Lecture 4:

Transforms

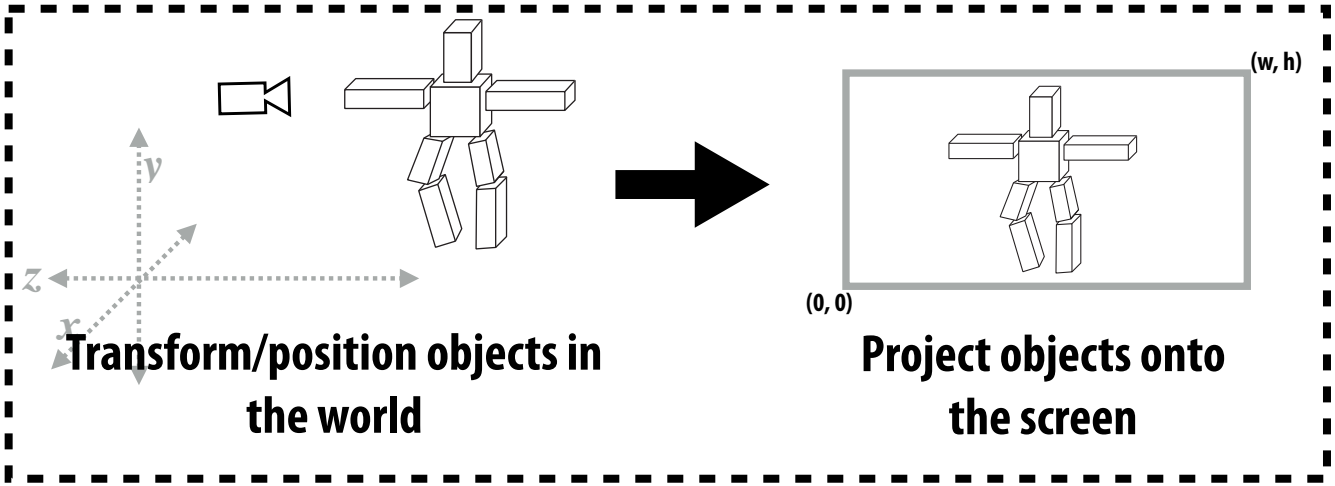
Computer Graphics 2025

Fuzhou University - Computer Science

Why Study Transforms?

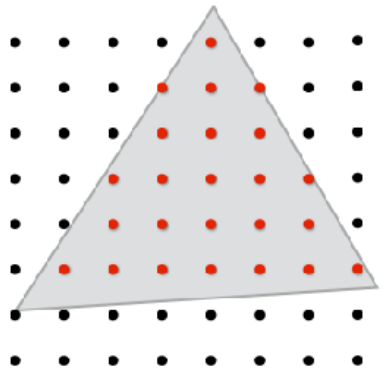
The Rasterization Pipeline

- Rough sketch of rasterization pipeline:

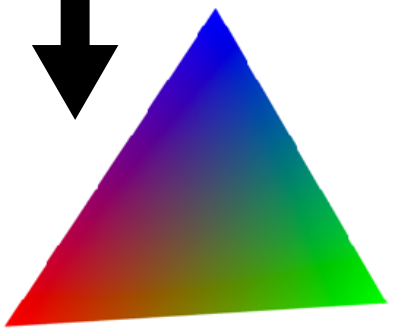


Transform/position objects in the world

Project objects onto the screen



Sample triangle coverage



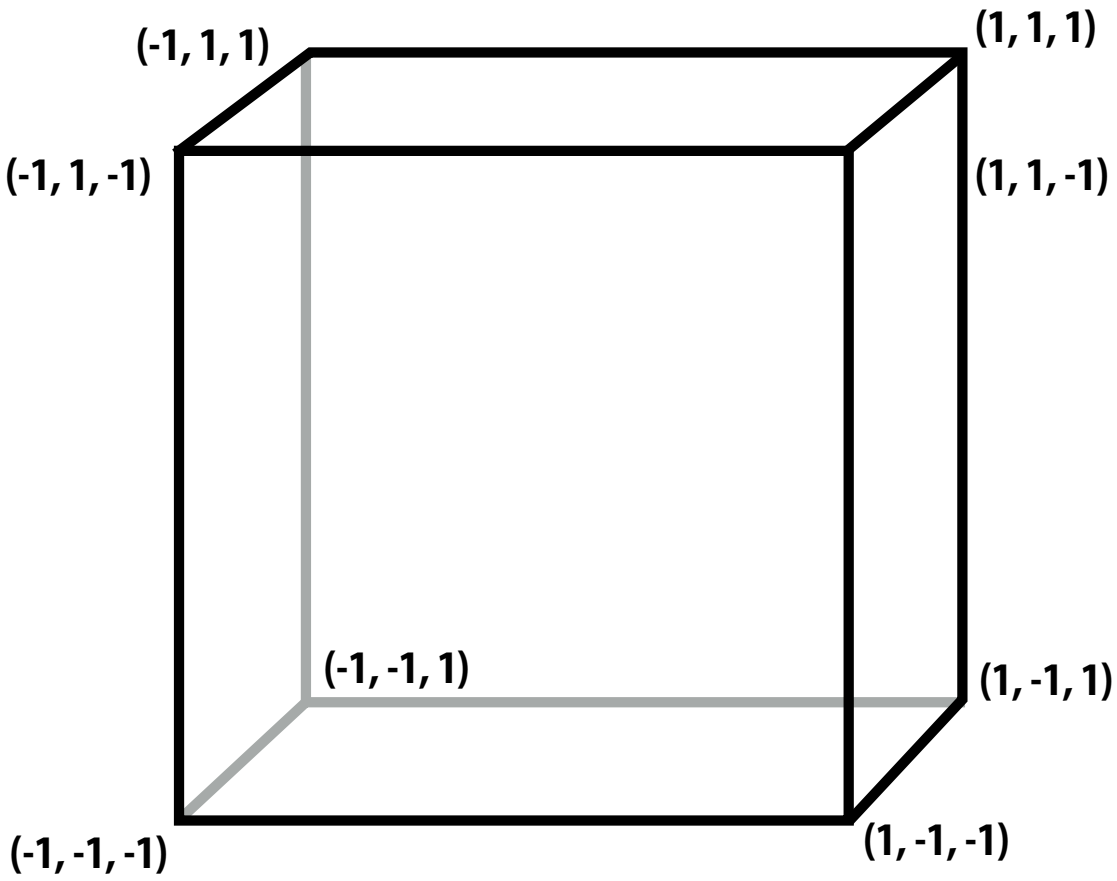
Interpolate triangle attributes at covered samples

Sample texture maps / evaluate shaders

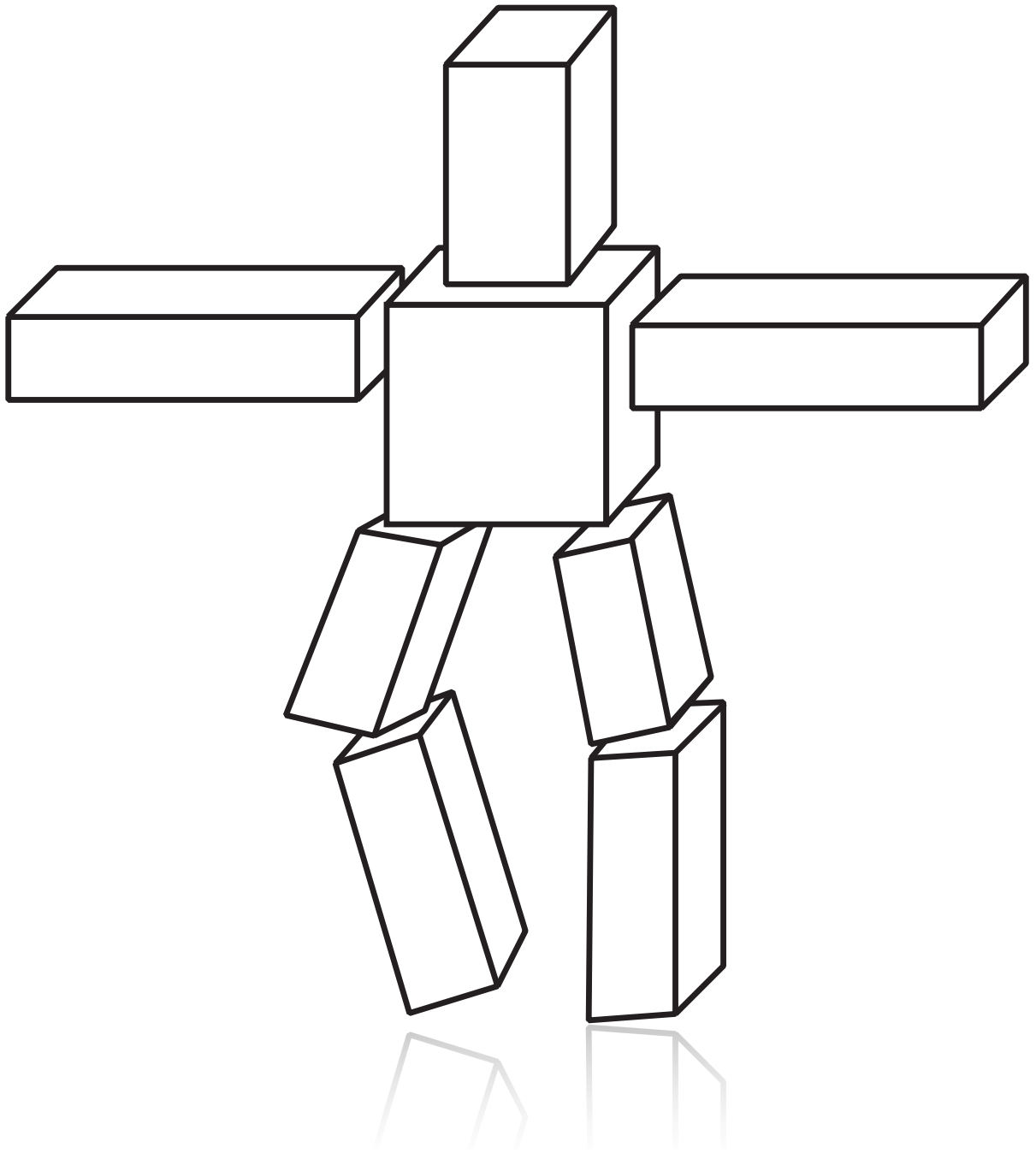
Combine samples into final image (depth, alpha, …)

- Reflects standard "real world" pipeline (OpenGL/Direct3D)

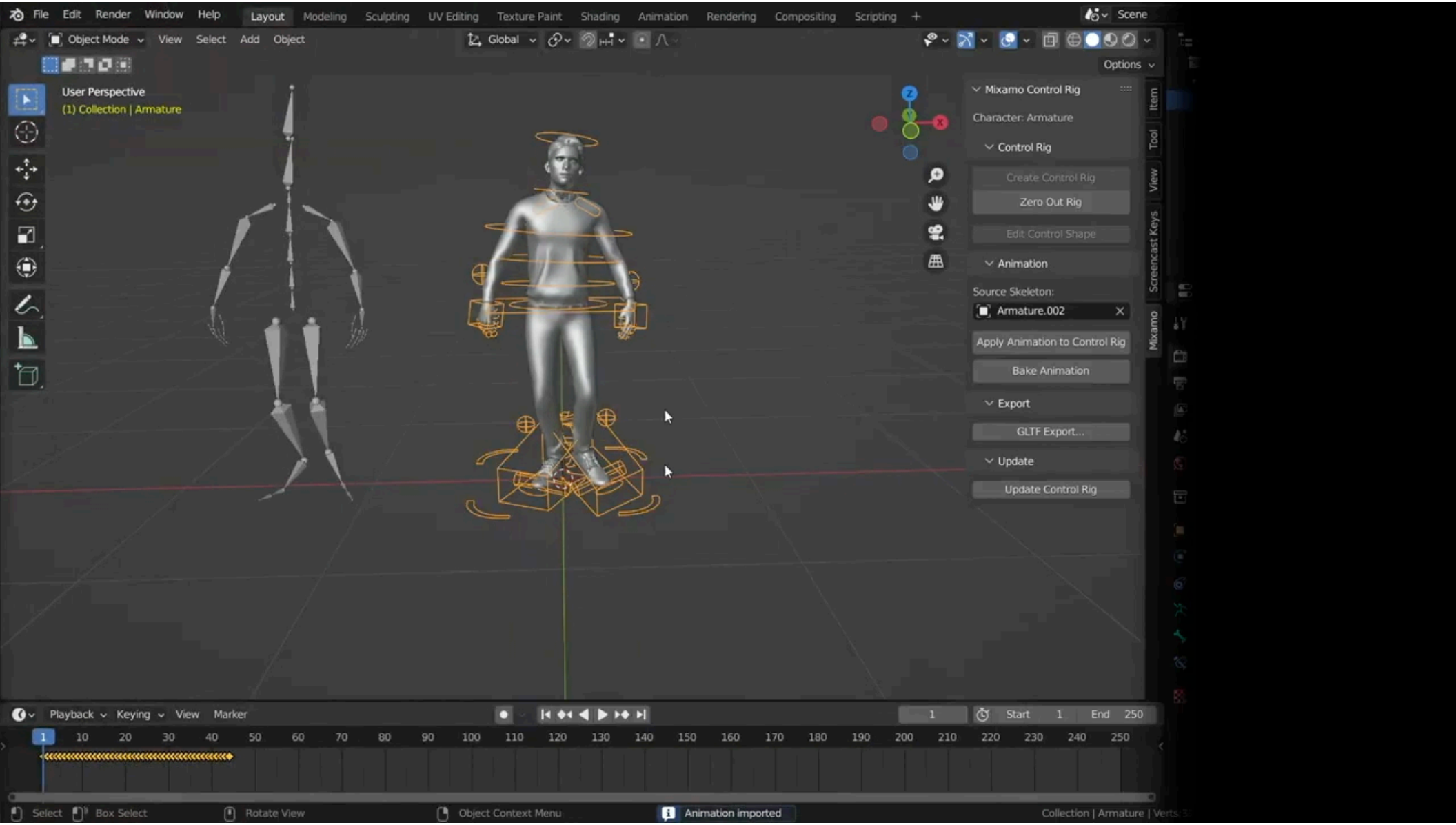
Cube



Cube Man

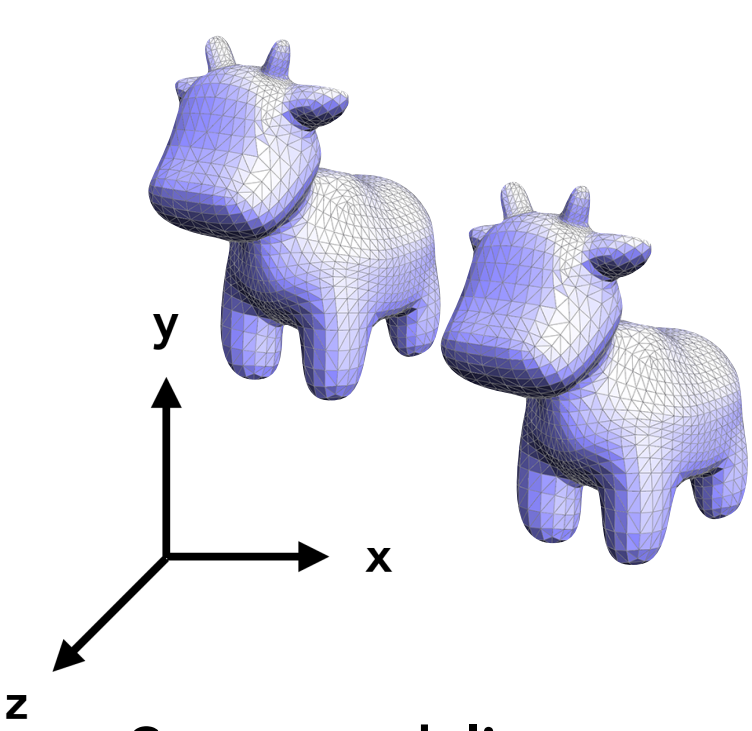


Modeling Objects

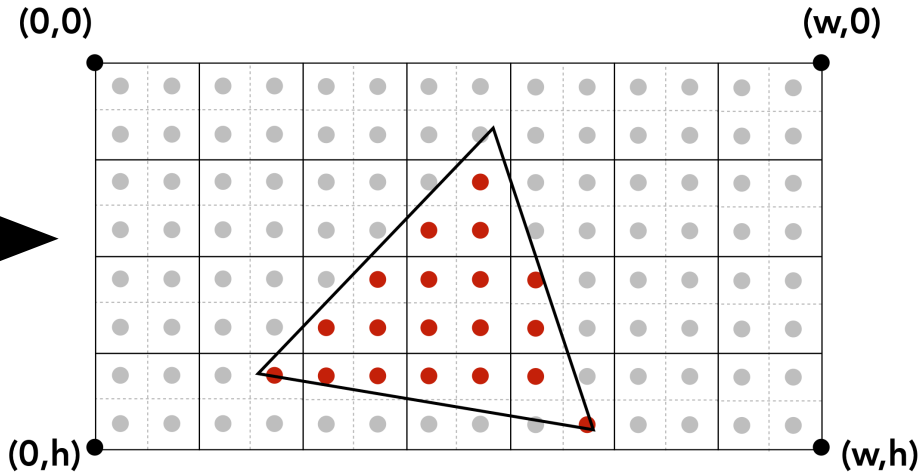


Transforms can describe positions and movements of body parts.

Viewing Objects



?



Scene modeling
3D world coordinates

Rasterization
2D screen coordinates

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

Today's Topics

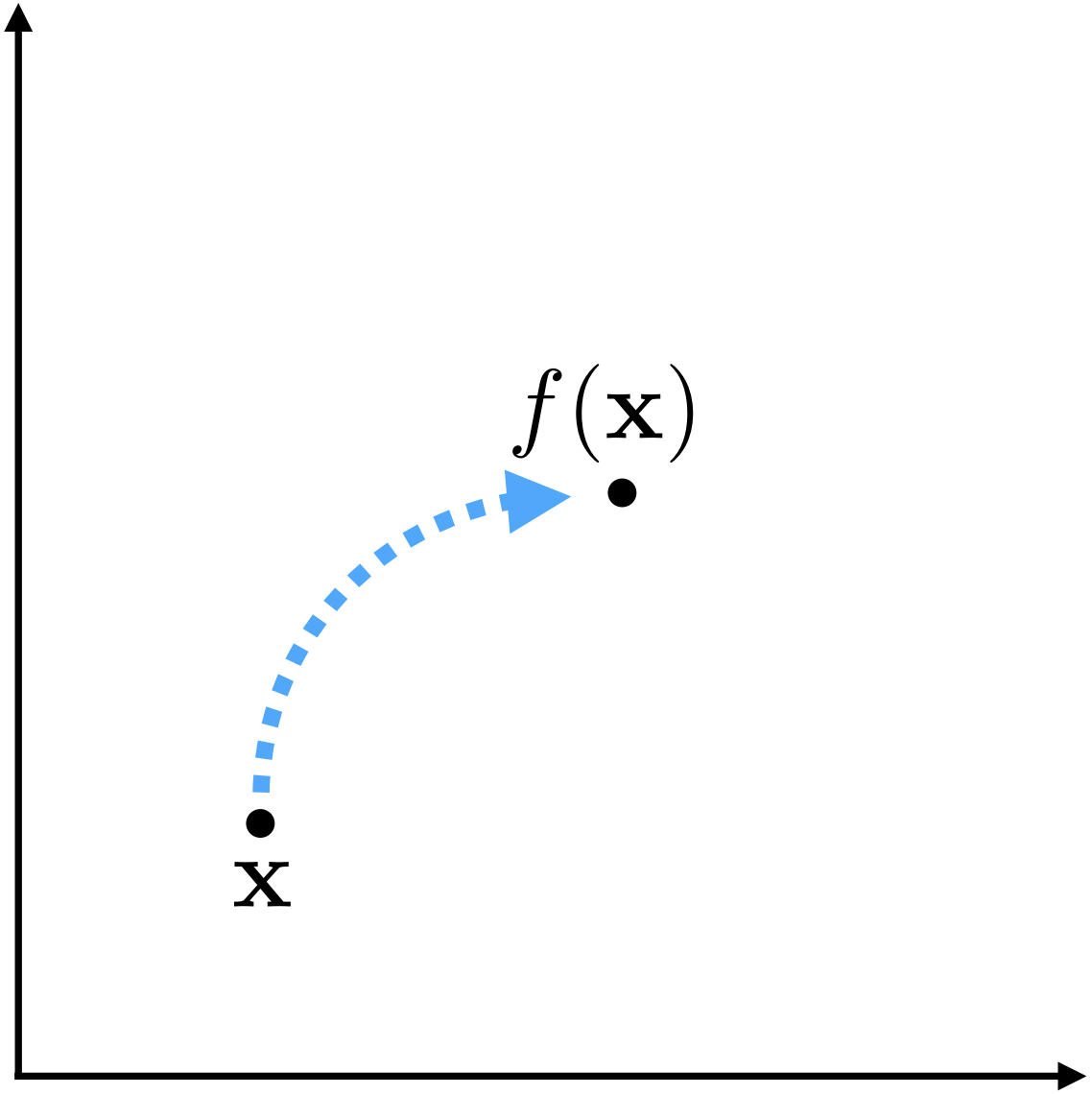
Why Study Transformation?

Basic Transformations

3D transformations

MVP Transformation

Basic idea: f transforms x to $f(x)$



What can we do with linear transformations?

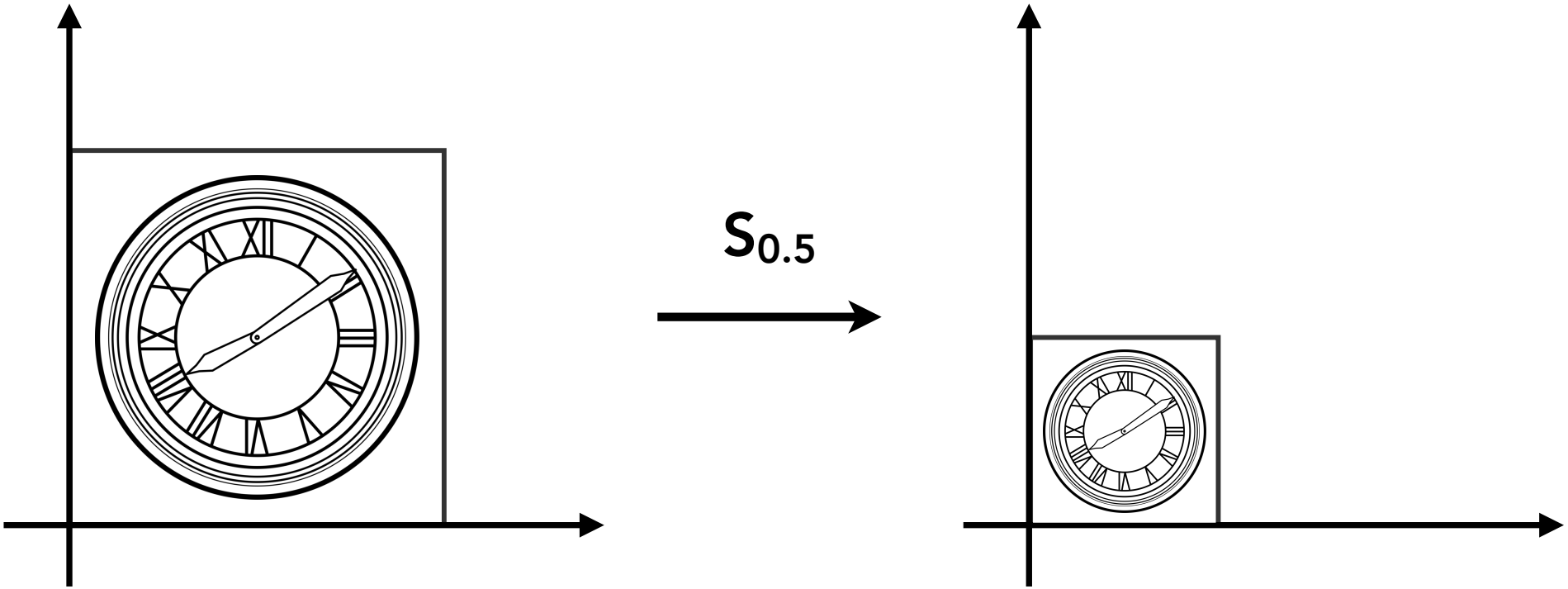
- What did linear mean? 指直线到直线?

$$f(x+y) = f(x) + f(y) \quad \text{可加性?}$$

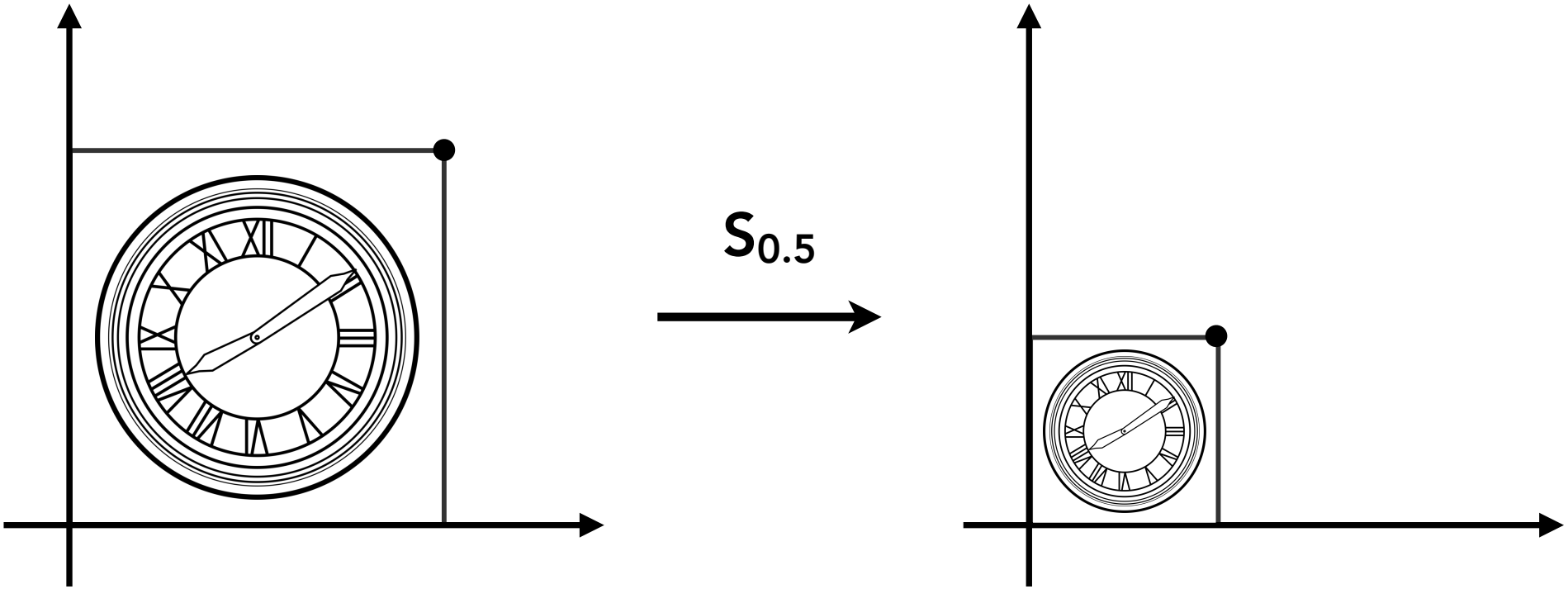
$$f(ax) = af(x) \quad \text{齐次性?}$$

- Cheap to compute
- Composition of linear transformation is linear
  - Leads to uniform representation of transformations
  - E.g., in graphics card (GPU) or graphics API

Scale Transform

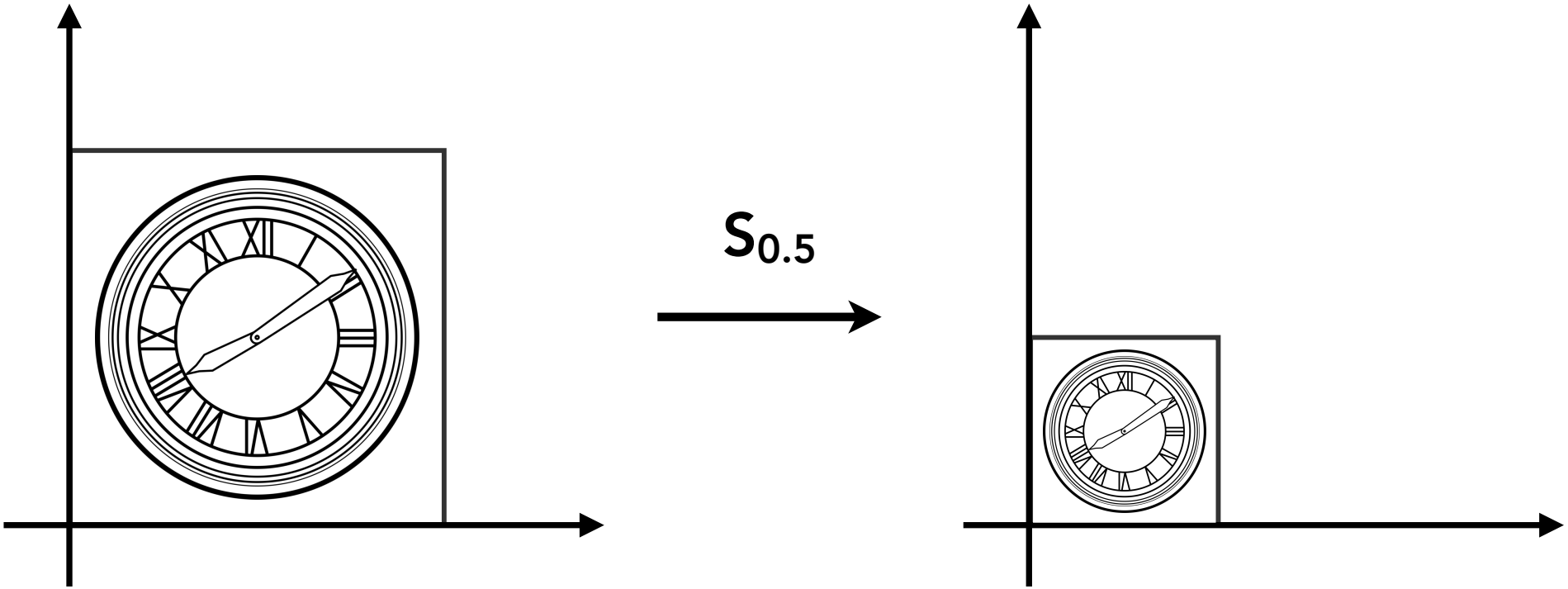


Scale Transform



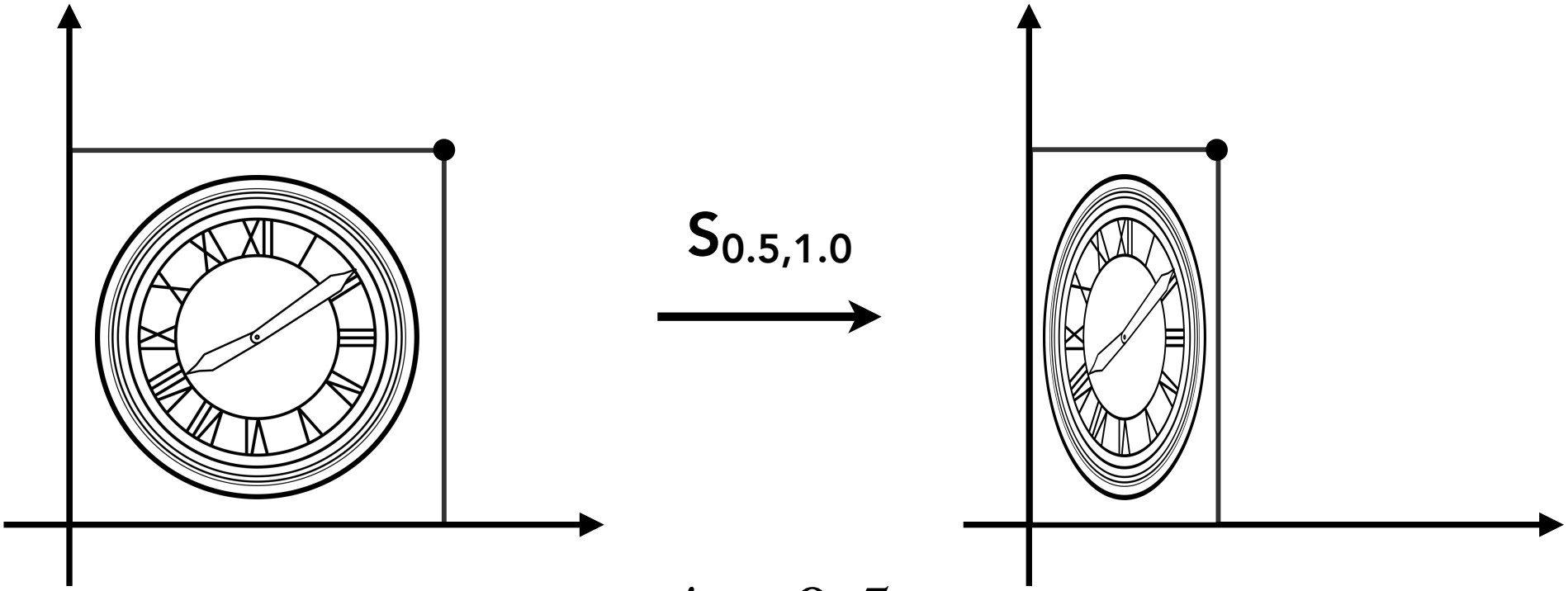
$$x' = 0.5x$$
$$y' = 0.5y$$

Scale Matrix



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

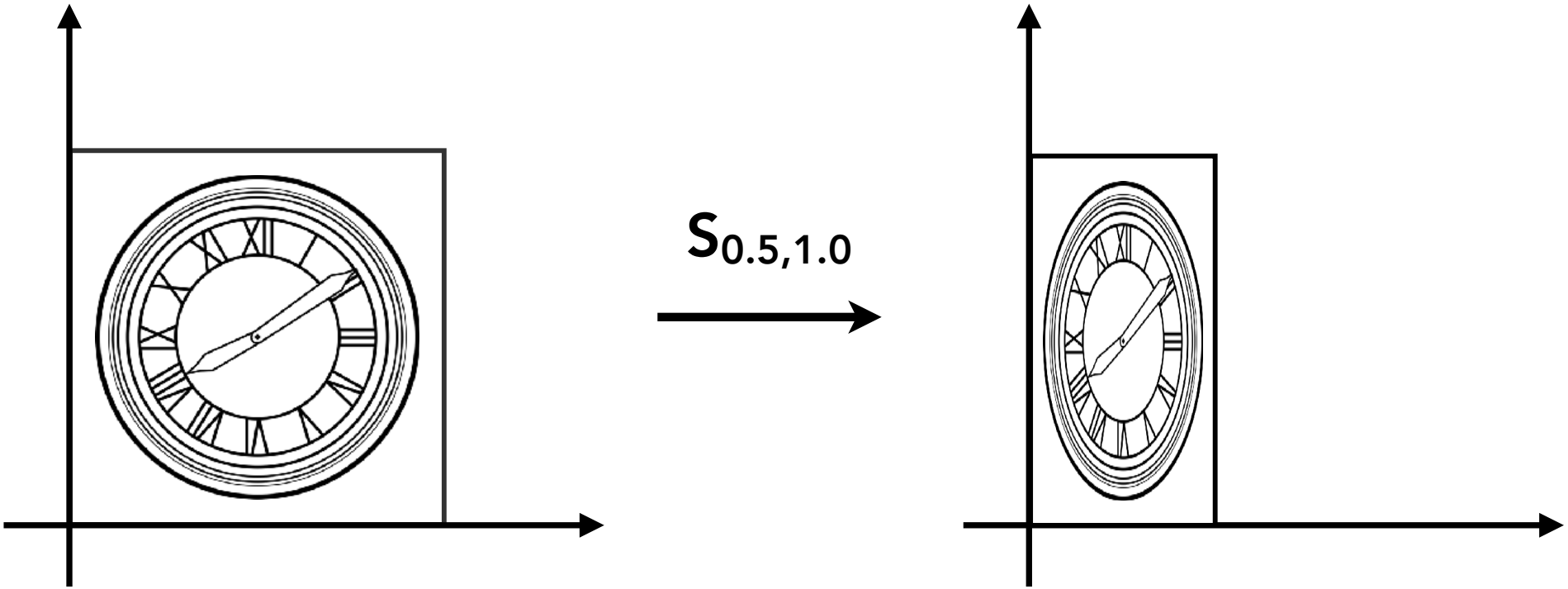
Scale Transform (non-uniform a.k.a anisotropic)



$$x' = 0.5x$$

$$y' = y$$

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



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

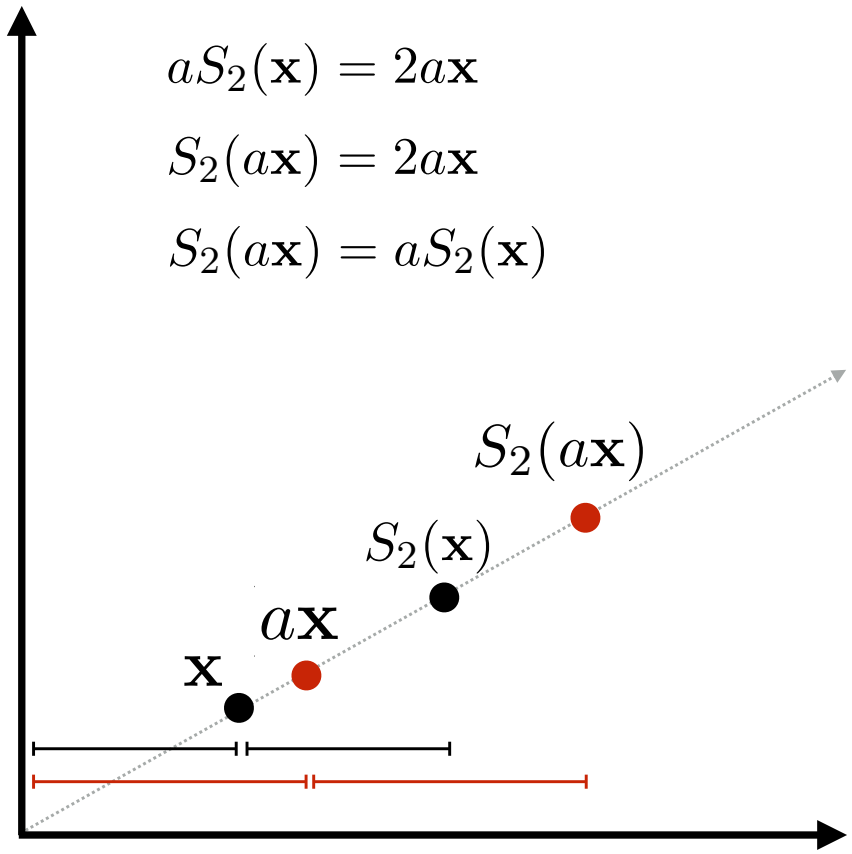
Is scale a linear transform?

$$S_2(\mathbf{x}) = 2\mathbf{x}$$

$$aS_2(\mathbf{x}) = 2a\mathbf{x}$$

$$S_2(a\mathbf{x}) = 2a\mathbf{x}$$

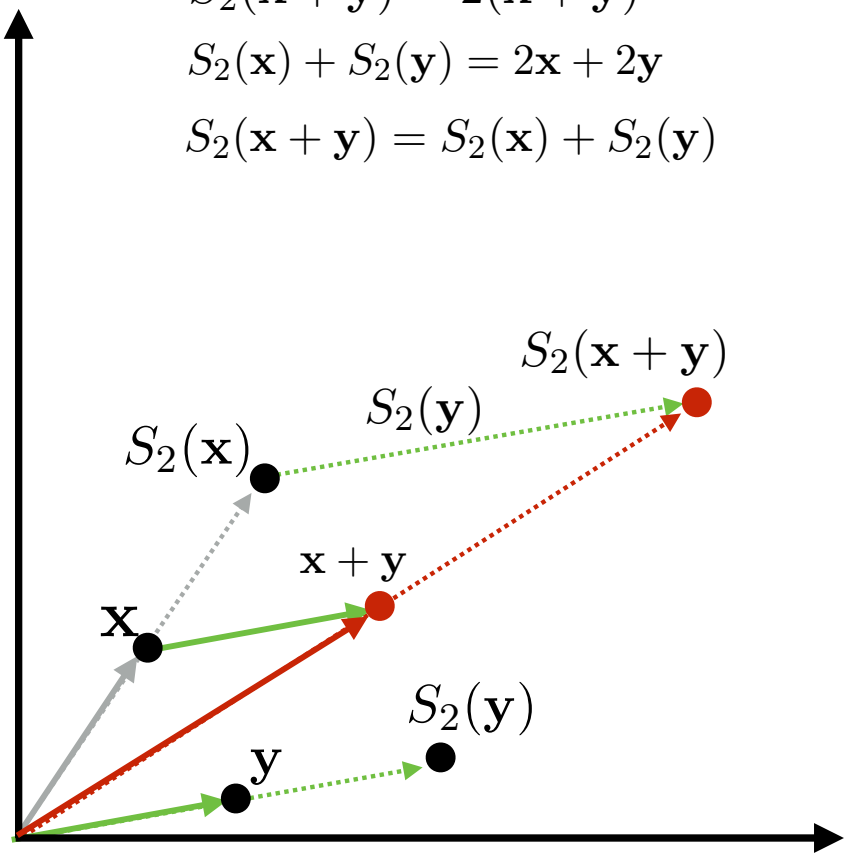
$$S_2(a\mathbf{x}) = aS_2(\mathbf{x})$$



$$S_2(\mathbf{x}+\mathbf{y}) = 2(\mathbf{x}+\mathbf{y})$$

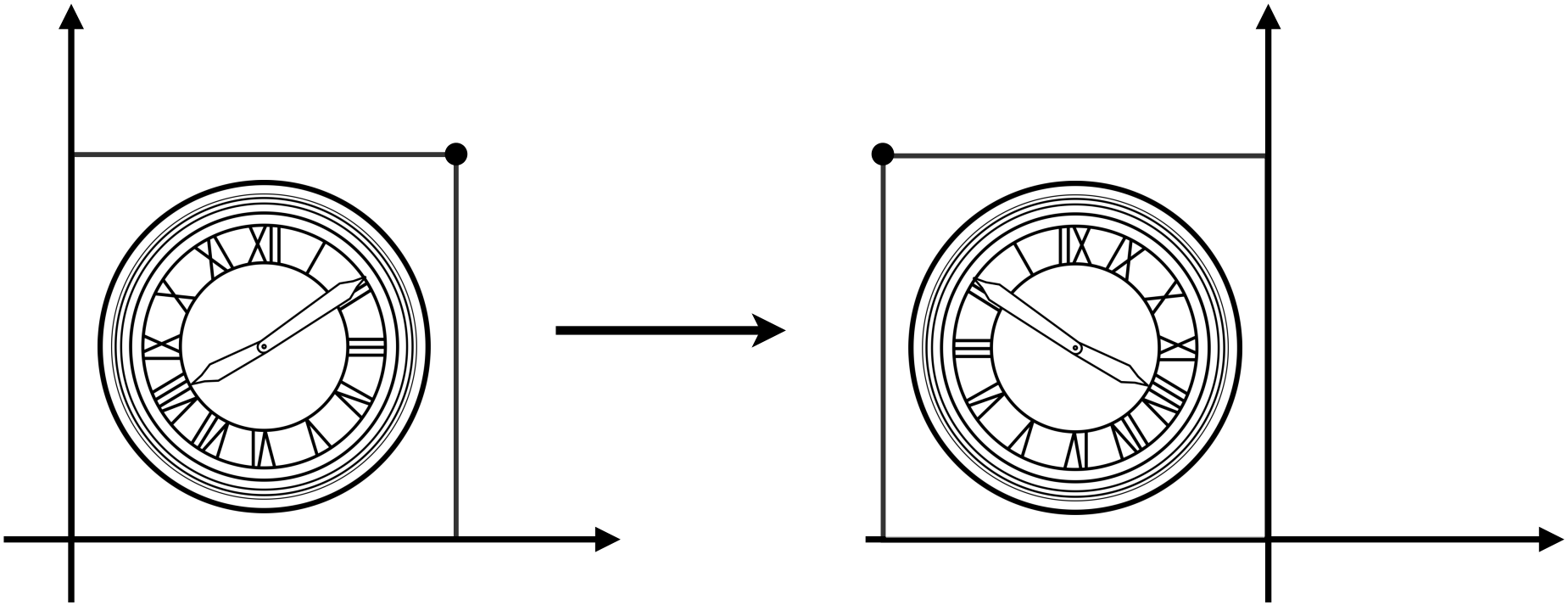
$$S_2(\mathbf{x}) + S_2(\mathbf{y}) = 2\mathbf{x} + 2\mathbf{y}$$

$$S_2(\mathbf{x}+\mathbf{y}) = S_2(\mathbf{x}) + S_2(\mathbf{y})$$



Yes!

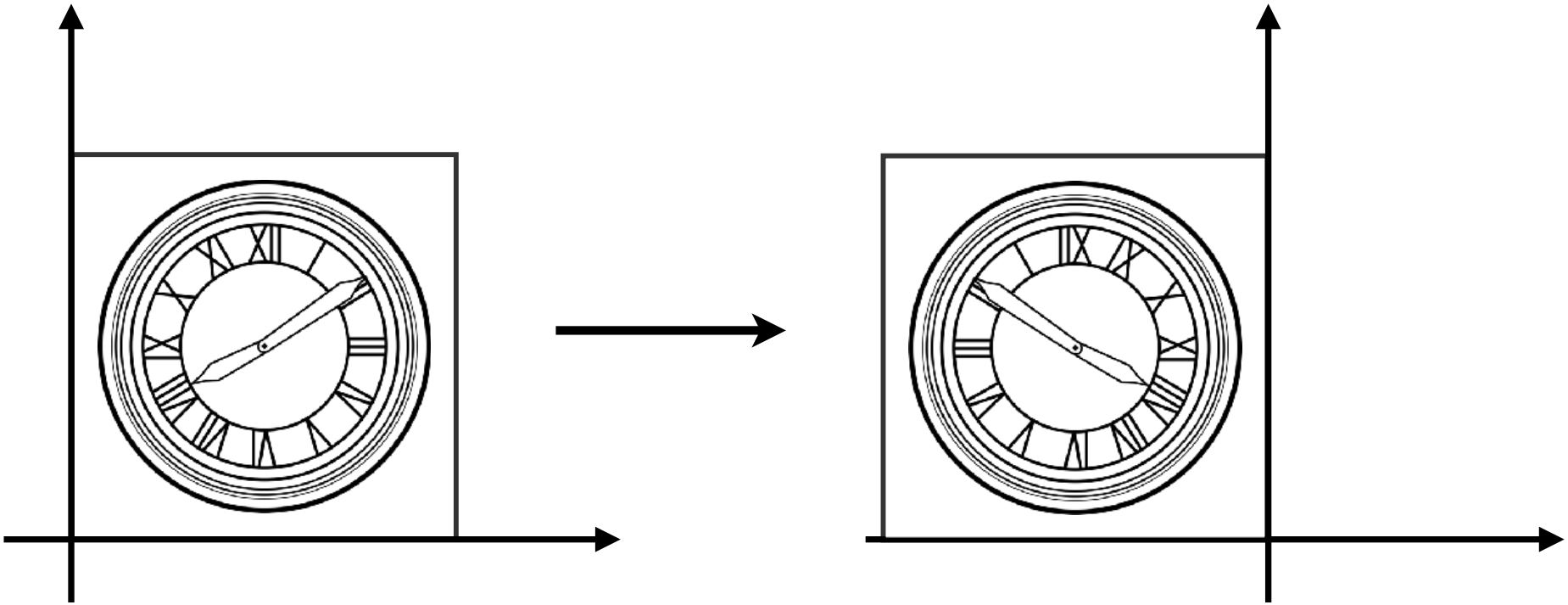
Reflection Transform



$$x' = -1.0x$$

$$y' = y$$

Reflection Matrix

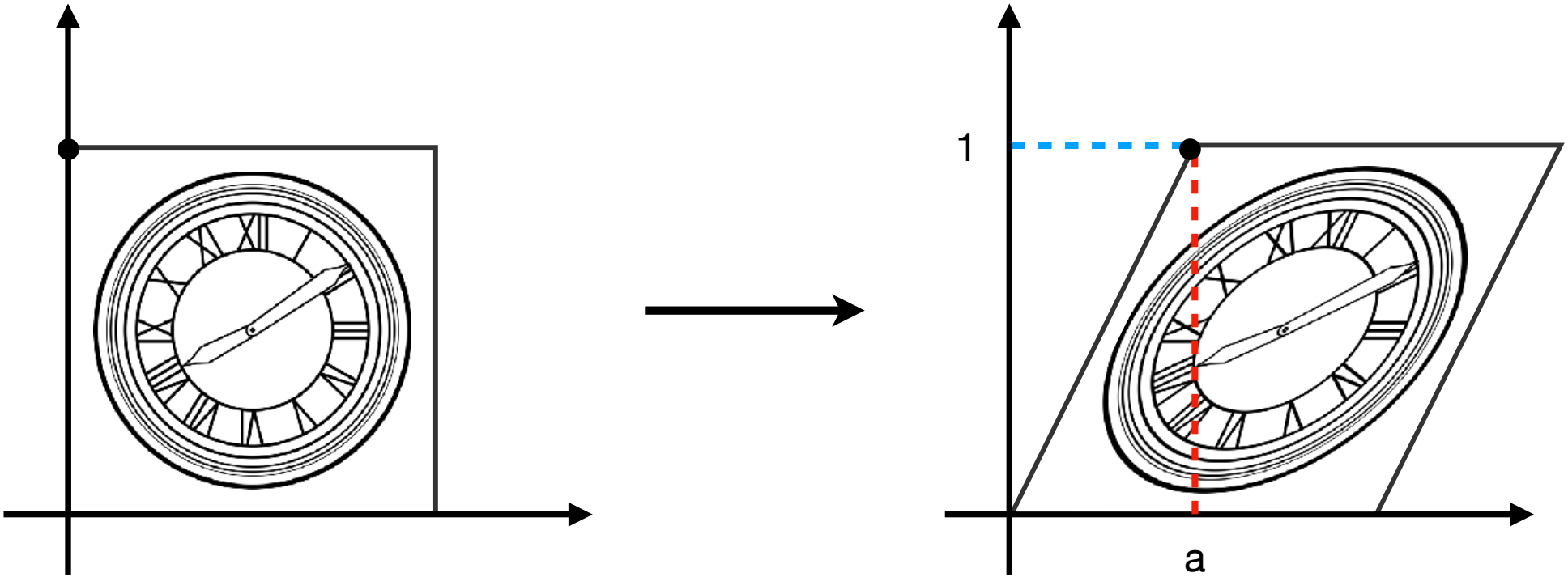


Horizontal reflection:

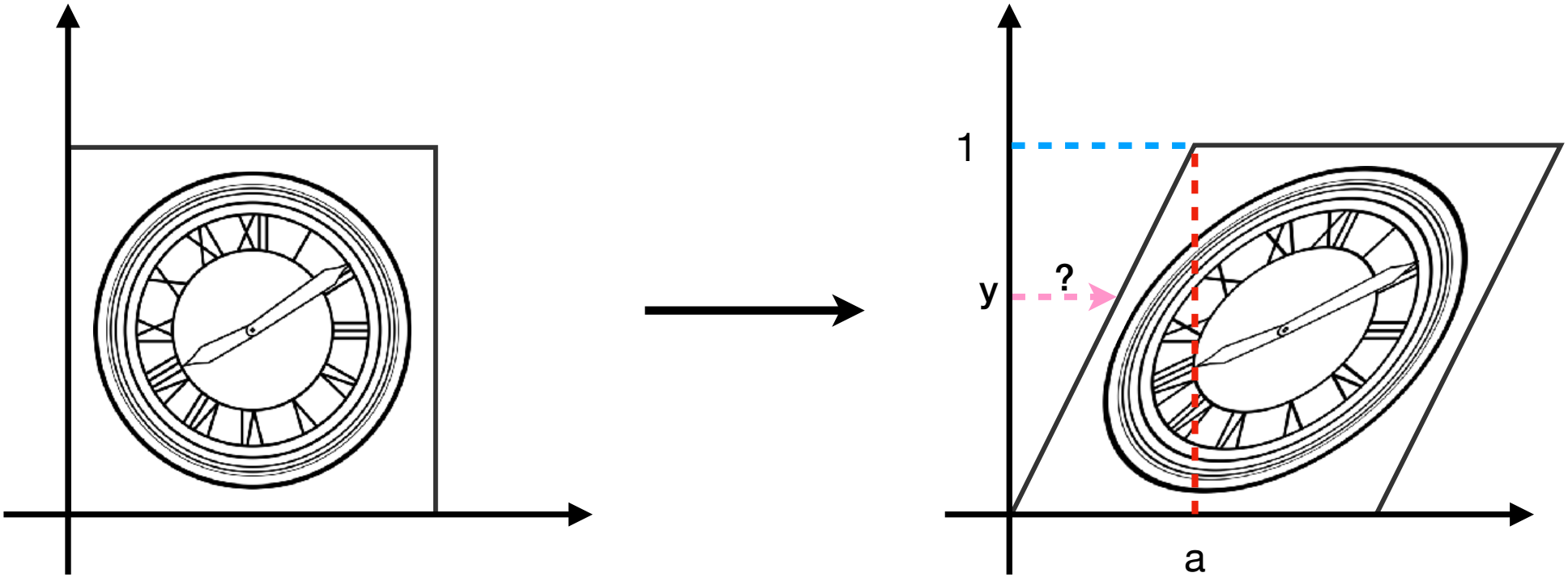
$$x' = -x$$
$$y' = y$$

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

Shear Matrix



Shear Matrix



Hints:

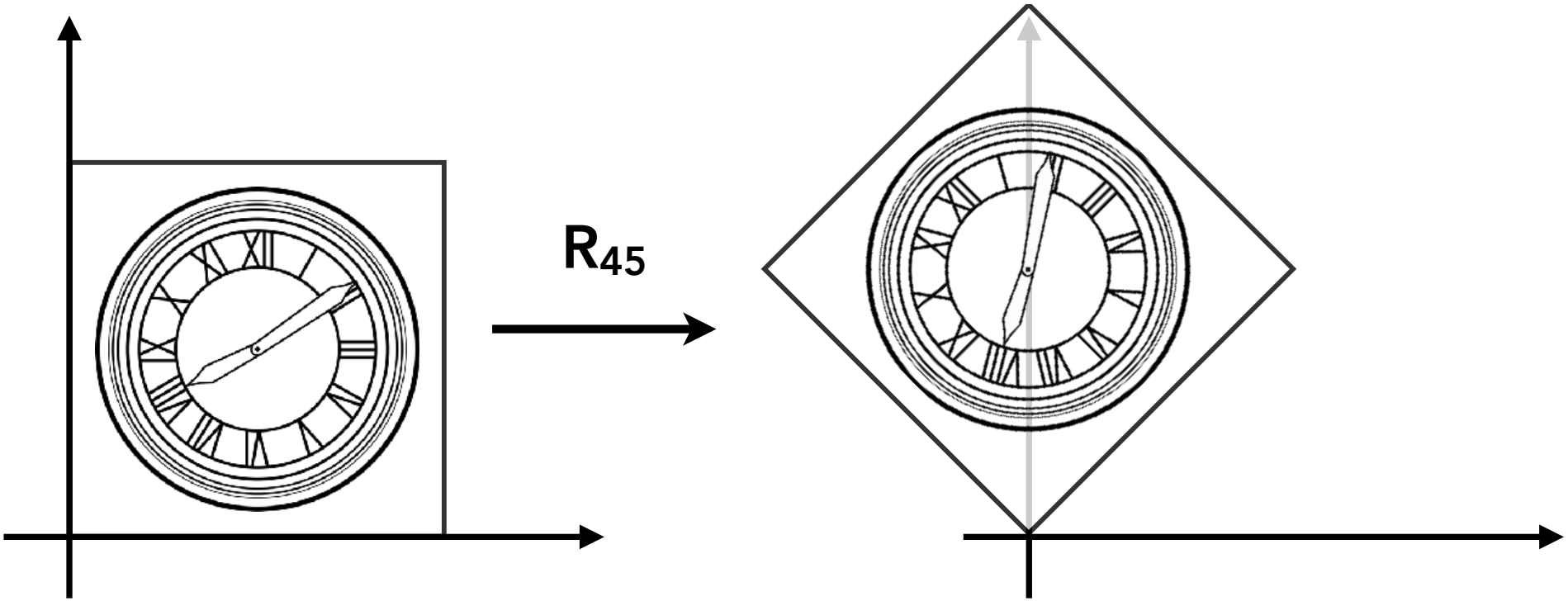
Horizontal shift is 0 at y=0

Horizontal shift is a at y=1

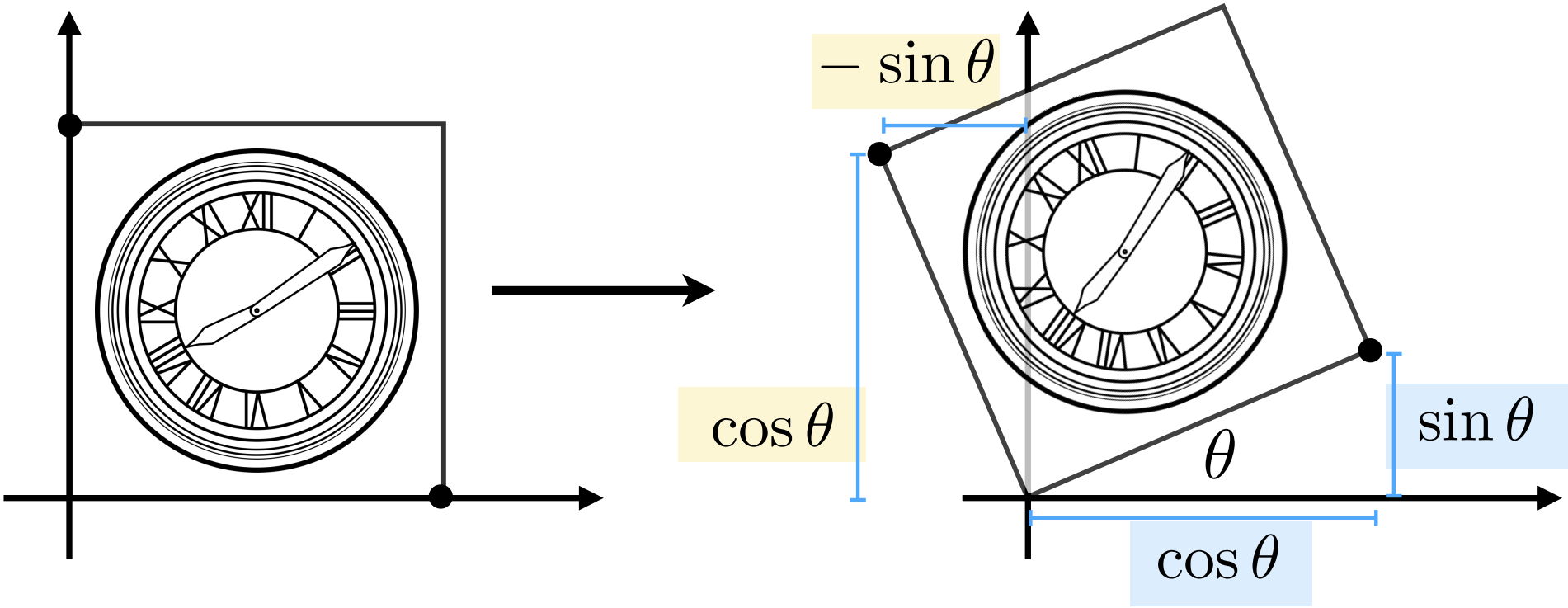
Vertical shift is always 0

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

Rotate (about the origin (0, 0), CCW by default)

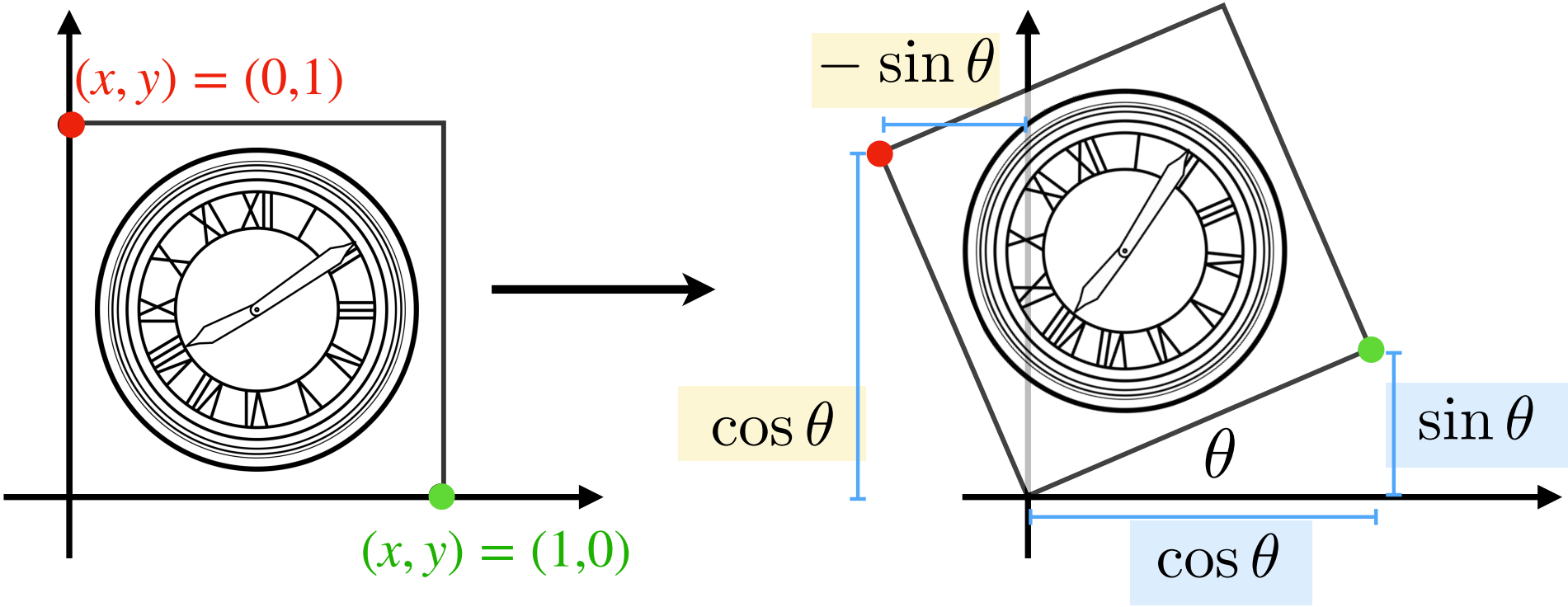


Rotation Matrix



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

Rotation Matrix



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

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

$$x' = cos\theta \times 1$$

$$y' = sin\theta \times 1$$

$$x' = -\,sin\theta \times 1$$

$$y' = cos\theta \times 1$$

Linear Transformation = Matrix

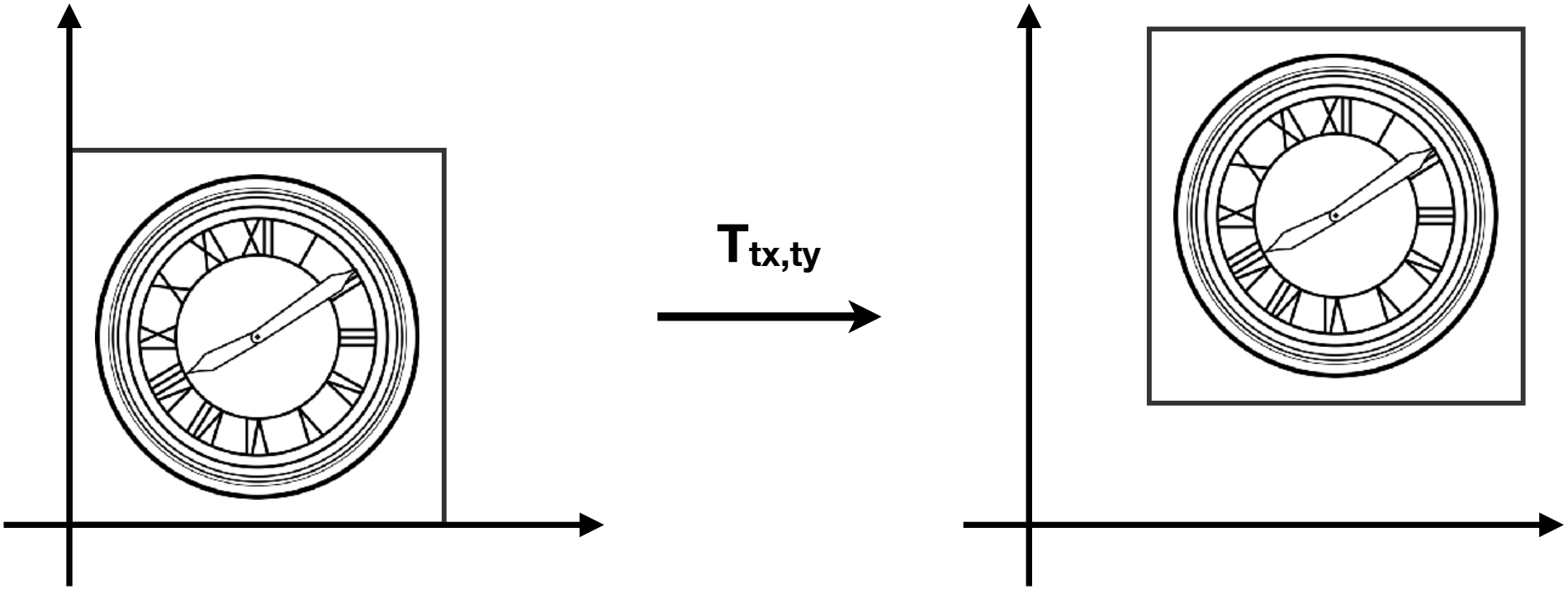
$$x' = a\,x + b\,y$$
$$y' = c\,x + d\,y$$

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

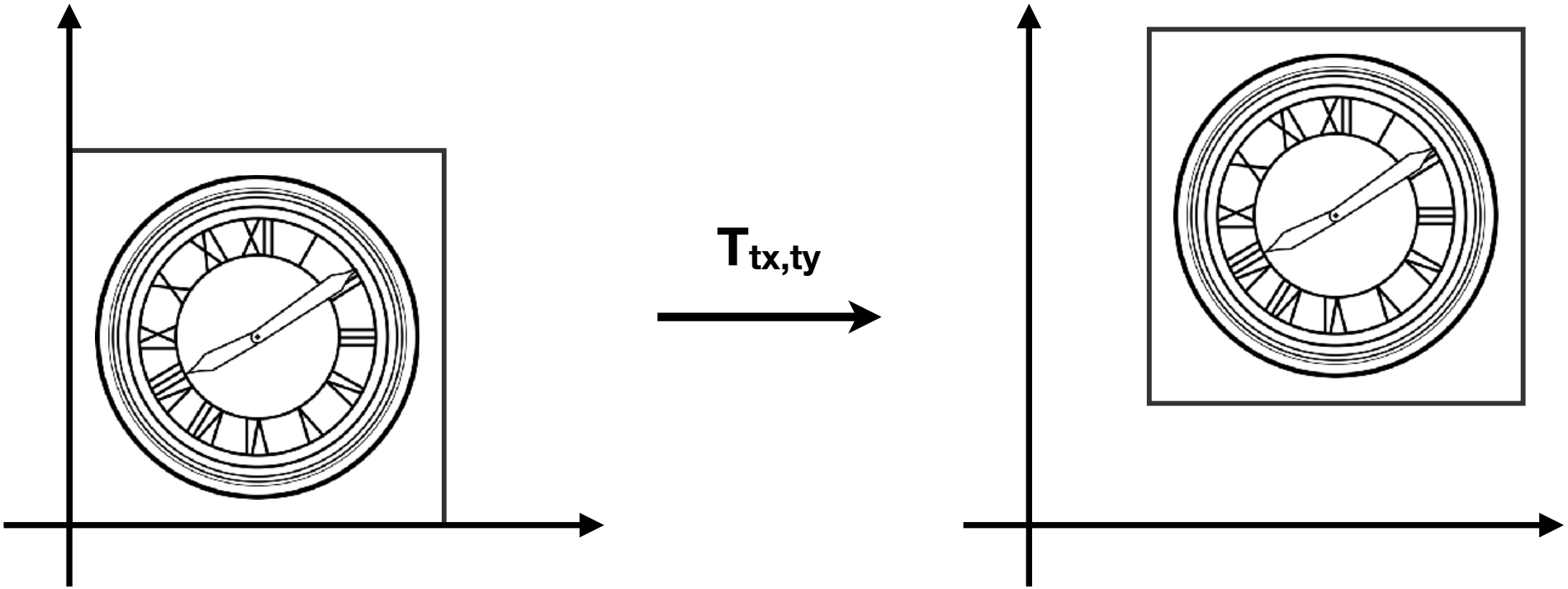
$$\mathbf{x}' = \mathbf{M}\ \mathbf{x}$$

2D线性变换可以用2x2矩阵表示 $f(x) = A_{2\times 2}x$

Translation

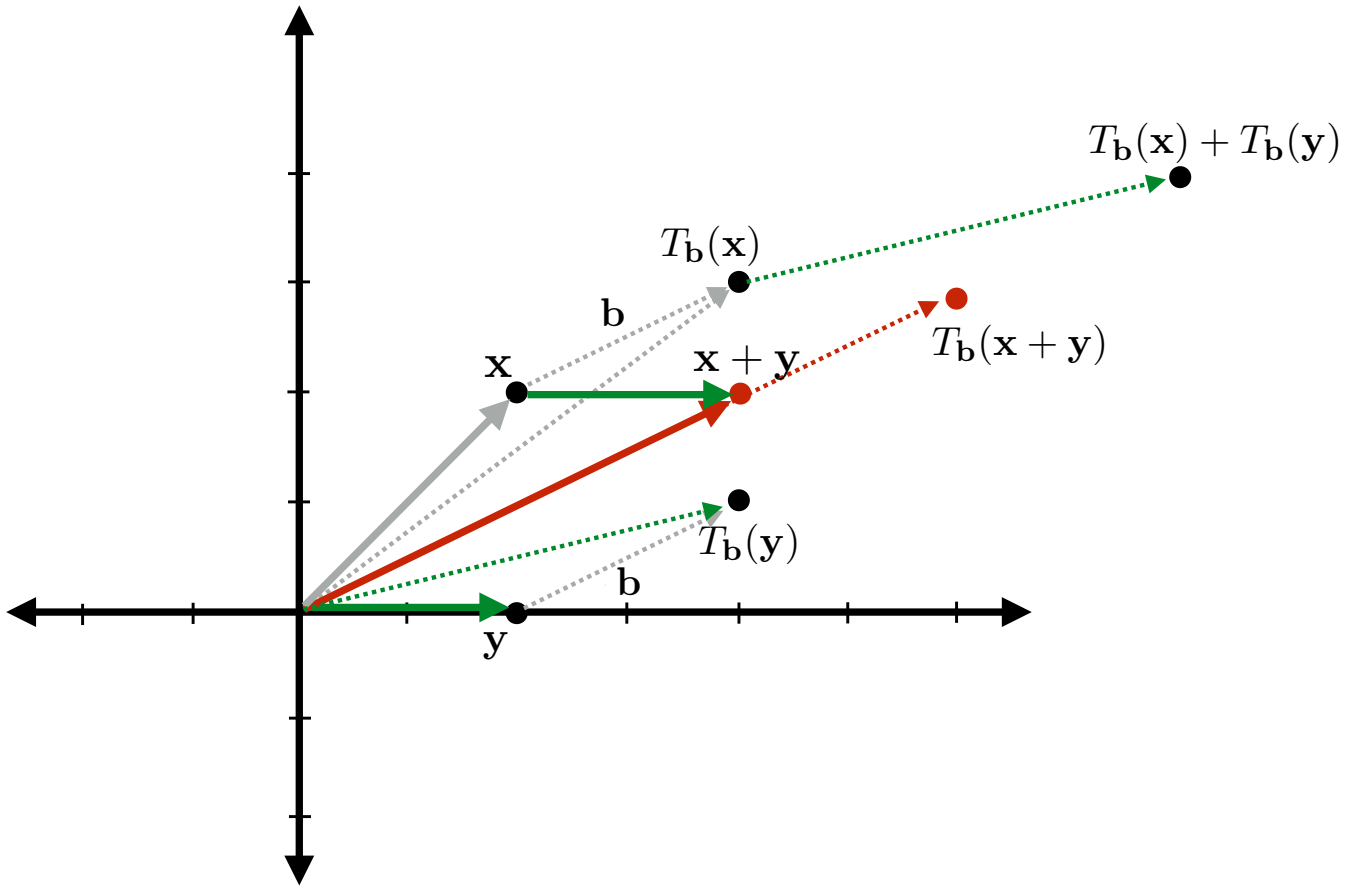


Translation??



$$x' = x + t_x$$
$$y' = y + t_y$$

Is translation a linear transform?



No! Translation is affine.

Can we represent translation in matrix form?

- Translation cannot be represented in matrix form

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

(So, translation is NOT linear transform!)

- But we don't want translation to be a special case
- Is there a unified way to represent all transformations?

Solution: Homogenous Coordinates

Add a third coordinate (*w-coordinate*) (x, y, w)

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

Now you can express translation as a matrix!!

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

优点：统一表示、简化几何变换的计算

Homogenous Coordinates

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

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

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

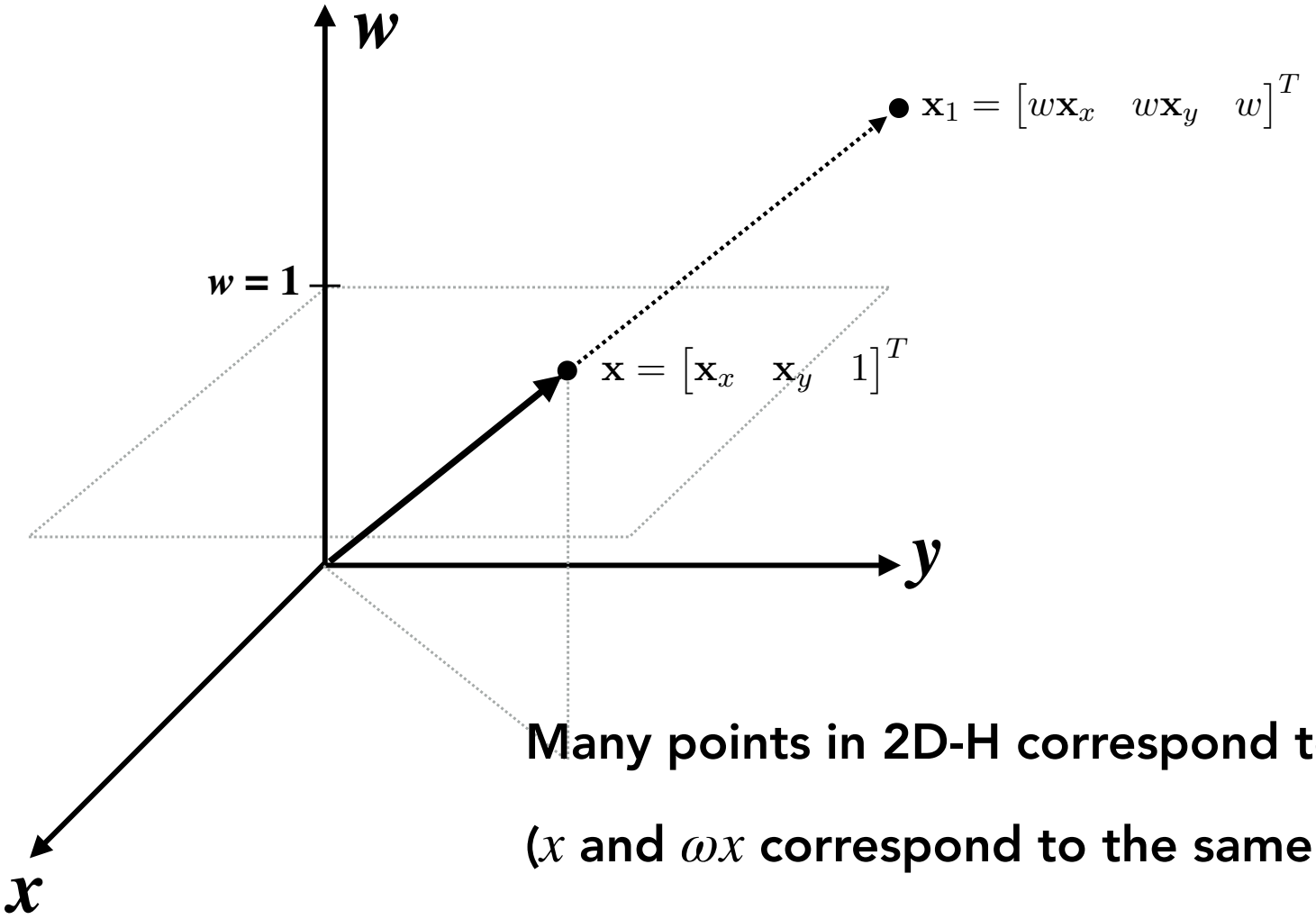
- vector + vector = vector
- point – point = vector
- point + vector = point
- point + point = ??

In homogeneous coordinates,

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} \text{ is the 2D point } \begin{pmatrix} x/w \\ y/w \\ 1 \end{pmatrix}, \; w \neq 0$$

规范化齐次坐标

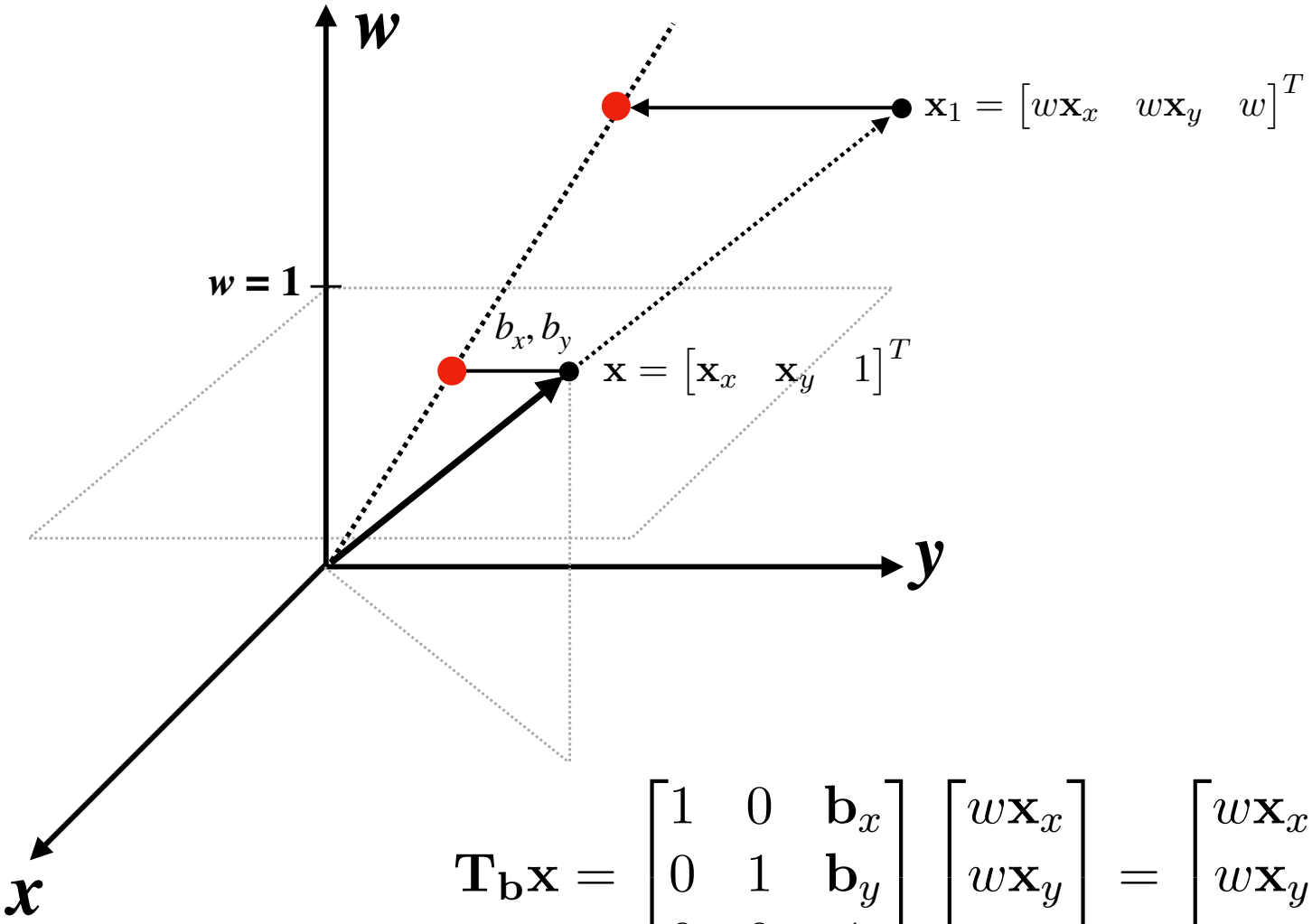
Homogeneous coordinates: some intuition

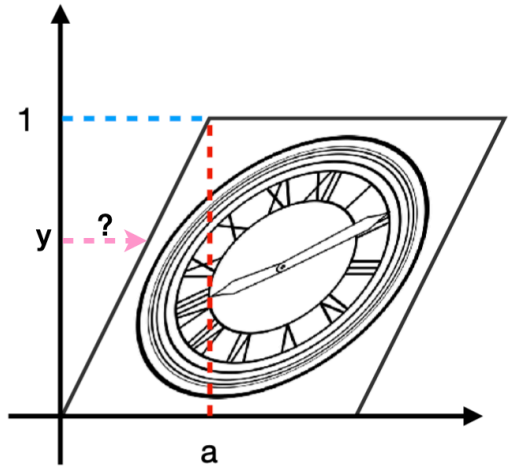


Many points in 2D-H correspond to same point in 2D

(x and ωx correspond to the same 2D point)

Homogeneous coordinates: some intuition





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

2D Shear

$$\mathbf{T_b x} = \begin{bmatrix} 1 & 0 & \mathbf{b}_x \\ 0 & 1 & \mathbf{b}_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} w\mathbf{x}_x \\ w\mathbf{x}_y \\ w \end{bmatrix} = \begin{bmatrix} w\mathbf{x}_x + w\mathbf{b}_x \\ w\mathbf{x}_y + w\mathbf{b}_y \\ w \end{bmatrix}$$

Translation is a shear in x and y in 2D-H space

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:

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

Homogeneous coordinates let us encode translations as linear transformations!

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

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

2D Transformations
(homogenous coordinates)

"Projective Transform"
(any matrix)

"Affine Transform"

Scale

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

"Similarity Transform if Sx=Sy"

Rotation

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

"Rigid Transform"

Translation

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

Summary of basic transformations

Linear:

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$$
$$f(a\mathbf{x}) = af(\mathbf{x})$$

Scale
Rotation
Reflection
Shear

Affine:

Composition of linear transform + translation

$$f(\mathbf{x}) = g(\mathbf{x}) + \mathbf{b}$$

Not affine: perspective projection

Euclidean:

Preserve distance between points

$$|f(\mathbf{x}) - f(\mathbf{y})| = |\mathbf{x} - \mathbf{y}|$$

Rotation
Reflection
Translation

Not linear:

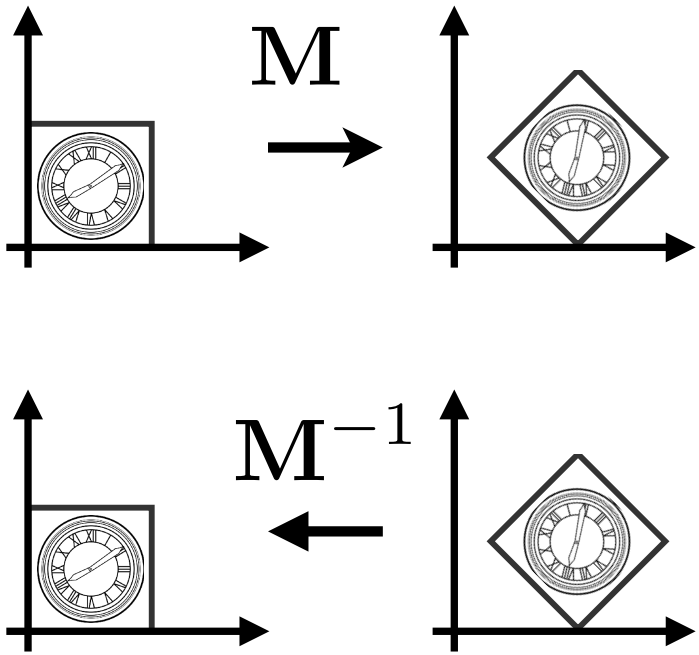
Translation

"Rigid body" transformations are distance-preserving motions that also preserve orientation (i.e., does not include reflection)

Inverse Transform

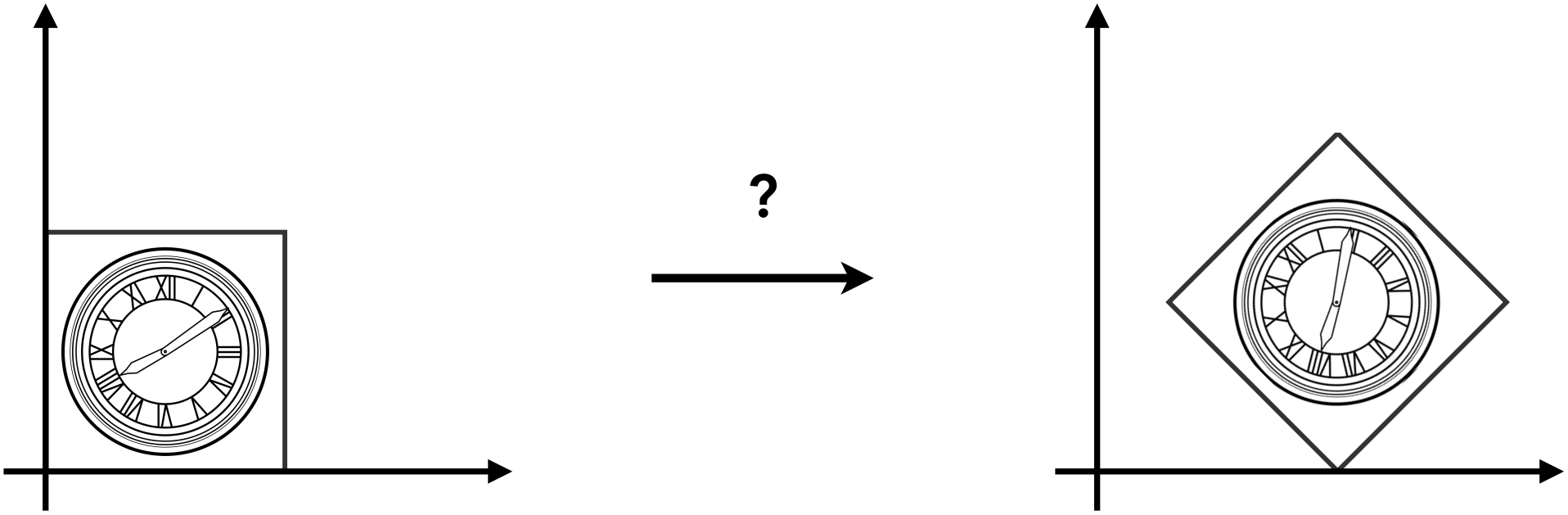
$$\mathbf{M}^{-1}$$

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

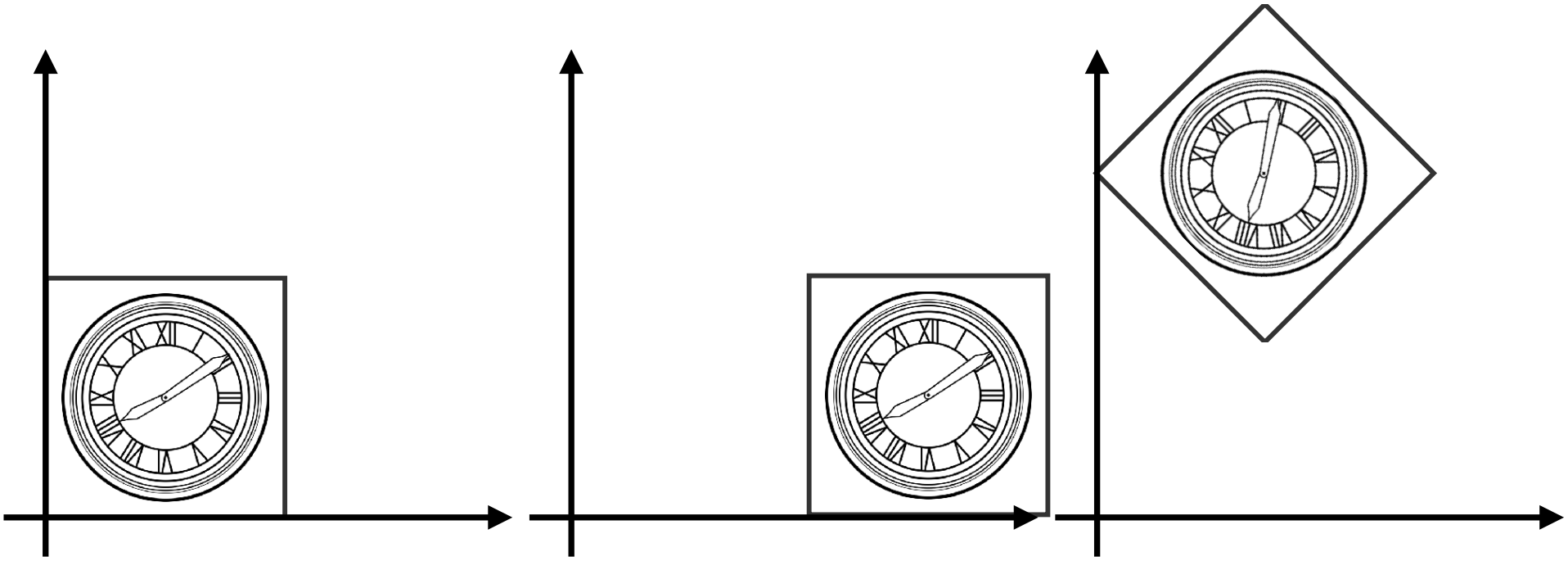


Composing Transforms

Composite Transform

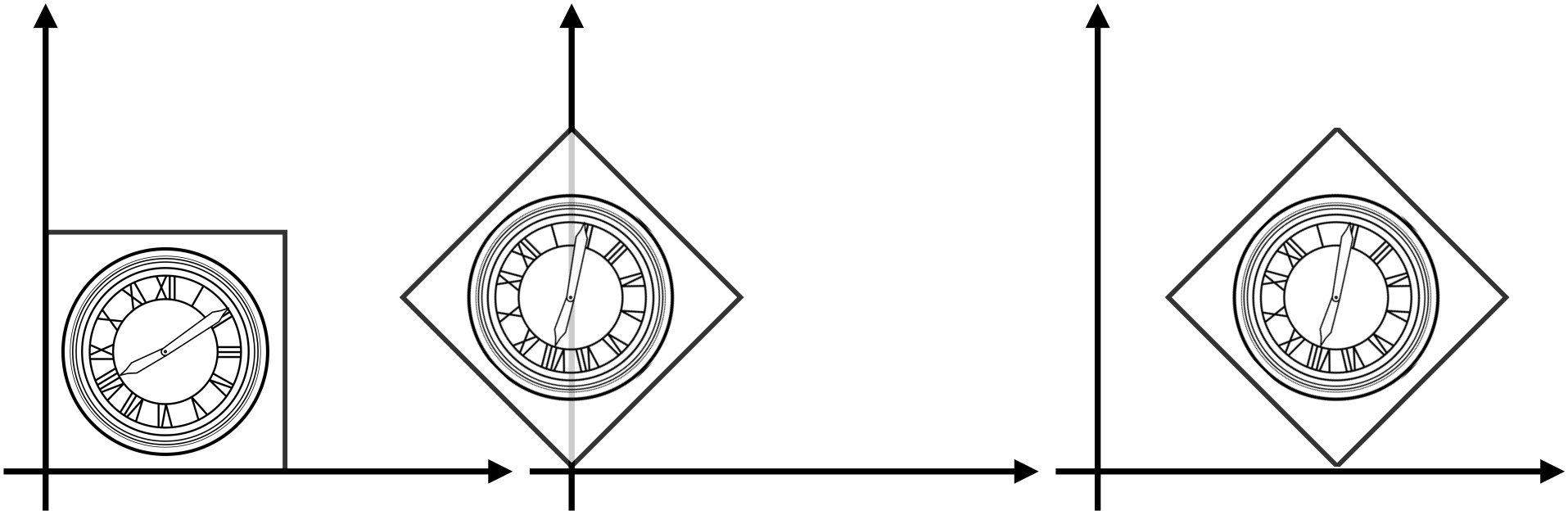


Translate Then Rotate?



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

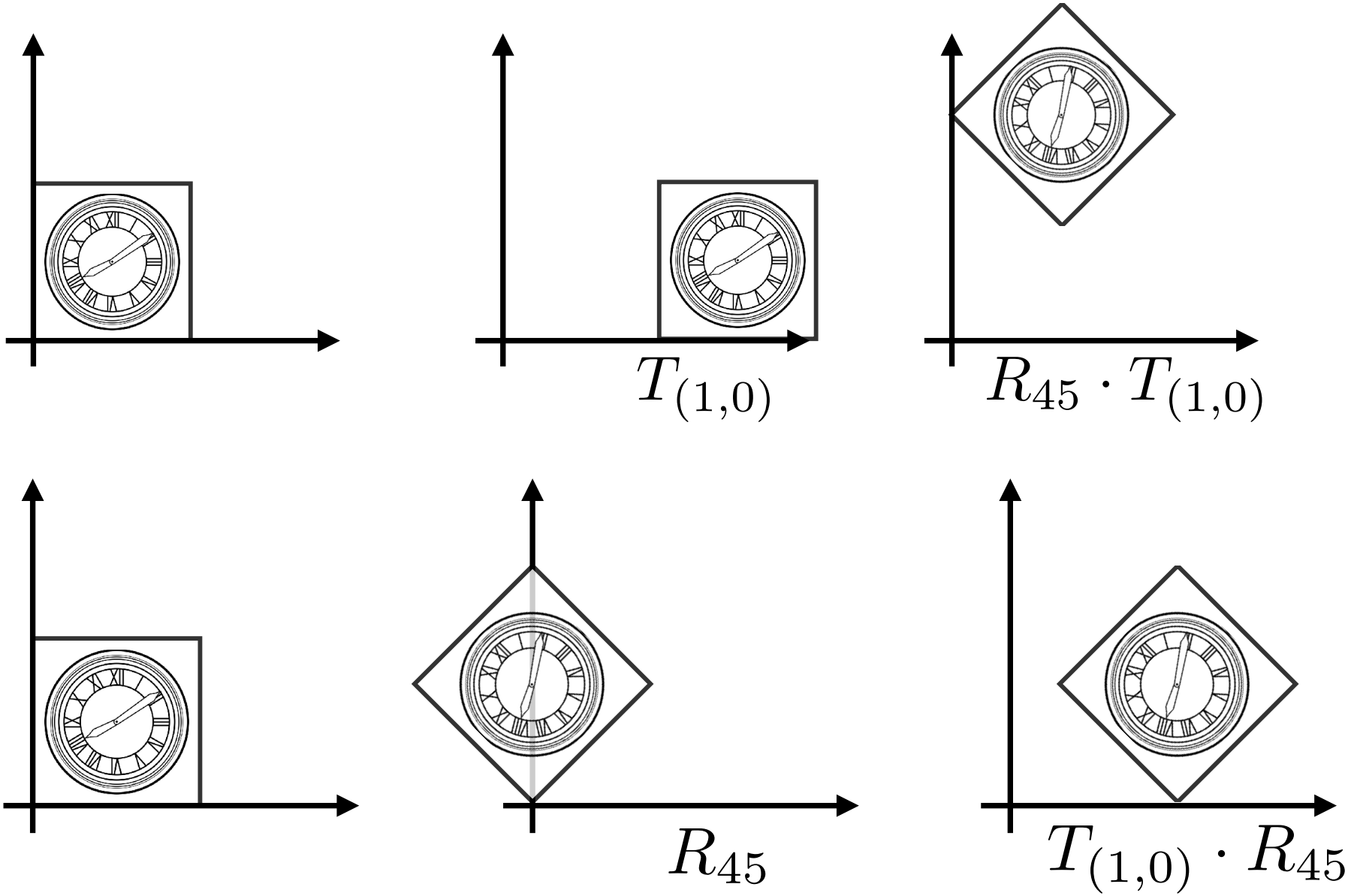
Rotate Then Translate



R_{45}

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

Transform Ordering Matters!



Transform Ordering Matters!

Matrix multiplication is <u>not</u> commutative

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

Note that matrices are applied right to left:

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

Composing Transforms

Sequence of transforms A_1, A_2, A_3, ...

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

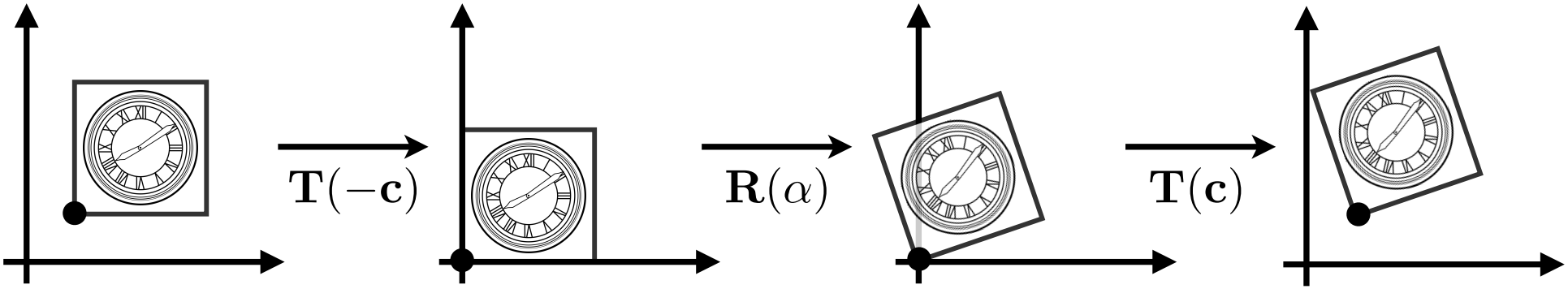
$$A_n(\ldots A_2(A_1(\mathbf{x}))) \;=\; \mathbf{A}_n \cdots \mathbf{A}_2 \cdot \mathbf{A}_1 \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

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

Building Complex Transforms

How to rotate around a given point c?

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



Matrix representation?

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

Today's Topics

Why Study Transformation?

Basic transformations

3D transformations

MVP Transformation

3D Transformations

Use homogeneous coordinates again:

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

In general, $(x, y, z, w)(w != 0)$ is the 3D point $(x/w, y/w, z/w)$

3D Transformations

Scale

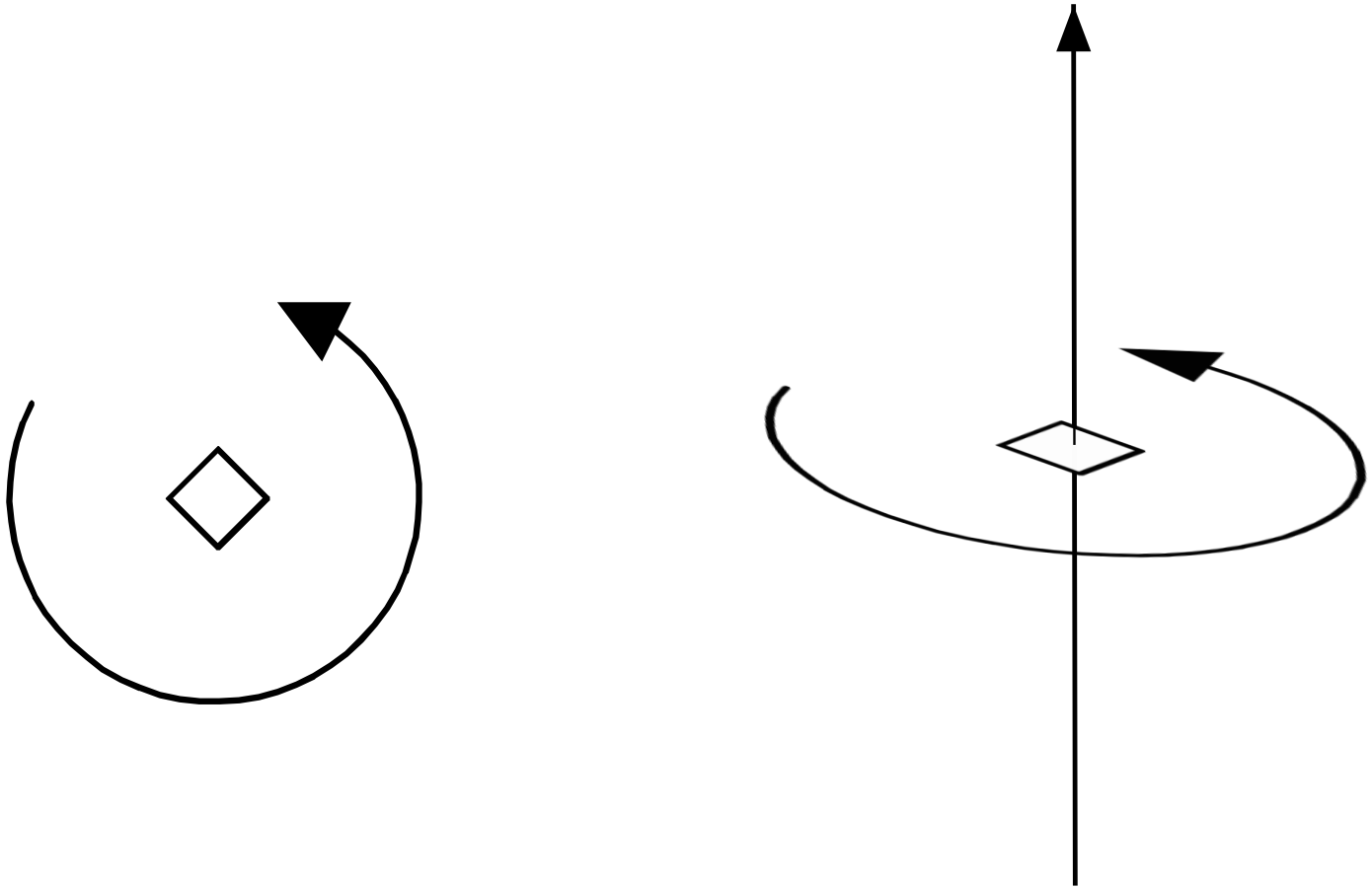
$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation

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

2D and 3D Rotations

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



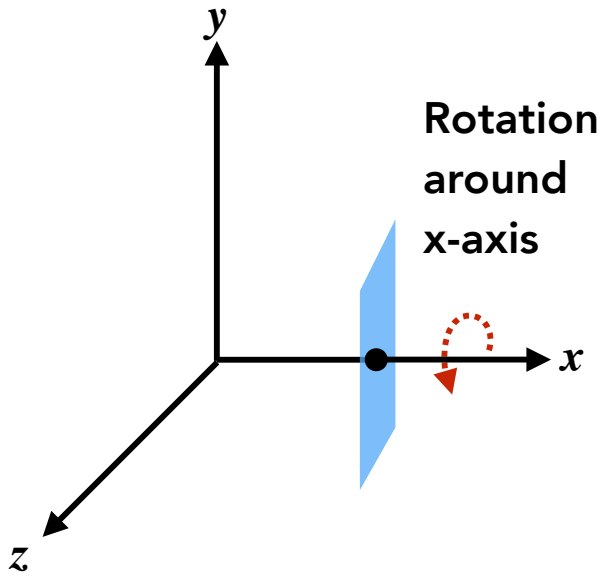
3D Transformations

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

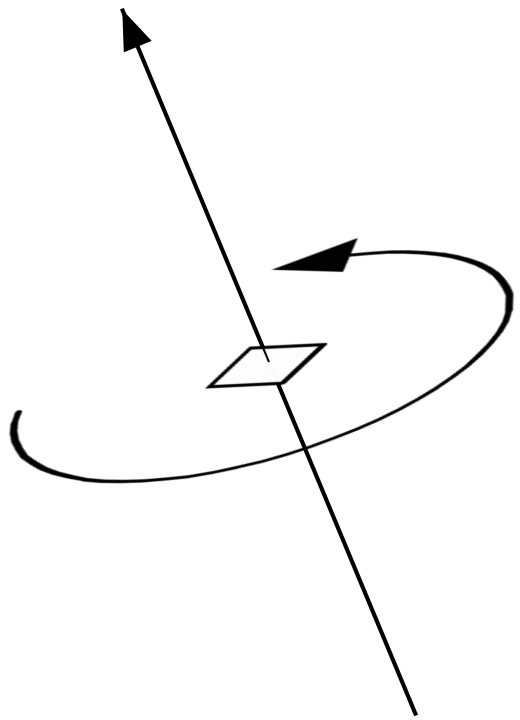
$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

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



3D Rotation Around Arbitrary Axis

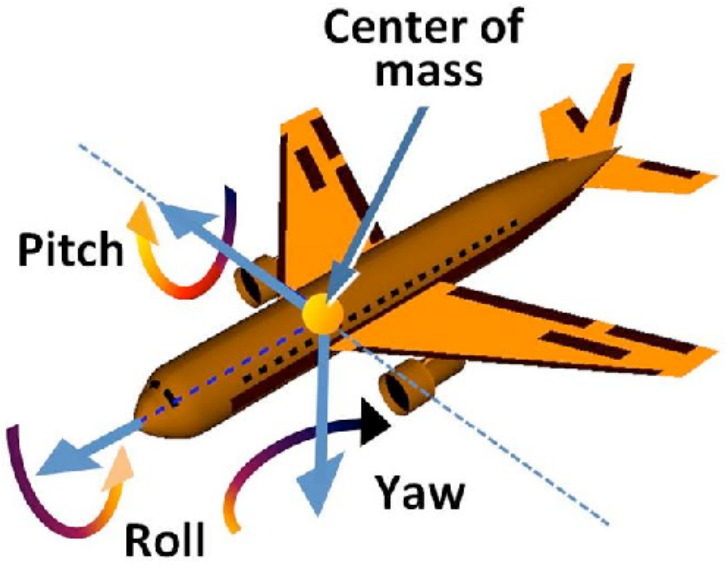


3D Rotations

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



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?

Rodrigues' Rotation Formula

Rodrigues' rotation formula - W

en.wikipedia.org/wiki/Rodrigues%27_rotation_formula

Update

Contents [hide]

Derivation [edit]

Let $\mathbf{k}$ be a unit vector defining a rotation axis, and let $\mathbf{v}$ be any vector to rotate about $\mathbf{k}$ by angle θ (right hand rule, anticlockwise in the figure), producing the rotated vector $\mathbf{v}_{\text{rot}}$.

Using the dot and cross products, the vector $\mathbf{v}$ can be decomposed into components parallel and perpendicular to the axis $\mathbf{k}$,

$$\mathbf{v} = \mathbf{v}_{\parallel} + \mathbf{v}_{\perp},$$

where the component parallel to $\mathbf{k}$ is called the vector projection of $\mathbf{v}$ on $\mathbf{k}$,

$$\mathbf{v}_{\parallel} = (\mathbf{v} \cdot \mathbf{k})\mathbf{k},$$

and the component perpendicular to $\mathbf{k}$ is called the vector rejection of $\mathbf{v}$ from $\mathbf{k}$:

$$\mathbf{v}_{\perp} = \mathbf{v} - \mathbf{v}_{\parallel} = \mathbf{v} - (\mathbf{k} \cdot \mathbf{v})\mathbf{k} = -\mathbf{k} \times (\mathbf{k} \times \mathbf{v}),$$

where the last equality follows from the vector triple product formula: $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \cdot \mathbf{c})\mathbf{b} - (\mathbf{a} \cdot \mathbf{b})\mathbf{c}$. Finally, the vector $\mathbf{k} \times \mathbf{v}_{\perp} = \mathbf{k} \times \mathbf{v}$ is a copy of $\mathbf{v}_{\perp}$ rotated 90° around $\mathbf{k}$. Thus the three vectors

$$\mathbf{k},\ \mathbf{v}_{\perp},\ \mathbf{k} \times \mathbf{v}$$

form a right-handed orthogonal basis of $\mathbb{R}^3$, with the last two vectors of equal length.

Under the rotation, the component $\mathbf{v}_{\parallel}$ parallel to the axis will not change magnitude nor direction:

$$\mathbf{v}_{\parallel \text{rot}} = \mathbf{v}_{\parallel};$$

while the perpendicular component will retain its magnitude but rotate its direction in the perpendicular plane spanned by $\mathbf{v}_{\perp}$ and $\mathbf{k} \times \mathbf{v}$, according to

$$\mathbf{v}_{\perp \text{rot}} = \cos(\theta)\mathbf{v}_{\perp} + \sin(\theta)\mathbf{k} \times \mathbf{v}_{\perp} = \cos(\theta)\mathbf{v}_{\perp} + \sin(\theta)\mathbf{k} \times \mathbf{v},$$

in analogy with the planar polar coordinates (r, θ) in the Cartesian basis $\mathbf{e}_x$, $\mathbf{e}_y$:

$$\mathbf{r} = r\cos(\theta)\mathbf{e}_x + r\sin(\theta)\mathbf{e}_y.$$

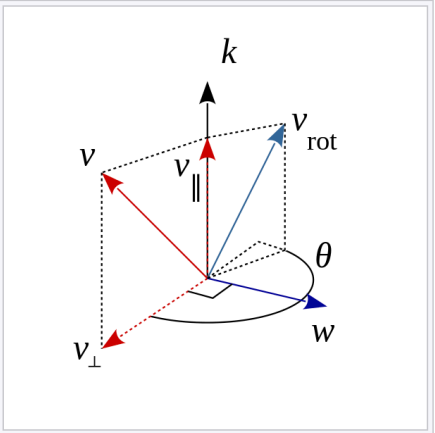
Now the full rotated vector is:

$$\mathbf{v}_{\text{rot}} = \mathbf{v}_{\parallel \text{rot}} + \mathbf{v}_{\perp \text{rot}} = \mathbf{v}_{\parallel} + \cos(\theta)\,\mathbf{v}_{\perp} + \sin(\theta)\mathbf{k} \times \mathbf{v}.$$

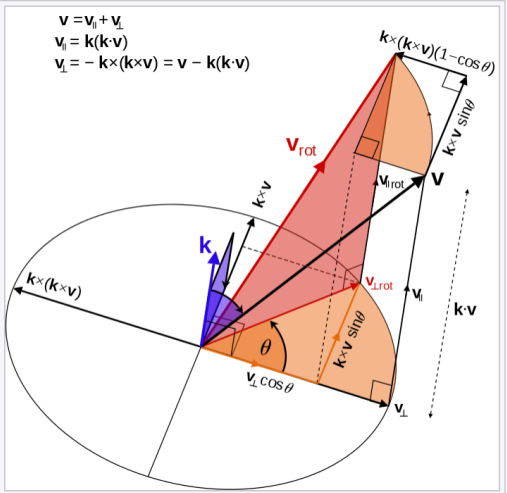
Substituting $\mathbf{v}_{\perp} = \mathbf{v} - \mathbf{v}_{\parallel}$ or $\mathbf{v}_{\parallel} = \mathbf{v} - \mathbf{v}_{\perp}$ in the last expression gives respectively:

$$\mathbf{v}_{\text{rot}} = \cos(\theta)\,\mathbf{v} + (1 - \cos\theta)(\mathbf{k} \cdot \mathbf{v})\mathbf{k} + \sin(\theta)\mathbf{k} \times \mathbf{v},$$

$$\mathbf{v}_{\text{rot}} = \mathbf{v} + (1 - \cos\theta)\mathbf{k} \times \mathbf{k} \times \mathbf{v} + \sin(\theta)\mathbf{k} \times \mathbf{v}.$$



Rodrigues' rotation formula rotates $\mathbf{v}$ by an angle θ around vector k by decomposing it into its components parallel and perpendicular to k, and rotating only the perpendicular component.

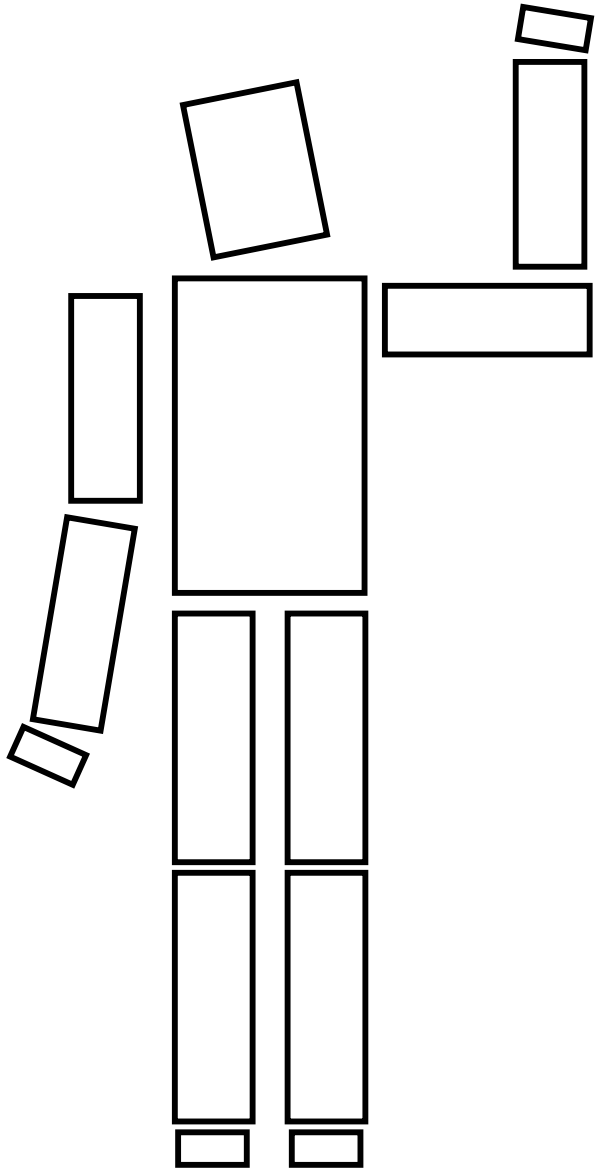


Vector geometry of Rodrigues' rotation formula, as well as the decomposition into parallel and perpendicular components.

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

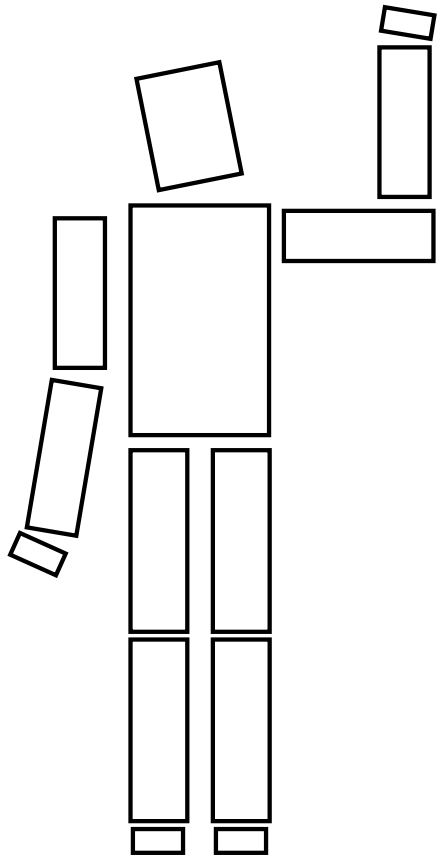


Linear Representation

Each shape associated with its own transform

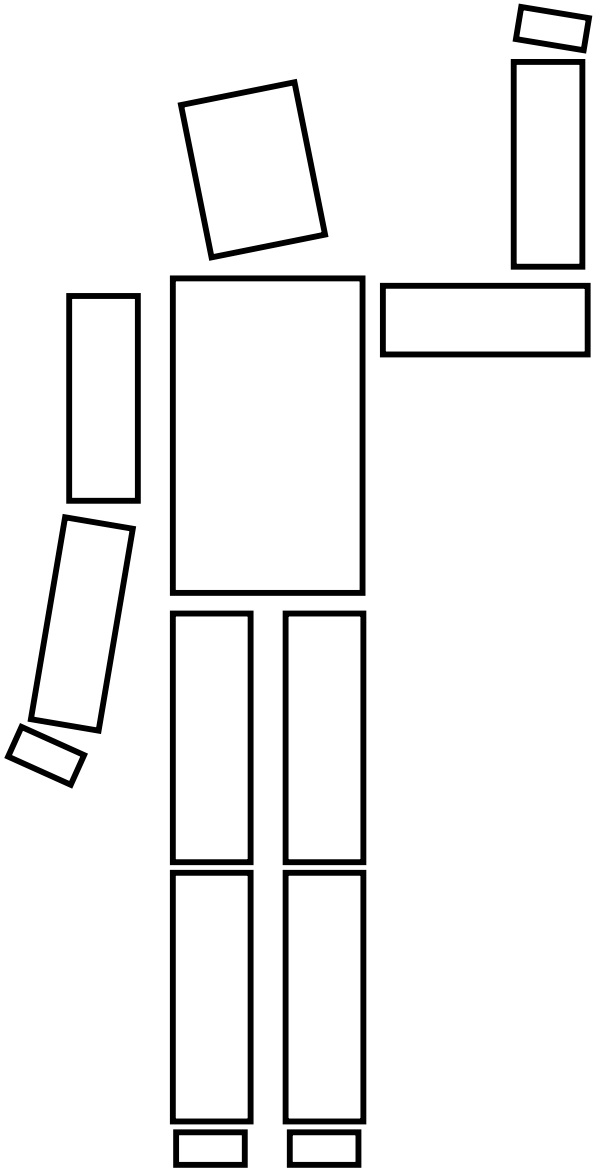
A single edit can require updating many transforms

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



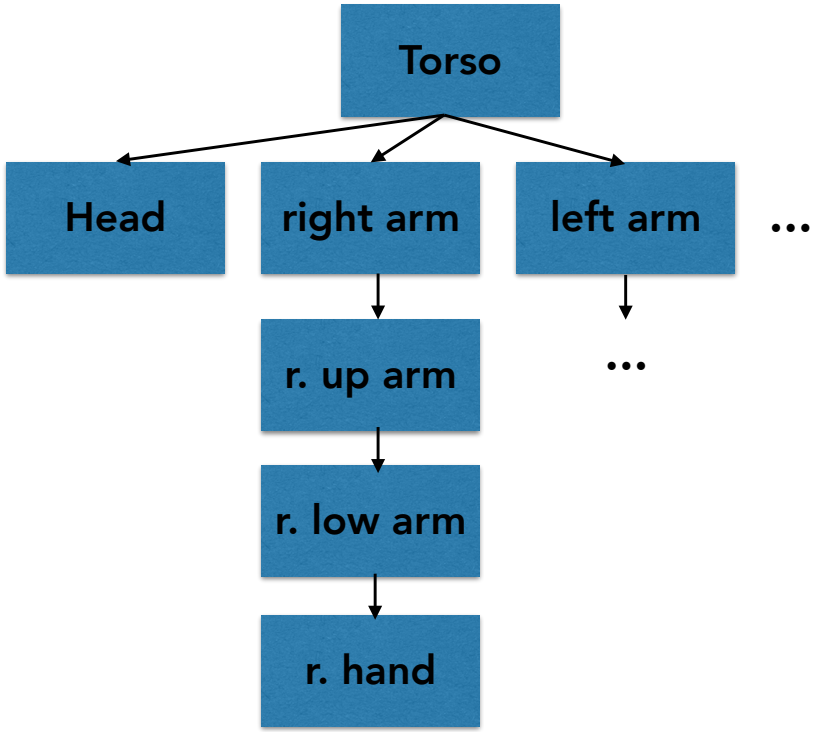
Skeleton - Hierarchical Representation

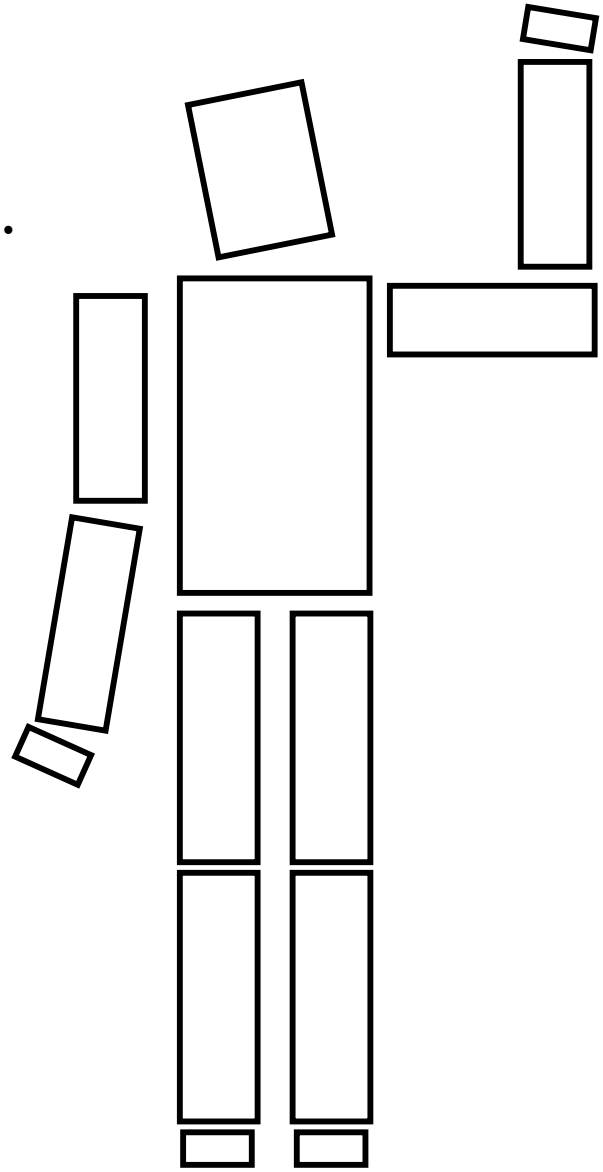
- torso
  - head
  - right arm
    - upper arm
      - lower arm
        - hand
  - left arm
    - upper arm
      - lower arm
        - hand
  - right leg
    - upper leg
      - lower leg
        - foot
  - left leg
    - upper leg
      - lower leg
        - foot



Skeleton - Hierarchical Representation

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

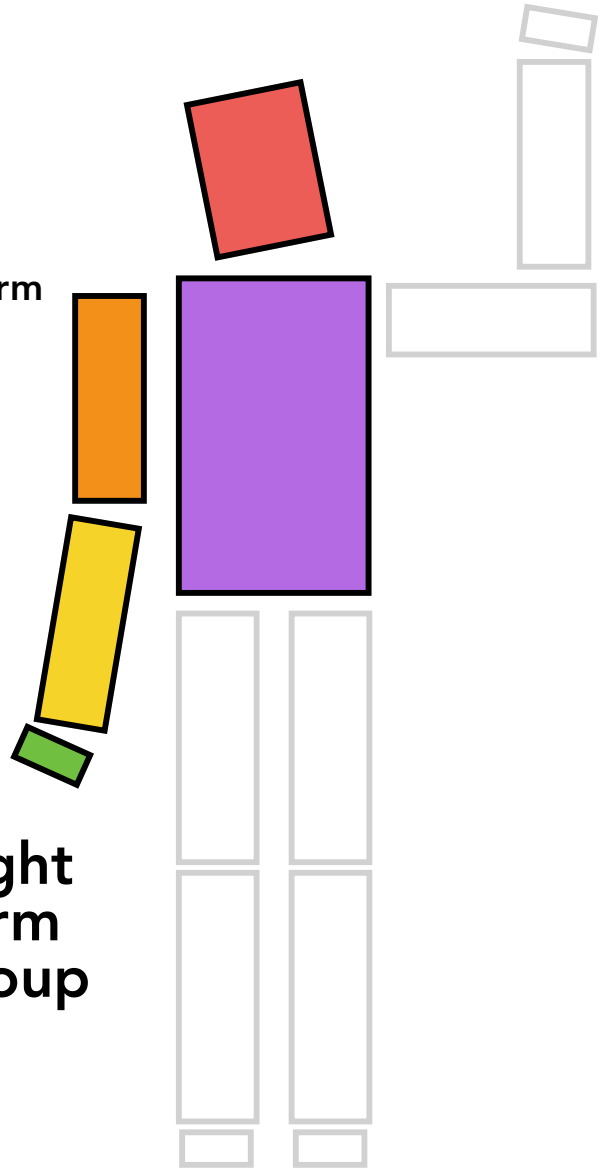
Skeleton - Hierarchical Representation

```
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

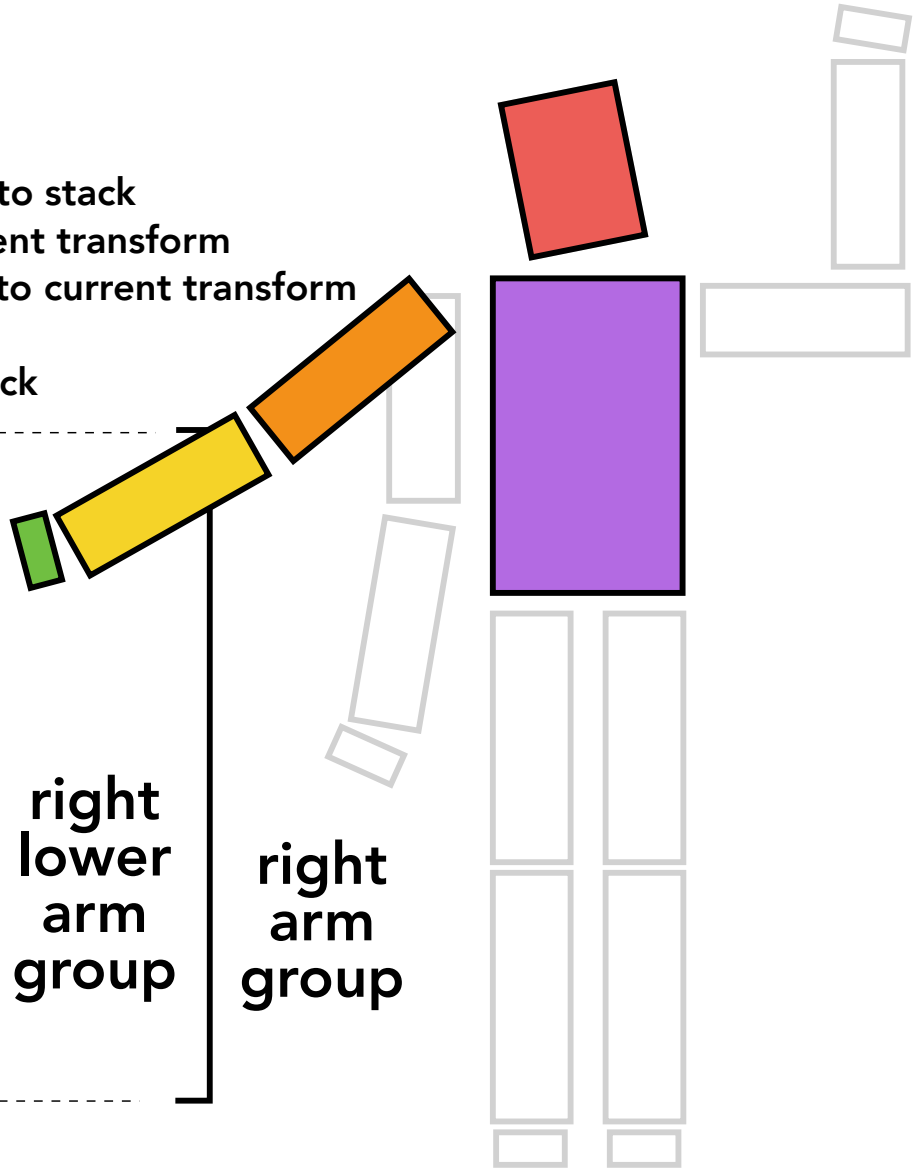
right lower arm group

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();
        ....
```



Today's Topics

Why Study Transformation?

Basic transformations

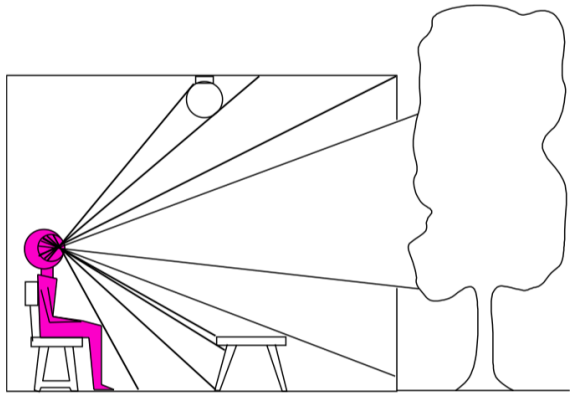
3D transformations

MVP Transformation

View / Camera Transformation

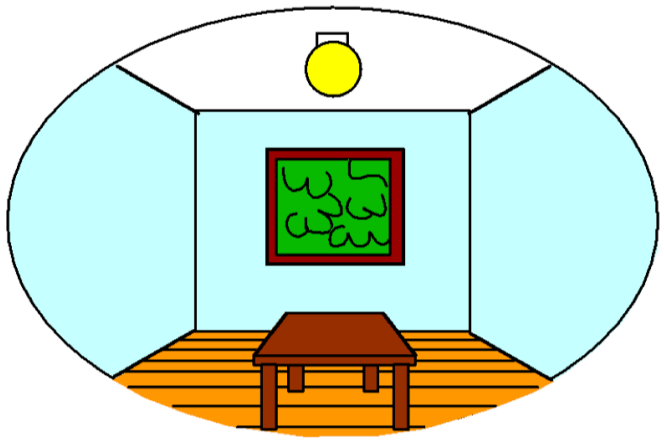
- What is view transformation?
- Think about how to take a photo MVP变换
  - Find a good place and arrange people (Model transformation)
  - Find a good "angle" to put the camera (View transformation)
  - Cheese! (Projection transformation)

3D world

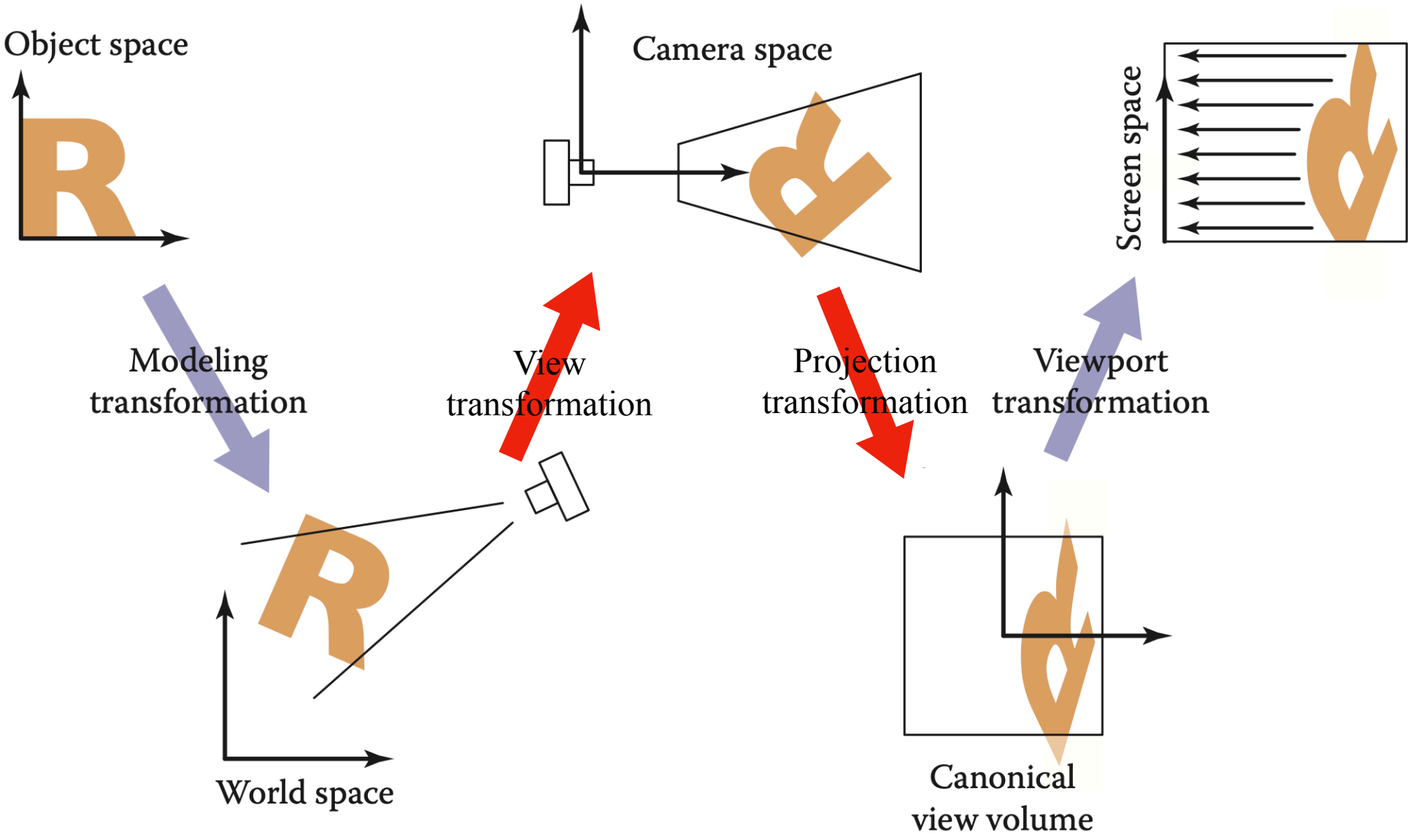


Point of observation

2D image



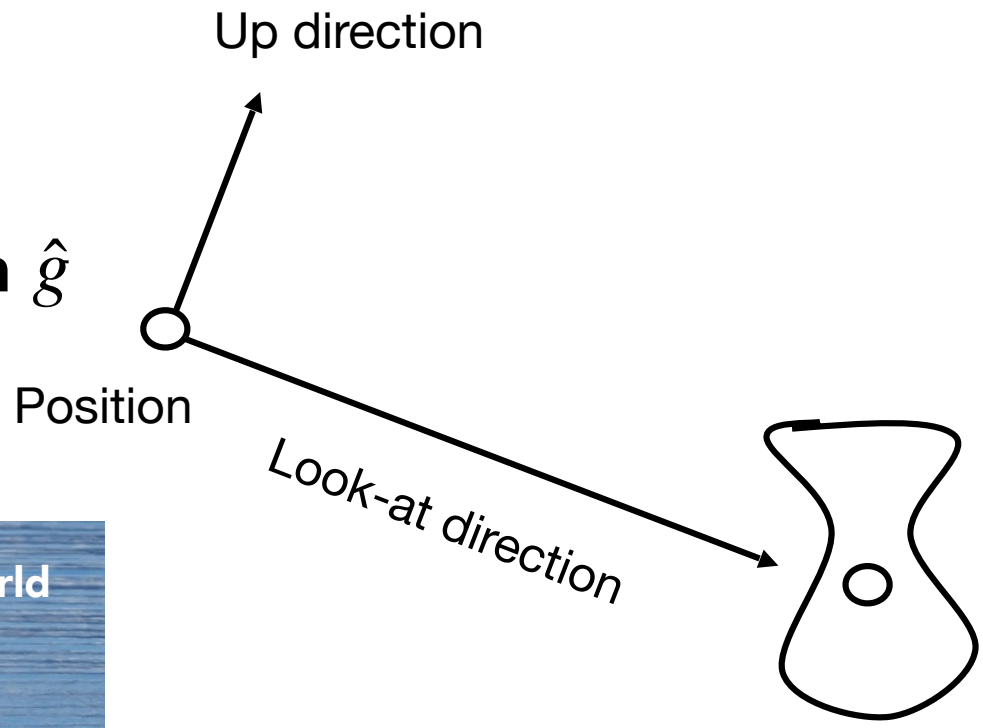
Overview of Transforms

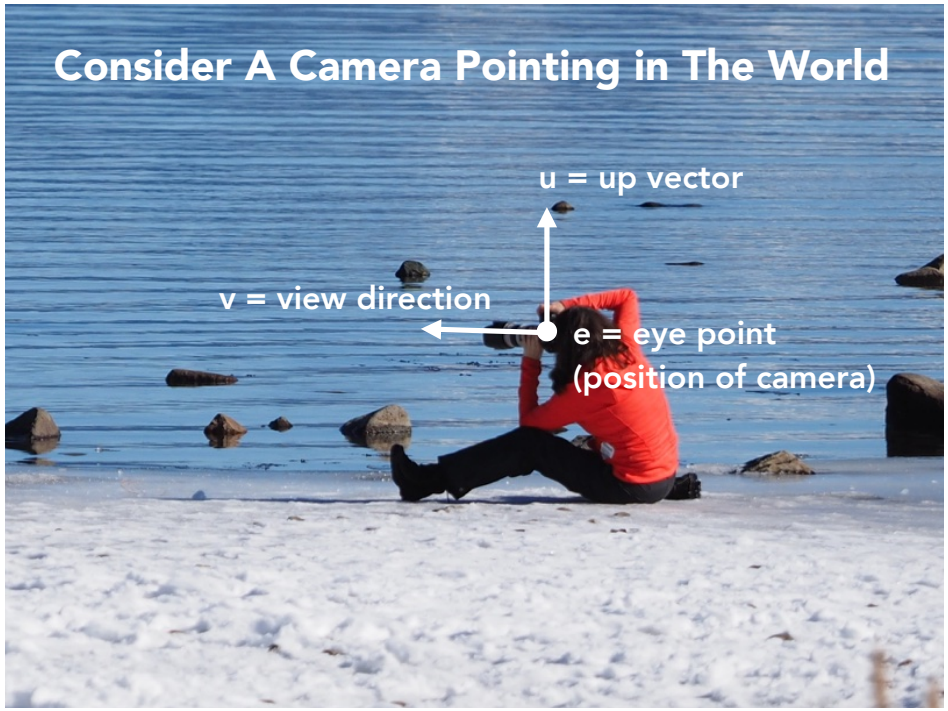


The sequence of spaces and transformations that gets objects from their original coordinates into screen space.

View / Camera Transformation

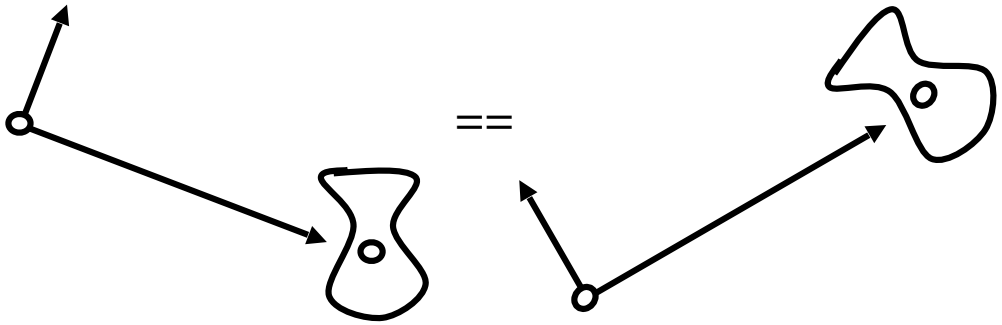
- Define the camera first
  - Position $\vec{e}$
  - Look-at/gaze direction $\hat{g}$
  - Up direction $\hat{t}$





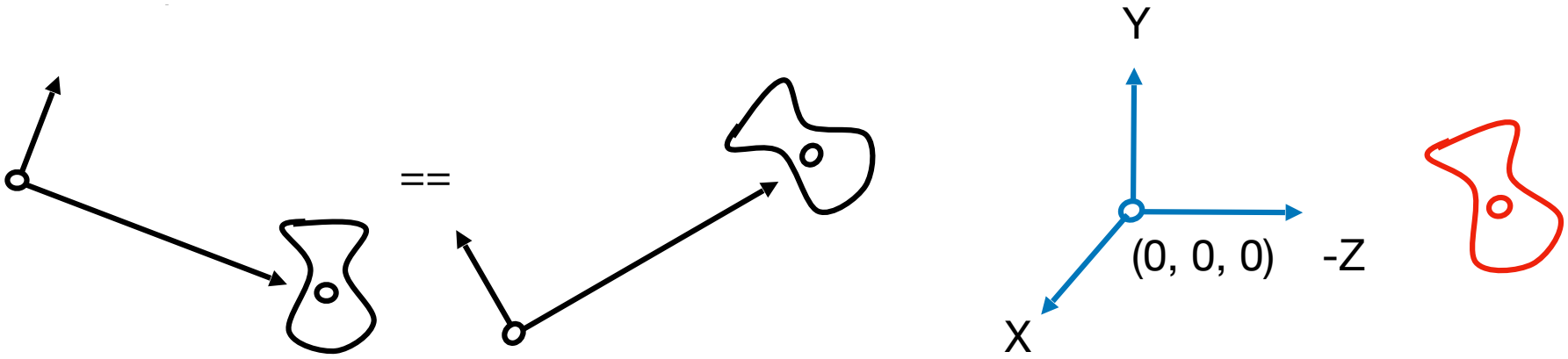
View / Camera Transformation

- Key observation
  - If the camera and all objects move together, the “photo” will be the same



View / Camera Transformation

- Key observation
  - If the camera and all objects move together, the “photo” will be the same

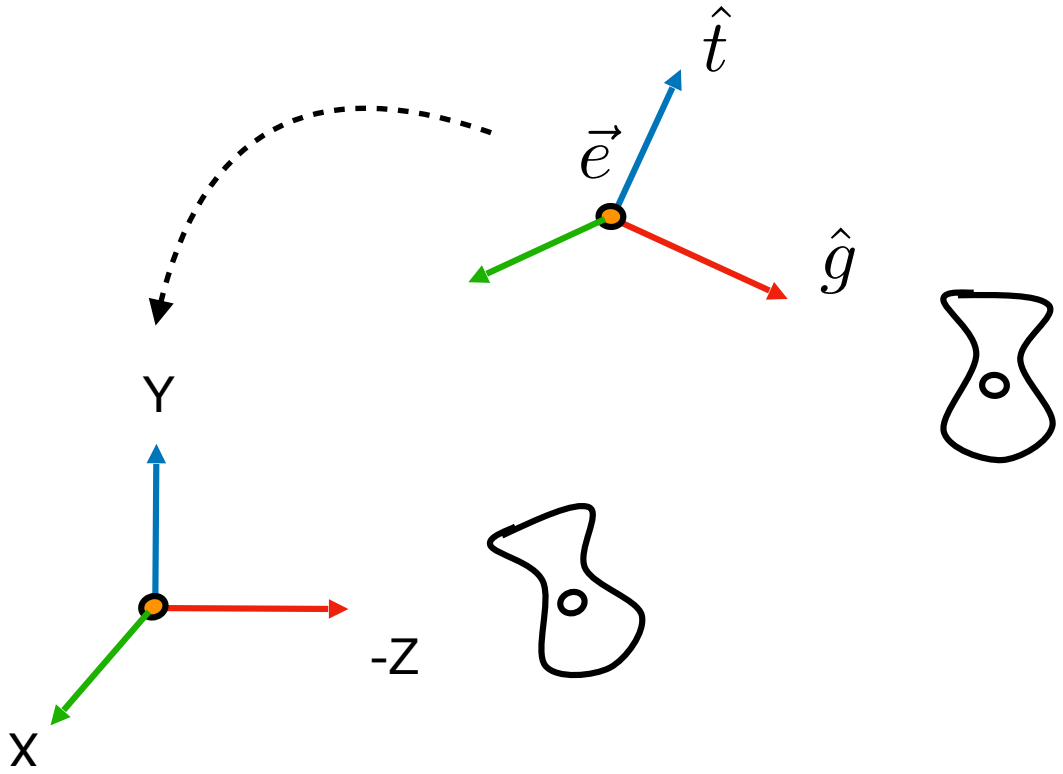


- How about that we always transform the camera to
  - The origin, up at Y, look at -Z
  - And transform the objects along with the camera

View / Camera Transformation

- Transform the camera by M_{view}
  - So it's located at the origin, up at Y, look at -Z

- M_{view} in math?
  - Translate $\vec{e}$ to origin
  - Rotates $\hat{g}$ to -Z
  - Rotates $\hat{t}$ to Y
  - Rotates $(\hat{g} \times \hat{t})$ to X



View / Camera Transformation

- M_{view} in math?
  - Let's write M_{view}=$R_{view}T_{view}$
  - Translate to origin

$$T_{view} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

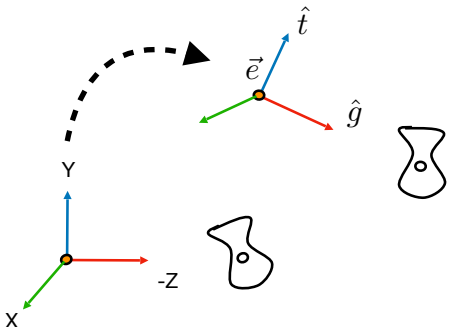
  - Rotates $\hat{g}$ to -Z, $\hat{t}$ to Y, $(\hat{g} \times \hat{t})$ to X
  - Consider its inverse rotation: X to $(\hat{g} \times \hat{t})$, Y to $\hat{t}$, Z to $-\hat{g}$

$$R_{view}^{-1} = \begin{bmatrix} x_{\hat{g}\times\hat{t}} & x_t & x_{-g} & 0 \\ y_{\hat{g}\times\hat{t}} & y_t & y_{-g} & 0 \\ z_{\hat{g}\times\hat{t}} & z_t & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

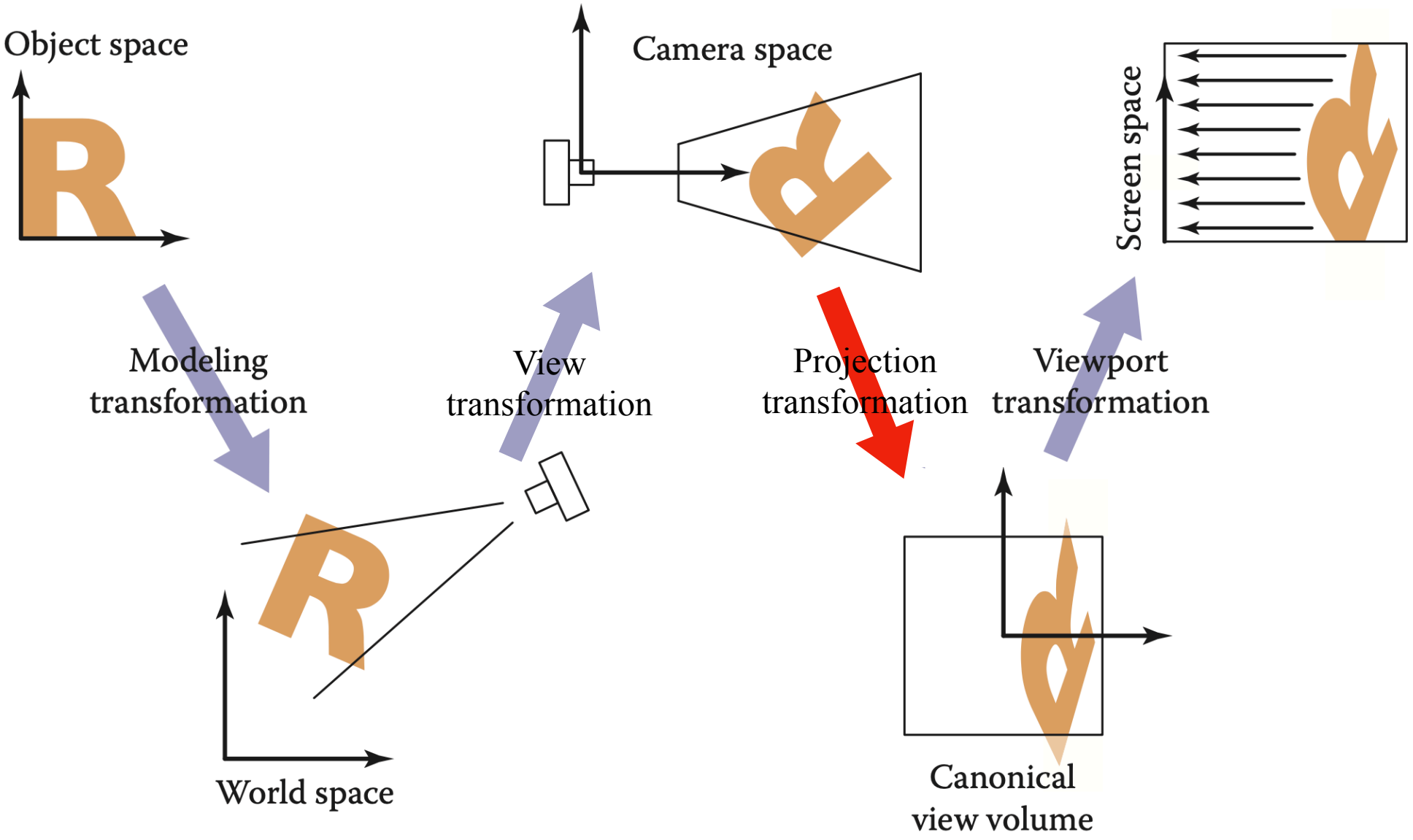
WHY?

$$R_{view} = \begin{bmatrix} x_{\hat{g}\times\hat{t}} & y_{\hat{g}\times\hat{t}} & z_{\hat{g}\times\hat{t}} & 0 \\ x_t & y_t & z_t & 0 \\ x_{-g} & y_{-g} & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{view}^{-1} = R_{view}^{T}$$



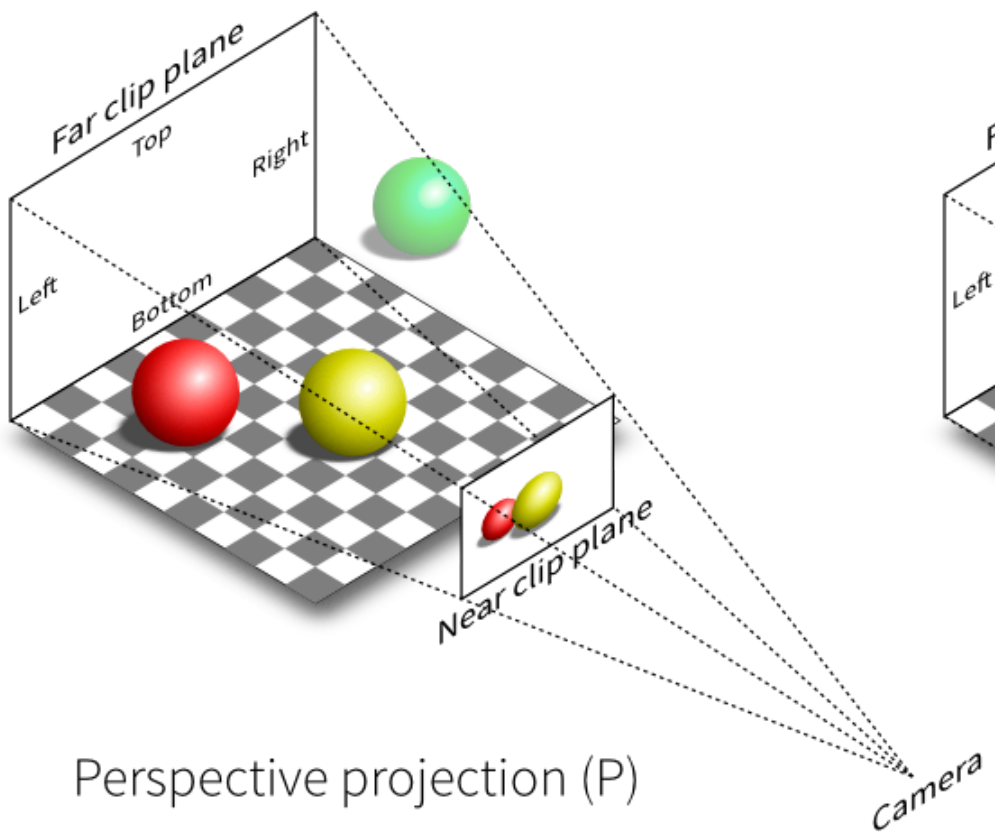
Overview of Transforms



The sequence of spaces and transformations that gets objects from their original coordinates into screen space.

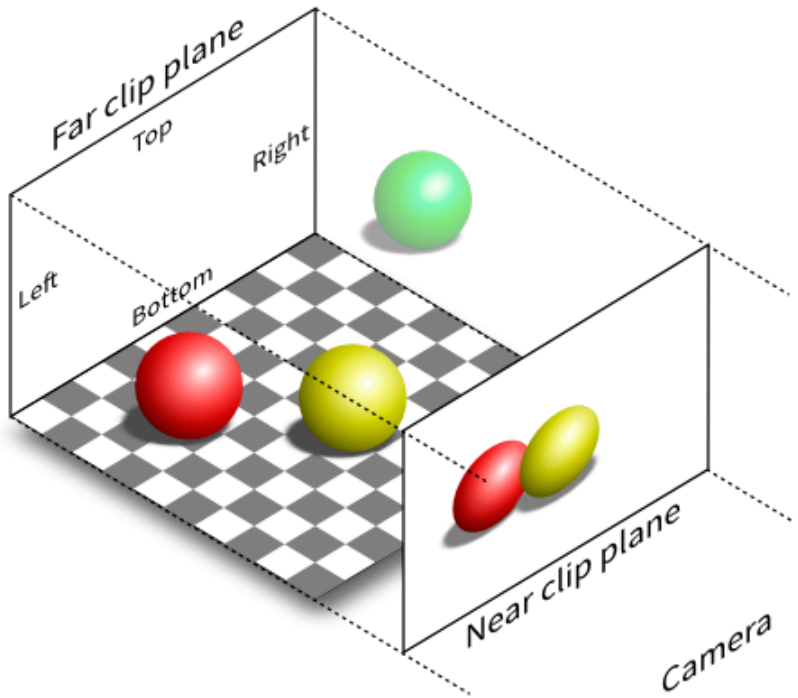
Projection Transformation

- Perspective projection vs. orthographic projection



Perspective projection (P)

透视投影

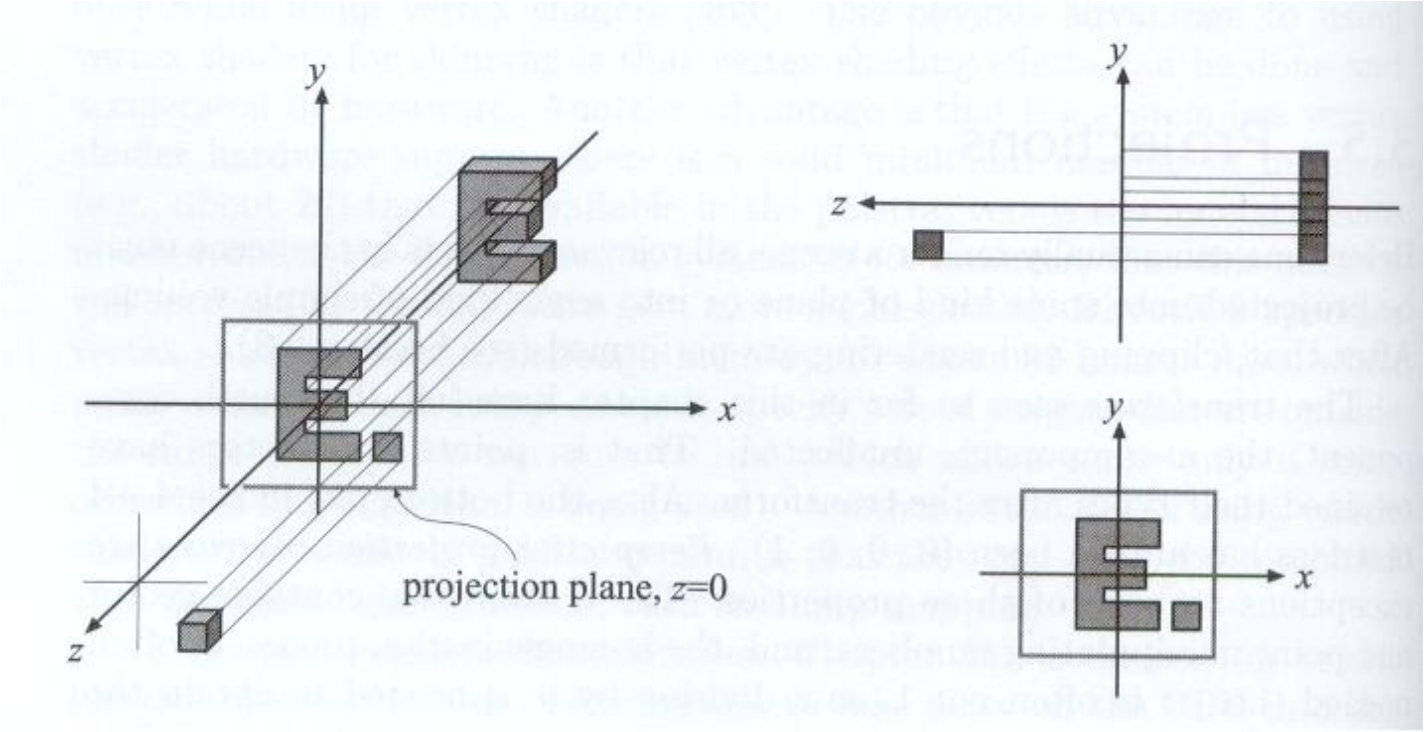


Orthographic projection (O)

正交投影

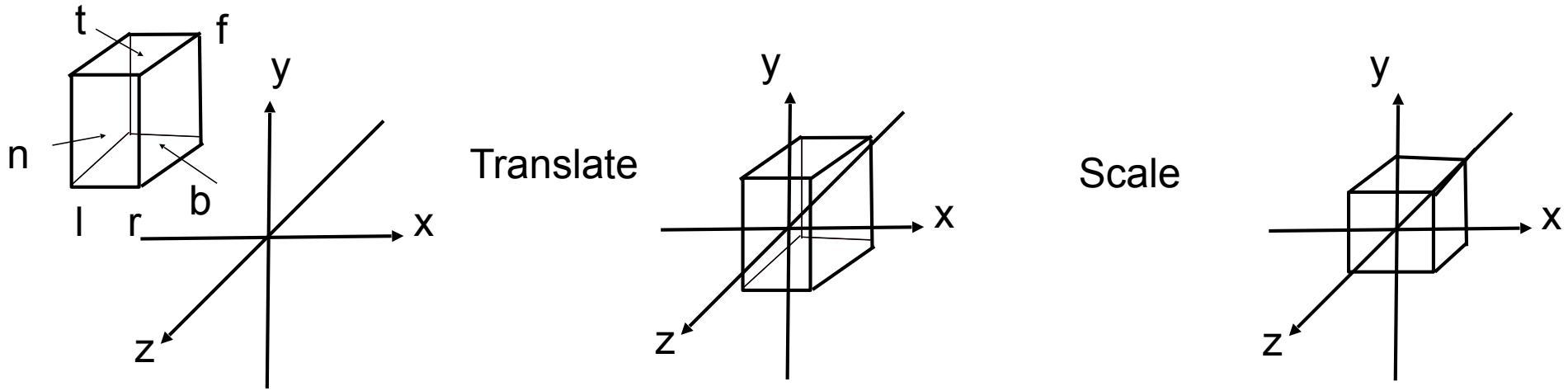
Orthographic Projection

- A simple way of understanding
  - Camera located at origin, looking at -Z, up at Y
  - Drop Z coordinate
  - Translate and scale the resulting rectangle to $[-1,1]^2$



Orthographic Projection

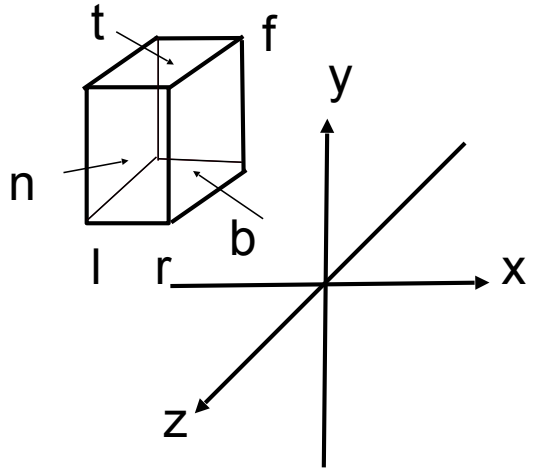
- In general
  - We want to map a cuboid ($[l,r]\times[b,t]\times[f,n]$) to the "canonical" cube $[-1,1]^3$



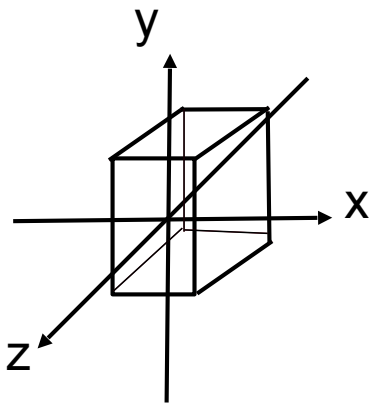
Orthographic Projection

- Transformation matrix?
  - Translate (cuboid center to origin) first, then scale (length/width/height to 2)

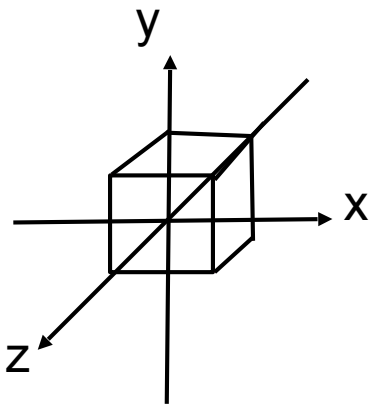
$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Translate

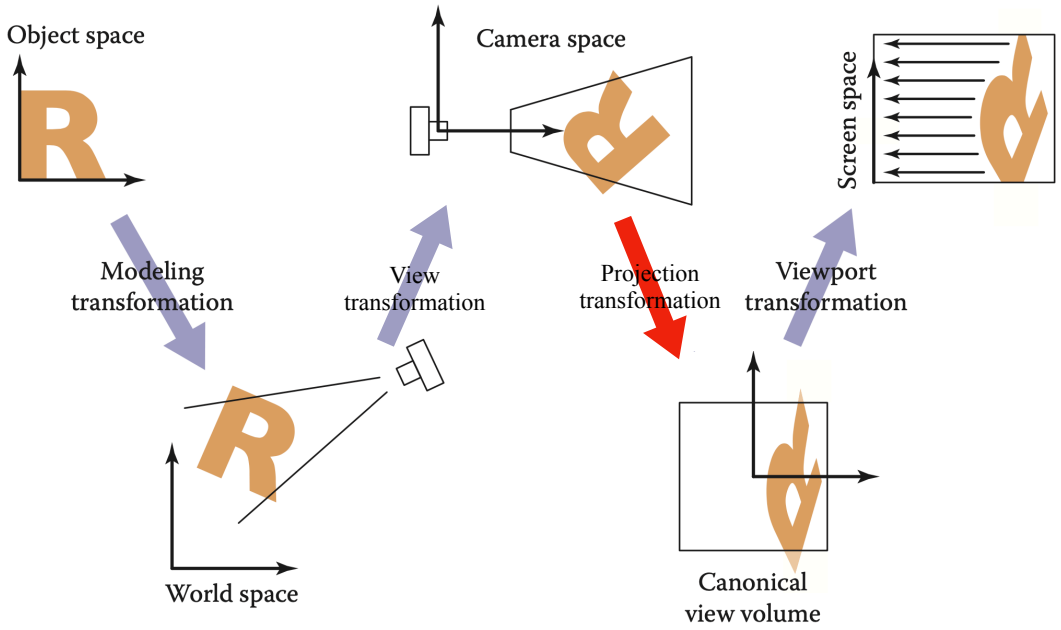


Scale



Orthographic Projection

- map a cuboid ([l,r]×[b,t]×[f,n]) to the "canonical" cube $[-1,1]^3$



Q: Why not directly map the cuboid to the 2D square?

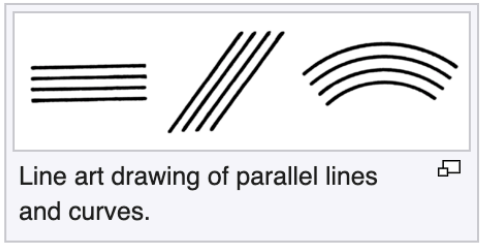
A: The z-coordinate will now be in [−1, 1]. We don't take advantage of this now, but it will be useful when we examine z-buffer algorithms.

Perspective Projection

- Most common in Computer Graphics, art, visual system
- Further objects are smaller
- Parallel lines not parallel; converge to a single point

Perspective Projection

- Euclid was wrong??!!

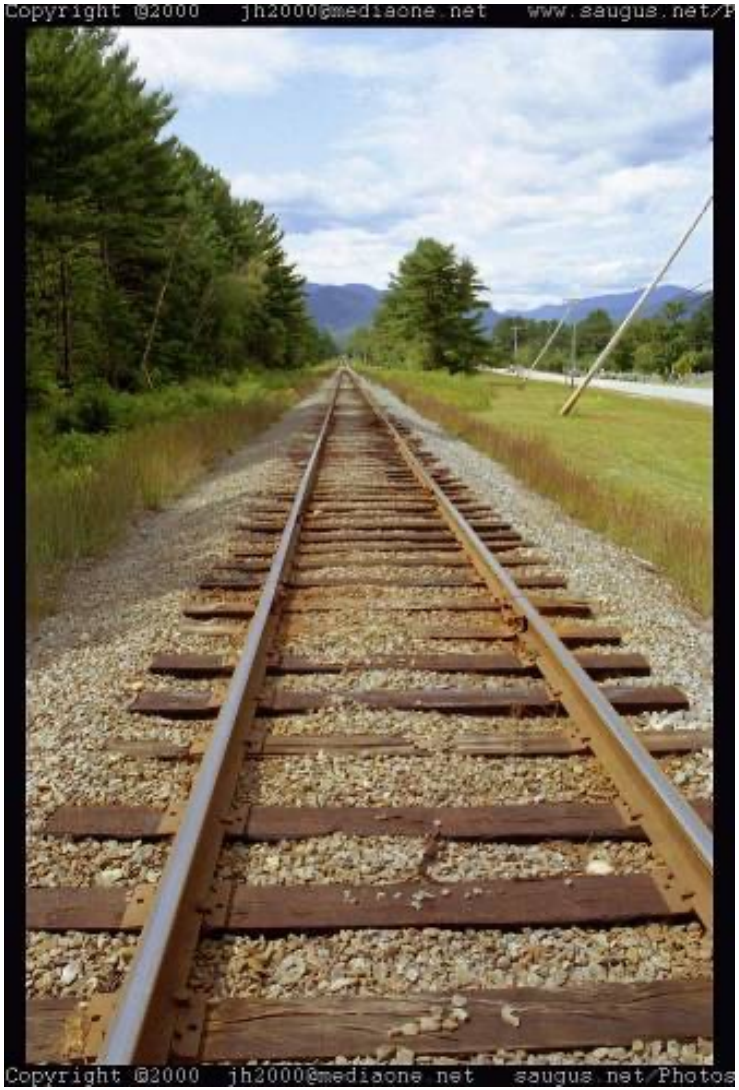


Line art drawing of parallel lines and curves.

In geometry, **parallel** lines are lines in a plane which do not meet; that is, two lines in a plane that do not intersect or touch each other at any point are said to be parallel. By extension, a line and a plane, or two planes, in three-dimensional Euclidean space that do not share a point are said to be parallel. However, two lines in three-dimensional space which do not meet must be in a common plane to be considered parallel; otherwise they are called skew lines. Parallel planes are planes in the same three-dimensional space that never meet.

Parallel lines are the subject of Euclid's parallel postulate.[1] Parallelism is primarily a property of affine geometries and Euclidean geometry is a special instance of this type of geometry. In some other geometries, such as hyperbolic geometry, lines can have analogous properties that are referred to as parallelism.

https://en.wikipedia.org/wiki/Parallel_(geometry)

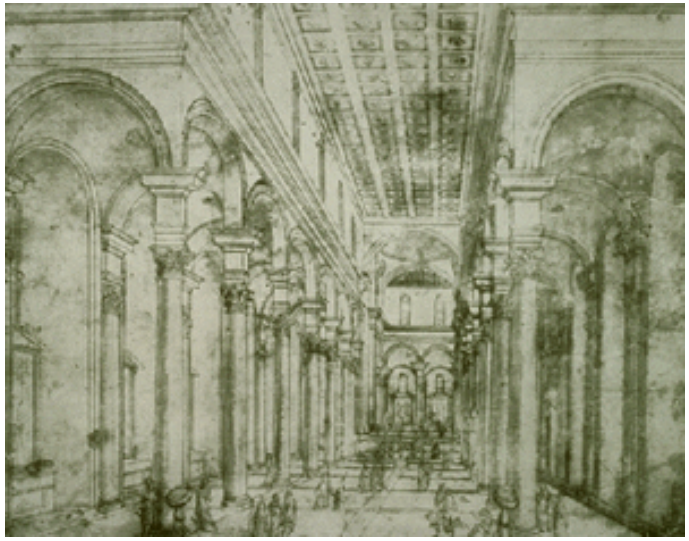


Example: Incorrect Perspective Drawing

Carolingian painting from the 8-9th century

Example: Towards correct Perspective Drawing

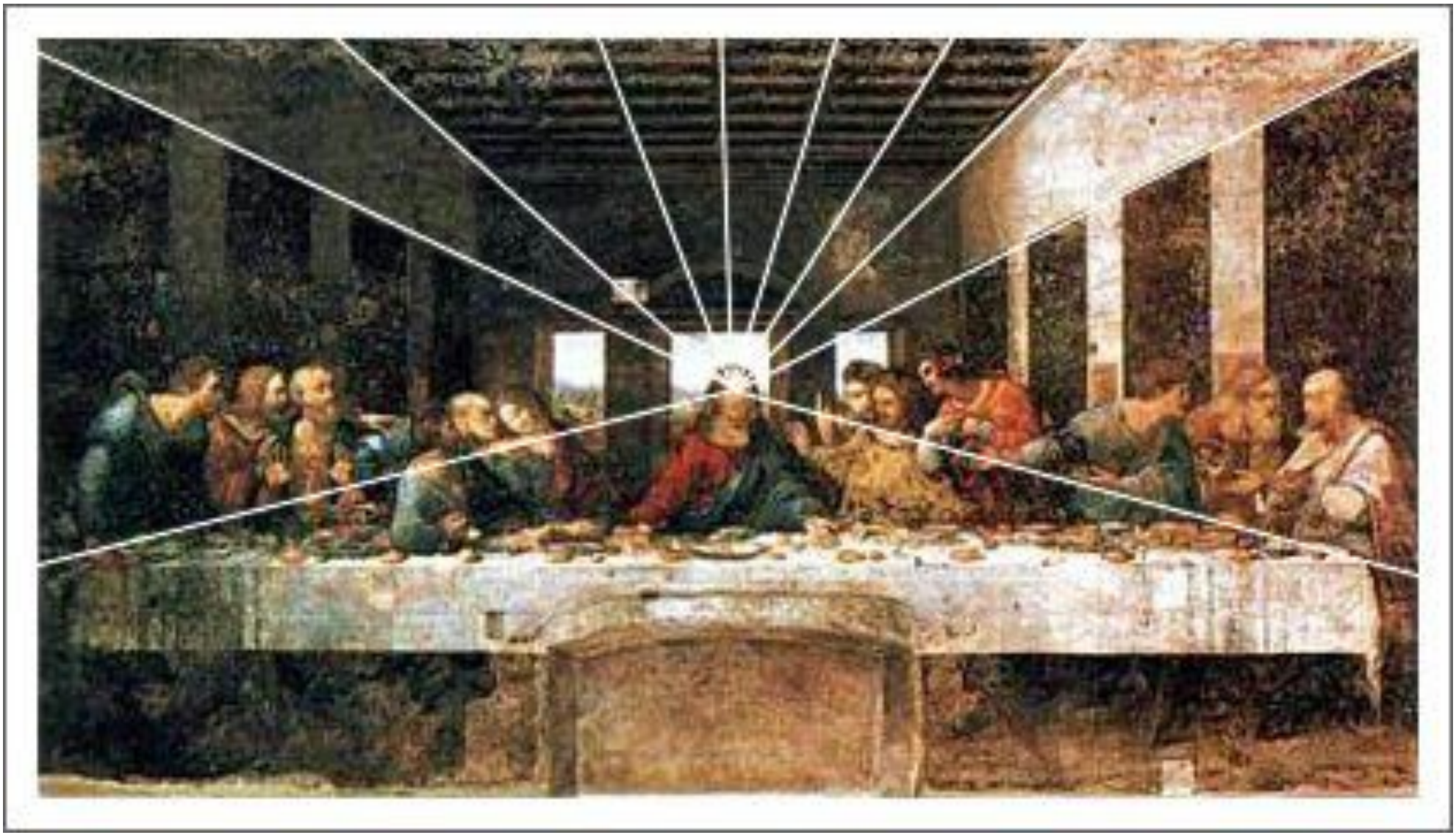
Ambrogio Lorenzetti
Annunciation, 1344



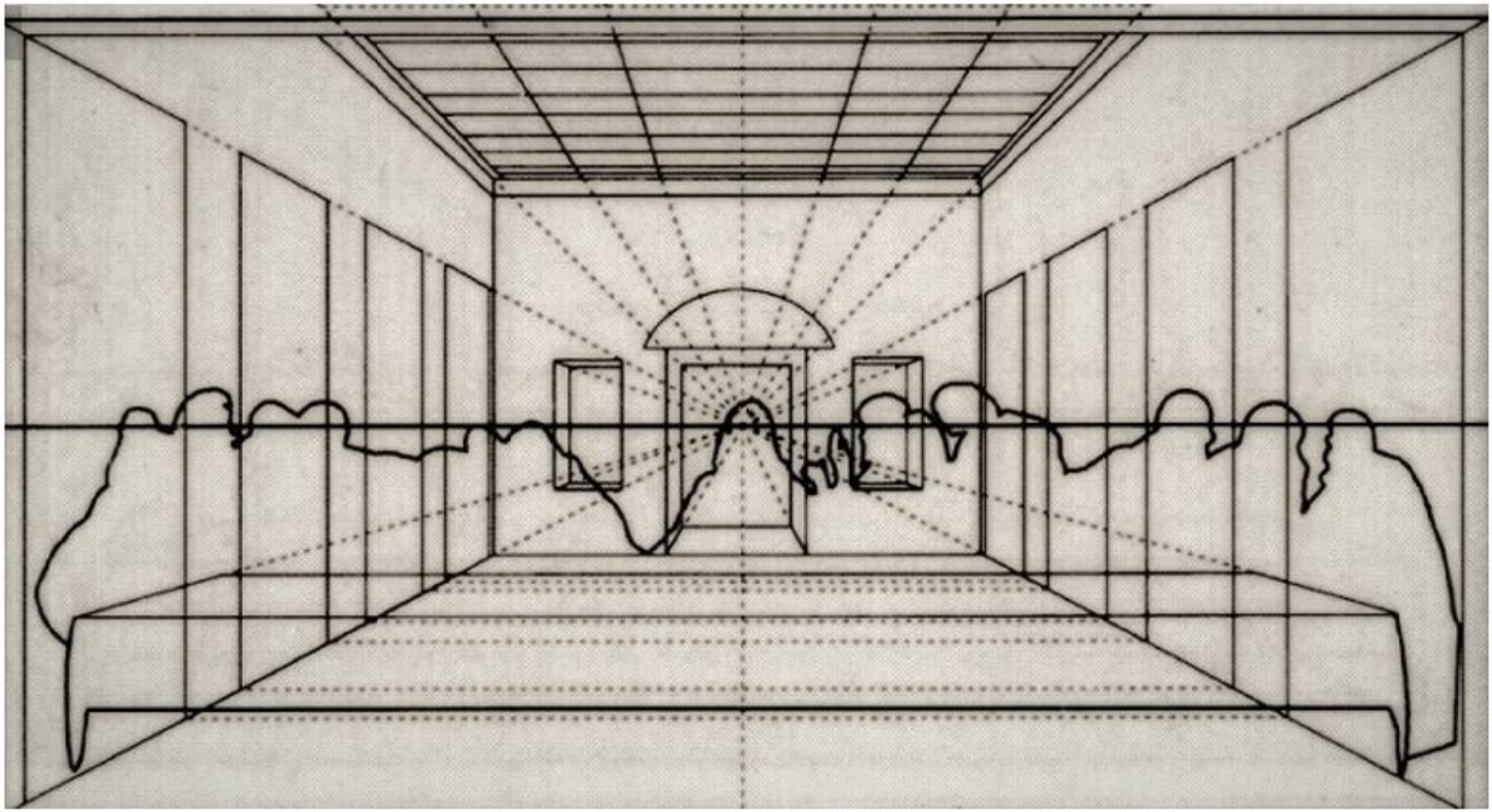
Brunelleschi, elevation of Santo Spirito,
1434-83, Florence

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

Example: Perspective Drawing



Example: Perspective Lines



Pinhole Camera Model

Pinhole Camera

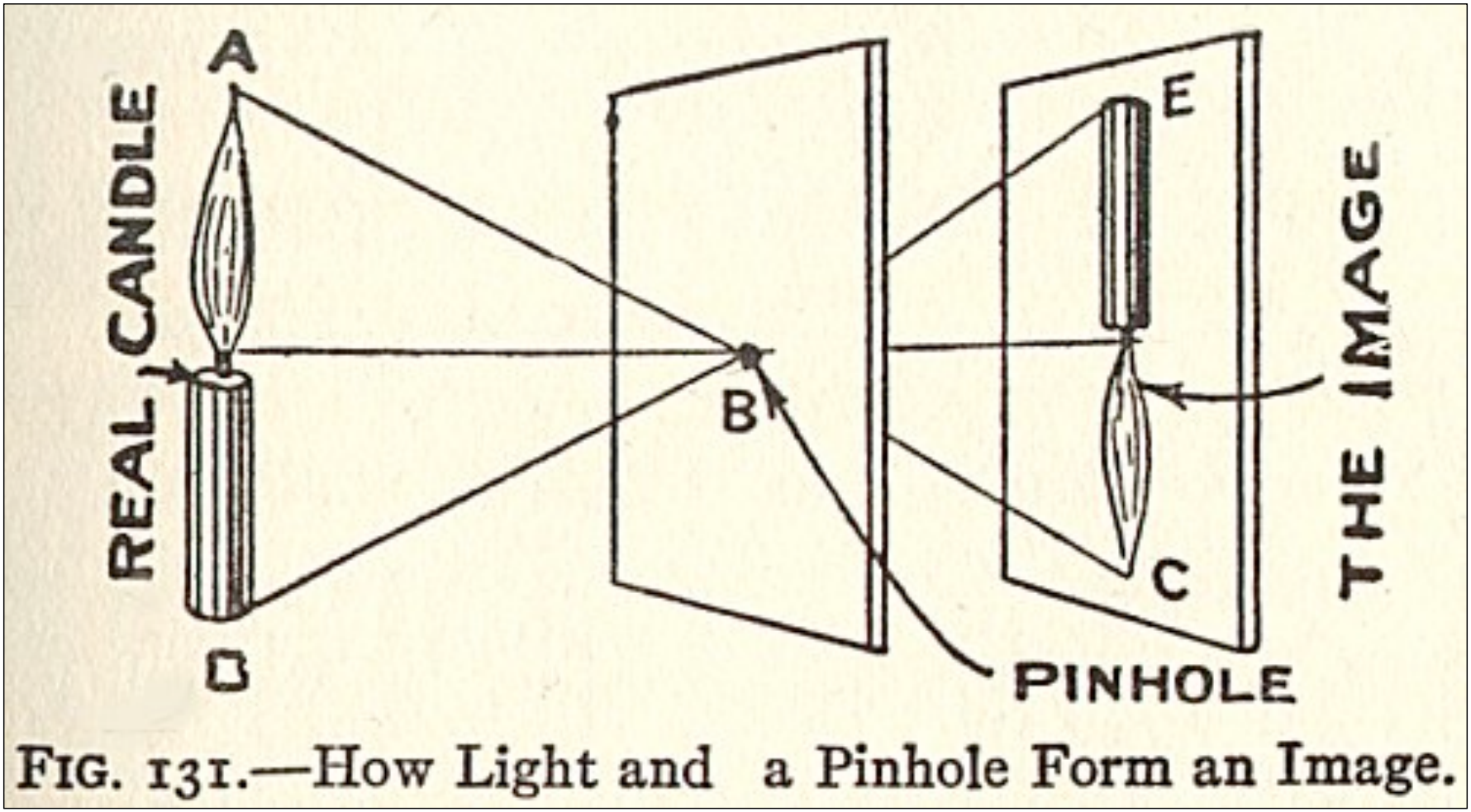
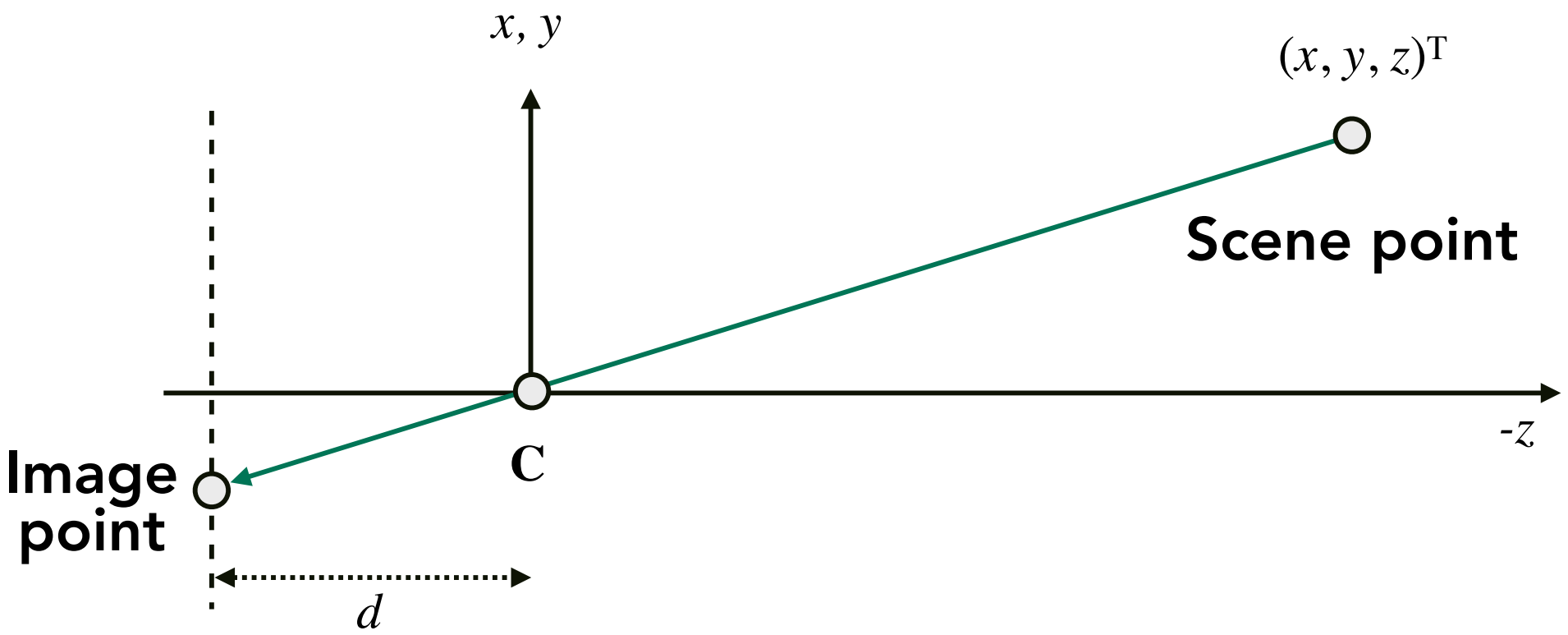


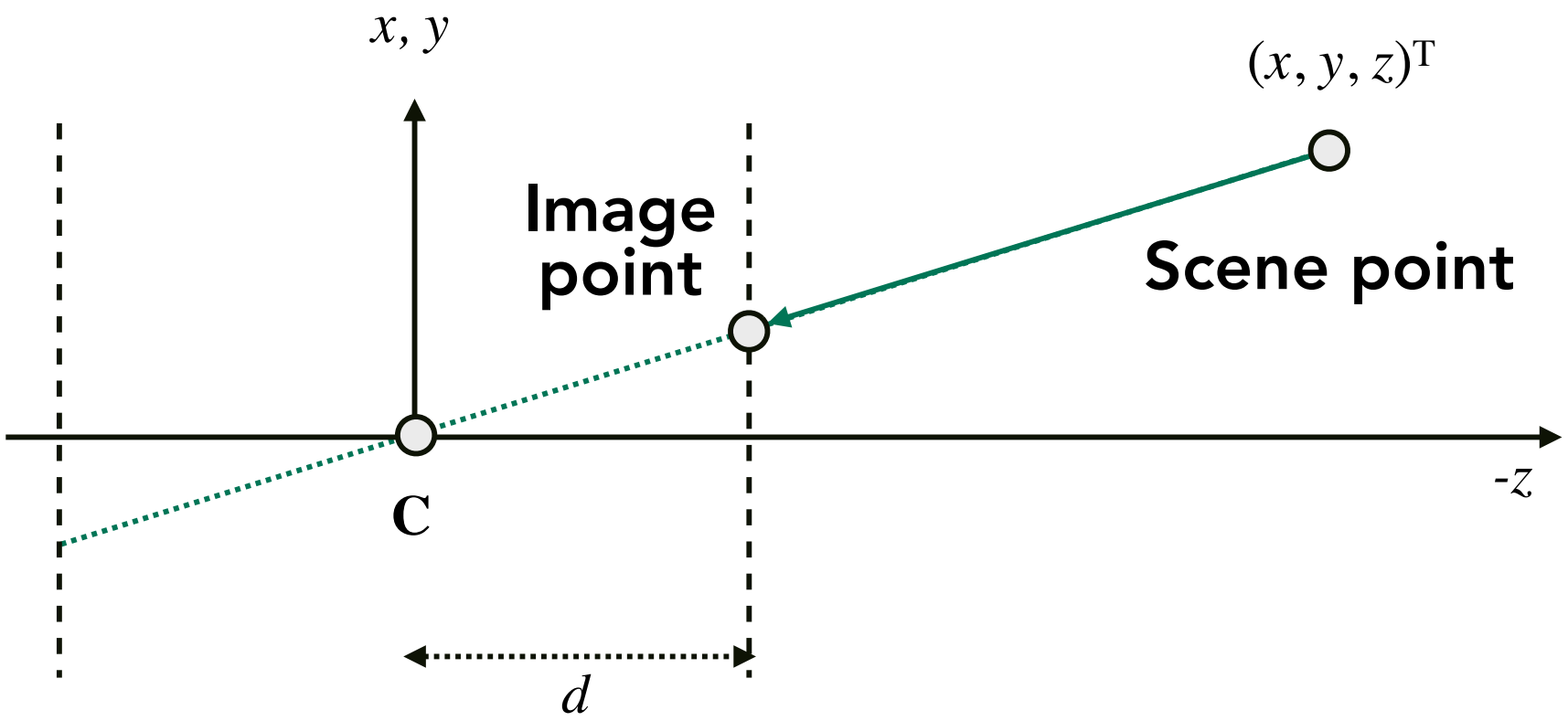
FIG. 131.—How Light and a Pinhole Form an Image.

Projective Transform



Inverted image (as in real pinhole camera)

Pinhole Camera Projective Transform

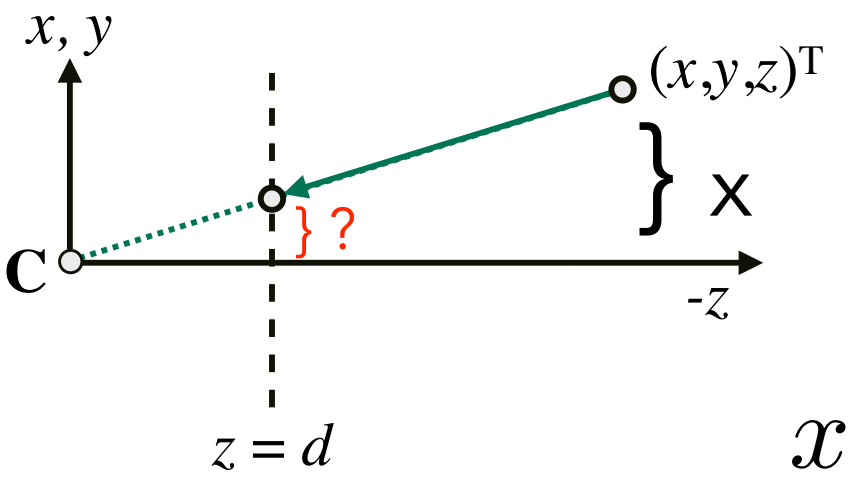


Upright image

Projective Transforms

Standard perspective projection

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



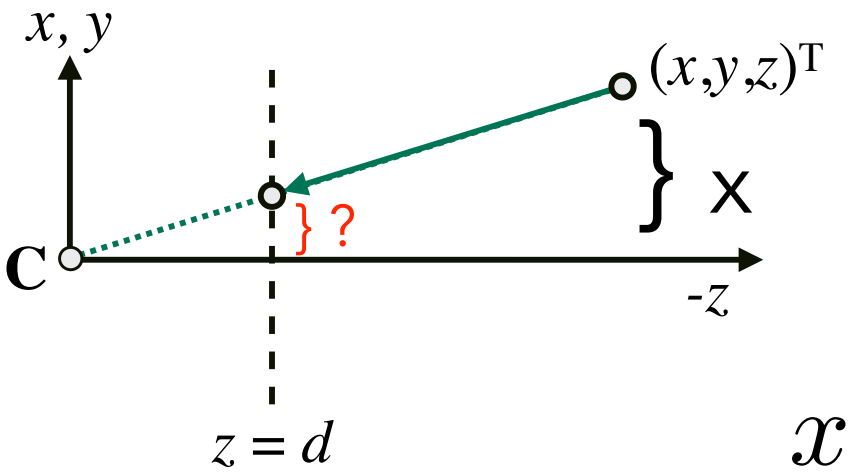
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \\ \\ d \end{pmatrix}$$

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

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 (objects shrink in distance, 解释了为什么近大远小)
- 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.

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

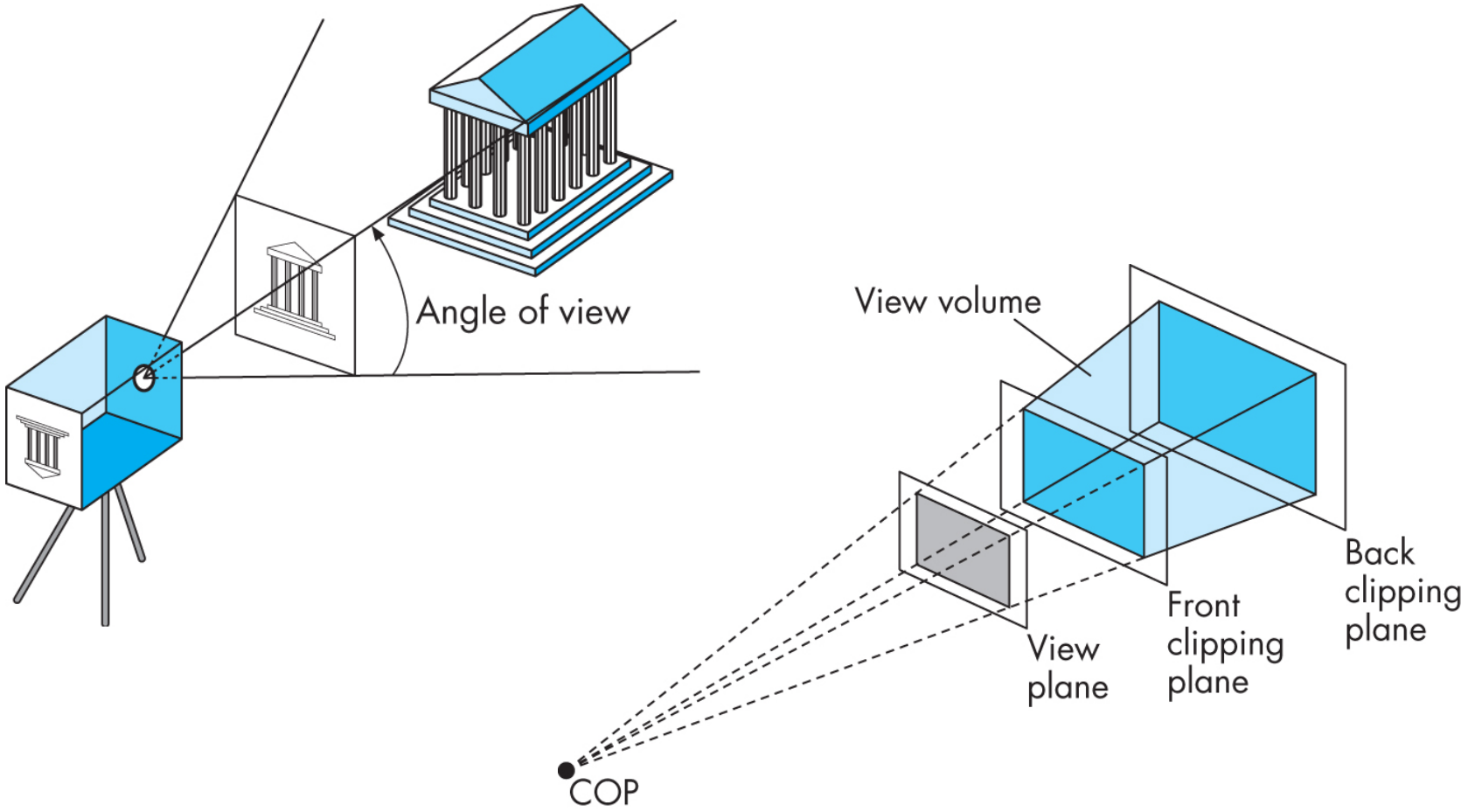
M将物体投影到相机坐标系下的相机投影平面上。

Perspective Projection

- In general
  - Similar to orthographic projection that maps a cuboid ($[l,r]\times[b,t]\times[f,n]$) to the "canonical" cube $[-1,1]^3$, for perspective projection, we want to maps a frustum to the "canonical" cube $[-1,1]^3$, not 2D plane.

Specifying Real Camera Perspectives

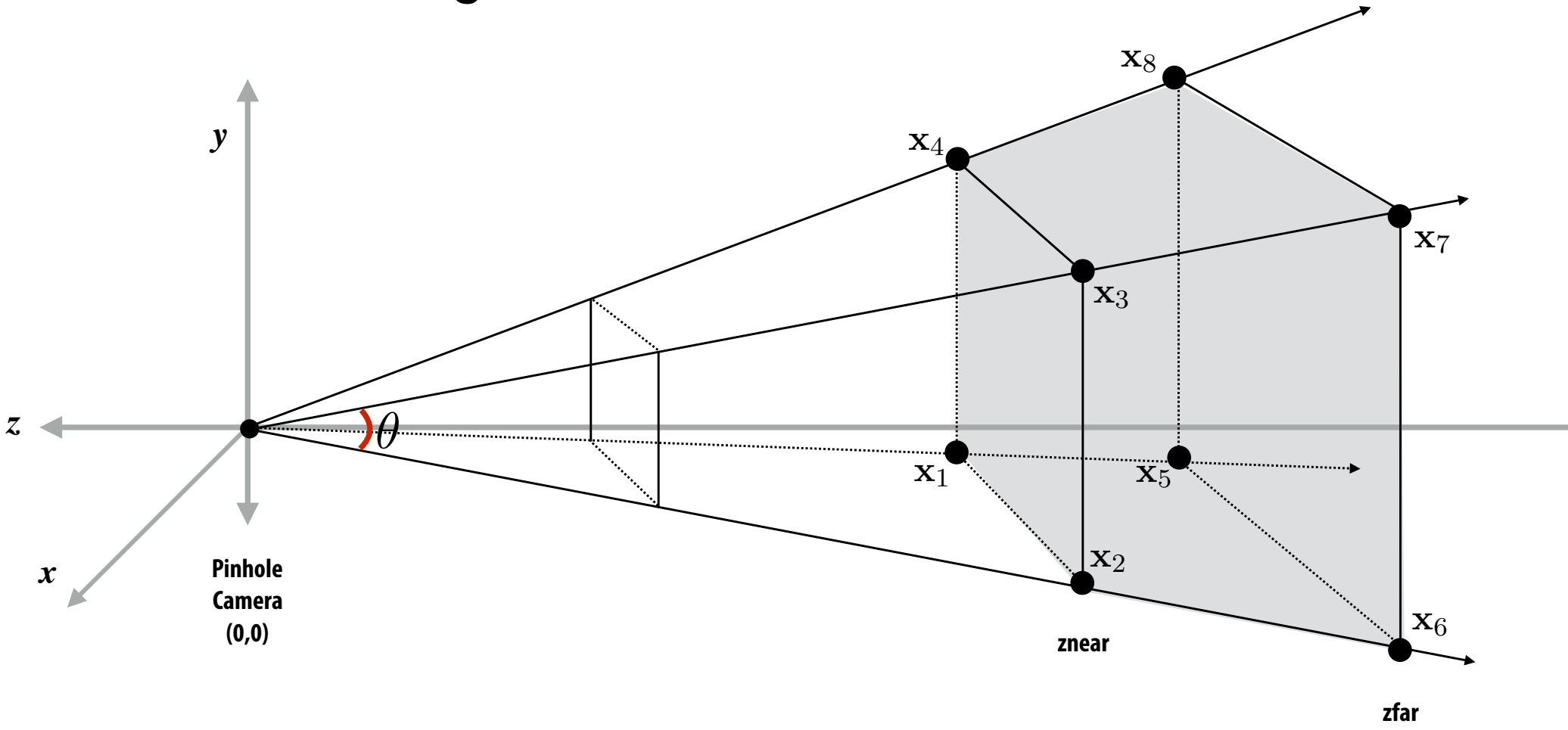
Specifying Perspective Projection



From Angel and Shreiner, Interactive Computer Graphics

View Frustum in Camera Space

View frustum is region the camera can see:



- Top/bottom/left/right planes correspond to the sides of screen
- Near/far planes correspond to closest/furthest thing we want to draw

Specifying Perspective Viewing Volume

Parameterized by

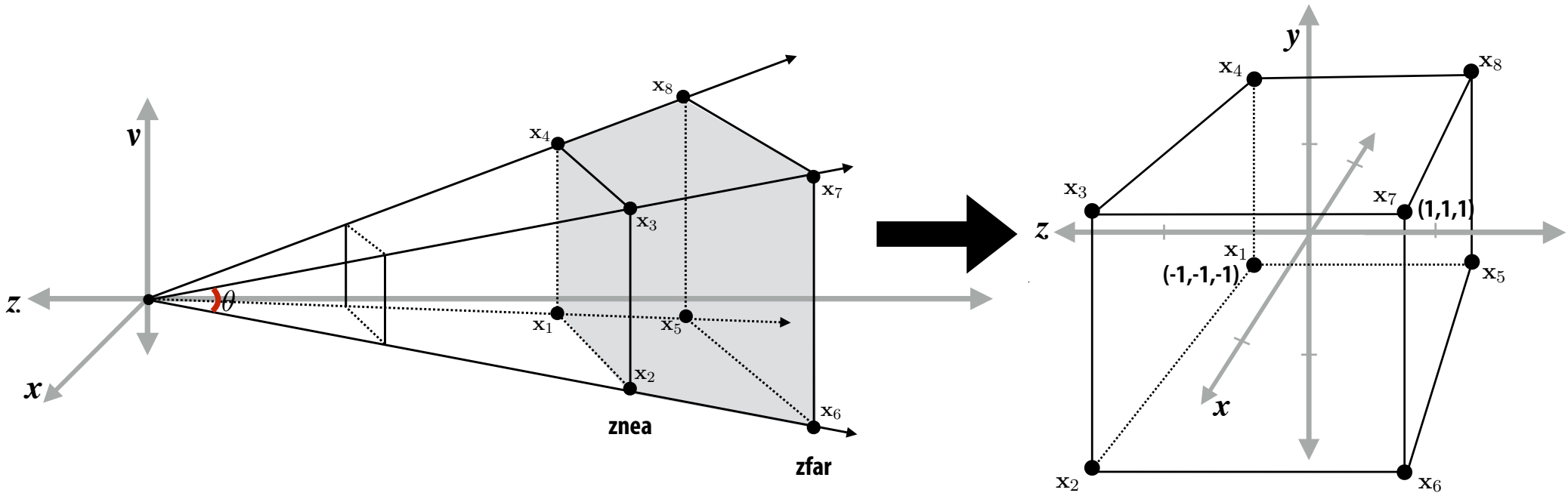
- fovy : vertical angular field of view
- aspect ratio : width / height of field of view
- near : depth of near clipping plane
- far : depth of far clipping plane

Derived quantities

- top = near * tan(fovy/2)
- bottom = – top
- right = top * aspect
- left = – right

Mapping frustum to cube

Before mapping to 2D, map corners of frustum to corners of cube:



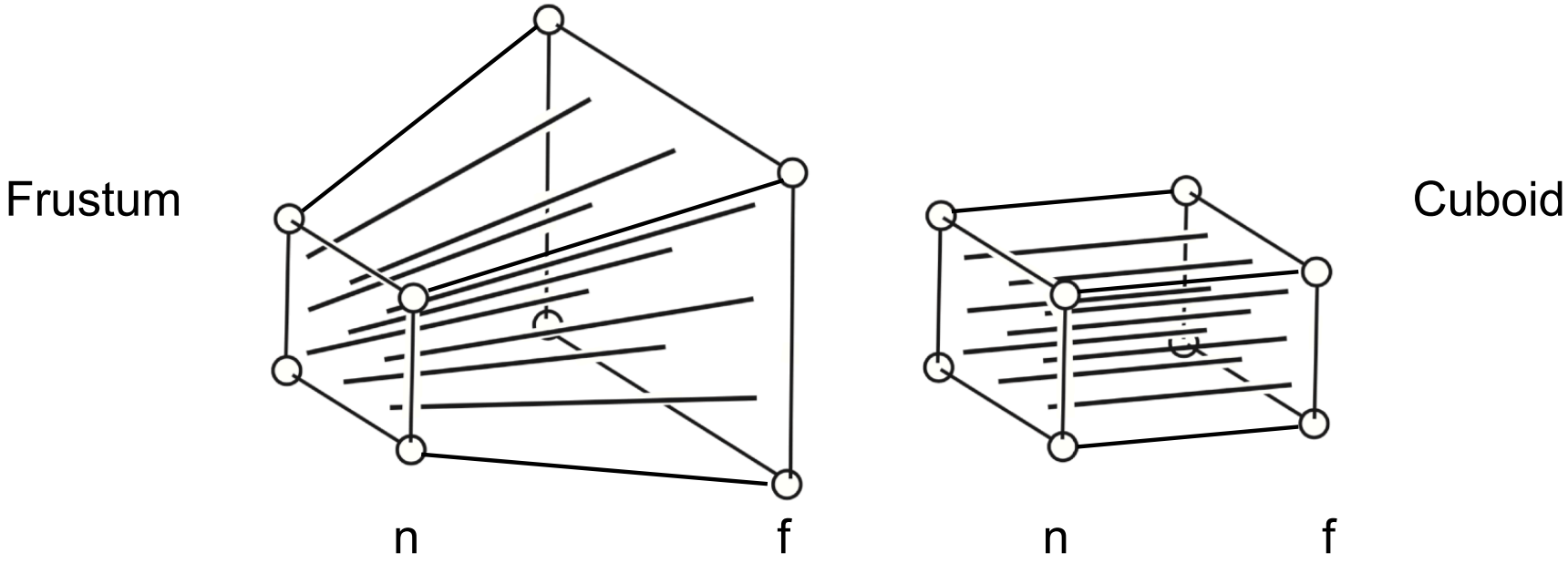
Camera Coordinates

Normalized Device Coords "NDC"

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

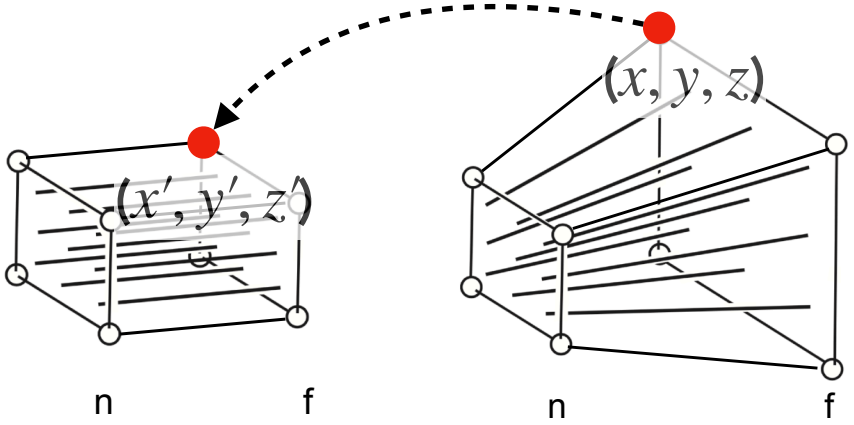
Perspective Projection

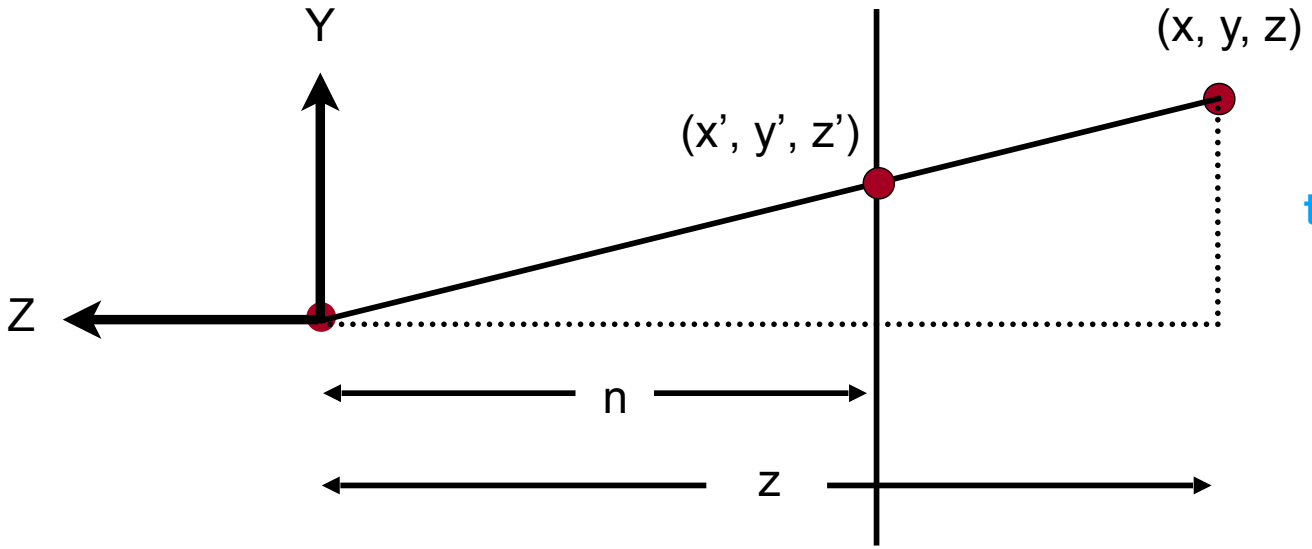
- How to do perspective projection
  - First "squish" the frustum into a cuboid ($M_{persp \to ortho}$)
  - Do orthographic projection (M_{ortho})



Perspective Projection

- In order to find a transformation
  - Recall the key idea: Find the relationship between transformed point (x', y', z') and the original points (x, y, z)





similar triangle

$$y' = \frac{n}{z} y$$

Perspective Projection

- In order to find a transformation
  - Recall the key idea: Find the relationship between transformed point (x', y', z') and the original points (x, y, z)

$$y' = \frac{n}{z} y \qquad x' = \frac{n}{z} x \text{ (similar to y')}$$

- In homogeneous coordinates,

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} nx/z \\ ny/z \\ \text{unknown} \\ 1 \end{pmatrix} \overset{\text{mult. by z}}{==} \begin{pmatrix} nx \\ ny \\ \text{still unknown} \\ z \end{pmatrix}$$

Perspective Projection

- So the "squish" (persp to ortho) projection does this

$$M^{(4\times4)}_{persp\to ortho}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix}$$

- Already good enough to figure out part of $M_{persp\to ortho}$

$$M_{persp\to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

WHY?

Perspective Projection

- How to figure out the third row of $M_{persp \to ortho}$
  - Any information that we can use?

$$M_{persp \to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- Observation: the third row is responsible for z'
  - Any point on the near plane will not change
  - Any point on the far plane will not change

Perspective Projection

- Any point on the near plane will not change

$$M^{(4\times4)}_{persp\to ortho}\begin{pmatrix}x\\y\\z\\1\end{pmatrix}=\begin{pmatrix}nx\\ny\\\text{unknown}\\z\end{pmatrix}$$

replace z with n

$$\begin{pmatrix}x\\y\\n\\1\end{pmatrix}\Rightarrow\begin{pmatrix}x\\y\\n\\1\end{pmatrix}==\begin{pmatrix}nx\\ny\\n^2\\n\end{pmatrix}$$

- So the third row must be the form (0 0 A B)

$$\begin{pmatrix}0 & 0 & A & B\end{pmatrix}\begin{pmatrix}x\\y\\n\\1\end{pmatrix}=n^2$$

n^2 has nothing to do with x and y

Perspective Projection

- What do we have now?

$$\begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = n^2 \quad \Rightarrow \quad An + B = n^2$$

- Any point's z on the far plane will not change
  - Similar to near plane

$$Af + B = f^2$$

Perspective Projection

- Solve for A and B

$$An + B = n^2$$
$$Af + B = f^2$$

$$A = n + f$$
$$B = -nf$$

- Finally, every entry in $M_{persp \to ortho}$ is known!

$$M_{persp \to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Matrix for Perspective Transform

- What's next?
  - Do orthographic projection (M_{ortho}) to finish
  - $M_{persp} = M_{ortho} M_{persp \to ortho}$

$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M_{persp \to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$M_{persp} = M_{ortho} M_{persp \to ortho} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Matrix for Perspective Transform

What does this do to z-values?

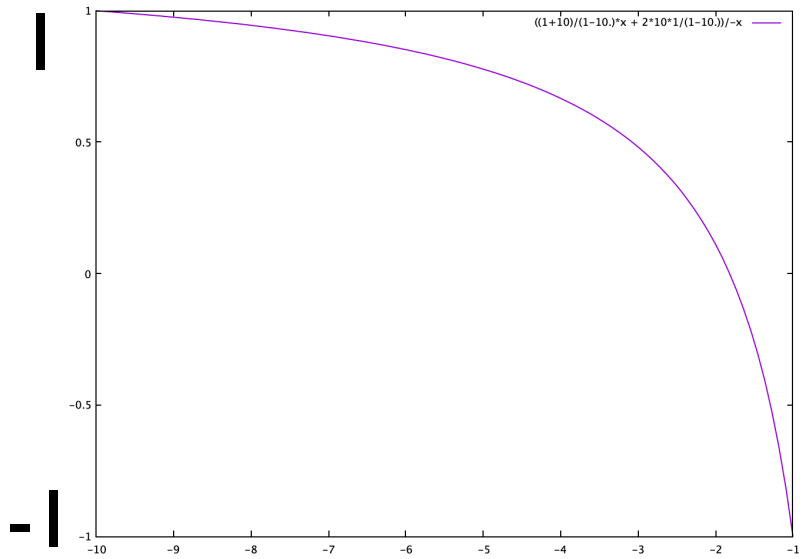
$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In equation form:

$$z \mapsto \frac{f+n}{f-n} + \frac{1}{z}\frac{2fn}{f-n}$$

当z'=1时，$z=-n$，此时点刚好位于视锥体的近平面处。
当z'=-1时，$z=-f$，此时点刚好位于视锥体的远平面处。

Graphically:

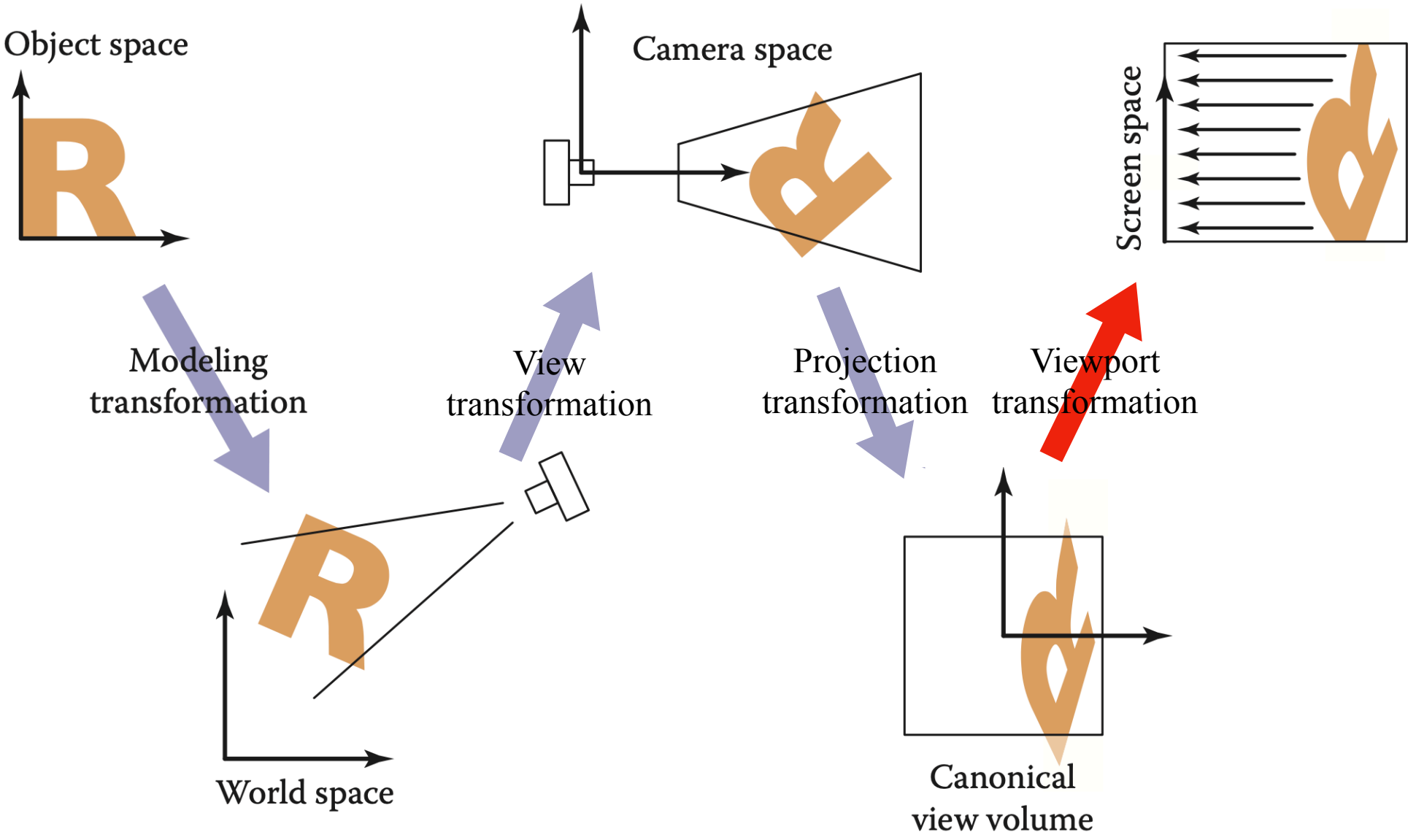


(Why are these z values negative?)

图中可见z'与z是非线性关系。

透视变换不是线性变换！

Overview of Transforms



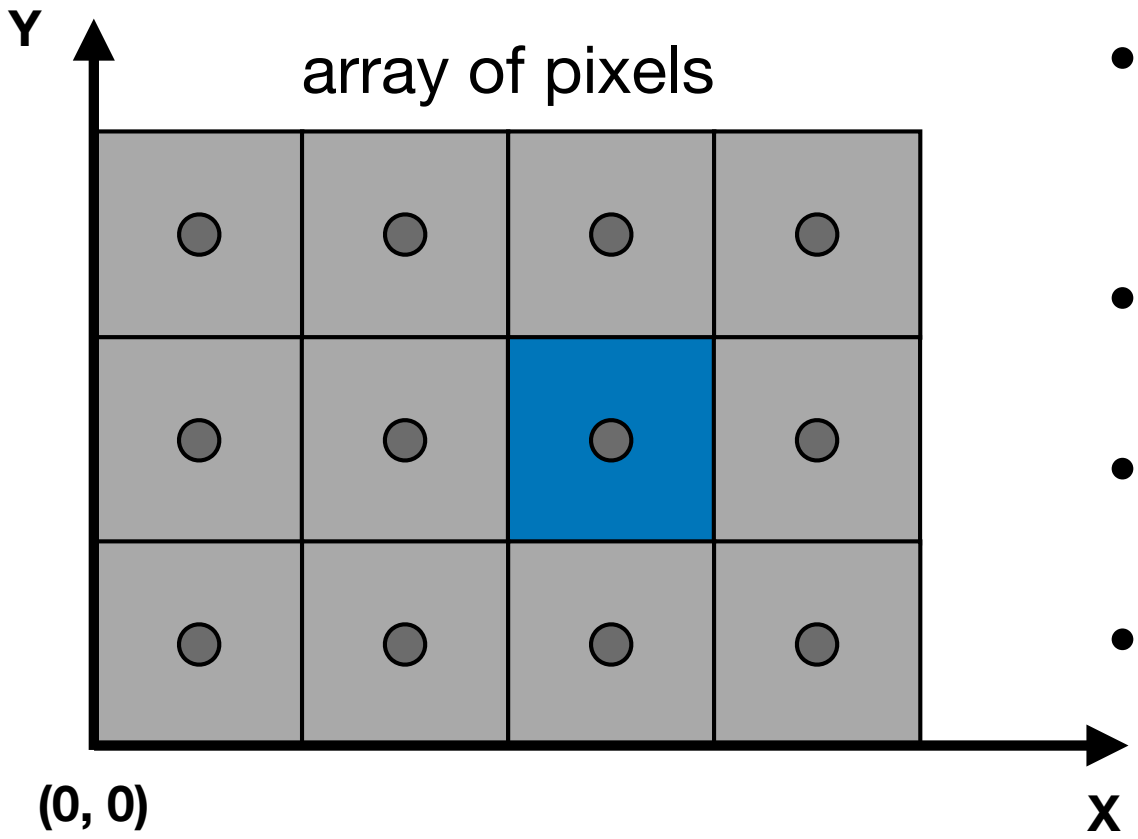
The sequence of spaces and transformations that gets objects from their original coordinates into screen space.

Canonical Cube to Screen

- What is a screen?
  - An array of pixels
  - Size of the array: resolution
  - A typical kind of raster display
- Raster == screen in German
  - Rasterize == drawing on the screen
- Pixel (FYI, short for “picture element”)
  - For now: A pixel is a little square with uniform color
  - Color is a mixture of (red, green, blue)

Canonical Cube to Screen

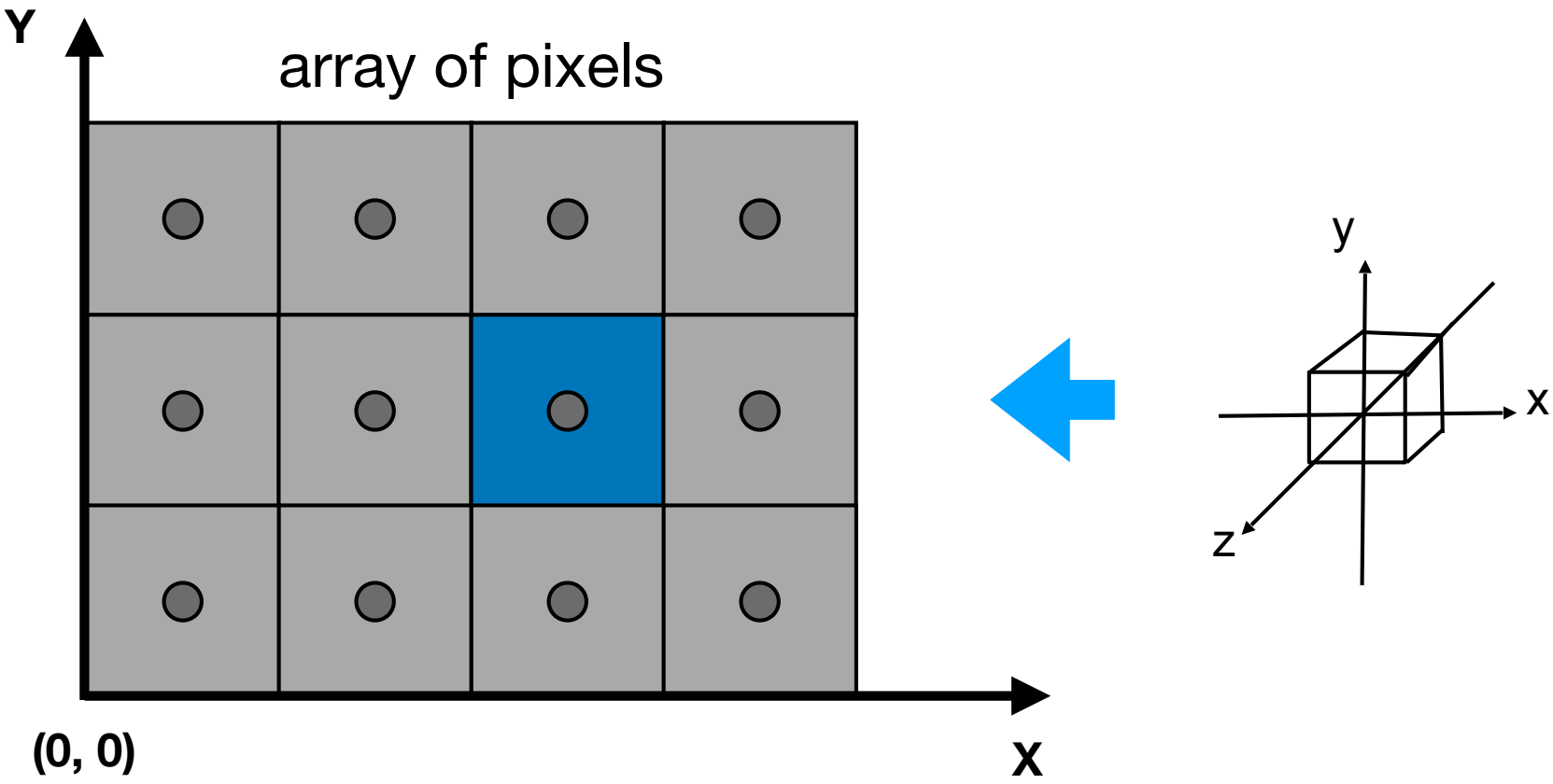
- Defining the screen space
  - Slightly different from the "tiger book"



- Pixels' indices are in the form of (x,y), where both x and y are integers
- Pixels' indices are from (0,0) to (width-1, height-1)
- Pixel (x,y) is centered at (x+0.5, y+0.5)
- The screen covers range from (0,0) to (width, height)

Canonical Cube to Screen

- Irrelevant to z
- Transform in xy plane: $[-1,1]^2$ to [0, width]×[0, height]



Canonical Cube to Screen

- Irrelevant to z
- Transform in xy plane: $[-1,1]^2$ to $[0, width]\times[0, height]$
- View port transform matrix:

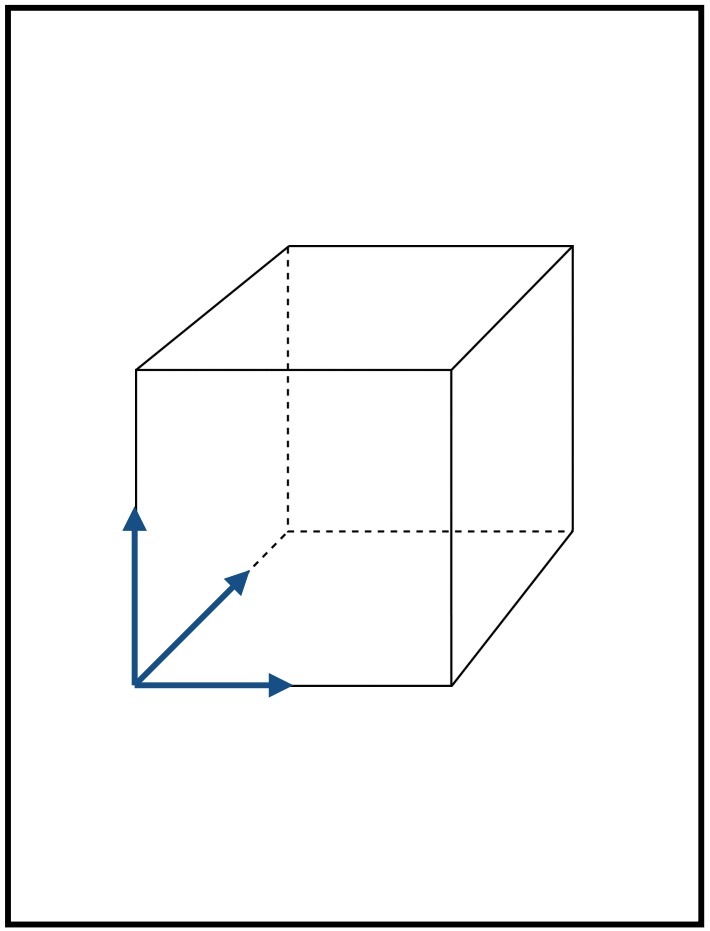
$$M_{viewport} = \begin{pmatrix} \frac{width}{2} & 0 & 0 & \frac{width}{2} \\ 0 & \frac{height}{2} & 0 & \frac{height}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transforms Recap

Coordinate Systems

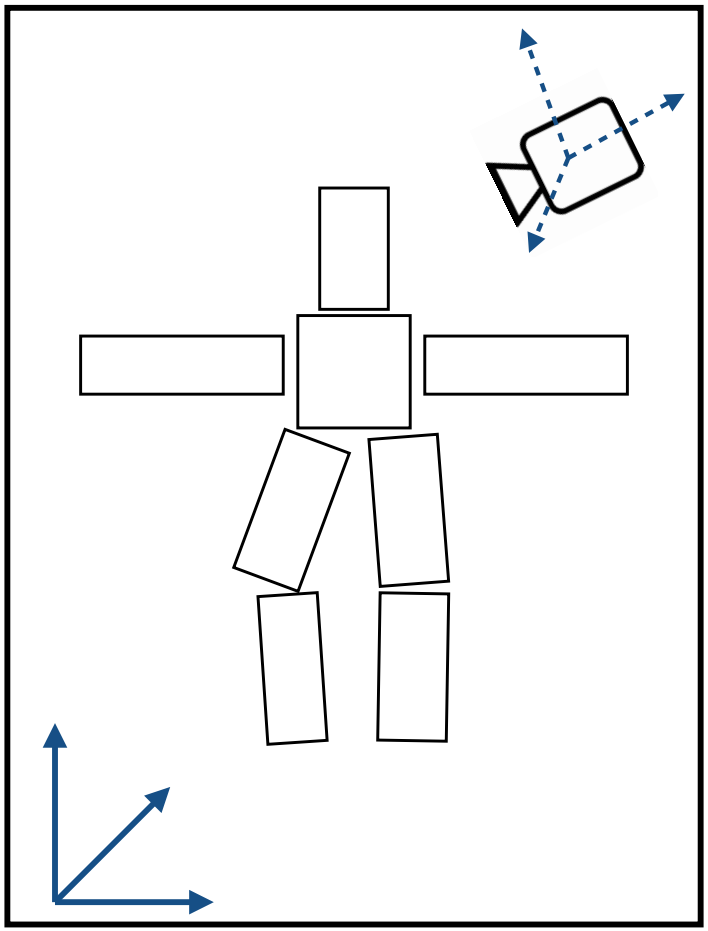
- Object coordinates
  - Apply modeling transforms…
- World (scene) coordinates
  - Apply viewing transform…
- Camera (eye) coordinates
  - Apply perspective transform + homog. divide…
- Normalized device coordinates
  - Apply 2D screen transform…
- Screen coordinates

Transforms Recap



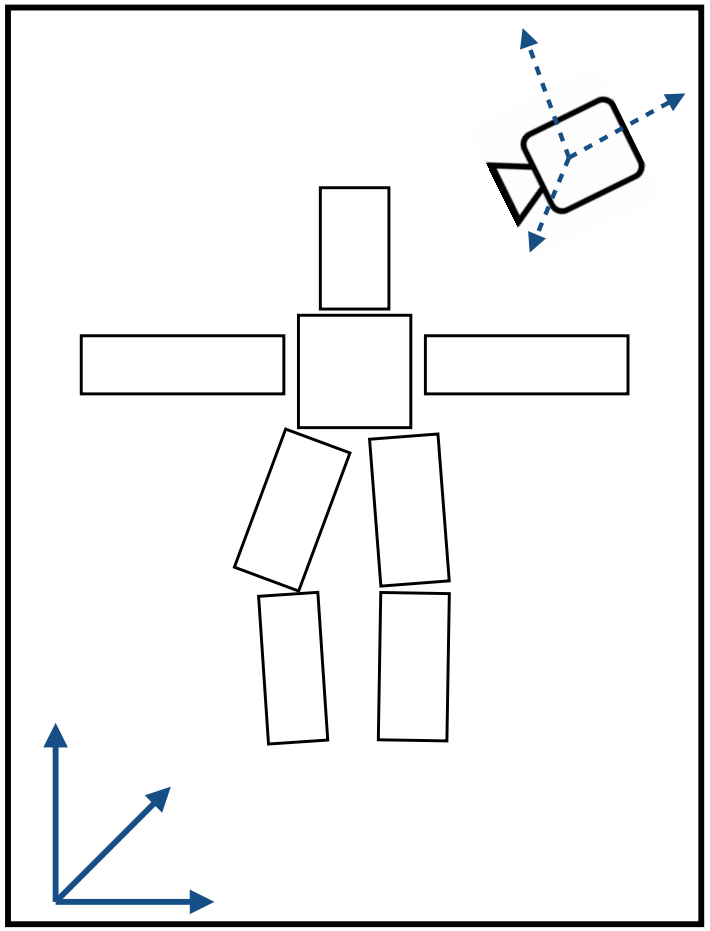
Object coords

Modeling transforms



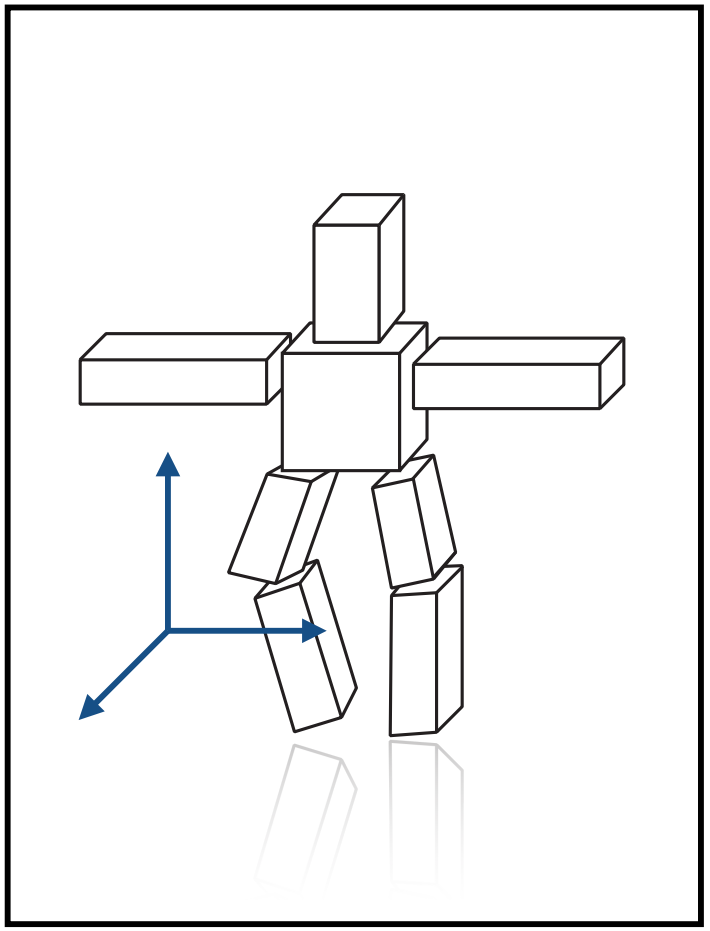
World coords

Transforms Recap



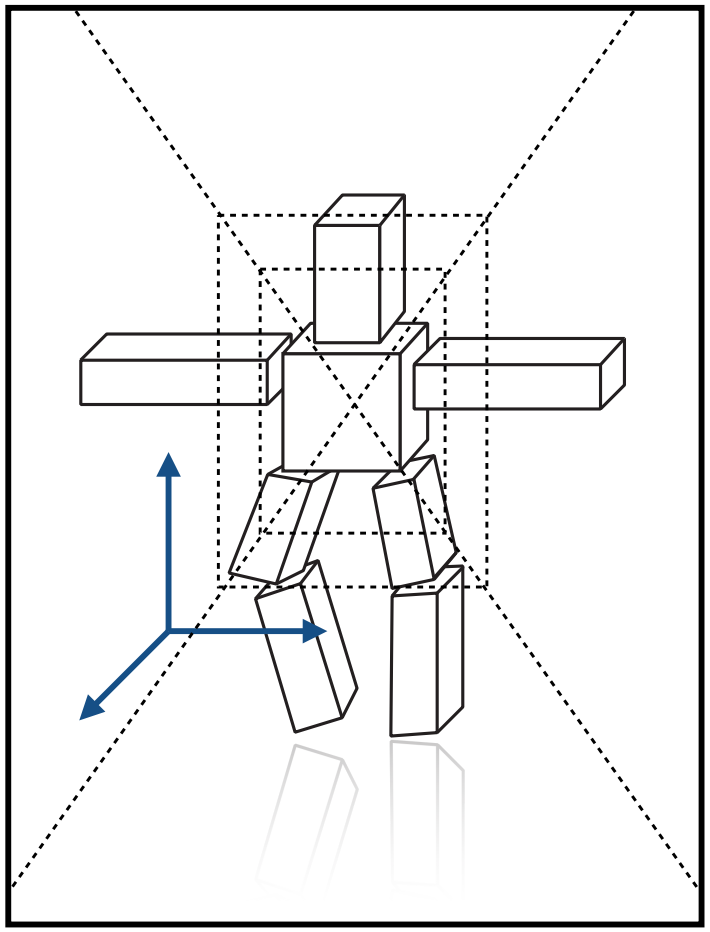
World coords

Viewing transform



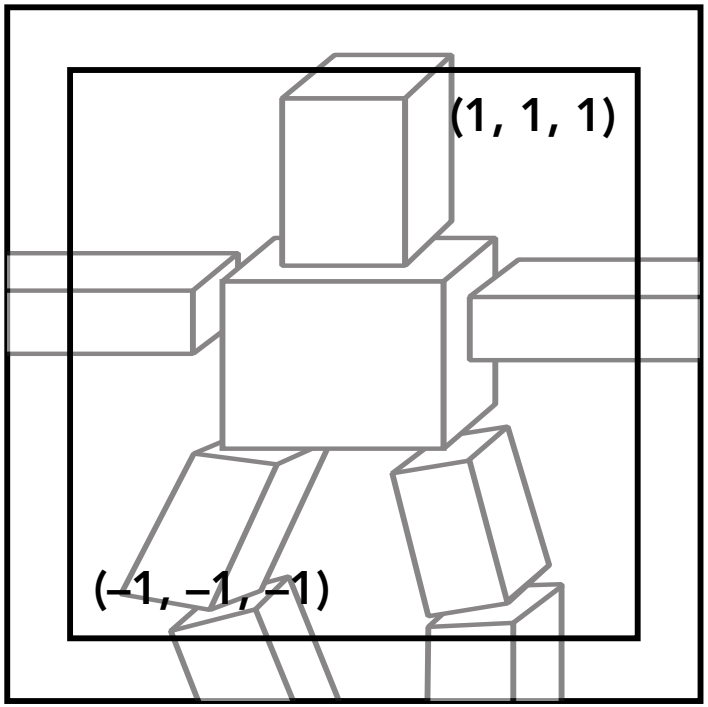
Camera coords

Transforms Recap



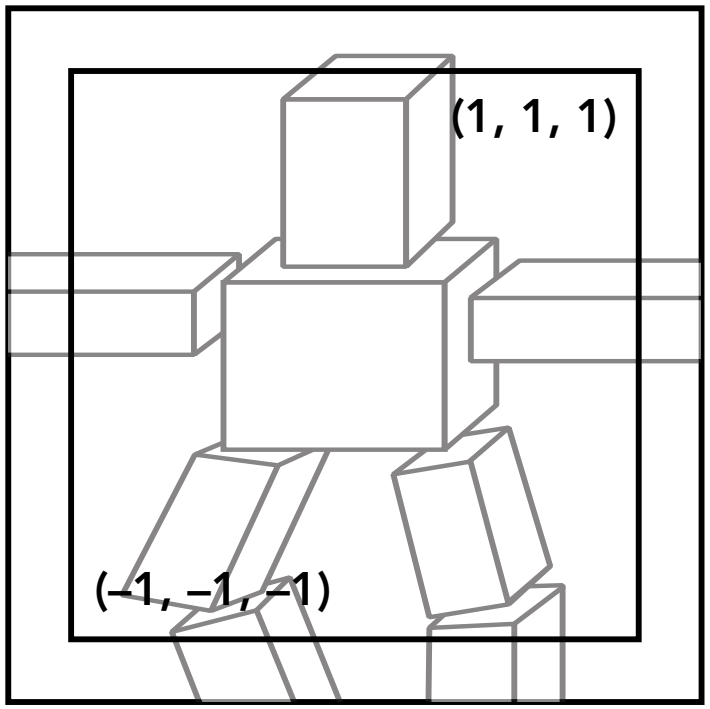
Camera coords

Perspective projection and homogeneous divide



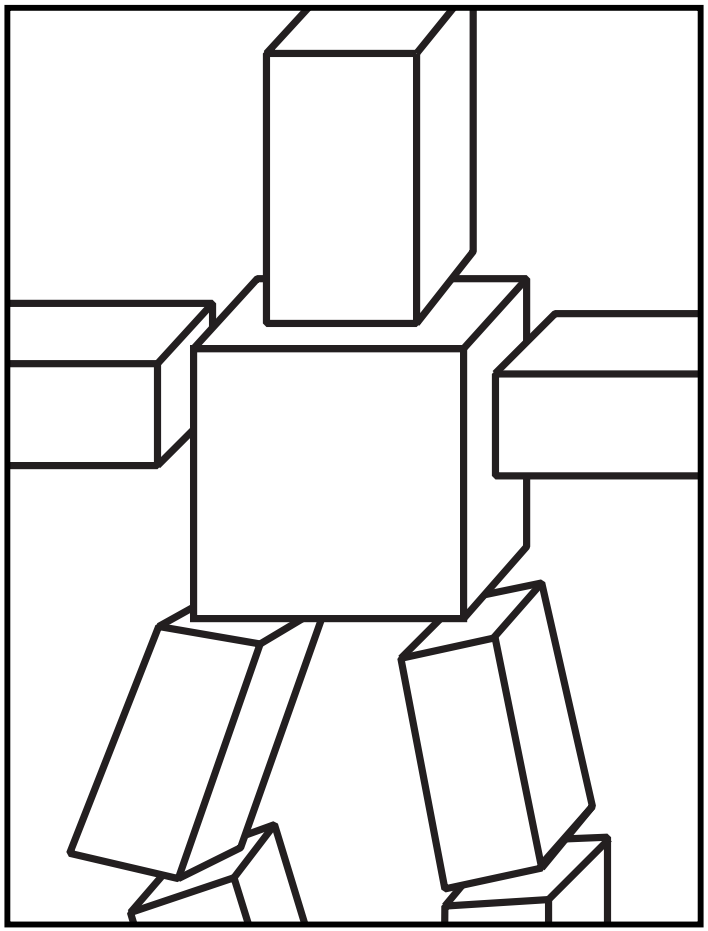
NDC

Transforms Recap



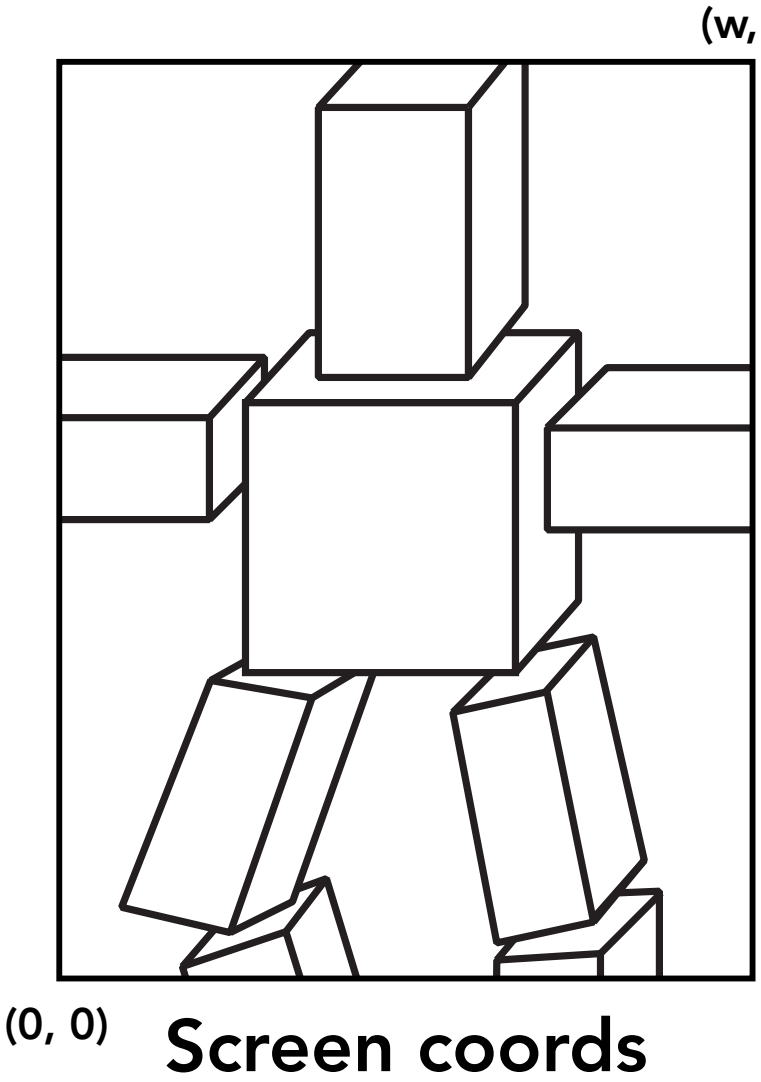
NDC

Screen transform



Screen coords

Transforms Recap



Screen coords

Rasterization