UNIVERSITETI I EVROPËS JUGLINDORE
УНИВЕРЗИТЕТ НА ЈУГОИСТОЧНА ЕВРОПА
SOUTH EAST EUROPEAN UNIVERSITY

Computer Graphics

CCS-601

Points and vectors

- Three important mathematical concepts, used in Computer Graphics are
  - Points
  - Vectors
  - Matrices

Points

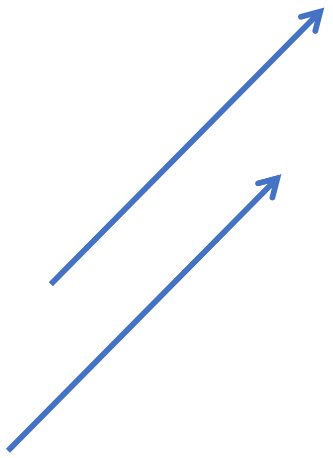
- Locations in space
- Must be referenced to something else
  - In CG this would be the coordinate system
  - The coordinate system has three vectors: X, Y and Z
  - The center of the coordinate system is an arbitrary point we have agreed upon
- Are there any other reference systems you can think of?
- Mathematically, a point will be defined as:

$$\begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix} = [x, y, z, 1]^T$$

Vectors

- By definition, vectors have their magnitude (length) and direction
- We usually draw them as arrows
- We denote vectors with lowercase letters (a, b, c)
  - The length of a vector is represented as |a|
  - A unit vector is a vector with a length of 1
- Mathematically, a vector is represented as

$$\boldsymbol{a} = \begin{vmatrix} x \\ y \\ z \\ 0 \end{vmatrix} = [x, y, z, 0]^T$$



Length of a vector

- You can calculate the length of a vector using the following formula:

$$|a| = \sqrt{x^2 + y^2 + z^2}$$

- E.g., the length of the vector [3, 2, 1, 0] will be:

$$|a| = \sqrt{3^2 + 2^2 + 1^2} = \sqrt{9 + 4 + 1} = \sqrt{14}$$

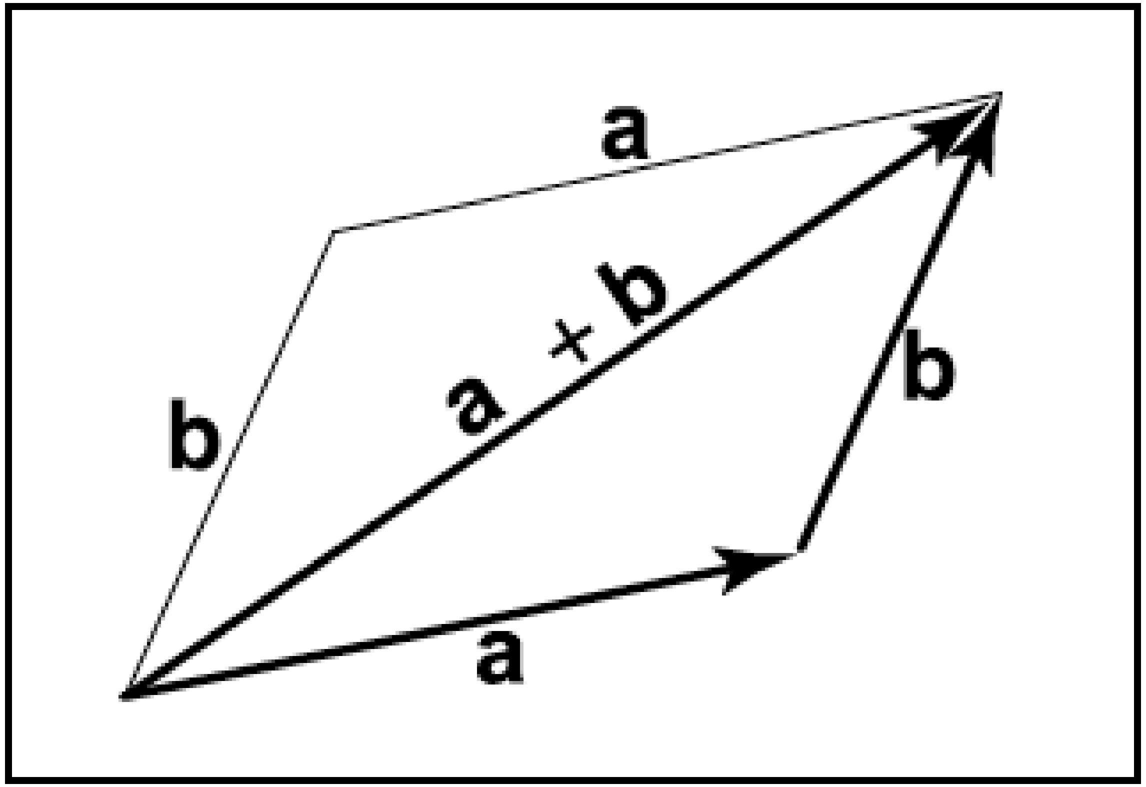
Operations on vectors - Adding

- Vectors are added using the parallelogram rule

$$\begin{vmatrix} x1 \\ y1 \\ z1 \\ 0 \end{vmatrix} + \begin{vmatrix} x2 \\ y2 \\ z2 \\ 0 \end{vmatrix} = \begin{vmatrix} x1 + x2 \\ y1 + y2 \\ z1 + z2 \\ 0 \end{vmatrix}$$

- The commutative rule applies:

a + b = b + a

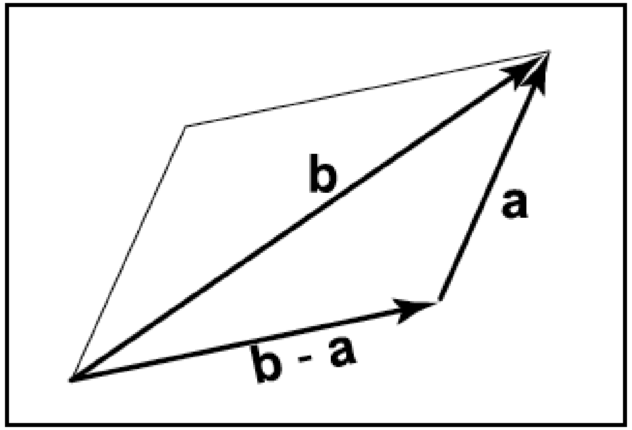


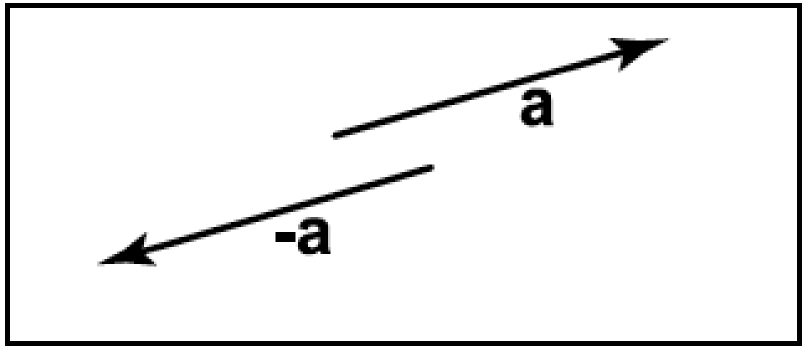
Operations on vectors - subtracting

- Similar to addition, we use the same rule

c = b – a = -a + (b)

$$\begin{vmatrix} x1 \\ y1 \\ z1 \\ 0 \end{vmatrix} - \begin{vmatrix} x2 \\ y2 \\ z2 \\ 0 \end{vmatrix} = \begin{vmatrix} x1 - x2 \\ y1 - y2 \\ z1 - z2 \\ 0 \end{vmatrix}$$

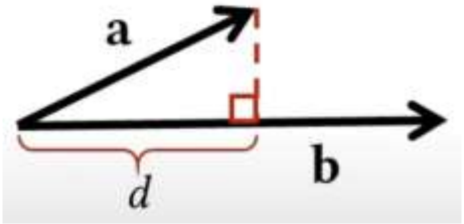




Operations on vectors – dot product

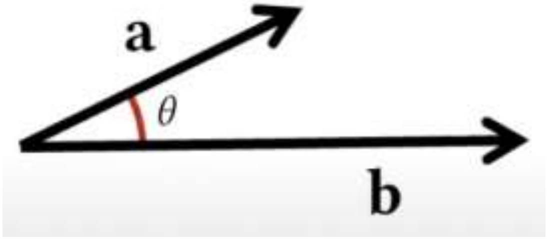
- One way to multiply two vectors

$$\boldsymbol{a} \circ \boldsymbol{b} = a_x b_x + a_y b_y + a_z b_z$$



$$\boldsymbol{d} = (\boldsymbol{a} \circ \boldsymbol{b}) \,/\, ||\,\boldsymbol{b}\,||$$

$$\boldsymbol{a} \circ \boldsymbol{b} = |a||b| \cos\theta$$

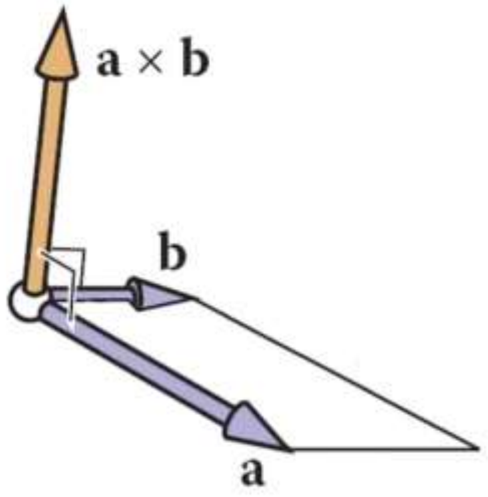


- In Computer Graphics dot product is used to:
  - Lighting and shading
  - Projection
  - Reflection

Operations on vectors – cross product

- Two vectors form a plane
- The cross product of two vectors will give us a new vector, perpendicular to these two vectors

$$\boldsymbol{a}\ x\ \boldsymbol{b} = [a_y b_z - b_y a_{z,} a_z\, b_x - a_x b_z\, , a_x b_y\ - b_x a_y\,]$$



- Cross product is used in Computer Graphics to:
  - Calculate normals in shading algorithms, colision detection etc.
  - Transformations and rotations
  - Texture mapping
  - Camera setup (e.g., look up)

Matrices

- We will use them much more in future lectures

$$A = \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix}$$

$$A\ x\ \boldsymbol{b} = \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} \begin{vmatrix} b_x \\ b_y \\ b_z \end{vmatrix} = \begin{vmatrix} a_{00}b_x & a_{01}b_y & a_{02}b_z \\ a_{10}b_x & a_{11}b_y & a_{12}b_z \\ a_{20}b_x & a_{21}b_y & a_{22}b_z \end{vmatrix}$$

Representation in Three.js

- Vectors / Points

```
const a = new THREE.Vector3( 0, 1, 0 );
```

- Matrices

```
const m = new Matrix3();
m.set(
        11, 12, 13,
        21, 22, 23,
        31, 32, 33
        );
```

Some helpful Three.js methods

- Adding two vectors:
  - a.add(b)
  - c.addVectors(a, b)
- Dot product
  - a.dot(b)
- Cross product
  - a.cross(b)
  - c.crossVectors(a, b)

Affine transformations in 2D

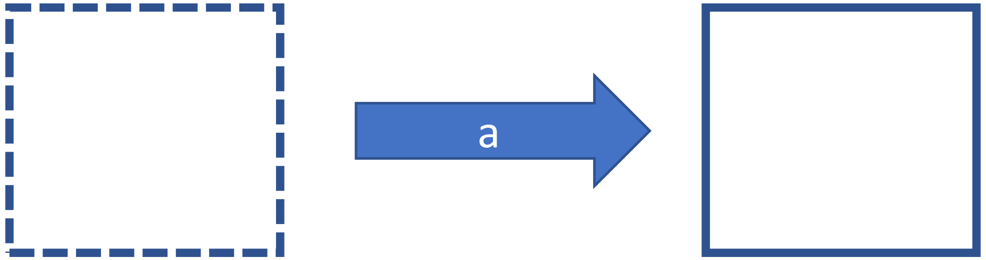
- In Computer Graphics we often want to apply some kind of transformations to our objects
- We will discuss the following transformations
    - Translate
    - Scale
    - Rotate
- In these three transformations the shape of the object changes but lines remain lines and parallel lines will still remain parallel

Translation

- In this basic translation, we move every point of an object in a constant <u>distance</u> in a given <u>direction</u>!
- In this case we have a translation vector ***a***

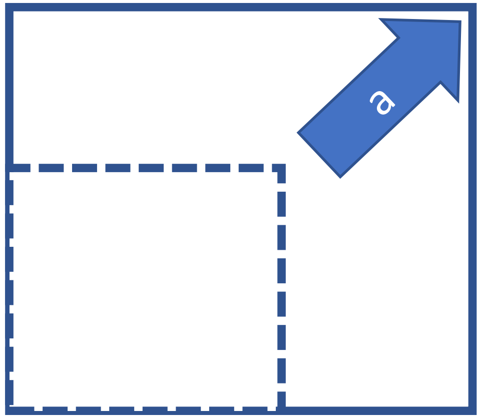
$$P' = P + \boldsymbol{a}$$

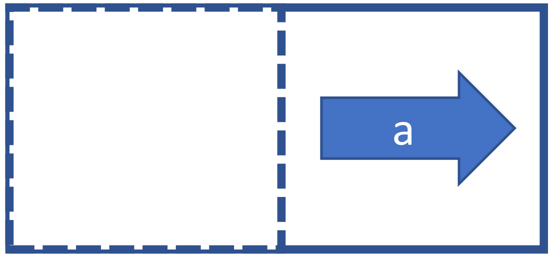
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + a_x \\ y + a_y \end{bmatrix}$$



Scaling

- A scaling transformation will change the size of the object
- Uniform scale is when we scale in all dimensions for the same amount





$$P' = P * \boldsymbol{s}$$

Uniform scaling

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x * s \\ y * s \end{bmatrix}$$

Non-uniform scaling

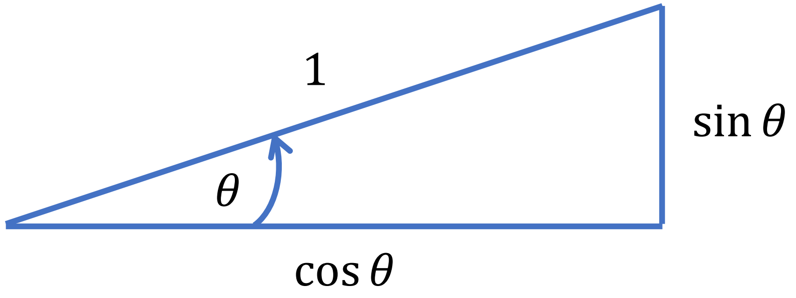
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x * \ s_x \\ y * s_y \end{bmatrix}$$

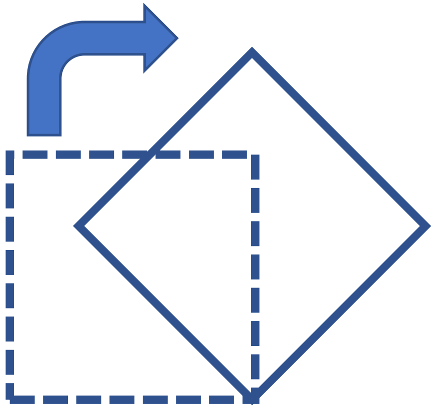
Matrix form

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x + 0 \\ 0 + s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotation

- Rotates the object for a given angle
- The new matrix will be **<u>orthogonal</u>**





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

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

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

Order of operations

- Up to this point we have been mostly ignoring the order of operations.
- We have performed translation, rotation and scaling without worrying about the order.
- The order of transformation matters.
- Three.js uses the following order to apply the transforms on an object:

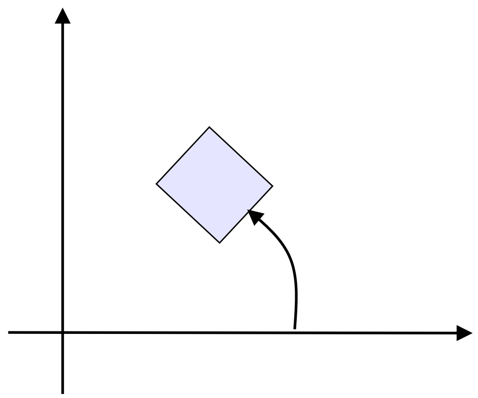
scale, rotate, translate

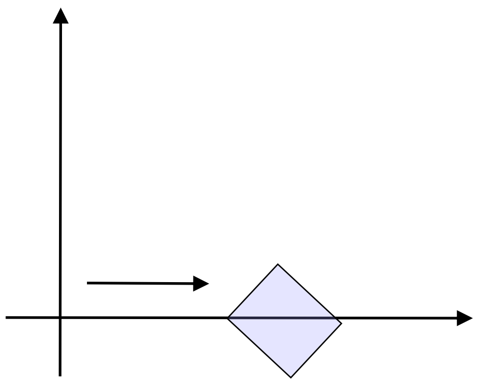
Order of operations

$$A \cdot B \neq B \cdot A$$

The sequence of transformations generally is not commutative. We can see this with translation and rotation.

$$Translation \cdot Rotation \neq Rotation \cdot Translation$$





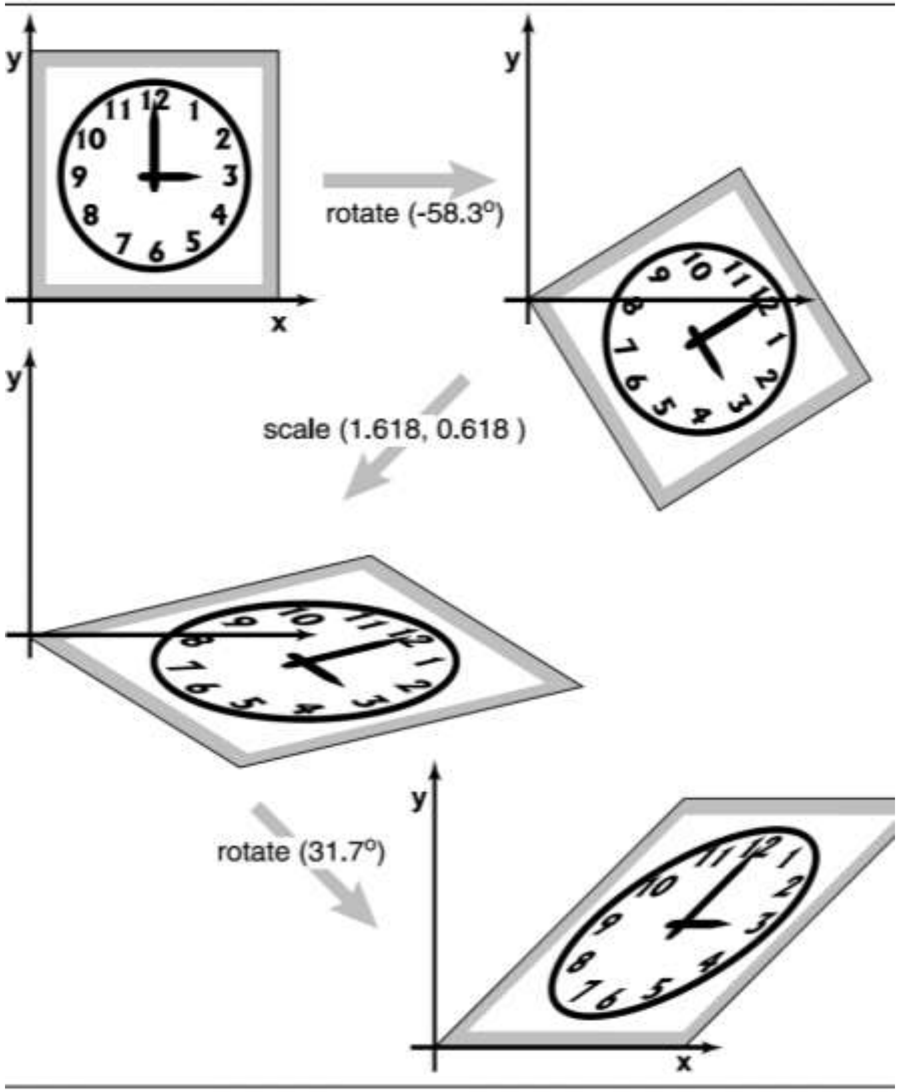
Transformations can be combined

- In theory any matrix can be represented as a combination of rotation and scale (R and S)
  - In practice this is a bit difficult though (SVD)
- E.g., for the skew transformation we can have:

p’ = RSRp

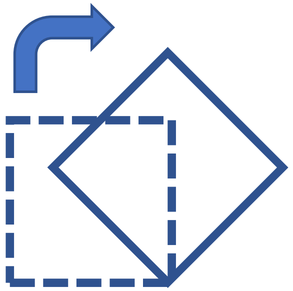
RSR = M

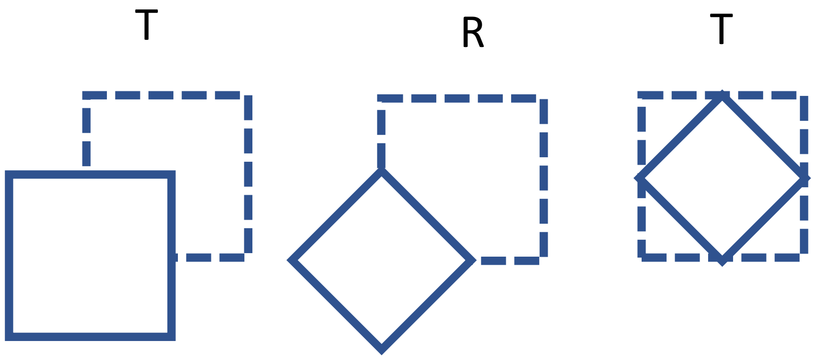
P’ = Mp



Rotation and translation

- Usually applied when we want to change the pivot (origin of the rotation)
- E.g., p’ = TRTp = Mp





Homogenous coordinates - translate

- We already covered this in last lecture
- The homogenous coordinate for translation is

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

Homogenous coordinates - scale

- Scale is a diagonal matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Homogenous coordinates - rotation

- Rotation is an orthogonal matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3D Homogenous coordinates - translate

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Homogenous coordinates - scale

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

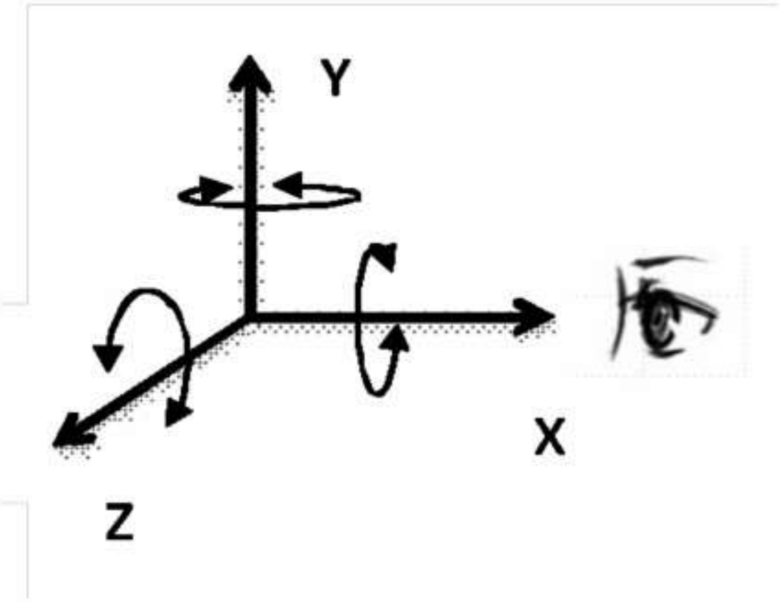
3D Homogenous coordinates - rotate

- Three matrices for rotate: rotate x, rotate y and rotate z

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

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

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



Hierarchy of objects

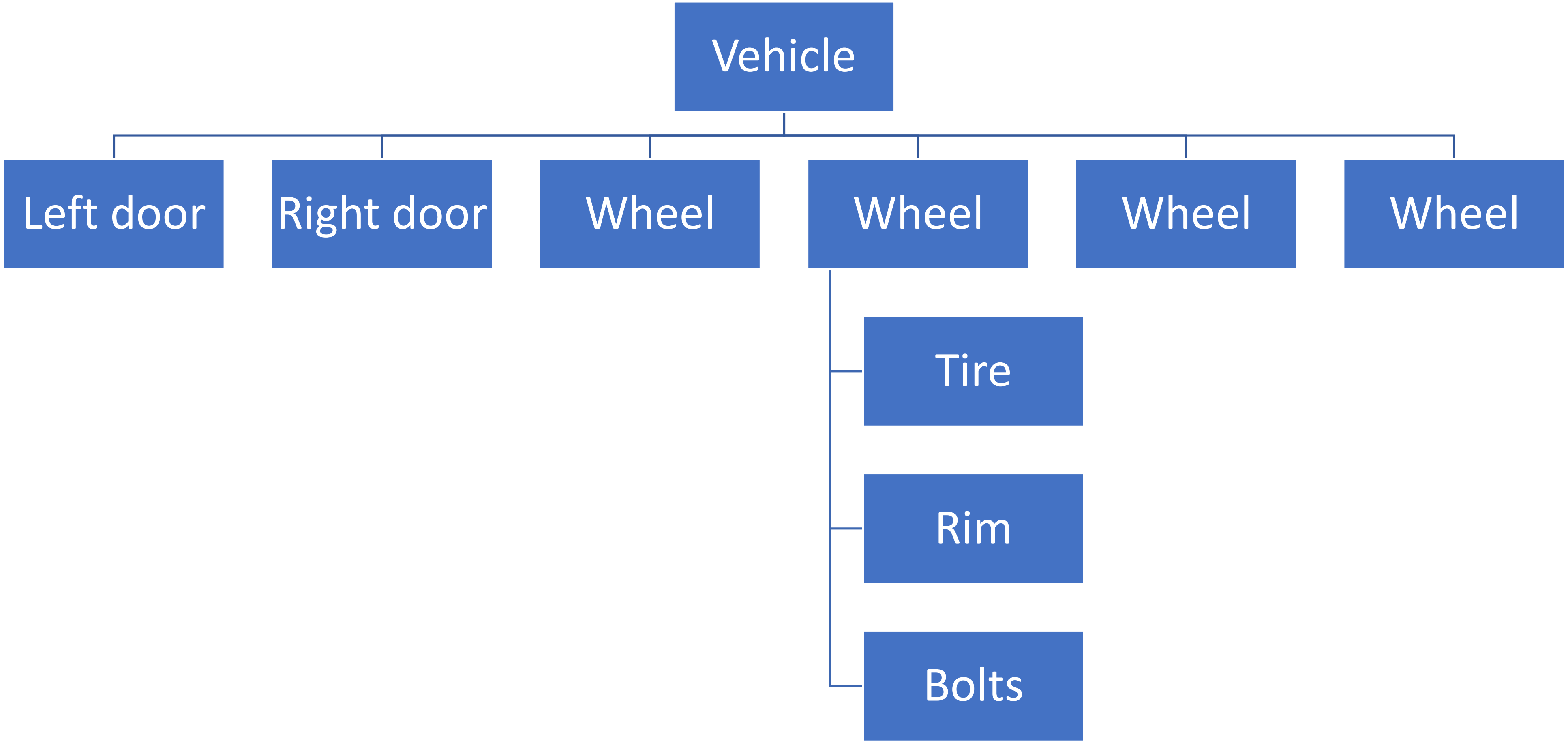
- Sometimes we want to position objects and the rotate accordingly.
- But, threejs rotates first then positions.
- Three JS allows creation of new objects which can be in itself transformed as wished.
- An example:

```
var block = new THREE.MeshBasicMaterial( {color: 0x00ff00} );
block.position.x = 4;

var clockHand = new THREE.Group();
clockHand.add(block);

clockHand.rotation.x = -70 * Math.PI / 180;
scene.add(clockHand);
```

Hierarchy of objects

- THREE.Group to give us access to a few extra transforms in the chain.
- However, THREE.Group is designed for another purpose that is extremely useful.
- What THREE.Group does is create a parent-child relationship between two objects.
- Once an object is a child of another object, that child is affected by whatever is done to the parent.

Hierarchical model of a vehicle



Transformations in Three.js

- Three.js supports all the above mentioned transformations
- You can either apply them directly on the object:
  - camera.position.z = 5;
  - cube.rotation.x += 0.01;
  - cube.rotation.y += 0.01;
- Or you can use three.js Object3D:
  - .rotateX (rad : Float)
  - .rotateY (rad : Float)
  - .translateX (distance : Float)
  - .translateY (distance : Float)
  - …