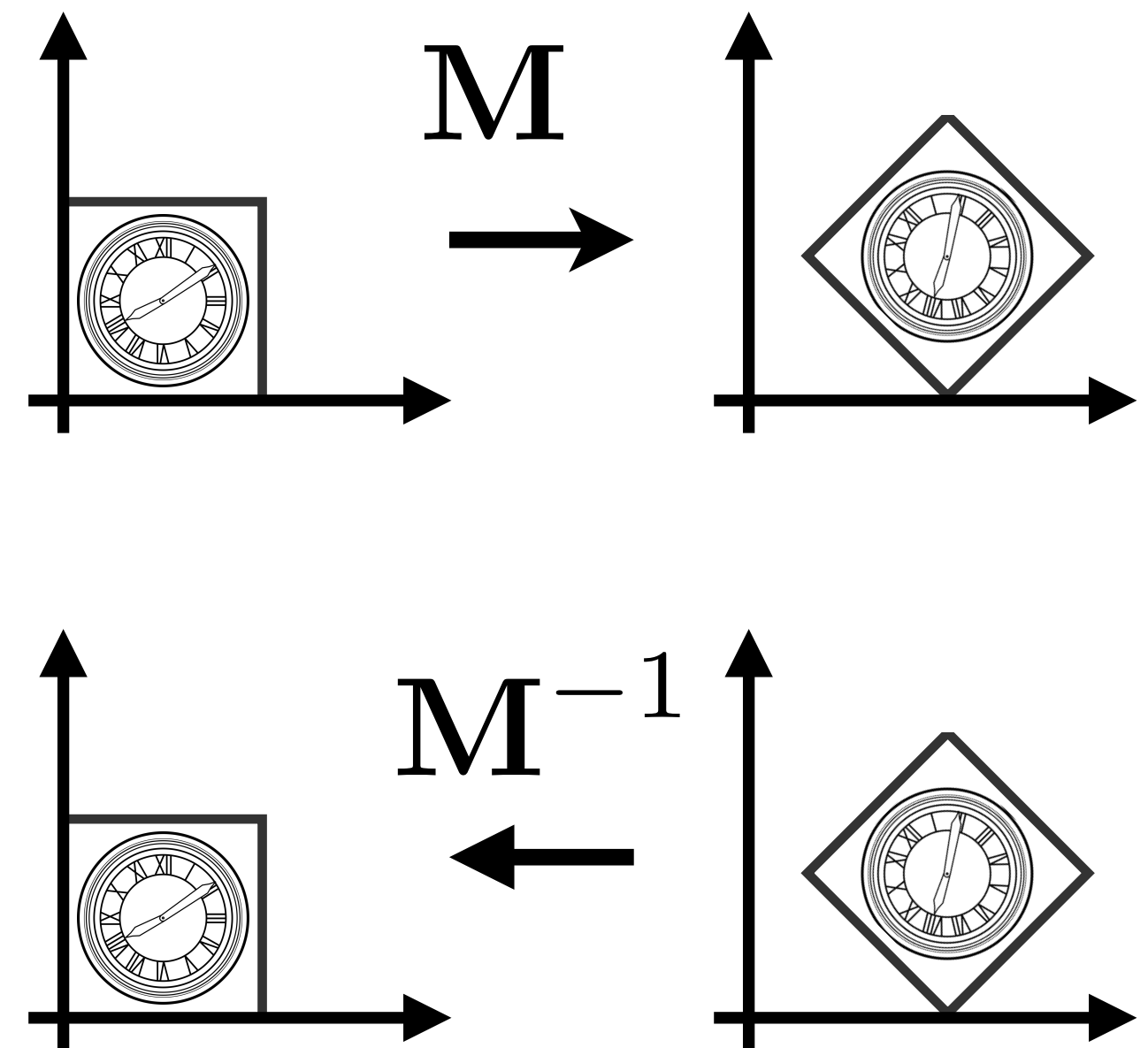
"Rigid Transform"

Inverse Transform

$$\mathbf{M}^{-1}$$

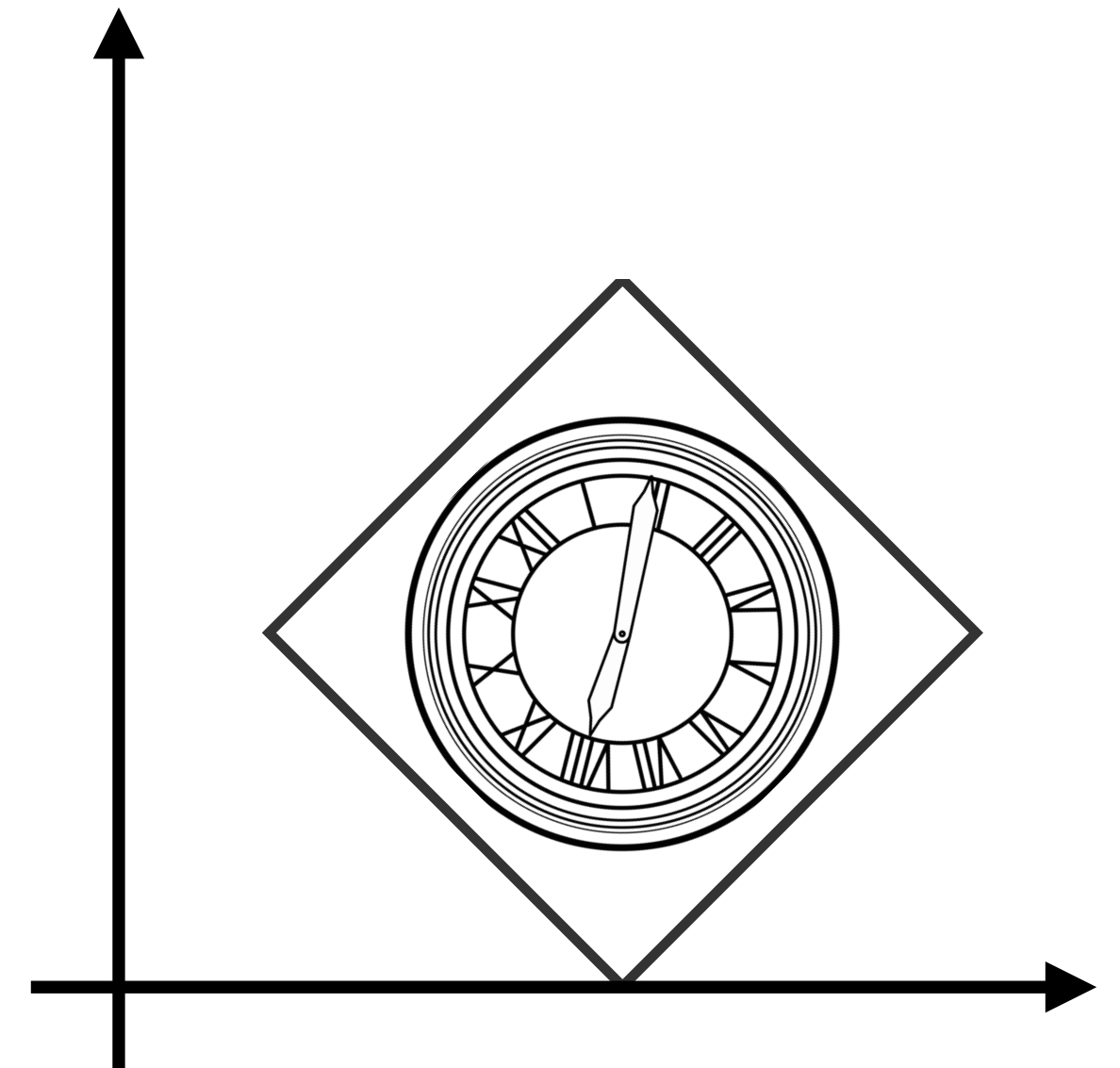
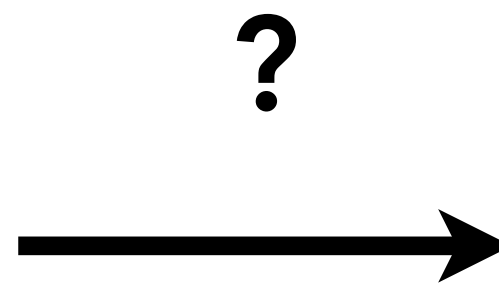
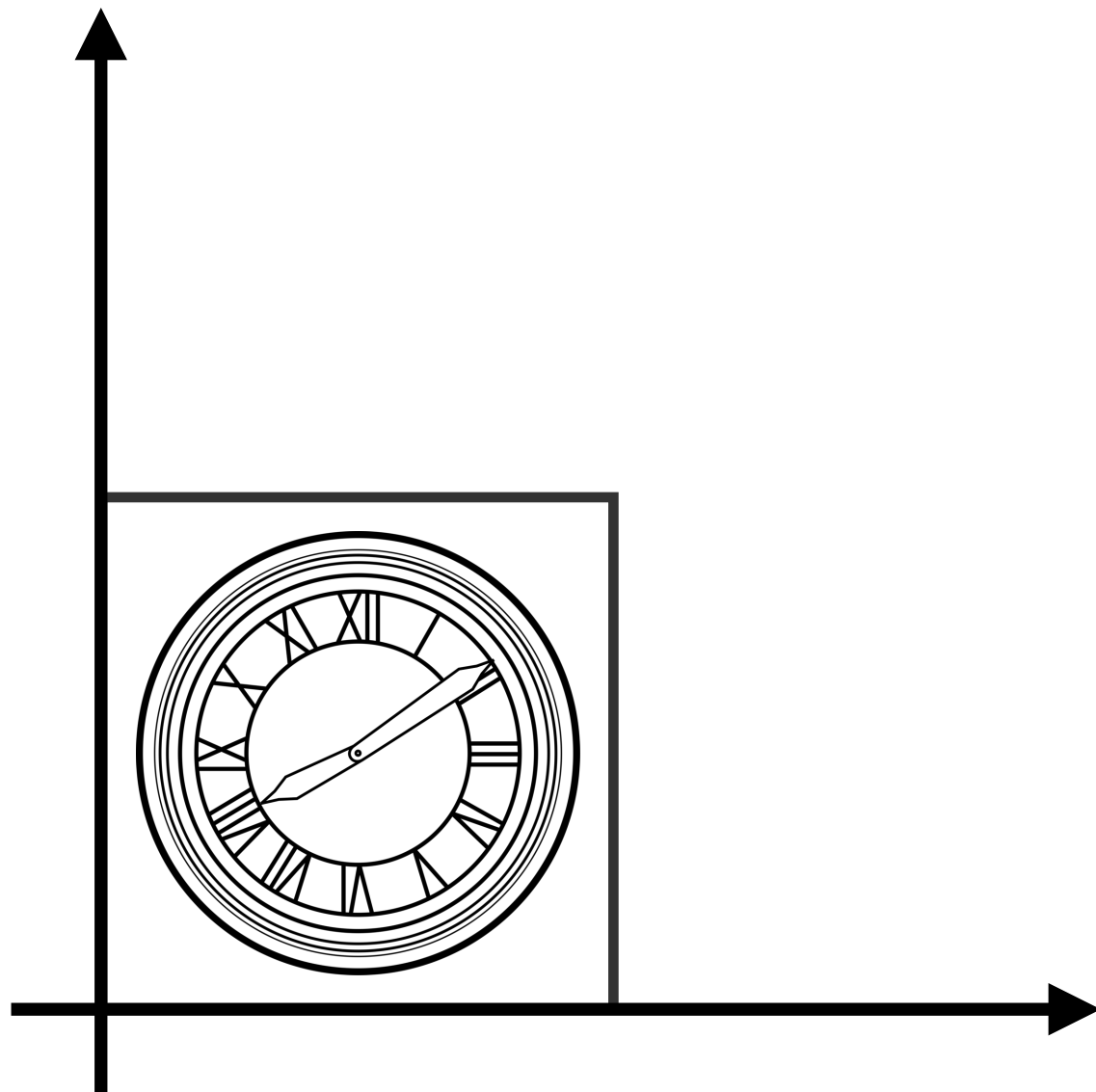
$\mathbf{M}^{-1}$ is the inverse of transform $\mathbf{M}$ in both a matrix and geometric sense

- Special cases:
 - Translation: negate t_x and t_y
 - Rotation: transpose
 - Scale: invert diagonal (axis-aligned scales)
- Others:
 - Invert matrix
 - Invert SVD matrices

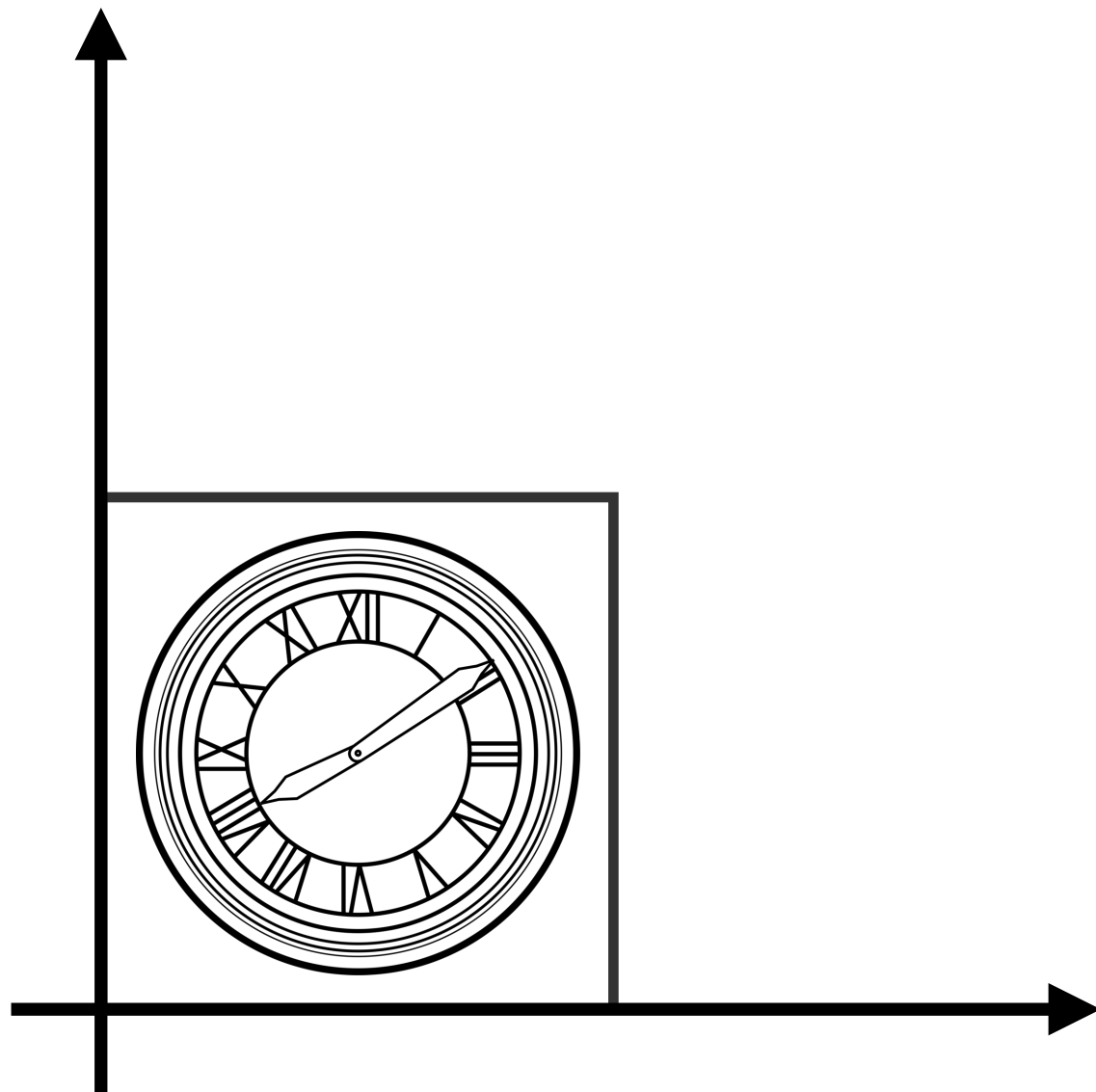


Composing Transforms

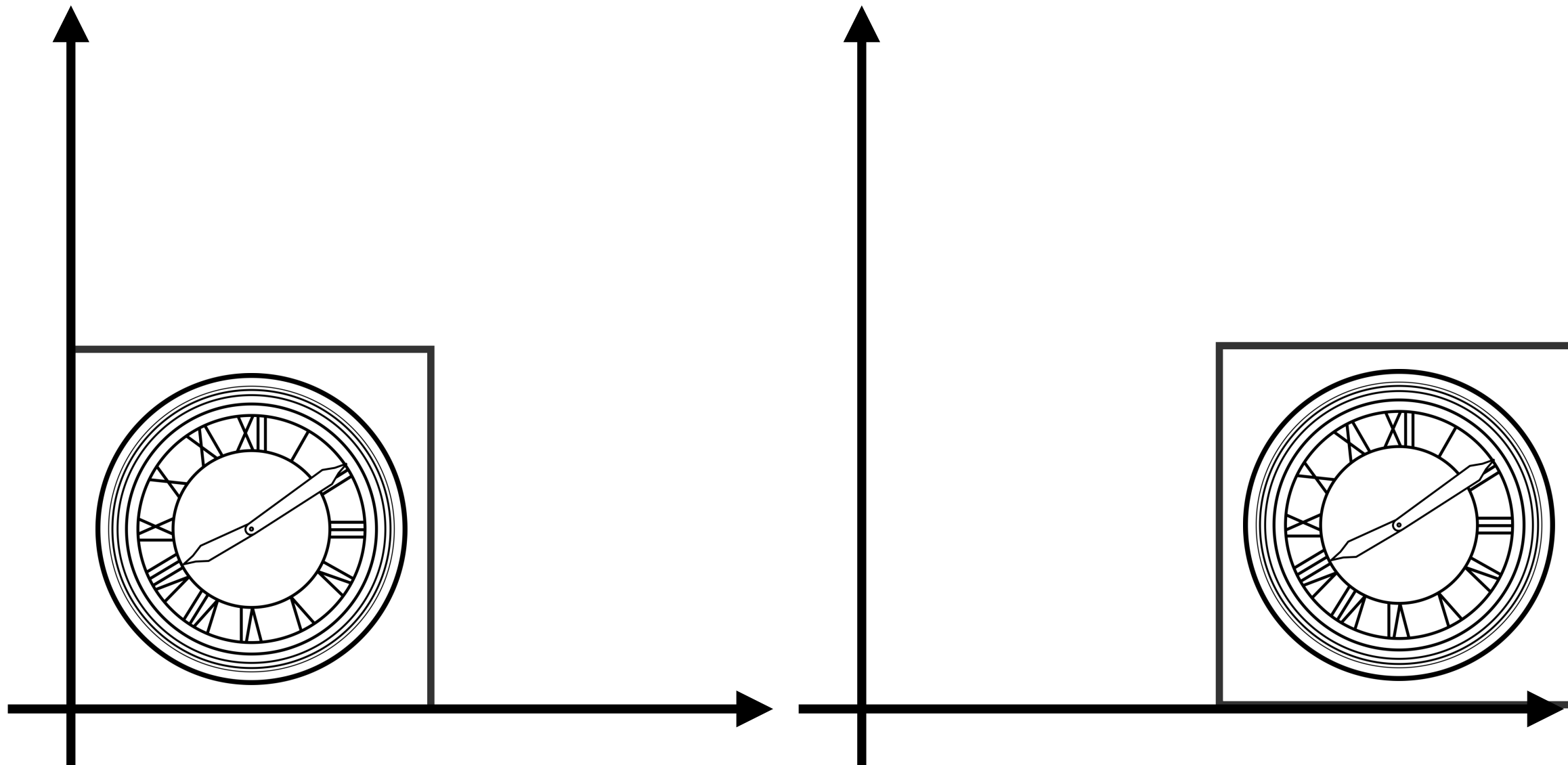
Composite Transform



Translate Then Rotate?

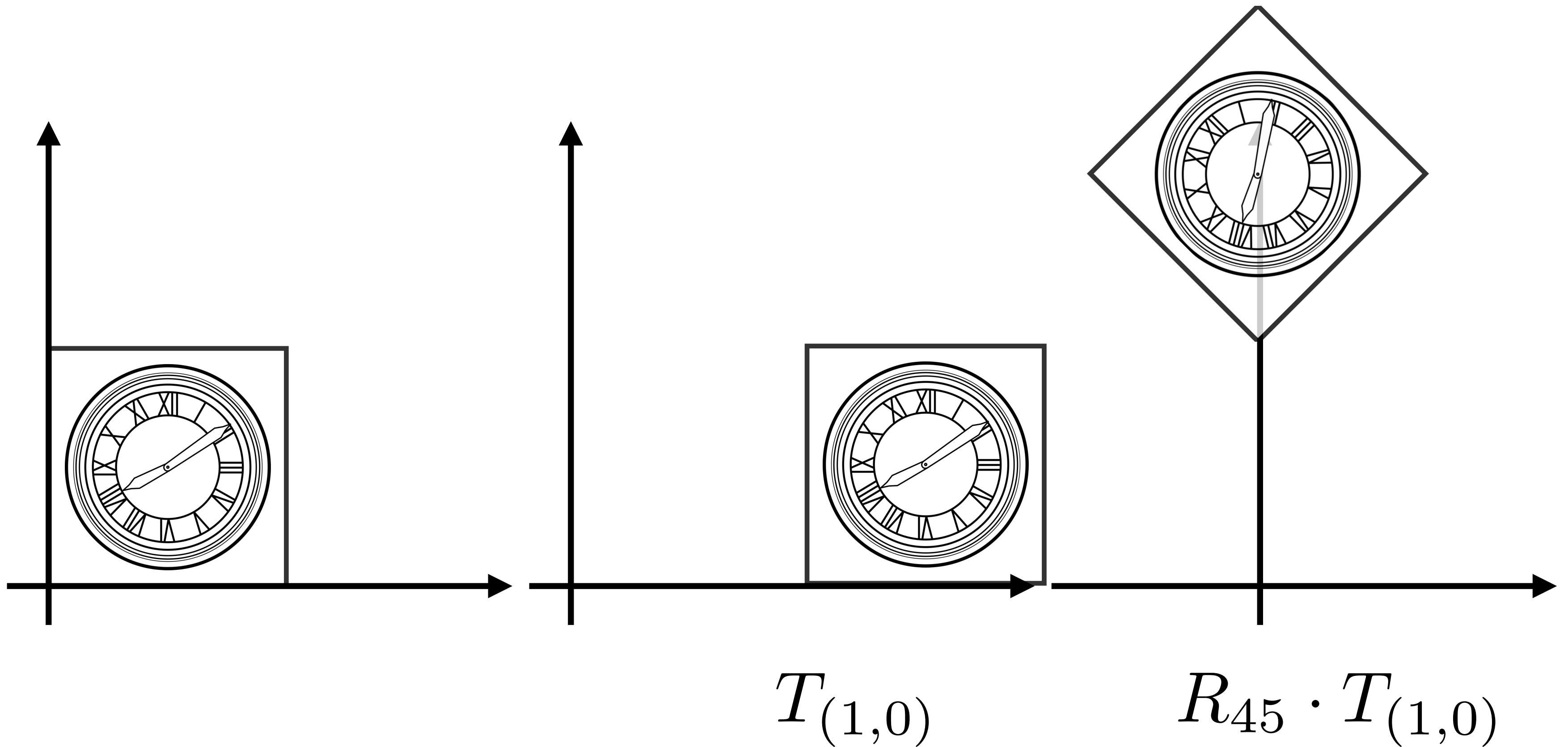


Translate Then Rotate?

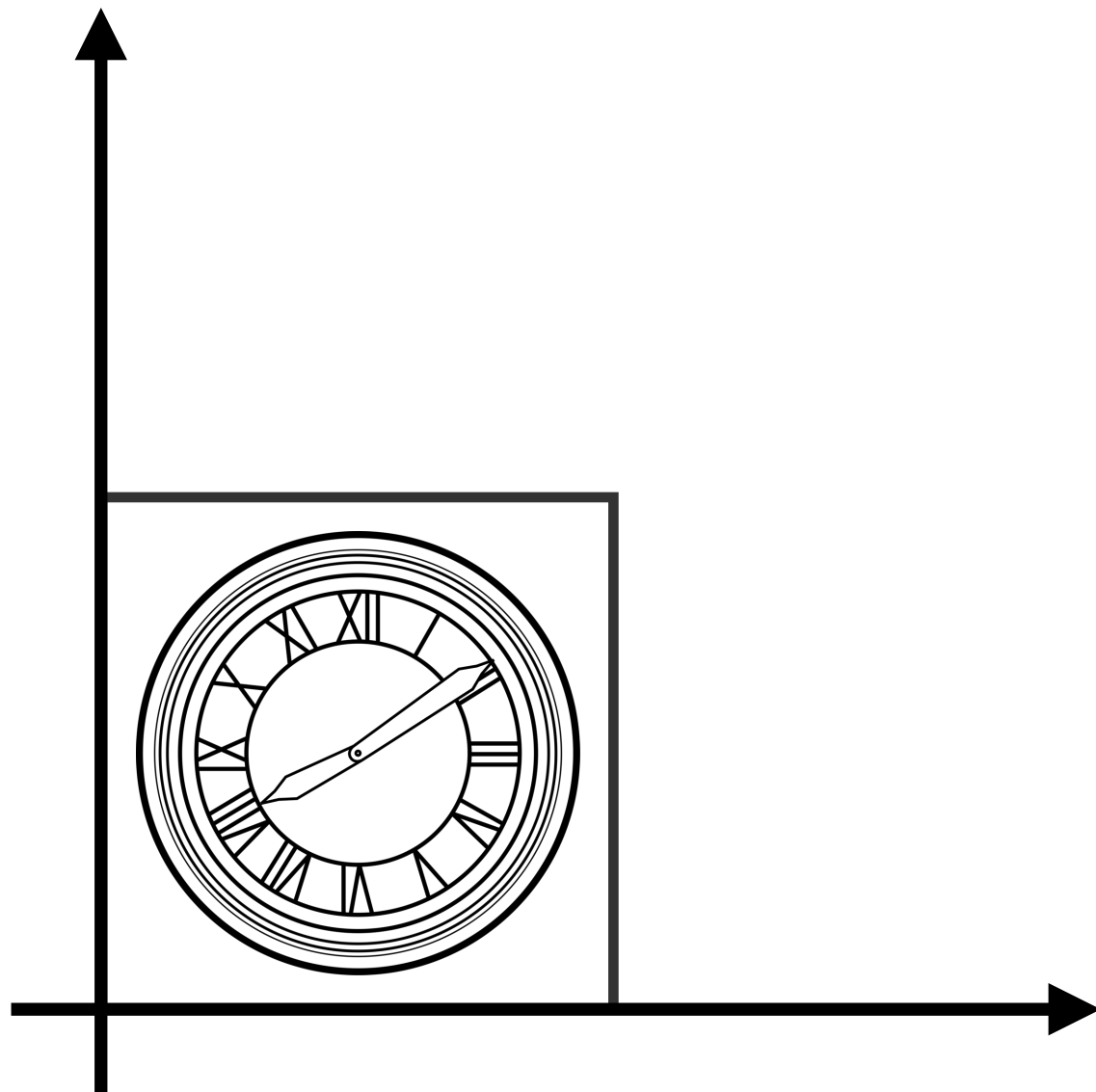


$$T_{(1,0)}$$

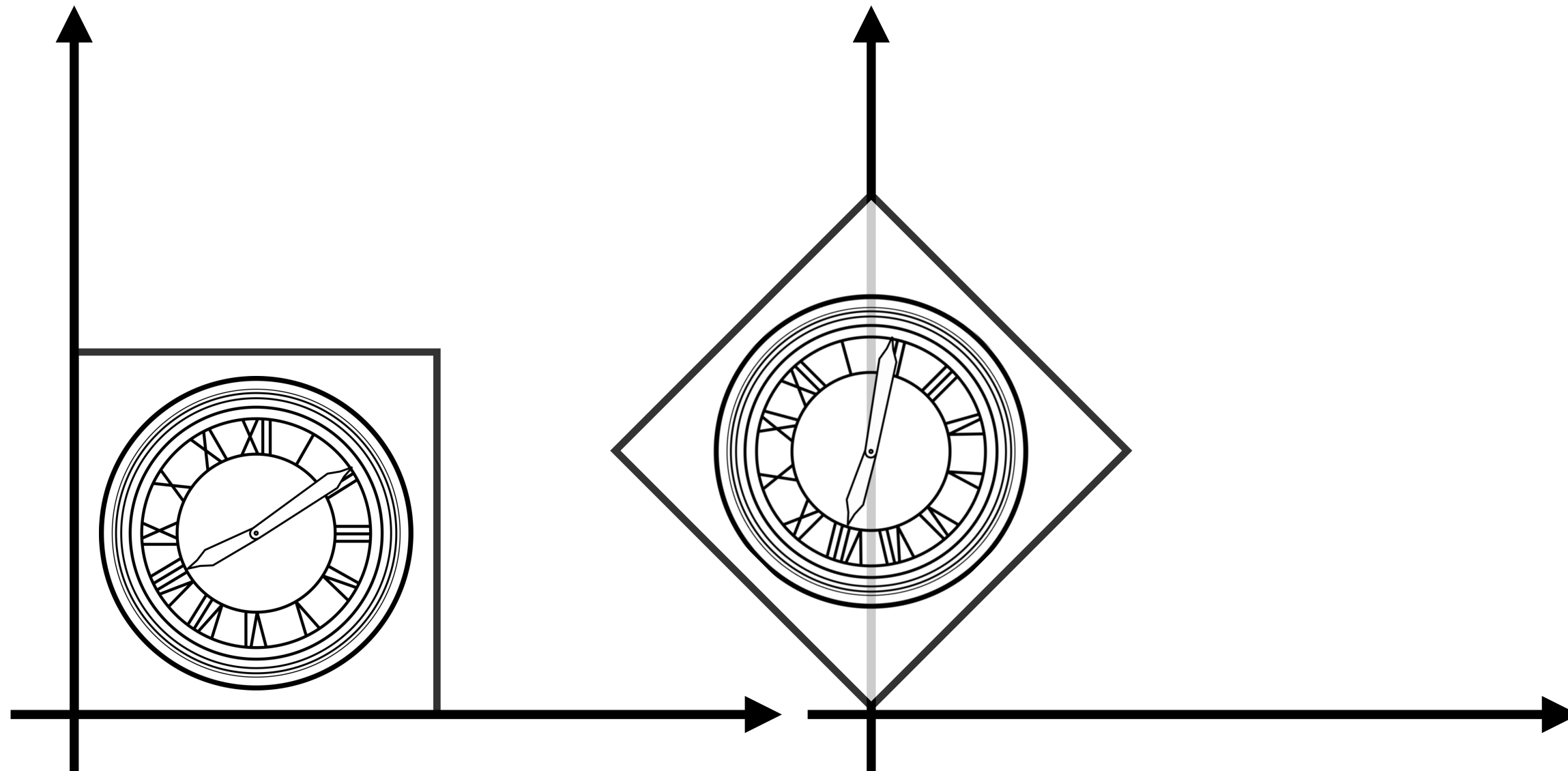
Translate Then Rotate?



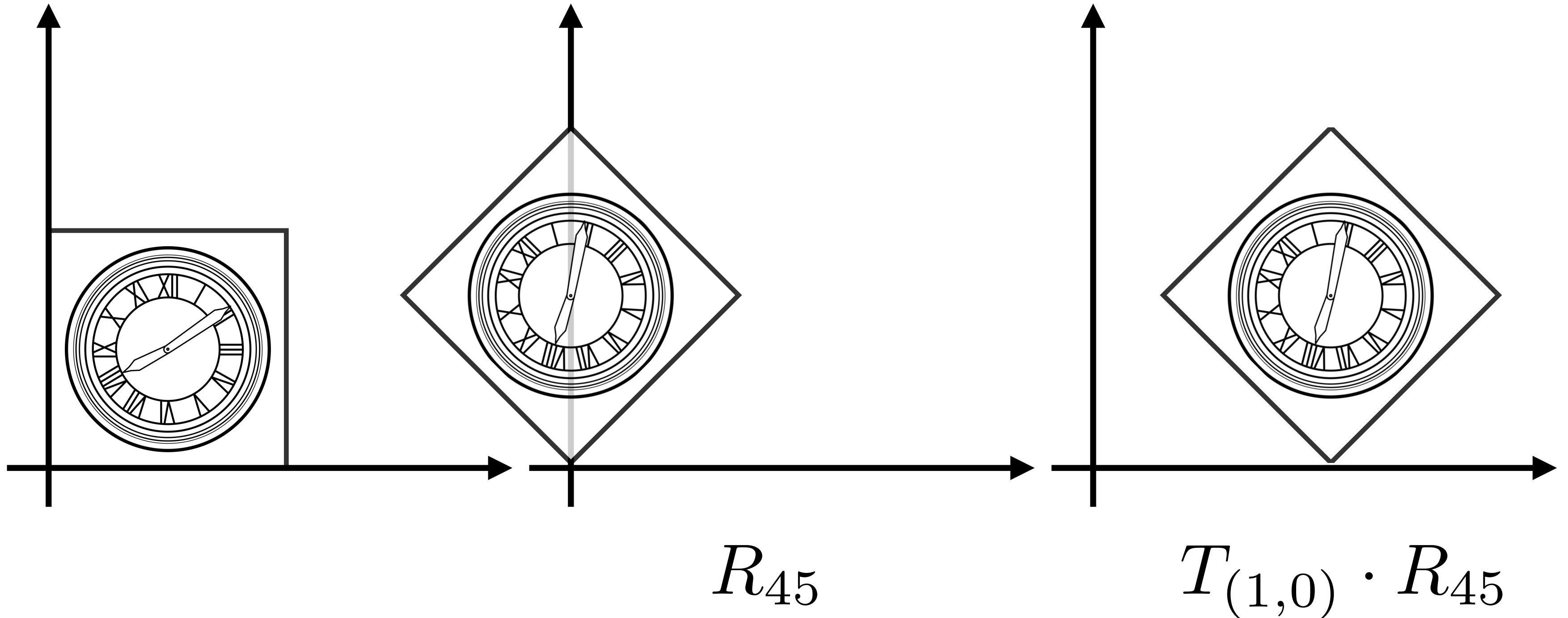
Rotate Then Translate



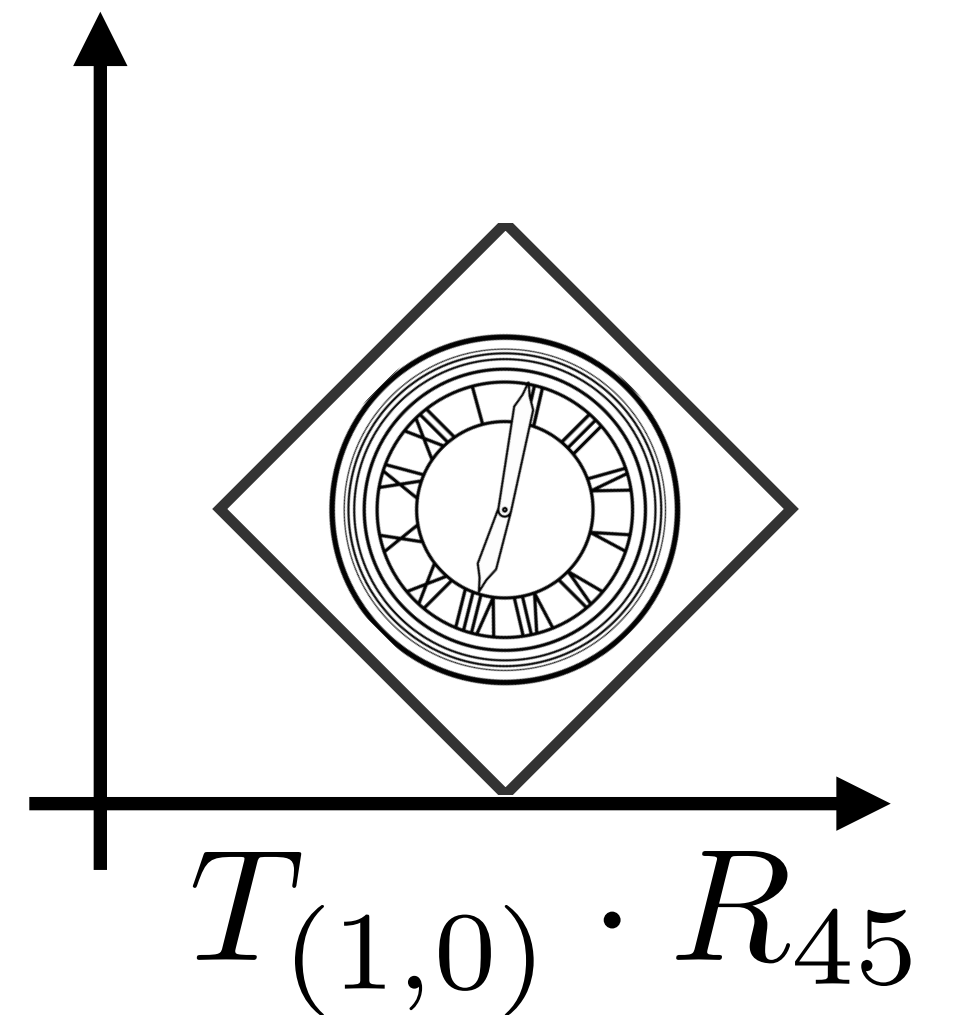
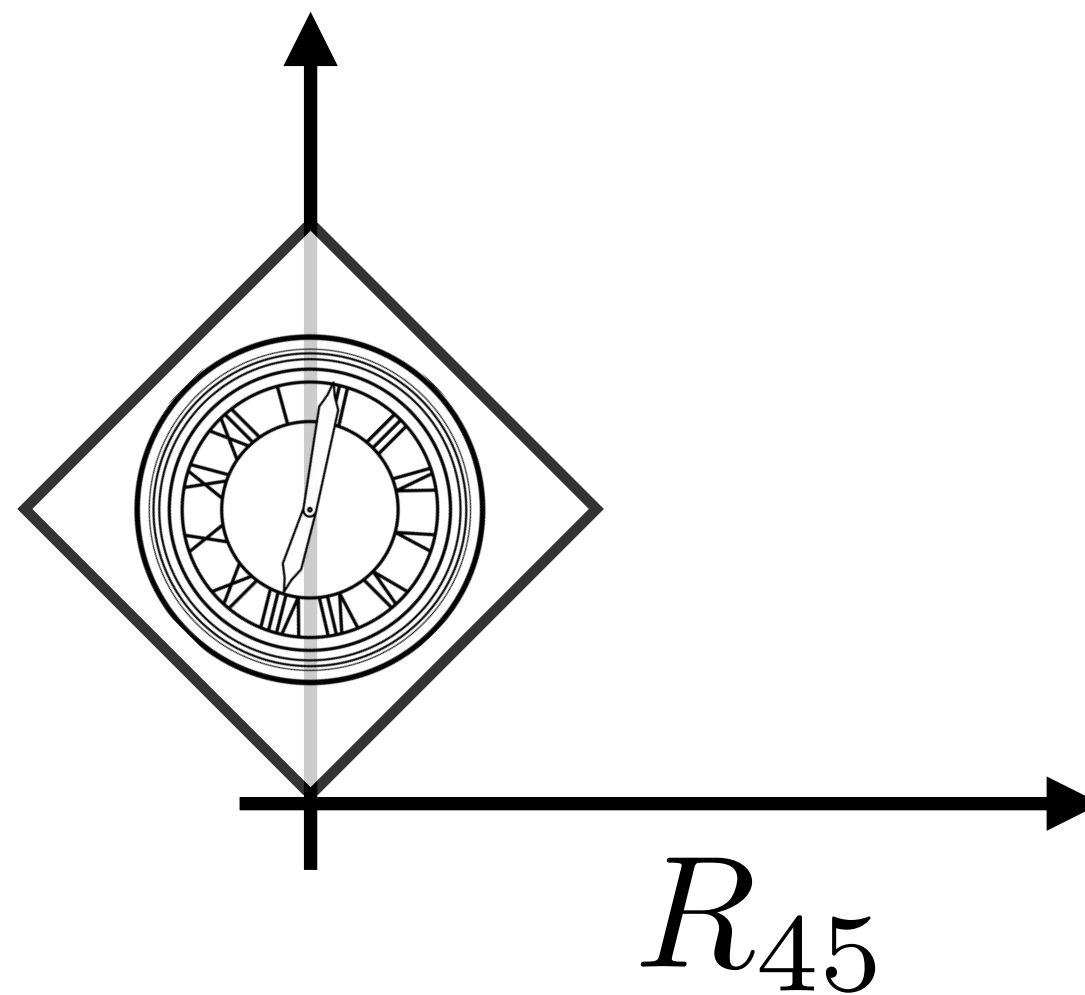
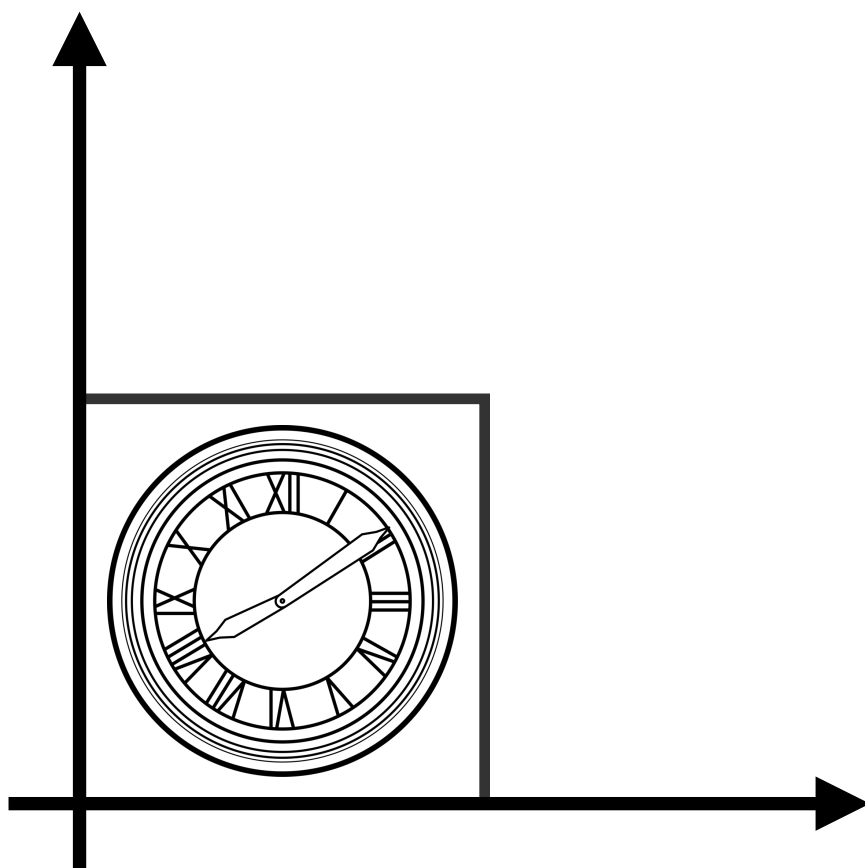
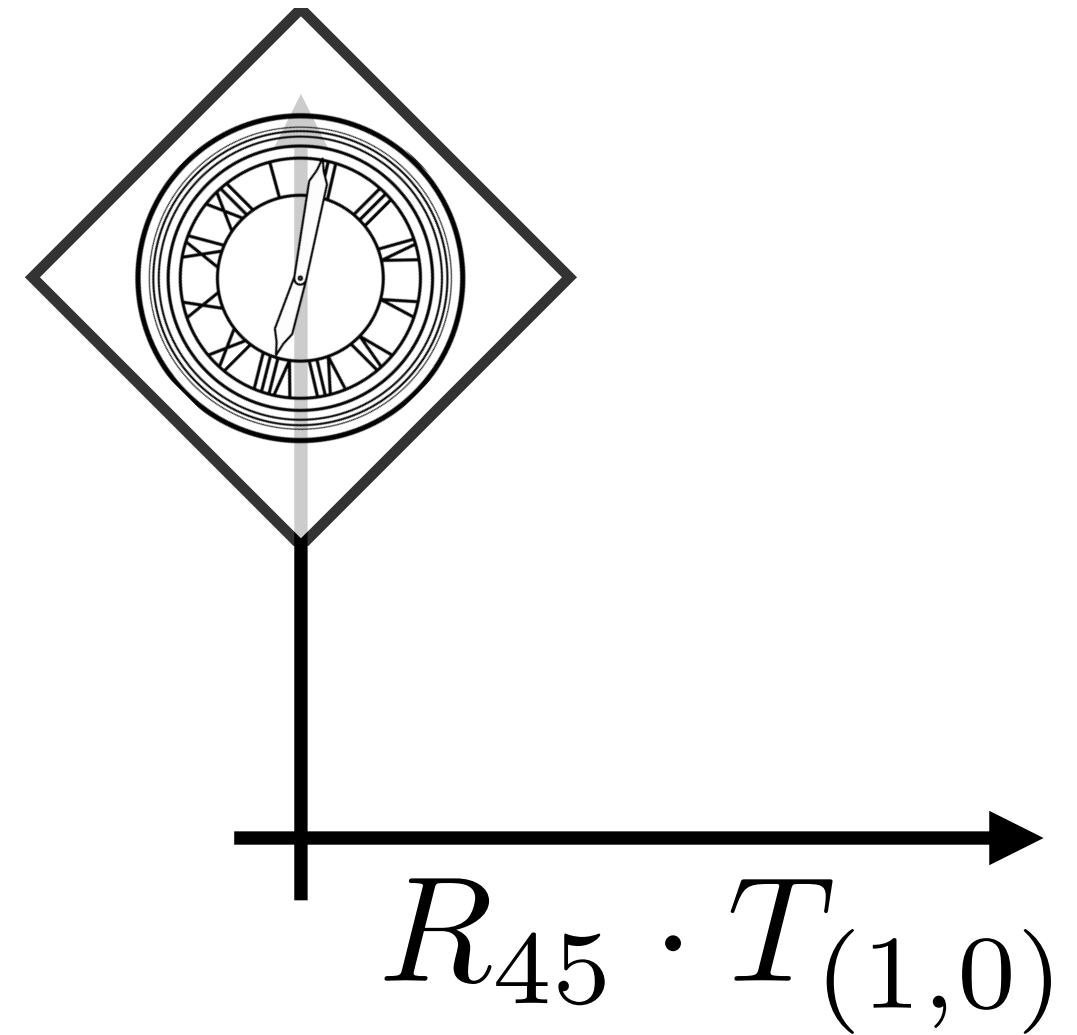
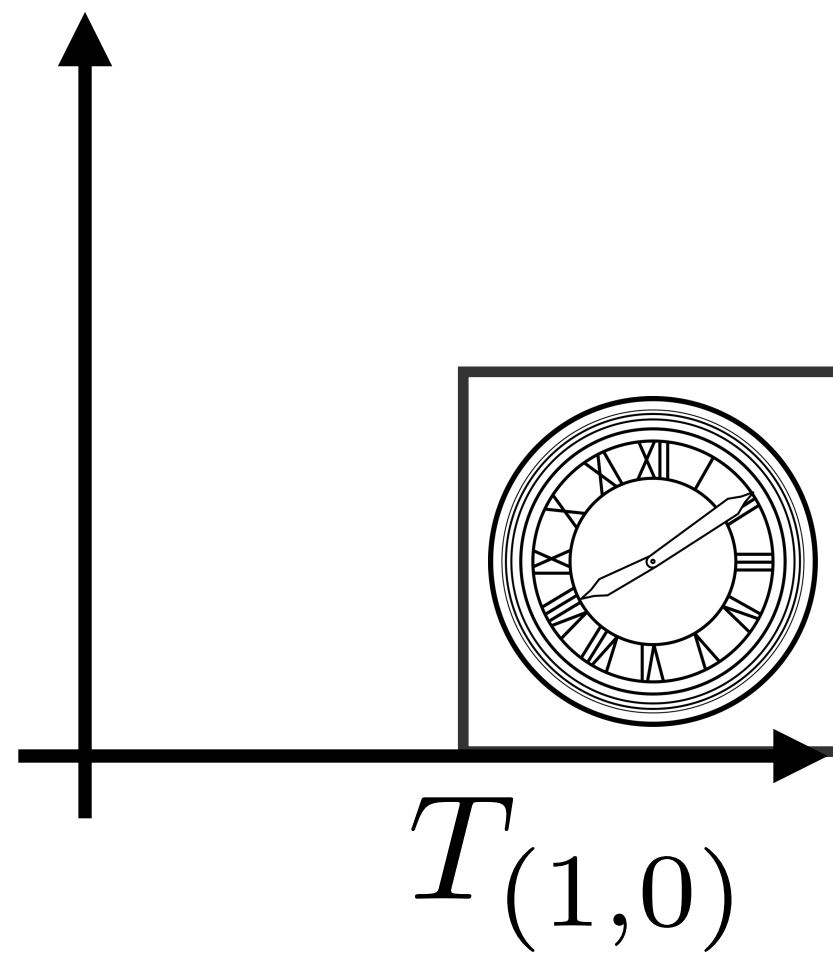
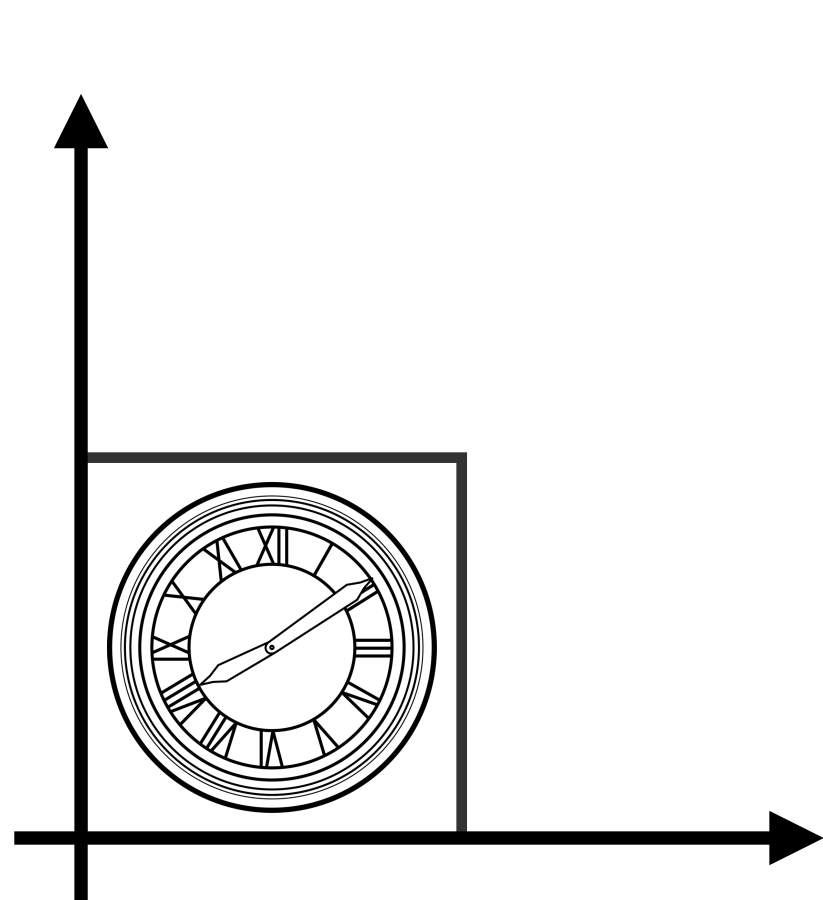
Rotate Then Translate



Rotate Then Translate



Transform Ordering Matters!



Transform Ordering Matters!

Matrix multiplication is not commutative

$$R_{45} \cdot T_{(1,0)} \neq T_{(1,0)} \cdot R_{45}$$

Transform Ordering Matters!

Matrix multiplication is not commutative

$$R_{45} \cdot T_{(1,0)} \neq T_{(1,0)} \cdot R_{45}$$

Recall the matrix math represented by these symbols:

$$\begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \neq \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transform Ordering Matters!

Matrix multiplication is not commutative

$$R_{45} \cdot T_{(1,0)} \neq T_{(1,0)} \cdot R_{45}$$

Recall the matrix math represented by these symbols:

$$\begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \neq \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that matrices are applied right to left:

$$T_{(1,0)} \cdot R_{45} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Composing Transforms

Sequence of transforms $A_1, A_2, A_3, \dots$

- Compose by matrix multiplication
 - Very important for performance!
 - Very important for complexity!

Composing Transforms

Sequence of transforms $A_1, A_2, A_3, \dots$

- Compose by matrix multiplication
 - Very important for performance!
 - Very important for complexity!

$$A_n(\dots A_2(A_1(\mathbf{x}))) = \mathbf{A}_n \cdots \mathbf{A}_2 \cdot \mathbf{A}_1 \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Composing Transforms

Sequence of transforms $A_1, A_2, A_3, \dots$

- Compose by matrix multiplication
 - Very important for performance!
 - Very important for complexity!

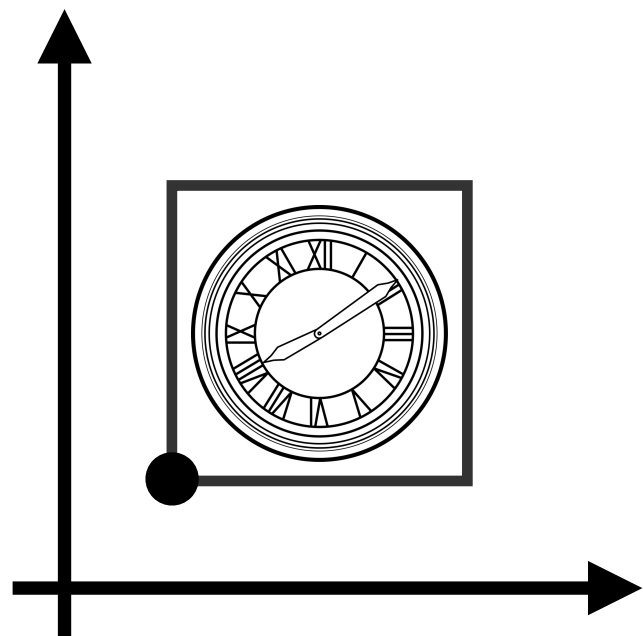
$$A_n(\dots A_2(A_1(\mathbf{x}))) = \underbrace{\mathbf{A}_n \cdots \mathbf{A}_2 \cdot \mathbf{A}_1}_{\text{Pre-multiply } n \text{ matrices to obtain a single matrix representing combined transform}} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Pre-multiply n matrices to obtain a single matrix representing combined transform

Building Complex Transforms

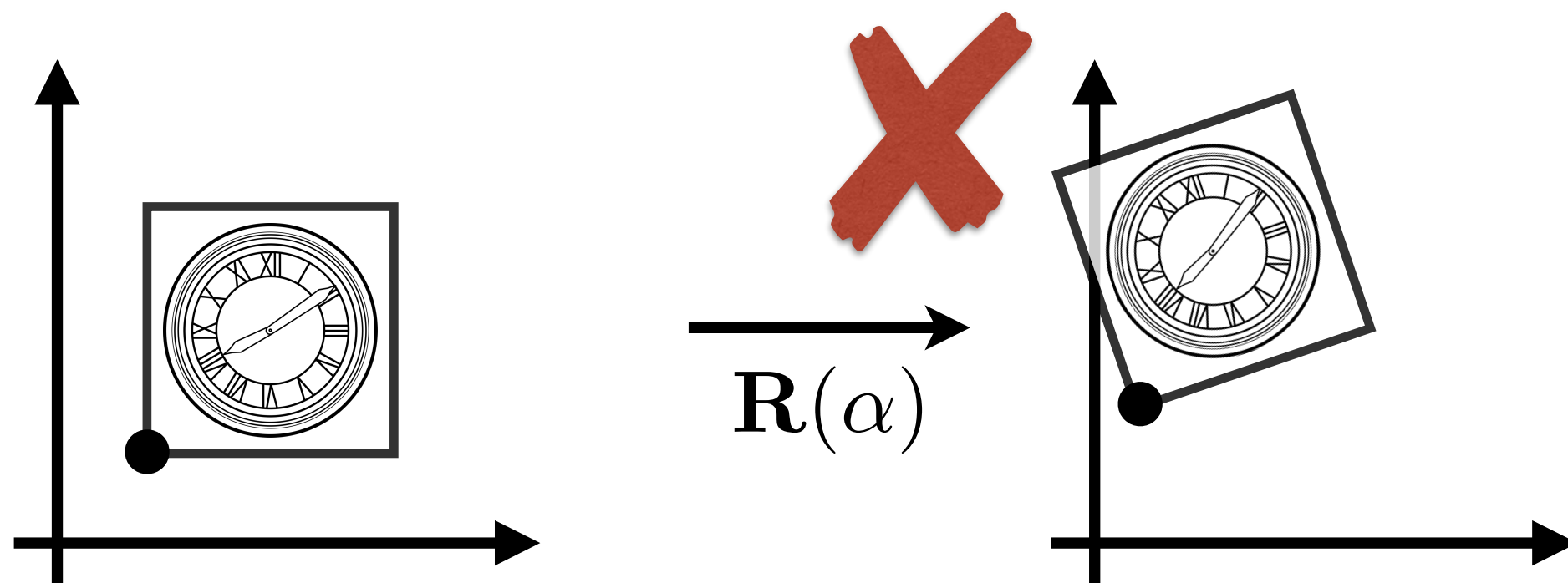
Building Complex Transforms

How to rotate around a given point c ?



Building Complex Transforms

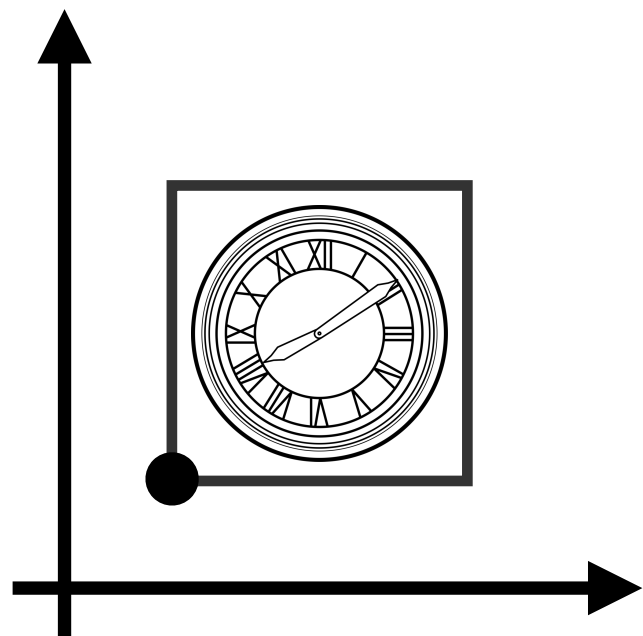
How to rotate around a given point c ?



Building Complex Transforms

Building Complex Transforms

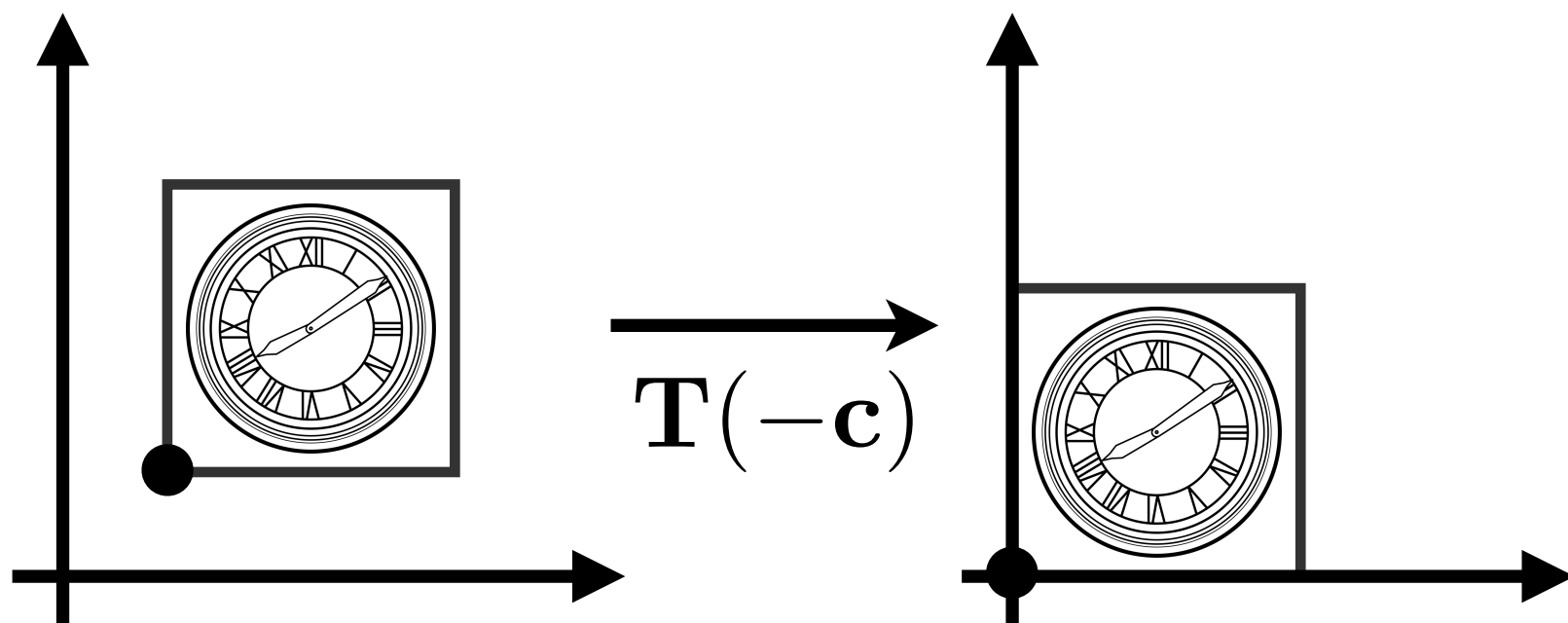
How to rotate around a given point c ?



Building Complex Transforms

How to rotate around a given point c ?

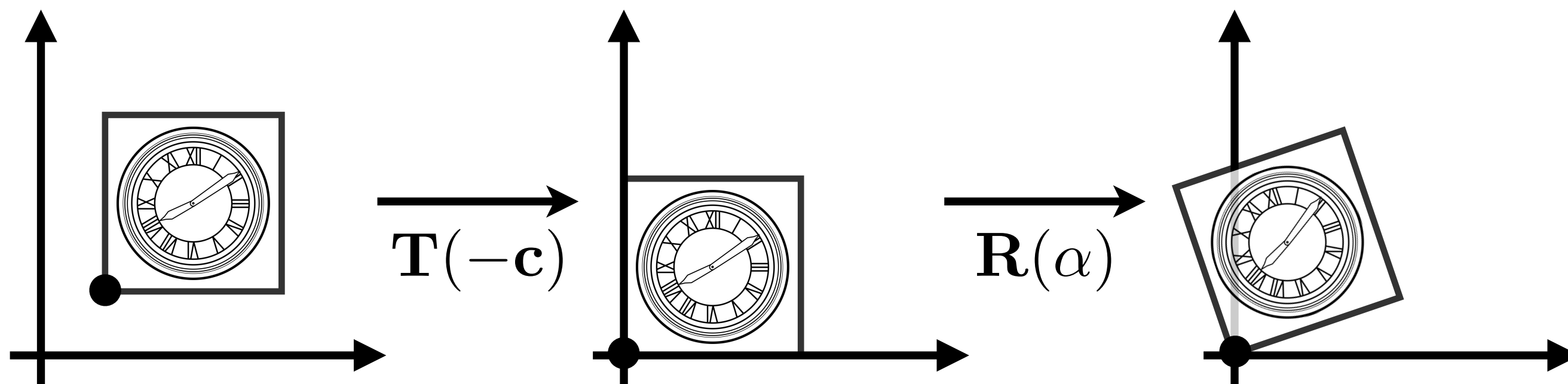
1. Translate center to origin



Building Complex Transforms

How to rotate around a given point c ?

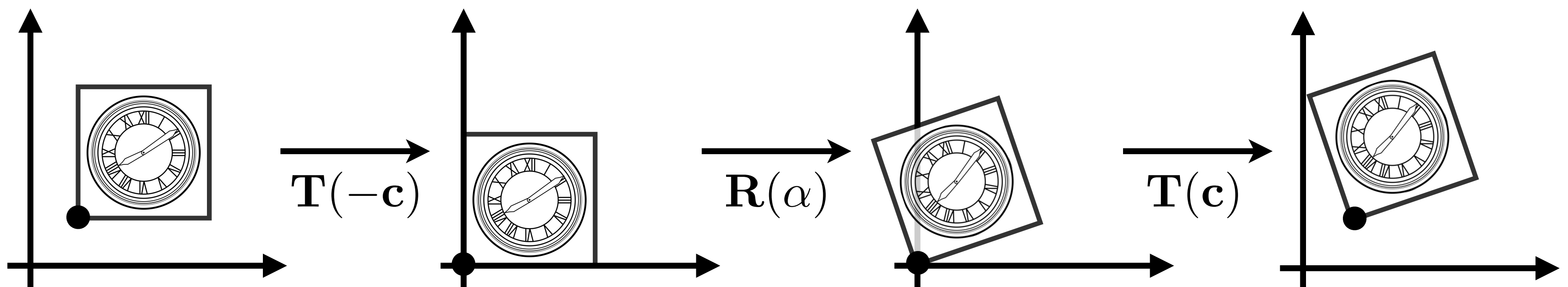
1. Translate center to origin
2. Rotate



Building Complex Transforms

How to rotate around a given point c ?

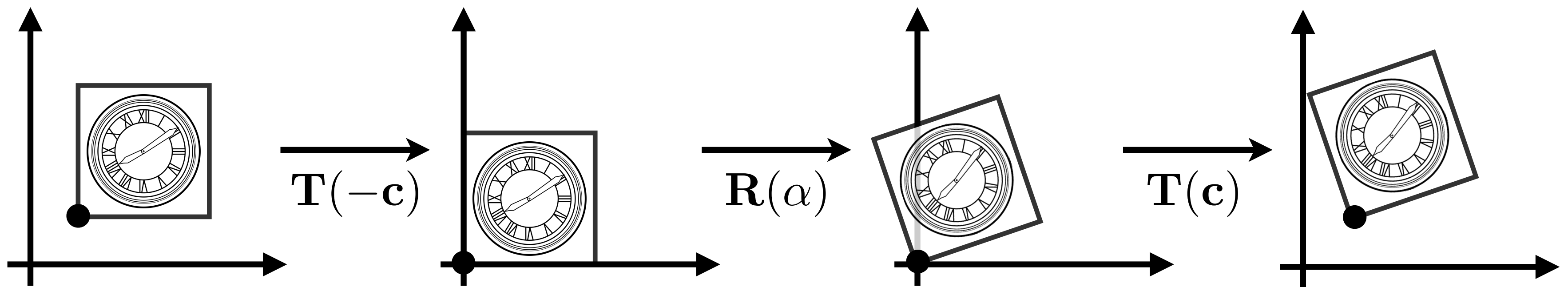
1. Translate center to origin
2. Rotate
3. Translate back



Building Complex Transforms

How to rotate around a given point c ?

1. Translate center to origin
2. Rotate
3. Translate back



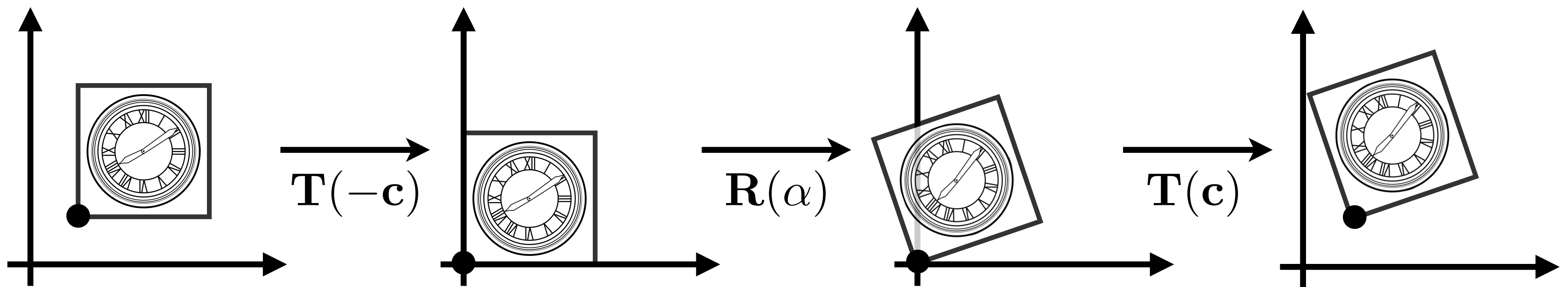
Matrix representation

$$T(c) \cdot R(\alpha) \cdot T(-c)$$

Building Complex Transforms

How to rotate around a given point $\mathbf{c}$?

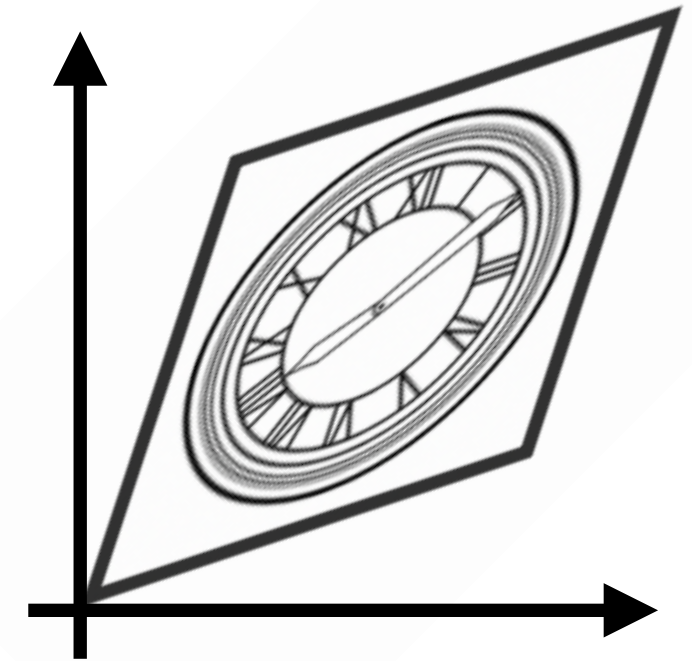
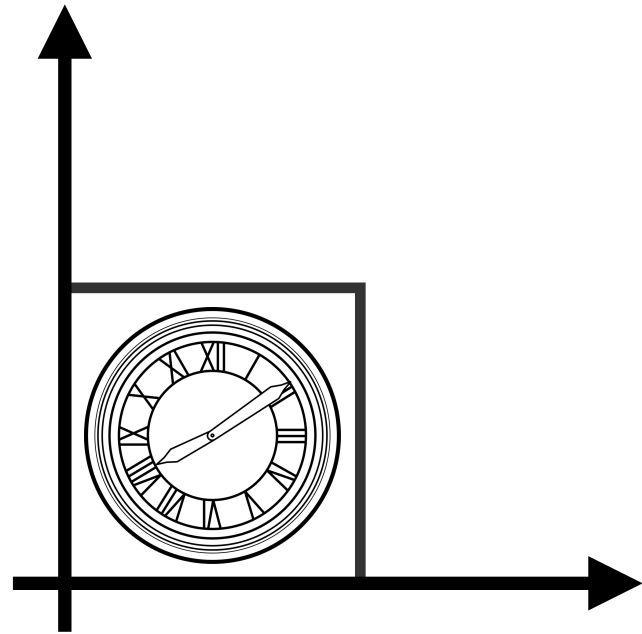
1. Translate center to origin
2. Rotate
3. Translate back



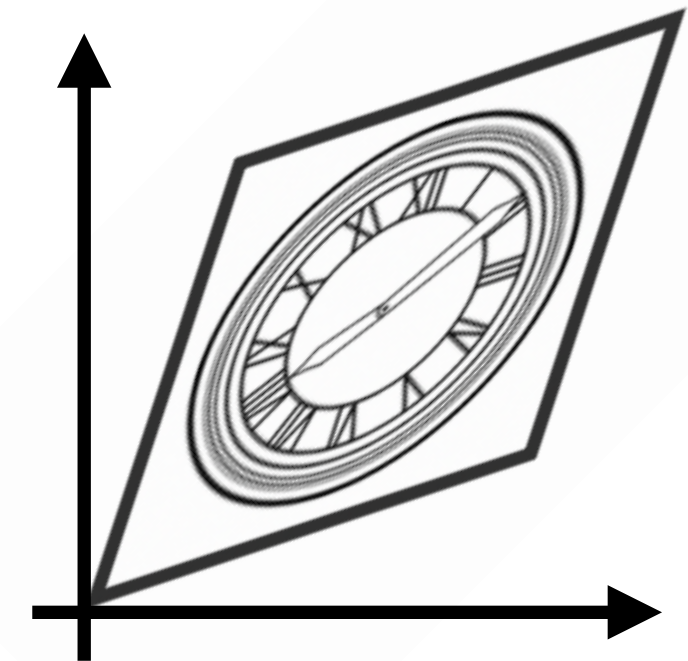
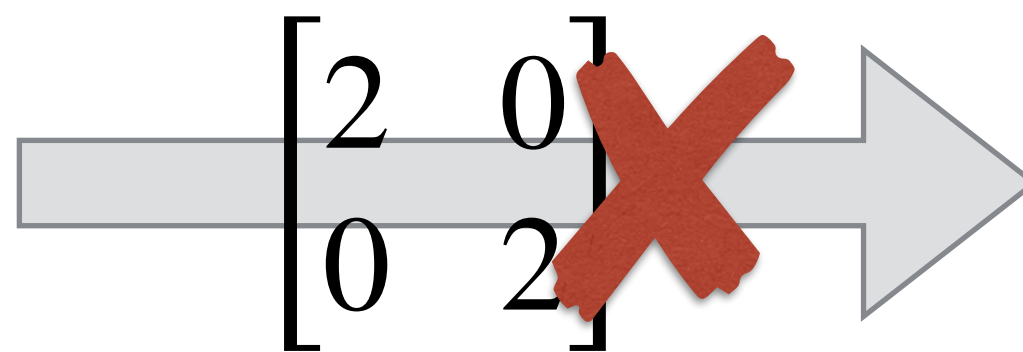
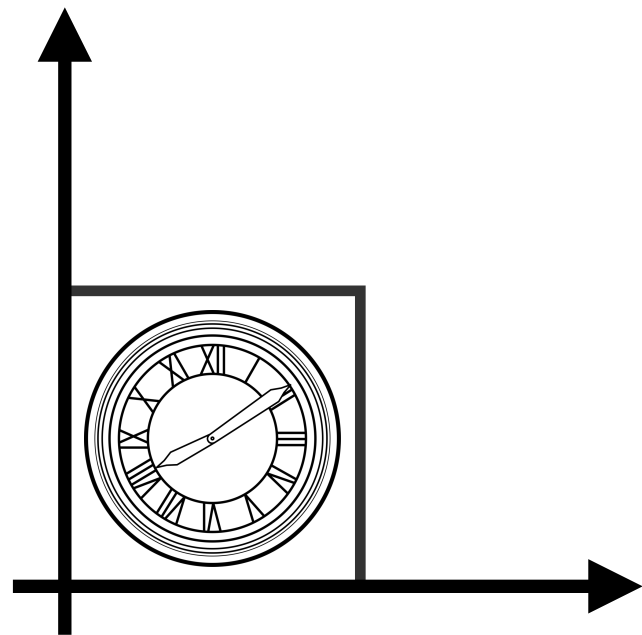
Matrix representation

$$\mathbf{M}_{\text{composite}} = \mathbf{T}(\mathbf{c}) \cdot \mathbf{R}(\alpha) \cdot \mathbf{T}(-\mathbf{c})$$

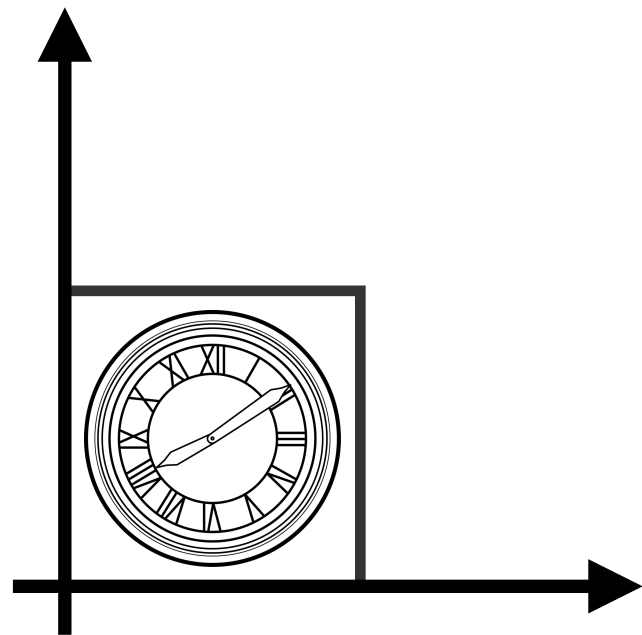
Building Not-Axis-Aligned Scale Matrix



Building Not-Axis-Aligned Scale Matrix

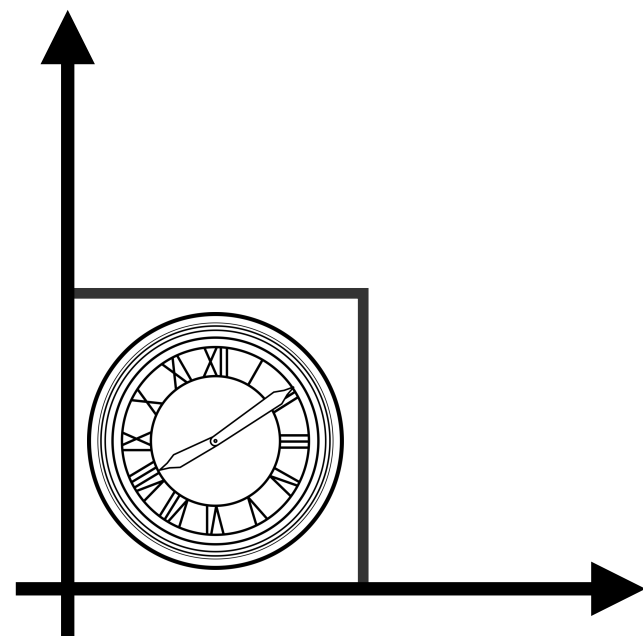
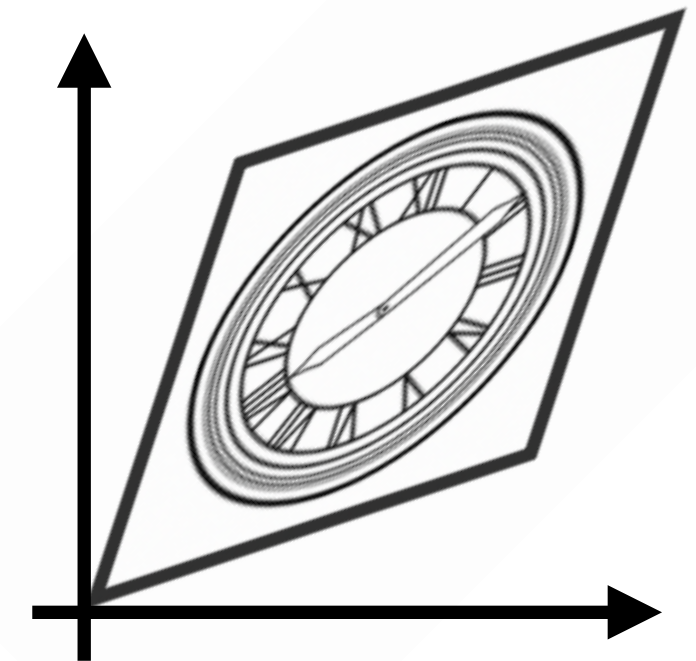


Building Not-Axis-Aligned Scale Matrix

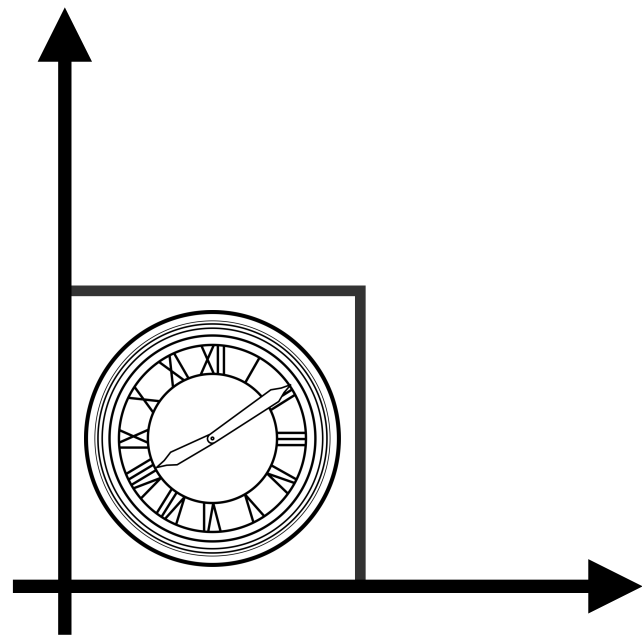


$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

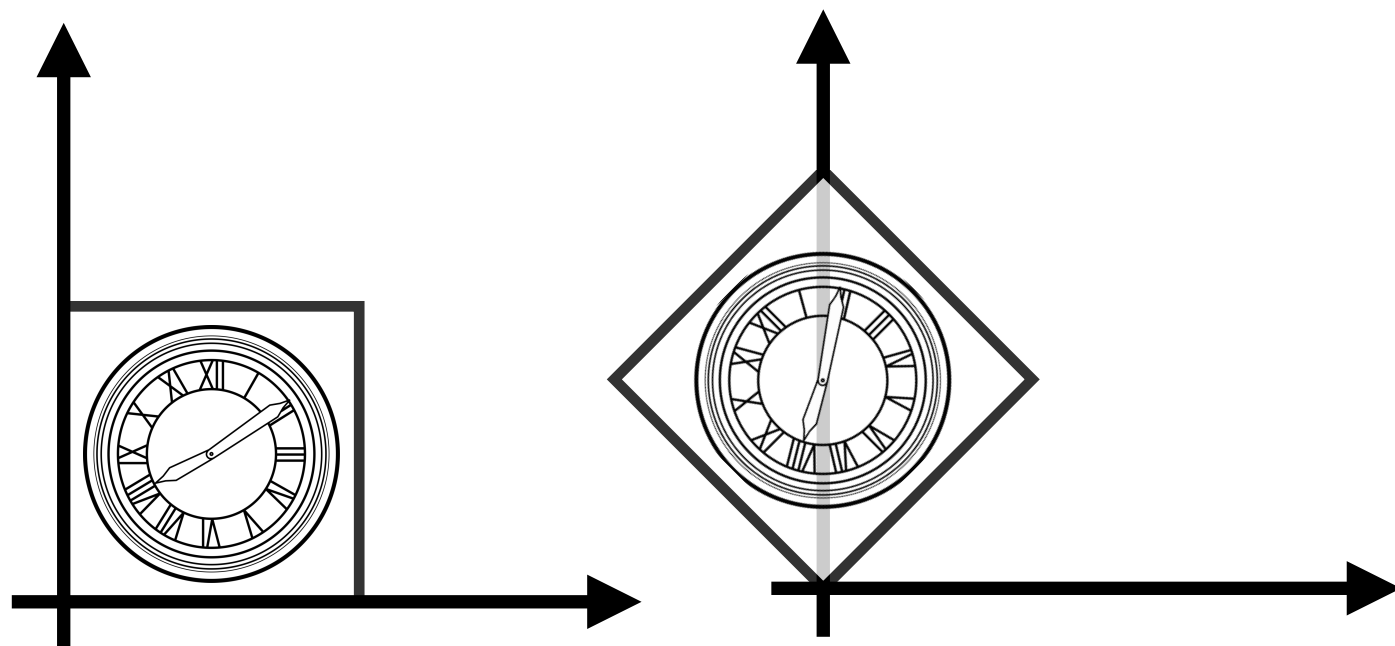
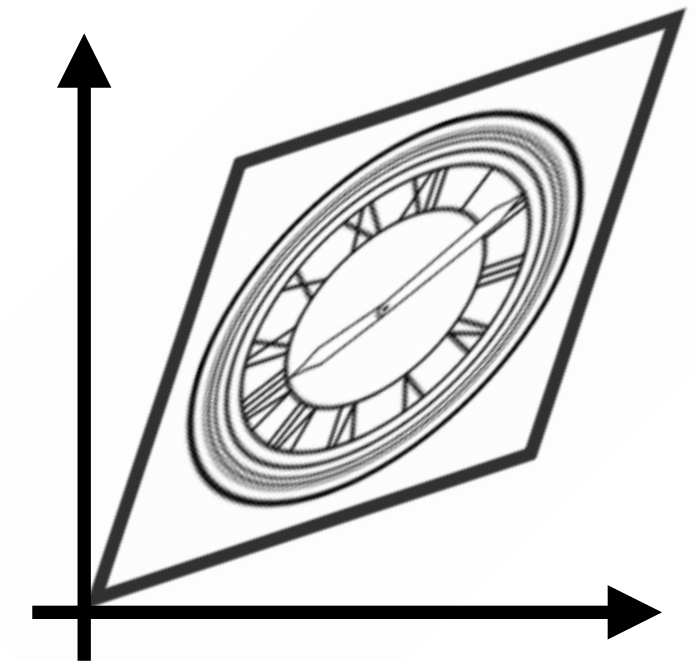
A large red 'X' is drawn over the matrix, indicating it is incorrect for non-axis-aligned scaling. A gray arrow points from the matrix towards the right.



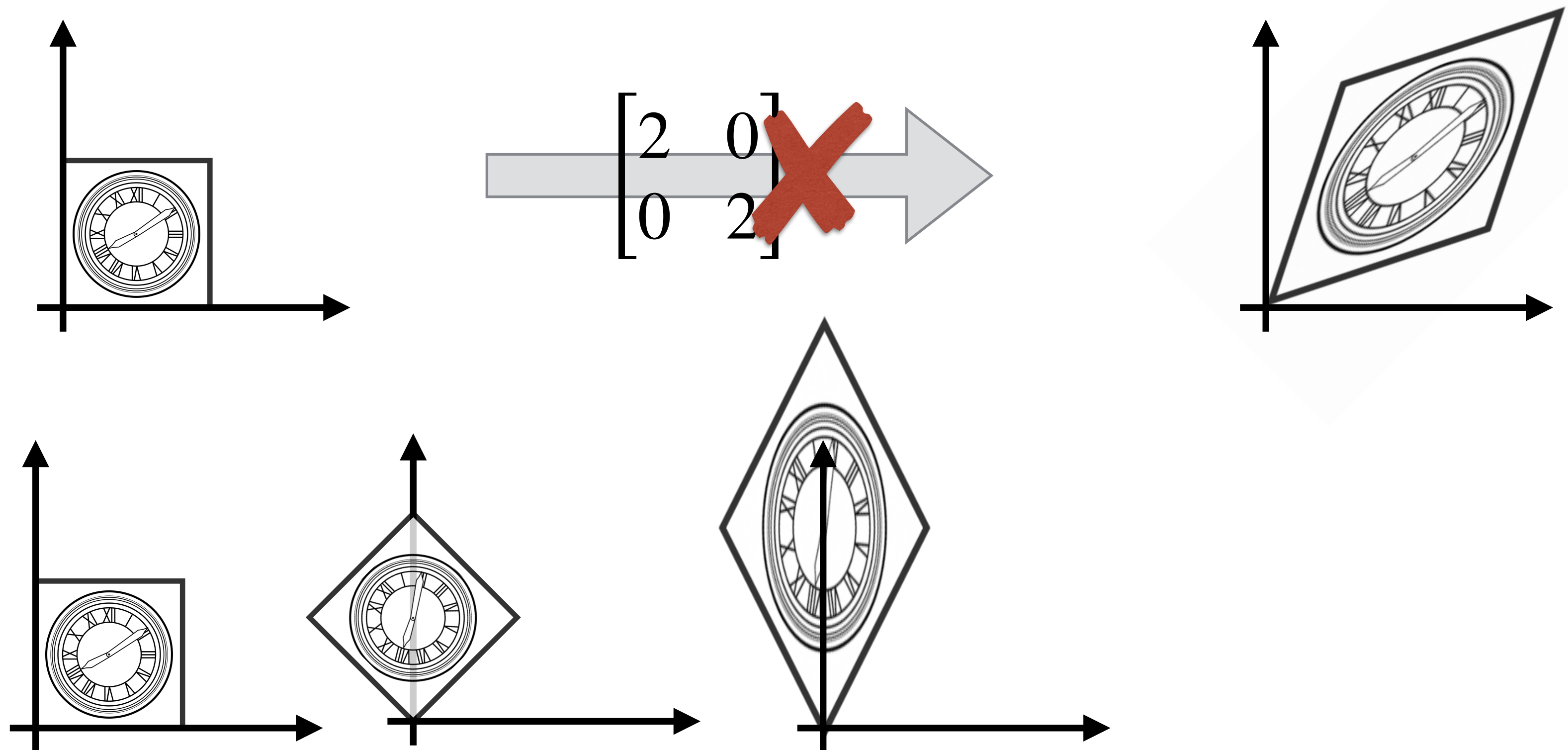
Building Not-Axis-Aligned Scale Matrix



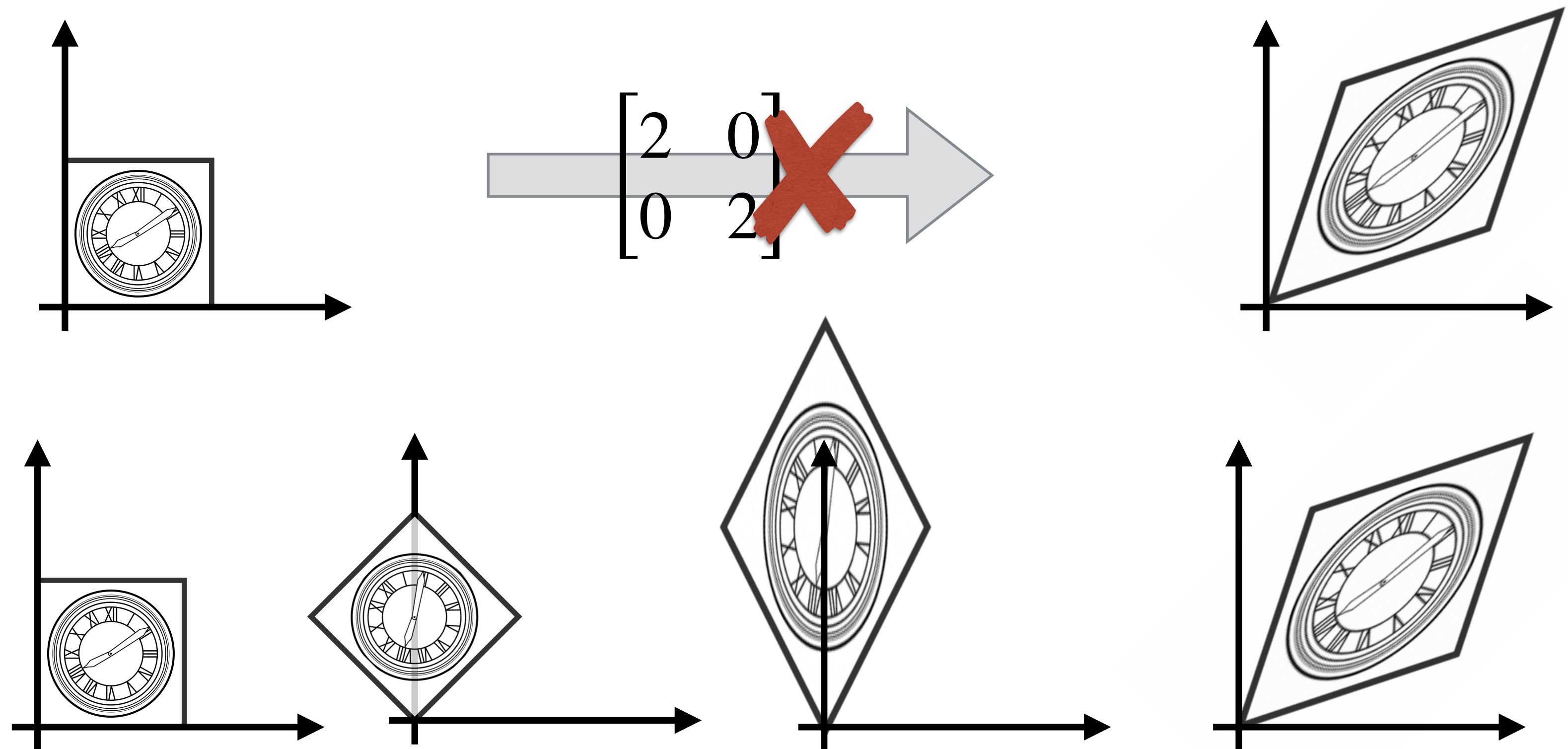
→ $\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ ✗ →



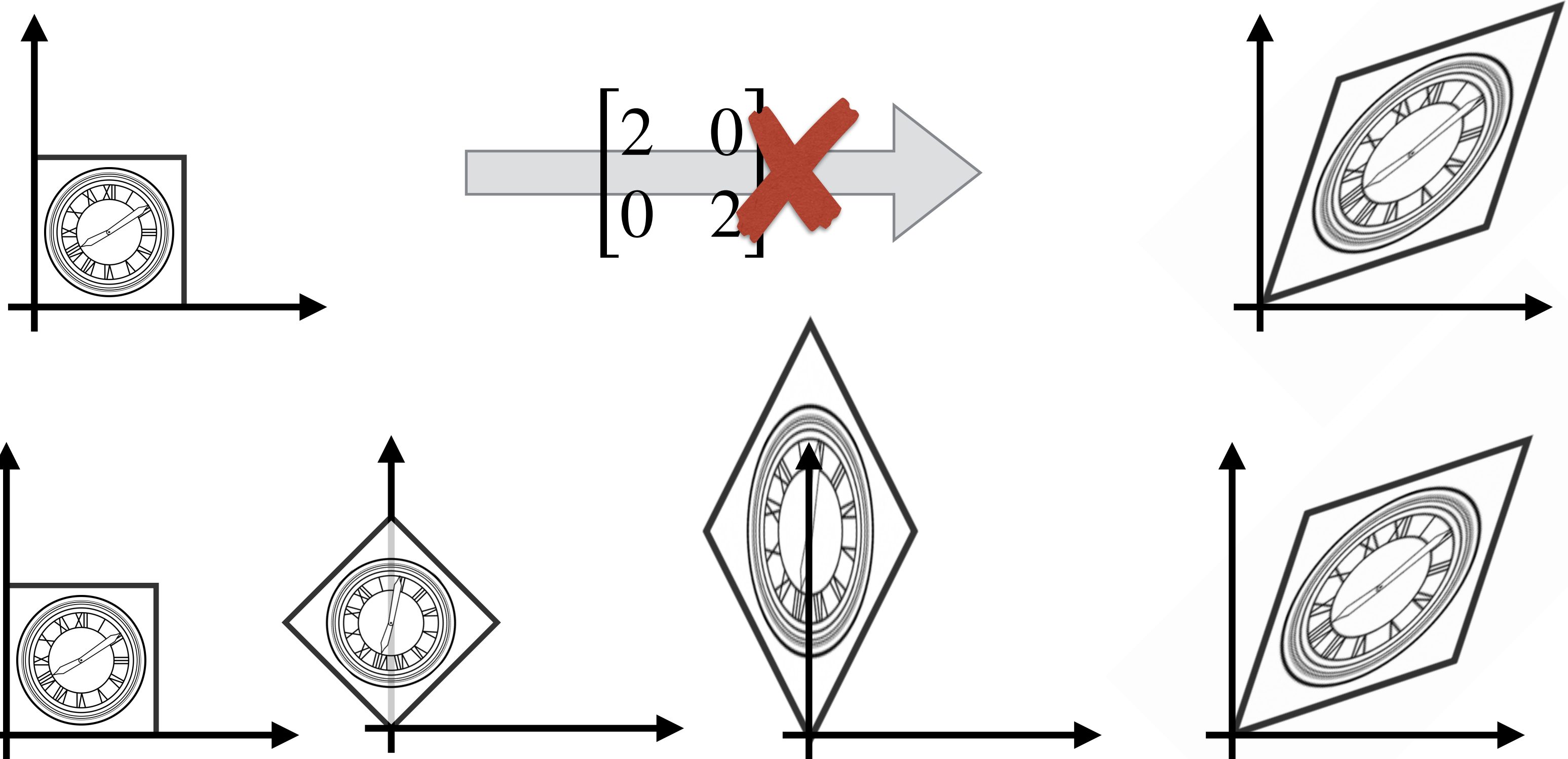
Building Not-Axis-Aligned Scale Matrix



Building Not-Axis-Aligned Scale Matrix



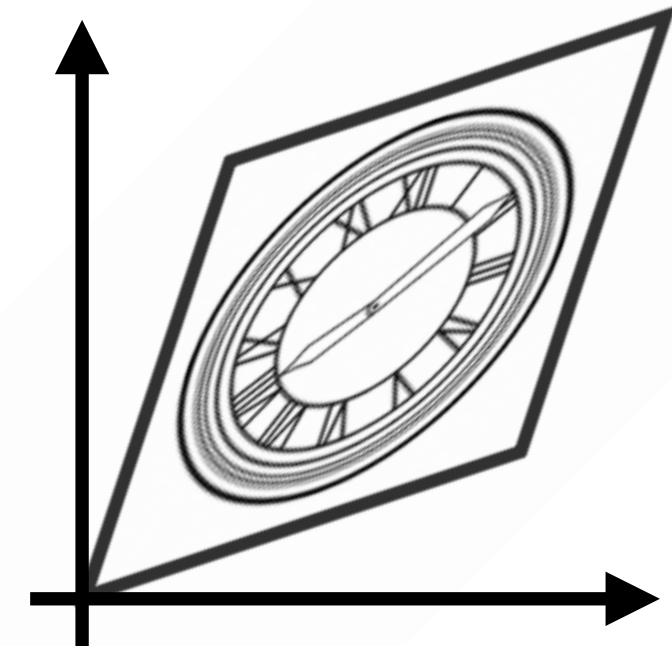
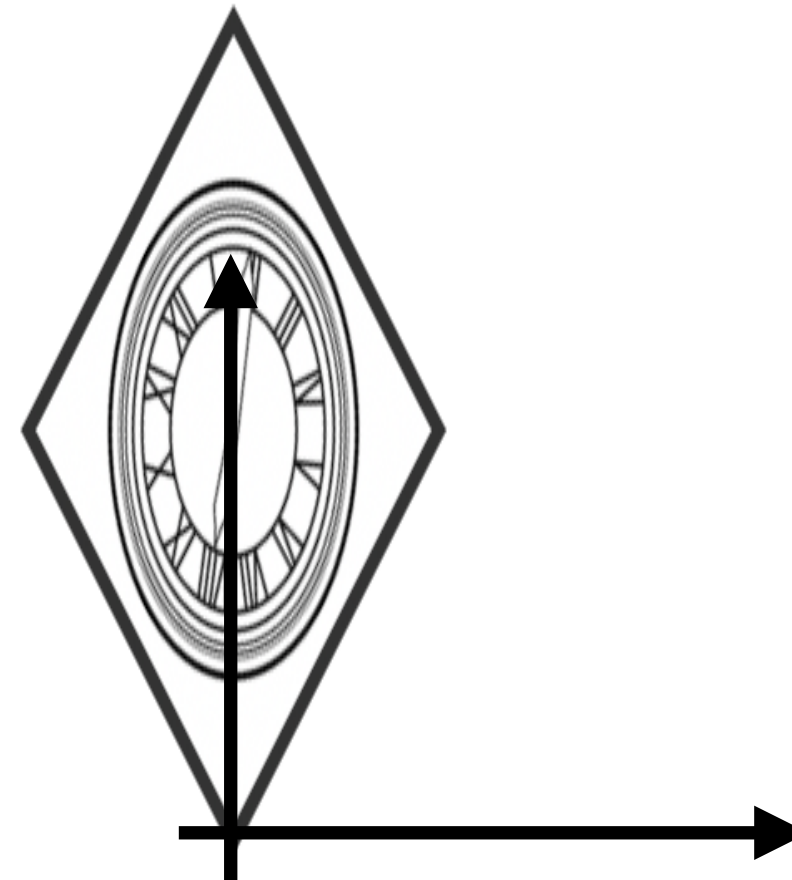
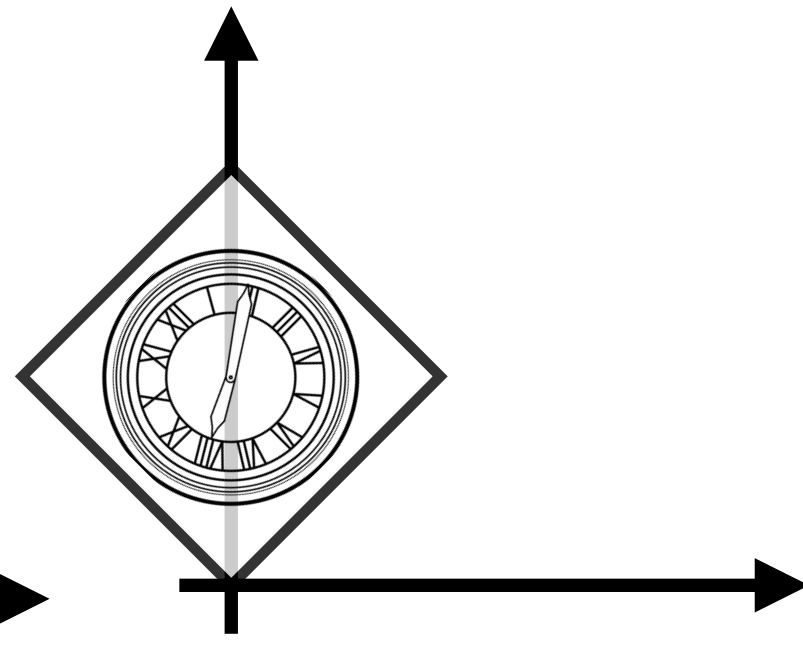
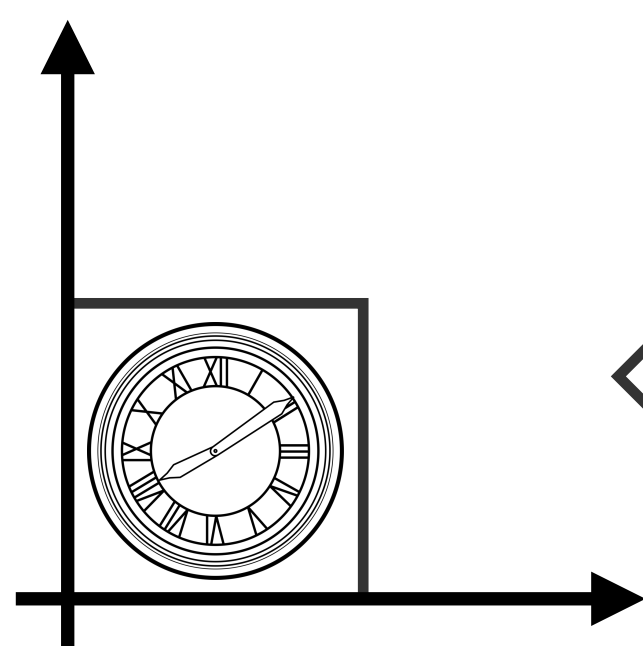
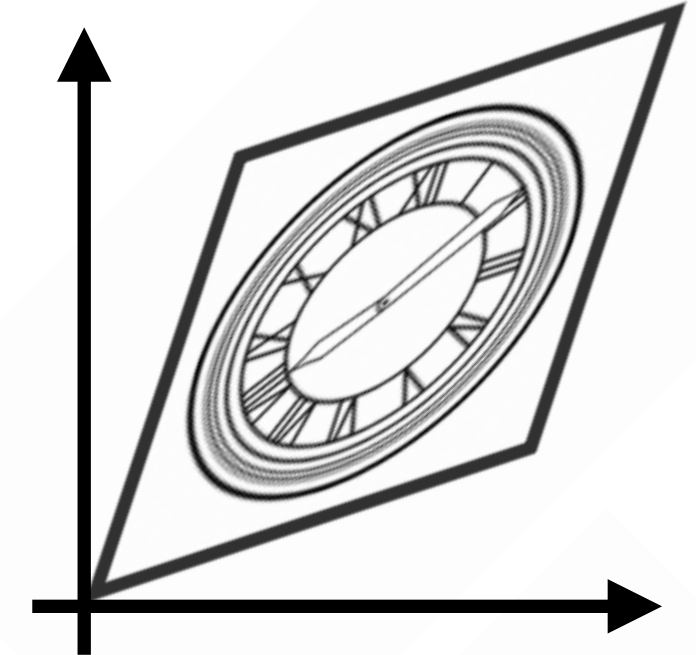
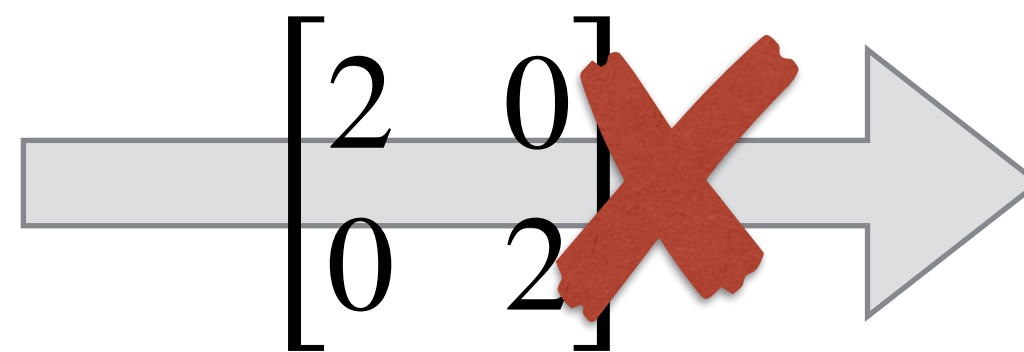
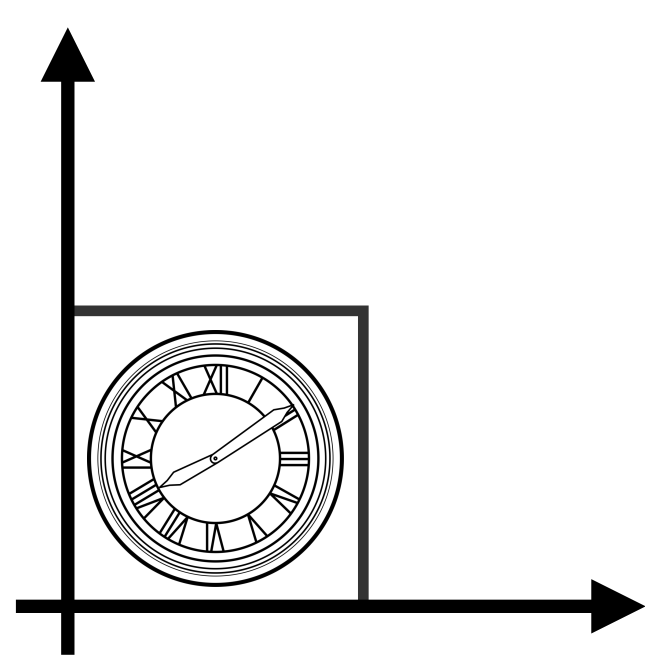
Building Not-Axis-Aligned Scale Matrix



$$\begin{bmatrix} \cos(45) & -\sin(45) \\ \sin(45) & \cos(45) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} \cos(-45) & -\sin(-45) \\ \sin(-45) & \cos(-45) \end{bmatrix} = \begin{bmatrix} \frac{3}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{3}{2} \end{bmatrix}$$

$R(45)$ $s(1,2)$ $R(-45)$

Building Not-Axis-Aligned Scale Matrix



$$\begin{bmatrix} \cos(45) & -\sin(45) \\ \sin(45) & \cos(45) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} \cos(-45) & -\sin(-45) \\ \sin(-45) & \cos(-45) \end{bmatrix} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 1.5 \end{bmatrix}$$

R(45)

s(1,2)

R(-45)

Useful Matrix Decompositions

Eigen system

Lambda is diagonal

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$$

Singular Value Decomposition (SVD)

S is diagonal

Q and R are orthonormal

$$\mathbf{A} = \mathbf{Q}\mathbf{S}\mathbf{R}^T$$

Polar Decomposition

S is diagonal

P, W and R are orthonormal

$$\mathbf{P}\mathbf{R} = \mathbf{Q}$$

$$\mathbf{A} = \mathbf{P}\mathbf{R}\mathbf{S}\mathbf{R}^T$$

Note: $\mathbf{V}=\mathbf{Q}=\mathbf{R}$ if $\mathbf{A}$ symmetric and real

Useful Matrix Decompositions

Eigen system

Lambda is diagonal

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$$

Singular Value Decomposition (SVD)

S is diagonal

Q and R are orthonormal

$$\mathbf{A} = \mathbf{Q}\mathbf{S}\mathbf{R}^T$$

Polar Decomposition

S is diagonal

P, W and R are orthonormal

$$\mathbf{P}\mathbf{R} = \mathbf{Q}$$

$$\mathbf{A} = \mathbf{P}\mathbf{R}\mathbf{S}\mathbf{R}^T$$

Note: $\mathbf{V}=\mathbf{Q}=\mathbf{R}$ if $\mathbf{A}$ symmetric and real

Rotation Matrix $\cong$ Orthonormal (det+1)

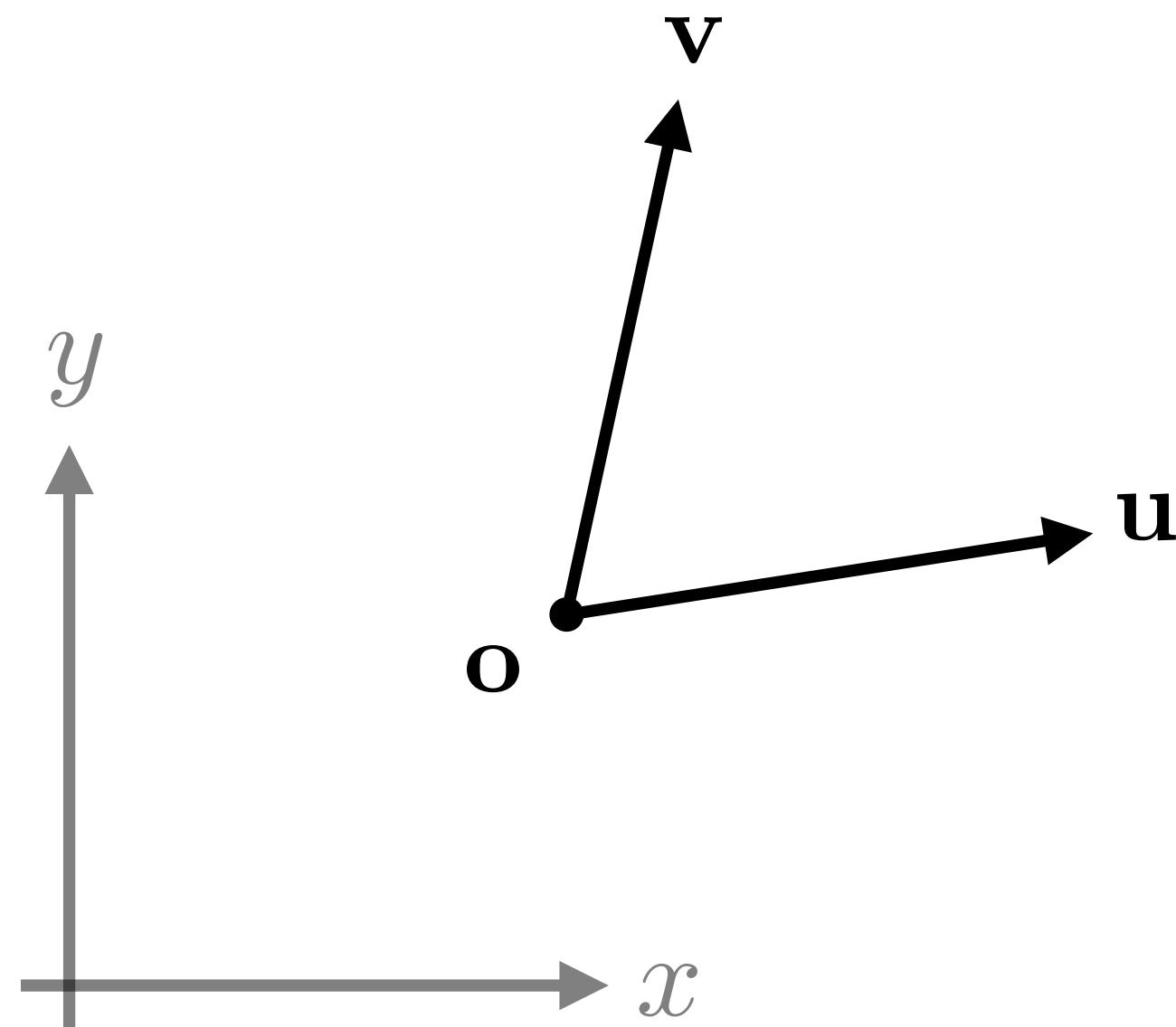
Axis-aligned Scale Matrix $\cong$ Diagonal

Scale Matrix $\cong$ OrthonormalTranspose . Diagonal . Orthonormal

Coordinate Systems

Coordinate System Transform

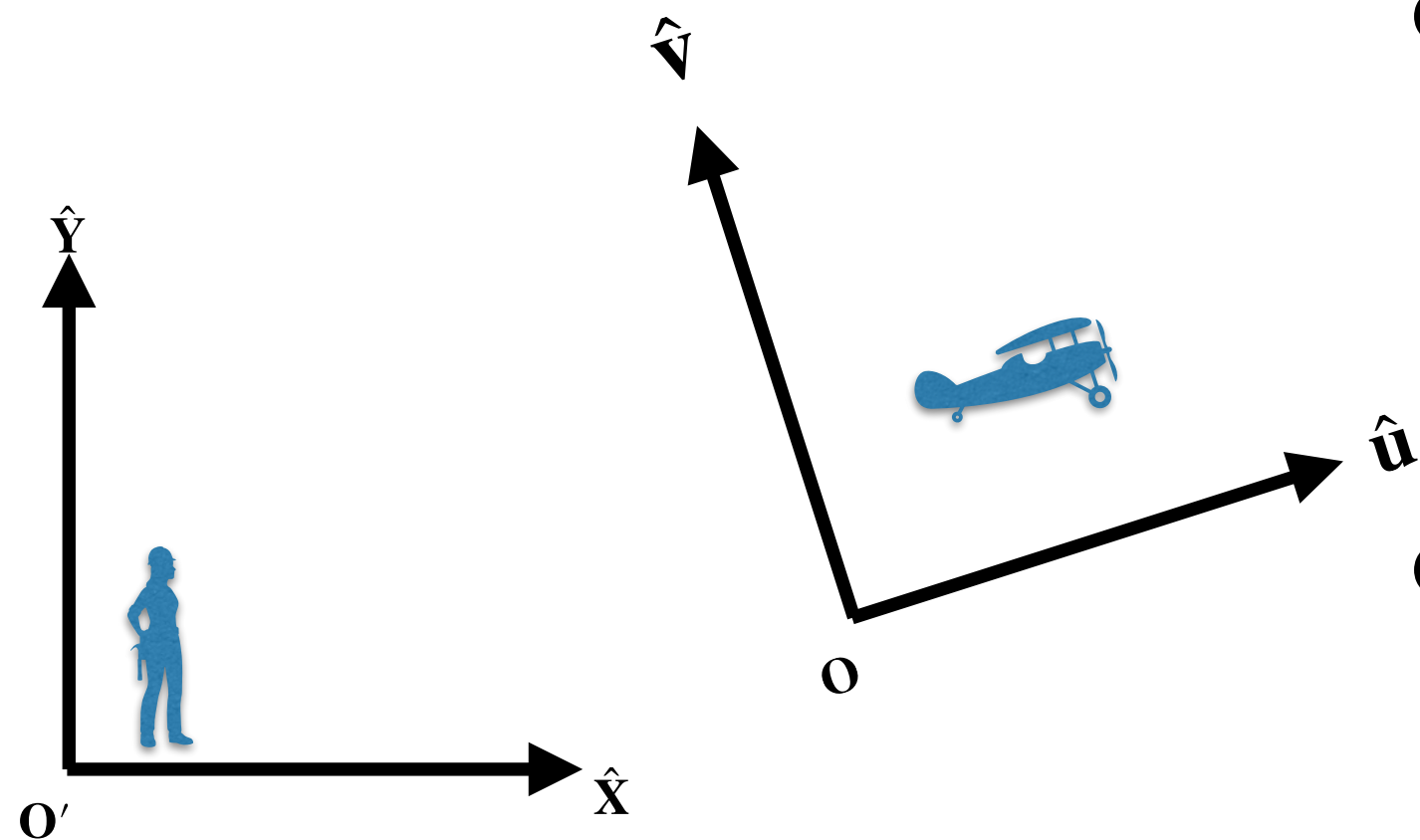
In general, a new coordinate frame is defined by an origin (point) and two unit axes (vectors)



Given coordinates in the (o,u,v) reference frame, what is the transform to coordinates in the (x,y) frame?

Coordinate System Transform Matrix

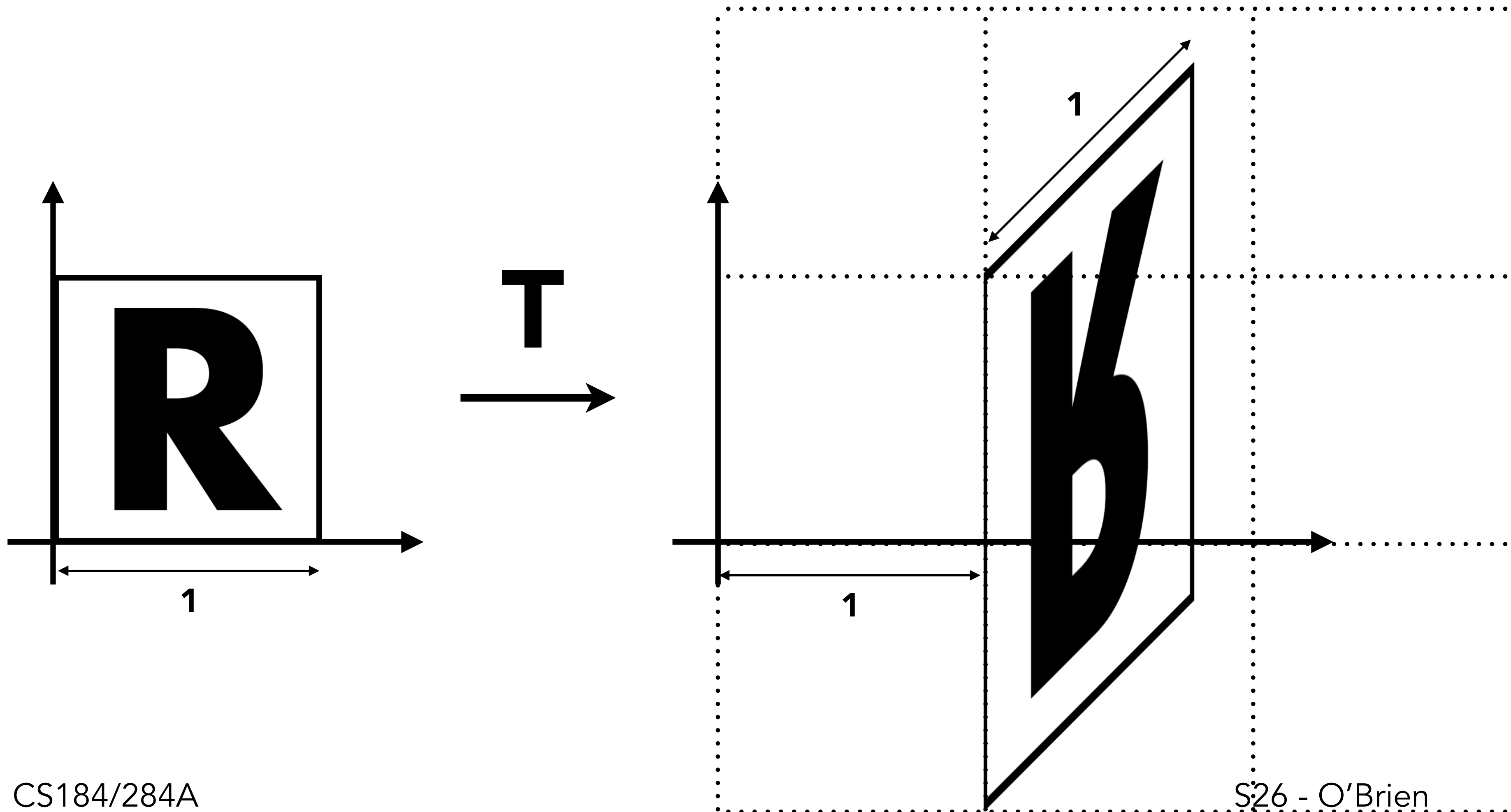
$$F = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{o} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u_x & v_x & o_x \\ u_y & v_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$



- Columns of matrix are defined by the reference frame's coordinates in the world
- Gives a new way to "read off" columns of matrix

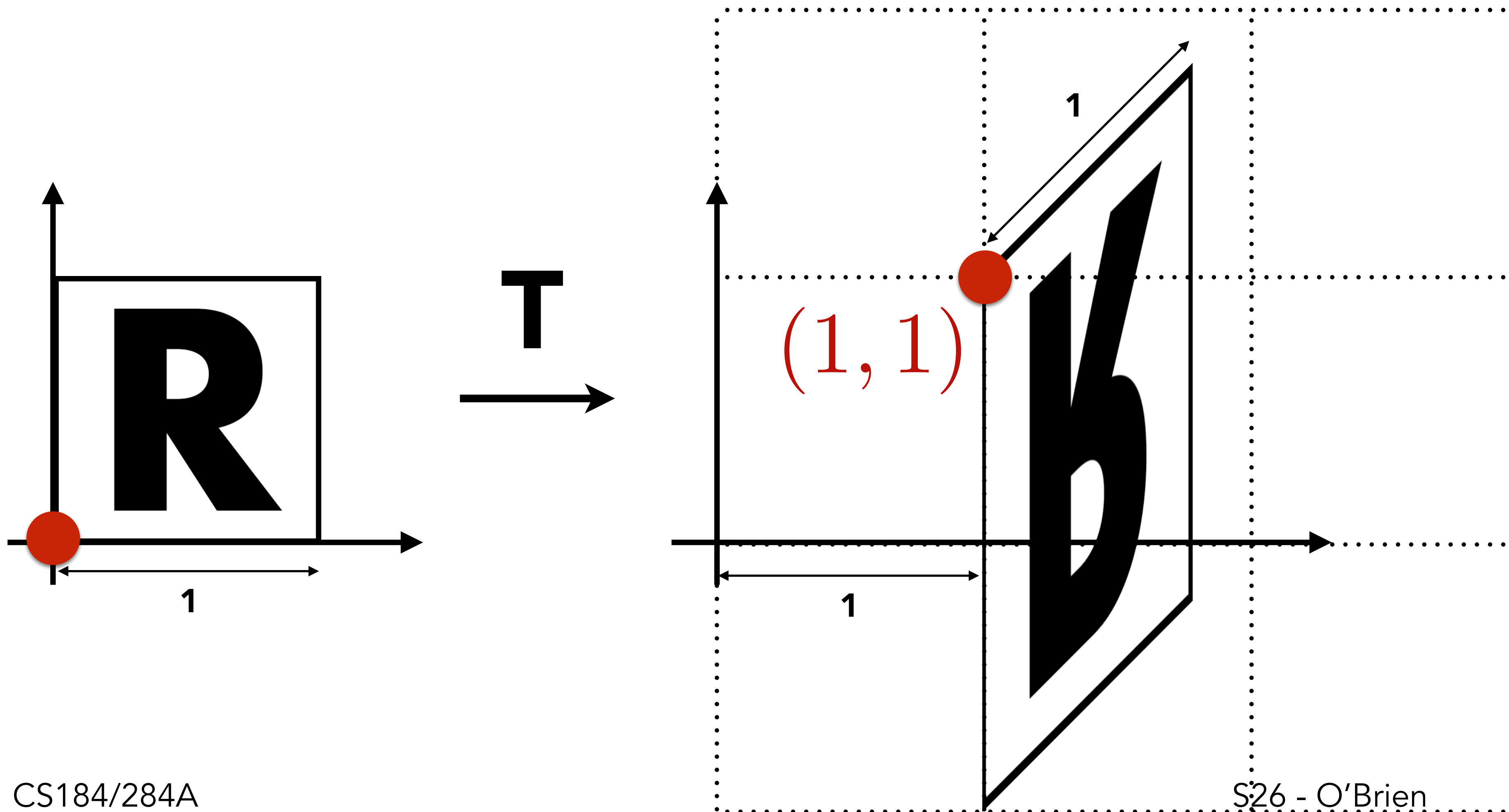
Coordinate System Transform - Example

Write down a matrix T representing this transform:



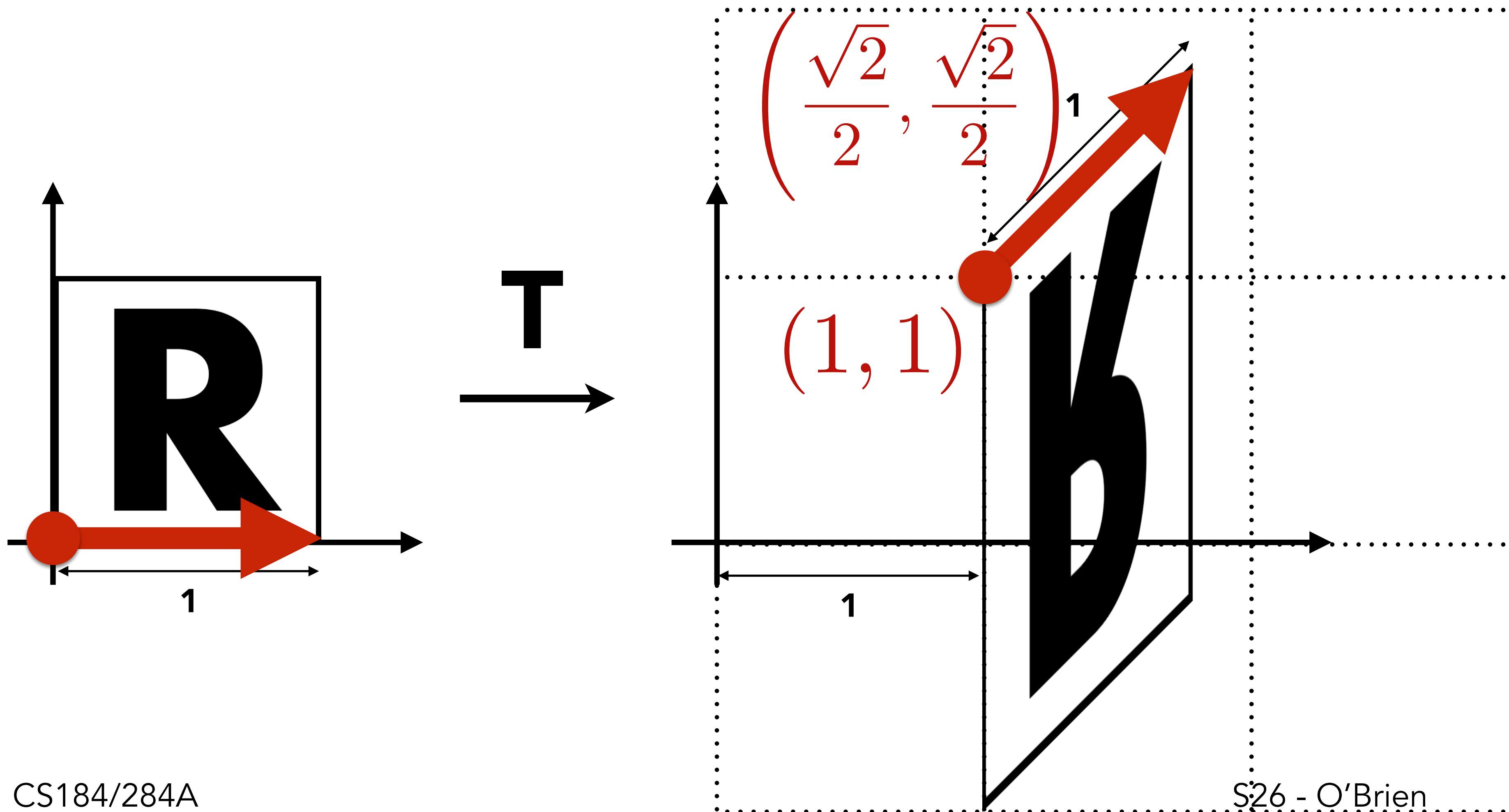
Coordinate System Transform - Example

Write down a matrix T representing this transform:



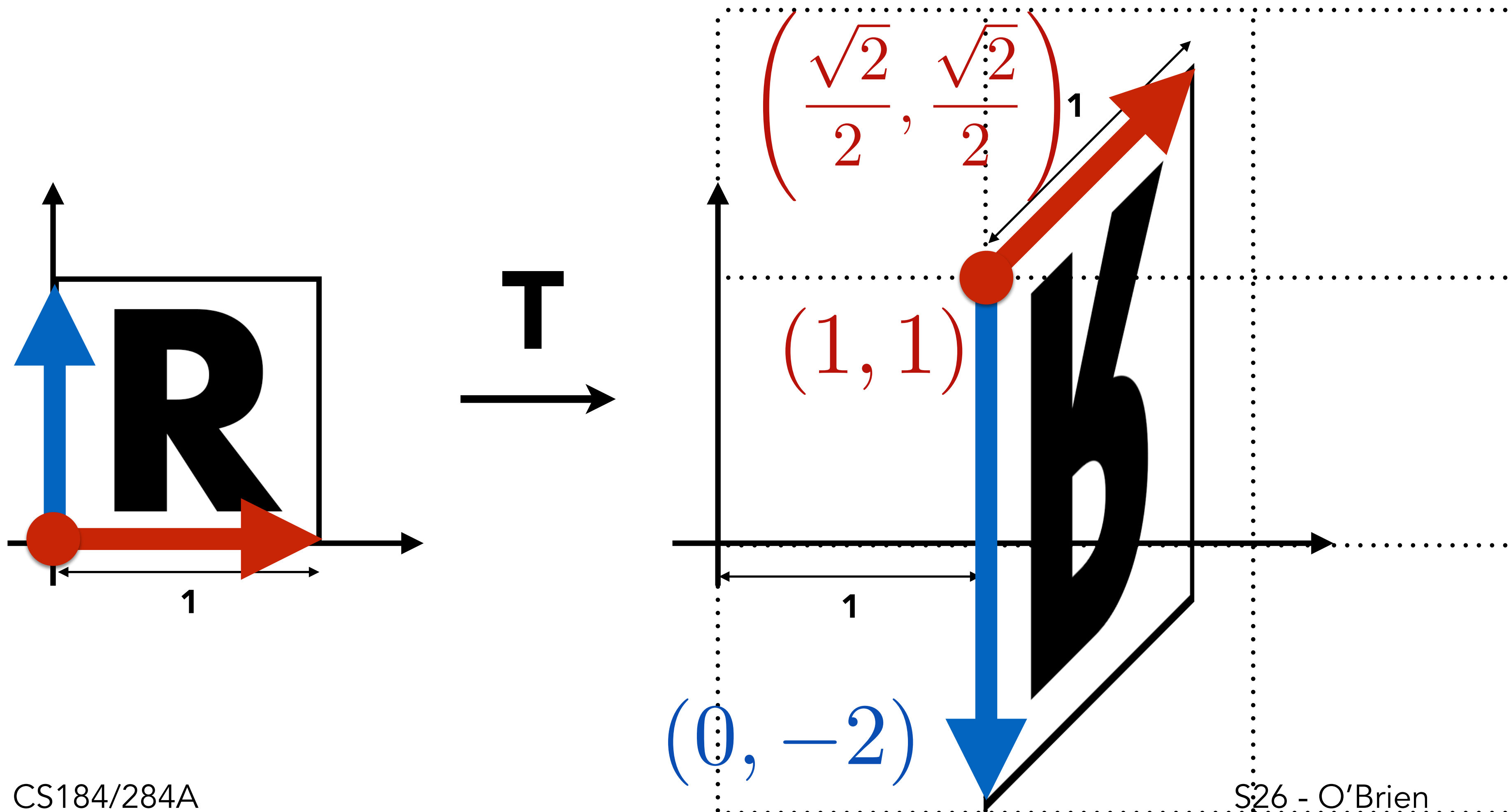
Coordinate System Transform - Example

Write down a matrix T representing this transform:



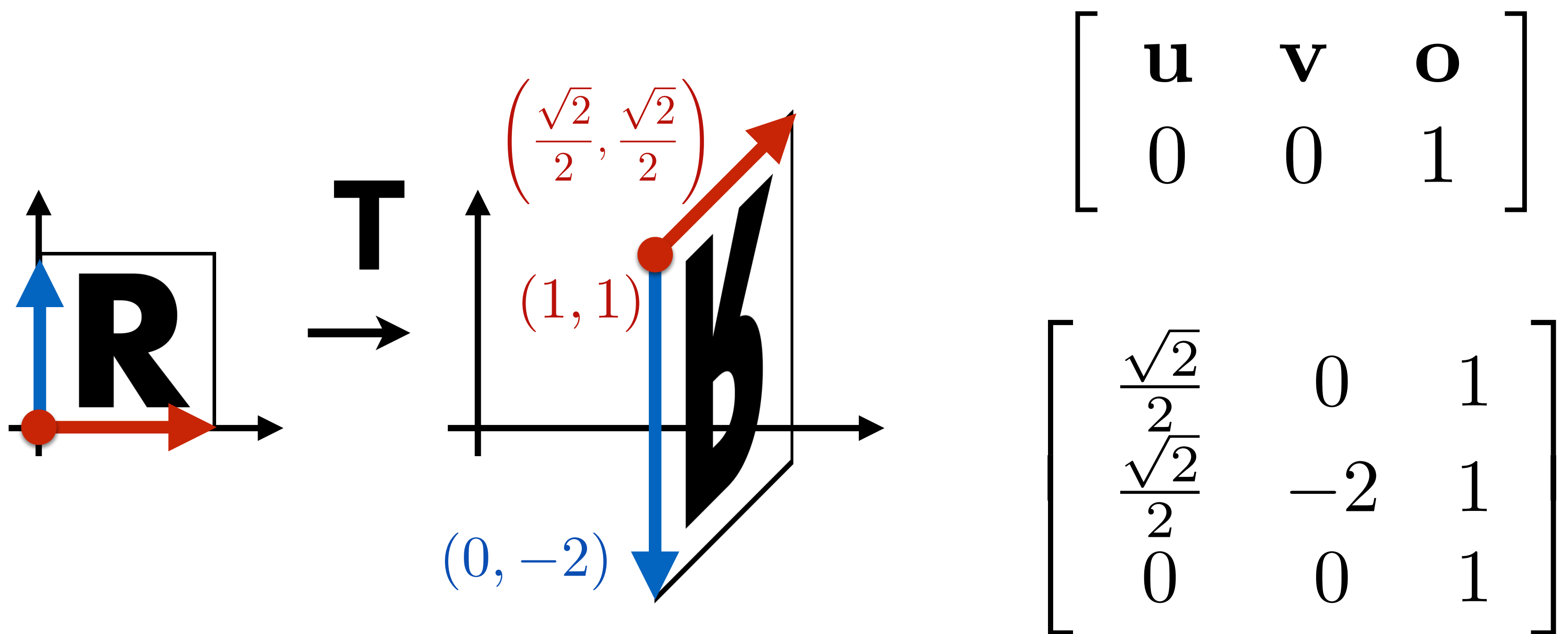
Coordinate System Transform - Example

Write down a matrix T representing this transform:



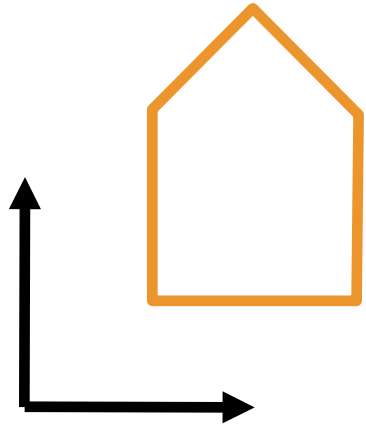
Coordinate System Transform - Example

Write down a matrix T representing this transform:



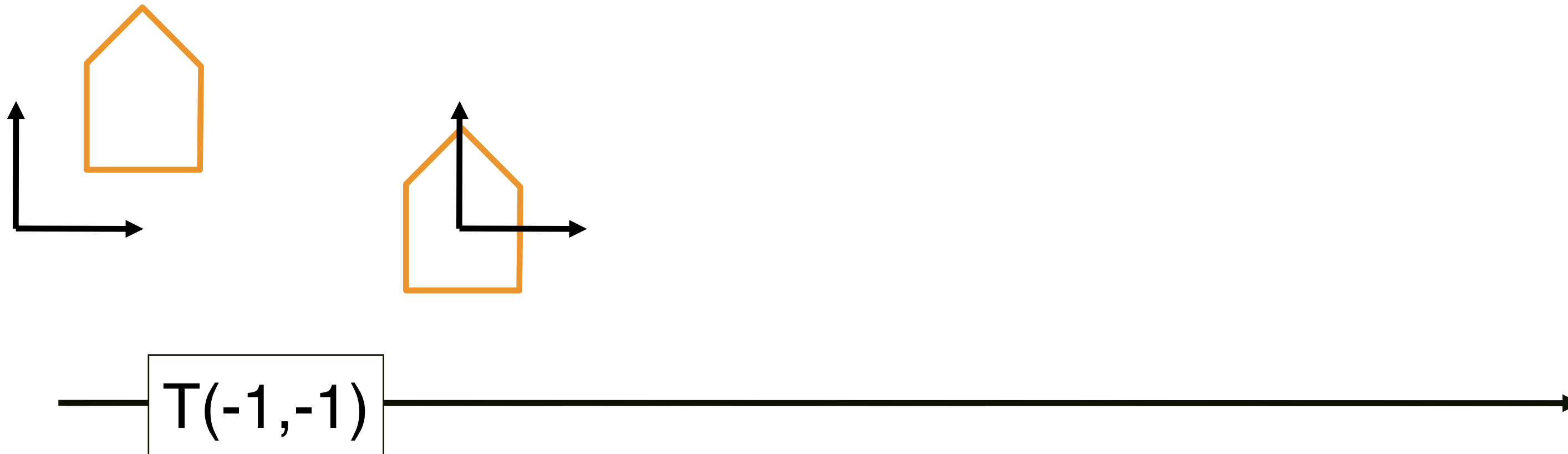
Two Interpretations of A Transform

Interpretation 1: Transforms object points



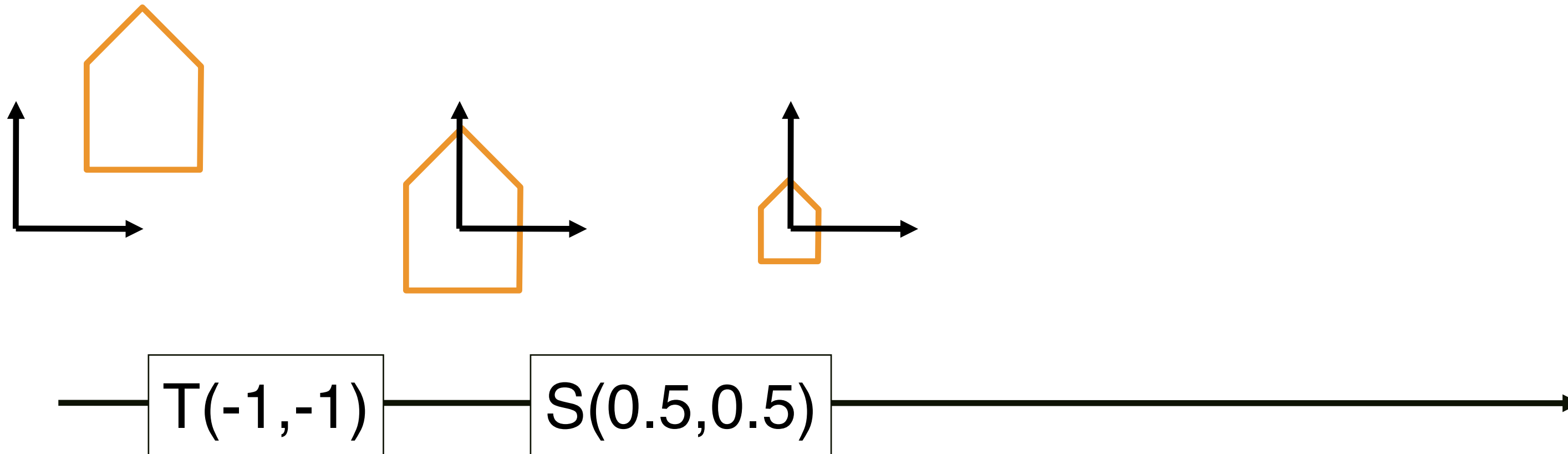
Two Interpretations of A Transform

Interpretation 1: Transforms object points



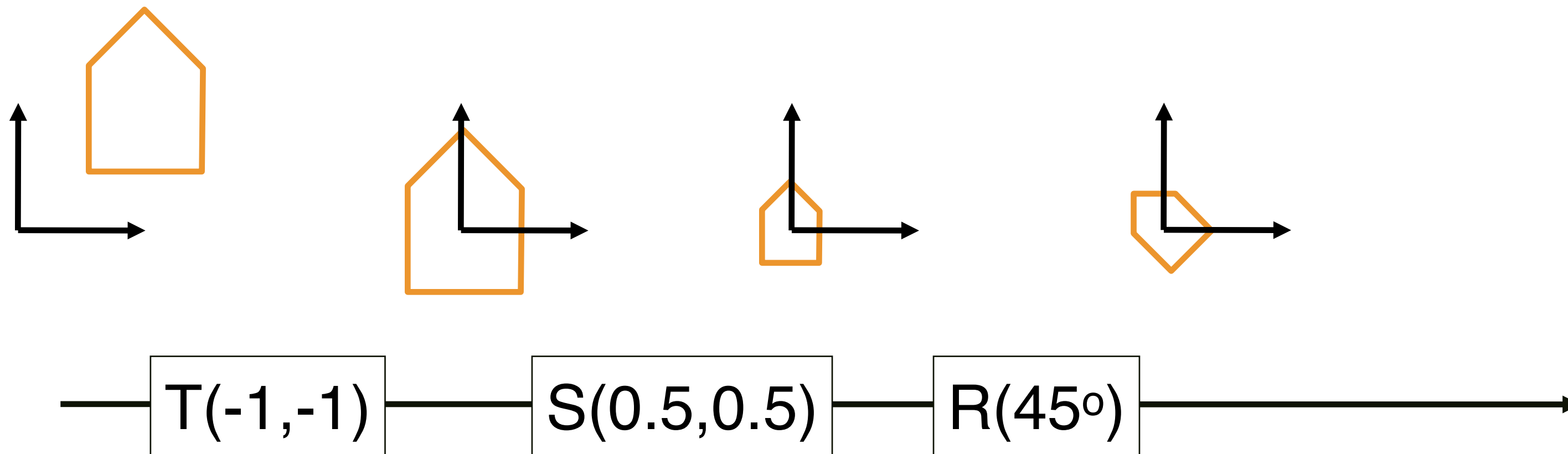
Two Interpretations of A Transform

Interpretation 1: Transforms object points



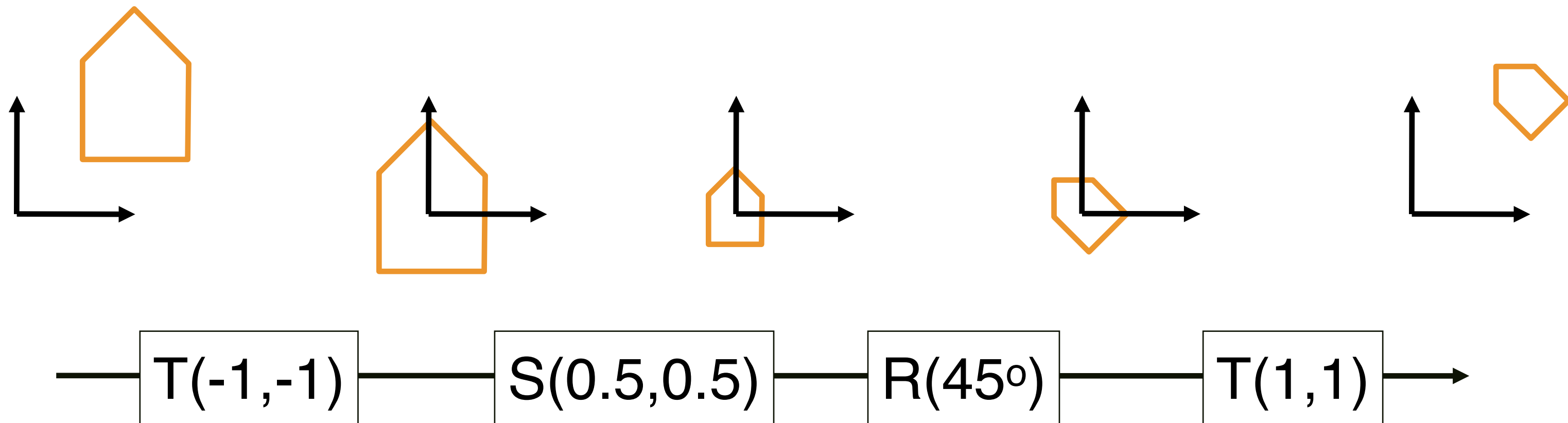
Two Interpretations of A Transform

Interpretation 1: Transforms object points



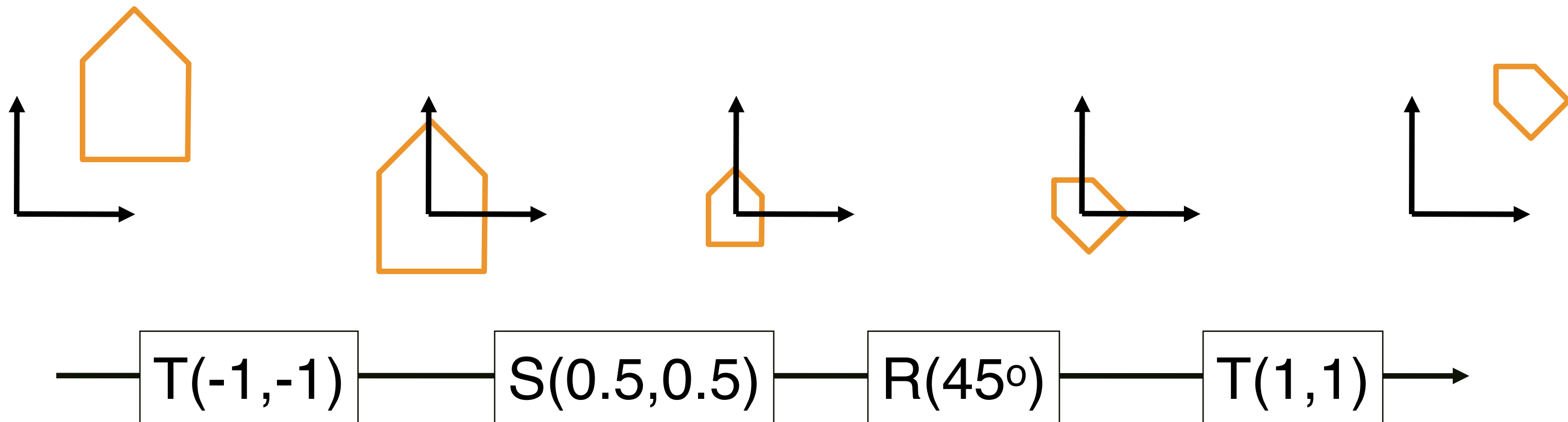
Two Interpretations of A Transform

Interpretation 1: Transforms object points



Two Interpretations of A Transform

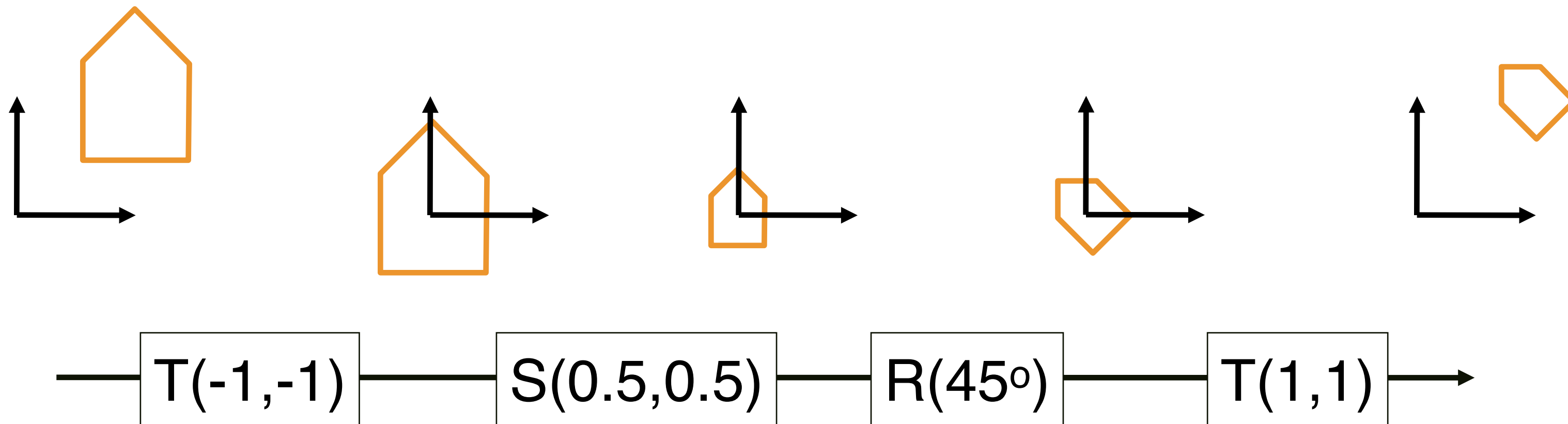
Interpretation 1: Transforms object points



Interpretation 2: Transforms coordinate system

Two Interpretations of A Transform

Interpretation 1: Transforms object points

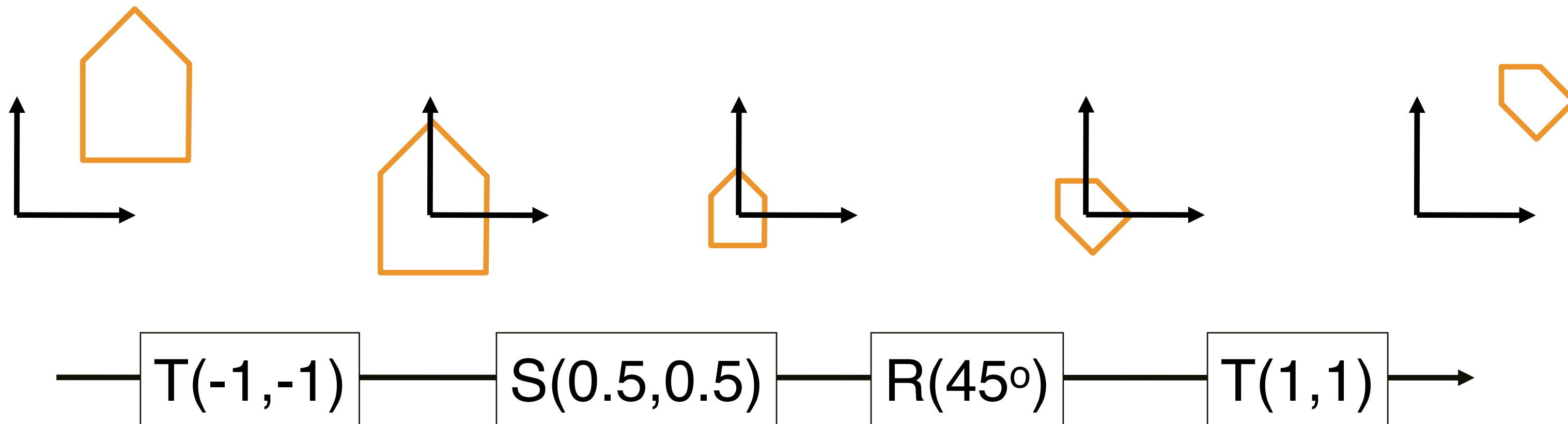


Interpretation 2: Transforms coordinate system



Two Interpretations of A Transform

Interpretation 1: Transforms object points

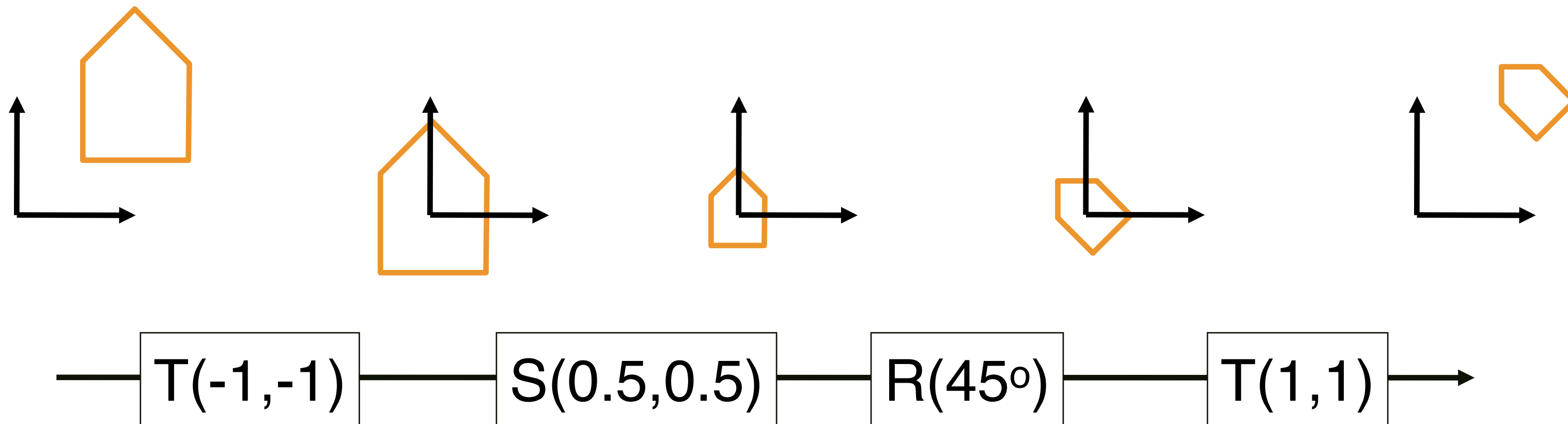


Interpretation 2: Transforms coordinate system

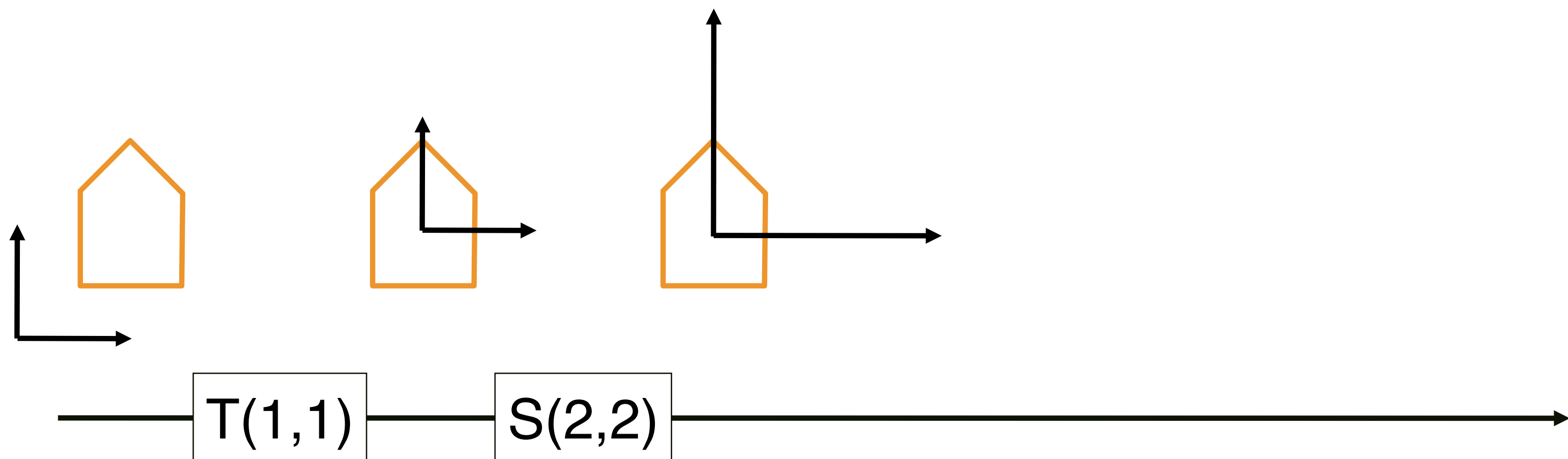


Two Interpretations of A Transform

Interpretation 1: Transforms object points

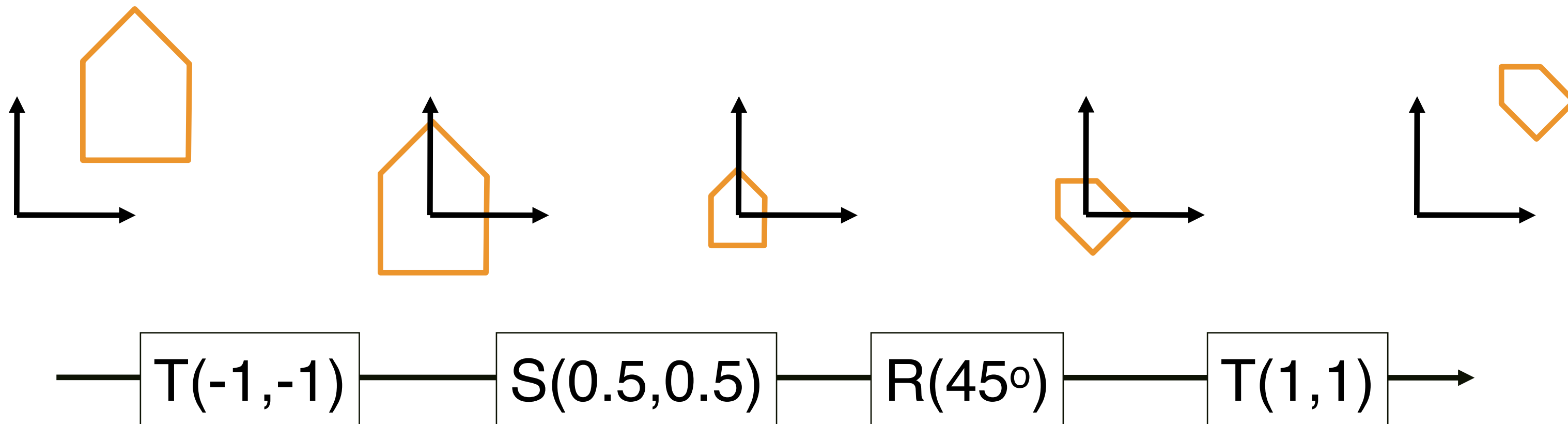


Interpretation 2: Transforms coordinate system

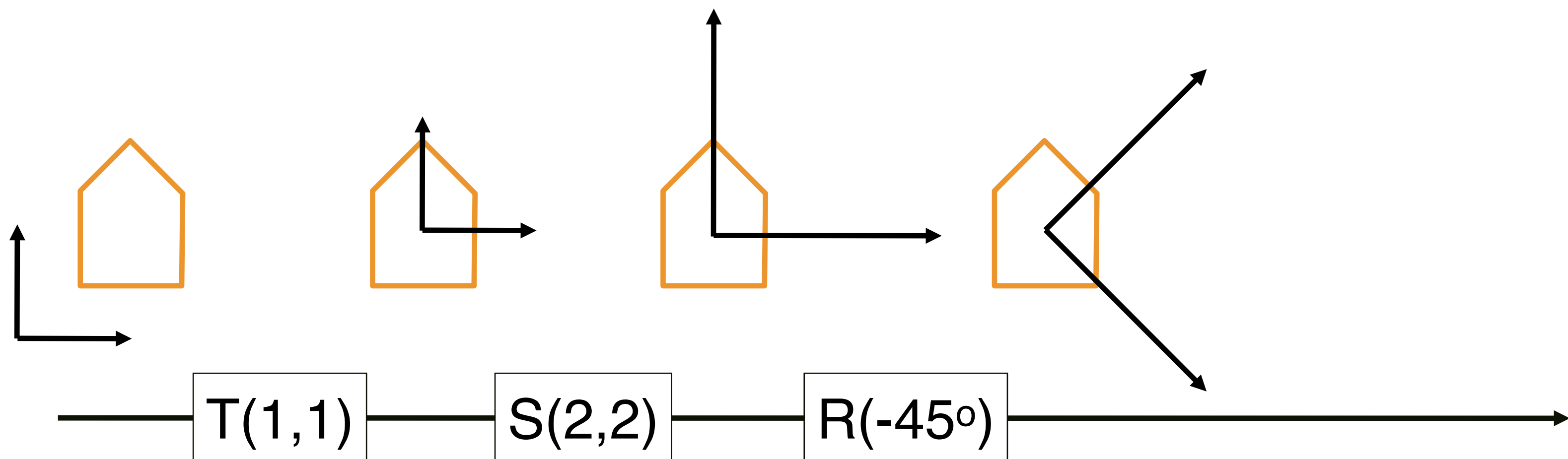


Two Interpretations of A Transform

Interpretation 1: Transforms object points

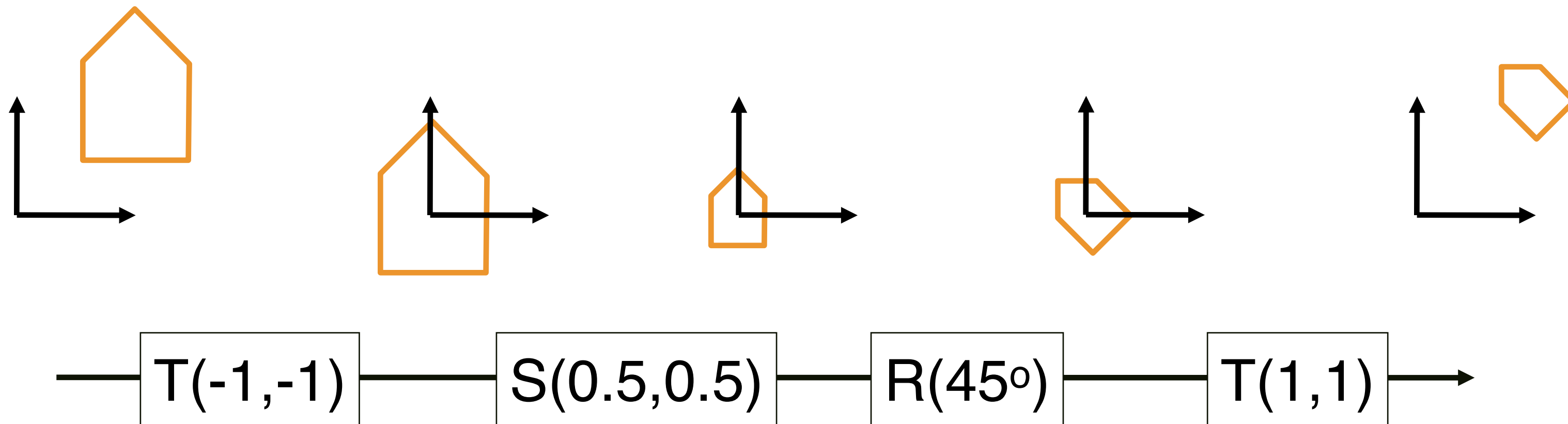


Interpretation 2: Transforms coordinate system

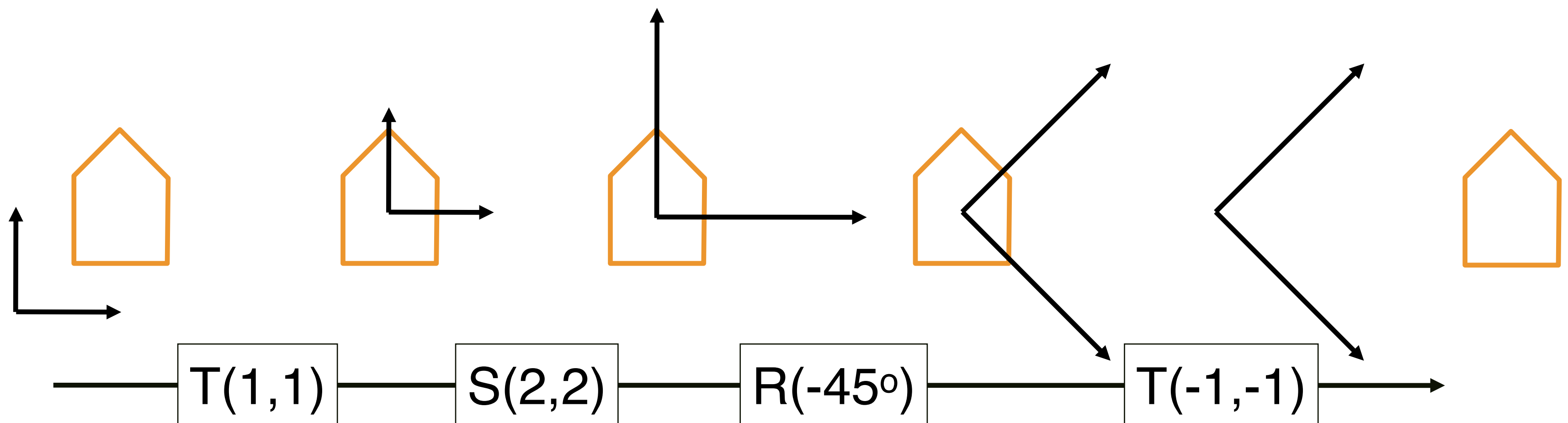


Two Interpretations of A Transform

Interpretation 1: Transforms object points



Interpretation 2: Transforms coordinate system



3D Transforms

Attendance

If you are seated in class, go to this form and sign in:

- <https://tinyurl.com/184class>



Word of the day:

- rotations

3D Transformations

Scale

$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Coordinate Change (Frame-to-world)

$$\mathbf{F}(\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{o}) = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{o} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Transformations

Use homogeneous coordinates again:

- 3D point = $(x, y, z, 1)^T$
- 3D vector = $(x, y, z, 0)^T$

Use 4x4 matrices for affine transformations

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotations

Rotations still orthonormal

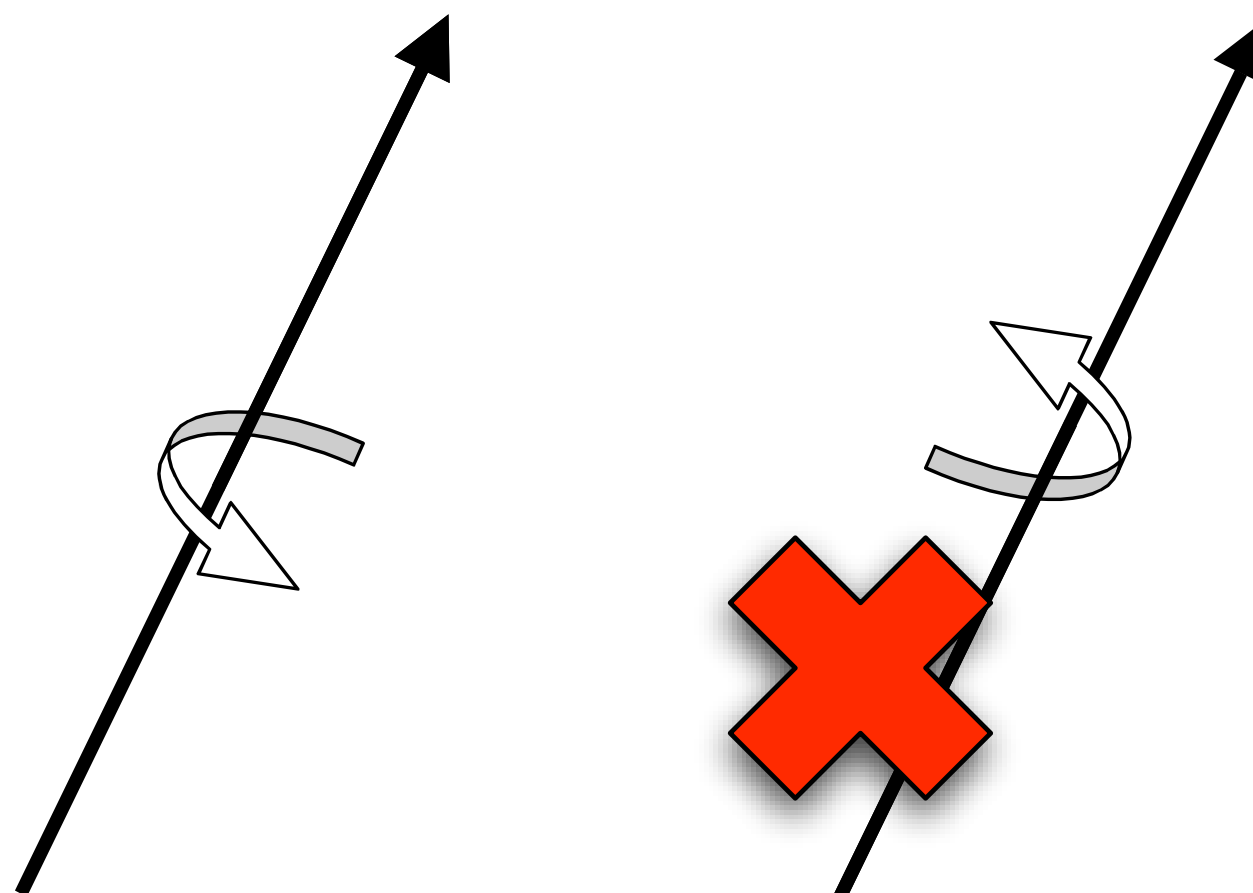
$$\text{Det}(\mathbf{R}) = 1 \neq -1$$

Preserve lengths and distance to origin

3D rotations DO NOT COMMUTE!

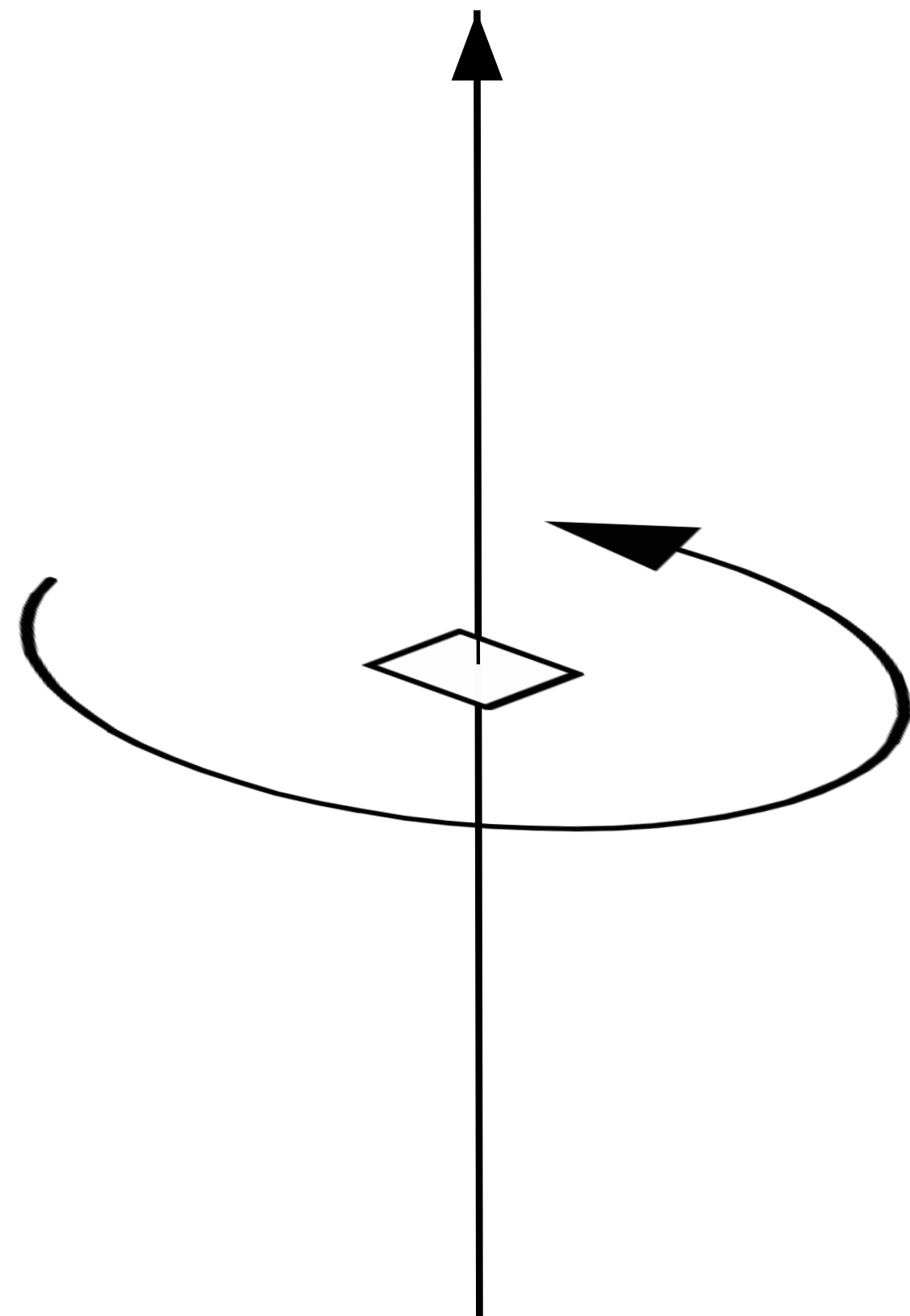
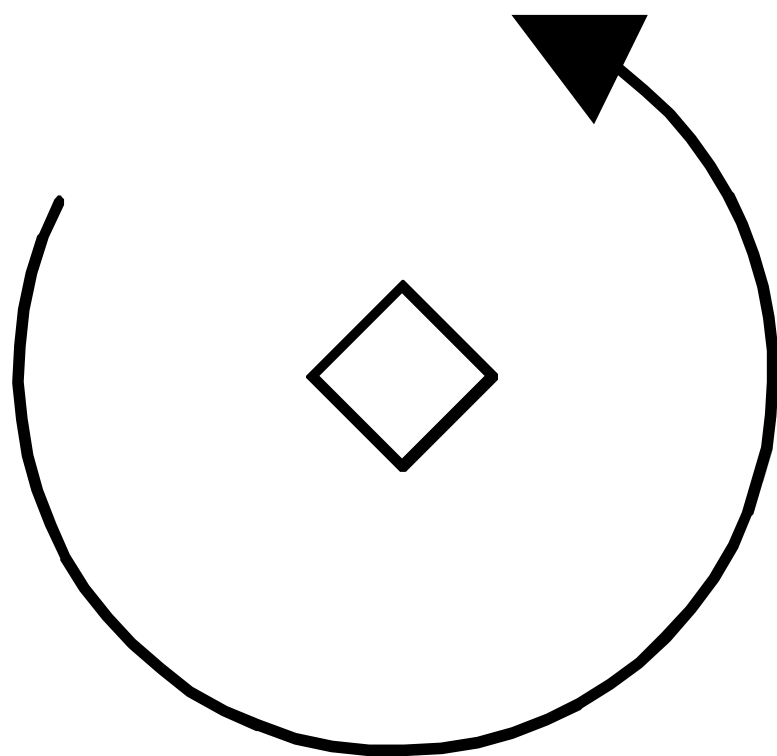
Right-hand rule

Unique matrices



2D and 3D Rotations

2D rotations implicitly rotate about a third out-of-plane axis



3D Transformations

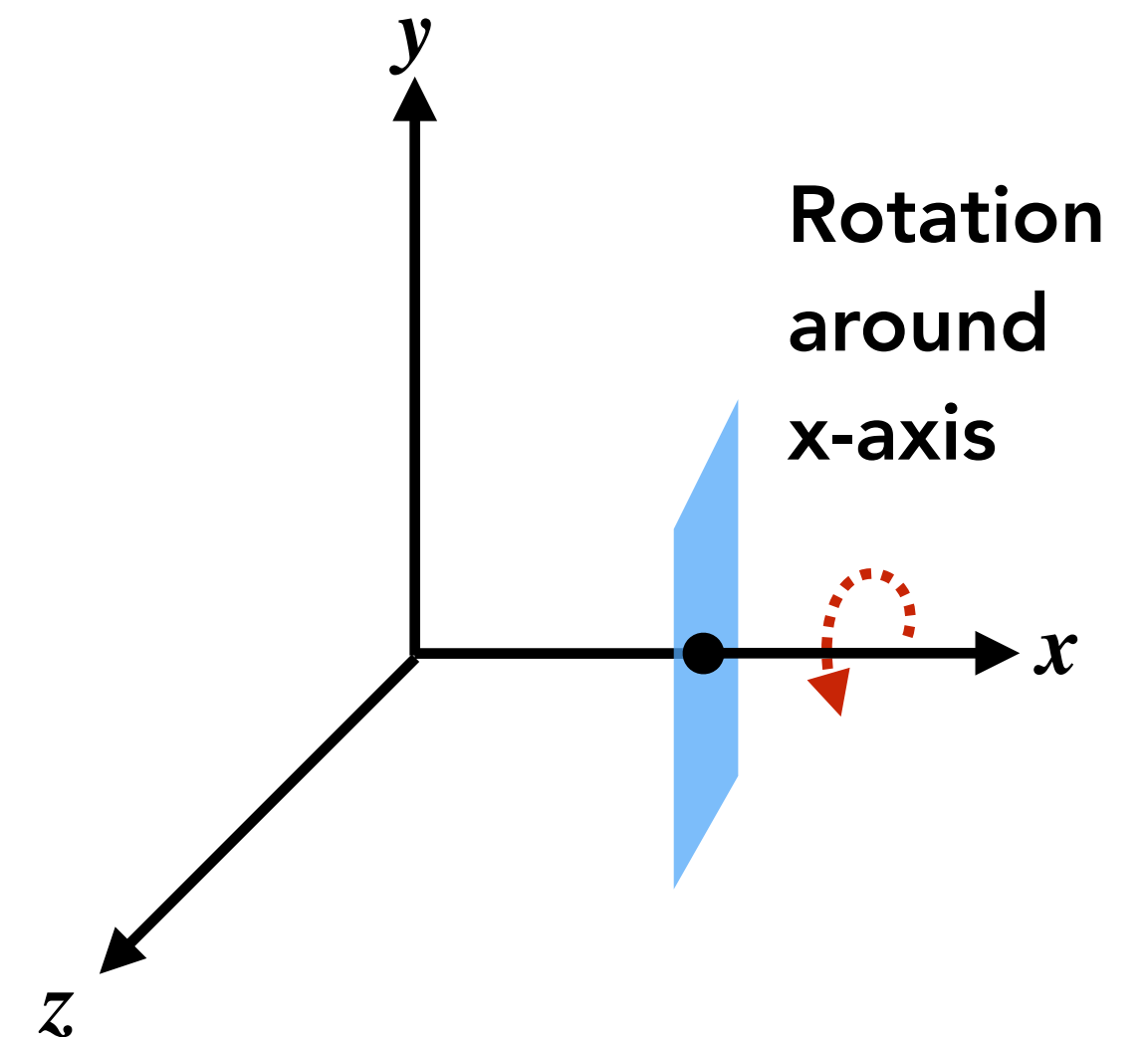
Rotation around x-, y-, or z-axis

- Also known as "direction-cosine" matrices

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

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

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



3D Rotations

Compose any 3D rotation from $\mathbf{R}_x$, $\mathbf{R}_y$, $\mathbf{R}_z$?

$$\mathbf{R}_{xyz}(\alpha, \beta, \gamma) = \mathbf{R}_x(\alpha) \mathbf{R}_y(\beta) \mathbf{R}_z(\gamma)$$

A list of the Euler angles is NOT a vector!

3D Rotations

Compose any 3D rotation from $\mathbf{R}_x$, $\mathbf{R}_y$, $\mathbf{R}_z$?

$$\mathbf{R}_{xyz}(\alpha, \beta, \gamma) = \mathbf{R}_x(\alpha) \mathbf{R}_y(\beta) \mathbf{R}_z(\gamma)$$

- So-called *Euler angles*

A list of the Euler angles is NOT a vector!

3D Rotations

Compose any 3D rotation from $\mathbf{R}_x$, $\mathbf{R}_y$, $\mathbf{R}_z$?

$$\mathbf{R}_{xyz}(\alpha, \beta, \gamma) = \mathbf{R}_x(\alpha) \mathbf{R}_y(\beta) \mathbf{R}_z(\gamma)$$

- So-called *Euler angles*
- Often used in flight simulators: roll, pitch, yaw

A list of the Euler angles is NOT a vector!

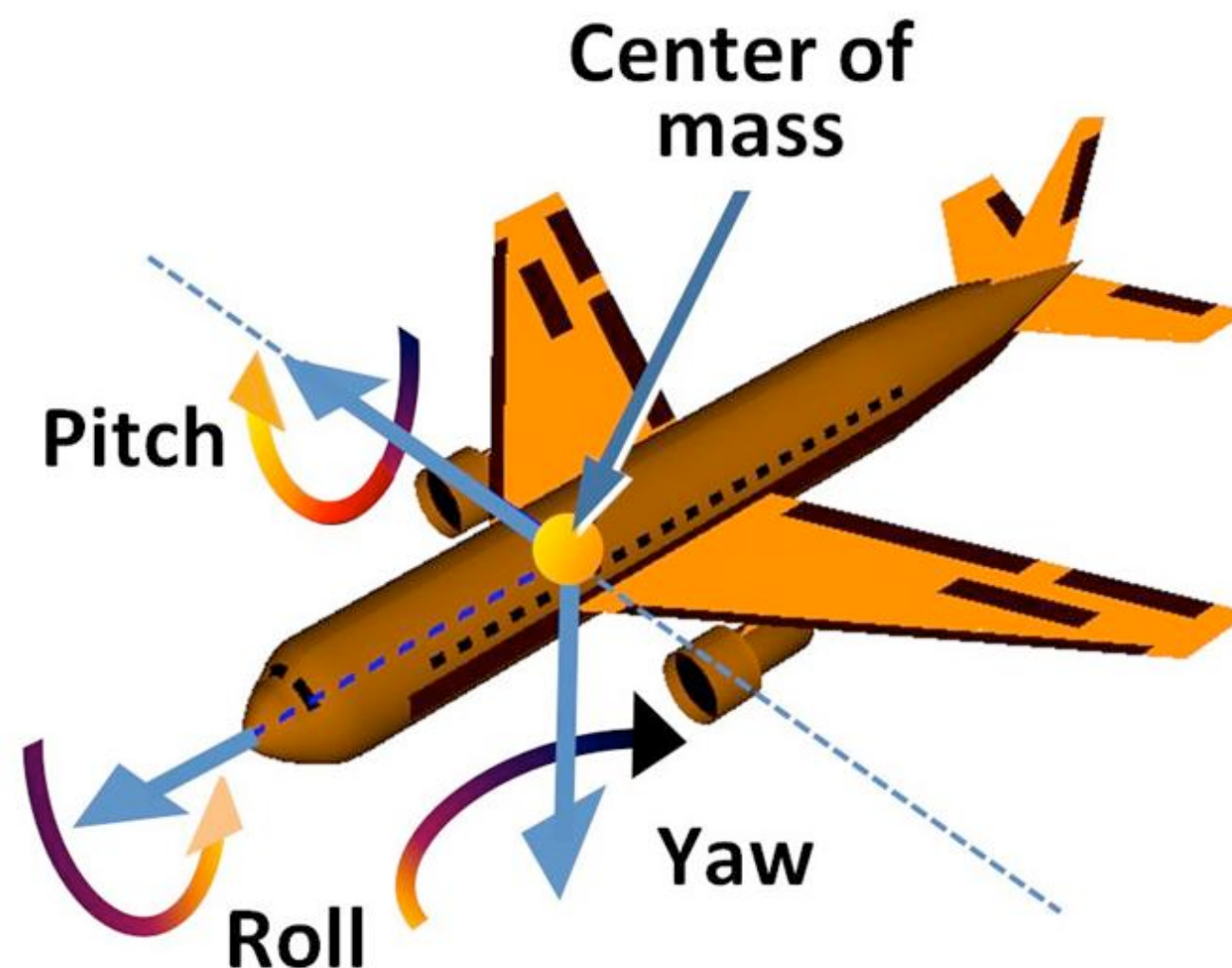
3D Rotations

Compose any 3D rotation from $\mathbf{R}_x$, $\mathbf{R}_y$, $\mathbf{R}_z$?

$$\mathbf{R}_{xyz}(\alpha, \beta, \gamma) = \mathbf{R}_x(\alpha) \mathbf{R}_y(\beta) \mathbf{R}_z(\gamma)$$

- So-called *Euler angles*
- Often used in flight simulators: roll, pitch, yaw

A list of the Euler angles is NOT a vector!



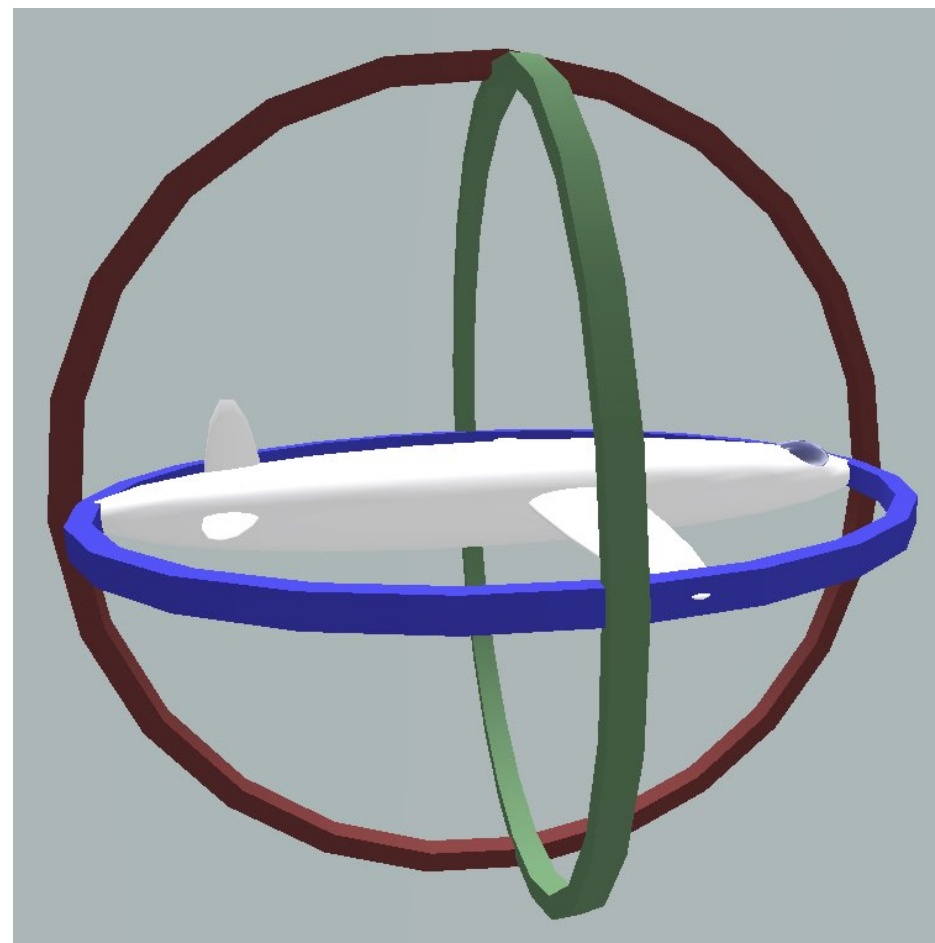
3D Rotations

Compose any 3D rotation from $\mathbf{R}_x, \mathbf{R}_y, \mathbf{R}_z$?

$$\mathbf{R}_{xyz}(\alpha, \beta, \gamma) = \mathbf{R}_x(\alpha) \mathbf{R}_y(\beta) \mathbf{R}_z(\gamma)$$

- So-called *Euler angles*
- Often used in flight simulators: roll, pitch, yaw

- **Problems:**
Gimbal Lock!
Not unique!
Ugly manifold!



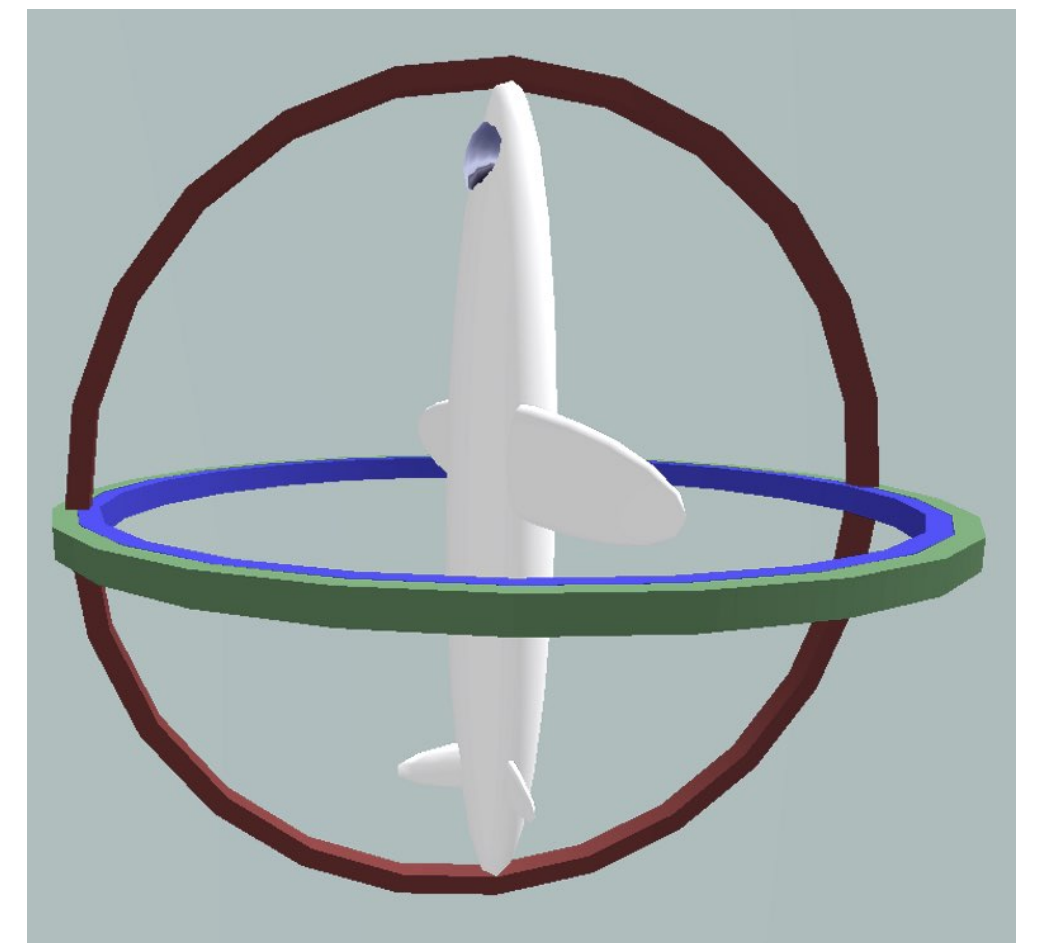
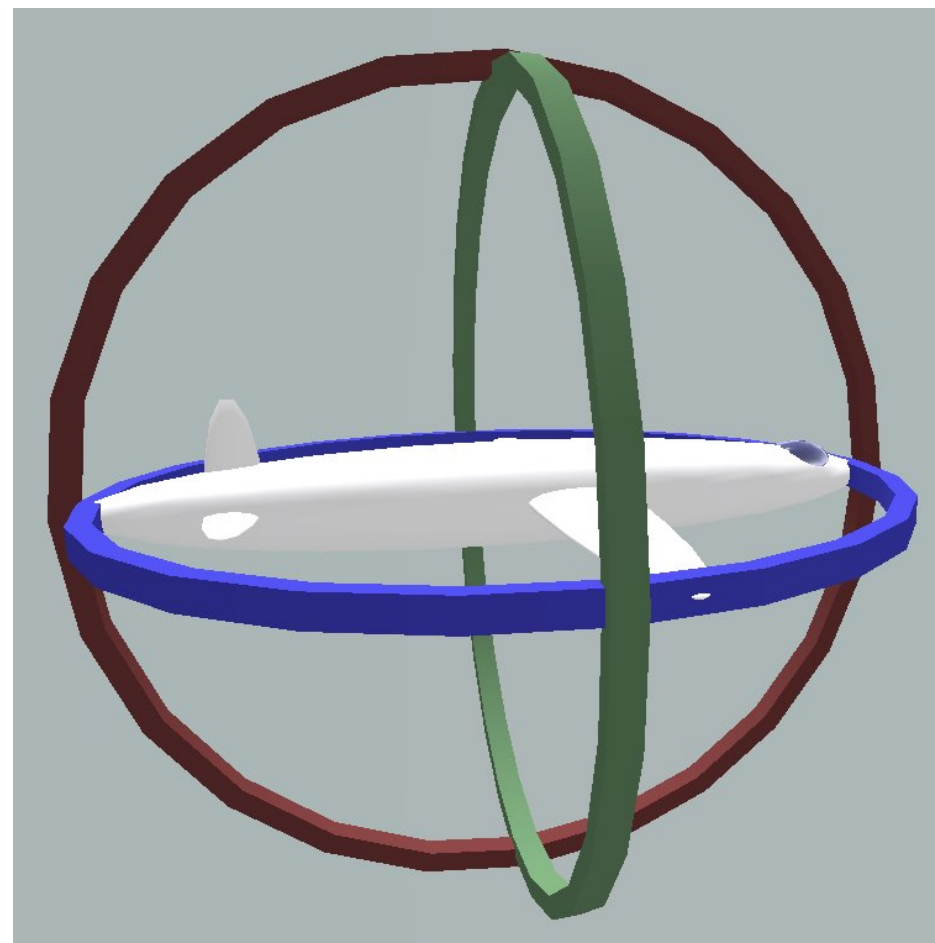
3D Rotations

Compose any 3D rotation from $\mathbf{R}_x, \mathbf{R}_y, \mathbf{R}_z$?

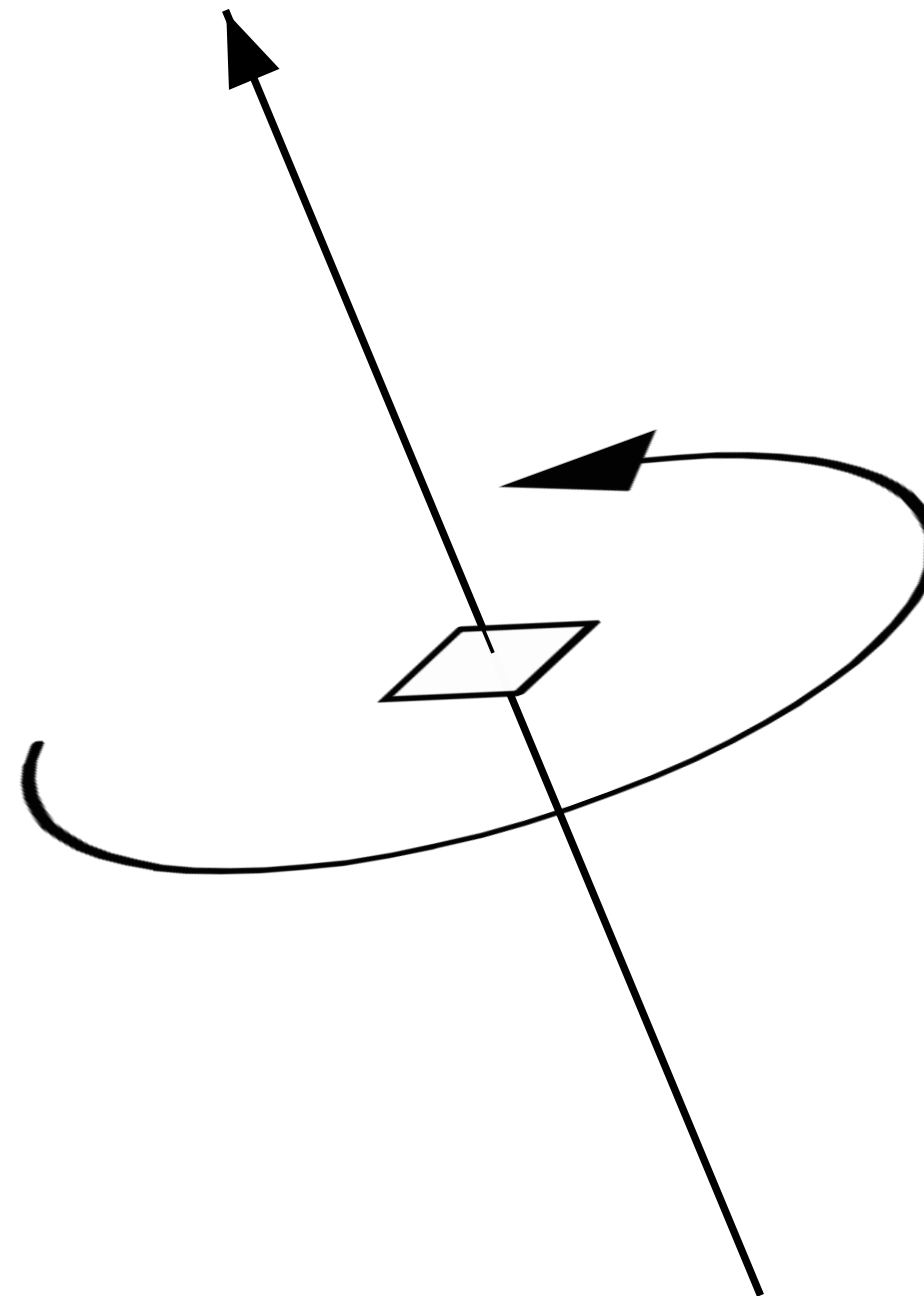
$$\mathbf{R}_{xyz}(\alpha, \beta, \gamma) = \mathbf{R}_x(\alpha) \mathbf{R}_y(\beta) \mathbf{R}_z(\gamma)$$

- So-called *Euler angles*
- Often used in flight simulators: roll, pitch, yaw

- **Problems:**
Gimbal Lock!
Not unique!
Ugly manifold!



3D Rotation Around Arbitrary Axis

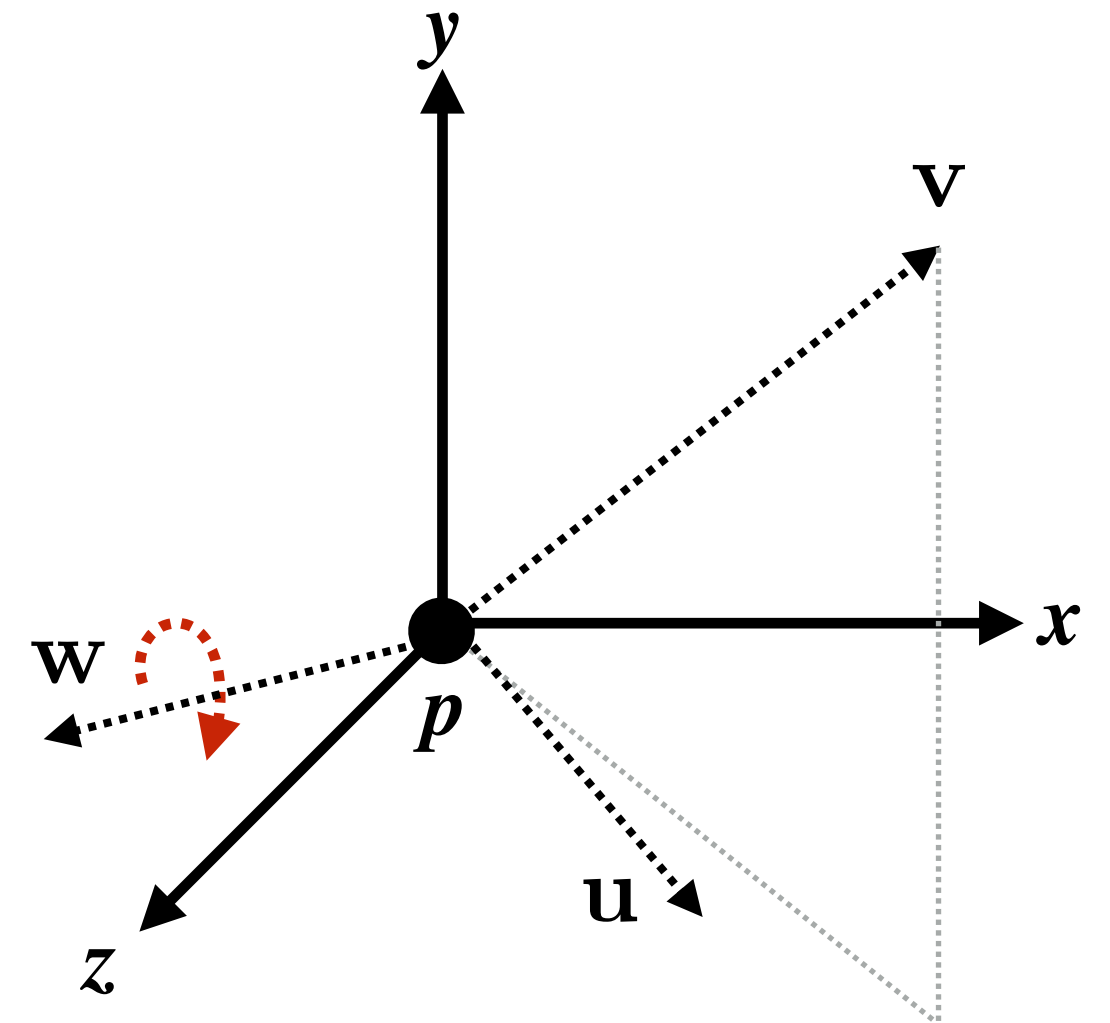


3D Rotation Around Arbitrary Axis

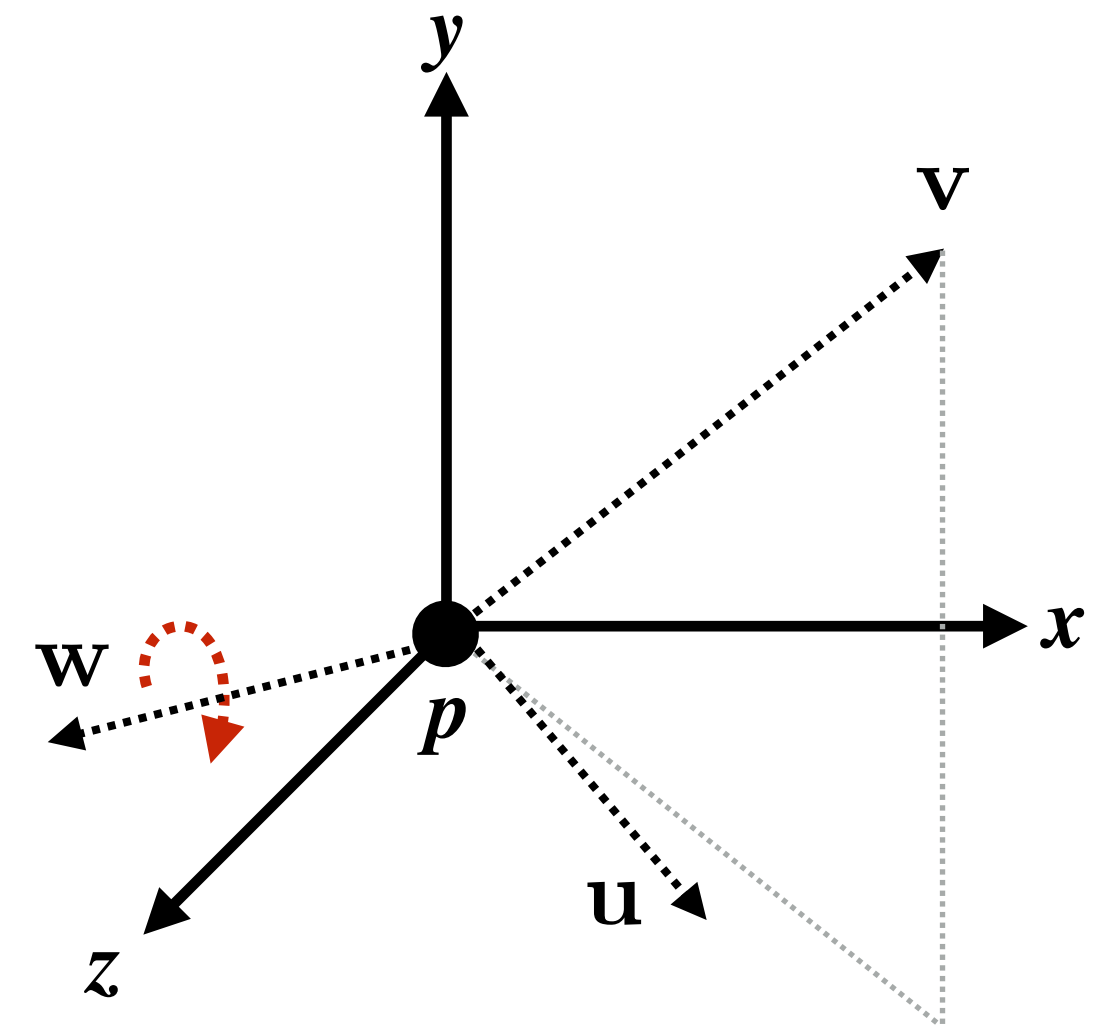
- **Easy to understand but less useful method:**

1. rotate about x axis to put w into the x - y plane
2. rotate about z axis align w with the x axis
3. rotate θ degrees about x axis
4. undo #2 and then #1
5. composite together

How would you take a derivative w.r.t. w ?

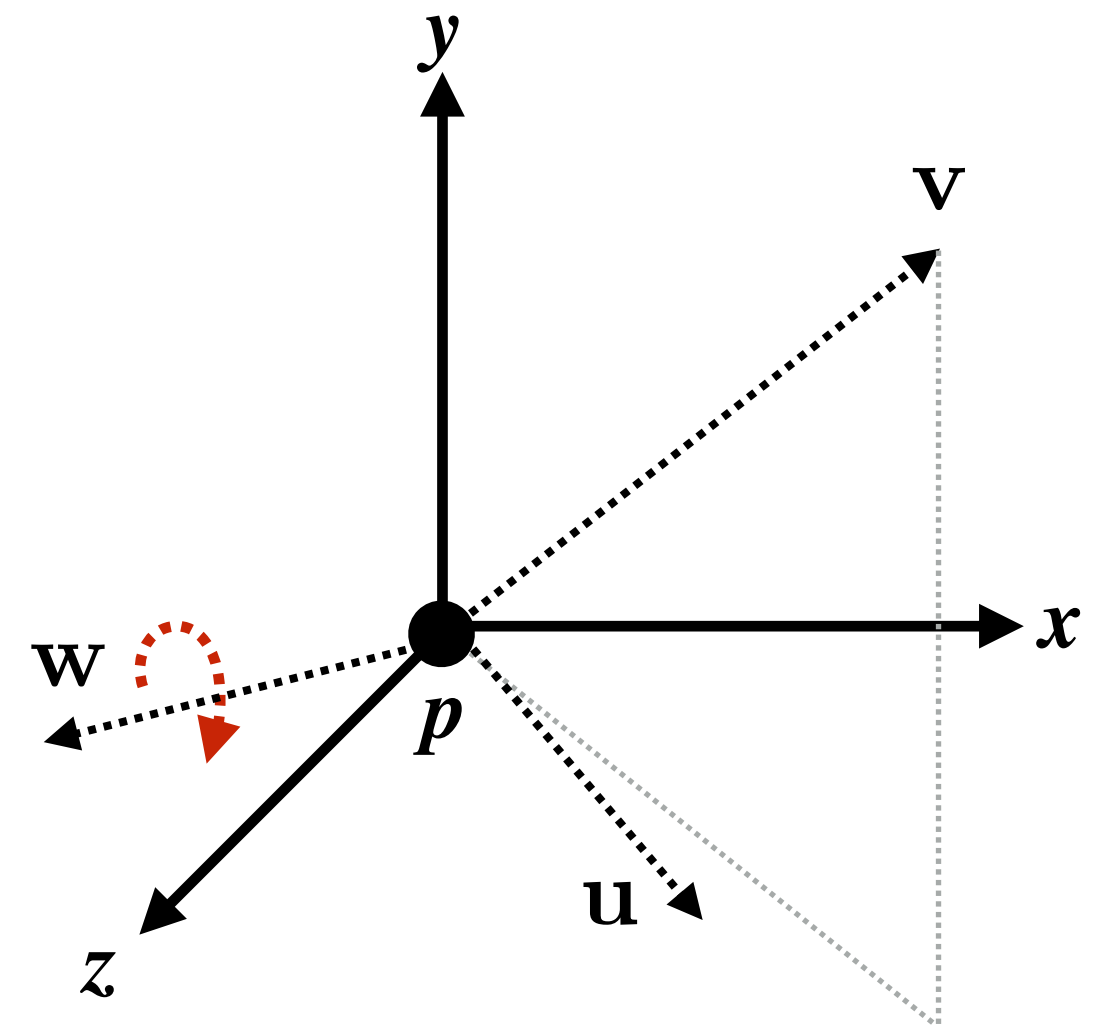


3D Rotation Around Arbitrary Axis



3D Rotation Around Arbitrary Axis

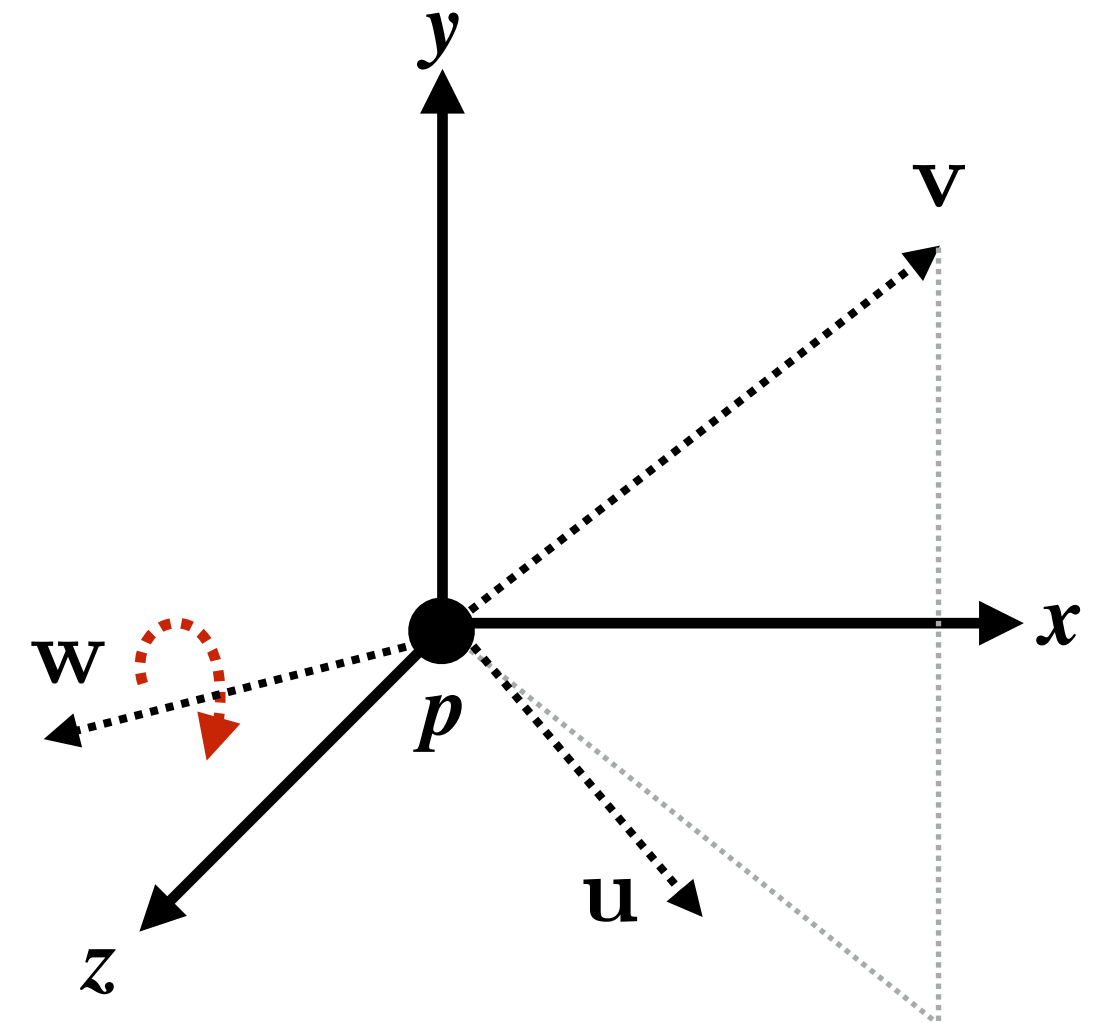
Construct orthonormal frame transformation F with p, u, v, w , where p and w match the rotation axis



3D Rotation Around Arbitrary Axis

Construct orthonormal frame transformation F with p, u, v, w , where p and w match the rotation axis

Apply the transform $(F R_z(\theta) F^{-1})$

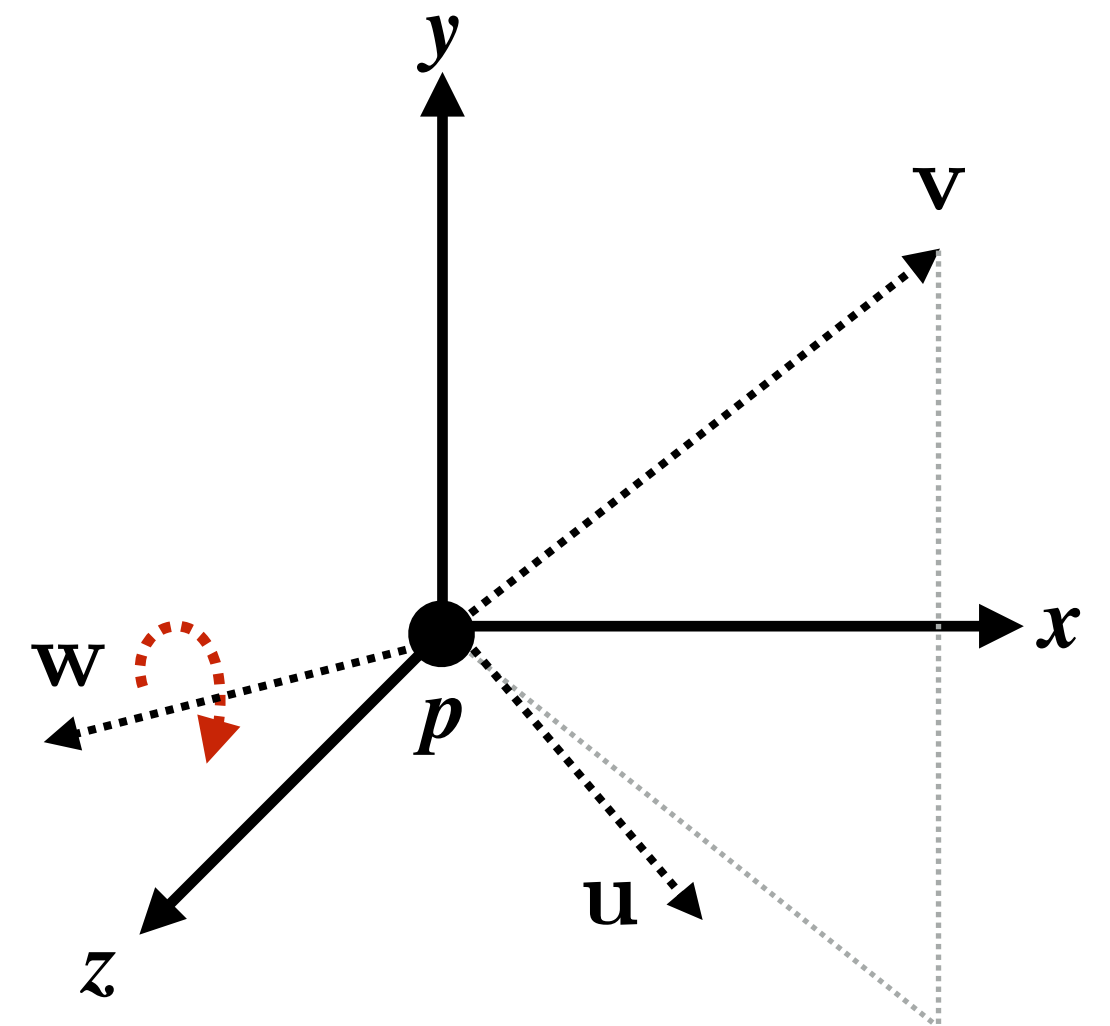


3D Rotation Around Arbitrary Axis

Construct orthonormal frame transformation F with p, u, v, w , where p and w match the rotation axis

Apply the transform $(F R_z(\theta) F^{-1})$

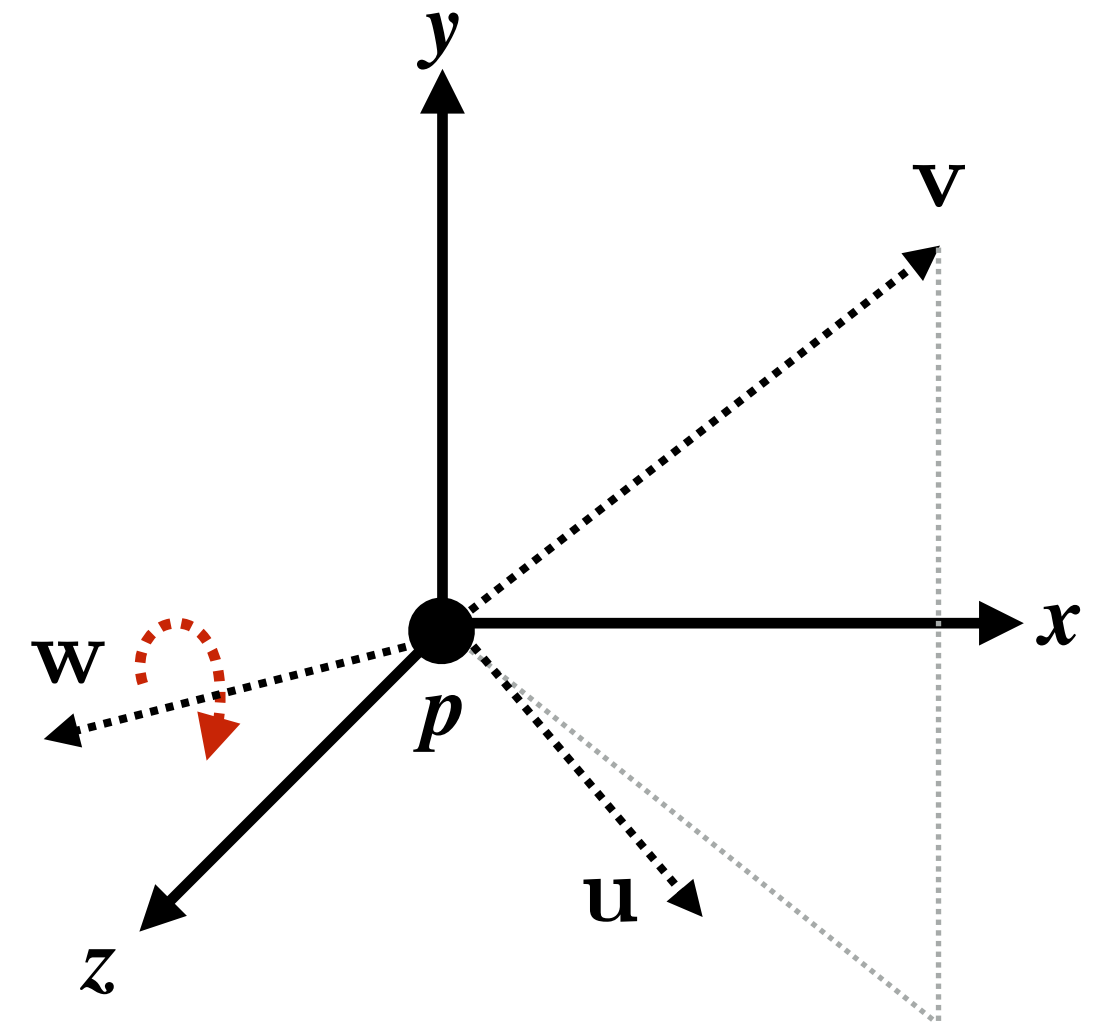
Interpretation:



3D Rotation Around Arbitrary Axis

Construct orthonormal frame transformation F with p, u, v, w , where p and w match the rotation axis

Apply the transform $(F R_z(\theta) F^{-1})$



Interpretation:

- Rotate to Z axis, rotate, then move back

Rodrigues' Rotation Formula

Rotation by angle α around axis $\mathbf{n}$

$$\mathbf{R}(\mathbf{n}, \alpha) = \cos(\alpha) \mathbf{I} + (1 - \cos(\alpha)) \mathbf{n}\mathbf{n}^T + \sin(\alpha) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathbf{N}}$$

**Turn cross-product into
matrix multiplication!**

$$\mathbf{n} \times \mathbf{v} = \mathbf{N}\mathbf{v}$$

Alternative notation

$$\mathbf{n} \times \mathbf{v} = [\mathbf{n}]\mathbf{v}$$

$$\mathbf{n} \times \mathbf{v} = (\mathbf{n}\times)\mathbf{v}$$

Rodrigues' Rotation Formula

Rotation by angle α around axis $\mathbf{n}$

$$\mathbf{R}(\mathbf{n}, \alpha) = \cos(\alpha) \mathbf{I} + (1 - \cos(\alpha)) \mathbf{n}\mathbf{n}^T + \sin(\alpha) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathbf{N}}$$

How to prove this magic formula?

Turn cross-product into
matrix multiplication!

$$\mathbf{n} \times \mathbf{v} = \mathbf{N}\mathbf{v}$$

Alternative notation

$$\mathbf{n} \times \mathbf{v} = [\mathbf{n}]\mathbf{v}$$

$$\mathbf{n} \times \mathbf{v} = (\mathbf{n}\times)\mathbf{v}$$

Rodrigues' Rotation Formula

Rotation by angle α around axis $\mathbf{n}$

$$\mathbf{R}(\mathbf{n}, \alpha) = \cos(\alpha) \mathbf{I} + (1 - \cos(\alpha)) \mathbf{n}\mathbf{n}^T + \sin(\alpha) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathbf{N}}$$

How to prove this magic formula?

- Matrix $\mathbf{N}$ computes a cross-product:

Turn cross-product into
matrix multiplication!

$$\mathbf{n} \times \mathbf{v} = \mathbf{N}\mathbf{v}$$

Alternative notation

$$\mathbf{n} \times \mathbf{v} = [\mathbf{n}]\mathbf{v}$$

$$\mathbf{n} \times \mathbf{v} = (\mathbf{n}\times)\mathbf{v}$$

Rodrigues' Rotation Formula

Rotation by angle α around axis $\mathbf{n}$

$$\mathbf{R}(\mathbf{n}, \alpha) = \cos(\alpha) \mathbf{I} + (1 - \cos(\alpha)) \mathbf{n}\mathbf{n}^T + \sin(\alpha) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathbf{N}}$$

How to prove this magic formula?

- Matrix $\mathbf{N}$ computes a cross-product:
- Assume orthonormal system $\mathbf{e}_1, \mathbf{e}_2, \mathbf{n}$

Turn cross-product into
matrix multiplication!

$$\mathbf{n} \times \mathbf{v} = \mathbf{N}\mathbf{v}$$

Alternative notation

$$\mathbf{n} \times \mathbf{v} = [\mathbf{n}]\mathbf{v}$$

$$\mathbf{n} \times \mathbf{v} = (\mathbf{n}\times)\mathbf{v}$$

Rodrigues' Rotation Formula

Rotation by angle α around axis $\mathbf{n}$

$$\mathbf{R}(\mathbf{n}, \alpha) = \cos(\alpha) \mathbf{I} + (1 - \cos(\alpha)) \mathbf{n}\mathbf{n}^T + \sin(\alpha) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathbf{N}}$$

How to prove this magic formula?

- Matrix $\mathbf{N}$ computes a cross-product:
- Assume orthonormal system $\mathbf{e}_1, \mathbf{e}_2, \mathbf{n}$

$$\begin{aligned}\mathbf{R}\mathbf{n} &= \mathbf{n} \\ \mathbf{R}\mathbf{e}_1 &= \cos \alpha \mathbf{e}_1 + \sin \alpha \mathbf{e}_2 \\ \mathbf{R}\mathbf{e}_2 &= -\sin \alpha \mathbf{e}_1 + \cos \alpha \mathbf{e}_2\end{aligned}$$

Turn cross-product into
matrix multiplication!

$$\mathbf{n} \times \mathbf{v} = \mathbf{N}\mathbf{v}$$

Alternative notation

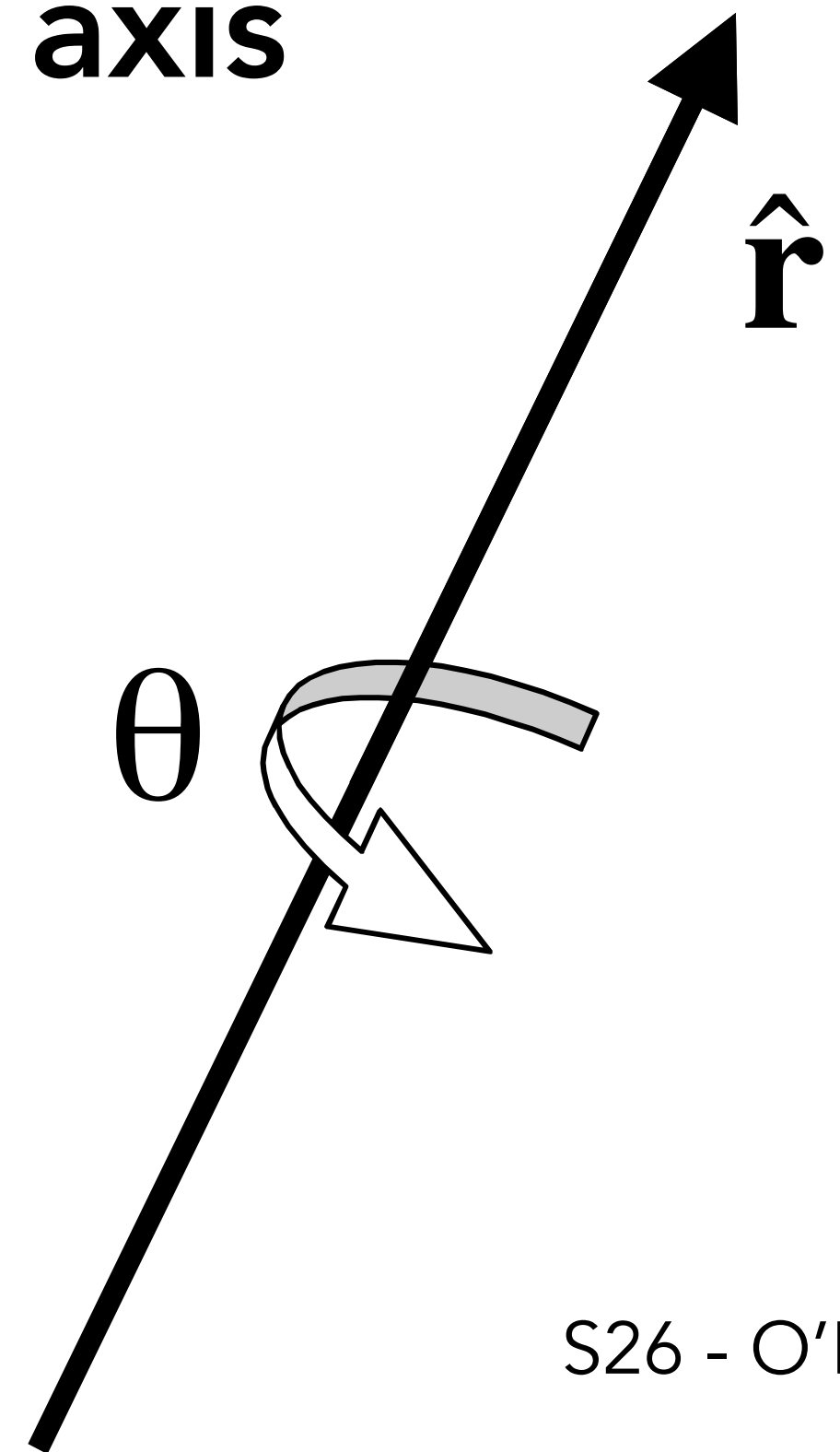
$$\mathbf{n} \times \mathbf{v} = [\mathbf{n}]\mathbf{v}$$

$$\mathbf{n} \times \mathbf{v} = (\mathbf{n}\times)\mathbf{v}$$

Exponential Maps

- Direct representation of arbitrary rotation
- AKA: *axis-angle*, angular displacement vector
- Rotate θ degrees about some axis
- Encode θ by length of vector

$$\theta = |\mathbf{r}|$$

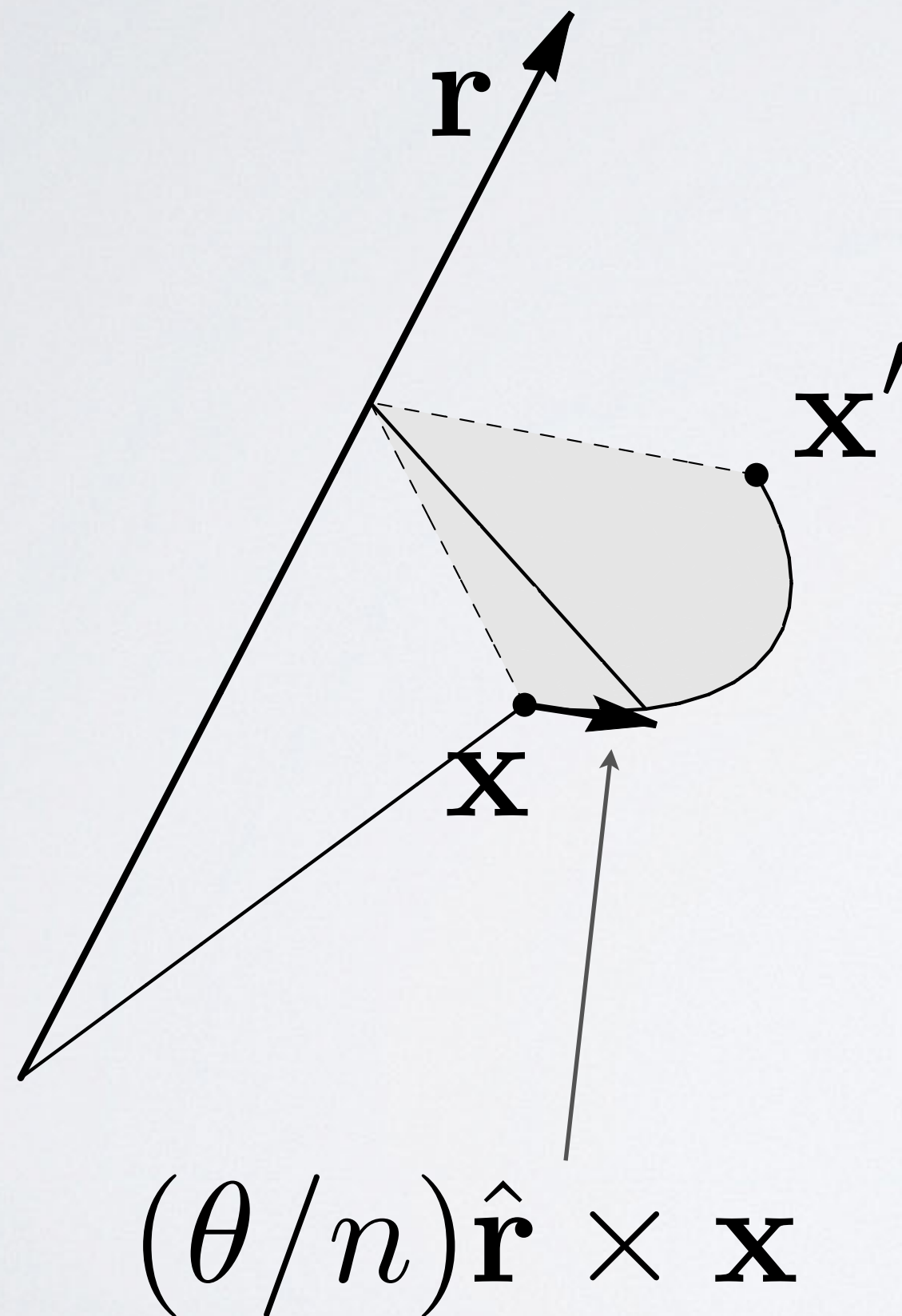


Exponential Maps

- Allows tumbling
- No gimbal-lock!
- Orientations are space within π -radius ball
- Nearly unique representation
- Singularities on shells at 2π
- Nice for interpolation

Exponential Maps

- Why exponential?



- Instead of rotating once by θ , let's do n small rotations of θ/n
- Now the angle is small, so the rotated $\mathbf{x}$ is approximately

$$\mathbf{x} + (\theta/n)\hat{\mathbf{r}} \times \mathbf{x}$$

$$= \left(\mathbf{I} + \frac{(\hat{\mathbf{r}} \times)\theta}{n} \right) \mathbf{x}$$

- Do it n times and you get

$$\mathbf{x}' = \left(\mathbf{I} + \frac{(\hat{\mathbf{r}} \times)\theta}{n} \right)^n \mathbf{x}$$

Exponential Maps

$$\mathbf{x}' = \lim_{n \rightarrow \infty} \left(\mathbf{I} + \frac{(\hat{\mathbf{r}} \times) \theta}{n} \right)^n \mathbf{x}$$

- Remind you of anything?

$$\lim_{n \rightarrow \infty} \left(1 + \frac{a}{n} \right)^n \text{ is a definition of } e^a$$

- So the rotation we want is the exponential of $(\hat{\mathbf{r}} \times) \theta$!

$$\mathbf{x}' = \underbrace{e^{(\hat{\mathbf{r}} \times) \theta}}_{\text{Rotation matrix!}} \mathbf{x}$$

Rotation matrix!

- In fact you can just plug it into the infinite series...

Exponential Maps

- Why exponential?
- Recall series expansion of e^x

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Exponential Maps

- Why exponential?
- Recall series expansion of e^x
- Euler: what happens if you put in $i\theta$ for x

$$e^{i\theta} = 1 + \frac{i\theta}{1!} + \frac{-\theta^2}{2!} + \frac{-i\theta^3}{3!} + \frac{\theta^4}{4!} + \dots$$

$$= \left(1 + \frac{-\theta^2}{2!} + \frac{\theta^4}{4!} + \dots \right) + i \left(\frac{\theta}{1!} + \frac{-\theta^3}{3!} + \dots \right)$$

$$e^{i\theta} = \cos(\theta) + i \sin(\theta)$$

Exponential Maps

- Do same trick with matrices!

$$e^{(\hat{\mathbf{r}} \times) \theta} = \mathbf{I} + \frac{(\hat{\mathbf{r}} \times) \theta}{1!} + \frac{(\hat{\mathbf{r}} \times)^2 \theta^2}{2!} + \frac{(\hat{\mathbf{r}} \times)^3 \theta^3}{3!} + \frac{(\hat{\mathbf{r}} \times)^4 \theta^4}{4!} + \dots$$

But notice that: $(\hat{\mathbf{r}} \times)^3 = -(\hat{\mathbf{r}} \times)$

$$e^{(\hat{\mathbf{r}} \times) \theta} = \mathbf{I} + \frac{(\hat{\mathbf{r}} \times) \theta}{1!} + \frac{(\hat{\mathbf{r}} \times)^2 \theta^2}{2!} + \frac{-(\hat{\mathbf{r}} \times) \theta^3}{3!} + \frac{-(\hat{\mathbf{r}} \times)^2 \theta^4}{4!} + \dots$$

Exponential Maps

$$e^{(\hat{\mathbf{r}} \times) \theta} = \mathbf{I} + \frac{(\hat{\mathbf{r}} \times) \theta}{1!} + \frac{(\hat{\mathbf{r}} \times)^2 \theta^2}{2!} + \frac{-(\hat{\mathbf{r}} \times) \theta^3}{3!} + \frac{-(\hat{\mathbf{r}} \times)^2 \theta^4}{4!} + \dots$$

$$e^{(\hat{\mathbf{r}} \times) \theta} = (\hat{\mathbf{r}} \times) \left(\frac{\theta}{1!} - \frac{\theta^3}{3!} + \dots \right) + \mathbf{I} + (\hat{\mathbf{r}} \times)^2 \left(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \dots \right)$$

$$e^{(\hat{\mathbf{r}} \times) \theta} = (\hat{\mathbf{r}} \times) \sin(\theta) + \mathbf{I} + (\hat{\mathbf{r}} \times)^2 (1 - \cos(\theta))$$

Rotation Matrices

- Eigen system
 - One real eigenvalue
 - Real axis is axis of rotation
 - Imaginary values are 2D rotation as complex number
- Logarithmic formula

$$(\hat{\mathbf{r}} \times) = \ln(\mathbf{R}) = \frac{\theta}{2 \sin \theta} (\mathbf{R} - \mathbf{R}^T)$$

$$\theta = \cos^{-1} \left(\frac{\text{Tr}(\mathbf{R}) - 1}{2} \right)$$

Quaternions

- More popular than exponential maps
- Natural extension of $e^{i\theta} = \cos(\theta) + i\sin(\theta)$
- Due to Hamilton (1843)
 - Interesting bit of drama....

Quaternions

- Three-Times-As-Complex Numbers

$$q = (z_1, z_2, z_3, s) = (\mathbf{z}, s)$$

$$q = iz_1 + jz_2 + kz_3 + s$$

$$i^2 = j^2 = k^2 = -1$$

$$ij = k \quad ji = -k$$

$$jk = i \quad kj = -i$$

$$ki = j \quad ik = -j$$

Quaternions

- Multiplication natural consequence of defn.

$$\mathbf{q} \cdot \mathbf{p} = (\mathbf{z}_q s_p + \mathbf{z}_p s_q + \mathbf{z}_p \times \mathbf{z}_q, s_p s_q - \mathbf{z}_p \cdot \mathbf{z}_q)$$

- Conjugate

$$\mathbf{q}^* = (-\mathbf{z}, s)$$

- Magnitude

$$||\mathbf{q}||^2 = \mathbf{z} \cdot \mathbf{z} + s^2 = \mathbf{q} \cdot \mathbf{q}^*$$

Quaternions

- Vectors as quaternions

$$\mathbf{v} = (\mathbf{v}, 0)$$

- Rotations as quaternions

$$\mathbf{r} = (\hat{\mathbf{r}} \sin \frac{\theta}{2}, \cos \frac{\theta}{2})$$

- Rotating a vector

$$\mathbf{x}' = \mathbf{r} \cdot \mathbf{x} \cdot \mathbf{r}^*$$

- Composing rotations

$$\mathbf{r} = \mathbf{r}_1 \cdot \mathbf{r}_2$$



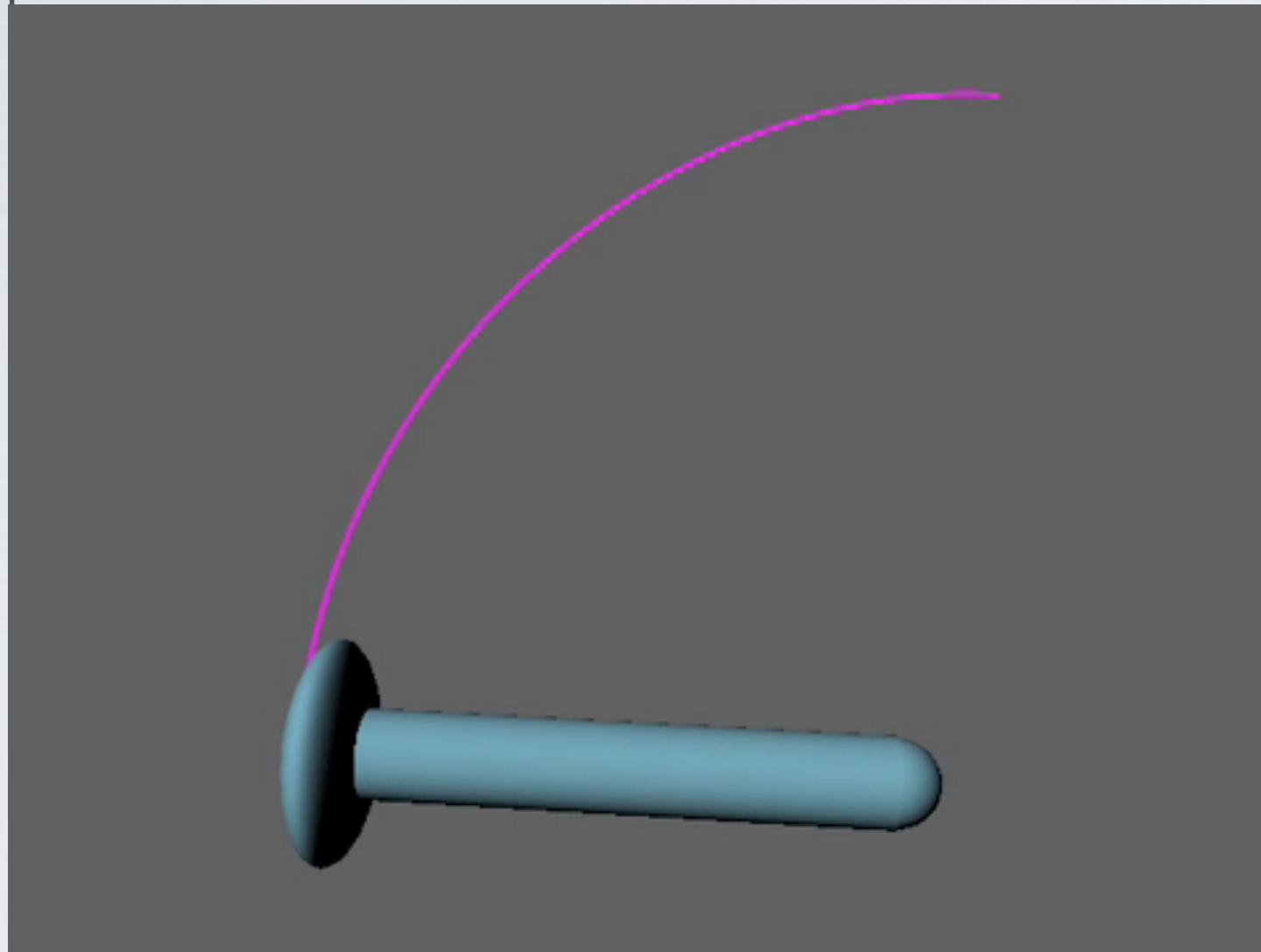
Compare to Exp. Map

Quaternions

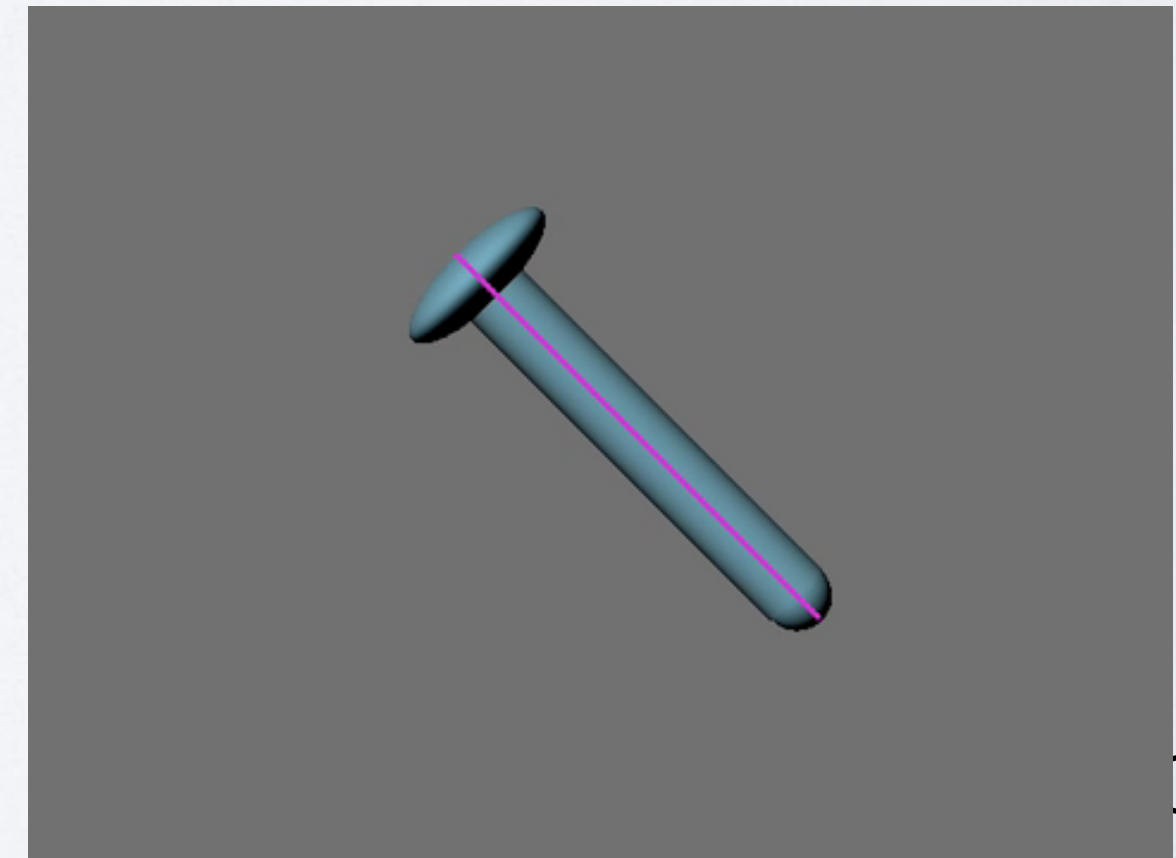
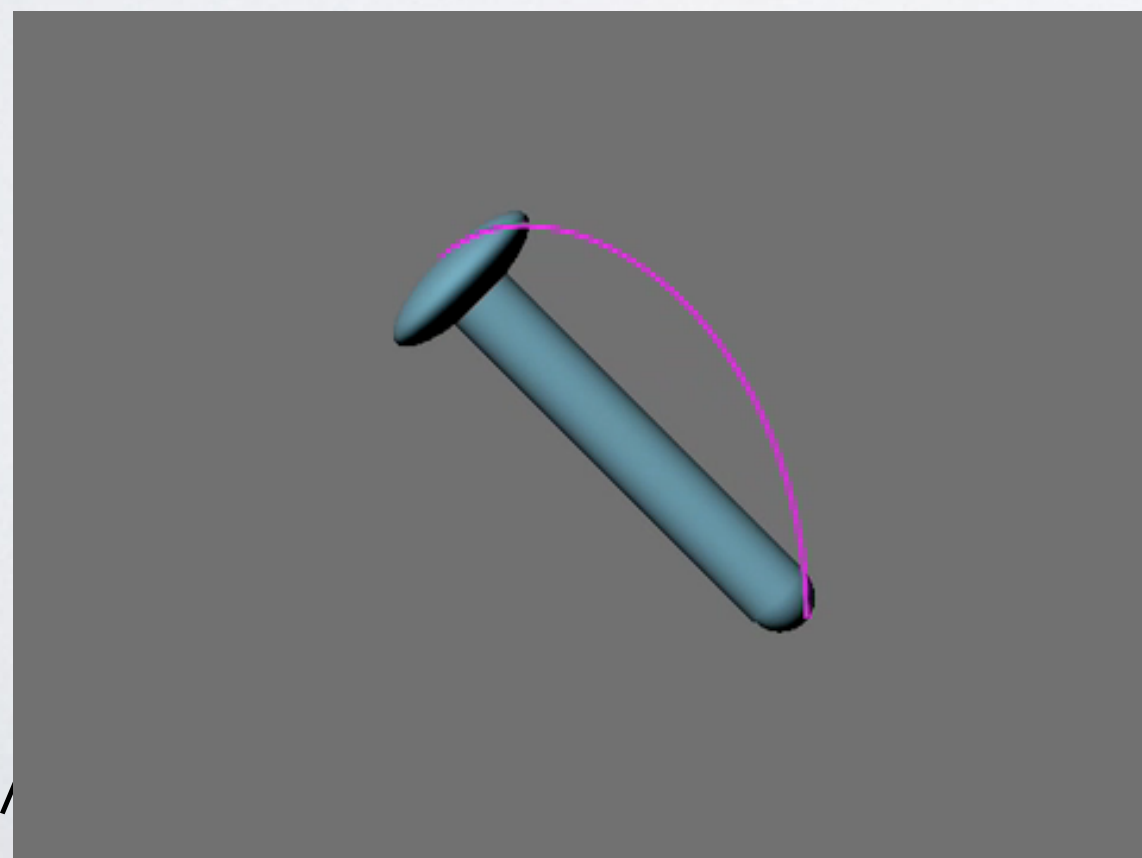
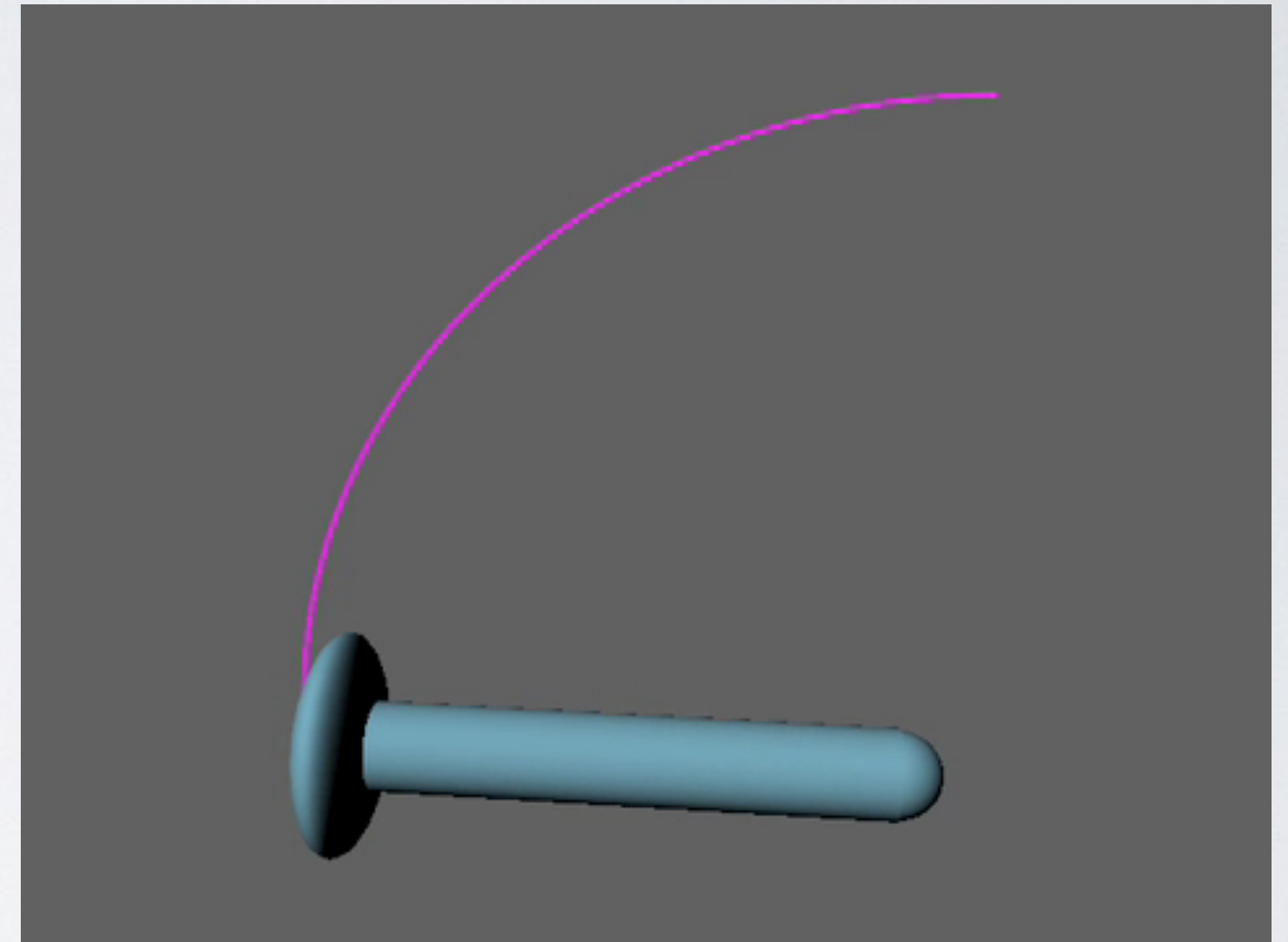
- No tumbling
- No gimbal-lock
- Orientations are “double unique”
- Surface of a 3-sphere in 4D $||r|| = 1$
- Nice for interpolation

Interpolation

Euler angles

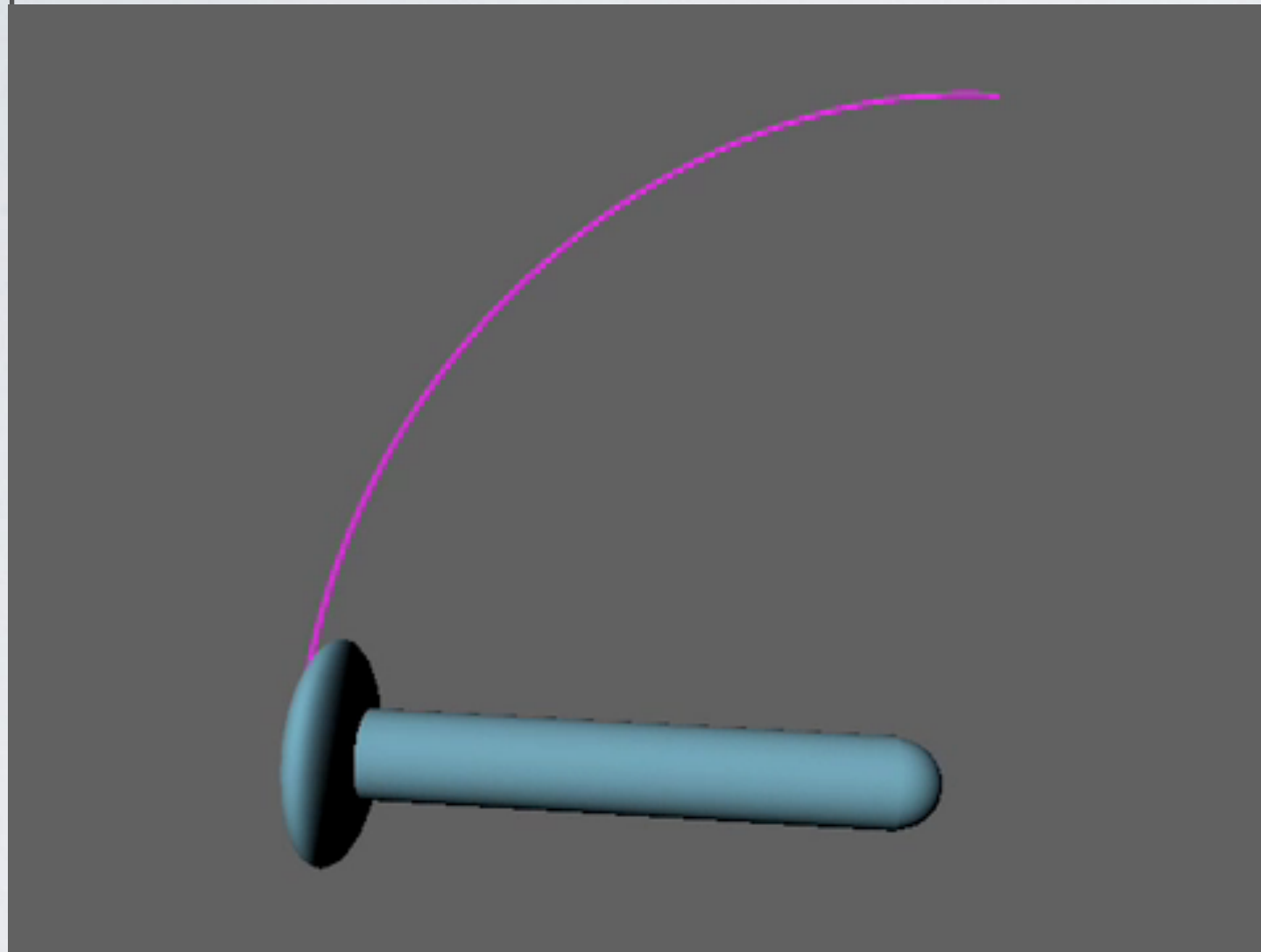


Exponential maps or quaternions

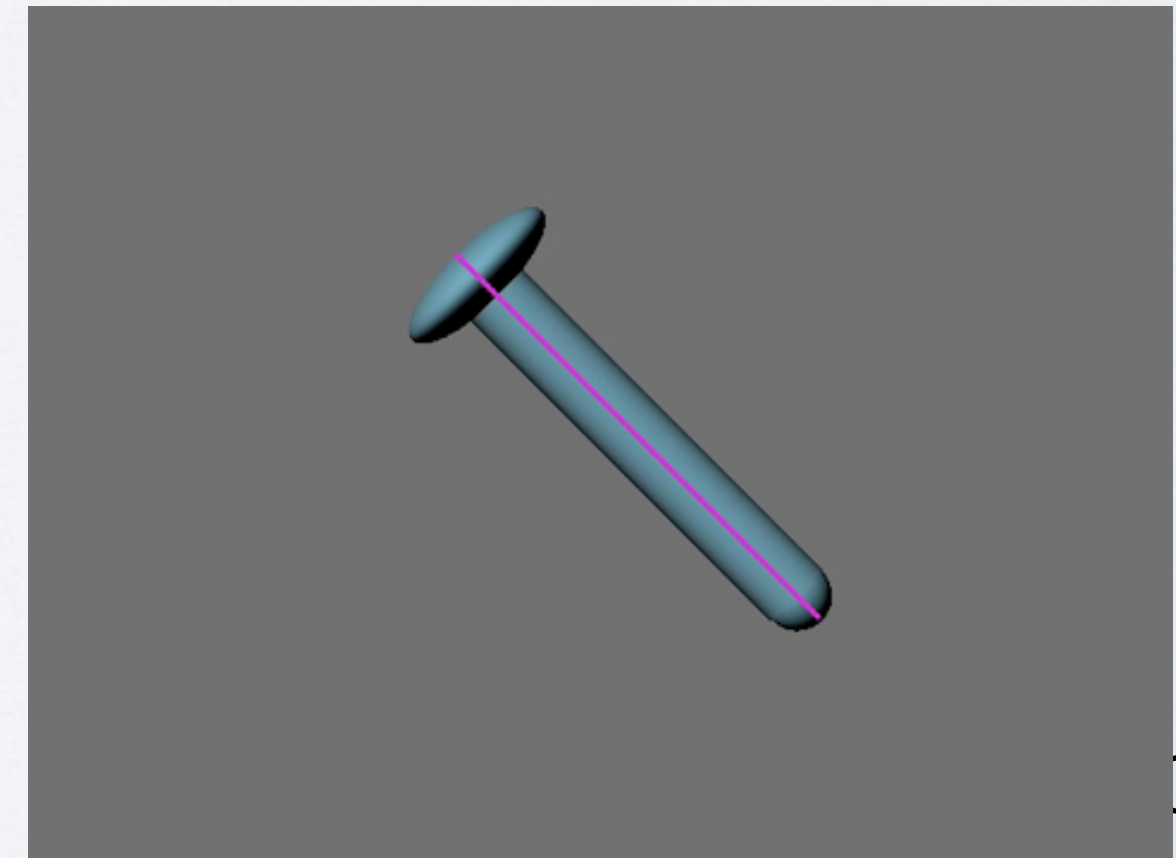
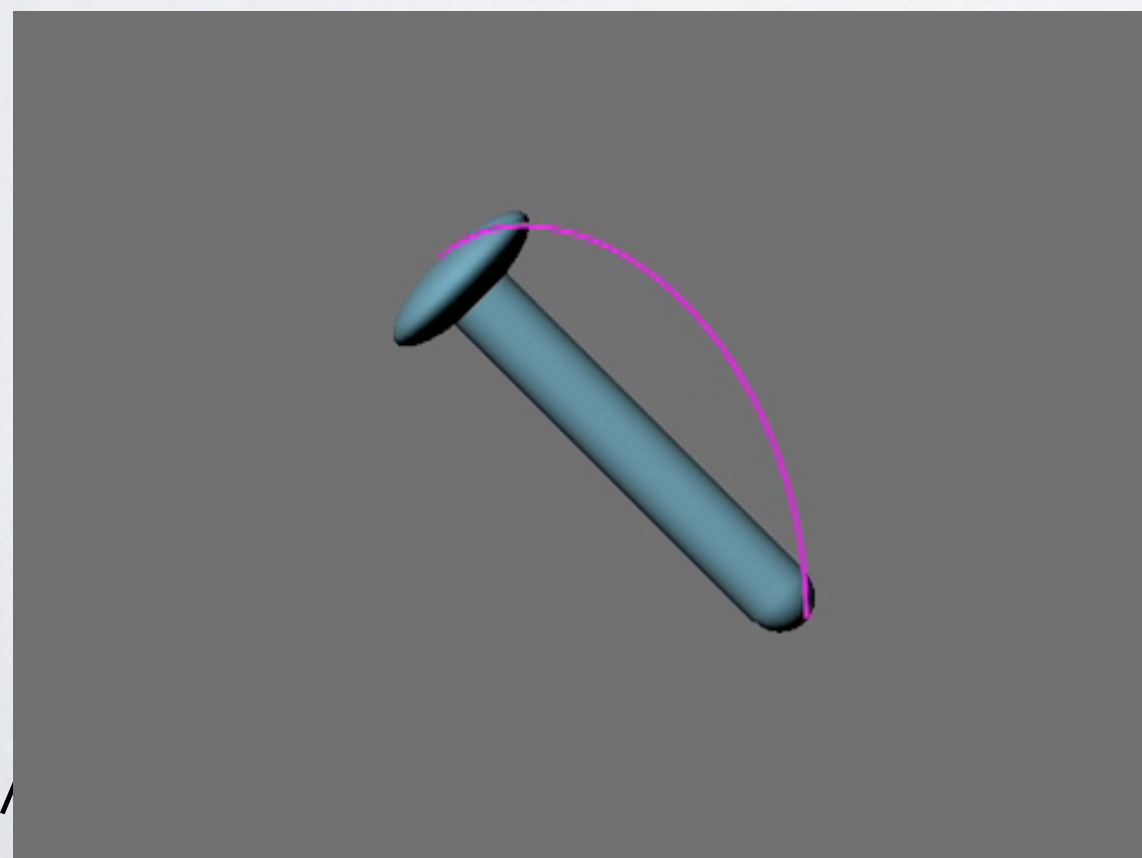
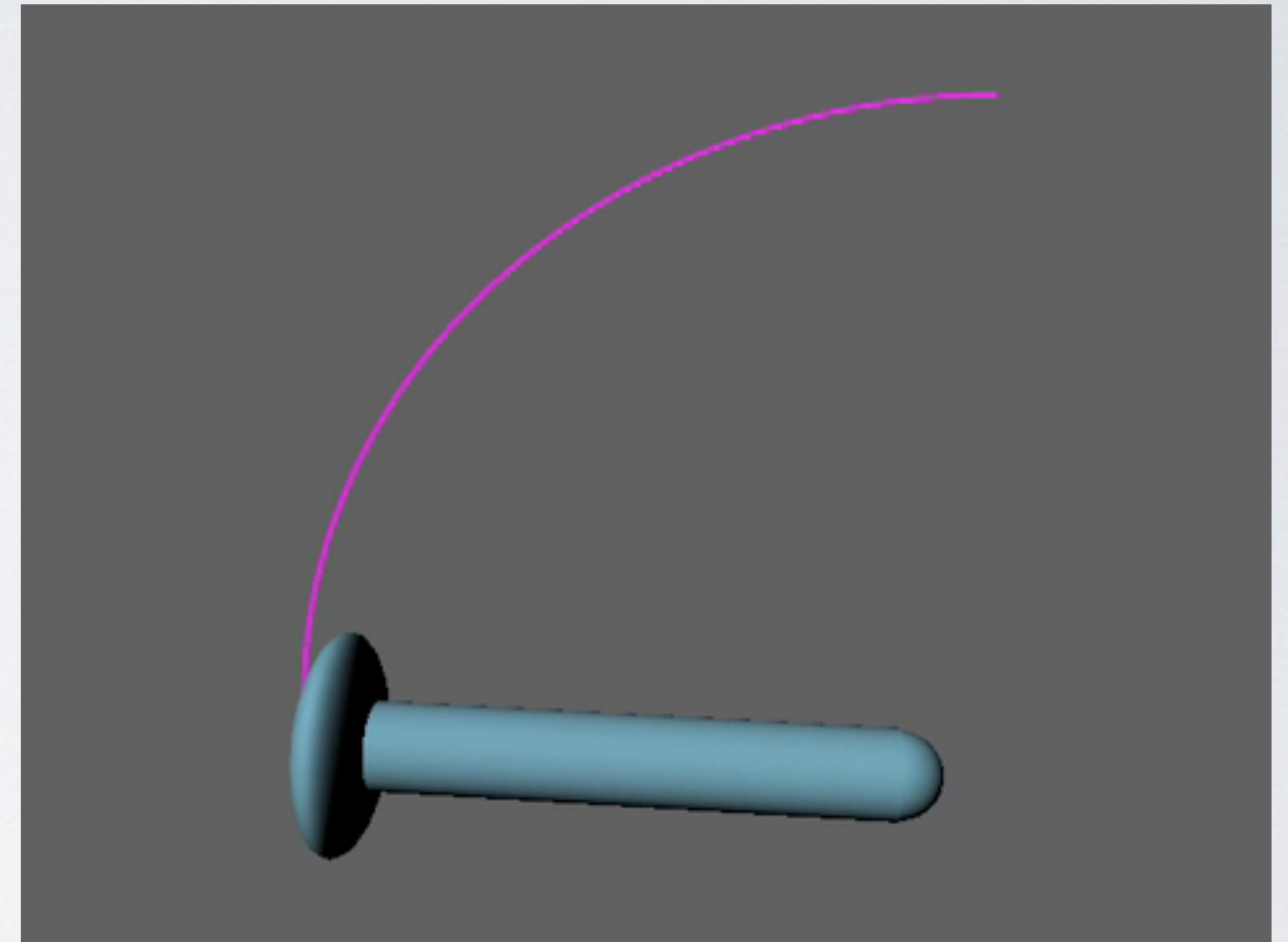


Interpolation

Euler angles



Exponential maps or quaternions



Hierarchical Transforms

Skeleton - Linear Representation

head

torso

right upper arm

right lower arm

right hand

left upper arm

left lower arm

left hand

right upper leg

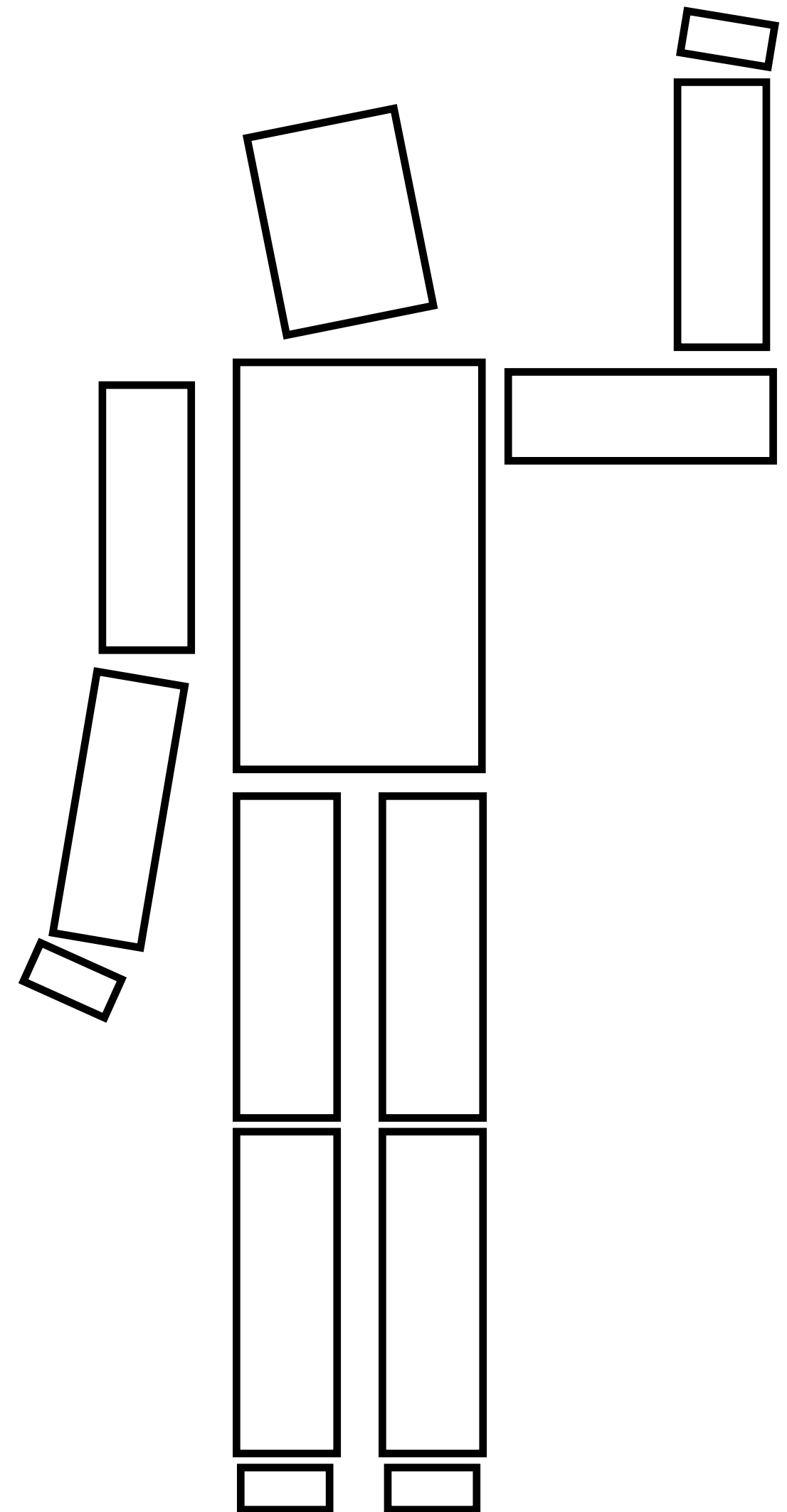
right lower leg

right foot

left upper leg

left lower leg

left foot



Linear Representation

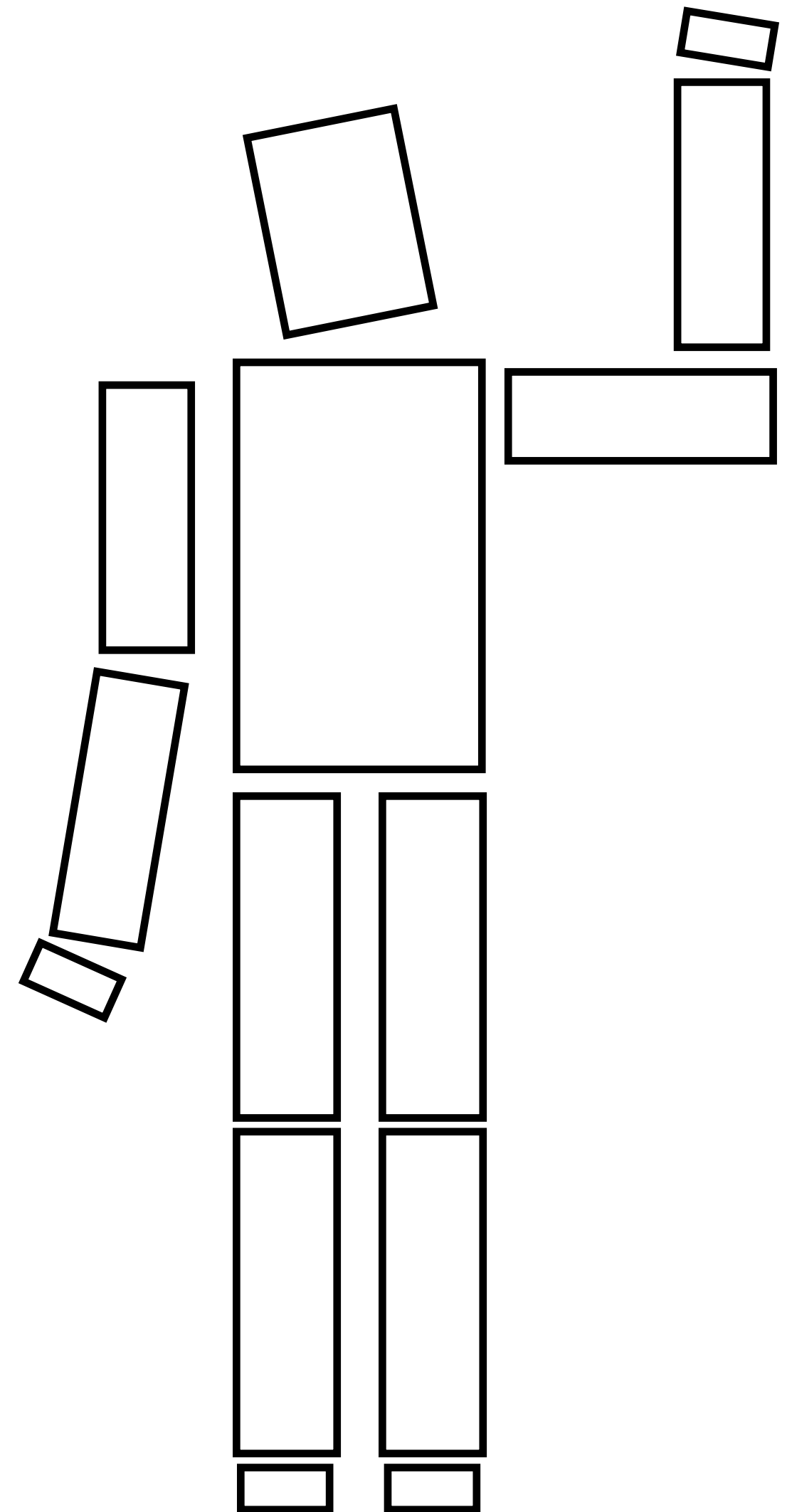
Each shape associated with its own transform

A single edit can require updating many transforms

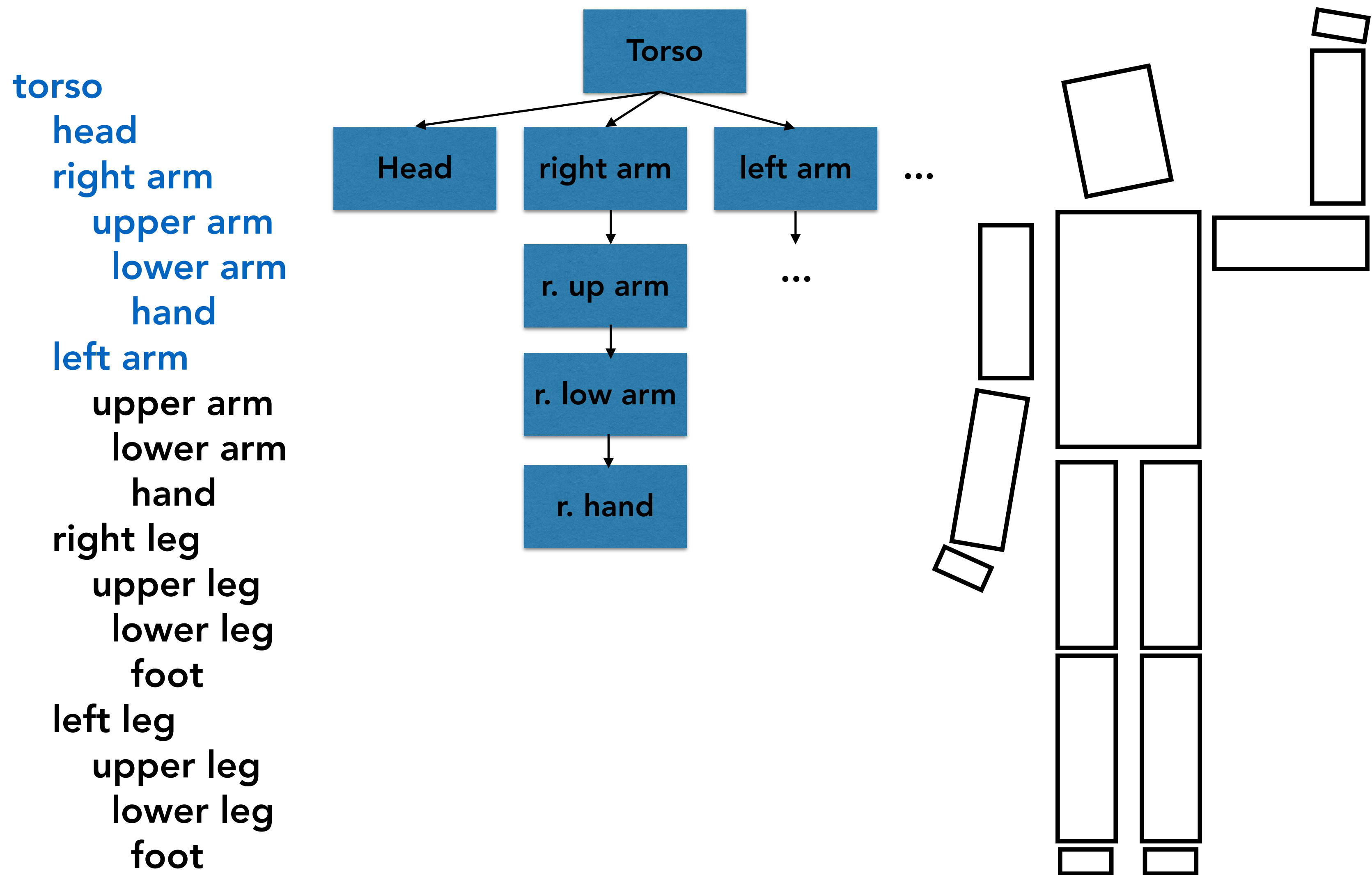
- E.g. raising arm requires updating transforms for all arm parts

Skeleton - Hierarchical Representation

torso
head
right arm
 upper arm
 lower arm
 hand
left arm
 upper arm
 lower arm
 hand
right leg
 upper leg
 lower leg
 foot
left leg
 upper leg
 lower leg
 foot



Skeleton - Hierarchical Representation



Hierarchical Representation

Hierarchical Representation

Grouped representation (tree)

Hierarchical Representation

Grouped representation (tree)

- Each group contains subgroups and/or shapes

Hierarchical Representation

Grouped representation (tree)

- Each group contains subgroups and/or shapes
- Each group is associated with a transform relative to parent group

Hierarchical Representation

Grouped representation (tree)

- Each group contains subgroups and/or shapes
- Each group is associated with a transform relative to parent group
- Transform on leaf-node shape is concatenation of all transforms on path from root node to leaf

Hierarchical Representation

Grouped representation (tree)

- Each group contains subgroups and/or shapes
- Each group is associated with a transform relative to parent group
- Transform on leaf-node shape is concatenation of all transforms on path from root node to leaf
- Changing a group's transform affects all parts

Hierarchical Representation

Grouped representation (tree)

- Each group contains subgroups and/or shapes
- Each group is associated with a transform relative to parent group
- Transform on leaf-node shape is concatenation of all transforms on path from root node to leaf
- Changing a group's transform affects all parts
 - Allows high level editing by changing only one node

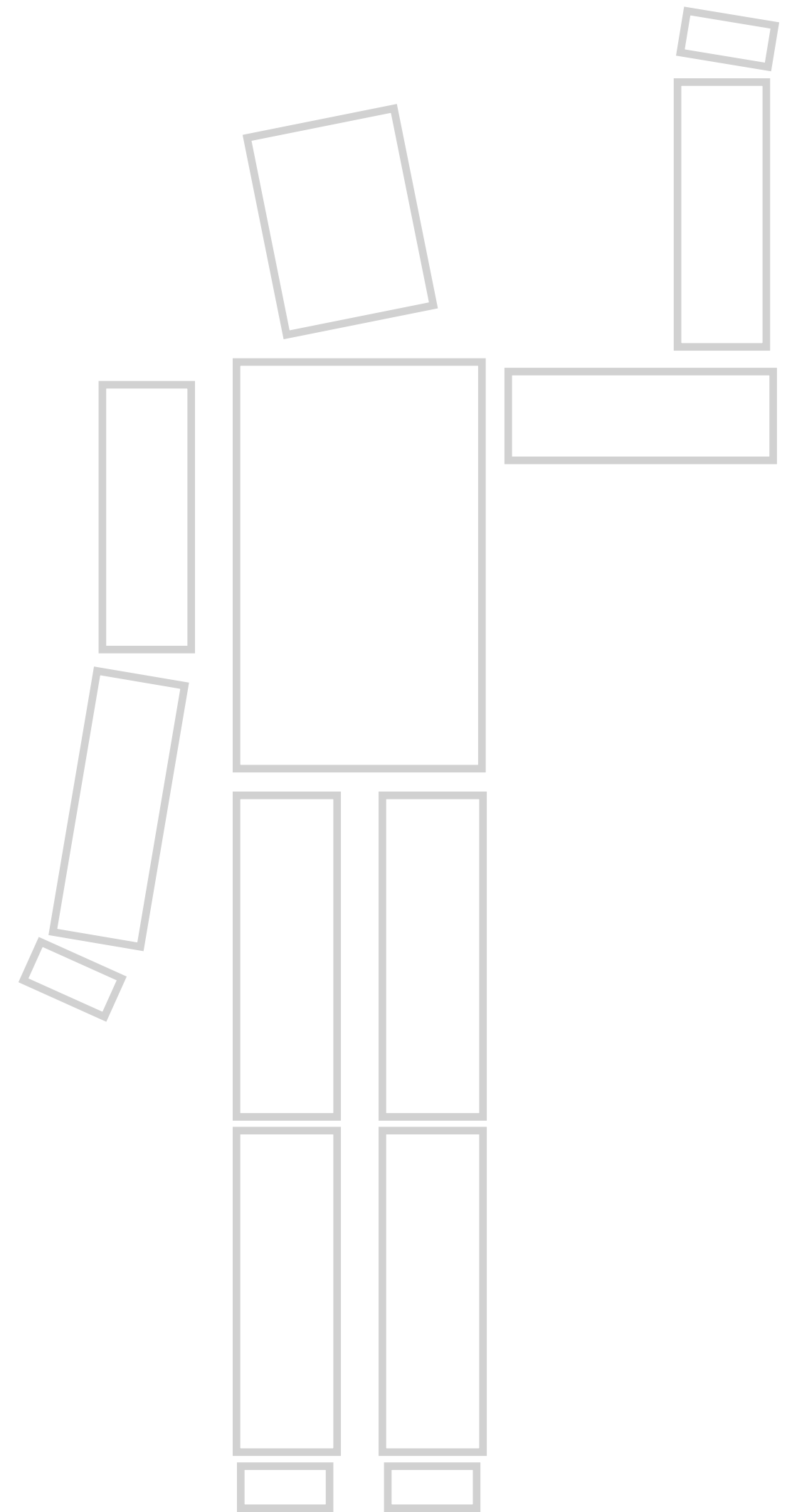
Hierarchical Representation

Grouped representation (tree)

- Each group contains subgroups and/or shapes
- Each group is associated with a transform relative to parent group
- Transform on leaf-node shape is concatenation of all transforms on path from root node to leaf
- Changing a group's transform affects all parts
 - Allows high level editing by changing only one node
 - E.g. raising left arm requires changing only one transform for that group

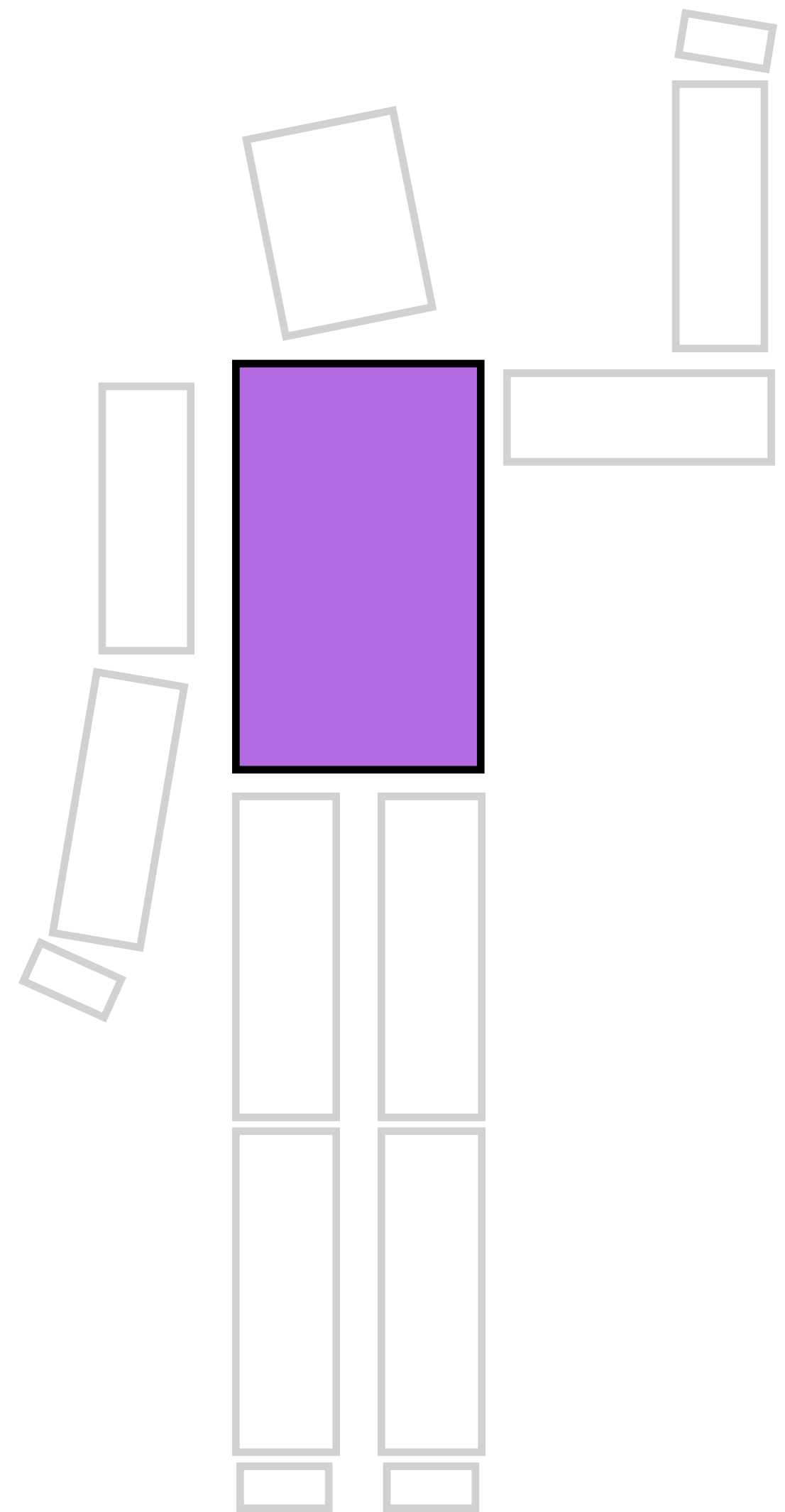
Skeleton - Hierarchical Representation

```
translate(0, 10);  
drawTorso();
```



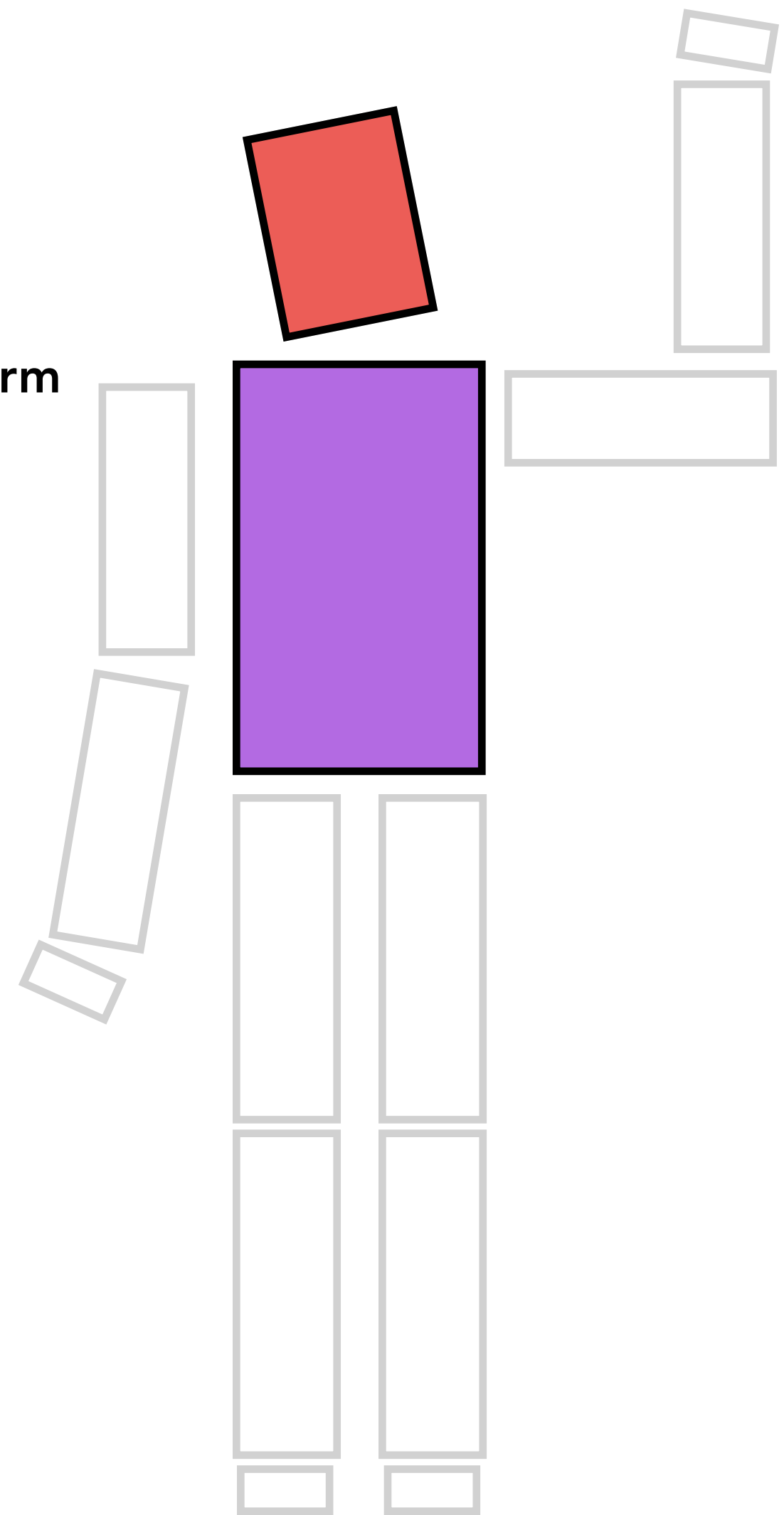
Skeleton - Hierarchical Representation

```
translate(0, 10);  
drawTorso();
```



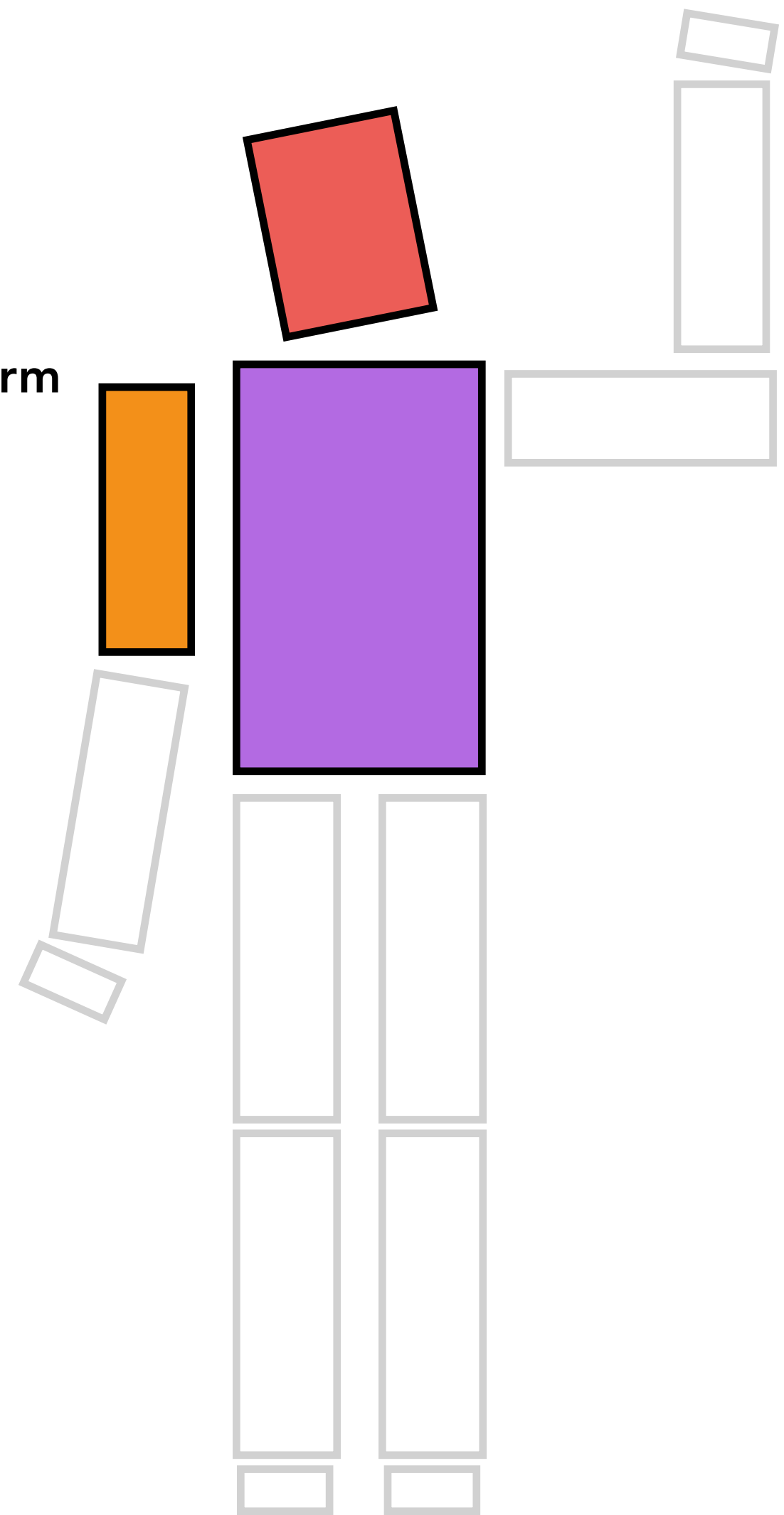
Skeleton - Hierarchical Representation

```
translate(0, 10);  
drawTorso();  
  pushmatrix(); // push a copy of transform onto stack  
  translate(0, 5); // right-multiply onto current transform  
  rotate(headRotation); // right-multiply onto current transform  
  drawHead();  
popmatrix(); // pop current transform off stack
```



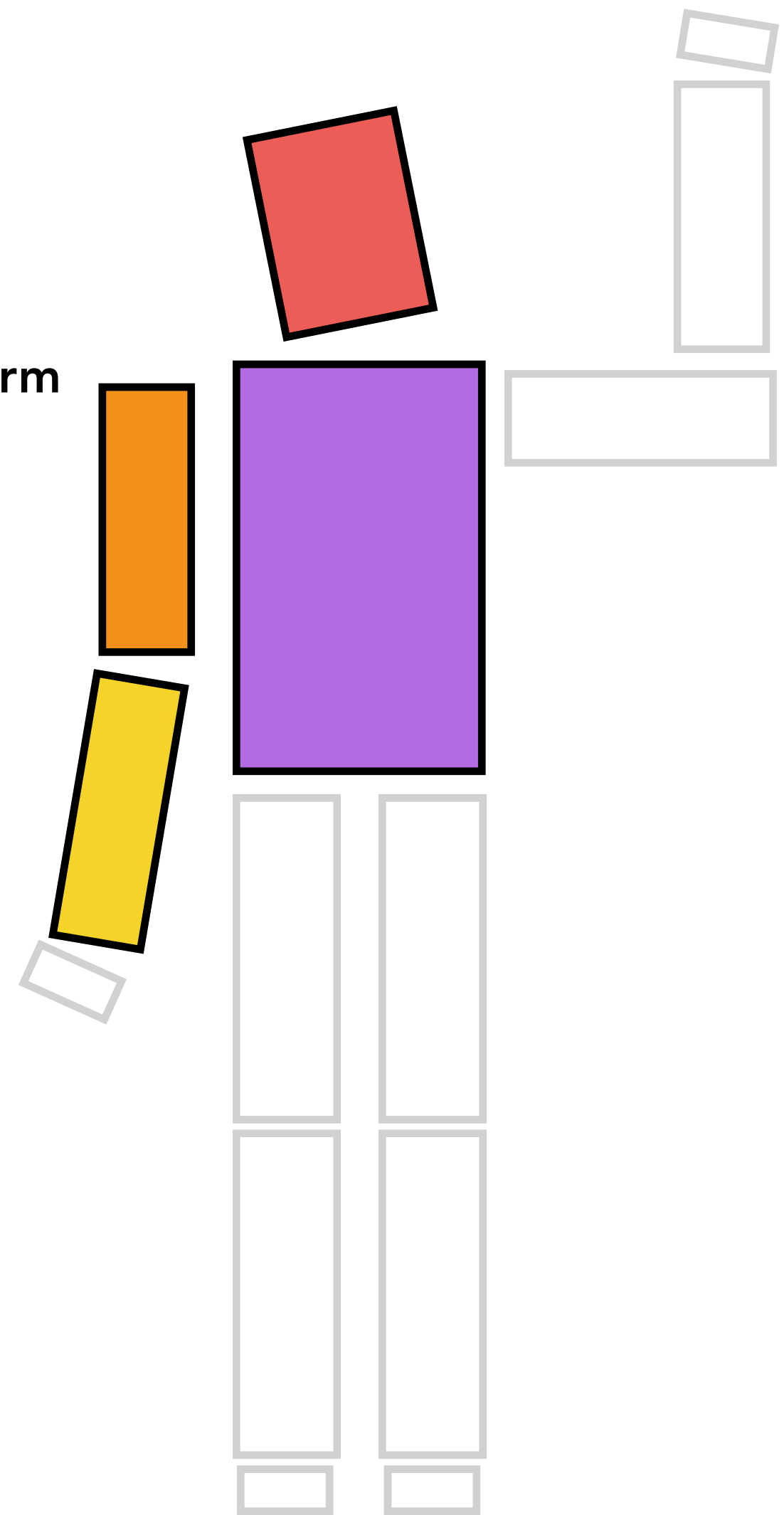
Skeleton - Hierarchical Representation

```
translate(0, 10);  
drawTorso();  
  pushmatrix(); // push a copy of transform onto stack  
  translate(0, 5); // right-multiply onto current transform  
  rotate(headRotation); // right-multiply onto current transform  
  drawHead();  
  popmatrix(); // pop current transform off stack  
  pushmatrix();  
  translate(-2, 3);  
  rotate(rightShoulderRotation);  
  drawUpperArm();
```



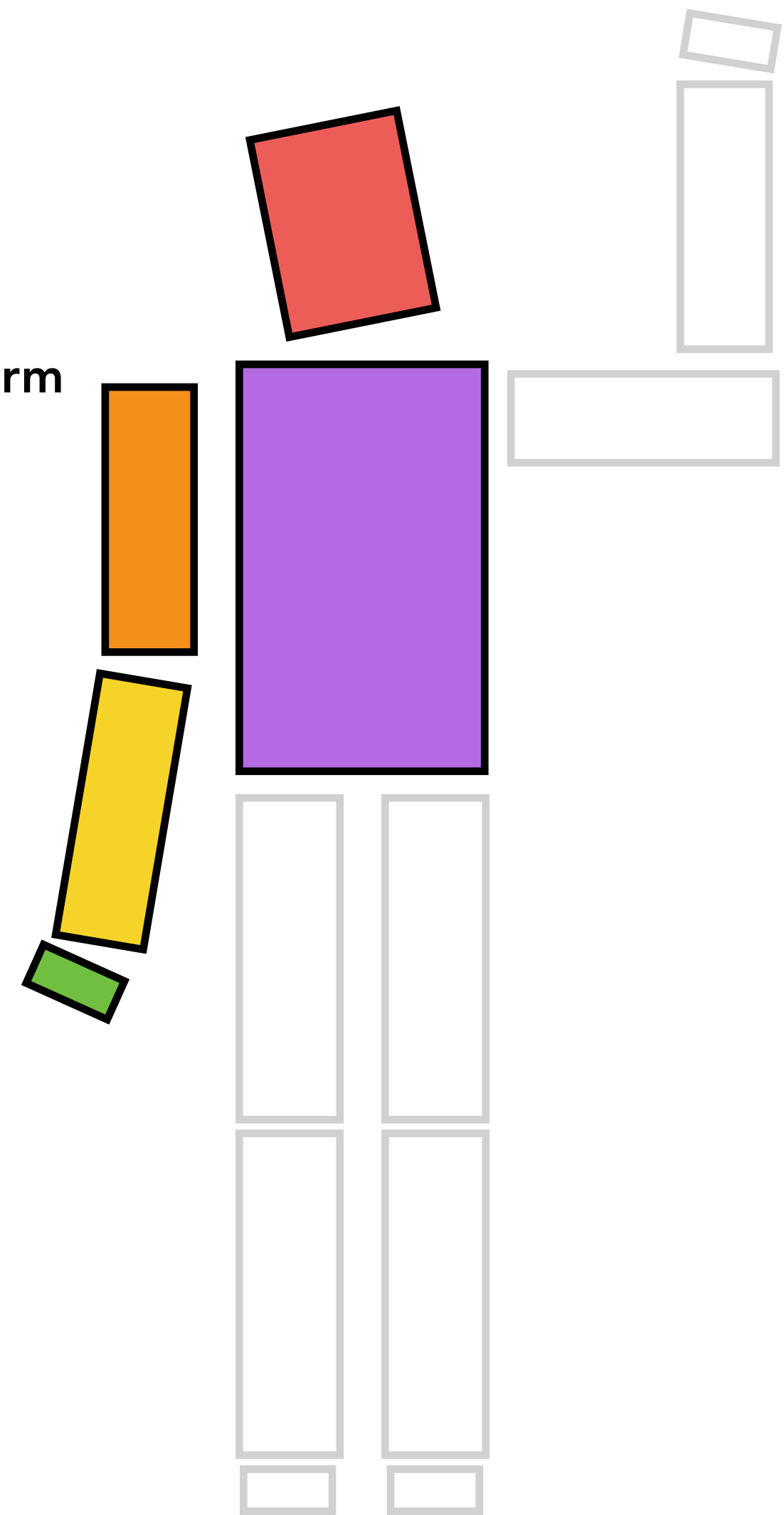
Skeleton - Hierarchical Representation

```
translate(0, 10);  
drawTorso();  
  pushmatrix(); // push a copy of transform onto stack  
  translate(0, 5); // right-multiply onto current transform  
  rotate(headRotation); // right-multiply onto current transform  
  drawHead();  
  popmatrix(); // pop current transform off stack  
  pushmatrix();  
  translate(-2, 3);  
  rotate(rightShoulderRotation);  
  drawUpperArm();  
  pushmatrix();  
  translate(0, -3);  
  rotate(elbowRotation);  
  drawLowerArm();
```



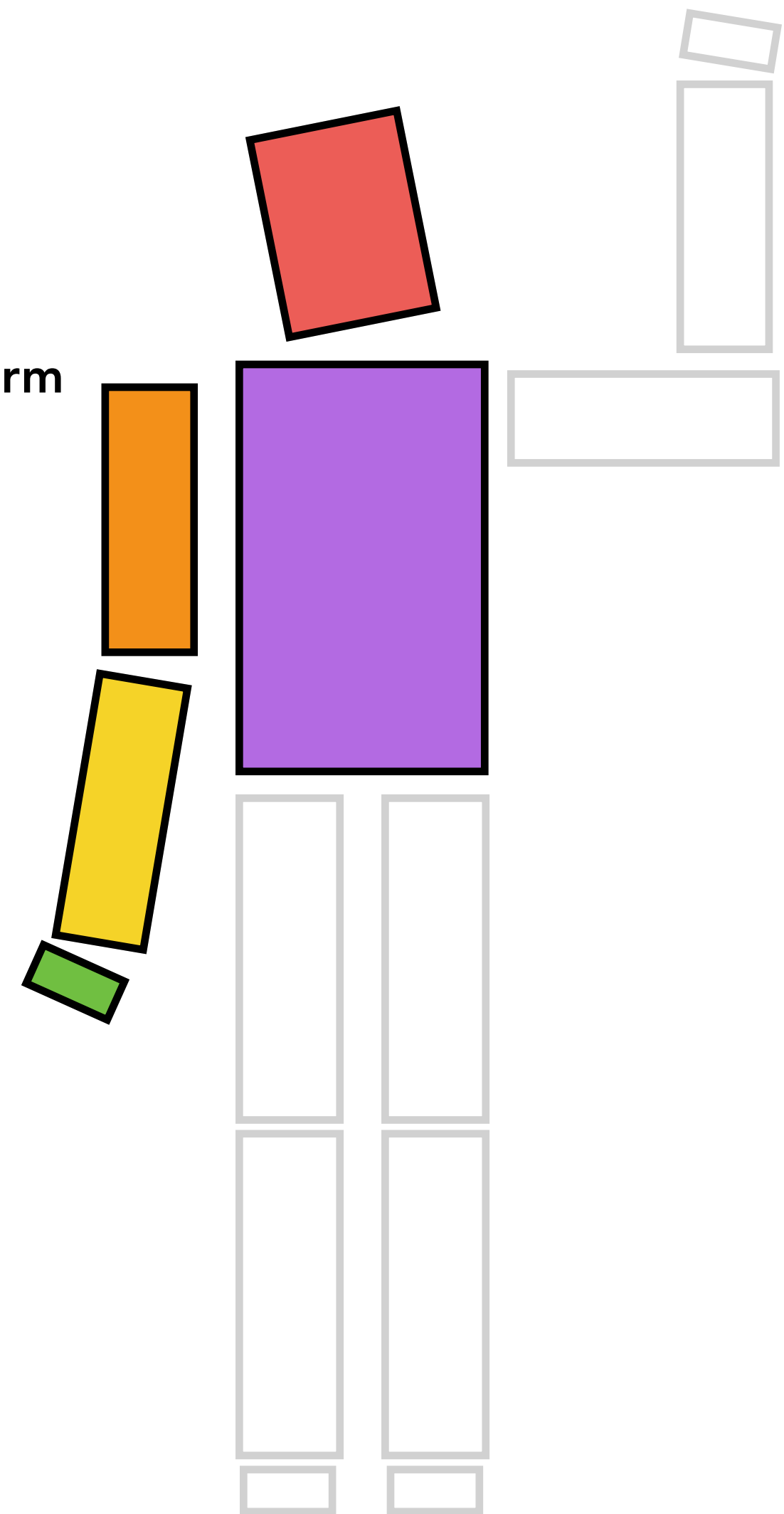
Skeleton - Hierarchical Representation

```
translate(0, 10);
drawTorso();
  pushmatrix(); // push a copy of transform onto stack
  translate(0, 5); // right-multiply onto current transform
  rotate(headRotation); // right-multiply onto current transform
  drawHead();
popmatrix(); // pop current transform off stack
pushmatrix();
  translate(-2, 3);
  rotate(rightShoulderRotation);
  drawUpperArm();
  pushmatrix();
    translate(0, -3);
    rotate(elbowRotation);
    drawLowerArm();
    pushmatrix();
      translate(0, -3);
      rotate(wristRotation);
      drawHand();
    popmatrix();
```



Skeleton - Hierarchical Representation

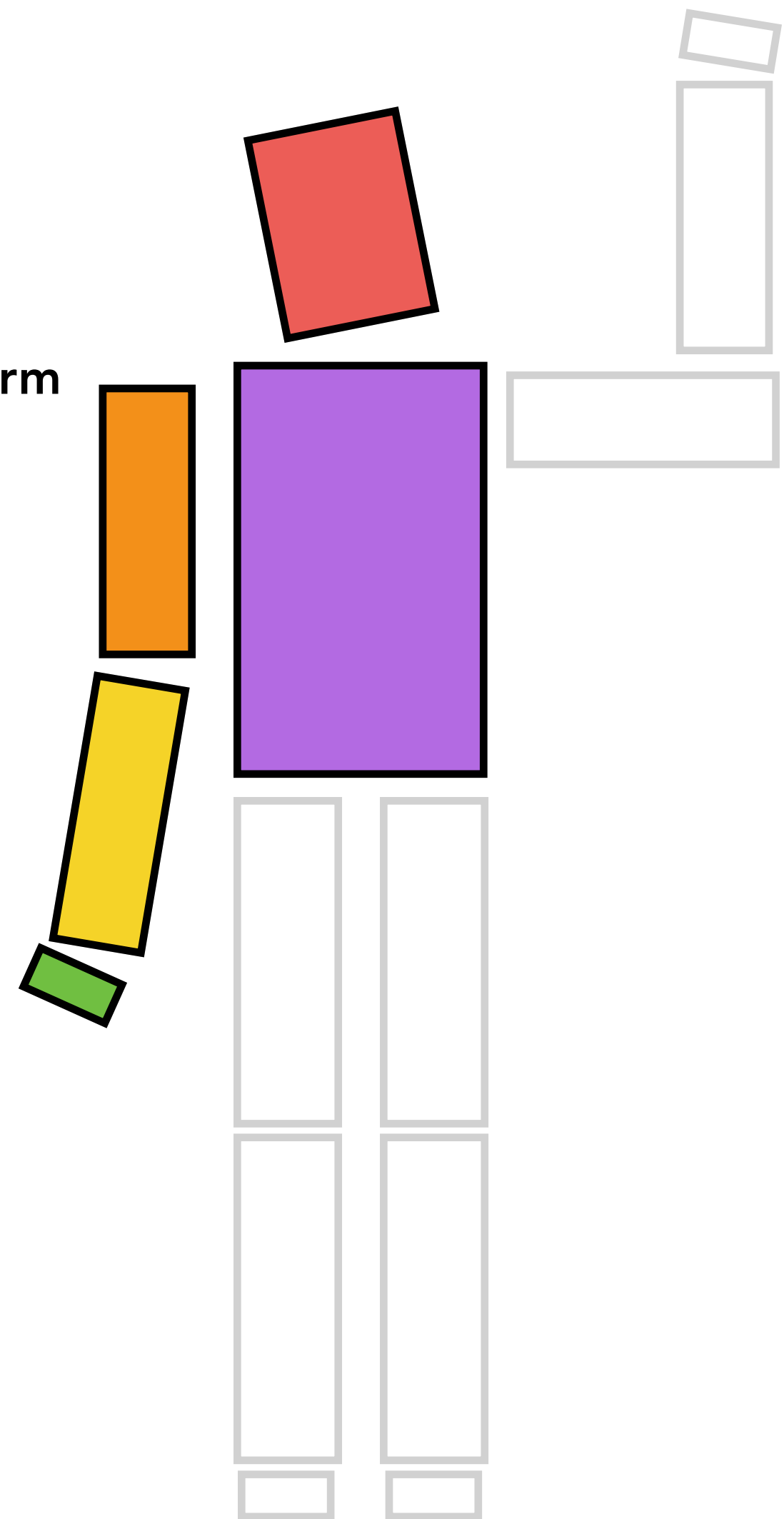
```
translate(0, 10);
drawTorso();
  pushmatrix(); // push a copy of transform onto stack
    translate(0, 5); // right-multiply onto current transform
    rotate(headRotation); // right-multiply onto current transform
    drawHead();
  popmatrix(); // pop current transform off stack
  pushmatrix();
    translate(-2, 3);
    rotate(rightShoulderRotation);
    drawUpperArm();
    pushmatrix();
      translate(0, -3);
      rotate(elbowRotation);
      drawLowerArm();
    popmatrix();
      translate(0, -3);
      rotate(wristRotation);
      drawHand();
    popmatrix();
  popmatrix();
....
```



Skeleton - Hierarchical Representation

```
translate(0, 10);
drawTorso();
  pushmatrix(); // push a copy of transform onto stack
    translate(0, 5); // right-multiply onto current transform
    rotate(headRotation); // right-multiply onto current transform
    drawHead();
  popmatrix(); // pop current transform off stack
  pushmatrix();
    translate(-2, 3);
    rotate(rightShoulderRotation);
    drawUpperArm();
    pushmatrix();
      translate(0, -3);
      rotate(elbowRotation);
      drawLowerArm();
      pushmatrix(); -----
        translate(0, -3);
        rotate(wristRotation);
        drawHand();
      popmatrix(); -----
    popmatrix();
  popmatrix();
....
```

right
hand

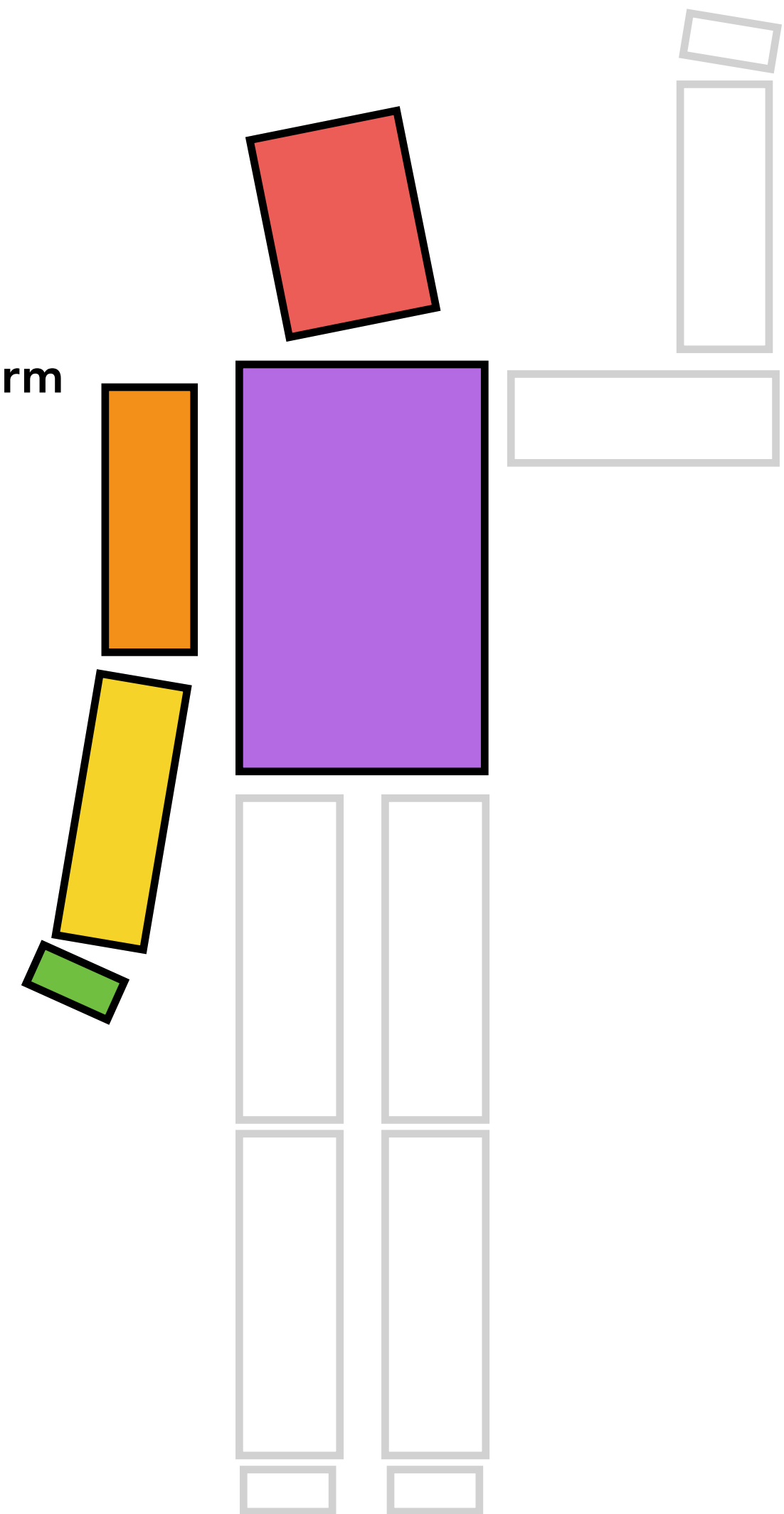


Skeleton - Hierarchical Representation

```
translate(0, 10);
drawTorso();
pushmatrix(); // push a copy of transform onto stack
  translate(0, 5); // right-multiply onto current transform
  rotate(headRotation); // right-multiply onto current transform
  drawHead();
popmatrix(); // pop current transform off stack
pushmatrix();
  translate(-2, 3);
  rotate(rightShoulderRotation);
  drawUpperArm();
  pushmatrix(); -----
    translate(0, -3);
    rotate(elbowRotation);
    drawLowerArm();
    pushmatrix(); -----
      translate(0, -3);
      rotate(wristRotation);
      drawHand();
      popmatrix(); -----
    popmatrix(); -----
  popmatrix();
....
```

right
hand

right
lower
arm
group



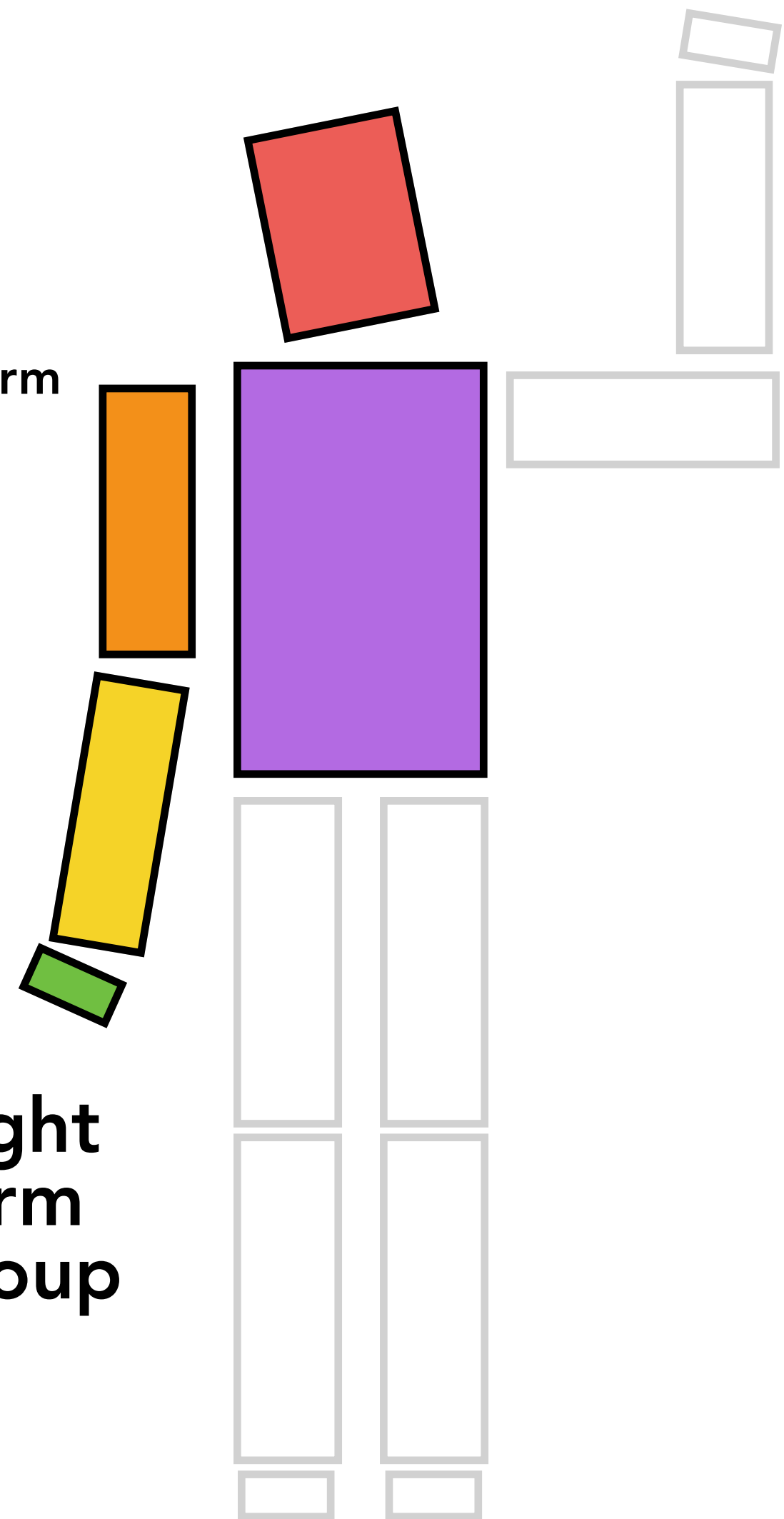
Skeleton - Hierarchical Representation

```
translate(0, 10);
drawTorso();
pushmatrix(); // push a copy of transform onto stack
  translate(0, 5); // right-multiply onto current transform
  rotate(headRotation); // right-multiply onto current transform
  drawHead();
popmatrix(); // pop current transform off stack
pushmatrix(); -----
  translate(-2, 3);
  rotate(rightShoulderRotation);
  drawUpperArm();
  pushmatrix(); -----
    translate(0, -3);
    rotate(elbowRotation);
    drawLowerArm();
    pushmatrix(); -----
      translate(0, -3);
      rotate(wristRotation);
      drawHand();
      popmatrix(); -----
    popmatrix(); -----
  popmatrix(); -----
....
```

right
hand

right
lower
arm
group

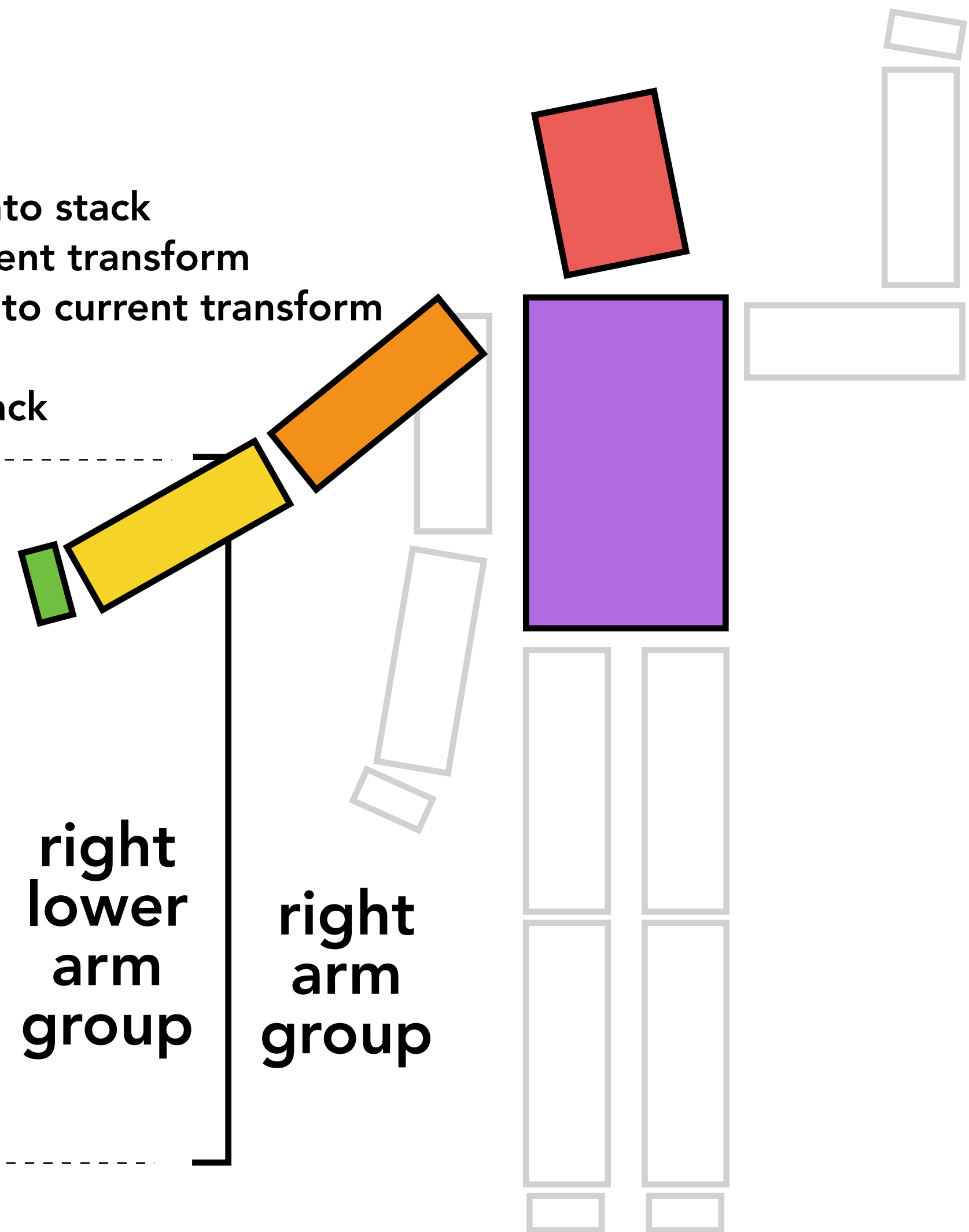
right
arm
group

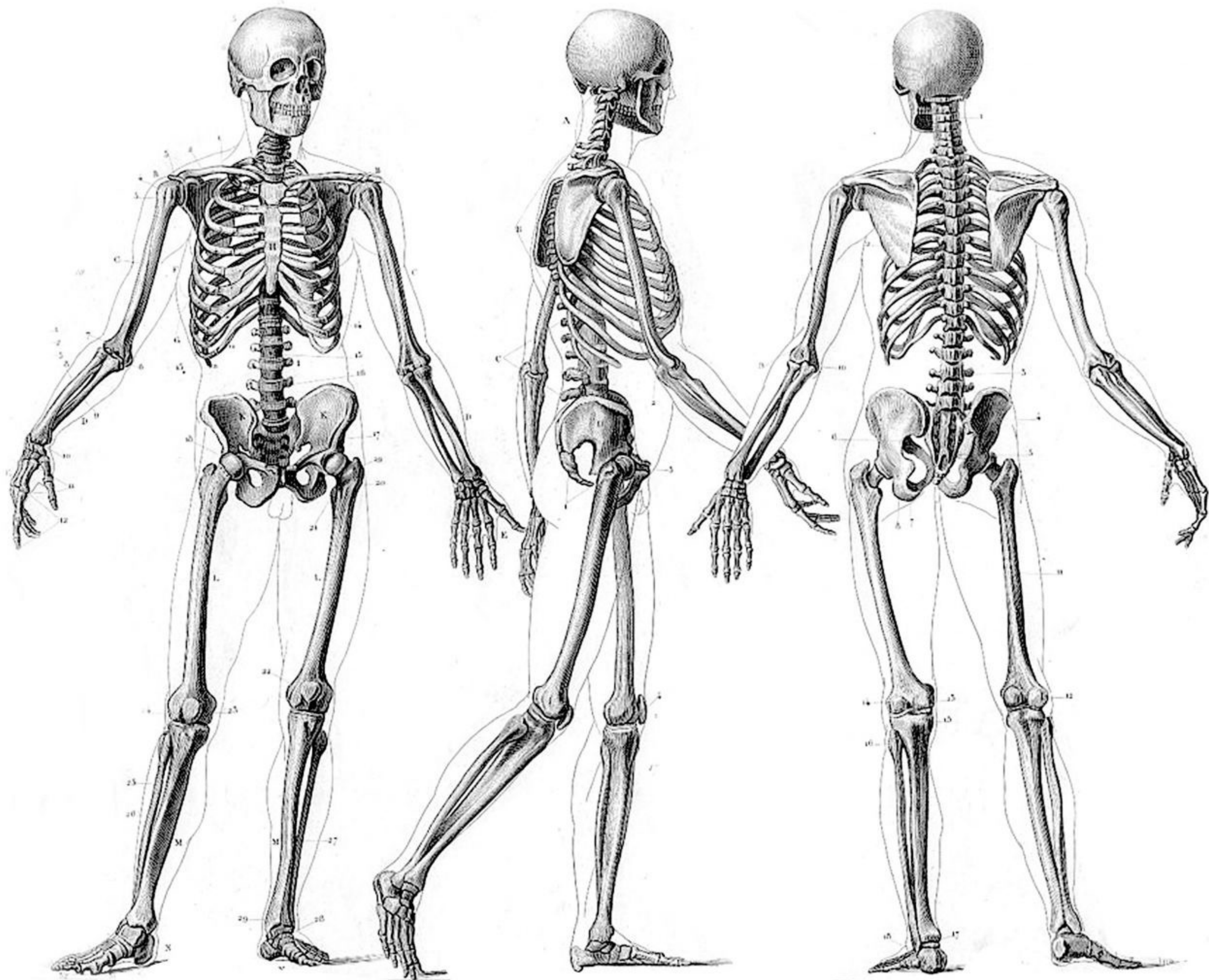


Skeleton - Hierarchical Representation

```
translate(0, 10);
drawTorso();
pushmatrix(); // push a copy of transform onto stack
  translate(0, 5); // right-multiply onto current transform
  rotate(headRotation); // right-multiply onto current transform
  drawHead();
popmatrix(); // pop current transform off stack
pushmatrix(); -----
  translate(-2, 3);
  rotate(rightShoulderRotation);
  drawUpperArm();
  pushmatrix(); -----
    translate(0, -3);
    rotate(elbowRotation);
    drawLowerArm();
    pushmatrix(); -----
      translate(0, -3);
      rotate(wristRotation);
      drawHand();
      popmatrix(); -----
    popmatrix(); -----
  popmatrix(); -----
....
```

→

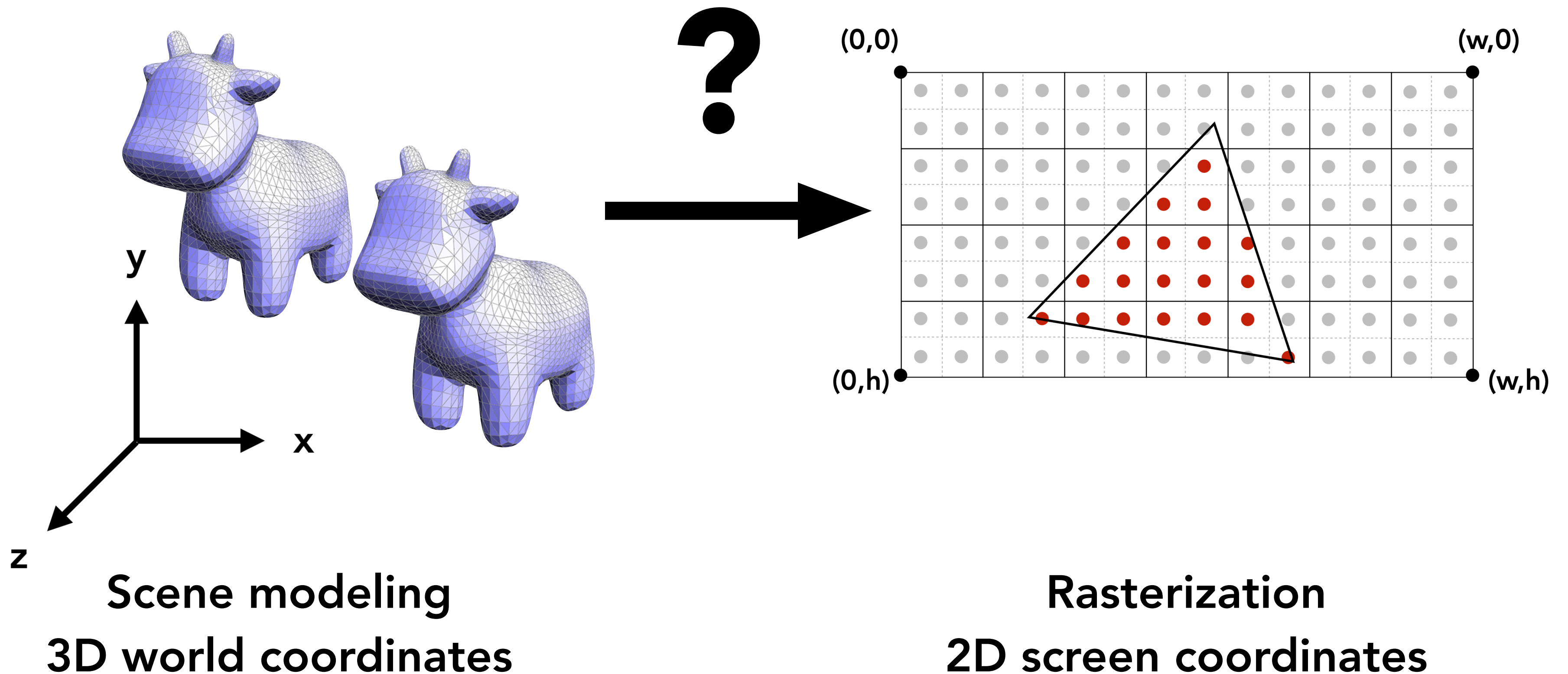




Viewing and Perspective



Viewing and Perspective Transforms



Camera Space

"Standard" Camera Space



We will use this convention for "standard" camera coordinates:

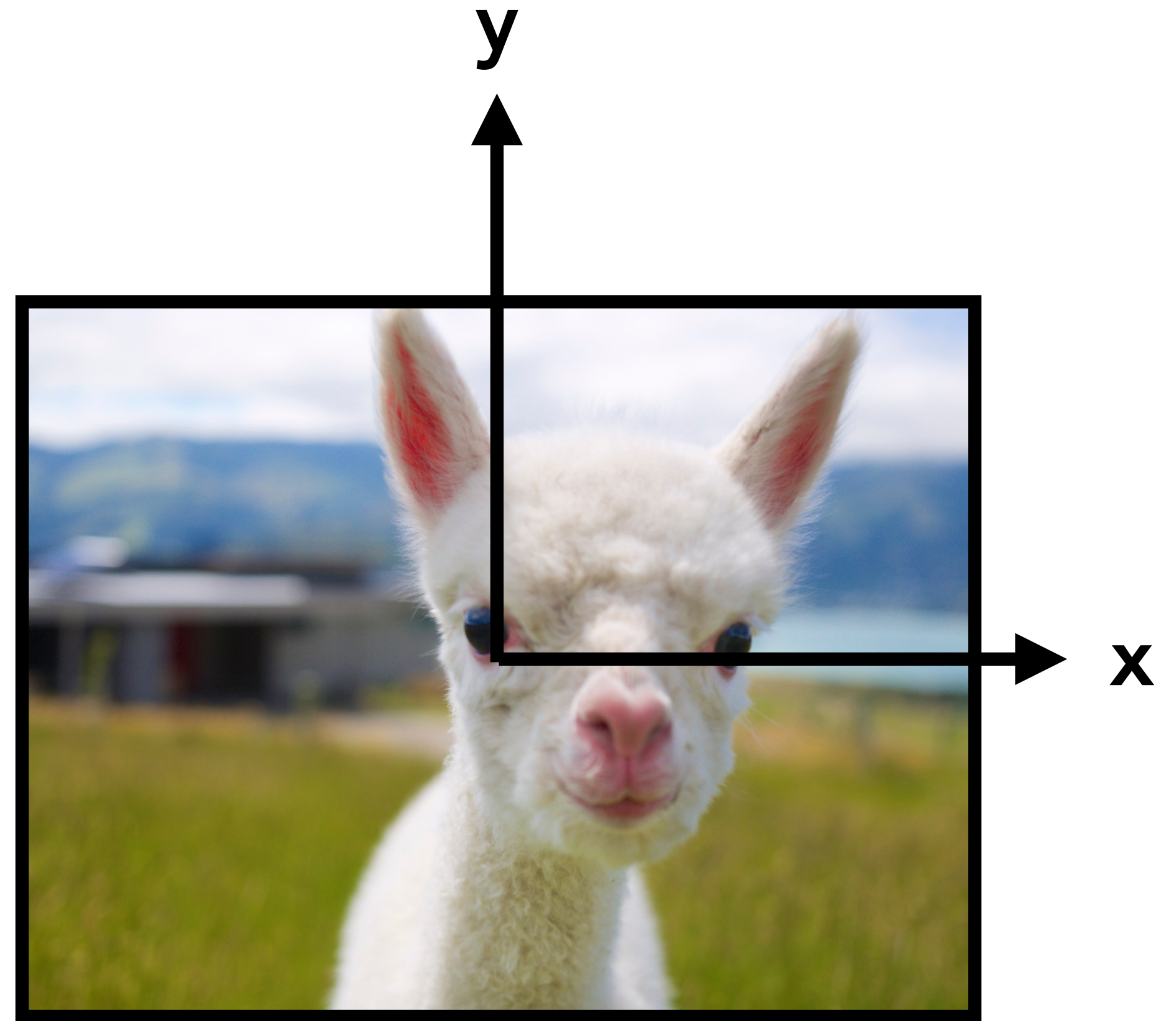
- camera located at the origin
- looking down *negative* z-axis
- vertical vector is y-axis
- (x-axis) orthogonal to y & z

"Standard" Camera Coordinates



Resulting image

"Standard" Camera Coordinates

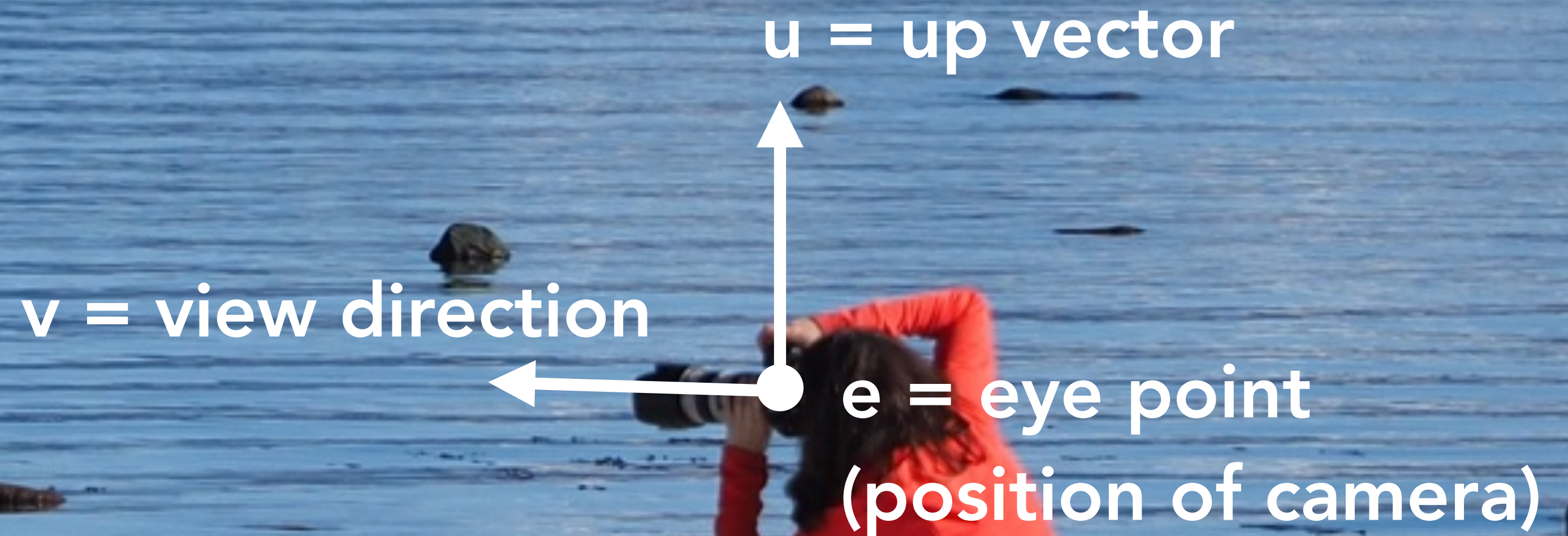


Resulting image
(z-axis pointing away from scene)

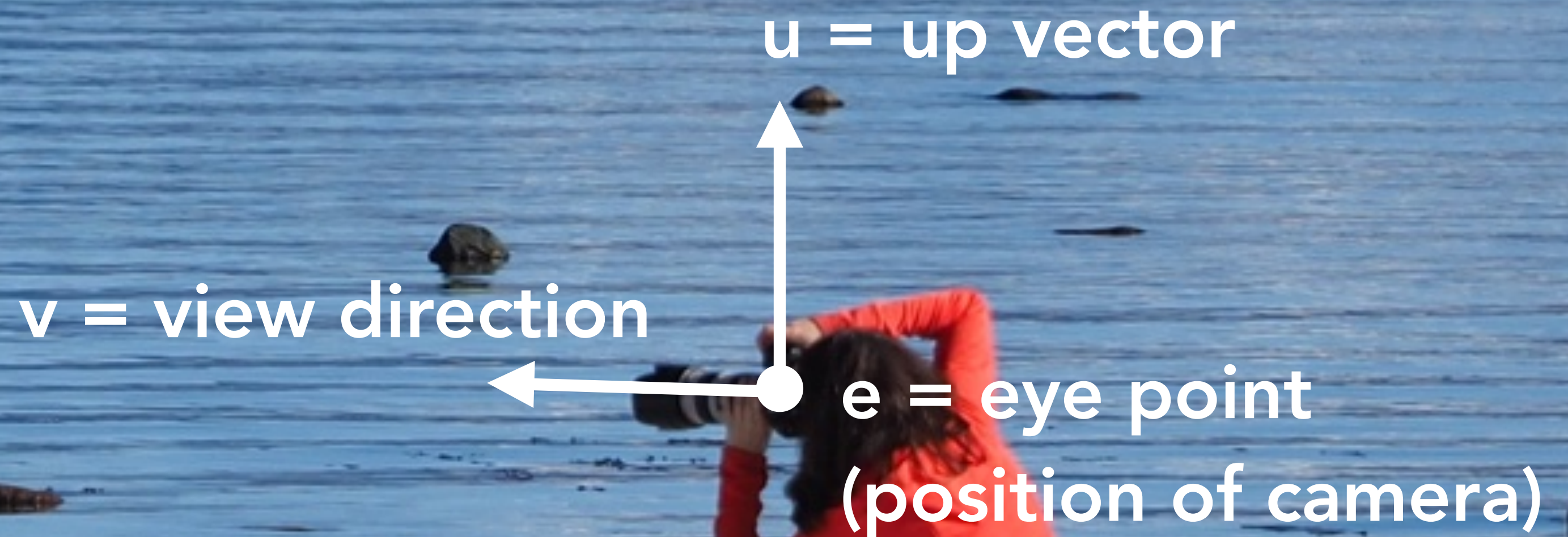
Consider A Camera Pointing in The World



Consider A Camera Pointing in The World



Camera “Look-At” Transformation



Input: e , u & v given in world space coordinates
Output: transform matrix from world space
to standard camera space

Camera “Look-At” Transformation

“Look-At” transformation

Camera “Look-At” Transformation

“Look-At” transformation

- Given eye point e , *up* vector u , *view* direction v

Camera “Look-At” Transformation

“Look-At” transformation

- Given eye point e , *up* vector u , *view* direction v
- Assume u and v are orthonormal

Camera “Look-At” Transformation

“Look-At” transformation

- Given eye point e , *up* vector u , *view* direction v
 - Assume u and v are orthonormal
 - Find *right* vector $r = v \times u$

Camera “Look-At” Transformation

“Look-At” transformation

- Given eye point e , *up* vector u , *view* direction v
 - Assume u and v are orthonormal
 - Find *right* vector $r = v \times u$
- Transform camera (e, r, u, v) to standard camera:

Camera “Look-At” Transformation

“Look-At” transformation

- Given eye point e , *up* vector u , *view* direction v
 - Assume u and v are orthonormal
 - Find *right* vector $r = v \times u$
- Transform camera (e, r, u, v) to standard camera:
 - located at the origin

Camera “Look-At” Transformation

“Look-At” transformation

- Given eye point e , *up* vector u , *view* direction v
 - Assume u and v are orthonormal
 - Find *right* vector $r = v \times u$
- Transform camera (e, r, u, v) to standard camera:
 - located at the origin
 - looking down *negative* z axis

Camera “Look-At” Transformation

“Look-At” transformation

- Given eye point e , *up* vector u , *view* direction v
 - Assume u and v are orthonormal
 - Find *right* vector $r = v \times u$
- Transform camera (e, r, u, v) to standard camera:
 - located at the origin
 - looking down *negative* z axis
 - up vector is y axis

Camera “Look-At” Transformation

Camera “Look-At” Transformation

Inverse: Matrix from standard camera to world space

$$\begin{pmatrix} r_x & u_x & -v_x & e_x \\ r_y & u_y & -v_y & e_y \\ r_z & u_z & -v_z & e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Camera "Look-At" Transformation

Inverse: Matrix from standard camera to world space
(Why? This is a coordinate frame transform to (e,r,u,-v))

$$\begin{pmatrix} r_x & u_x & -v_x & e_x \\ r_y & u_y & -v_y & e_y \\ r_z & u_z & -v_z & e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Camera “Look-At” Transformation

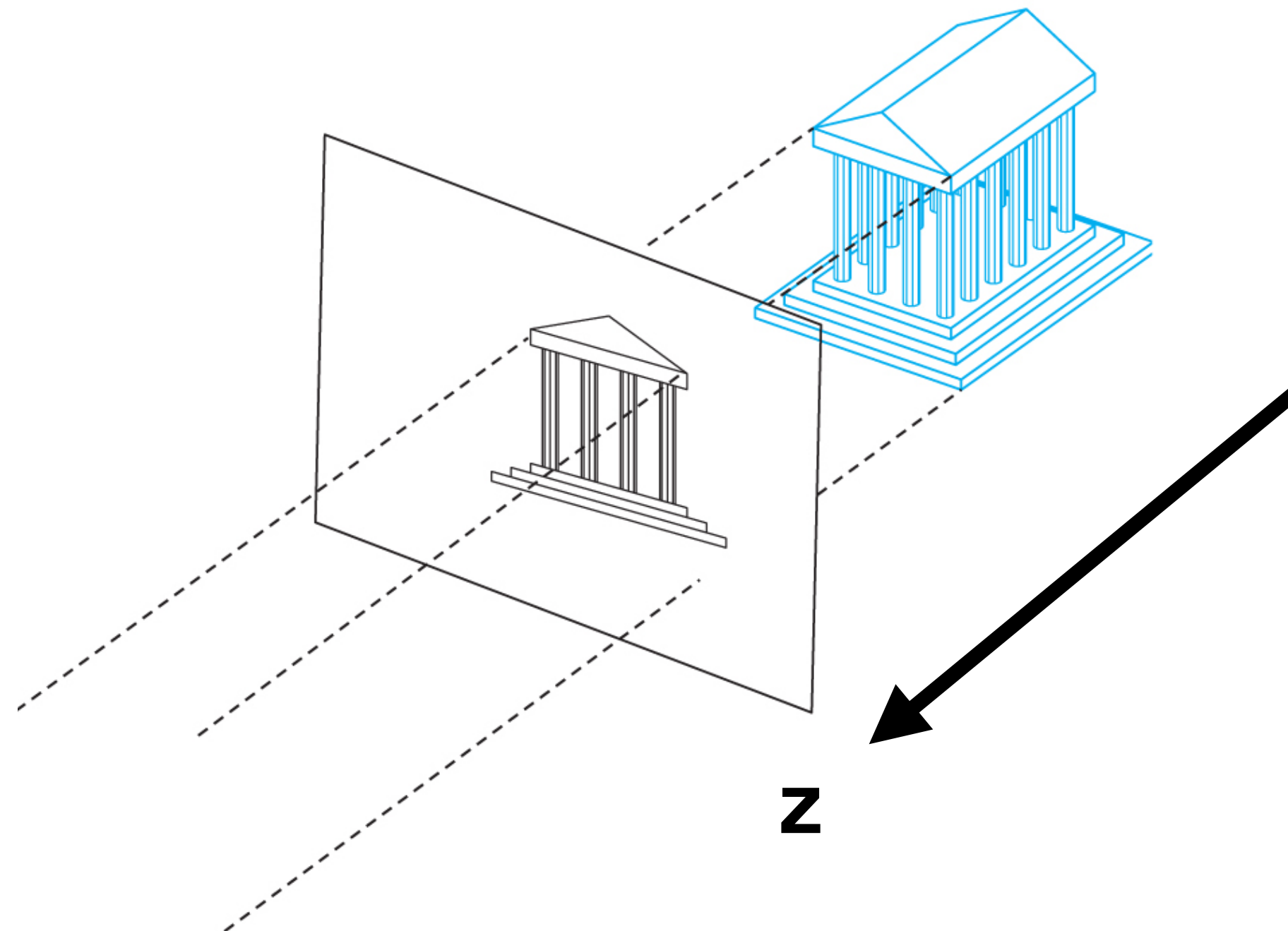
Inverse: Matrix from standard camera to world space
(Why? This is a coordinate frame transform to (e,r,u,-v))

$$\begin{pmatrix} r_x & u_x & -v_x & e_x \\ r_y & u_y & -v_y & e_y \\ r_z & u_z & -v_z & e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

“Look-at” (world->camera) transform is the inverse of above matrix:

$$\begin{pmatrix} r_x & u_x & -v_x & e_x \\ r_y & u_y & -v_y & e_y \\ r_z & u_z & -v_z & e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} r_x & r_y & r_z & 0 \\ u_x & u_y & u_z & 0 \\ -v_x & -v_y & -v_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transform Camera Space to Image Plane?



How to transform from 3D camera space to 2D image plane?

- One option: orthographic projection (just delete z)
- Useful, e.g. for engineering drawings
- But is this the whole story?

Perspective



Perspective in Art



CS184/284A

Berlinghieri 1235

104

S26 - O'Brien

Perspective in Art



CS184/284A

Giotto 1290

105

S26 - O'Brien

Perspective in Art



CS184/284A

Giotto 1290

105

S26 - O'Brien

Perspective in Art



Something wrong here!

Perspective Composition = Camera Position + Angle of View

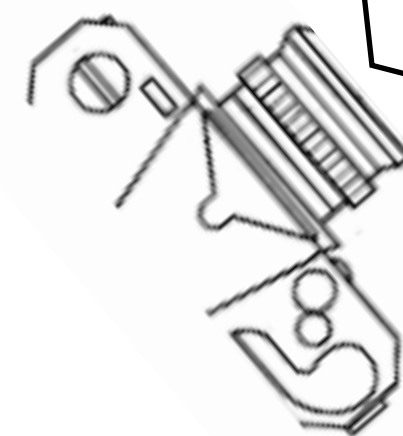
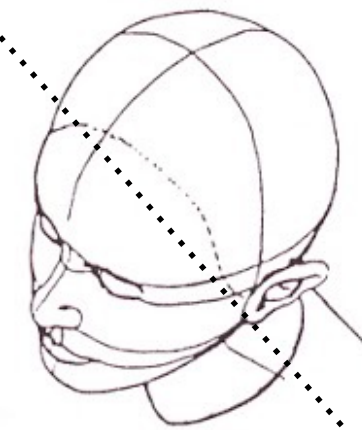


In this sequence, angle of view decreases as distance from subject increases, to size of human subject in image.

Notice the dramatic change in background perspective.

From Canon EF Lens Work III

Perspective Composition



16 mm (110°)

Up close and zoomed wide
with short focal length

Perspective Composition

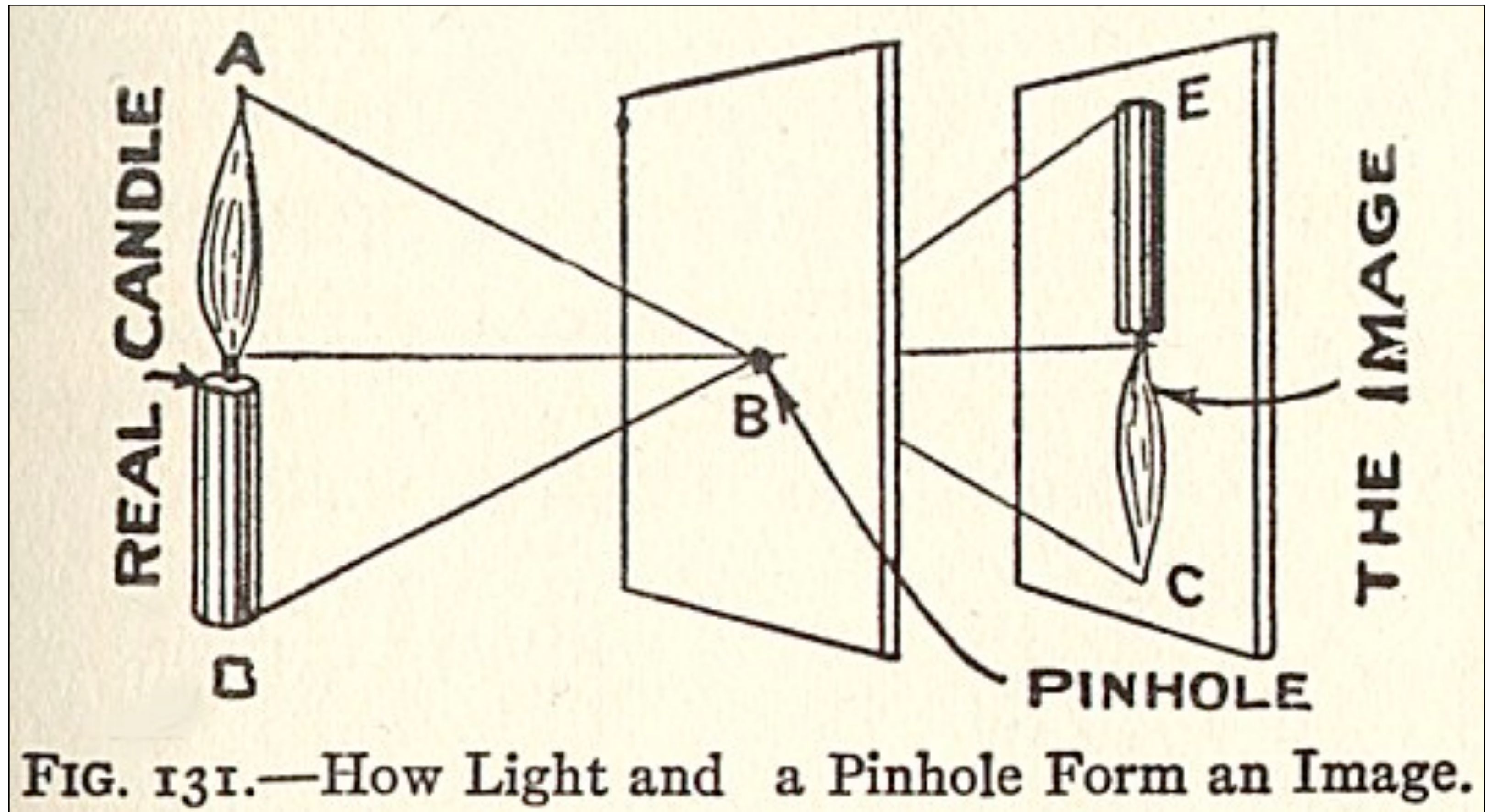


200 mm (12°)

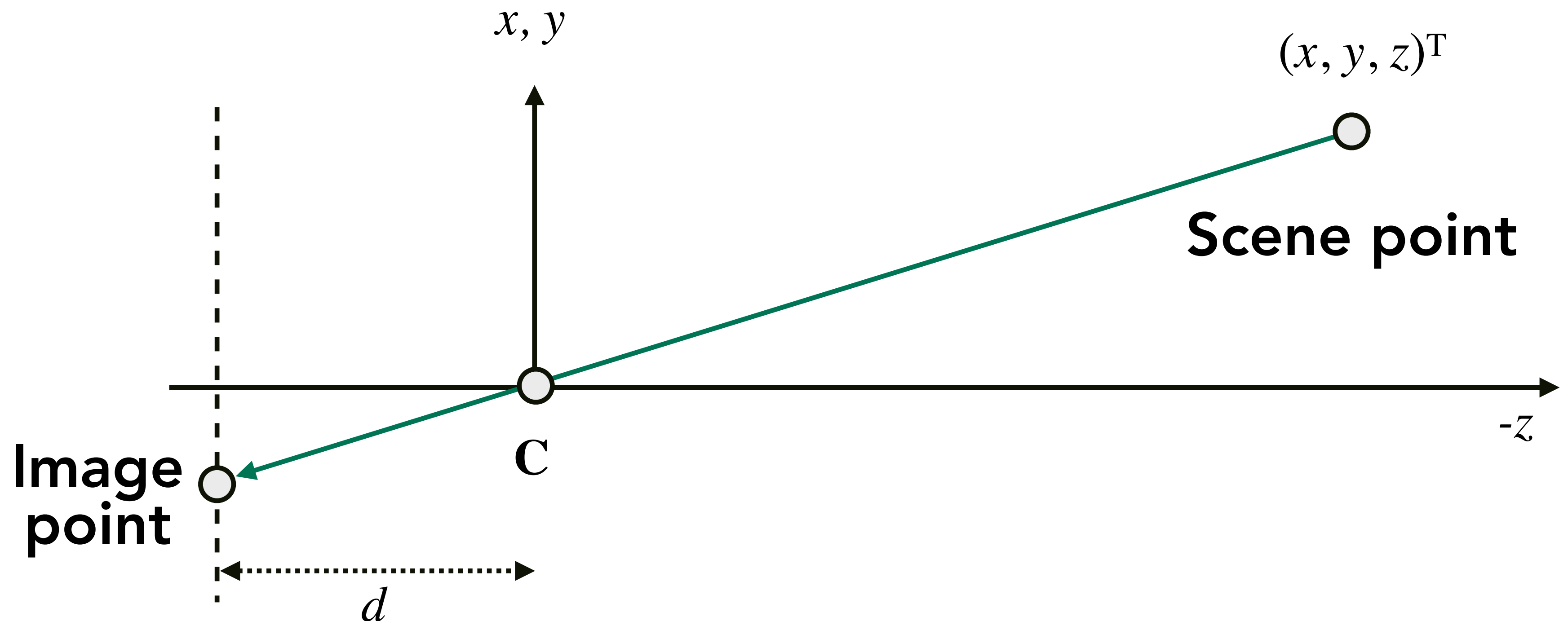
Walk back and zoom in
with long focal length

Pinhole Camera Model

Pinhole Camera

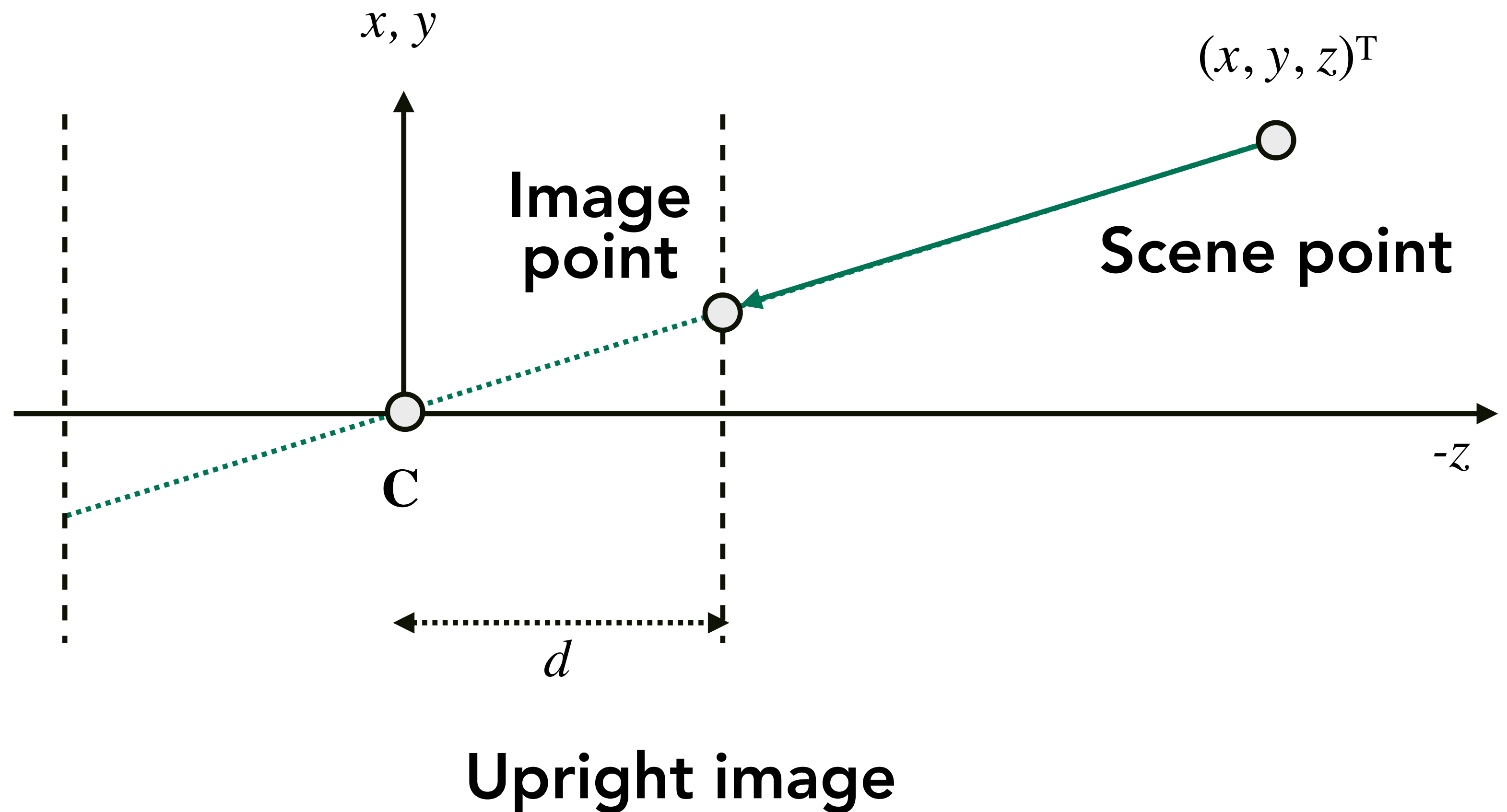


Projective Transform



Inverted image (as in real pinhole camera)

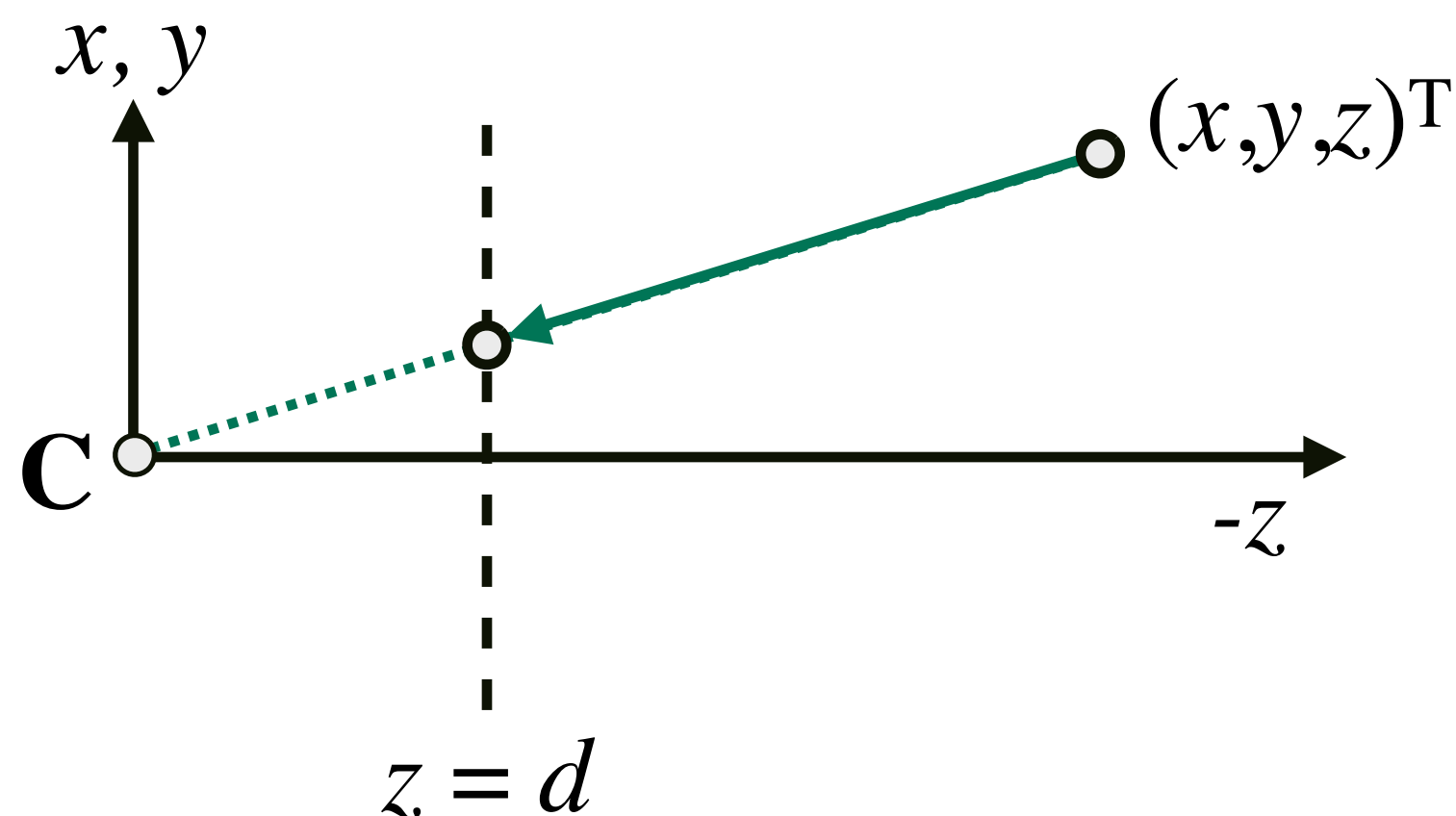
Pinhole Camera Projective Transform



Projective Transforms

Standard perspective projection

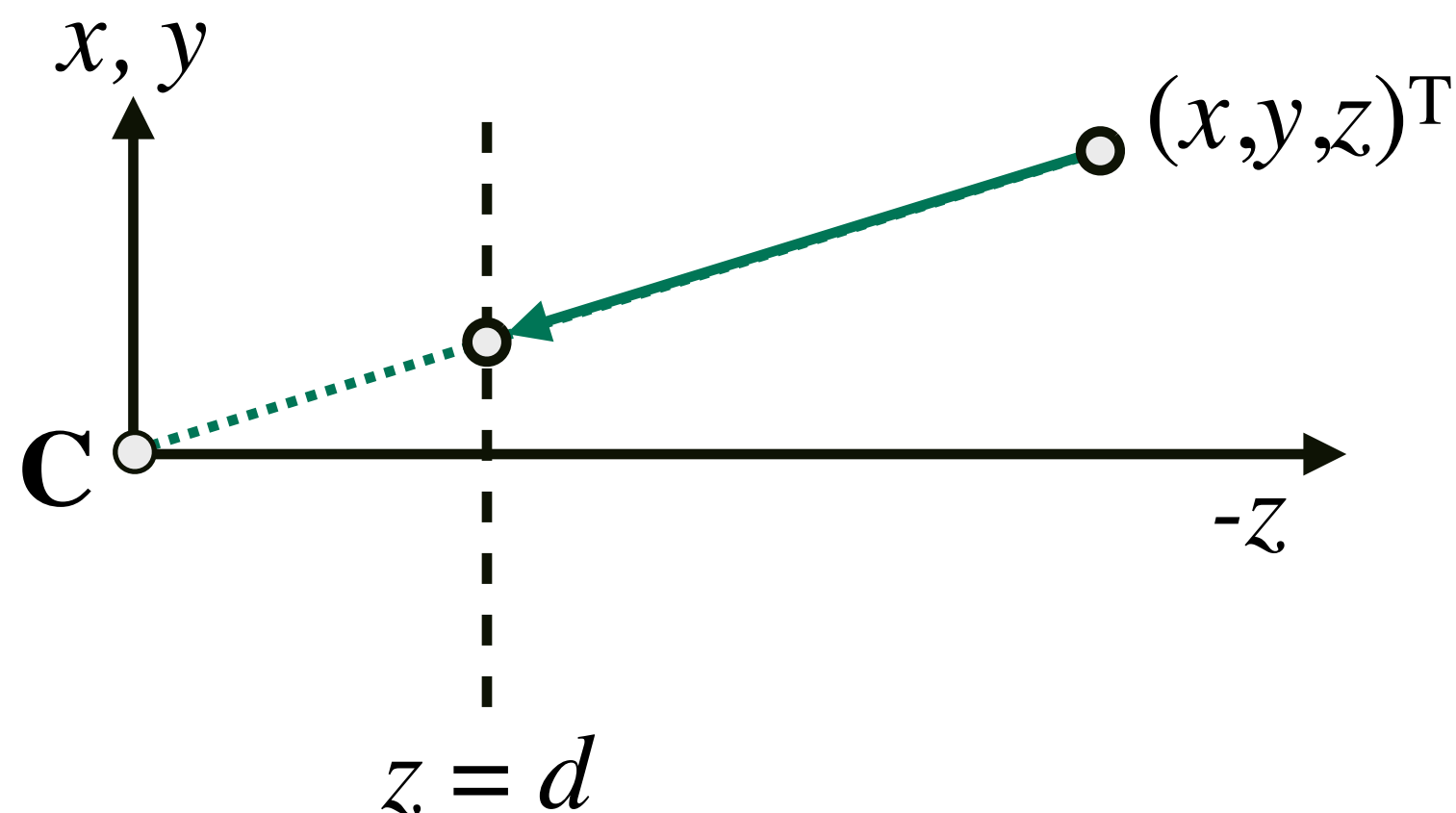
- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$



Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$

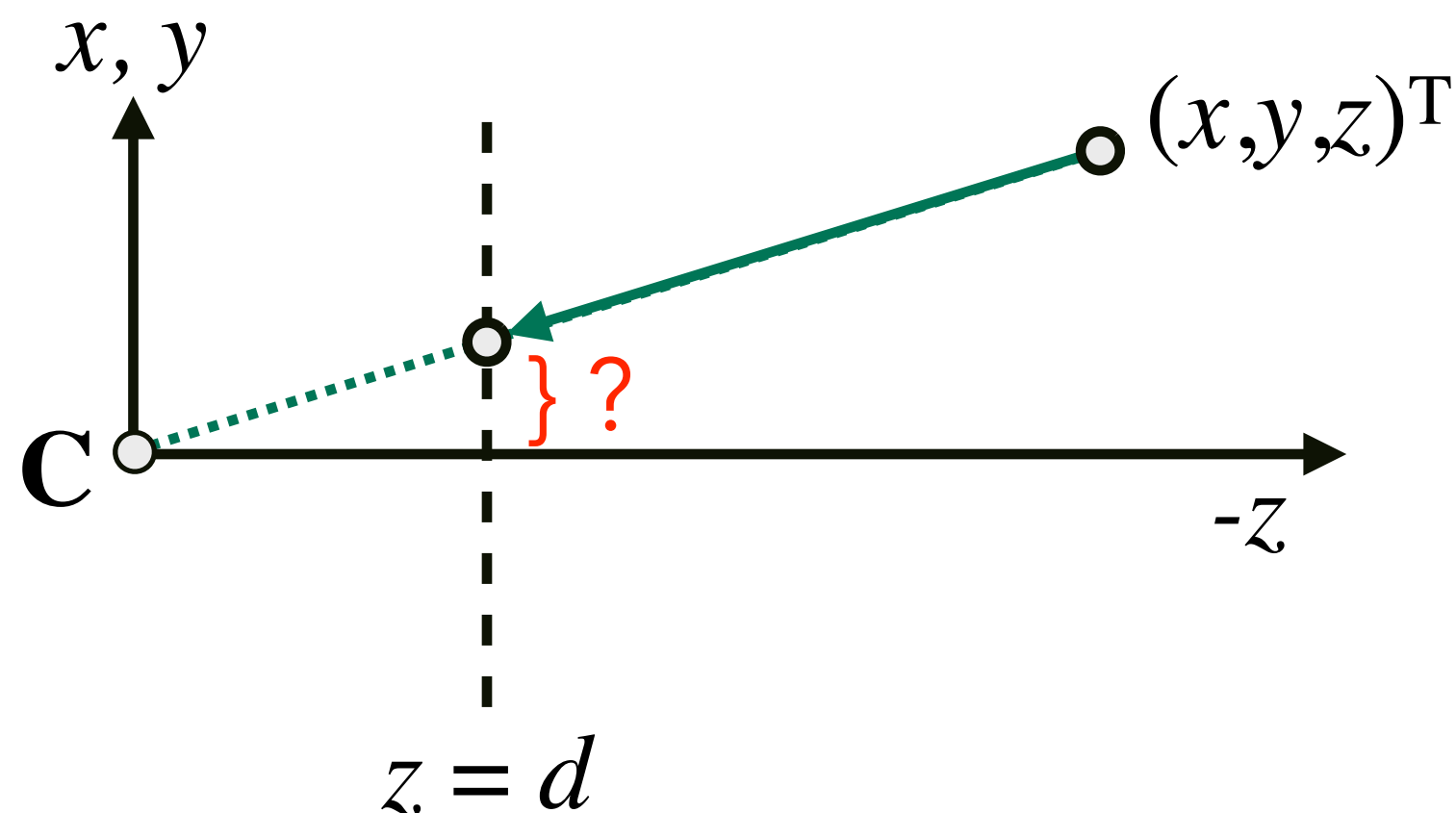


$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \\ \\ d \end{pmatrix}$$

Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$

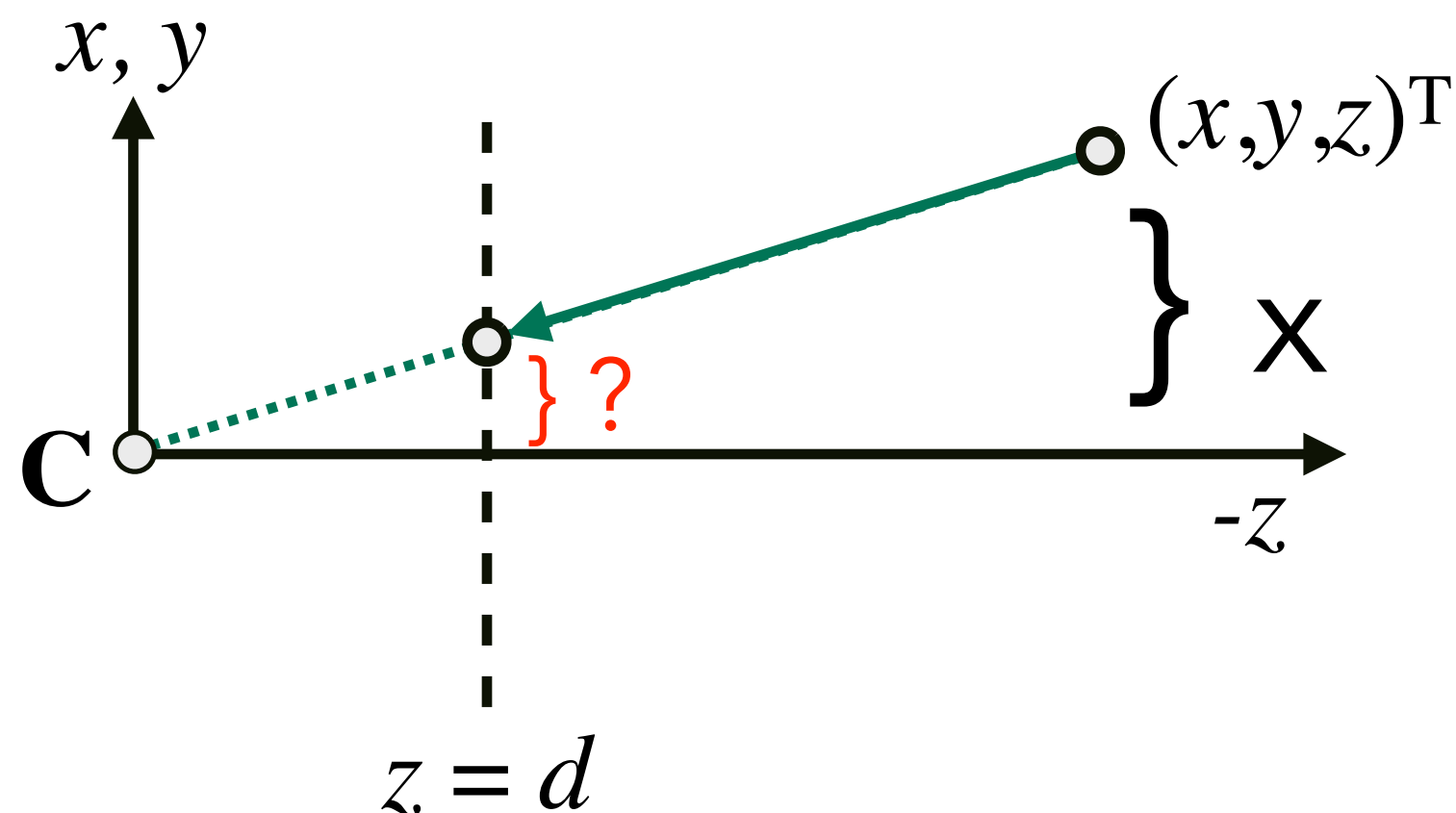


$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \\ \\ d \end{pmatrix}$$

Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$

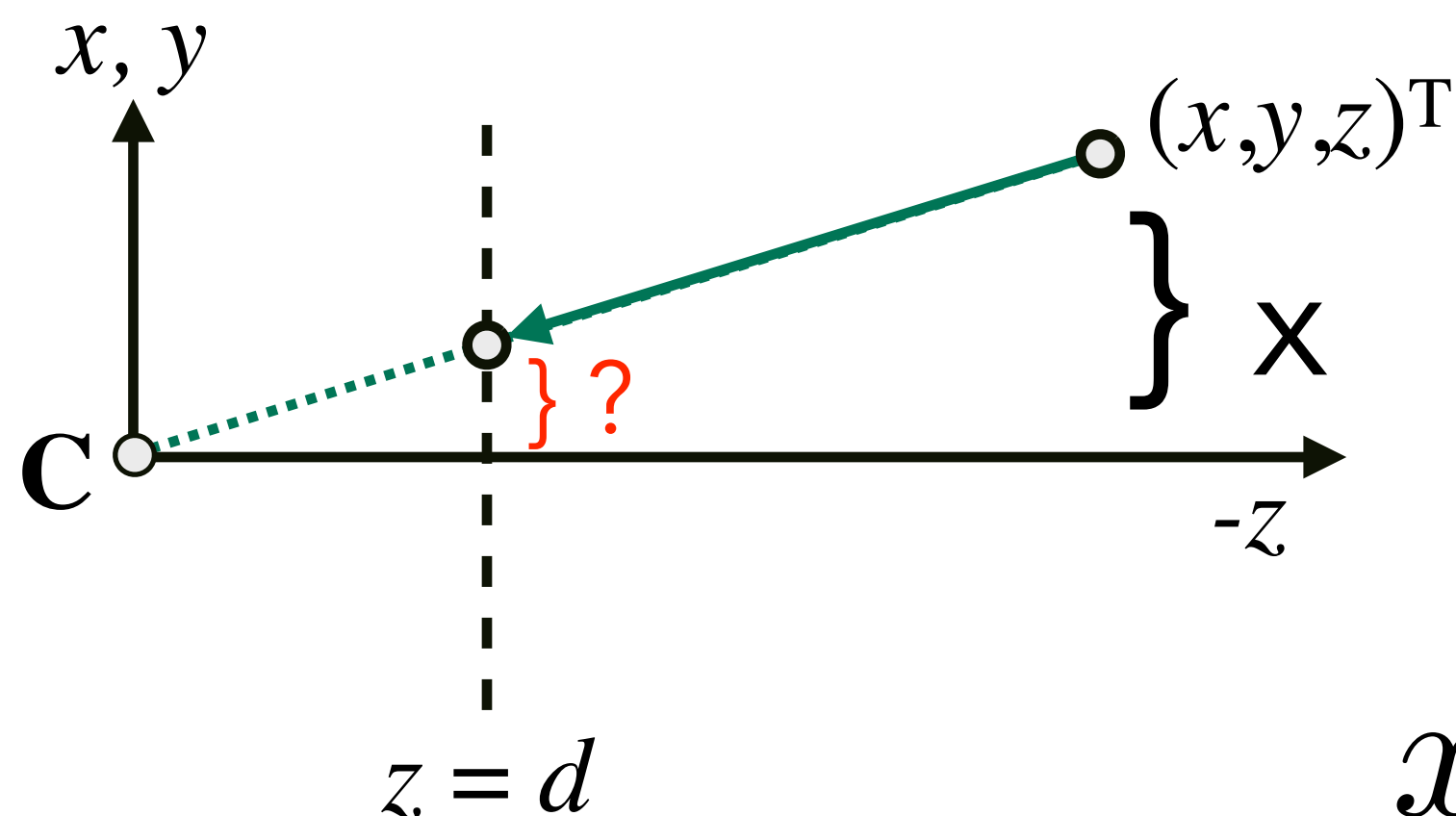


$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \\ \\ d \end{pmatrix}$$

Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$



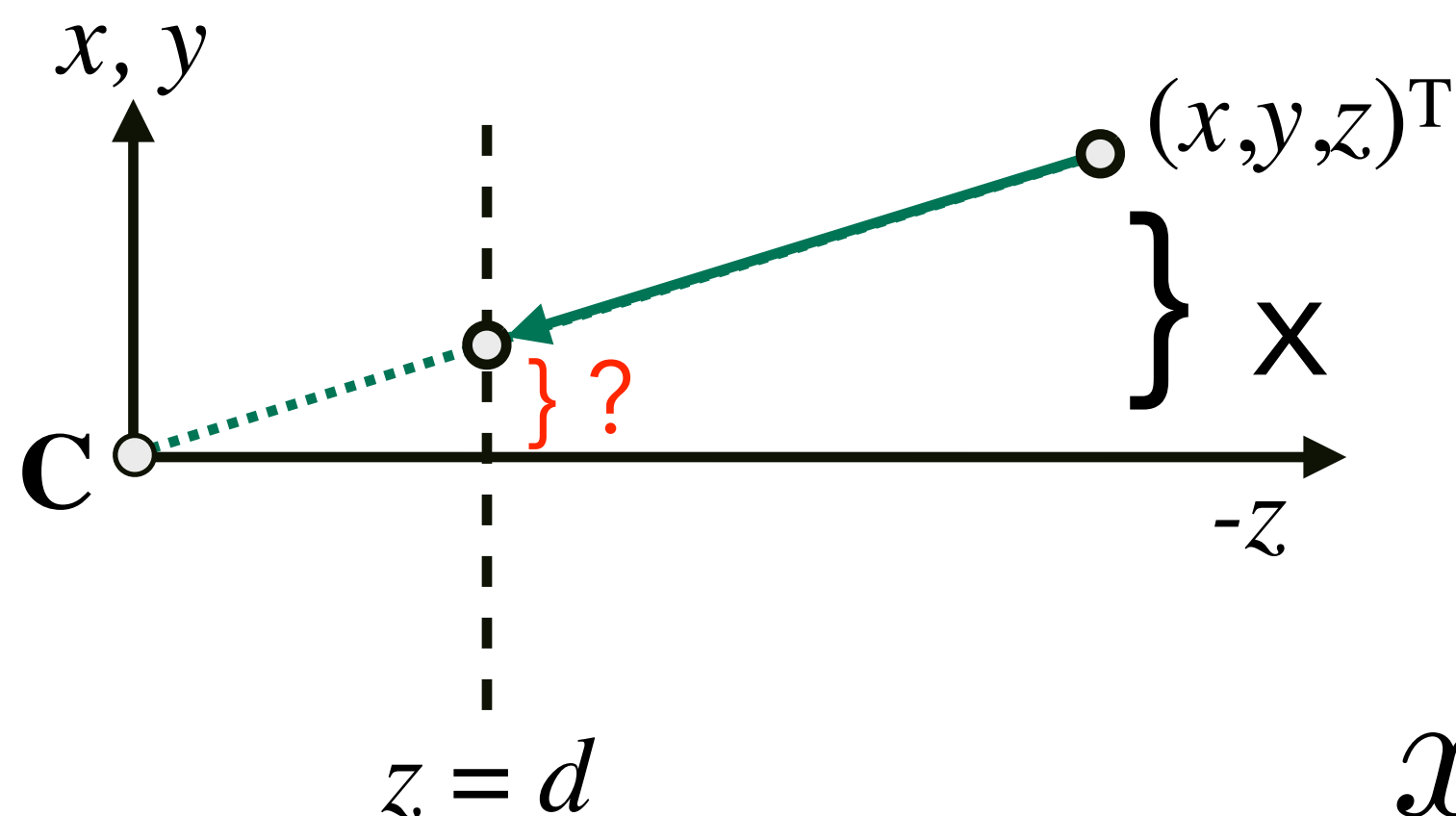
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \\ d \end{pmatrix}$$

$$\frac{x}{z} = \frac{?}{d}$$

Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$



$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot d/z \\ y \cdot d/z \\ d \end{pmatrix}$$

$$\frac{x}{z} = \frac{?}{d}$$

Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot d/z \\ y \cdot d/z \\ d \end{pmatrix}$$

Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot d/z \\ y \cdot d/z \\ d \end{pmatrix}$$

Perspective foreshortening

Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot d/z \\ y \cdot d/z \\ d \end{pmatrix}$$

Perspective foreshortening

- Need division by z

Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot d/z \\ y \cdot d/z \\ d \end{pmatrix}$$

Perspective foreshortening

- Need division by z
- Matrix representation?

Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot d/z \\ y \cdot d/z \\ d \end{pmatrix}$$

Perspective foreshortening

- Need division by z
 - Matrix representation?
- ➡ Homogeneous coordinates!

Homogenous Coordinates (3D)

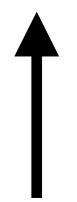
$$\mathbf{p} = \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix}$$

Homogenous Coordinates (3D)

$$\mathbf{p} = \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} \longleftrightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Homogenous Coordinates (3D)

$$\mathbf{p} = \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} \longleftrightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

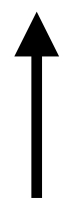
$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$


**Note non-zero term in final row.
First time we have seen this.**

Homogenous Coordinates (3D)

$$\mathbf{p} = \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} \longleftrightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\mathbf{q} = \mathbf{M} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix}$$

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$


**Note non-zero term in final row.
First time we have seen this.**

Homogenous Coordinates (3D)

$$\mathbf{p} = \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} \longleftrightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$

$$\mathbf{q} = \mathbf{M} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} \longleftrightarrow \begin{pmatrix} xd/z \\ yd/z \\ d \\ 1 \end{pmatrix}$$

**Note non-zero term in final row.
First time we have seen this.**

Homogenous Coordinates (3D)

$$\mathbf{p} = \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} \longleftrightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$

$$\mathbf{q} = \mathbf{M} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} \longleftrightarrow \begin{pmatrix} xd/z \\ yd/z \\ d \\ 1 \end{pmatrix}$$

**Note non-zero term in final row.
First time we have seen this.**



Homogenous Coordinates (3D)

$$\mathbf{p} = \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} \longleftrightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$

$$\mathbf{q} = \mathbf{M} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} \longleftrightarrow \begin{pmatrix} xd/z \\ yd/z \\ d \\ 1 \end{pmatrix}$$

**Note non-zero term in final row.
First time we have seen this.**

**Note zeros in 4th column.
We'll change this later for
hidden surface determination.**



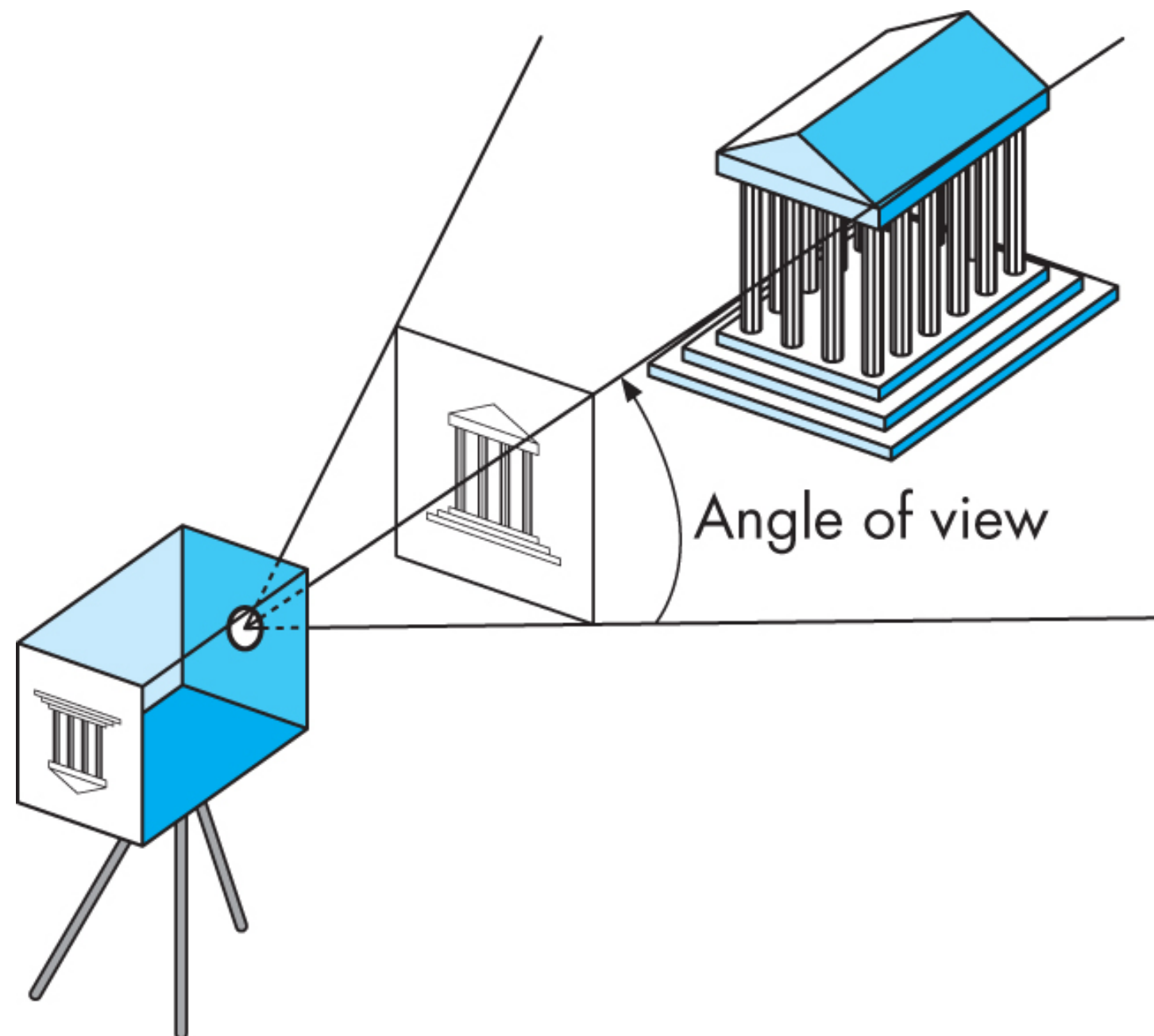
Pinhole Camera Model

This mathematical model produces all linear perspective effects!

- **Converging lines + vanishing points**
- **Closer objects appear larger in images**
- **...**

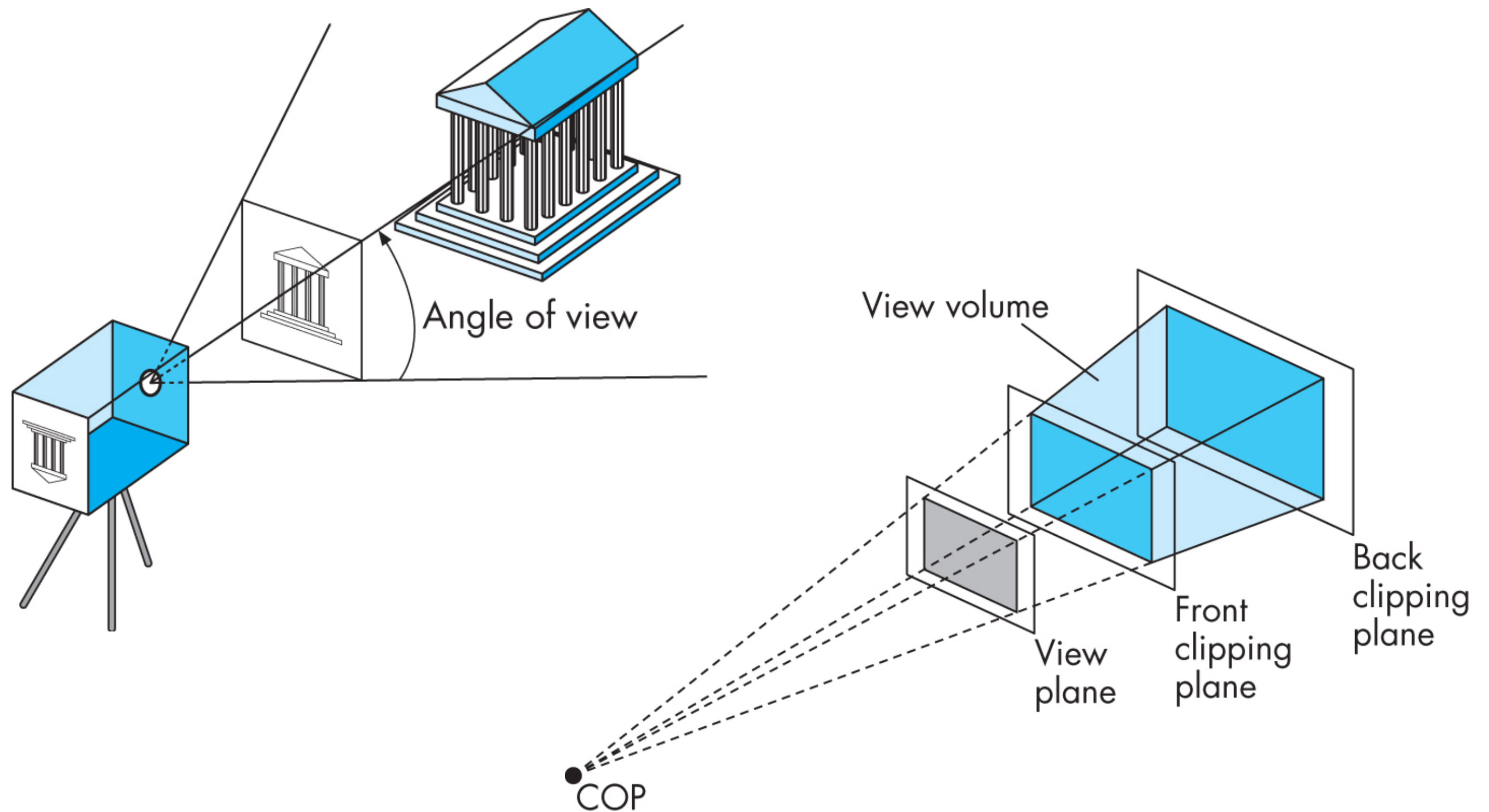
Specifying Real Camera Perspectives

Specifying Perspective Projection



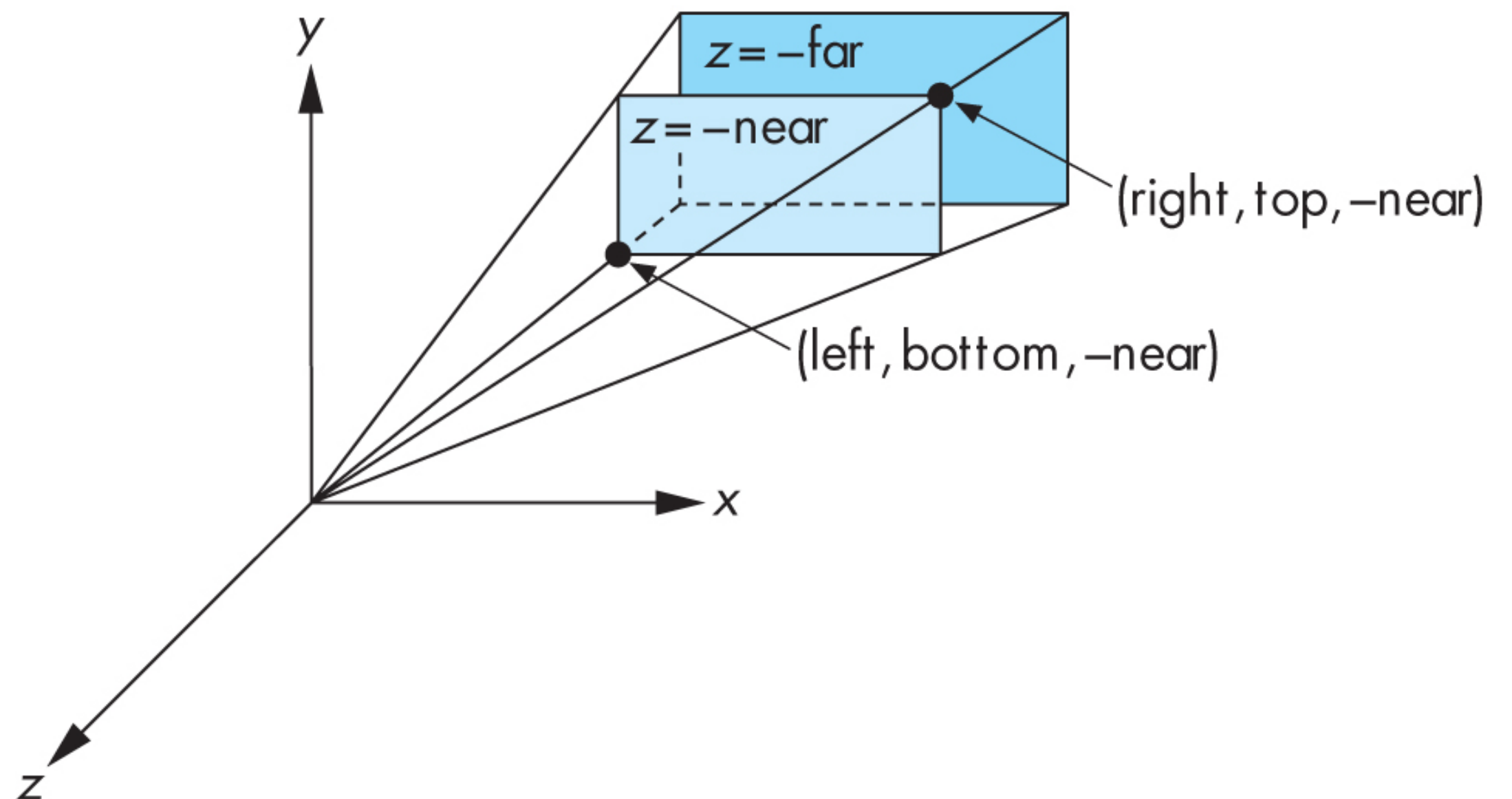
From Angel and Shreiner, Interactive Computer Graphics

Specifying Perspective Projection



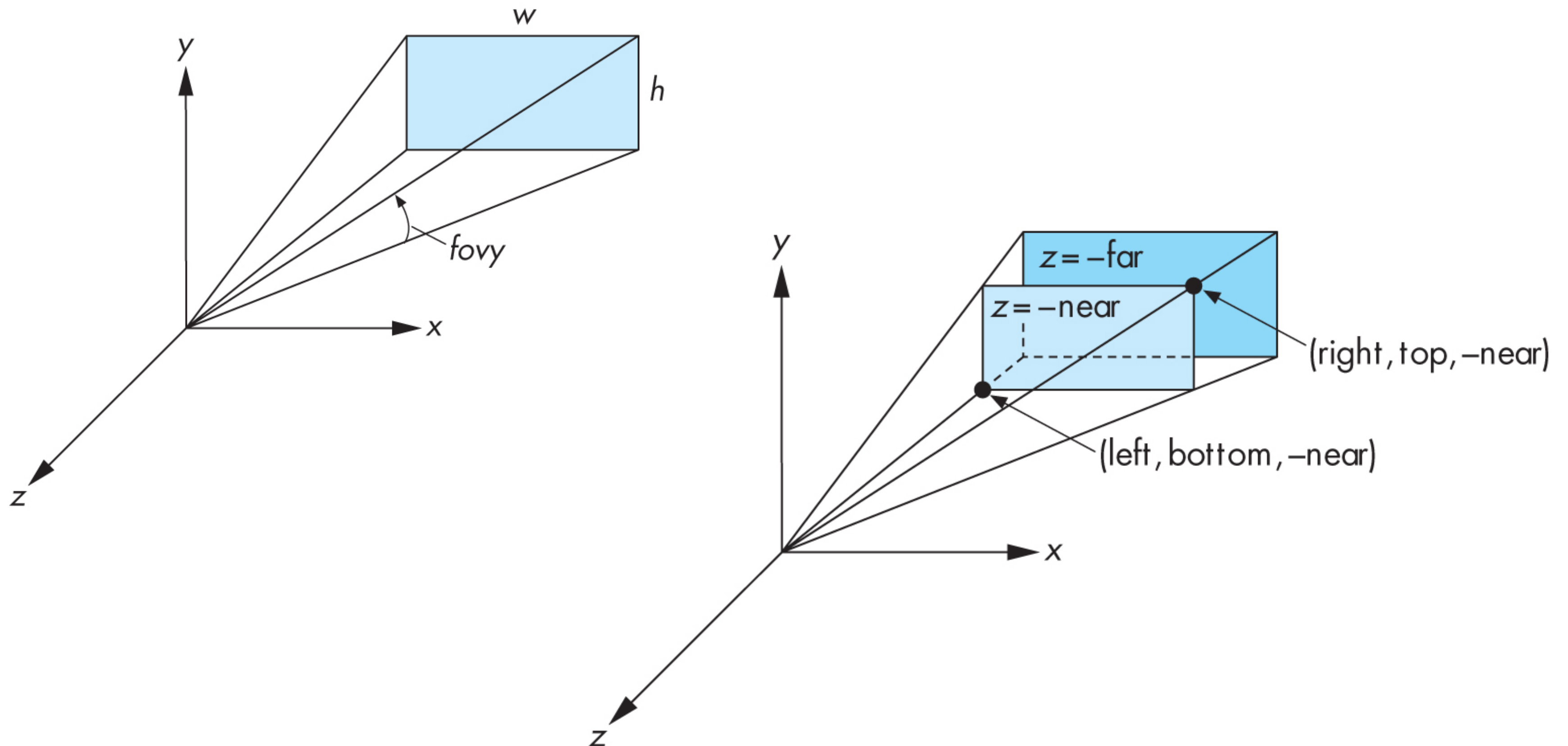
From Angel and Shreiner, Interactive Computer Graphics

Specifying Perspective Viewing Volume



From Angel and Shreiner, Interactive Computer Graphics

Specifying Perspective Viewing Volume



From Angel and Shreiner, Interactive Computer Graphics

Specifying Perspective Viewing Volume

Parameterized by

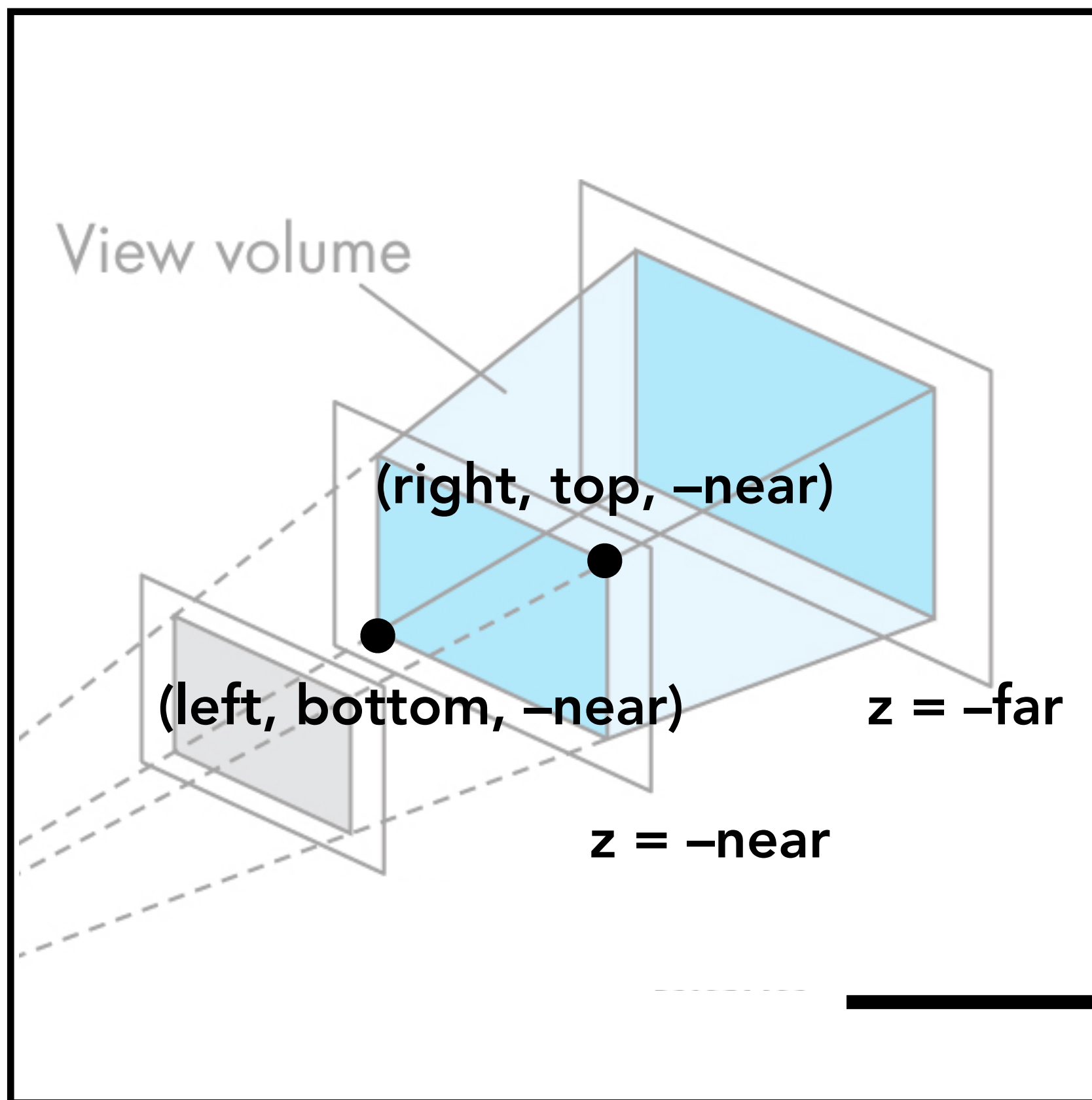
- **fovy** : vertical angular field of view
- **aspect ratio** : width / height of field of view
- **near** : depth of near clipping plane
- **far** : depth of far clipping plane

Derived quantities

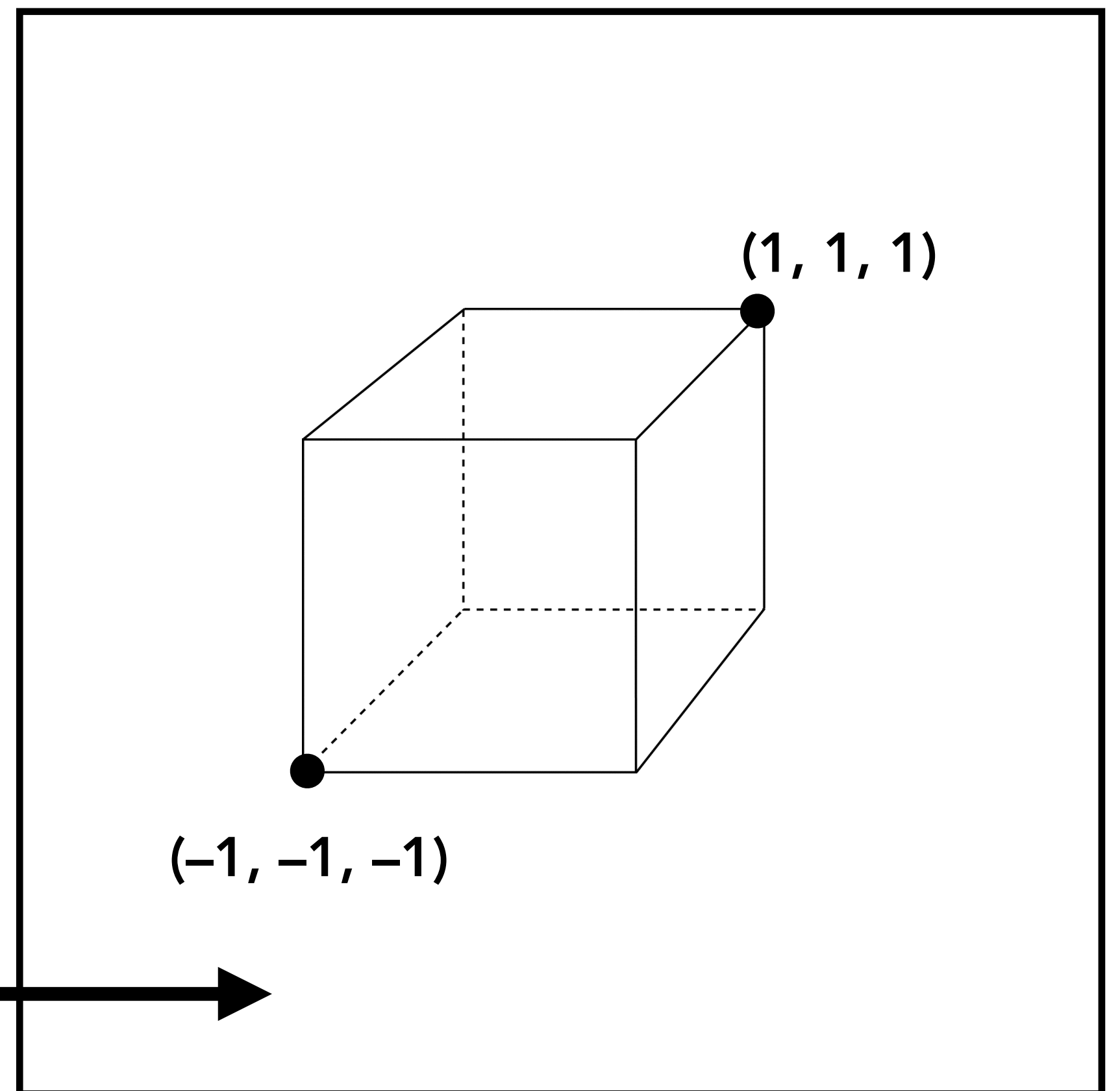
- **top** = $\text{near} * \tan(\text{fovy})$
- **bottom** = $-\text{top}$
- **right** = $\text{top} * \text{aspect}$
- **left** = $-\text{right}$

Perspective Projection Implementation

Perspective Projection Transform



Camera Coordinates



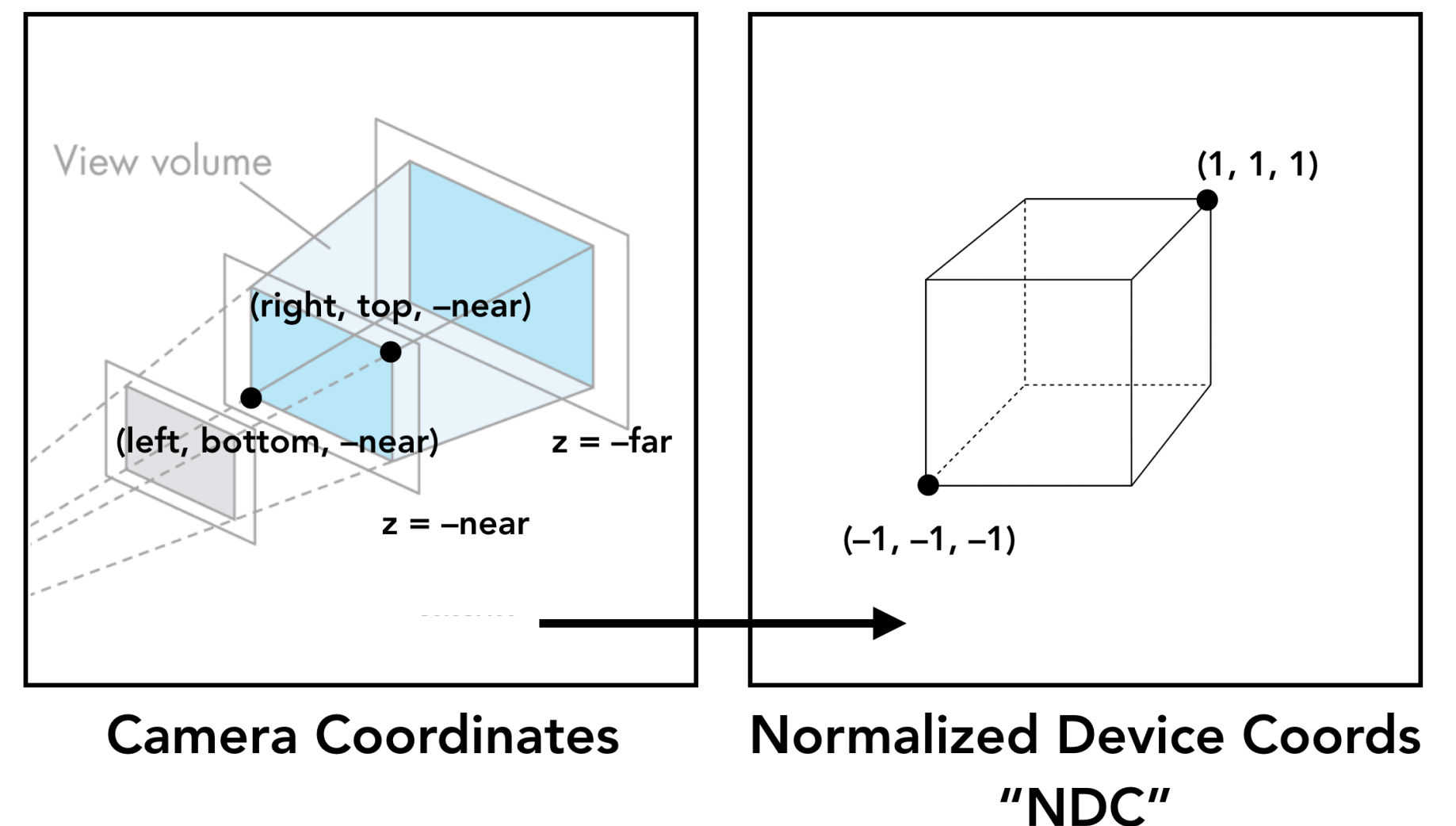
Normalized Device Coords
"NDC"

Later we will "flatten and scale " NDC to get framebuffer coordinates

Perspective Projection Transform

Notes:

- *Need not be symmetric about z-axis, but for simplicity here we assume so*
- This transform will preserve depth information (ordering) in NDC



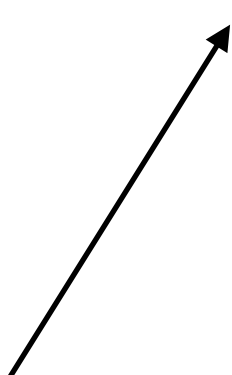
Perspective Transform Matrix

$$\mathbf{P} = \begin{bmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & \frac{-2far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Perspective Transform Matrix

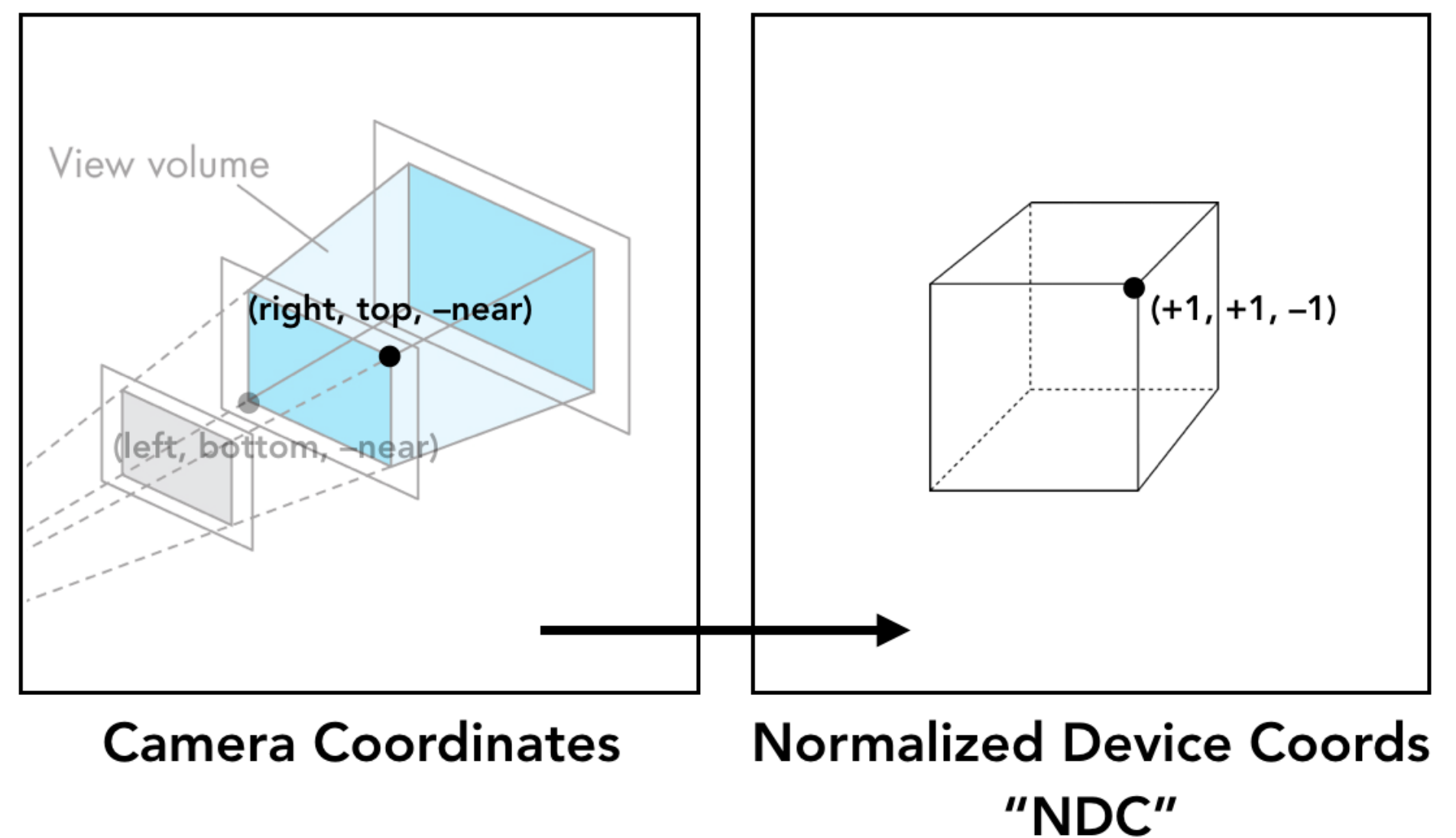
$$\mathbf{P} = \begin{bmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & \frac{-2far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Note entry in 4th column



Perspective Transform Matrix Example

$$\mathbf{P} = \begin{bmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & \frac{-2far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} right \\ top \\ -near \\ 1 \end{bmatrix}$$



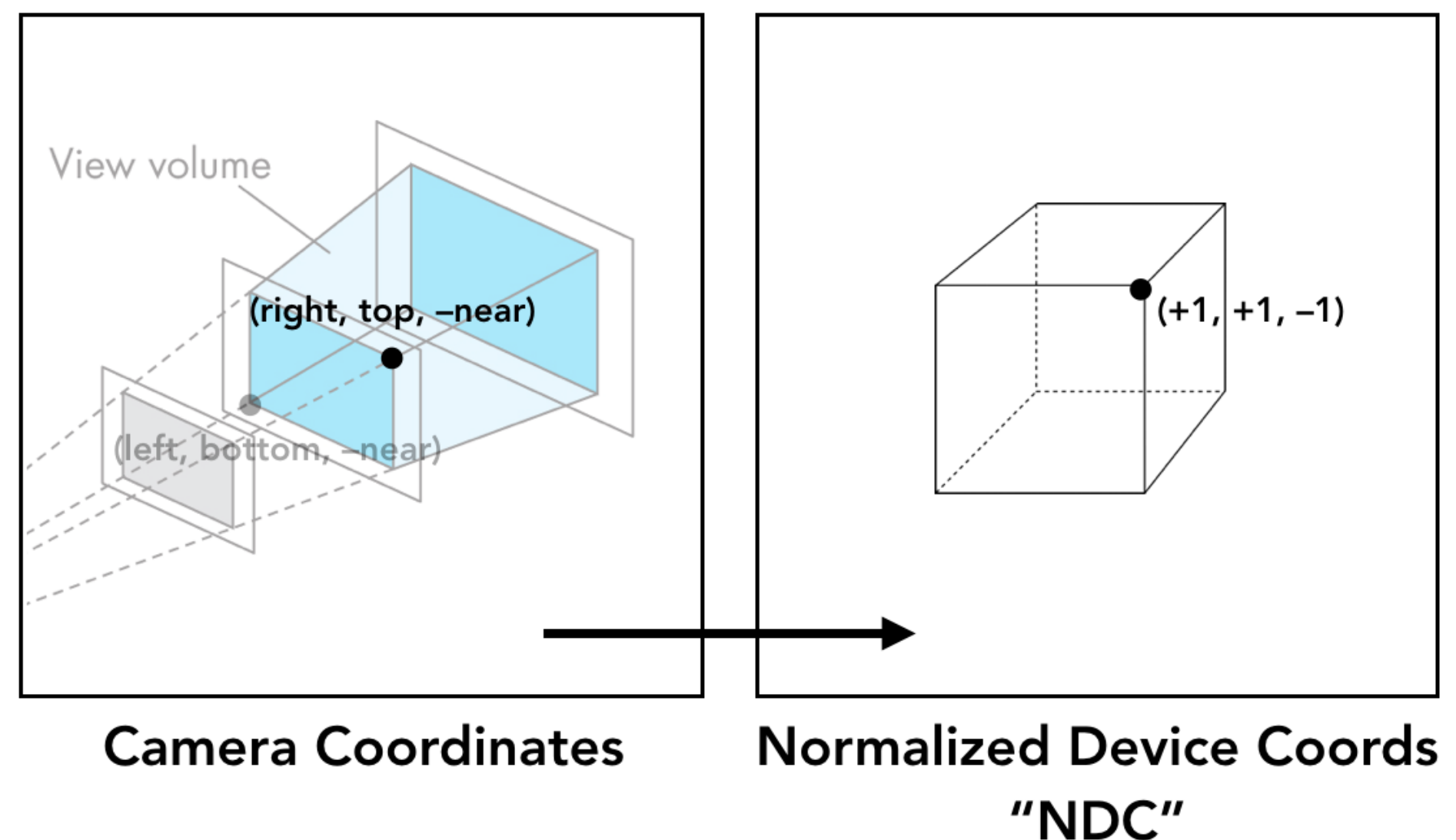
Perspective Transform Matrix Example

$$\mathbf{P} = \begin{bmatrix} \frac{\text{near}}{\text{right}} & 0 & 0 & 0 \\ 0 & \frac{\text{near}}{\text{top}} & 0 & 0 \\ 0 & 0 & -\frac{\text{far}+\text{near}}{\text{far}-\text{near}} & \frac{-2\text{far}*\text{near}}{\text{far}-\text{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \text{right} \\ \text{top} \\ -\text{near} \\ 1 \end{bmatrix}$$

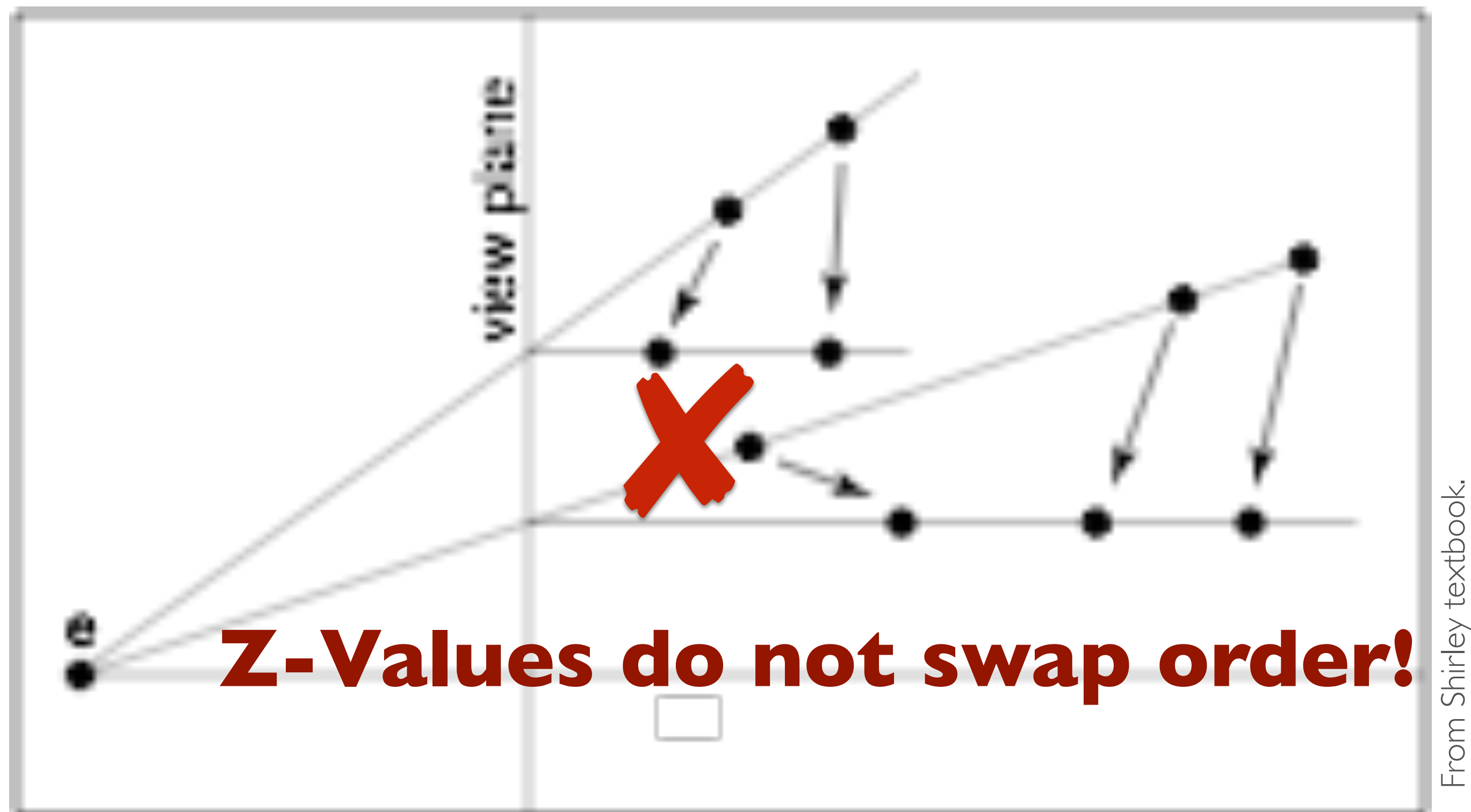
$$= \begin{bmatrix} \text{near} \\ \text{near} \\ \text{near} * \frac{\text{far}+\text{near}}{\text{far}-\text{near}} - \frac{2\text{far}*\text{near}}{\text{far}-\text{near}} \\ \text{near} \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 1 \\ \frac{-\text{far}+\text{near}}{\text{far}-\text{near}} \\ 1 \end{bmatrix}$$

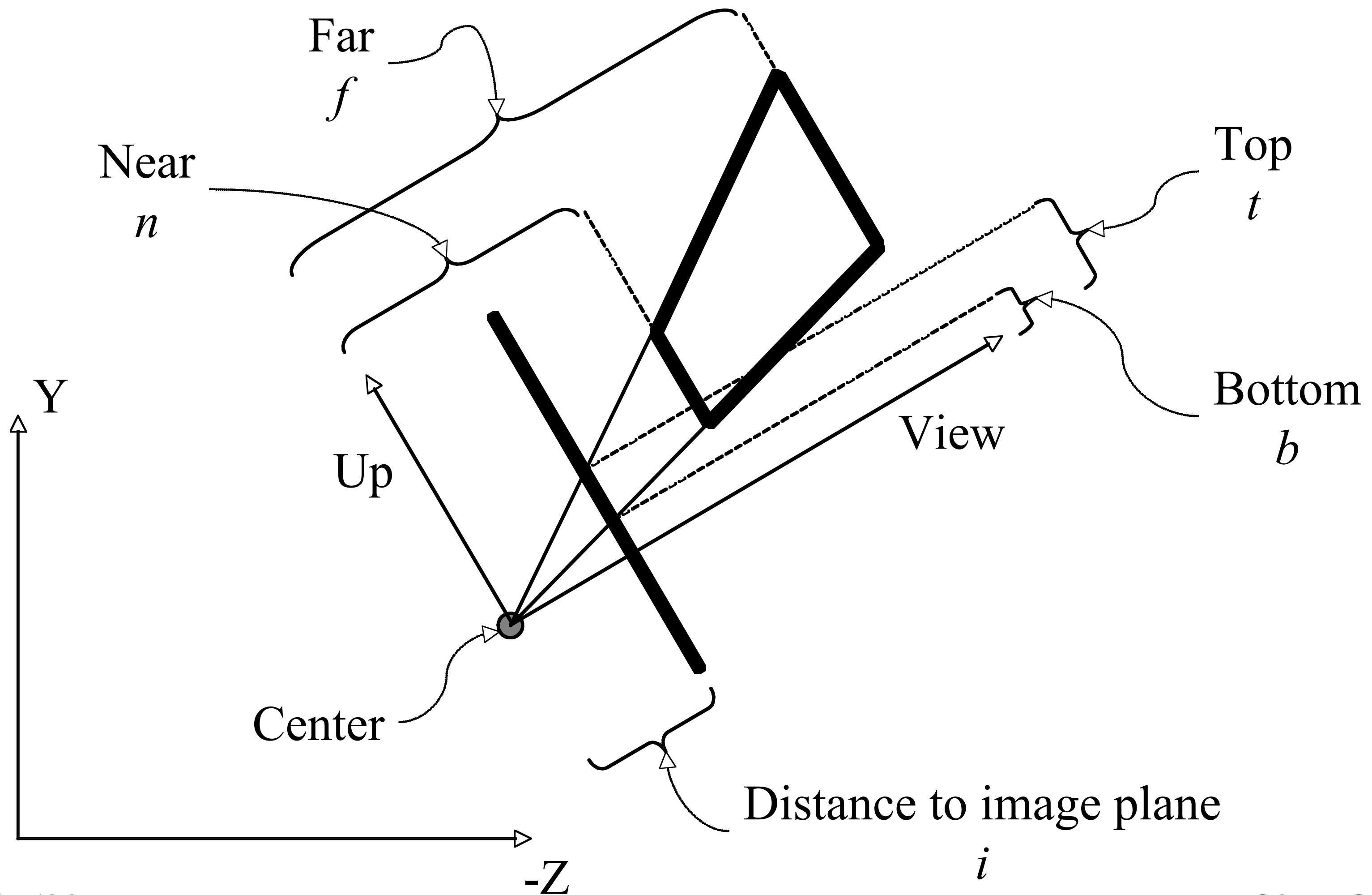
$$= \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$



Perspective Projection



Perspective Projection Alternatives



Perspective Tricks



Attendance

If you are seated in class, go to this form and sign in:

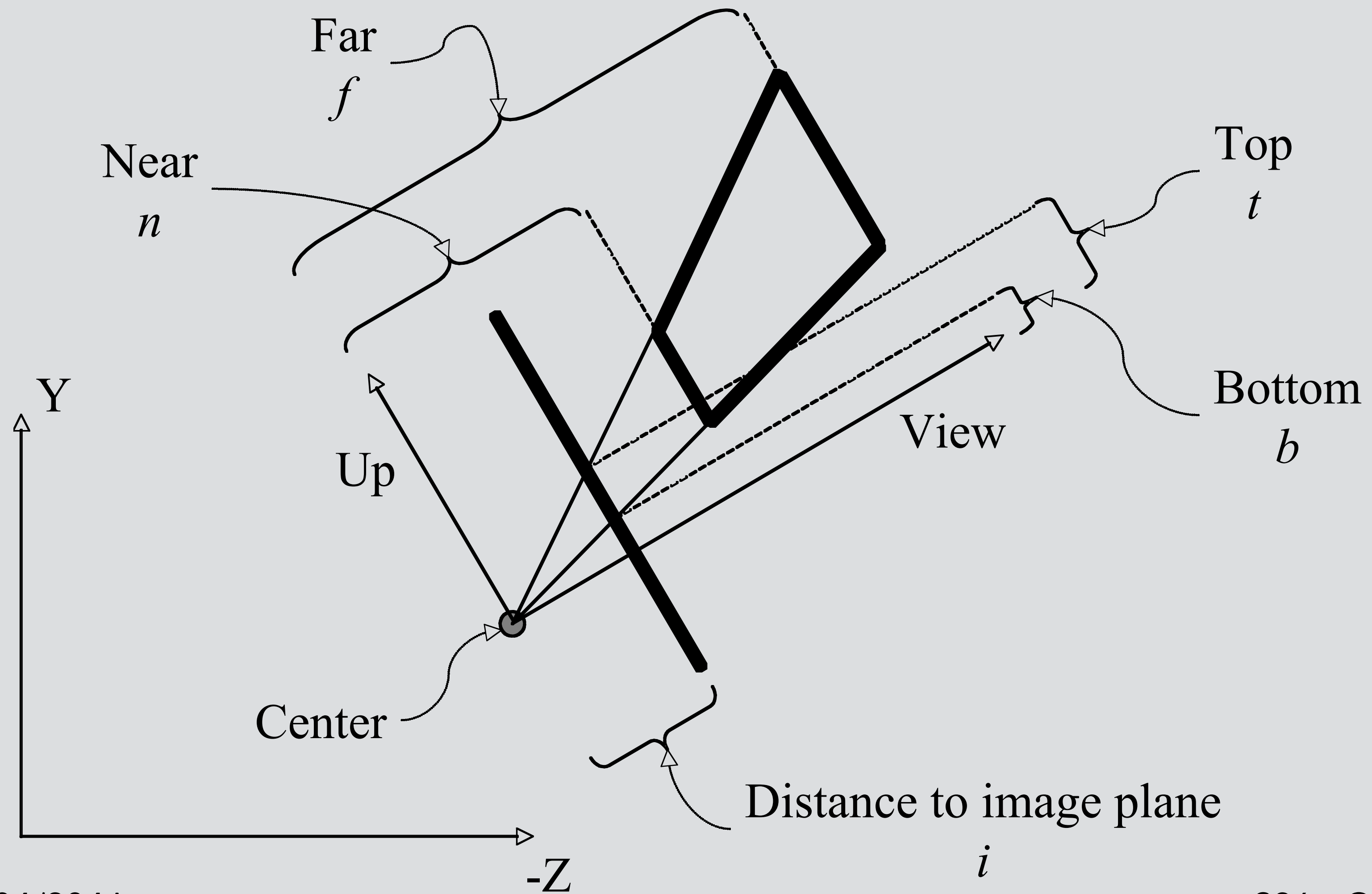
- <https://tinyurl.com/184class>



Word of the day:

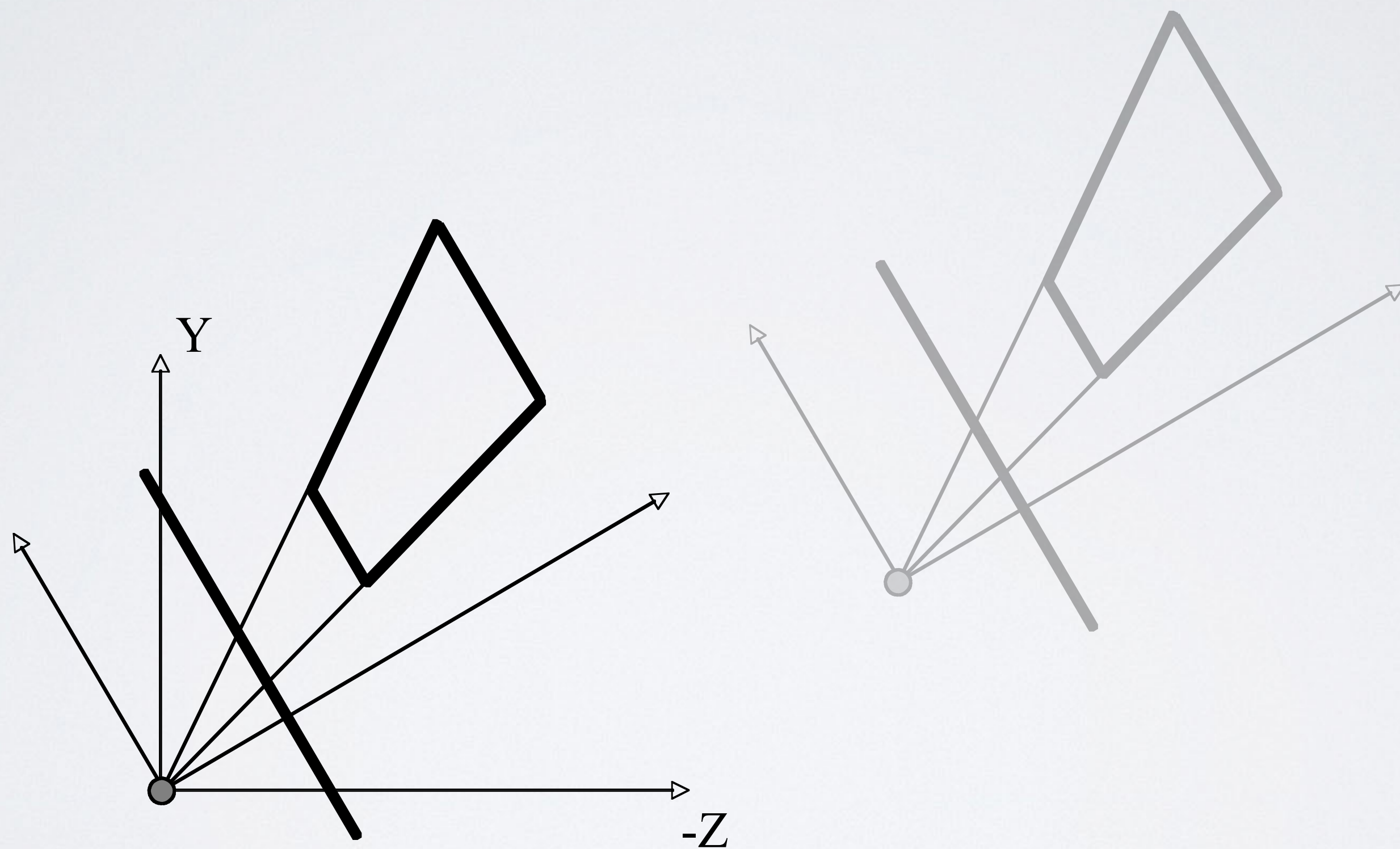
- ????

Perspective Projection Alternatives



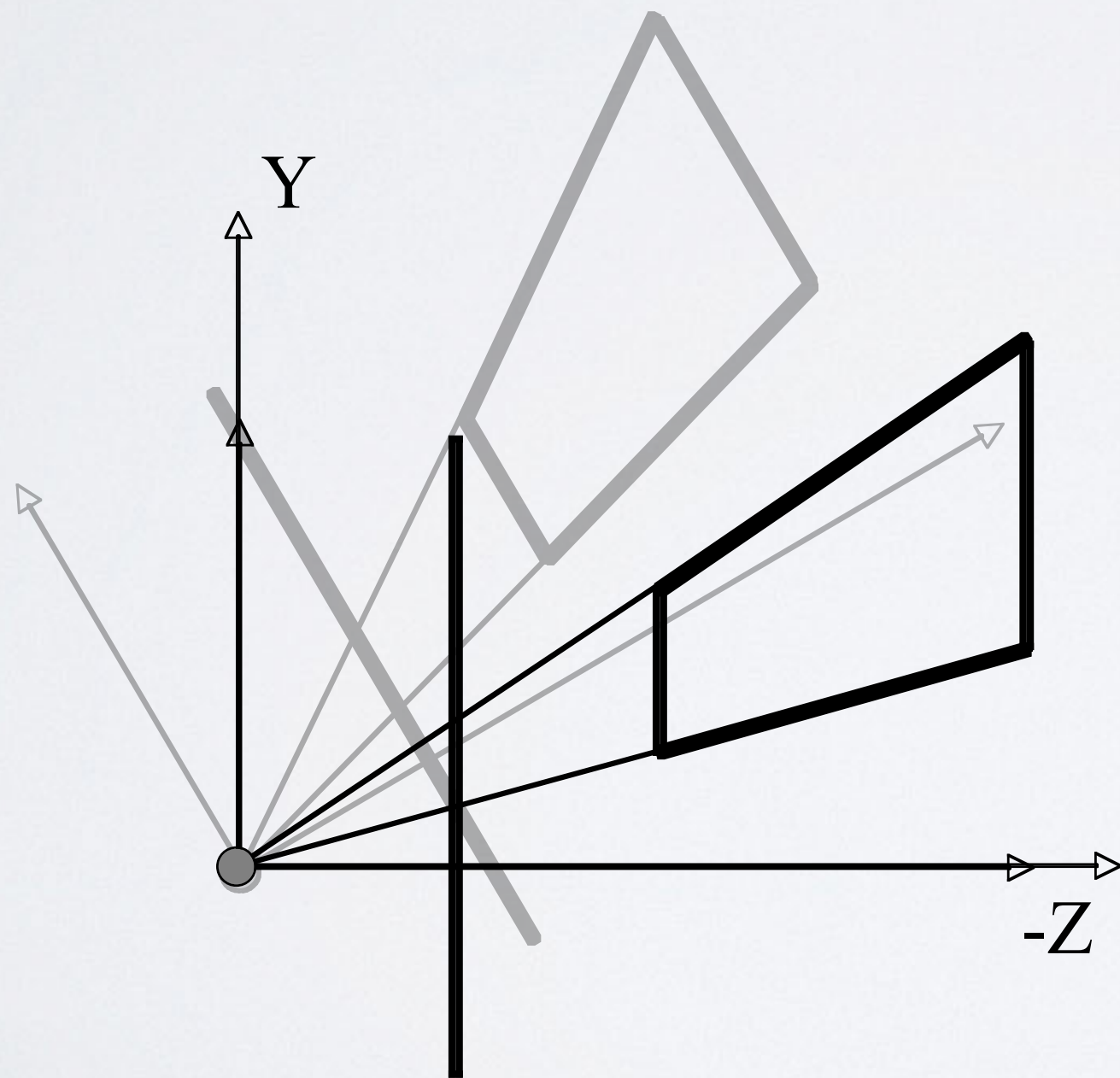
Perspective Projection Alternatives

- Step 1: Translate *center* to origin



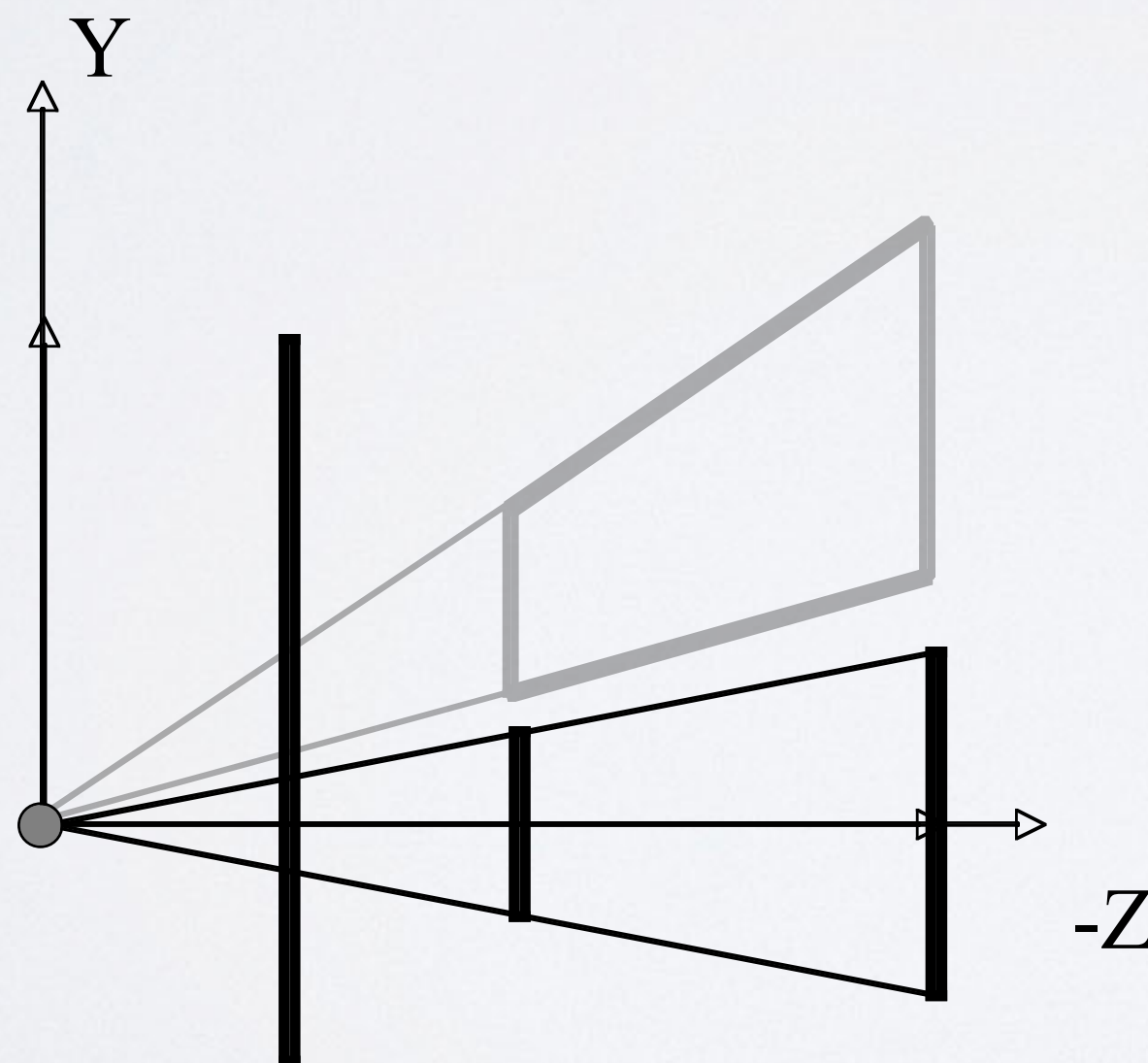
Perspective Projection Alternatives

- Step 1: Translate **center** to origin
- Step 2: Rotate **view** to **-Z**, *up* to **+Y**



Perspective Projection Alternatives

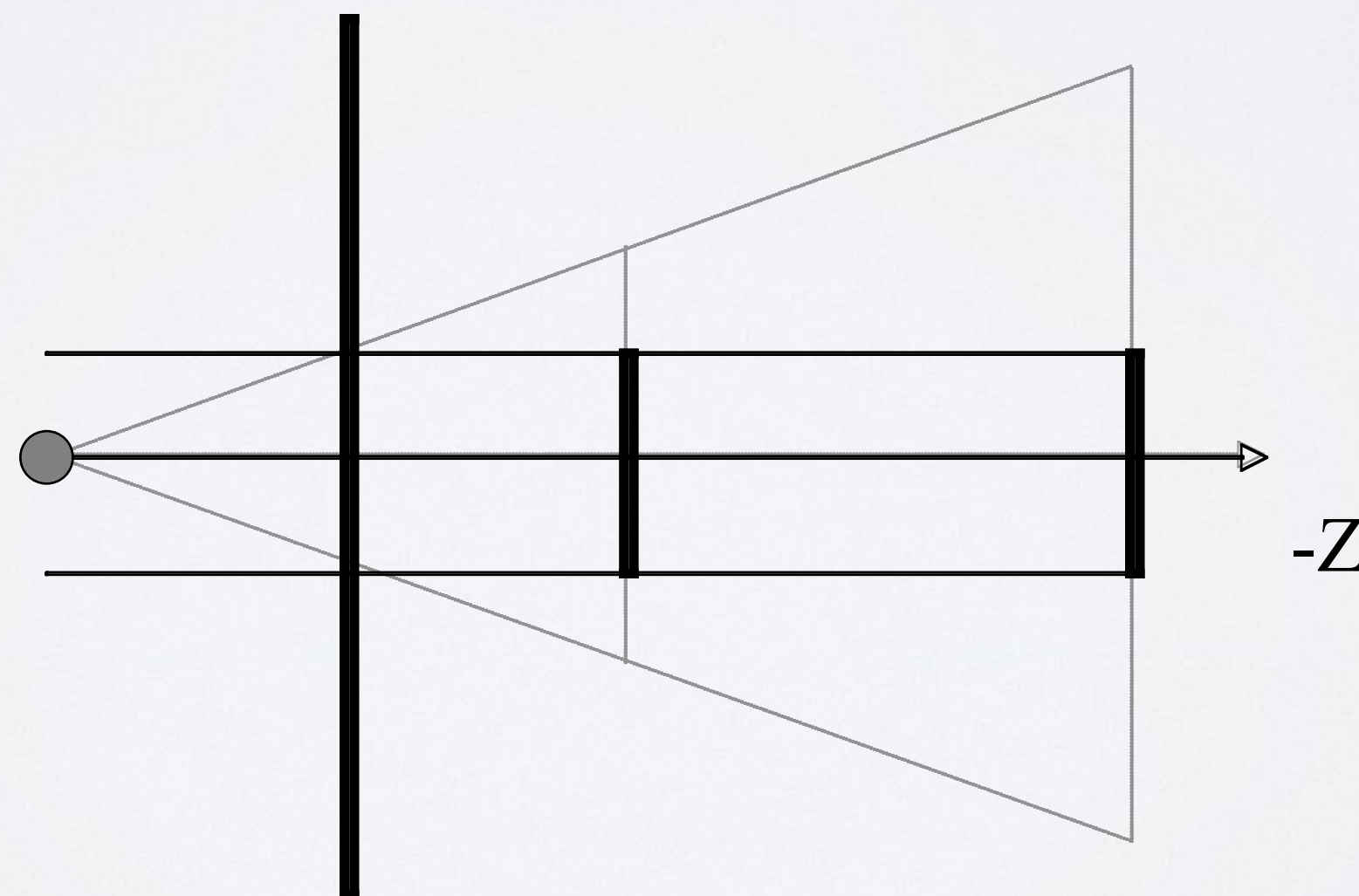
- Step 1: Translate **center** to origin
- Step 2: Rotate **view** to **-Z**, *up* to **+Y**
- Step 3: Shear center-line to **-Z** axis



Perspective Projection Alternatives

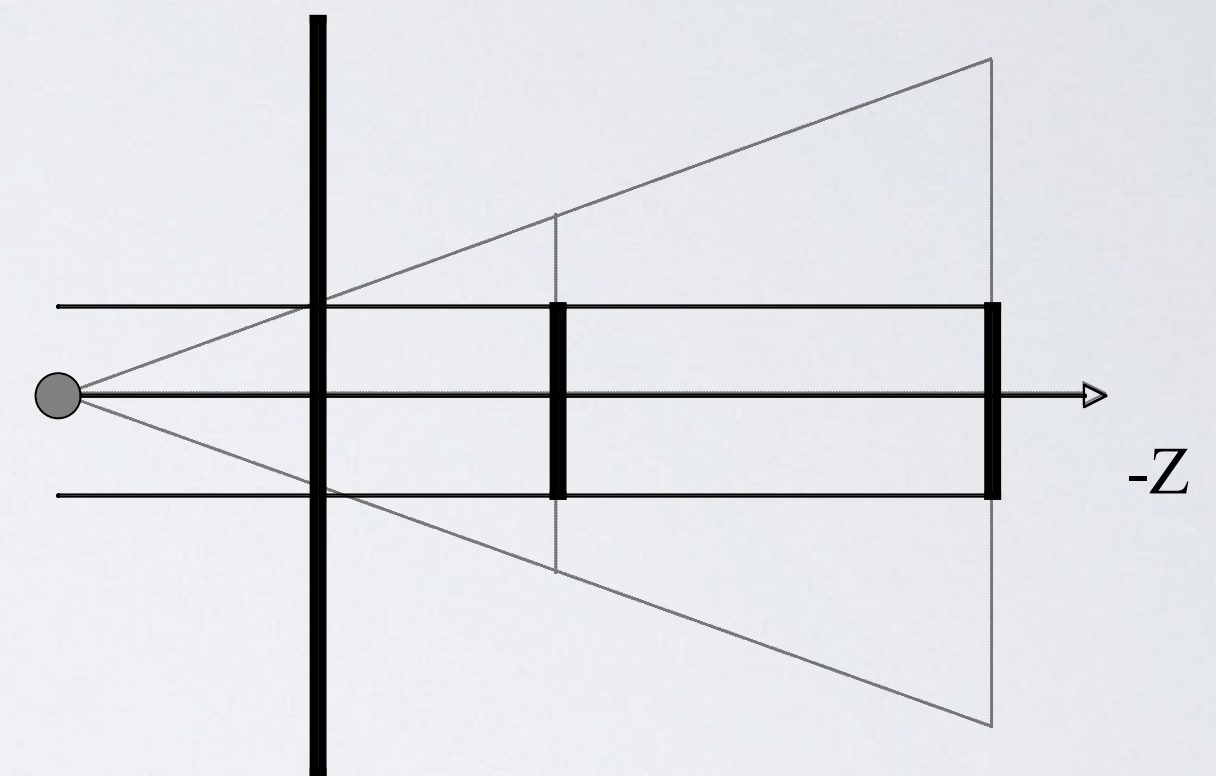
- Step 1: Translate **center** to origin
- Step 2: Rotate **view** to **-Z**, *up* to **+Y**
- Step 3: Shear center-line to **-Z** axis
- Step 4: Perspective

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{i+f}{i} & f \\ 0 & 0 & \frac{-1}{i} & 0 \end{bmatrix}$$



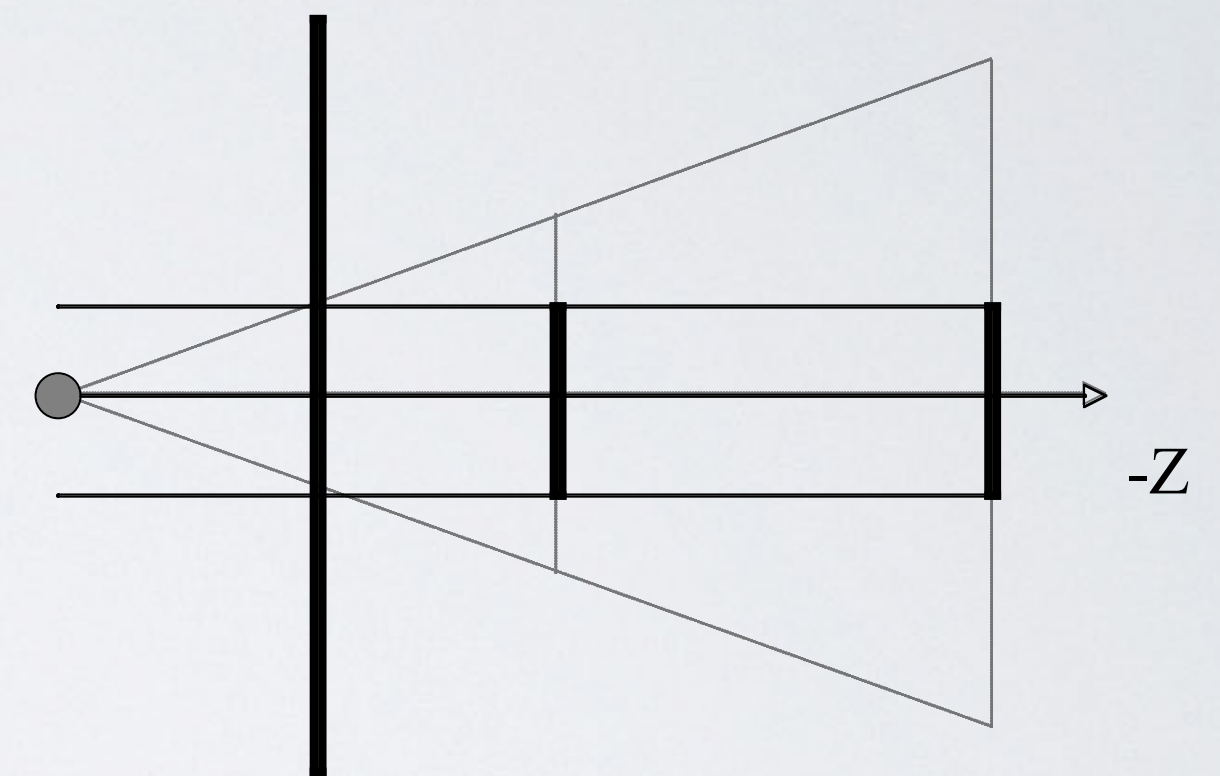
Perspective Projection Alternatives

- Step 4: Perspective
 - Points at $z=-i$ stay at $z=-i$
 - Points at $z=-f$ stay at $z=-f$
 - Points at $z=0$ goto $z=\pm\infty$
 - Points at $z=-\infty$ goto $z=-(i+f)$
- x and y values divided by $-z/i$
- Straight lines stay straight
- Depth ordering preserved in $[-i, -f]$
- Movement along lines distorted



Perspective Projection Alternatives

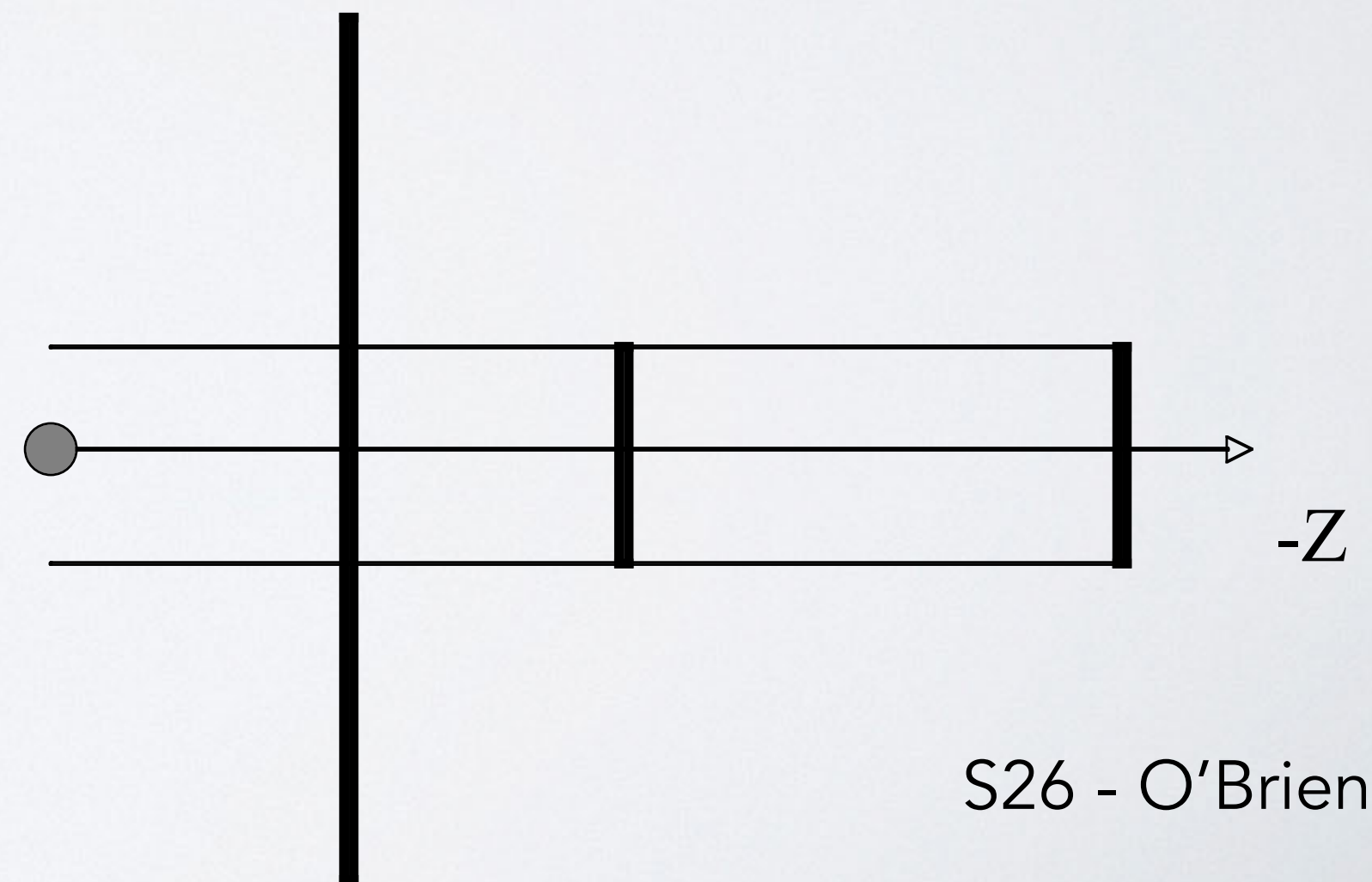
- Step 4: Perspective
 - Points at $z=-i$ stay at $z=-i$
 - Points at $z=-f$ stay at $z=-f$
 - Points at $z=0$ goto $z=\pm\infty$
 - Points at $z=-\infty$ goto $z=-(i+f)$
- x and y values divided by $-z/i$
- Straight lines stay straight
- Depth ordering preserved in $[-i, -f]$
- Movement along lines distorted



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{i+f}{i} & f \\ 0 & 0 & \frac{-1}{i} & 0 \end{bmatrix}$$

Perspective Projection Alternatives

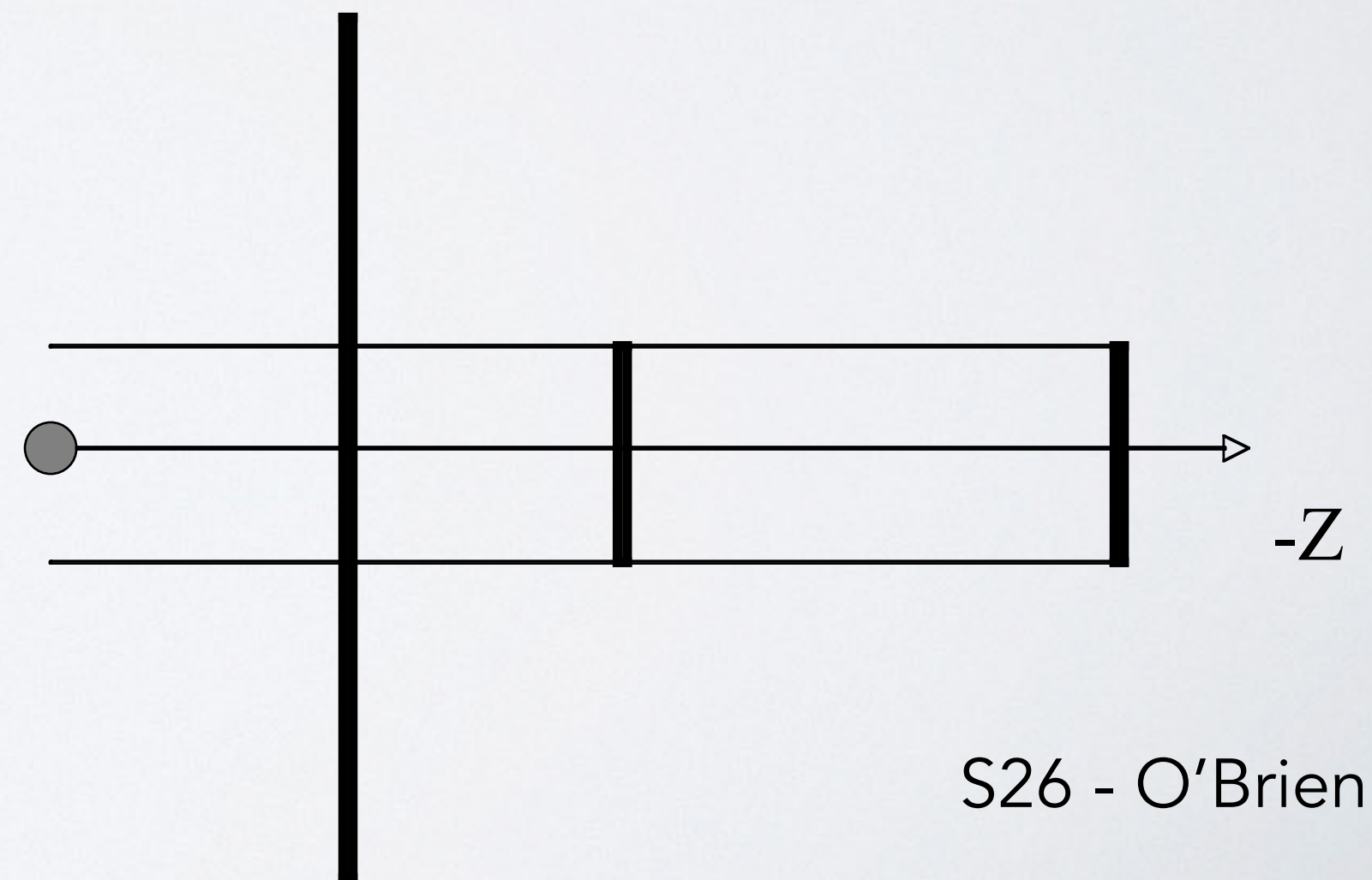
- Step 1: Translate **center** to orange
- Step 2: Rotate **view** to **-Z**, *up* to **+Y**
- Step 3: Shear center-line to **-Z** axis
- Step 4: Perspective
- Step 5: center view volume
- Step 6: scale to canonical size



Perspective Projection Alternatives

- Step 1: Translate **center** to orange
- Step 2: Rotate **view** to **-Z**, **up** to **+Y** } $\mathbf{M}_v$
- Step 3: Shear center-line to **-Z** axis } $\mathbf{M}_p$
- Step 4: Perspective
- Step 5: center view volume
- Step 6: scale to canonical size } $\mathbf{M}_o$

$$\mathbf{M} = \mathbf{M}_o \cdot \mathbf{M}_p \cdot \mathbf{M}_v$$



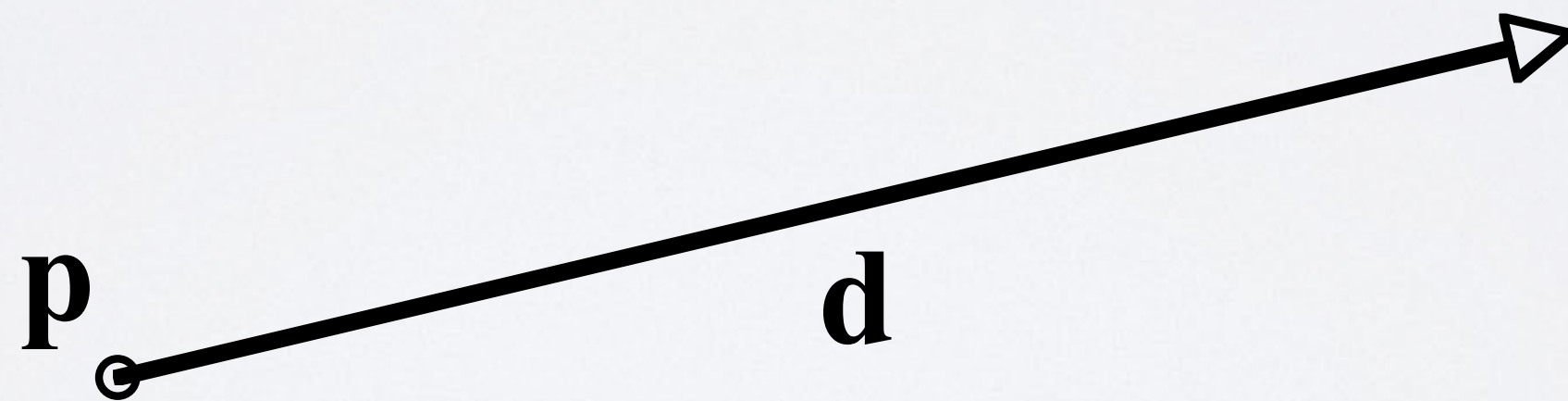
Perspective Projection Alternatives

- There are other ways to set up the projection matrix
 - View plane at $z=0$ zero
 - Looking down another axis
 - *etc...*
- Functionally equivalent

Vanishing Points

- Consider a ray:

$$\mathbf{r}(t) = \mathbf{p} + t \mathbf{d}$$



Vanishing Points

- Ignore **Z** part of matrix
- **X** and **Y** will give location in image plane
- Assume image plane at $z=-i$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \text{whatever} & & & \\ 0 & 0 & -1 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} I_x \\ I_y \\ I_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Vanishing Points

$$\begin{bmatrix} I_x \\ I_y \\ I_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ -z \end{bmatrix}$$

$$\begin{bmatrix} I_x / I_w \\ I_y / I_w \end{bmatrix} = \begin{bmatrix} -x / z \\ -y / z \end{bmatrix}$$

Vanishing Points

- Assume

$$d_z = -1$$

$$\begin{bmatrix} I_x / I_w \\ I_y / I_w \end{bmatrix} = \begin{bmatrix} -x / z \\ -y / z \end{bmatrix} = \begin{bmatrix} \frac{p_x + td_x}{-p_z + t} \\ \frac{p_y + td_y}{-p_z + t} \end{bmatrix}$$

$$\lim_{t \rightarrow \pm\infty} = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

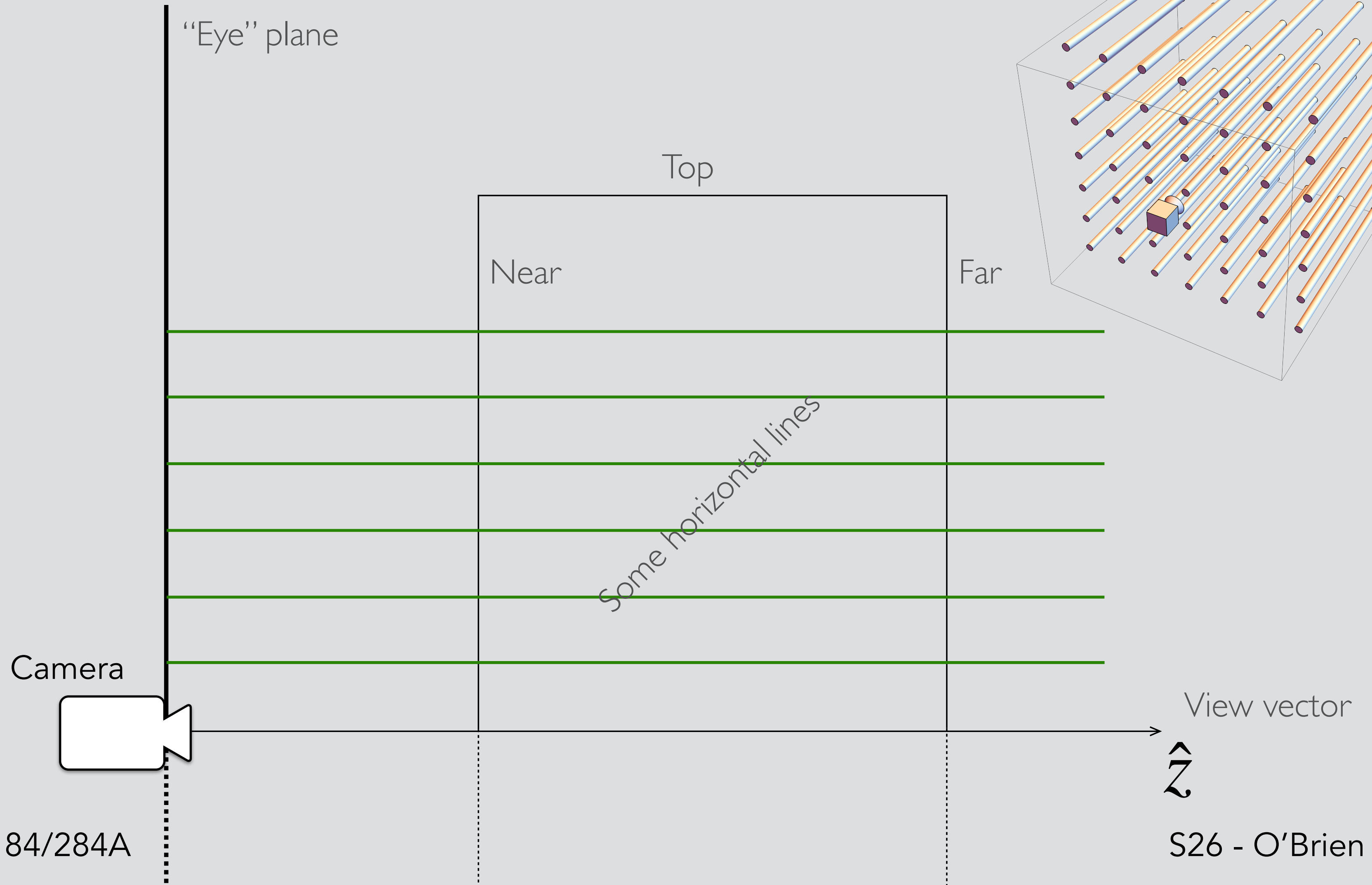
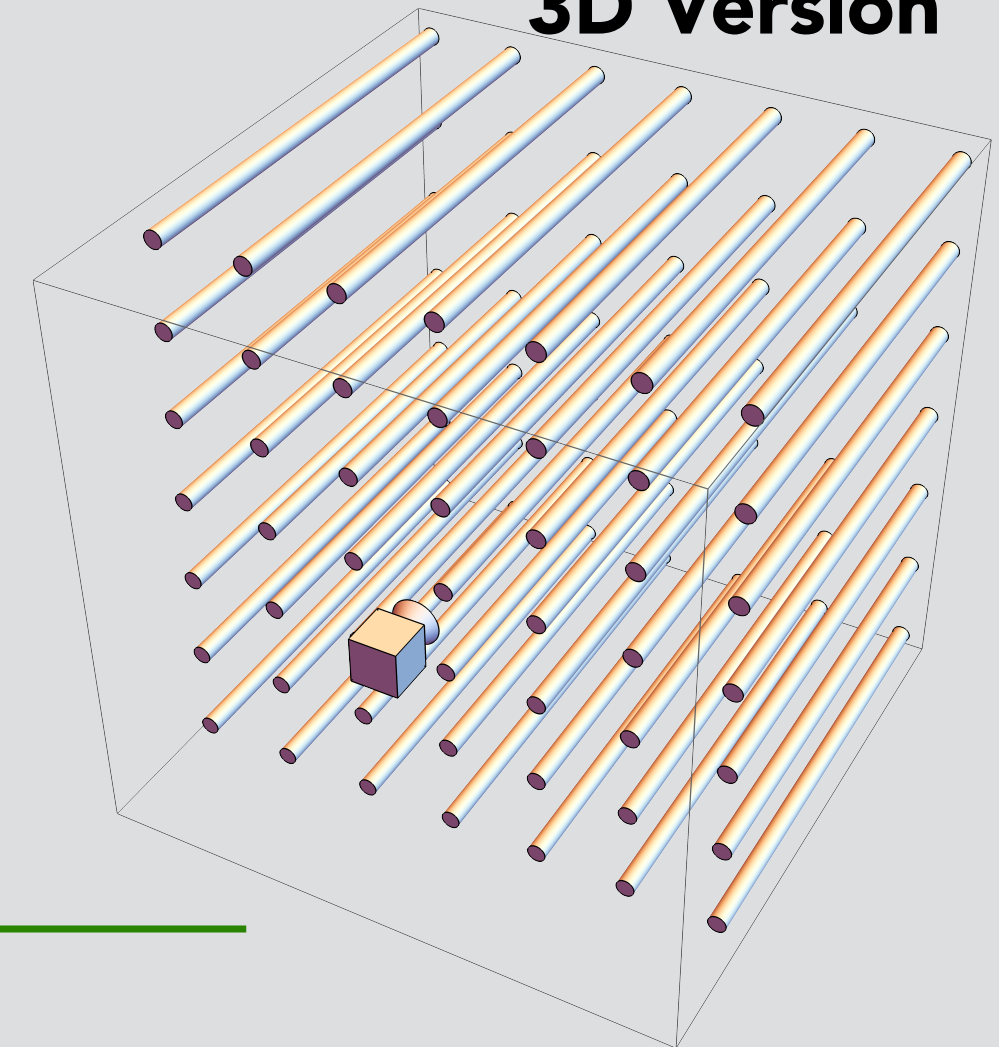
Vanishing Points

$$\lim_{t \rightarrow \pm\infty} = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

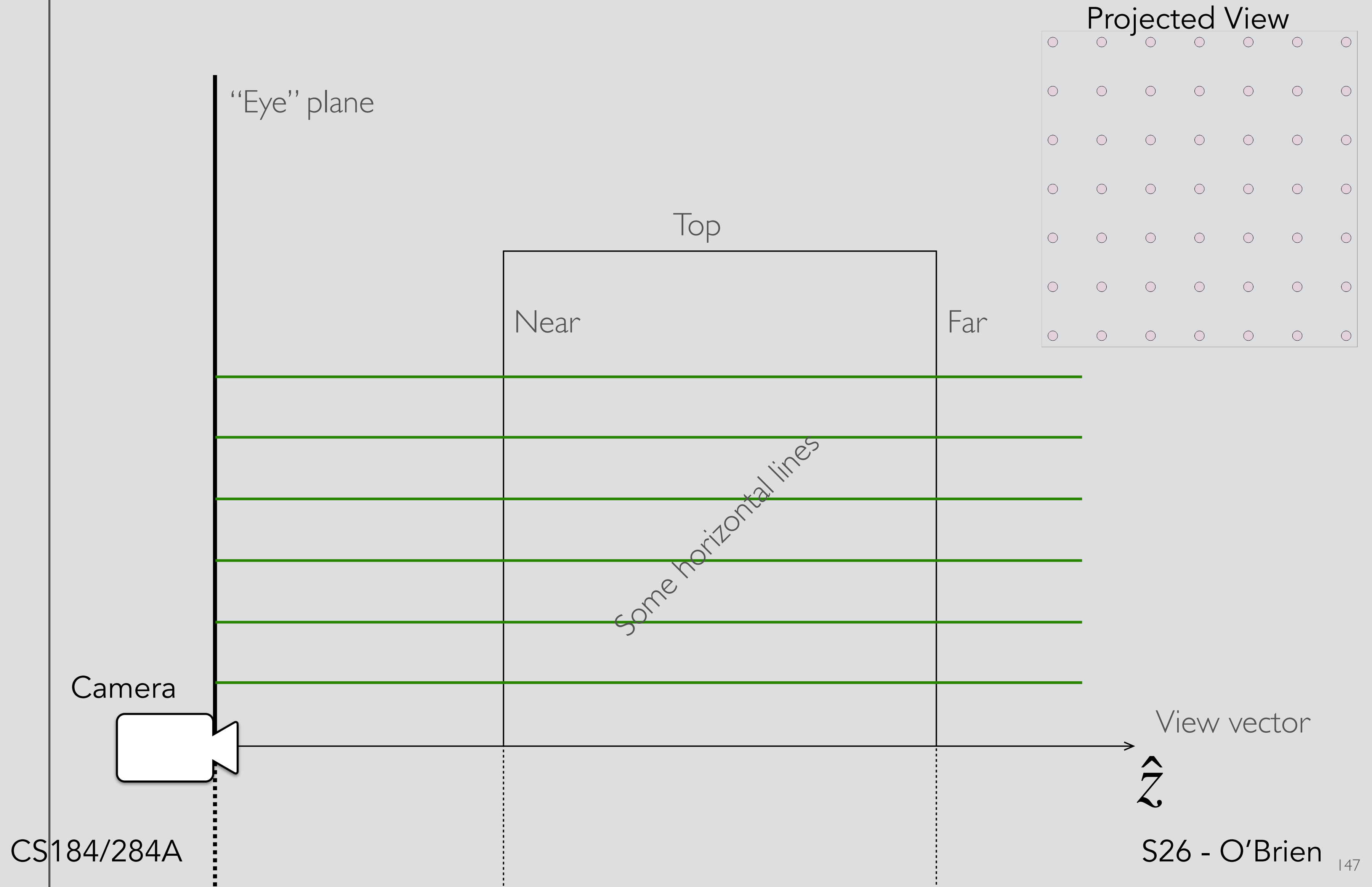
- All lines in direction $\mathbf{d}$ converge to same point in the image plane -- the vanishing point
- Every point in plane is a v.p. for some set of lines
- Lines parallel to image plane ($d_z \Rightarrow 0$) vanish at infinity

Perspective Projection

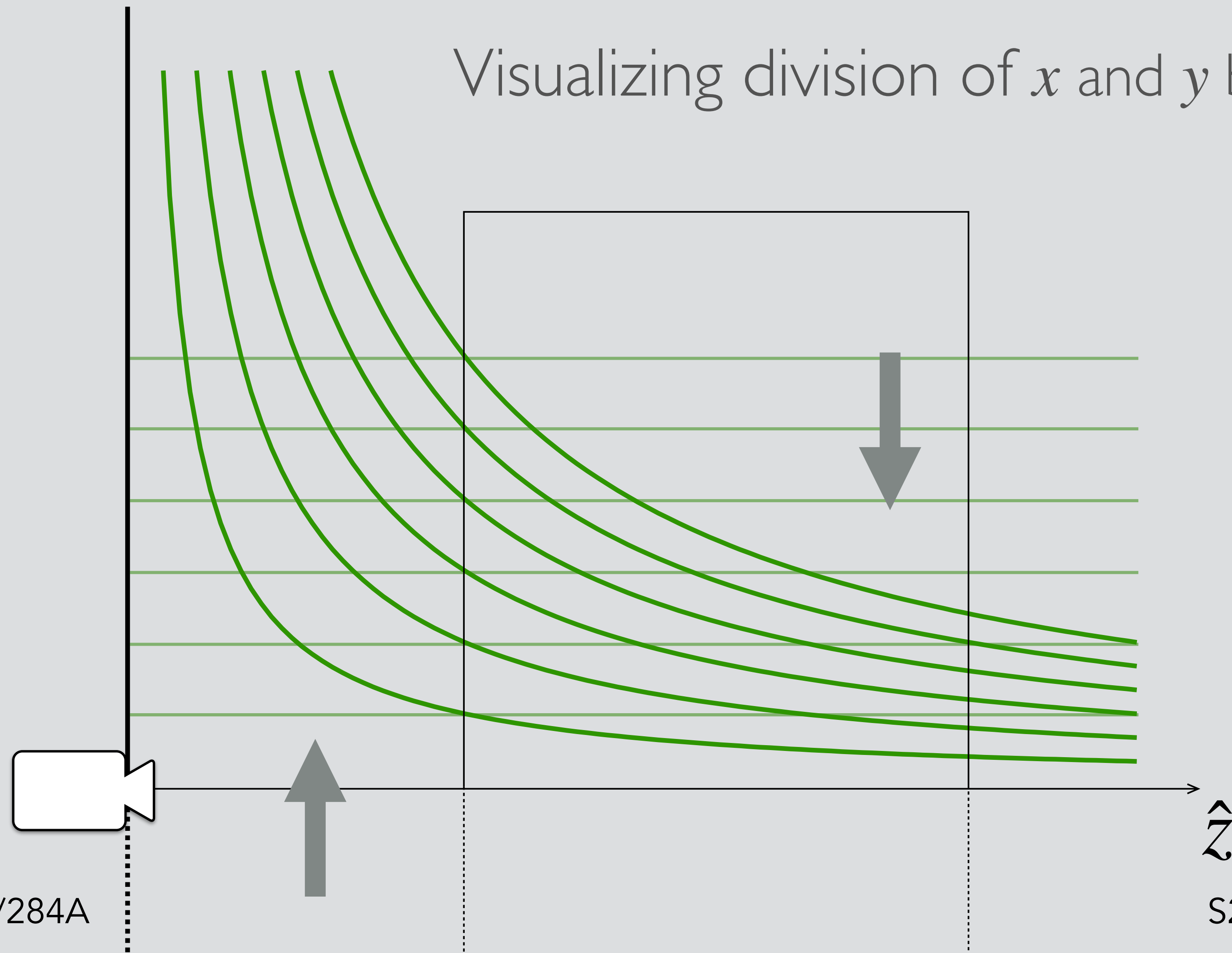
3D Version



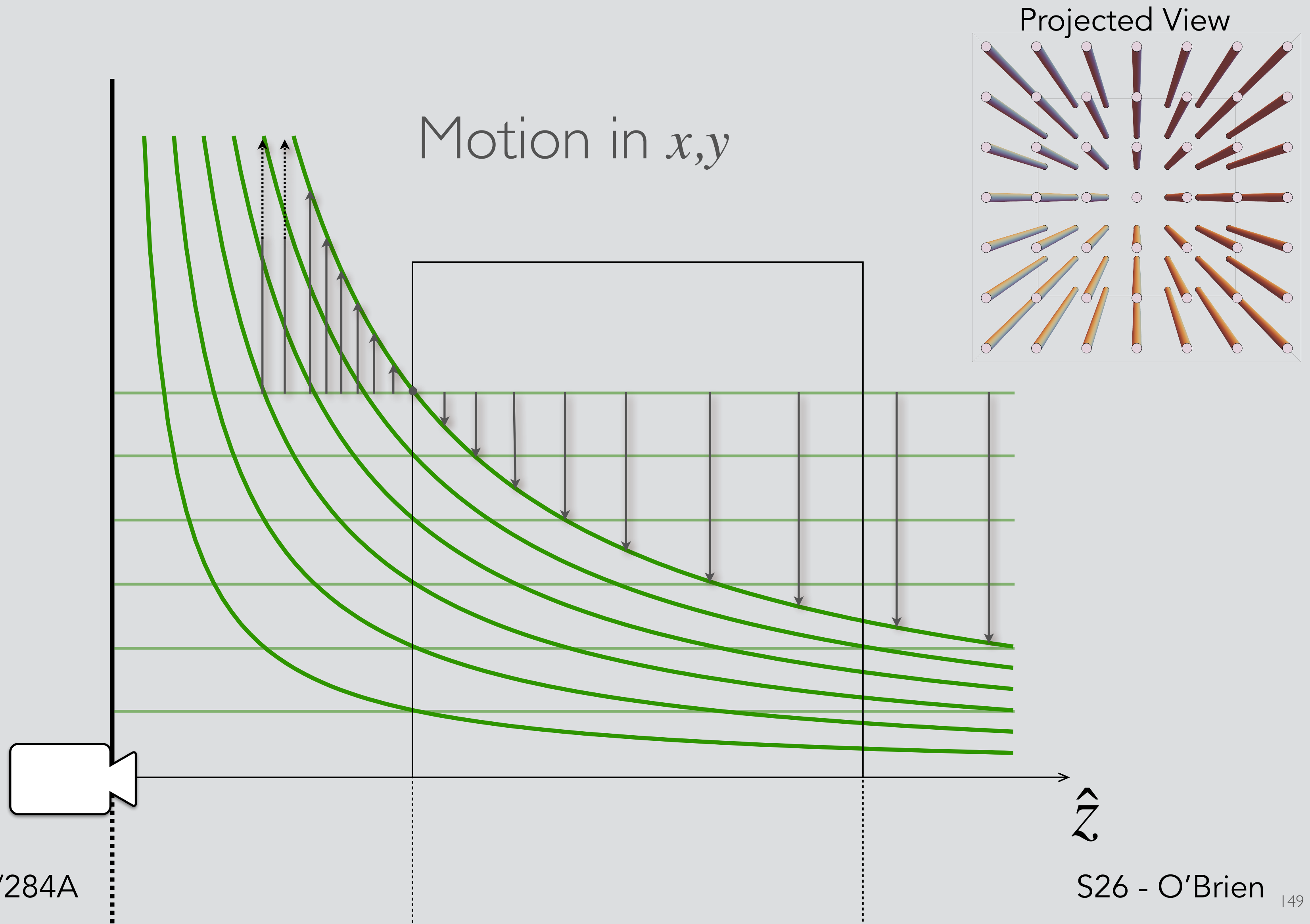
Perspective Projection



Perspective Projection

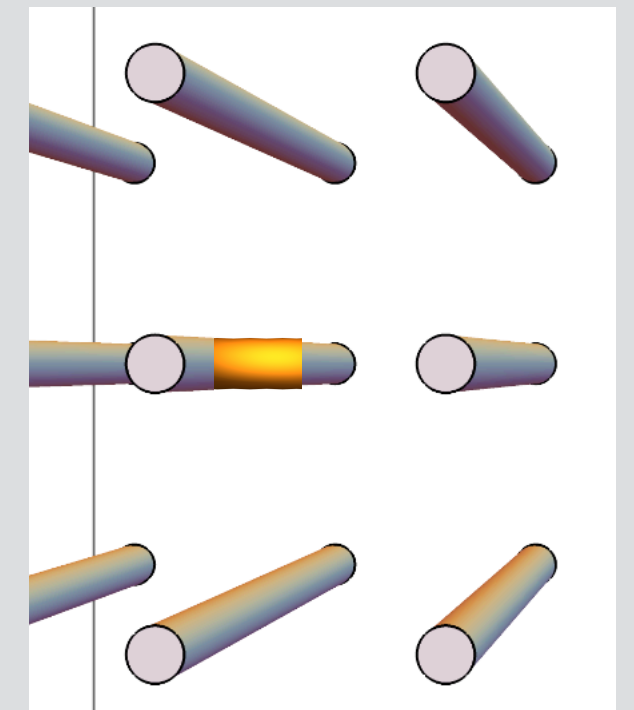
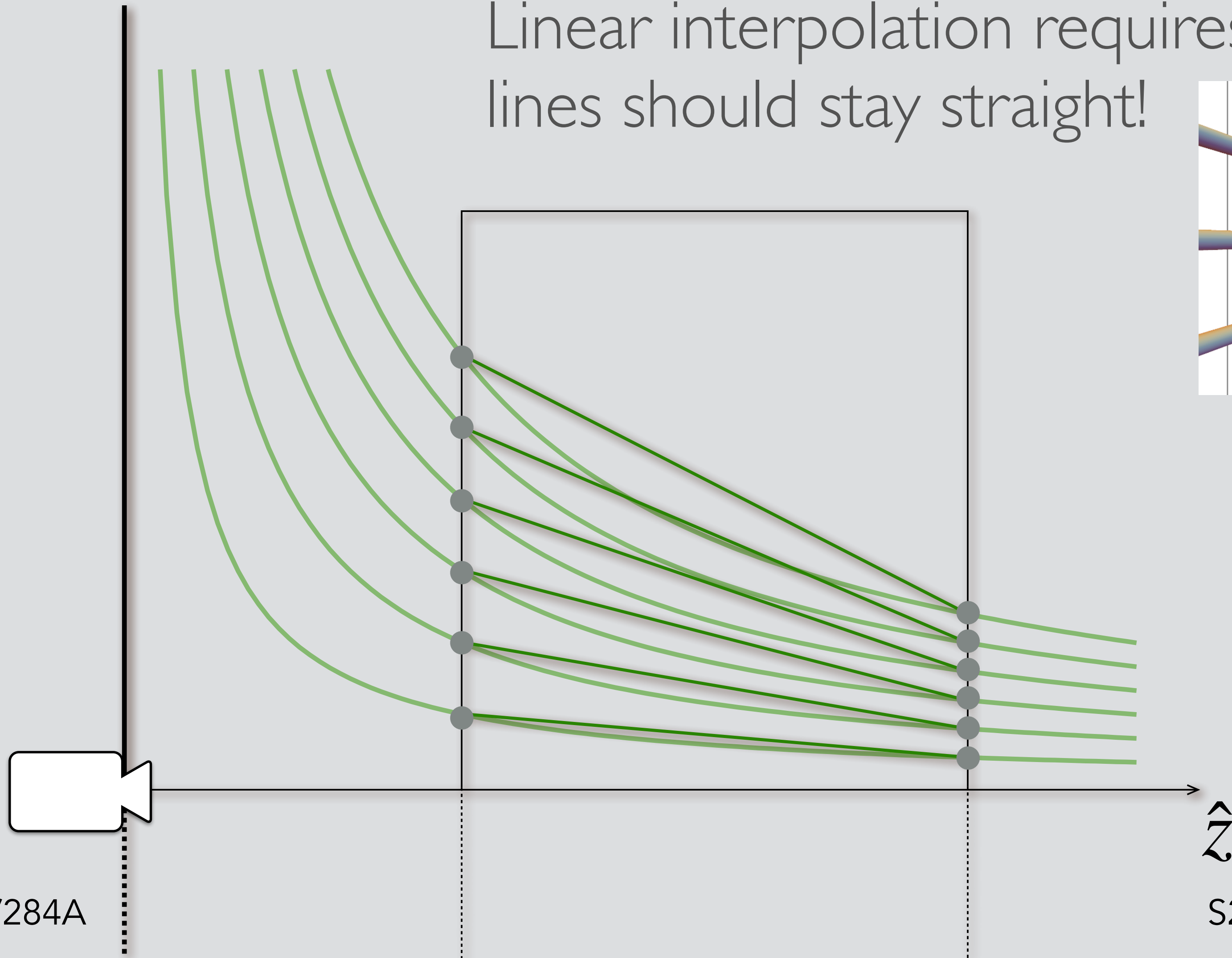


Perspective Projection



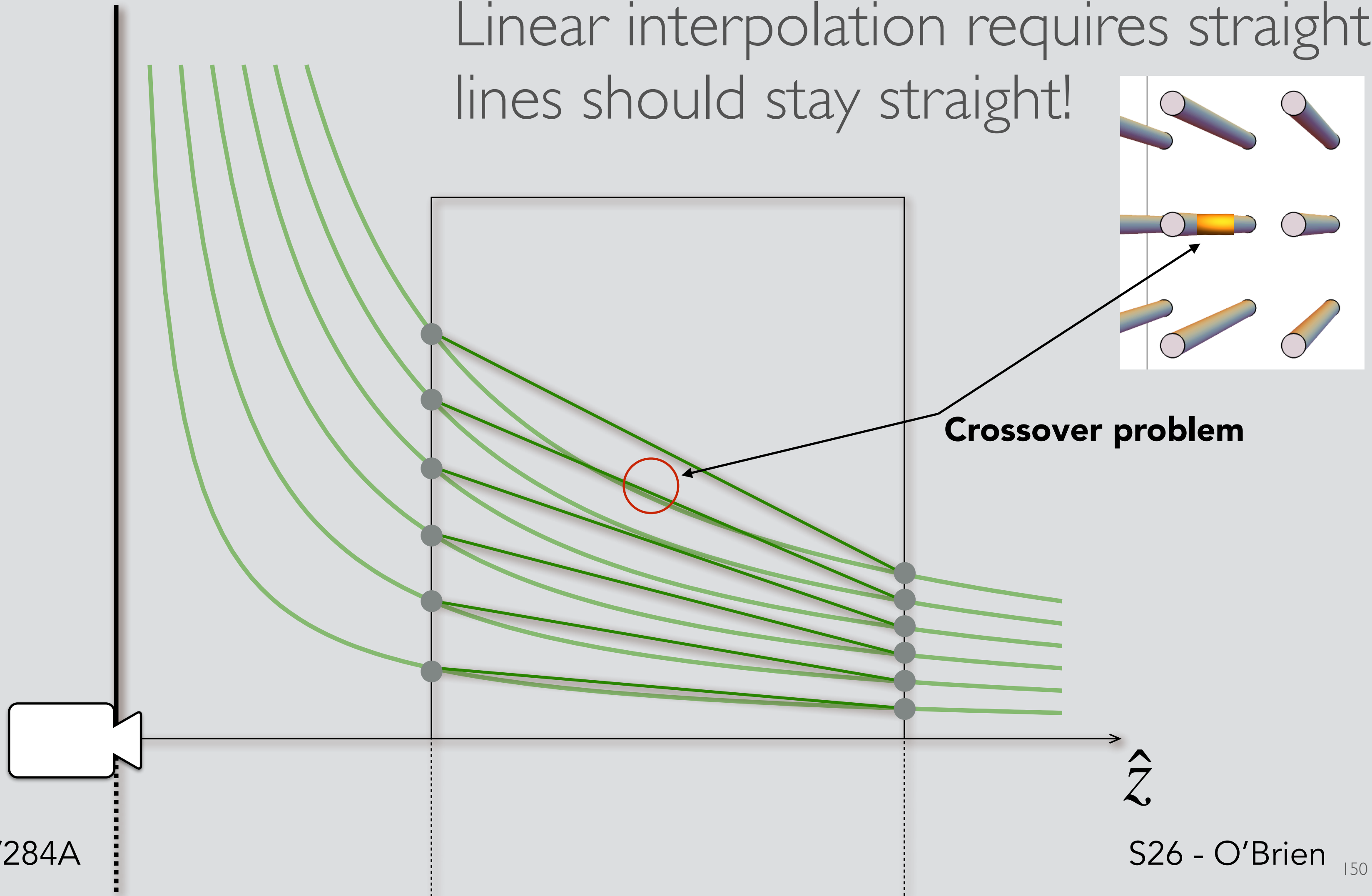
Perspective Projection

Linear interpolation requires straight lines should stay straight!



Perspective Projection

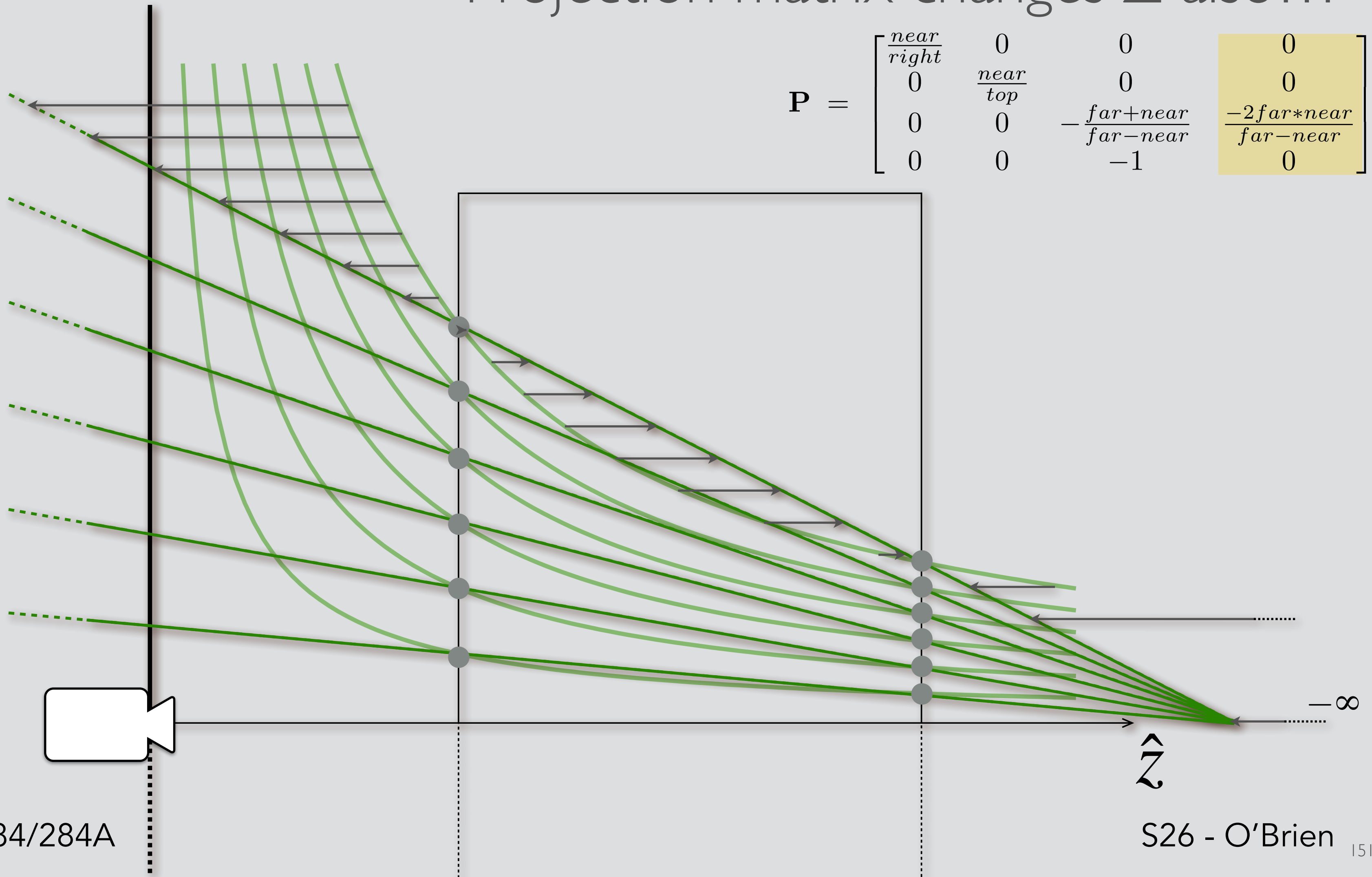
Linear interpolation requires straight lines should stay straight!



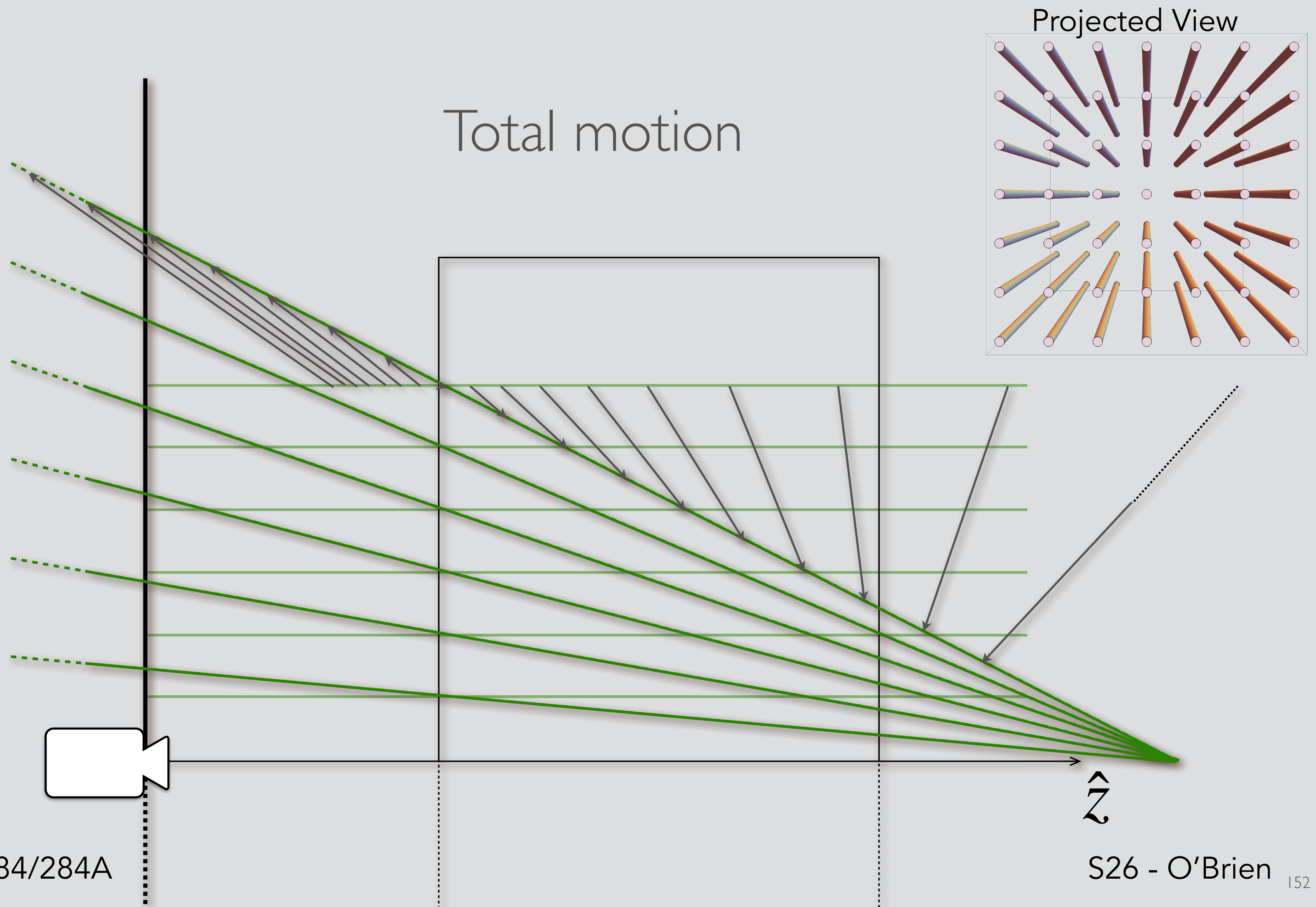
Perspective Projection

Projection matrix changes Z also...

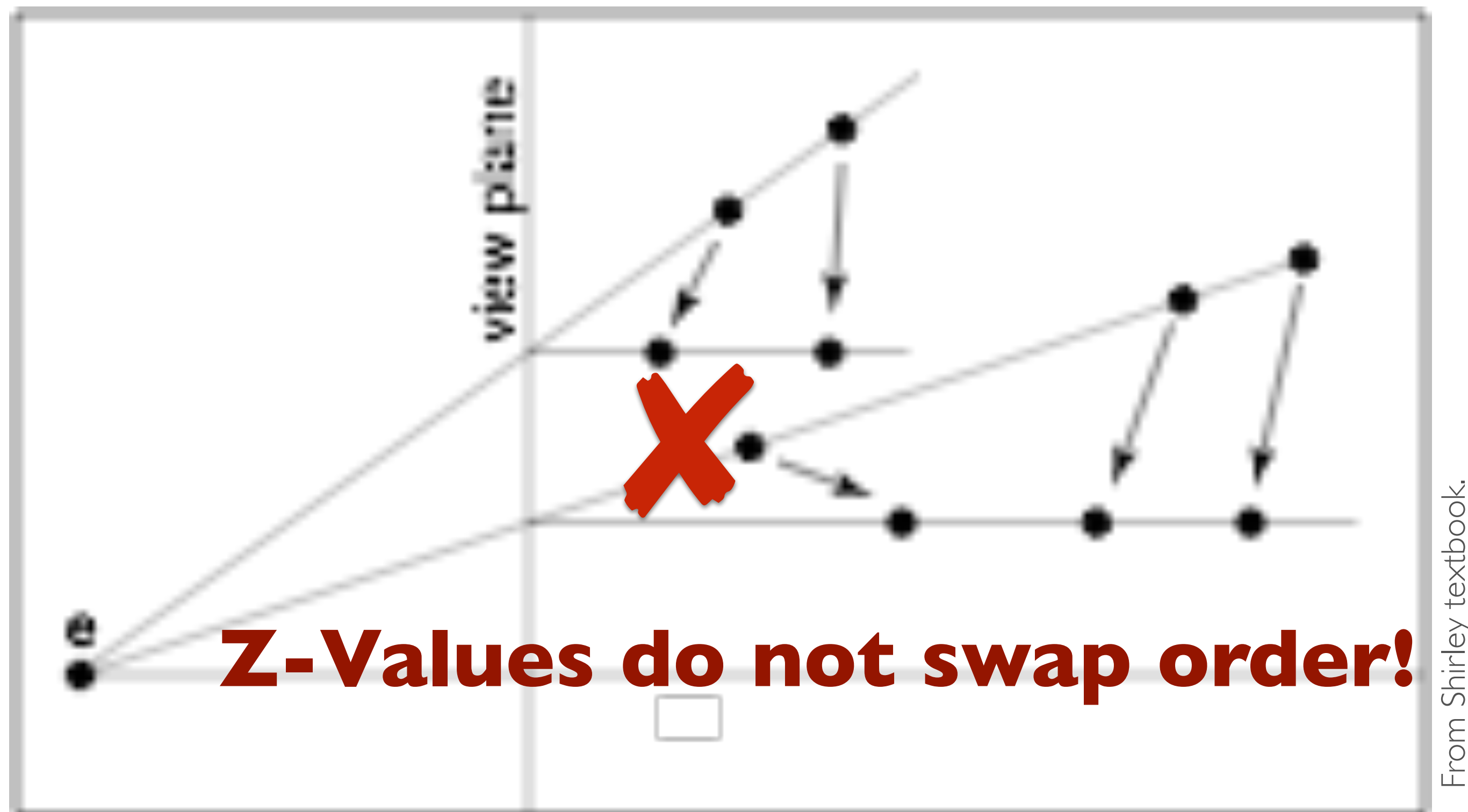
$$\mathbf{P} = \begin{bmatrix} \frac{\text{near}}{\text{right}} & 0 & 0 & 0 \\ 0 & \frac{\text{near}}{\text{top}} & 0 & 0 \\ 0 & 0 & -\frac{\text{far}+\text{near}}{\text{far}-\text{near}} & \frac{-2\text{far}*\text{near}}{\text{far}-\text{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



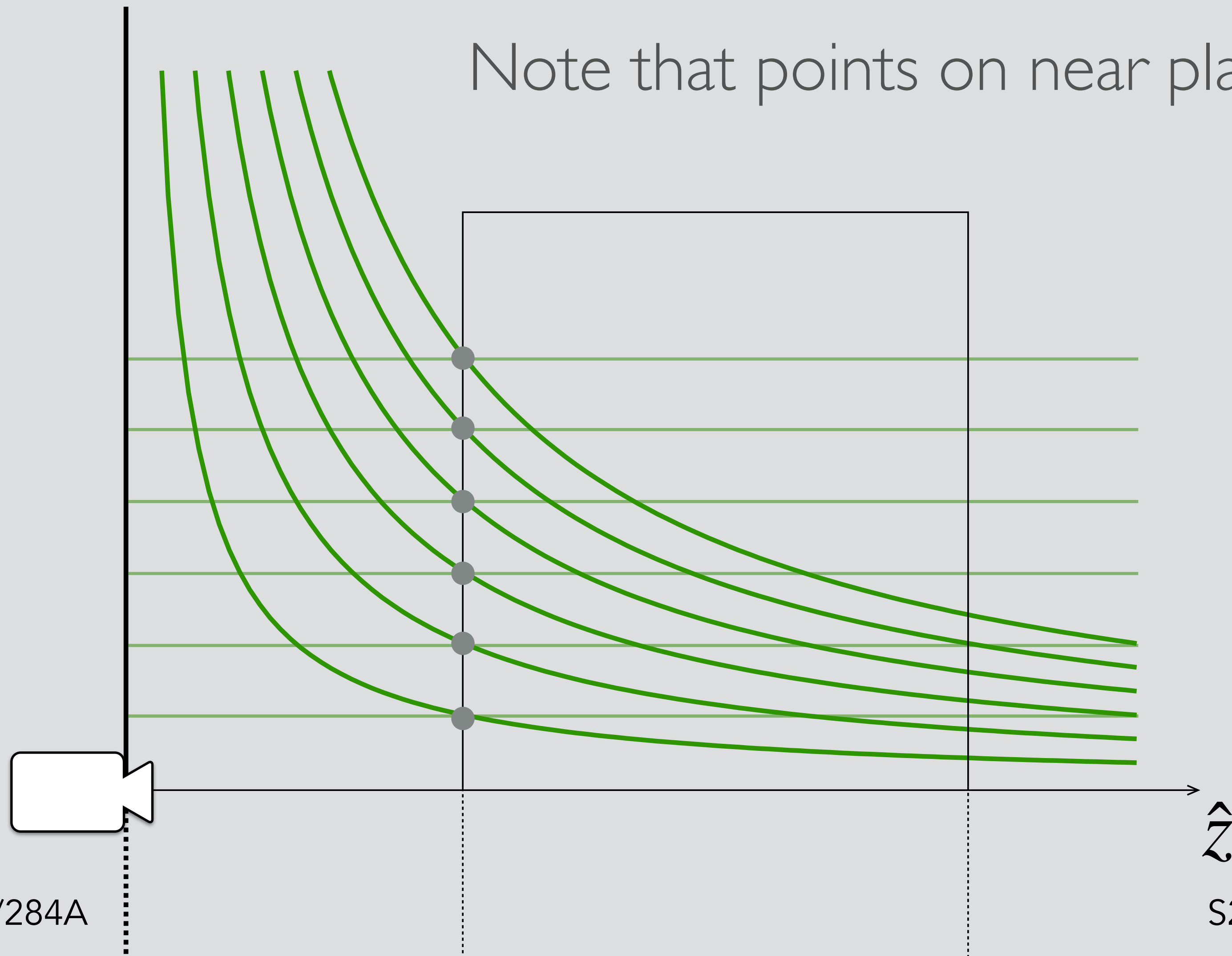
Perspective Projection



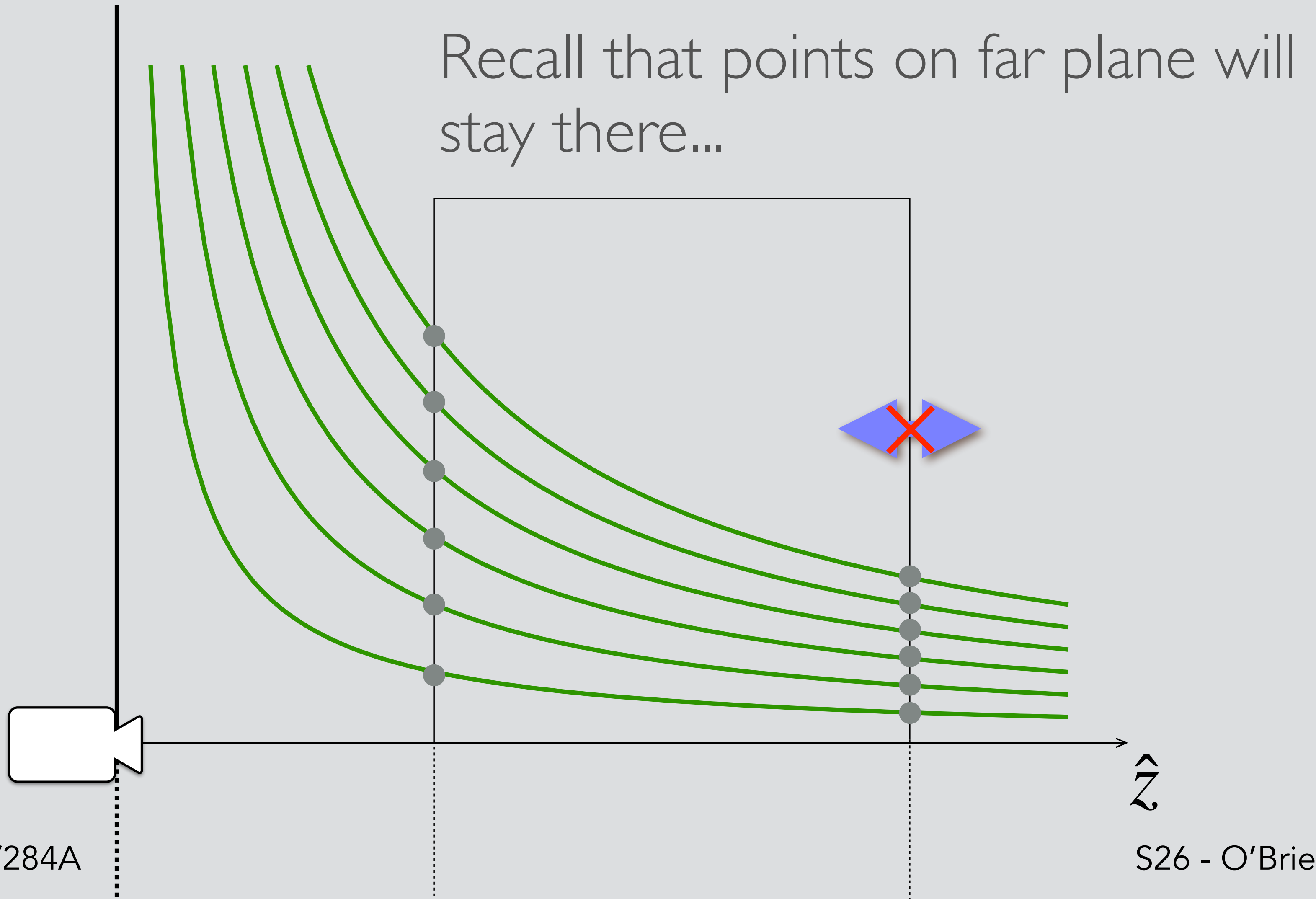
Perspective Projection



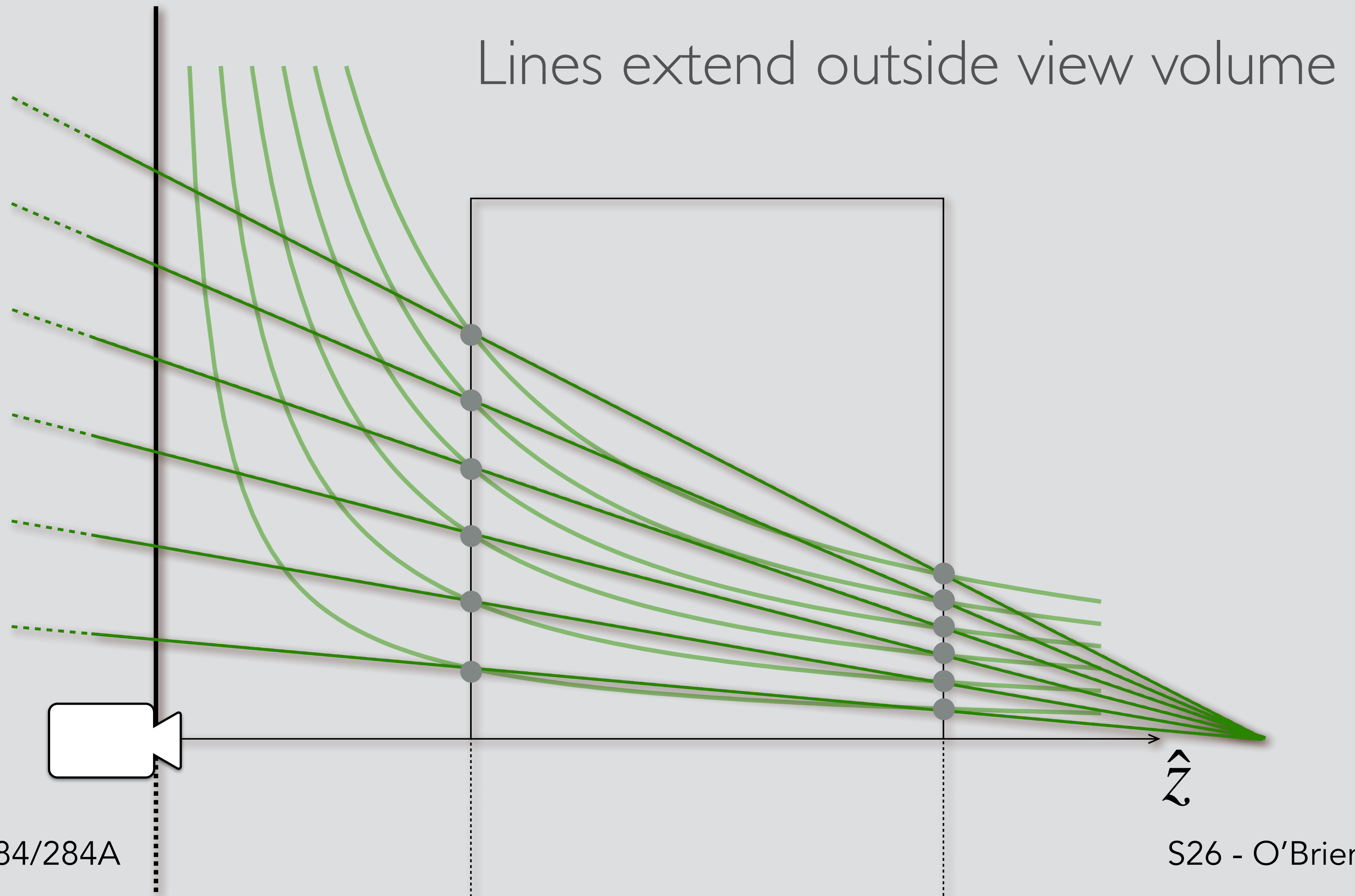
Perspective Projection



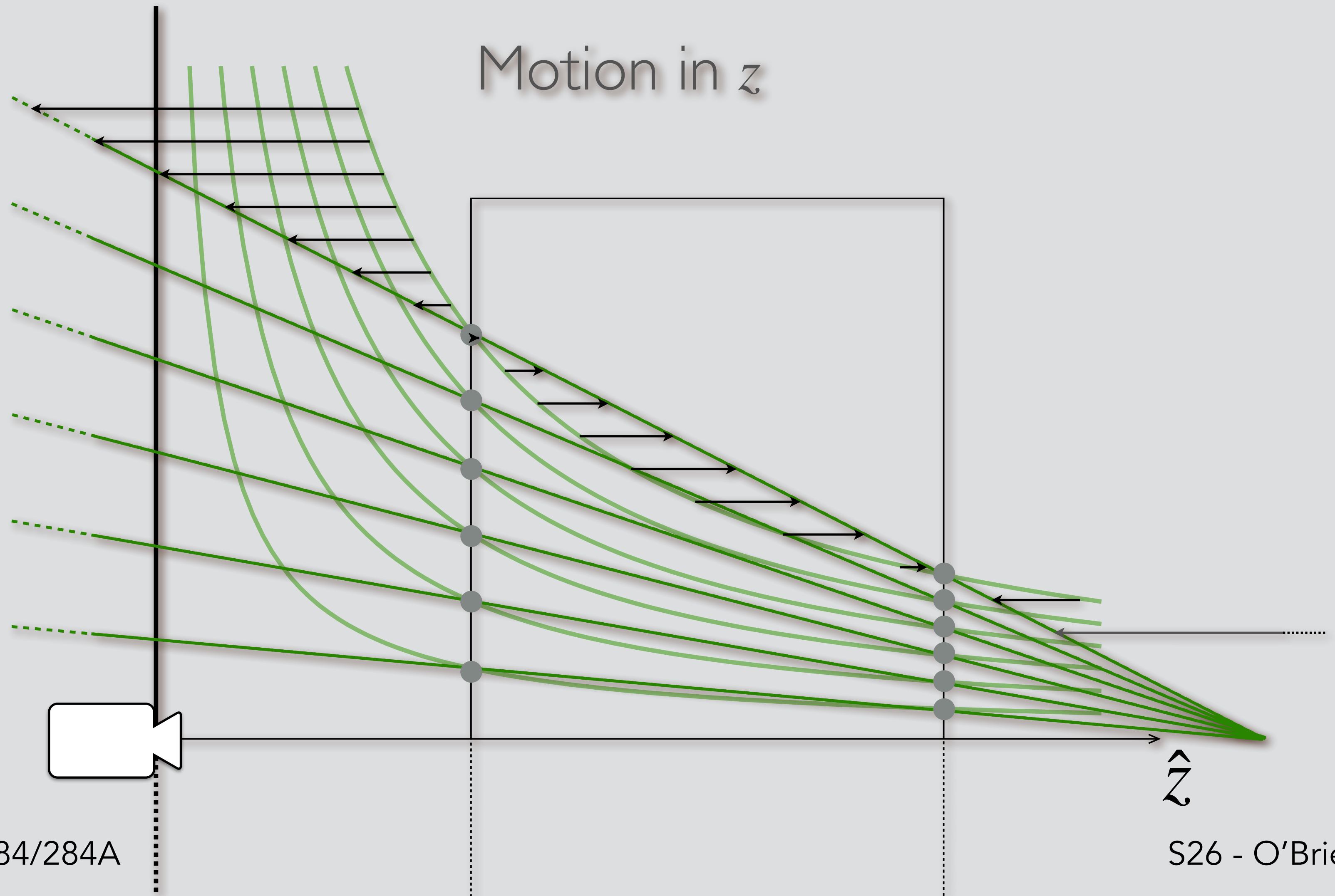
Perspective Projection



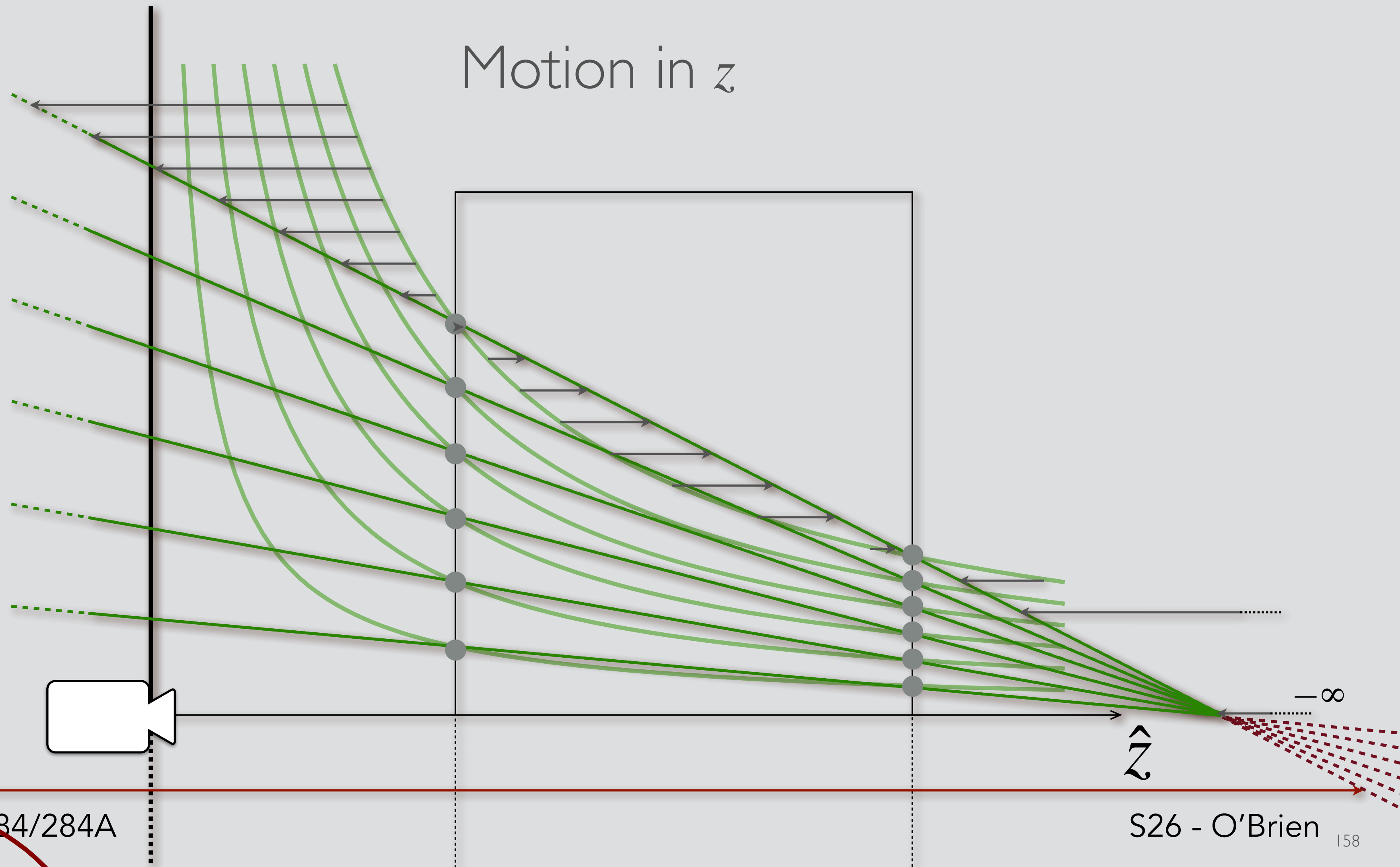
Perspective Projection



Perspective Projection



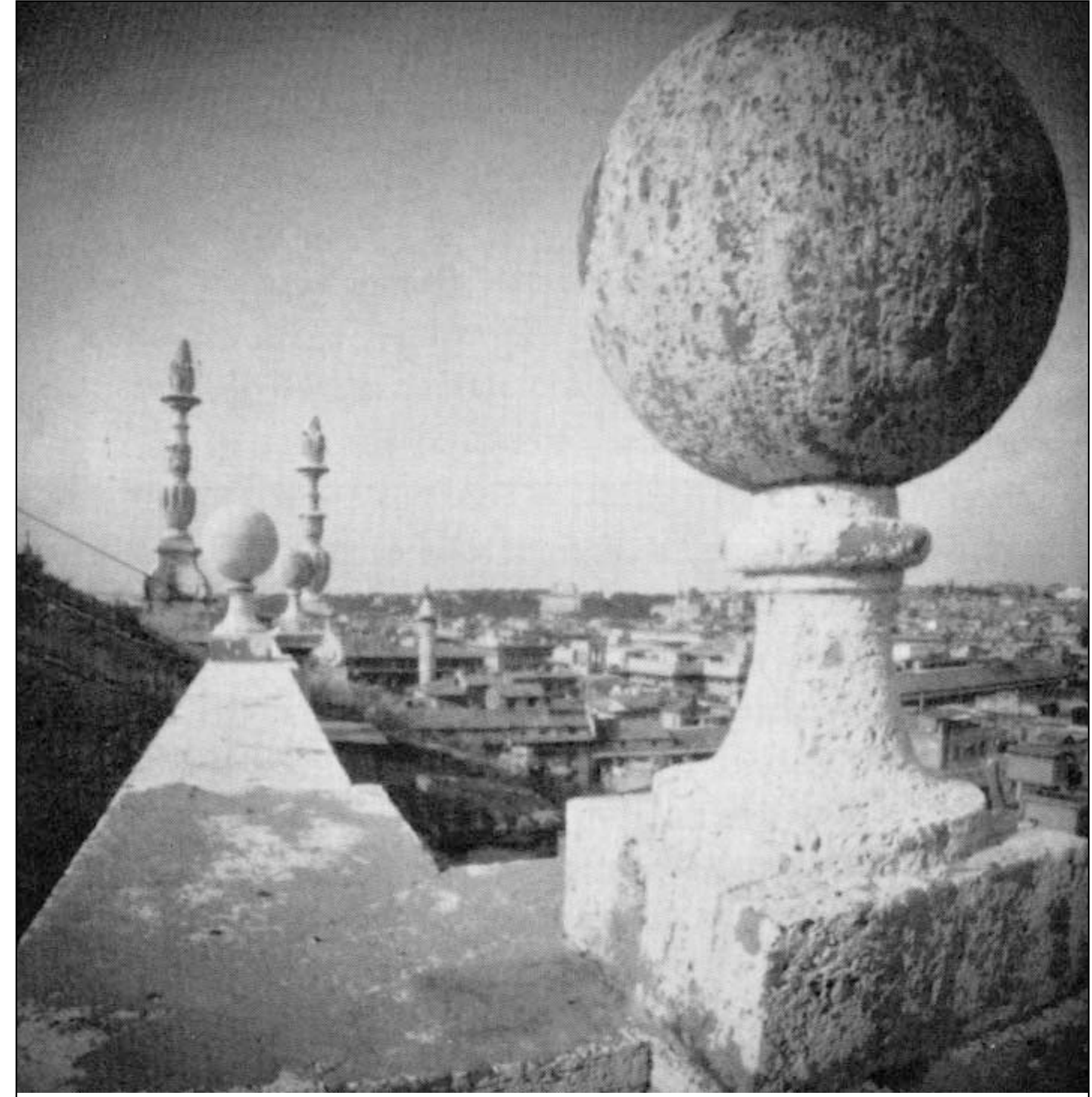
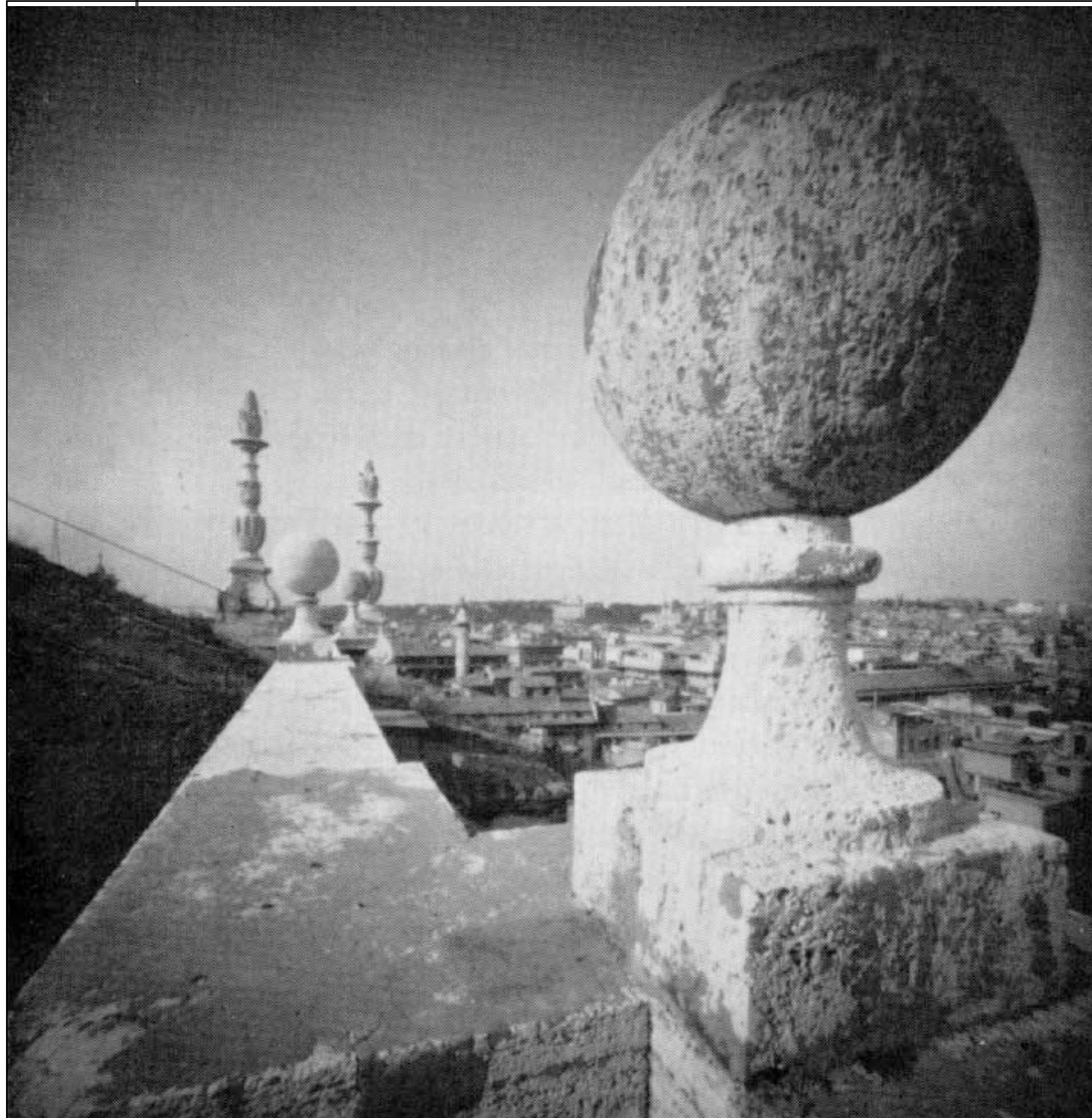
Perspective Projection



Perspective Tricks



Right Looks Wrong (Sometimes)

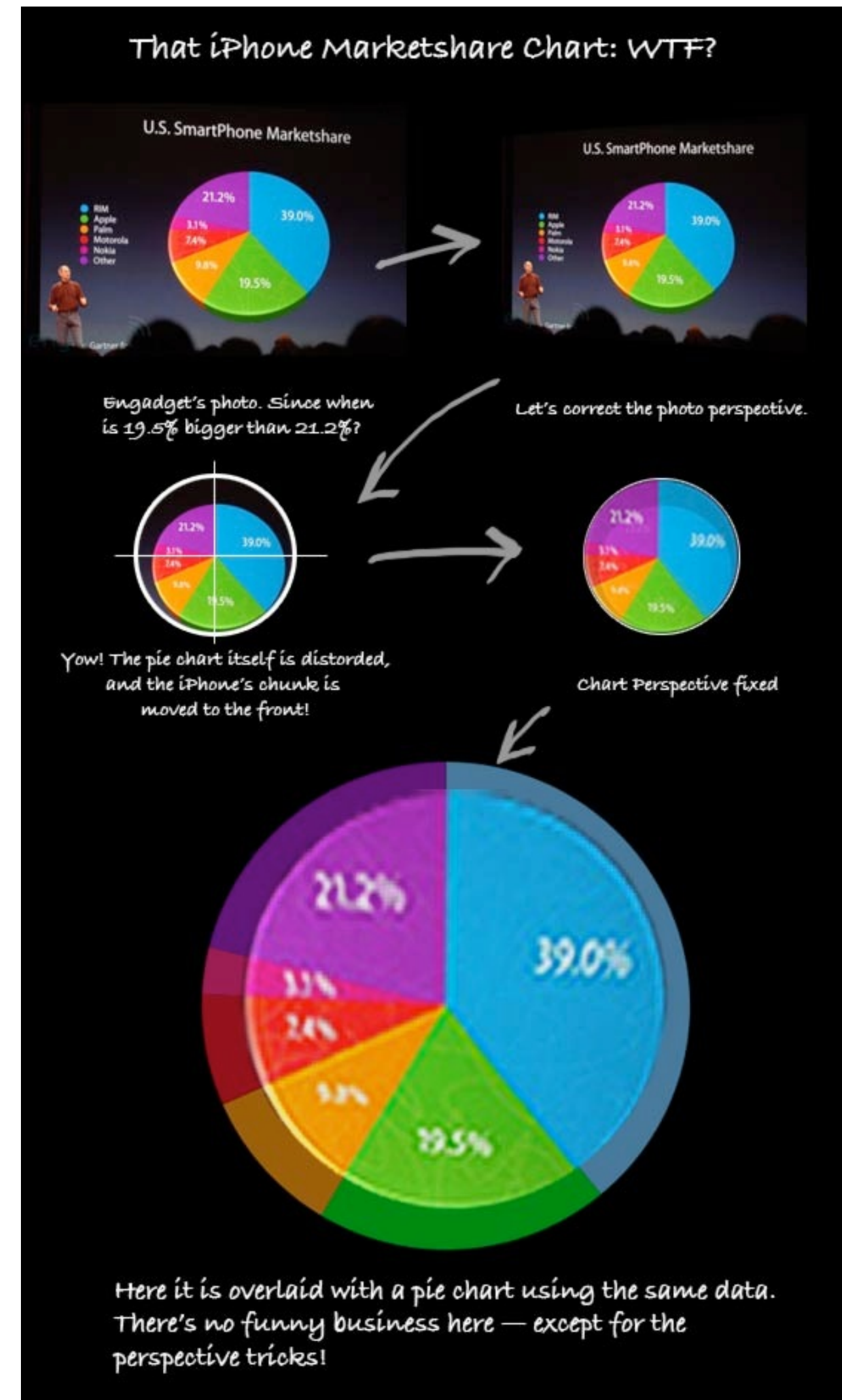
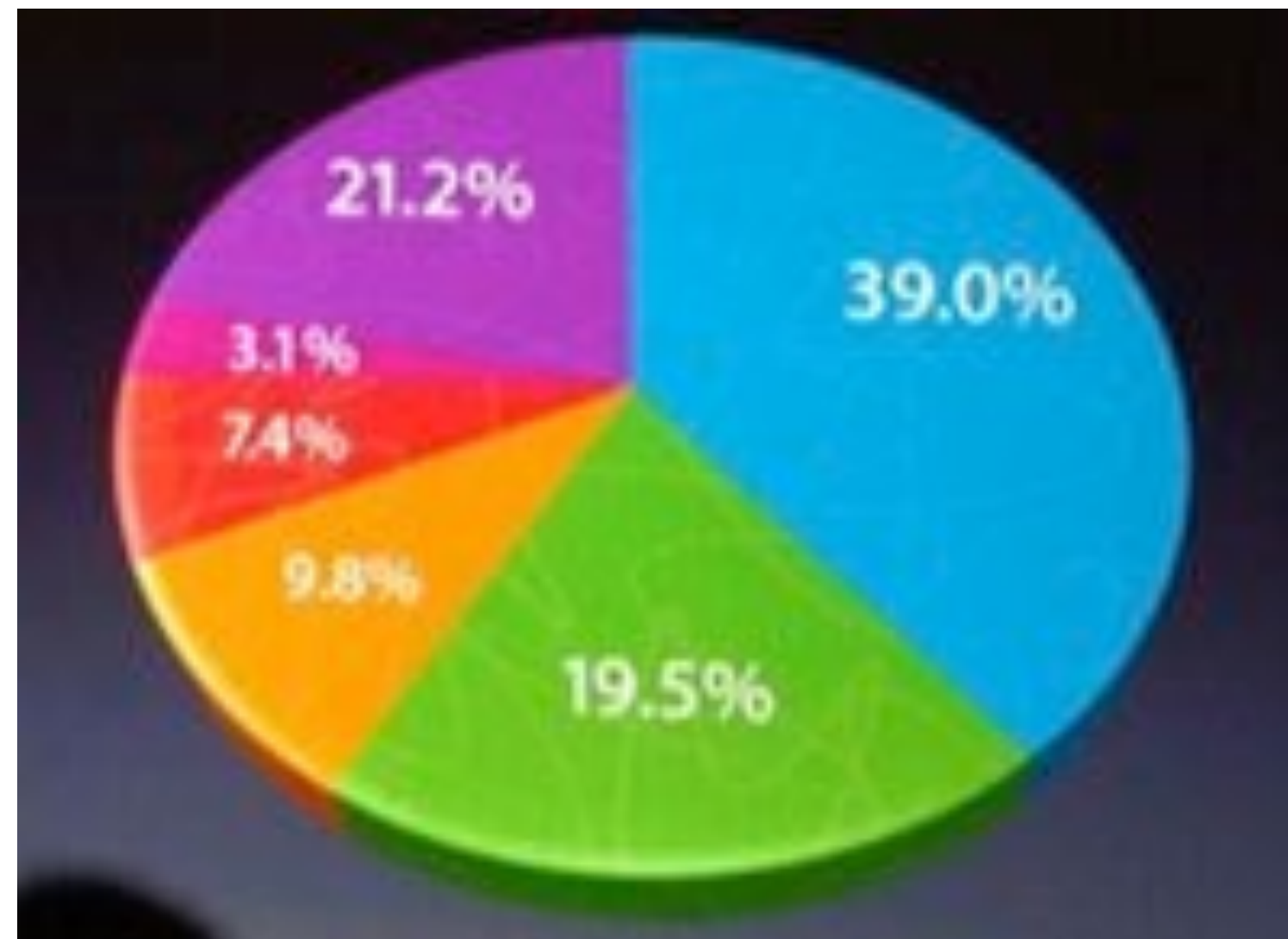


From *Correction of Geometric Perceptual Distortions in Pictures*, Zorin and Barr SIGGRAPH 1995

CS184/284A

S26 - O'Brien

Right Looks Wrong (Sometimes)

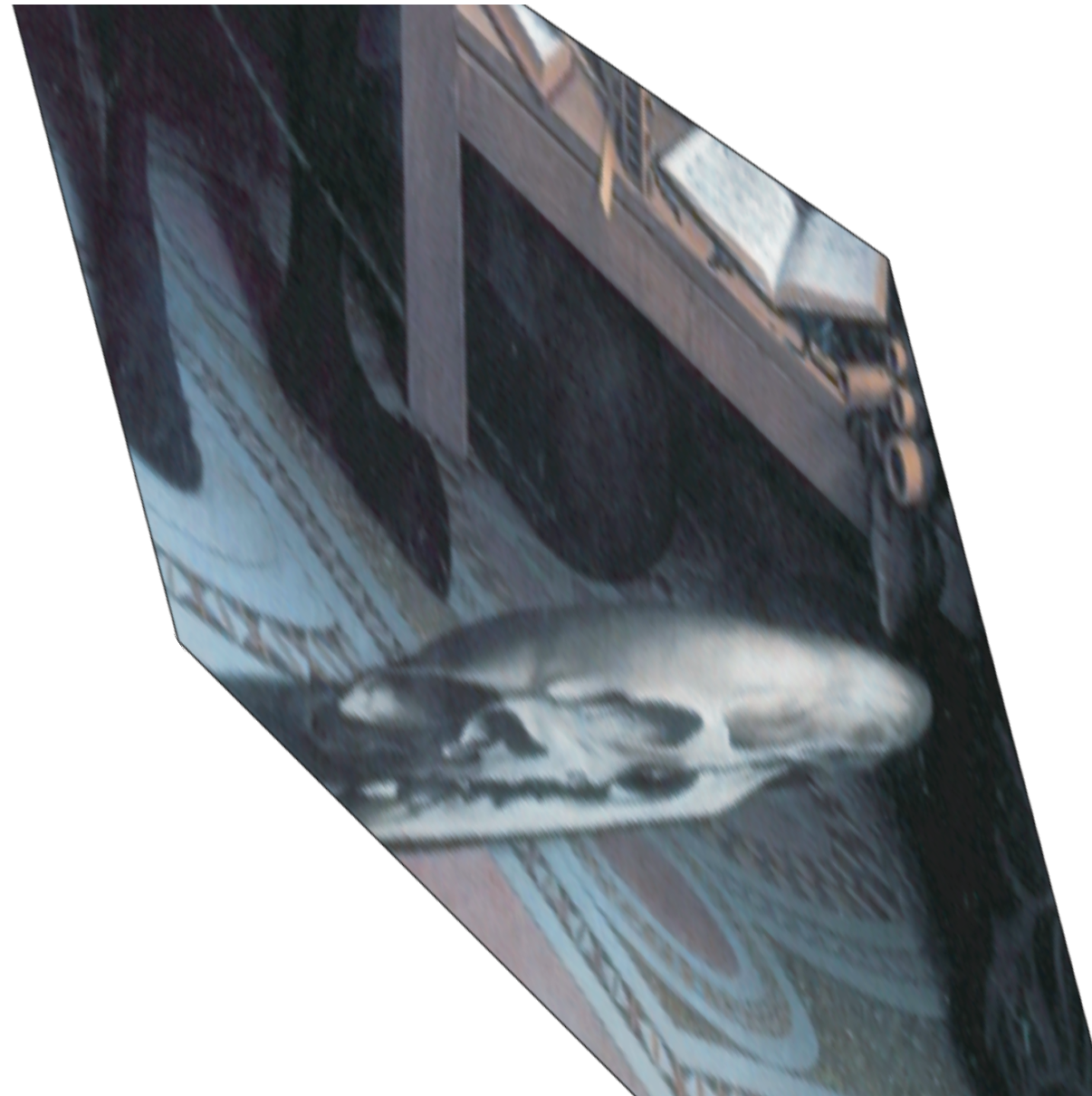


Strangeness



The Ambassadors
by Hans Holbein the Younger

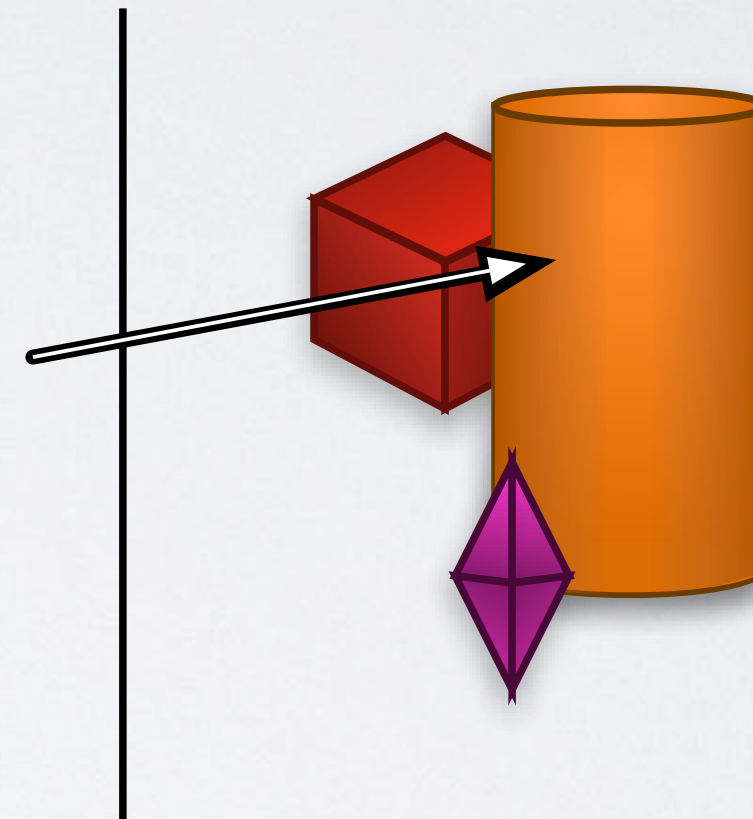
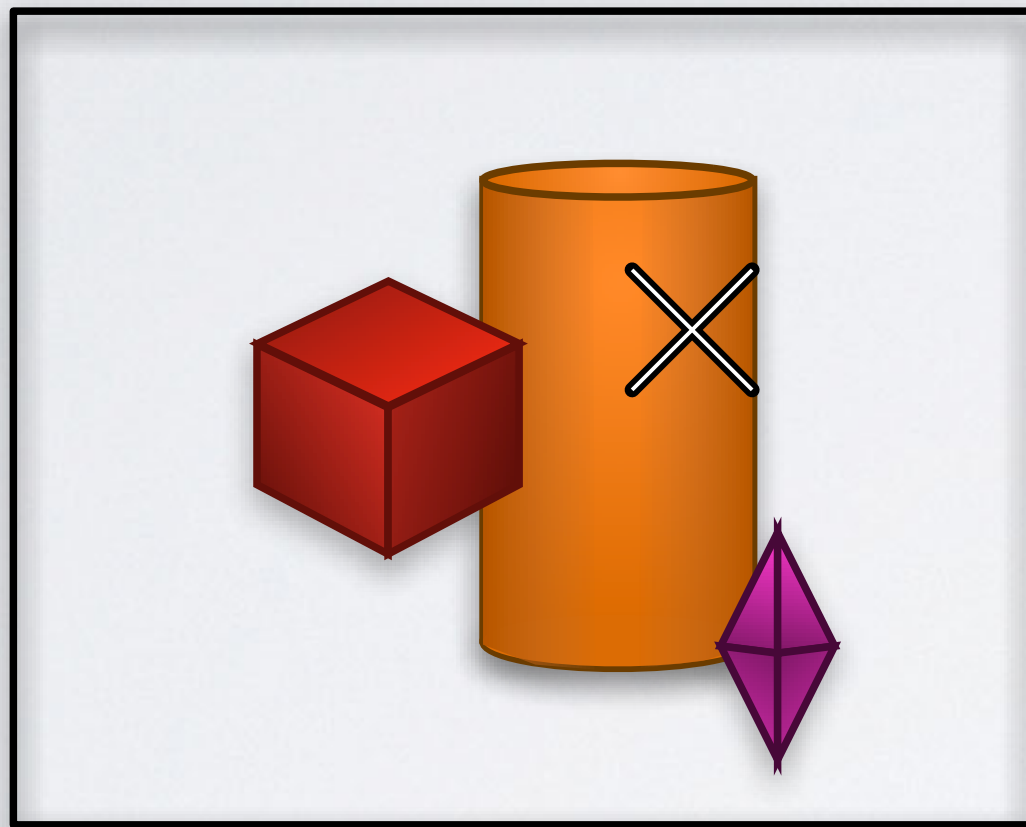
Strangeness



The Ambassadors
by Hans Holbein the Younger

Ray Picking

- Pick object by picking point on screen



- Compute ray from pixel coordinates.

Ray Picking

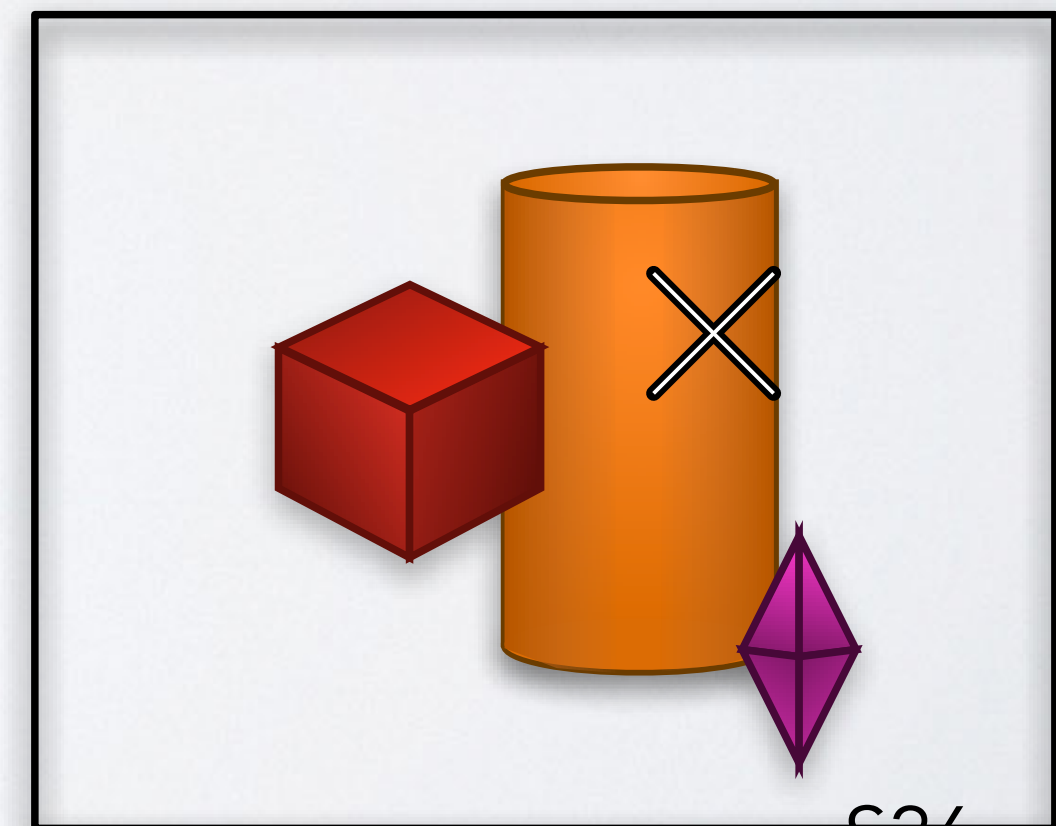
- Transform from World to Screen is:

$$\begin{bmatrix} I_x \\ I_y \\ I_z \\ I_w \end{bmatrix} = \mathbf{M} \begin{bmatrix} W_x \\ W_y \\ W_z \\ W_w \end{bmatrix}$$

- Inverse:

$$\begin{bmatrix} W_x \\ W_y \\ W_z \\ W_w \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} I_x \\ I_y \\ I_z \\ I_w \end{bmatrix}$$

- What **Z** value?



Ray Picking

- Recall that:

- Points at $z=-i$ stay at $z=-i$
- Points at $z=-f$ stay at $z=-f$

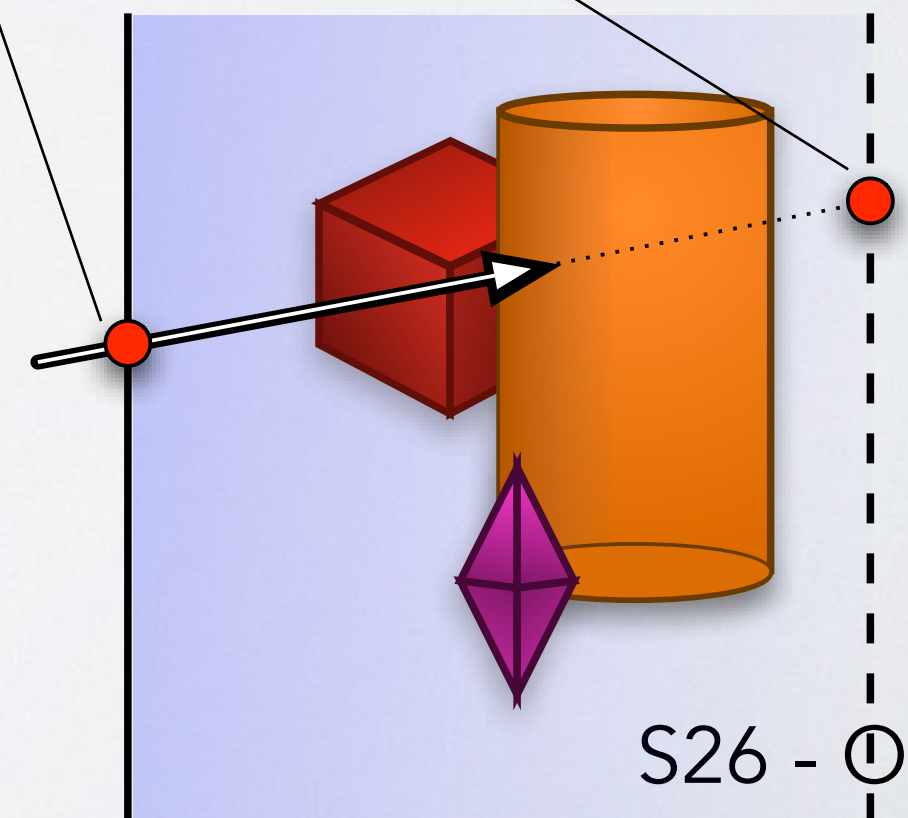
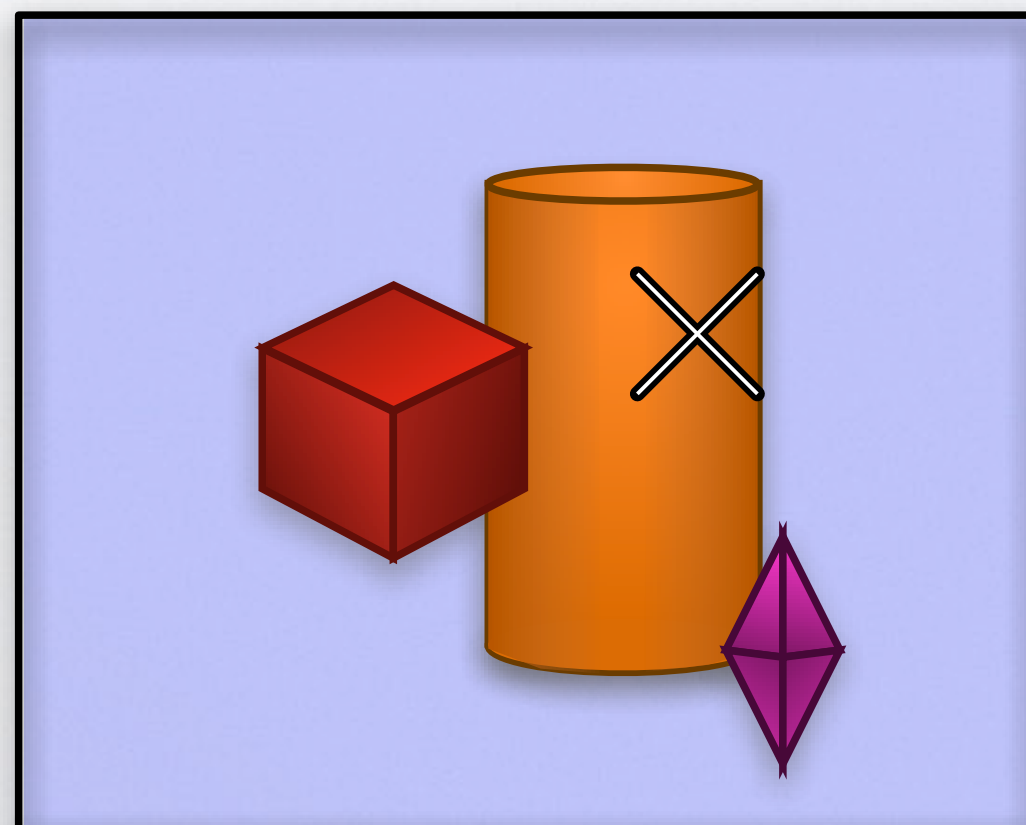
$$\mathbf{r}(t) = \mathbf{p} + t \mathbf{d}$$

$$\mathbf{r}(t) = \mathbf{a}_w + t(\mathbf{b}_w - \mathbf{a}_w)$$

Depends on screen details, YMMV
General idea should translate...

$$\mathbf{a}_s = [s_x, s_y, -i]$$

$$\mathbf{b}_s = [s_x, s_y, -f]$$



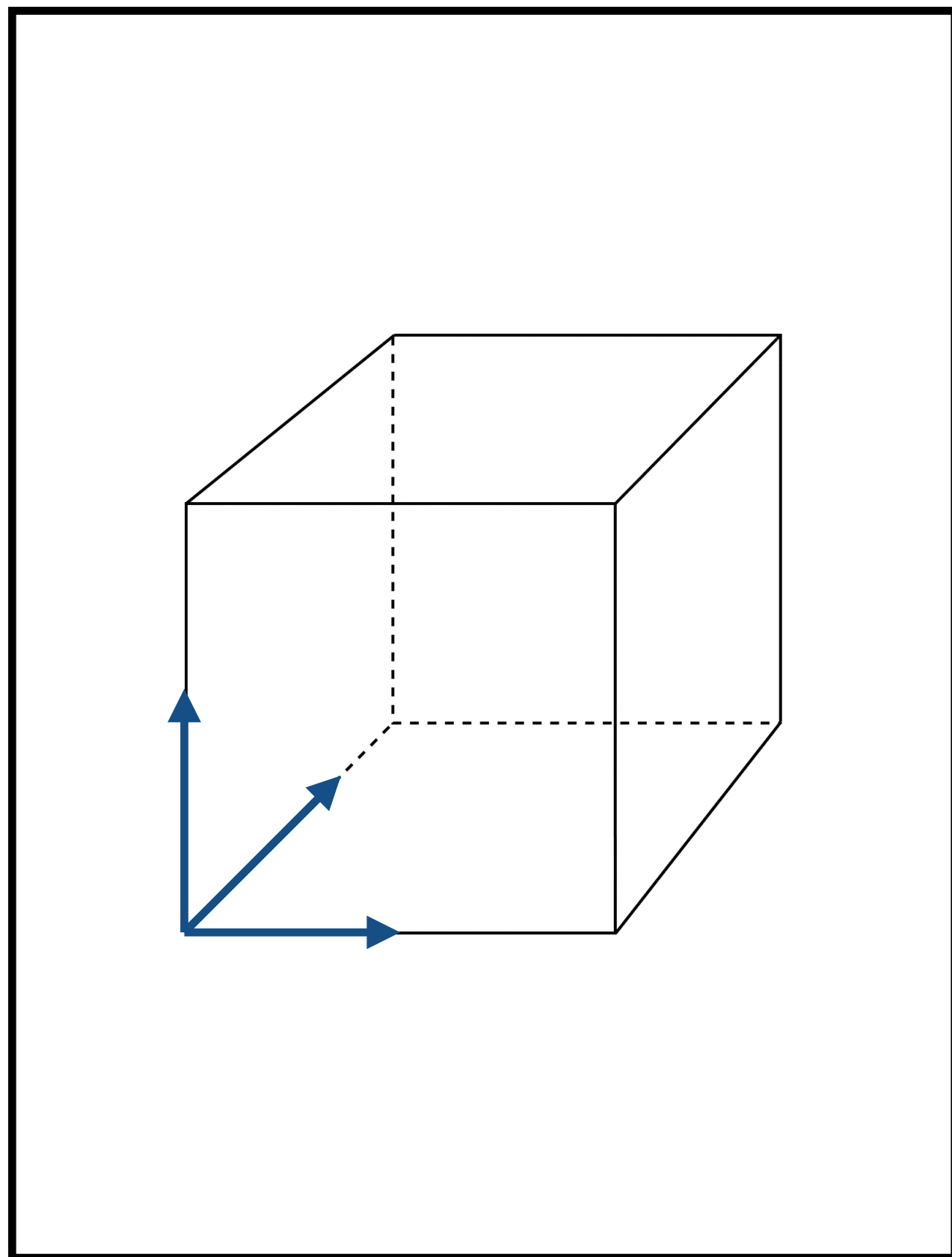
Transforms Recap

Transforms Recap

Coordinate Systems

- Object coordinates
 - Apply modeling transforms...
- World (scene) coordinates
 - Apply viewing transform...
- Camera (eye) coordinates
 - Apply perspective transform + homog. divide...
- Normalized device coordinates
 - Apply 2D screen transform...
- Screen coordinates

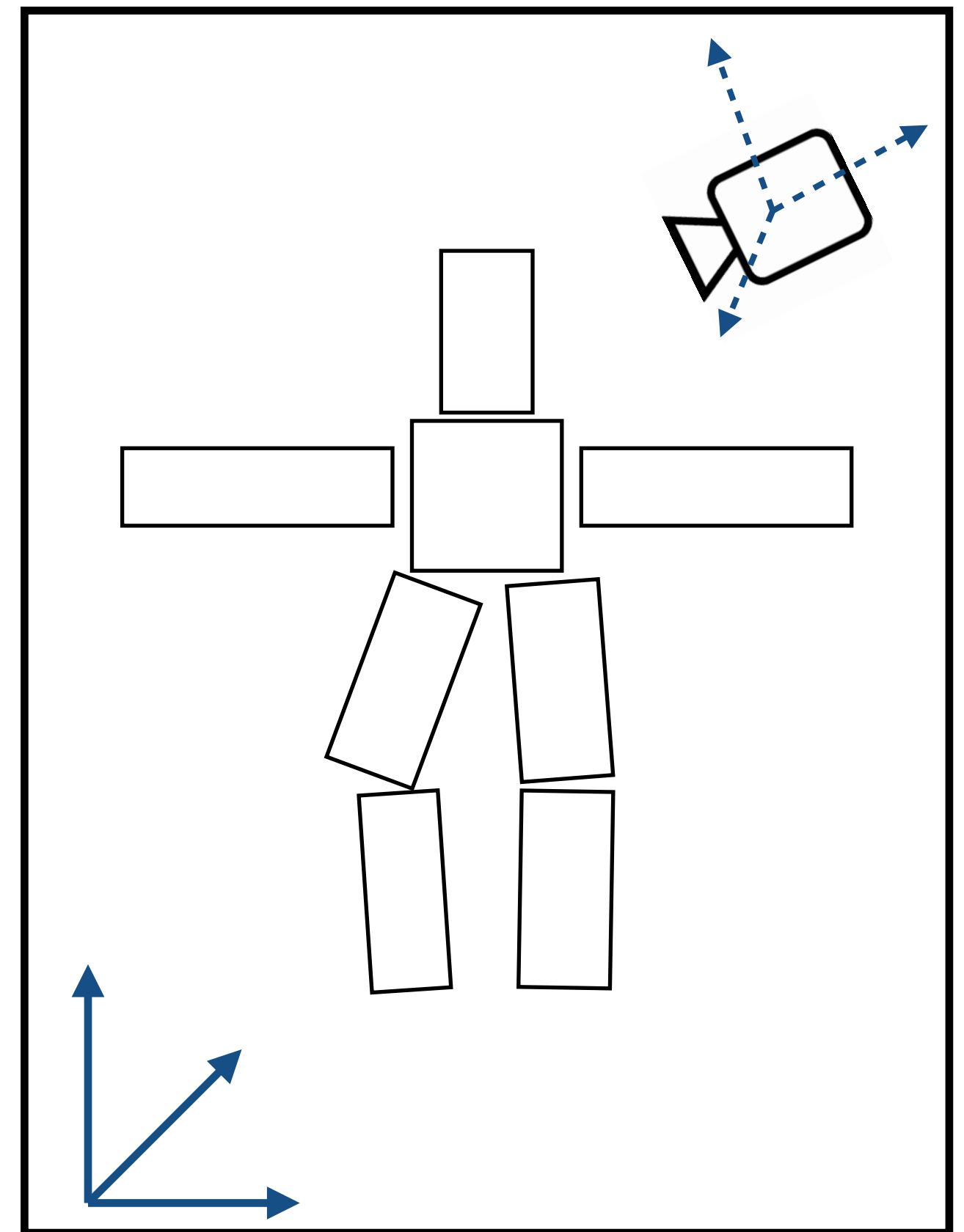
Transforms Recap



Object coords

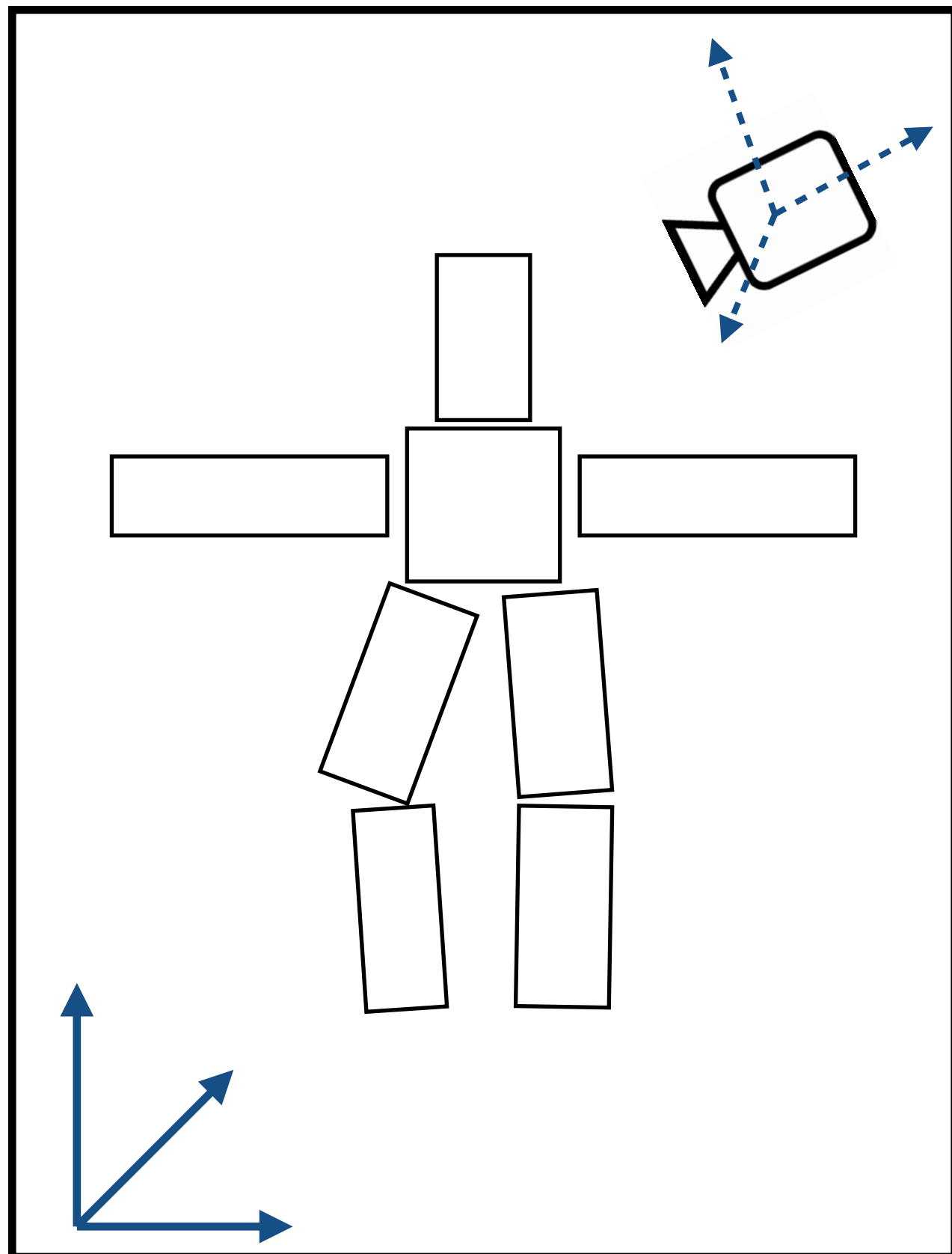


**Modeling
transforms**



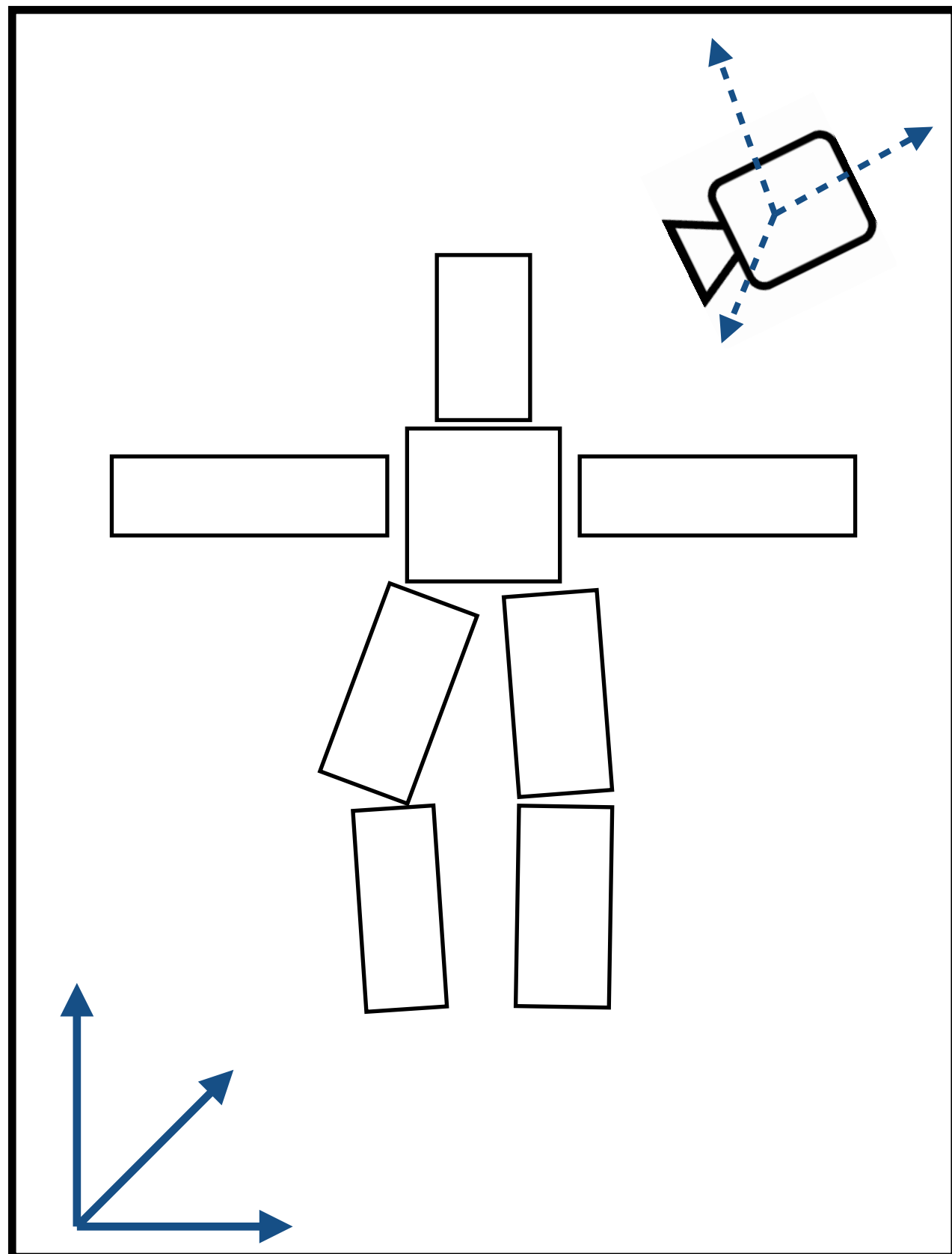
World coords

Transforms Recap



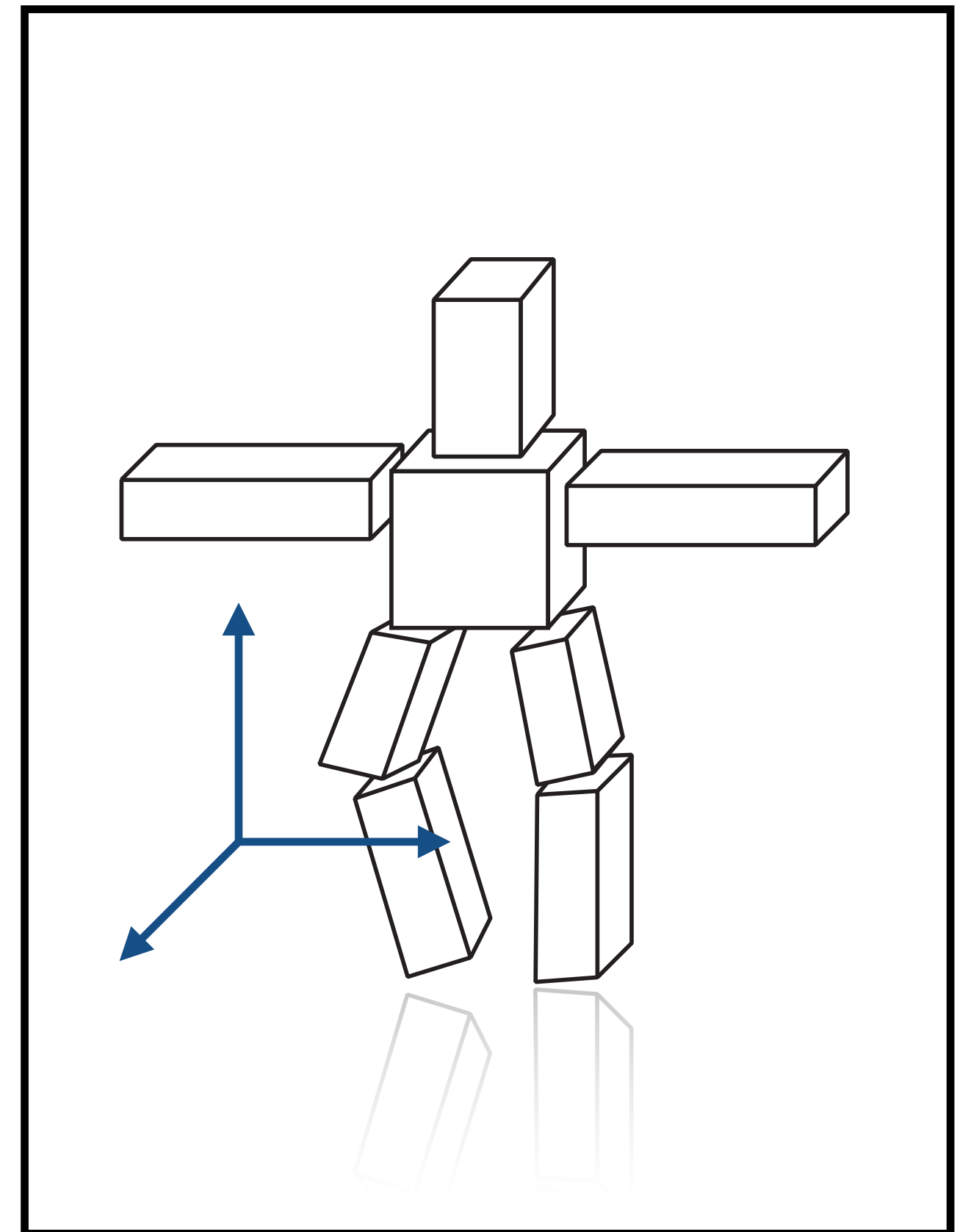
World coords

Transforms Recap



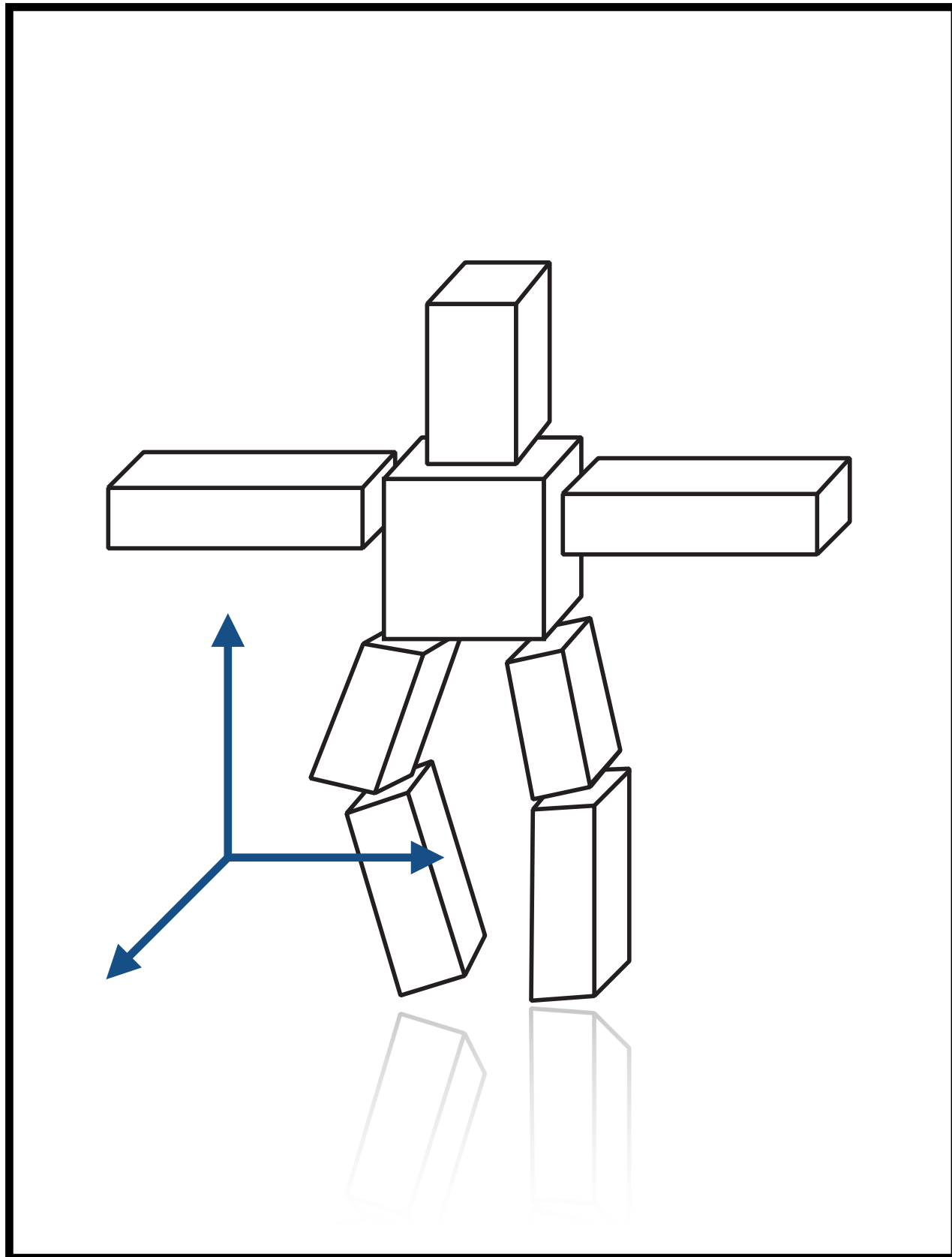
World coords

→
Viewing
transform



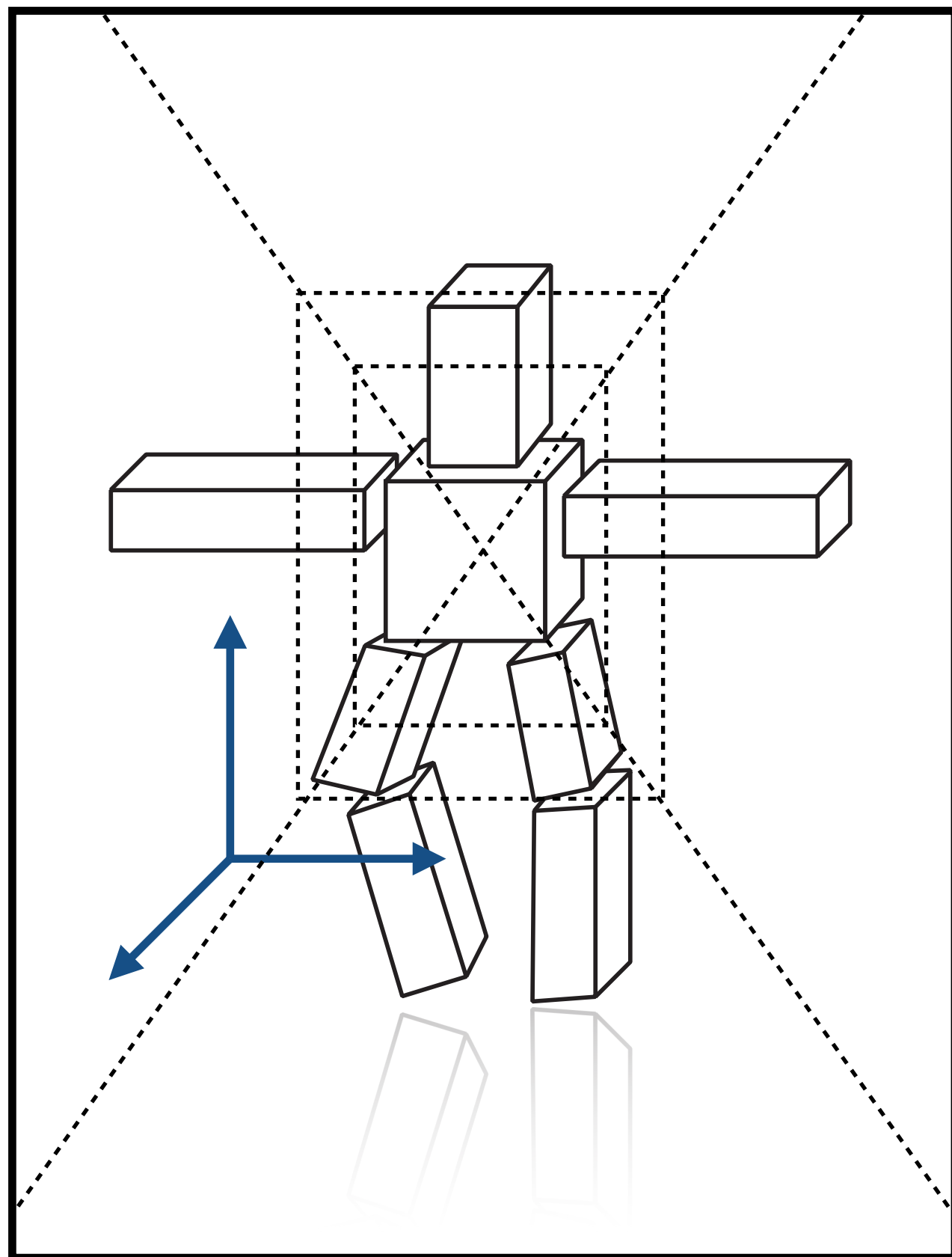
Camera coords

Transforms Recap



Camera coords

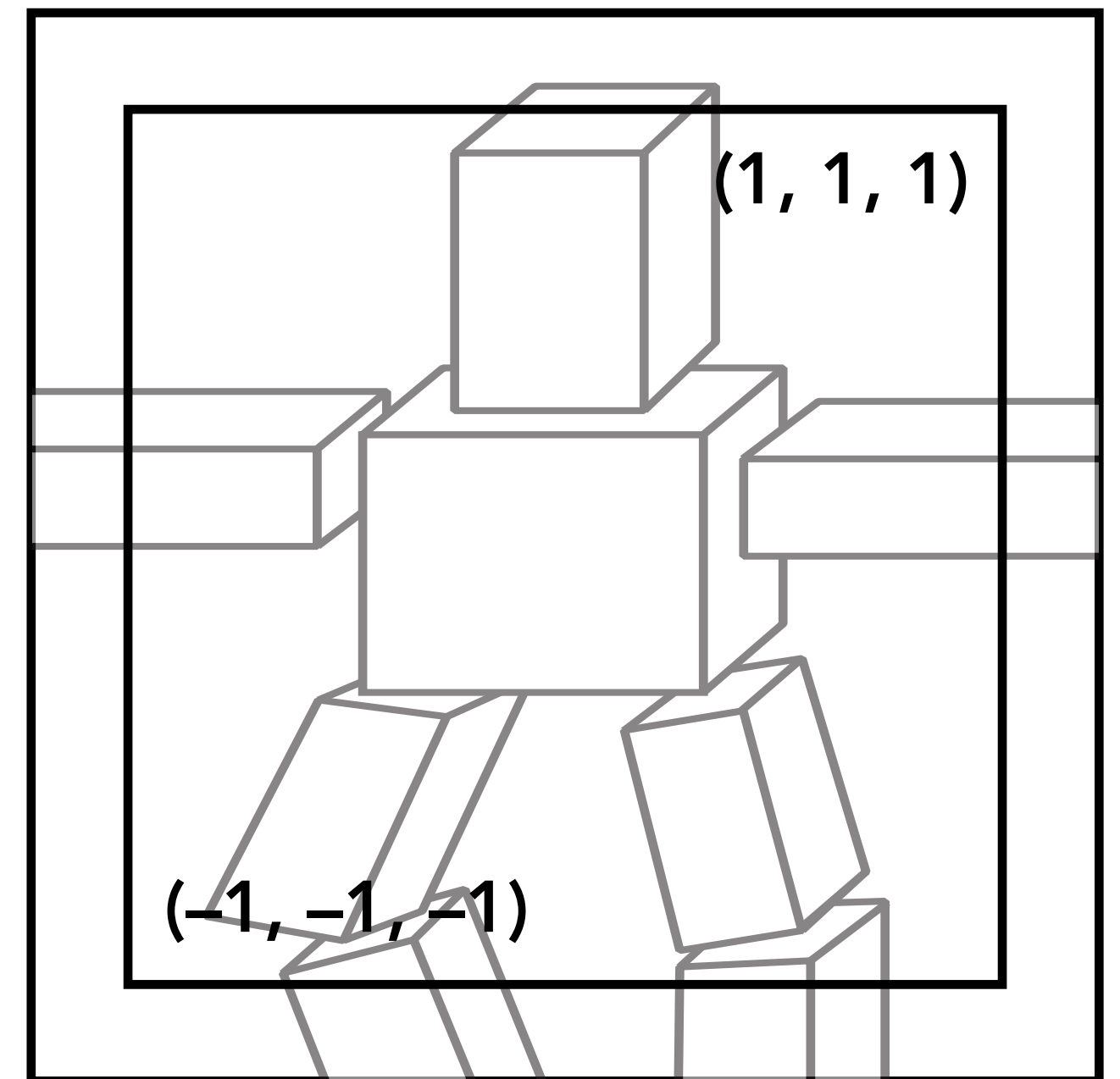
Transforms Recap



Camera coords

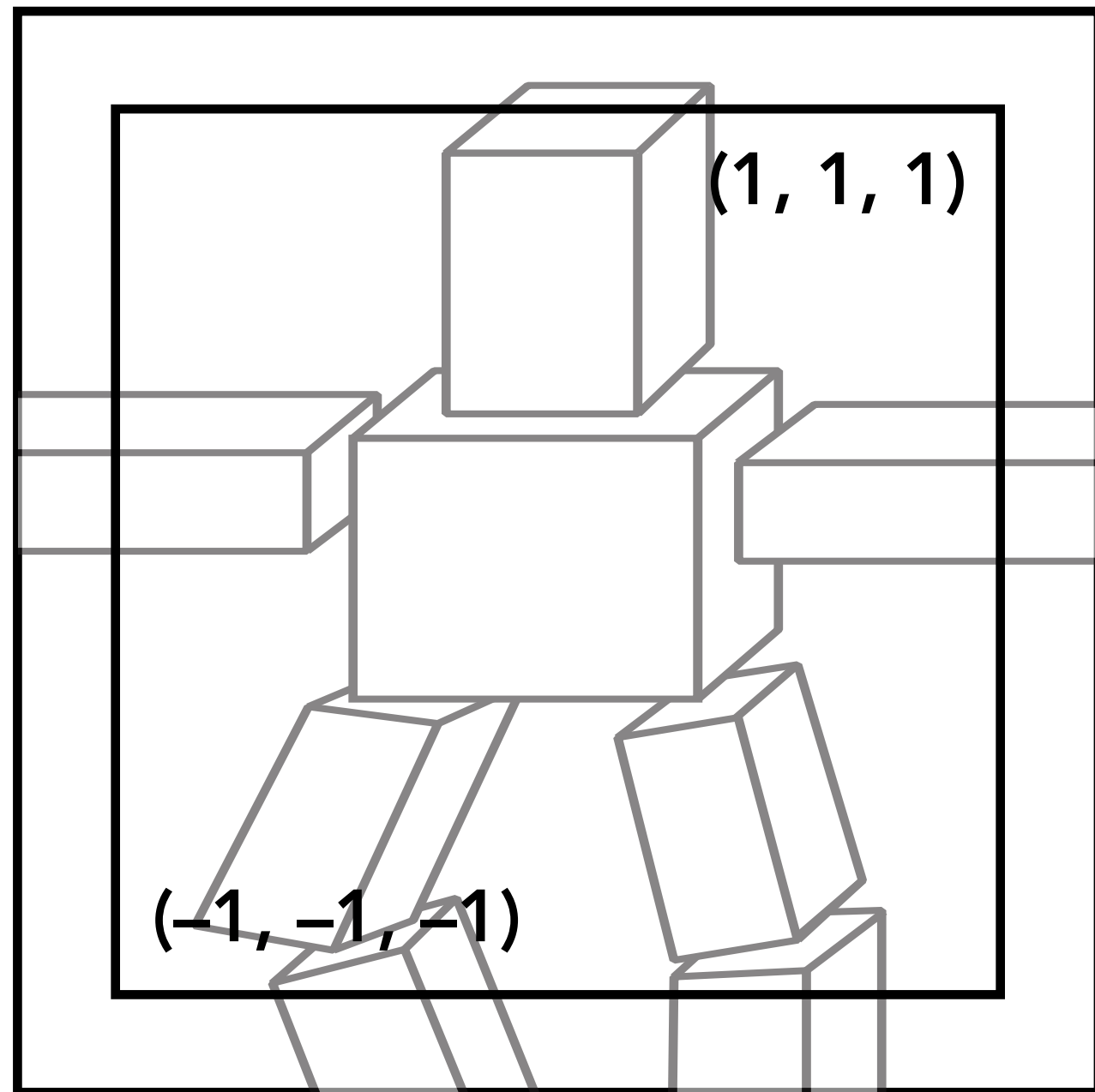


**Perspective
projection
and
homogeneous
divide**



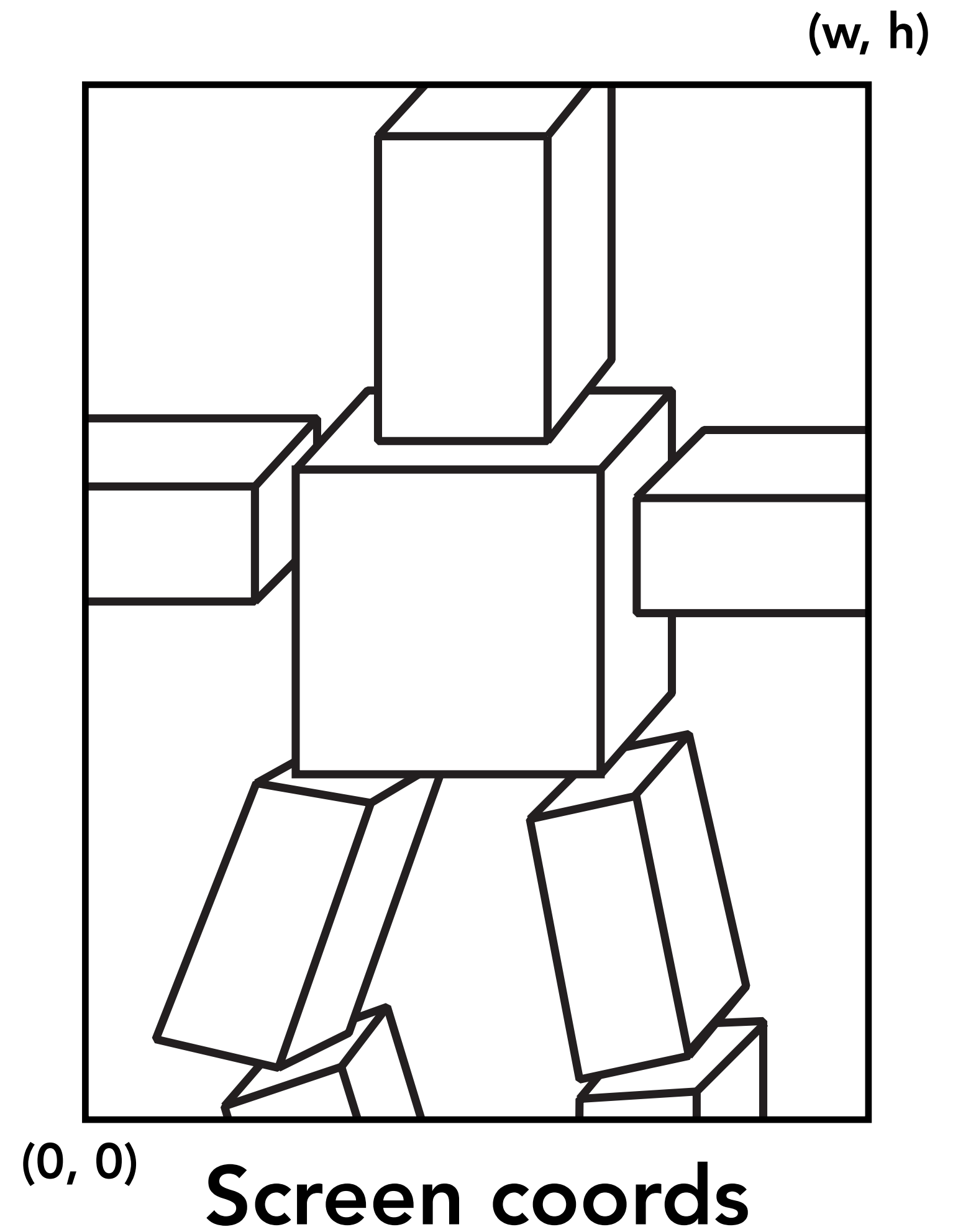
NDC

Transforms Recap



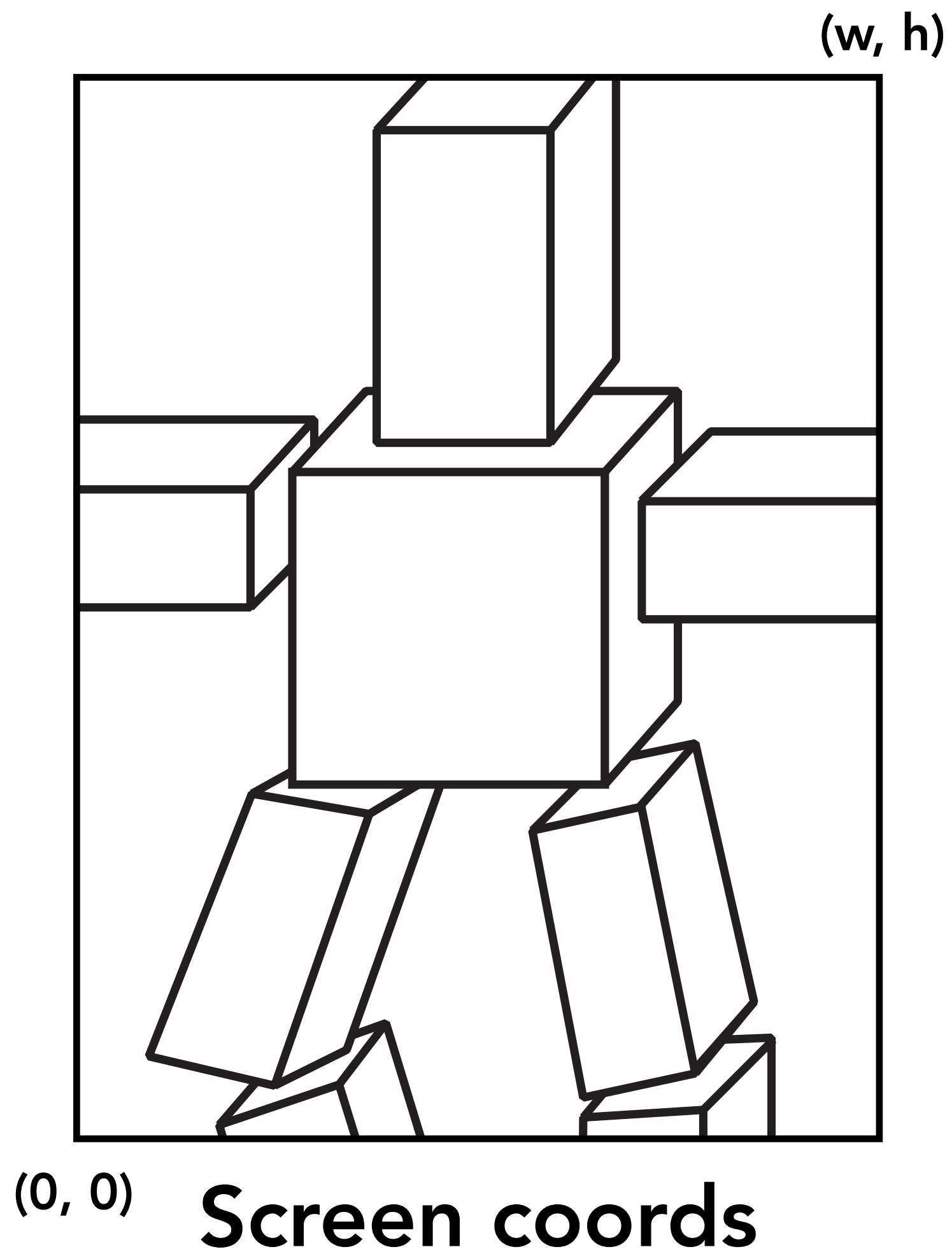
NDC

→
**Screen
transform**

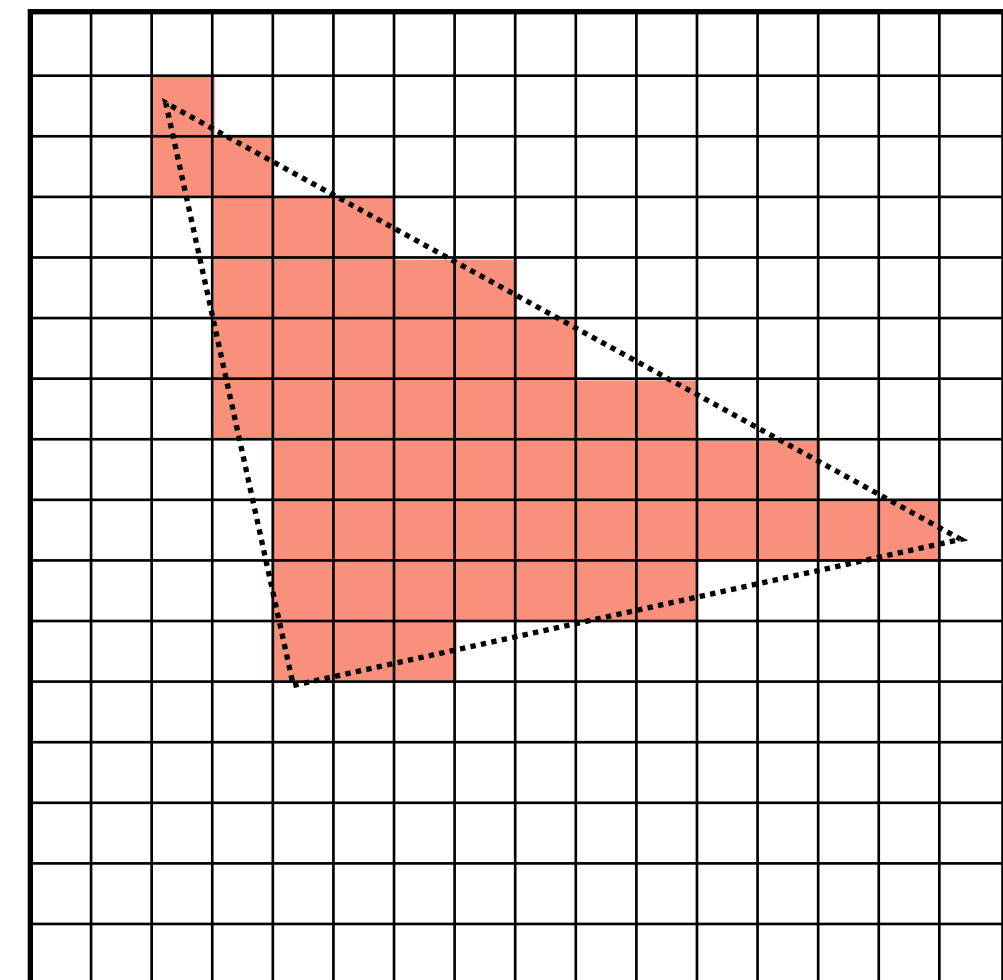


Screen coords

Transforms Recap



Rasterization



Things to Remember

Transform uses

- Basic transforms: rotate, scale, translate, ...
- Modeling, viewing, projection, perspective
- Change in coordinate system
- Hierarchical scene descriptions by push/pop

Implementing transforms

- Linear transforms = matrices
- Transform composition = matrix multiplication
- Homogeneous coordinates for translation, projection

Acknowledgments

This slide set contain contributions from:

- **Kayvon Fatahalian**
- **David Forsyth**
- **Pat Hanrahan**
- **Angjoo Kanazawa**
- **Ren Ng**
- **James O'Brien**
- **Mark Pauly**