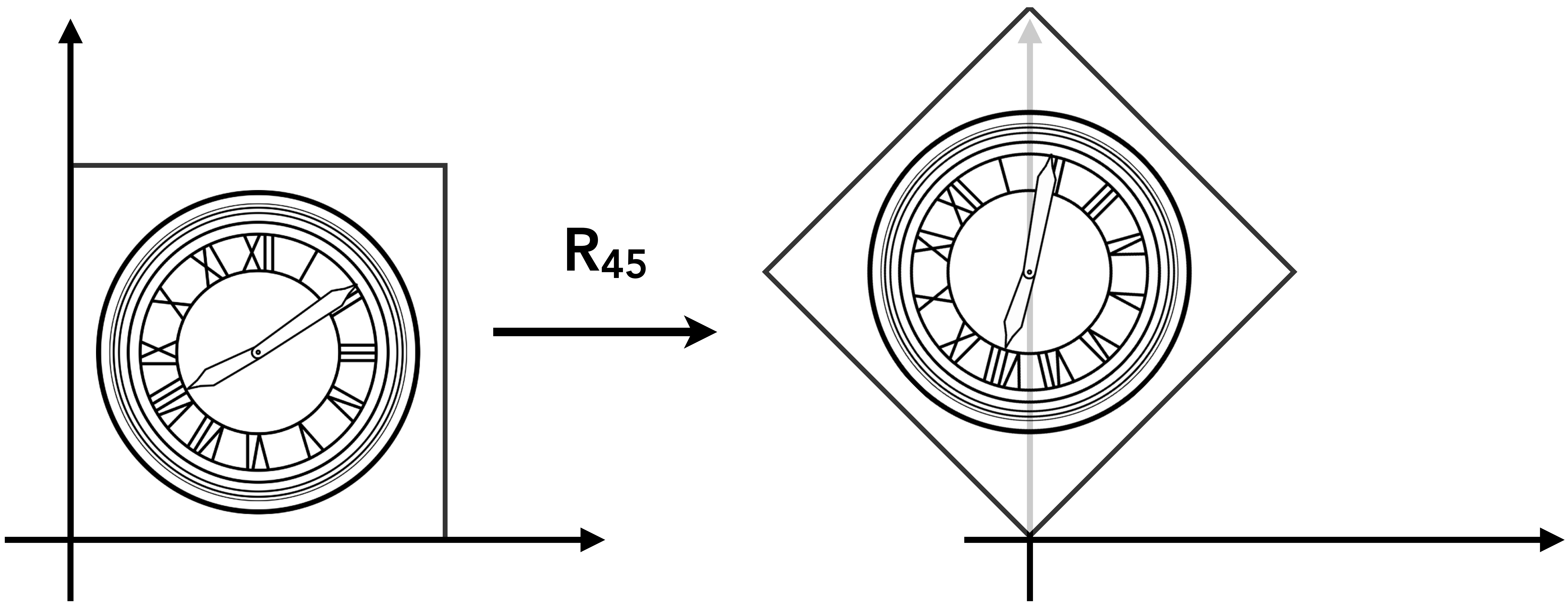
Lecture 4:

Transforms

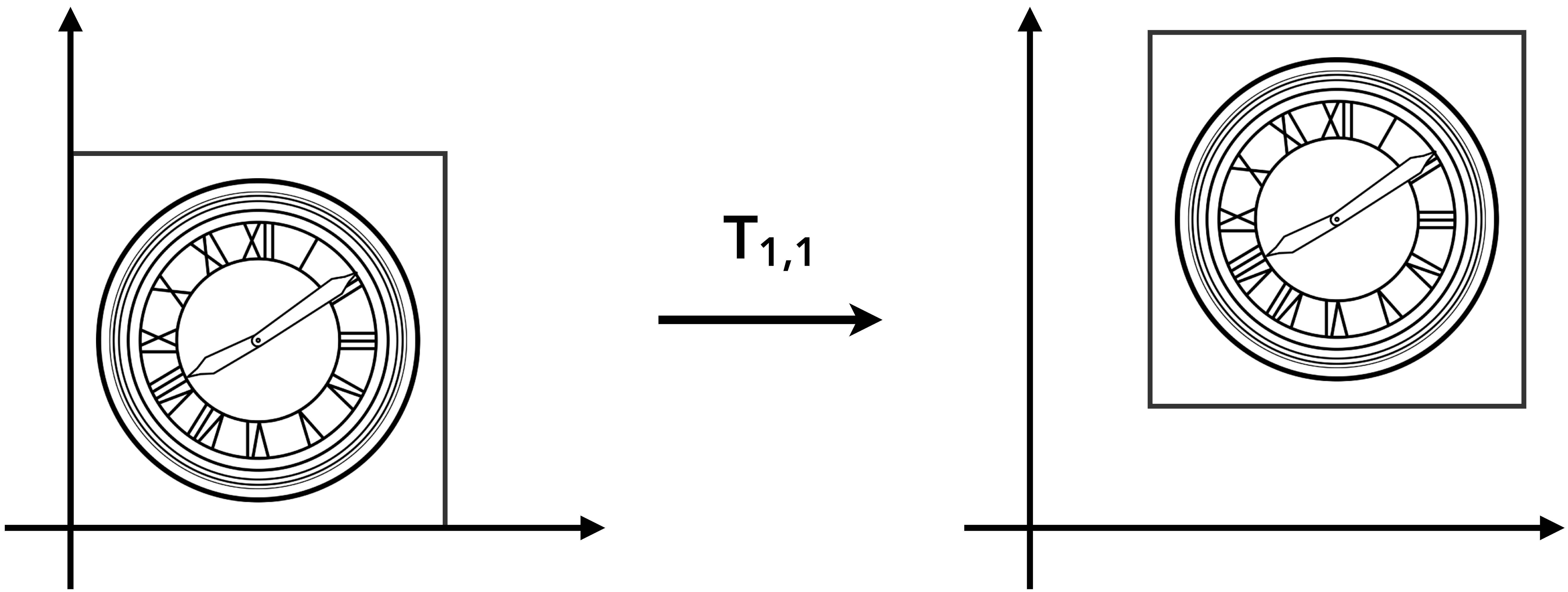
Computer Graphics and Imaging
UC Berkeley CS184/284A

Basic Transforms

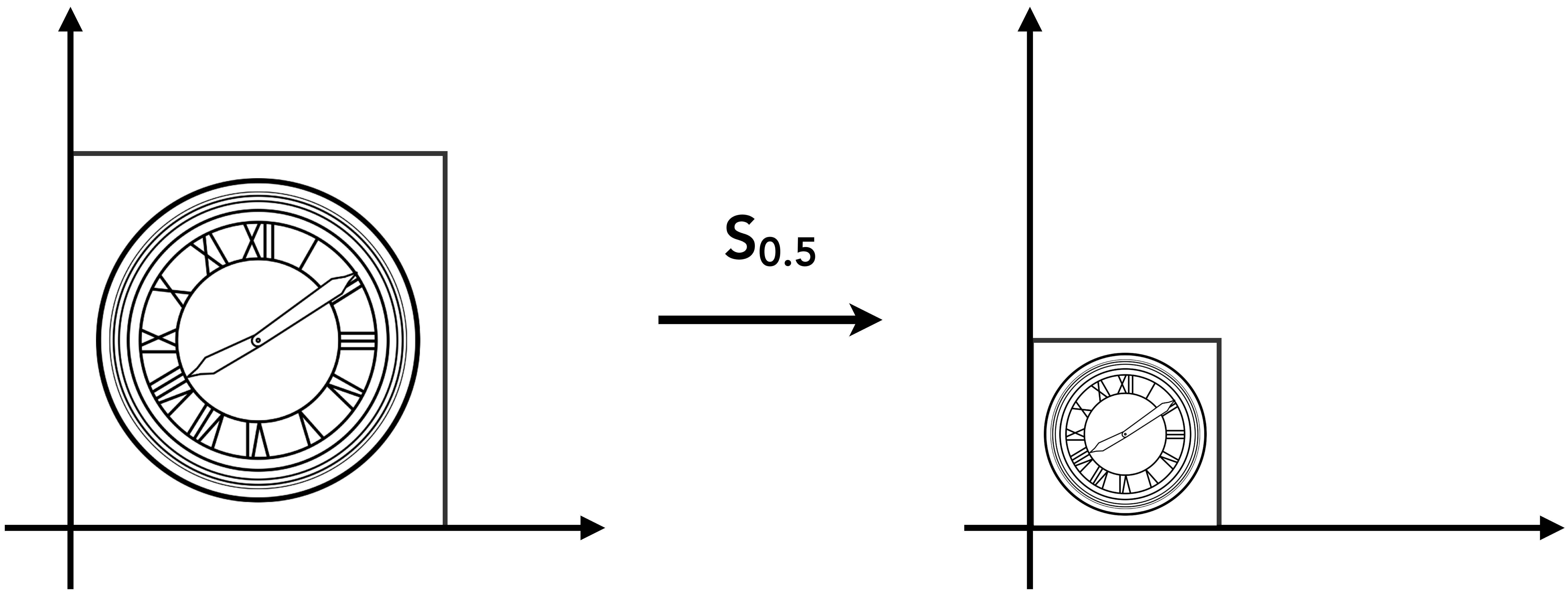
Rotate



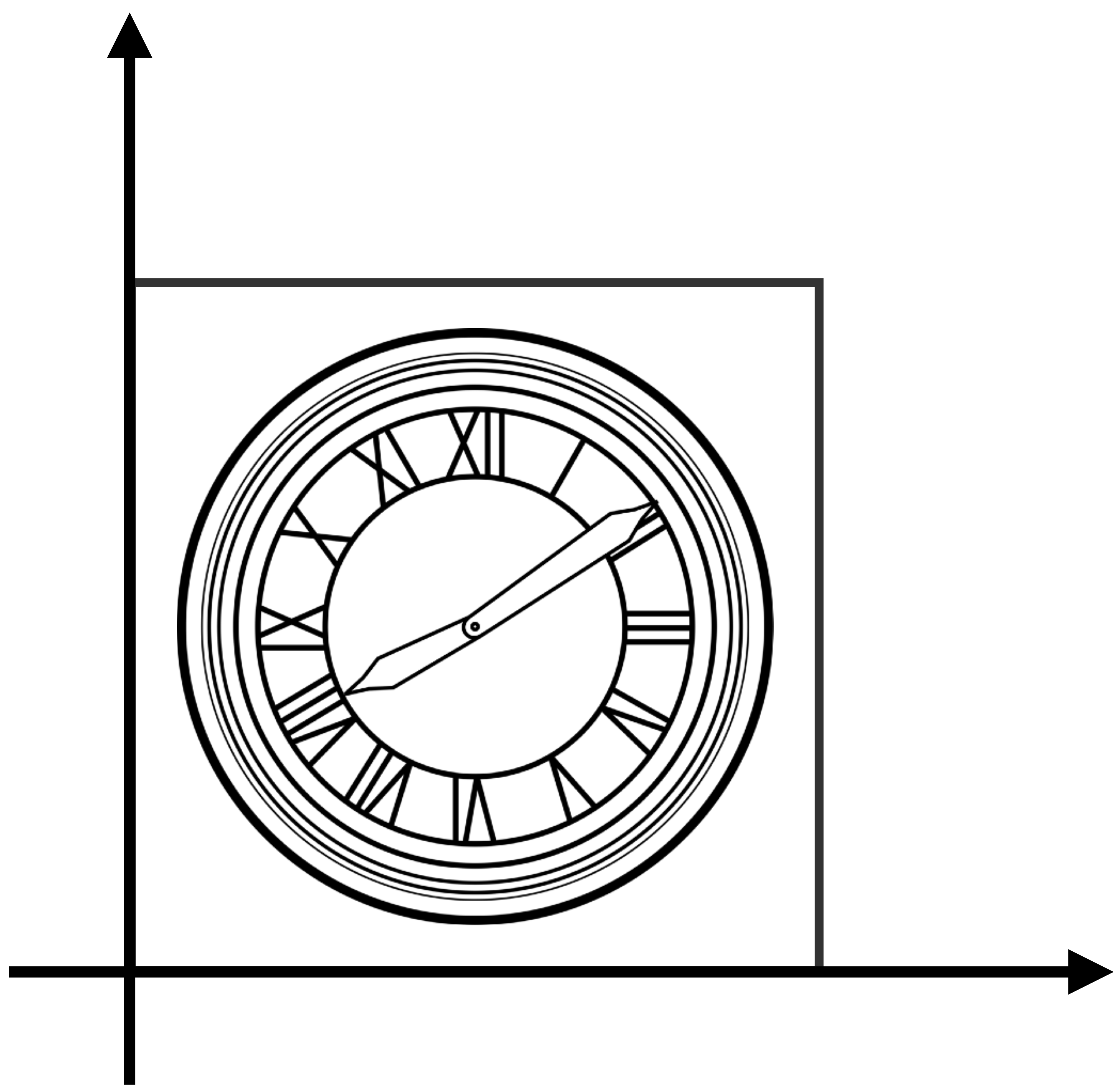
Translate



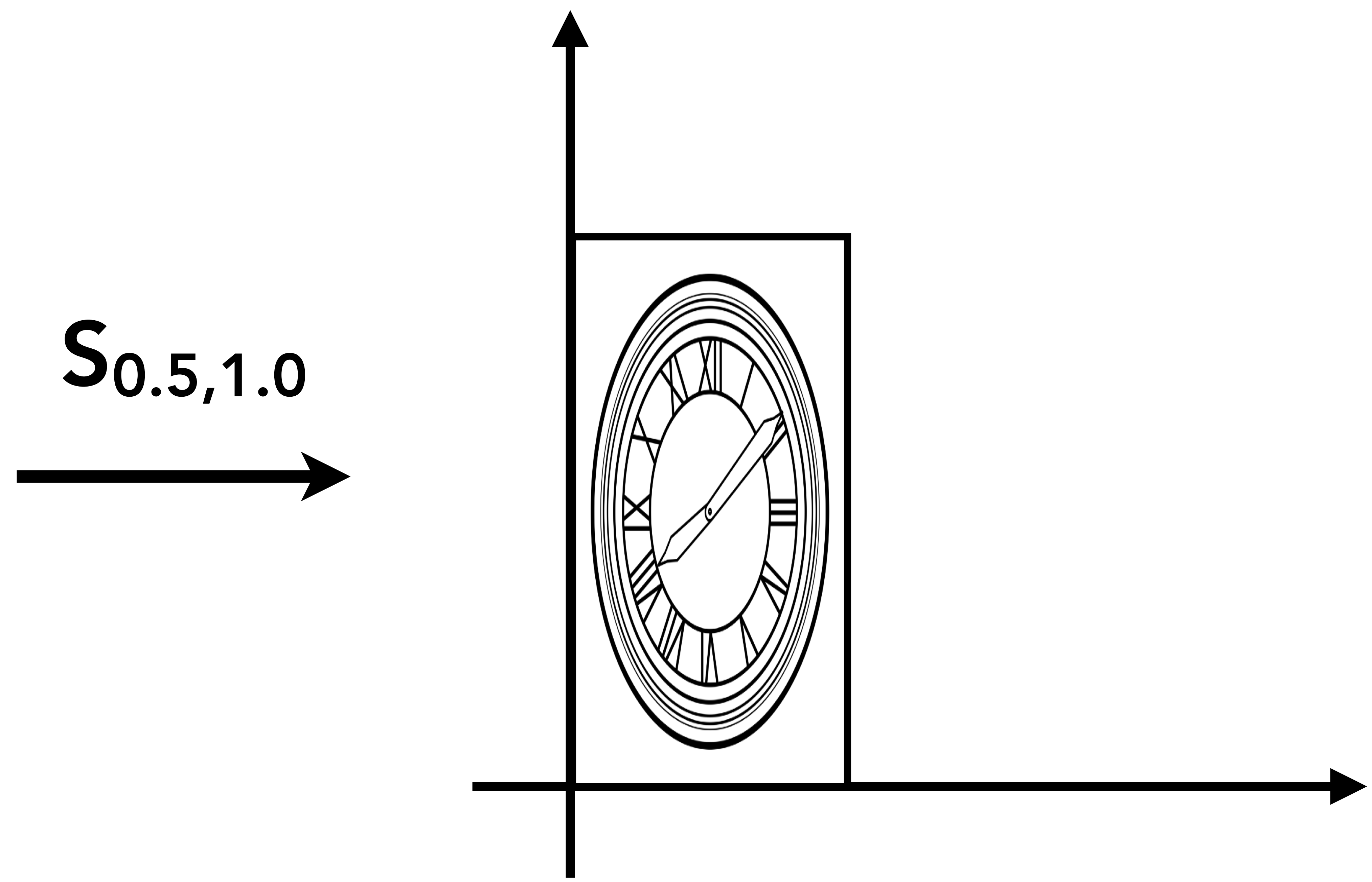
Scale



Scale (Non-Uniform)



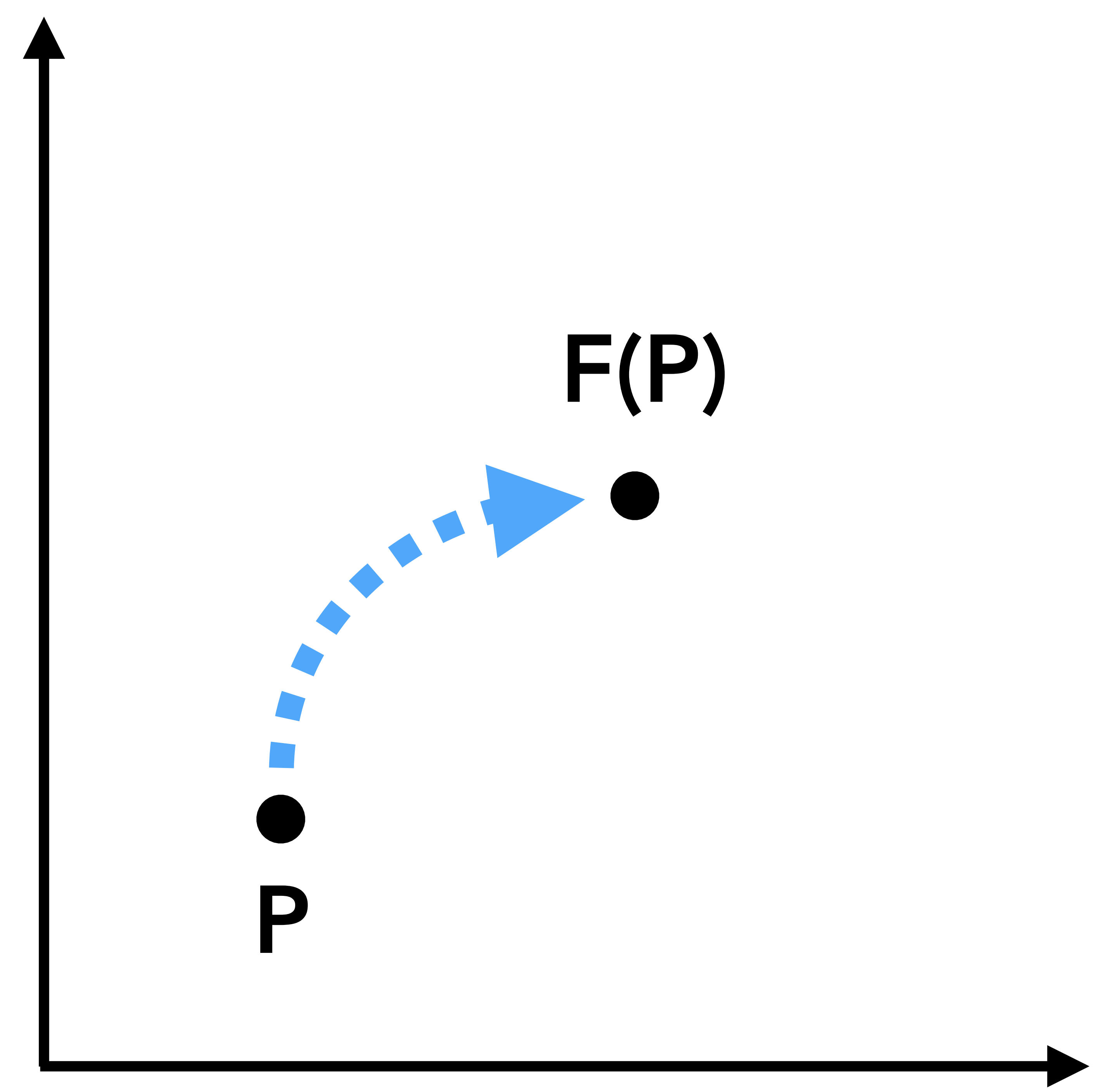
$S_{0.5,1.0}$



What Are Transforms?

Just functions acting on points

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



Why Study Transforms?

Modeling A Crowd

Minions

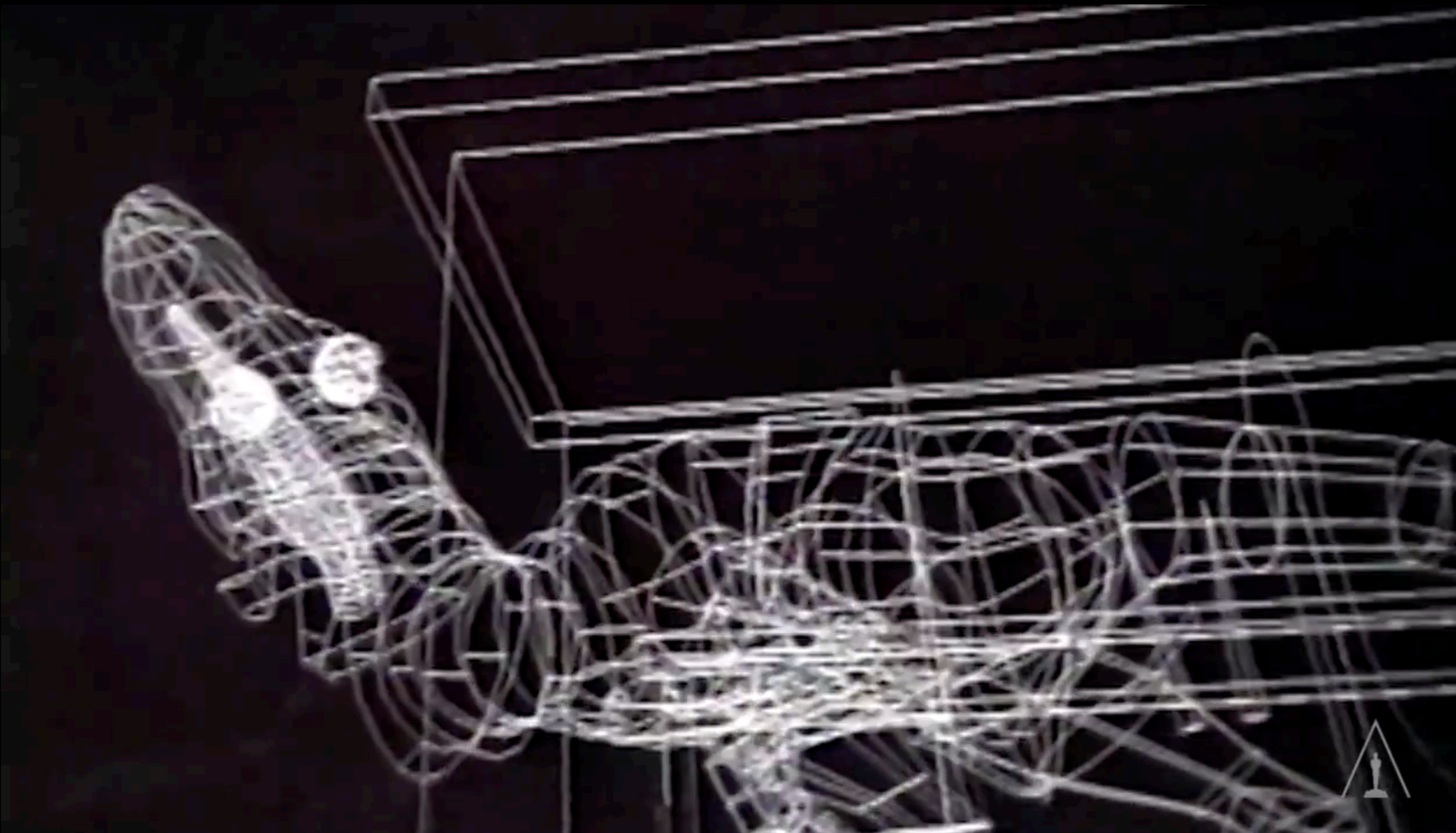
Transforms can describe position of object instances

Posing a Character's Skeleton

Minions

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

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

Lecture Outline

How to think about and use transformations

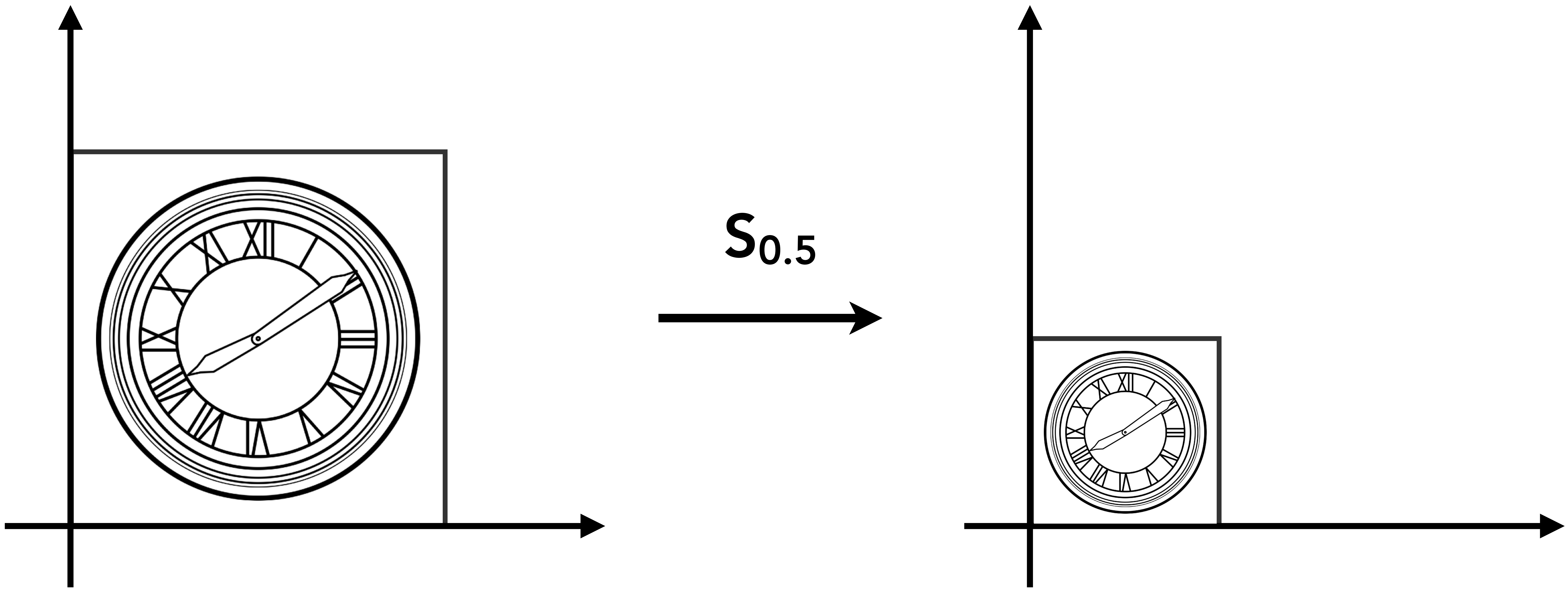
- Types: rotate, translate, scale, …
- Coordinate frames
- Composing multiple transformations
- Hierarchical transforms
- Perspective projection

How to implement?

- Represent transforms as matrices
- Homogeneous coordinates

Linear Transforms = Matrices

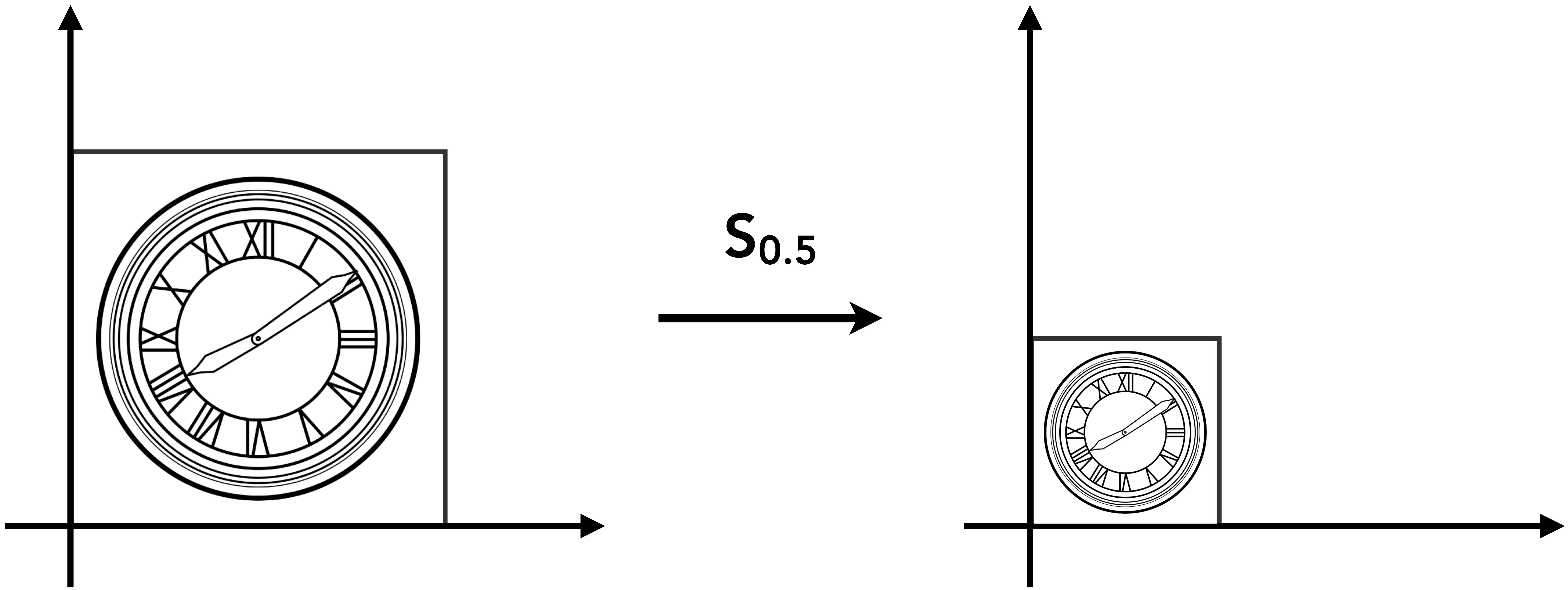
Scale Transform



$$x' = sx$$

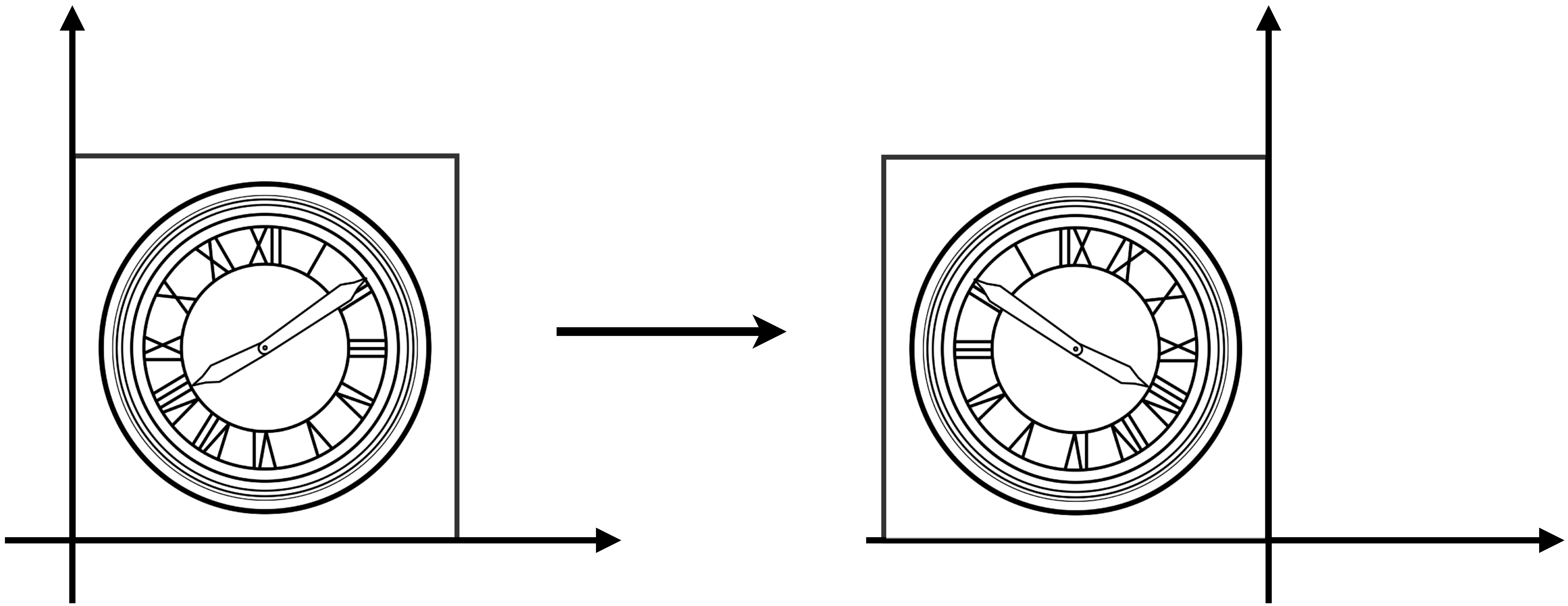
$$y' = sy$$

Scale Matrix



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

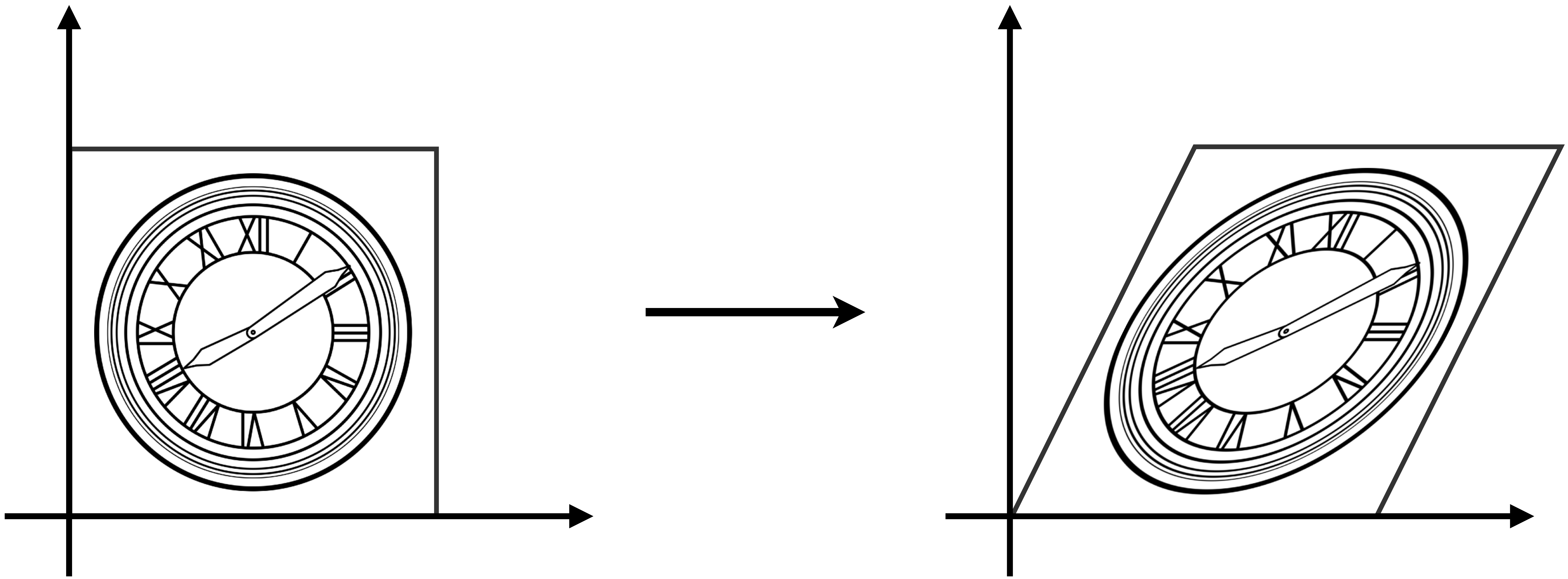
Reflection Matrix



$$x' = ??$$

$$y' = ??$$

Shear Matrix



$$x' = ??$$

$$y' = ??$$

Linear Transforms = Matrices

$$x' = a\,x + b\,y$$
$$y' = c\,x + d\,y$$

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

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

2D Coordinate Systems

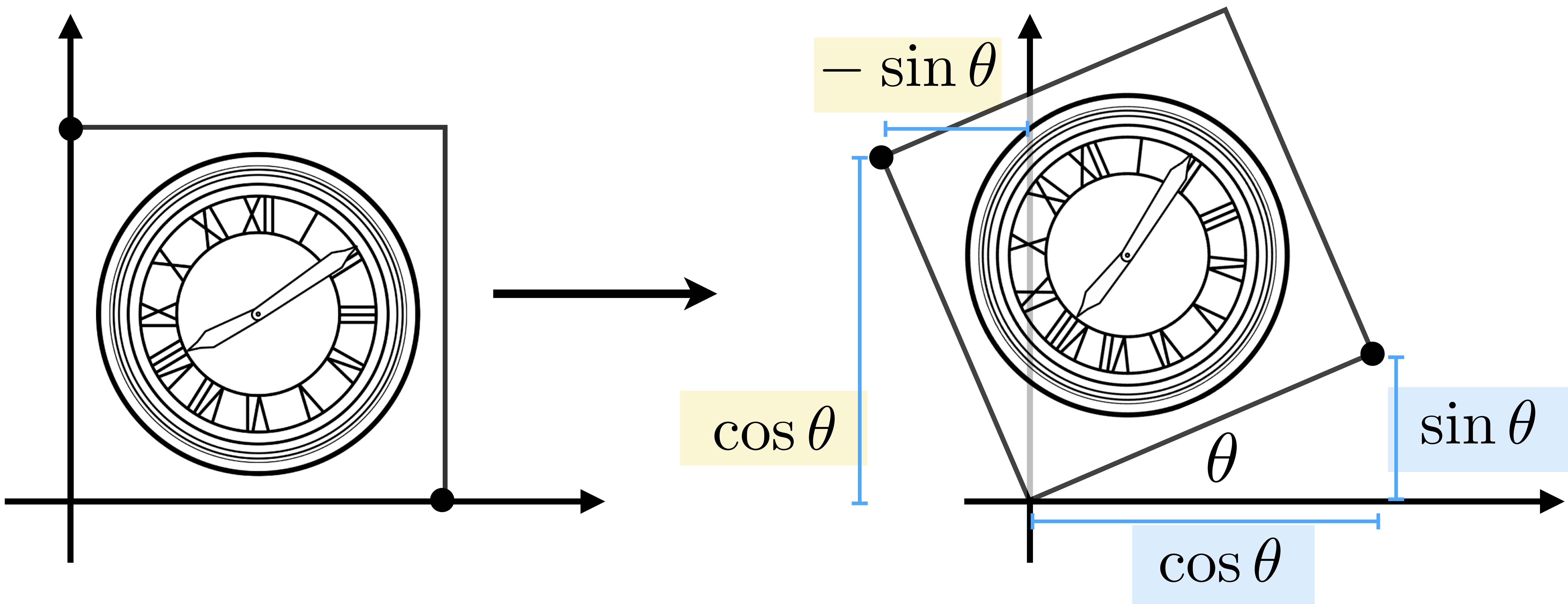
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

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

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

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

Rotation Matrix

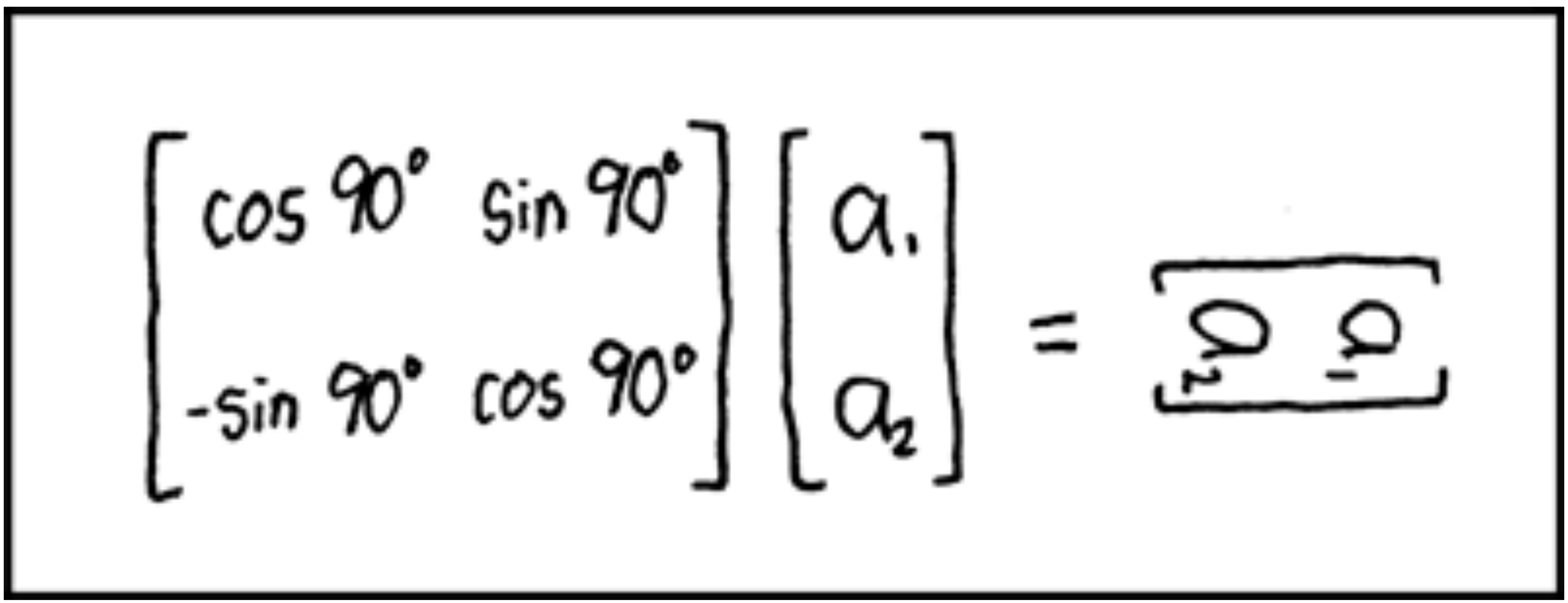


$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

2D Rotation Matrix: Another Way

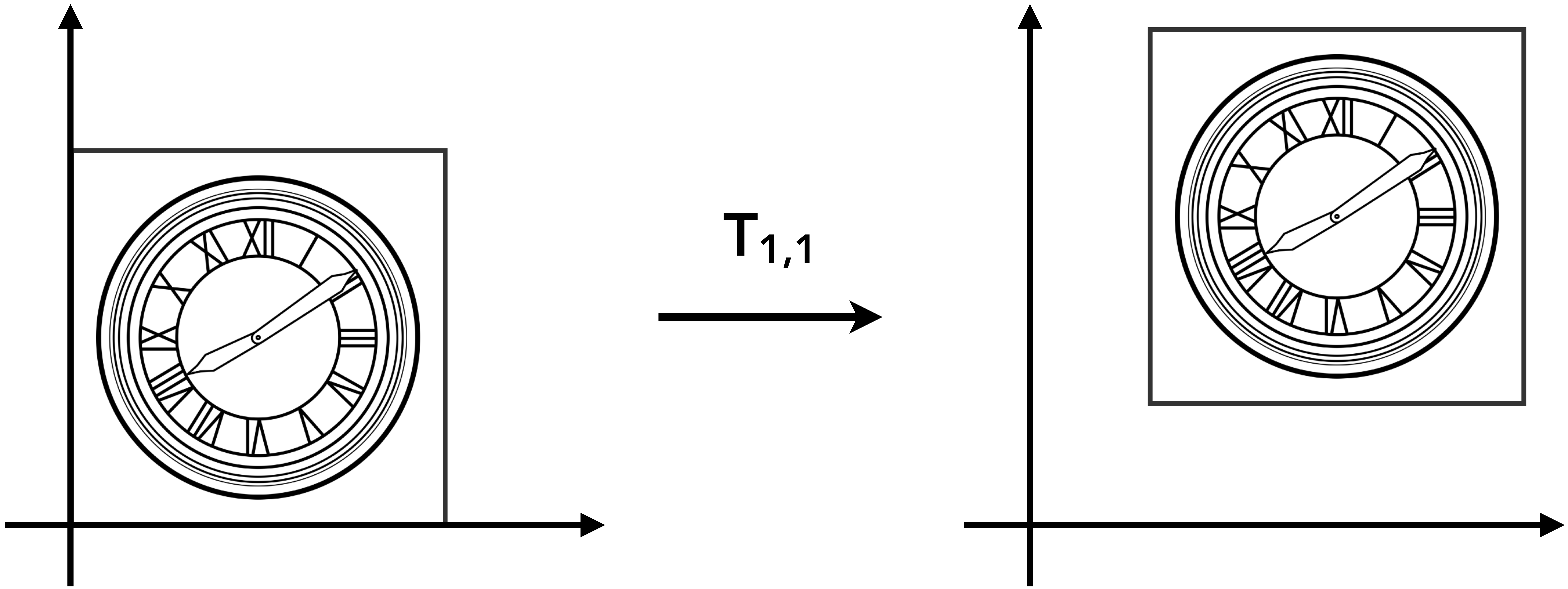
xkcd

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



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 = $(x, y, 0)^T$

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

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

Scale

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

Rotation

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

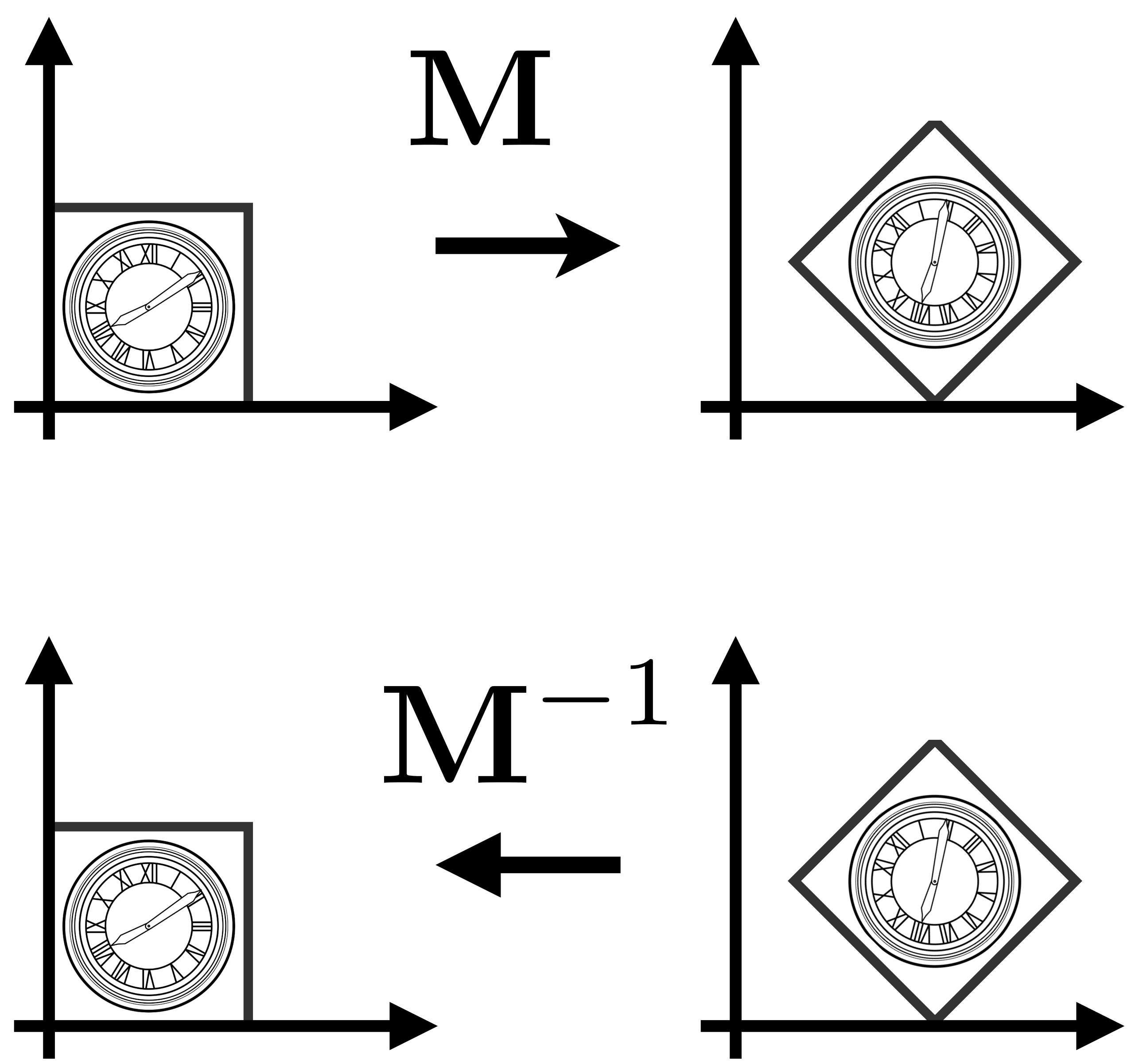
Translation

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

Inverse Transform

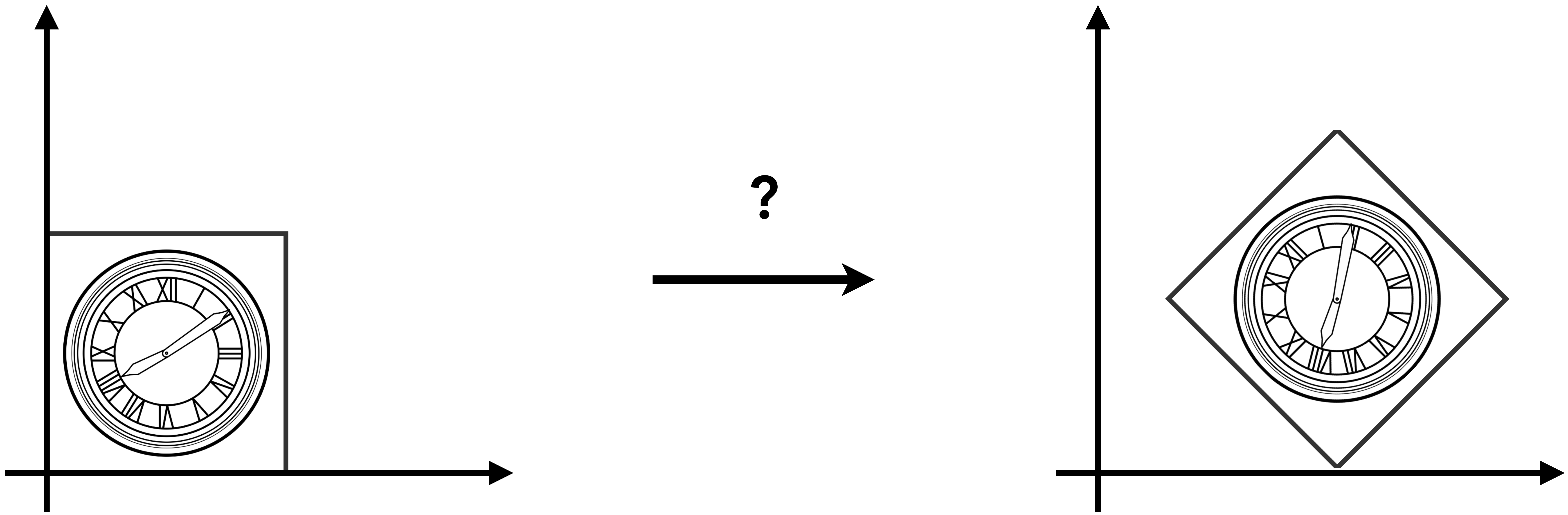
$$\mathrm{M}^{-1}$$

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

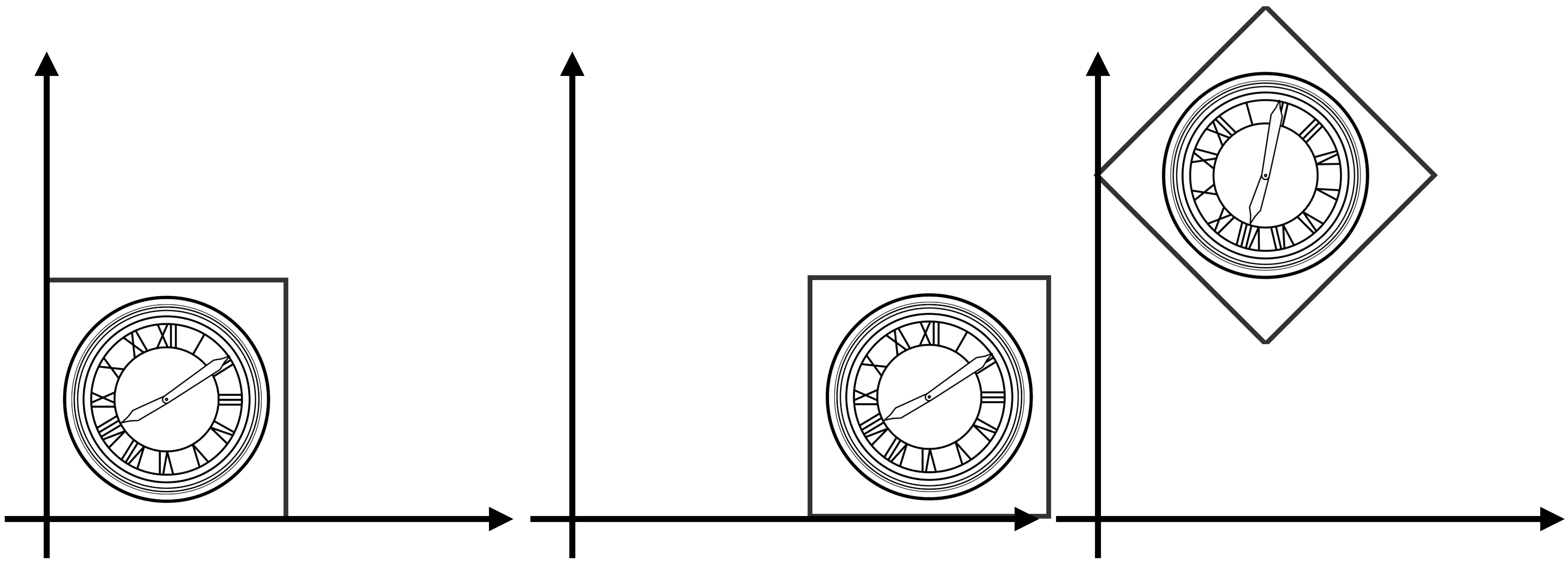


Composing Transforms

Composite Transform



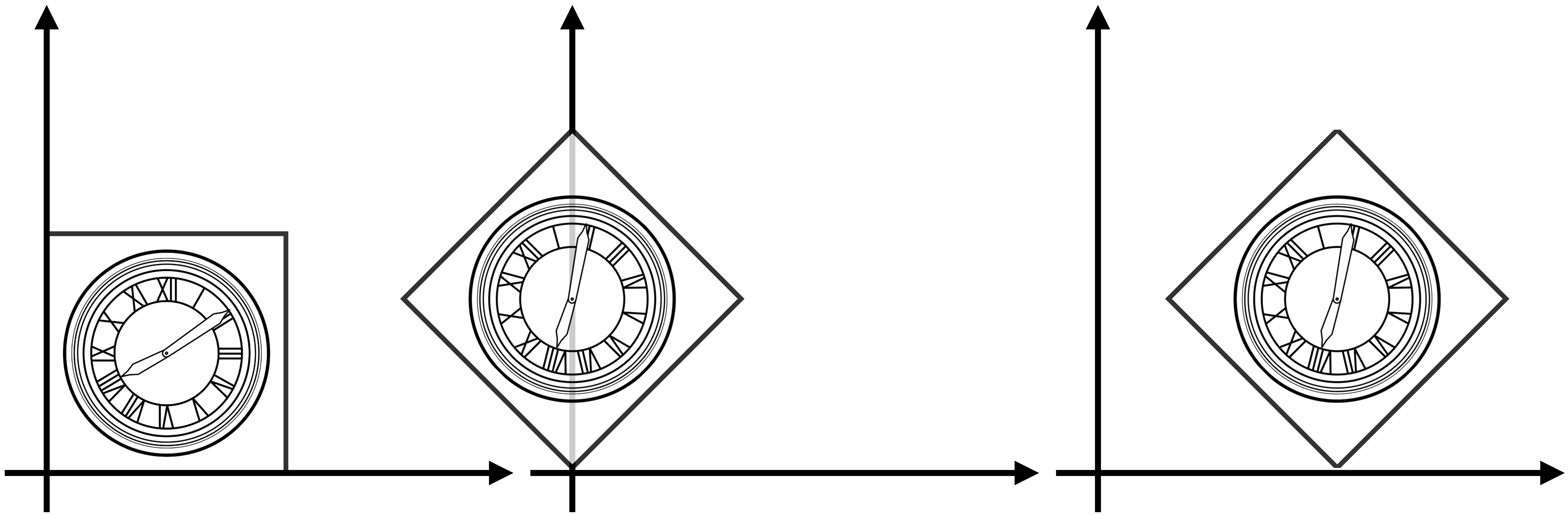
Translate Then Rotate?



$T_{(1,0)}$

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

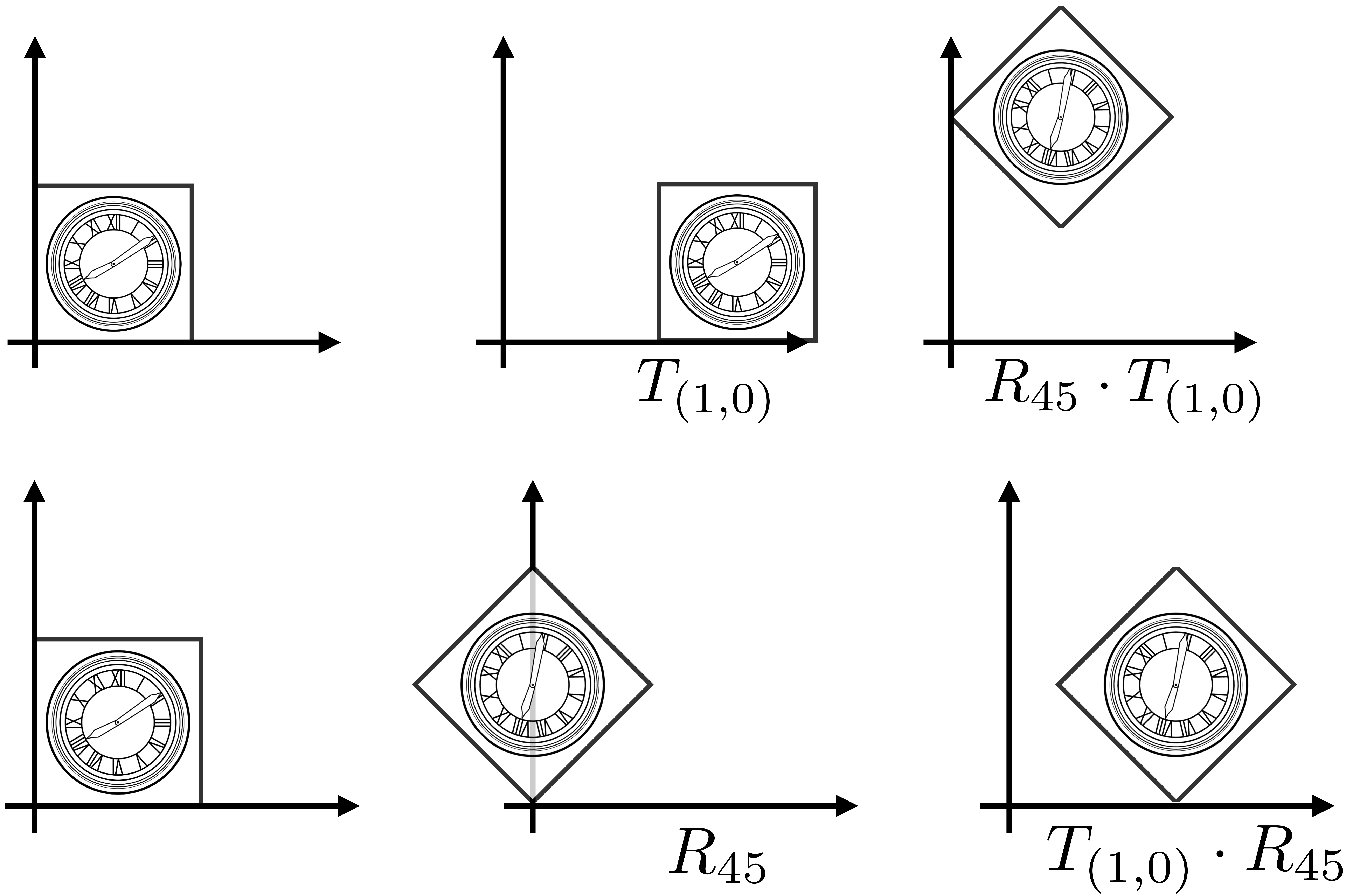
Rotate Then Translate



R_{45}

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

Transform Ordering Matters!



Transform Ordering Matters!

Matrix multiplication is <u>not</u> 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 affine transforms A_1, A_2, A_3, ...

- Compose by matrix multiplication
  - Very important for performance!

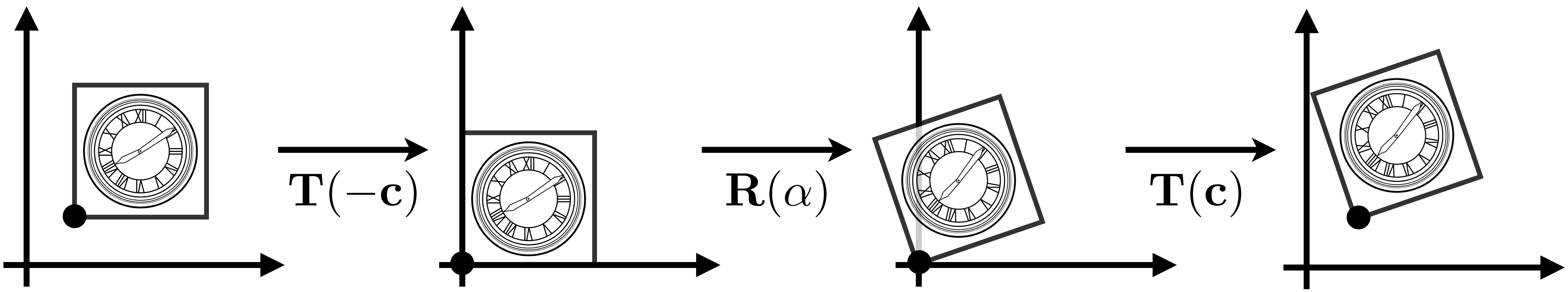
$$A_n(\ldots A_2(A_1(\mathbf{x}))) \;=\; \mathbf{A}_n \cdots \mathbf{A}_2 \cdot \mathbf{A}_1 \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

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

Decomposing Complex Transforms

How to rotate around a given point c?

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



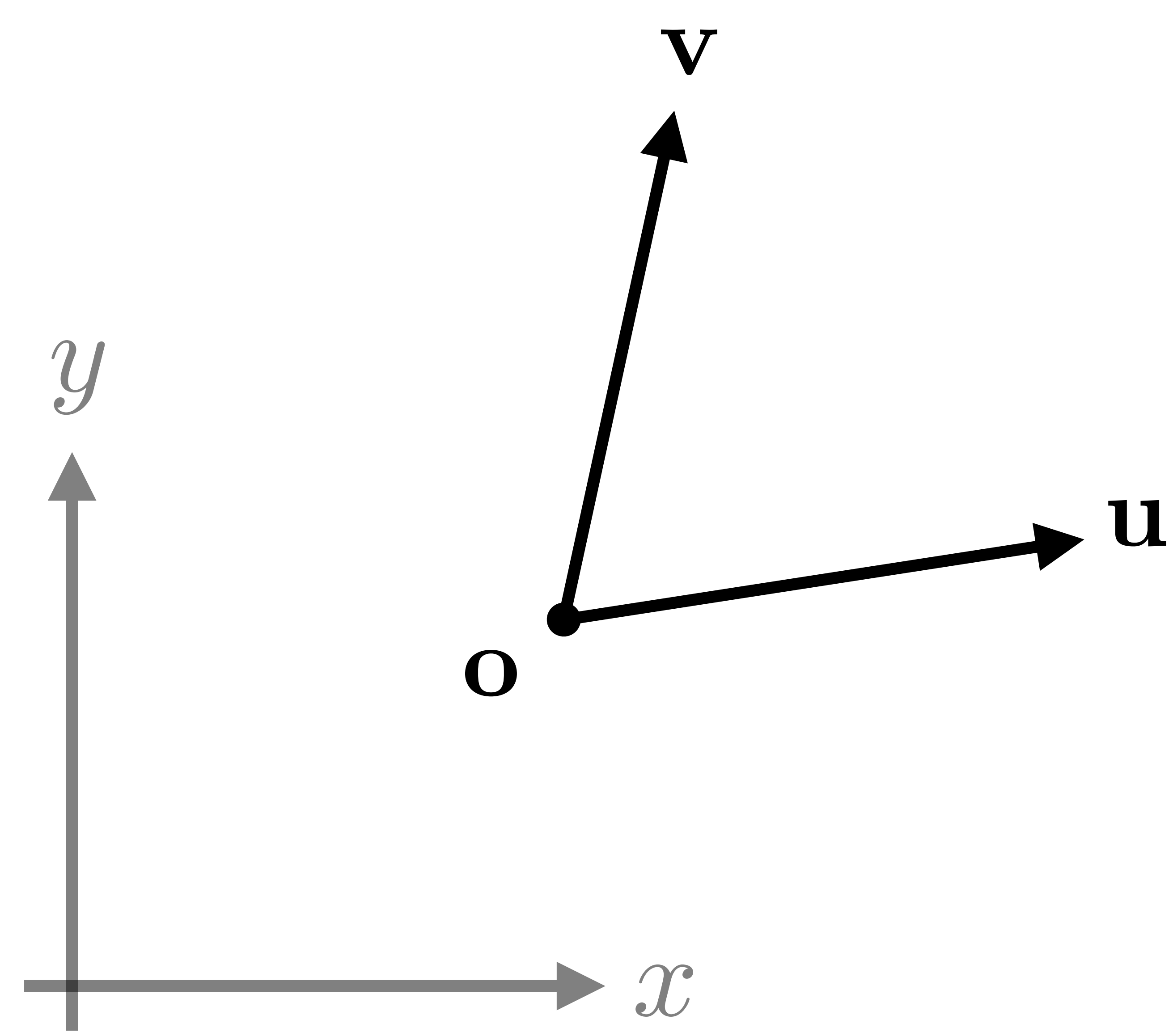
Matrix representation?

$$\mathbf{T}(\mathbf{c}) \cdot \mathbf{R}(\alpha) \cdot \mathbf{T}(-\mathbf{c})$$

Coordinate Systems

Coordinate System Transform

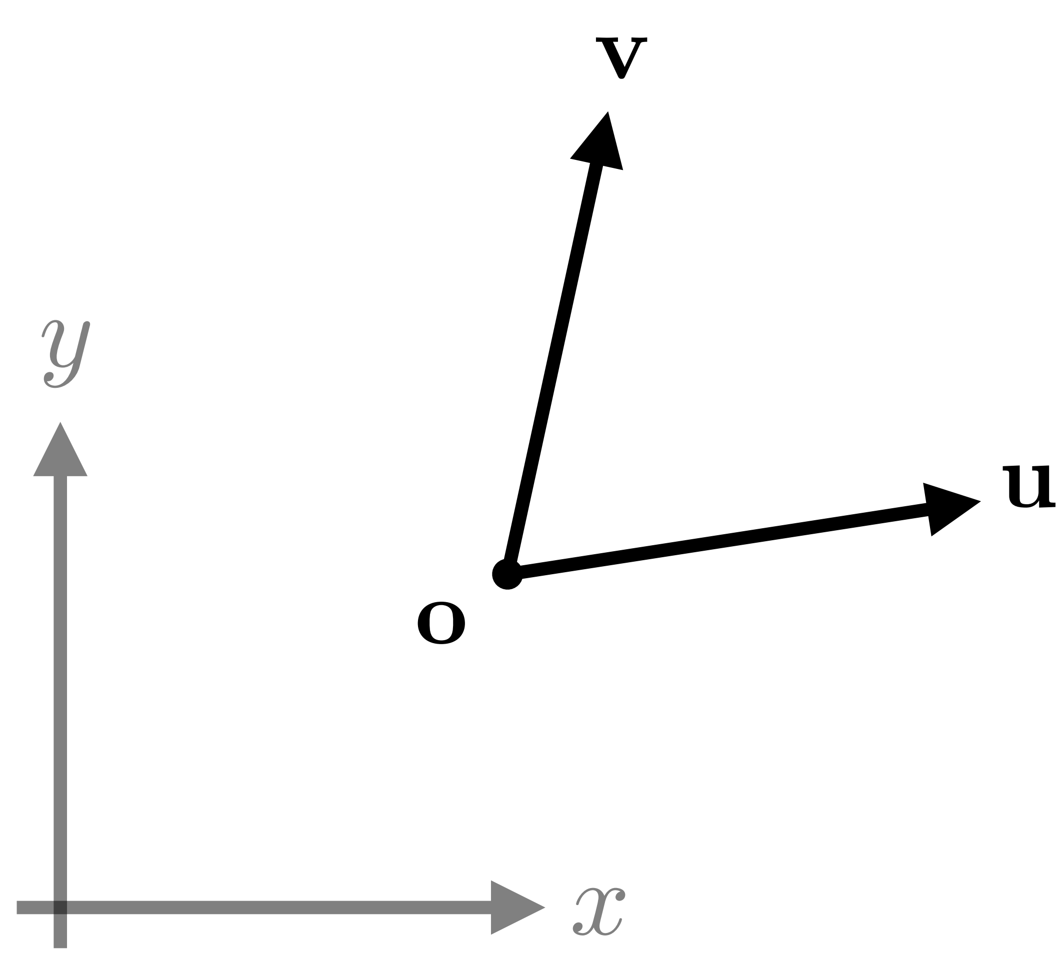
In general, a new coordinate frame is defined by an origin (point) and two unit axes (vectors)



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

Coordinate System Transform Matrix

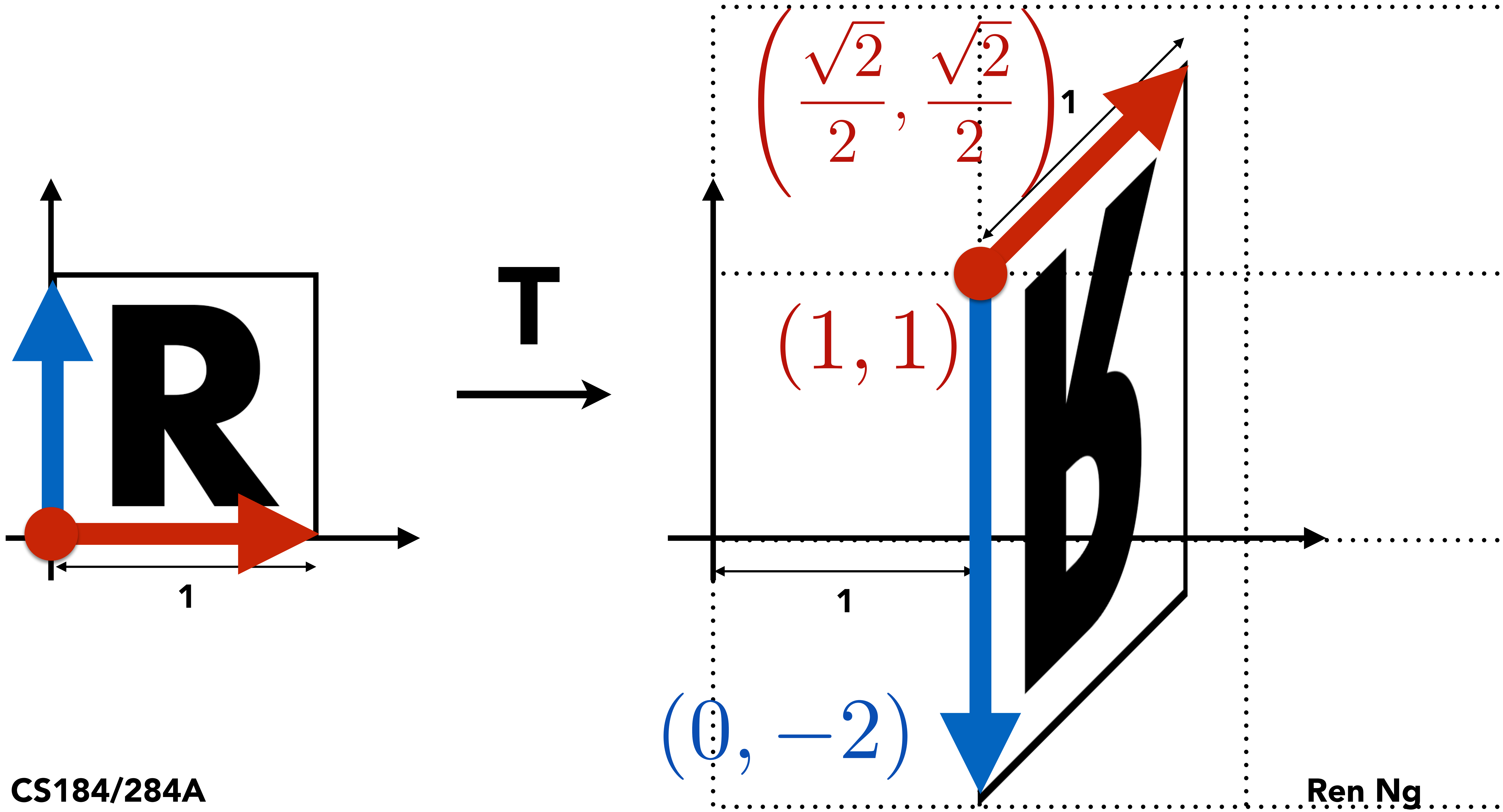
$$F = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{o} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u_x & v_x & o_x \\ u_y & v_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$



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

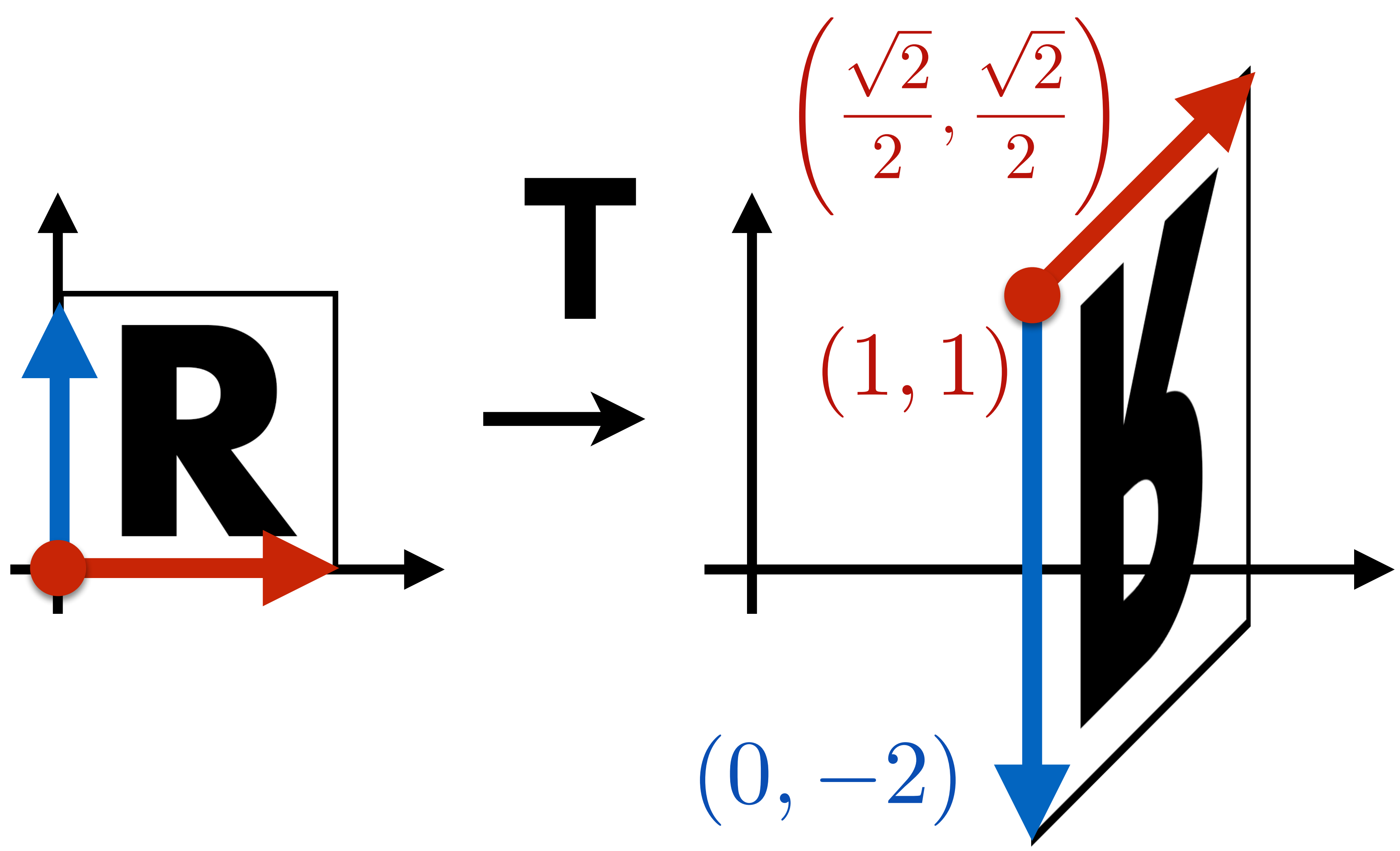
Coordinate System Transform - Example

Write down a matrix T representing this transform:



Coordinate System Transform - Example

Write down a matrix T representing this transform:

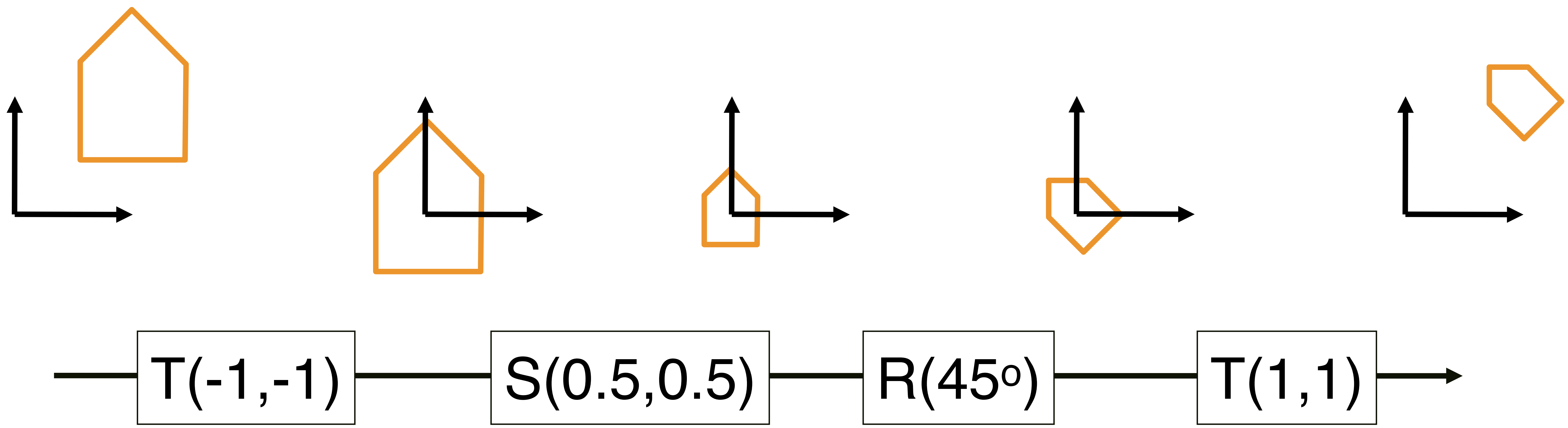


$$\begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{o} \\ 0 & 0 & 1 \end{bmatrix}$$

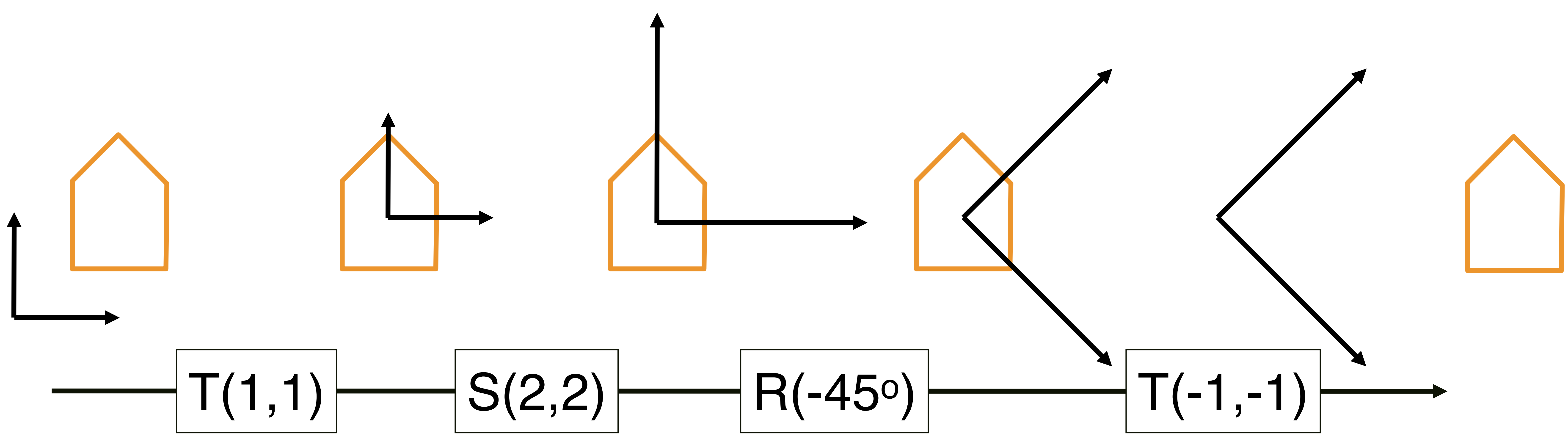
$$\begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & 1 \\ \frac{\sqrt{2}}{2} & -2 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Two Interpretations of A Transform

Interpretation 1: Transforms object points



Interpretation 2: Transforms coordinate system



3D Transforms

3D Transformations

Use homogeneous coordinates again:

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

Use 4×4 matrices for affine transformations

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

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)

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

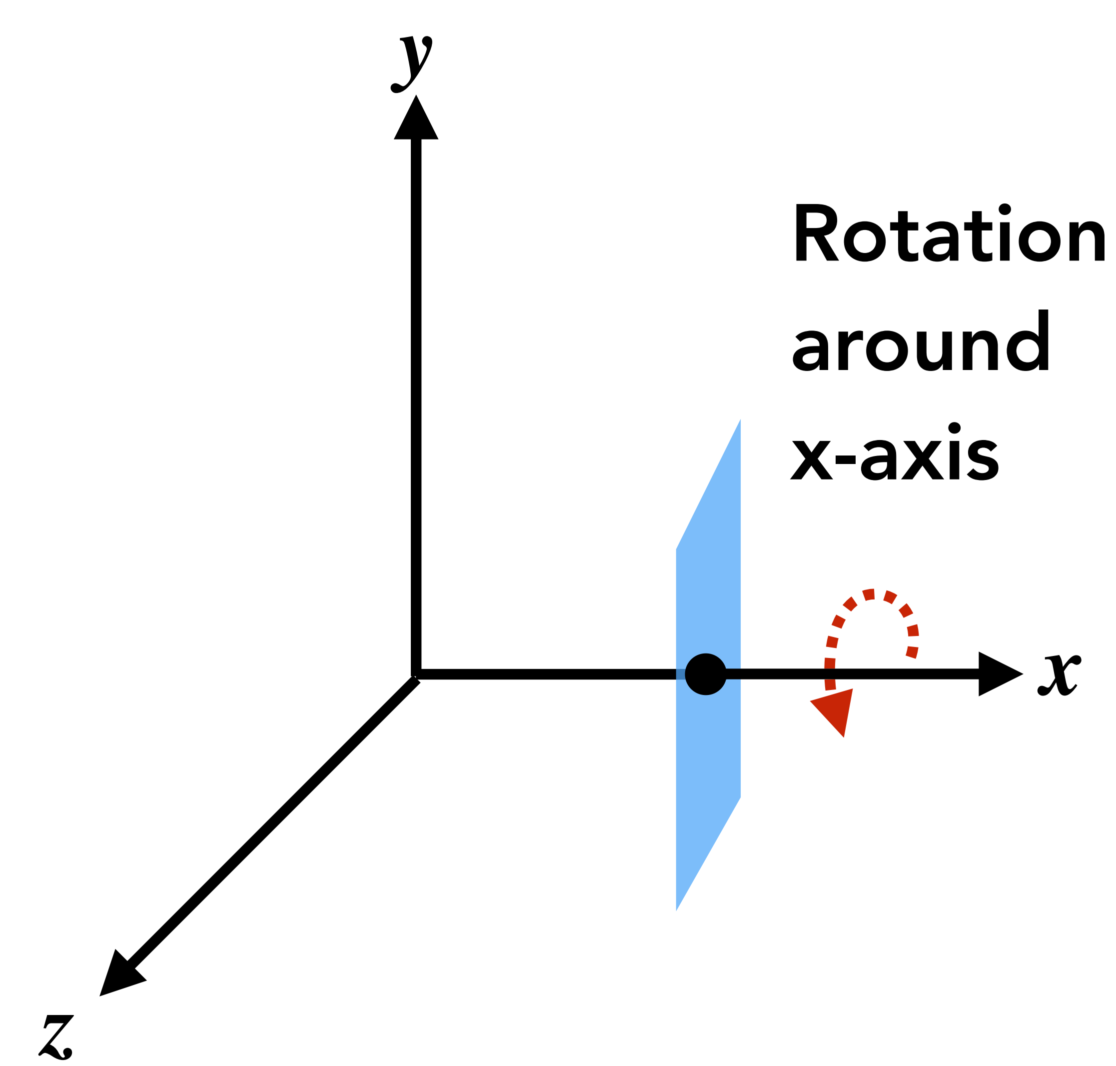
3D Transformations

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

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

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

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

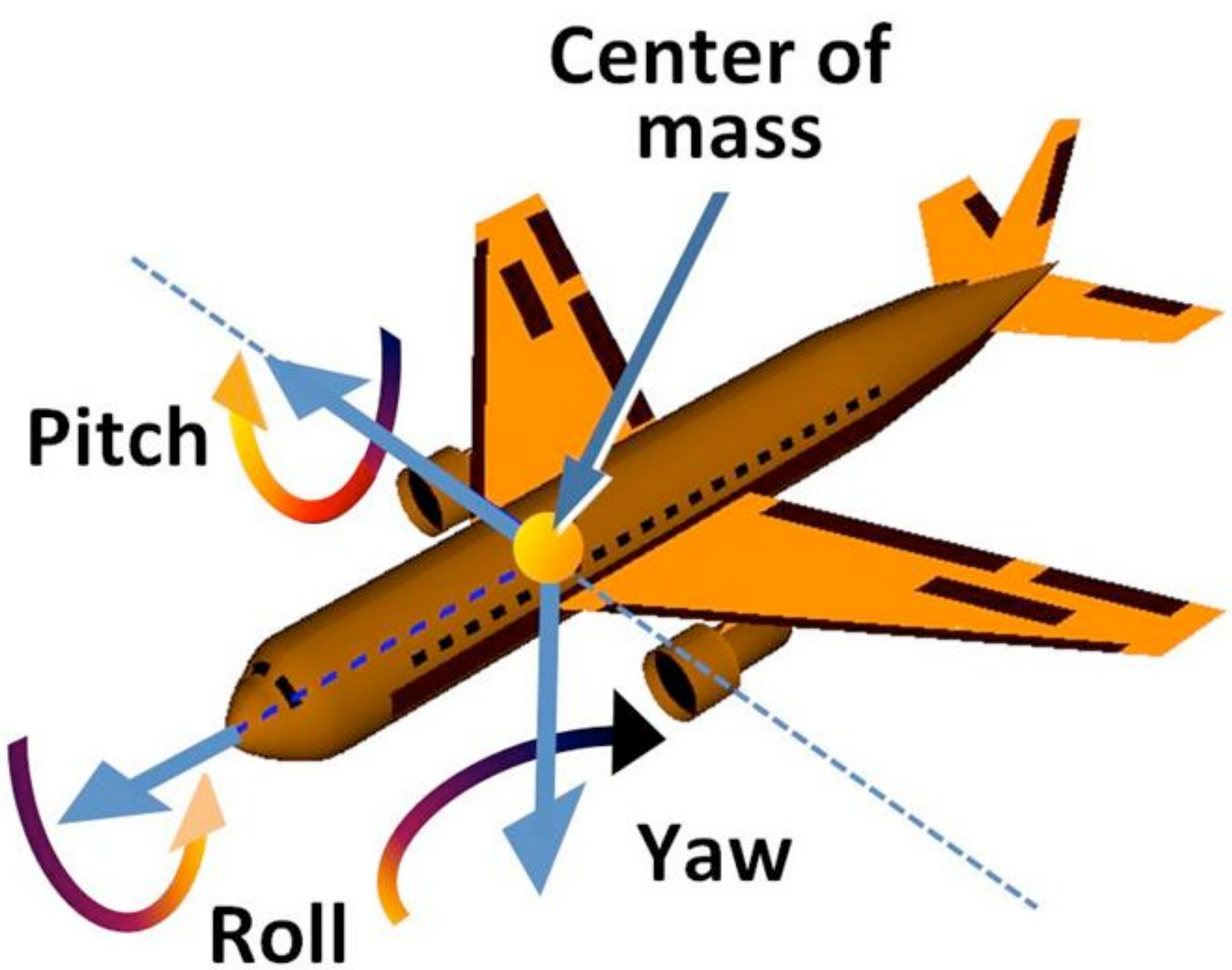


3D Rotations

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

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

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



3D Rotations

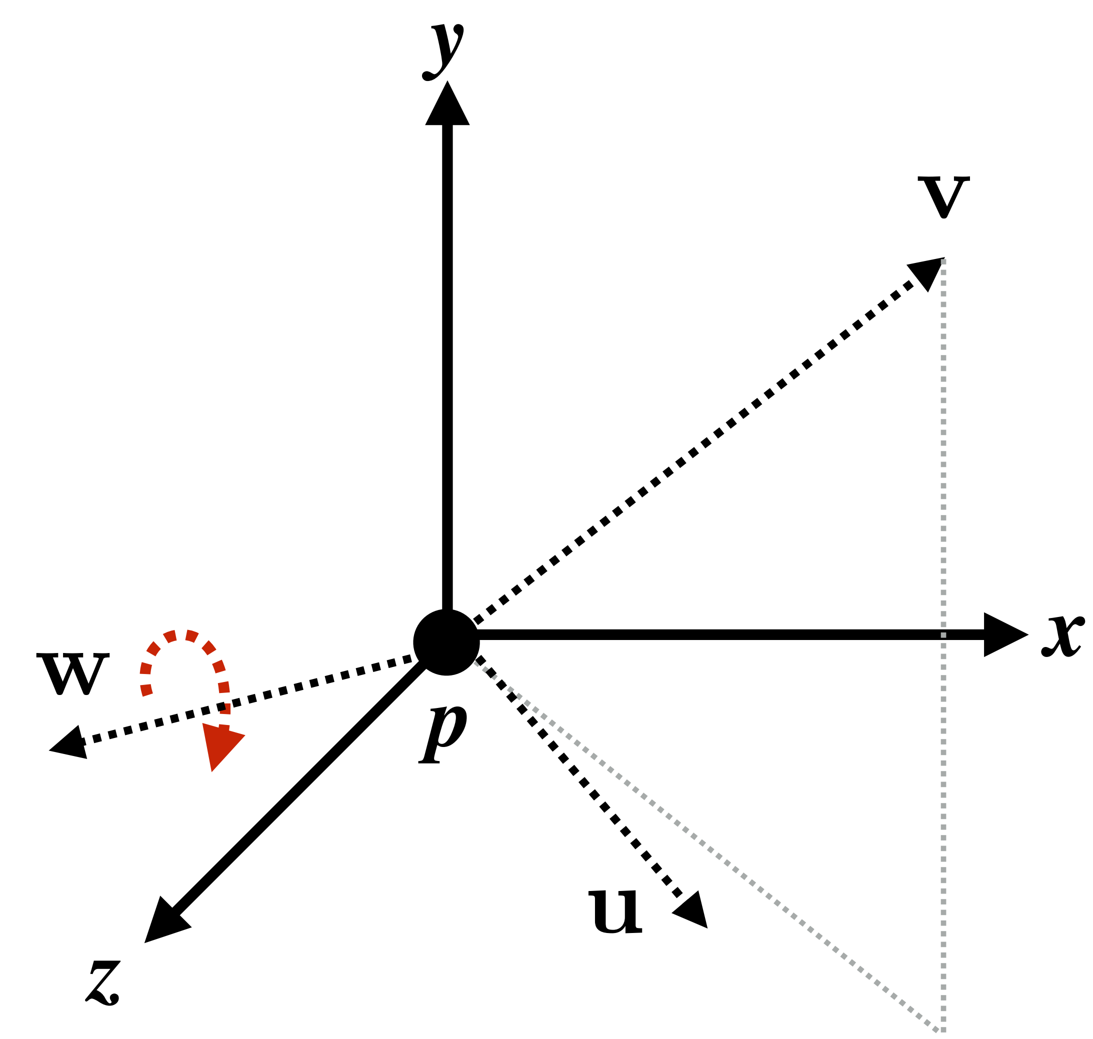
Compose any 3D rotation from $\mathbf{R}_x, \mathbf{R}_y, \mathbf{R}_z$?

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

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

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:

- Move to Z axis, rotate, then move back

Rodrigues' Rotation Formula

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

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

How to prove this magic formula?

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

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

Many Other Representations of Rotations

Quaternions

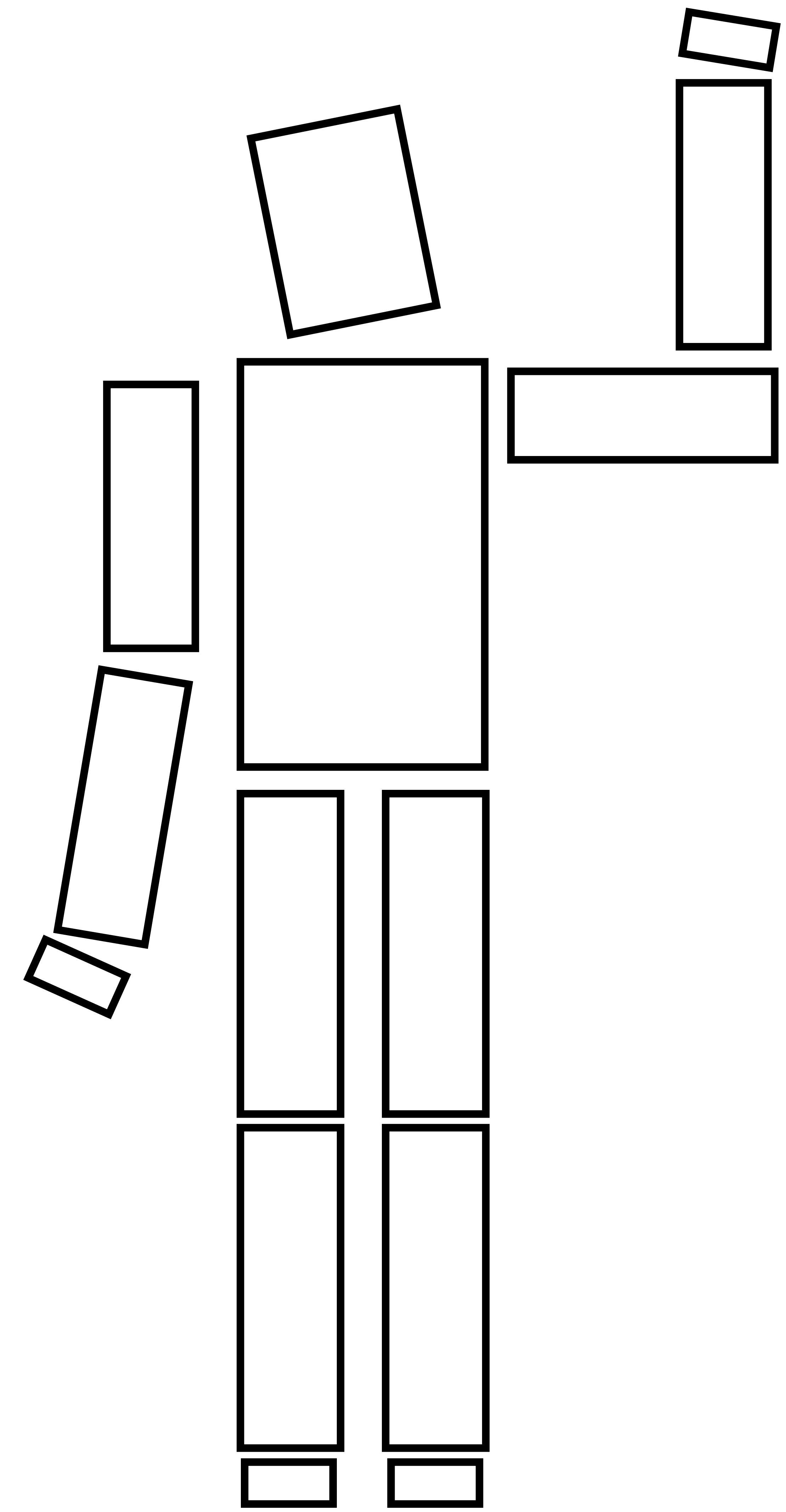
Exponential map

…

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

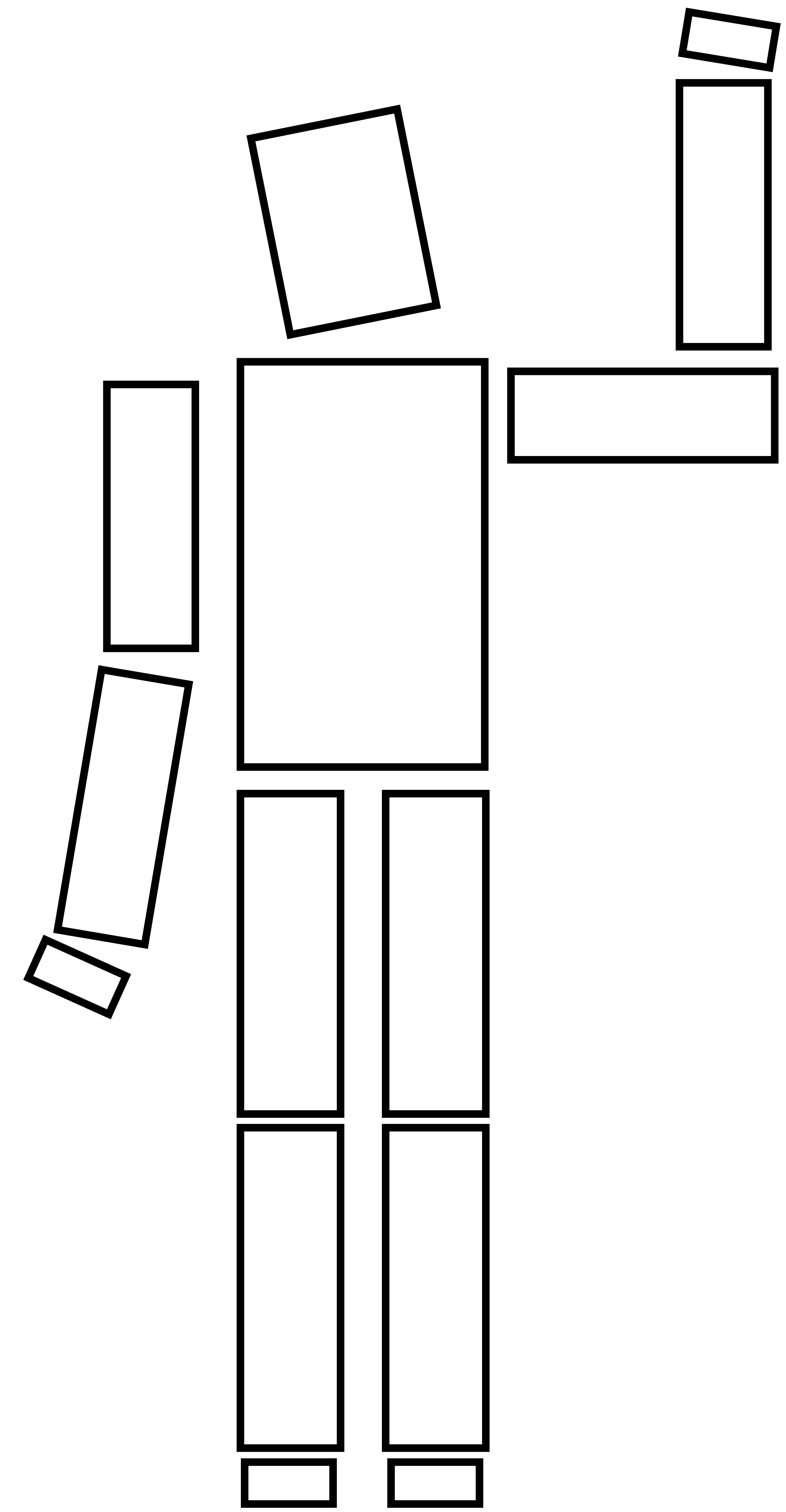
Each shape associated with its own transform

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



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

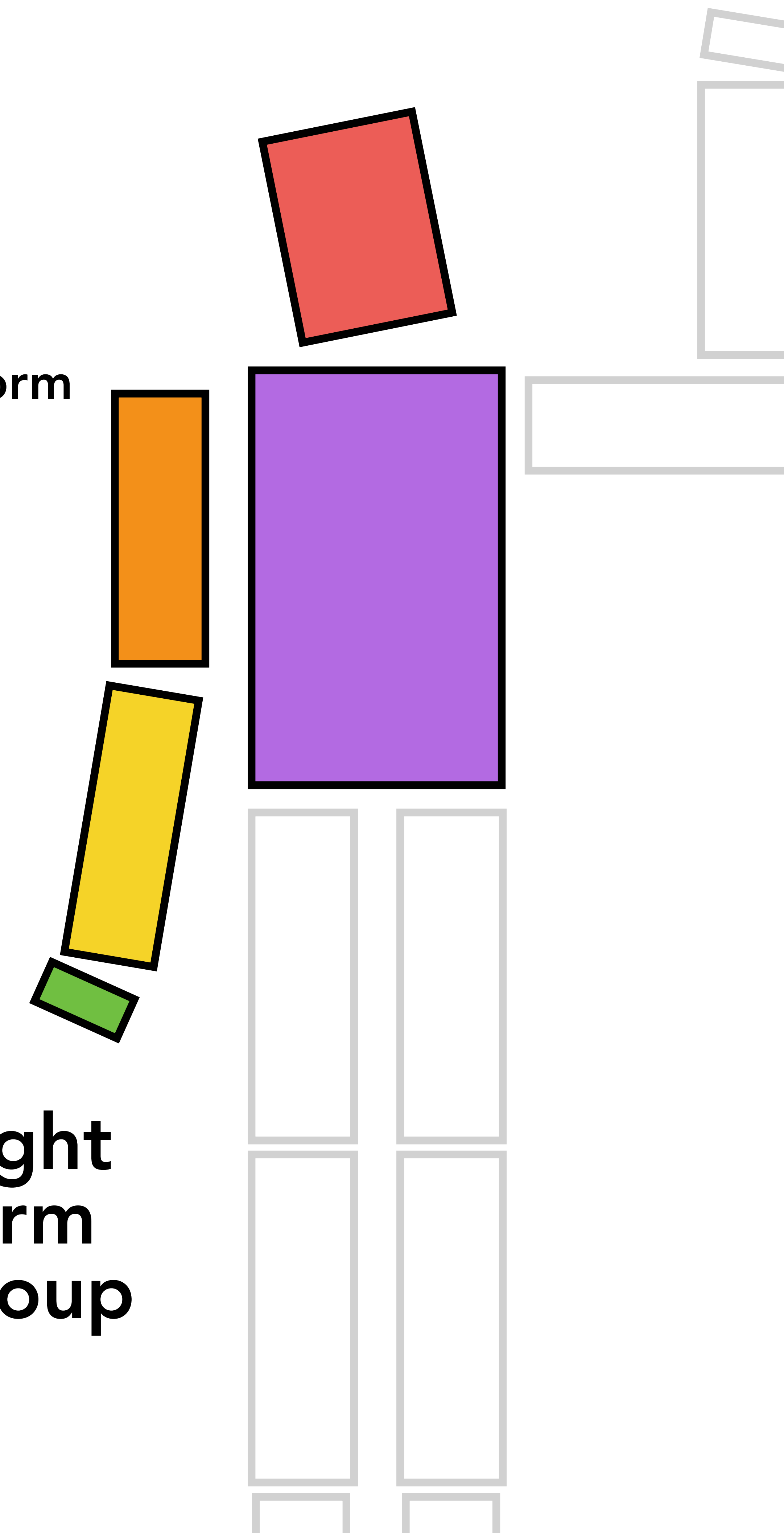
Skeleton - Hierarchical Representation

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



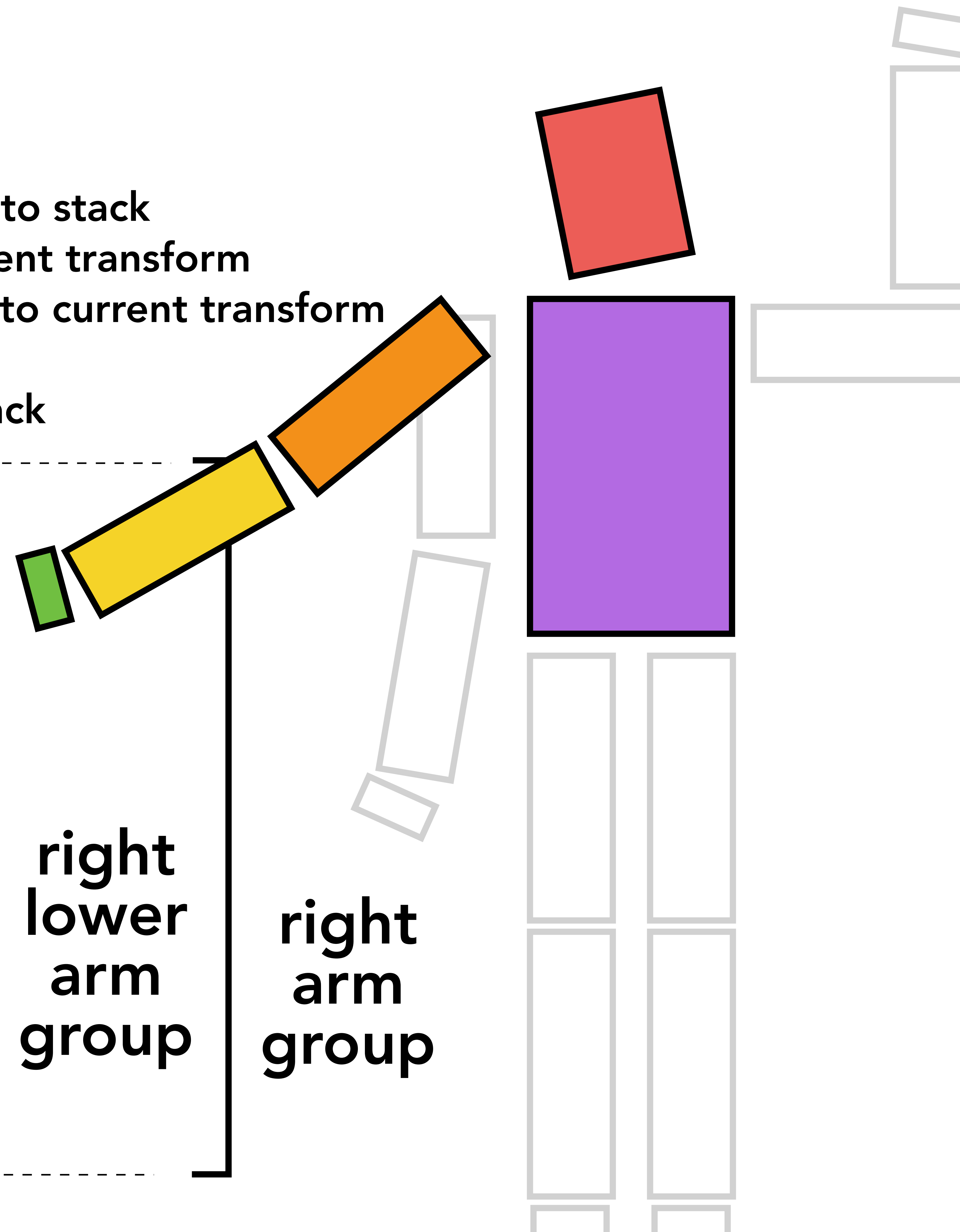
Skeleton - Hierarchical Representation

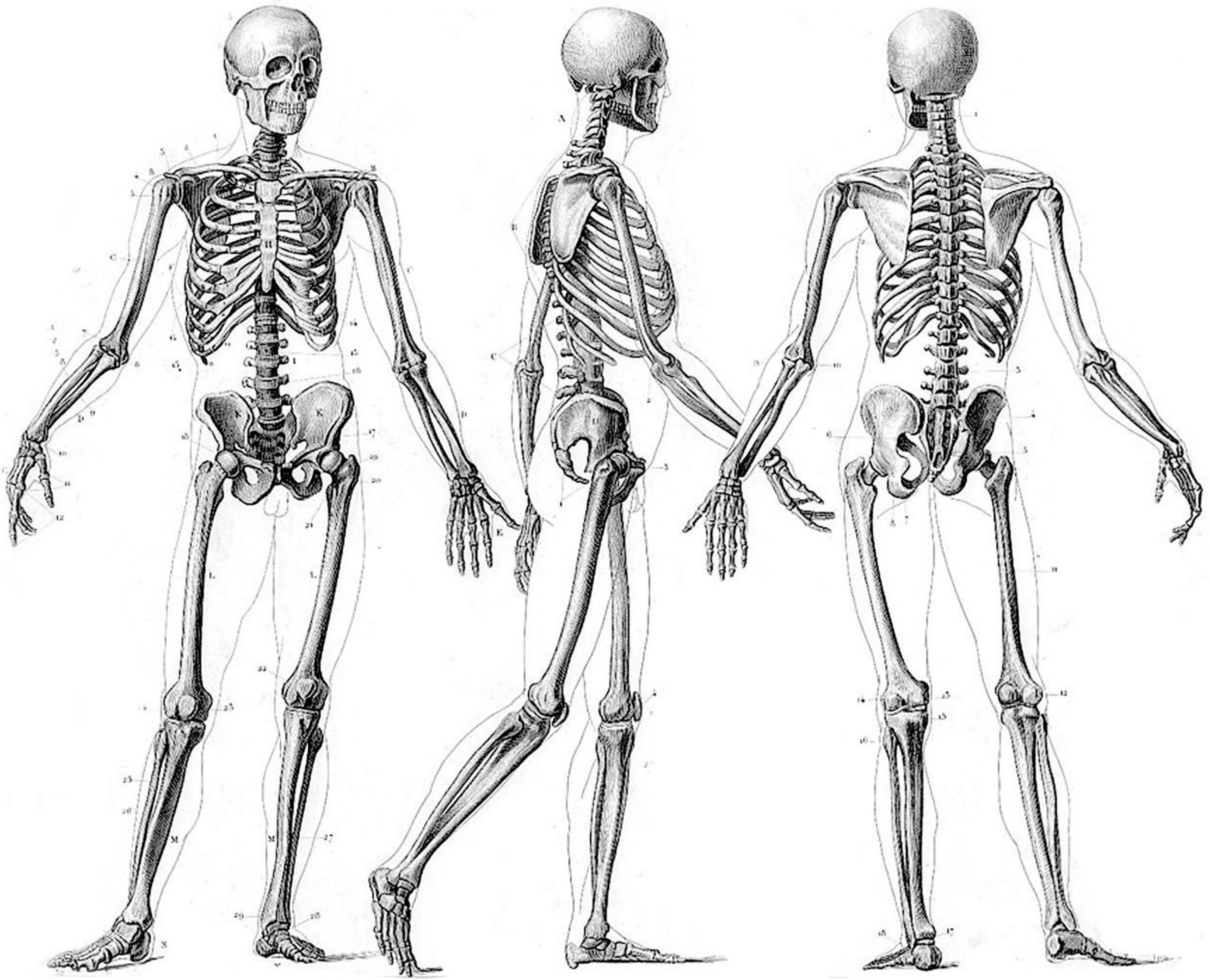
```
translate(0, 10);
    drawTorso();
        pushmatrix();  // push a copy of transform onto stack
            translate(0, 5);  // right-multiply onto current transform
            rotate(headRotation);  // right-multiply onto current transform
            drawHead();
        popmatrix();  // pop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

right hand

right lower arm group

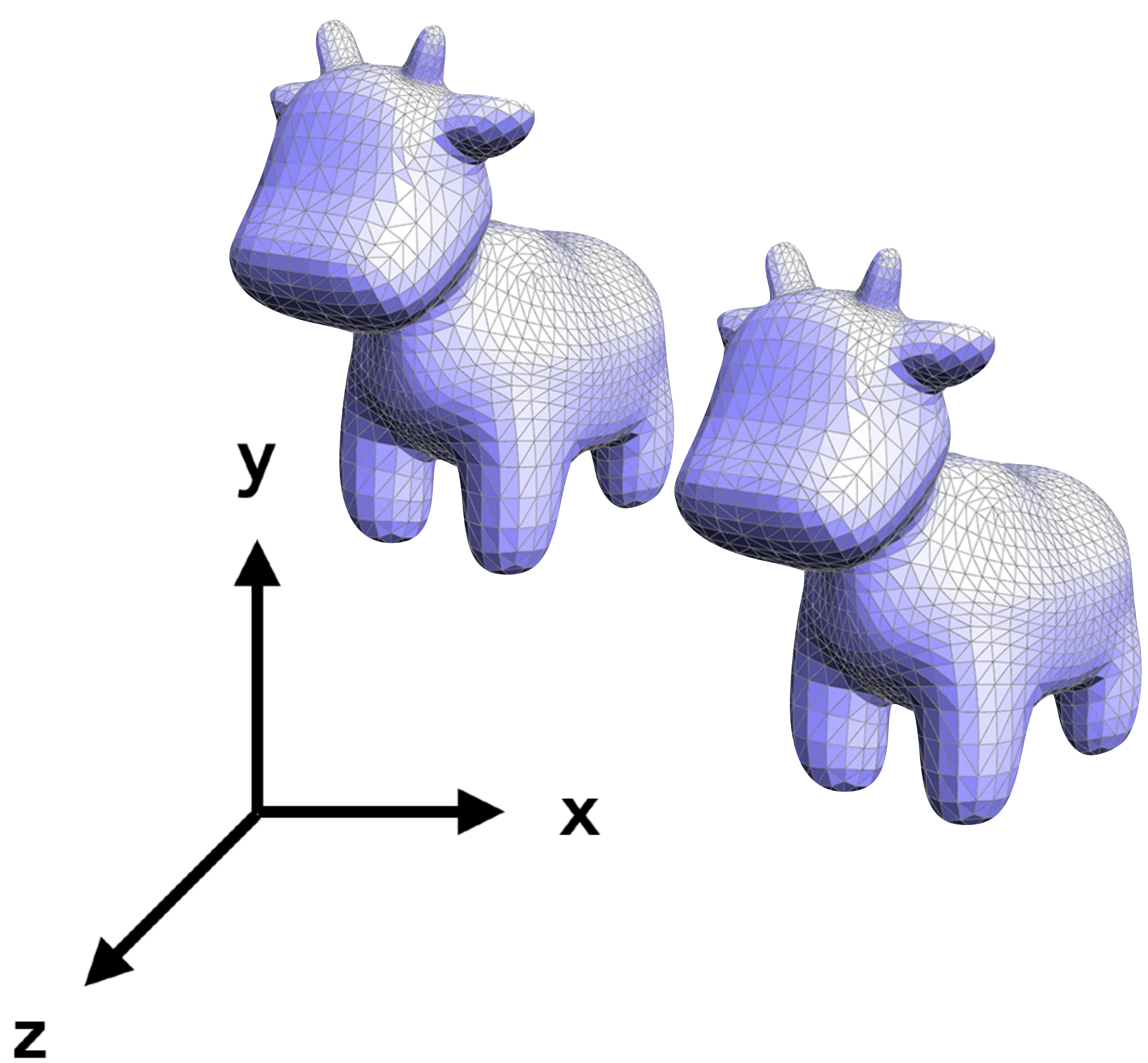
right arm group

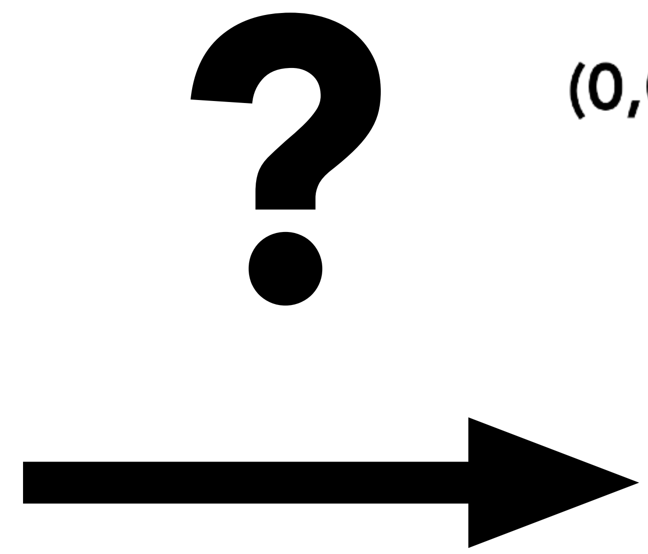


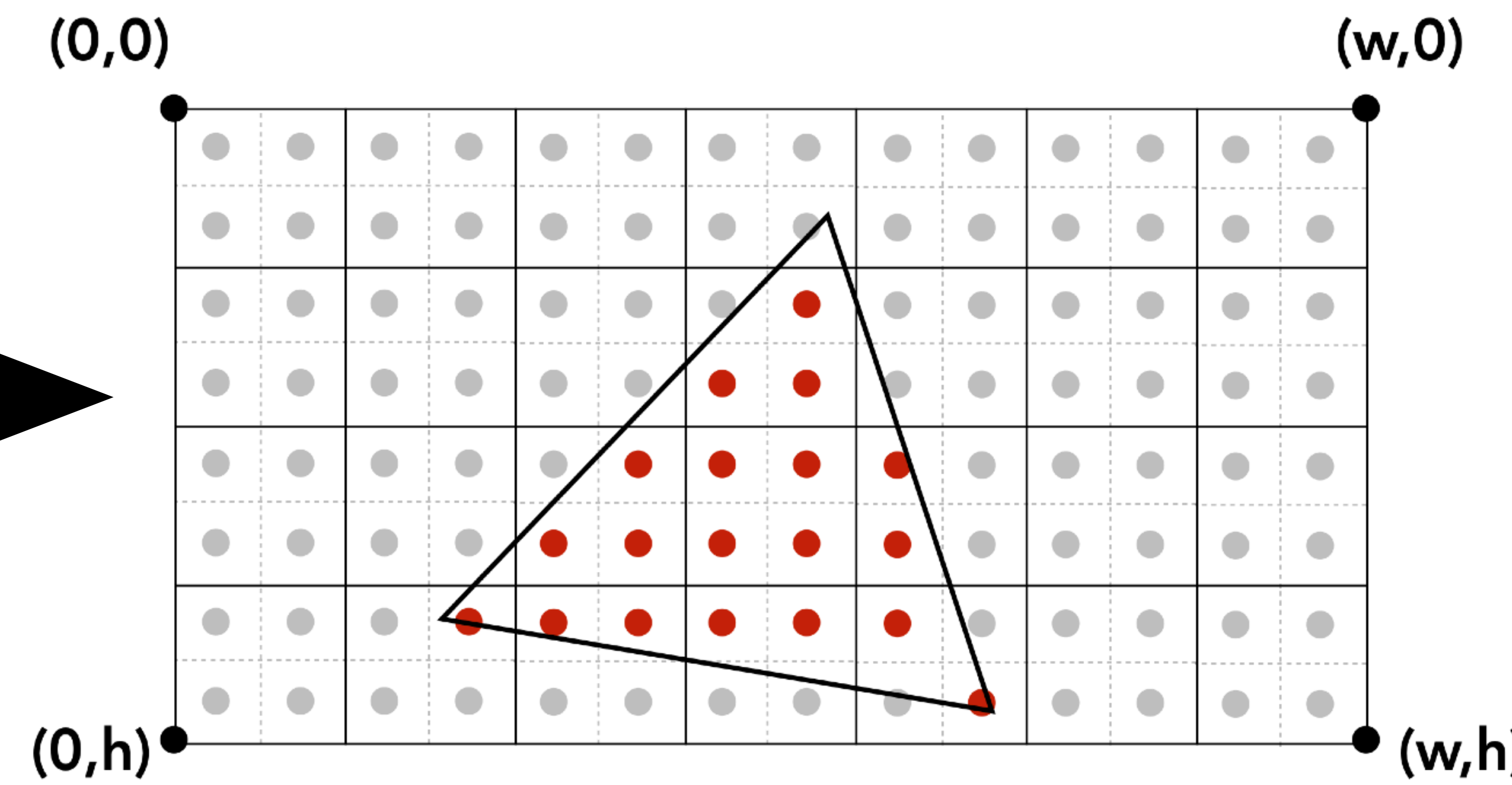


Viewing and Perspective

Viewing and Perspective Transforms







Scene modeling
3D world coordinates

Rasterization
2D screen coordinates

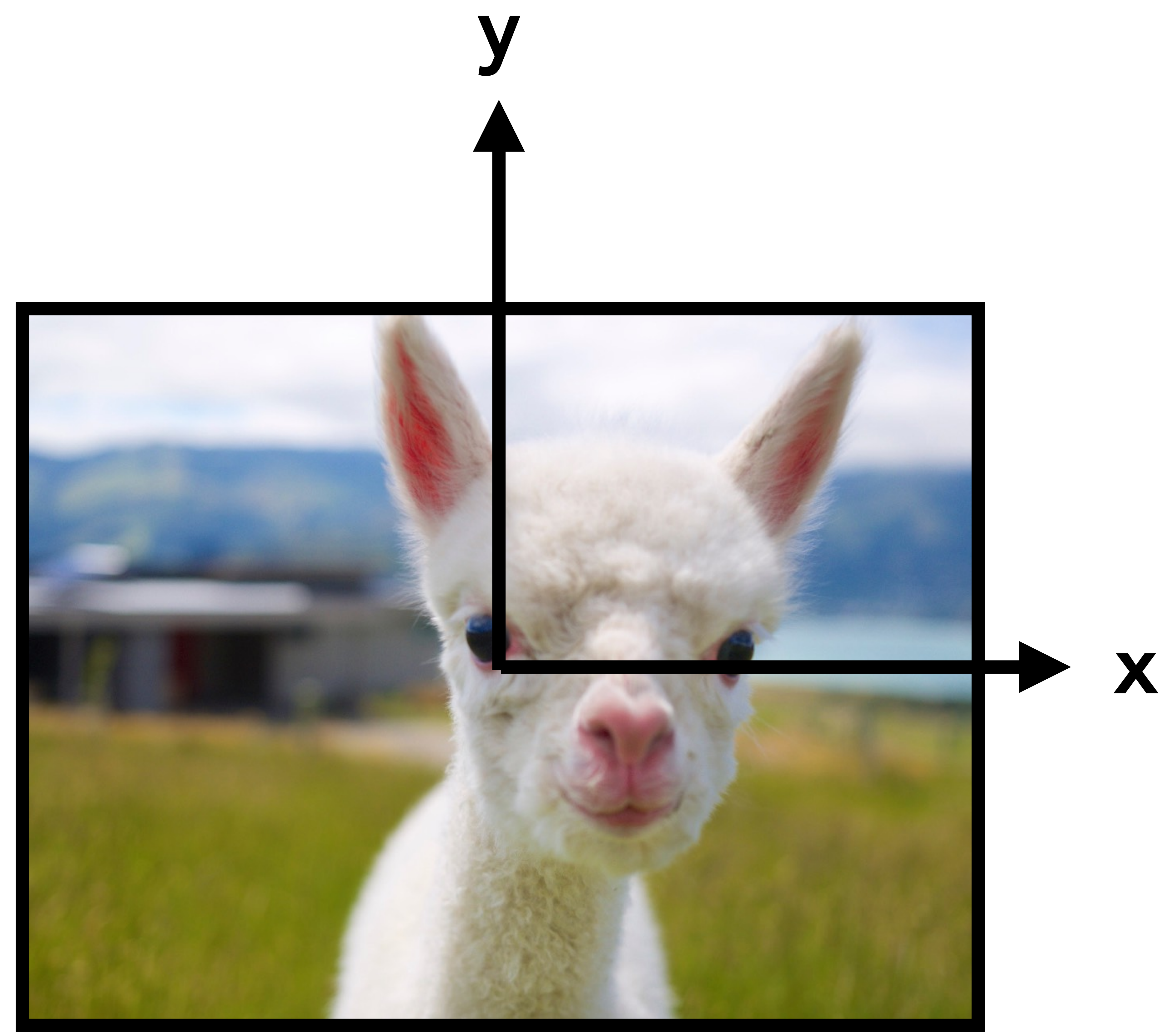
Camera Space

“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

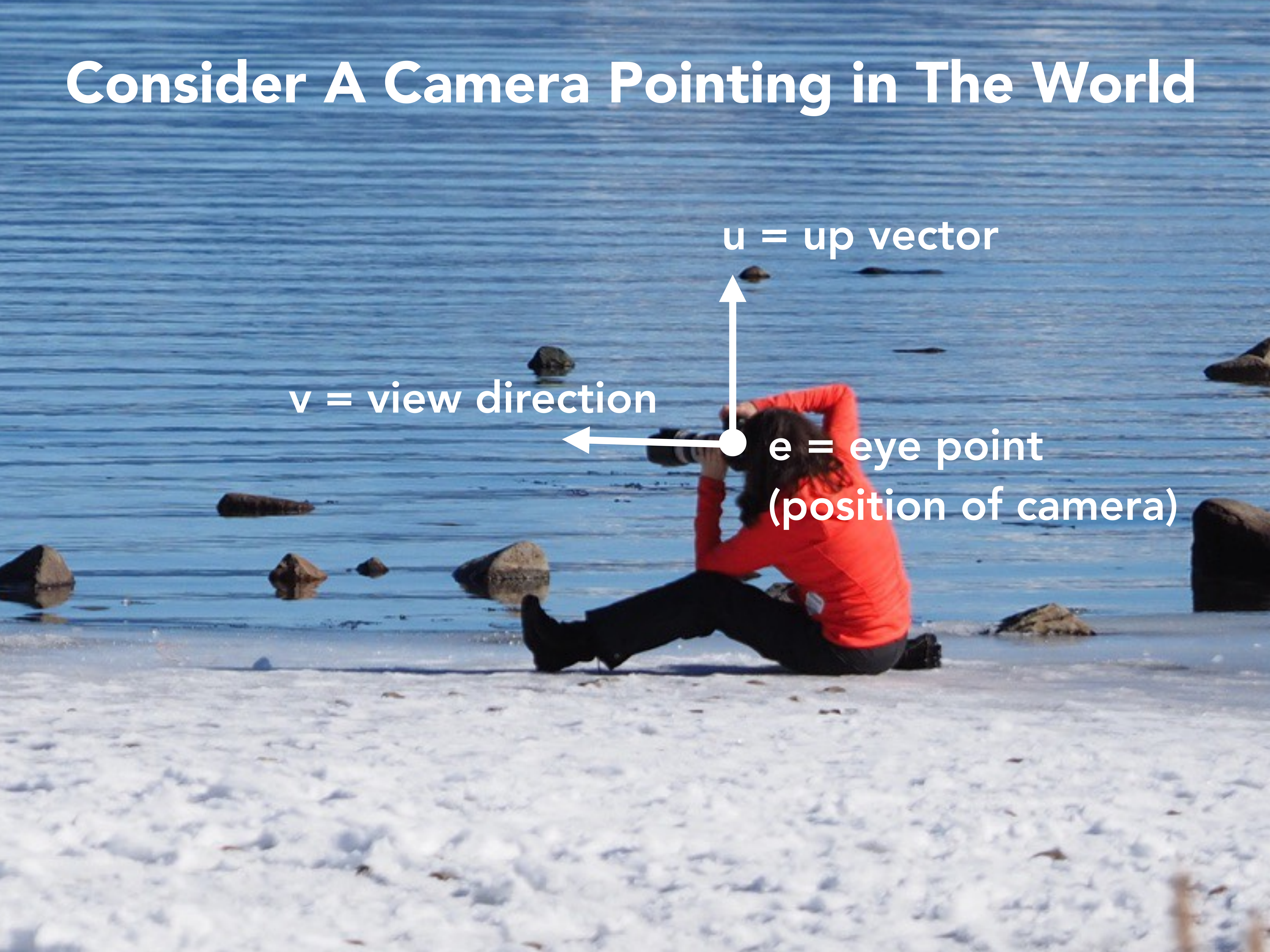


Resulting image

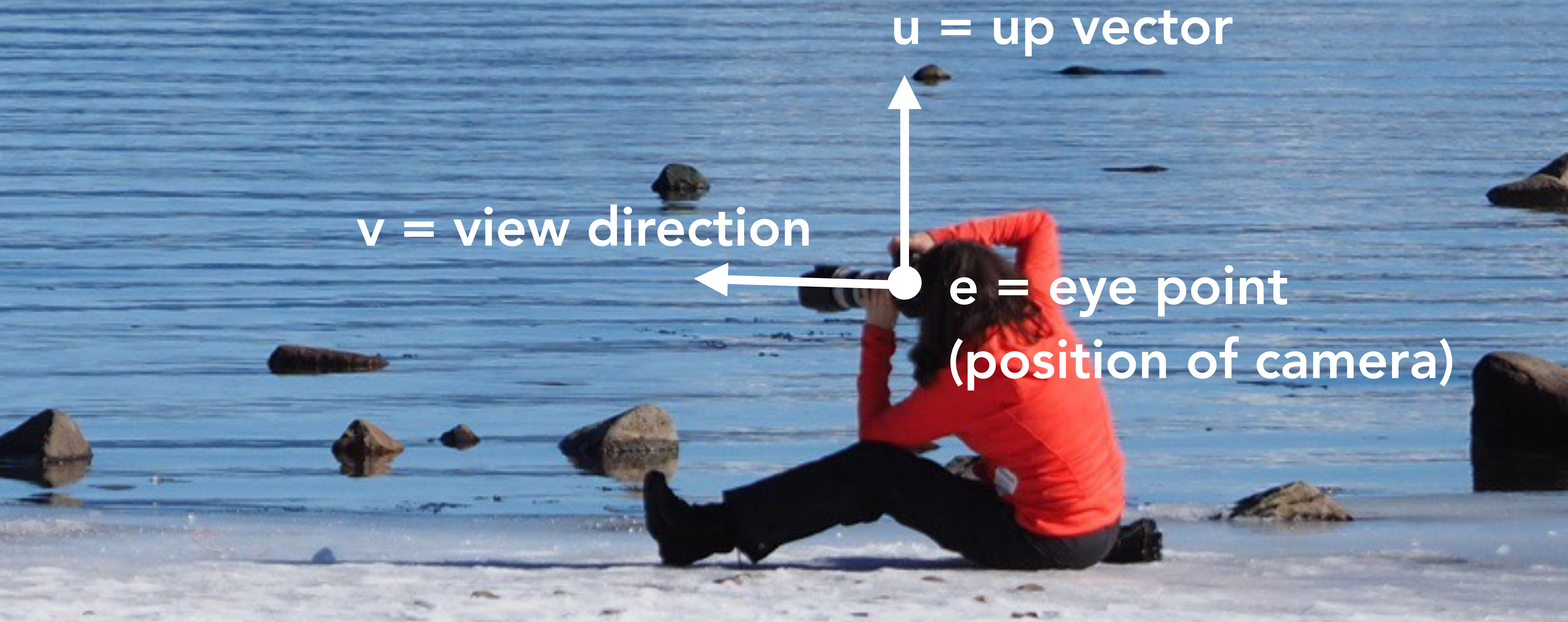
(z-axis pointing away from scene)

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

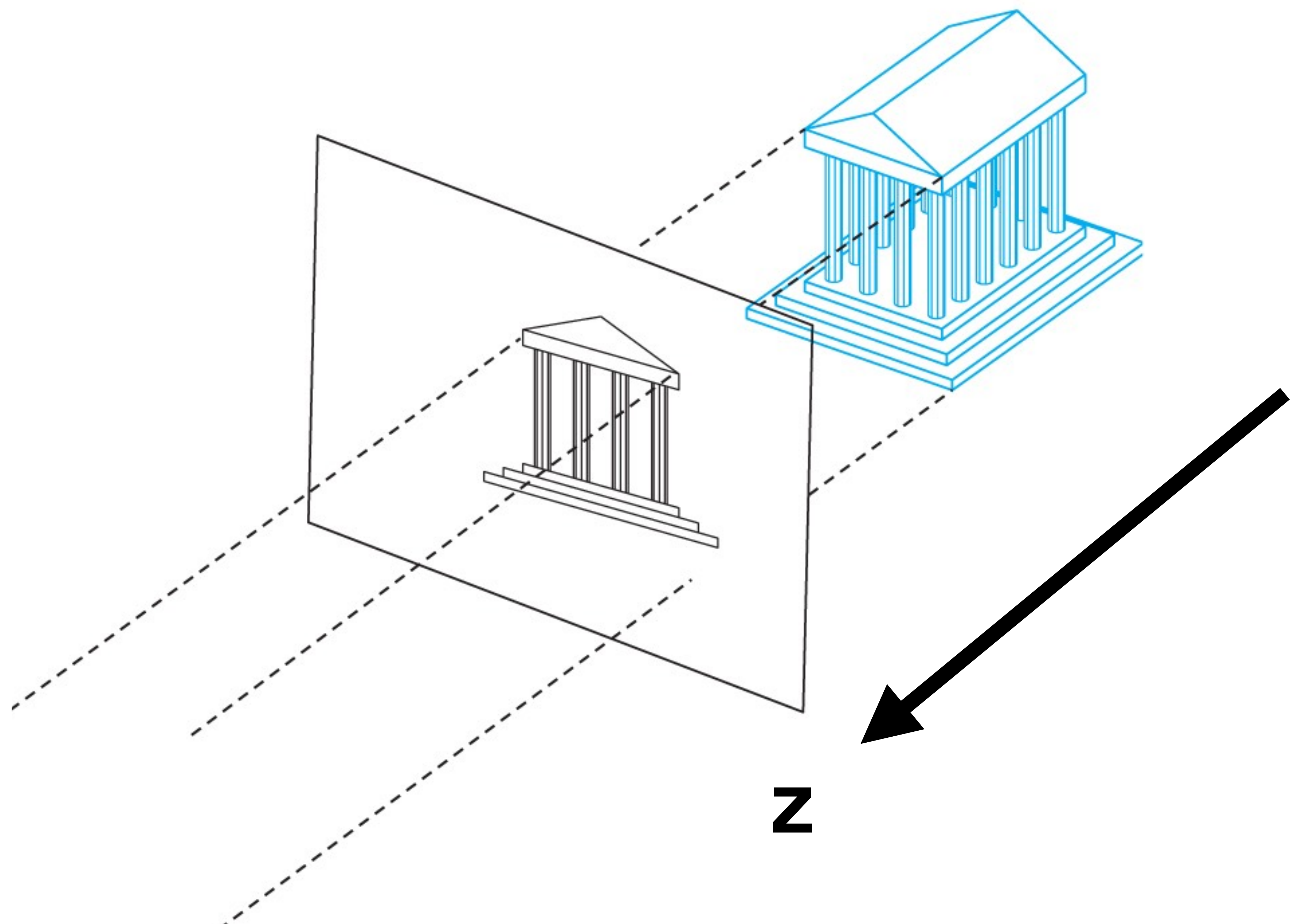
Inverse: Matrix from standard camera to world space (Why? This is a coordinate frame transform to (e,r,u,-v))

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

"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?

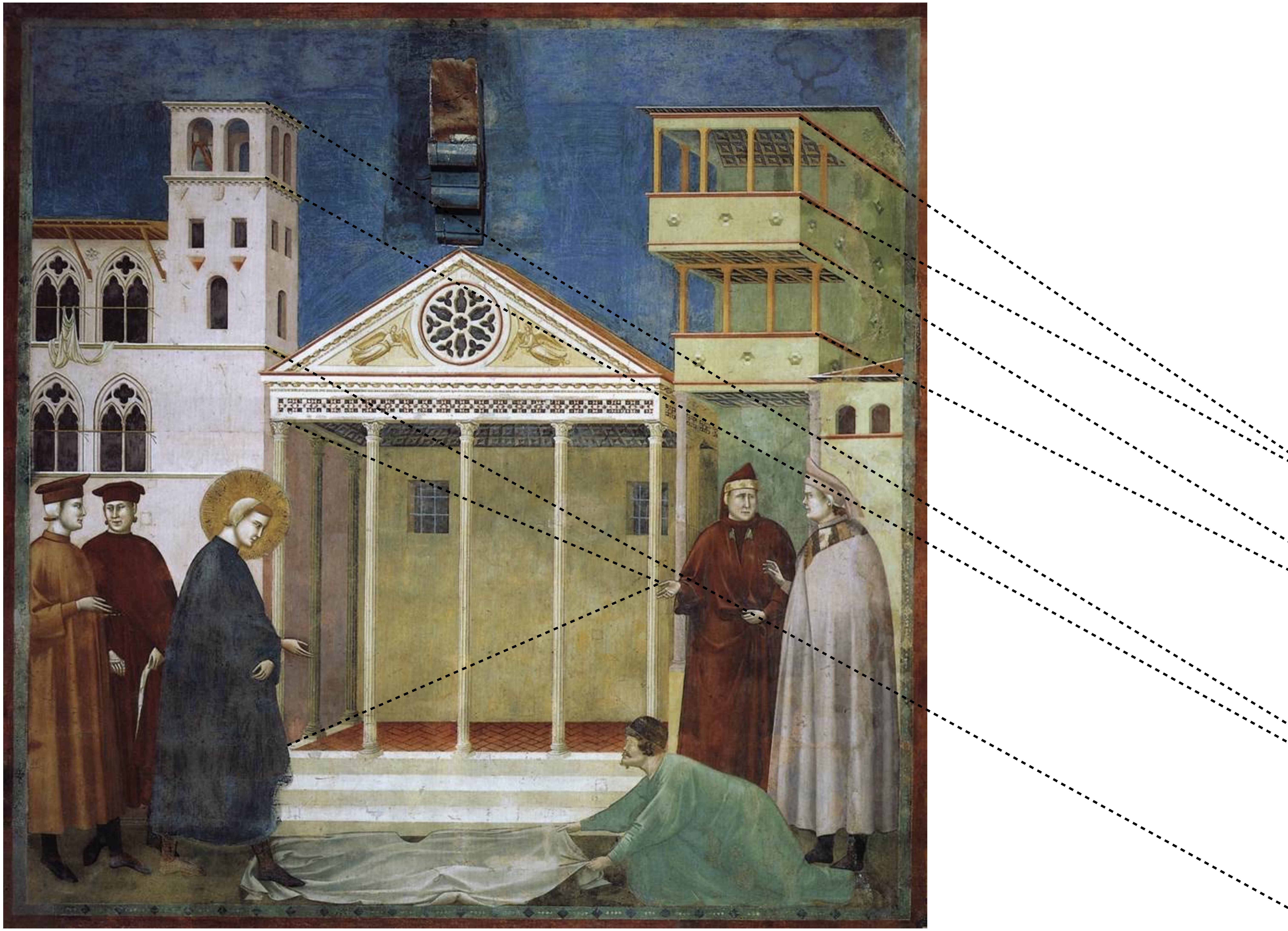
- One option: orthographic projection (just delete z)
- Useful, e.g. for engineering drawings
- But is this the whole story?


Perspective
Credit: jefflynchphoto.com

Perspective in Art

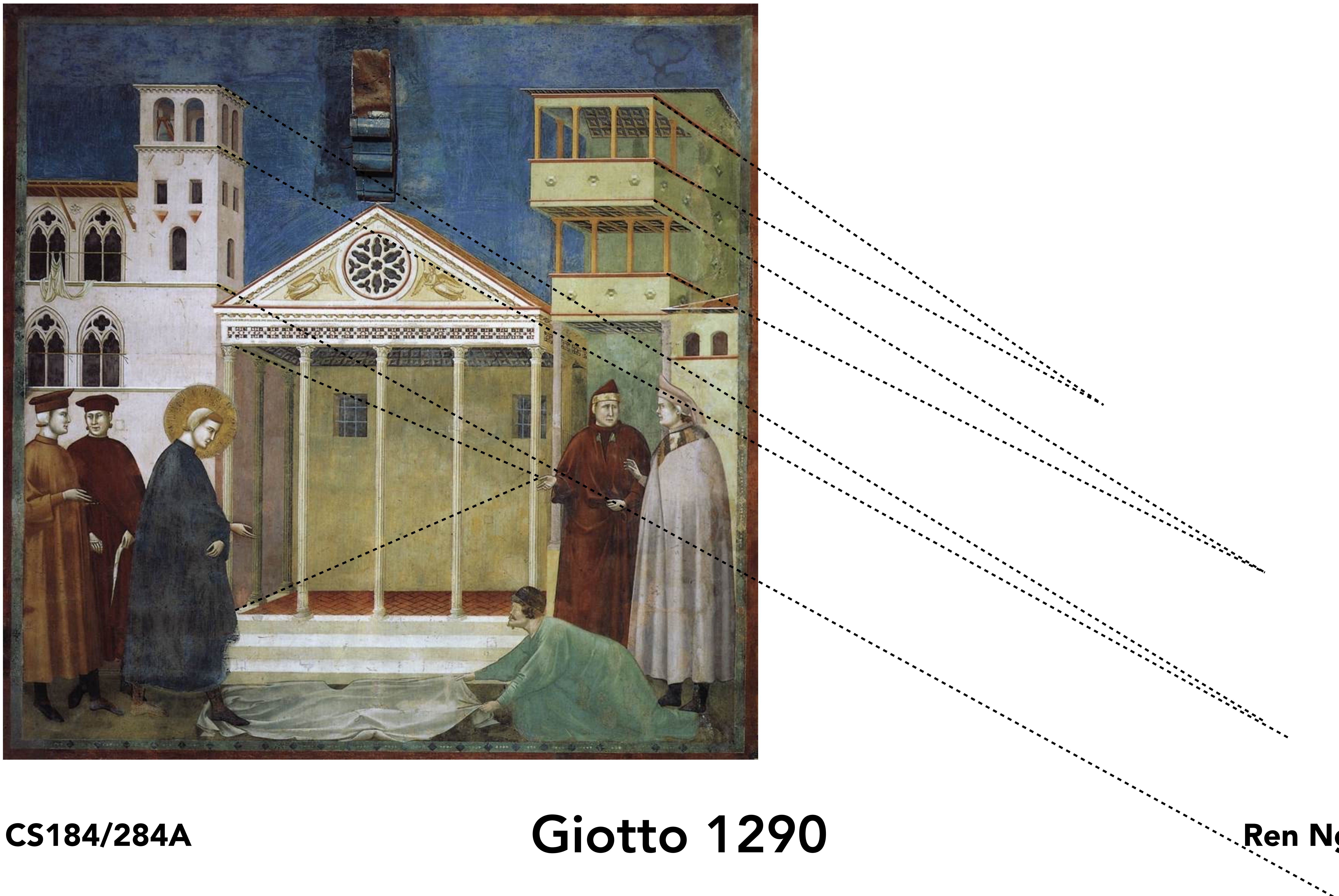
Berlinghieri 1235

Perspective in Art



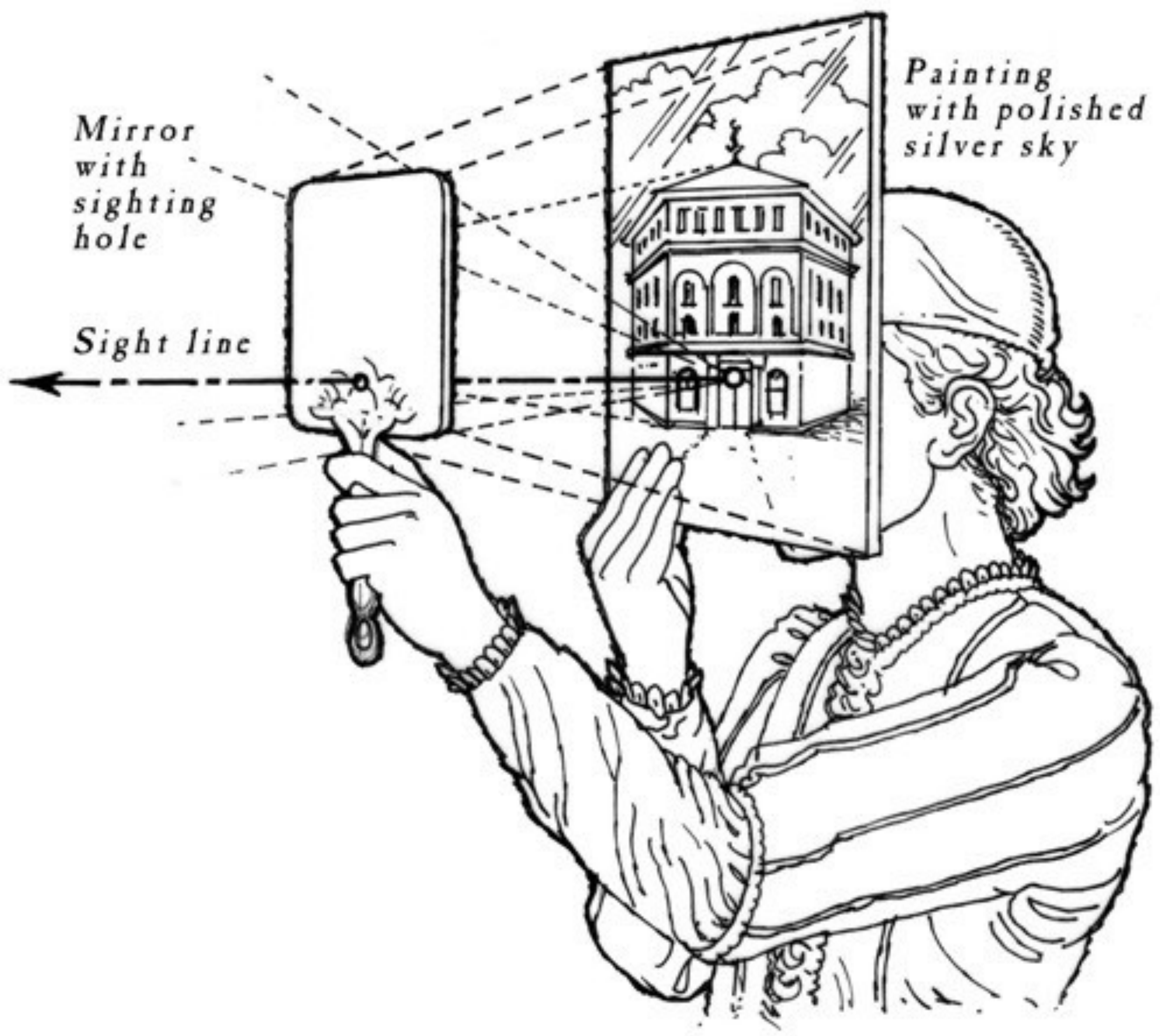
Giotto 1290

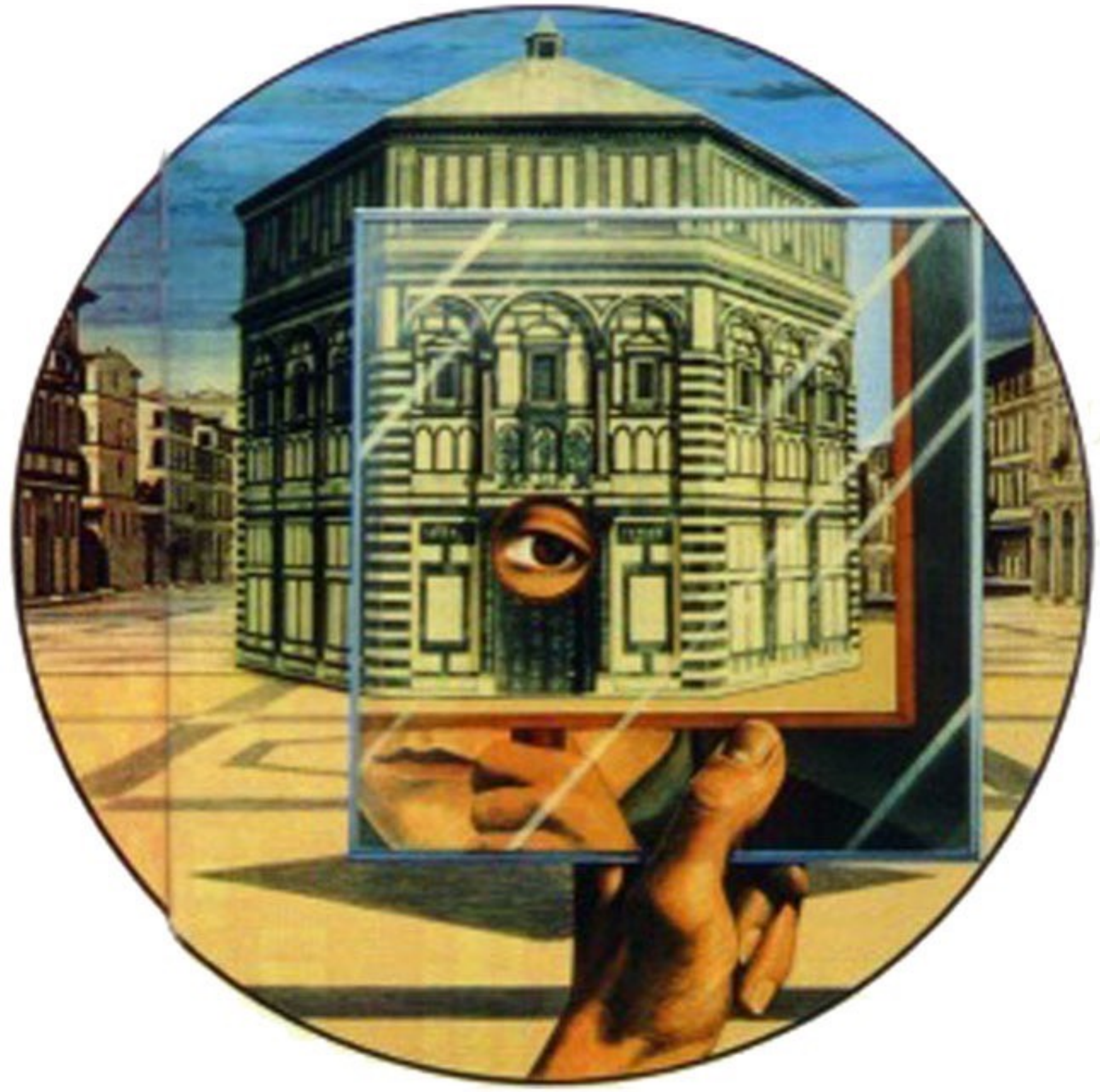
Perspective in Art



Giotto 1290

Perspective in Art





Brunelleschi experiment c. 1413

Florence Cathedral

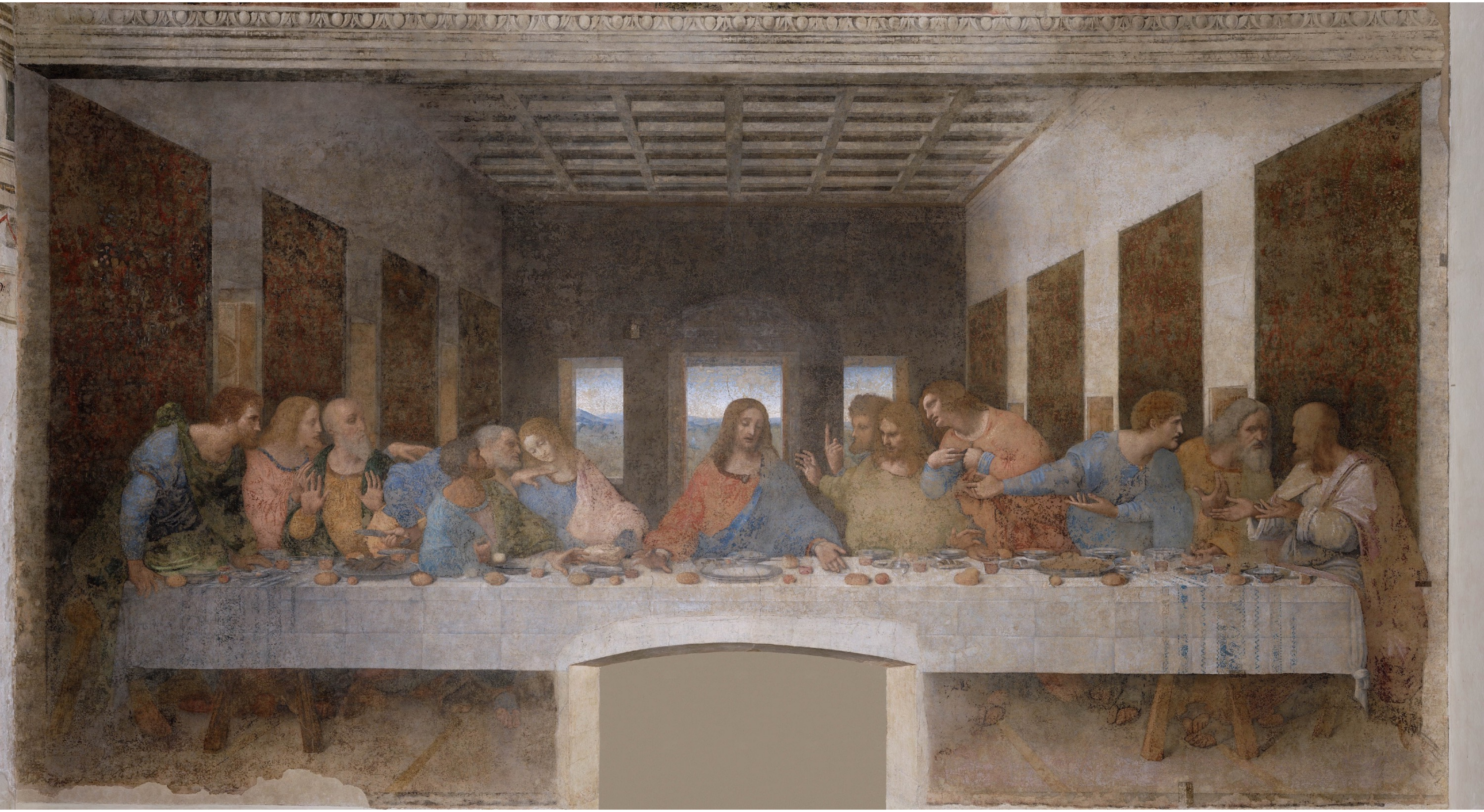
North Door (1403~) East Door (1425~)

Both works by Lorenzo Ghiberti

Perspective in Art

Delivery of the Keys (Sistine Chapel), Perugino, 1482

Perspective in Art



The Last Supper, Leonardo da Vinci, 1499

Pinhole Camera Model

Pinhole Camera

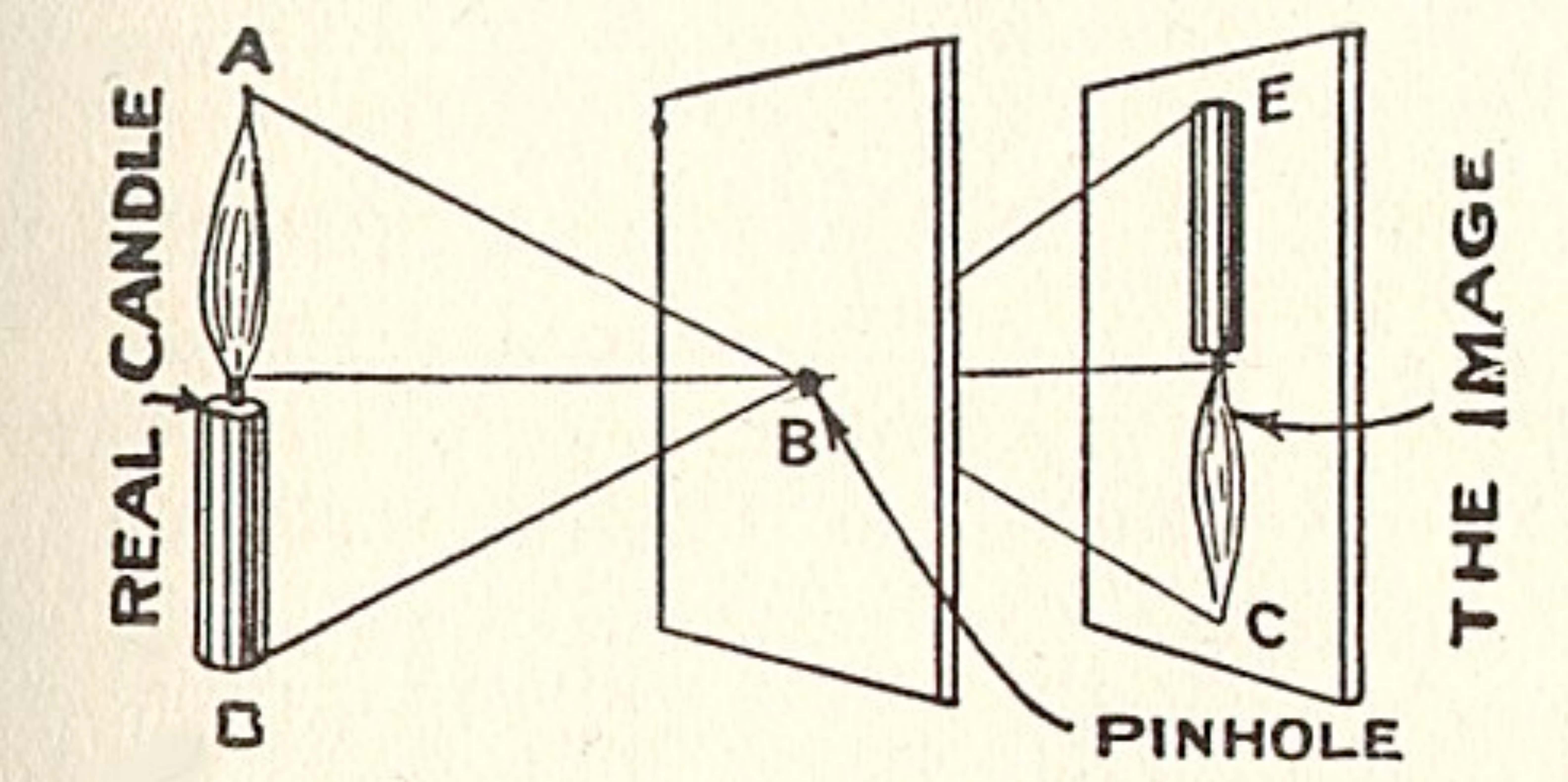
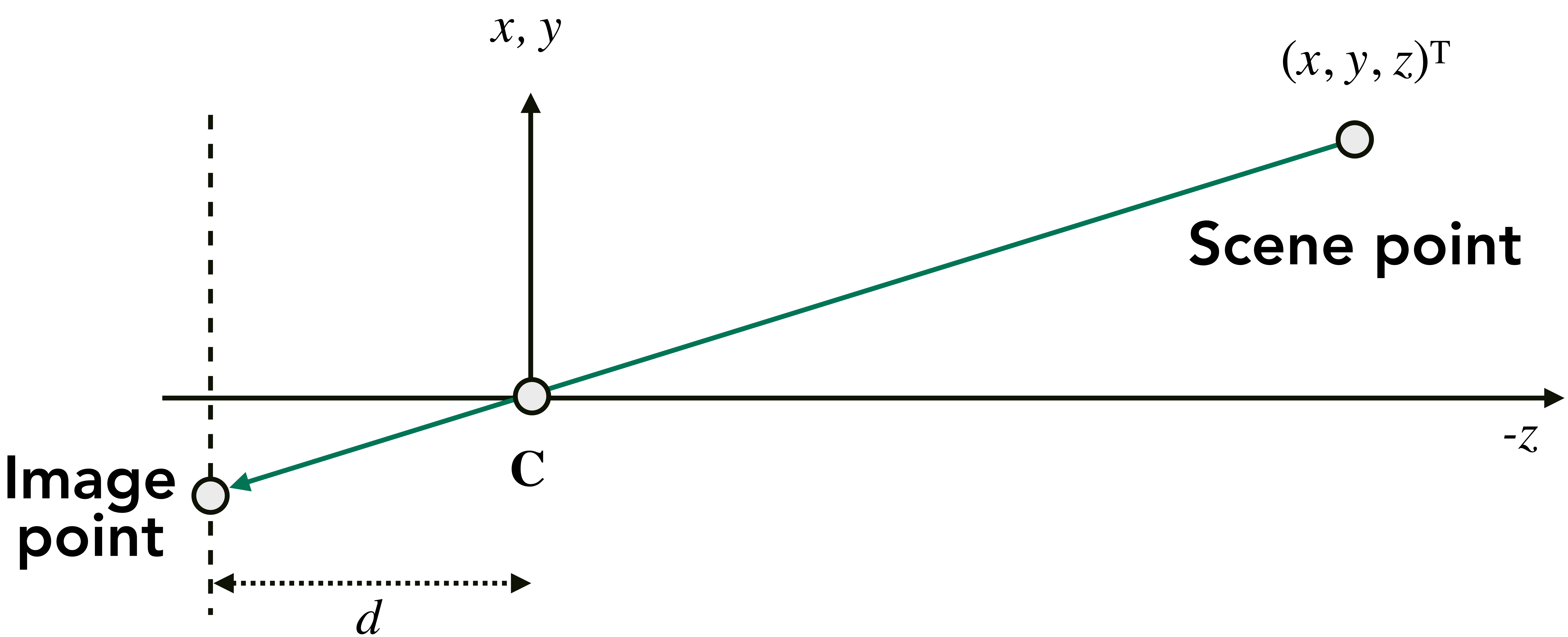


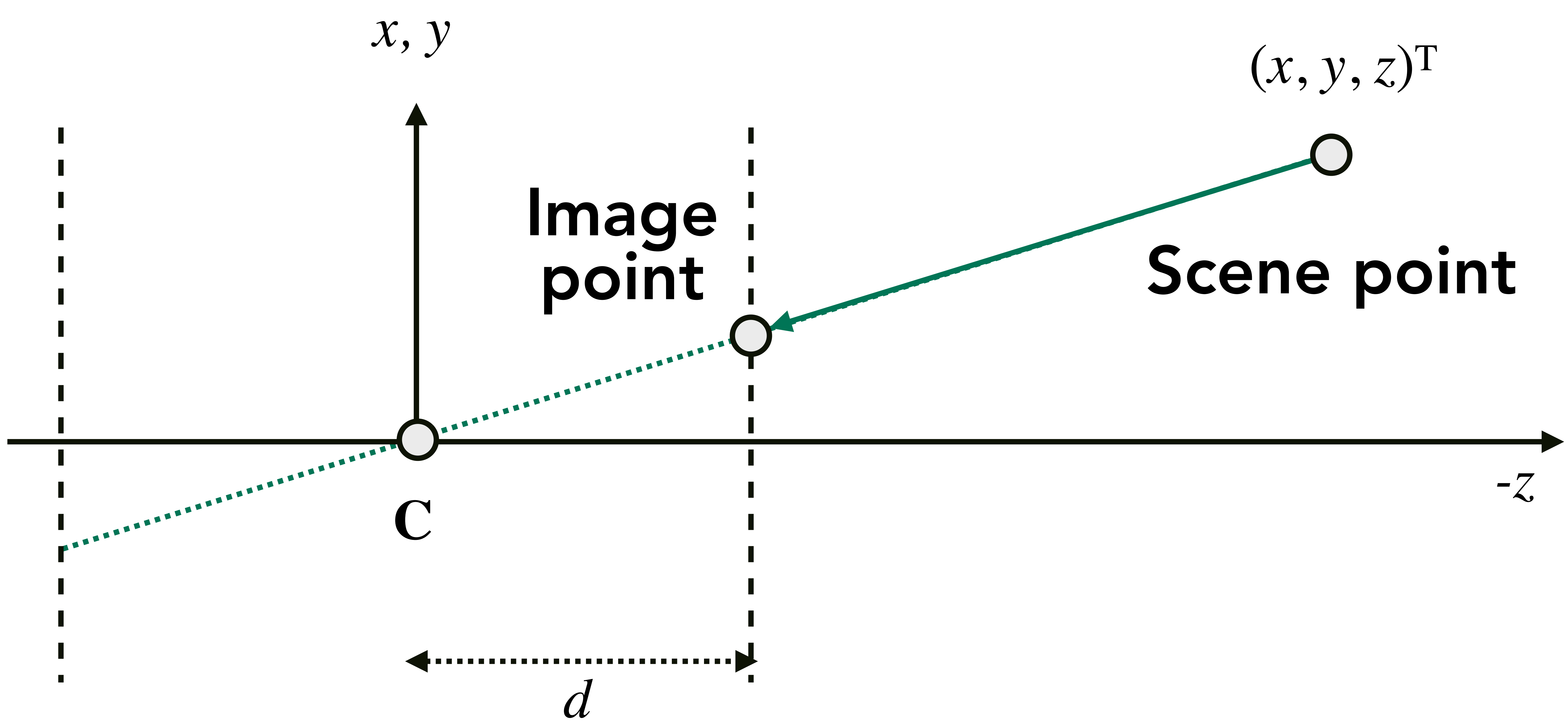
FIG. 131.—How Light and a Pinhole Form an Image.

Projective Transform



Inverted image (as in real pinhole camera)

Pinhole Camera Projective Transform

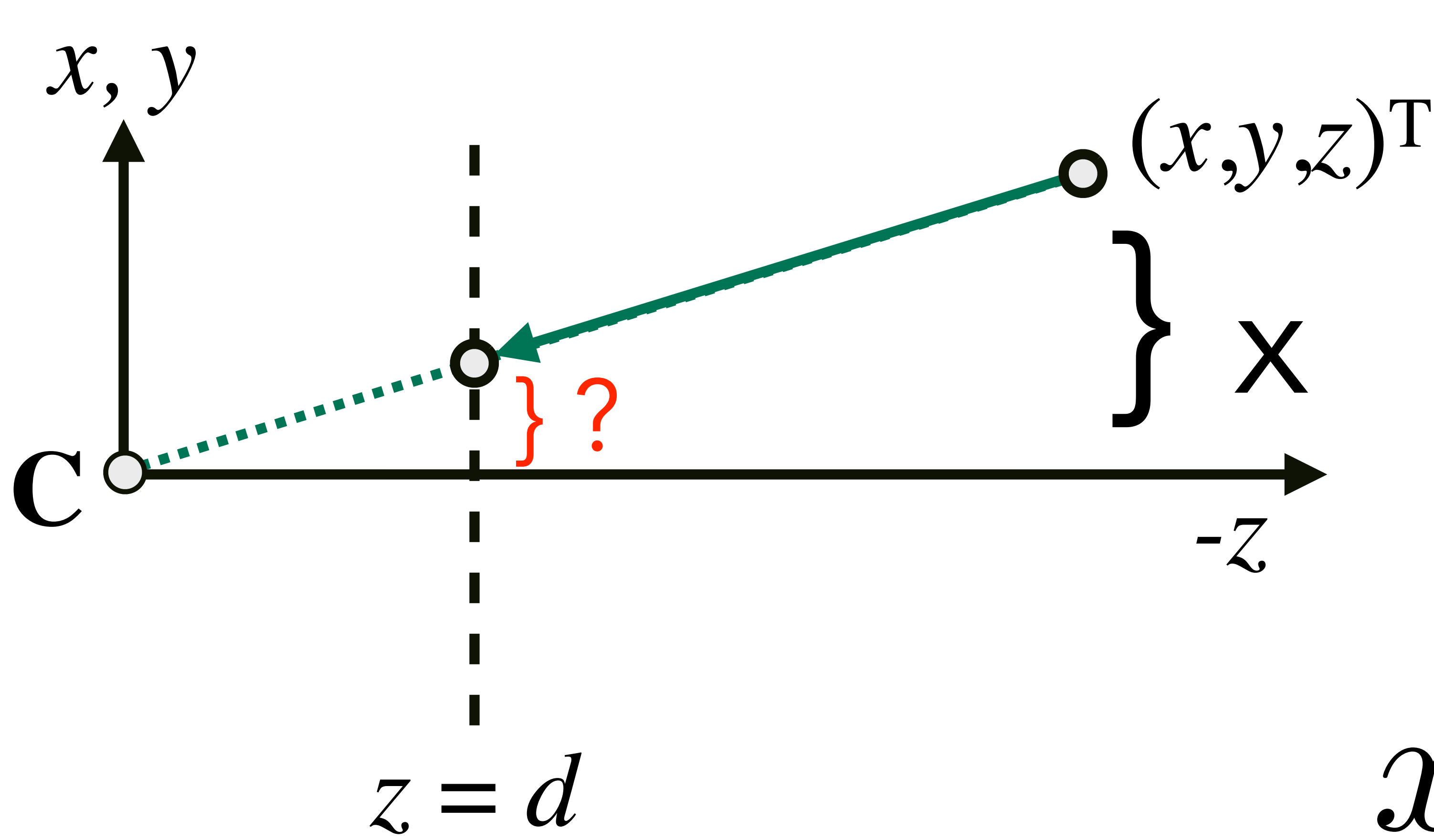


Upright image

Projective Transforms

Standard perspective projection

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



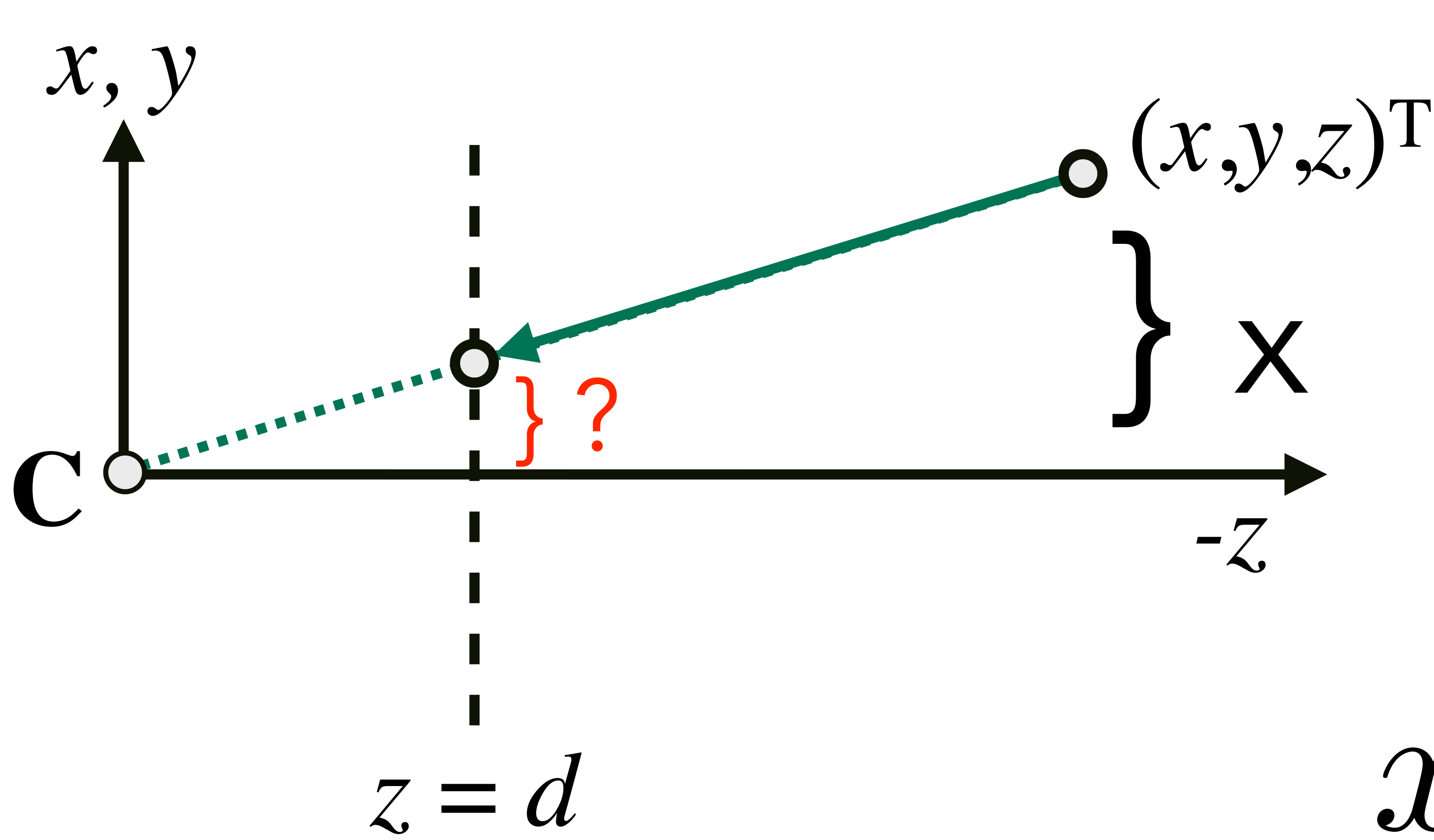
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \\ \\ d \end{pmatrix}$$

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

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 $z = d$

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

Perspective foreshortening

- Need division by z
- Matrix representation?

➡Homogeneous coordinates!

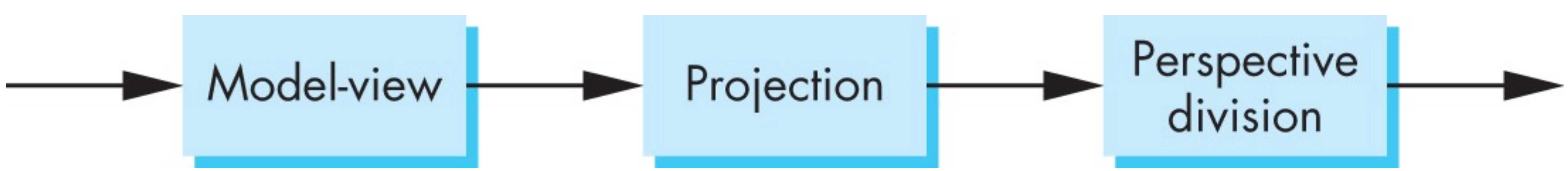
Homogenous Coordinates (3D)

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

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

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

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

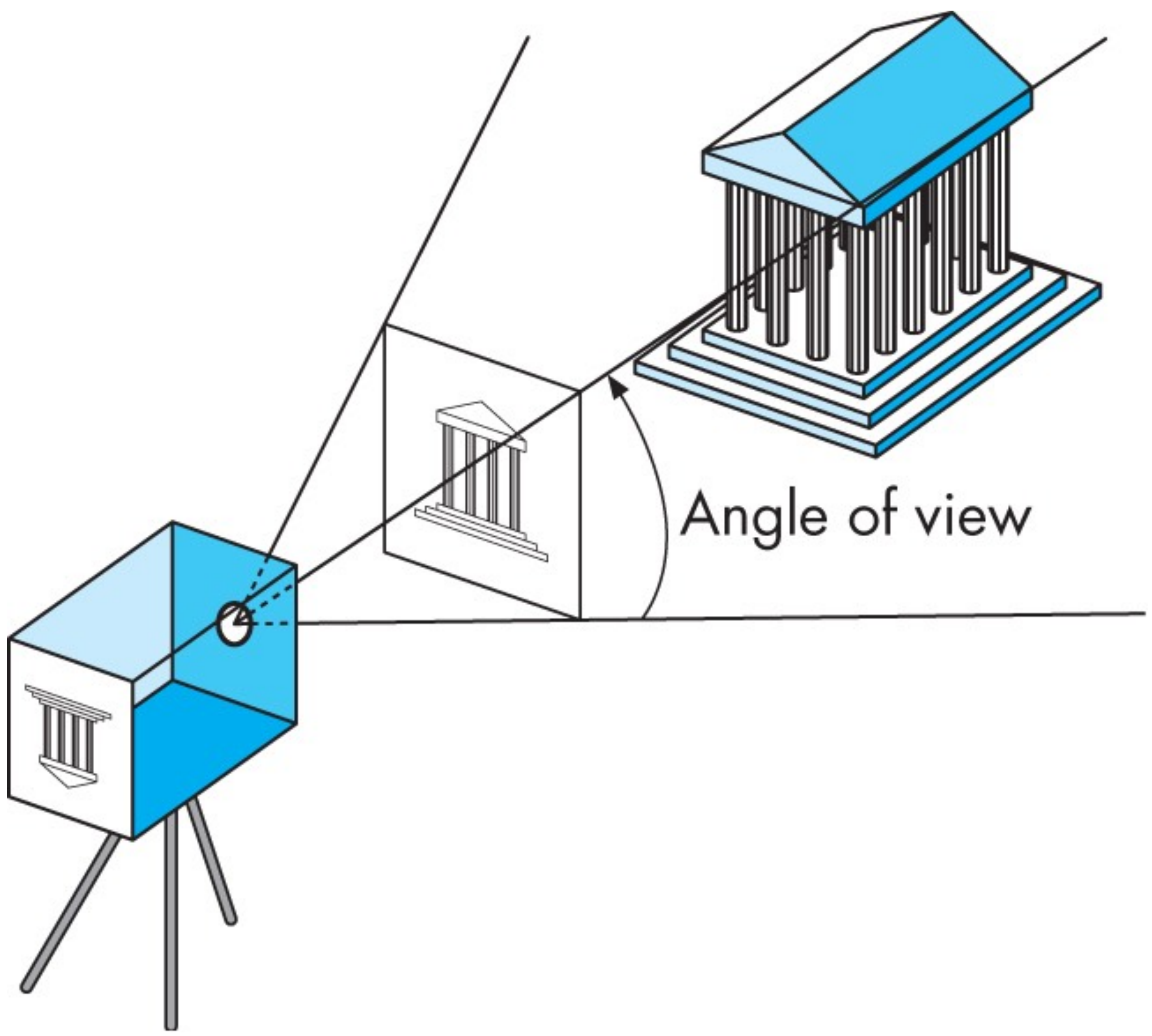


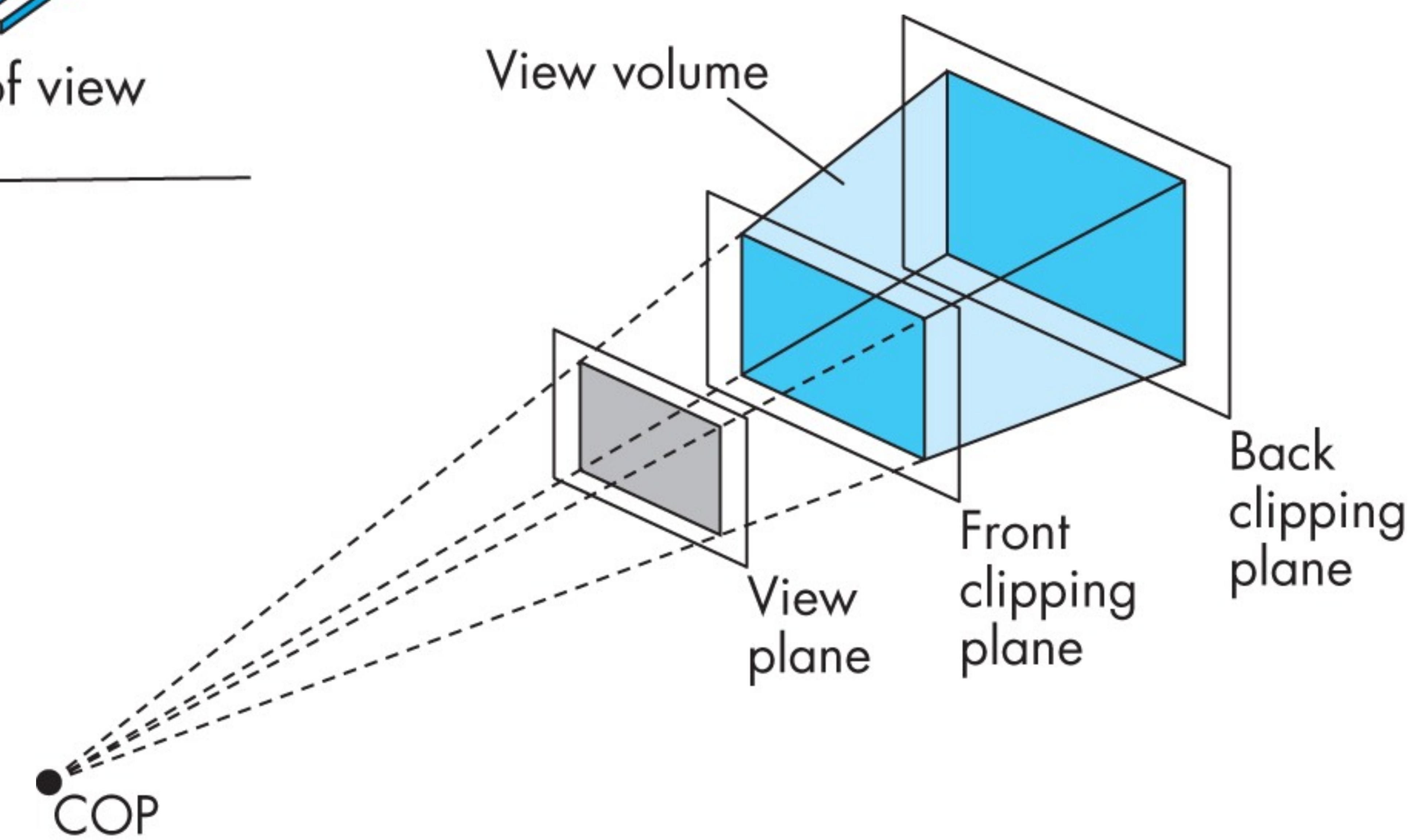
Pinhole Camera Model

This mathematical model produces all linear perspective effects!

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

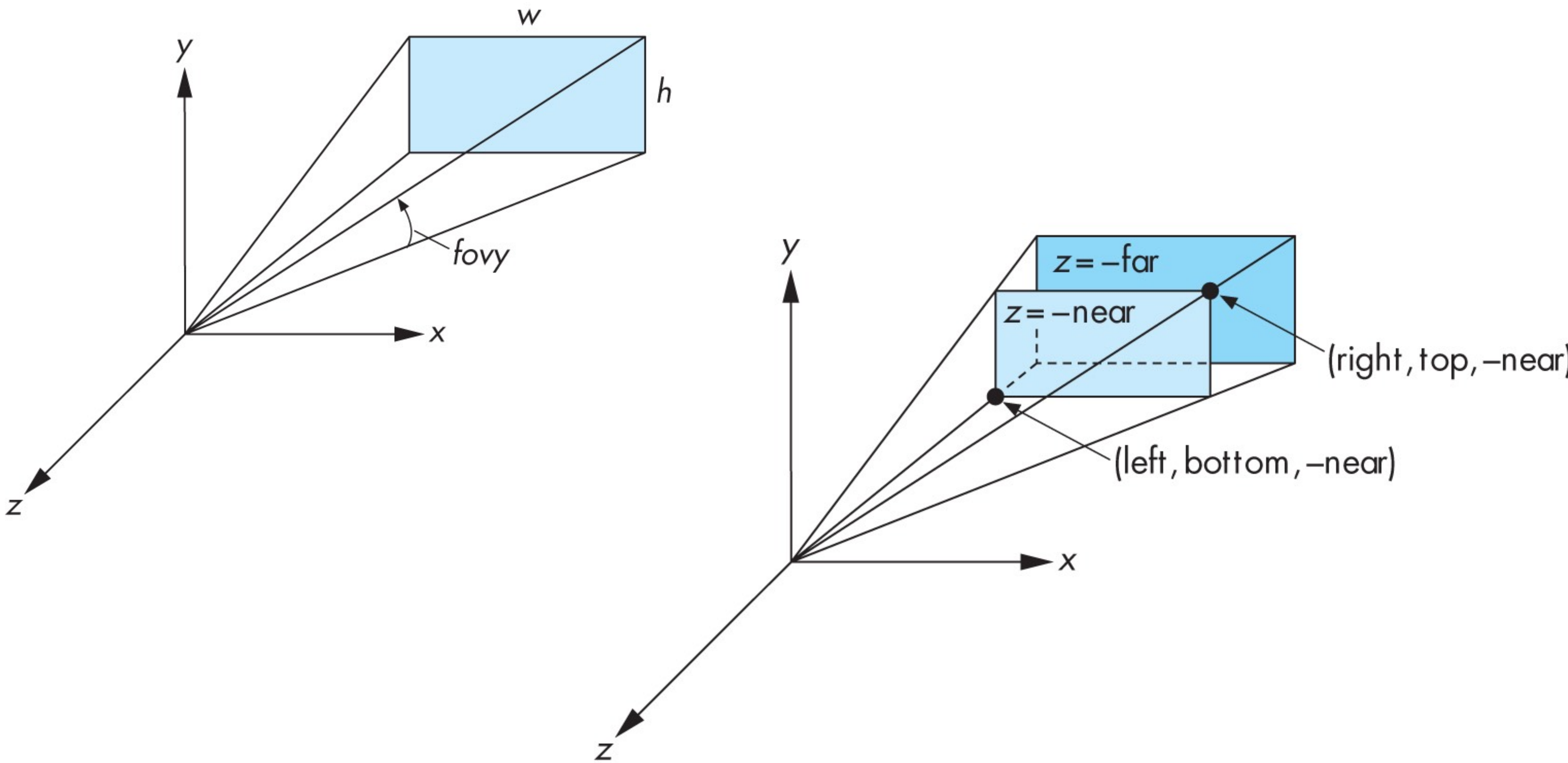
Specifying Perspective Projection





From Angel and Shreiner, Interactive Computer Graphics

Specifying Perspective Viewing Volume



From Angel and Shreiner, Interactive Computer Graphics

Specifying Perspective Viewing Volume

Parameterized by

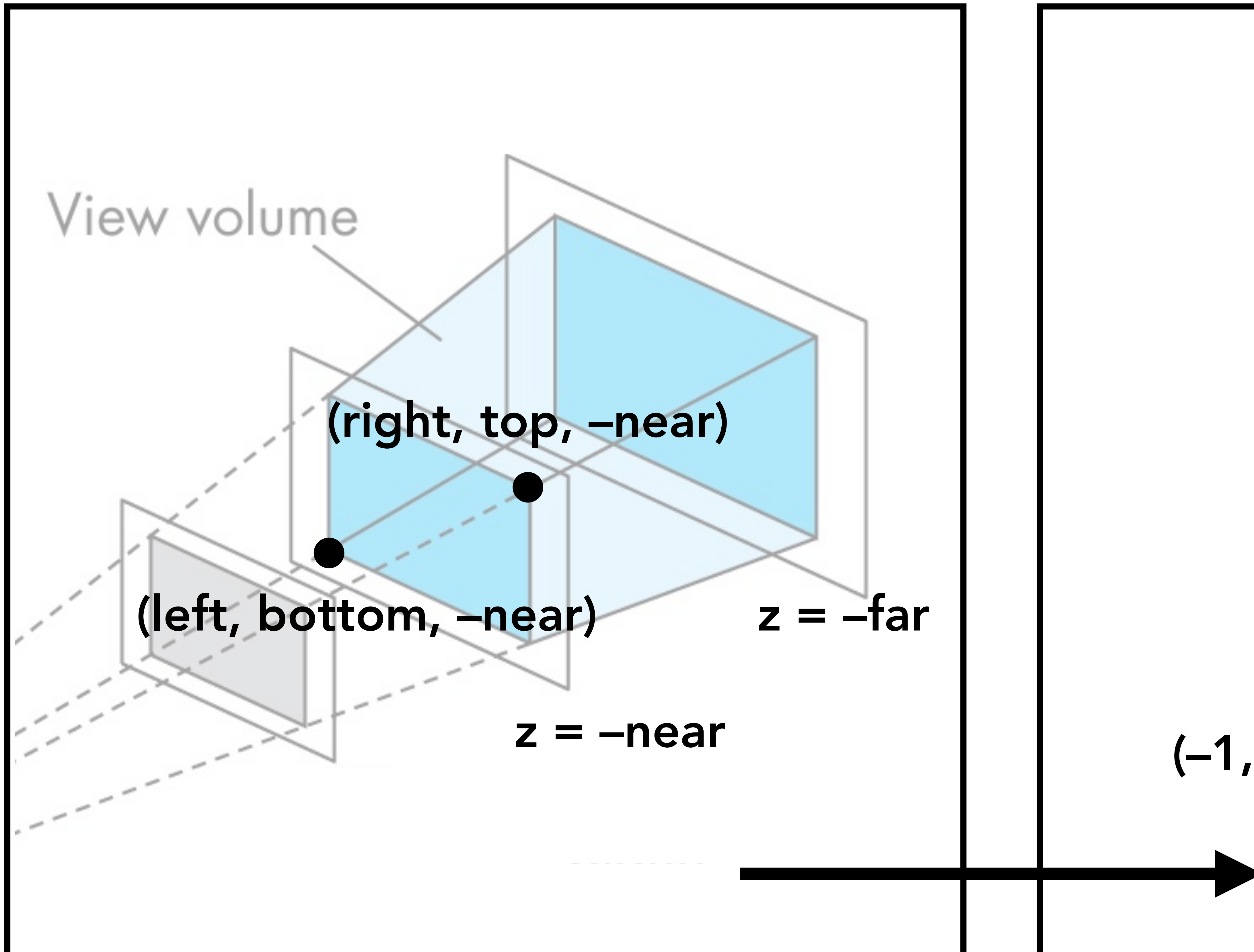
- fovy : vertical angular field of view
- aspect ratio : width / height of field of view
- near : depth of near clipping plane
- far : depth of far clipping plane

Derived quantities

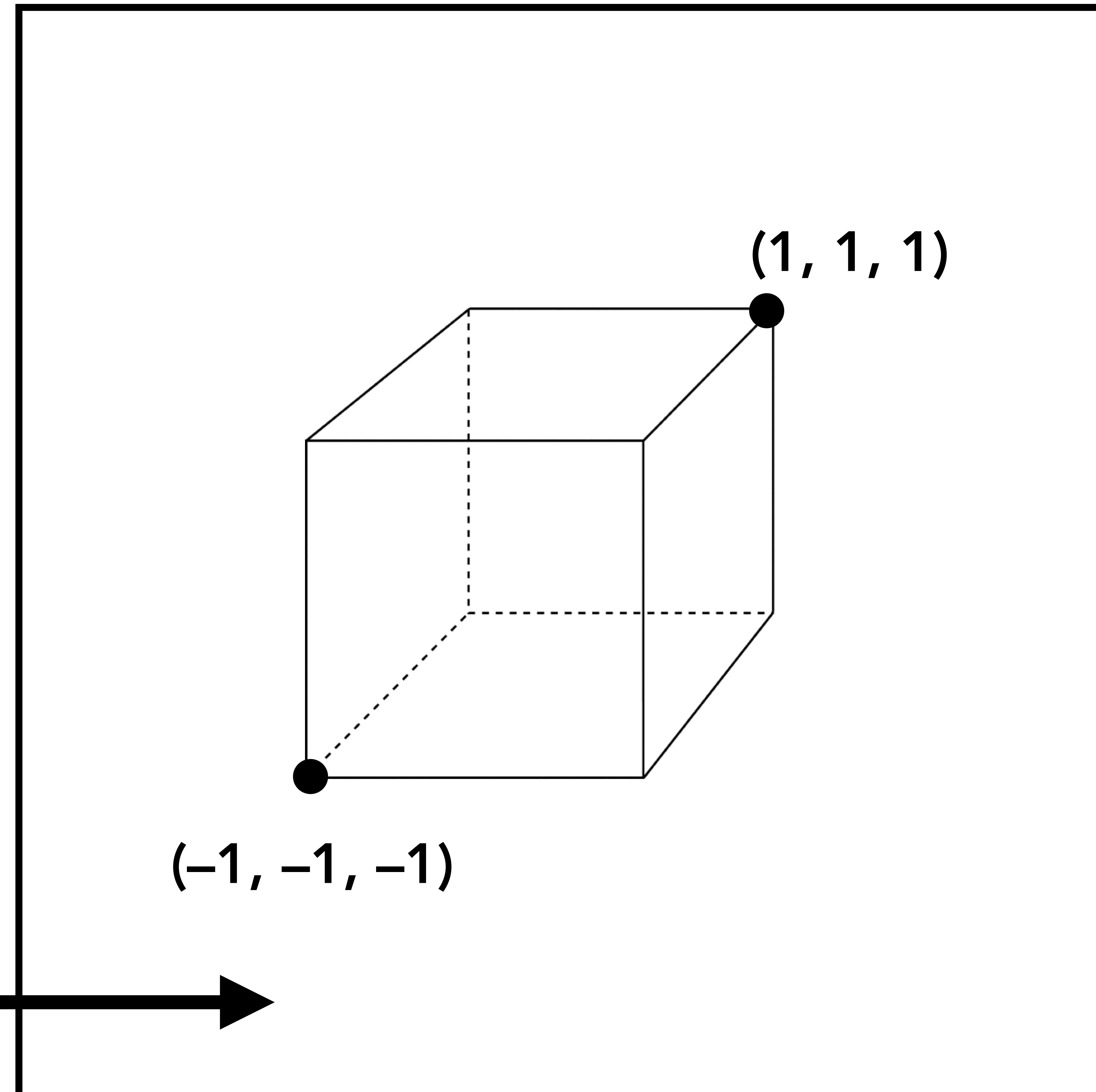
- top = near * tan (fovy)
- bottom = – top
- right = top * aspect
- left = – right

Perspective Projection Implementation

Perspective Projection Transform



Camera Coordinates



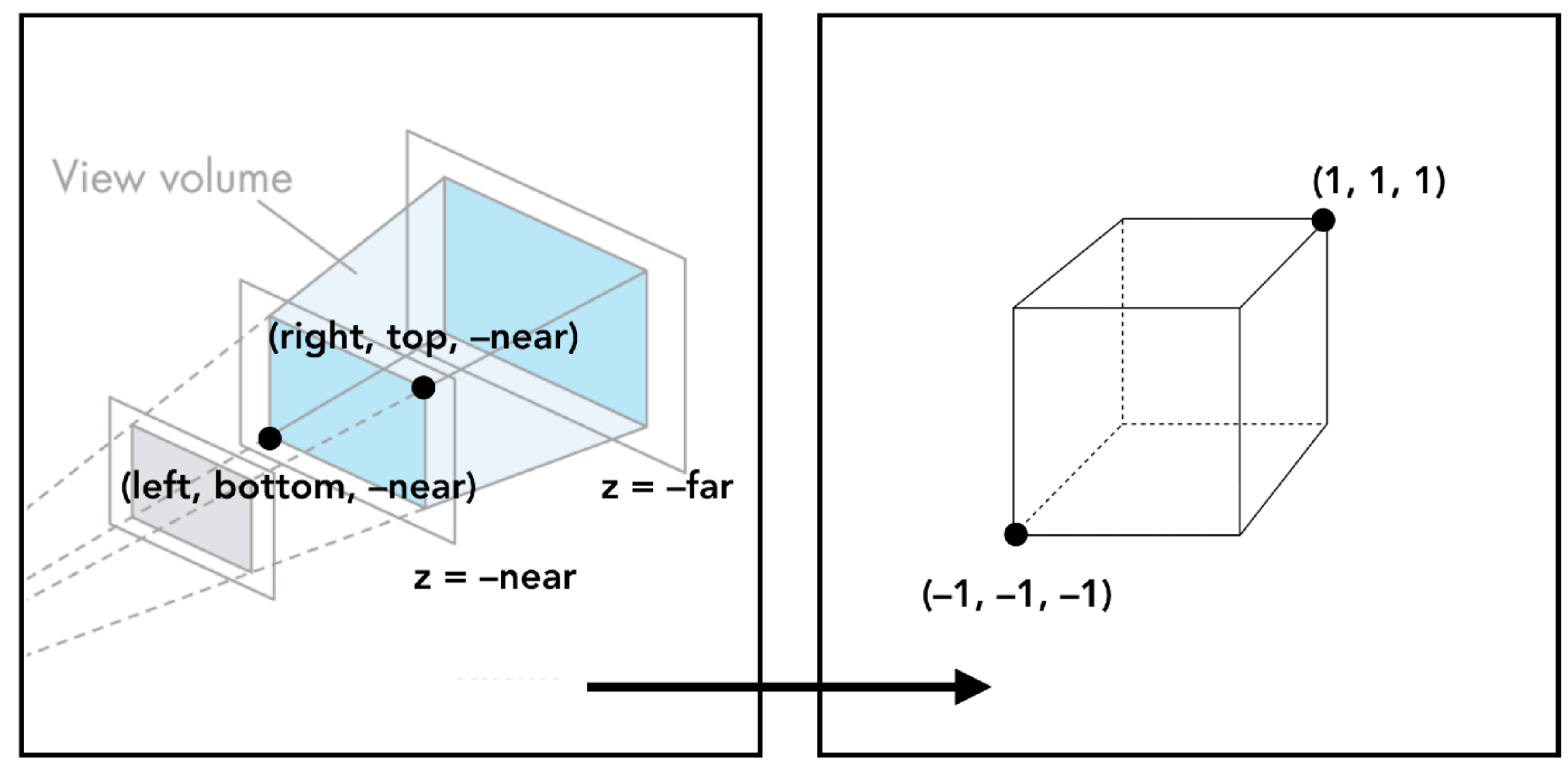
Normalized Device Coords
"NDC"

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



Camera Coordinates

Normalized Device Coords "NDC"

Perspective Transform Matrix

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

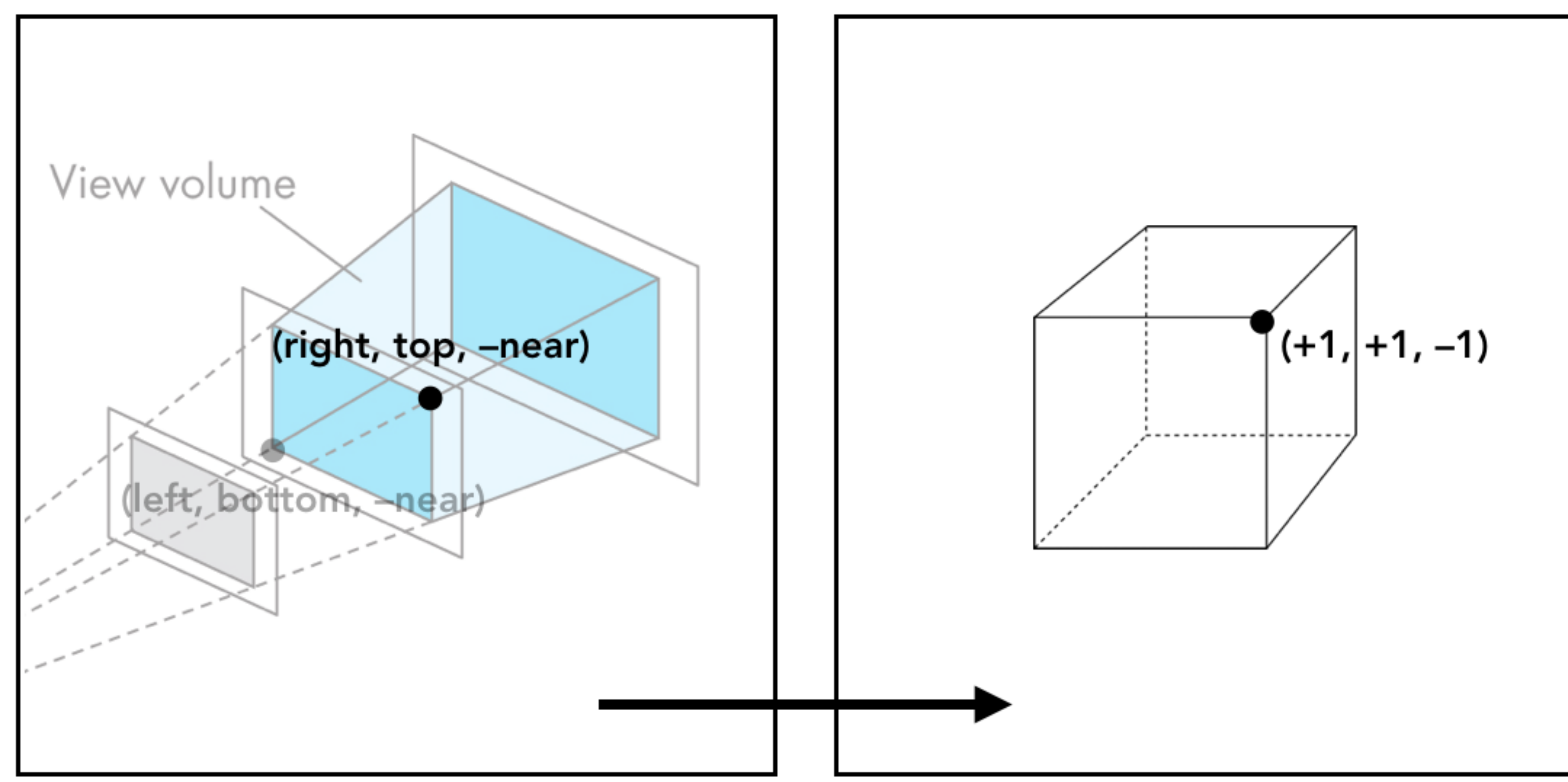
Perspective Transform Matrix Example

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

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

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

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



Camera Coordinates

Normalized Device Coords "NDC"

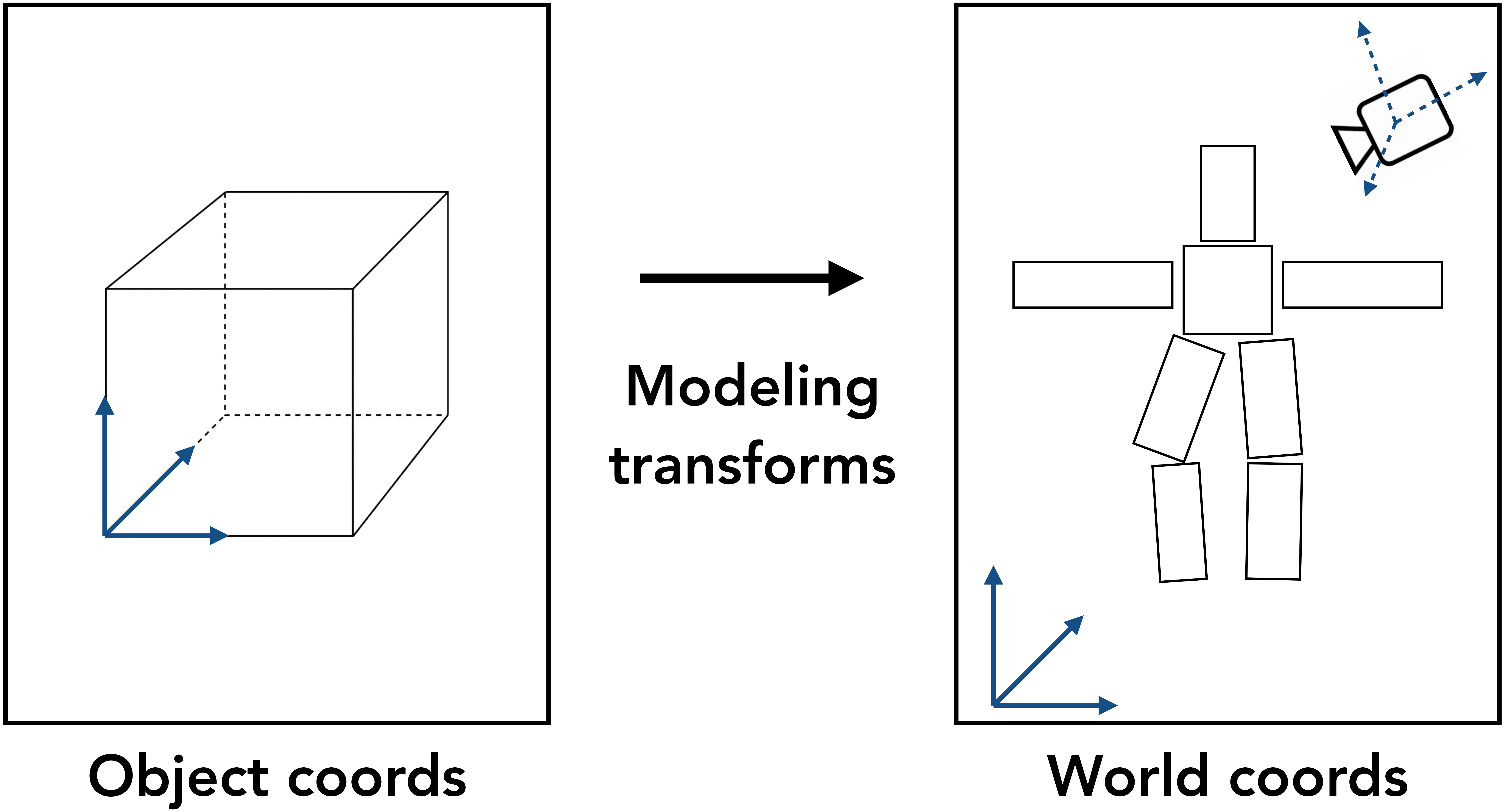
Transforms Recap

Transforms Recap

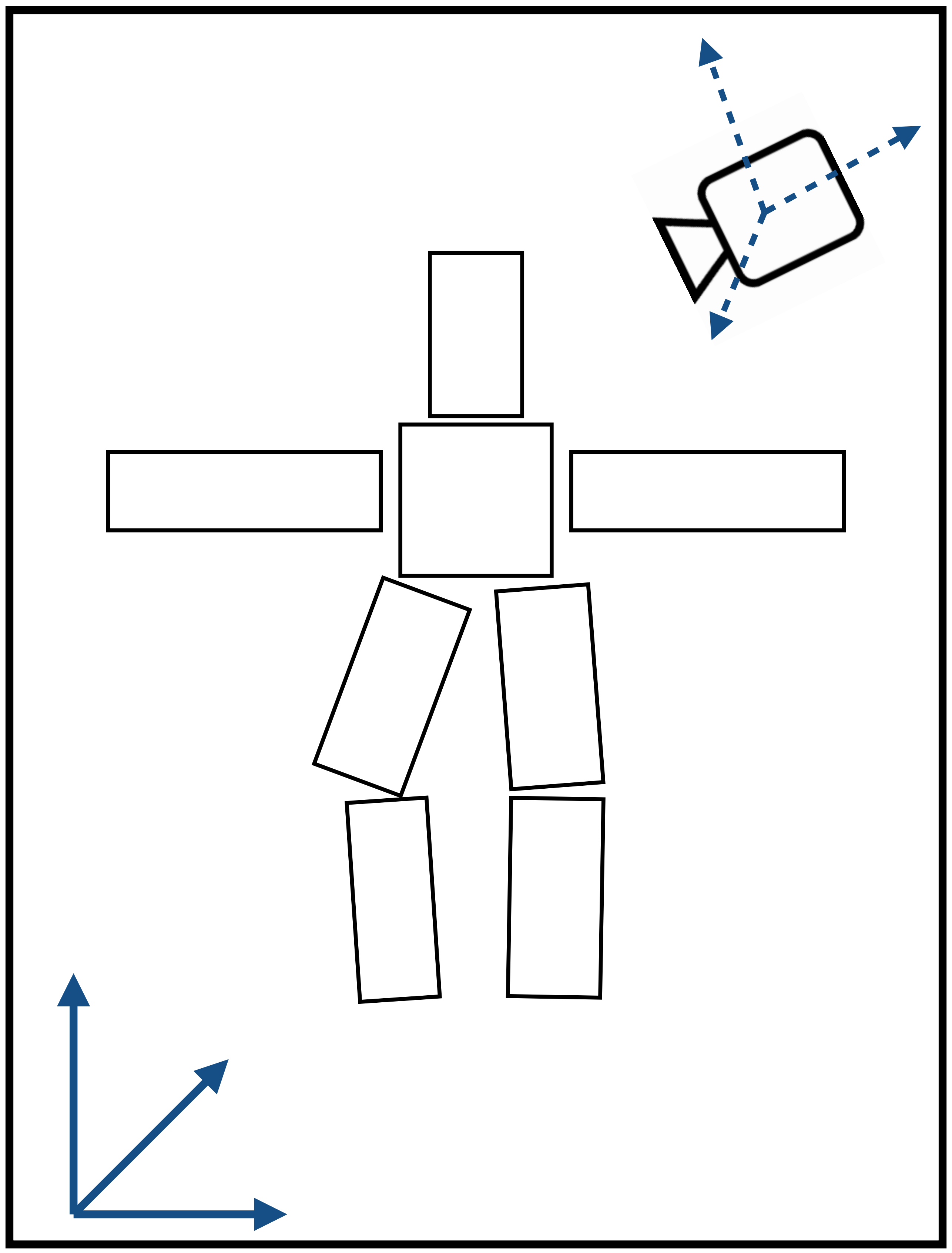
Coordinate Systems

- 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

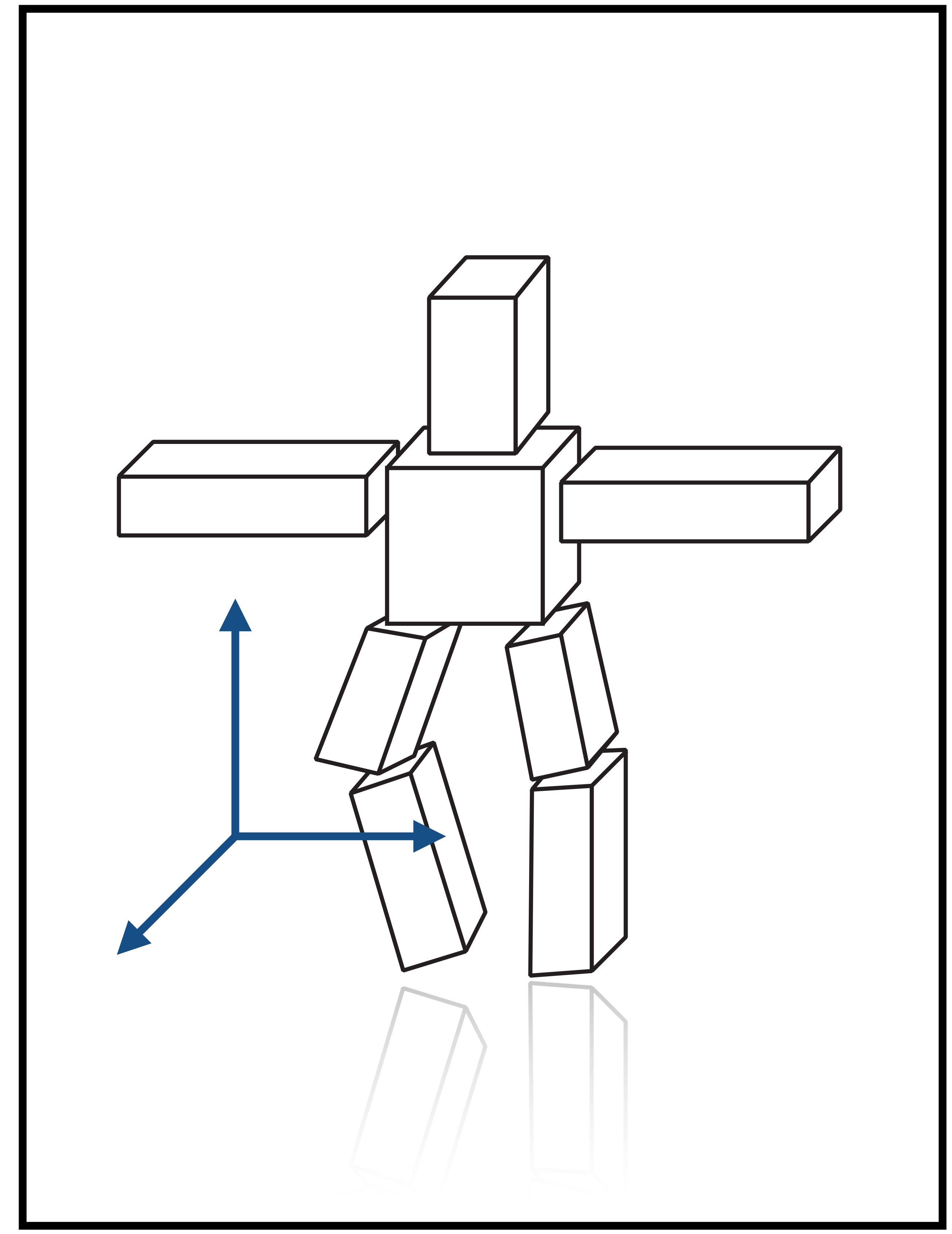


Transforms Recap



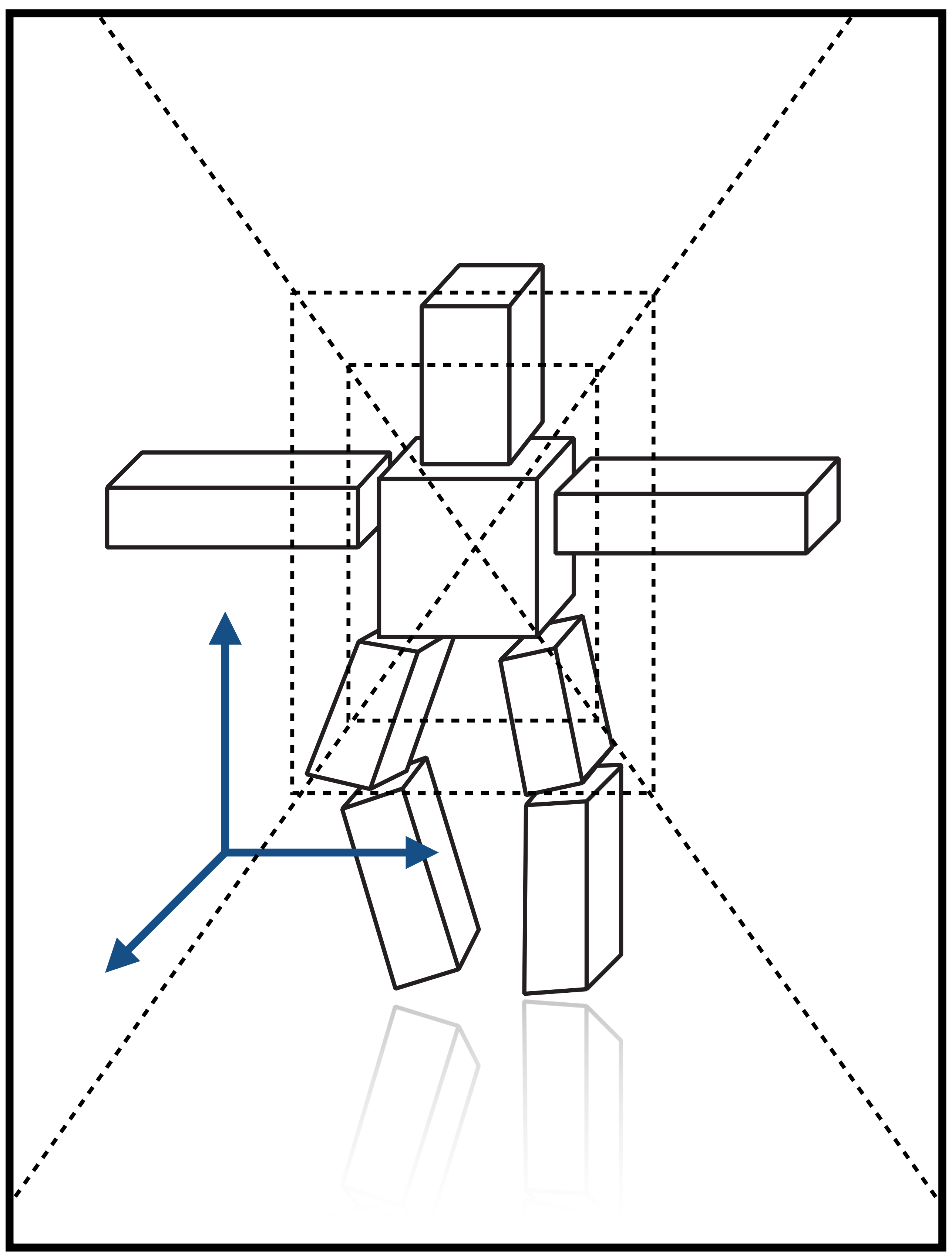
World coords

Viewing transform



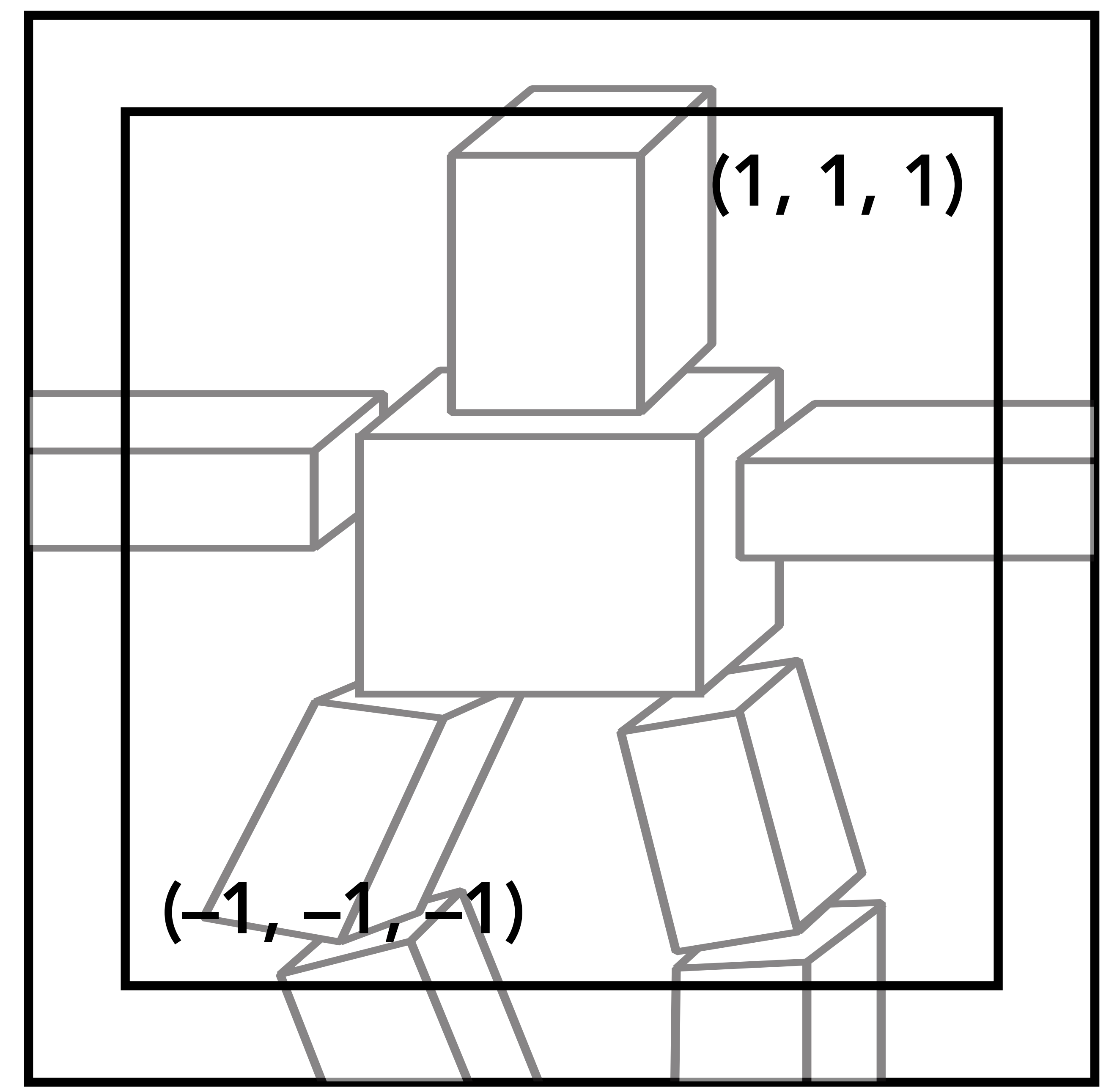
Camera coords

Transforms Recap



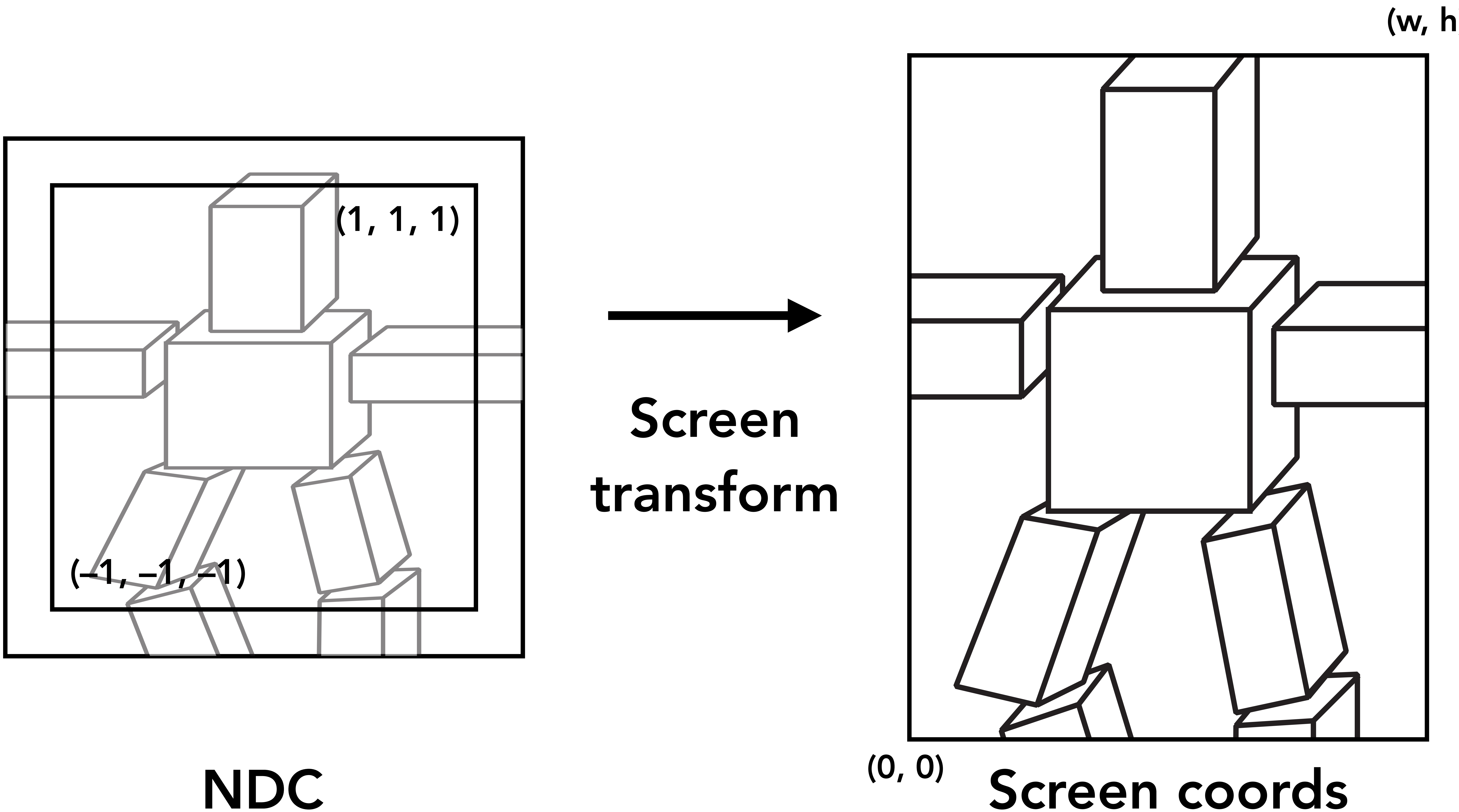
Camera coords

Perspective projection and homogeneous divide

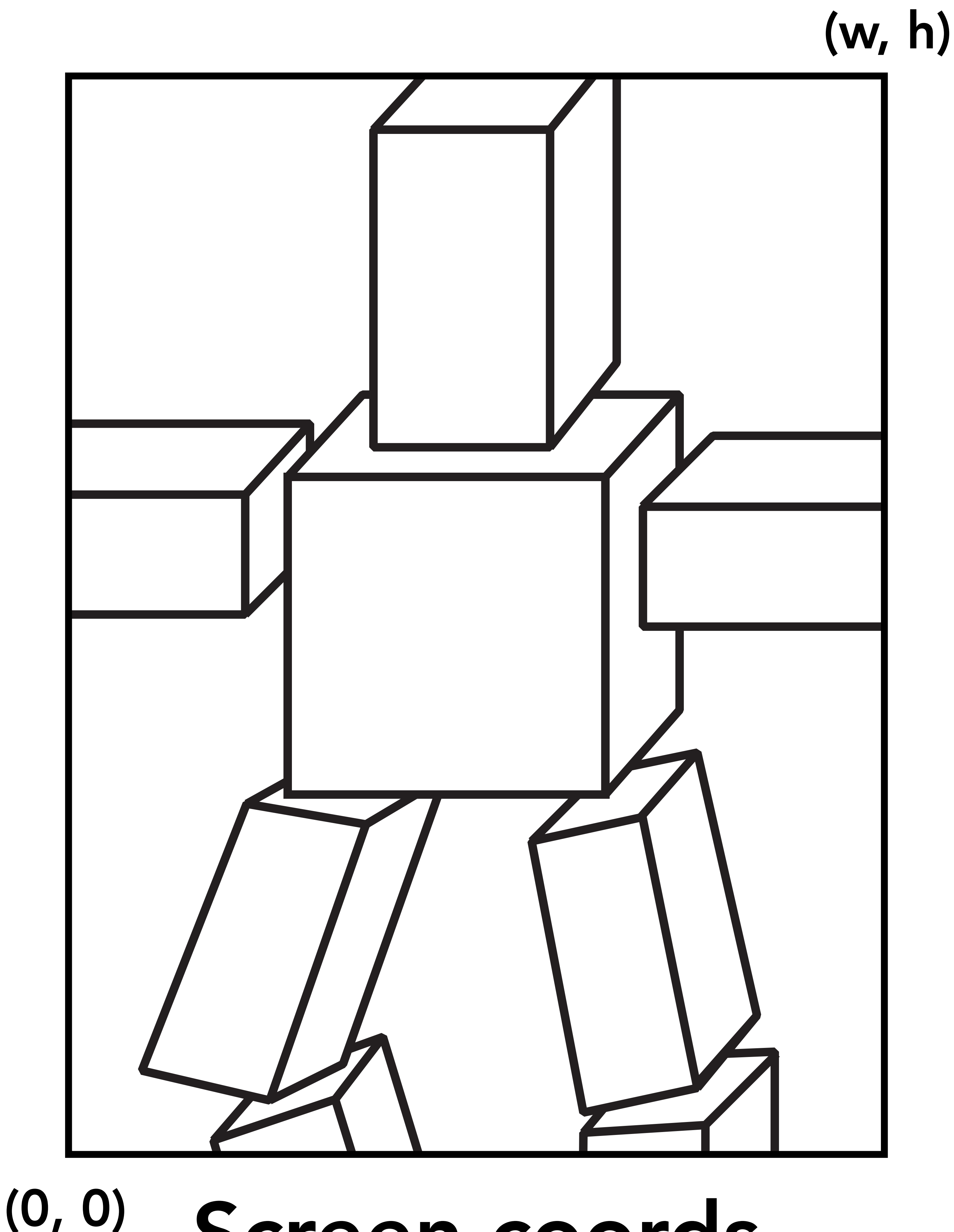


NDC

Transforms Recap

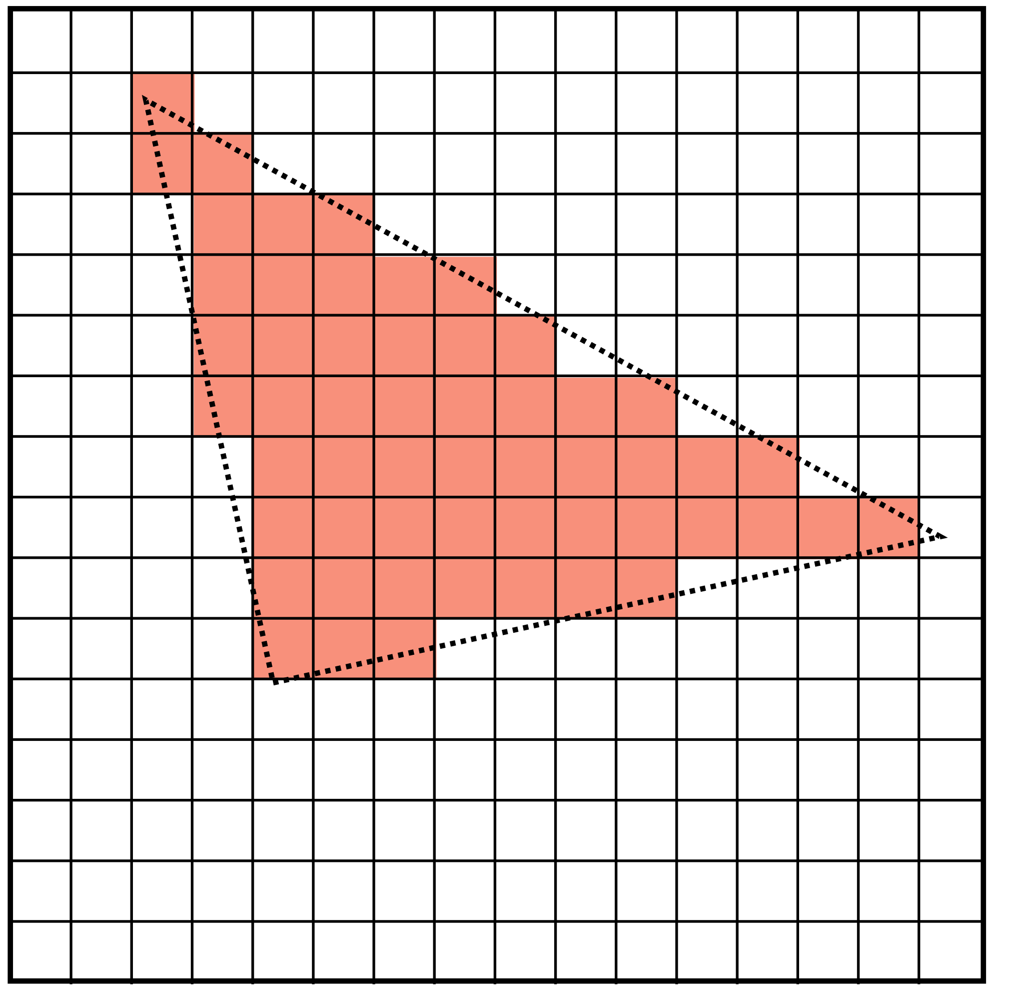


Transforms Recap



Screen coords

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

Thanks to Pat Hanrahan, Kayvon Fatahalian, Mark Pauly and Angjoo Kanazawa for presentation resources.