Introduction to Computer Graphics (CS360A)

Instructor: Soumya Dutta

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur (IITK)

email: soumyad@cse.iitk.ac.in

Acknowledgements

- A subset of the slides that I will present throughout the course are adapted/inspired by excellent courses on Computer Graphics offered by Prof. Han-Wei Shen, Prof. Wojciech Matusik, Prof. Frédo Durand, Prof. Abe Davis, and Prof. Cem Yuksel

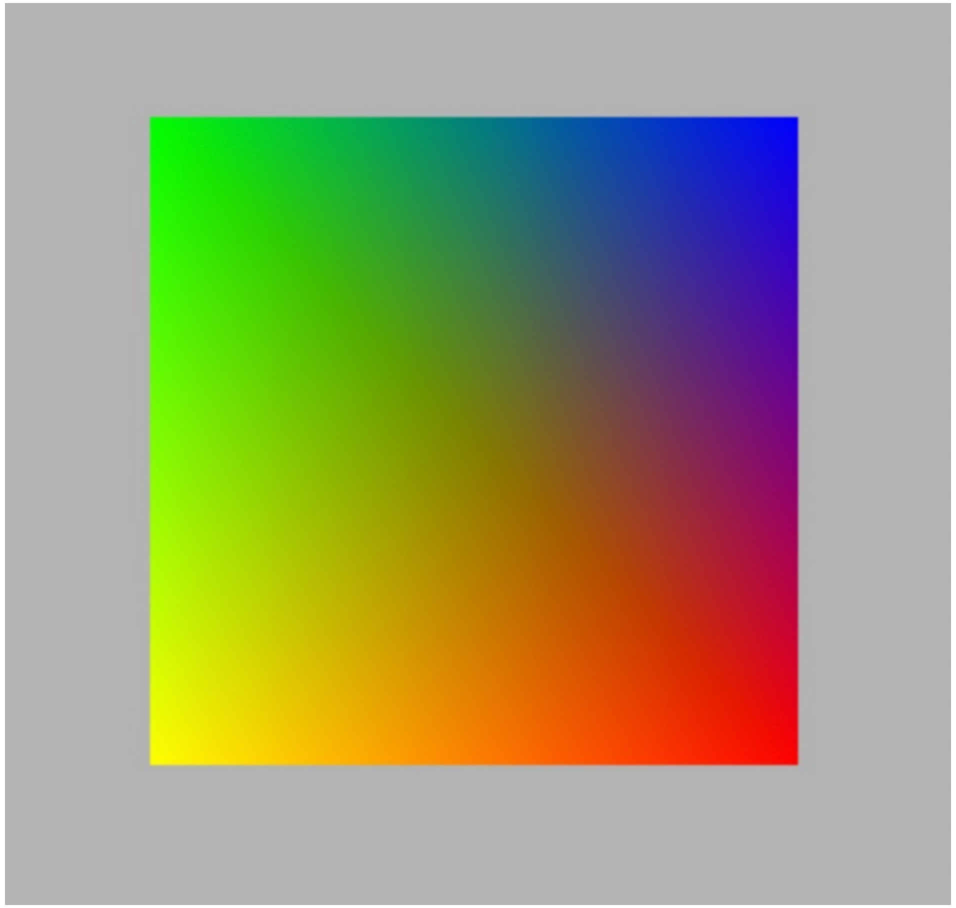
Draw using gl.DrawElements()


```
// buffer for point locations
const bufData = new Float32Array([
  0.5, 0.5,
  -0.5, 0.5,
  0.5, -0.5,
  -0.5, -0.5,
]);
```

```
// buffer for point indices
const indices = new Uint16Array([
  0, 1, 2,
  1, 2, 3]);
```

```
// buffer for the point colors
const colors = new Float32Array([
  0.0, 0.0, 1.0,
  0.0, 1.0, 0.0,
  1.0, 0.0, 0.0,
  1.0, 1.0, 0.0,
]);
```

```
// A single draw call is sufficient
gl.drawElements(gl.TRIANGLES, 6, gl.UNSIGNED_SHORT, 0);
```



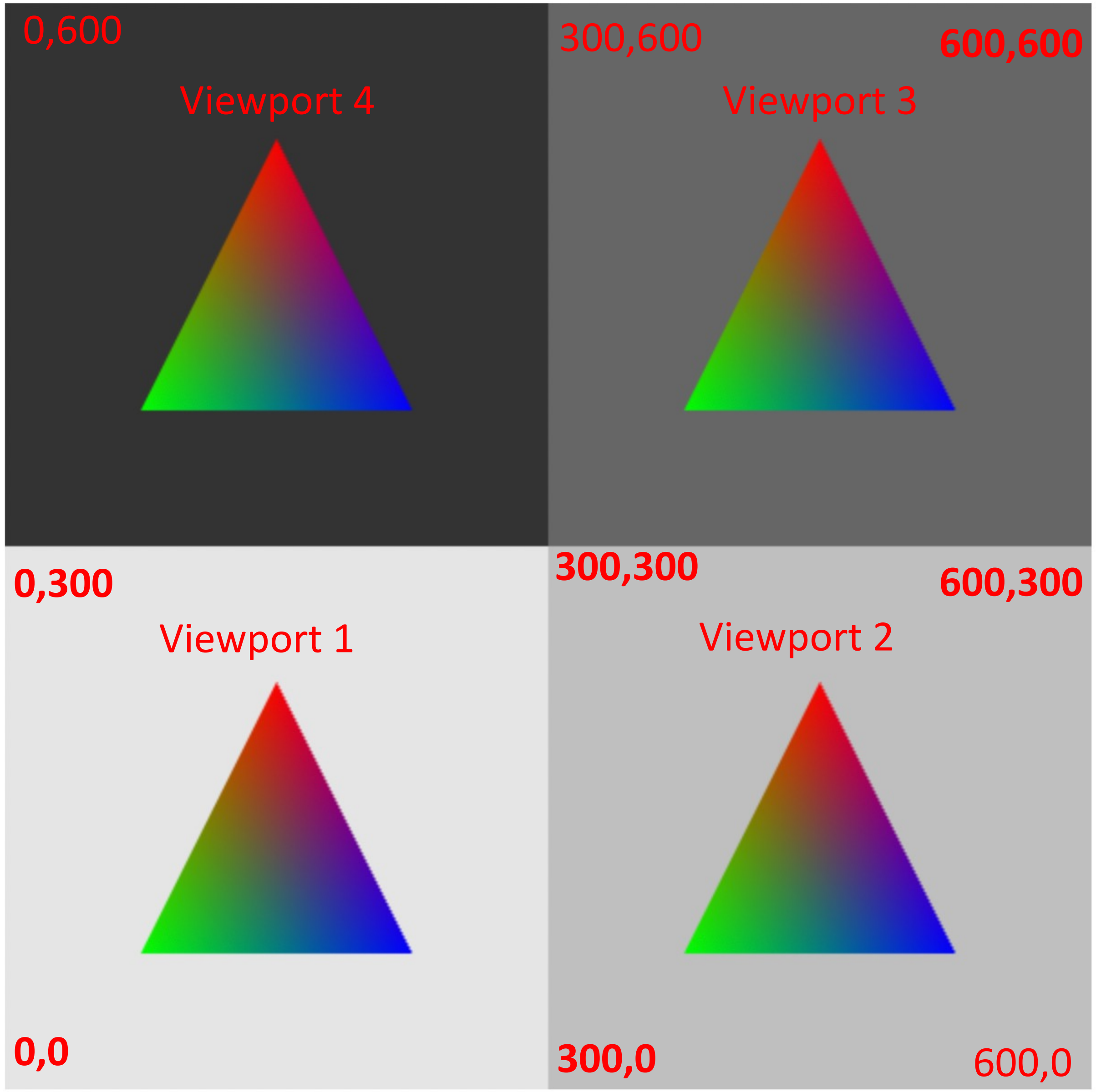
Viewport

IITK CS360A

- Viewport indicates the current drawing area on the screen space coordinate system
- Specifies the affine transformation of x and y from normalized device coordinates to window coordinates
- *gl.viewportWidth = canvas.width;*
- *gl.viewportHeight = canvas.height;*
- *gl.viewport(0,*

```
gl.viewport(0,
            0,
            gl.viewportWidth,
            gl.viewportHeight);
```

Multiple Viewports



Multiple Viewports

- Divide the canvas area into multiple drawing areas (viewports)
- You need to enable **SCISSOR_TEST** to be able to use multiple viewports
  - *gl.enable(gl.SCISSOR_TEST);*

```
////////////////////////////////////////
// Lower left viewport area
gl.viewport(0, 0, 300, 300);
gl.scissor(0, 0, 300, 300);
gl.clearColor(0.9, 0.9, 0.9, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
gl.drawArrays(gl.TRIANGLES, 0, 3);

////////////////////////////////////////
// Lower right viewport area
gl.viewport(300, 0, 300, 300);
gl.scissor(300, 0, 300, 300);
gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

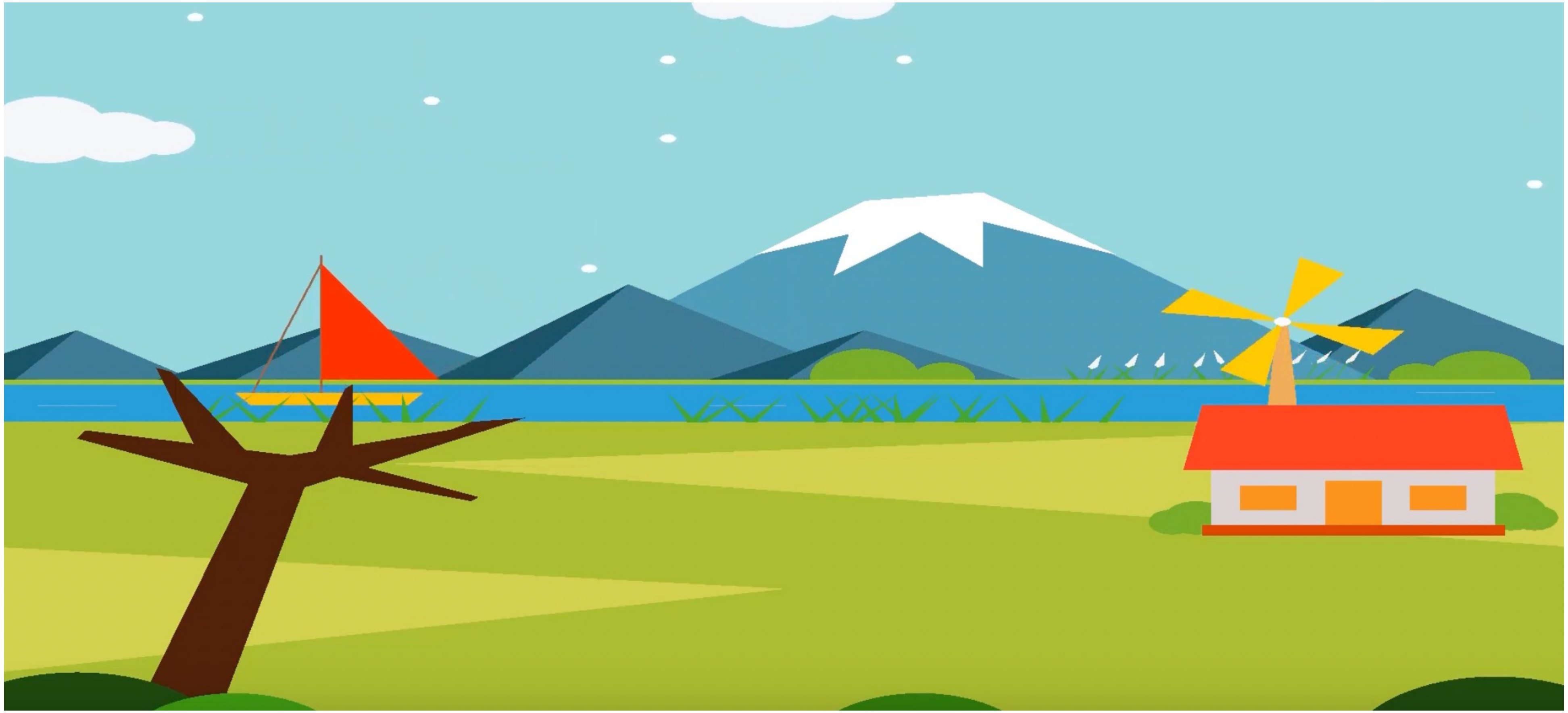
```
////////////////////////////////////////
// upper right viewport area
gl.viewport(300, 300, 300, 300);
gl.scissor(300, 300, 300, 300);
gl.clearColor(0.4, 0.4, 0.4, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
gl.drawArrays(gl.TRIANGLES, 0, 3);

////////////////////////////////////////
// upper left viewport area
gl.viewport(0, 300, 300, 300);
gl.scissor(0, 300, 300, 300);
gl.clearColor(0.2, 0.2, 0.2, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Transformations

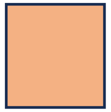
IITK CS360A

- Let’s say, we want to build a (2D) scene like the following

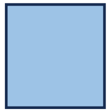


Affine Transformations

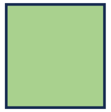
- Translation



- Rotation



- Scale

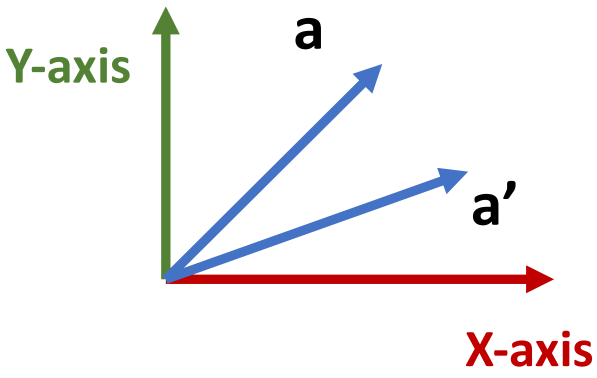


- Skew

  - A combination of rotation and scale

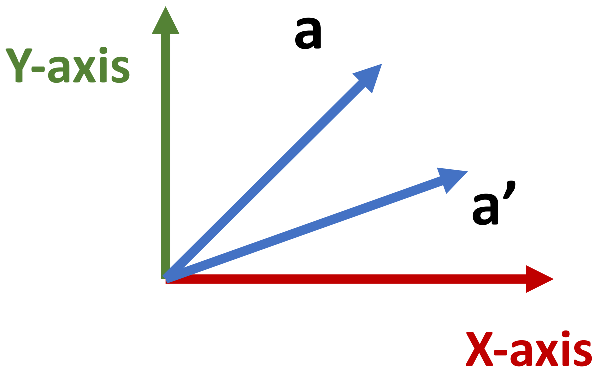
- We will learn about Translation, Rotation, and Scale operations in 2D and 3D
- Affine Transformation
  - Linear combination
  - All lines will remain as lines after affine transformation
  - All parallel lines will remain parallel after affine transformation

Translation



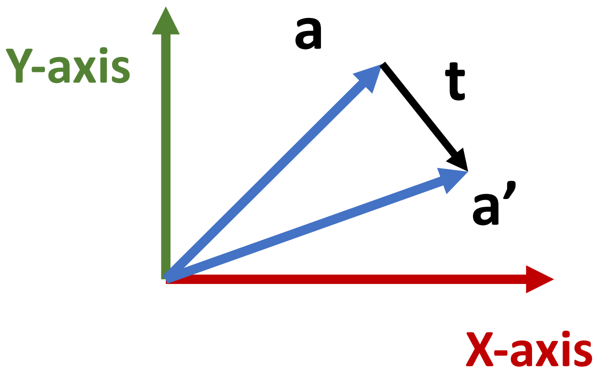
- Translate vector **a** to **a’**

Translation



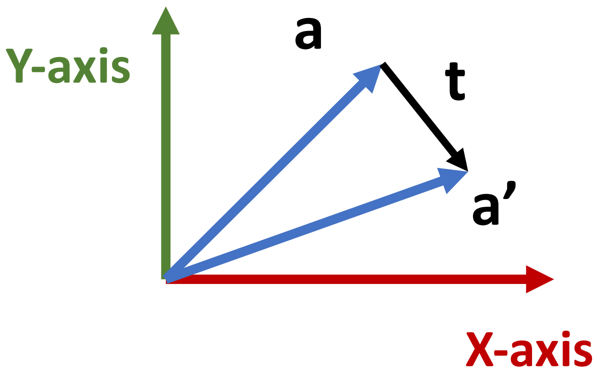
- Translate vector **a** to **a'**
- We can do this by adding vector **t** with **a**

Translation



- Translate vector **a** to **a’**
- We can do this by adding vector **t** with **a**

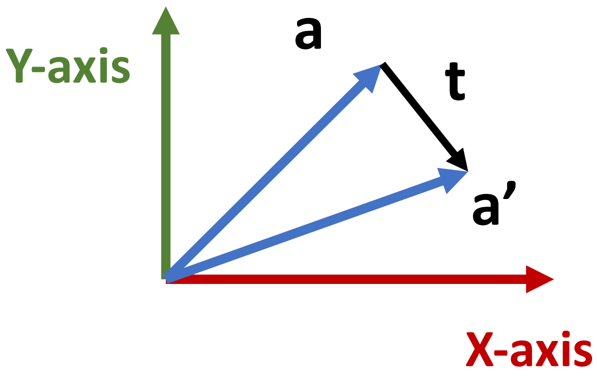
Translation



a’ = a + t

- Translate vector **a** to **a’**
- We can do this by adding vector **t** with **a**

Translation

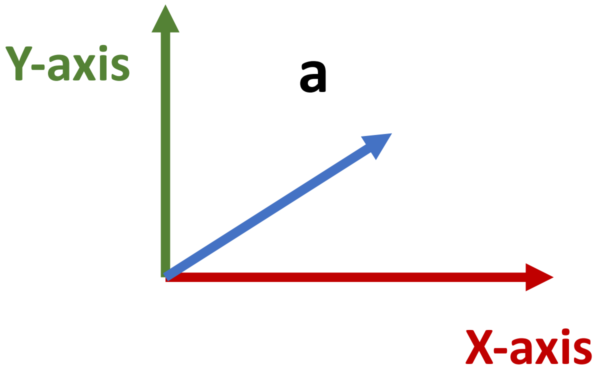


- Translate vector **a** to **a’**
- We can do this by adding vector **t** with **a**

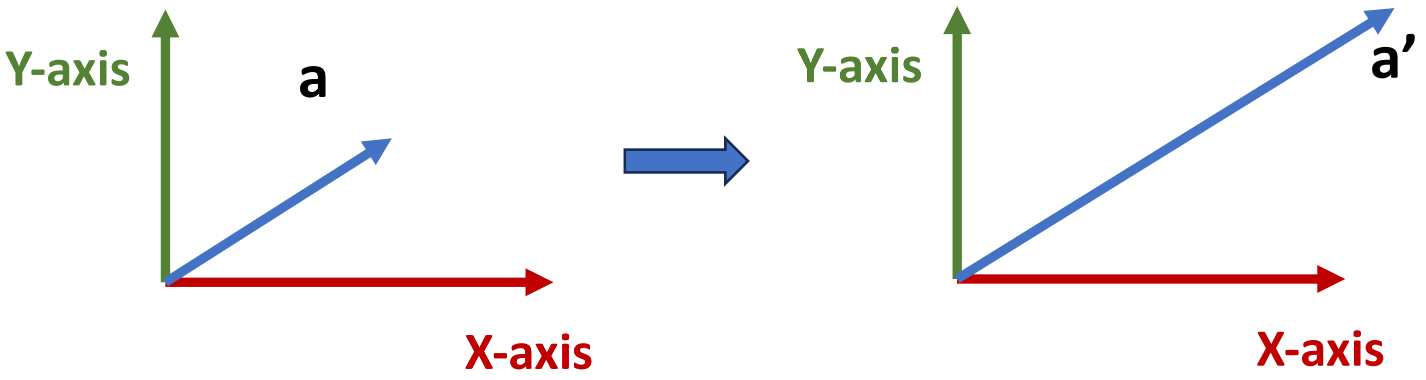
a’ = a + t

$$\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = \begin{bmatrix} a_x & + & tx \\ a_y & + & ty \end{bmatrix}$$

Scaling



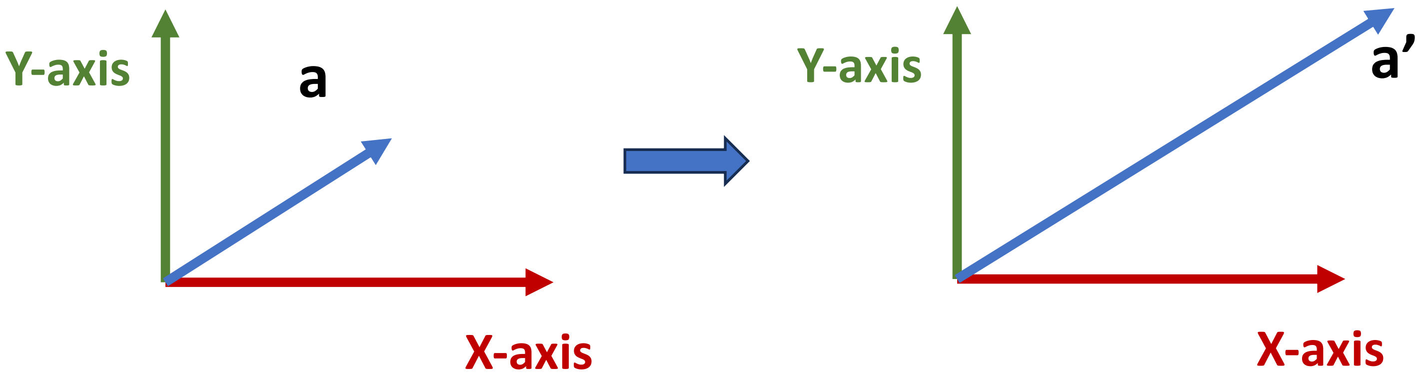
Scaling



- Scale vector **a** to **a’**

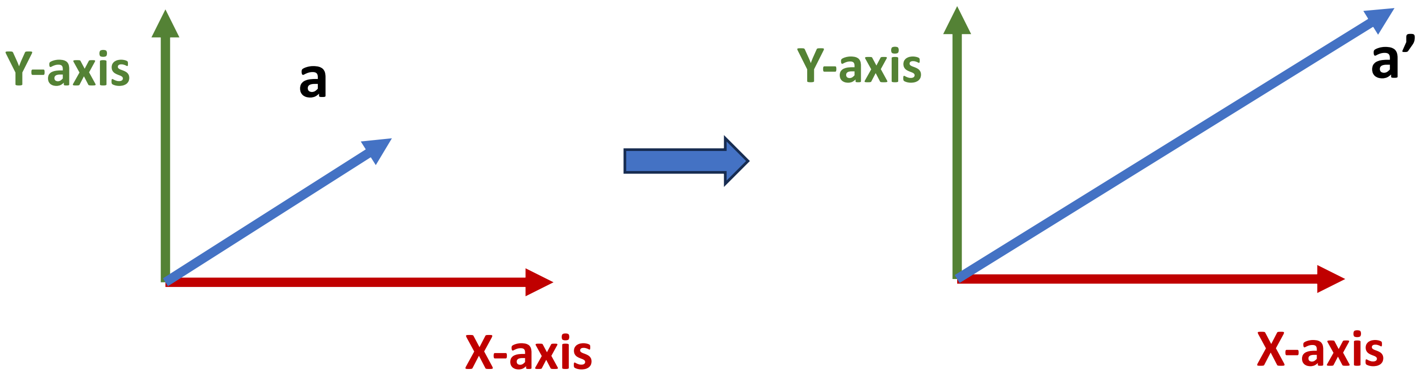
Scaling

IITK CS360A



- Scale vector **a** to **a’**
- We can do this by multiplying scalar s with vector **a**

Scaling

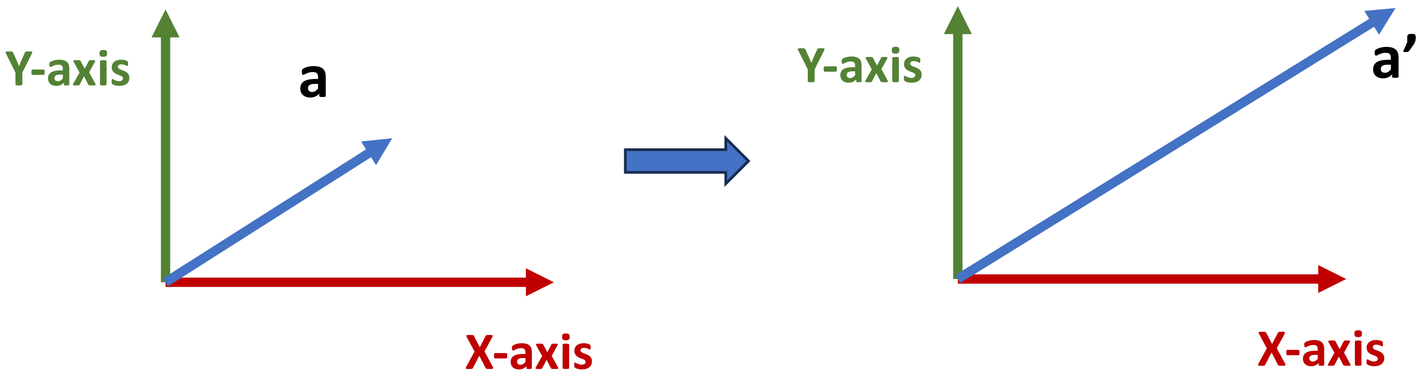


- Scale vector **a** to **a’**
- We can do this by multiplying scalar s with vector **a**

a’ = s **a**   Uniform scaling

Scaling

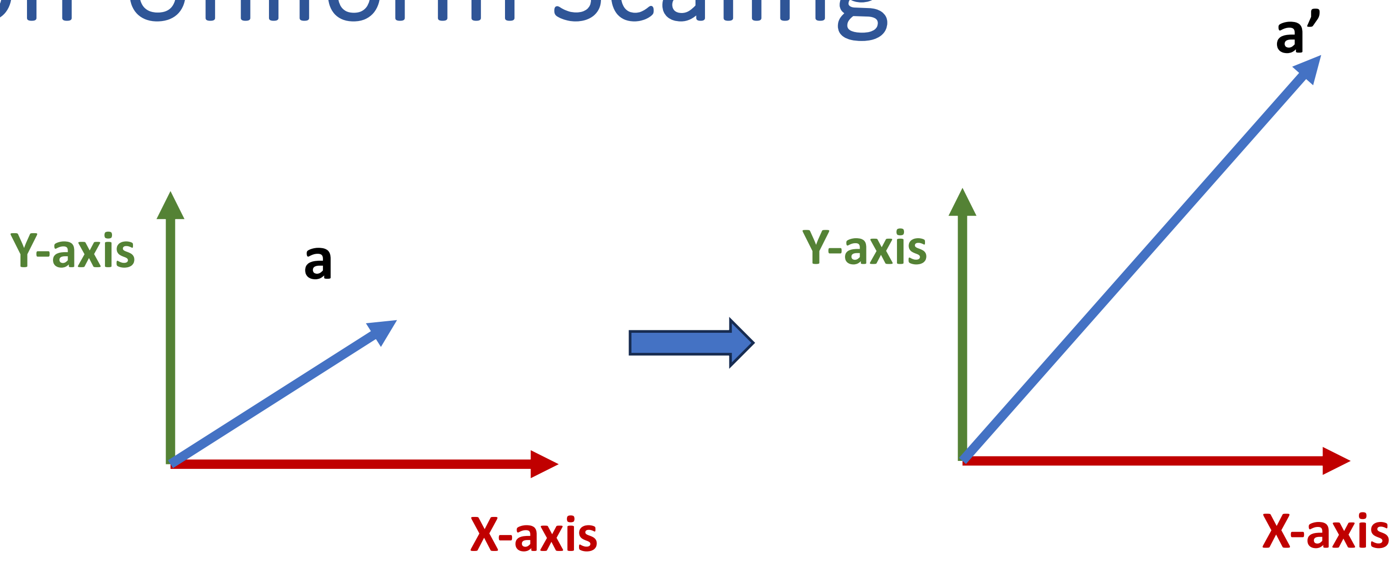
IITK CS360A



- Scale vector **a** to **a’**
- We can do this by multiplying scalar s with vector **a**

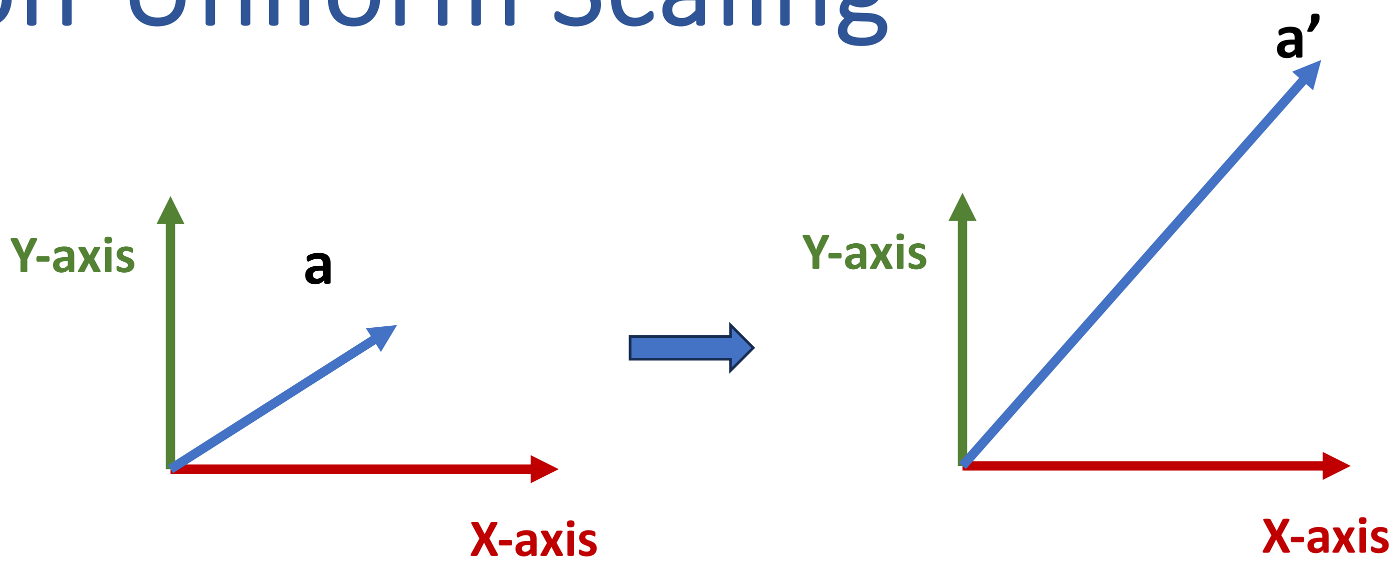
a’ = s **a** Uniform scaling $\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = \begin{bmatrix} sa_x \\ sa_y \end{bmatrix}$

Non-Uniform Scaling



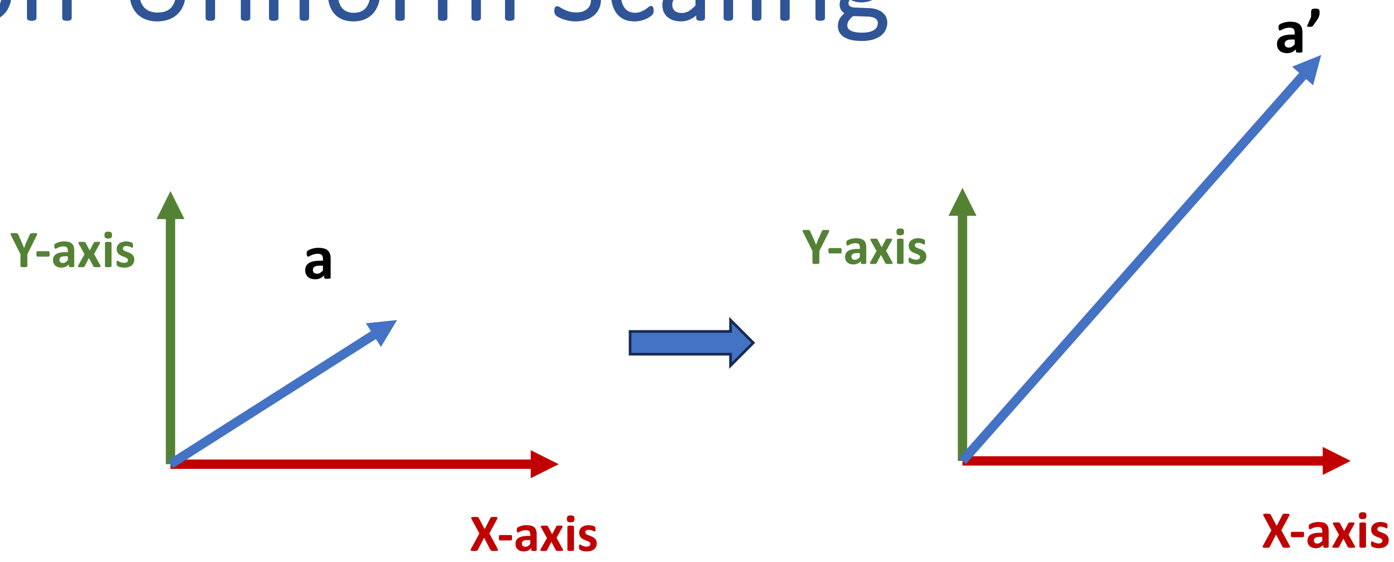
- Scale vector **a** to **a’**

Non-Uniform Scaling



- Scale vector **a** to **a’**
- We can do this by multiplying scalars s_x and s_y with vector **a**

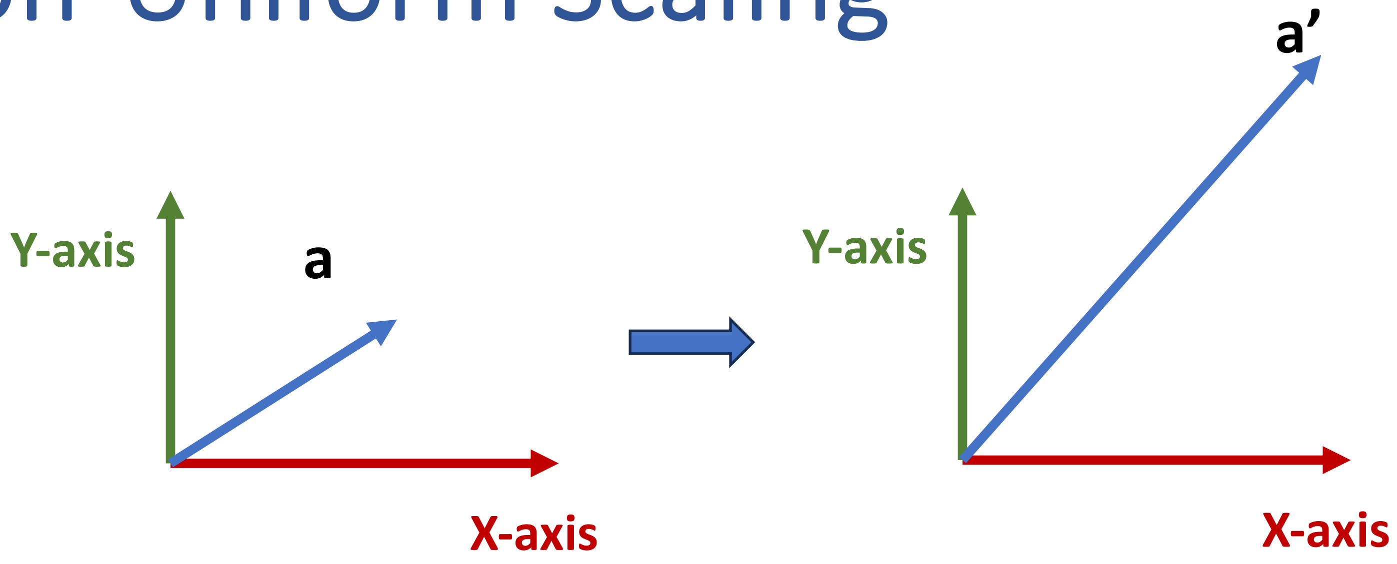
Non-Uniform Scaling



- Scale vector **a** to **a’**
- We can do this by multiplying scalars s_x and s_y with vector **a**

$\cancel{\mathbf{a}' = s\,\mathbf{a}}$ Non-Uniform scaling

Non-Uniform Scaling

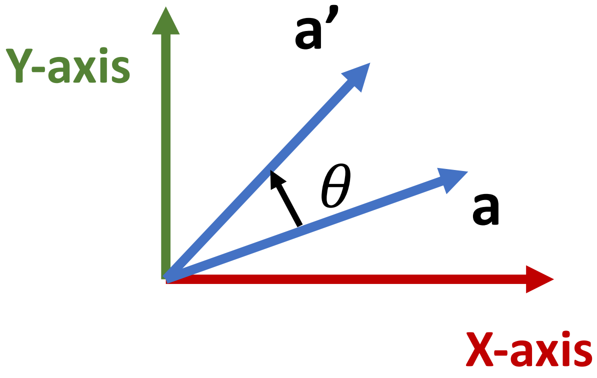


- Scale vector **a** to **a'**
- We can do this by multiplying scalars s_x and s_y with vector **a**

~~**a' = s a**~~ Non-Uniform scaling

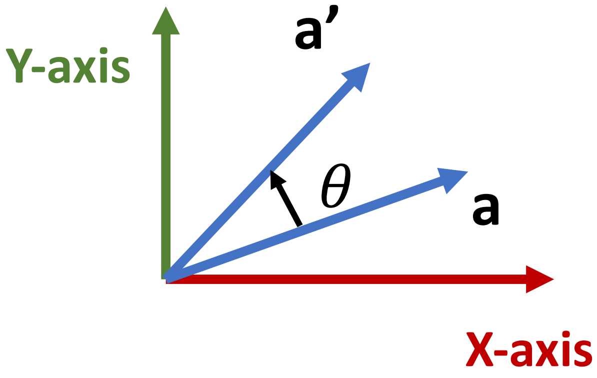
$$\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = \begin{bmatrix} s_x a_x \\ s_y a_y \end{bmatrix}$$

Rotation



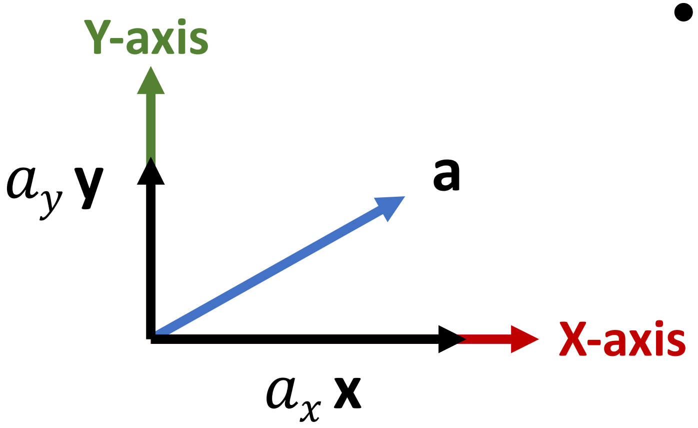
- Rotate vector **a** to **a’** counterclockwise by angle θ

Rotation



- Rotate vector **a** to **a’** counterclockwise by angle θ
- How do we find the transformation?
  - Not so straightforward as translation and scaling!

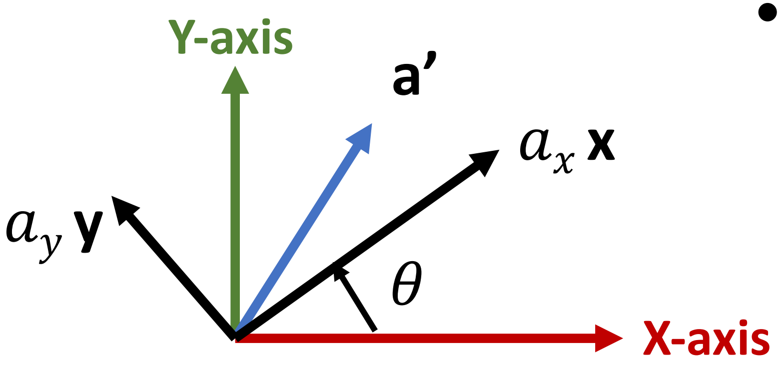
Rotation



$\mathbf{a} = a_x\,\mathbf{x} + a_y\,\mathbf{y}$

- Rotate vector **a** to **a’** counterclockwise by angle θ

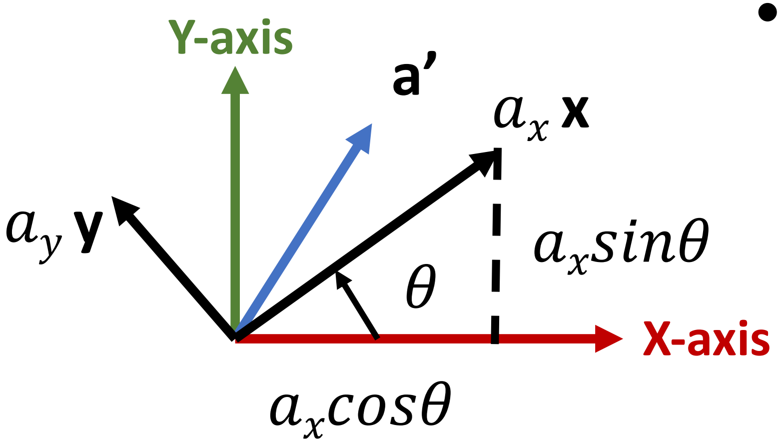
Rotation



- Rotate vector **a** to **a’** counterclockwise by angle θ

$$\mathbf{a} = a_x\,\mathbf{x} + a_y\,\mathbf{y}$$

Rotation

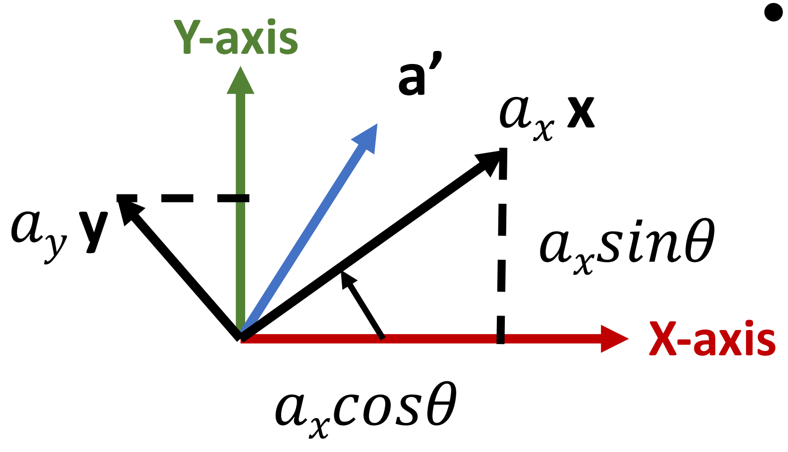


$$a_x \begin{bmatrix} cos\theta \\ sin\theta \end{bmatrix}$$

- Rotate vector **a** to **a’** counterclockwise by angle θ

a = a_x **x** + a_y **y**

Rotation

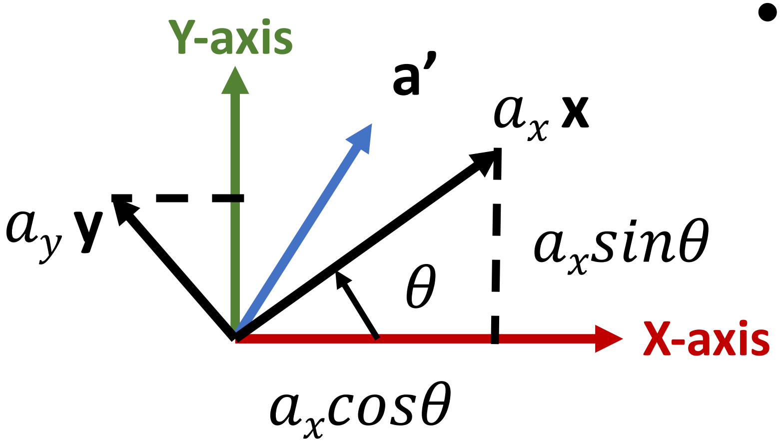


- Rotate vector **a** to **a'** counterclockwise by angle θ

a = a_x **x** + a_y **y**

$$a_x \begin{bmatrix} cos\theta \\ sin\theta \end{bmatrix} \qquad a_y \begin{bmatrix} -sin\theta \\ cos\theta \end{bmatrix}$$

Rotation



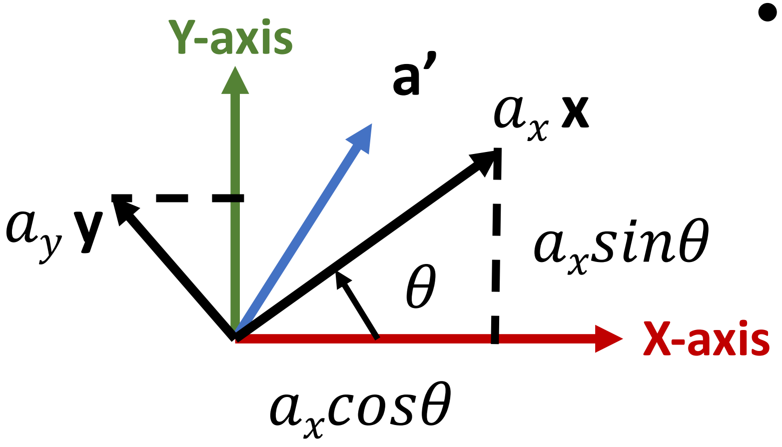
- Rotate vector **a** to **a'** counterclockwise by angle θ

a = a_x **x** + a_y **y**

$$\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = a_x \begin{bmatrix} cos\theta \\ sin\theta \end{bmatrix} + a_y \begin{bmatrix} -sin\theta \\ cos\theta \end{bmatrix}$$

$$\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$

Rotation



- Rotate vector **a** to **a'** counterclockwise by angle θ

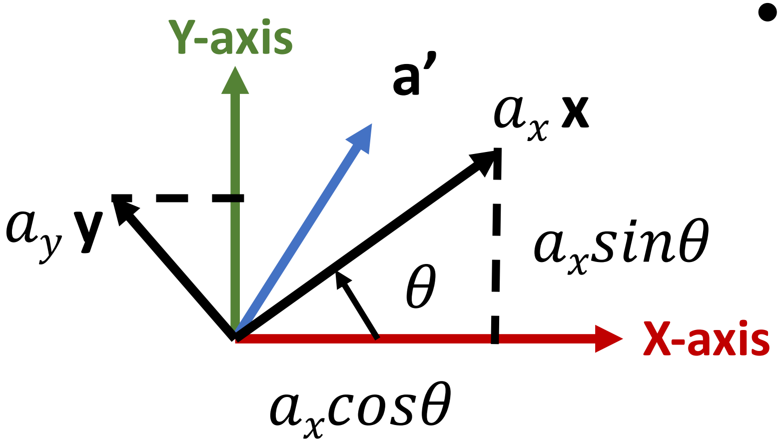
a = a_x **x** + a_y **y**

a' = **R a** (**R** is the rotation matrix)

$$\begin{bmatrix} a_x{}' \\ a_y{}' \end{bmatrix} = a_x \begin{bmatrix} cos\theta \\ sin\theta \end{bmatrix} + a_y \begin{bmatrix} -sin\theta \\ cos\theta \end{bmatrix}$$

$$\begin{bmatrix} a_x{}' \\ a_y{}' \end{bmatrix} = \begin{bmatrix} \boldsymbol{cos\theta} & \boldsymbol{-sin\theta} \\ \boldsymbol{sin\theta} & \boldsymbol{cos\theta} \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$

Rotation



- Rotate vector **a** to **a’** counterclockwise by angle θ

$$\mathbf{a} = a_x \mathbf{x} + a_y \mathbf{y}$$

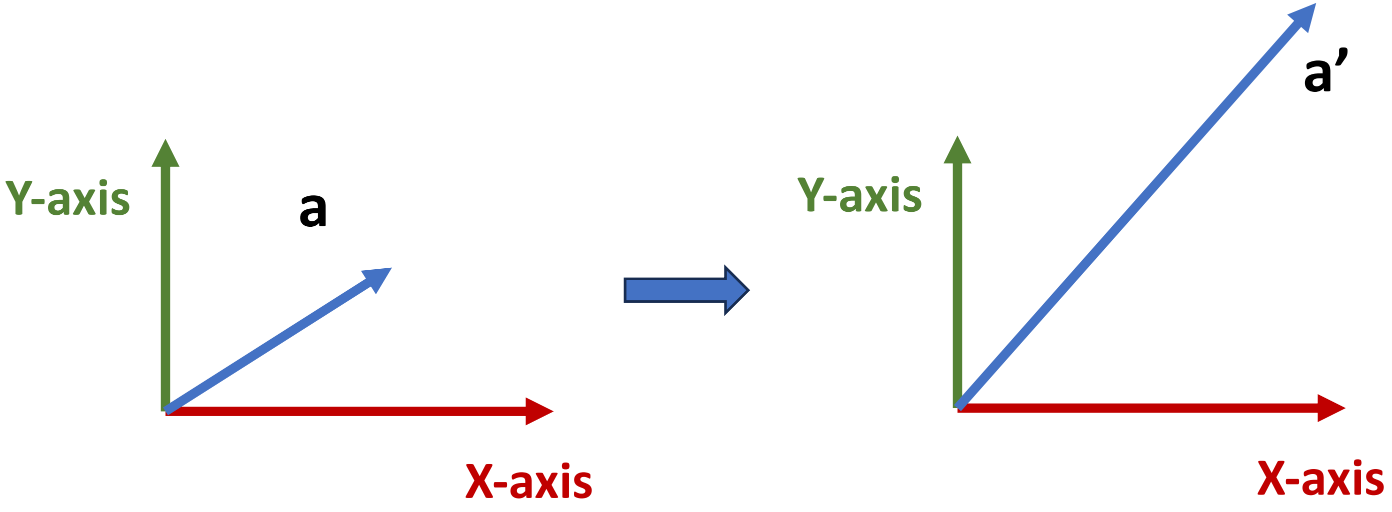
a’ = **R a** (**R** is the rotation matrix)

$$\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = a_x \begin{bmatrix} cos\theta \\ sin\theta \end{bmatrix} + a_y \begin{bmatrix} -sin\theta \\ cos\theta \end{bmatrix}$$

$$\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = \begin{bmatrix} \boldsymbol{cos\theta} & \boldsymbol{-sin\theta} \\ \boldsymbol{sin\theta} & \boldsymbol{cos\theta} \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$

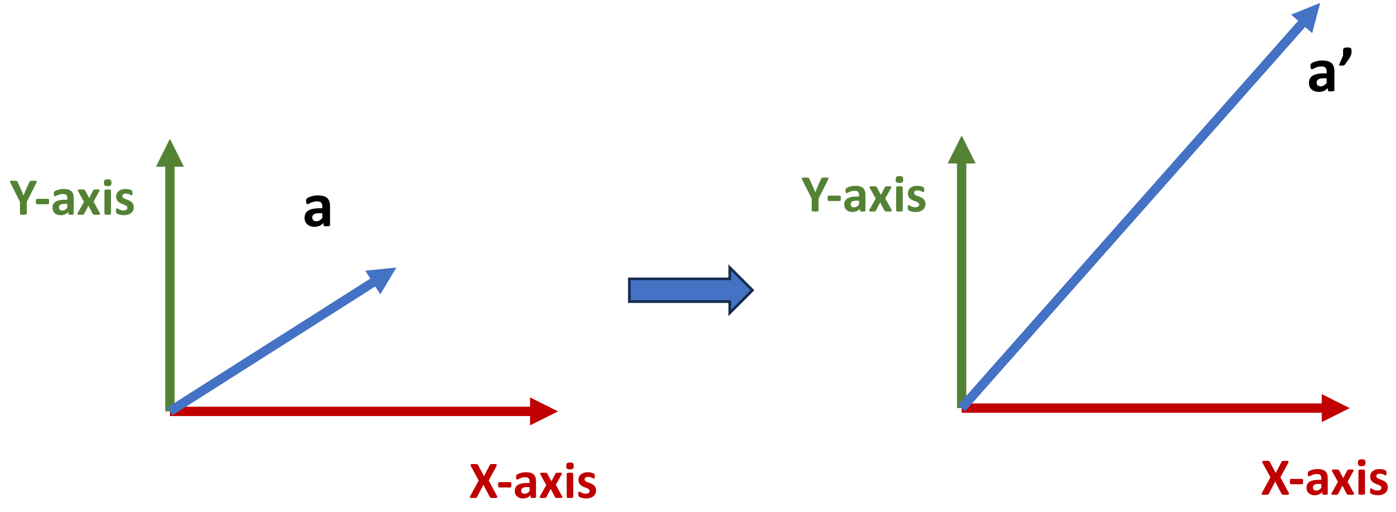
- Rotation matrices are orthogonal matrices
  - Column/Row vectors are perpendicular to each other
- $\mathbf{R}^T = \mathbf{R}^{-1}$
- $\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}$ (Identity matrix)

Scaling: Represent as a Matrix!



- Scale vector **a** to **a'**

Scaling: Represent as a Matrix!



- Scale vector **a** to **a'**
- We can do this by multiplying scalars s_x and s_y with vector **a**

$$\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = \begin{bmatrix} \boldsymbol{s_x} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{s_y} \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} = \begin{bmatrix} s_x a_x \\ s_y a_y \end{bmatrix}$$

- Scale matrices are <u>diagonal</u> matrices

a' = **S a** (**S** is the scaling matrix)

Skew Operation

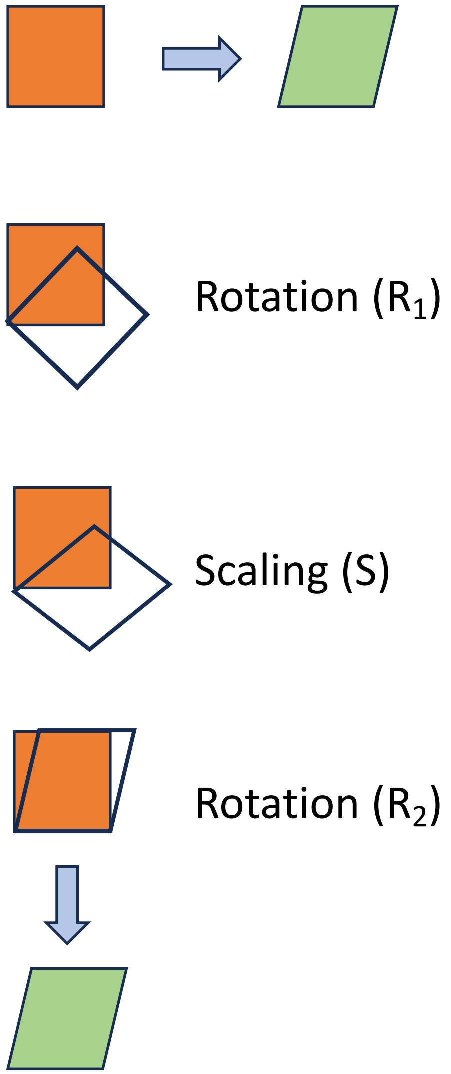
- Skew can be shown as a combination of rotation, followed by non-uniform scaling, and another rotation

$$\mathbf{a}' = R_2\, S\, R_1\, \mathbf{a}$$
$$= S_k\, \mathbf{a} \quad (S_k \text{ is the skew matrix})$$

- Any sequence of rotation and scaling operations now can be combined to a single matrix!

$$\mathbf{a}' = R_4\, S_3\, R_3\, S_2\, R_2\, S_1\, R_1\, \mathbf{a}$$
$$= M\, \mathbf{a}$$

- Great, let's form the Translation matrix!



Translation: Represent as a Matrix!

- Translation?

$$\mathbf{a'} = \mathbf{M\ a + t}$$
$$\mathbf{a'} = \mathbf{M_2\ (M_1\ a + t)}$$
$$\mathbf{a'} = \mathbf{M_2\ (M_1\ a + t_1) + t_2}$$

This does not seem to be a nice way to add translation into a sequence of transformations!

Homogeneous Coordinates

a' = a + t $\Rightarrow$ **a' = T a**

$$\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = \begin{bmatrix} a_x & + & tx \\ a_y & + & ty \end{bmatrix}$$

$$\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = \begin{bmatrix} ? \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$

Homogeneous Coordinates

a' = a + t → **a' = T a**

$$\begin{bmatrix} a_x' \\ a_y' \end{bmatrix} = \begin{bmatrix} a_x & + & tx \\ a_y & + & ty \end{bmatrix}$$

$$\begin{bmatrix} a_x' \\ a_y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ 1 \end{bmatrix}$$

Do addition in a matrix multiplication form

Add an extra dimension, to make the math work Homogeneous coordinate!

Great! Now we can do the following!

a' = R T S R T S R T S R **a** **Composite transformation**

= M **a**

Summary of 2D Transformations

- Translation matrix

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

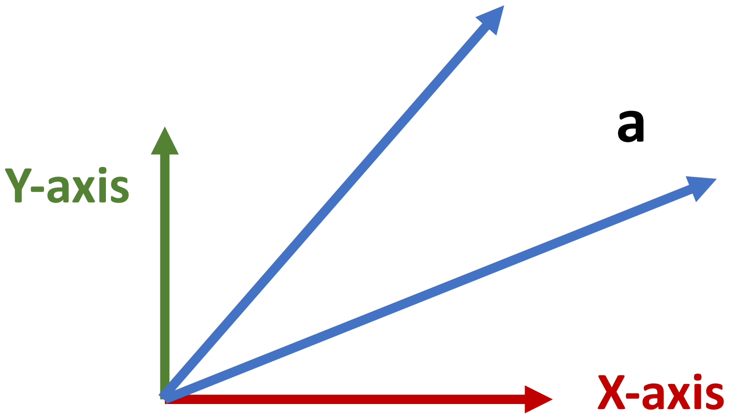
- Scale matrix

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

- Rotation matrix

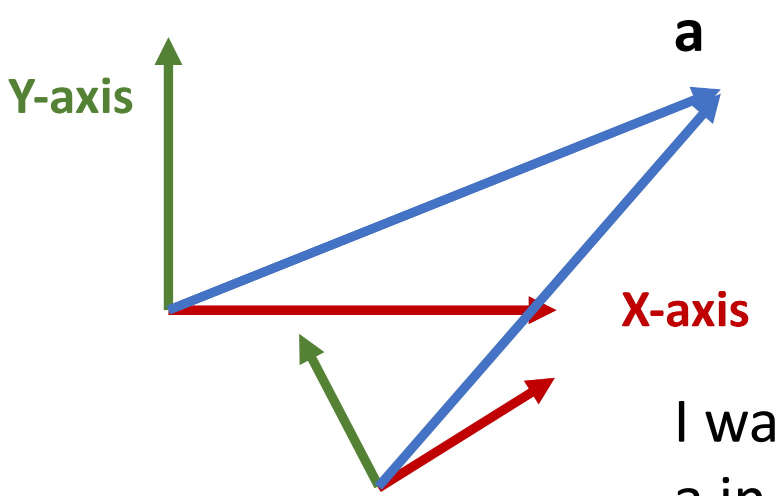
$$R = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Thinking About Affine Transformations



Thinking transformations as deforming the vectors or objects

Thinking About Affine Transformations

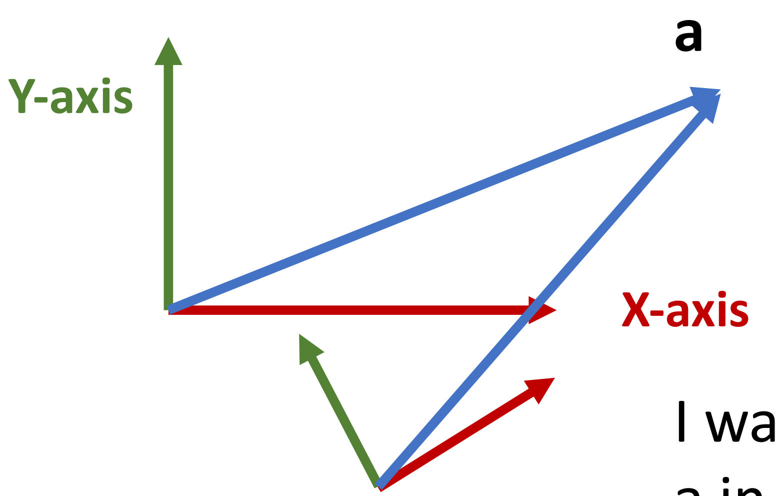


I want to know the position of point at a in a different coordinate system

Thinking transformations as changing the coordinate frame

Thinking About Affine Transformations

IITK CS360A



I want to know the position of point at a in a different coordinate system

Thinking transformations as changing the coordinate frame

Transformation Matrix Interpretation

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

Affine transformation matrix

Transformation Matrix Interpretation

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation and Scaling components

Affine transformation matrix

Transformation Matrix Interpretation

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

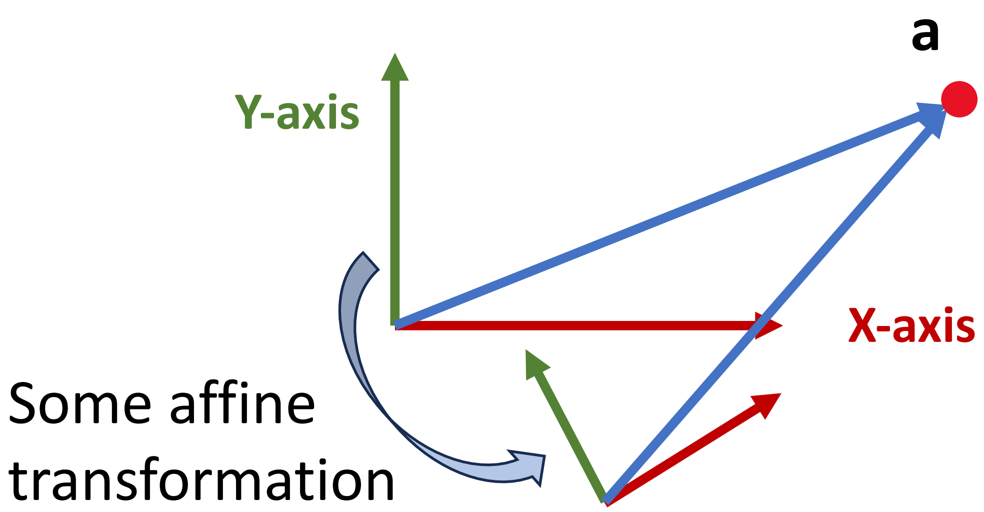
Rotation and Scaling components

Translation components

Affine transformation matrix

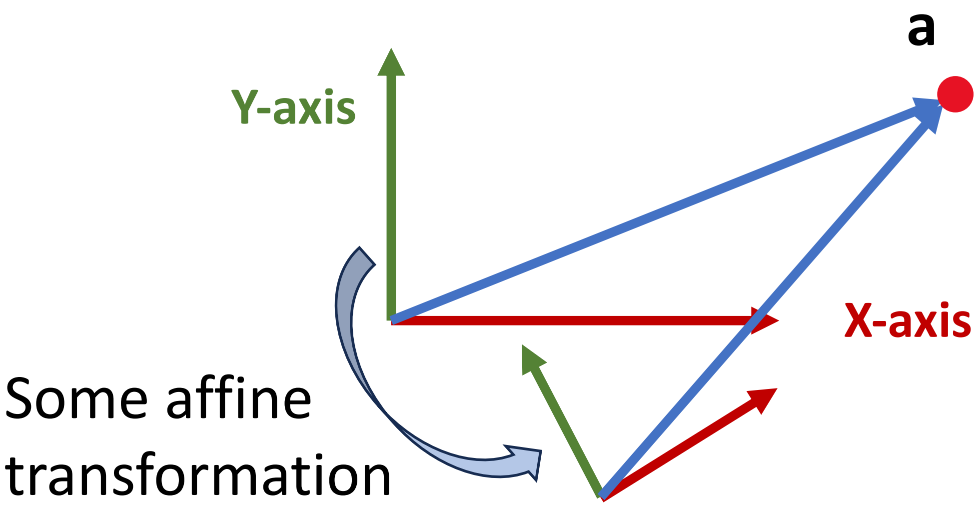
Positions vs Directions

- Vectors represent both positions and directions
  - When transforming, are the rules same for position and direction vectors?



Positions vs Directions

- Vectors represent both positions and directions
  - When transforming, are the rules same for position and direction vectors?

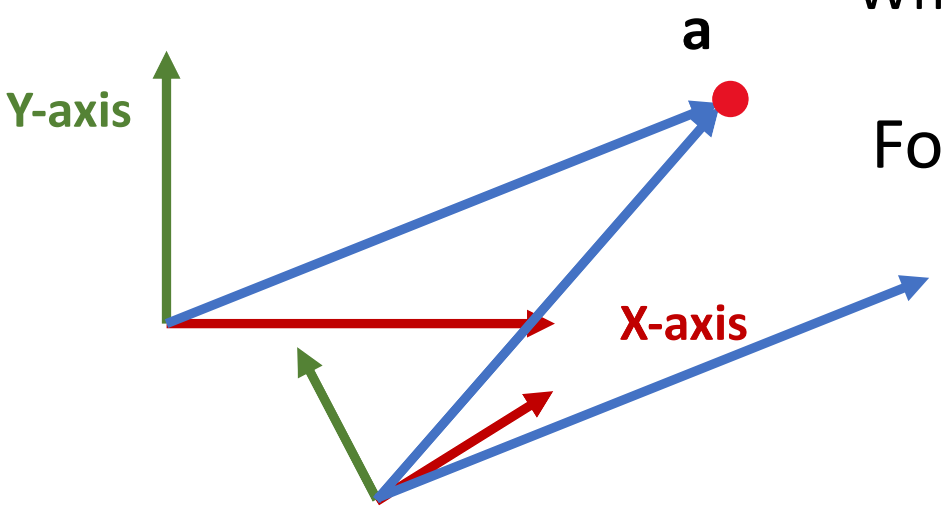


For Position vectors:

$$\begin{bmatrix} a_x' \\ a_y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ 1 \end{bmatrix}$$

Positions vs Directions

- Vectors represent both positions and directions
  - When transforming, are the rules same for position and direction vectors?



What if we are interested in the direction?

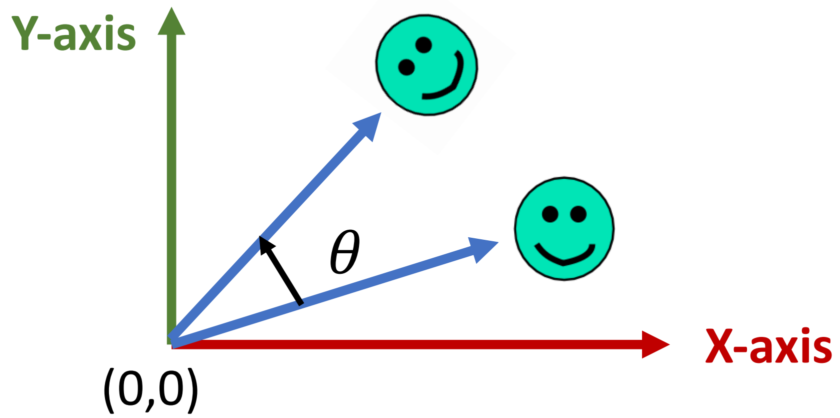
For Direction vectors:

$$\begin{bmatrix} a_x{}' \\ a_y{}' \\ 0 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ 0 \end{bmatrix}$$

Revisit Rotation

- The standard rotation matrix rotates objects around the origin (0,0)

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

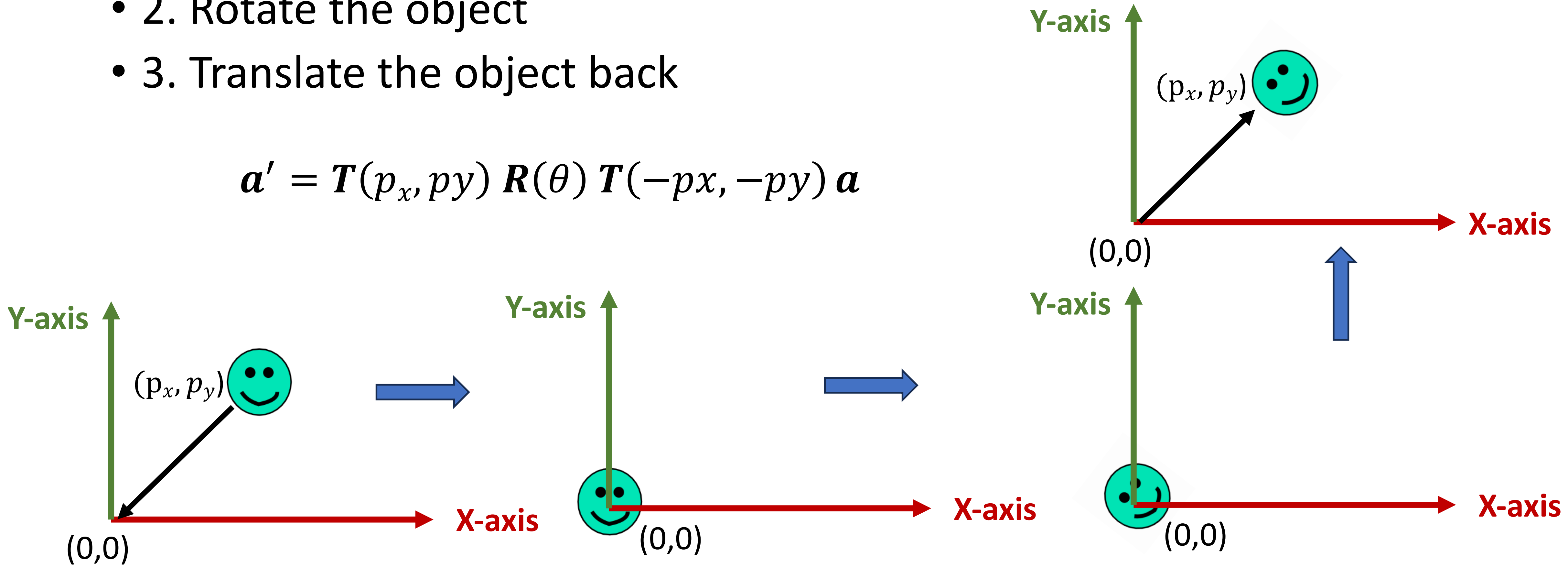


- But what if I want to rotate around an arbitrary center point?

Revisit Rotation

- To rotate about an arbitrary point P (p_x, p_y) by θ :
- 1. Translate the object so that P will coincide with the origin
- 2. Rotate the object
- 3. Translate the object back

$$\boldsymbol{a}' = \boldsymbol{T}(p_x, py)\,\boldsymbol{R}(\theta)\,\boldsymbol{T}(-px, -py)\,\boldsymbol{a}$$

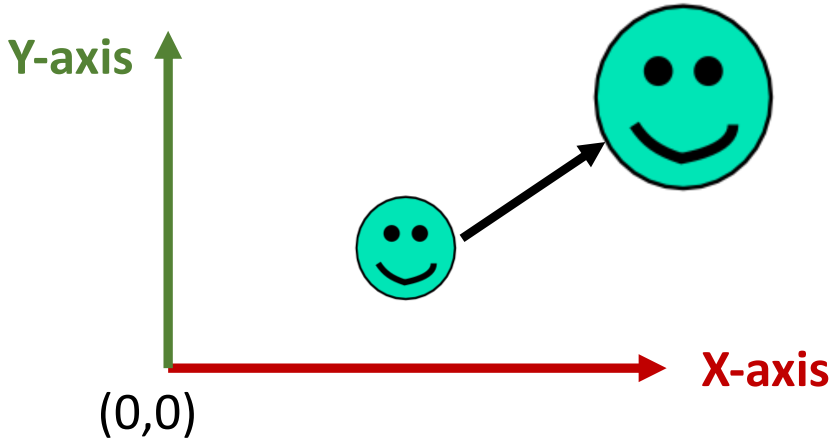


Revisit Scaling

IITK CS360A

- The standard scaling matrix only pivot around origin (0,0)

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

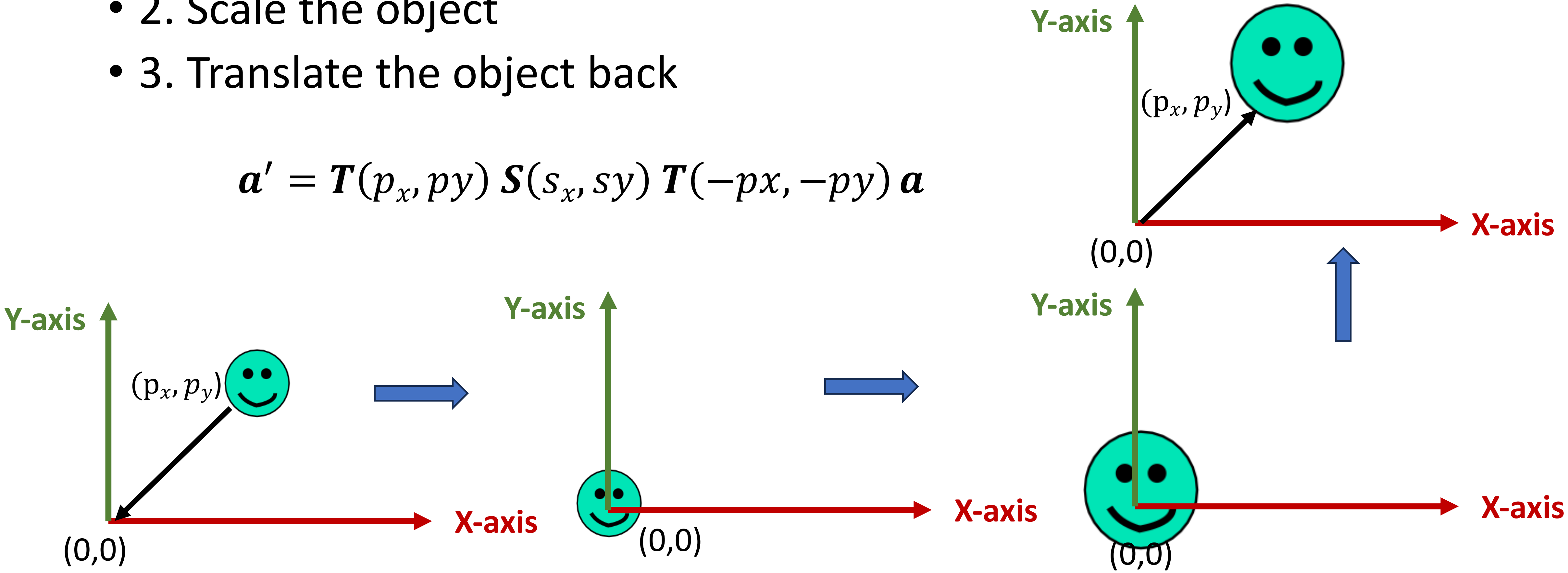


- But what if I want to scale about an arbitrary pivot point?

Revisit Scaling

- To scale about an arbitrary pivot point P (p_x, p_y) by θ :
- 1. Translate the object so that P will coincide with the origin
- 2. Scale the object
- 3. Translate the object back

$$\boldsymbol{a}' = \boldsymbol{T}(p_x, py)\,\boldsymbol{S}(s_x, sy)\,\boldsymbol{T}(-px, -py)\,\boldsymbol{a}$$

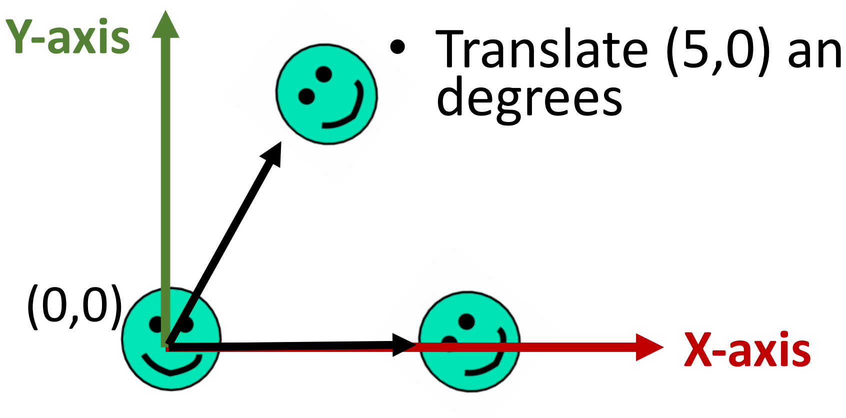


Transformation Order Matters!

- Example: Rotation and Translation are not commutative!

Transformation Order Matters!

- Example: Rotation and Translation are not commutative!



- Translate (5,0) and then rotate 75 degrees
- Rotate 75 degrees and then Translate (5,0)