Basic Linear Algebra

Acknowledgements: Daniele Panozzo

Overview

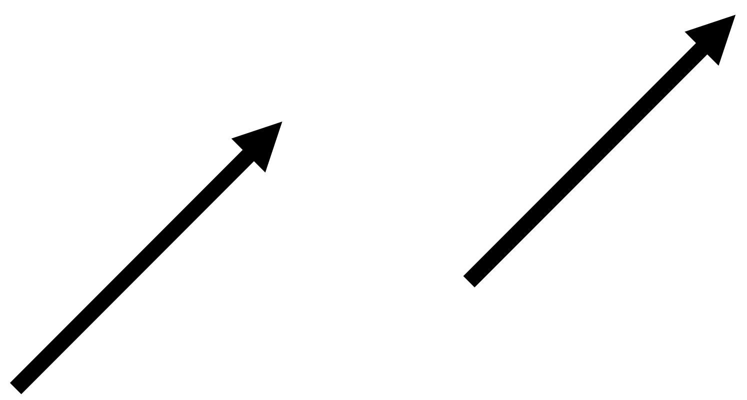
- We will briefly overview the basic linear algebra concepts that we will need in the class
- You will not be able to follow the next lectures without a clear understanding of this material

Vectors

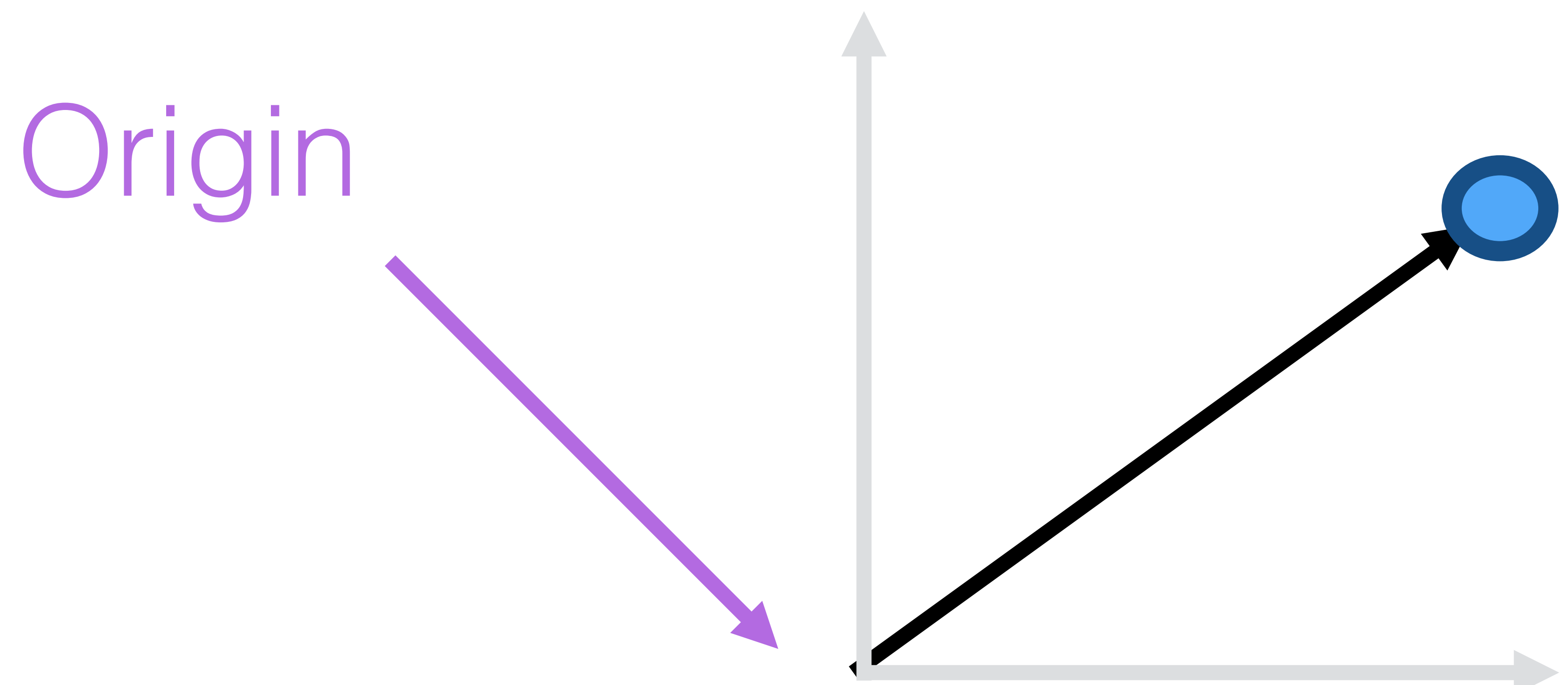
Vectors

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!

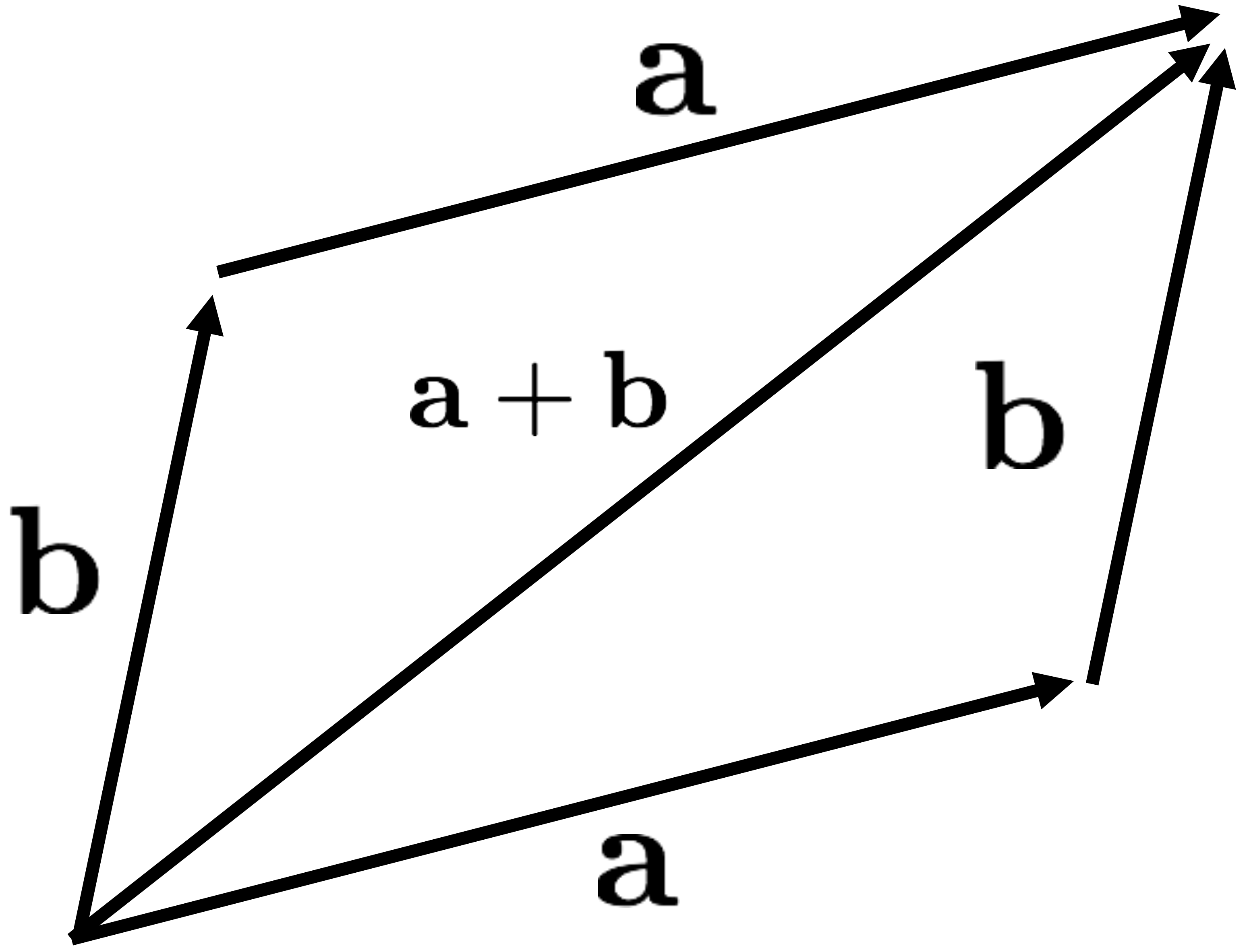


Vectors represent displacements. If you represent the displacement wrt the origin, then they *encode* a location.

Sum

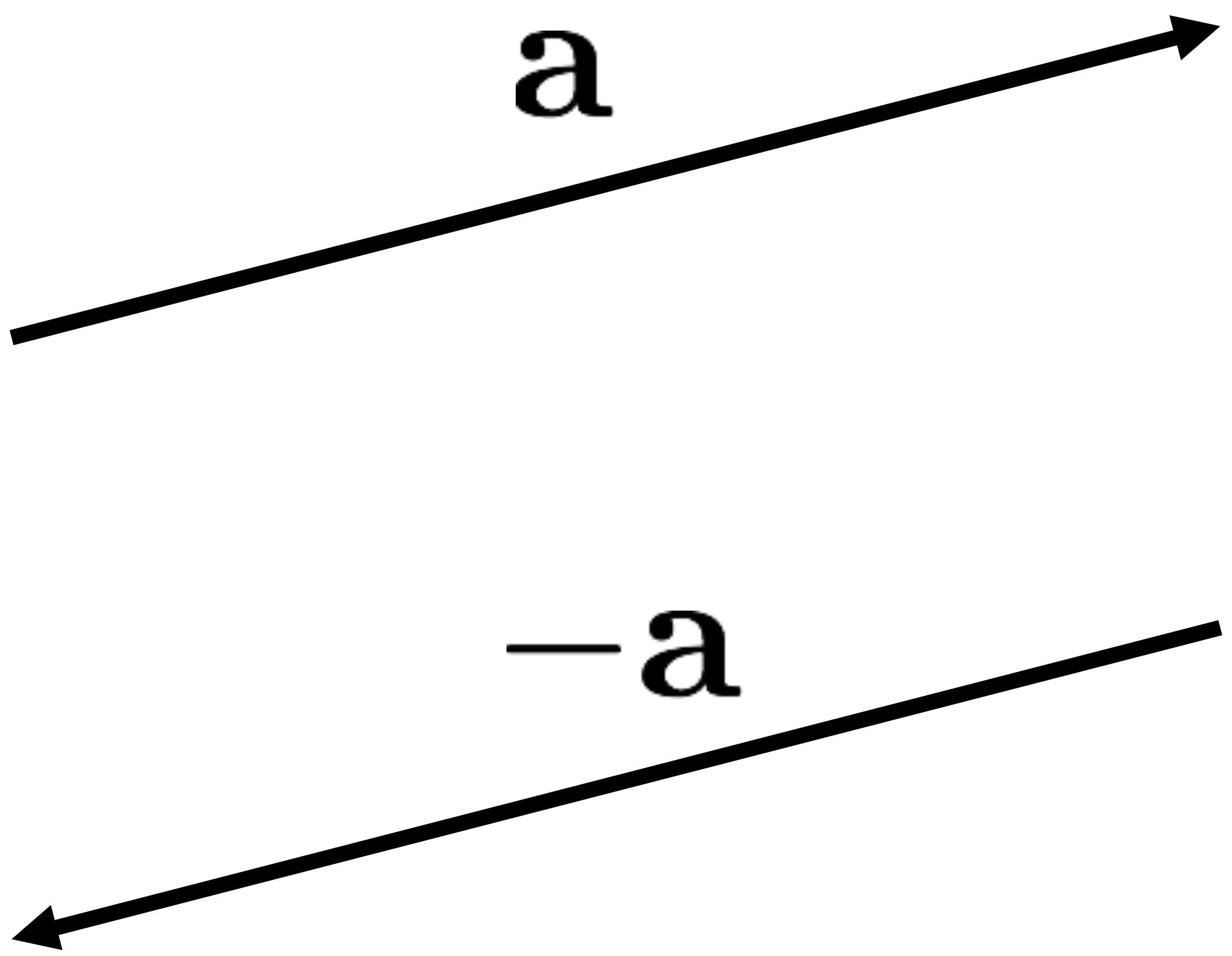
Operator +

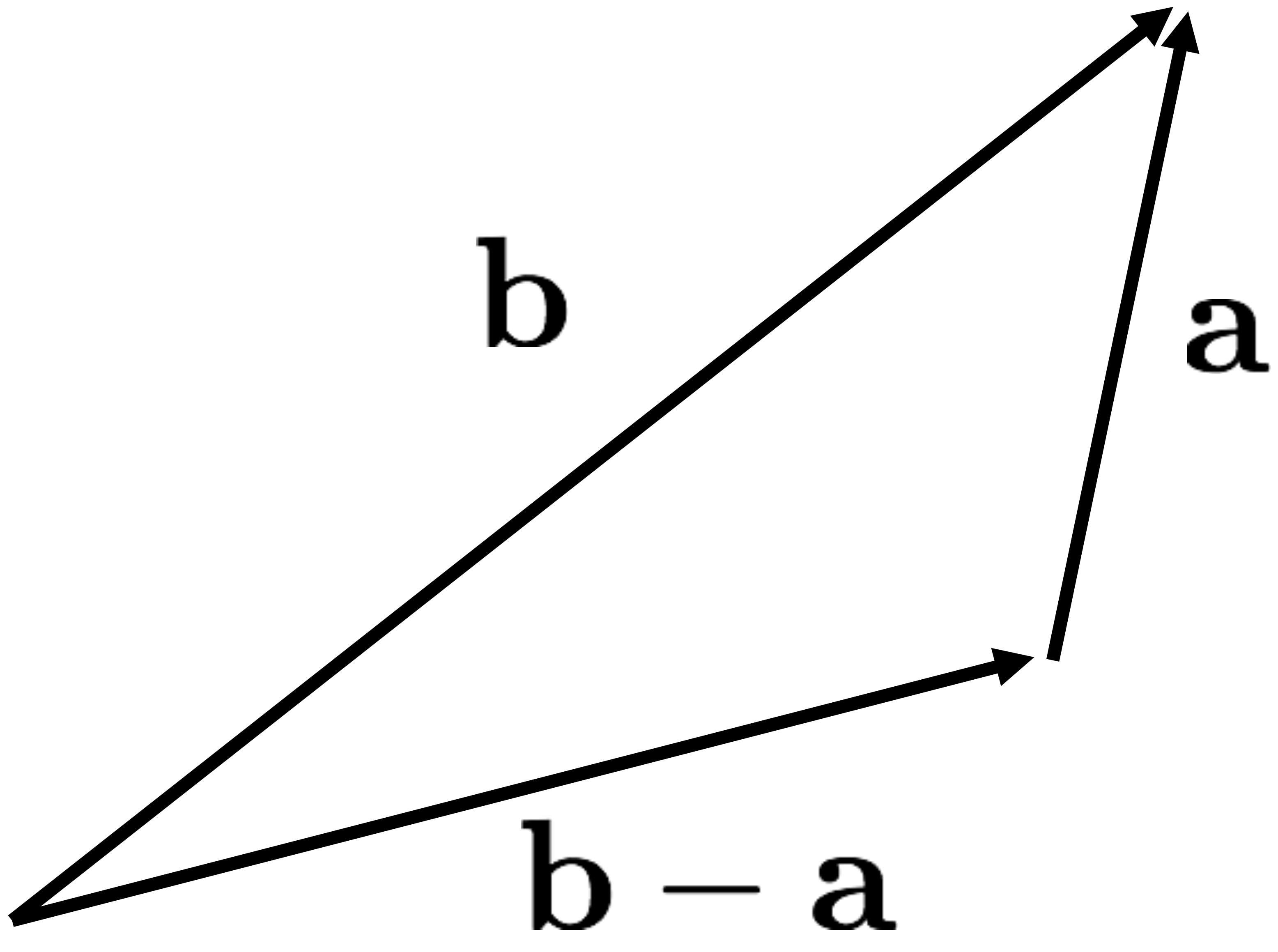
$$\mathbf{a} + \mathbf{b} = \mathbf{b} + \mathbf{a}$$



Difference

Operator -





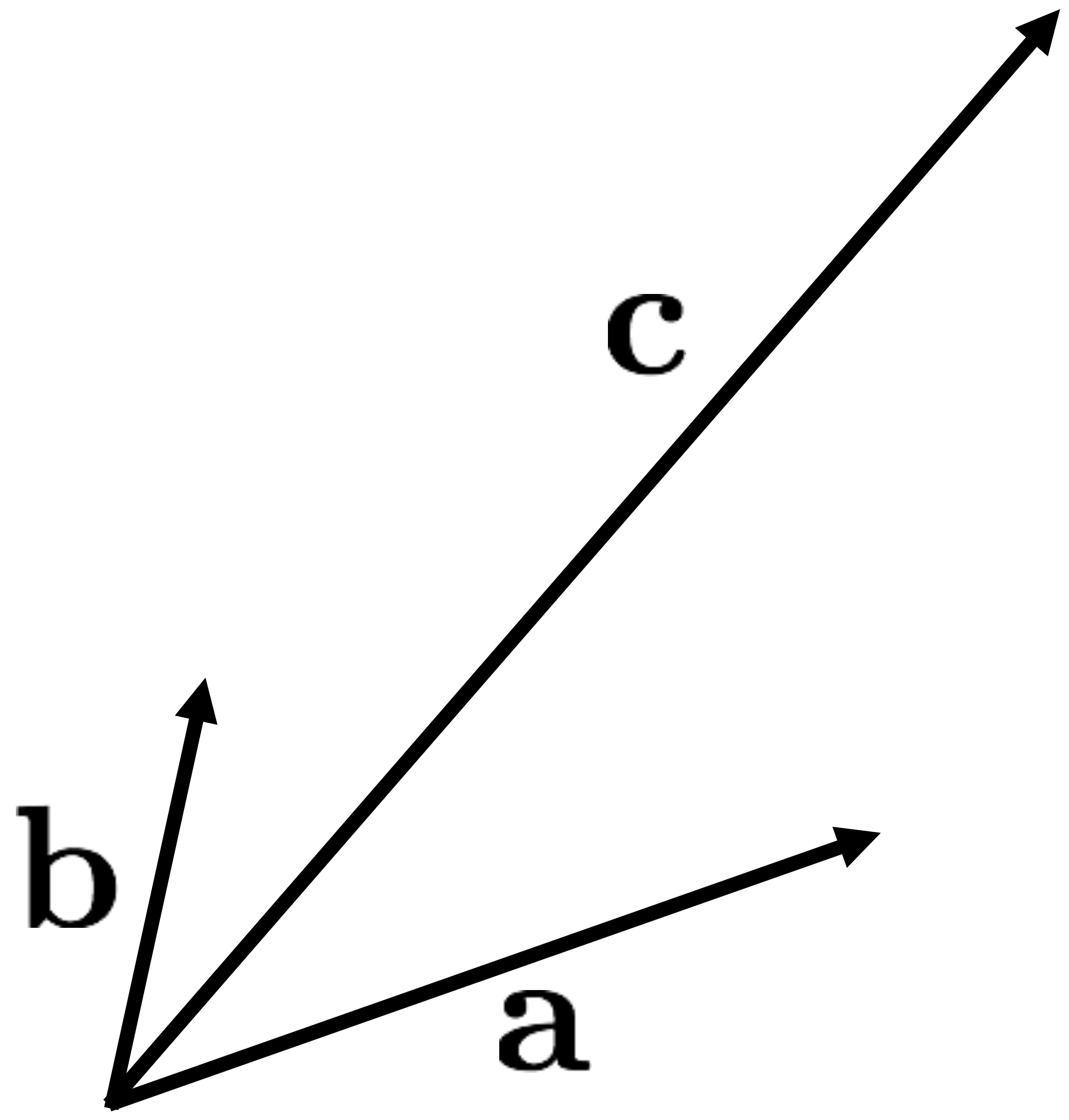
$$\mathbf{b} - \mathbf{a} = -\mathbf{a} + \mathbf{b}$$

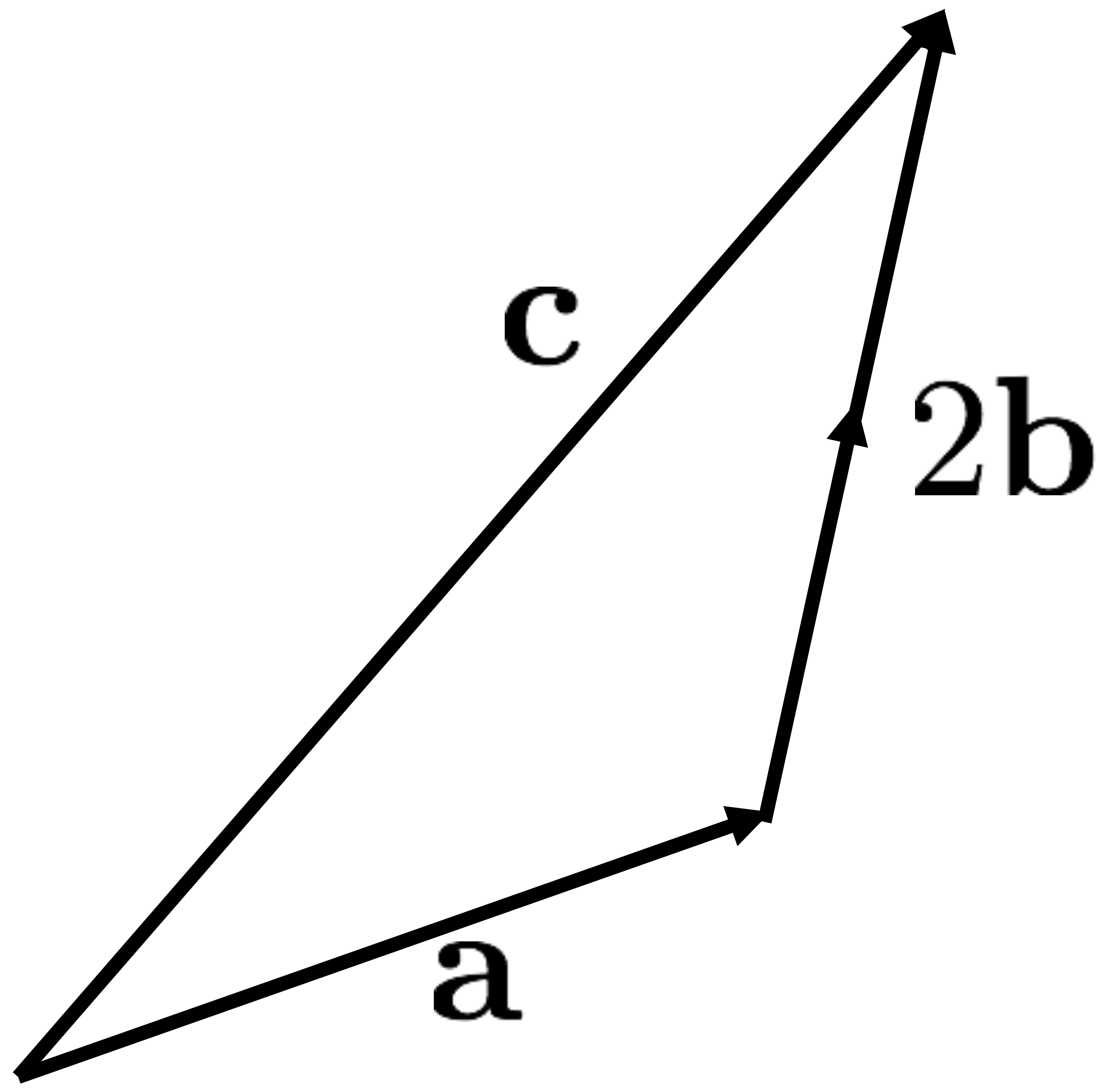
Coordinates

Operator []

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

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

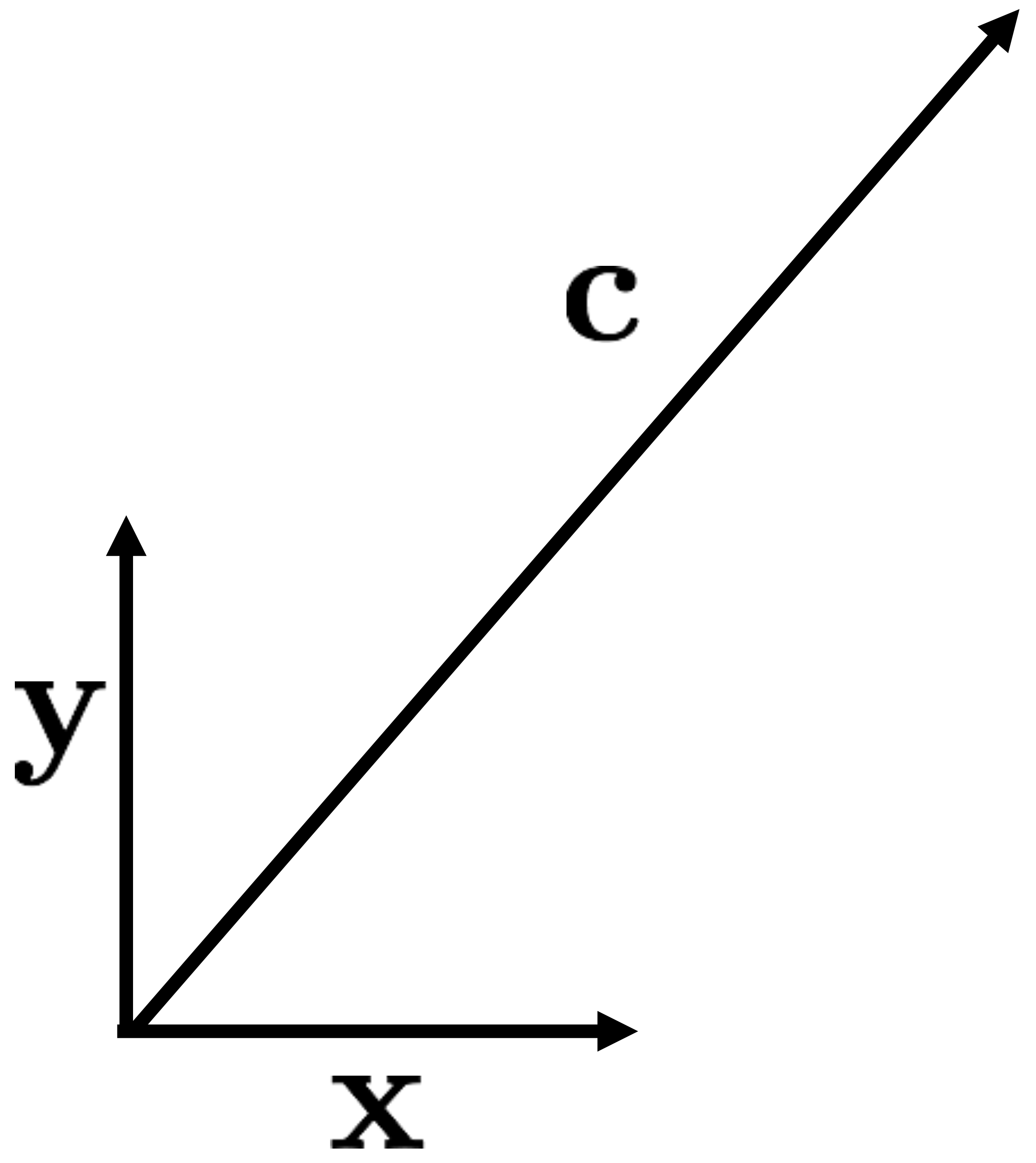




a and **b** form a 2D basis

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 $||\mathbf{a}||$

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:

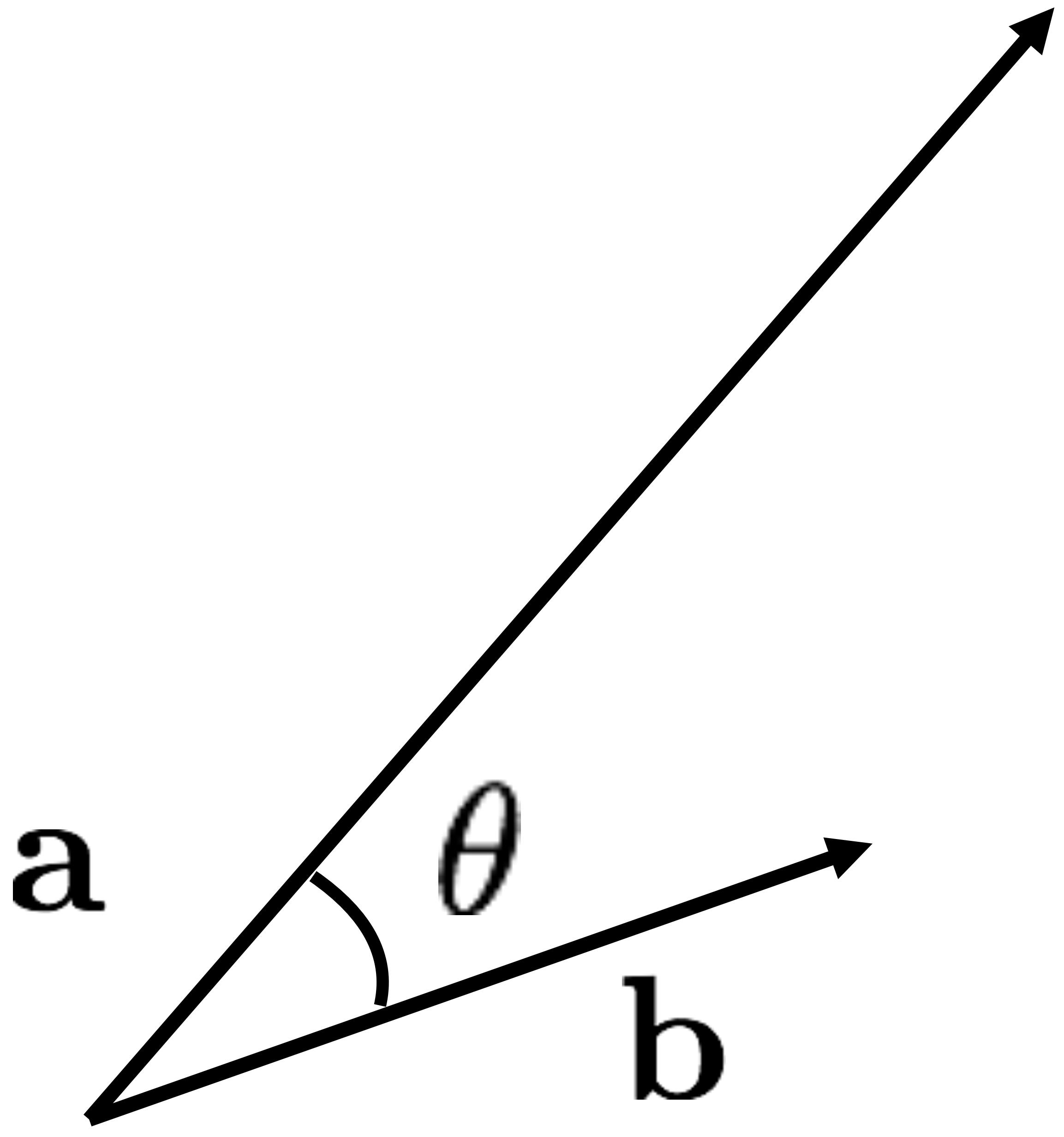
$$\mathbf{b} = \frac{\mathbf{a}}{||\mathbf{a}||}$$

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

Dot Product

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

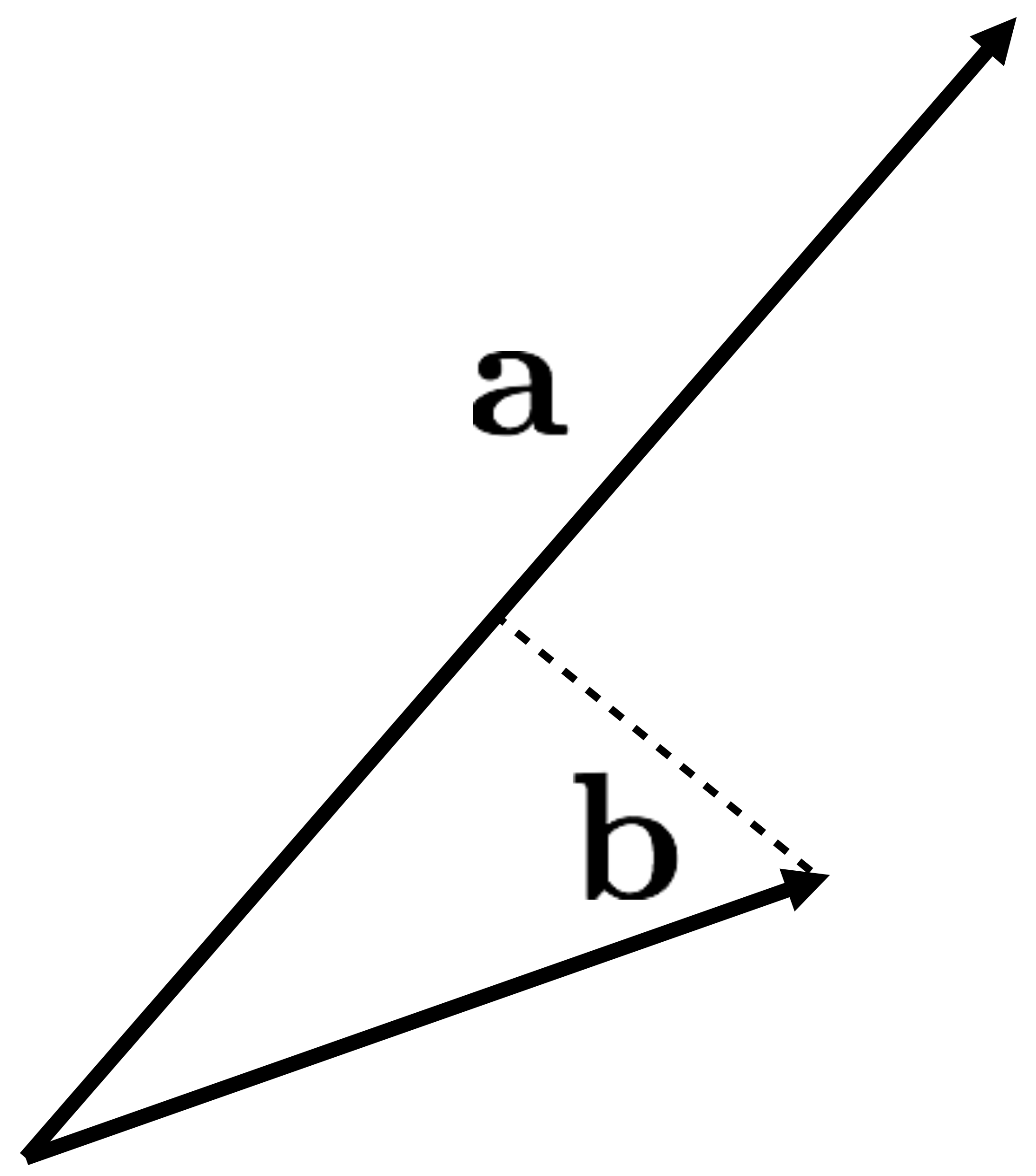
$$\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}|| \; ||\mathbf{b}|| \cos\theta$$



- 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

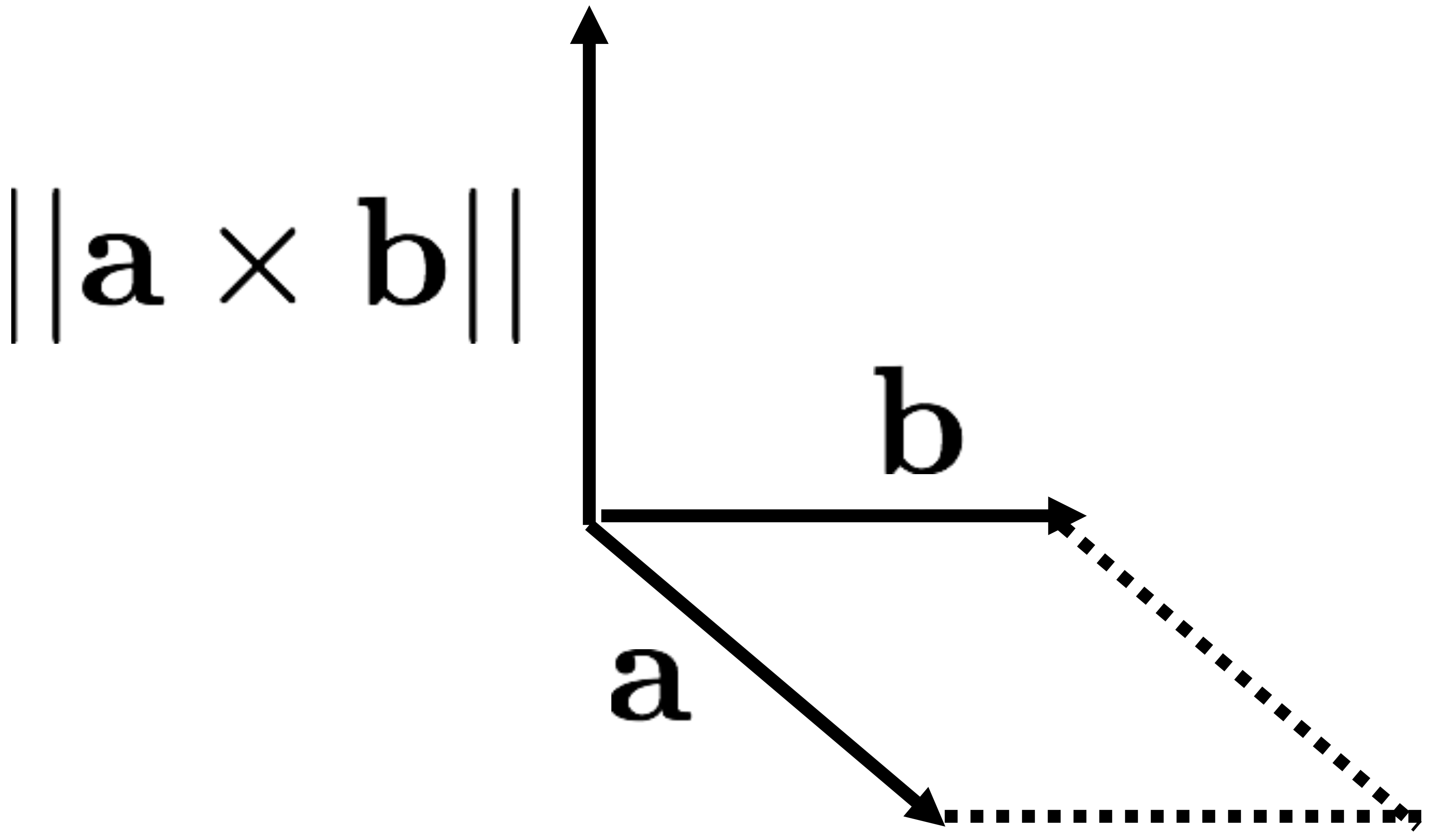
$$\mathbf{b} \rightarrow \mathbf{a} = ||\mathbf{b}|| \cos\theta = \frac{\mathbf{b} \cdot \mathbf{a}}{||\mathbf{a}||}$$

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

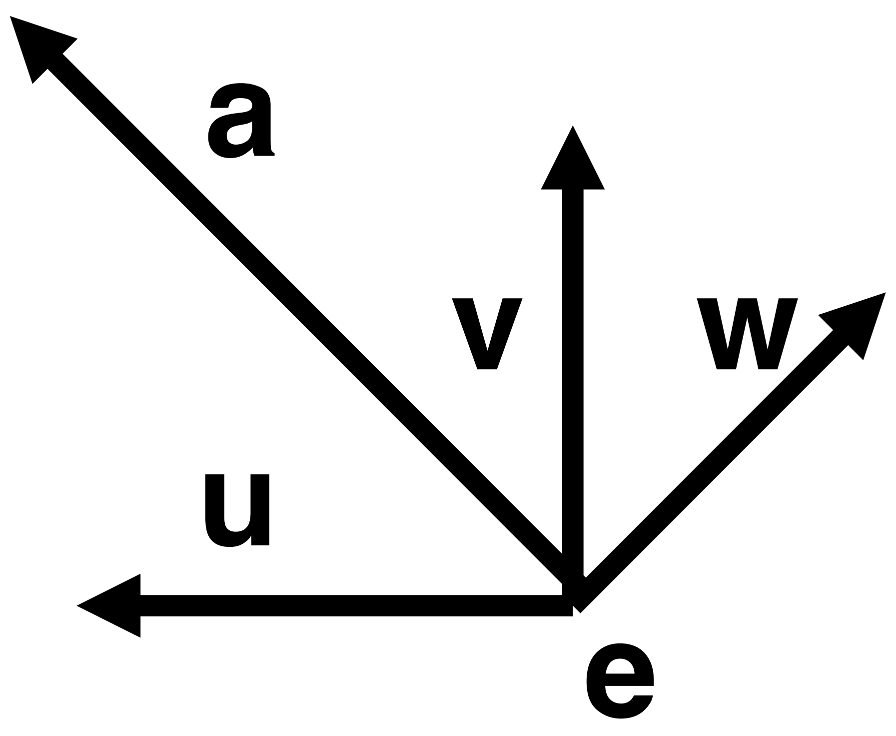
$$||\mathbf{u}|| = ||\mathbf{v}|| = 1,$$
$$\mathbf{u} \cdot \mathbf{v} = 0$$

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: $\mathbf{w} = \mathbf{u} \times \mathbf{v}$

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

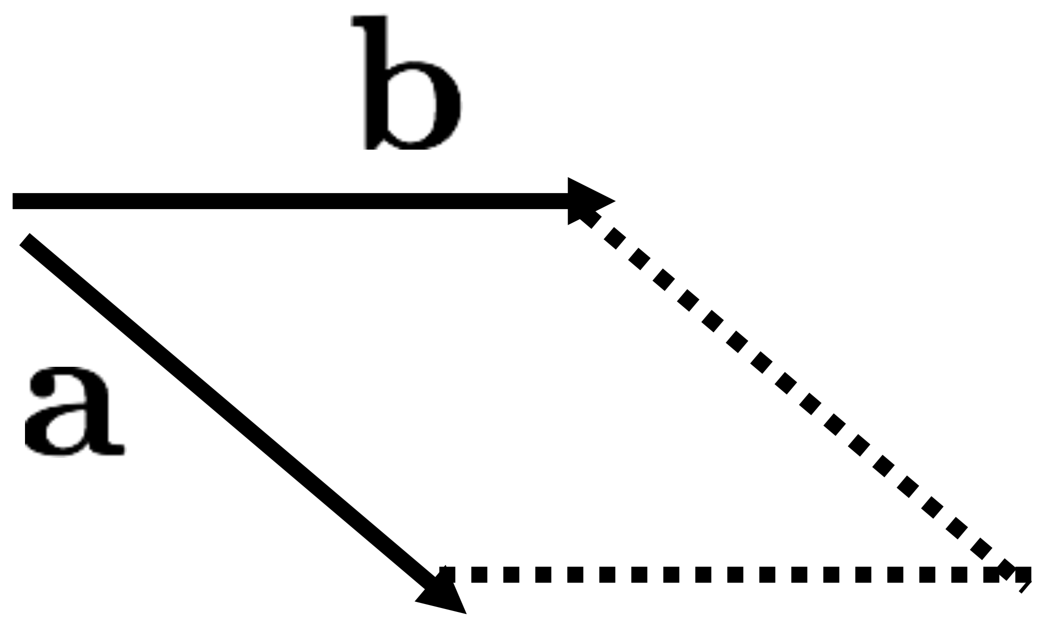
Matrices

Overview

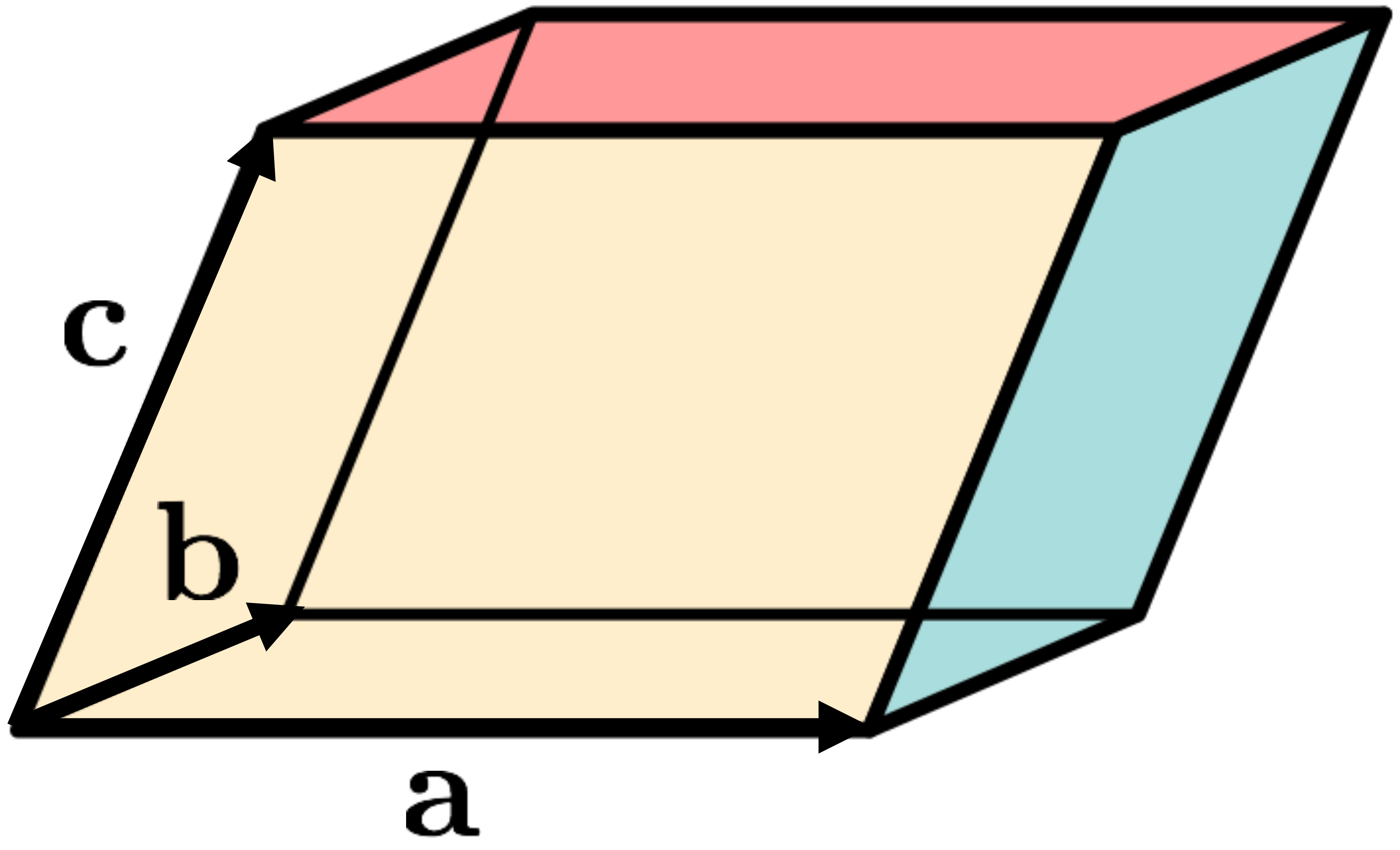
- Matrices will allow us to conveniently represent and ally transformations on vectors, such as translation, scaling and rotation
- Similarly to what we did for vectors, we will briefly overview their basic operations

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

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

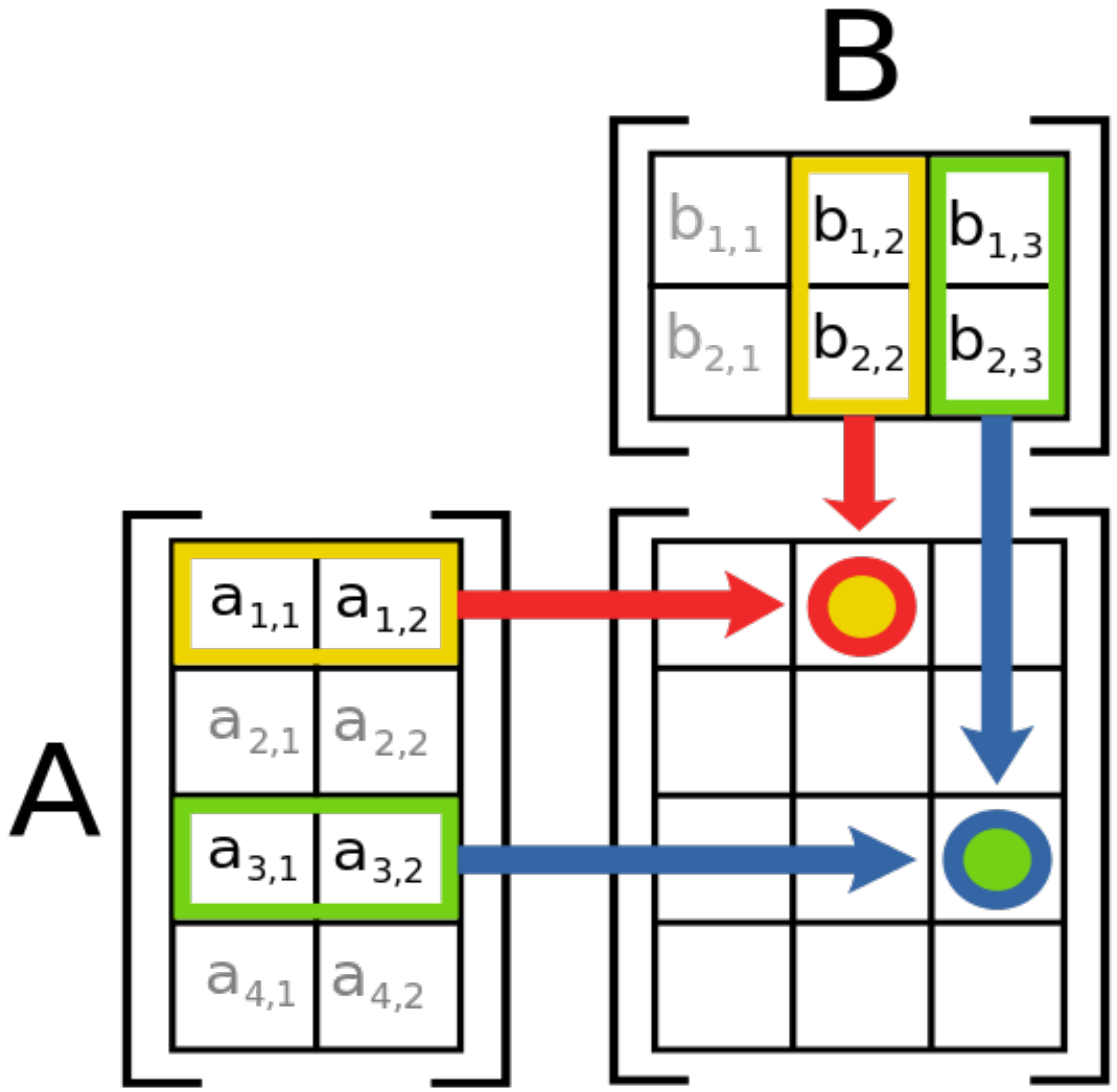
Matrix Product

- 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):

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

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



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:

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

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;
```

Linear Systems

- Direct Methods
  - LU-decomposition by Gaussian Elimination
  - Cholesky Algorithm for $A = LL^T$ (A is positive definite)

https://web.stanford.edu/class/cme324/saad.pdf

Linear Systems

- Iterative Methods
  - large sparse system
  - may not require any extra storage
  - One disadvantage is that after solving Ax = b1, one must start over again from the beginning in order to solve Ax = b2

https://web.stanford.edu/class/cme324/saad.pdf

Linear Systems

- Iterative Methods
  - Jacobi method
  - Gauss-Seidel method

They differs in how C and M are constructed!

- Given $Ax = b$, let $A = C - M$, where C is nonsingular and easily invertible.
- $Ax = b \Rightarrow (C - M)x = b \Rightarrow Cx = Mx + bx \Rightarrow x = Bx + c$, where $B = C^{-1}M$, $c = C^{-1}b$
- Suppose we start with an initial $x(0)$, then $x(1) = Bx(0) + c$ and $x(k + 1) = Bx(k) + c$

https://web.stanford.edu/class/cme324/saad.pdf

http://www.cs.nthu.edu.tw/~cchen/CS6331/ch2.pdf

References

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

Chapter 5

MIT Open Course: https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/