Lecture 2

Some Mathematical and Geometrical Foundations

FUNDAMENTALS OF COMPUTER GRAPHICS

Animation & Simulation

Stanford CS248B, Fall 2022

PROFS. KAREN LIU & DOUG JAMES

Overview

- **Mathematical and geometric foundations**
  - **Let's explore some familiar and maybe new concepts together**
- **TOPICS:**
  - **Vectors and matrices**
  - **2D and 3D geometry**
    - **Projections**
    - **Barycentric coordinates (line, triangle)**
  - **Transformations**
    - **Linear vs affine**
    - **Rotations (2D & 3D) ... including derivation of rotation matrix (!)**
    - **Hierarchical modeling**
  - **Ordinary Differential Equations (ODEs)**
    - **Reduction to first order form. Autonomous vs nonautonomous. Linearization. Model equation. Stability.**
    - **Time-stepping schemes (forward Euler, backward Euler). Stiffness and stability.**

Vector and matrix operations

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \ \mathbf{A} = \begin{pmatrix} A_{11} & \dots & A_{1n} \\ \vdots & & \vdots \\ A_{n1} & \dots & A_{nn} \end{pmatrix}$$

- **Notation**
- **Vector operations**
  - **Euclidean length of vector (two norm):**

$$\|\mathbf{x}\| \equiv \|\mathbf{x}\|_2 = \sqrt{\mathbf{x} \cdot \mathbf{x}}$$

  - **Dot product:**

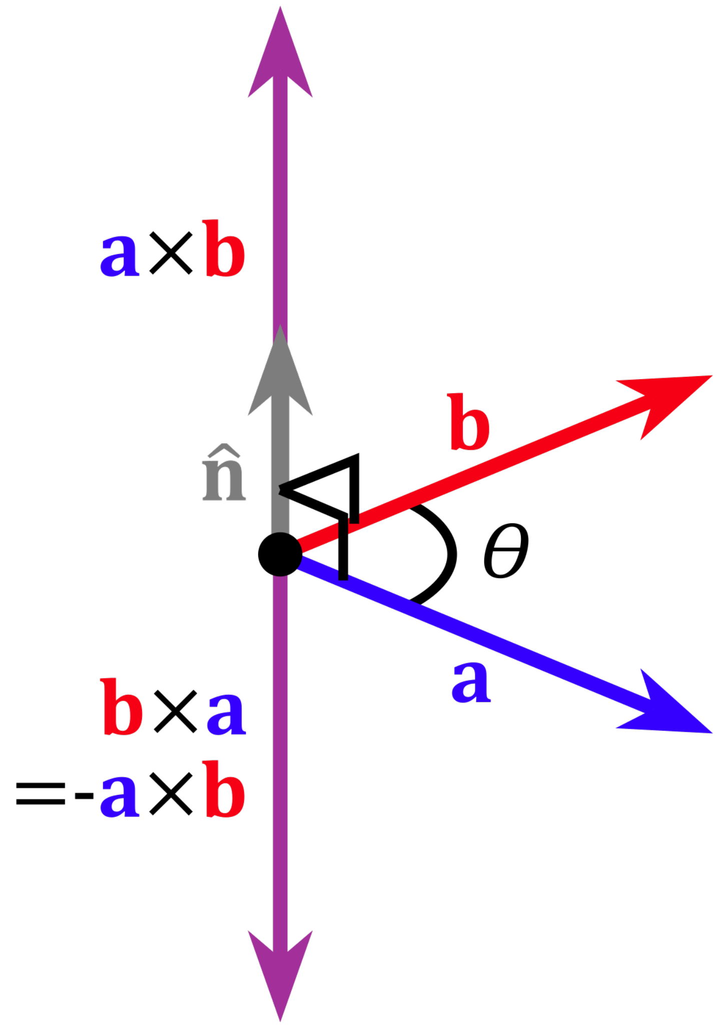
$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i y_i = \|\mathbf{x}\| \|\mathbf{y}\| \cos\theta$$

Note: Defines angle between vectors

  - **Cross product (3D)**

$$\mathbf{a} \times \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \sin\theta \, \hat{\mathbf{n}} = \begin{pmatrix} +a_2b_3 - a_3b_2 \\ -a_1b_3 + a_3b_1 \\ +a_1b_2 - a_2b_1 \end{pmatrix}$$

Note: Defines area of parallelogram



Vector perpendicular to a and b

Vector and matrix operations

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \; \mathbf{A} = \begin{pmatrix} A_{11} & \dots & A_{1n} \\ \vdots & & \vdots \\ A_{n1} & \dots & A_{nn} \end{pmatrix}$$

- **Vector operations**
  - **Normalization**

$$\hat{\mathbf{n}} = \frac{\mathbf{n}}{||\mathbf{n}||}$$

Example: Angle between two vectors is given by $\arccos(\hat{\mathbf{a}} \cdot \hat{\mathbf{b}}) \in [0,\pi]$

  - **Triple scalar product (3D)**

$$\mathbf{a} \cdot \mathbf{b} \times \mathbf{c} = \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = \det([\mathbf{a}\,\mathbf{b}\,\mathbf{c}])$$

- Defines volume of parallelepiped formed by $[\mathbf{a}\,\mathbf{b}\,\mathbf{c}]$

Vectors and matrices in OpenProcessing

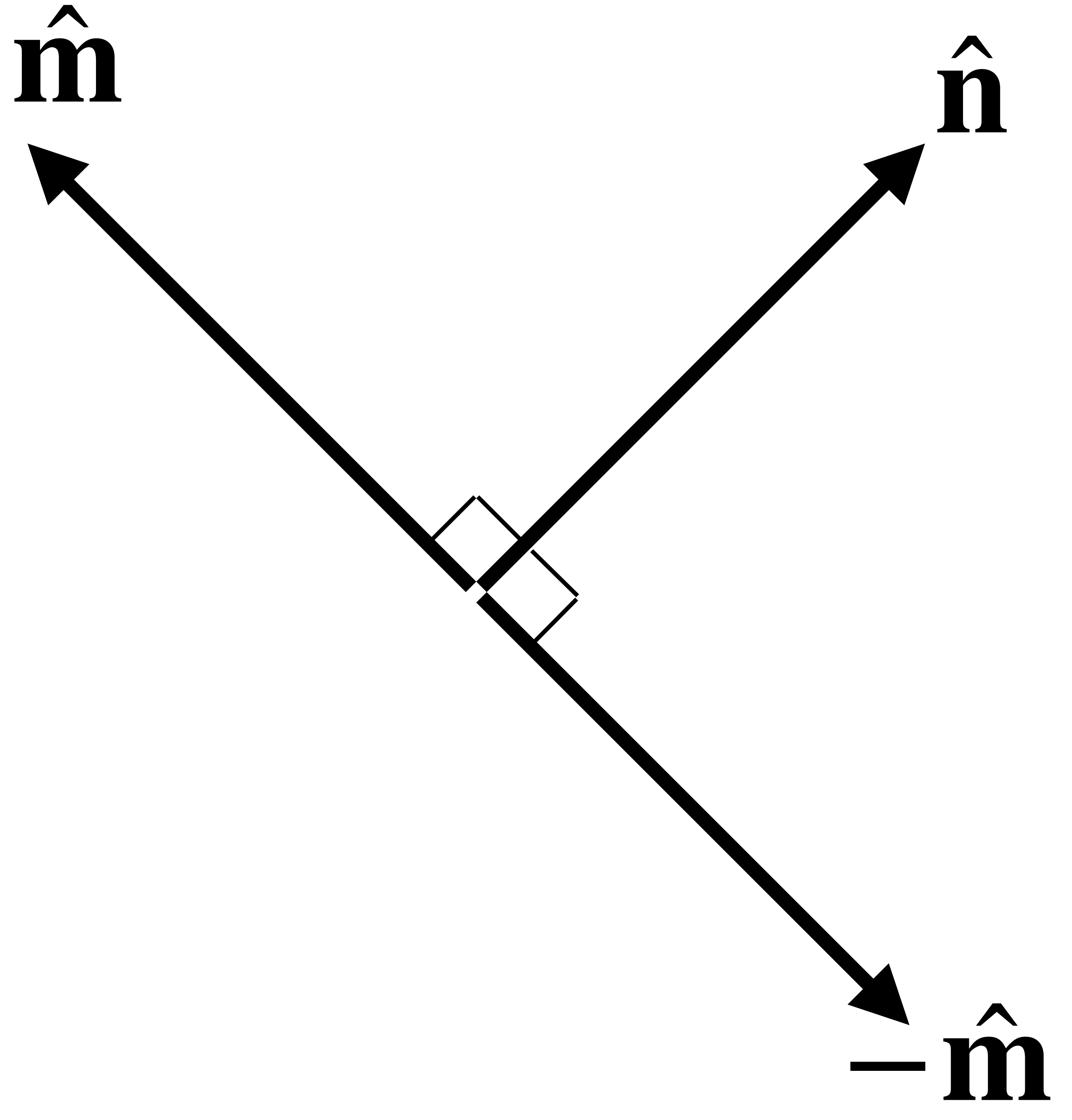
- p5.Vector class for 2D and 3D vectors
  - https://p5js.org/reference/#/p5.Vector
  - `createVector()`
    - https://p5js.org/reference/#/p5/createVector
- Other vector and matrix representations:
  - Array: Regular JavaScript arrays:

```
let v = [1,2];
let A = [[11,12],[21,22]];
```

  - math.js library (https://mathjs.org)

```
const v = math.matrix([1, 4, 9, 16, 25])
print(v) // [1, 4, 9, 16, 25]
const A = math.matrix(math.ones([2, 3]))
print(A) // [[1, 1, 1], [1, 1, 1]]
print(A.size()) // [2, 3]
```

Making an orthogonal basis (2D)



- **Application: Normal & tangent space directions for contact**
- **Given an initial unit vector,** $\hat{\mathbf{n}} = \begin{pmatrix} n_x \\ n_y \end{pmatrix}$
- **A perpendicular unit vector** $\hat{\mathbf{m}}$ **must satisfy** $\hat{\mathbf{n}} \cdot \hat{\mathbf{m}} = 0$
- **Suitable choice is** $\hat{\mathbf{m}} = \begin{pmatrix} -n_y \\ n_x \end{pmatrix}$
- **Handedness?**
  - **Could have also chosen** $-\hat{\mathbf{m}} = \begin{pmatrix} n_y \\ -n_x \end{pmatrix}$
  - **Choose right-handed basis, such that** $\det([\hat{\mathbf{n}}\,\hat{\mathbf{m}}]) = \det \begin{bmatrix} n_x & m_x \\ n_y & m_y \end{bmatrix} = n_x m_y - n_y m_x = +1$

Projection of a vector $\mathbf{v}$ onto a direction, $\hat{\mathbf{n}}$

- The scalar component of $\mathbf{v}$ along $\hat{\mathbf{n}}$ is $v_n = \mathbf{v} \cdot \hat{\mathbf{n}}$
- The projected vector is

$$v_n \, \hat{\mathbf{n}} = (\mathbf{v} \cdot \hat{\mathbf{n}}) \, \hat{\mathbf{n}} = \hat{\mathbf{n}} \, (\hat{\mathbf{n}}^T \mathbf{v}) = \hat{\mathbf{n}} \, \hat{\mathbf{n}}^T \mathbf{v}$$

- The projection matrix is

$$\mathbf{P}_{\hat{\mathbf{n}}} = \hat{\mathbf{n}} \, \hat{\mathbf{n}}^T = \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} \begin{pmatrix} n_1 & n_2 & n_3 \end{pmatrix} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}$$

  - A rank-one matrix

- Vector with component along $\hat{\mathbf{n}}$ removed is $\mathbf{v} - \mathbf{P}_{\hat{\mathbf{n}}} \mathbf{v} = \left(\mathbf{I} - \mathbf{P}_{\hat{\mathbf{n}}} \right) \mathbf{v}$
- Decomposition into parallel and perpendicular components:

$$\mathbf{v} = \mathbf{P}_{\hat{\mathbf{n}}} \mathbf{v} + \left(\mathbf{I} - \mathbf{P}_{\hat{\mathbf{n}}} \right) \mathbf{v}$$

Projection onto a line

- Line equation: Consider a line given by the parametric equation,

$$\mathbf{r}(t) = \mathbf{q} + t\,\hat{\mathbf{n}}, \quad t \in \mathbb{R}$$

where $\mathbf{q}$ is a point on the line, its direction is $\hat{\mathbf{n}}$, and t is the distance along the line.

- Given a point $\mathbf{x}$, it's projection onto the line, $\bar{\mathbf{x}}$, must be

$$\bar{\mathbf{x}} = \mathbf{q} + \mathbf{P}_{\hat{\mathbf{n}}}(\mathbf{x} - \mathbf{q}) = \mathbf{q} + \hat{\mathbf{n}}\,\hat{\mathbf{n}}^T(\mathbf{x} - \mathbf{q})$$

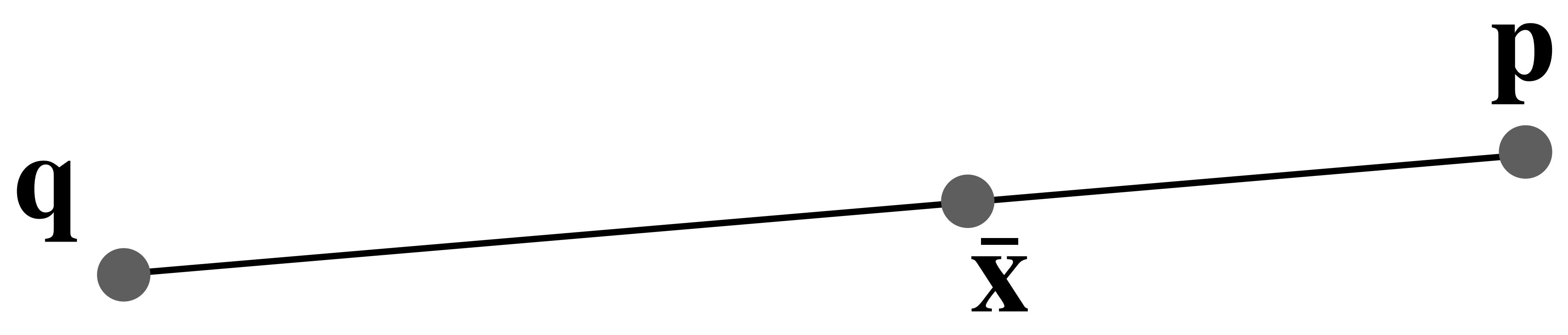
- t is the signed distance along the line:

$$t = \hat{\mathbf{n}}^T(\mathbf{r}(t) - \mathbf{q}) = \hat{\mathbf{n}}^T(\bar{\mathbf{x}} - \mathbf{q}) = \hat{\mathbf{n}}^T\,\mathbf{P}_{\hat{\mathbf{n}}}(\mathbf{x} - \mathbf{q}) = \hat{\mathbf{n}}^T(\mathbf{x} - \mathbf{q})$$

Barycentric coordinates of a point on a line segment

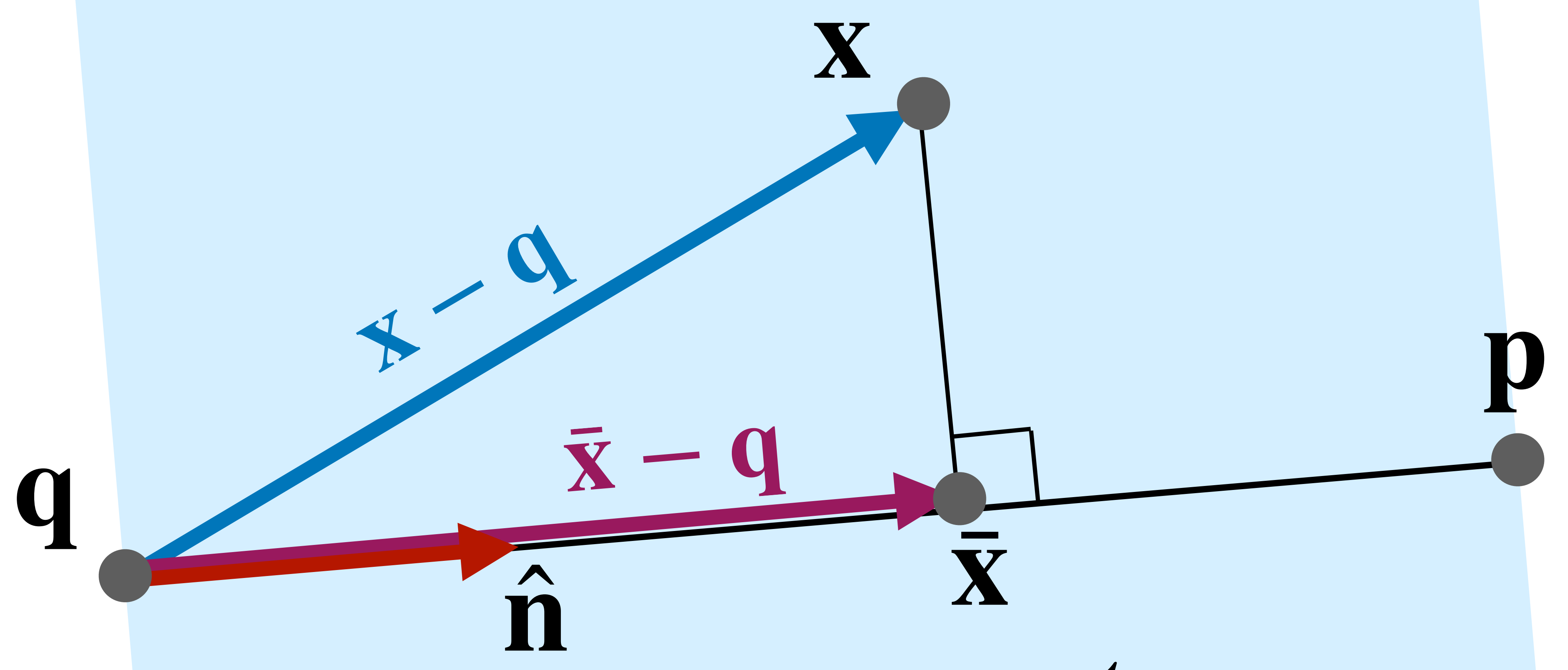
- Representation guarantees point is on line segment
- Barycentric coordinate, $\alpha \in [0,1]$
- Point on line segment given by

$$\bar{\mathbf{x}} = (1-\alpha)\mathbf{q} + \alpha\mathbf{p}$$



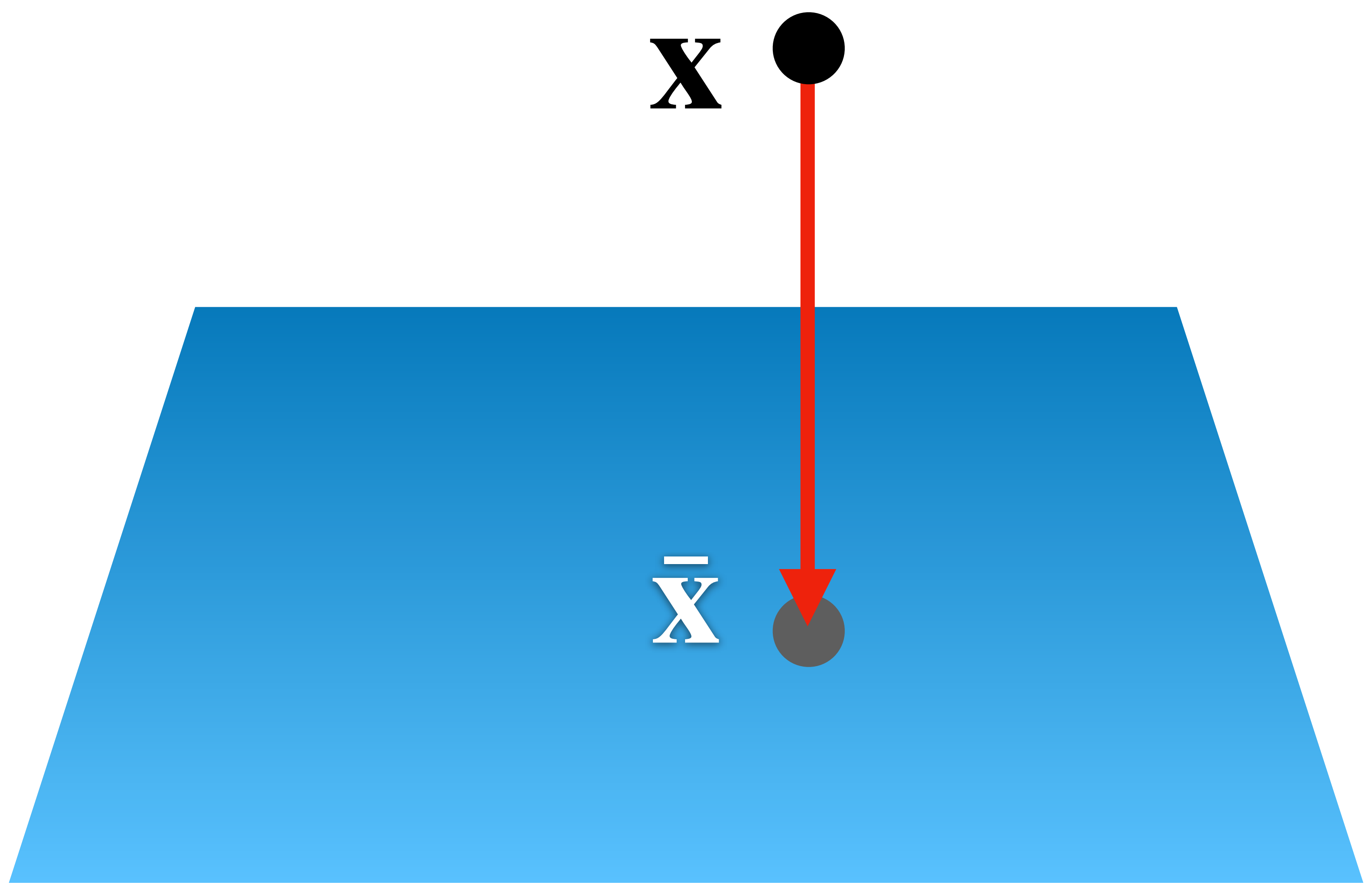
Closest-point projection onto a line segment, $\mathbf{pq}$

- Define $\hat{\mathbf{n}} = \dfrac{\mathbf{p}-\mathbf{q}}{\|\mathbf{p}-\mathbf{q}\|}$
- Given $\mathbf{x}$, obtain its projection onto the infinite line, $\bar{\mathbf{x}}$, and parameter, t



- Define the normalized coordinate, $\bar{\alpha} = \dfrac{t}{\|\mathbf{p}-\mathbf{q}\|} \in \mathbb{R}$
- The closest point is then given at barycentric coordinate $\alpha = \text{clamp}(\bar{\alpha}, 0, 1)$
- DEMO: https://openprocessing.org/sketch/1673434

Projection: Closest point on a plane



- Given the implicit equation of a plane

$$C(\mathbf{x}) = \hat{\mathbf{n}} \cdot \mathbf{x} + d = 0$$

where $\hat{\mathbf{n}} = \nabla_{\mathbf{x}} C(\mathbf{x})$ is a *unit* normal vector to the plane.

- The closest point on the plane is located at

$$\bar{\mathbf{x}} = \mathbf{x} + \lambda \hat{\mathbf{n}}$$

where $\lambda \in \mathbb{R}$ is the unknown projection distance.

- Substituting into plane equation

$$0 = C(\bar{\mathbf{x}}) = \hat{\mathbf{n}} \cdot (\mathbf{x} + \lambda \hat{\mathbf{n}}) + d = \hat{\mathbf{n}} \cdot \mathbf{x} + \lambda + d = C(\mathbf{x}) + \lambda$$

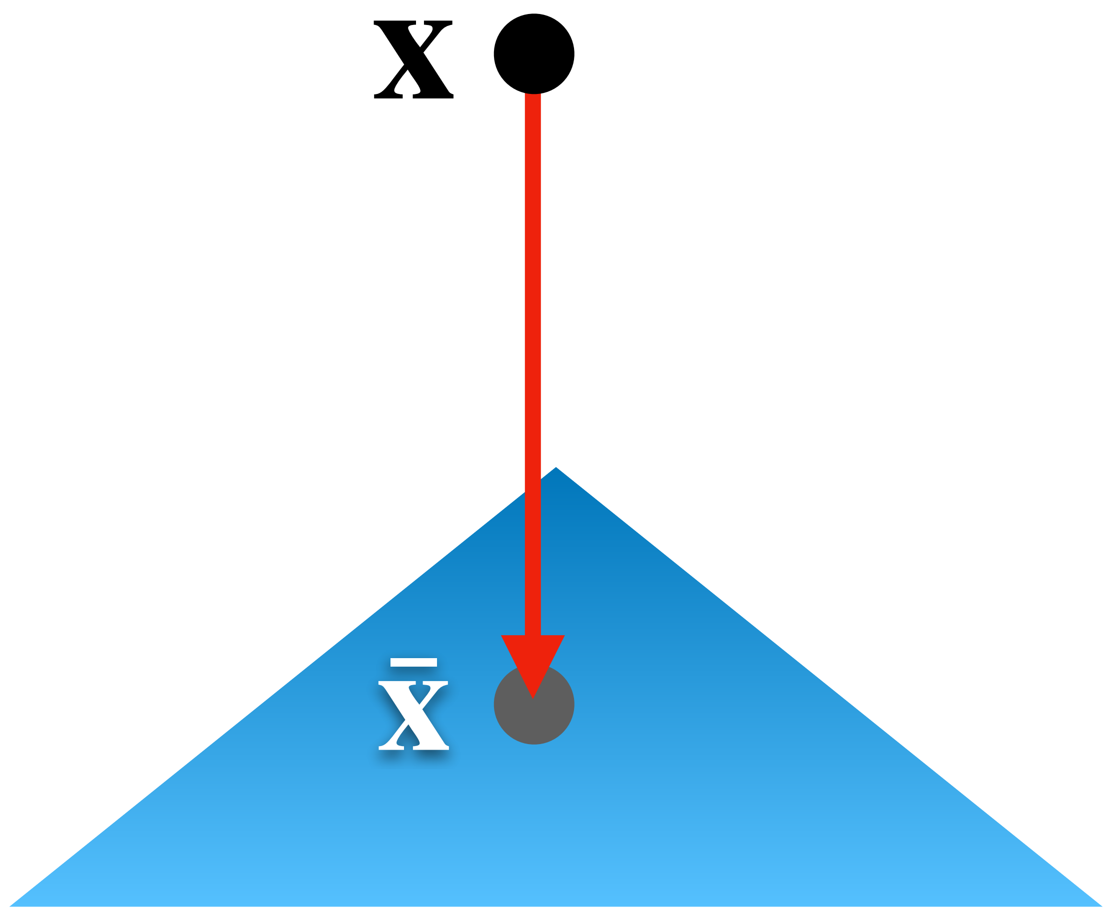
so that $\lambda = -\,C(\mathbf{x})$ and therefore the closest point on the plane is

$$\bar{\mathbf{x}} = \mathbf{x} - C(\mathbf{x})\,\hat{\mathbf{n}}$$

Barycentric coordinates of a point on a triangle

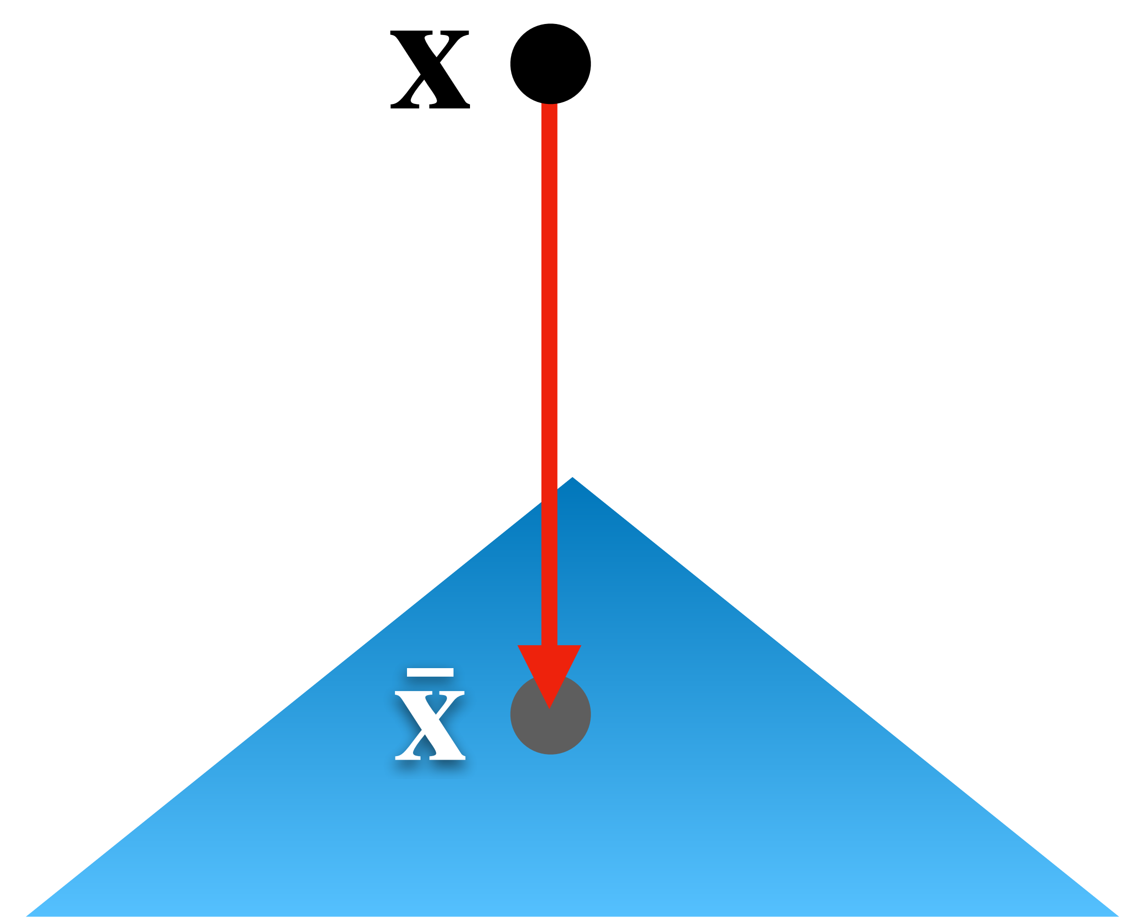
- Representation guarantees point is on triangle
- Given triangle with vertices $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$
- Construct edge vectors $\mathbf{u} = \mathbf{p}_1 - \mathbf{p}_0$ and $\mathbf{v} = \mathbf{p}_2 - \mathbf{p}_0$
- Point on plane can be expressed as
  - $\alpha_0\mathbf{p}_0 + \alpha_1\mathbf{p}_1 + \alpha_2\mathbf{p}_2$ where $\alpha_0 + \alpha_1 + \alpha_2 = 1.$
- Or
  - $\mathbf{p}_0 + \alpha_1\mathbf{u} + \alpha_2\mathbf{v}$
- Point is inside the triangle when all $\alpha_i \in [0,1].$
- How to get α_i?

Closest-point projection onto a triangle

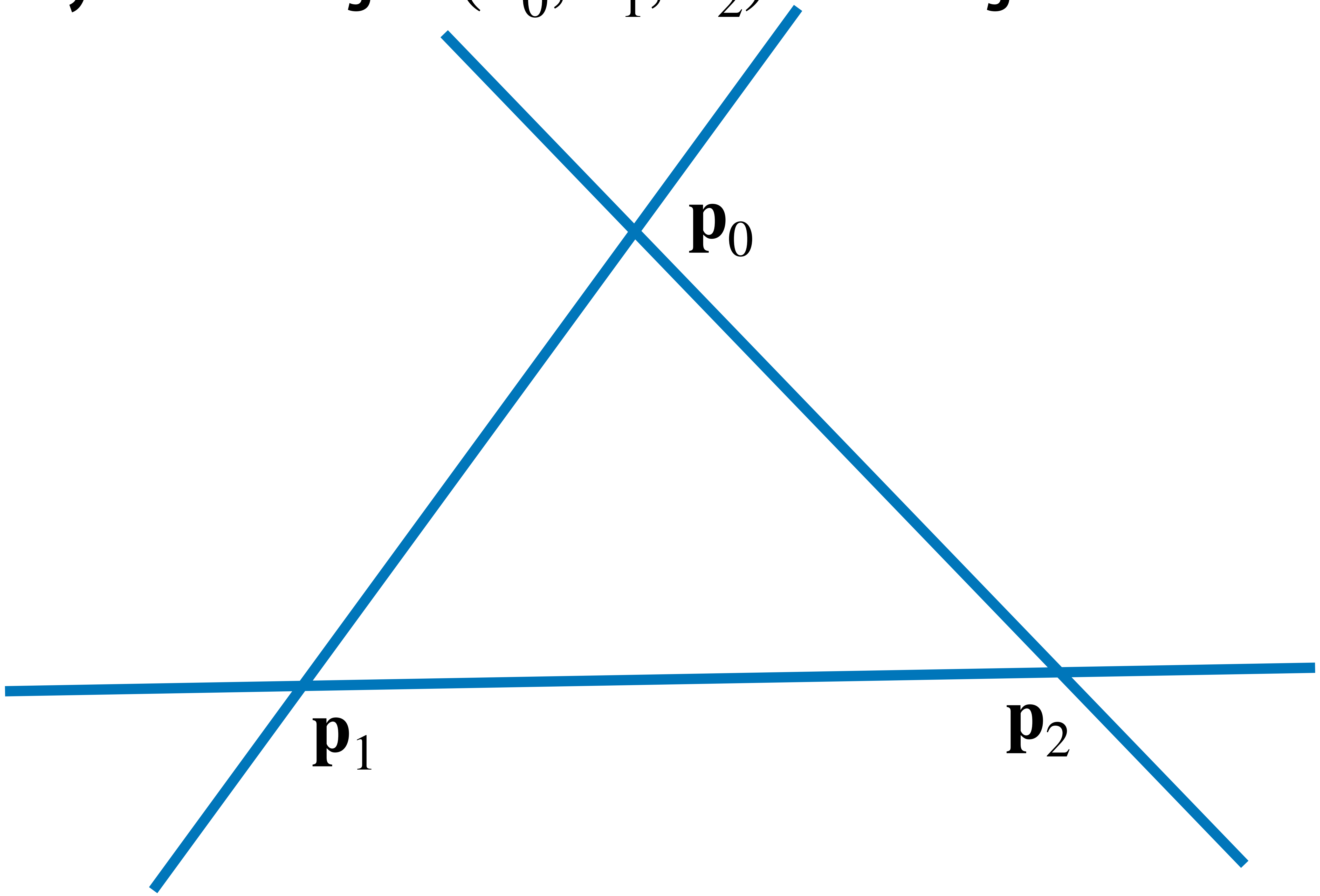


- Related to homework question
- Can estimate barycentric weights using least squares solution of
  - $\mathbf{x} \approx \bar{\mathbf{x}}(\alpha) = \mathbf{p}_0 + \alpha_1 \mathbf{u} + \alpha_2 \mathbf{v} = \mathbf{p}_0 + [\mathbf{u}\ \mathbf{v}] \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} = \mathbf{p}_0 + \mathbf{J}\,\alpha$
  - Writing $\mathbf{J}\,\alpha = \mathbf{x} - \mathbf{p}_0$ (in a least-squares sense)
  - Solving least-squares project using the Normal equations (i.e., project using $\mathbf{J}^T$):
  - $\mathbf{J}^T \mathbf{J}\,\alpha = \mathbf{J}^T \left(\mathbf{x} - \mathbf{p}_0\right)$ implies that $\alpha = \left(\mathbf{J}^T \mathbf{J}\right)^{-1} \mathbf{J}^T \left(\mathbf{x} - \mathbf{p}_0\right)$.
  - The least-squares solution is $\bar{\mathbf{x}} = \mathbf{p}_0 + \mathbf{J} \left(\mathbf{J}^T \mathbf{J}\right)^{-1} \mathbf{J}^T \left(\mathbf{x} - \mathbf{p}_0\right)$
  - Compare to projection-onto-line equation (with $\mathbf{p}_0 = \mathbf{q}$ and $\mathbf{J} = \hat{\mathbf{n}}$)

Closest-point projection onto a triangle

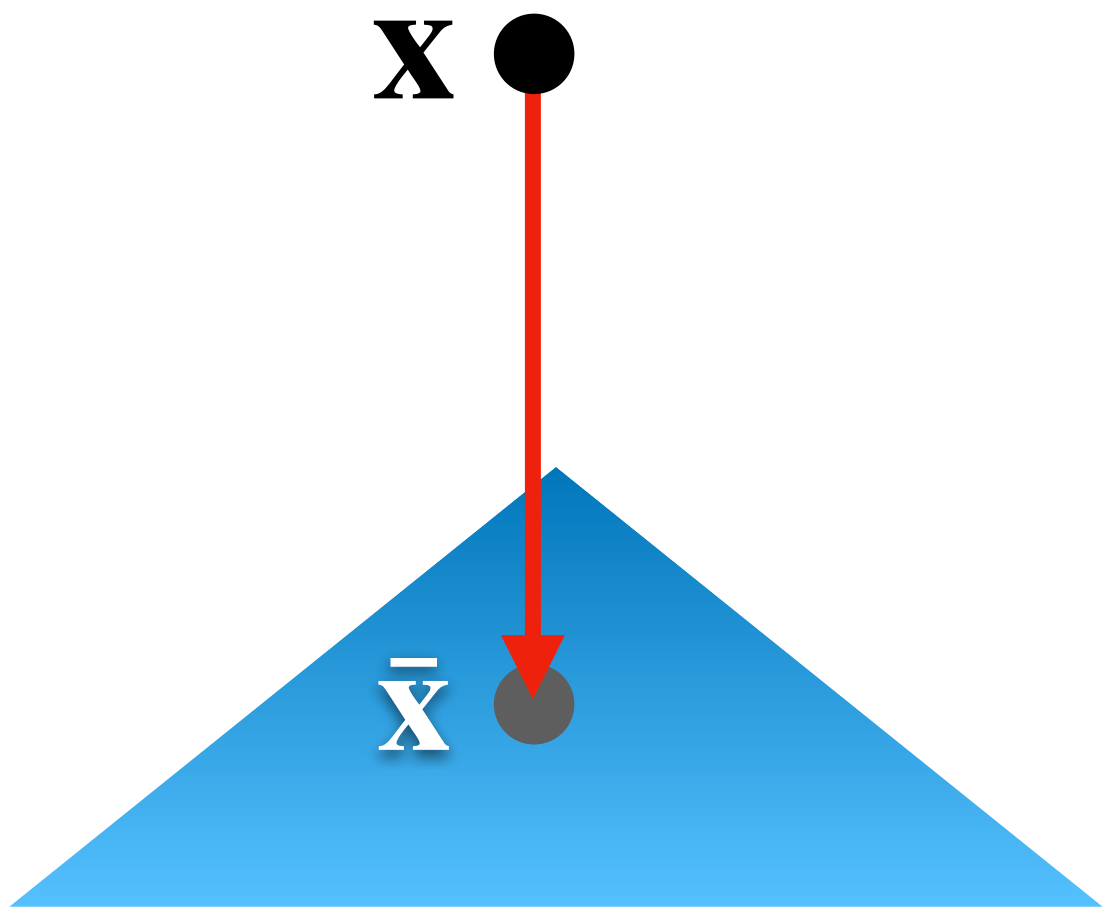


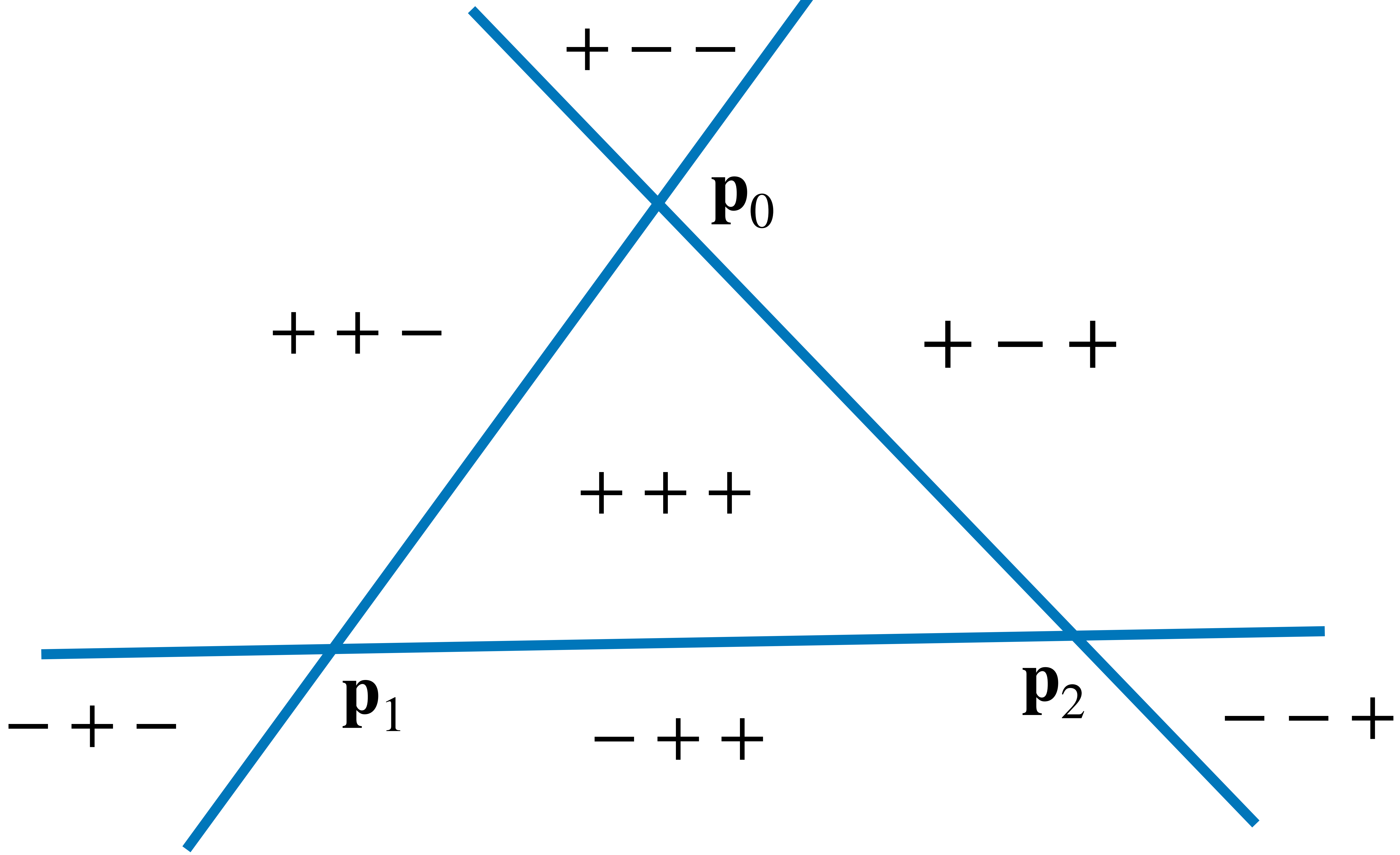
- Related to homework question
- Closest features could be vertex, point on edge, or point on face.
  - Can use barycentric weights $(\alpha_0, \alpha_1, \alpha_2)$ to distinguish locations
  - How?



Closest-point projection onto a triangle

- Related to homework question
- Closest features could be vertex, point on edge, or point on face.
  - Can use barycentric weights $(\alpha_0, \alpha_1, \alpha_2)$ to distinguish locations
  - How?

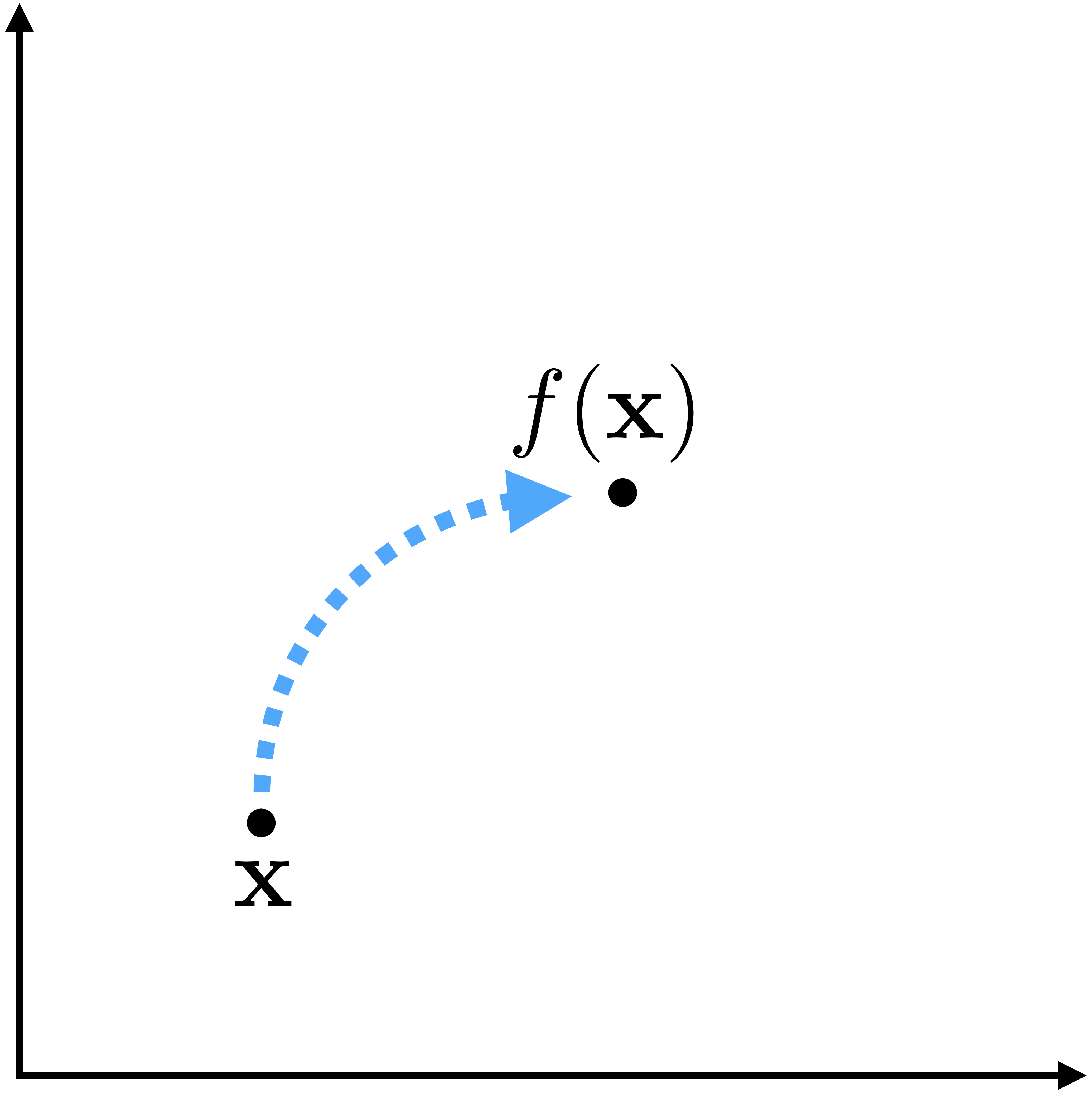




Transformations

(more in CS248A)

Basic idea: f transforms point $\mathbf{x}$ 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 transformations**

Linear transformation

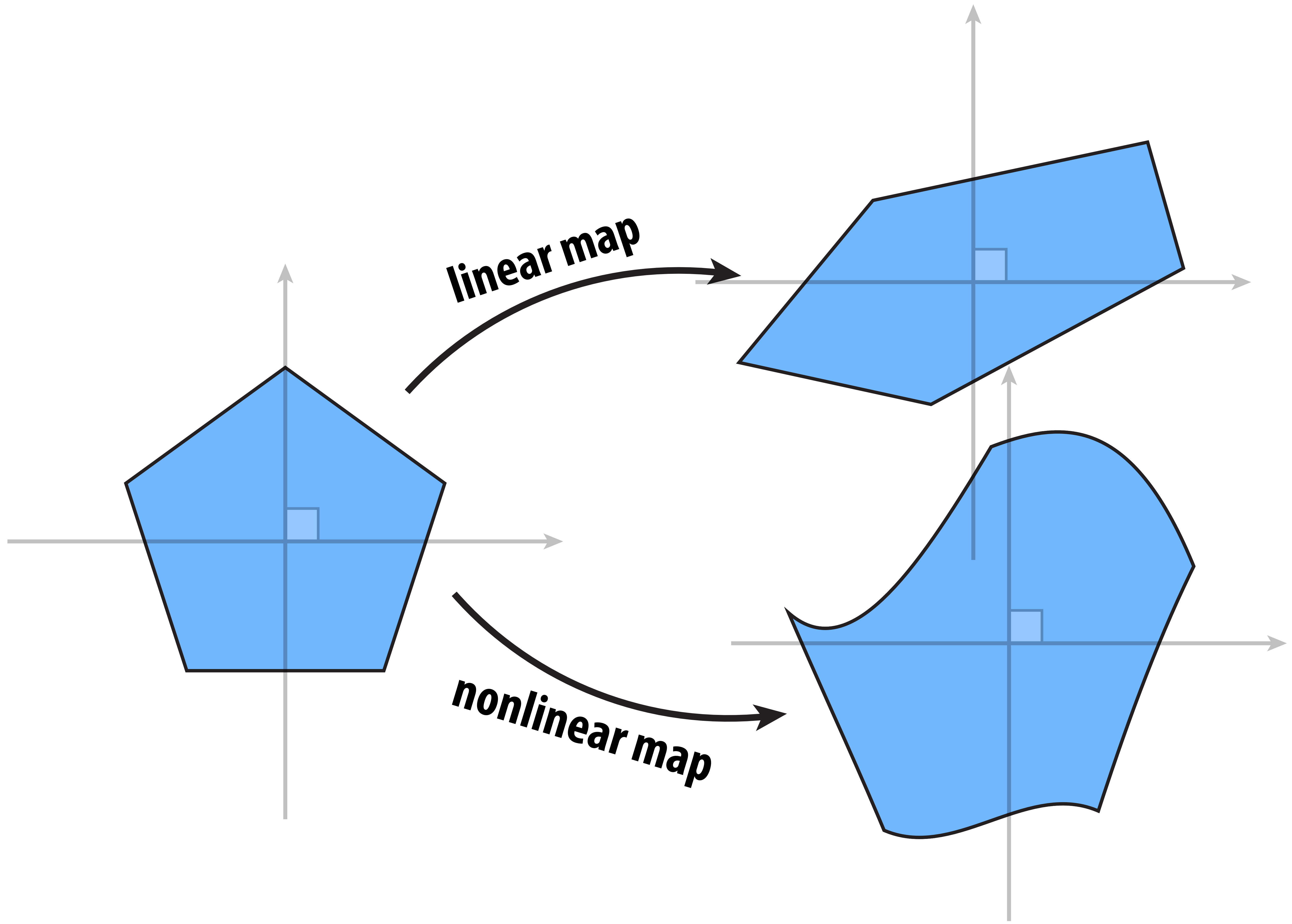
$$f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f(\mathbf{v})$$
$$f(a\mathbf{u}) = af(\mathbf{u})$$

- **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):**

$$\begin{tikzcd}
\mathbf{x}, \mathbf{y} \arrow[r, "\text{add first}"] \arrow[d, "\text{apply f first}"] & \mathbf{x}+\mathbf{y} \arrow[d, "\text{then apply f}"] \\
f(\mathbf{x}), f(\mathbf{y}) \arrow[r, "\text{then add}"] & \begin{array}{c} f(\mathbf{x})+f(\mathbf{y}) \\ = \\ f(\mathbf{x}+\mathbf{y}) \end{array}
\end{tikzcd}$$

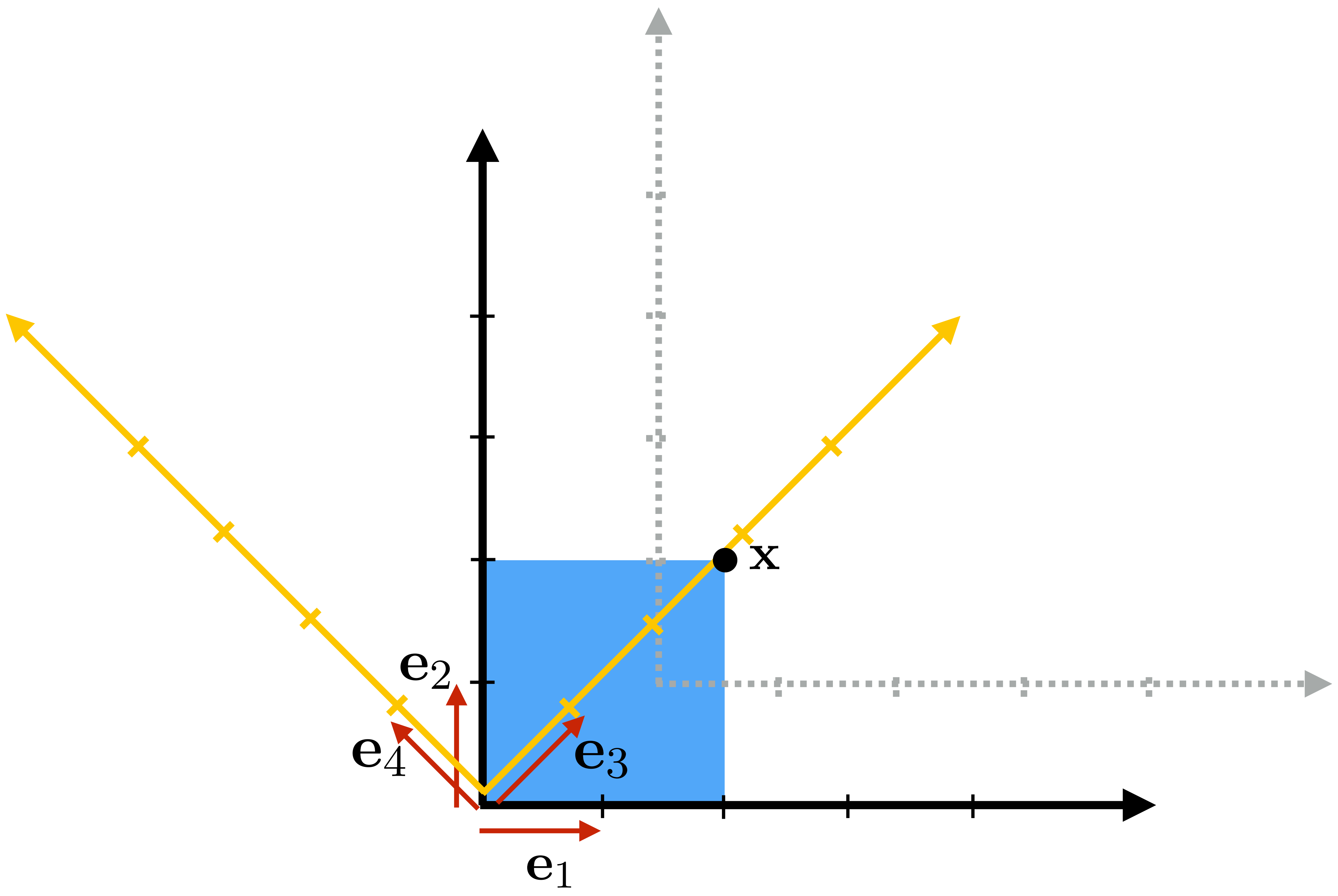
Linear transforms/maps—visualized

- **Example:**



Key idea: ***linear maps take lines to lines***

Review: representing points in a coordinate space



Consider coordinate space defined by orthogonal vectors $\mathbf{e}_1$ and $\mathbf{e}_2$

$$\mathbf{x} = 2\mathbf{e}_1 + 2\mathbf{e}_2$$

$$\mathbf{x} = \begin{bmatrix} 2 & 2 \end{bmatrix}$$

$\mathbf{x} = \begin{bmatrix} 0.5 & 1 \end{bmatrix}$ **in coordinate space defined by e_1 and e_2, with origin at (1.5, 1)**

$\mathbf{x} = \begin{bmatrix} \sqrt{8} & 0 \end{bmatrix}$ **in coordinate space defined by e_3 and e_4, with origin at (0, 0)**

Review: 2D matrix multiplication

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

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

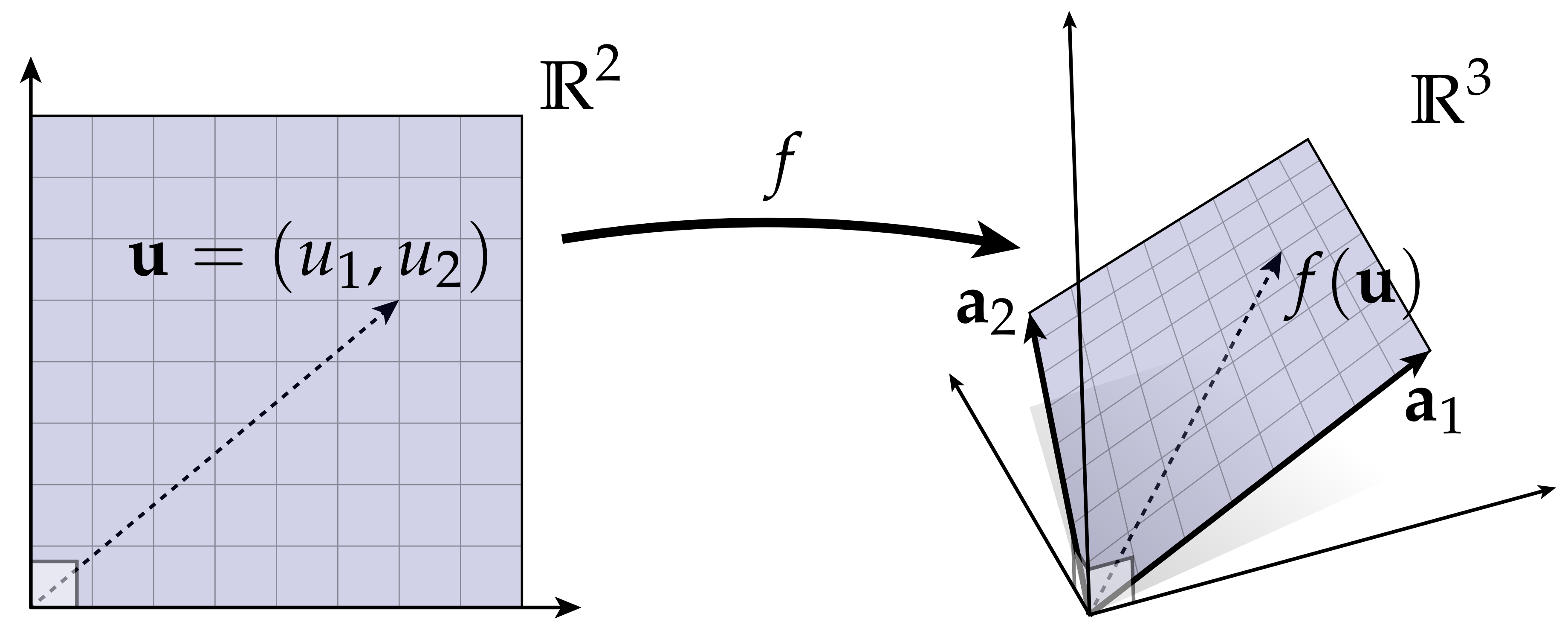
$$\begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

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

Linear maps via matrices

- **Example: Consider a linear map**

$$f(\mathbf{u}) = u_1\mathbf{a}_1 + u_2\mathbf{a}_2$$



- **Encoding as a matrix: "a" vectors become matrix columns:**

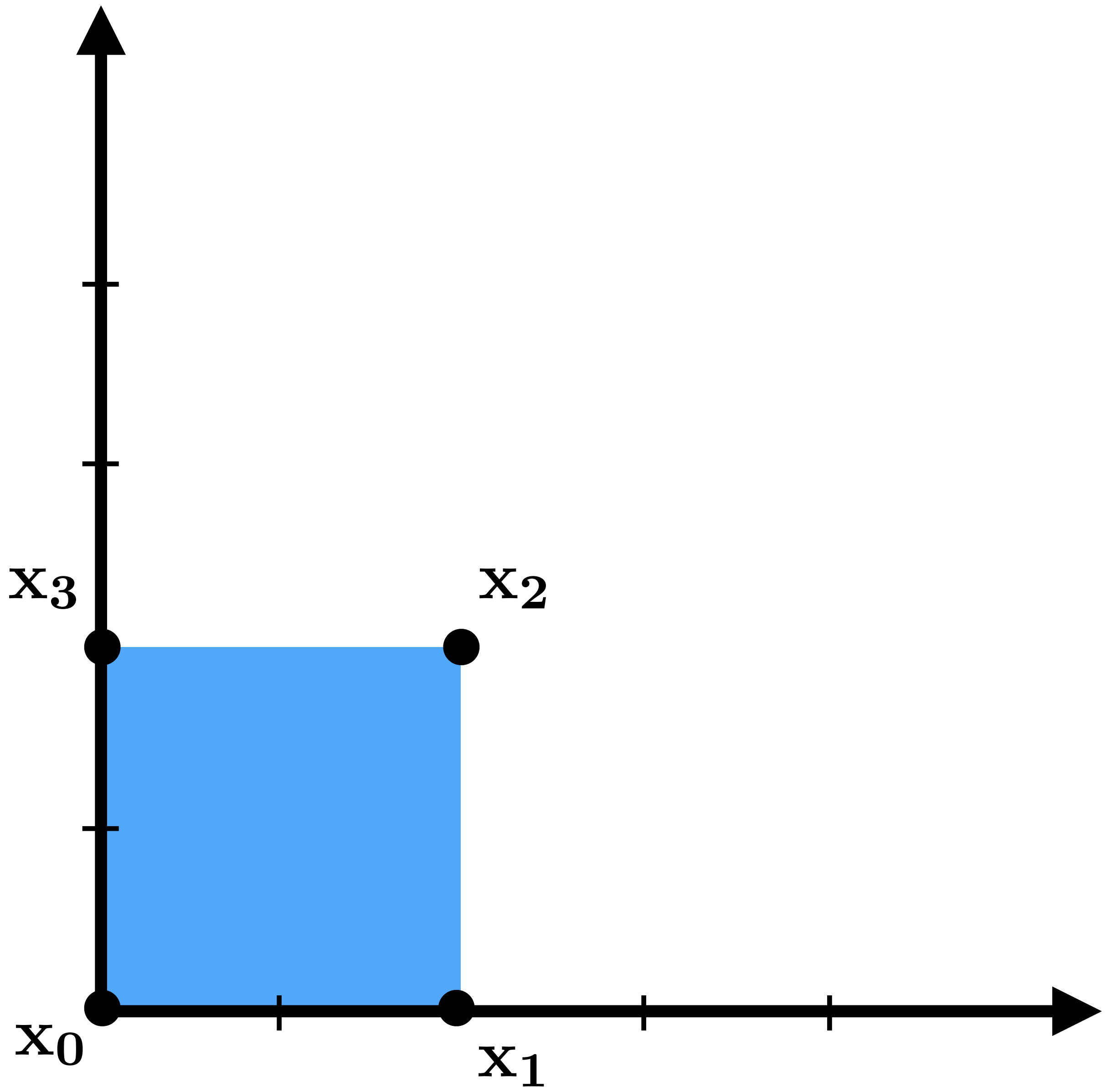
$$A := \begin{bmatrix} a_{1,x} & a_{2,x} \\ a_{1,y} & a_{2,y} \\ a_{1,z} & a_{2,z} \end{bmatrix}$$

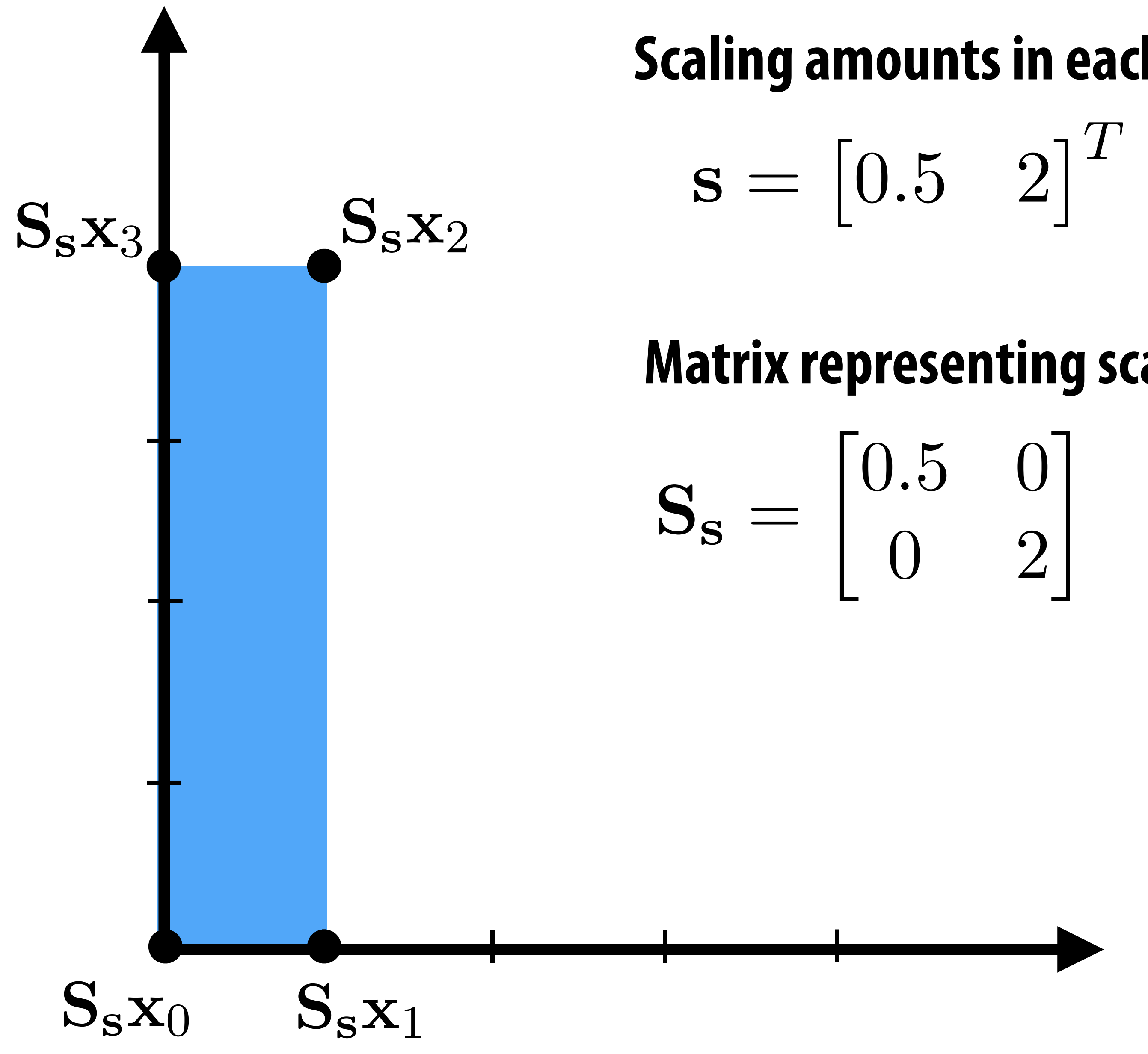
- **Matrix-vector multiply computes same output as original map:**

$$\begin{bmatrix} a_{1,x} & a_{2,x} \\ a_{1,y} & a_{2,y} \\ a_{1,z} & a_{2,z} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} a_{1,x}u_1 + a_{2,x}u_2 \\ a_{1,y}u_1 + a_{2,y}u_2 \\ a_{1,z}u_1 + a_{2,x}u_2 \end{bmatrix} = u_1\mathbf{a}_1 + u_2\mathbf{a}_2$$

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

Consider non-uniform scale: $\mathbf{S_s} = \begin{bmatrix} \mathbf{s}_x & 0 \\ 0 & \mathbf{s}_y \end{bmatrix}$





Scaling amounts in each direction:

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

Matrix representing scale transform:

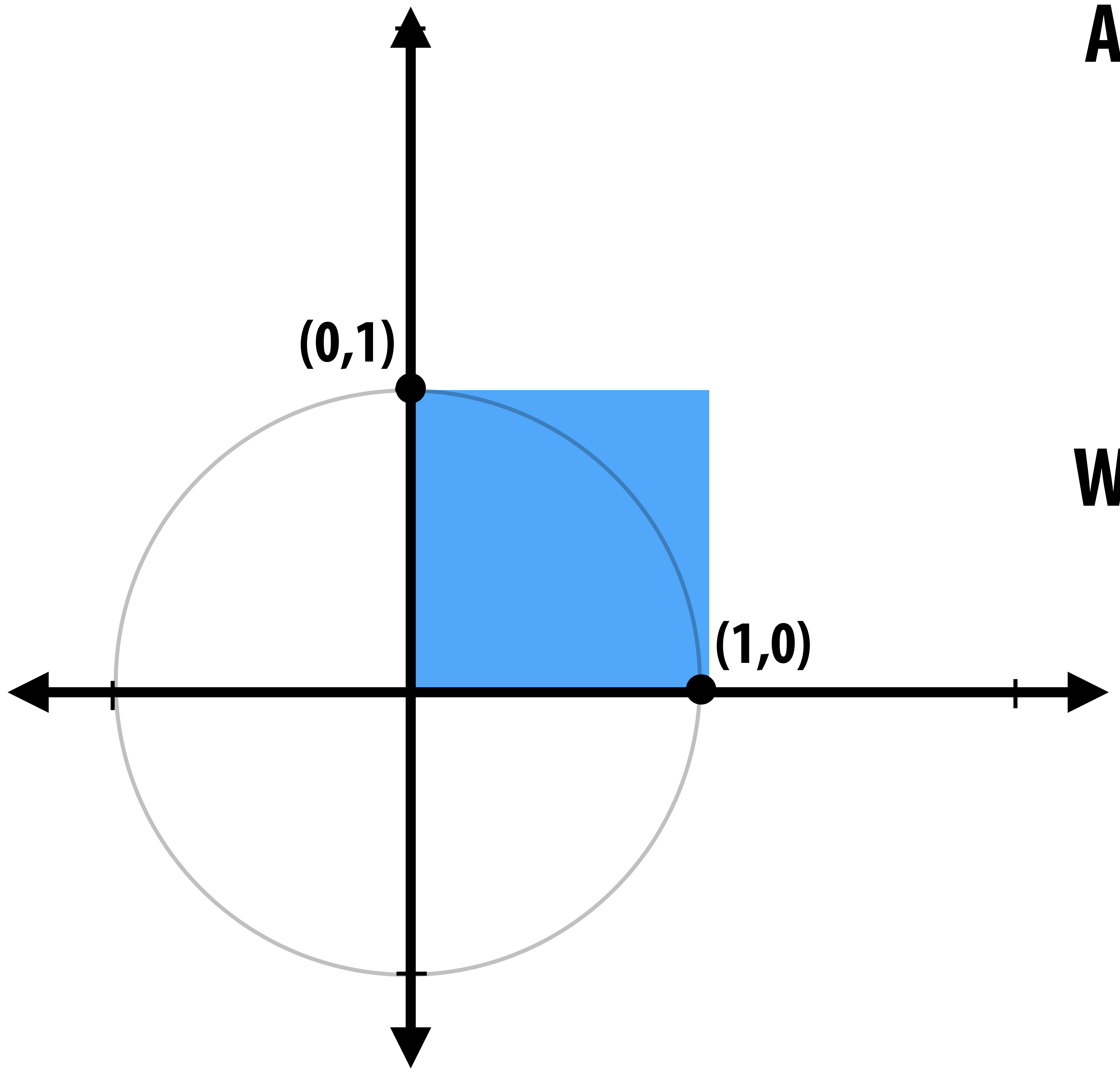
$$\mathbf{S_s} = \begin{bmatrix} 0.5 & 0 \\ 0 & 2 \end{bmatrix}$$

Rotation matrix (2D)

Question: what happens to (1, 0) and (0,1) after rotation by θ?

Reminder: rotation moves points along circular trajectories.

(Recall that $\cos\theta$ and $\sin\theta$ are the coordinates of a point on the unit circle.)



Answer:

$$R_\theta(1,0) = (\cos(\theta), \sin(\theta))$$

$$R_\theta(0,1) = (\cos(\theta + \pi/2), \sin(\theta + \pi/2))$$

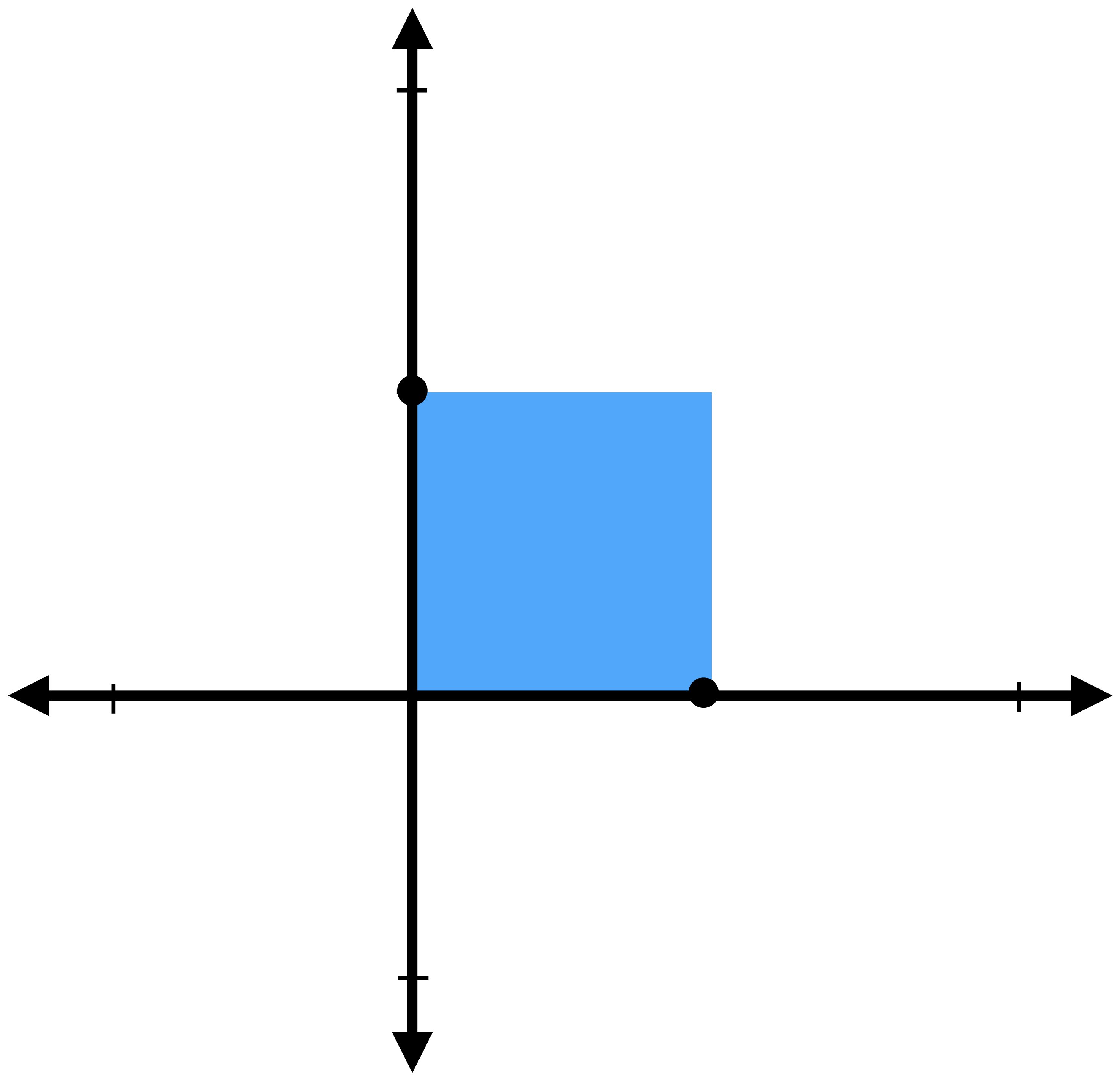
Which means the matrix must look like:

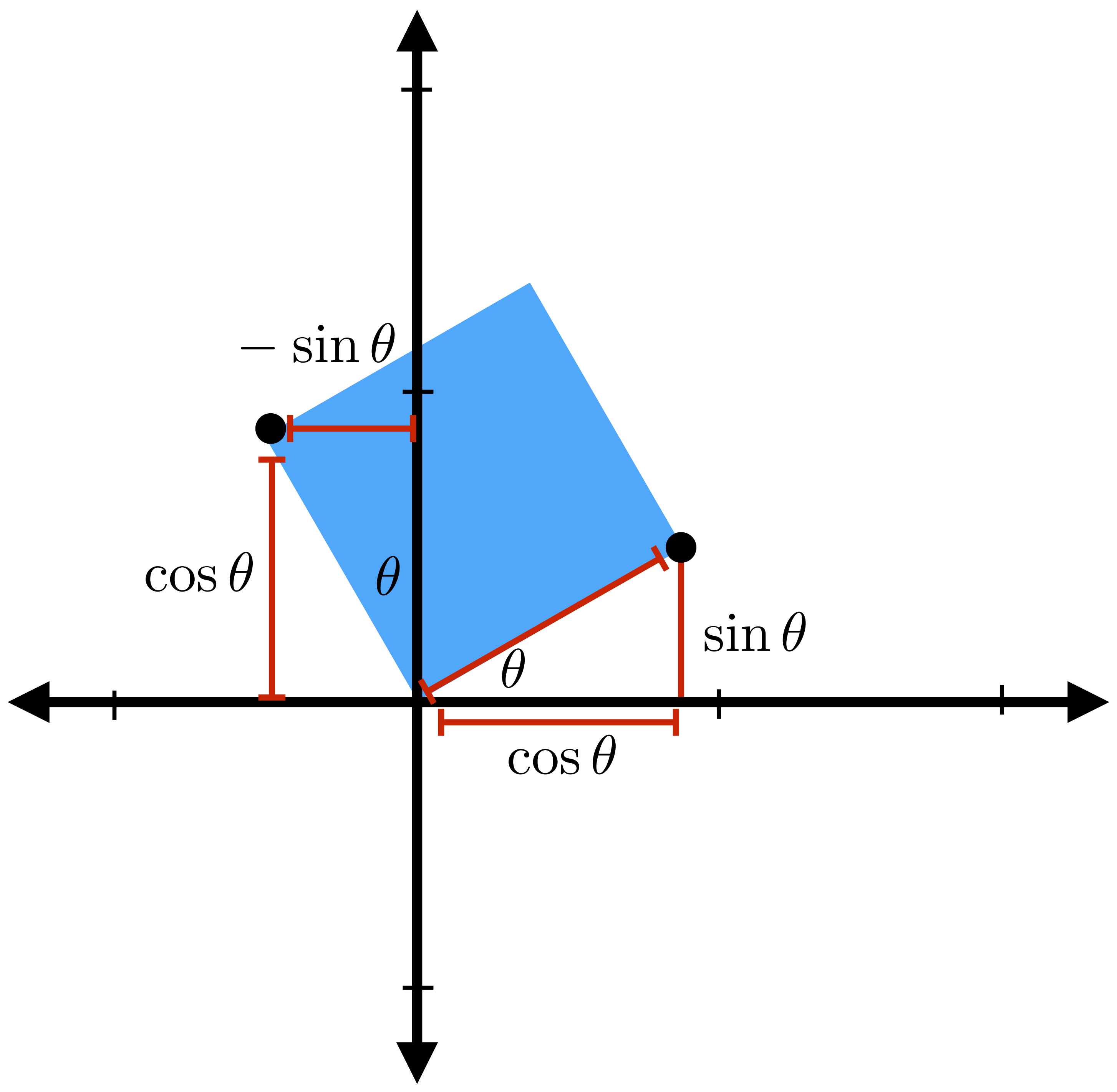
$$R_\theta = \begin{bmatrix} \cos(\theta) & \cos(\theta + \pi/2) \\ \sin(\theta) & \sin(\theta + \pi/2) \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

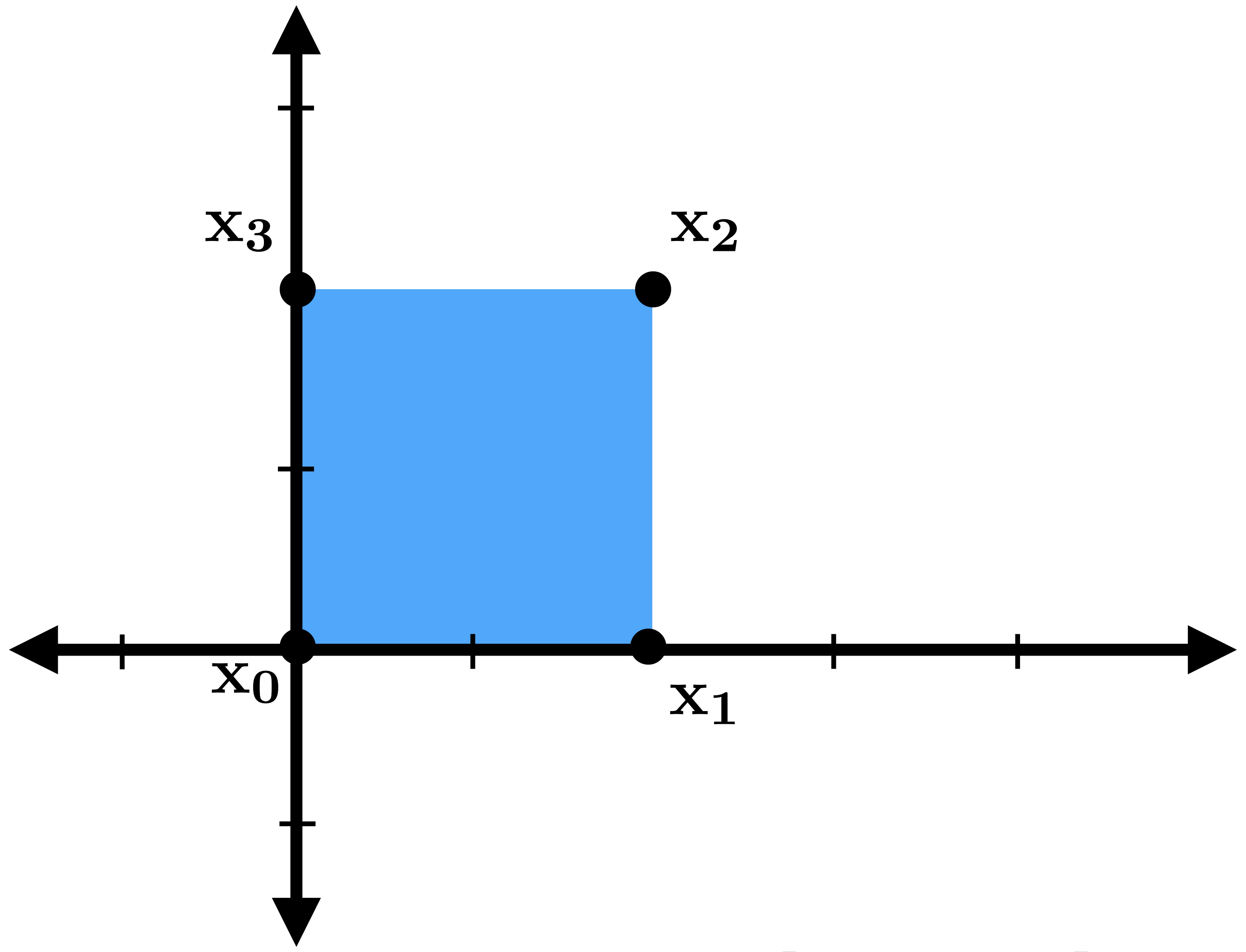
Rotation matrix (2D): another way...

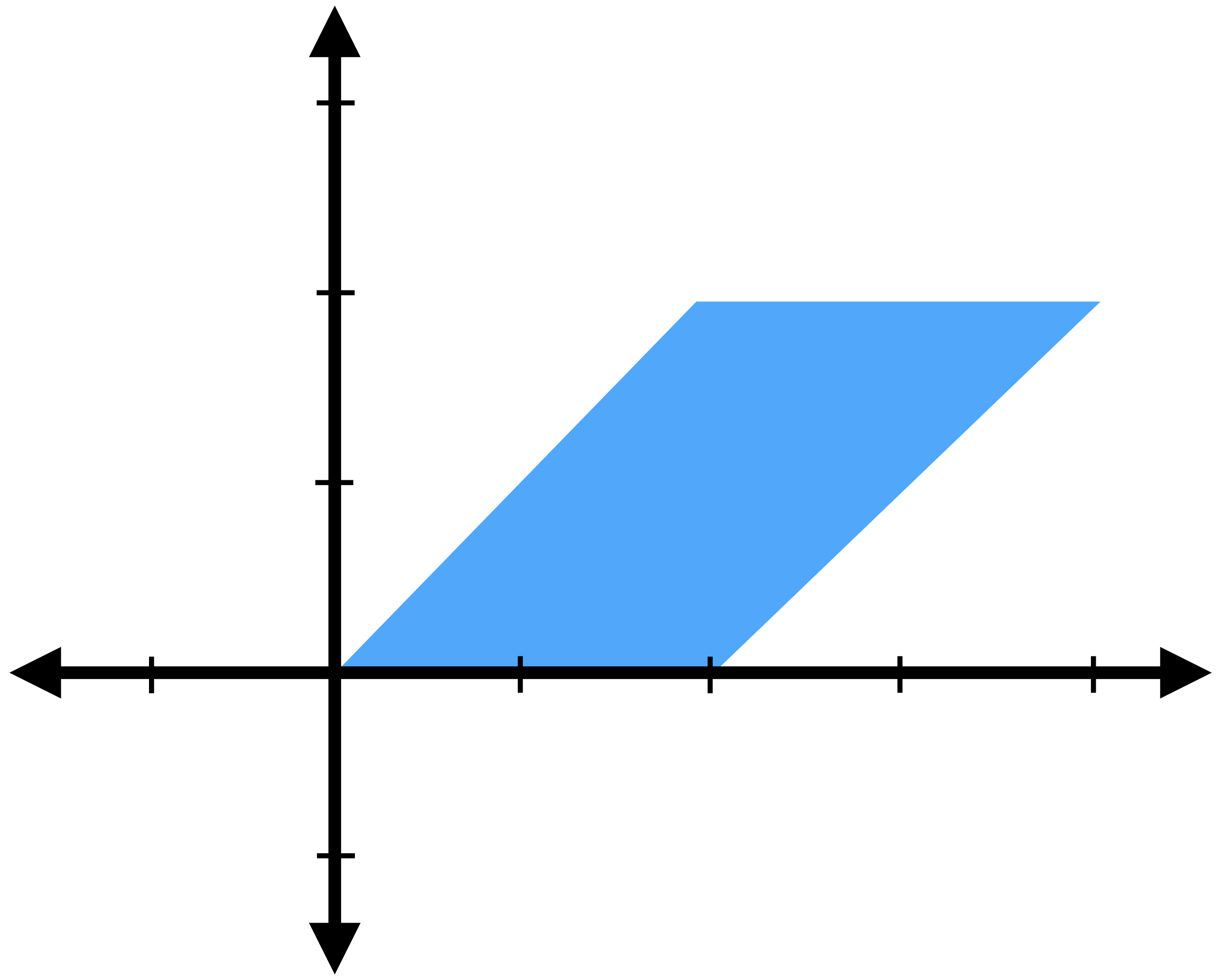
$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$





Shear



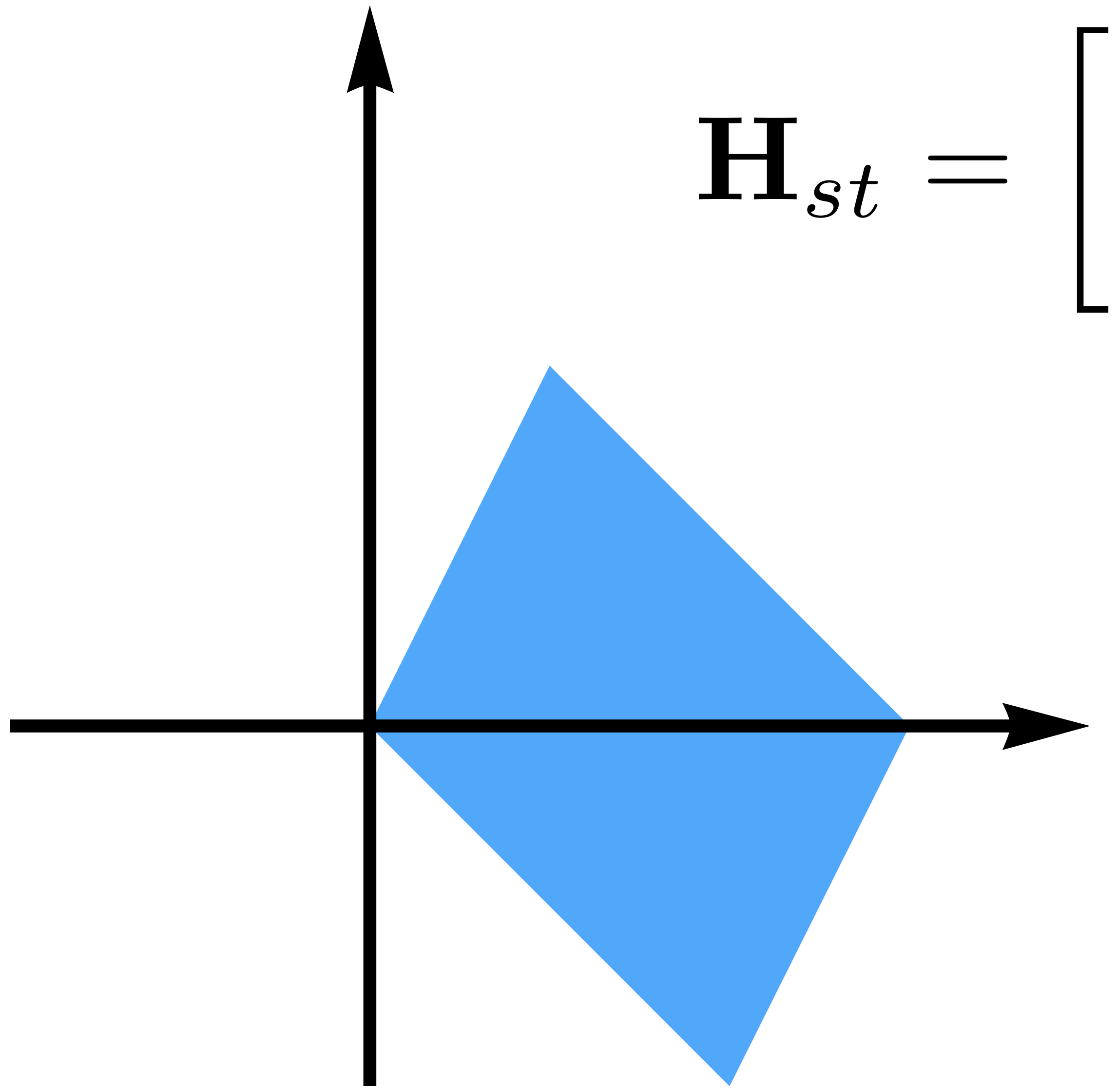


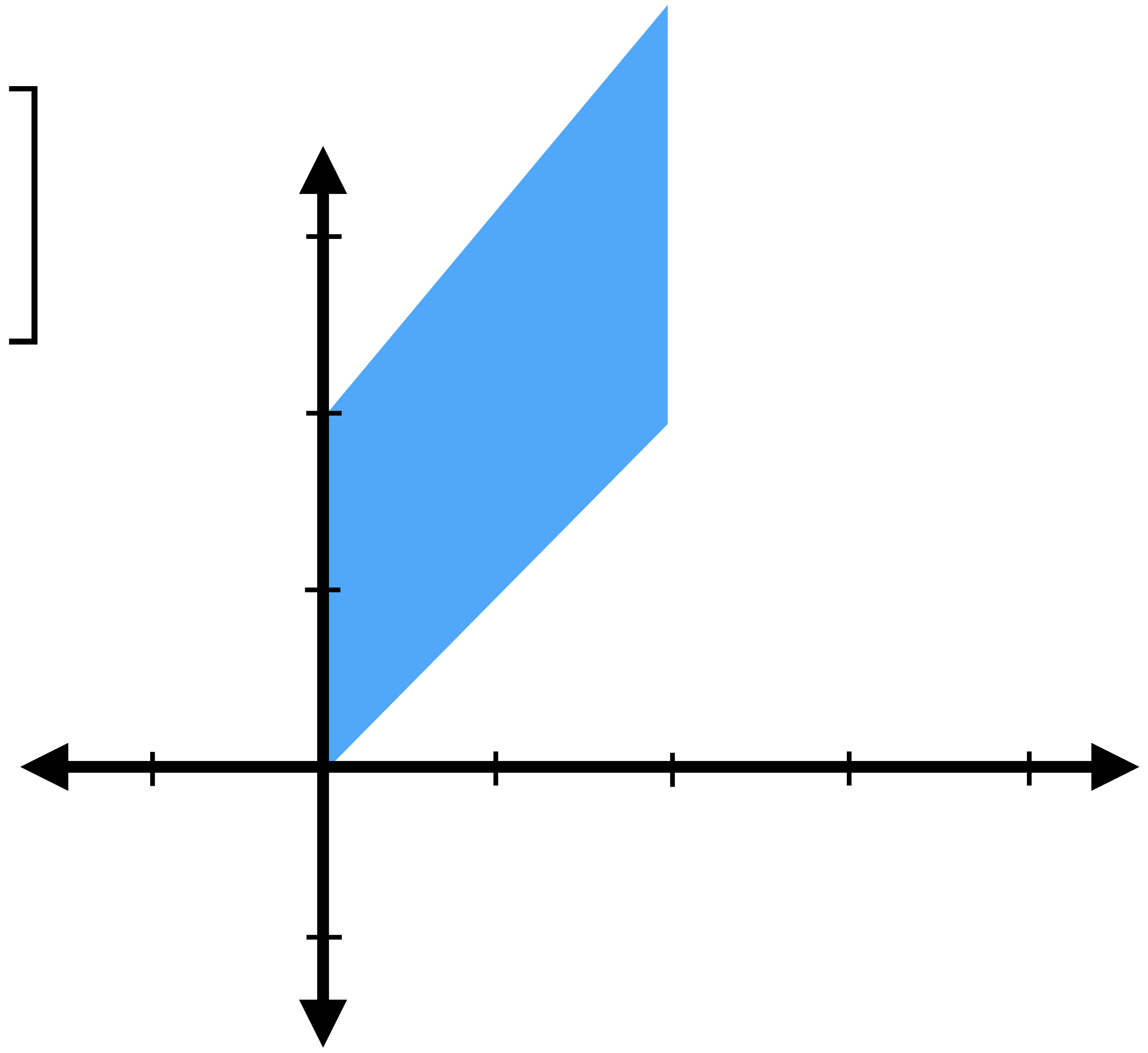
Shear in x:

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

Arbitrary shear:

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

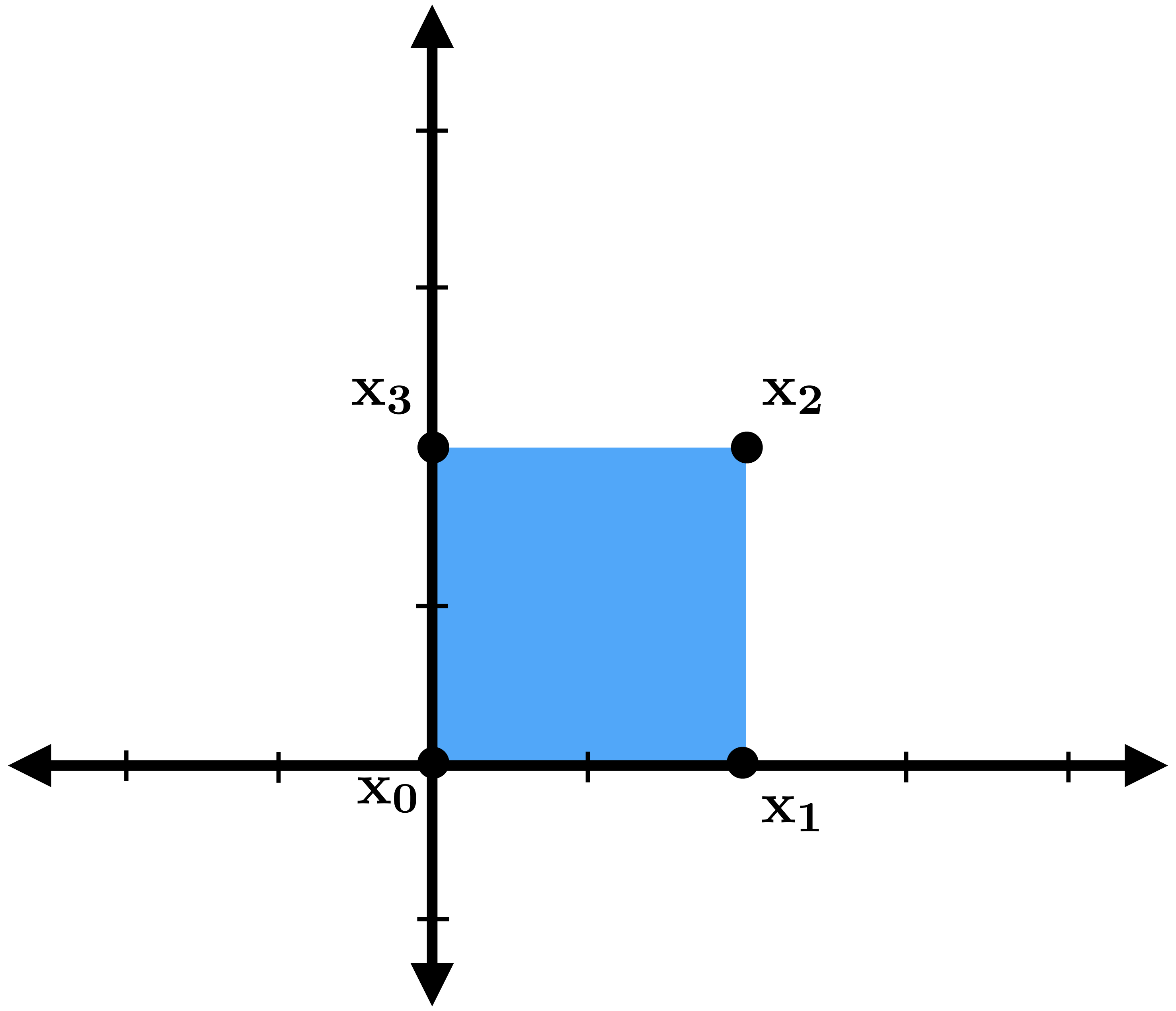


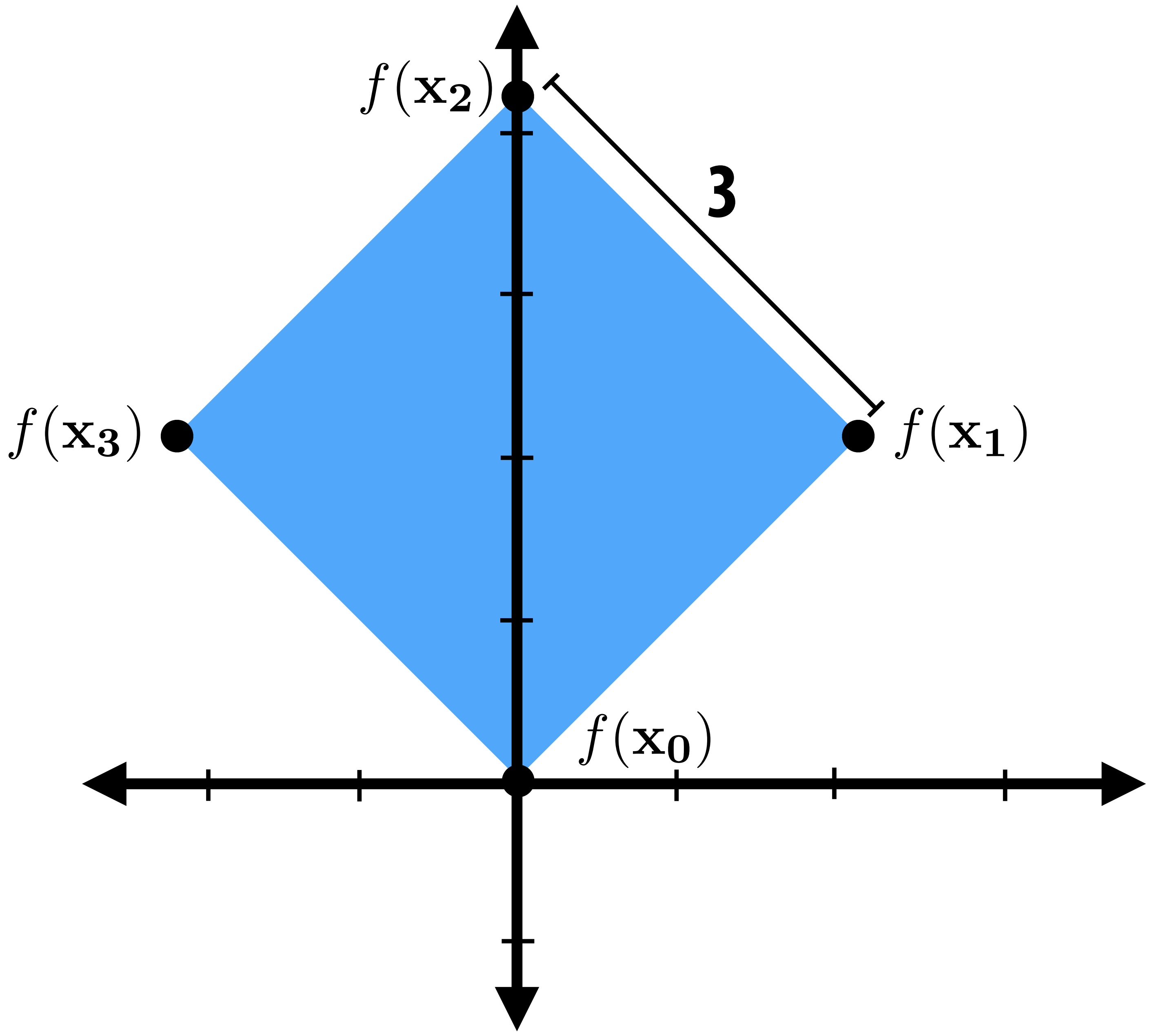


Shear in y:

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

How do we compose linear transformations?





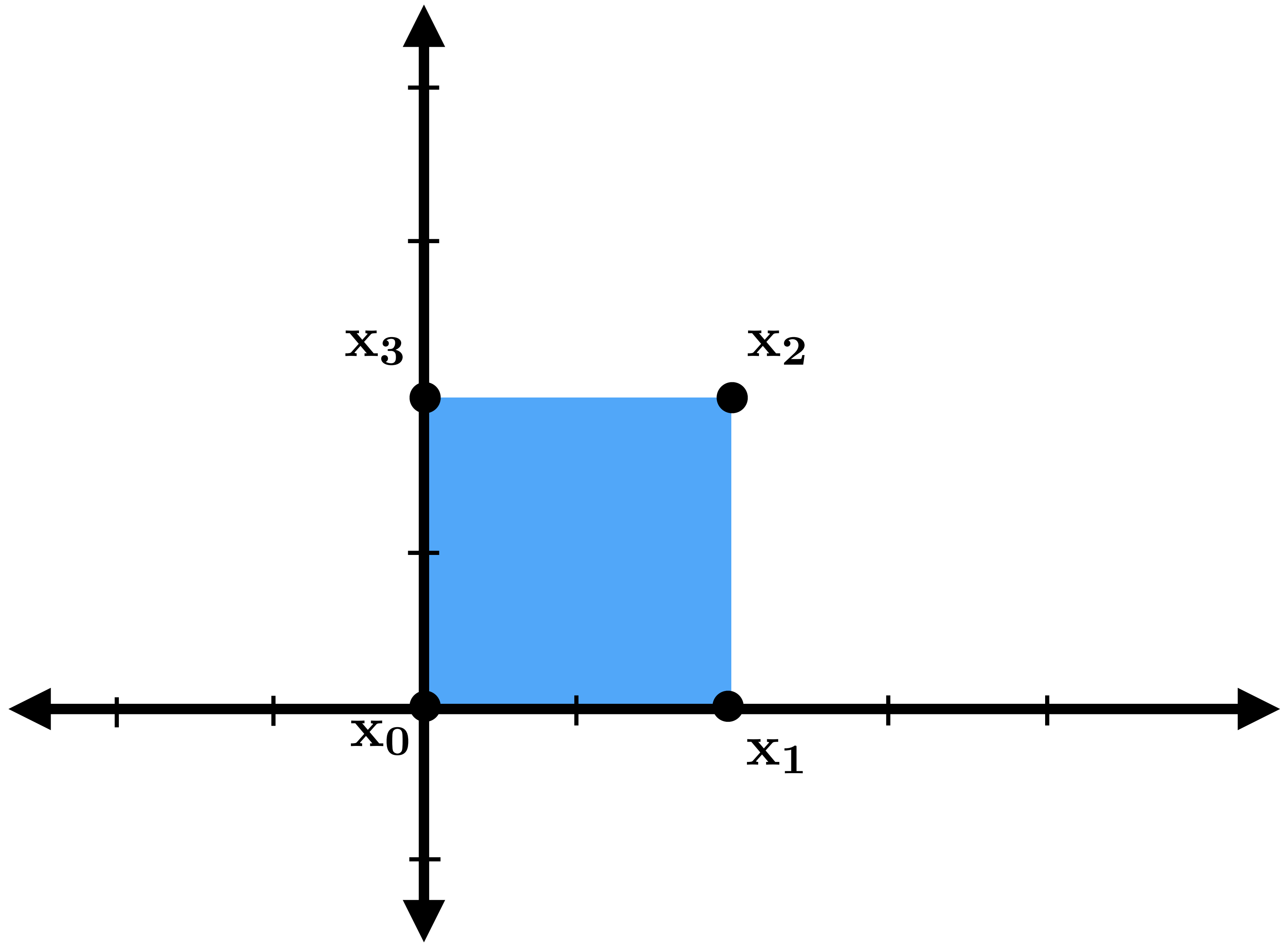
Compose linear transformations via matrix multiplication.
This example: uniform scale, followed by rotation

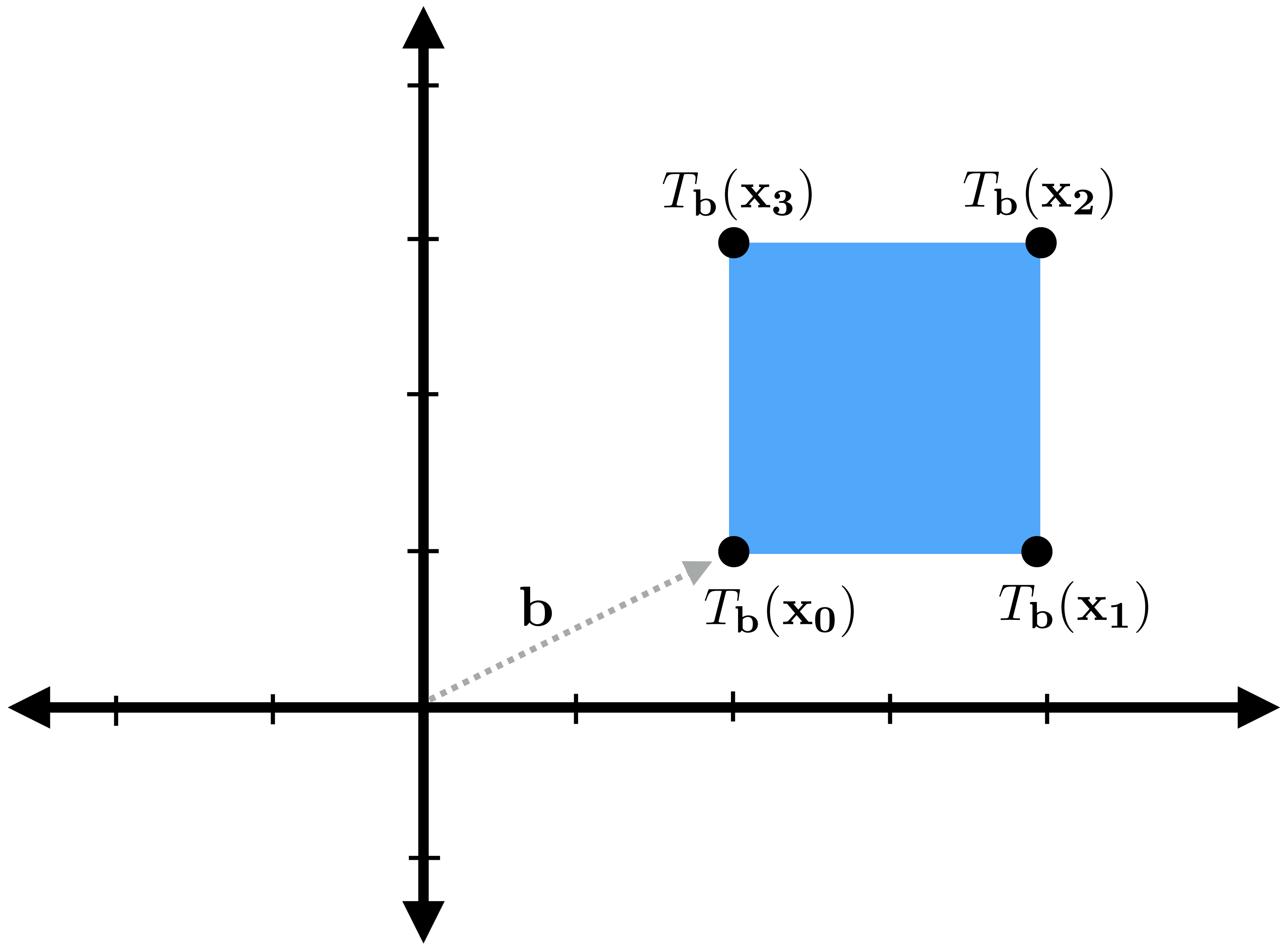
$$f(\mathbf{x}) = R_{\pi/4}\mathbf{S}_{[1.5,1.5]}\mathbf{x}$$

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

How do we deal with translation? (Not linear)

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





Unfortunately, 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

$$\mathbf{x}_{\mathbf{out}\,x} = \mathbf{x}_x + \mathbf{b}_x$$

$$\mathbf{x}_{\mathbf{out}\,y} = \mathbf{x}_y + \mathbf{b}_y$$

Translation math

Affine transformations

Common class of transformations in graphics and animation

Coordinate vector, $\mathbf{x}$, gets transformed as

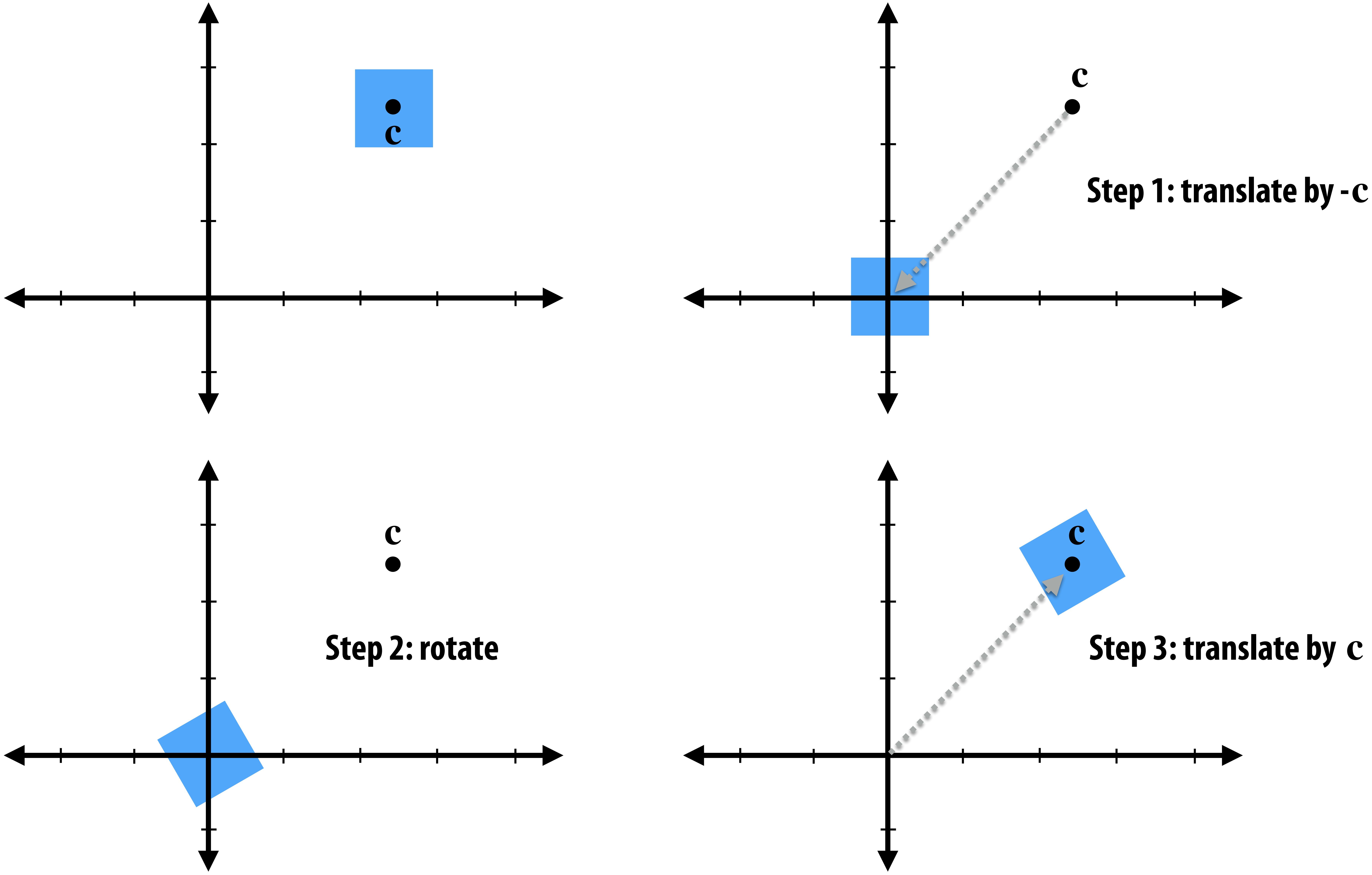
$$\mathbf{x}' = \mathbf{A}\,\mathbf{x} + \mathbf{b}$$

where

- $\mathbf{A}$ is a linear matrix operator representing scale, shear, rotation and reflections,
- $\mathbf{b}$ is a translation.

- Example: Rigid body motion $\mathbf{x}' = \mathbf{R}\,\mathbf{x} + \mathbf{b}$ where $\mathbf{R}$ is a rotation, and $\mathbf{b}$ is a translation.
- CS248A covers homogeneous coordinates
  - Used to write affine transformations as linear matrix operators
  - Used extensively in geometry and rendering

Common task: rotate about a point c



Common task: rotate $\mathbf{x}$ about a point $\mathbf{c}$

- Given $\mathbf{x}$, what is the overall affine transformation,

$$\mathbf{A}\mathbf{x} + \mathbf{b}\,?$$

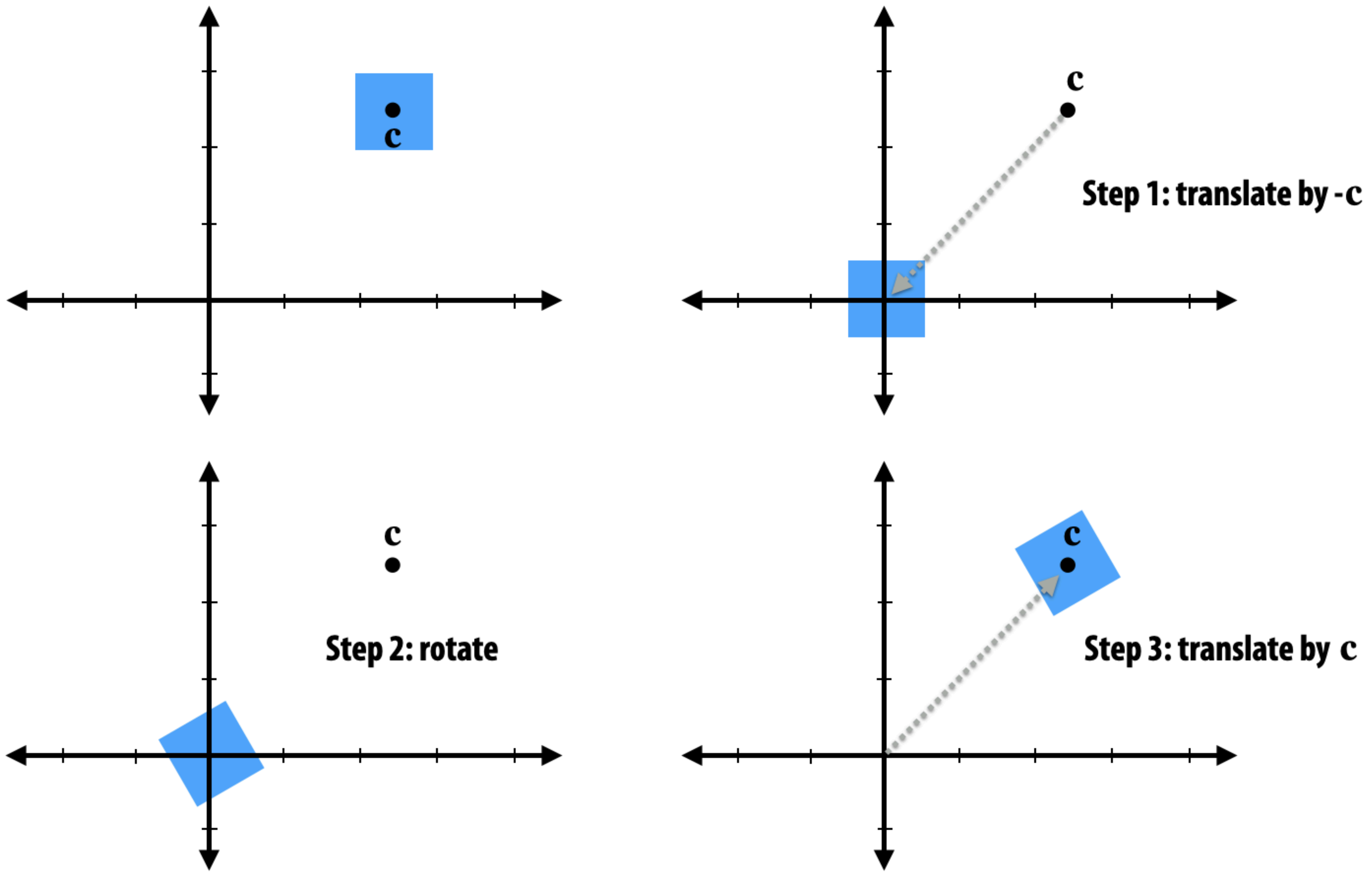
- Step 1: $\mathbf{x} - \mathbf{c}$
- Step 2: $\mathbf{R}\,(\mathbf{x} - \mathbf{c})$
- Step 3: $\mathbf{R}\,(\mathbf{x} - \mathbf{c}) + \mathbf{c}$
- Resulting transformation is

$$\mathbf{x}' = \mathbf{R}\,\mathbf{x} + (\mathbf{I} - \mathbf{R})\,\mathbf{c}$$

- So that

$$\mathbf{A} = \mathbf{R}$$

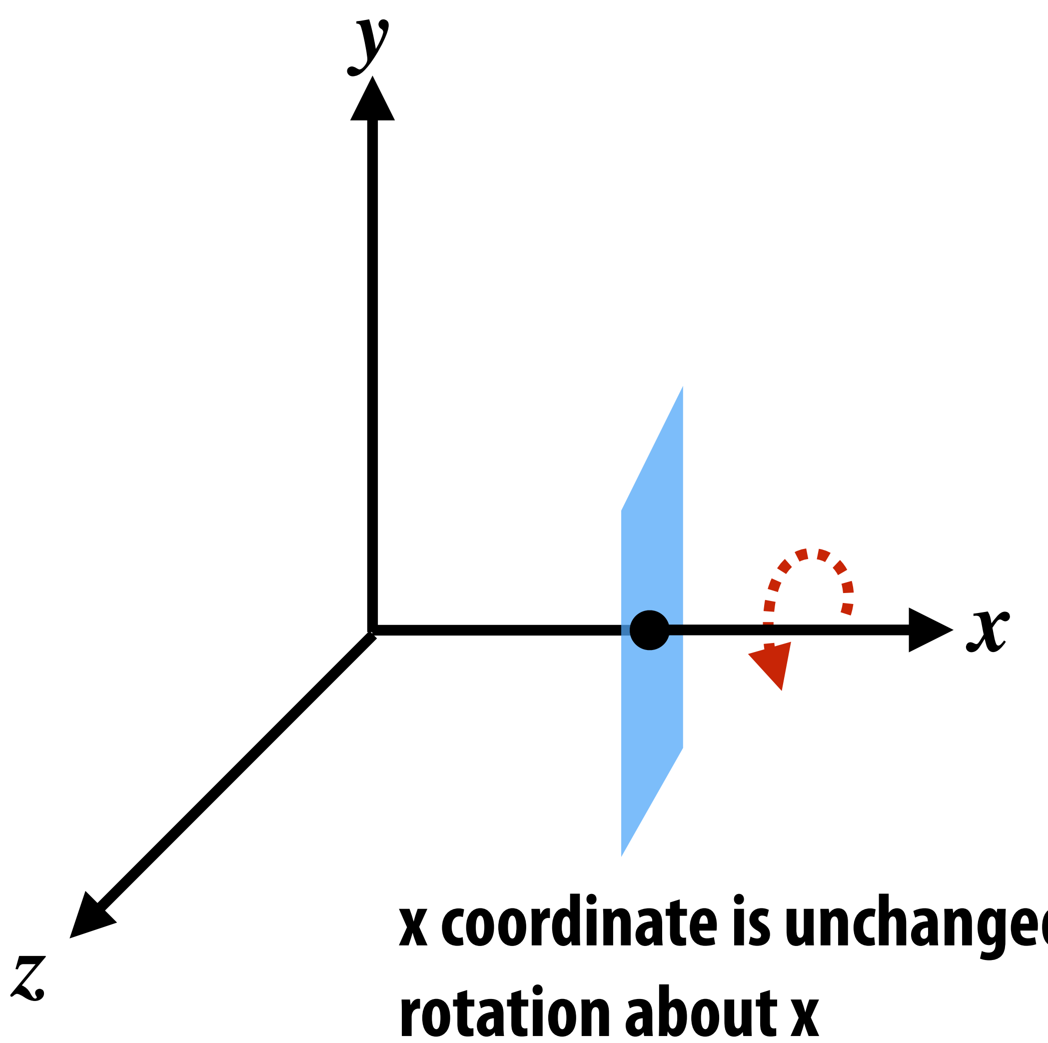
$$\mathbf{b} = \mathbf{c} - \mathbf{R}\mathbf{c}$$



Rotations in 3D

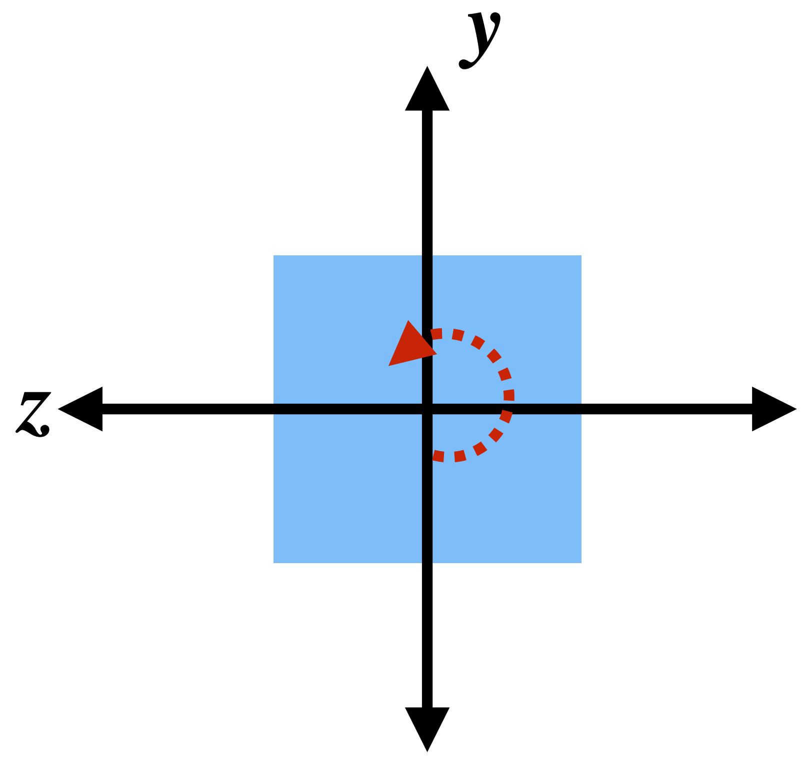
Rotation about x axis:

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



x coordinate is unchanged by rotation about x

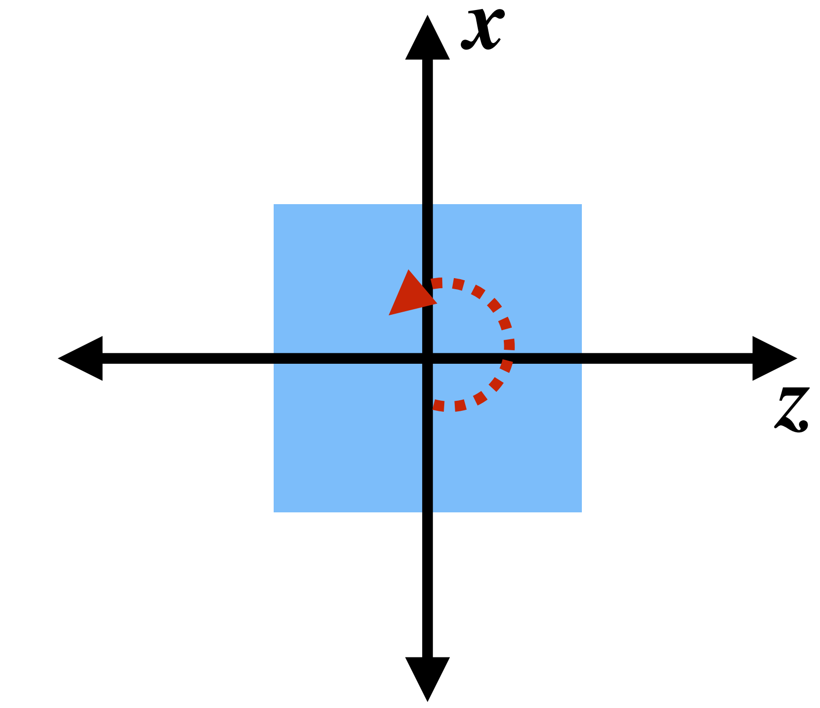
View looking down -x axis:



Rotation about y axis:

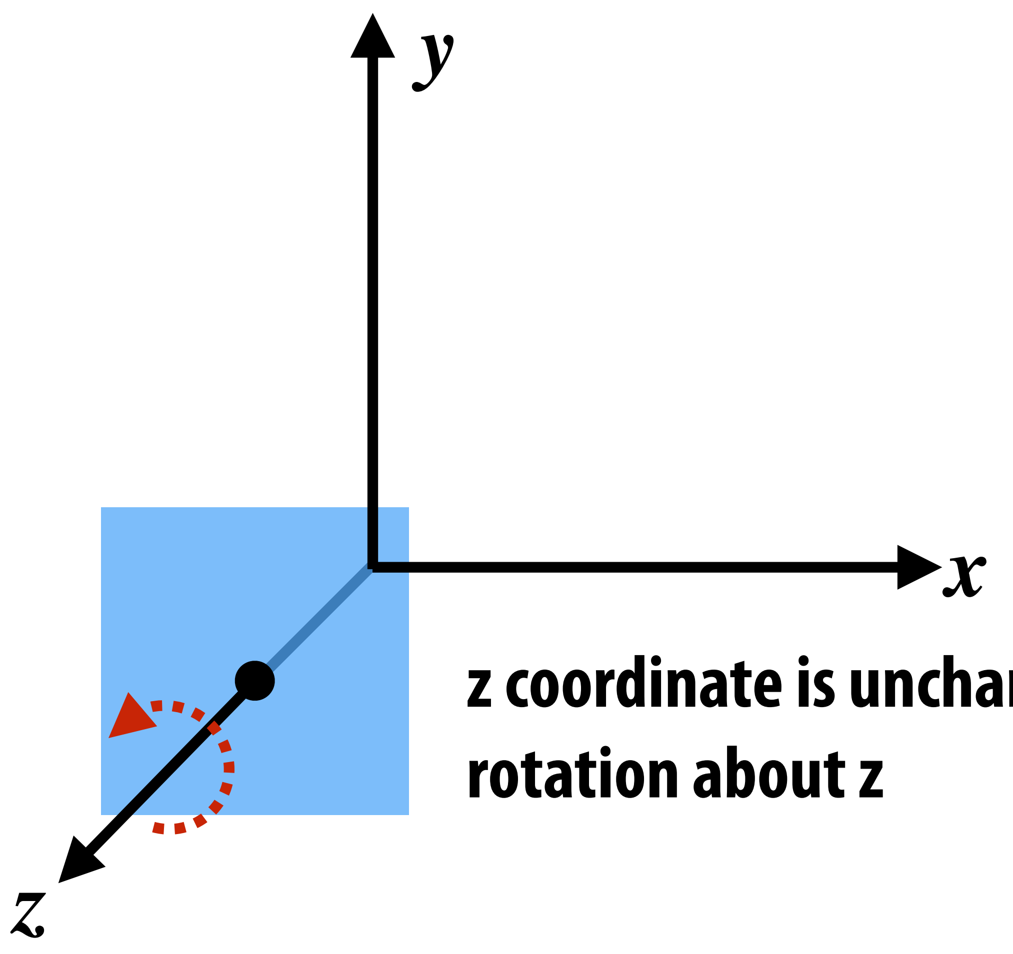
$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

View looking down -y axis:



Rotation about z axis:

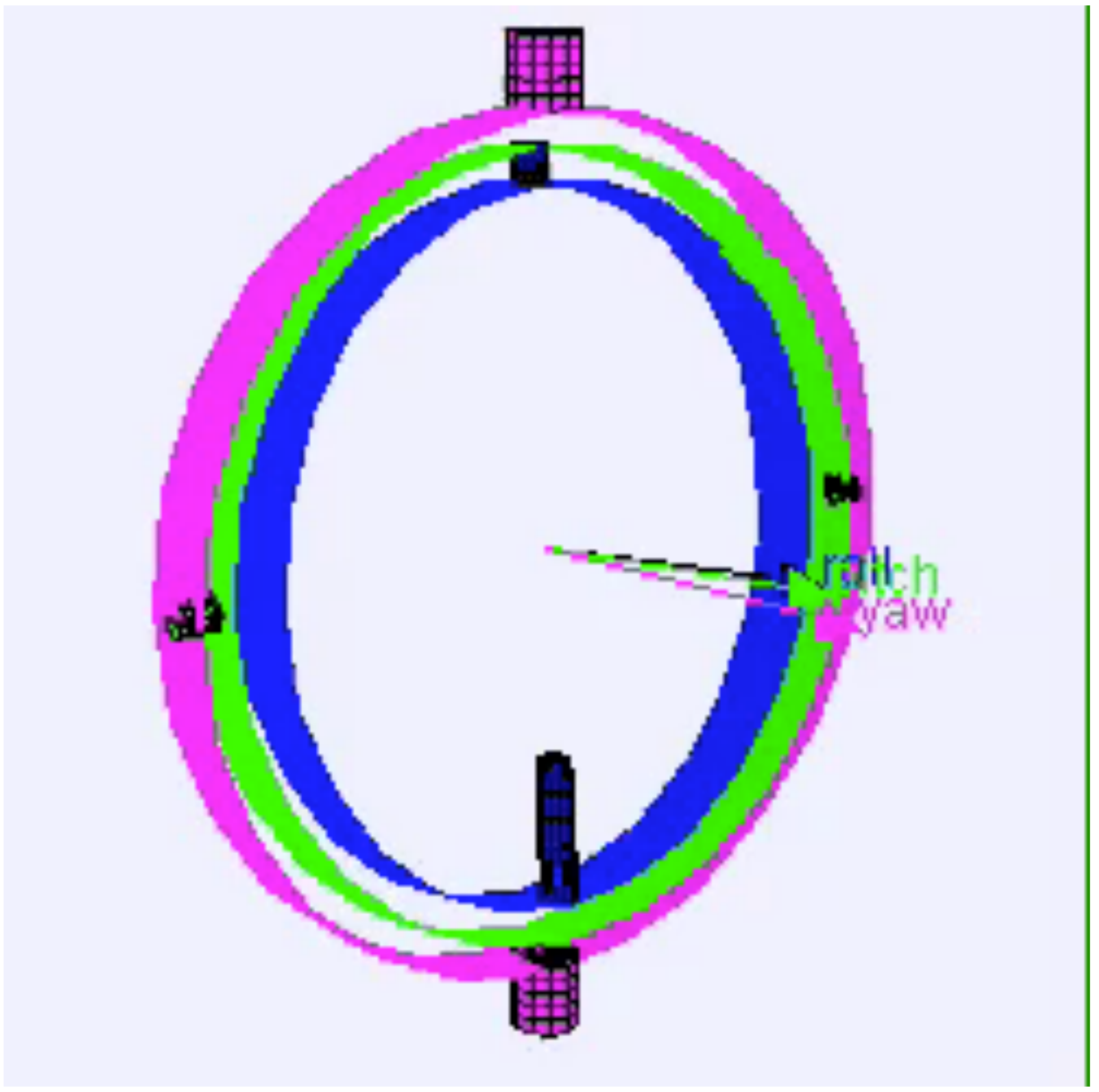
$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

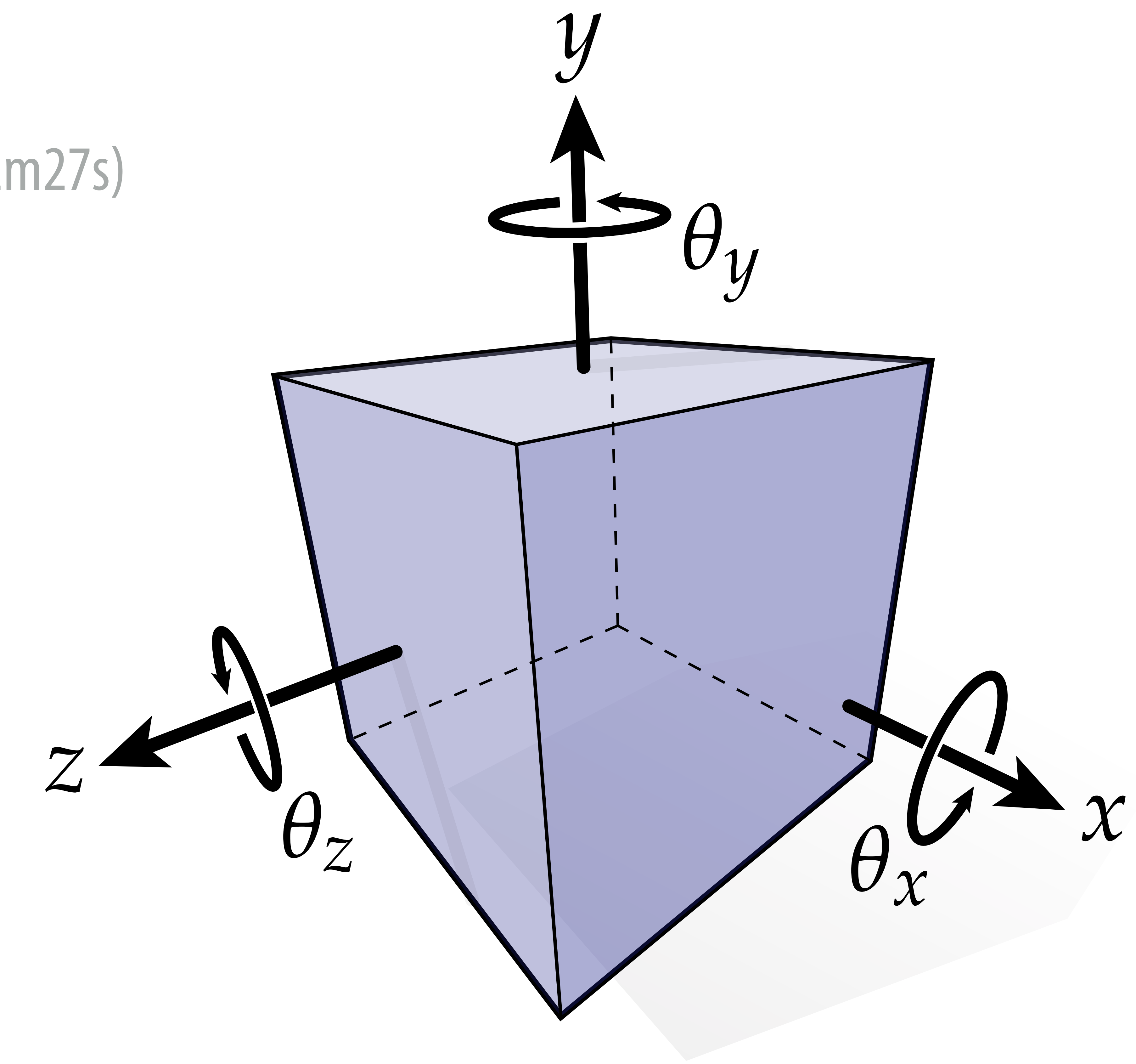


z coordinate is unchanged by rotation about z

Representing rotations in 3D—Euler angles

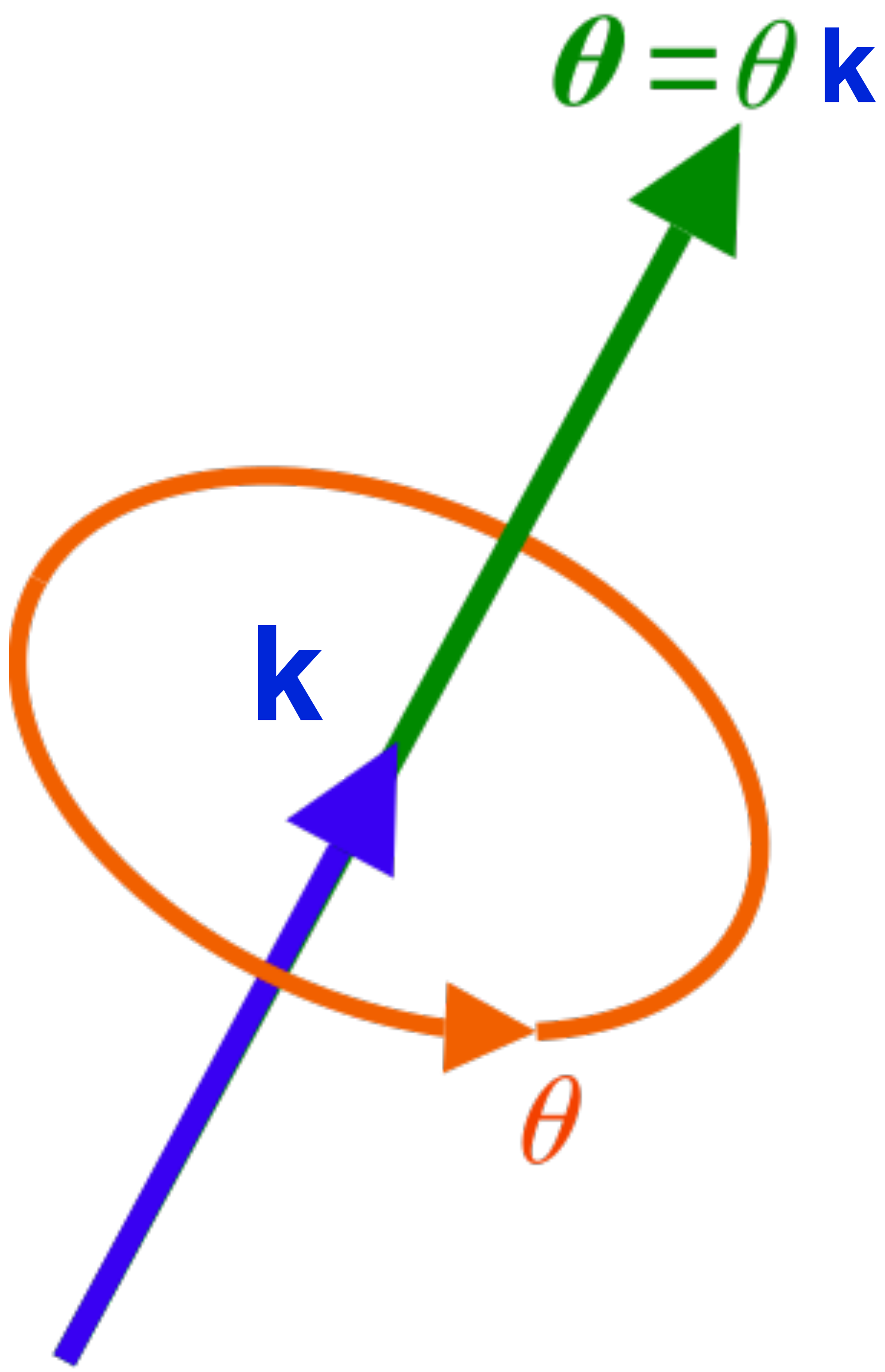
- How do we express rotations in 3D?
- One idea: we know how to do 2D rotations
- Why not simply apply rotations around the three axes? (X,Y,Z)
- Scheme is called *Euler angles*
- PROBLEM: "Gimbal Lock" [YouTube Video] (1m20s-2m27s)





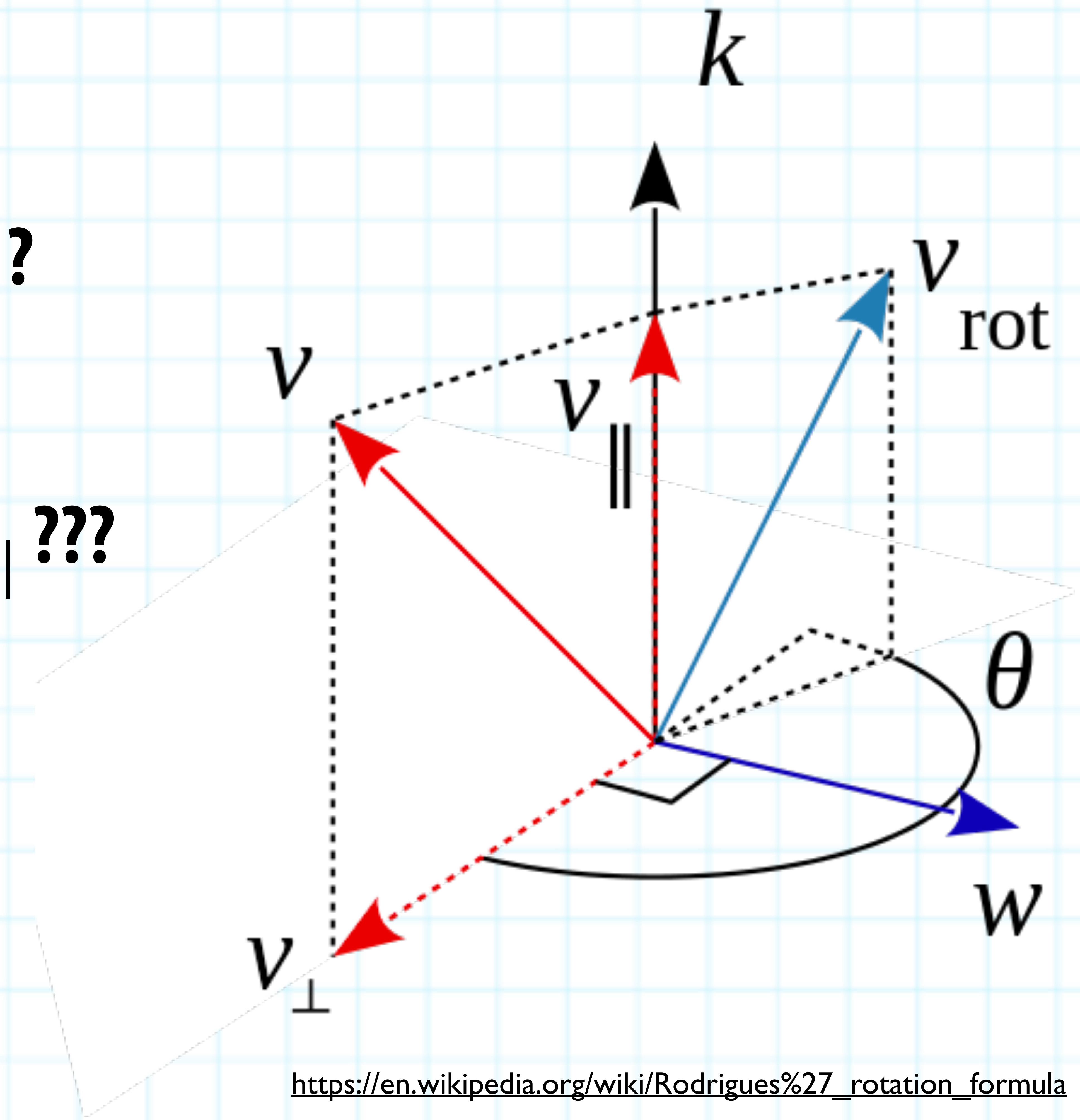
Alternative representations of 3D rotations

- Quaternions (Karen will cover later)
- Axis-angle rotations



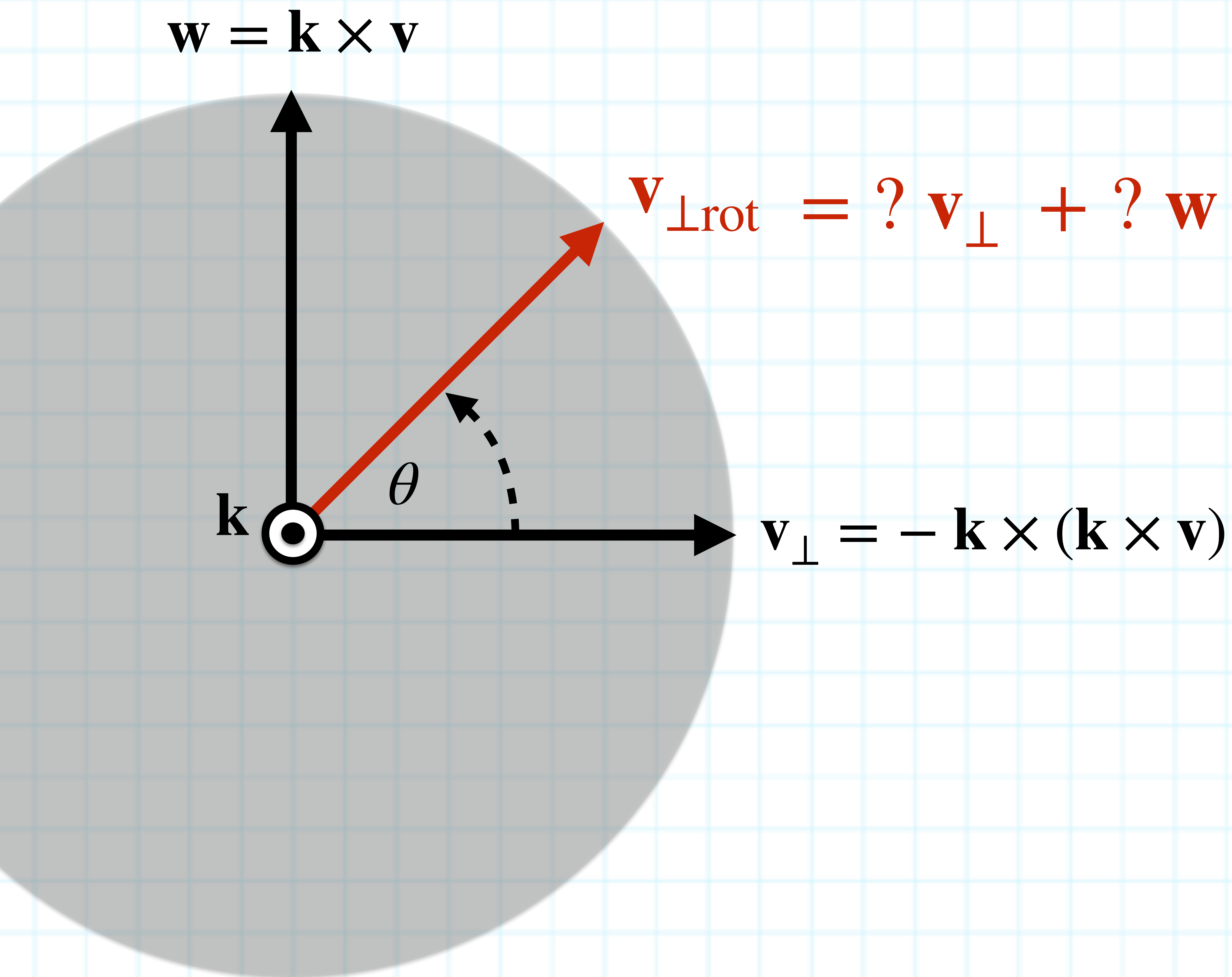
What is the 3D rotation matrix $\mathbf{R}$ for a rotation by θ about axis $\mathbf{k}$?

- Let's derive it! (It's actually not that hard)
- Consider the rotated vector, $\mathbf{v}_{\text{rot}} = \mathbf{R}\,\mathbf{v}$
- What happens if $\mathbf{v}$ is parallel to $\mathbf{k}$, e.g., $\mathbf{v}_{\|} = \alpha\,\mathbf{k}$?
  - Nothing! Since $\mathbf{R}\,\mathbf{k} = \mathbf{k}$, then $\mathbf{R}\,\mathbf{v}_{\|} = \mathbf{v}_{\|}$.
- What about the perpendicular part, $\mathbf{v}_{\perp} = \mathbf{v} - \mathbf{v}_{\|}$???
- Just use 2D rotation on $\mathbf{v}_{\perp}$!
  - $\mathbf{v}_{\perp} = -\,\mathbf{k} \times (\mathbf{k} \times \mathbf{v}) \xrightarrow{\mathbf{R}} \mathbf{v}_{\perp\text{rot}} = ??$
  - $\mathbf{w} = \mathbf{k} \times \mathbf{v}$

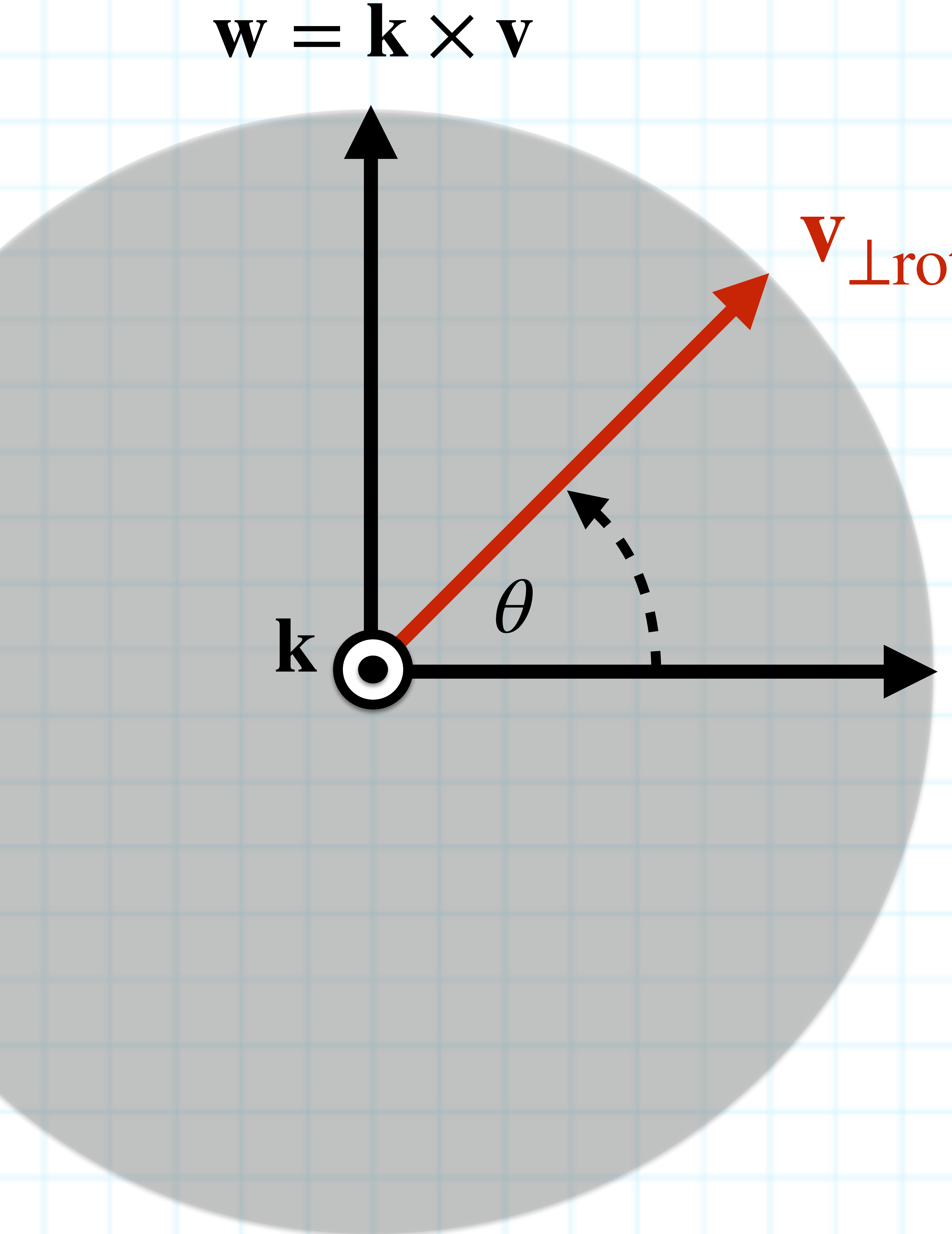


https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula

What is the 3D rotation matrix $\mathbf{R}$ for a rotation by θ about axis $\mathbf{k}$?



What is the 3D rotation matrix $\mathbf{R}$ for a rotation by θ about axis $\mathbf{k}$?



$$\mathbf{v}_{\perp\mathrm{rot}} = \cos\theta\ \mathbf{v}_\perp + \sin\theta\ \mathbf{w}$$
$$= -\cos\theta\ \mathbf{k} \times (\mathbf{k} \times \mathbf{v}) + \sin\theta\ \mathbf{k} \times \mathbf{v}$$

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

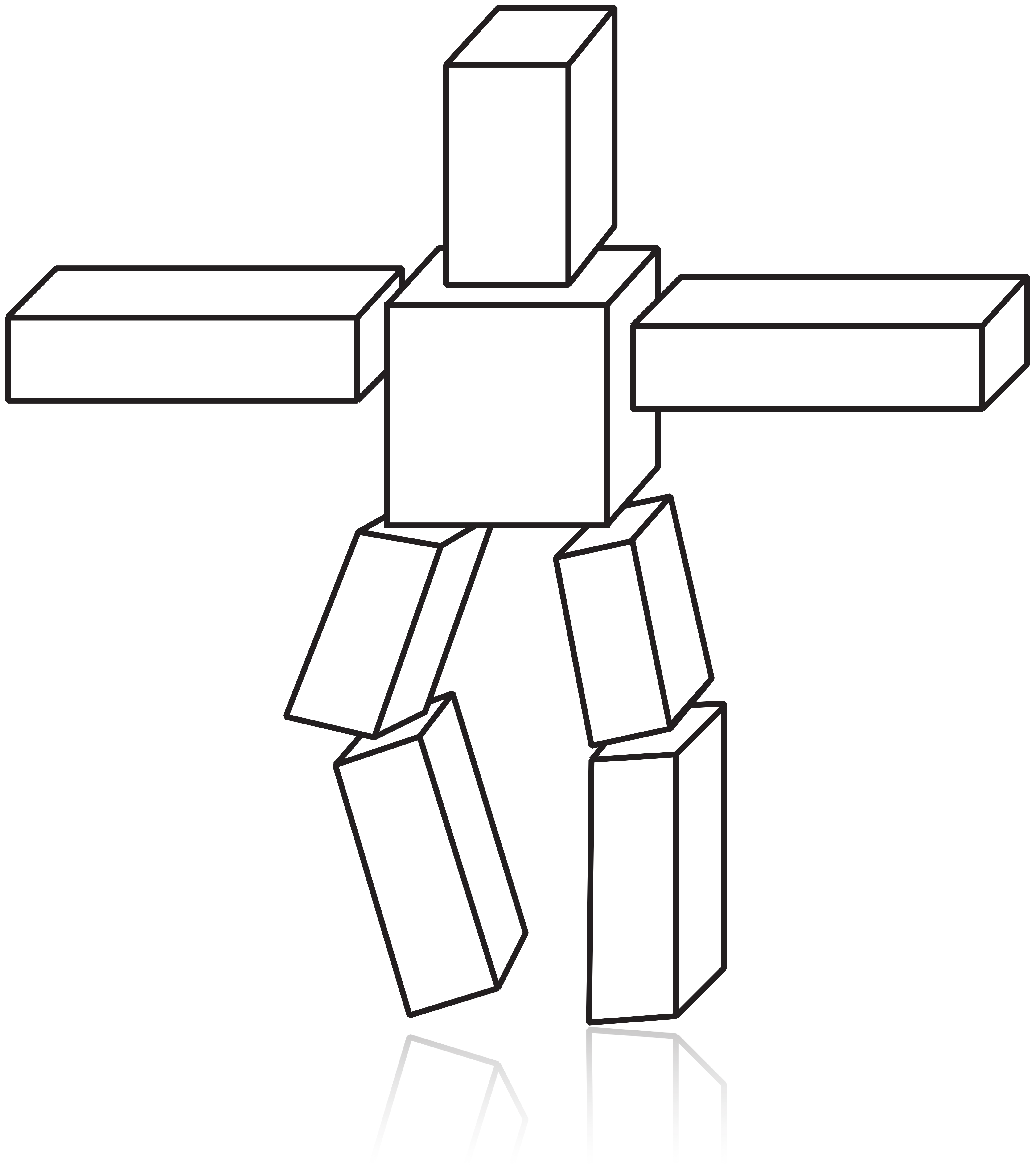
What is the 3D rotation matrix $\mathbf{R}$ for a rotation by θ about axis $\mathbf{k}$?

$$
\begin{aligned}
\mathbf{v}_{\mathrm{rot}} &= \mathbf{v}_{\|} + \mathbf{v}_{\perp\mathrm{rot}} \\
&= \left(\mathbf{v} - \mathbf{v}_{\perp}\right) + \mathbf{v}_{\perp\mathrm{rot}} \\
&= \left(\mathbf{v} + \mathbf{k} \times (\mathbf{k} \times \mathbf{v})\right) + \left(-\cos\theta\, \mathbf{k} \times (\mathbf{k} \times \mathbf{v}) + \sin\theta\, \mathbf{k} \times \mathbf{v}\right) \\
&= \mathbf{v} + (1 - \cos\theta)\, \mathbf{k} \times (\mathbf{k} \times \mathbf{v}) + \sin\theta\, \mathbf{k} \times \mathbf{v} \\
&= \left[\mathbf{I} + (1 - \cos\theta)\, \mathbf{K}^2 + \sin\theta\, \mathbf{K}\right] \mathbf{v} \quad \text{where} \quad \mathbf{K} \equiv \text{"}\mathbf{k} \times \text{"} \\
&= \mathbf{R}\, \mathbf{v}
\end{aligned}
$$

$$
= \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix}
$$

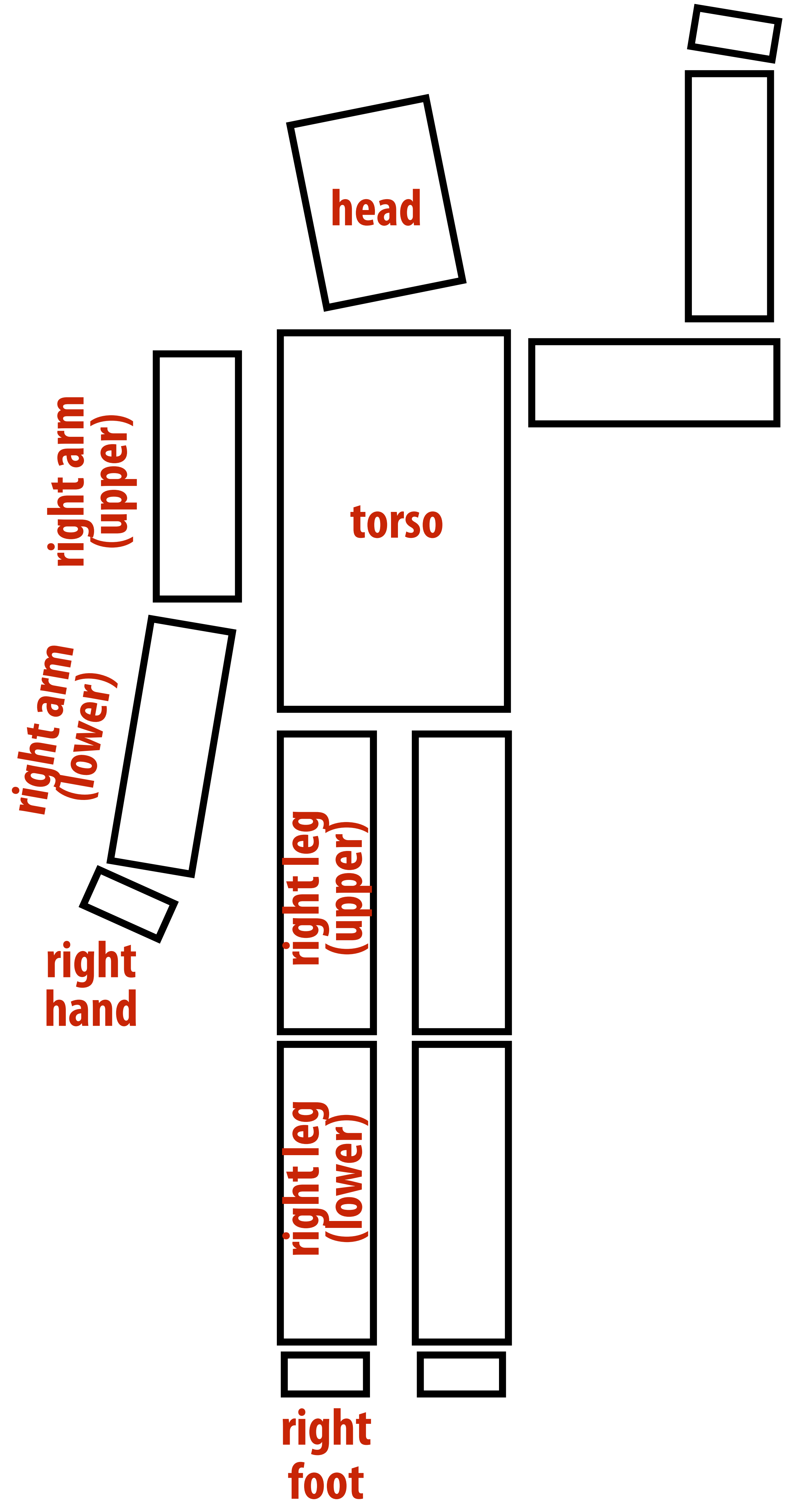
We just derived the famous <u>Rodrigues' Formula</u>!!!

Let's make a 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

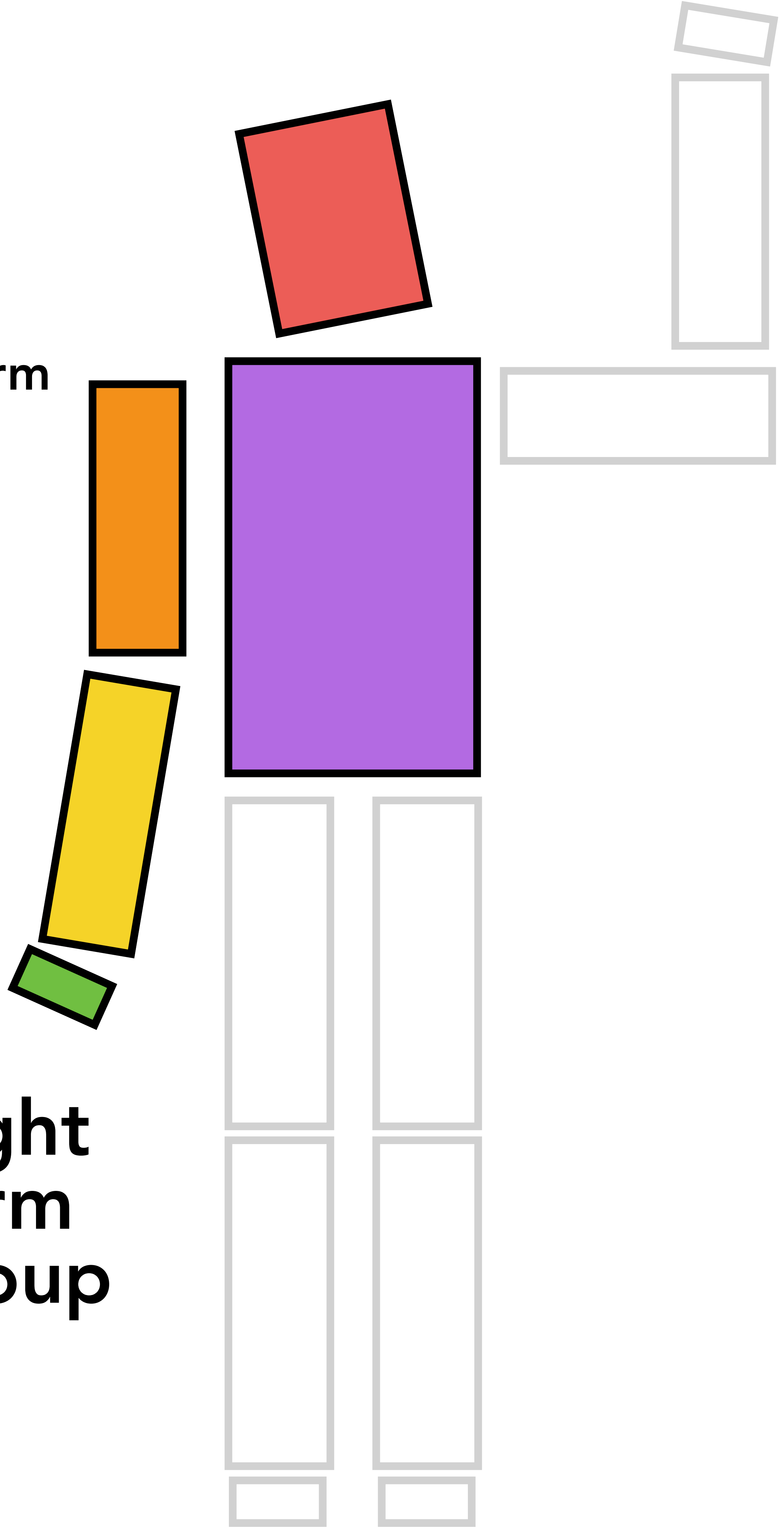
Skeleton - hierarchical representation

```
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();
                      translate(-2, 3);
                      rotate(rightShoulderRotation);
                      drawUpperArm();
                      pushmatrix();
                        translate(0, -3);
                        rotate(elbowRotation);
                        drawLowerArm();
                        pushmatrix();
                          translate(0, -3);
                          rotate(wristRotation);
                          drawHand();
                        popmatrix();
                      popmatrix();
                    popmatrix();
                    ....
```

right hand

right lower arm group

right arm group



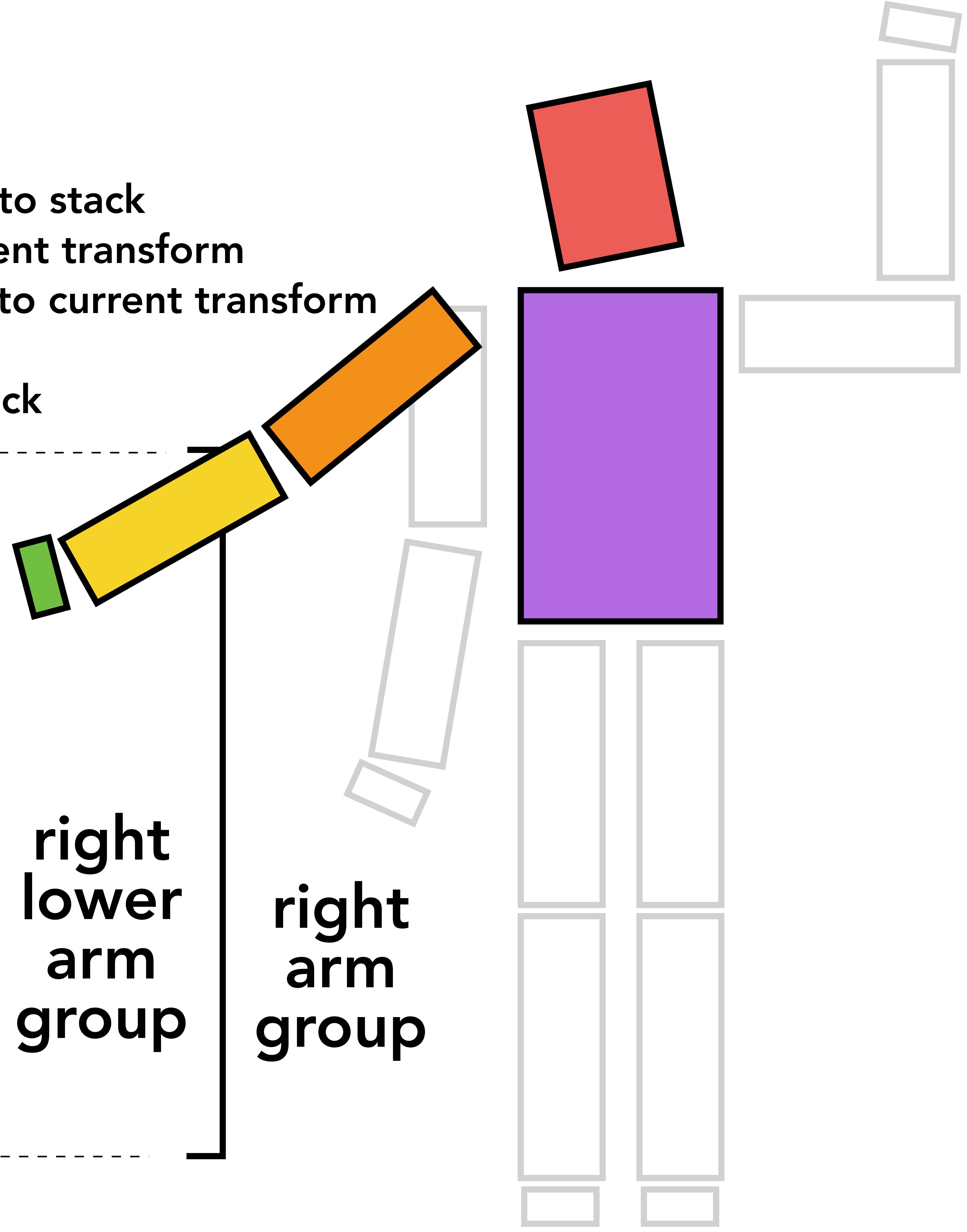
Skeleton - hierarchical representation

```
translate(0, 10);

    drawTorso();

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

right hand

right lower arm group

right arm group



Transformations in OpenProcessing

- https://p5js.org/reference/

Transform

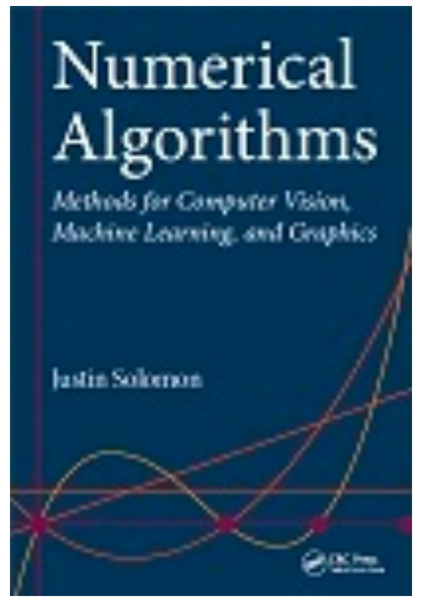
applyMatrix()
resetMatrix()
rotate()
rotateX()
rotateY()
rotateZ()
scale()
shearX()
shearY()
translate()

push()
pop()

Ordinary Differential Equations (ODEs)

- **Initial value problems (IVPs)**
  - **Common in animation**

- **Some slides from CS205A on Introduction to ODEs**
  - **More on time-stepping methods next class (some for end of HW1)**
  - **Textbook reference:**



- **Solomon, Justin. <u>Numerical Algorithms.</u> Textbook published by AK Peters/ CRC Press, 2015**

Next Class: Particle Systems

Ordinary Differential Equations I

CS 205A:
Mathematical Methods for Robotics, Vision, and Graphics

Doug James (and Justin Solomon)

Today: Initial Value Problems

$$\text{Find } f(t) : \mathbb{R} \to \mathbb{R}^n$$
$$\text{Satisfying } F[t, f(t), f'(t), f''(t), \ldots, f^{(k)}(t)] = 0$$
$$\text{Given } f(0), f'(0), f''(0), \ldots, f^{(k-1)}(0)$$

Most Famous Example

$$\vec{F} = m\vec{a}$$

Newton's second law

$\vec{F}(t, \vec{x}, \vec{x}')$ usual expression of force

n particles $\implies$ simulation in $\mathbb{R}^{3n}$

Examples of ODEs

- $y' = 1 + \cos t$: solved by integrating both sides
- $y' = ay$: linear in y, no dependence on t
- $y' = ay + e^t$: time and value-dependent
- $y'' + 3y' - y = t$: multiple derivatives of y
- $y'' \sin y = e^{ty'}$: nonlinear in y and t

Reasonable Assumption

Explicit ODE

An ODE is *explicit* if can be written in the form

$$f^{(k)}(t) = F[t, f(t), f'(t), f''(t), \dots, f^{(k-1)}(t)].$$

Reduction to First Order

$$f^{(k)}(t) = F[t, f(t), f'(t), f''(t), \ldots, f^{(k-1)}(t)]$$

Reduction to First Order

$$f^{(k)}(t) = F[t, f(t), f'(t), f''(t), \ldots, f^{(k-1)}(t)]$$

$$\frac{d}{dt}\begin{pmatrix} f_0(t) \\ f_1(t) \\ \vdots \\ f_{k-2}(t) \\ f_{k-1}(t) \end{pmatrix} = \begin{pmatrix} f_1(t) \\ f_2(t) \\ \vdots \\ f_{k-1}(t) \\ F[t, f_0(t), f_1(t), \ldots, f_{k-1}(t)] \end{pmatrix}$$

Example

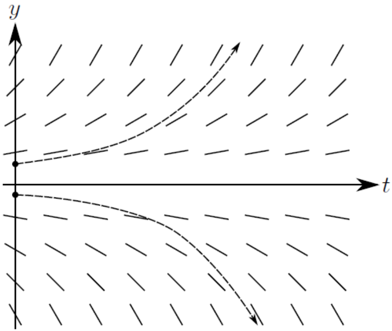
$$y''' = 3y'' - 2y' + y$$

Example

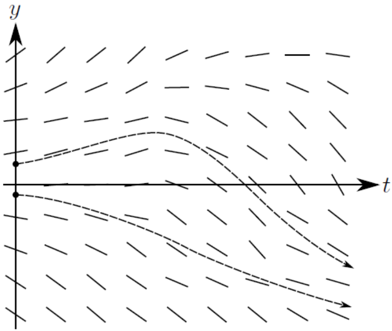
$$y''' = 3y'' - 2y' + y$$

$$\frac{d}{dt}\begin{pmatrix} y \\ z \\ w \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -2 & 3 \end{pmatrix}\begin{pmatrix} y \\ z \\ w \end{pmatrix}$$

Time Dependence

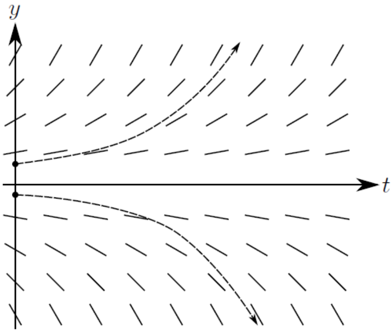


Time-independent

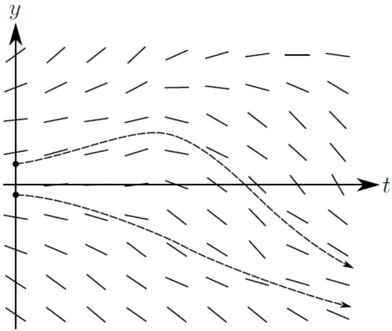


Time-dependent

Time Dependence



Time-independent



Time-dependent

Visualization: Slope field

Autonomous ODE

$$\vec{y}' = F[\vec{y}]$$

No dependence of F on t

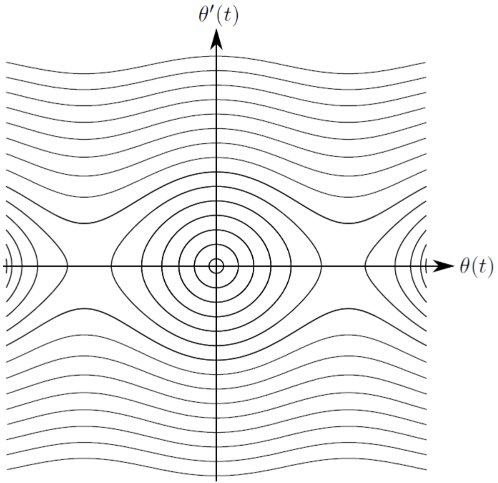
Autonomous ODE

$$\vec{y}' = F[\vec{y}]$$

No dependence of F on t

$$g'(t) = \begin{pmatrix} f'(t) \\ \bar{g}'(t) \end{pmatrix} = \begin{pmatrix} F[f(t), \bar{g}(t)] \\ 1 \end{pmatrix}$$

Visualization: Phase Space



$$\theta'' = -\sin\theta$$

Existence and Uniqueness

Theorem: Local existence and uniqueness

Suppose F is continuous and Lipschitz, that is, $\|F[\vec{y}] - F[\vec{x}]\|_2 \leq L\|\vec{y} - \vec{x}\|_2$ for some fixed $L \geq 0$. Then, the ODE $f'(t) = F[f(t)]$ admits exactly one solution for all $t \geq 0$ regardless of initial conditions.

Linearization of 1D ODEs

$$y' = F[y]$$

Linearization of 1D ODEs

$$y' = F[y]$$

$$\longrightarrow y' = ay + b$$

Linearization of 1D ODEs

$$y' = F[y]$$

$$\longrightarrow y' = ay + b$$

$$\longrightarrow \bar{y}' = a\bar{y}$$

Model Equation

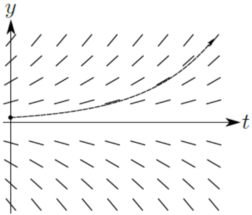
$$y' = ay$$

Model Equation

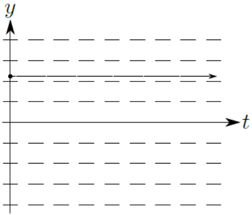
$$y' = ay$$

$$\implies y(t) = Ce^{at}$$

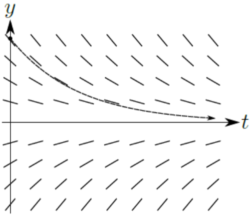
Stability: Visualization



$a > 0$



$a = 0$



$a < 0$

$$\boxed{y' = ay}$$

Three Cases

$$y' = ay, y(t) = Ce^{at}$$

1. $a = 0$: Stable

Three Cases

$$y' = ay, y(t) = Ce^{at}$$

1. $a = 0$: Stable
2. $a < 0$: Stable; solutions get closer

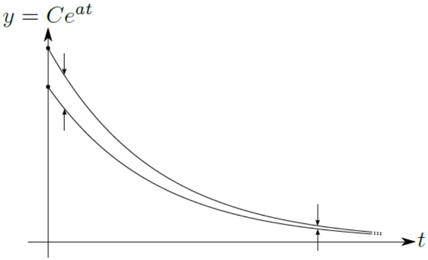
Three Cases

$$y' = ay, y(t) = Ce^{at}$$

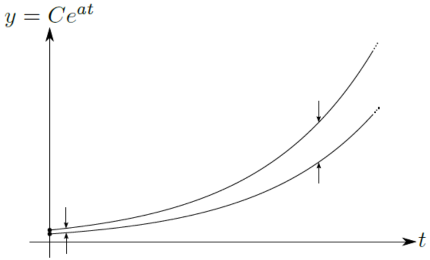
1. $a = 0$: Stable
2. $a < 0$: Stable; solutions get closer
3. $a > 0$: Unstable; mistakes in initial data amplified

Intuition for Stability

An *unstable* ODE magnifies mistakes in the initial conditions $y(0)$.



Stable $(a < 0)$



Unstable $(a > 0)$

Multidimensional Case

$$\vec{y}' = A\vec{y},\ A\vec{y}_i = \lambda_i \vec{y}_i \text{ (where } A = A^\top)$$
$$\vec{y}(0) = \sum_i c_i \vec{y}_i$$

Multidimensional Case

$$\vec{y}' = A\vec{y},\ \ A\vec{y}_i = \lambda_i \vec{y}_i \text{ (where } A = A^\top)$$

$$\vec{y}(0) = \sum_i c_i \vec{y}_i$$

$$\implies \vec{y}(t) = \sum_i c_i e^{\lambda_i t} \vec{y}_i$$

Multidimensional Case

$$\vec{y}' = A\vec{y}, \;\; A\vec{y}_i = \lambda_i \vec{y}_i \text{ (where } A = A^\top)$$

$$\vec{y}(0) = \sum_i c_i \vec{y}_i$$

$$\implies \vec{y}(t) = \sum_i c_i e^{\lambda_i t} \vec{y}_i$$

Stability depends on $\max_i \lambda_i$.

Integration Strategies

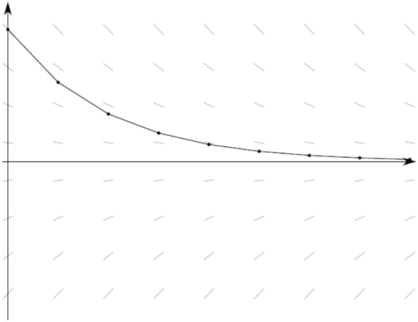
Given $\vec{y}_k$ at time t_k, generate $\vec{y}_{k+1}$ assuming $\vec{y}' = F[\vec{y}]$.

Forward Euler

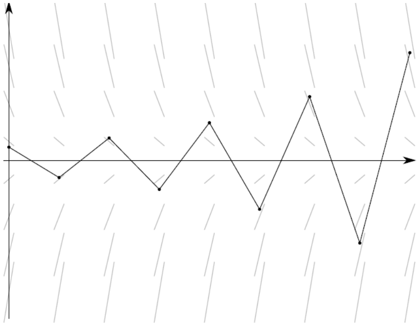
$$\vec{y}_{k+1} = \vec{y}_k + hF[\vec{y}_k]$$

- Explicit method
- $O(h^2)$ localized truncation error
- $O(h)$ global truncation error; "first order accurate"

Forward Euler: Stability



Stable $(a = -0.4)$



Unstable $(a = -2.3)$

Model Equation

$$y' = ay \longrightarrow y_{k+1} = (1 + ah)y_k$$

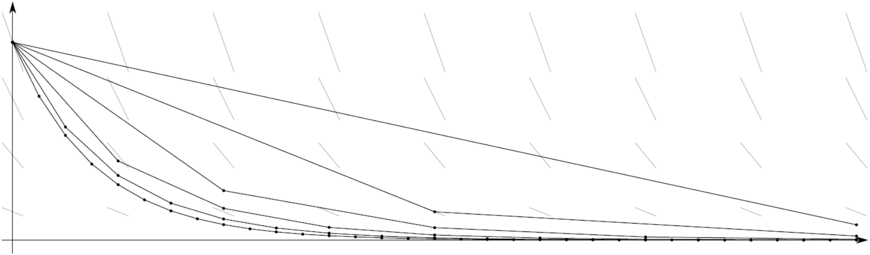
For $a < 0$, stable when $h < \frac{2}{|a|}$.

Backward Euler

$$\vec{y}_{k+1} = \vec{y}_k + hF[\vec{y}_{k+1}]$$

- Implicit method
- $O(h^2)$ localized truncation error
- $O(h)$ global truncation error; "first order accurate"

Backward Euler: Stability



Model Equation

$$y' = ay \longrightarrow y_{k+1} = \frac{1}{1 - ah} y_k$$

Unconditionally stable!

Model Equation

$$y' = ay \longrightarrow y_{k+1} = \frac{1}{1-ah} y_k$$

Unconditionally stable!

But this has nothing to do with accuracy.

Model Equation

$$y' = ay \longrightarrow y_{k+1} = \frac{1}{1-ah} y_k$$

Unconditionally stable!

But this has nothing to do with accuracy.

Good for *stiff* equations.

Numerical Stiffness for IVP ODEs

An IVP is said to be numerically "stiff" if stability requirements dictate a much smaller time step size than is needed to satisfy the approximation requirements alone.

[Ascher & Petzold 1998]

Forward and Backward Euler on Linear ODE

$$\vec{y}\,' = A\vec{y}$$

- Forward Euler: $\vec{y}_{k+1} = (I + hA)\vec{y}_k$
- Backward Euler: $\vec{y}_{k+1} = (I - hA)^{-1}\vec{y}_k$

Next