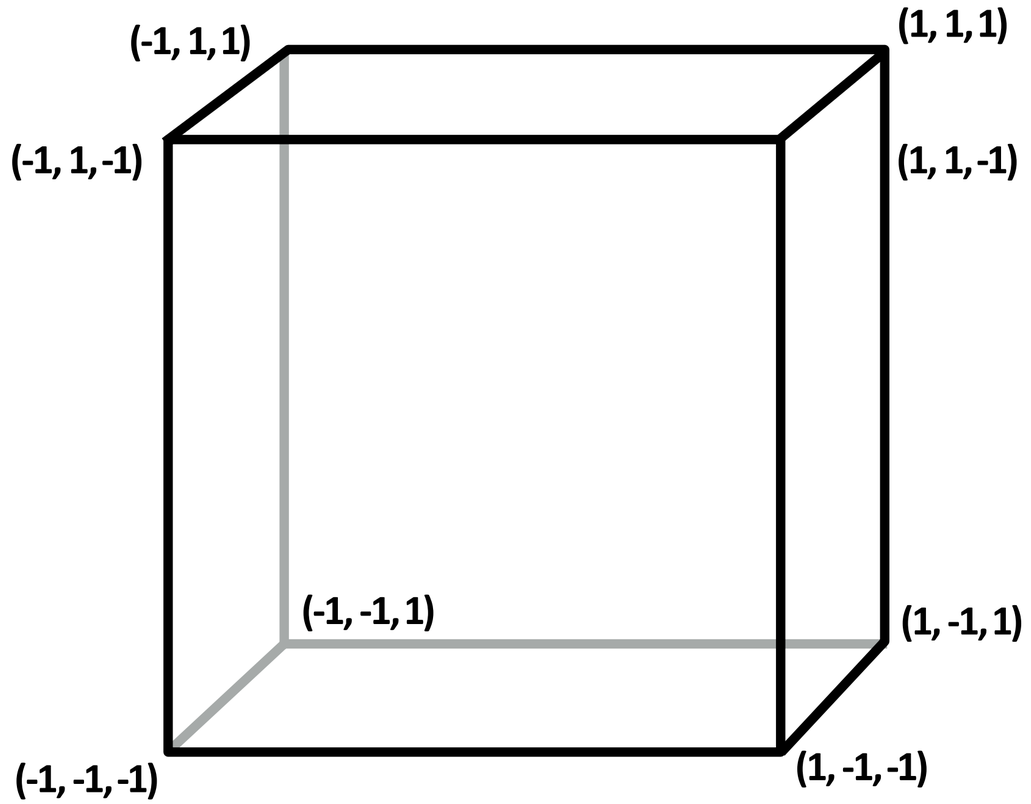
Week 4:

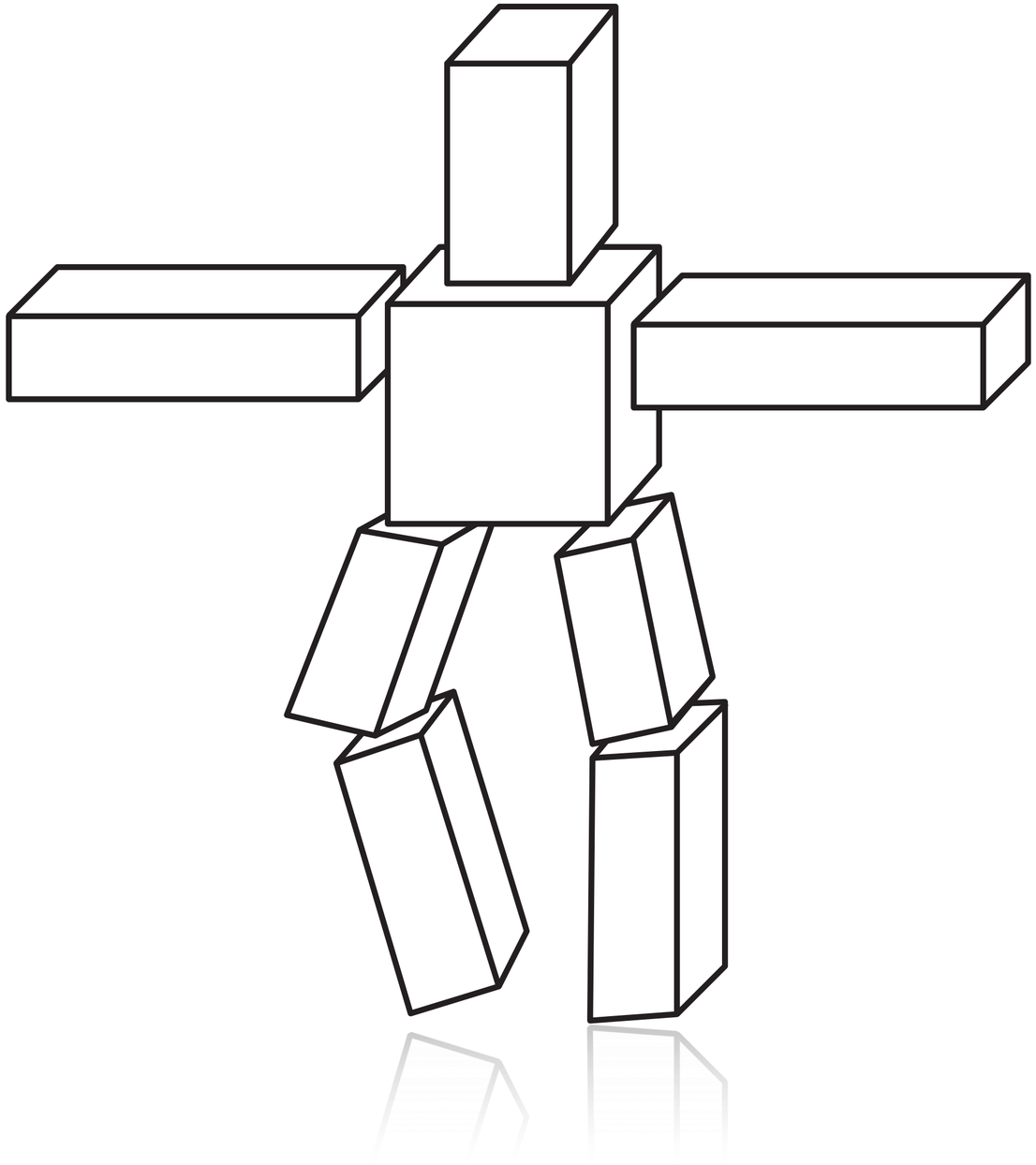
Coordinate Spaces and Transformations

Transformations

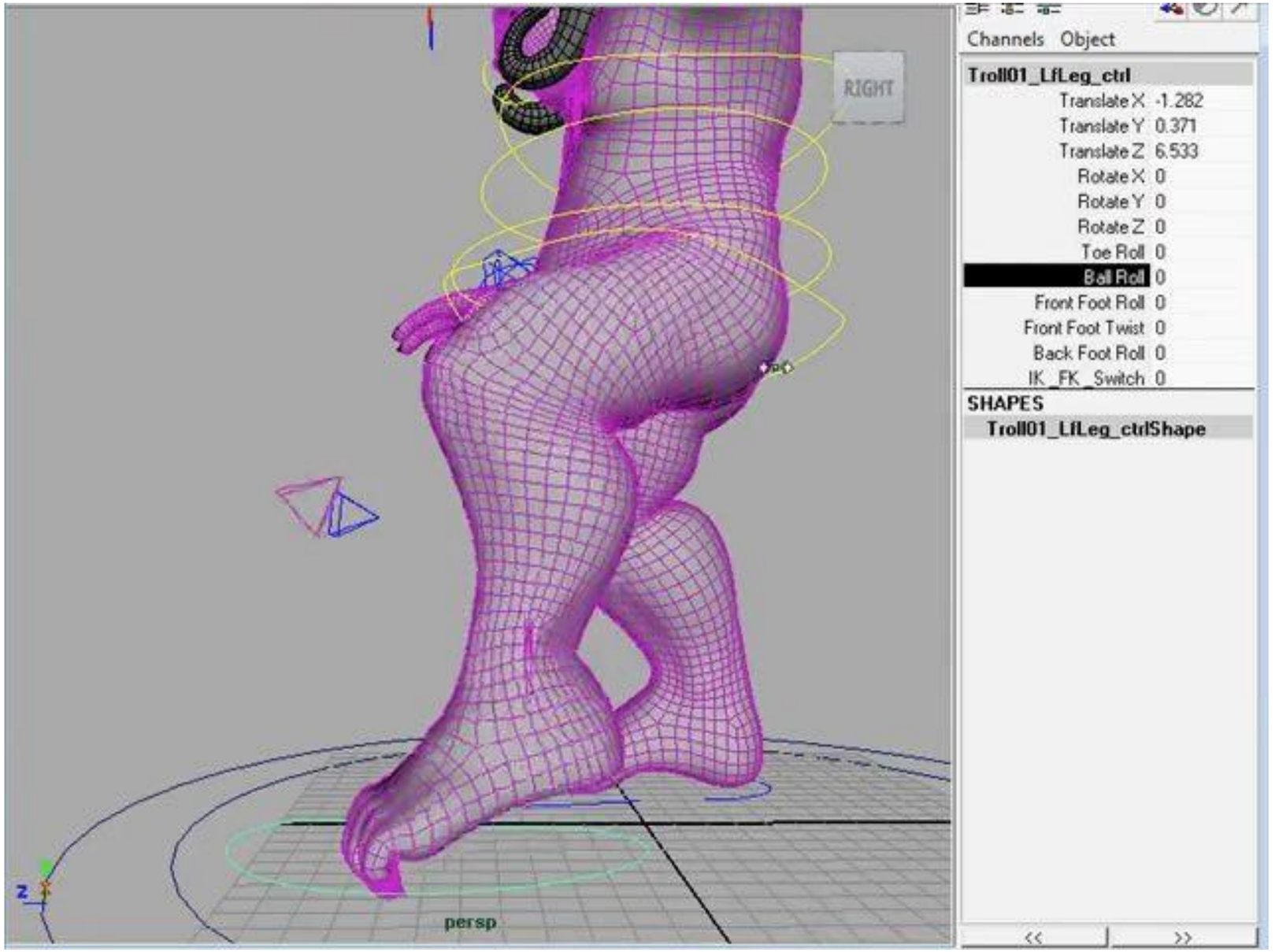
Acube, centered at the origin, with faces of size 2 x 2



Consider drawing a cube person



Transformations in character rigging



Transformations for geometry instancing

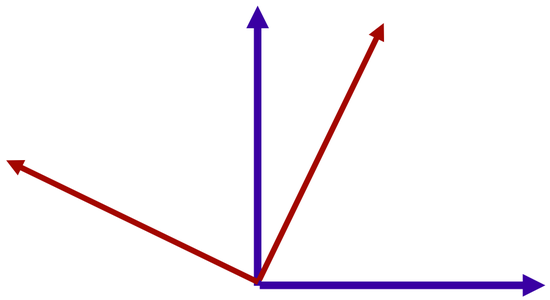
Credit: Jan-Walter Schliep, Burak Kahraman, and Timm Dapper

Coordinates

- We are used to represent points with tuples of coordinates such as $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$
- But the tuples are meaningless without a clear coordinate system

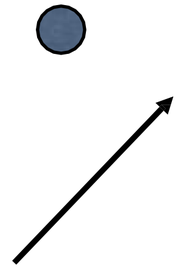
could be this point in the red coordinate system

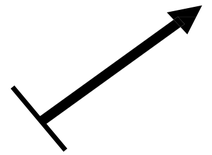
could be this point in the blue coordinate system



Different objects

- **Points**
  - represent locations
- **Vectors**
  - represent movement, force, displacement from A to B
- **Normals**
  - represent orientation, unit length
- **Coordinates**
  - numerical representation of the above objects **in a given coordinate system**



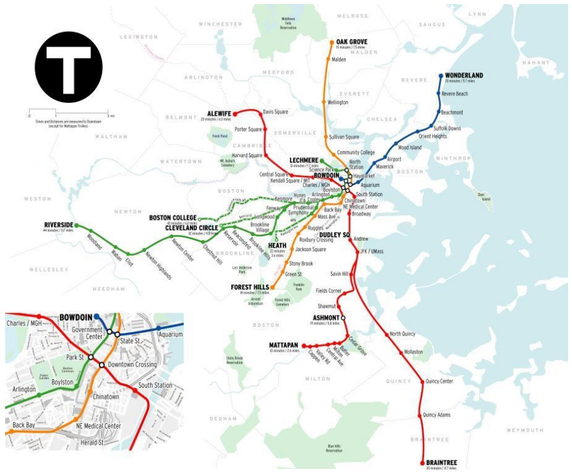


$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Points & vectors are different

- The 0 vector has a fundamental meaning: no movement, no force
- Why would there be a special 0 point?
  - It’s meaningful to add vectors, not points

  - Boston location + NYC location =?

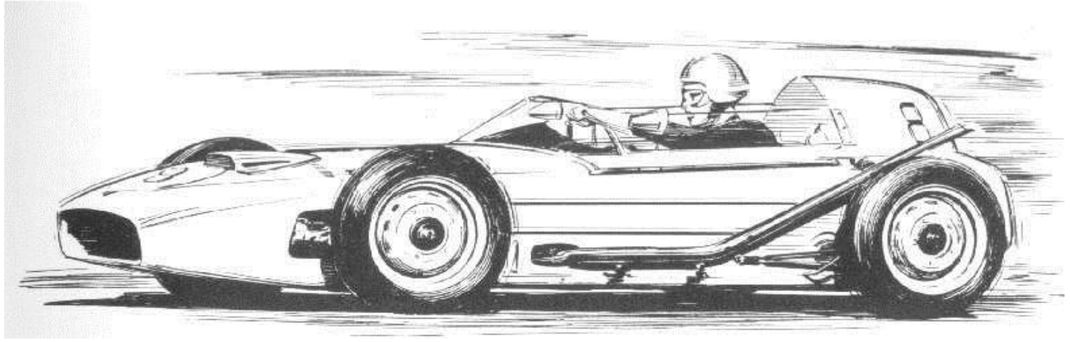


+

=?

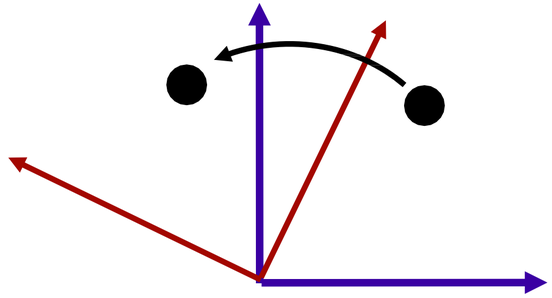
Points & vectors are different

- Moving car
  - points describe location of car elements
  - vectors describe velocity, distance between pairs of points
- If I **translate** the moving car to a different road
  - The points (location) change
  - The vectors (speed, distance between points) don't

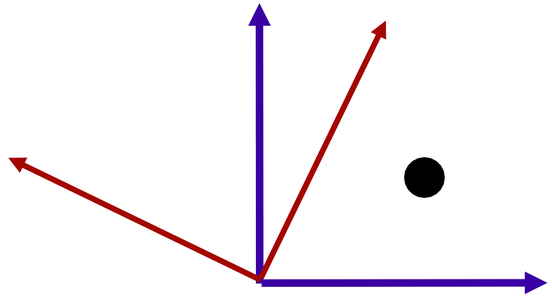


Matrices have two purposes

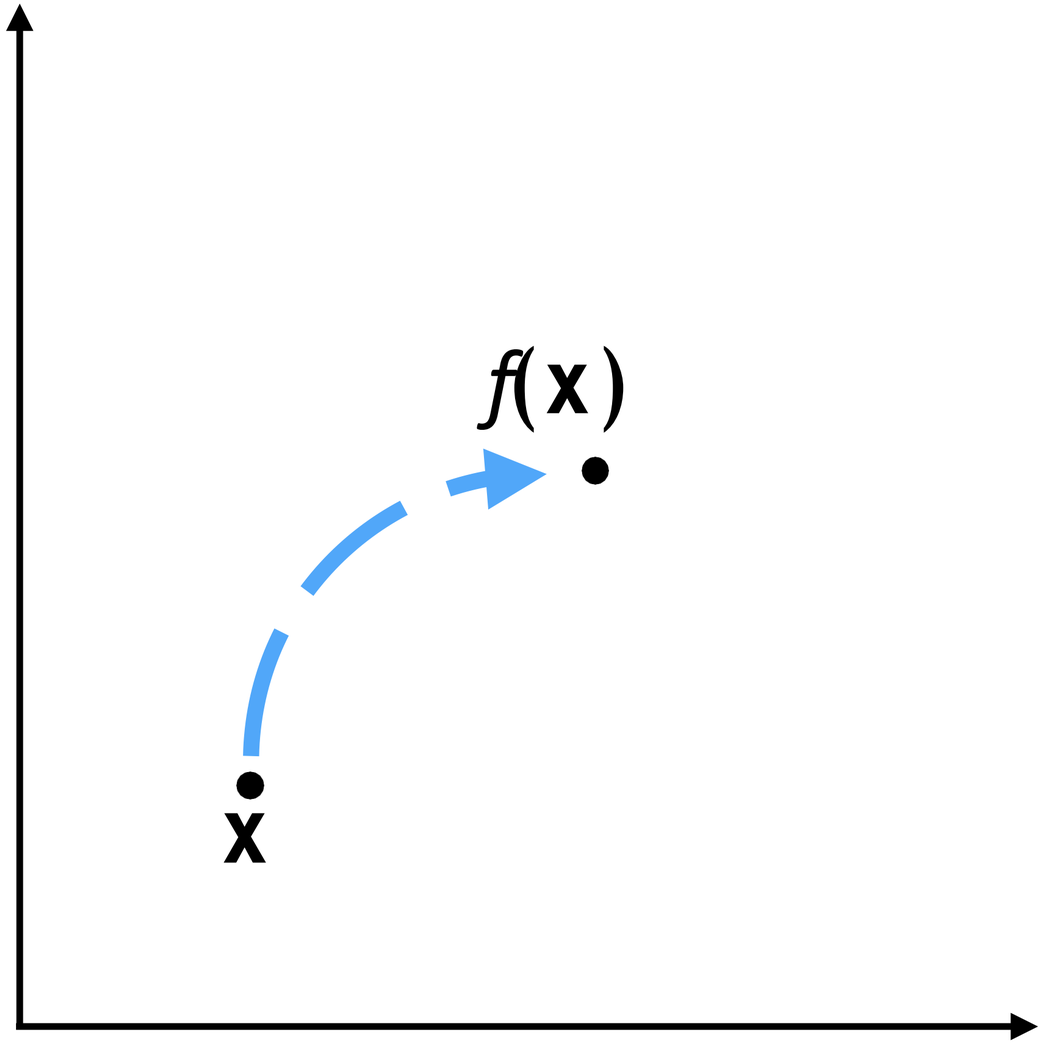
- Transform things
  - e.g. rotate the car from facing North to facing East



- Express coordinate system changes
  - e.g. given the driver's location in the coordinate system of the car, express it in the coordinate system of the world



Basic idea: f transforms $\mathbf{x}$ to $f(\mathbf{x})$



What can we do with *linear* transformations?

- **What does *linear* mean?**

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

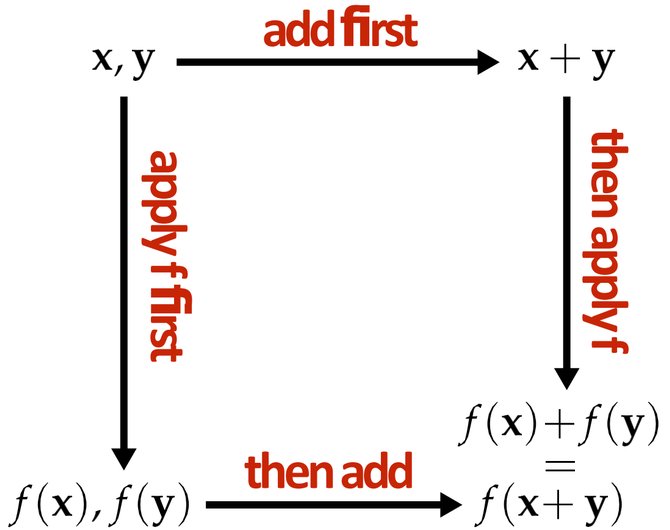
$$f(a\mathbf{x}) = af(\mathbf{x})$$

- **Cheap to compute**
- **Composition of linear transformations is linear**
  - **Leads to uniform representation of many types of transformations**

Linear transformation

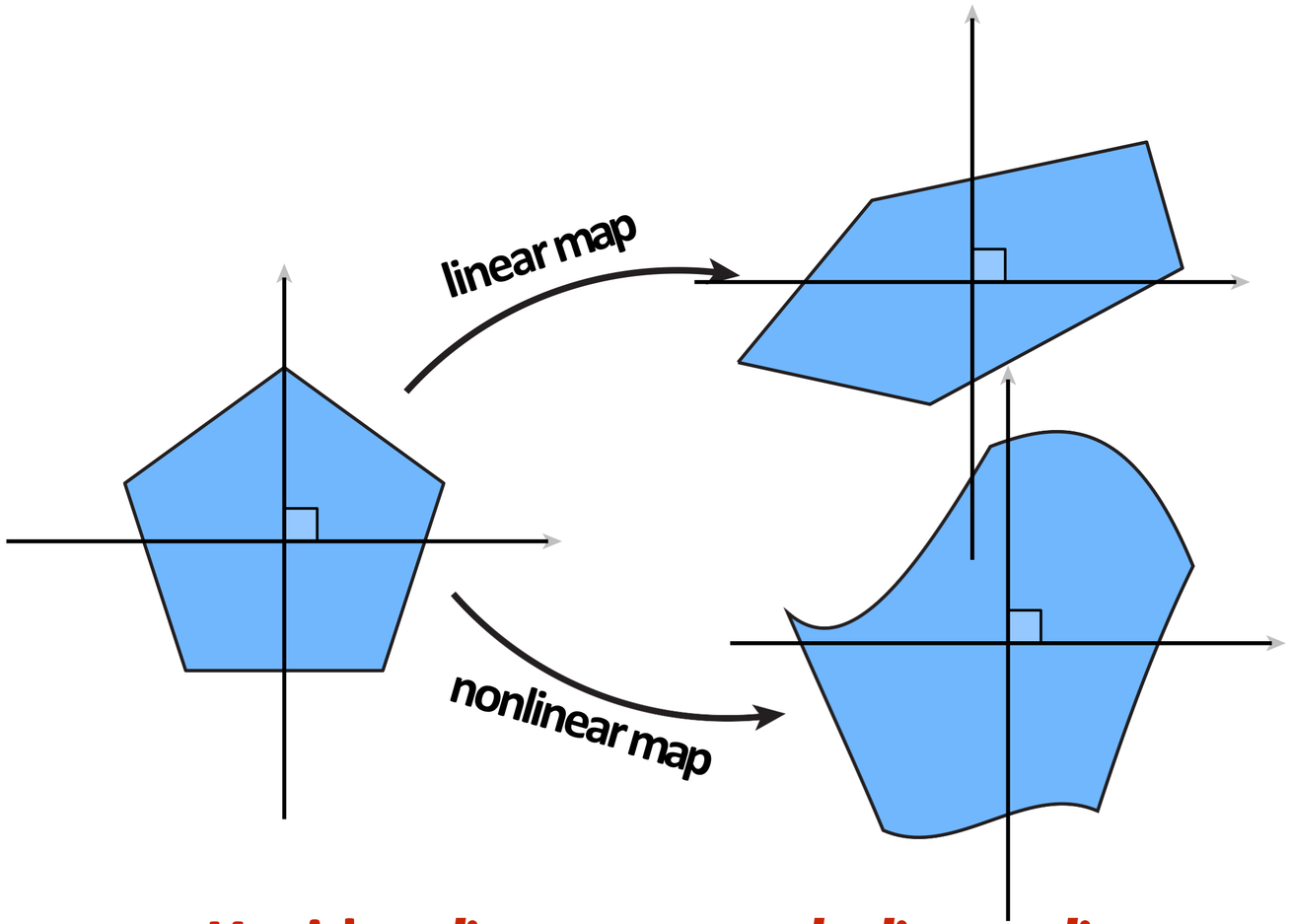
$$f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f(\mathbf{v})$$
$$f(a\mathbf{u}) = af(\mathbf{u})$$

- **In other words: if it doesn't matter whether we add the vectors and then apply the map, or apply the map and then add the vectors (and likewise for scaling):**



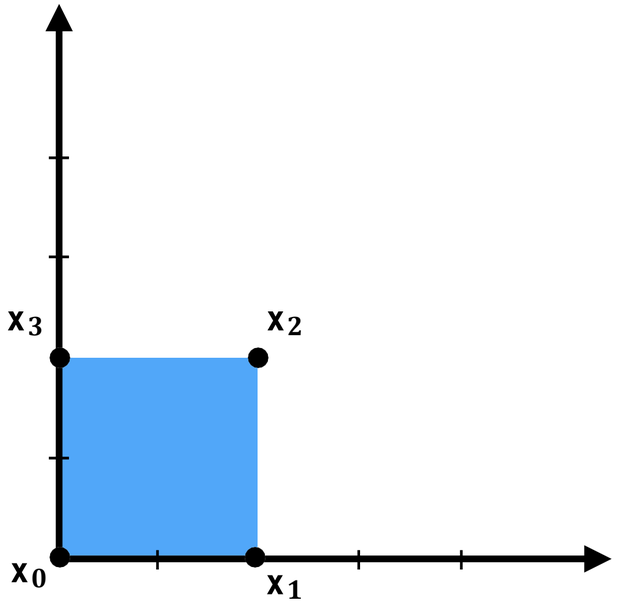
Linear transforms/maps—visualized

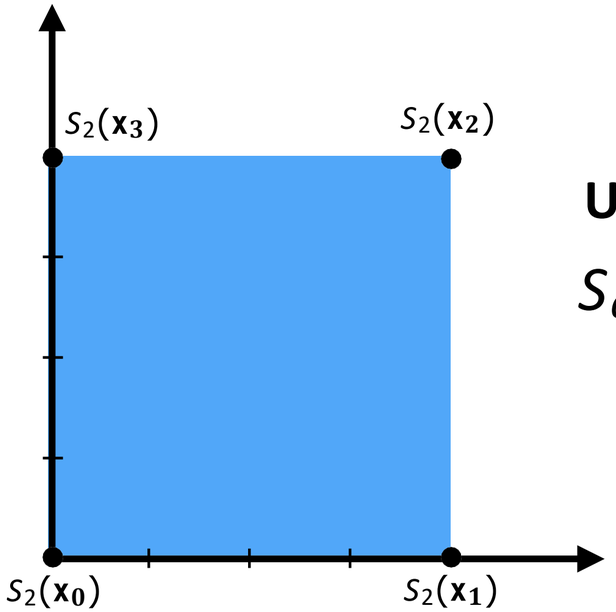
- **Example:**



Key idea: *linear maps take lines to lines*

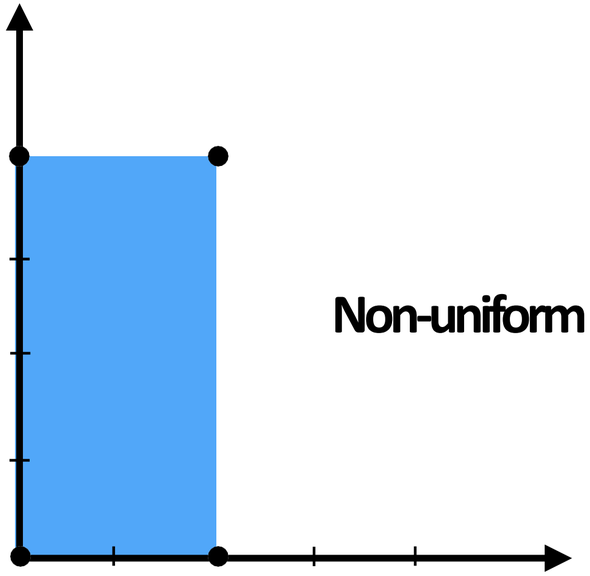
Scale





Uniform scale:

$$S_a(\mathbf{x}) = a\mathbf{x}$$



Non-uniform scale??

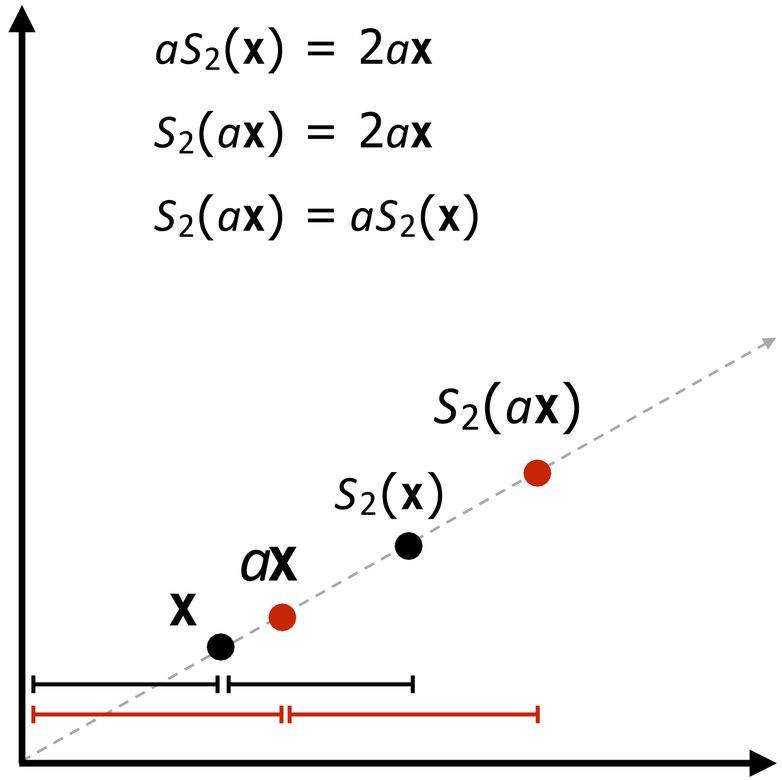
Is scale a linear transform?

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

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

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

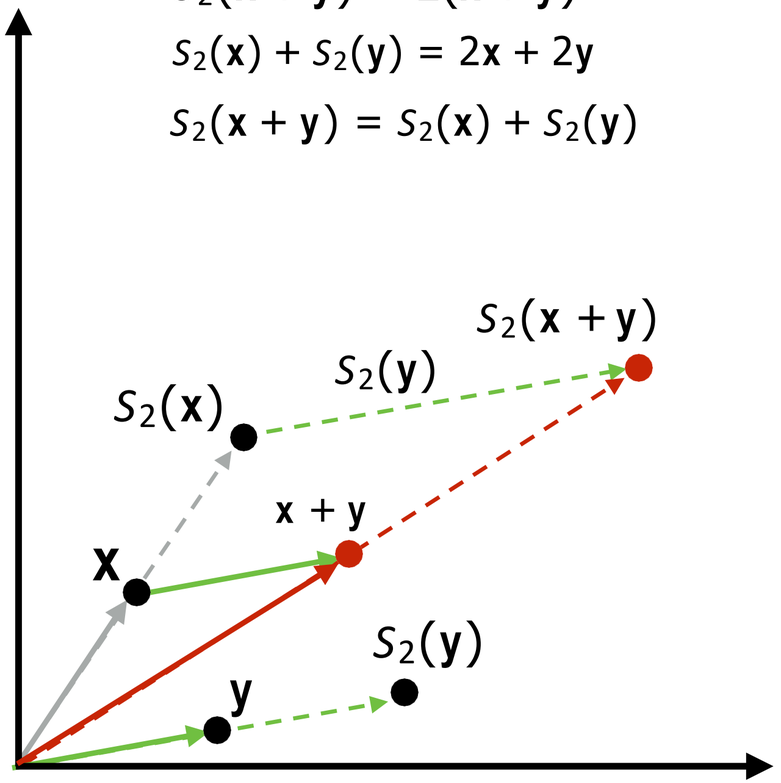
$S_2(a\mathbf{x}) = aS_2(\mathbf{x})$



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

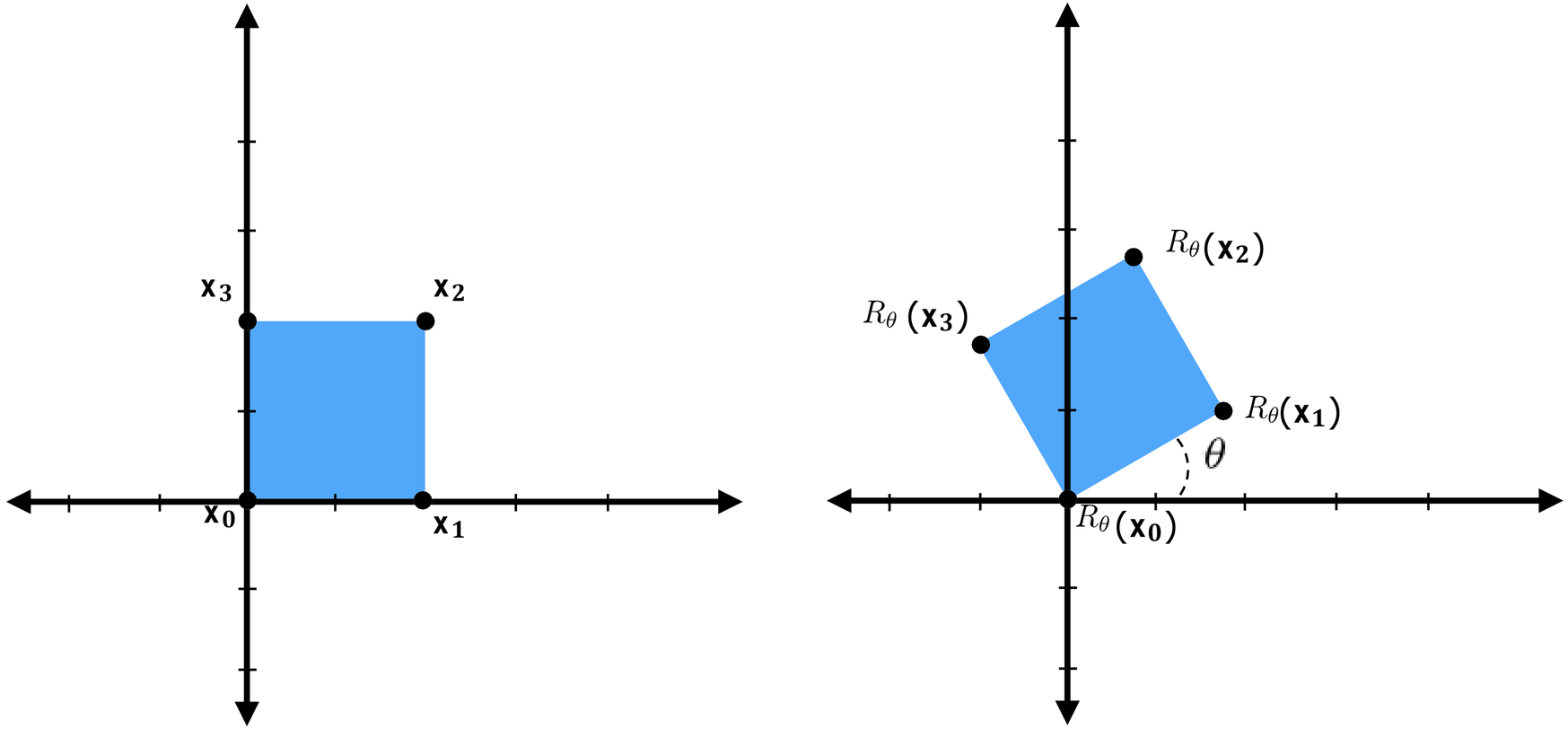
$S_2(\mathbf{x}) + S_2(\mathbf{y}) = 2\mathbf{x} + 2\mathbf{y}$

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



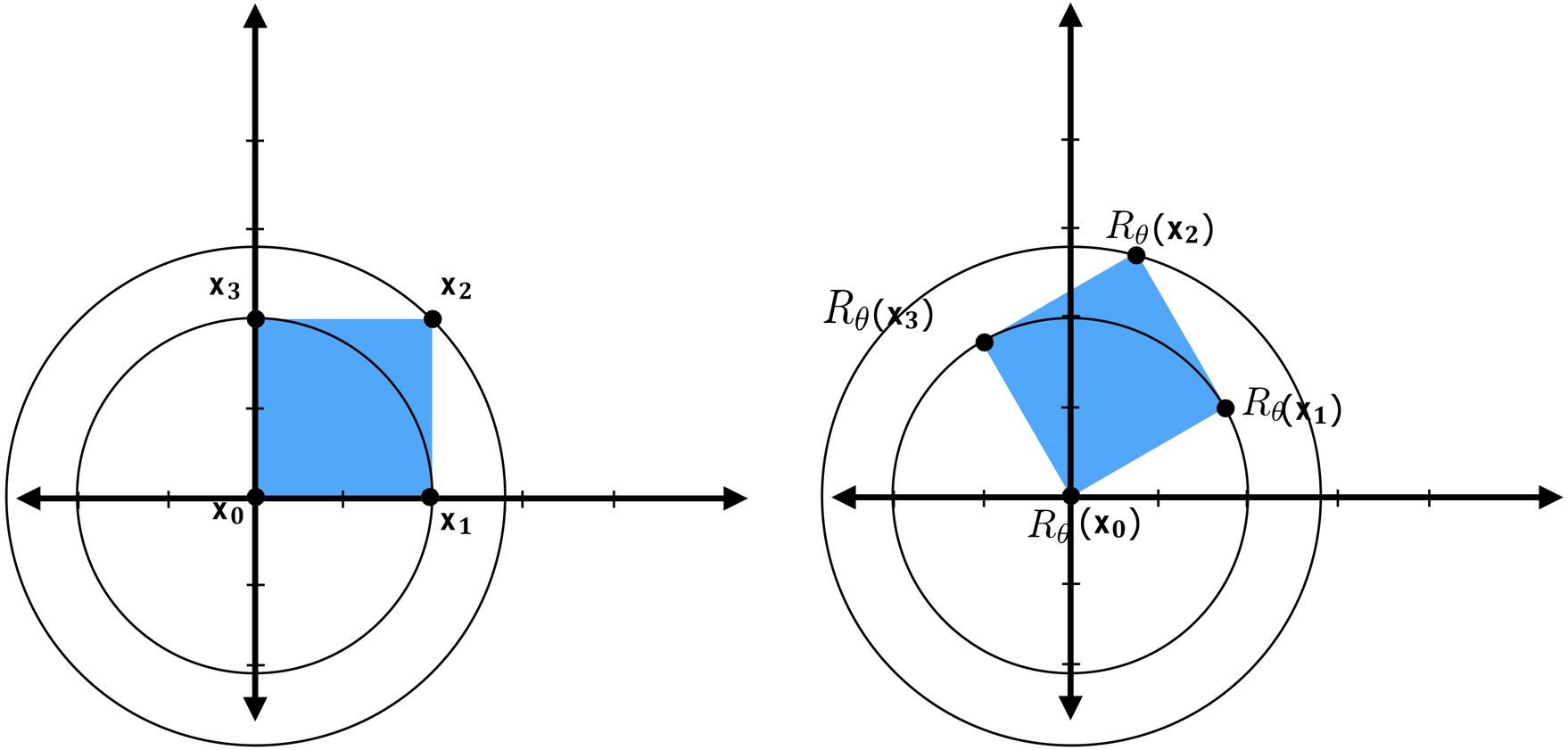
Yes!

Rotation



R_θ = **rotate counter-clockwise by** θ

Rotation as circular motion

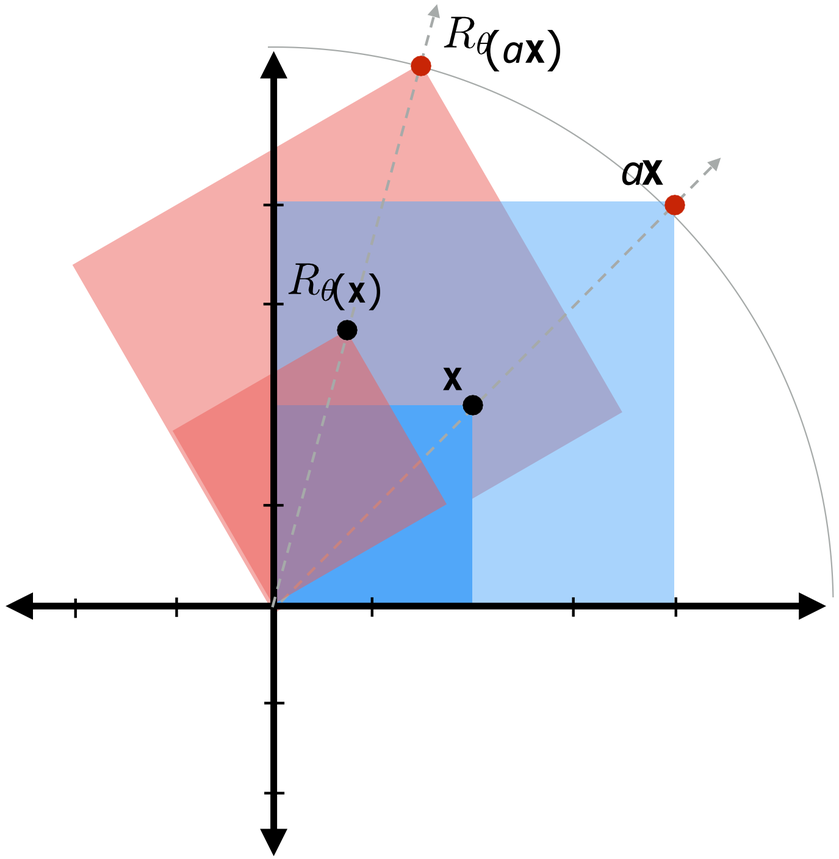


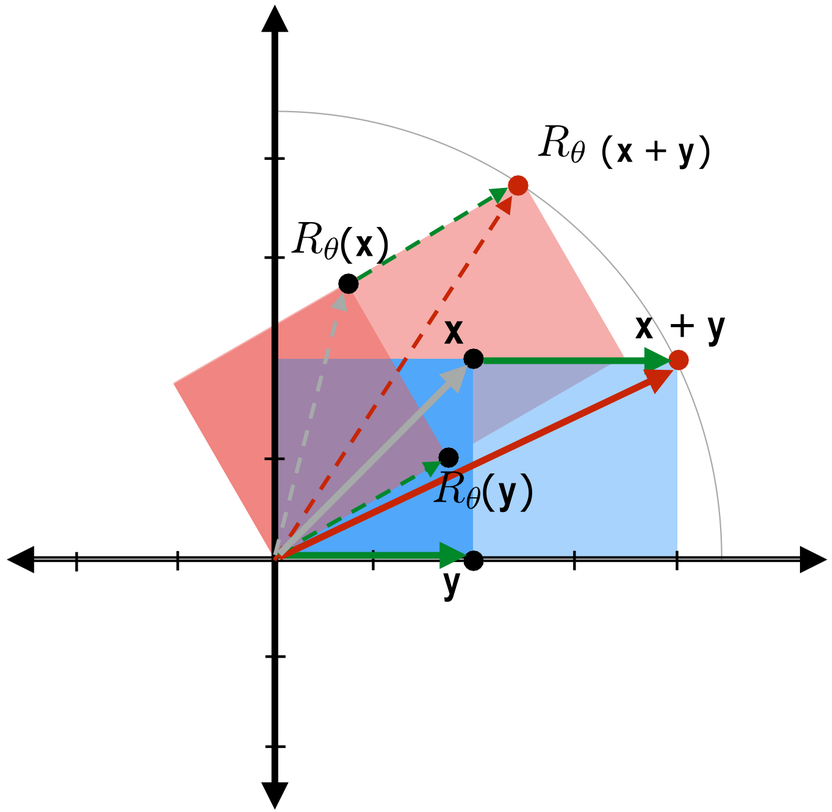
R_θ= rotate counter-clockwise by θ

As angle changes, points move along *circular* trajectories.

Hence, rotations preserve length of vectors: $|R_\theta(\mathbf{x})| = |\mathbf{x}|$

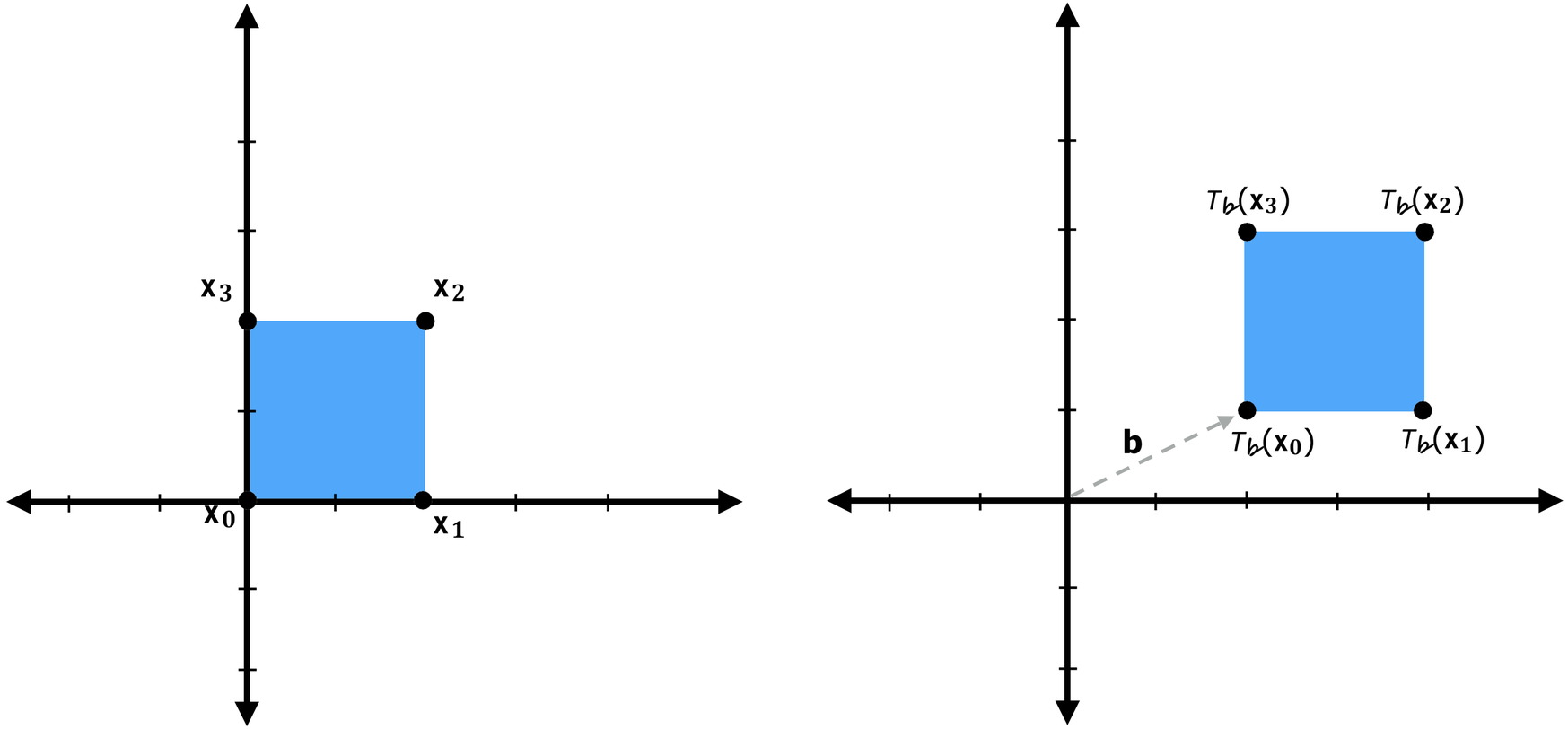
Is rotation linear?





Yes!

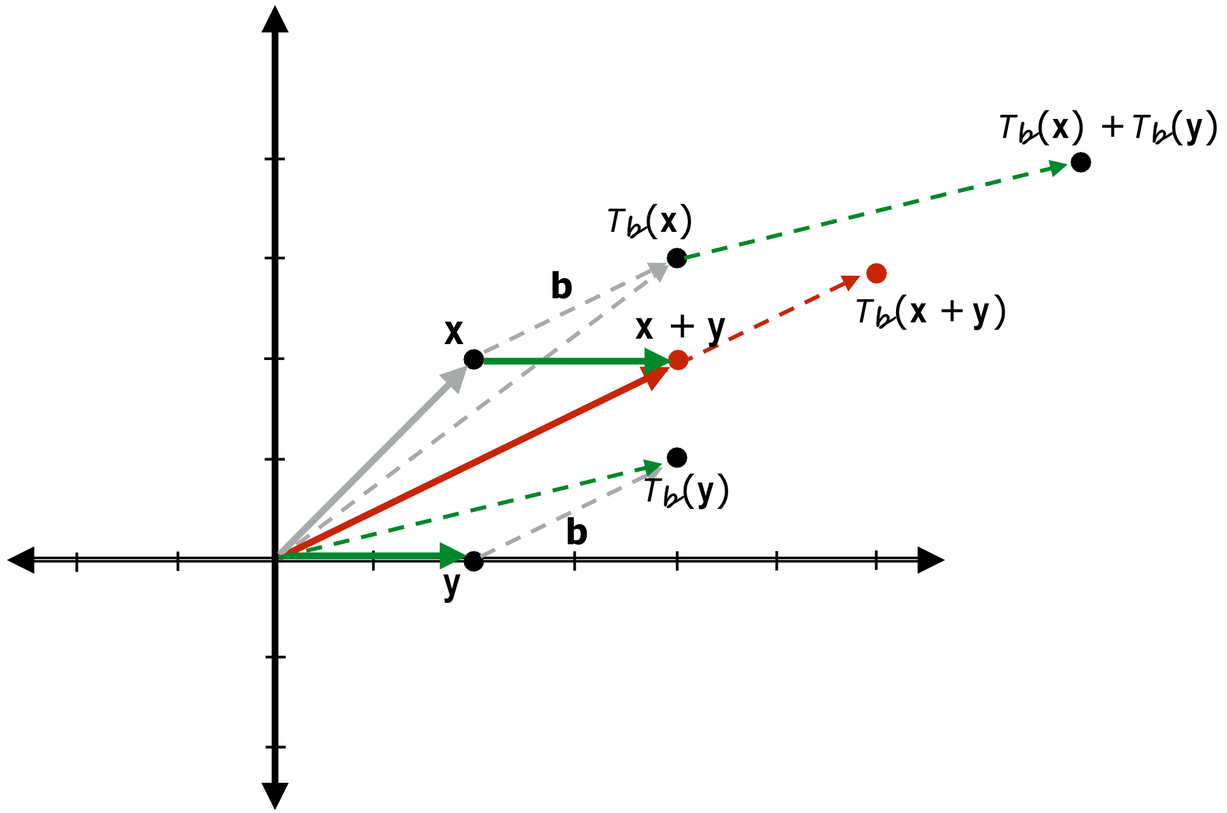
Translation



$T_{\mathbf{b}}$ — **"translate by b"**

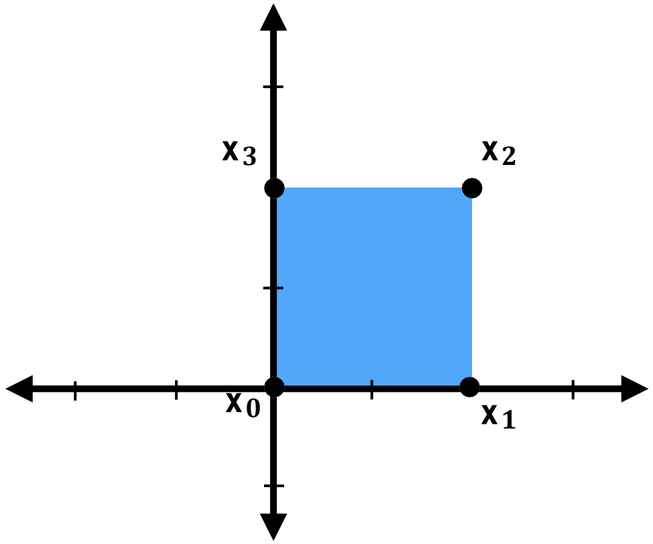
$T_b(\mathbf{x}) = \mathbf{x} + \mathbf{b}$

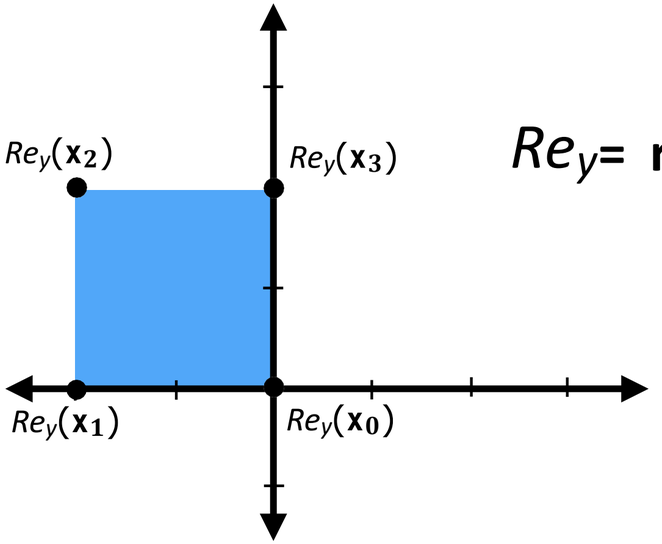
Is translation linear?



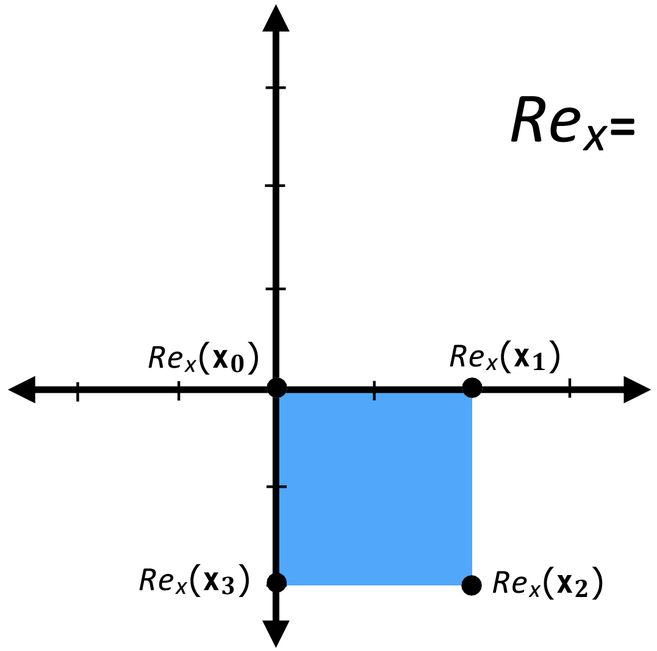
No. Translation is affine.

Reflection



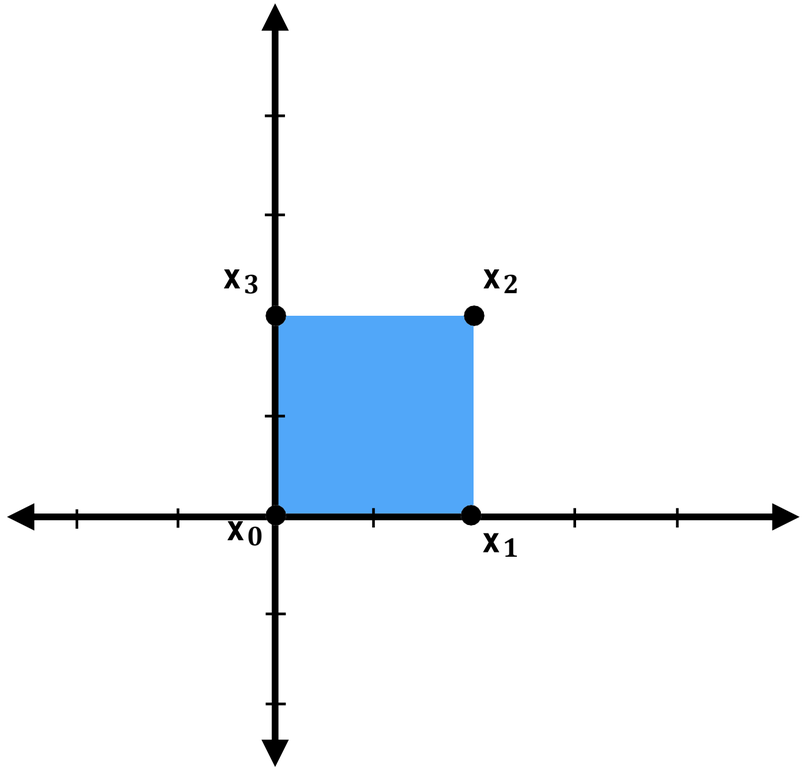


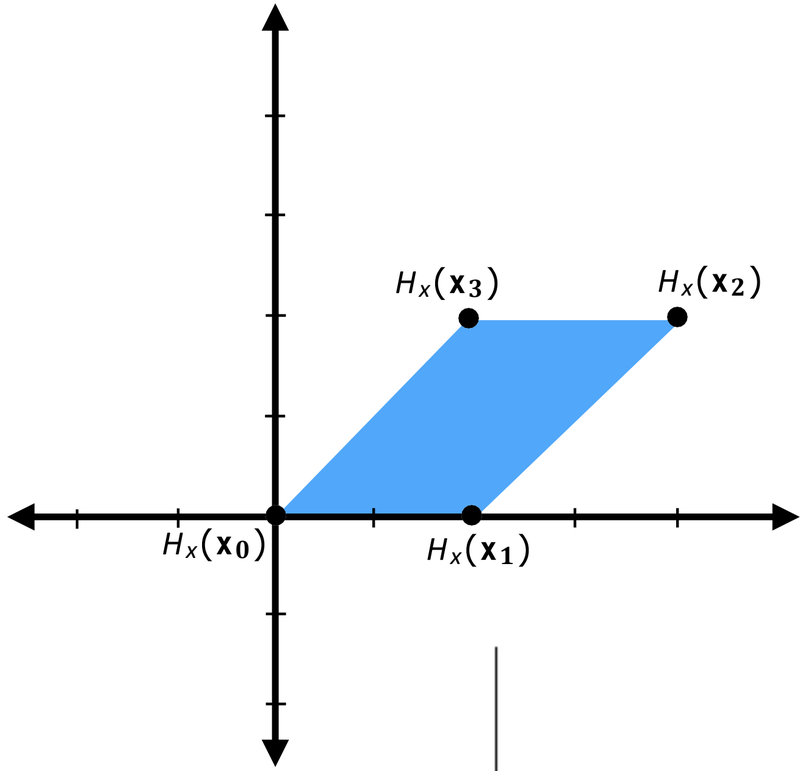
Re_y= **reflection about** y



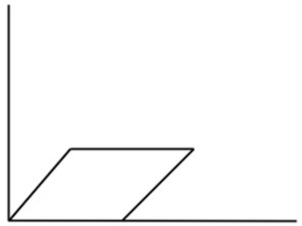
Re_x= **reflection about** x

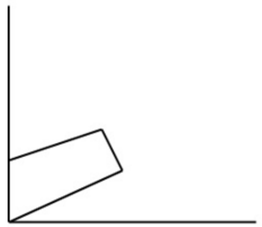
Shearing

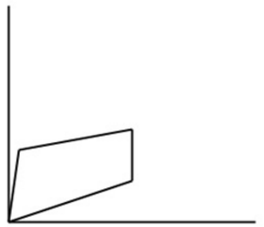




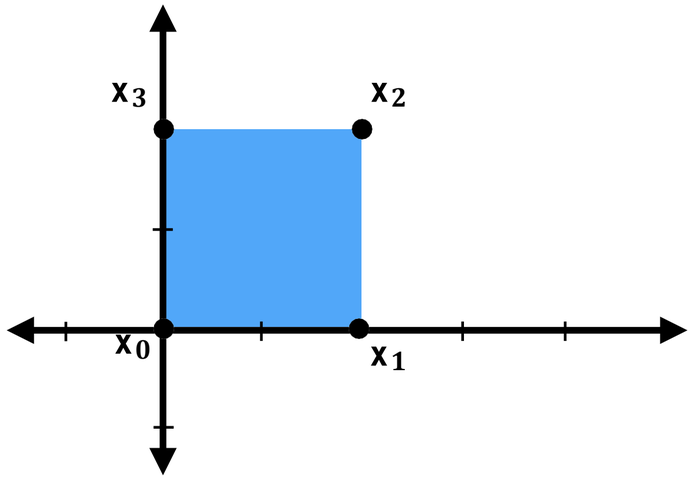
Original Object

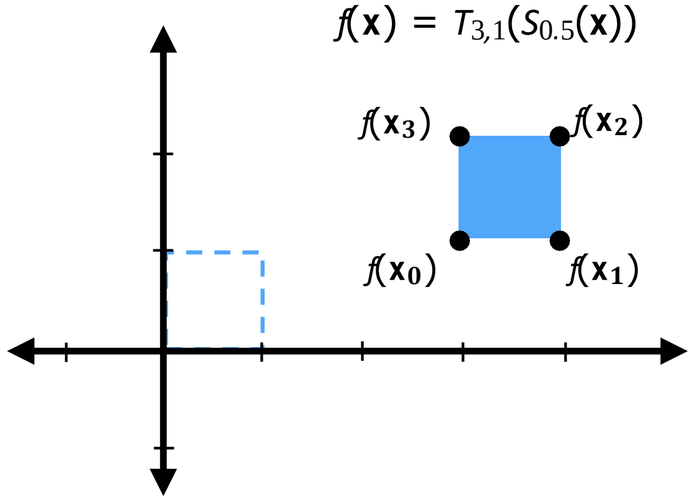

Shear in X direction

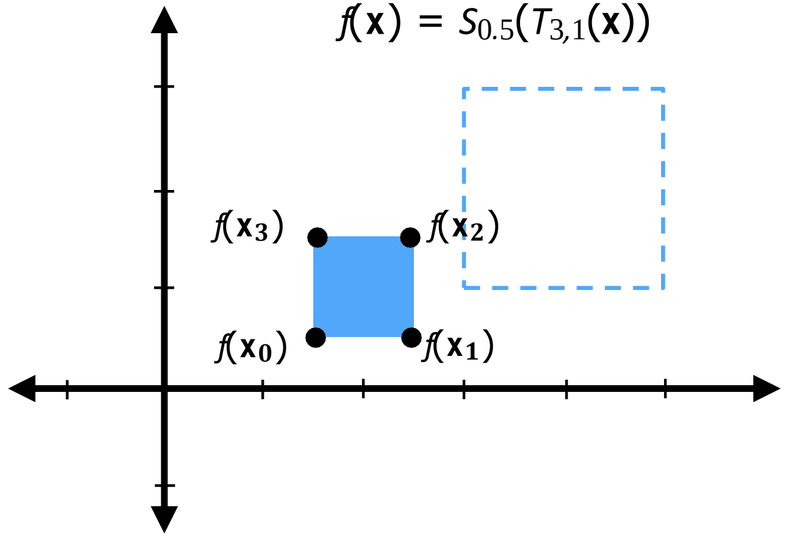

Shear in Y direction


Shear in both directions

Compose basic transformations to construct more **complicated ones**





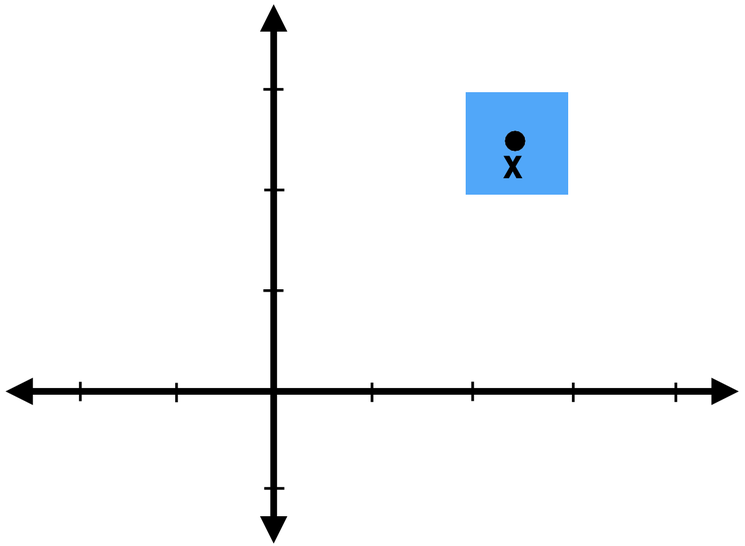


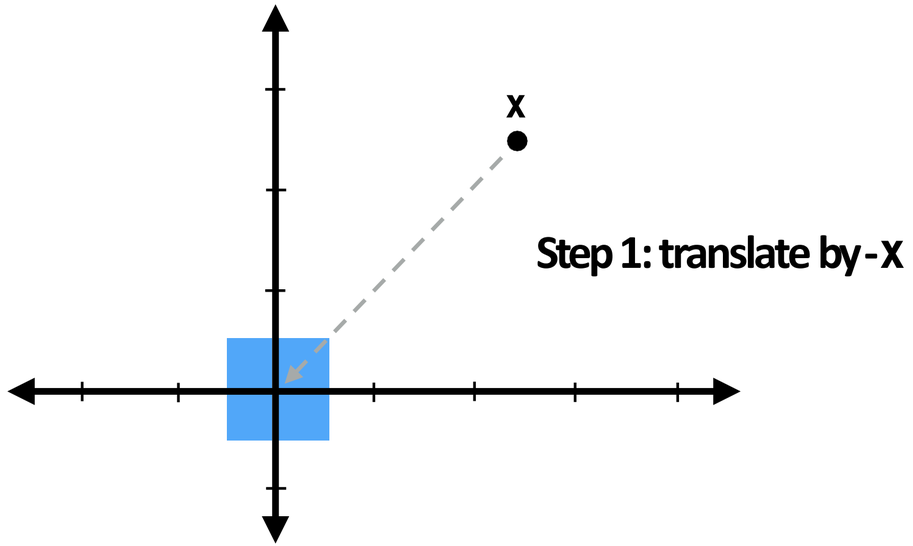
Note: order of composition matters

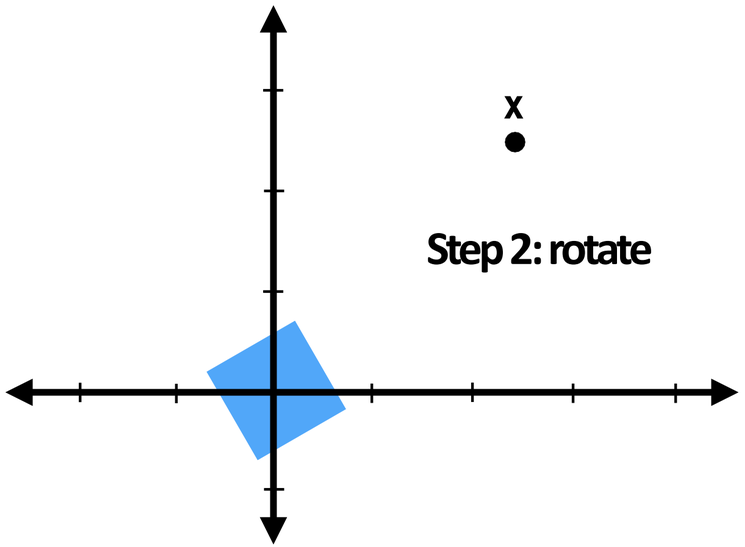
Top-right: scale, then translate

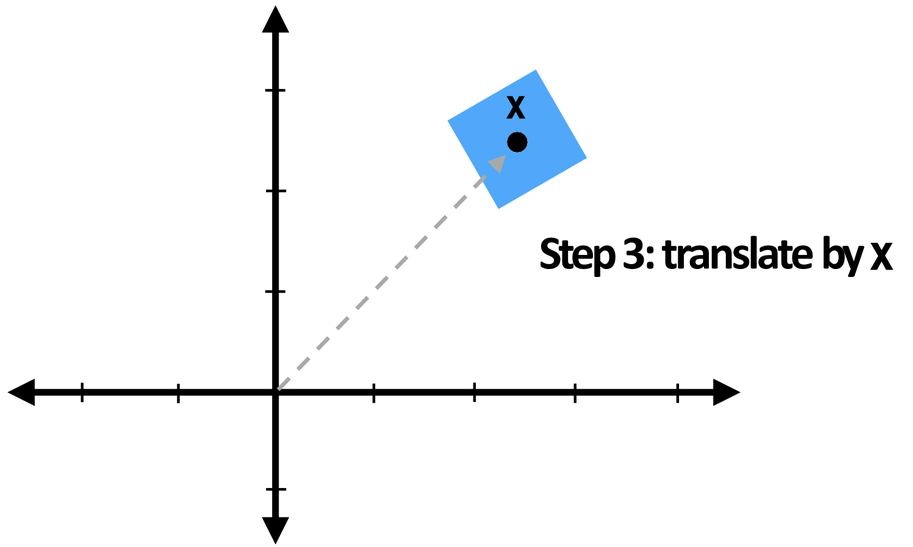
Bottom-right: translate, then scale

Common task: rotate about a point x









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

Not linear:

Translation

Affine:

Composition of linear transform + translation

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

Euclidean: (Isometries)

Preserve distance between points (preserves length)

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

Translation
Rotation
Reflection

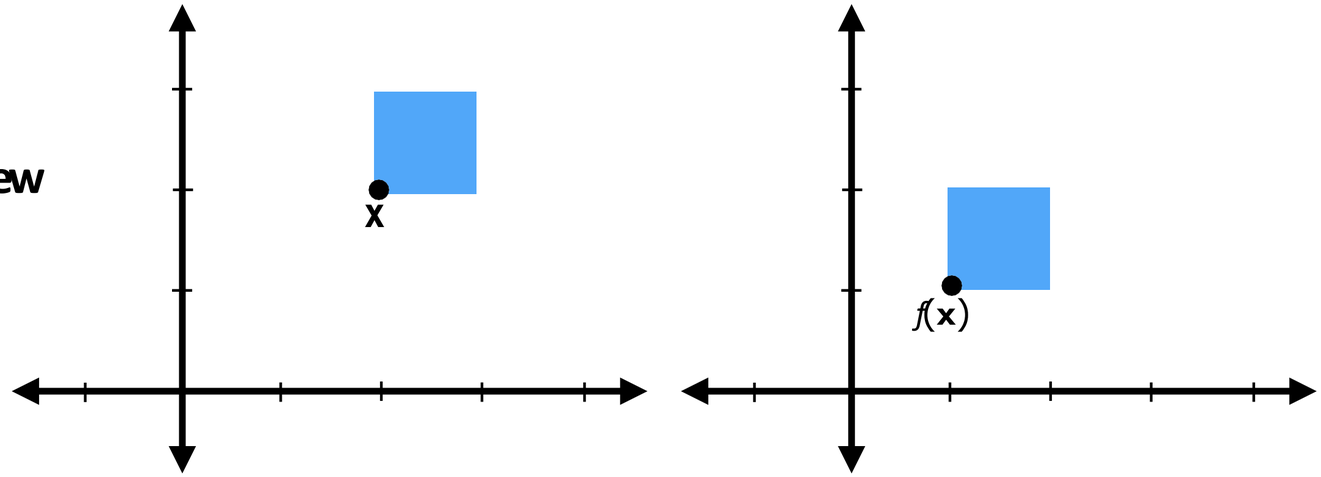
"Rigid body" transformations are distance-preserving motions that also preserve *orientation* (i.e., does not include reflection)

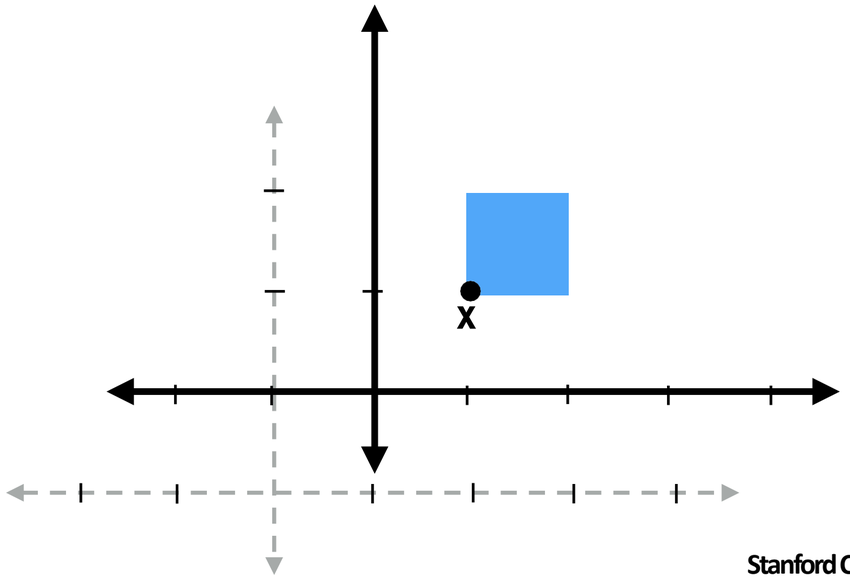
Representing Transformations in Coordinates

Another way to think about transformations: change of coordinates

Interpretation of transformations so far in this lecture: *transformations modify (move) points*

Point x moved to new position $f(x)$ so it has new coordinates in same coordinate space.





Alternative interpretation:

Transformations induce of change of coordinate frame: Representation of x changes since point is now expressed in new coordinates

2D matrix multiplication

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

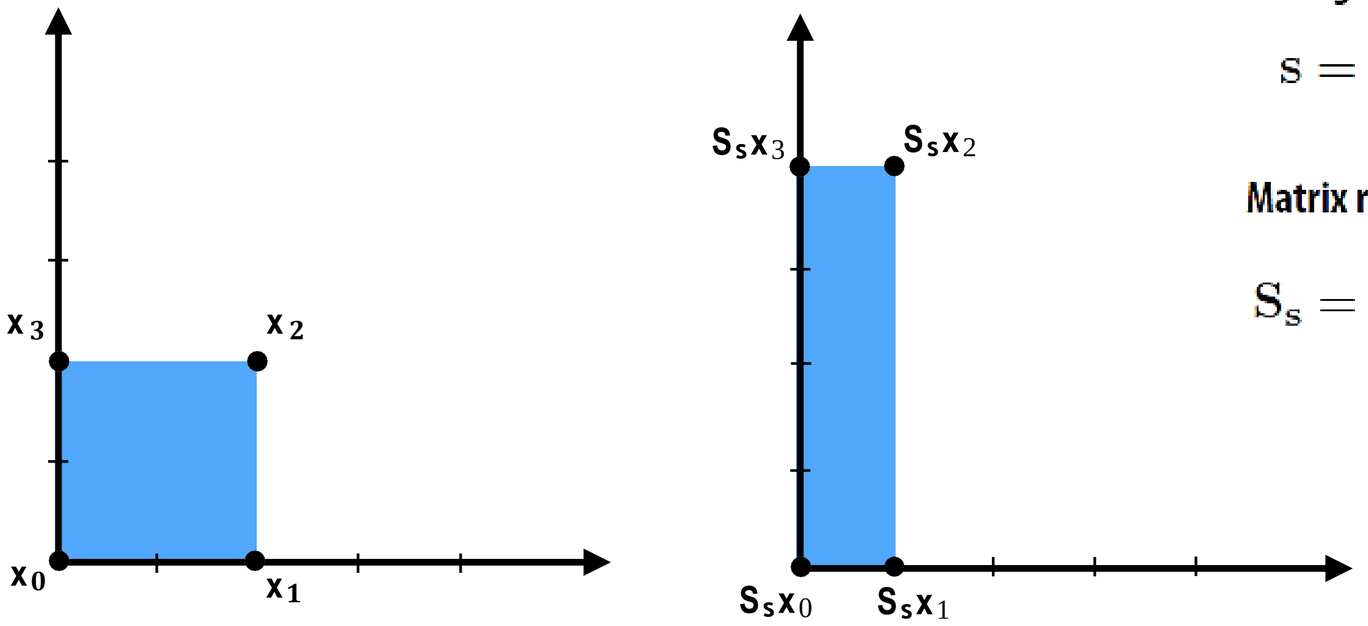
$$x \begin{bmatrix} a \\ c \end{bmatrix} + y \begin{bmatrix} b \\ d \end{bmatrix} =$$

$$\begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

- **Matrix multiplication is linear combination of columns**
- **Encodes a linear map!**

Linear transformations in 2D can be represented as 2x2 matrices

Consider non-uniform scale: $\mathbf{S_s} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$



Scaling amounts in each direction:

$$\mathbf{s} = \begin{bmatrix} 0.5 & 2 \end{bmatrix}^T$$

Matrix representing scale transform:

$$\mathbf{S_s} = \begin{bmatrix} 0.5 & 0 \\ 0 & 2 \end{bmatrix}$$

Scaling Example

- Consider Square with left-bottom corner at (2,2) and right top corner at (6,6) apply the transformation which makes its size half.

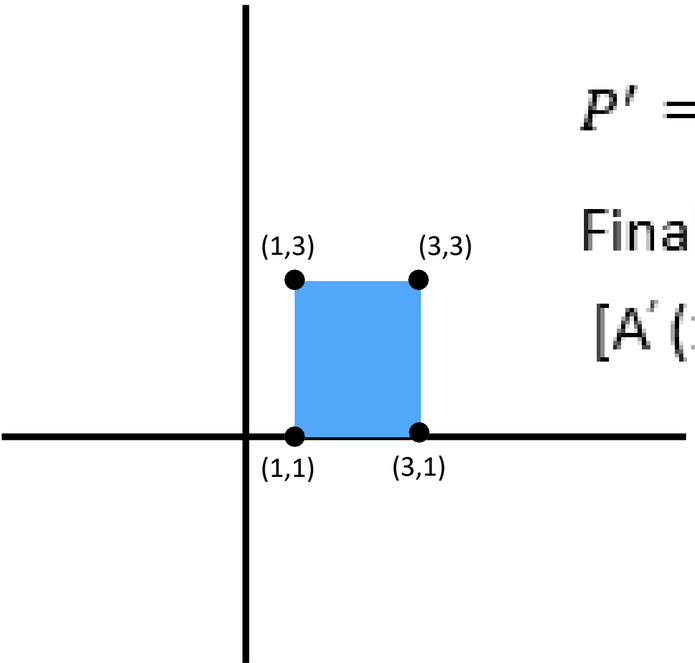
$$P' = S \cdot P$$

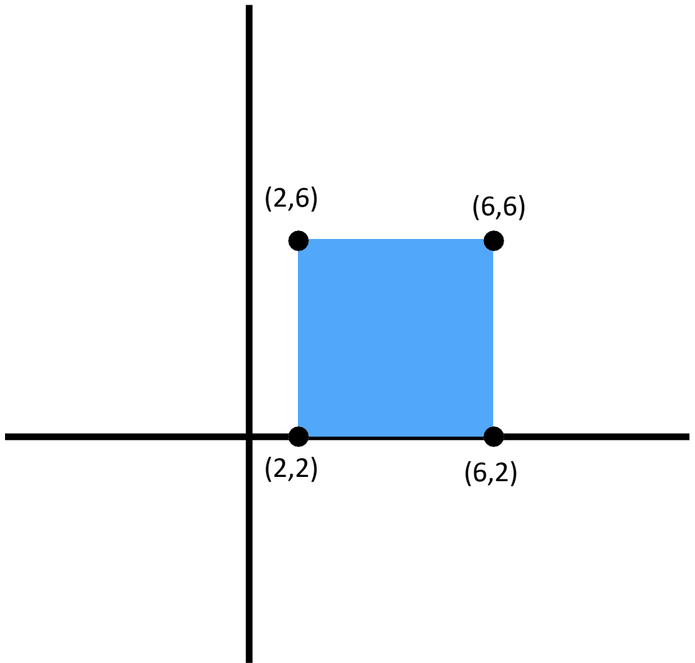
$$P' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 3 & 3 & 1 \\ 1 & 1 & 3 & 3 \end{bmatrix}$$

Final coordinate after scaling are:

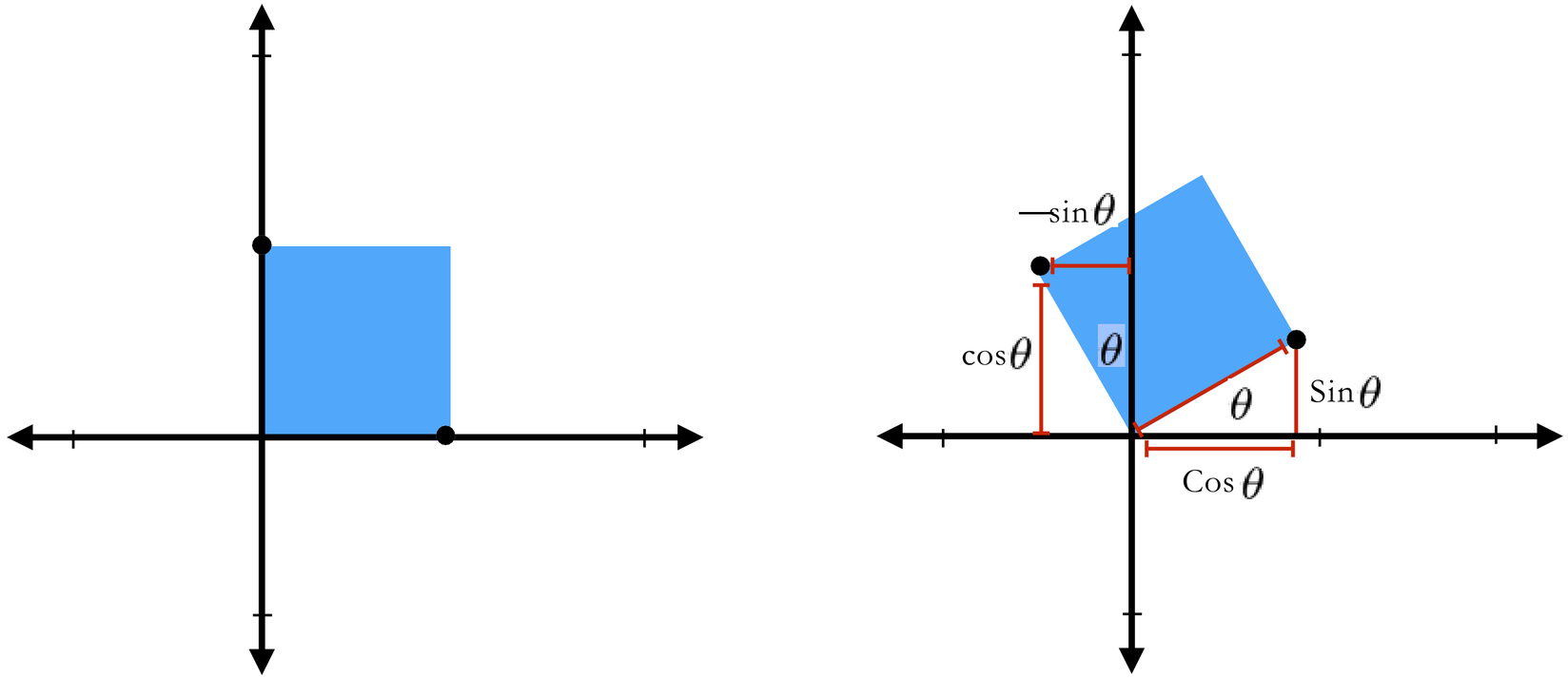
$[A'(1, 1), B'(3, 1), C'(3, 3), D'(1, 3)]$





Rotation matrix (2D):

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



Rotation Example

- Locate the new position of the triangle [A(5,4), B(8,3), C(8,8)] after its rotation by 90^o clockwise the origin.

$$P' = R \cdot P$$

$$P' = \begin{bmatrix} \cos(-90) & -\sin(-90) \\ \sin(-90) & \cos(-90) \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$$

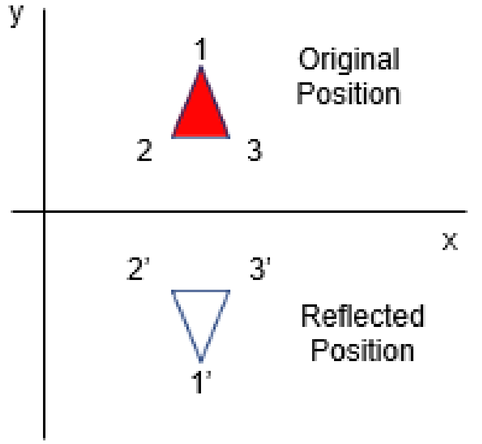
$$P' = \begin{bmatrix} 4 & 3 & 8 \\ -5 & -8 & -8 \end{bmatrix}$$

Final coordinates after rotation are: [A′ (4, -5), B′ (3, -8), C′ (8, -8)]

Reflection about

- For reflection about the line
  - Y=0, the x axis

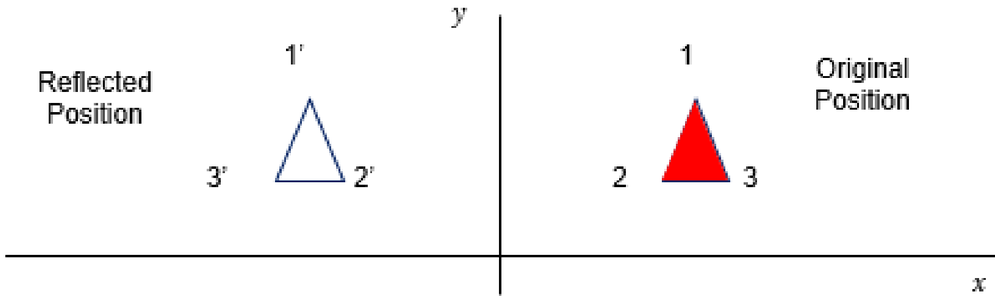
$$Ref_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



For reflection about the line

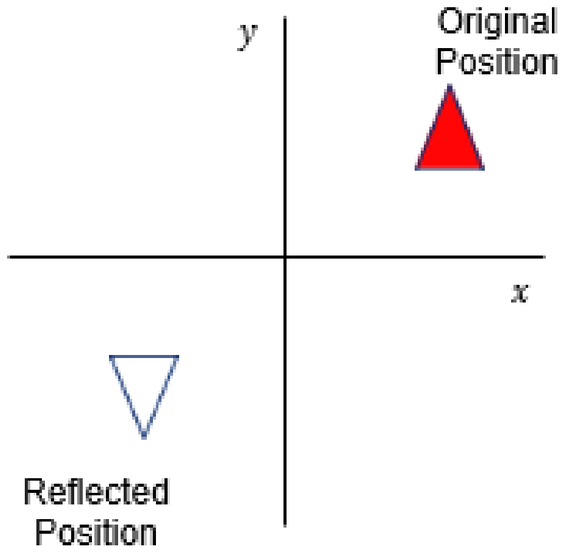
X=0, the Y axis

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



For reflection about the ***Origin.***

$$Ref_{origin} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Reflection Example

- Find the coordinates after reflection of the triangle [A(10,10), B(15,

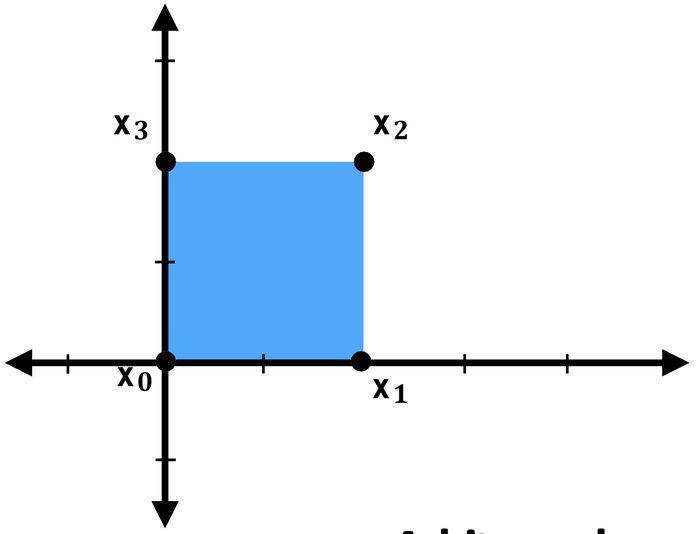
$$P' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 & 15 & 20 \\ 10 & 15 & 10 \\ 1 & 1 & 1 \end{bmatrix}$$

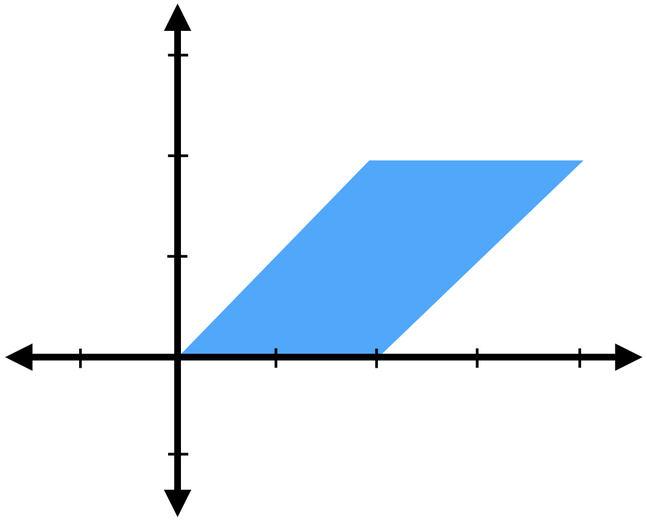
$$P' = \begin{bmatrix} 10 & 15 & 20 \\ -10 & -15 & -10 \\ 1 & 1 & 1 \end{bmatrix}$$

Final coordinate after reflection are:

[A′ (10, -10), B′ (15, -15), C′ (20, -10)]

Shear



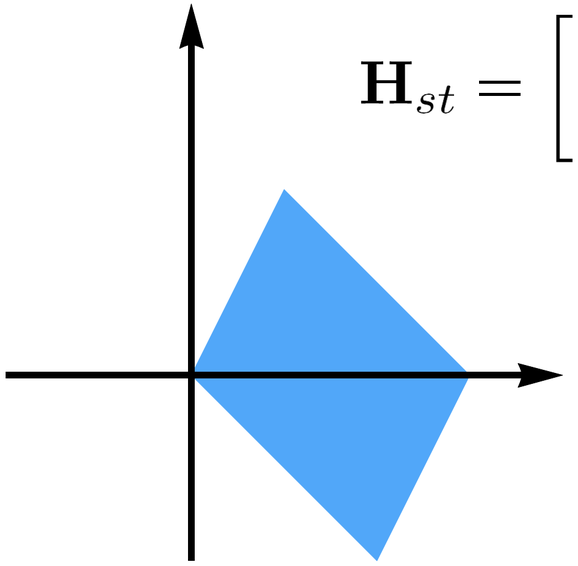


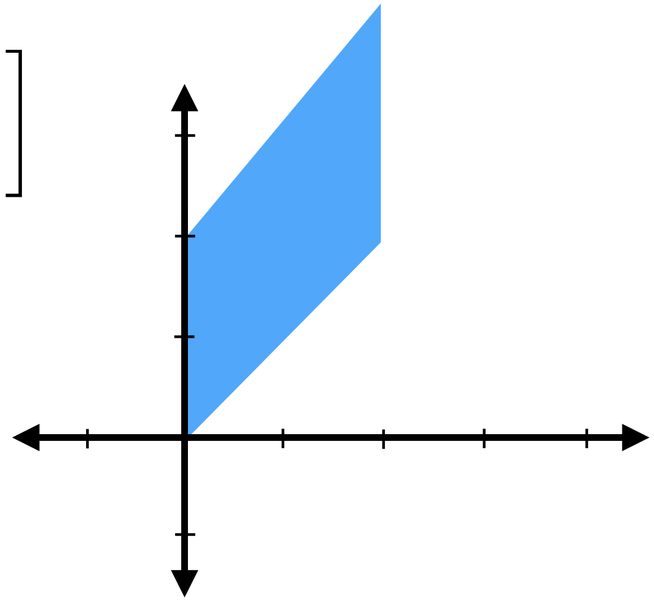
Shear in x:

$$\mathbf{H}_{xs} = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}$$

Arbitrary shear:

$$\mathbf{H}_{st} = \begin{bmatrix} 1 & s \\ t & 1 \end{bmatrix}$$

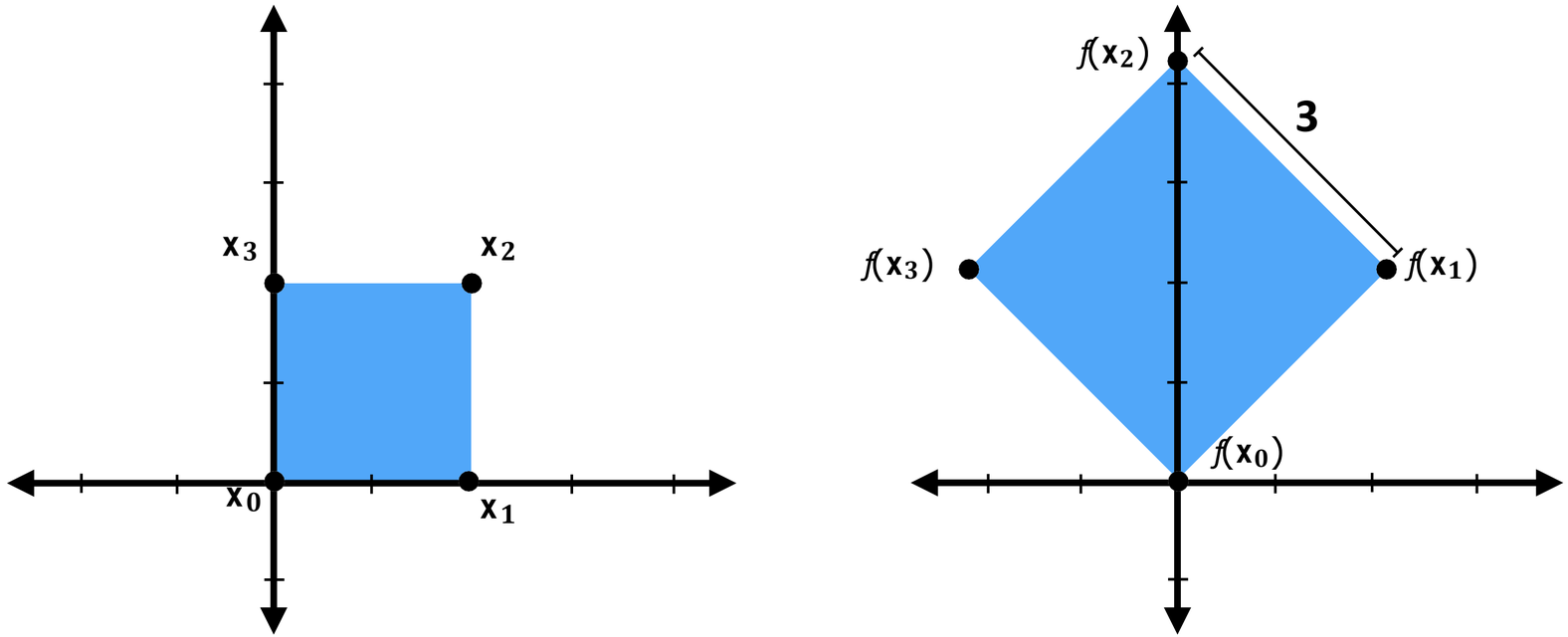




Shear in y:

$$\mathbf{H}_{ys} = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}$$

How do we compose linear transformations?



Compose linear transformations via matrix multiplication.

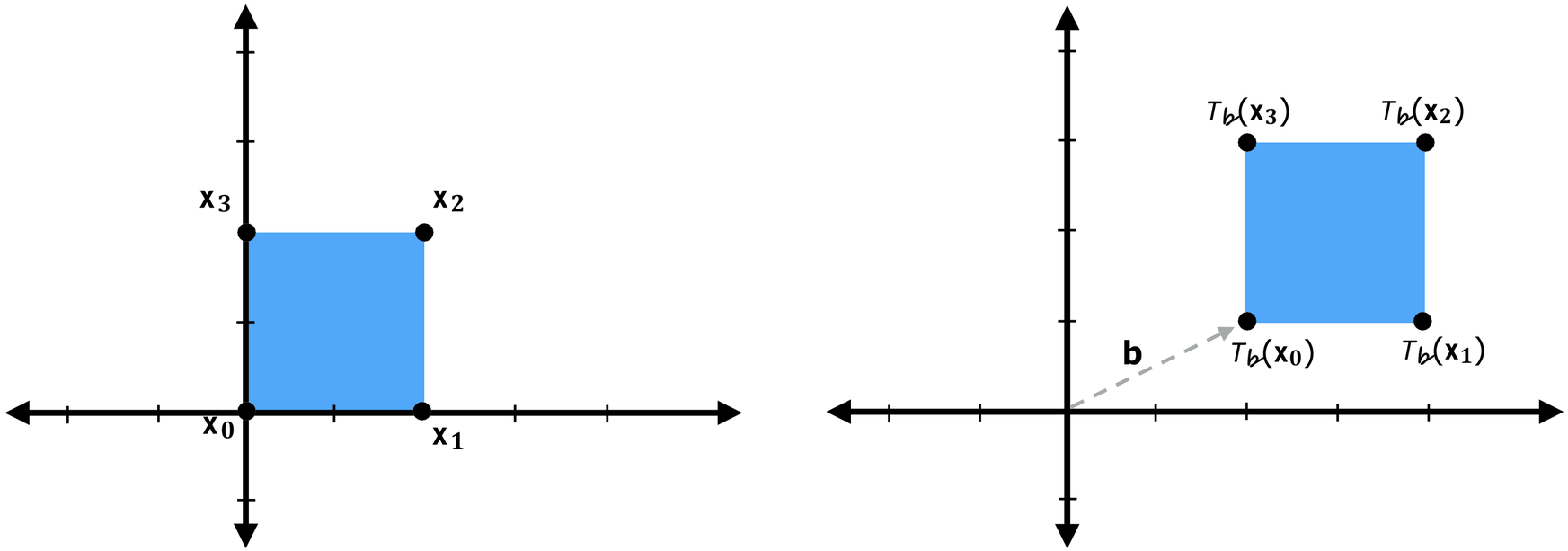
This example: uniform scale, followed by rotation

$$f(\mathbf{x}) = R_{30}\mathbf{S}_{[1.5,1.5]}\mathbf{x} = \mathbf{M}\,\mathbf{x}$$

Enables simple, efficient implementation: reduce complex chain of transformations to a single matrix multiplication!

How do we deal with translation? (Not linear)

$$T_b(x) = x + b$$



Recall: translation is not a linear transform

⟶ Output coefficients are not a linear combination of input coefficients

⟶ Translation operation cannot be represented by a 2x2 matrix

$$x_{out_x} = x_x + b_x$$

$$x_{out_y} = x_y + b_y$$

Translation math

2D homogeneous coordinates (2D-H)

Idea: represent 2D points with THREE values ("homogeneous coordinates")

So the point (x,y) is represented as the 3-vector: $\begin{bmatrix} x & y & 1 \end{bmatrix}^T$

And transformations are represented a 3x3 matrices that transform these vectors.

Recover final 2D coordinates by dividing by "extra" (third) coordinate

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow \begin{bmatrix} x/w \\ y/w \end{bmatrix}$$

Example: scale and rotation in 2D-H coords

- **For transformations that are already linear, not much changes:**

$$\mathbf{S_s} = \begin{bmatrix} \mathbf{S}_x & 0 & 0 \\ 0 & \mathbf{S}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Notice that the last row/column doesn't do anything interesting. E.g., for scaling:

$$\mathbf{S_s} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}_x x \\ \mathbf{S}_y y \\ 1 \end{bmatrix}$$

Now we divide by the 3rd coordinate to get our final 2D coordinates (not too exciting!)

$$\begin{bmatrix} \mathbf{S}_x x \\ \mathbf{S}_y y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{S}_x x/1 \\ \mathbf{S}_y y/1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}_x x \\ \mathbf{S}_y y \end{bmatrix}$$

Translation in 2D homogeneous coordinates

Translation expressed as 3x3 matrix multiplication:

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

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

(remember: just a linear combination of columns!)

Cool: homogeneous coordinates let us encode translations as *linear* transformations!

Example: Translation

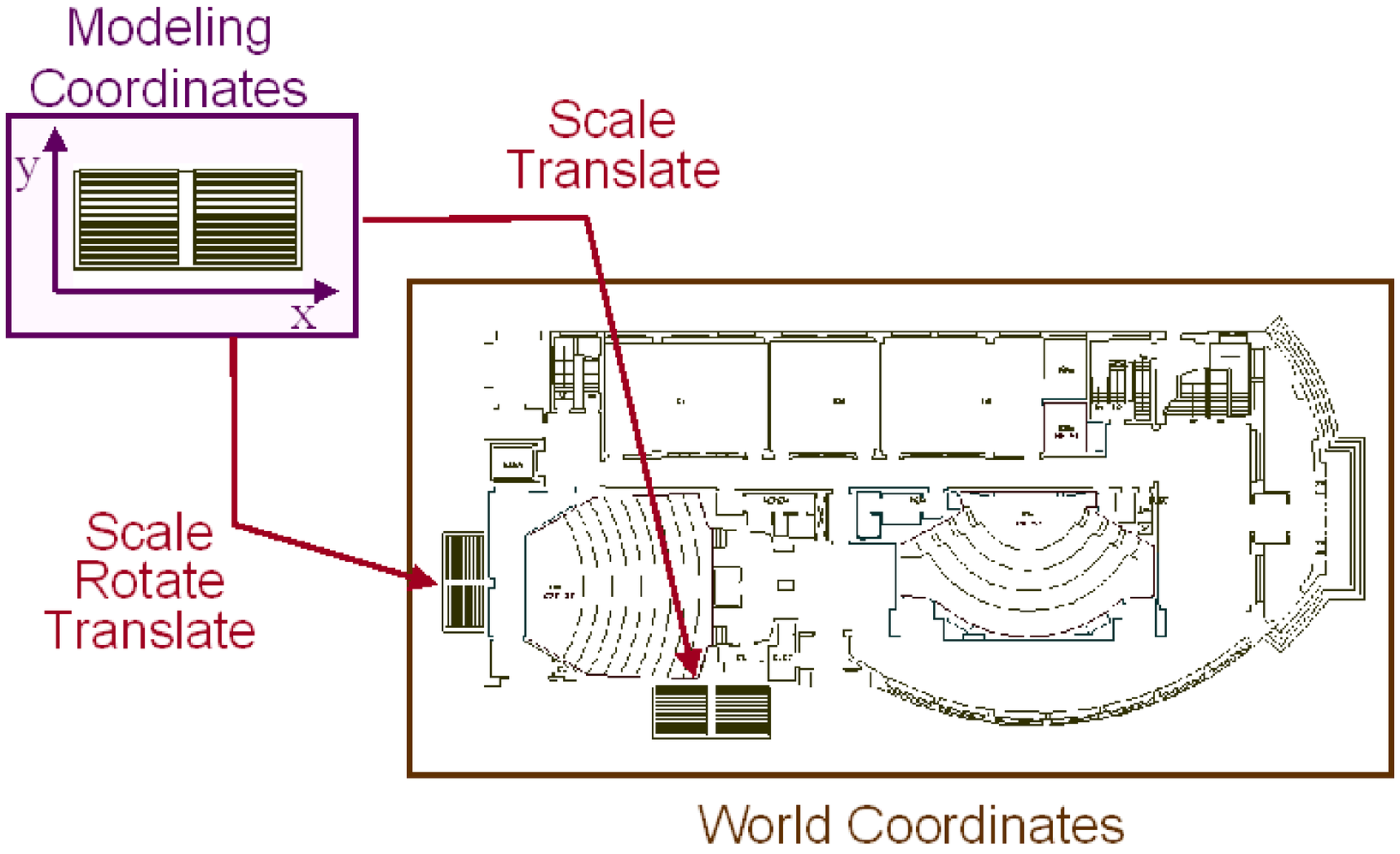
- Let’s translate the point **(3,2)(3, 2)(3,2)** by **t_x=4, ty=5.**

- Convert to homogeneous coordinates:
  - P_h=(3,2,1)

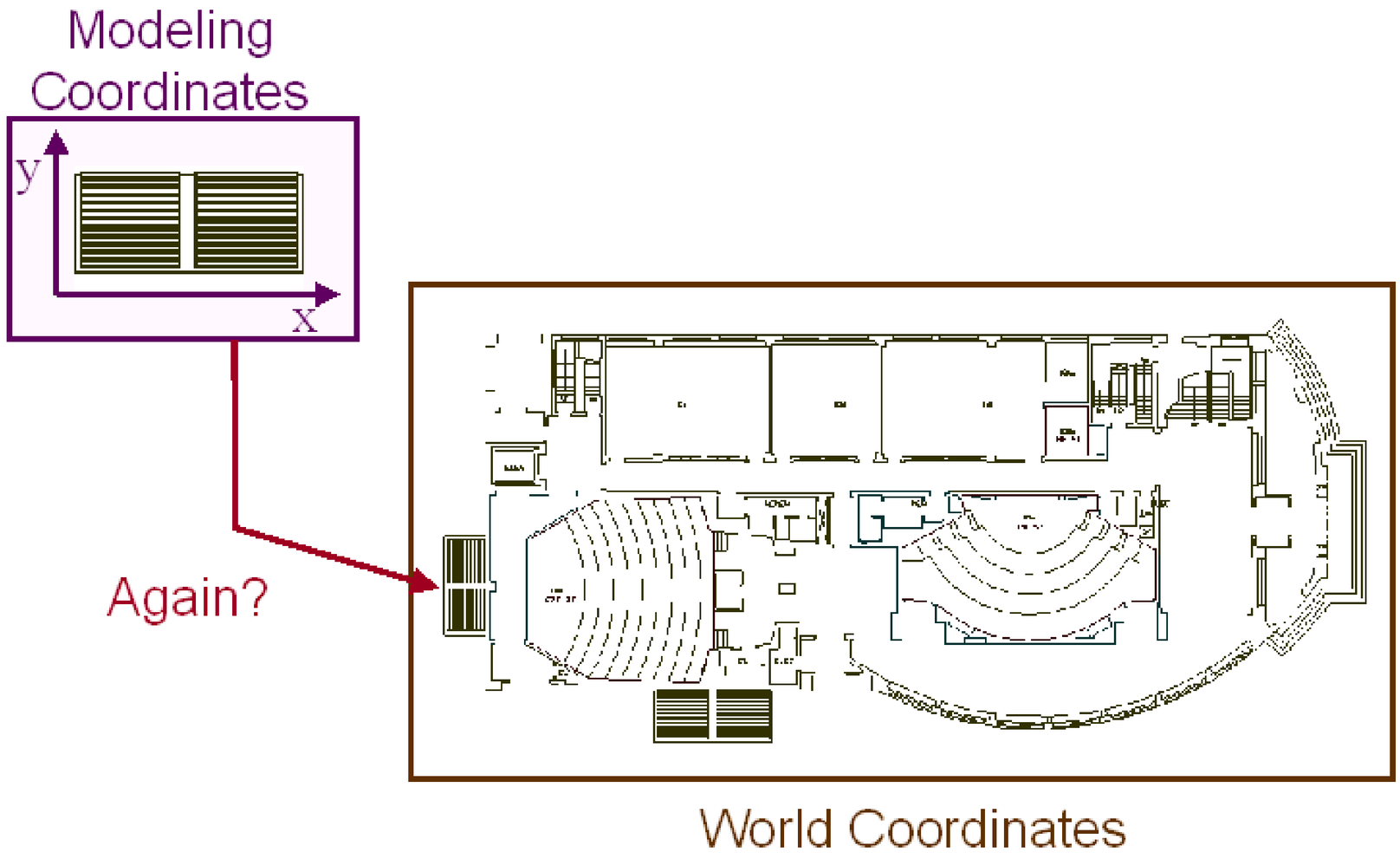
$$T \cdot P_H = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} (1 \times 3) + (0 \times 2) + (4 \times 1) \\ (0 \times 3) + (1 \times 2) + (5 \times 1) \\ (0 \times 3) + (0 \times 2) + (1 \times 1) \end{bmatrix} = \begin{bmatrix} 3 + 4 \\ 2 + 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \\ 1 \end{bmatrix}$$

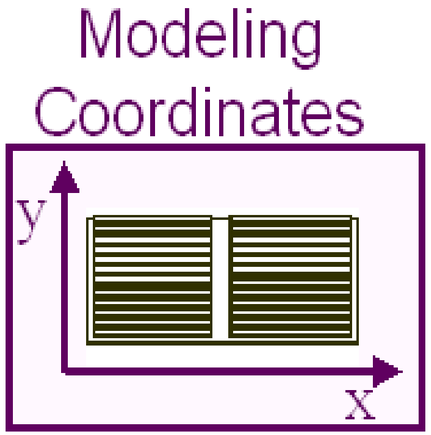
2-D Transformations

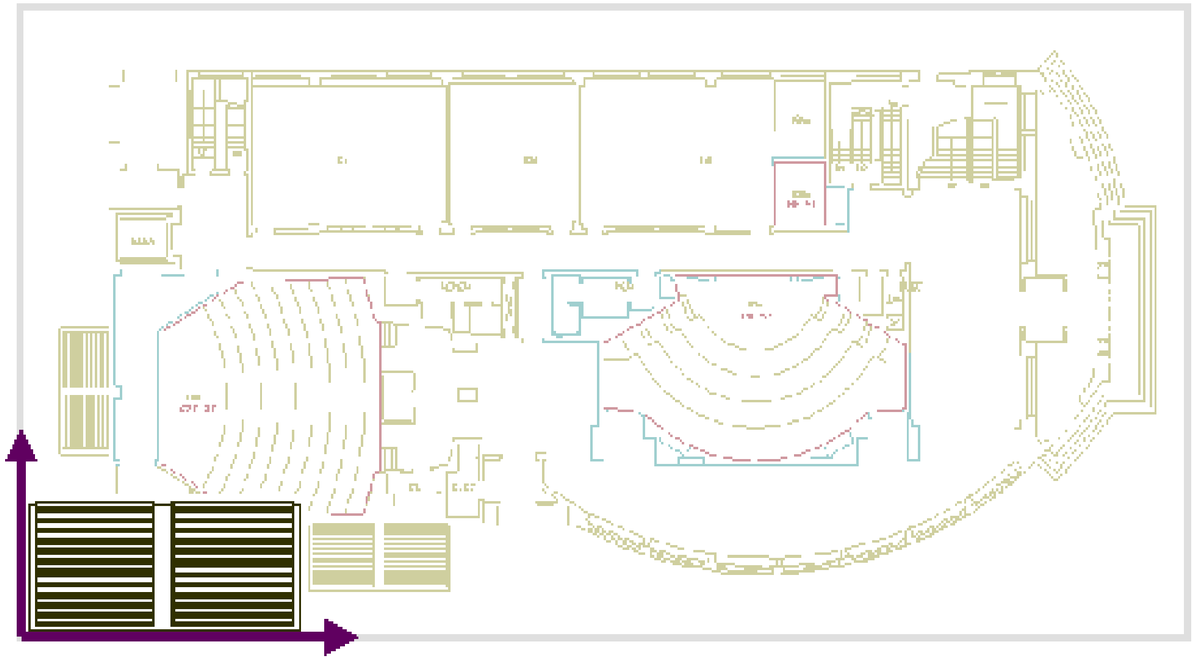


2-D Transformations

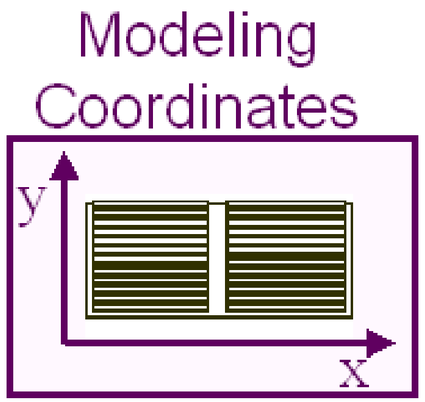


2-D Transformations

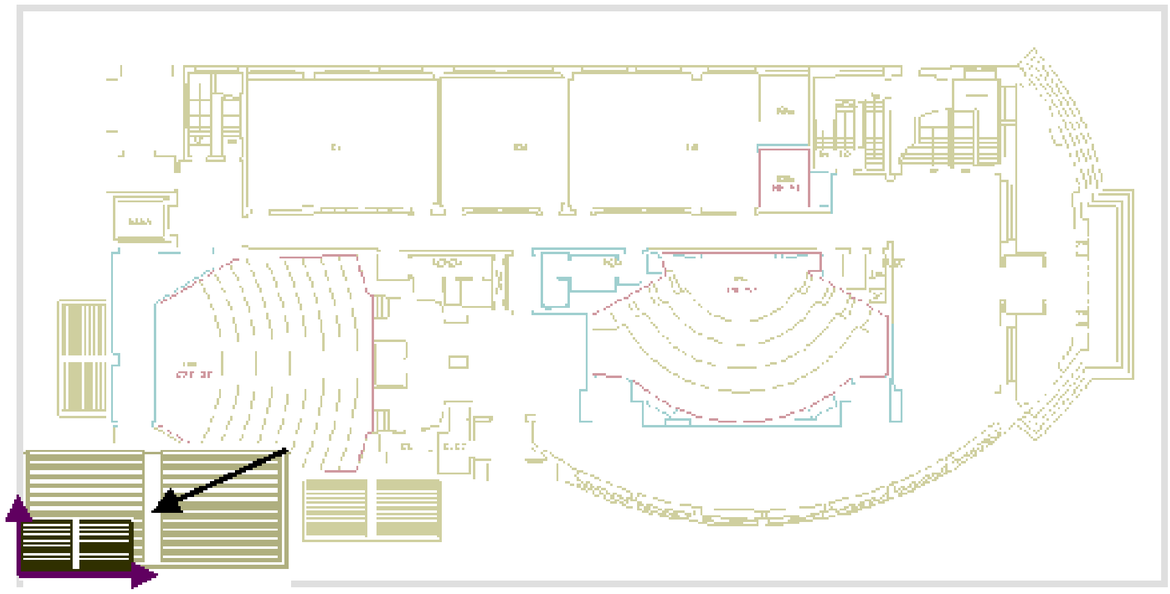




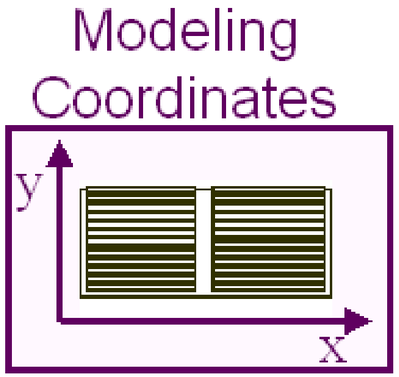
2-D Transformations



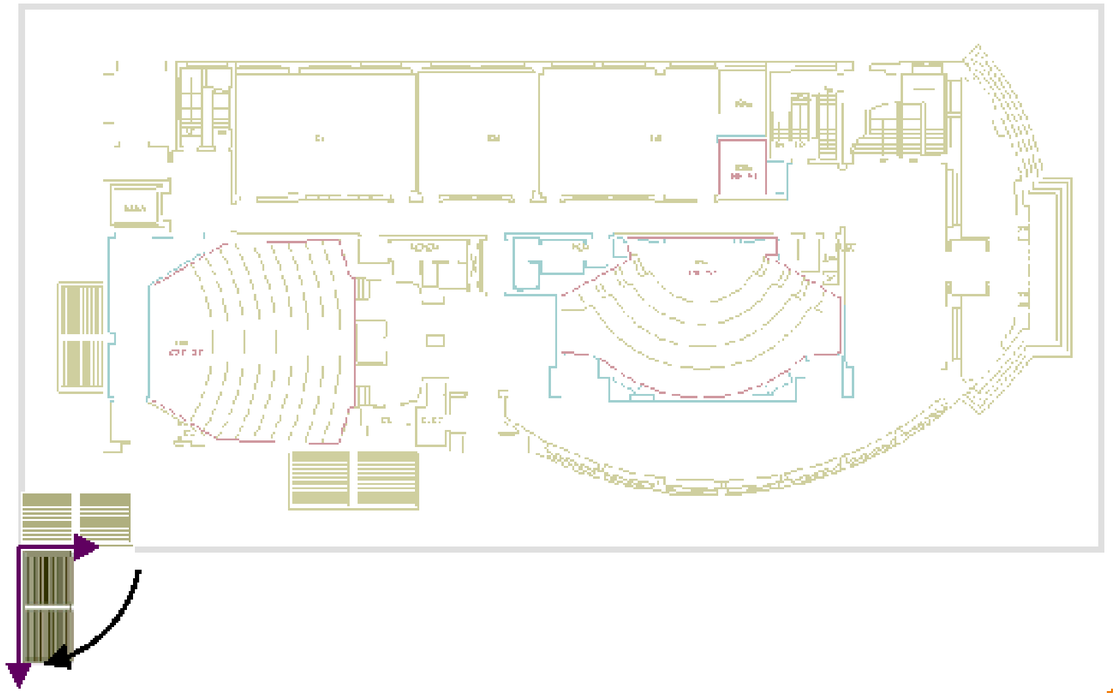
Scale .3, .3



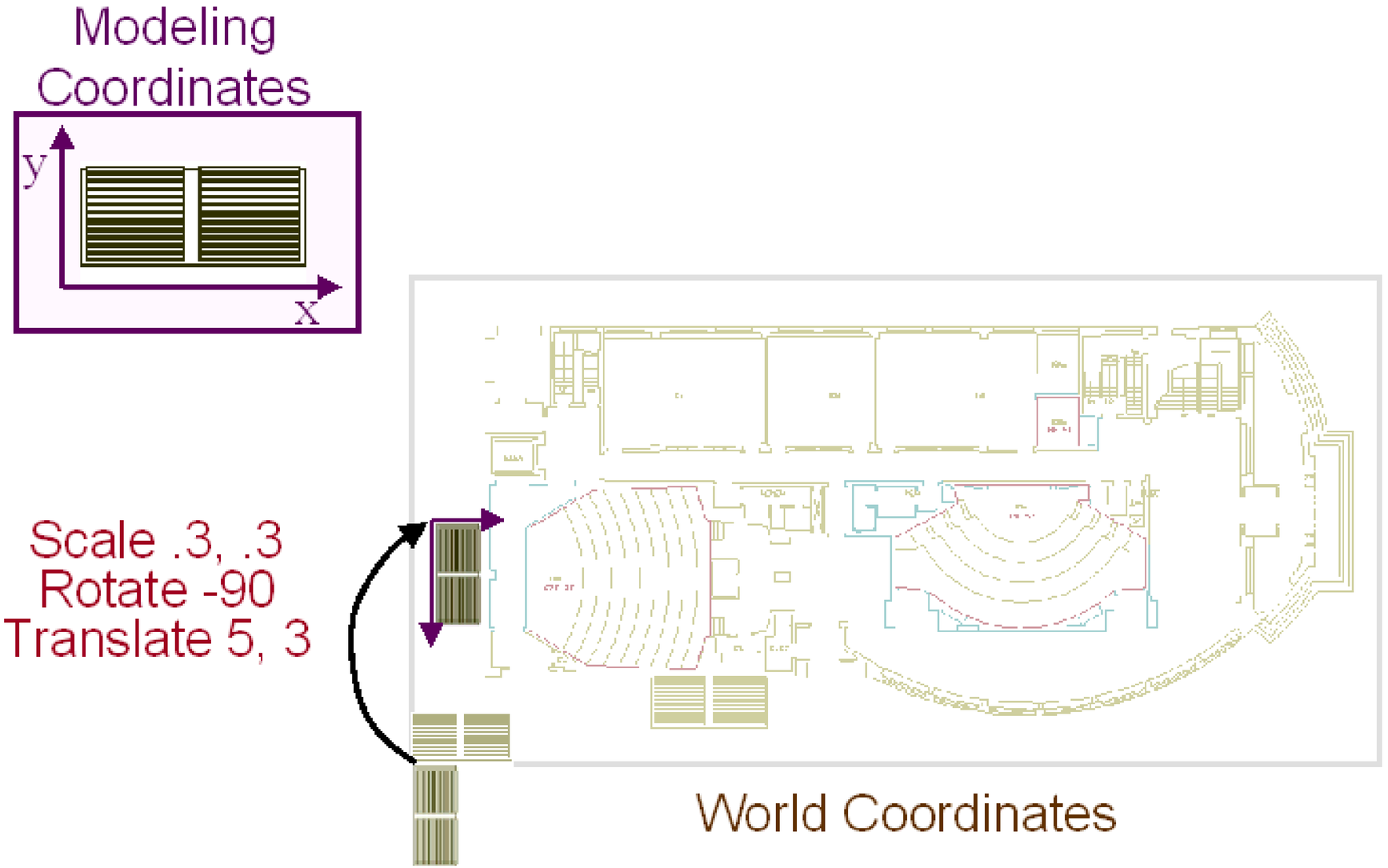
2-D Transformations



Scale .3, .3
Rotate -90

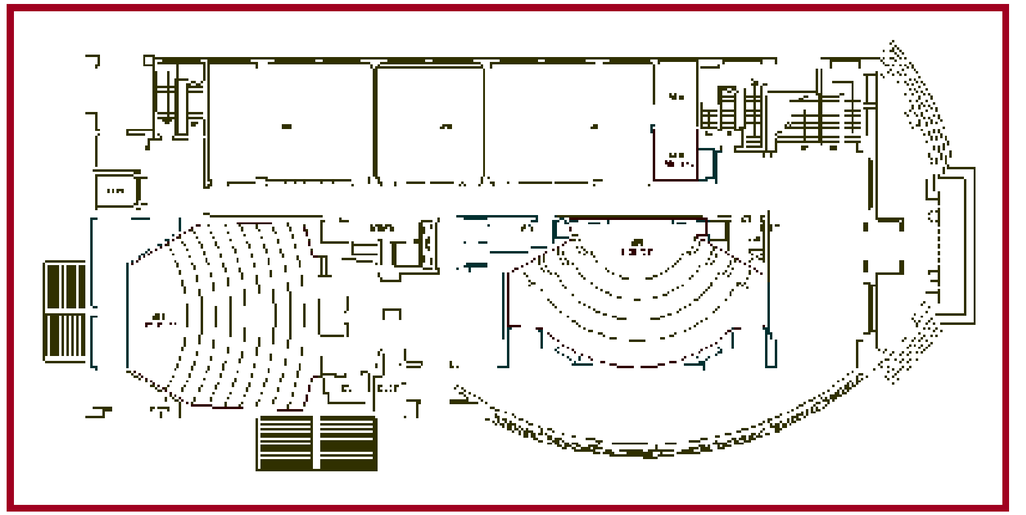


2-D Transformations



2-D Transformations

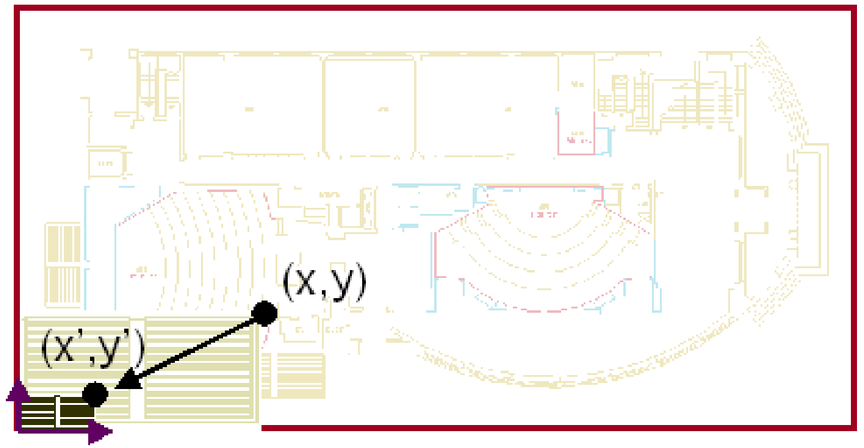
- Translation:
  - $x' = x + t_x$
  - $y' = y + t_y$
- Scale:
  - $x' = x \times S_x$
  - $y' = y \times S_y$
- Shear:
  - $x' = x + Sh_x \times y$
  - $y' = y + Sh_y \times x$
- Rotation:
  - $x' = x \times \cos\theta - y \times \sin\theta$
  - $y' = y \times \sin\theta + y \times \cos\theta$



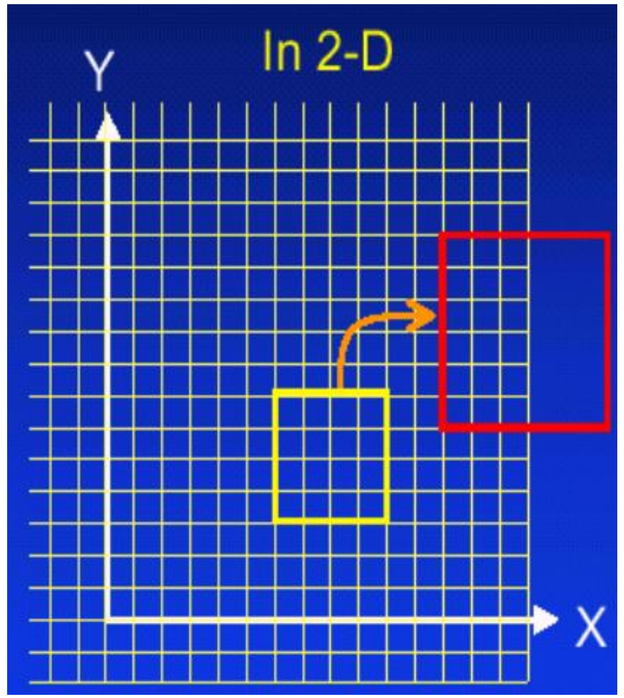
Transformations
can be combined
(with simple algebra)

2-D Transformations

- Translation:
  - $x' = x + t_x$
  - $y' = y + t_y$
- Scale:
  - $x' = x \times S_x$
  - $y' = y \times S_y$
- Shear:
  - $x' = x + Sh_x \times y$
  - $y' = y + Sh_y \times x$
- Rotation:
  - $x' = x \times \cos\theta - y \times \sin\theta$
  - $y' = y \times \sin\theta + y \times \cos\theta$

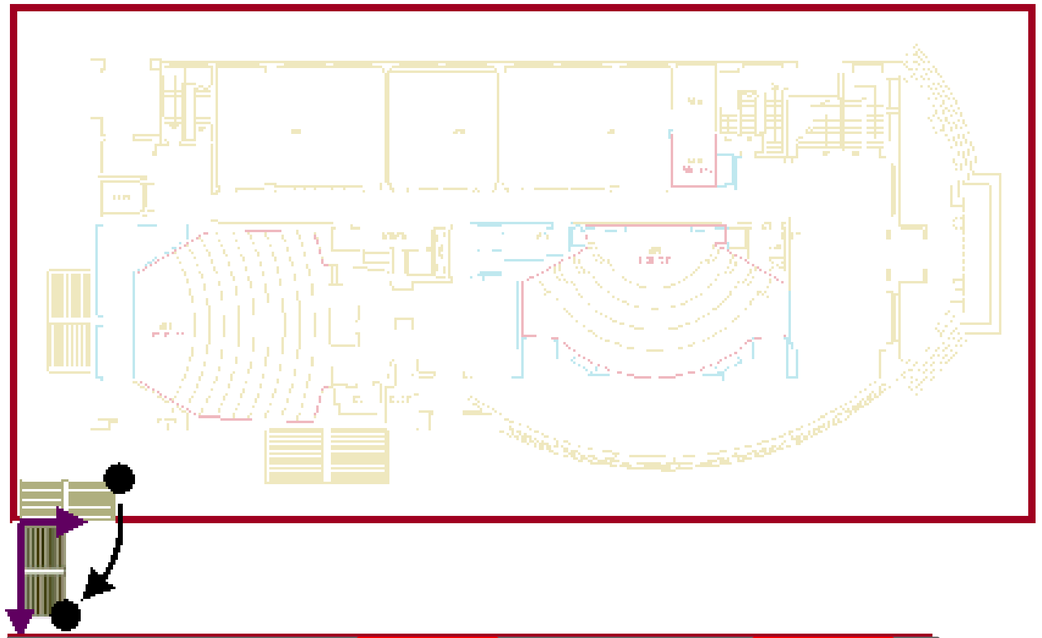


$$x' = x \times S_x$$
$$y' = y \times S_y$$



2-D Transformations

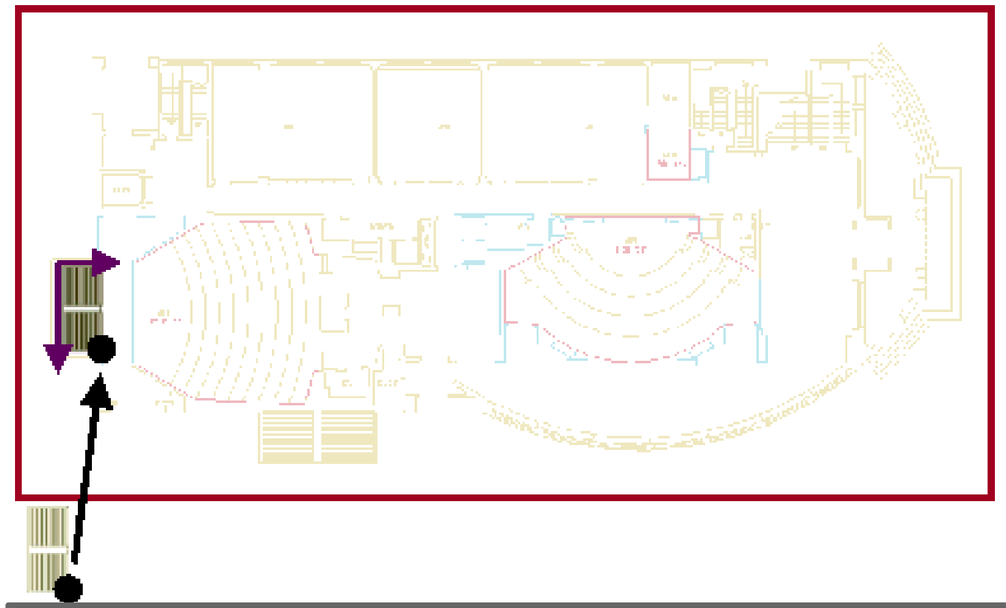
- Translation:
  - $x' = x + t_x$
  - $y' = y + t_y$
- Scale:
  - $x' = x \times S_x$
  - $y' = y \times S_y$
- Shear:
  - $x' = x + Sh_x \times y$
  - $y' = y + Sh_y \times x$
- Rotation:
  - $x' = x \times \cos\theta - y \times \sin\theta$
  - $y' = y \times \sin\theta + y \times \cos\theta$



$$x' = (x \times S_x) \times \cos\theta - (y \times S_y) \times \sin\theta$$
$$y' = (x \times S_x) \times \sin\theta + (y \times S_y) \times \cos\theta$$

2-D Transformations

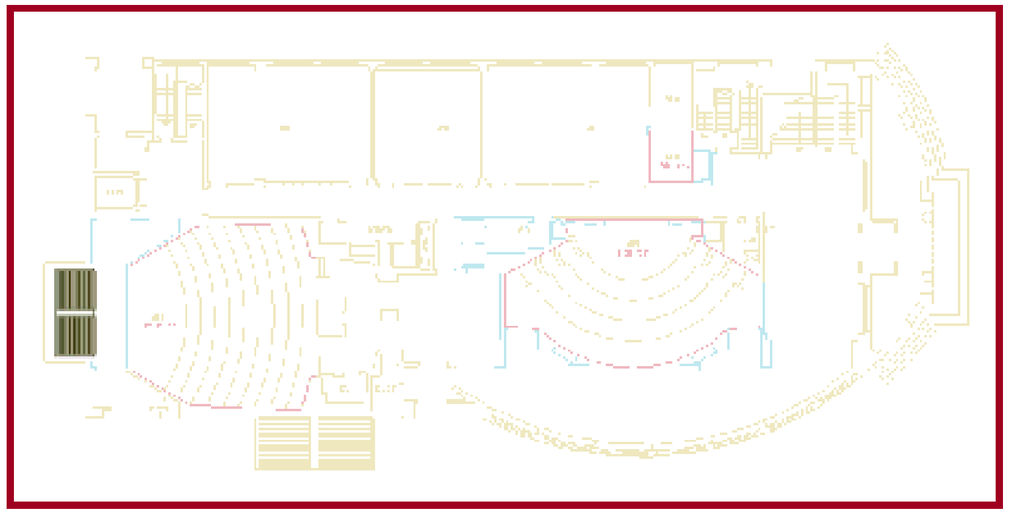
- Translation:
  - $x' = x + t_x$
  - $y' = y + t_y$
- Scale:
  - $x' = x \times S_x$
  - $y' = y \times S_y$
- Shear:
  - $x' = x + Sh_x \times y$
  - $y' = y + Sh_y \times x$
- Rotation:
  - $x' = x \times \cos\theta - y \times \sin\theta$
  - $y' = y \times \sin\theta + y \times \cos\theta$



$$x' = (x \times S_x) \times \cos\theta - (y \times S_y) \times \sin\theta + t_x$$
$$y' = (x \times S_x) \times \sin\theta + (y \times S_y) \times \cos\theta + t_y$$

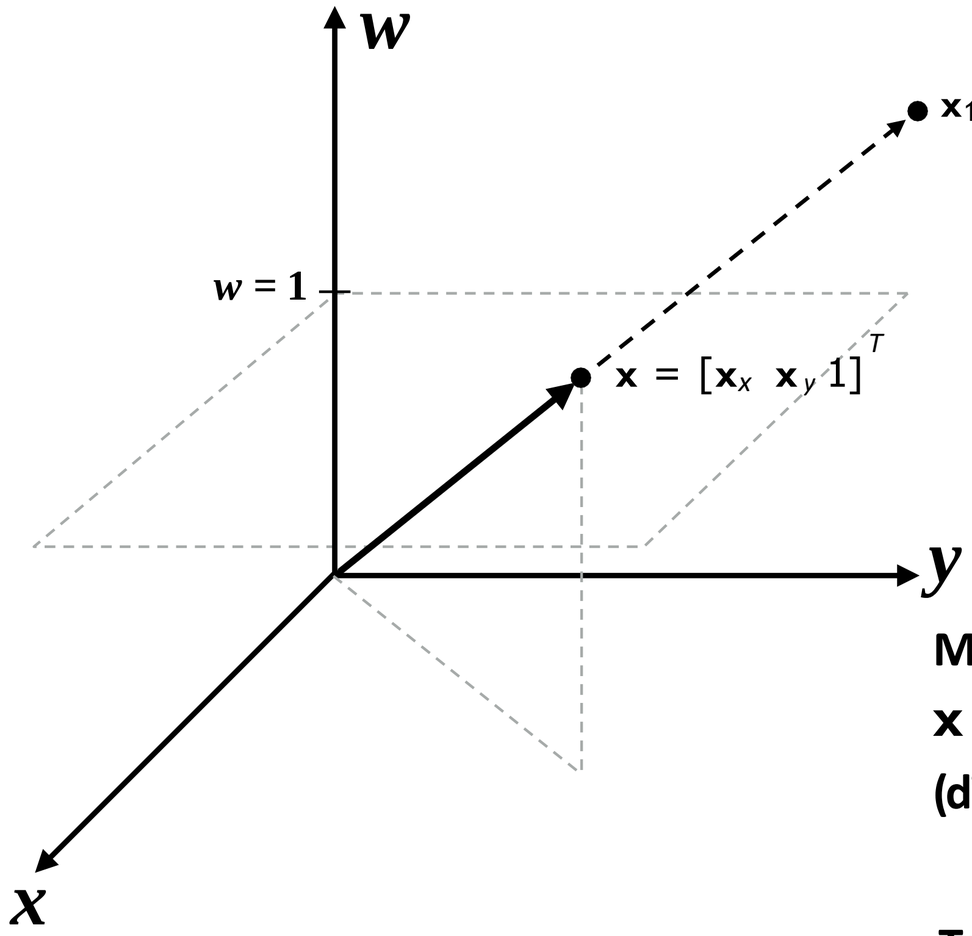
2-D Transformations

- Translation:
  - $x' = x + t_x$
  - $y' = y + t_y$
- Scale:
  - $x' = x \times S_x$
  - $y' = y \times S_y$
- Shear:
  - $x' = x + Sh_x \times y$
  - $y' = y + Sh_y \times x$
- Rotation:
  - $x' = x \times \cos\theta - y \times \sin\theta$
  - $y' = y \times \sin\theta + y \times \cos\theta$



$$x' = (x \times S_x) \times \cos\theta - (y \times S_y) \times \sin\theta + t_x$$
$$y' = (x \times S_x) \times \sin\theta + (y \times S_y) \times \cos\theta + t_y$$

Homogeneous coordinates: some intuition



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

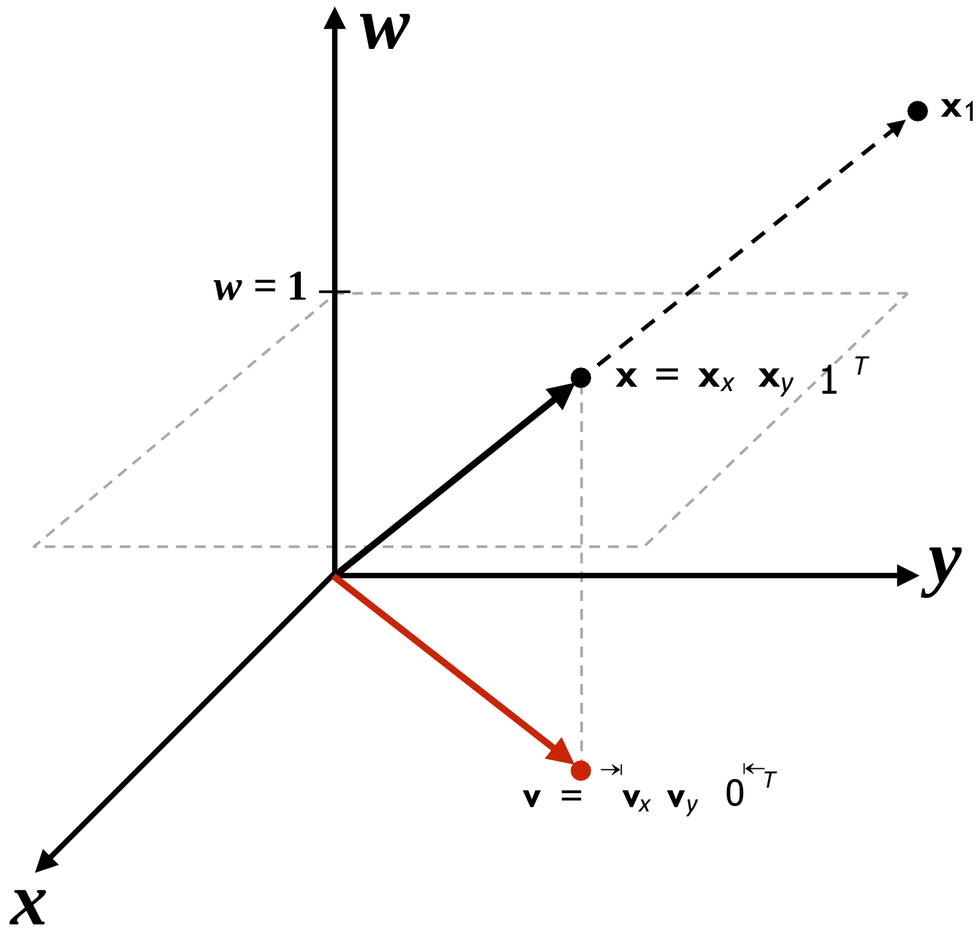
$\mathbf{x}$ and $w\mathbf{x}$ correspond to the same 2D point

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

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

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

Homogeneous coordinates: points vs. vectors



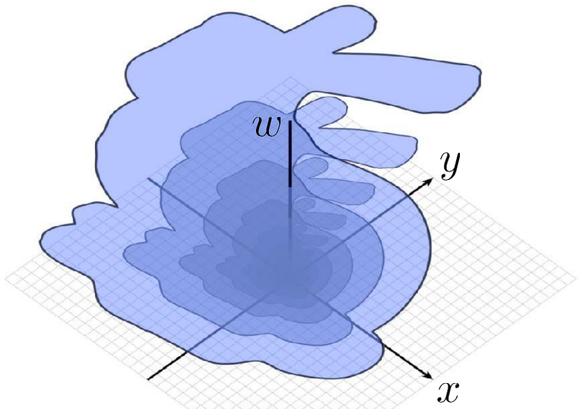
2D-H points with $w = 0$ represent 2D vectors (think: directions are points at infinity)

Unlike 2D, points and directions are distinguishable by their representation in 2D-H

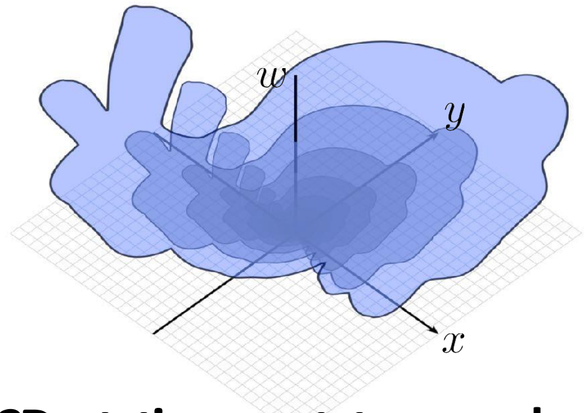
Note: translation does not modify directions:

$$\mathbf{T}_\mathbf{b}\mathbf{v} = \begin{bmatrix} 1 & 0 & \mathbf{b}_x \\ 0 & 1 & \mathbf{b}_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ 0 \end{bmatrix}$$

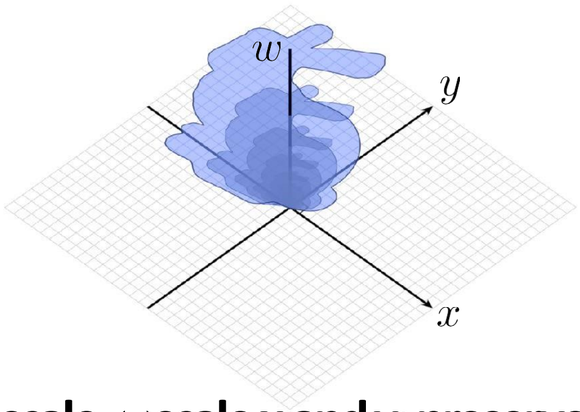
Visualizing 2D transformations in 2D-H



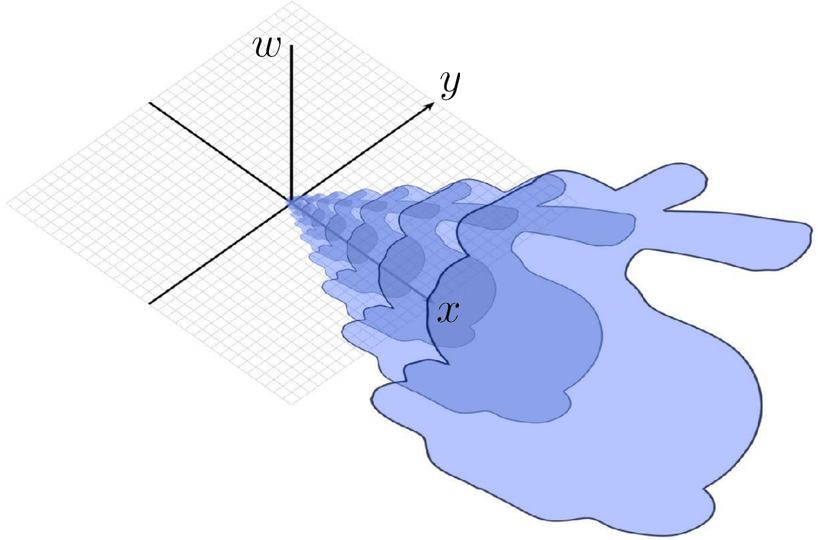
Original shape in 2D can be viewed as many copies, uniformly scaled by w.



2D rotation ↔ rotate around w



2D scale ↔ scale x and y; preserve w



2D translate ↔ shear in 2D-H

(LINEAR!)

Moving to 3D (and 3D-H)

Represent 3D transformations as 3x3 matrices and 3D-H transformations as 4x4 matrices

Scale:

3D

$$\mathbf{S}_\mathrm{s} = \begin{bmatrix} \mathbf{S}_x & 0 & 0 \\ 0 & \mathbf{S}_y & 0 \\ 0 & 0 & \mathbf{S}_z \end{bmatrix}$$

3D-H

$$\mathbf{S}_\mathrm{s} = \begin{bmatrix} \mathbf{S}_x & 0 & 0 & 0 \\ 0 & \mathbf{S}_y & 0 & 0 \\ 0 & 0 & \mathbf{S}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shear (in x, based on y,z position):

$$\mathbf{H}_{x,\mathrm{d}} = \begin{bmatrix} 1 & \mathbf{d}_y & \mathbf{d}_z \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}_{x,\mathrm{d}} = \begin{bmatrix} 1 & \mathbf{d}_y & \mathbf{d}_z & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

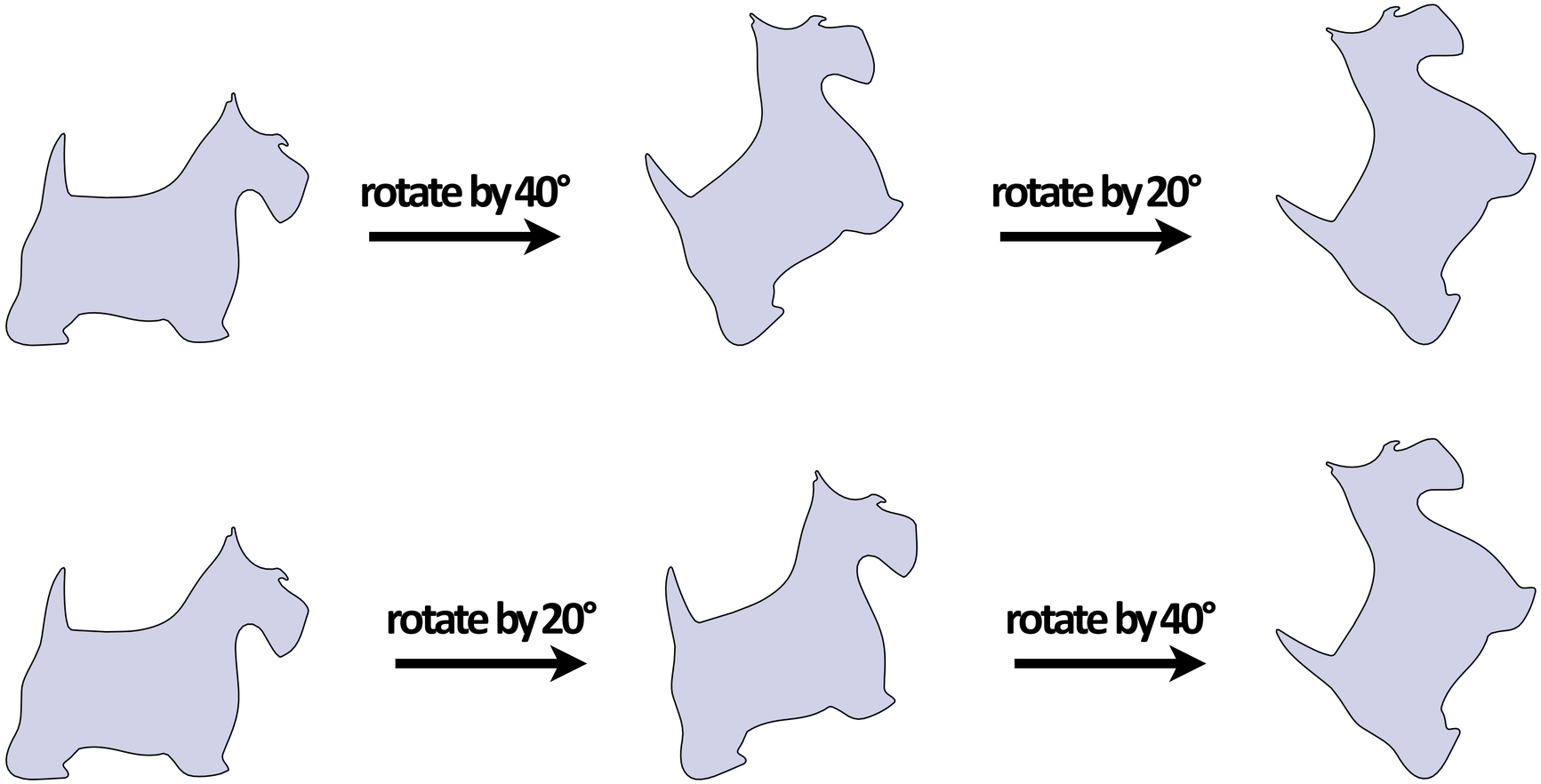
Translate:

3D-H

$$\mathbf{T}_\mathrm{b} = \begin{bmatrix} 1 & 0 & 0 & \mathbf{b}_x \\ 0 & 1 & 0 & \mathbf{b}_y \\ 0 & 0 & 1 & \mathbf{b}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

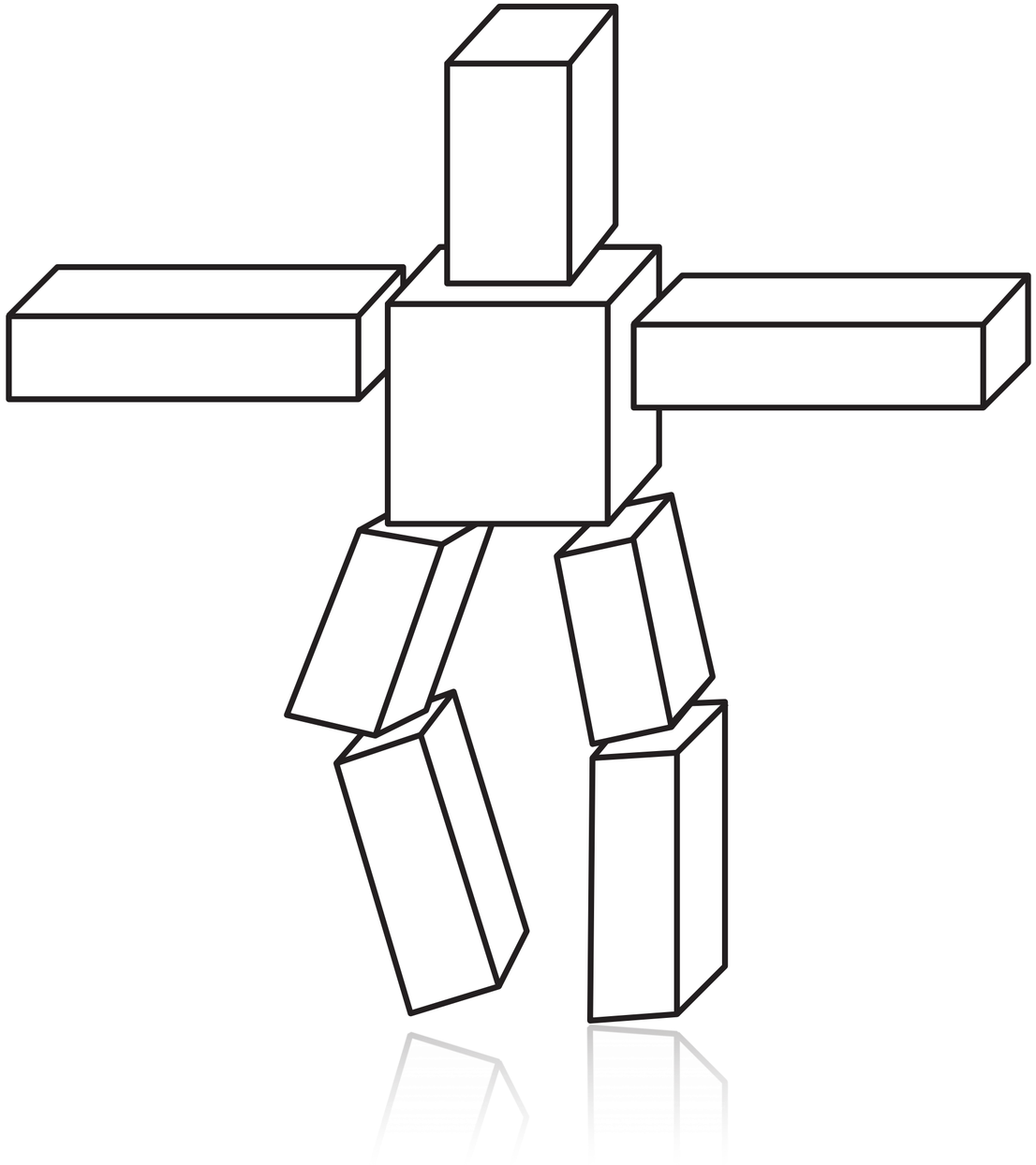
Commutativity of rotations—2D

- **In 2D, order of rotations doesn't matter:**



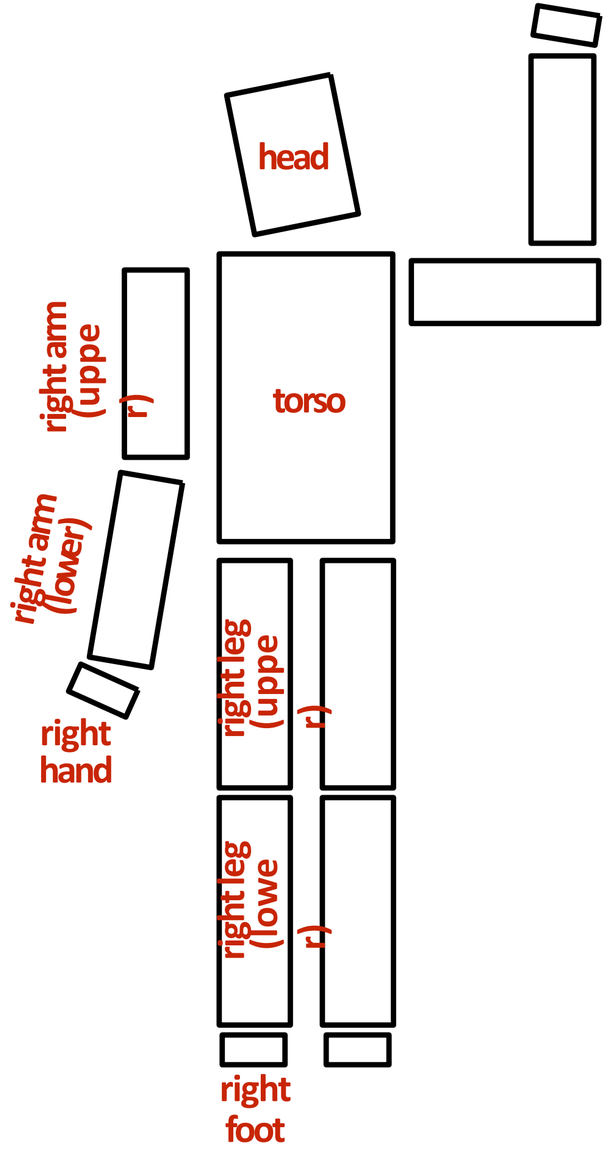
Same result! ("2D rotations commute")

Let's make that cube person…



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 descendent 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); // person centered at (0,10)

drawTorso();

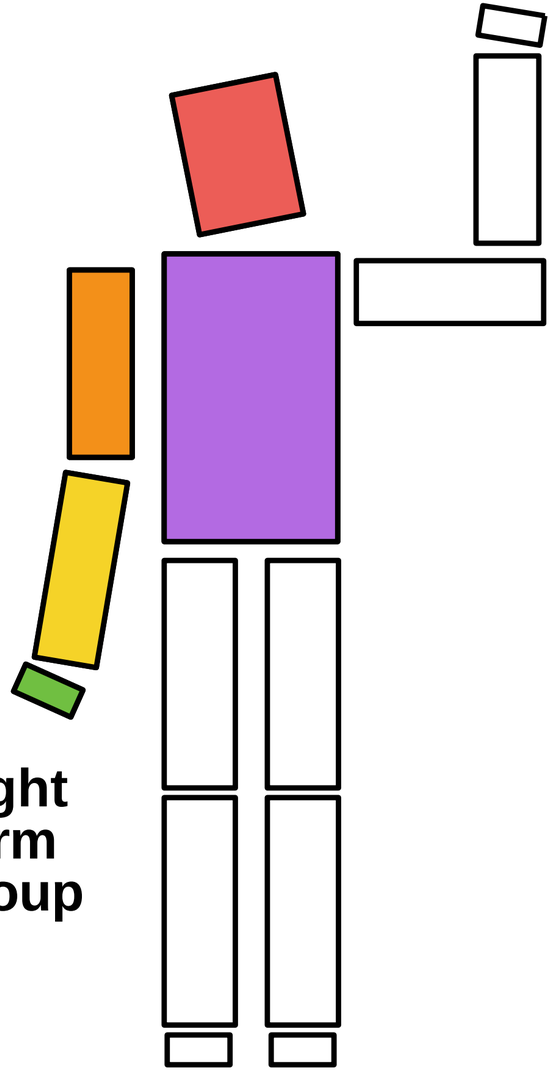
- pushmatrix(); // push a copy of transform onto stack
  translate(0, 5); // right-multiply onto current transform
- rotate(headRotation); // right-multiply onto current transform
  drawHead();
- popmatrix(); // pop current transform off stack
- pushmatrix();

drawLowerArm();
pushmatrix();
translate(0, -3);
rotate(wristRotation);
drawHand();
popmatrix();
popmatrix();
popmatrix();
....

right hand

right lower arm group

right arm group

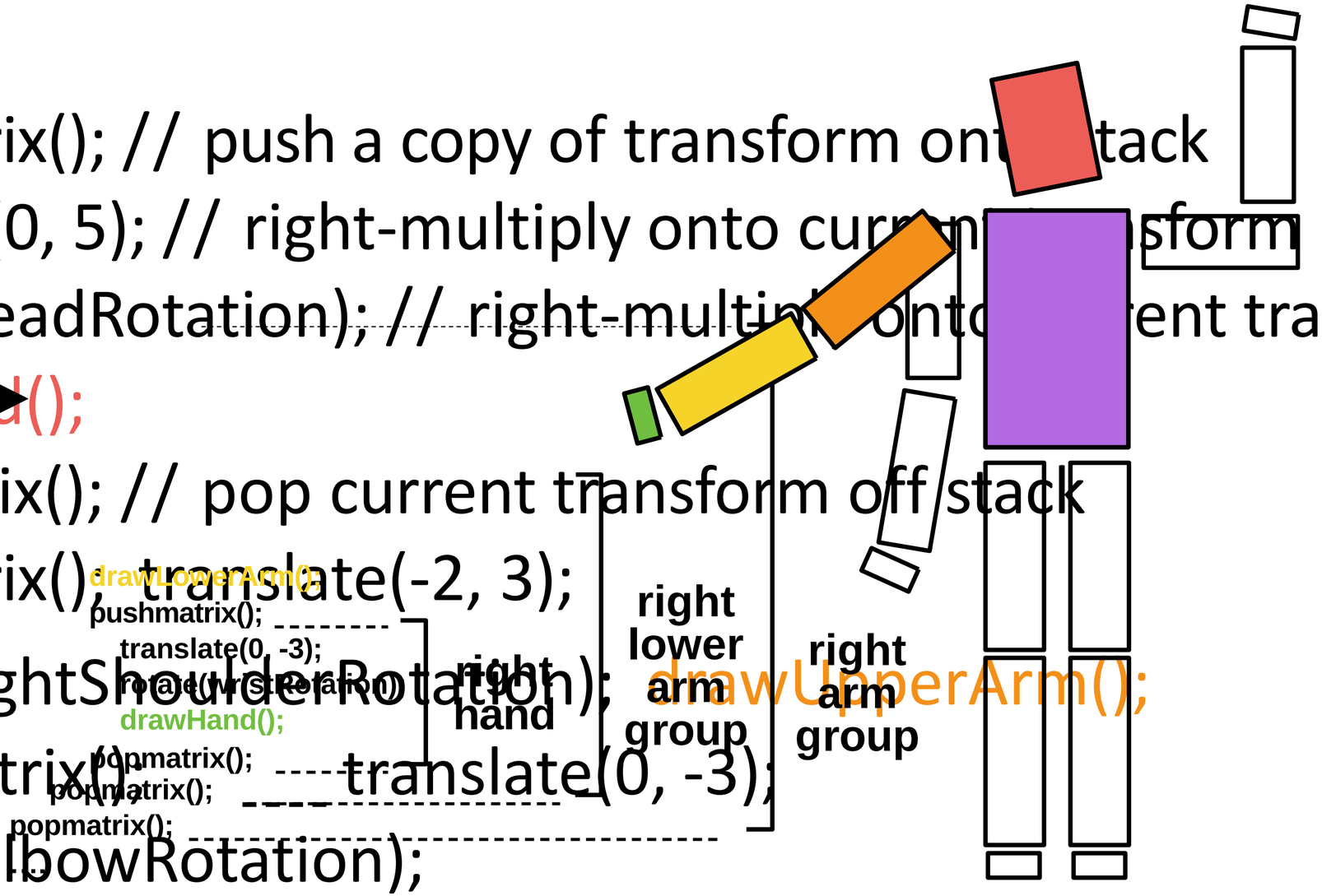


Skeleton - hierarchical representation

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);
  - pushmatrix(); translate(0, -3);
    rotate(elbowRotation);

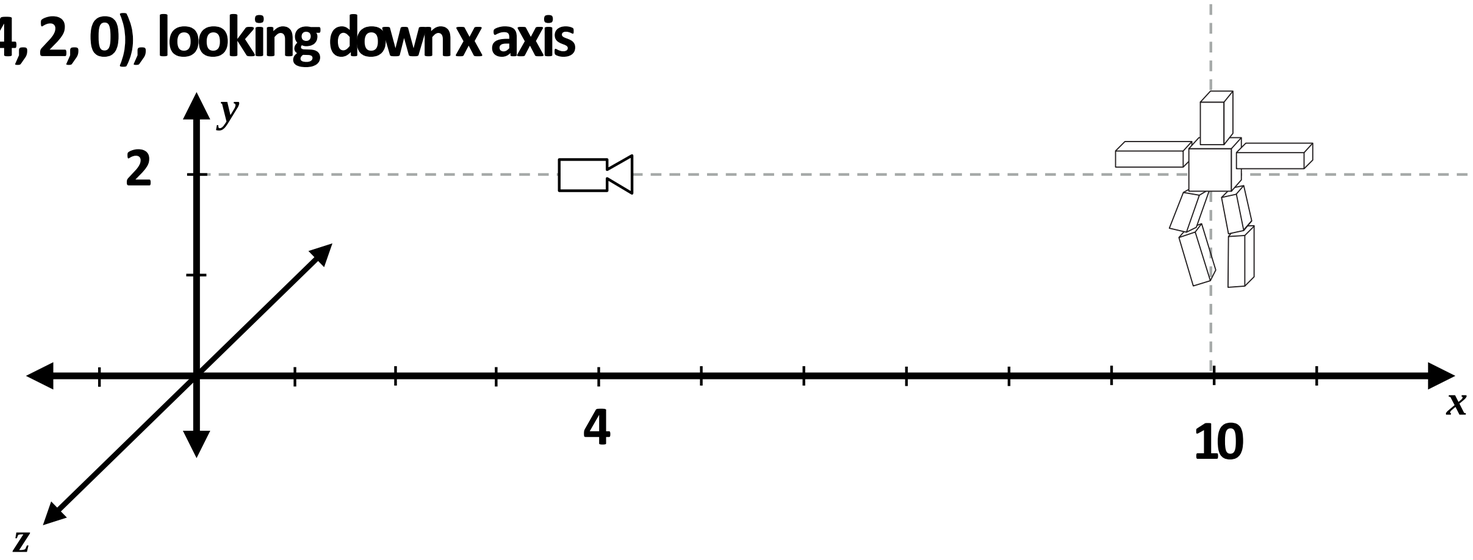


Transforming points into camera-relative coordinates

Example: simple camera transform

Consider object positioned in world at (10, 2, 0)

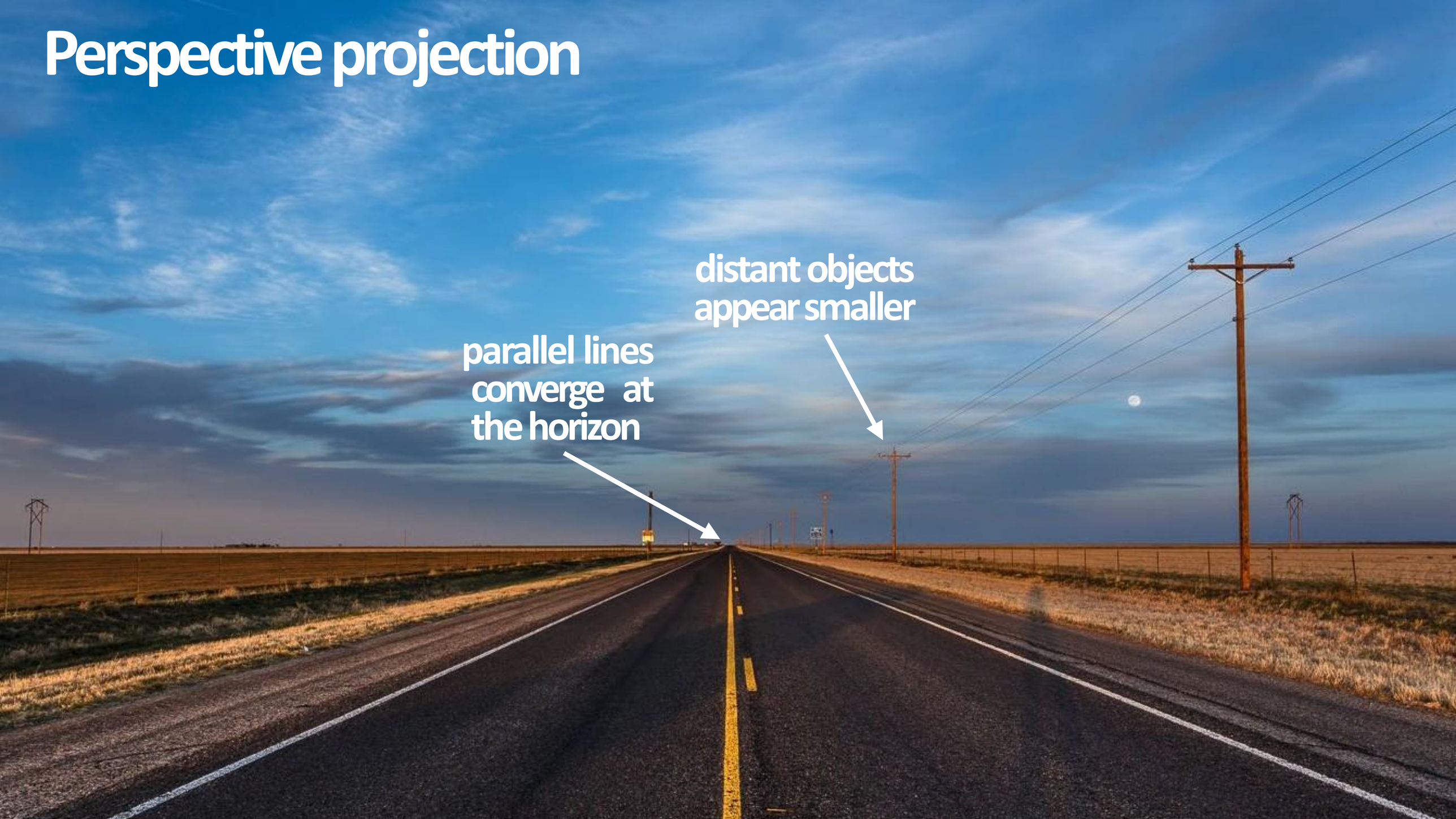
Consider camera at (4, 2, 0), looking down x axis



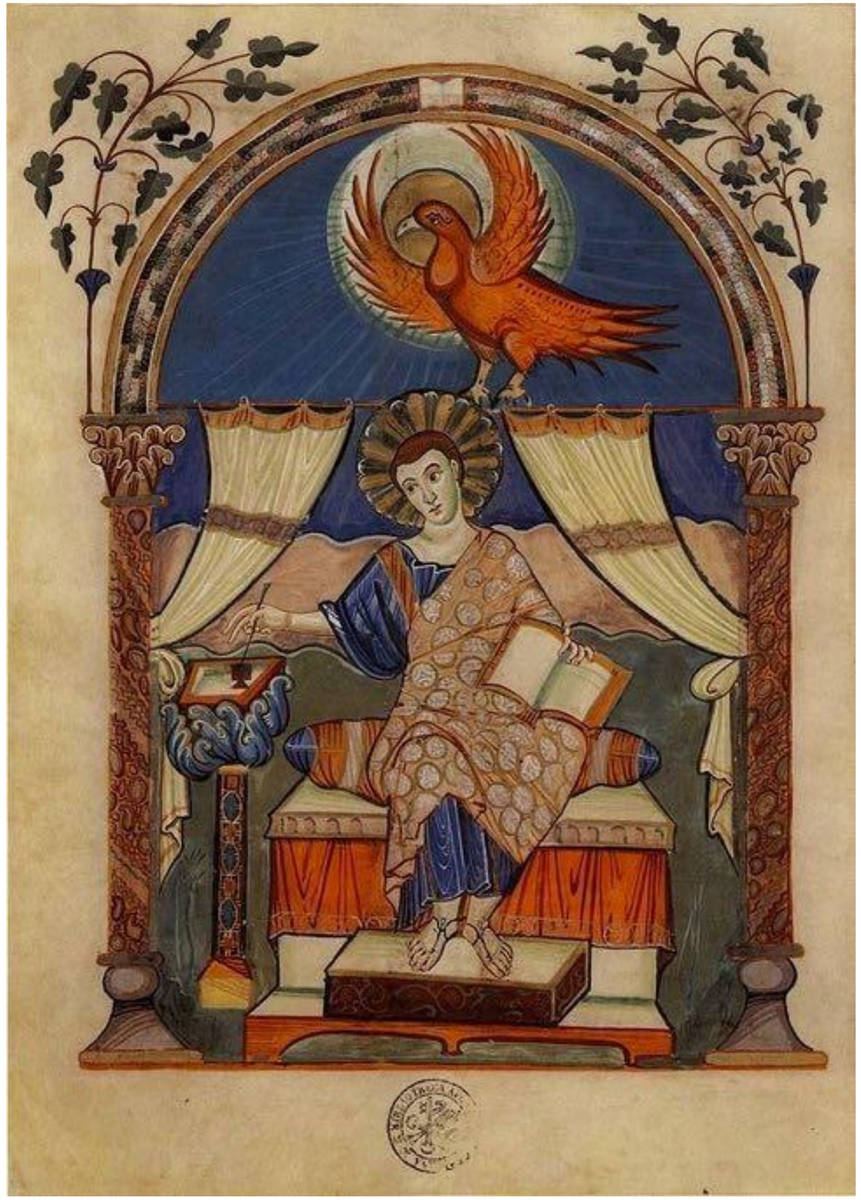
What transform places in the object in a coordinate space where the camera is at the origin and the camera is looking directly down the -z axis?

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

Perspective projection



Early painting: incorrect perspective



Carolingian painting from the 8-9th century

Perspective in art

Giotto 1290

Evolution toward correct perspective

Ambrogio Lorenzetti
Annunciation, 1344

First known perspective painting by Fillipo Brunelleshi

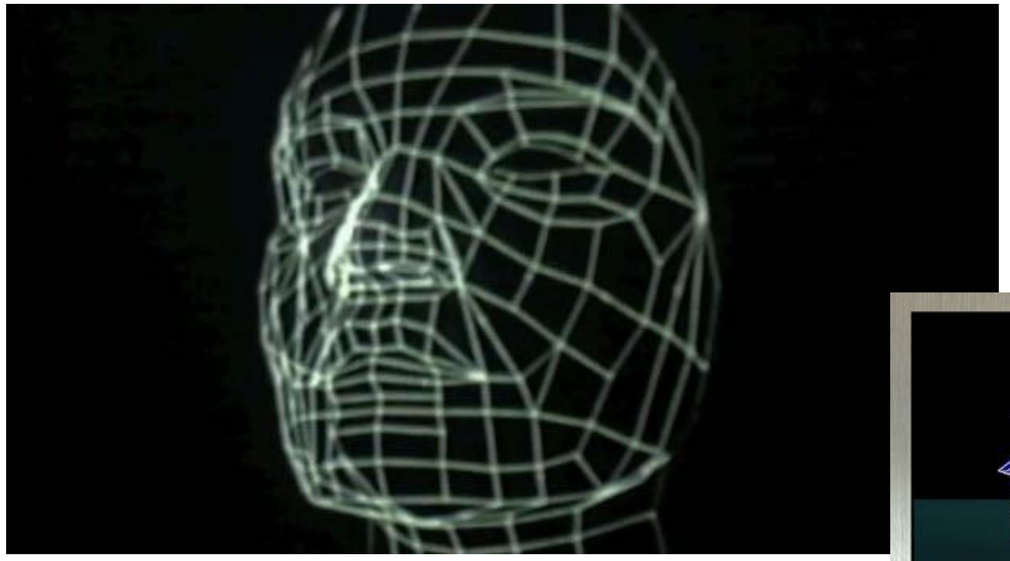
Brunelleschi, elevation of Santo Spirito, 1434-83, Florence

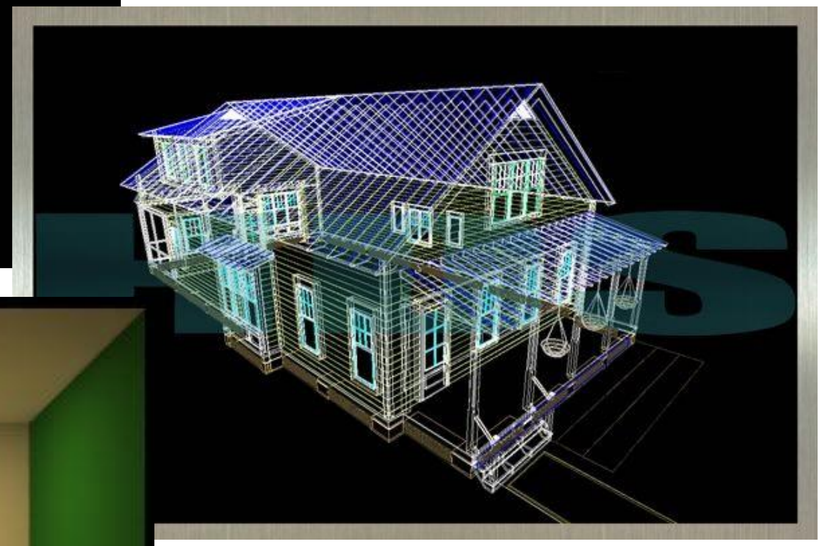
Masaccio – The Tribute Money c.1426-27
Fresco, The Brancacci Chapel, Florence

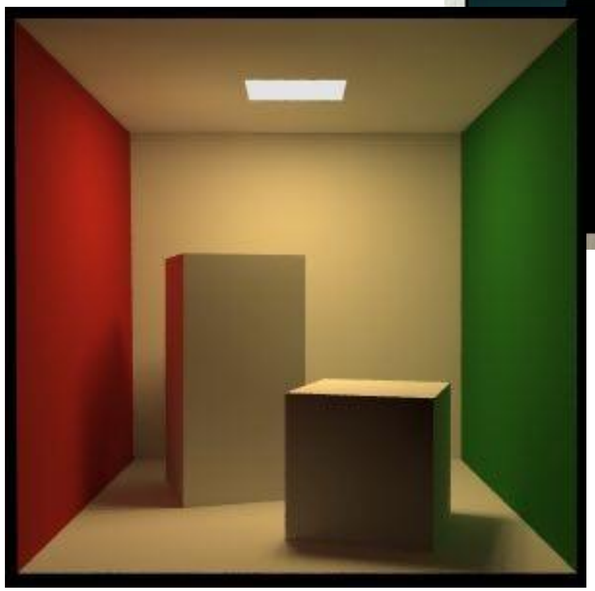
Perspective in art

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

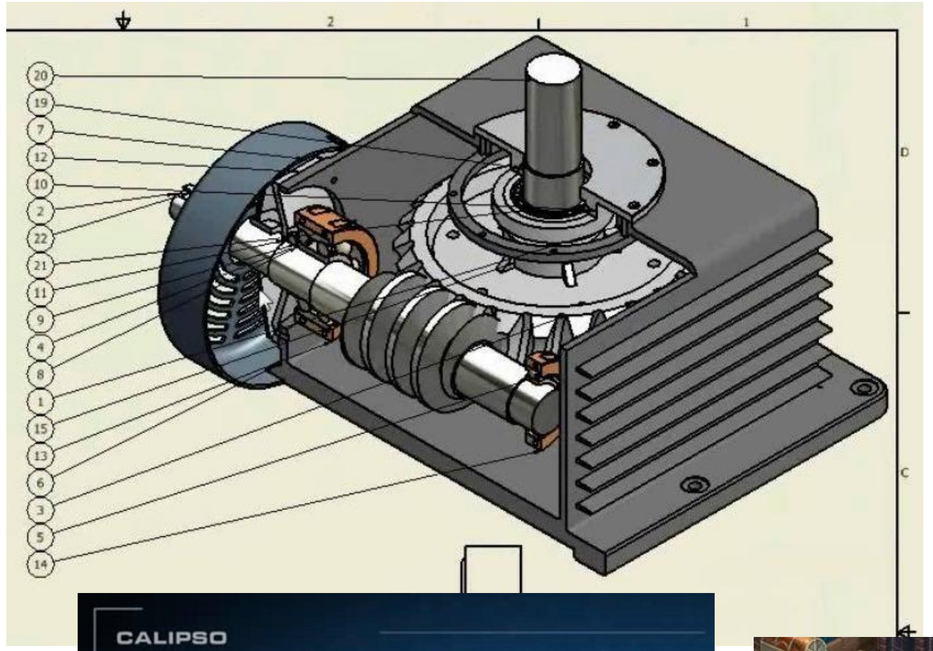
Correct perspective in computer graphics

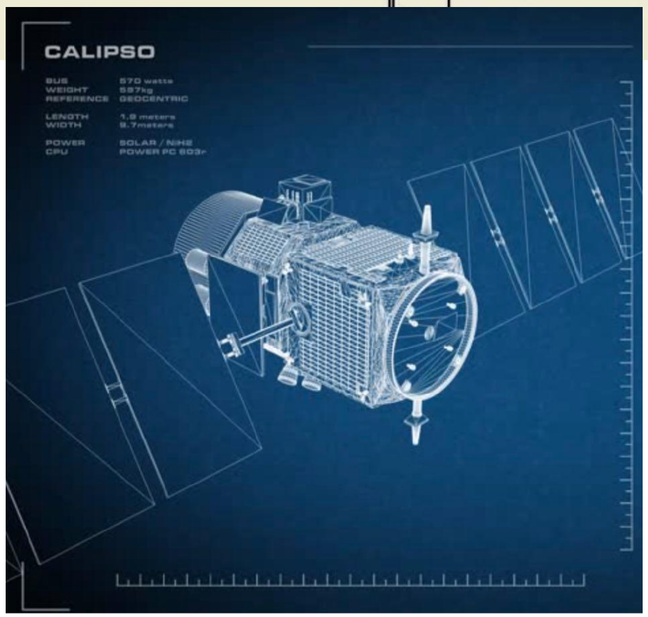




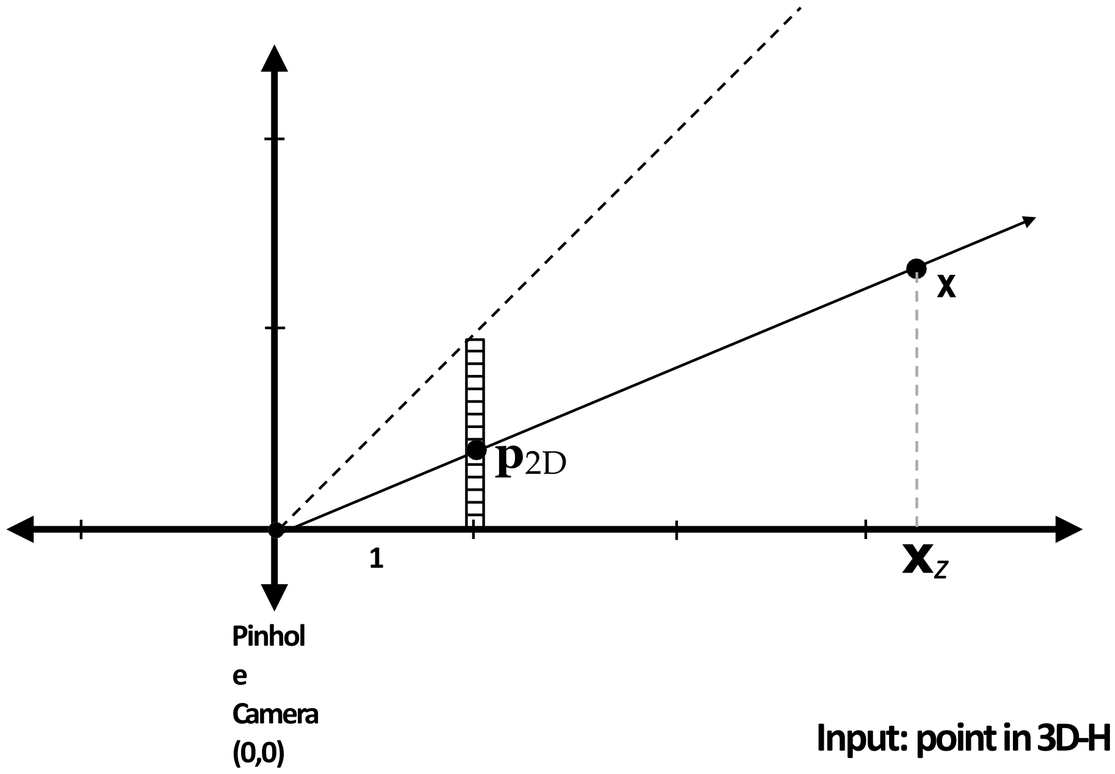


Rejection of perspective in computer graphics





Basic perspective projection



Desired perspective projected result (2D point):

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

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

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

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

After homogeneous divide: $\begin{bmatrix} \mathbf{x}_x/\mathbf{x}_z & \mathbf{x}_y/\mathbf{x}_z & 1 \end{bmatrix}^T$

(throw out third component to get 2D)

Assumption:

Pinhole camera at (0,0) looking down z

Perspective vs. orthographic projection

- **Most basic version of perspective projection matrix:**

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix} \longmapsto \begin{bmatrix} x/z \\ y/z \\ 1 \\ 1 \end{bmatrix}$$

objects shrink in distance

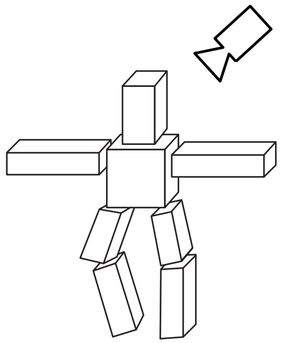
- **Most basic version of orthographic projection matrix:**

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

objects stay the same size

Transformations: from objects in 3D to their 2D screen positions

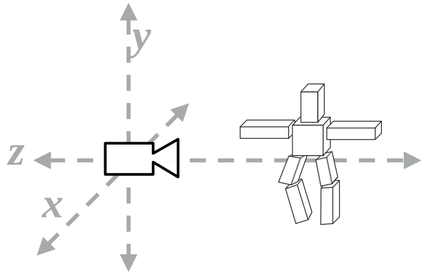
[WORLD COORDINATES]



original description of objects

view transform

[VIEW COORDINATES]

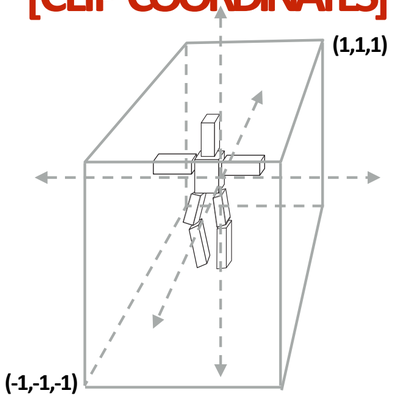


vertex positions now expressed relative to camera; camera is sitting at origin looking down -z direction

projection transform

[CLIP COORDINATES]

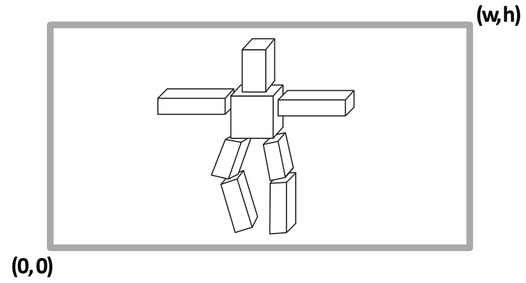
(Also called "normalized device coordinates")



everything visible to the camera is mapped to unit cube for easy triangle "clipping"

screen transform

[WINDOW COORDINATES]



objects now in 2D screen coordinates

primitives are now 2D and can be drawn via rasterization