03 - Basic Linear Algebra and 2D Transformations

NYU | COURANT

Overview

In this box, you will find references to Eigen

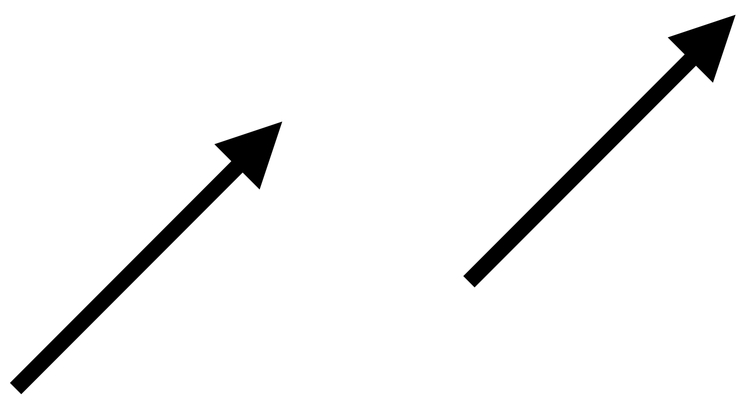
- We will briefly overview the basic linear algebra concepts that we will need in the class
- You will not be able to follow the next lectures without a clear understanding of this material

Vectors

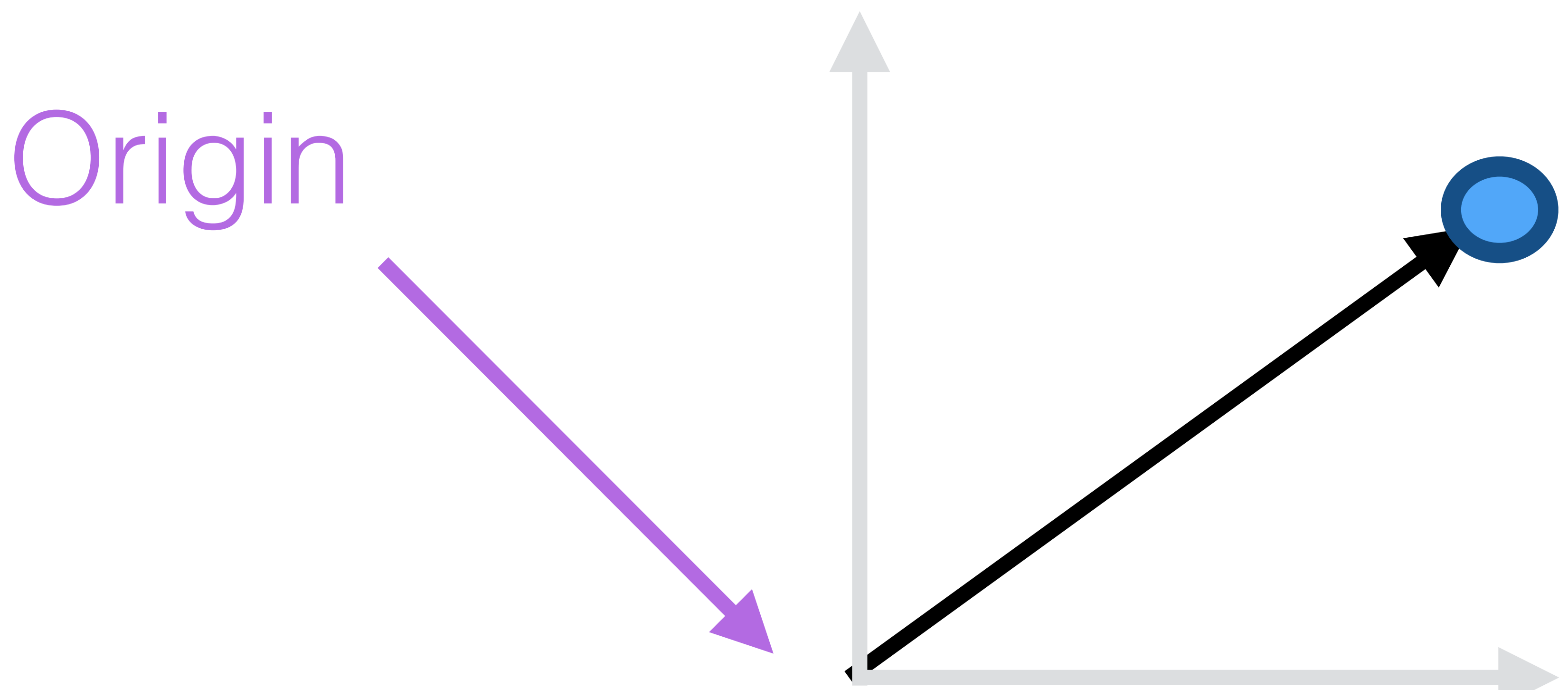
Vectors

Eigen::VectorXd

- A *vector* describes a direction and a length
- Do not confuse it with a location, which represent a position
- When you encode them in your program, they will both require 2 (or 3) numbers to be represented, but they are not the same object!



These two are identical!

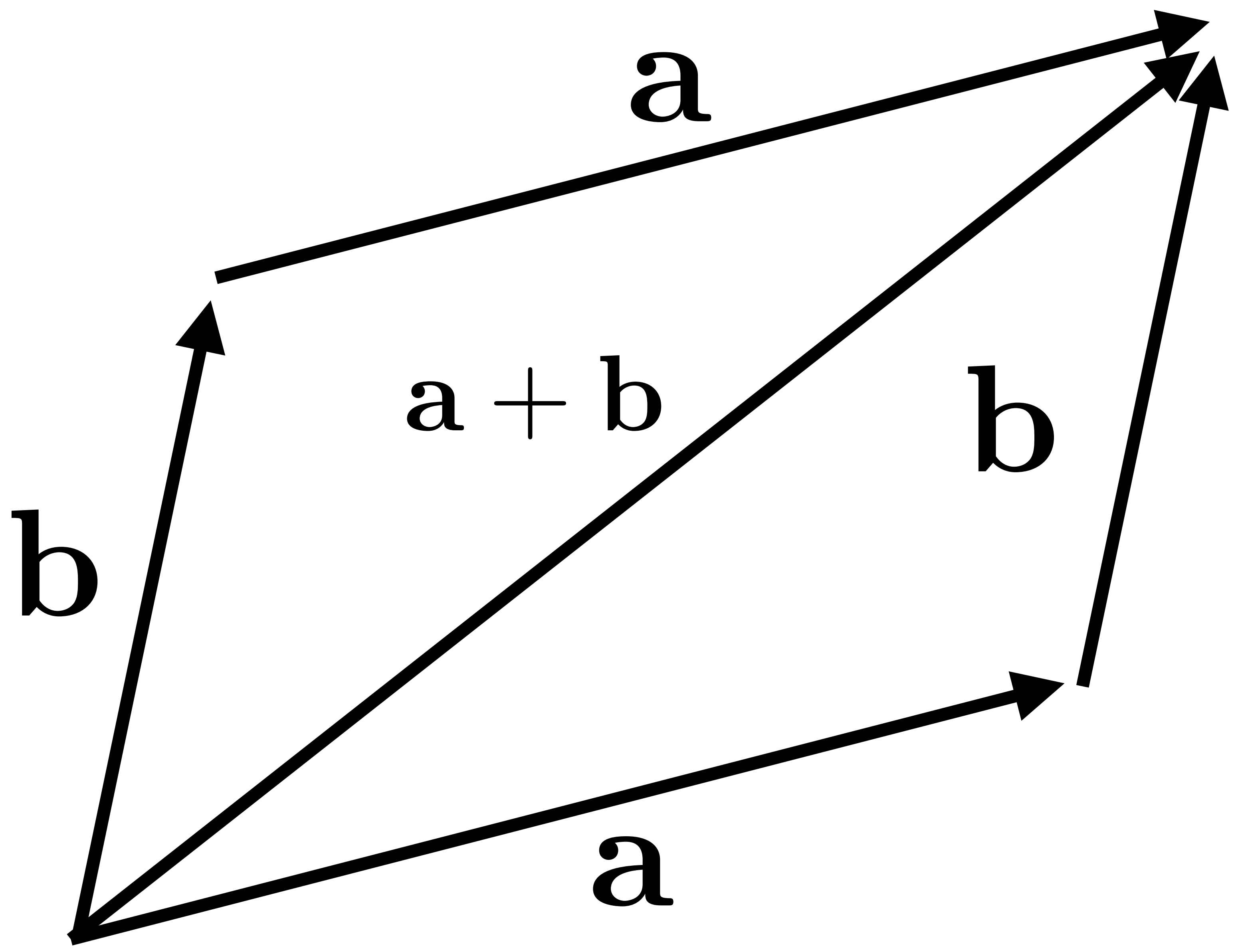


Vectors represent displacements. If you represent the displacement wrt the origin, then they *encode* a location.

Sum

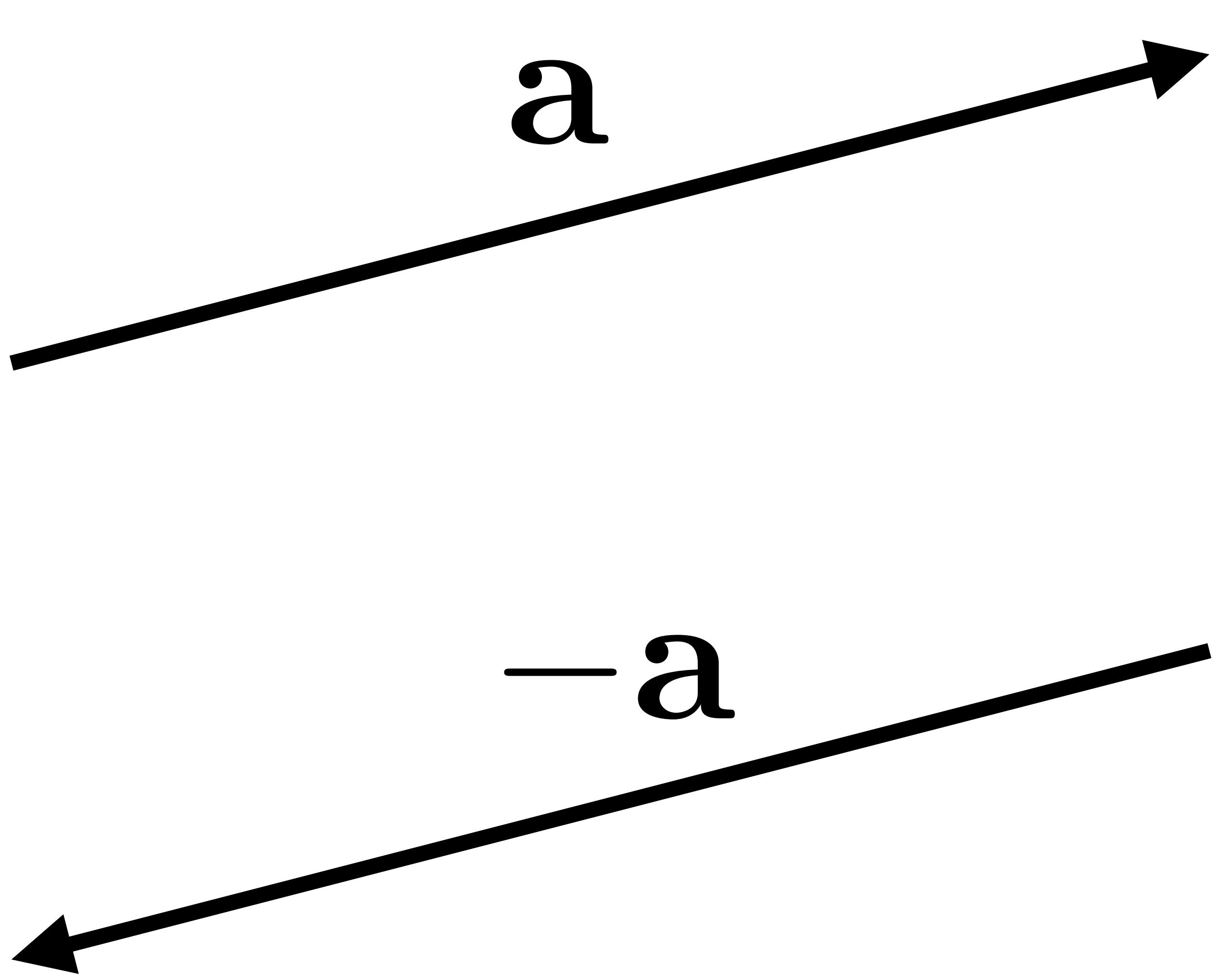
Operator +

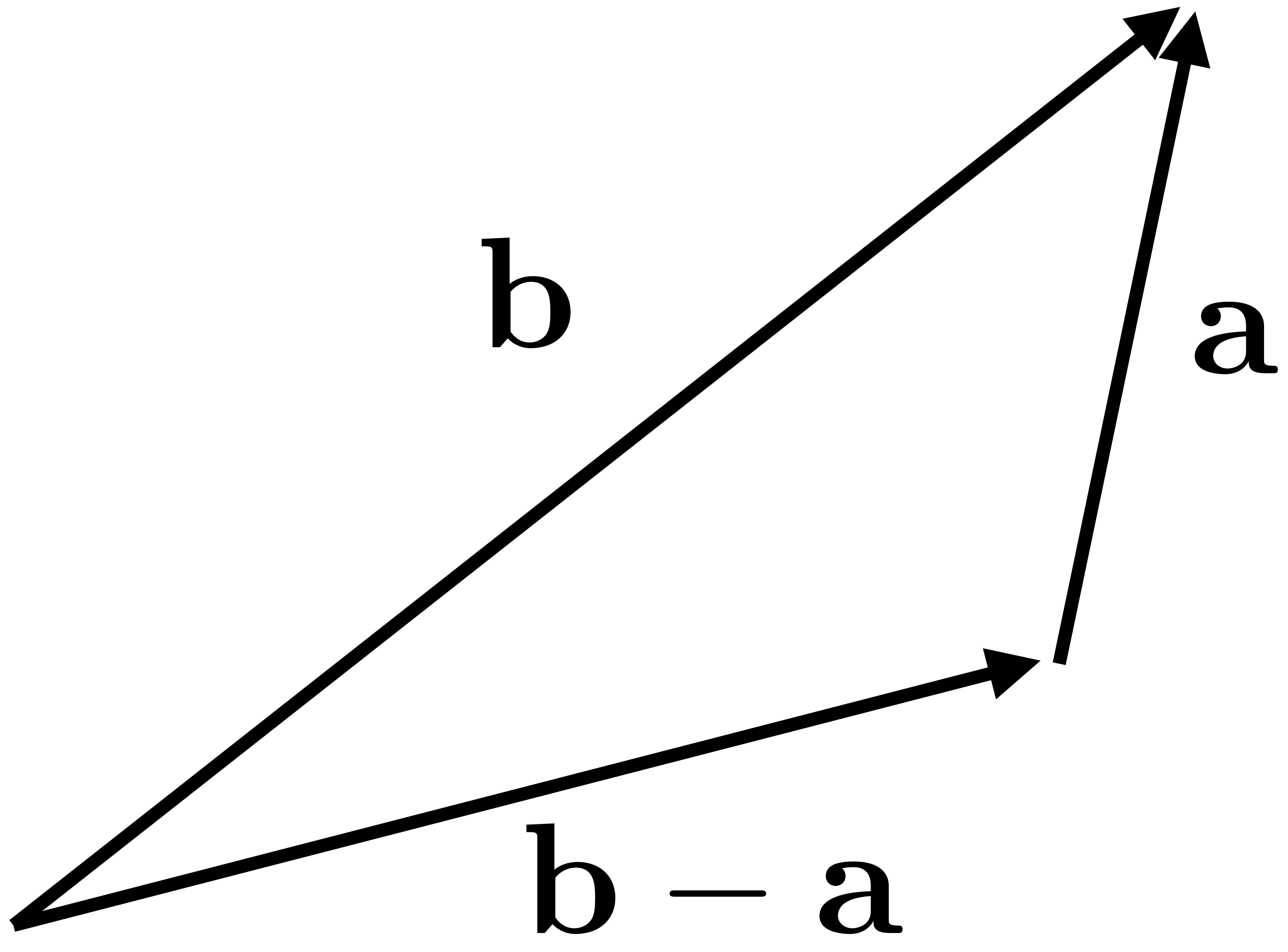
$$\mathbf{a} + \mathbf{b} = \mathbf{b} + \mathbf{a}$$



Difference

Operator -





$$\mathbf{b}-\mathbf{a}=-\mathbf{a}+\mathbf{b}$$

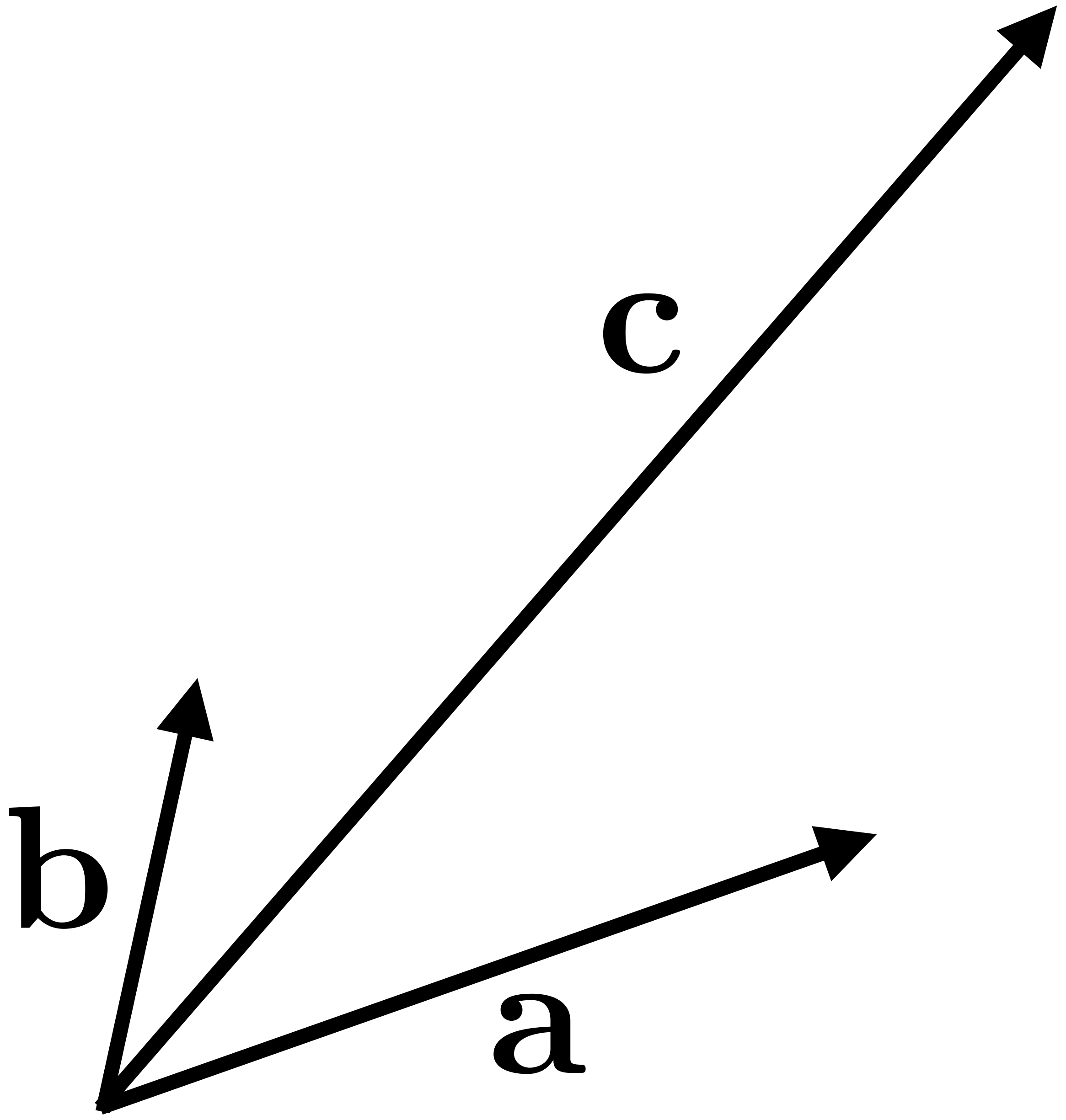
NYU | COURANT

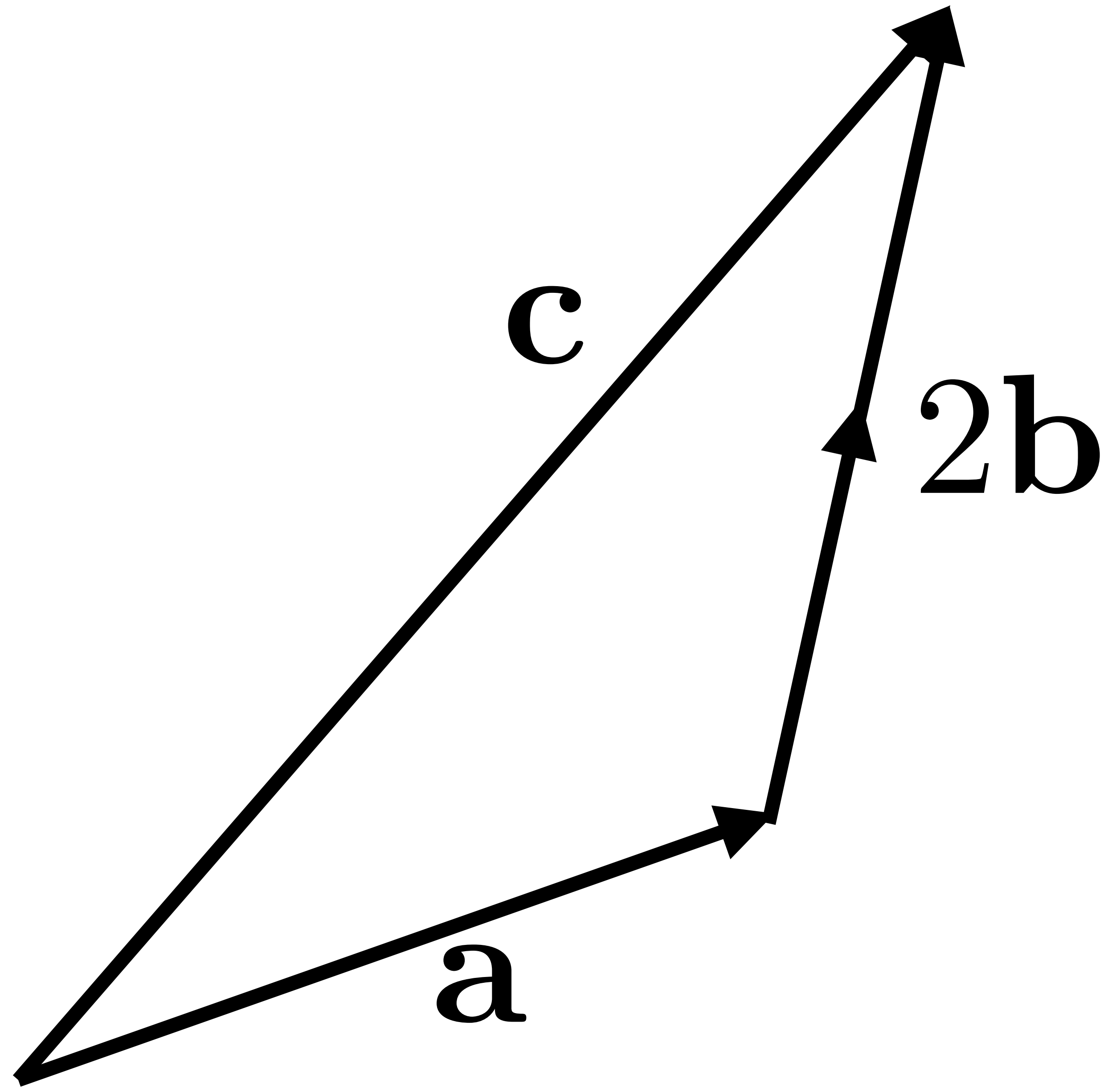
Coordinates

$\boxed{\text{Operator [ ]}}$

$$\mathbf{c} = c_1\mathbf{a} + c_2\mathbf{b}$$

$$\mathbf{c} = \mathbf{a} + 2\mathbf{b}$$

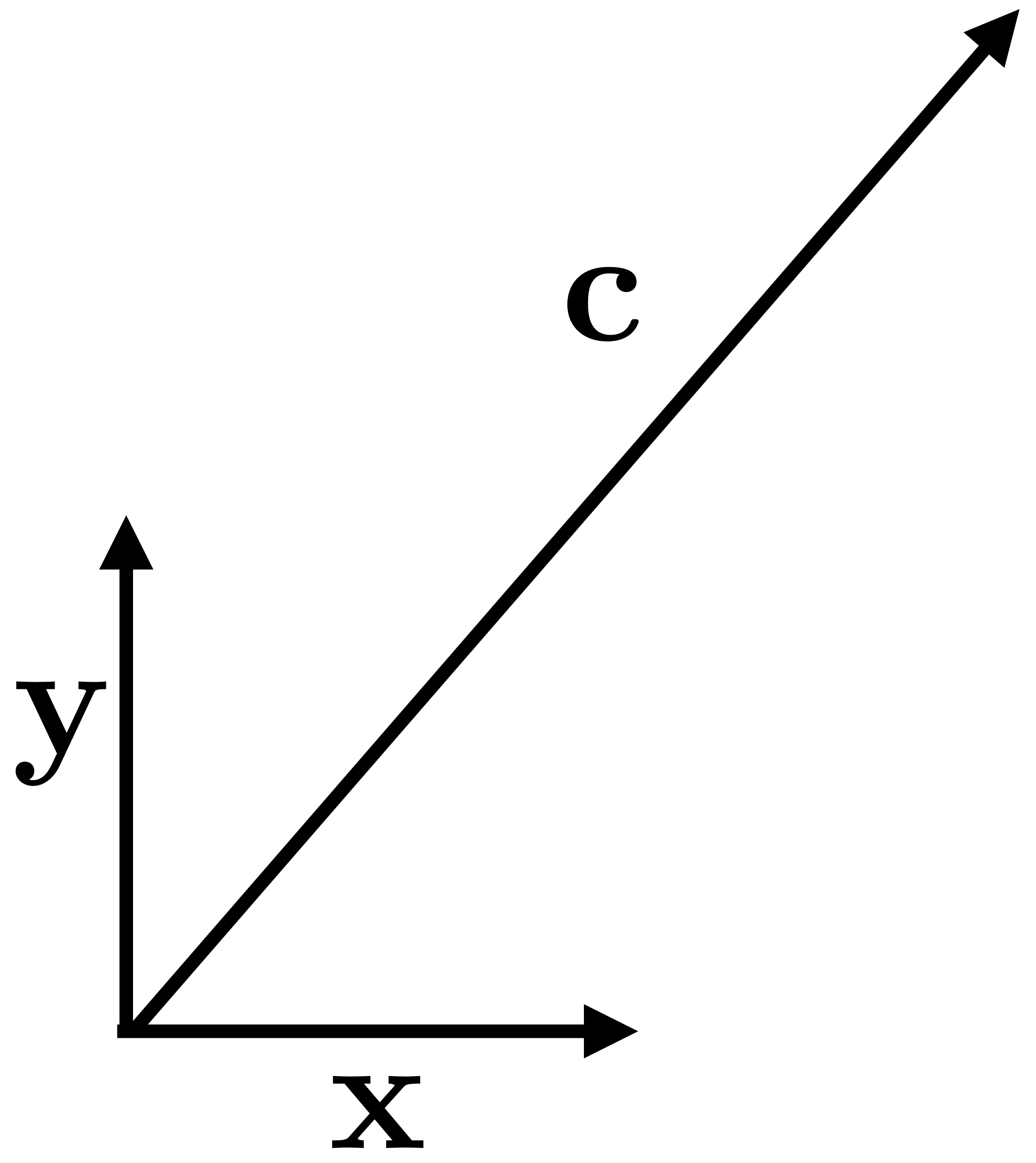




a and **b** form a 2D basis

Cartesian Coordinates

$$\mathbf{c} = c_1\mathbf{x} + c_2\mathbf{y}$$



- **x** and **y** form a canonical, Cartesian basis

Length

- The length of a vector is denoted as ||**a**||

a.norm()

- If the vector is represented in cartesian coordinates, then it is the L2 norm of the vector:

$$||\mathbf{a}|| = \sqrt{a_1^2 + a_2^2}$$

- A vector can be normalized, to change its length to 1, without affecting the direction:

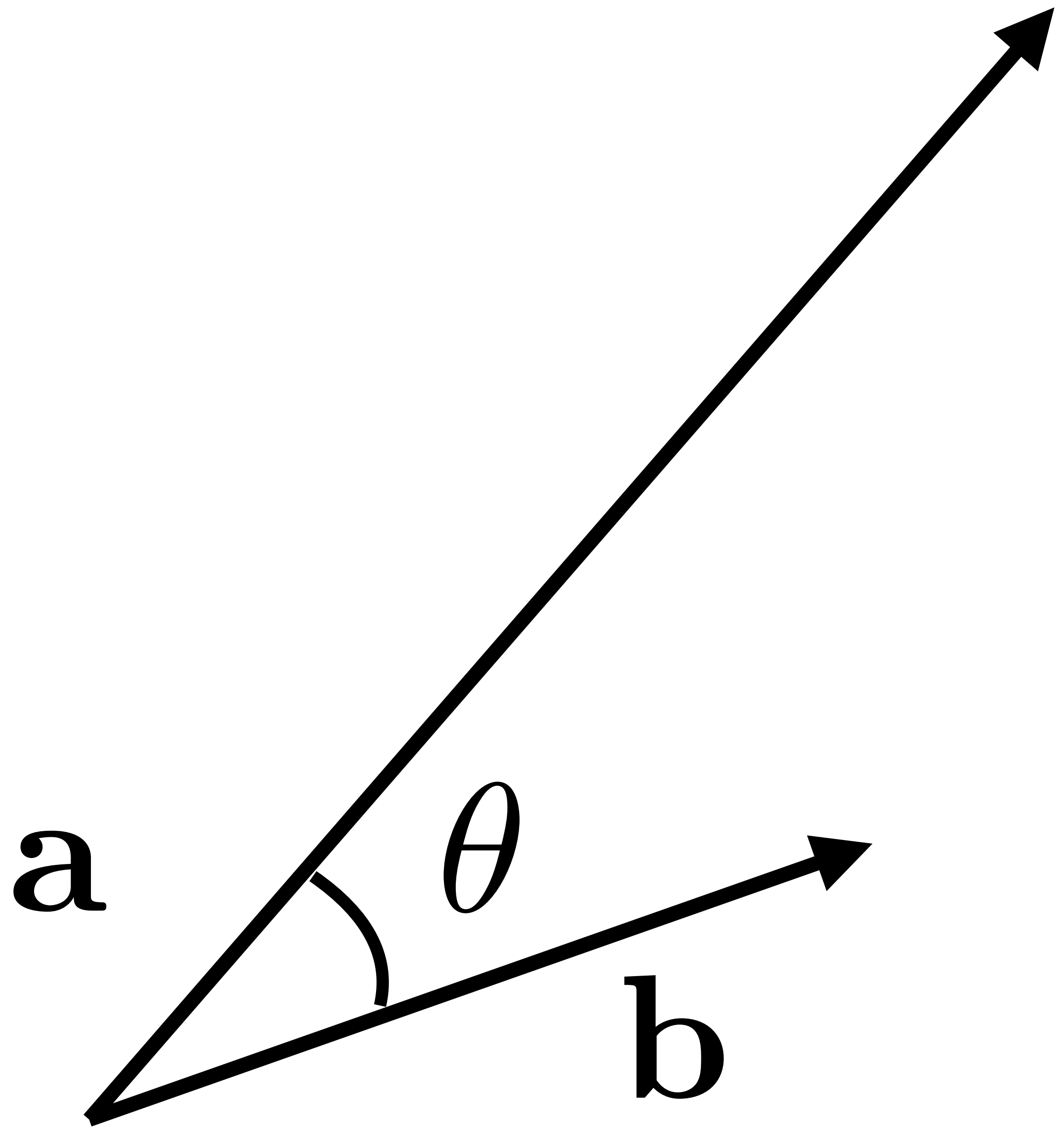
$$\mathbf{b} = \frac{\mathbf{a}}{||\mathbf{a}||}$$

CAREFUL:
b.normalize() <— in place
b.normalized() <— returns the normalized vector

Dot Product

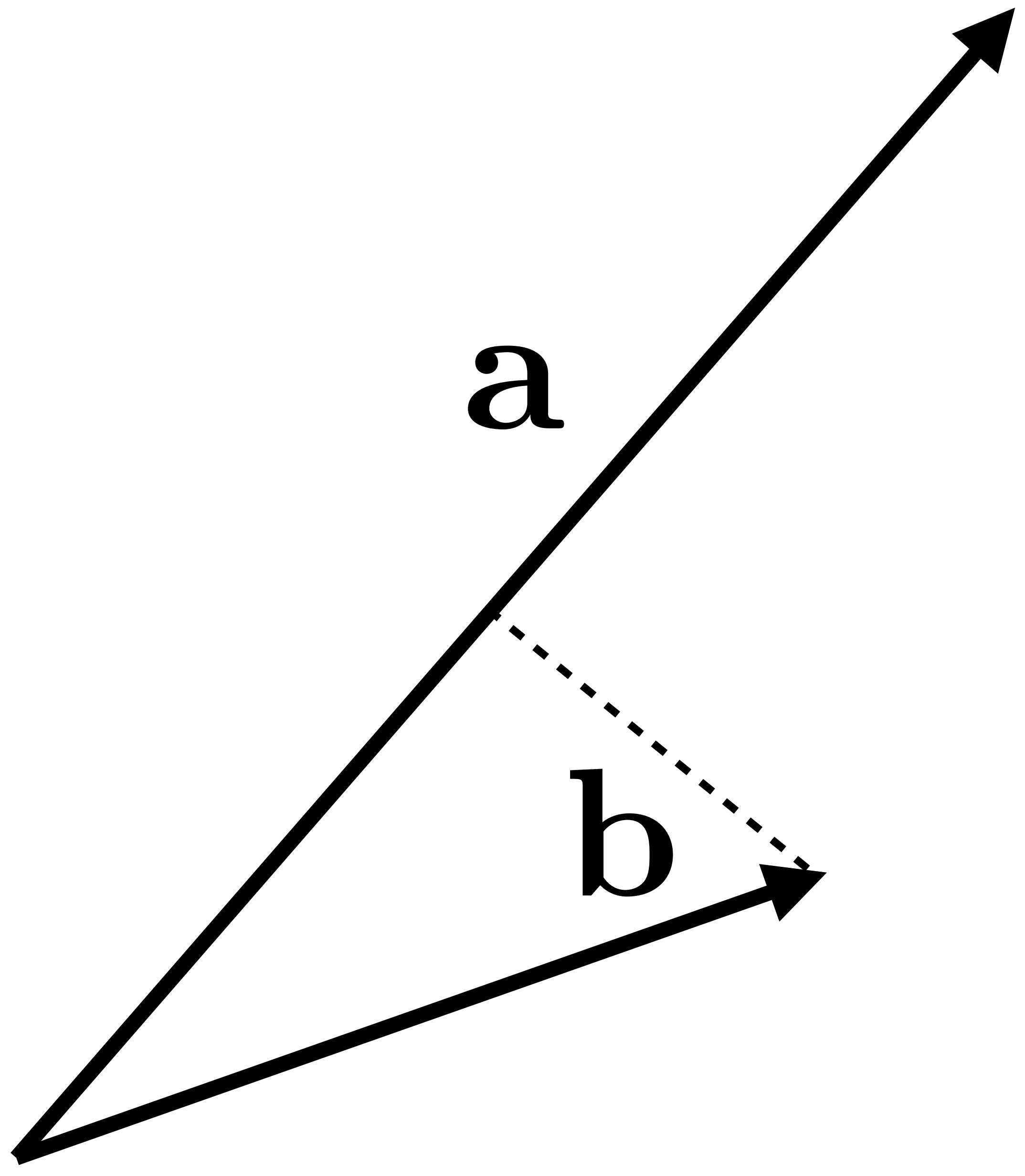
```
a.dot(b)
a.transpose()*b
```

$$\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}||\ ||\mathbf{b}|| \cos\theta$$



- The dot product is related to the length of vector and of the angle between them
- If both are normalized, it is directly the cosine of the angle between them

Dot Product - Projection



- The length of the projection of **b** onto **a** can be computed using the dot product

$$\mathbf{b} \to \mathbf{a} = ||\mathbf{b}|| \cos\theta = \frac{\mathbf{b} \cdot \mathbf{a}}{||\mathbf{a}||}$$

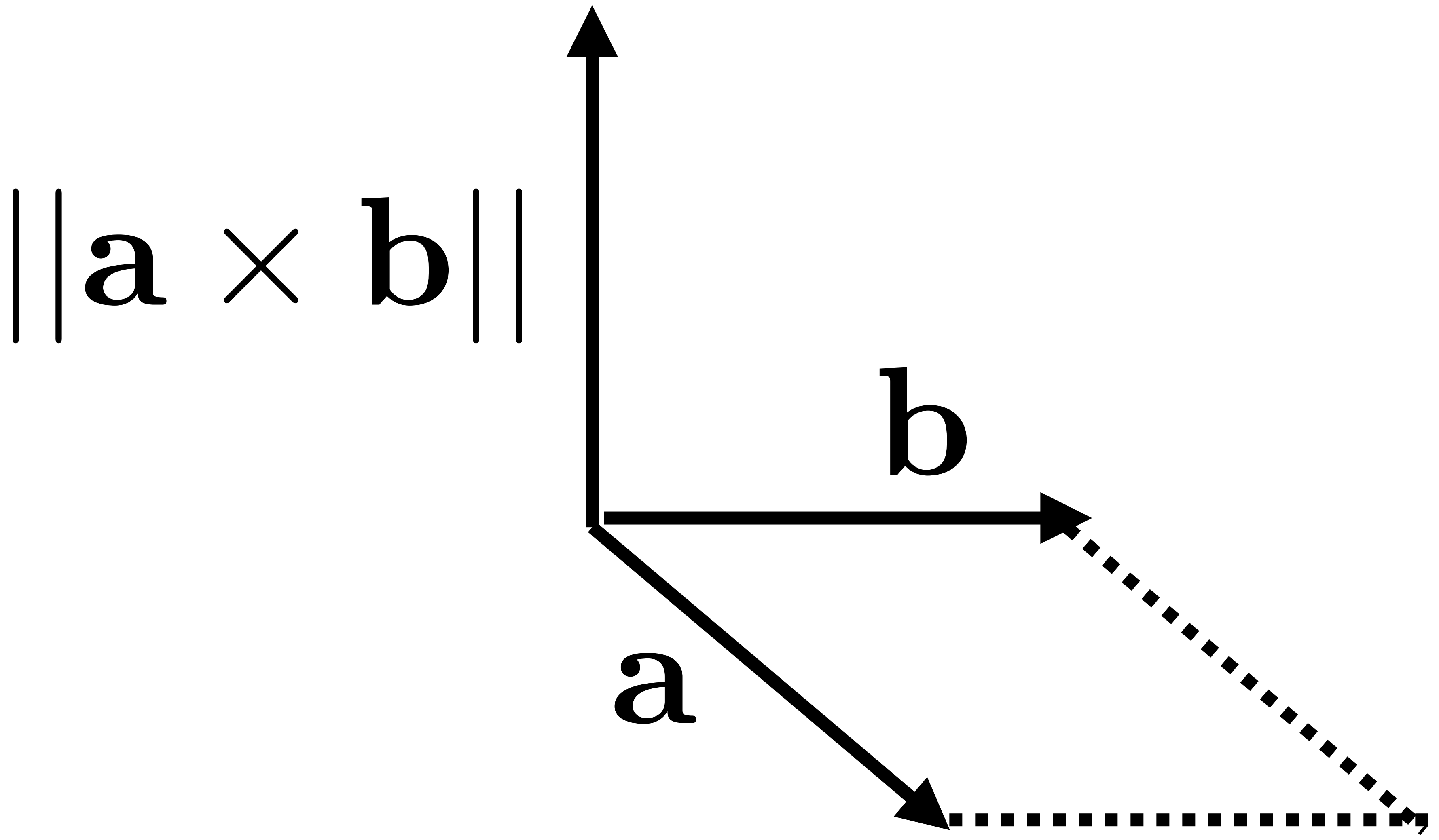
NYU COURANT

Cross Product

```
Eigen::Vector3d v(1, 2, 3);
Eigen::Vector3d w(4, 5, 6);
v.cross(w);
```

$$||\mathbf{a} \times \mathbf{b}|| = ||\mathbf{a}|| \; ||\mathbf{b}|| \sin \theta$$

- Defined only for 3D vectors
- The resulting vector is perpendicular to both **a** and **b**, the direction depends on the *right hand rule*
- The magnitude is equal to the area of the parallelogram formed by **a** and **b**



Coordinate Systems

- You will often need to manipulate coordinate systems (i.e. for finding the position of the pixels in Assignment 1)
- You will always use *orthonormal bases,* which are formed by pairwise orthogonal unit vectors :

2D

$$||\mathbf{u}|| = ||\mathbf{v}|| = 1,$$
$$\mathbf{u} \cdot \mathbf{v} = 0$$

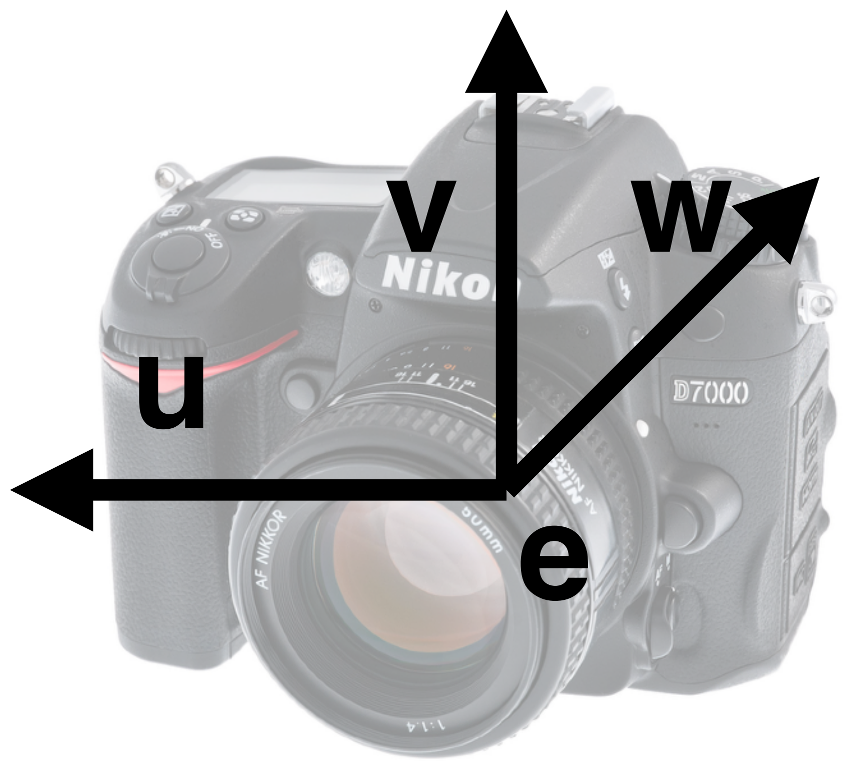
3D

$$||\mathbf{u}|| = ||\mathbf{v}|| = ||\mathbf{w}|| = 1,$$
$$\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{w} = \mathbf{w} \cdot \mathbf{u} = 0$$

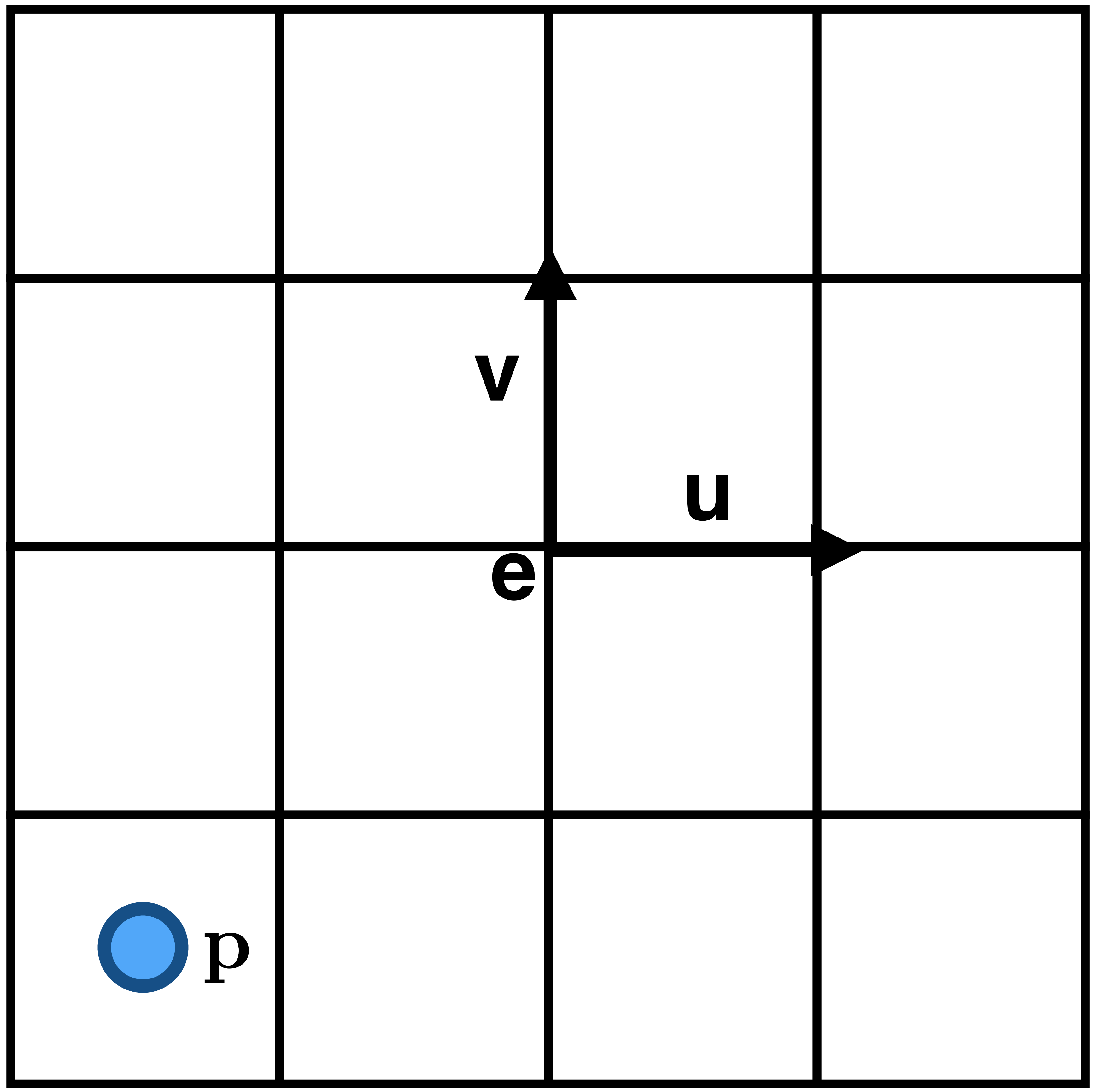
Right-handed if: $\mathbf{w} = \mathbf{u} \times \mathbf{v}$

Coordinate Frame

e is the origin of the reference system
p is the center of the pixel

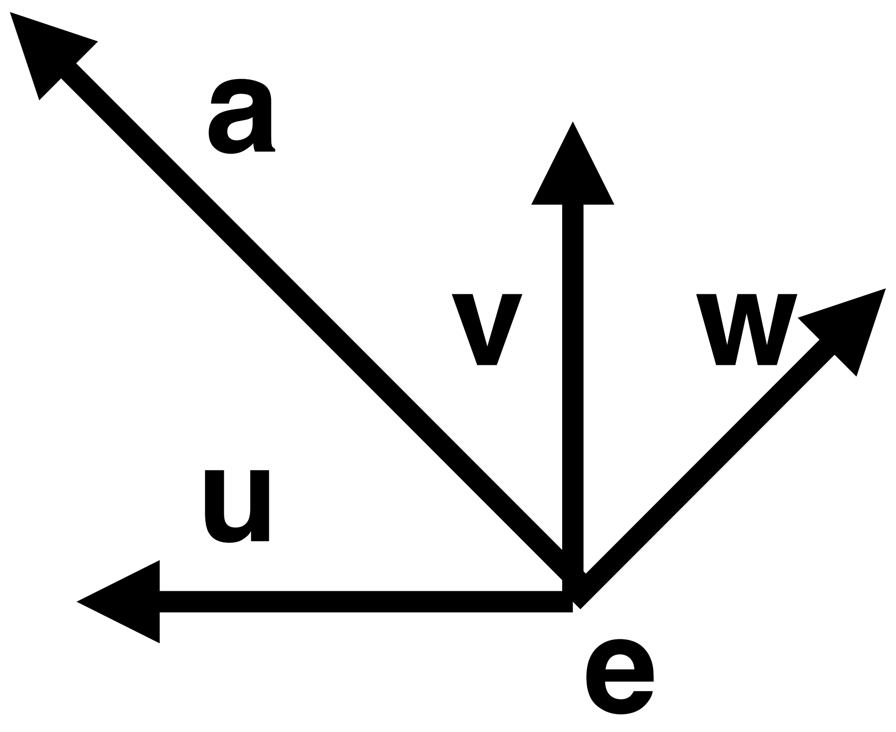


u, *v*, *w* are the coordinates of **p**
wrt the frame of reference or coordinate frame
(note that they depend also on the origin **e**)



$$\mathbf{p} = \mathbf{e} + u\mathbf{u} + v\mathbf{v} + w\mathbf{w}$$

Change of frame



- If you have a vector **a** expressed in global coordinates, and you want to convert it into a vector expressed in a local orthonormal **u-v-w** coordinate system, you can do it using projections of **a** onto **u, v, w** (which we assume are expressed in global coordinates):

$$\mathbf{a}^{\mathbf{C}} = (\mathbf{a} \cdot \mathbf{u}, \mathbf{a} \cdot \mathbf{v}, \mathbf{a} \cdot \mathbf{w})$$

References

Fundamentals of Computer Graphics, Fourth Edition
4th Edition **by Steve Marschner, Peter Shirley**

Chapter 2

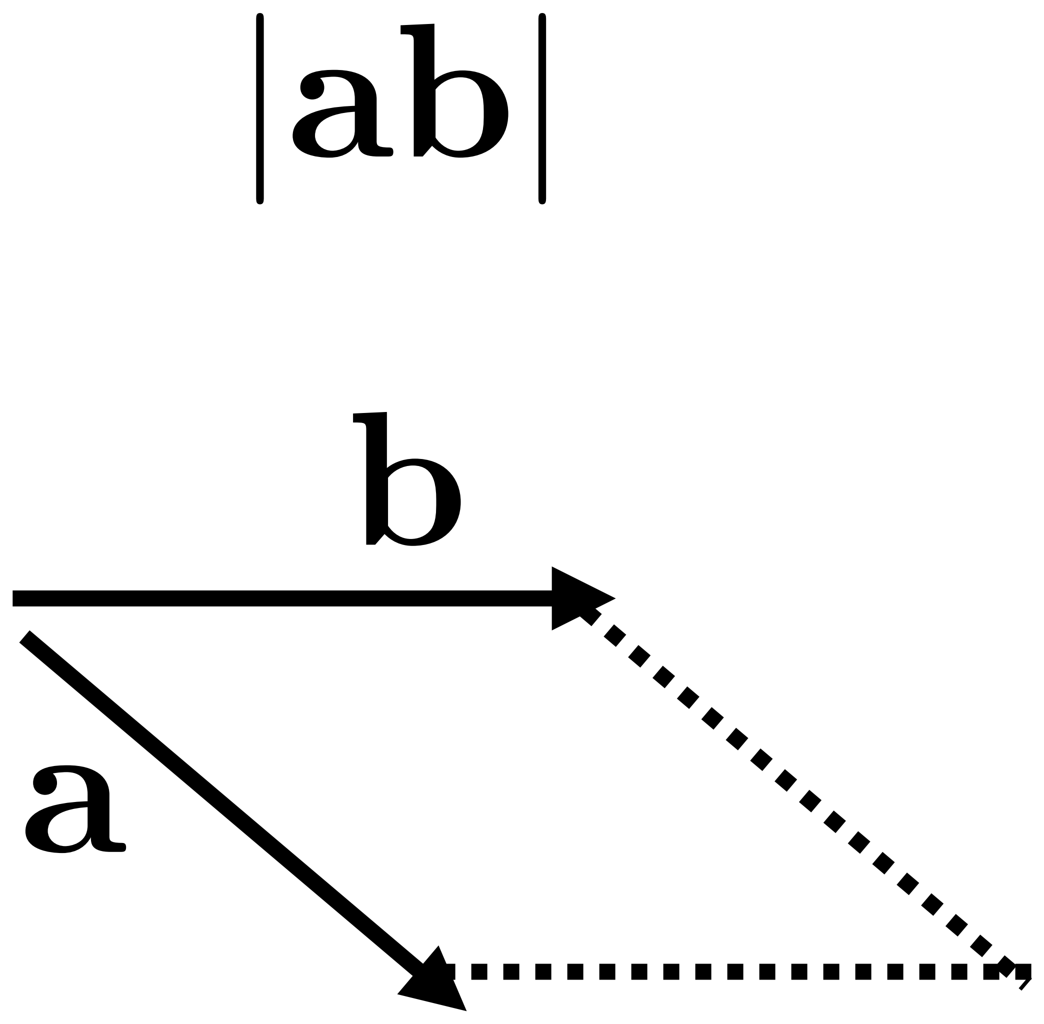
Matrices

Overview

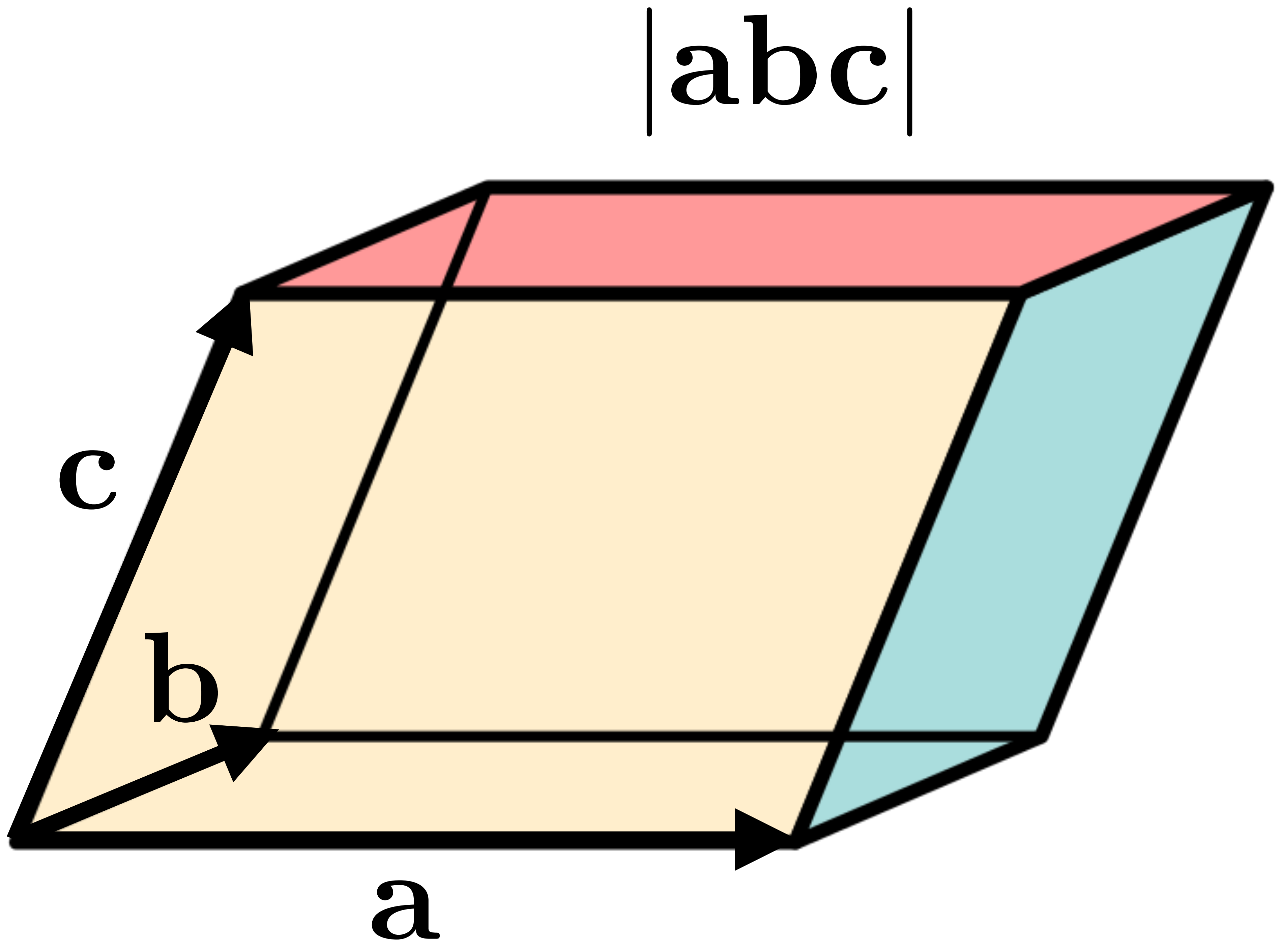
- Matrices will allow us to conveniently represent and ally transformations on vectors, such as translation, scaling and rotation
- Similarly to what we did for vectors, we will briefly overview their basic operations

Determinants

- Think of a determinant as an operation between vectors.



Area of the parallelogram



Volume of the parallelepiped
(positive since abc is a right-handed basis)

Matrices

```
Eigen::MatrixXd A(2,2)
```

- A matrix is an array of numeric elements

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$$

Sum $$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} + \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} = \begin{bmatrix} x_{11} + y_{11} & x_{12} + y_{12} \\ x_{21} + y_{21} & x_{22} + y_{22} \end{bmatrix}$$

```
A.array() + B.array()
```

Scalar Product $$y * \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} yx_{11} & yx_{12} \\ yx_{21} & yx_{22} \end{bmatrix}$$

```
A.array() * y
```

Transpose

```
B = A.transpose();
A.transposeInPlace();
```

- The transpose of a matrix is a new matrix whose entries are reflected over the diagonal

$$\begin{bmatrix}1 & 2\end{bmatrix}^T = \begin{bmatrix}1 \\ 2\end{bmatrix} \qquad \begin{bmatrix}1 & 2 \\ 3 & 4\end{bmatrix}^T = \begin{bmatrix}1 & 3 \\ 2 & 4\end{bmatrix} \qquad \begin{bmatrix}1 & 2 \\ 3 & 4 \\ 5 & 6\end{bmatrix}^T = \begin{bmatrix}1 & 3 & 5 \\ 2 & 4 & 6\end{bmatrix}$$

- The transpose of a product is the product of the transposed, in reverse order

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$$

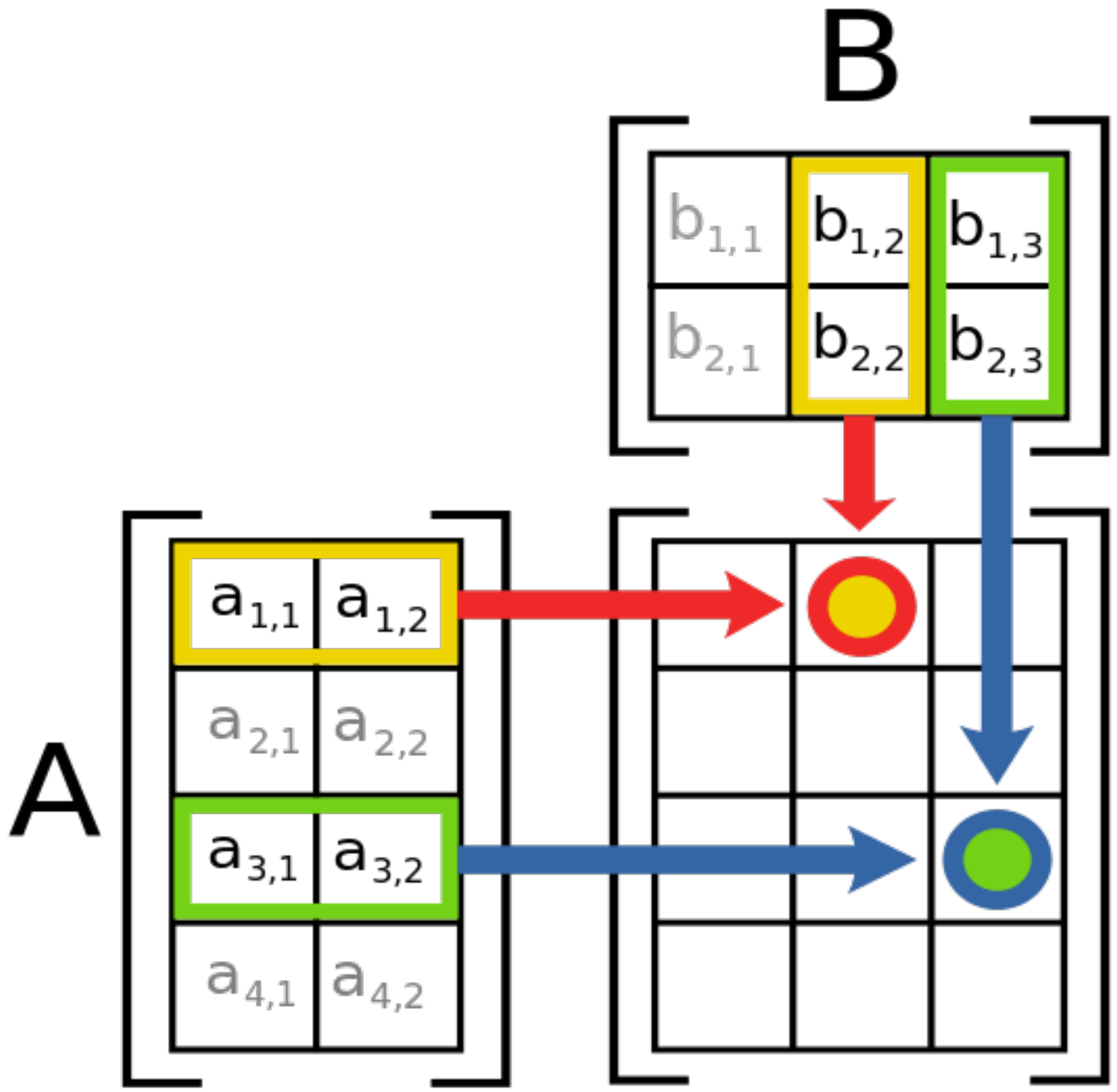
Matrix Product

```
Eigen::MatrixXd A(4,2);
Eigen::MatrixXd B(2,3);
A*B;
```

- The entry i,j is given by multiplying the entries on the i-th row of A with the entries of the j-th column of B and summing up the results

- It is NOT commutative (in general):

$$\mathbf{AB} \neq \mathbf{BA}$$



Intuition

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = \begin{bmatrix} -\mathbf{r_1}- \\ -\mathbf{r_2}- \\ -\mathbf{r_3}- \end{bmatrix} \begin{bmatrix} | \\ \mathbf{x} \\ | \end{bmatrix}$$

$$y_i = \mathbf{r_i} \cdot \mathbf{x}$$

Dot product on each row

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = \begin{bmatrix} | & | & | \\ \mathbf{c_1} & \mathbf{c_2} & \mathbf{c_3} \\ | & | & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\mathbf{y} = x_1\mathbf{c_1} + x_2\mathbf{c_2} + x_3\mathbf{c_3}$$

Weighted sum of the columns

Inverse Matrix

```
Eigen::MatrixXd A(4,4);
A.inverse() <— do not use this
to solve a linear system!
```

- The inverse of a matrix $\mathbf{A}$ is the matrix $\mathbf{A}^{-1}$ such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$

  where $\mathbf{I}$ is the *identity matrix* $\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- The inverse of a product is the product of the inverse in opposite order:

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$$

Diagonal Matrices

```
Eigen::Vector3d v(1,2,3);

A = v.asDiagonal()
```

- They are zero everywhere except the diagonal:

$$\mathbf{D} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}$$

- Useful properties:

$$\mathbf{D}^{-1} = \begin{bmatrix} a^-1 & 0 & 0 \\ 0 & b^-1 & 0 \\ 0 & 0 & c^{-1} \end{bmatrix}$$

$$\mathbf{D} = \mathbf{D}^T$$

Orthogonal Matrices

- An orthogonal matrix is a matrix where
  - each column is a vector of length 1
  - each column is orthogonal to all the others
- A useful property of orthogonal matrices that their inverse corresponds to their transpose:

$$(\mathbf{R}^T\mathbf{R}) = \mathbf{I} = (\mathbf{R}\mathbf{R}^T)$$

NYU | COURANT

Linear Systems

- We will often encounter in this class linear systems with *n* linear equations that depend on *n* variables.
- For example:

$$\begin{aligned} 5x + 3y - 7z &= 4 \\ -3x + 5y + 12z &= 9 \\ 9x - 2y - 2z &= -3 \end{aligned}$$

$$\begin{bmatrix} 5 & 3 & -7 \\ -3 & 5 & 12 \\ 9 & -2 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 9 \\ -3 \end{bmatrix}$$

- To find *x,y,z* you have to "solve" the linear system. Do not use an inverse, but rely on a direct solver:

```
Matrix3f A;
Vector3f b;
A << 5,3,-7,  -3,5,12,  9,-2,-2;
b << 4, 9, -3;
cout << "Here is the matrix A:\n" << A << endl;
cout << "Here is the vector b:\n" << b << endl;
Vector3f x = A.colPivHouseholderQr().solve(b);
cout << "The solution is:\n" << x << endl;
```

References

Fundamentals of Computer Graphics, Fourth Edition
4th Edition **by Steve Marschner**, **Peter Shirley**

Chapter 5

2D Transformations

2D Linear Transformations

- Each 2D linear map can be represented by a unique 2×2 matrix

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

- Concatenation of mappings corresponds to multiplication of matrices

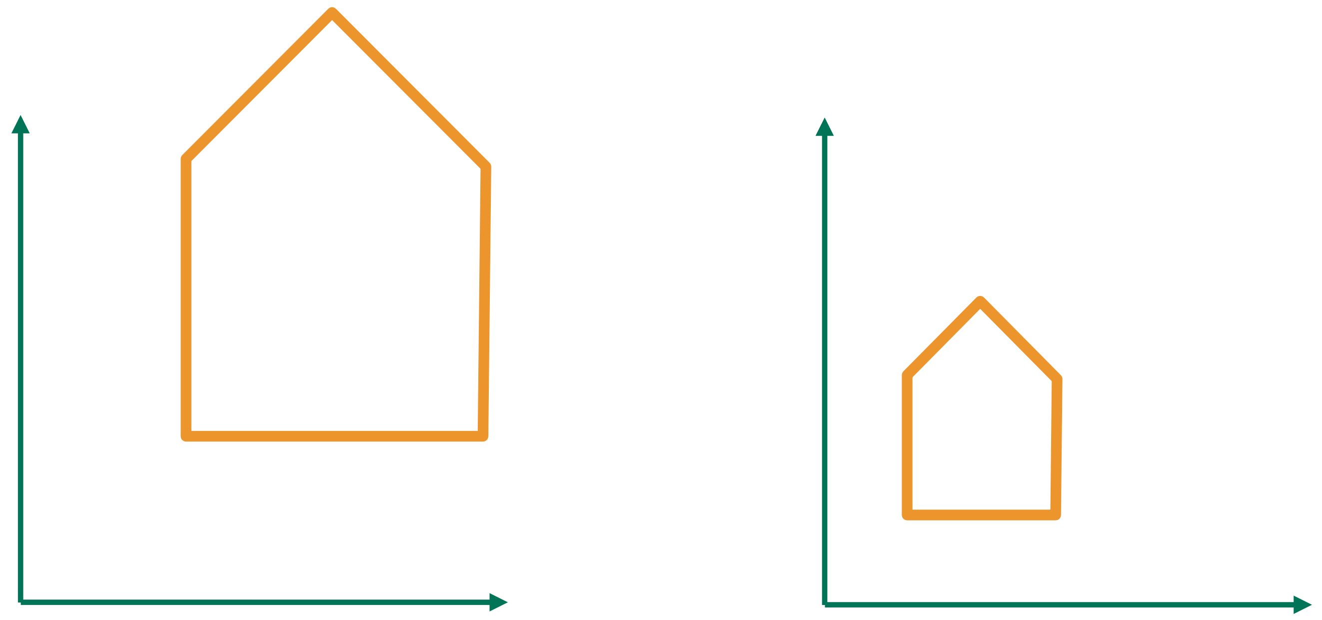
$$L_2(L_1(\mathbf{x})) = \mathbf{L}_2 \, \mathbf{L}_1 \, \mathbf{x}$$

```
L2 * L1 * x;
```

- Linear transformations are very common in computer graphics!

2D Scaling

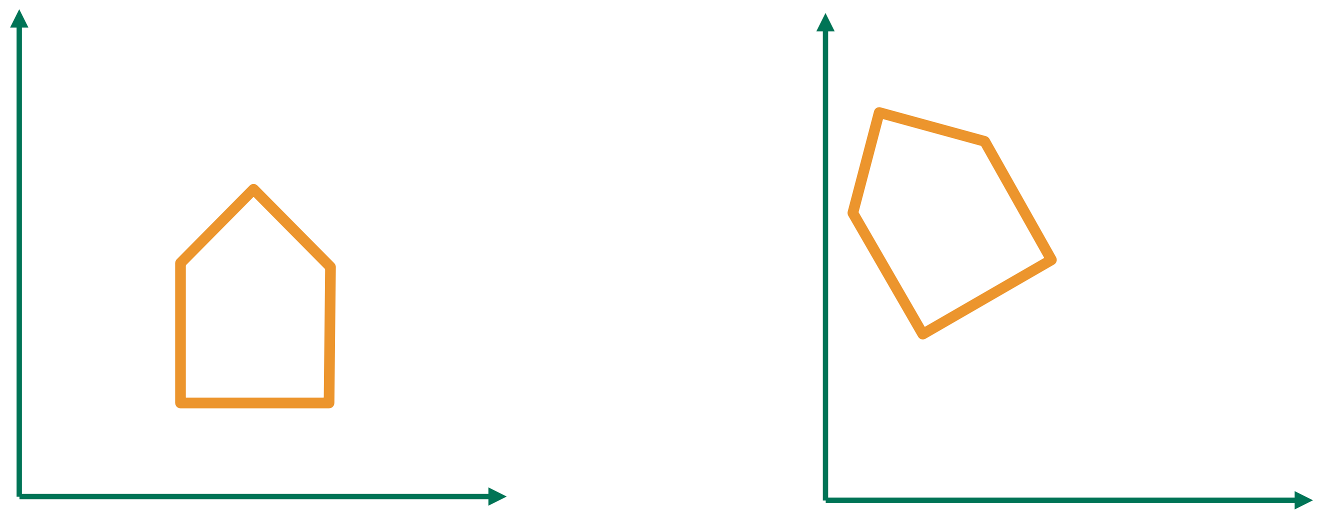
- Scaling $\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}}_{\mathbf{S}(s_x, s_y)} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$



$\mathbf{S}(0.5, 0.5)$

2D Rotation

- Rotation $\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}}_{\mathbf{R}(\alpha)} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$



$\mathbf{R}(20^\circ)$

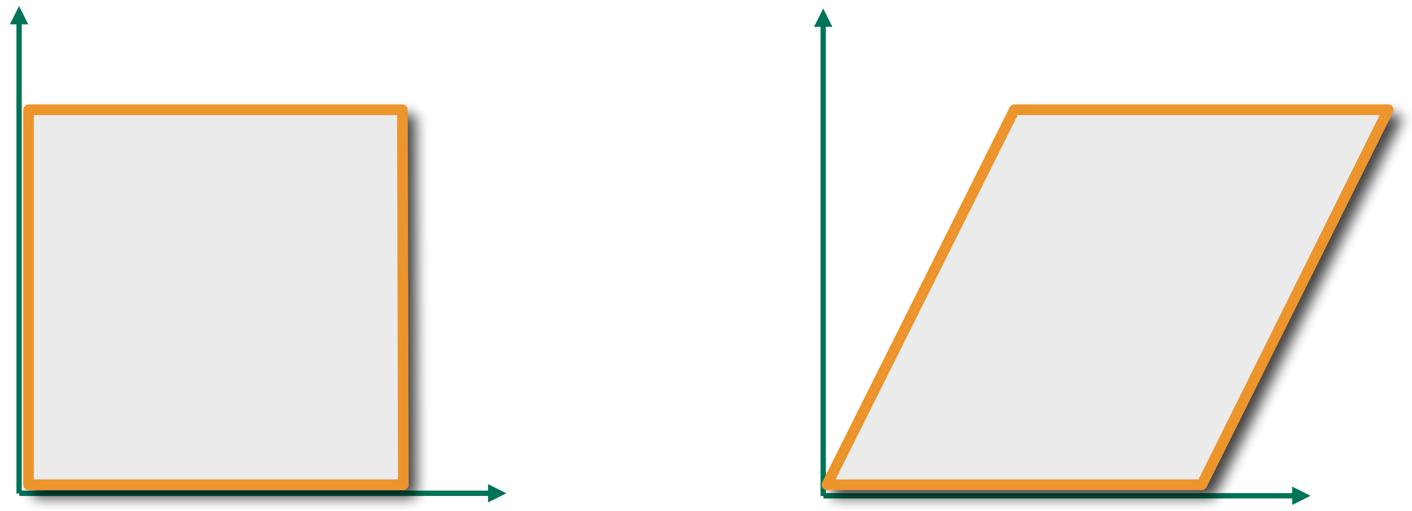
Special case: $\mathbf{R}(90) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$

NYU | COURANT

2D Shearing

- Shear along x-axis

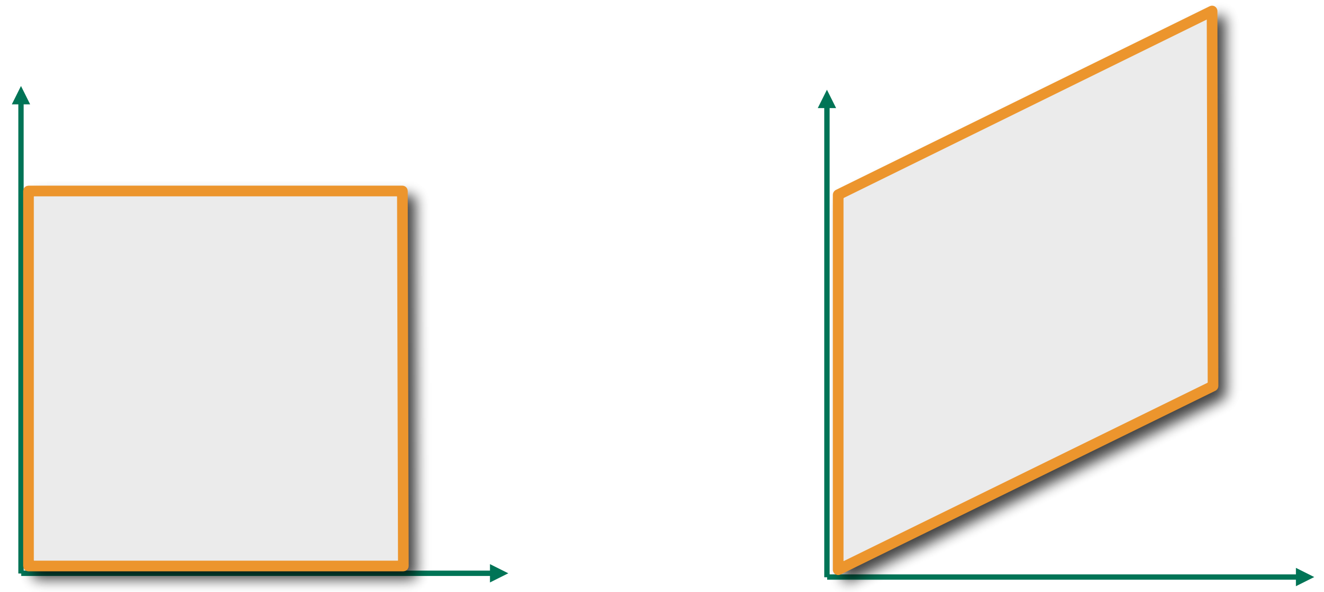
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix}}_{\mathbf{H}_x(a)} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$



$\mathbf{H}_x(0.5)$

- Shear along y-axis

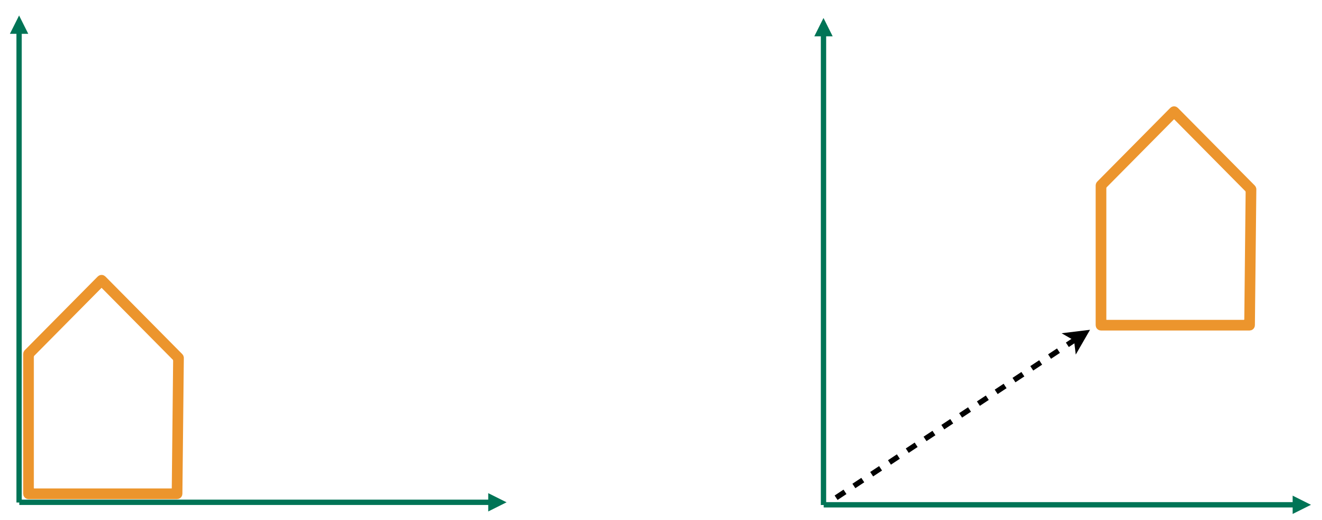
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 \\ b & 1 \end{pmatrix}}_{\mathbf{H}_y(b)} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$



$\mathbf{H}_y(0.5)$

2D Translation

- Translation $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$



- Matrix representation? $\begin{pmatrix} x' \\ y' \end{pmatrix} = \mathbf{T}(t_x, t_y) \cdot \begin{pmatrix} x \\ y \end{pmatrix}$

NYU COURANT

Affine Transformations

- Translation is not linear, but it is *affine*
  - Origin is no longer a fixed point
- Affine map = linear map + translation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} = \mathbf{Lx} + \mathbf{t}$$

- Is there a matrix representation for affine transformations?
  - We would like to handle all transformations in a unified framework -> simpler to code and easier to optimize!

Homogenous Coordinates

- Add a third coordinate (*w-coordinate*)
  - 2D point = $(x, y, 1)^T$
  - 2D vector = $(x, y, 0)^T$

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

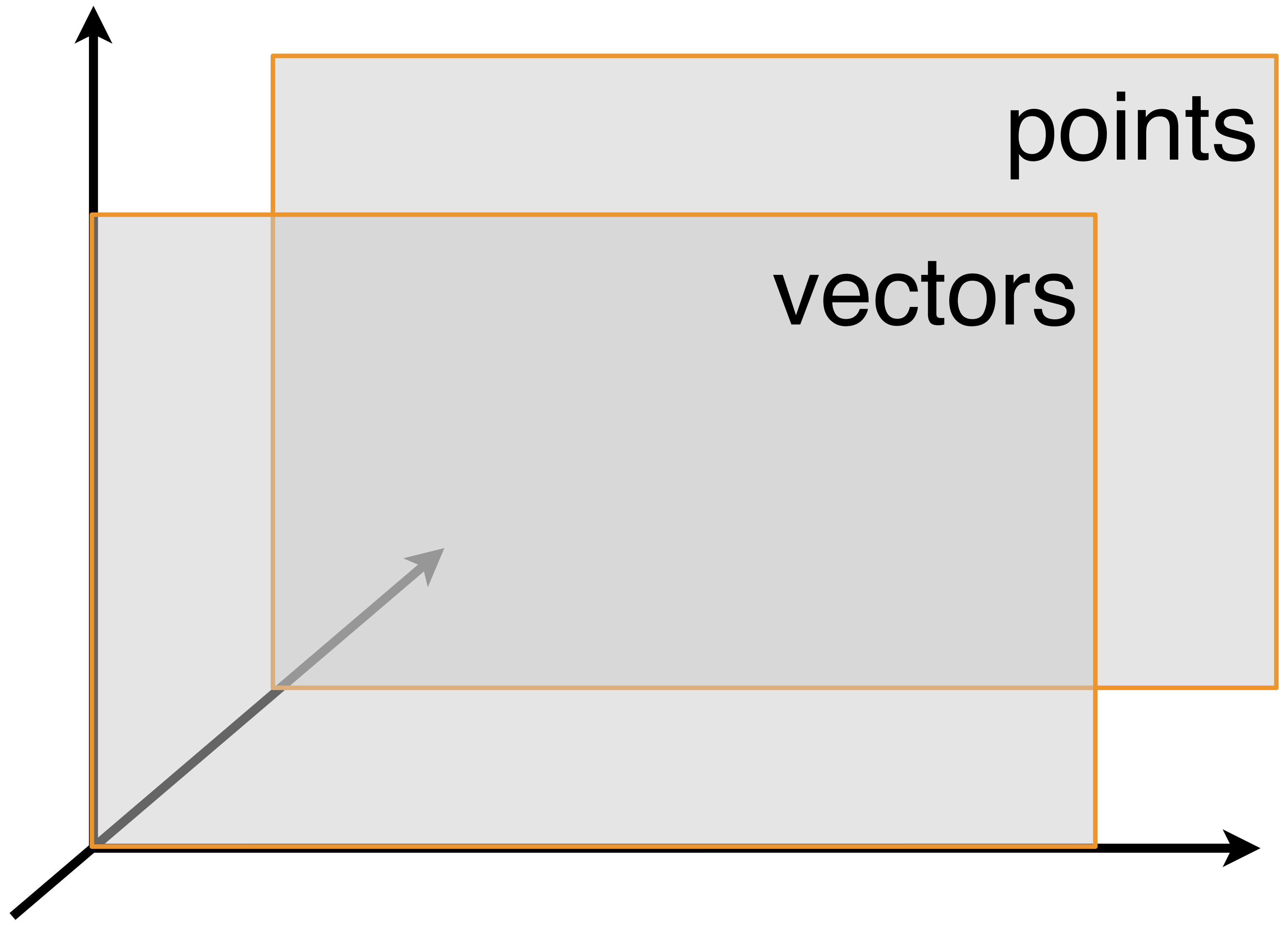
- Matrix representation of translations

Homogenous Coordinates

- Valid operation if the resulting w-coordinate is 1 or 0
  - vector + vector= vector
  - point - point = vector
  - point + vector= point
  - point + point = ???

Homogenous Coordinates

- Geometric interpretation: 2 hyperplanes in $\mathbf{R}^3$



Affine Transformations

- Affine map = linear map + translation

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

- Using homogenous coordinates:

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

2D Transformations

- Scale

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

- Rotation

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

- Translation

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

Concatenation of Transformations

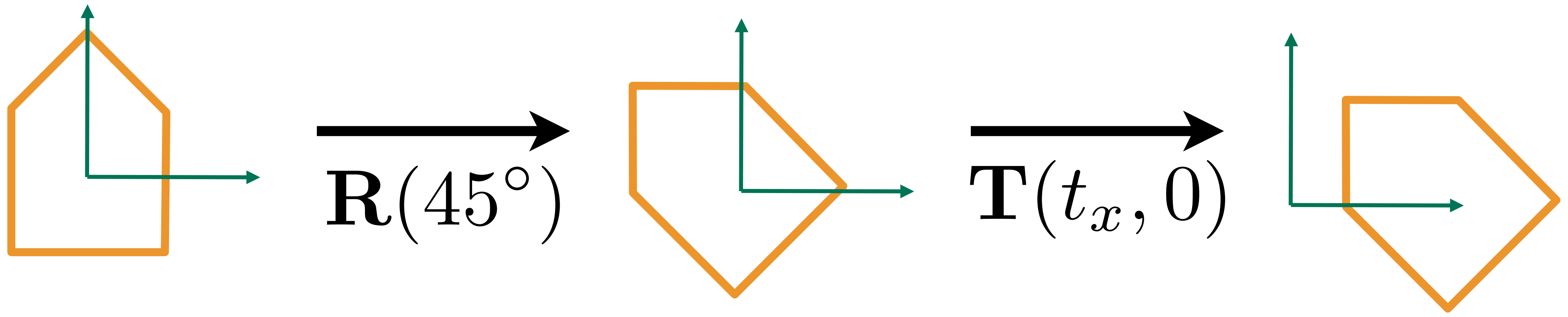
- Sequence of affine maps $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, ...
    - Concatenation by matrix multiplication

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

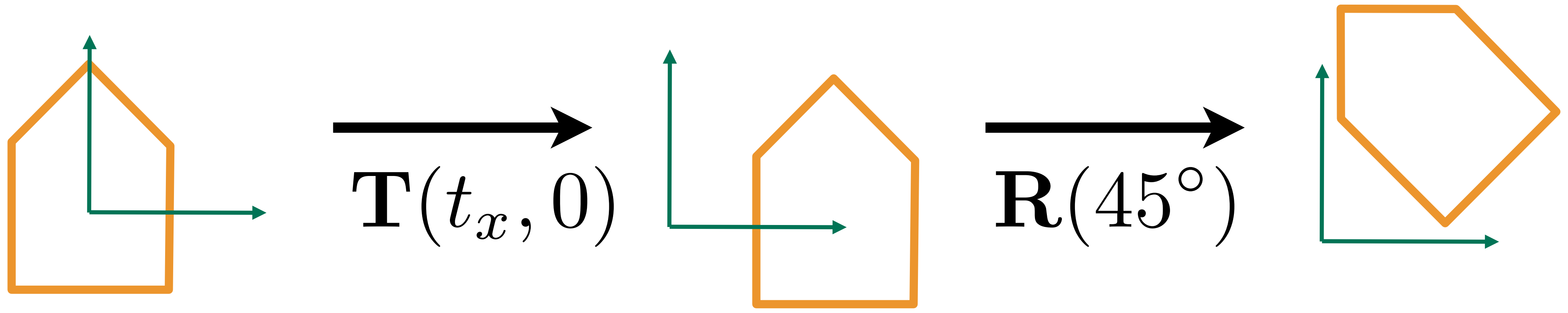
- Very important for performance!
- Matrix multiplication not commutative, ordering is important!

Rotation and Translation

- Matrix multiplication is not commutative!
  - First rotation, then translation

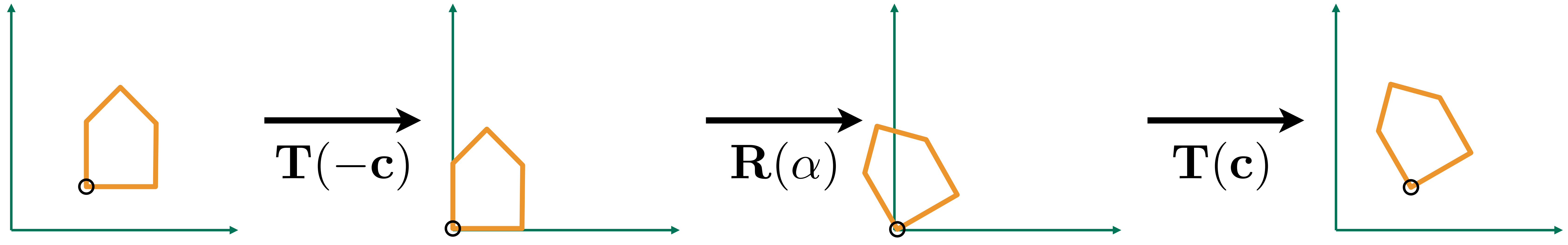


  - First translation, then rotation



2D Rotation

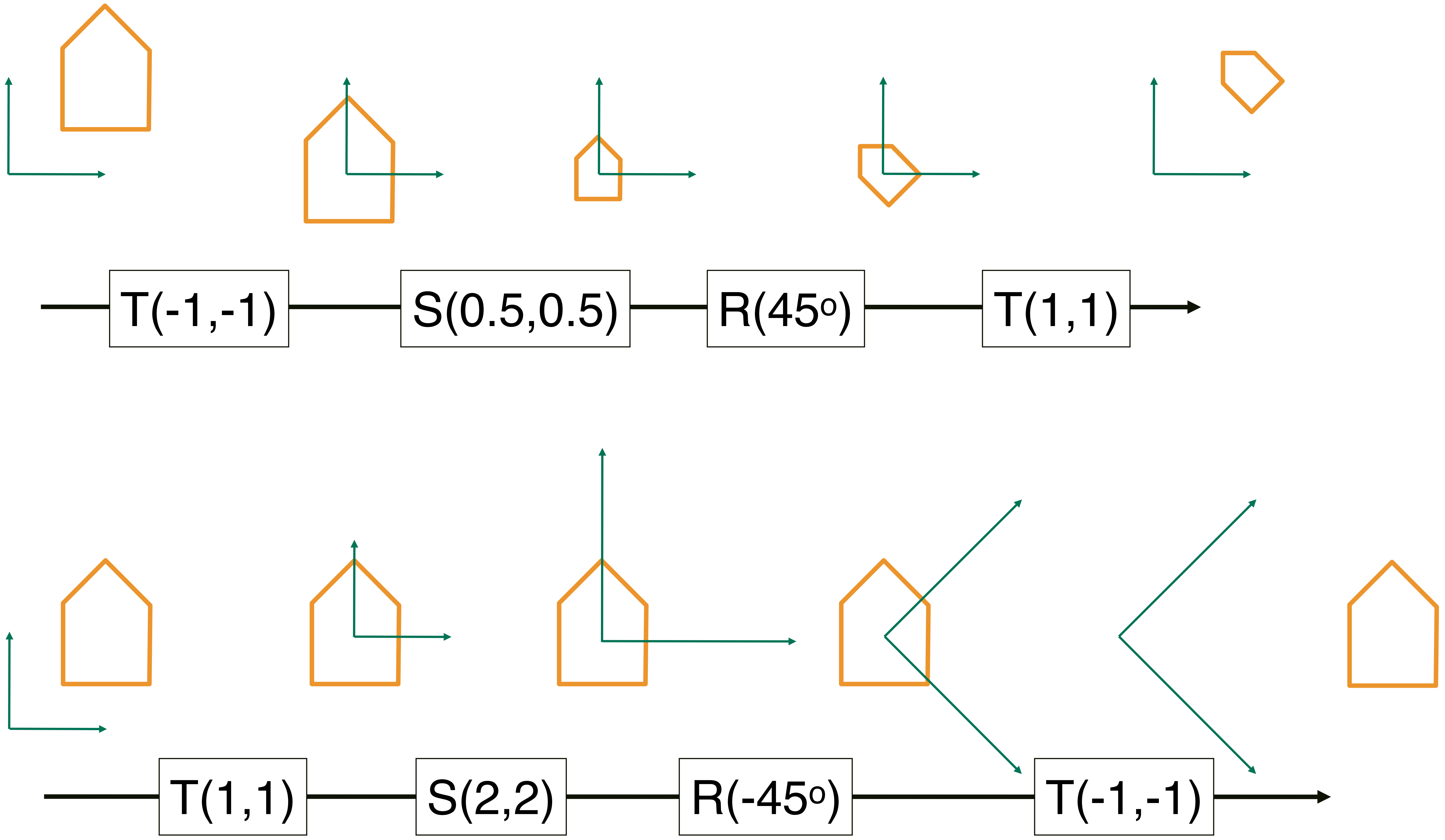
- How to rotate around a given point **c**?
  1. Translate **c** to origin
  2. Rotate
  3. Translate back



- Matrix representation?

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

Transform Object or Camera?



NYU COURANT

References

Fundamentals of Computer Graphics, Fourth Edition
4th Edition **by Steve Marschner**, **Peter Shirley**

Chapter 6