Lecture 12:

Transformations

Contents

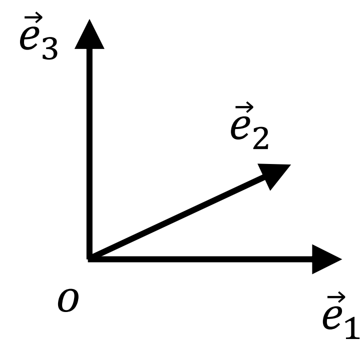
Vector Space

- The **vector space** V in 3D over the real numbers

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \in V^3 = \mathbb{R}^3$$

- Vectors in nD are written as $n \times 1$ matrices
- Vectors describe directions – **not positions**
  - All vectors conceptually start from the origin of the coordinate system
- 3 linear independent vectors create a basis

$$\{\vec{e}_1, \vec{e}_2, \vec{e}_3\} = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right\}$$

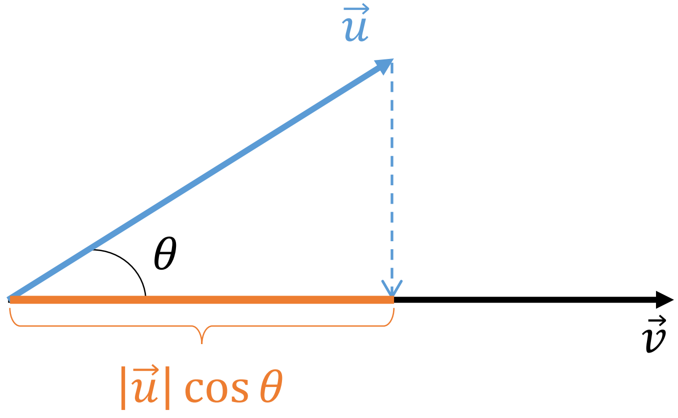


- Any 3D vector can be represented uniquely with coordinates

$$\vec{v} = v_1\vec{e}_1 + v_1\vec{e}_1 + v_1\vec{e}_1 \qquad v_1, v_2, v_3 \in \mathbb{R}$$

Vector Space - Metric

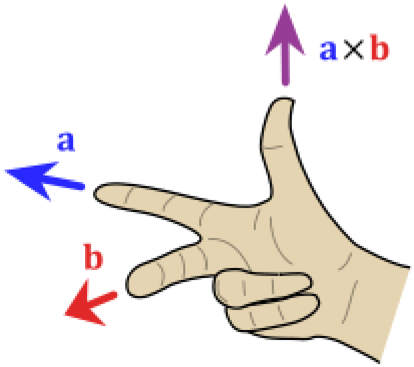
- Standard scalar product a.k.a. dot or inner product
  - Measure lengths
    $$|\vec{v}|^2 = \vec{v} \cdot \vec{v} = v_1^2 + v_2^2 + v_3^2$$
  - Compute angles
    $$\vec{v} \cdot \vec{u} = |\vec{v}||\vec{u}| \cos(u, v)$$
  - Projection of vectors onto other vectors
    $$|\vec{u}| \cos \theta = \frac{\vec{v} \cdot \vec{u}}{|\vec{v}|} = \frac{\vec{v} \cdot \vec{u}}{\sqrt{\vec{v} \cdot \vec{v}}}$$



Orthonormal basis

- Unit length vectors
  - $|\vec{e}_1| = |\vec{e}_2| = |\vec{e}_3| = 1$
- Orthogonal to each other
  - $\vec{e}_i \cdot \vec{e}_j = \delta_{ij}$

Handedness of the coordinate system



- Two options: $\vec{e}_1 \times \vec{e}_2 = \vec{e}_3$
  - Positive: Right-handed (RHS)
  - Negative: Left-handed (LHS)
- Example: Screen Space
  - Typical: X goes right, Y goes up (thumb & index finger, respectively)
  - In a RHS: Z goes out of the screen (middle finger)
- Be careful:
  - Most systems nowadays (including OpenRT) use a right handed coordinate system
  - But some are not (*e.g.* RenderMan) → can cause lots of confusion

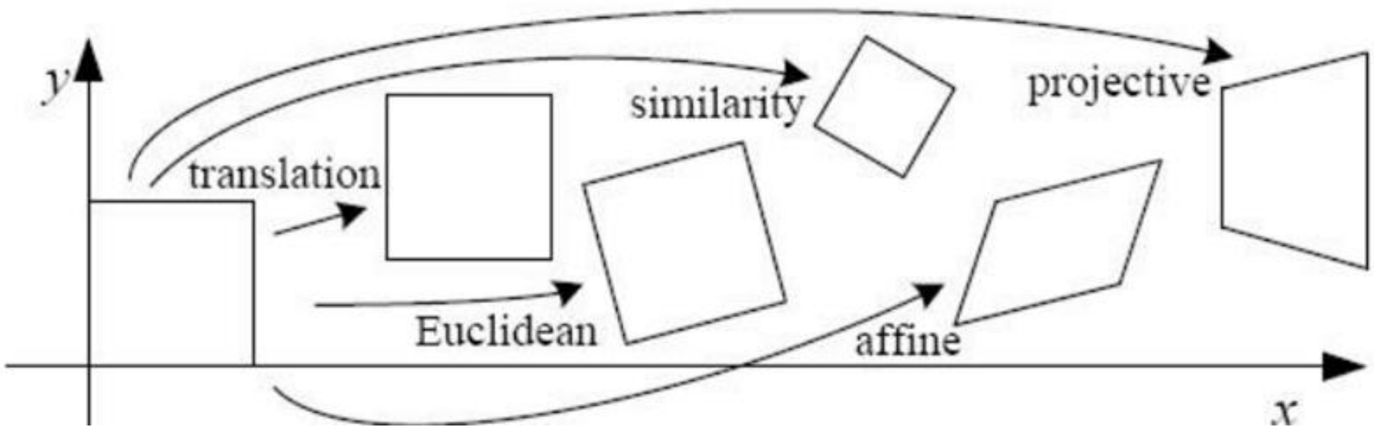
Transformation

- Matrix multiplication can be used to transform vectors through multiplication: $x' = Ax$
- A matrix used in this way is called a transformation matrix
  - Simplest is scaling:

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

Transformations

- Transformations can be divided into many classes

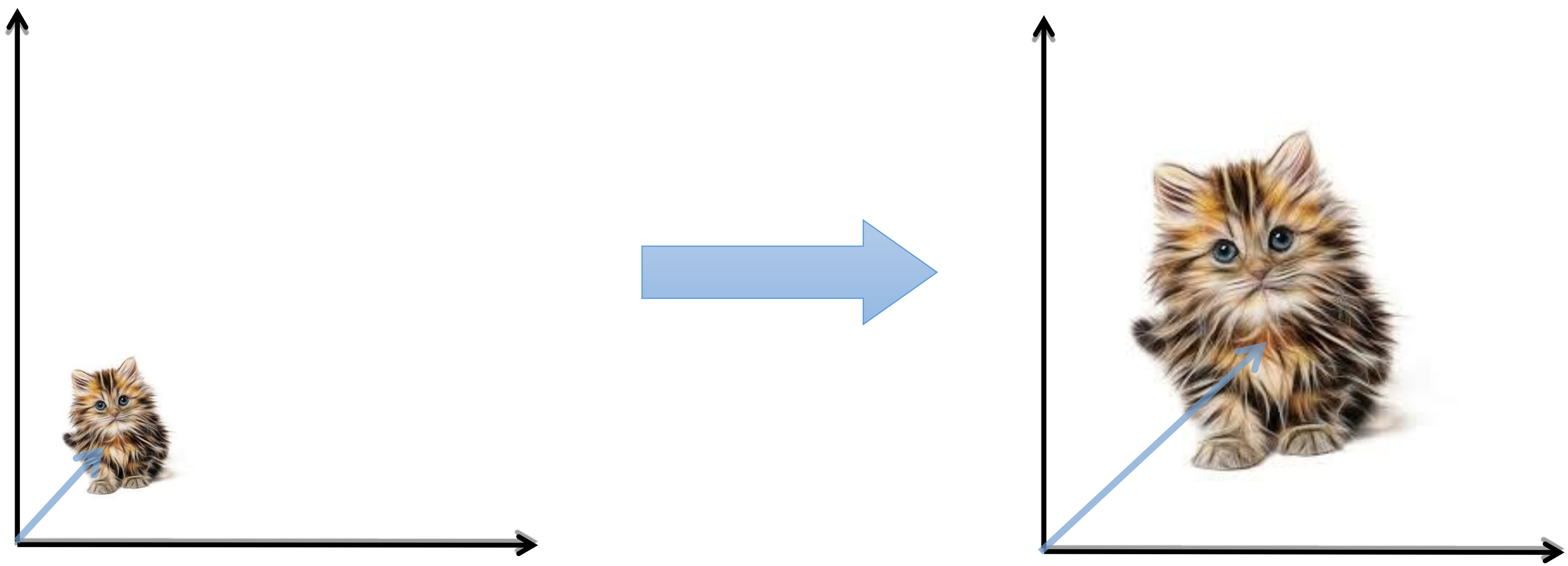


- **Rigid / Euclidian transformation:** preserves the Euclidean distance between every pair of points. The rigid transformations include *rotations*, *translations*, *reflections*, or their combination
- **Similarity transformation:** is an angle-preserving transformation whose transformation matrix A' can be written in the form $A' = B\ A\ B^{-1}$
- **Affine transformation:** is any transformation that preserves collinearity and ratios of distances. It can be a composition of two functions: a translation and a linear mapping
- **Projective transformation:** maps lines to lines (but does not necessarily preserve parallelism). Any plane projective transformation can be expressed by an invertible matrix in homogeneous coordinates (to be defined...)

Scaling (S)

rt::CTransform::scale()

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



Note that this operation happens w.r.t. to the origin of the coordinate system – the kitten has been "stretched" in x- and y-axis but also the position of its center has moved in the same proportion

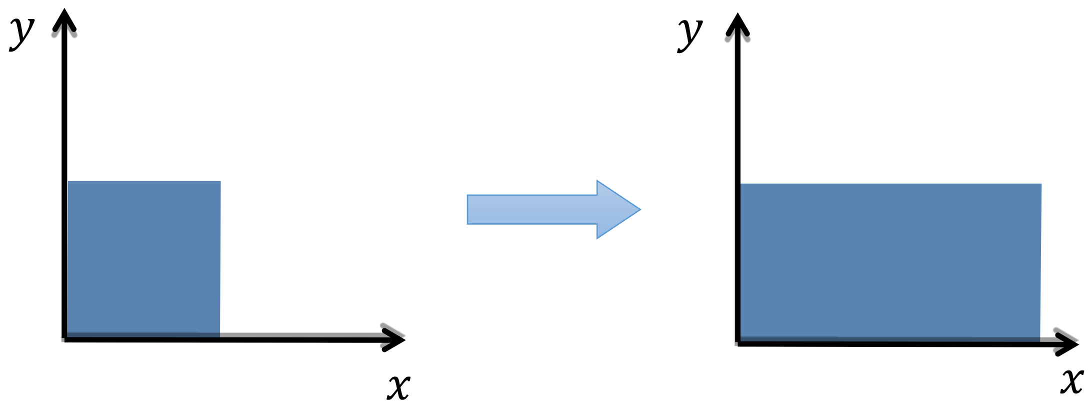
Scaling (S)

rt::CTransform::scale()

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- If s_x, s_y and s_z are equal, we talk about *uniform* scaling s: $s = s_x, s_y, s_z$
  - It can be represented by a simple scalar multiplication of the vector
- If the contrary is true, then it's non-uniform scaling
- Note: $s_x, s_y, s_z \geq 0$ (otherwise see mirror transformation)

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix}$$



Multiple points to transform

- Rather than transform individual points, all operations can be done in one step:

$$\begin{pmatrix} x'_1 \\ y'_1 \end{pmatrix} = T \times \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$\begin{pmatrix} x'_2 \\ y'_2 \end{pmatrix} = T \times \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$$

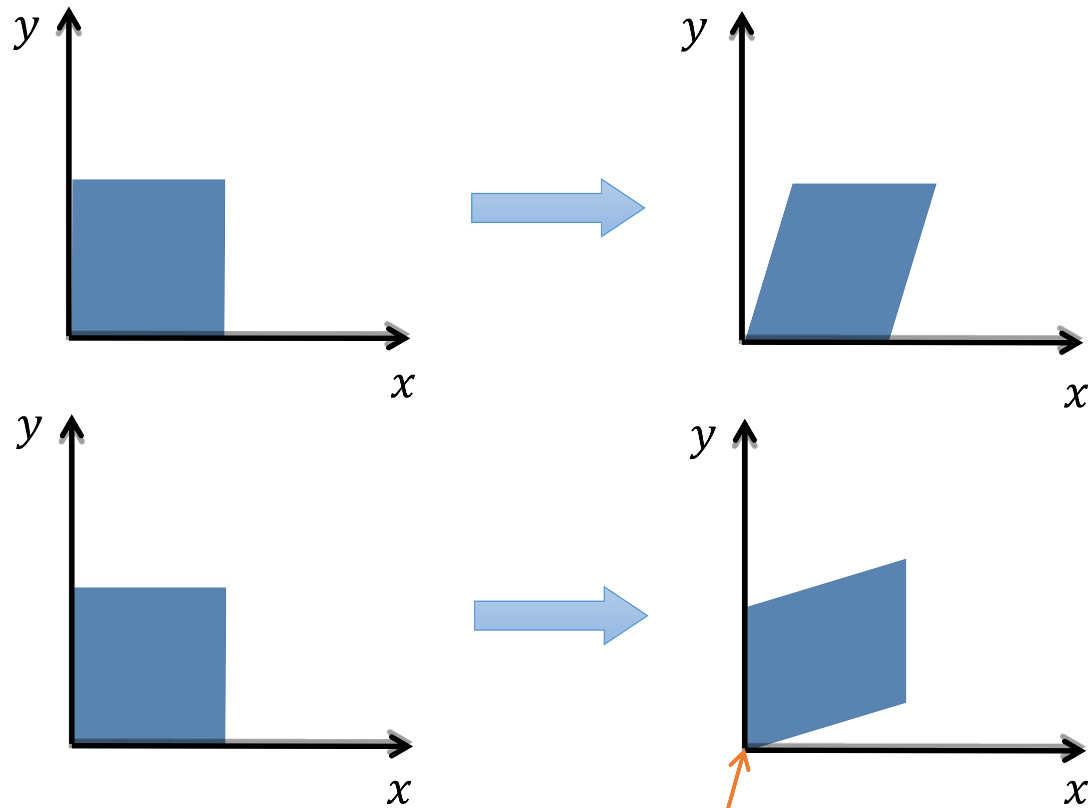
$$\begin{bmatrix} x'_1 & x'_2 & \cdots \\ y'_1 & y'_2 & \dots \end{bmatrix} = T \times \begin{bmatrix} x_1 & x_2 & \cdots \\ y_1 & y_2 & \dots \end{bmatrix}$$

Shear (H)

- Shear transformation **H** acts along the axes of the coordinate system:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 1 & h_x \\ 0 & 1 \end{bmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ h_y & 1 \end{bmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix}$$



General form: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & h_{xy} & h_{xz} \\ h_{yx} & 1 & h_{yz} \\ h_{zx} & h_{zy} & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Why did this point not move?

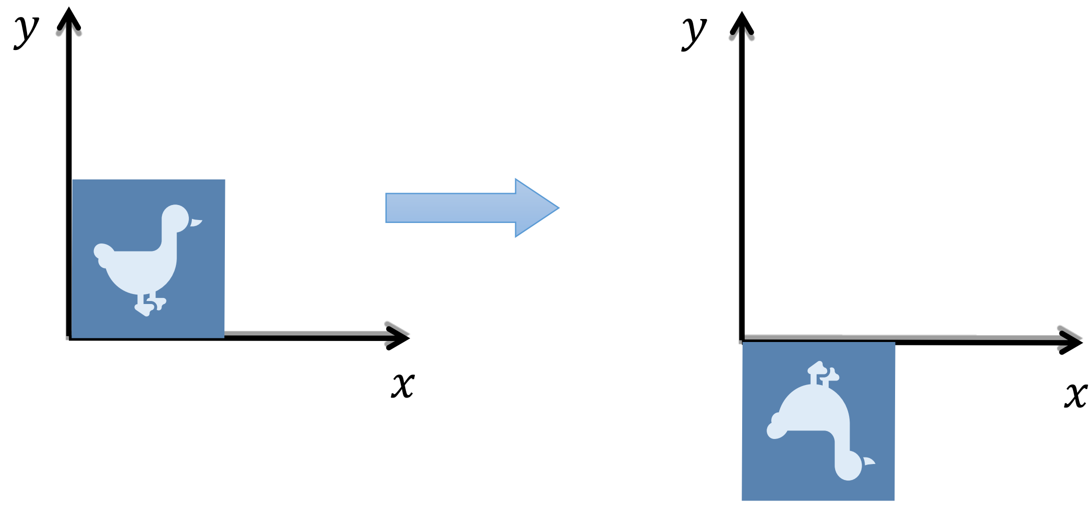
Answer: because it's an invariant point of this transformation!

Reflection / Mirror (M)

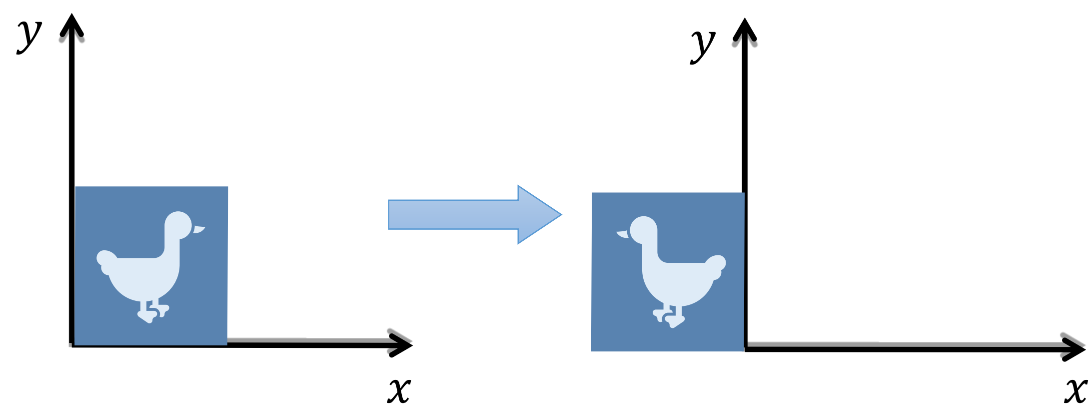
[rt::CTransform::mirror()]

- Mirror transformation **M** with respect to one of the axes
- Note: changes orientation

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



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

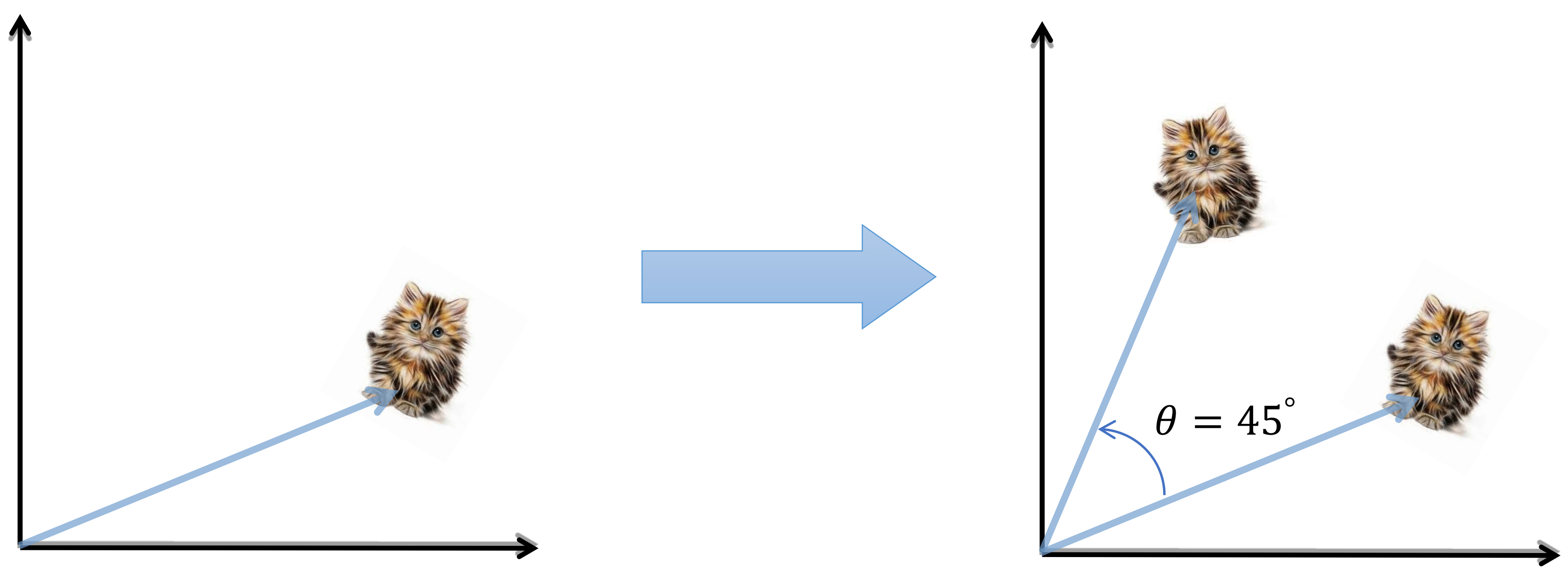


Can you guess the formula for mirroring w.r.t. x and y?

Rotation (R)

rt::CTransform::rotate()

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ y\cos\theta + x\sin\theta \end{pmatrix}$$



Note that this operation happens w.r.t. to the origin of the coordinate system – the kitten has been rotated around it's center but also its center moved with respect to the origin of the coordinate frame

Rotation around origin in 2D

- Representation in polar (or spherical for 3D) coordinates:
  - $x = r\cos\alpha$
  - $y = r\sin\alpha$

$$x' = r\cos(\alpha + \theta)$$

$$y' = r\sin(\alpha + \theta)$$

- Well know property
  - $\cos(\alpha + \theta) = \cos\alpha\cos\theta - \sin\alpha\sin\theta$
  - $sin(\alpha + \theta) = \cos\alpha\sin\theta - \sin\alpha\cos\theta$

- Gives
  - $x' = (r\cos\alpha)\cos\theta - (r\sin\alpha)\sin\theta = x\cos\theta - y\sin\theta$
  - $y' = (r\cos\alpha)\sin\theta - (r\sin\alpha)\cos\theta = x\sin\theta + y\cos\theta$

- Or in matrix form

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ y\cos\theta + x\sin\theta \end{pmatrix}$$

Rotation around major axes in 3D

- Rotation around z axis: $R_\theta^z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- Rotation around y axis: $R_\theta^y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$

- Rotation around x axis: $R_\theta^x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$

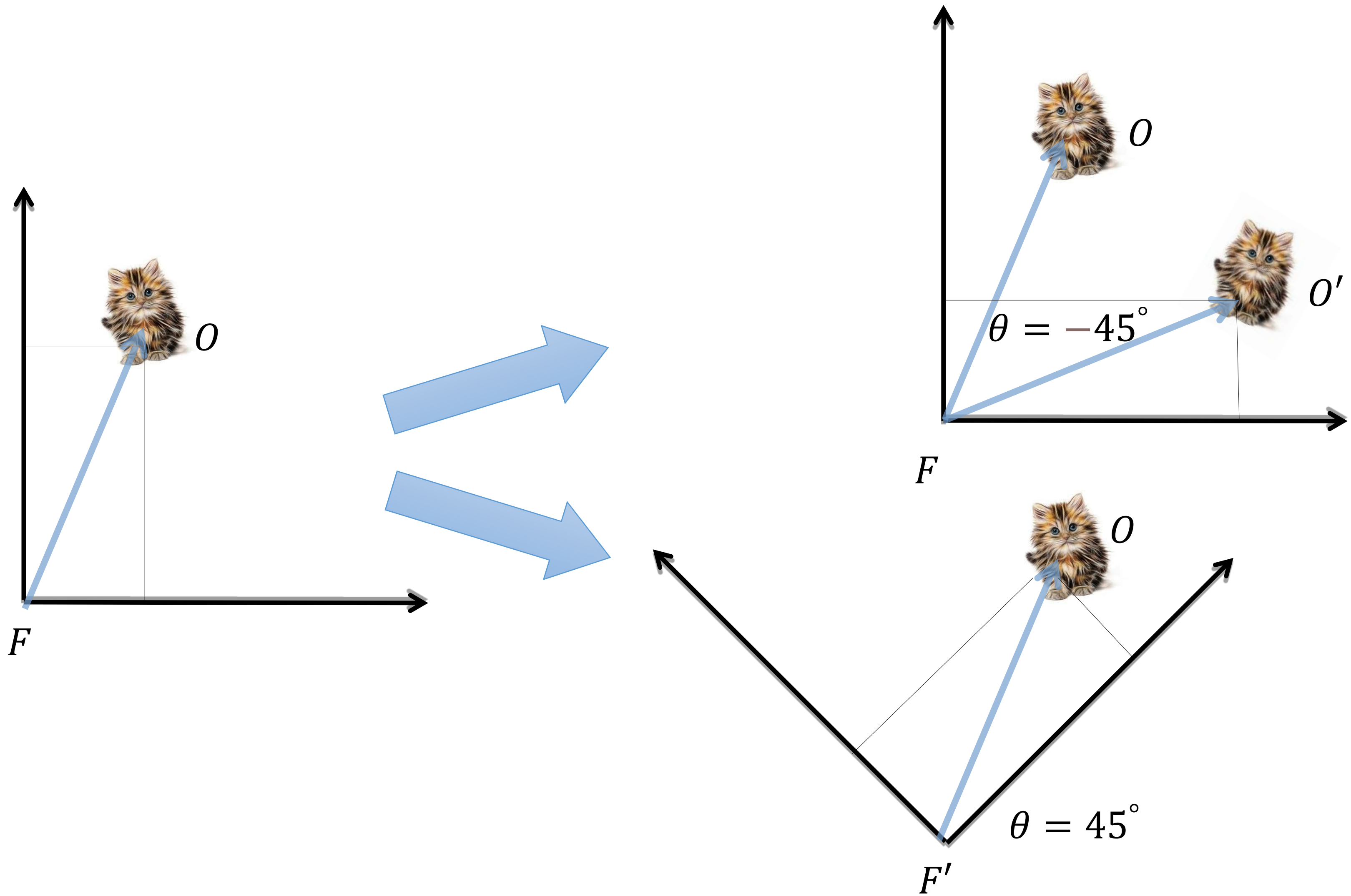
- 2D rotation around the respective axis
  - Assumes right-handed system, mathematically positive direction
- Be aware of change in sign on sines in R_θ^y
  - Due to relative orientation of other axis

Properties of Rotation Transform

- $R_0 = I$
- $\det(R)$ is always 1 (volume preservation)
- R is always invertible
  - Also, $R^{-1} = R^{\top}$
- Transpose of a rotation matrix produces a rotation in the opposite direction
  - $R_\theta^{-1} = R_{-\theta} = R_\theta^{\top}$
- Columns and rows of a rotation matrix are always orthogonal (they constitute the rotated coordinate axis!)
- Rotations around the same axis are commutative:
  - $R_\theta \times R_\alpha = R_{\theta+\alpha} = R_\alpha \times R_\theta$
- Rotations around different axes are <u>not</u> commutative
  - $R_\theta^x \times R_\alpha^y \neq R_\alpha^y \times R_\theta^x$
  - Order does matter for rotations around different axes

Vector vs coordinate frame transformation

- In general: transformation of a vector is equivalent to mathematical transformation of its coordinates
- Useful insight: rotating a vector is equivalent to rotating the coordinate frame in the opposite sense



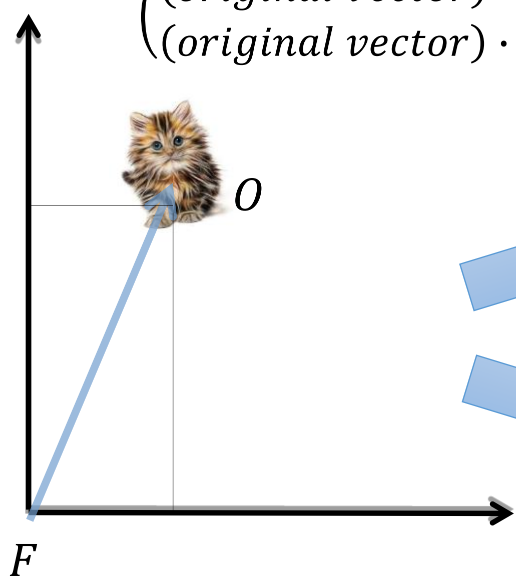
Vector vs coordinate frame transformation

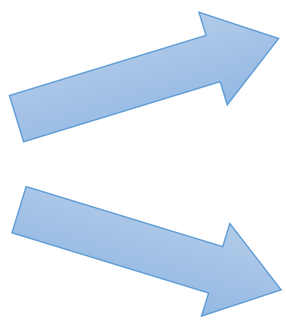
- The reason it works:
  - Original vector:

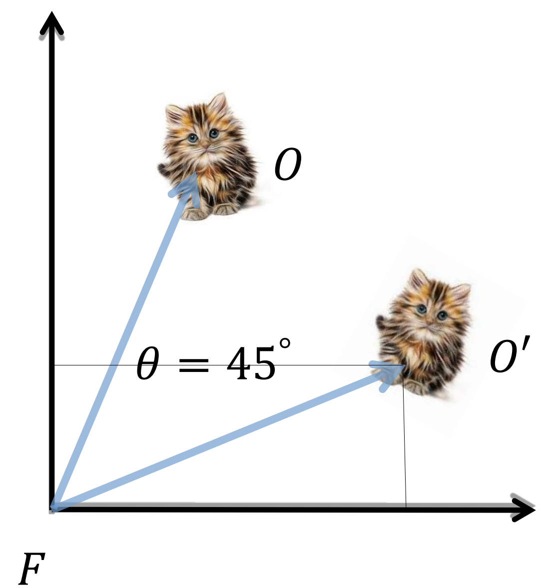
$$\begin{pmatrix}\text{component in direction of the frame's x axis}\\ \text{component in direction of y axis}\end{pmatrix}$$

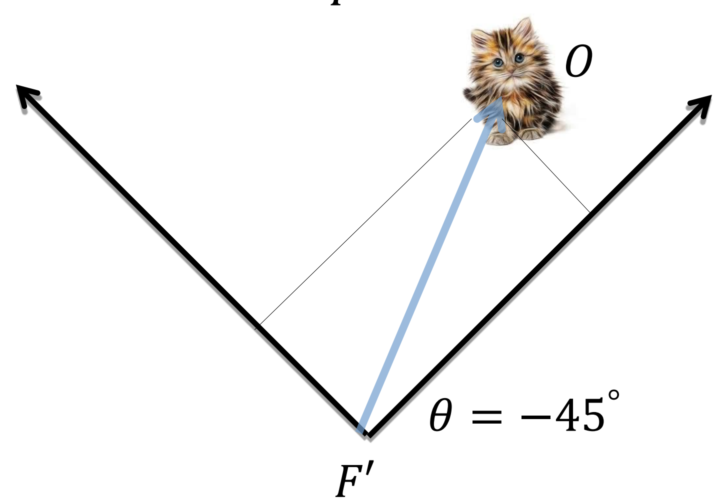
  - New vector:

$$\begin{pmatrix}(original\ vector)\cdot(the\ new\ x\ axis)\\ (original\ vector)\cdot(the\ new\ y\ axis)\end{pmatrix}$$









Combining transformations

- Multiple transformation matrices can be used to transform a point:

$$\vec{x}' = M \times T \times S \times \vec{x}$$

- The effect of this is to apply their transformations one after the other, from right to left. In the example above, the result is:

$$\vec{x}' = \big(M \times \big(T \times (S \times \vec{x})\big)\big)$$

- The result is exactly the same if we multiply the matrices first, to form a single transformation matrix:

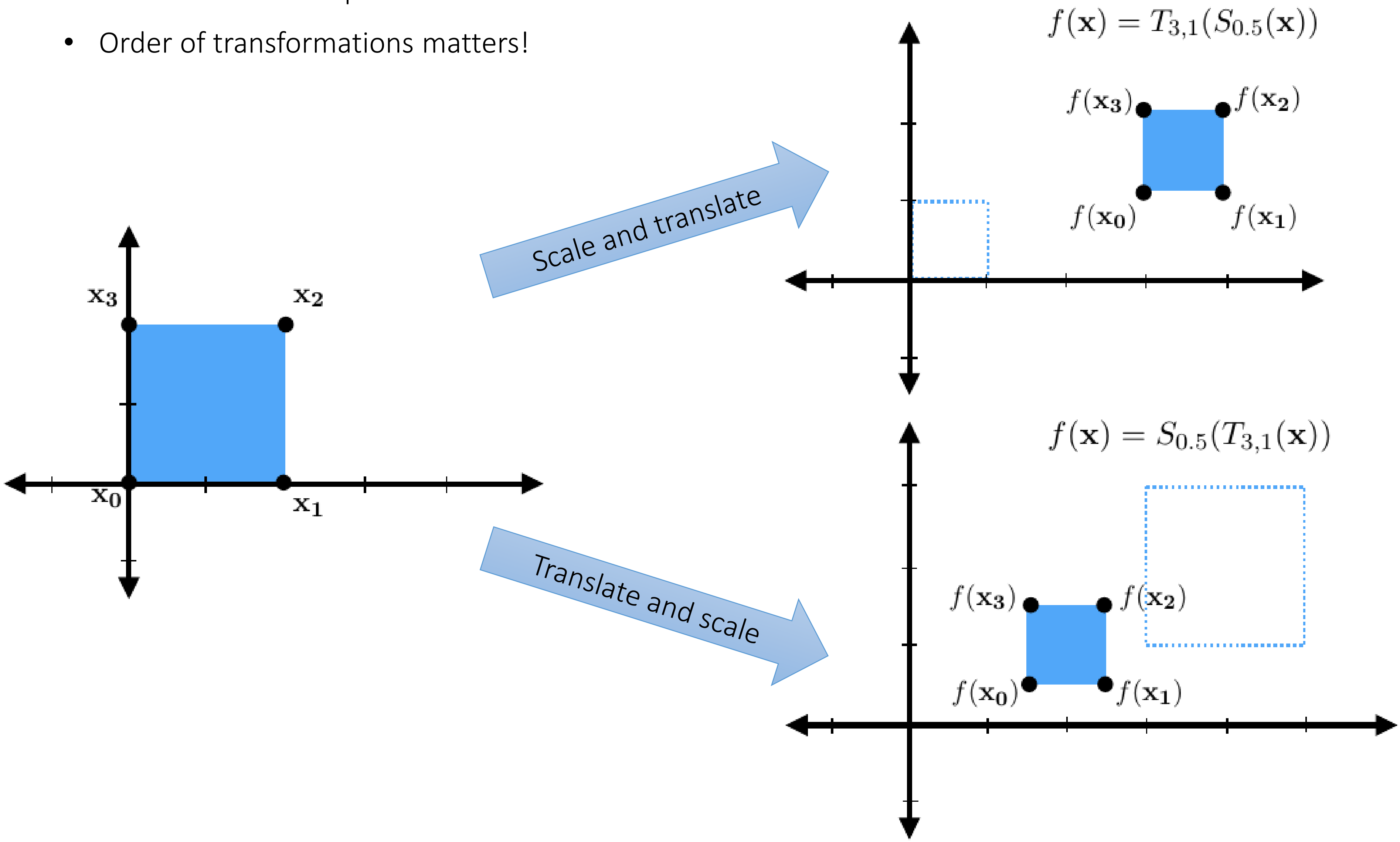
$$\vec{x}' = (M \times T \times S) \times \vec{x}$$

- In general all the transformations may be written as a single transformation matrix:

$$\vec{x}' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{pmatrix} y \\ x \\ z \end{pmatrix}$$

Multiply matrices to concatenate

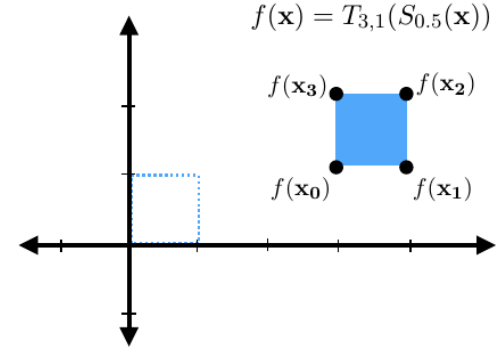
- Matrix-matrix multiplication is not commutative
- Order of transformations matters!



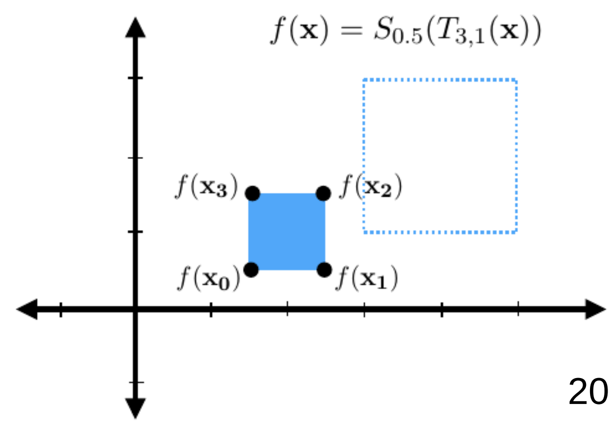
Fluent Interface

- OpenRT CTransform class support fluent interface
- Every method of the class return an instance of that class

```
CTransform transform;
Mat t = transform.scale(0.5f).translate(Vec3f(3, 1, 0)).get();
solidQuad.transform(t);
```




```
CTransform transform;
Mat t = transform.translate(Vec3f(3, 1, 0)).scale(0.5f).get();
solidQuad.transform(t);
```



Linear transformations are combinations of ...

- Scale
- Rotation
- Shear
- Mirror

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{pmatrix} y \\ x \\ z \end{pmatrix}$$

Properties of linear transformations:

- Origin maps to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition

... but why have we never mentioned translation?

Dealing with Translation

- Translation is a conceptually very simple operation: just add a vector to the vector representing a point
- Translation is <u>not linear</u> and thus does not have a 2×2 matrix representation
- A modified way of expressing coordinates will help us with the this problem:
  - let us add an extra field w with a constant value of 1 to our point representation:

$$\begin{pmatrix} x \\ y \end{pmatrix}^T \Rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}^T$$

- We have to adjust our transform matrices to deal with it:

$$S_s = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \Rightarrow S_s = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

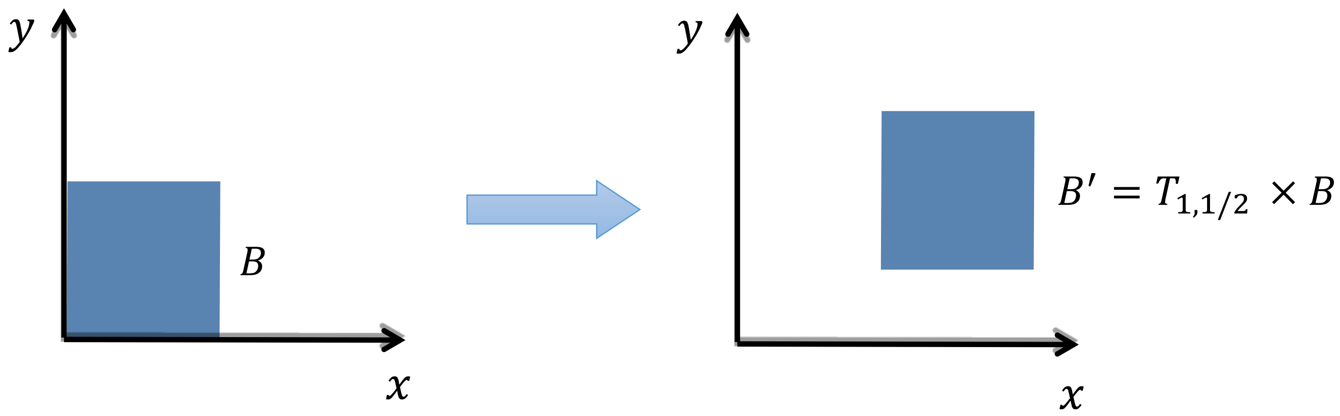
$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \Rightarrow R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation (T)

rt::CTransform::translate()

- Thanks to this, we can introduce a multiplicative translation transform:

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



Properties

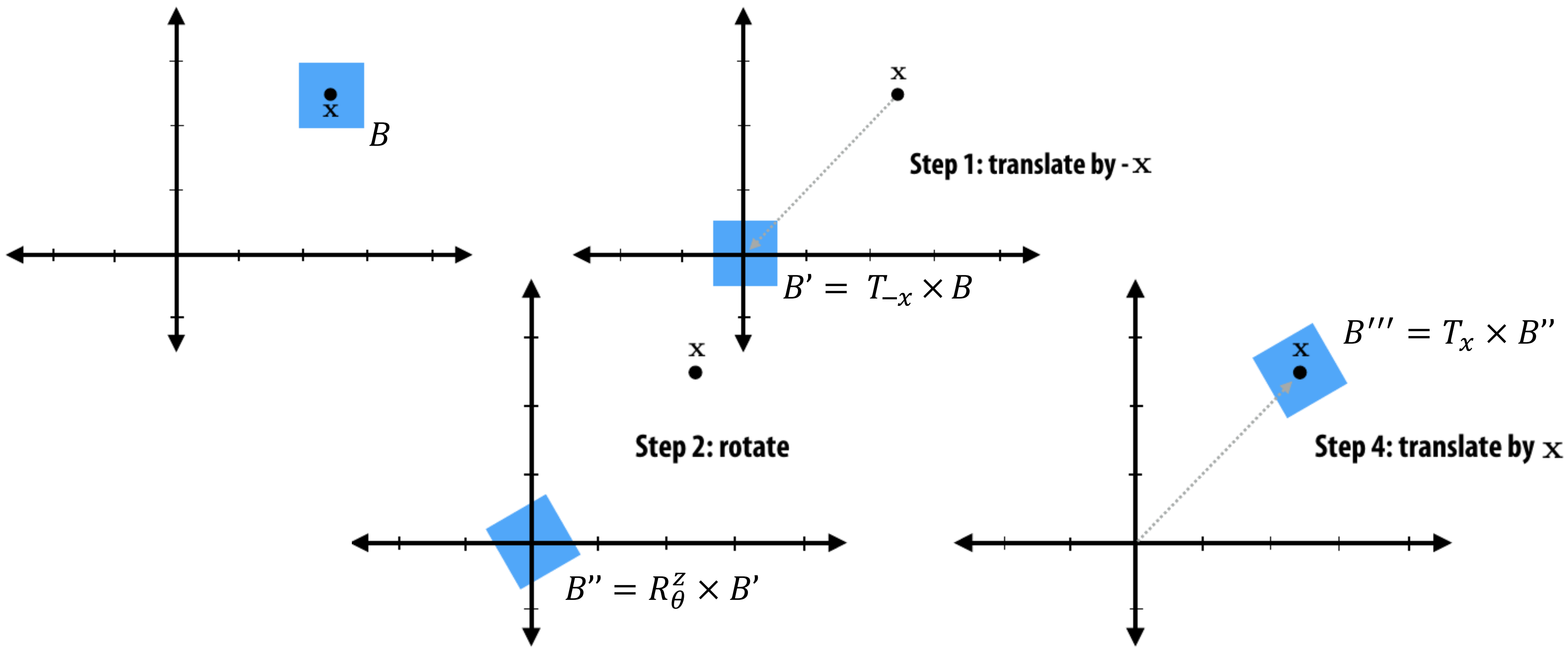
- Identity
  - $T_0 = I$
- Commutative (special case)
  - $T_t \times T_{t'} = T_{t+t'} = T_{t'} \times T_t$
- Inverse
  - $T_t^{-1} = T_{-t}$

Rotation and Translation

- Now we can include affine transforms into our calculations! For example, to move the element to the center, rotate it and move it back to its original point:
  - $T_t \times R_\theta \times T_{-t}$

Rotate object around a point x and axis z

- Point x is called *pivot point*
- $R^{z}_{x,\theta} = T_x \times R^{z}_{\theta} \times T_{-x}$ operation allows to rotate an shape around some point
- Rotation, as expressed in the initial form, is executed around $(0, 0)$



Rotate around a given point p and vector r $(|r| = 1)$

- Translate so that p is in the origin
- Transform with rotation $R = M^\intercal$
    - M given by orthonormal basis (k, s, t) such that k becomes the x axis
    - Requires construction of a orthonormal basis (k, s, t)
- Rotate around x axis
- Transform back with M^{-1}
- Translate back to point p

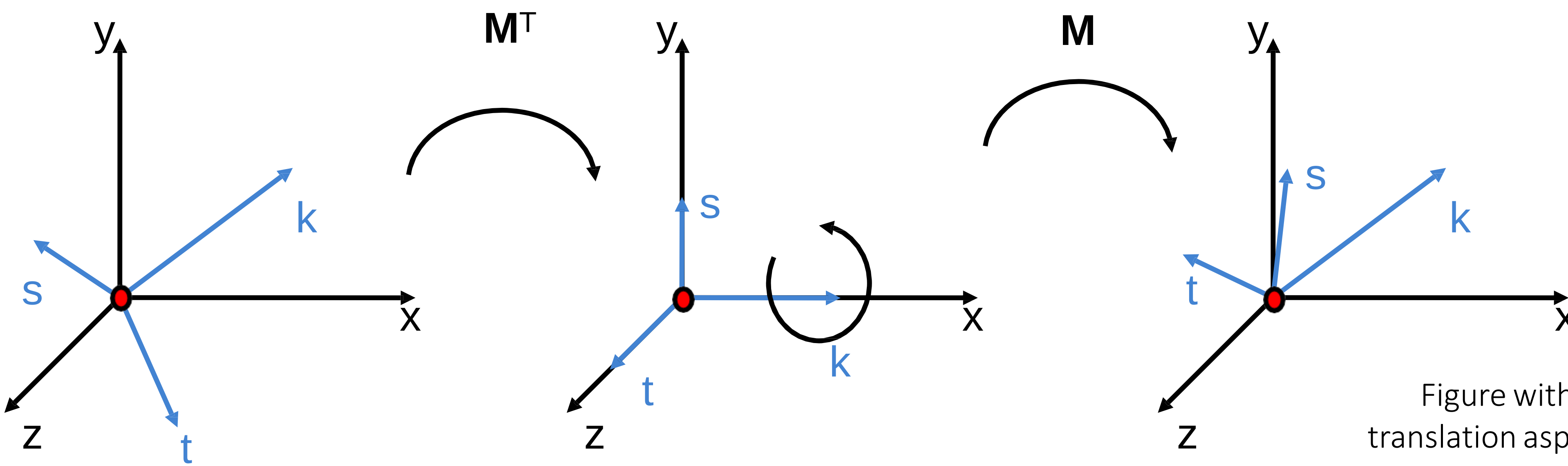


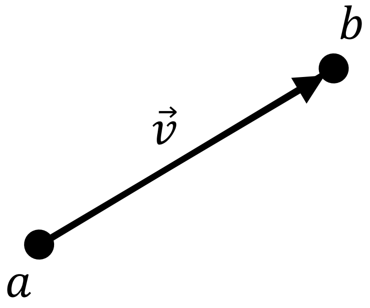
Figure without translation aspect

$$R(p, r, \theta) = T_p \times M(k) \times R_\theta^x \times M^\intercal\,(k) \times T_{-p}$$

```
// k – rotation axis; theta – angle of rotation in degrees
CTransform CTransform::rotate(const Vec3f& k, float theta) const
{
    Mat t = Mat::eye(3, 3, CV_32FC1);
    theta *= Pif/180;
    float cos_theta = cosf(theta);
    float sin_theta = sinf(theta);
    float x = k[0];
    float y = k[1];
    float z = k[2];

    t[0, 1] = (1 - cos_theta) * x * y - sin_theta * z;
    t[0, 0] = cos_theta + (1 - cos_theta) * x * x;
    t[0, 2] = (1 - cos_theta) * x * z + sin_theta * y;

    t[1, 0] = (1 - cos_theta) * y * x + sin_theta * z;
    t[1, 1] = cos_theta + (1 - cos_theta) * y * y;
    t[1, 2] = (1 - cos_theta) * y * z - sin_theta * x;

    t[2, 0] = (1 - cos_theta) * z * x - sin_theta * y;
    t[2, 1] = (1 - cos_theta) * z * y + sin_theta * x;
    t[2, 2] = cos_theta + (1 - cos_theta) * z * z;

    return CTransform(t * m_t);
}
```

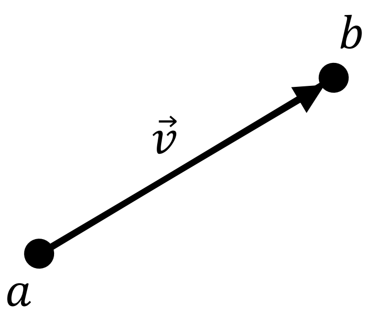
Basic mathematical concepts

- The **affine space** A
  - In contrast to vector space, affine space operates with objects of 2 types:
    - **Vectors:** represent *directions*: they always have $w = 0$
    - **Points:** represent *locations*
- Defined via its associated vector space V
  - $a, b \in A \Leftrightarrow \exists \vec{v} \in V : \vec{v} = b - a$
- Operations on affine space A
  - Subtraction of two points yields a vector
  - No addition of points (it is not clear what the some of two points would mean)
  - But: Addition of points and vectors:
    - $a + \vec{v} = b \in A^3$
  - Distance
    - $dist(a, b) = |a - b|$



Basic mathematical concepts

- The **affine space** A
  - In contrast to vector space, affine space operates with objects of 2 types:
    - **Vectors:** represent *directions*: they always have $w = 0$
    - **Points:** represent *locations*
- Difference between 2 points:
  - $\vec{v} = b - a = \begin{pmatrix} b_x \\ b_y \\ 1 \end{pmatrix} - \begin{pmatrix} a_x \\ a_y \\ 1 \end{pmatrix} = \begin{pmatrix} b_x - a_x \\ b_y - a_y \\ 0 \end{pmatrix}$



- Consequently: Translations do not affect vectors!
  - $T_t \times \vec{v} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{pmatrix} v_x \\ v_y \\ 0 \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ 0 \end{pmatrix} = \vec{v}$

Homogeneous Coordinates for 3D

- Homogeneous embedding of $\mathbb{R}^3$ into the affine 4D space $A(\mathbb{R}^4)$
  - Mapping a point into homogeneous space
    - $\mathbb{R}^3 \ni \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \in A(\mathbb{R}^4)$
  - Mapping back by dividing through fourth component
    - $A(\mathbb{R}^4) \ni \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \rightarrow \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix} \in \mathbb{R}^3$

Consequence

- This allows to represent affine transformations as 4x4 matrices
- Mathematical trick
  - Convenient representation to express rotations and translations as matrix multiplications