CSE-170 Computer Graphics

Lecture 4

Transformations (I)

Dr. Renato Farias
rfarias2@ucmerced.edu

Announcements

- Labs start this week!

- The first programming assignment has been posted, check CatCourses
  - pdf with instructions
  - zips with support code (assumes Visual Studio)

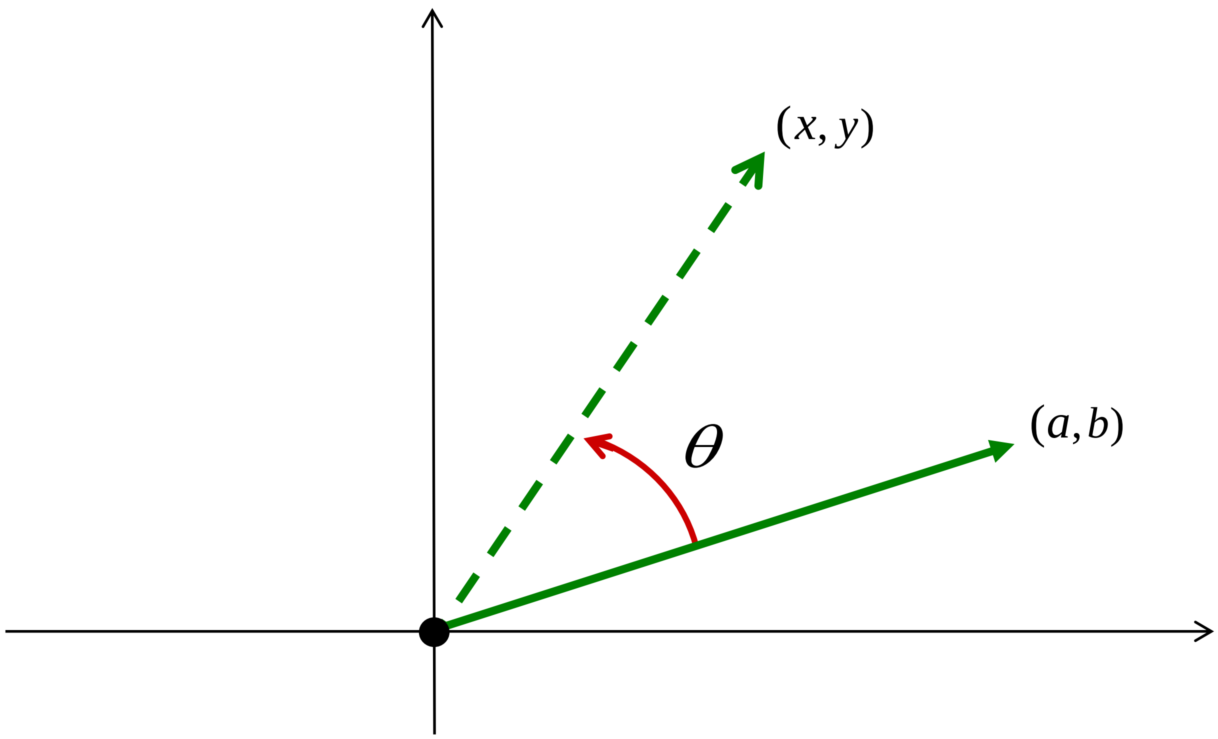
2D Transformation Matrices

2D Rotation

- What is a primitive rotation?
  - Rotation around origin of coordinate system
  - Can be easily encoded as a matrix multiplication!

2D Rotation

- How to compute a 2D Rotation of angle θ?
  - Use some trigonometry



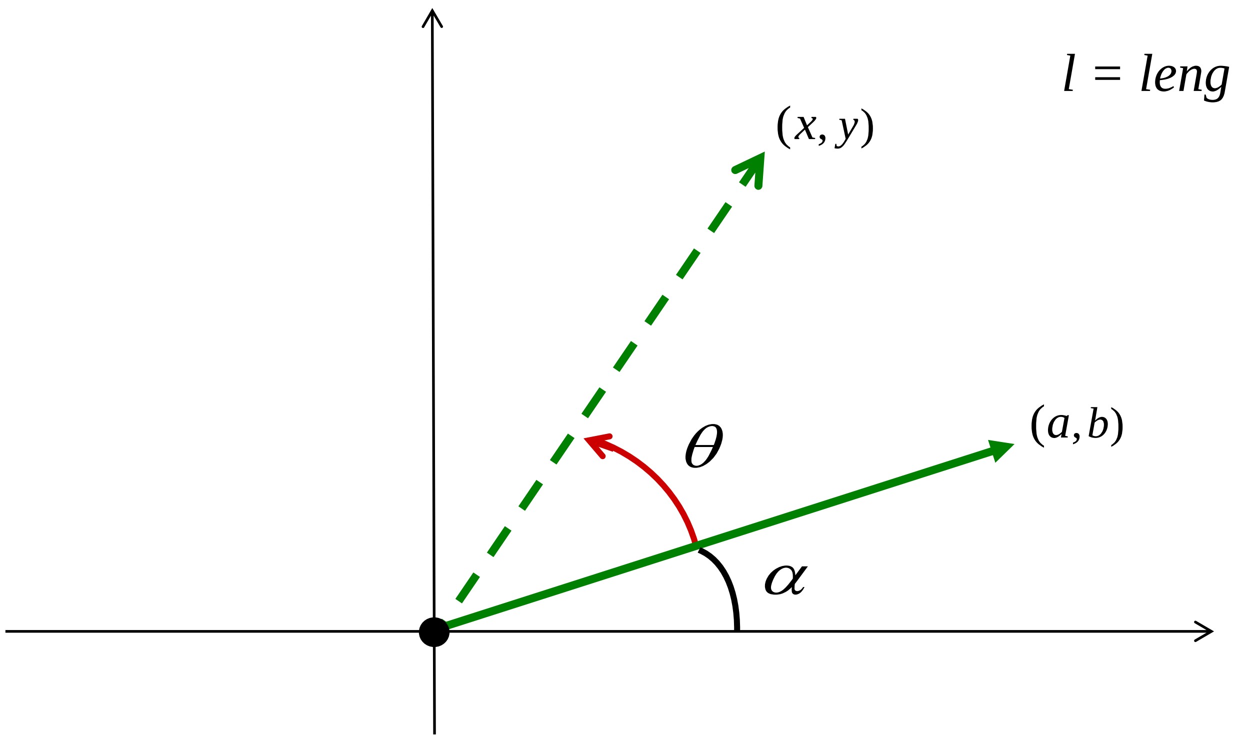
2D Rotation

- How to compute a 2D Rotation of angle θ?
  - Use some trigonometry

$$a = l\cos\alpha, \quad x = l\cos(\alpha+\theta),$$

$$b = l\sin\alpha, \quad y = l\sin(\alpha+\theta)$$

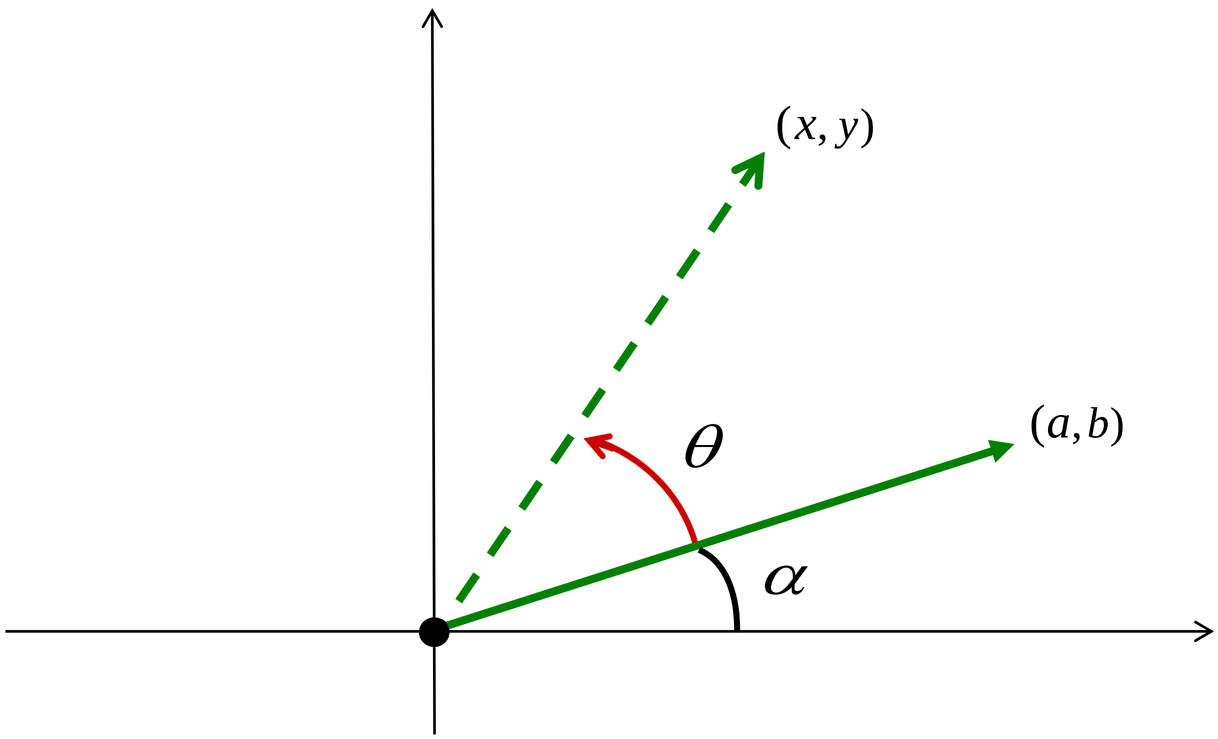
l = length of vector (a,b)



2D Rotation

- How to compute a 2D Rotation of angle θ?
  - Use some trigonometry
  - And remember: $\sin(\alpha+\theta)=\sin\alpha\cos\theta+\cos\alpha\sin\theta$

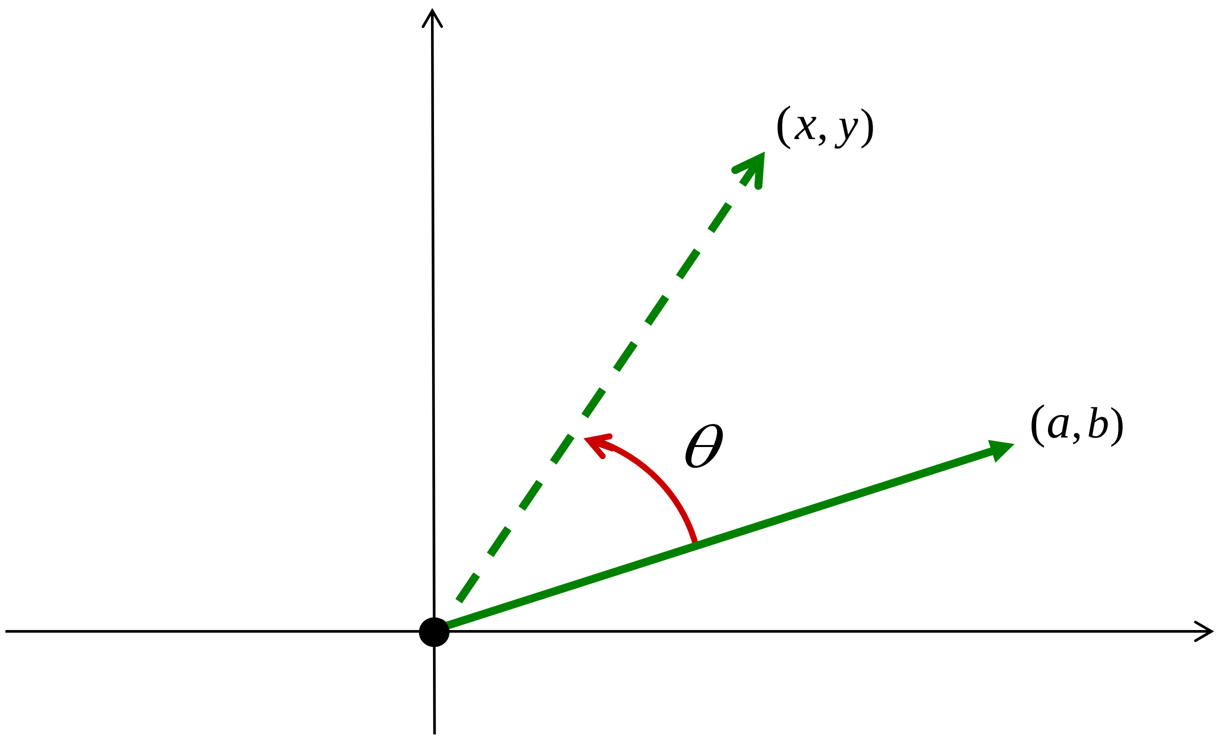
    $\cos(\alpha+\theta)=\cos\alpha\cos\theta-\sin\alpha\sin\theta$



2D Rotation

- How to compute a 2D Rotation of angle θ?
  - Use some trigonometry
  - Result:

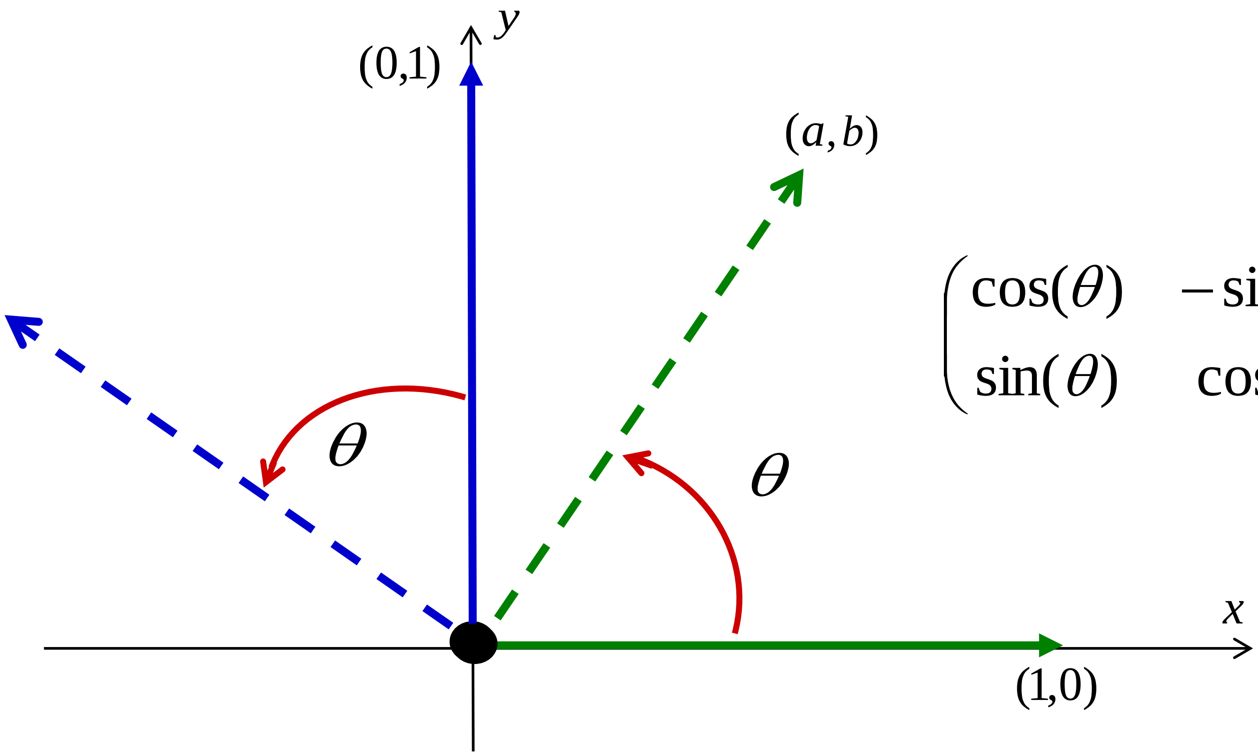
$$x = a\cos\theta - b\sin\theta$$
$$y = a\sin\theta + b\cos\theta$$
$$\Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$



2D Rotation Matrix Encoding

- 2D Rotation by given angle: $\mathbf{R}_\theta = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$

  - Examples:



$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

Rotations

- Rotations are **linear rigid transformations**
- Matrix inverse: inverses the transformation
- For rotations:
  – The inverse transformation of a rotation matrix of angle θ is another rotation matrix of angle $-\theta$.
  – For rotations, the transpose of the matrix works as the inverse transformation
    - So no need to compute the inverse of the matrix to rotate objects back

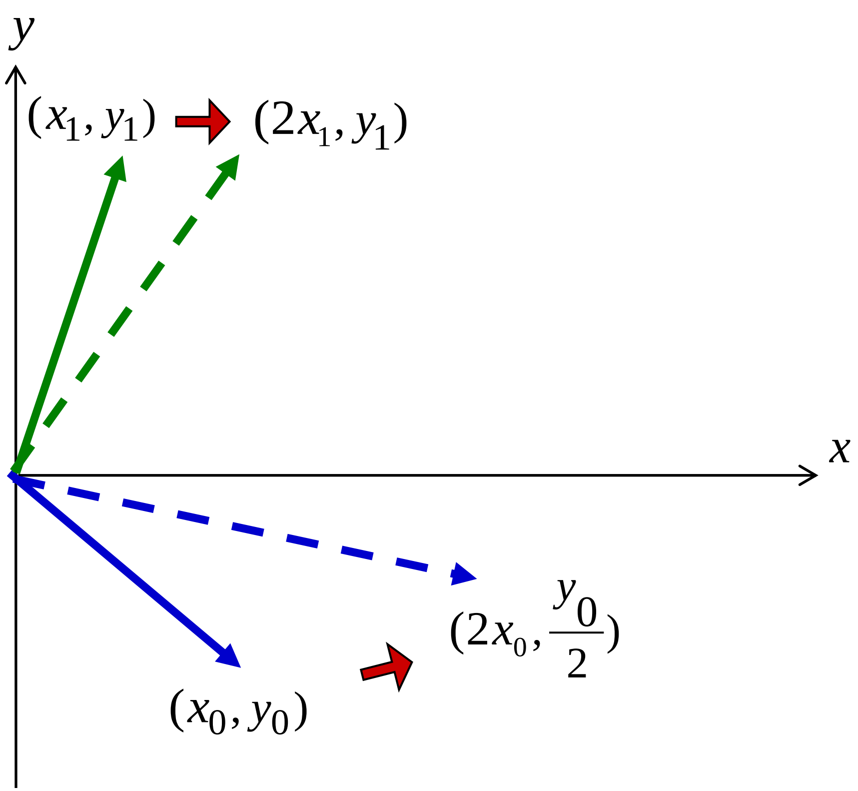
  (this is also valid for rotations in 3D)

2D Scaling

- 2D Scaling Matrix

$$S_{r,s} = \begin{pmatrix} r & 0 \\ 0 & s \end{pmatrix}$$

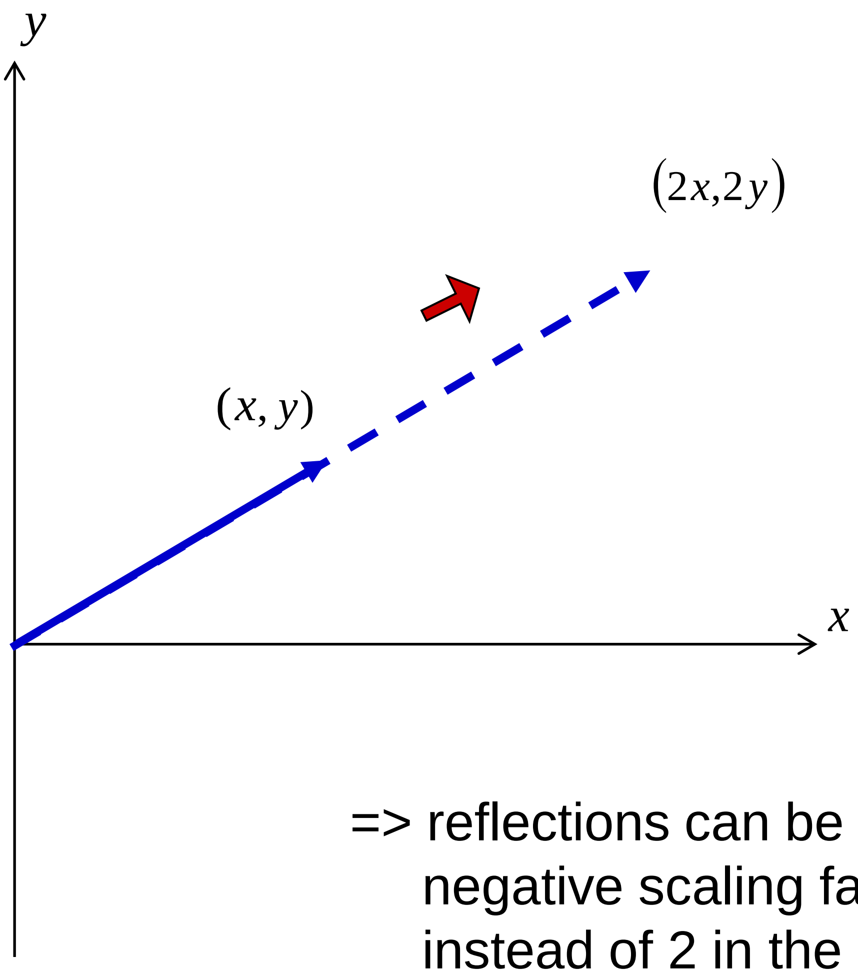
- Examples



$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 2x_1 \\ y_1 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 2x_0 \\ \frac{y_0}{2} \end{pmatrix}$$

Uniform Scaling

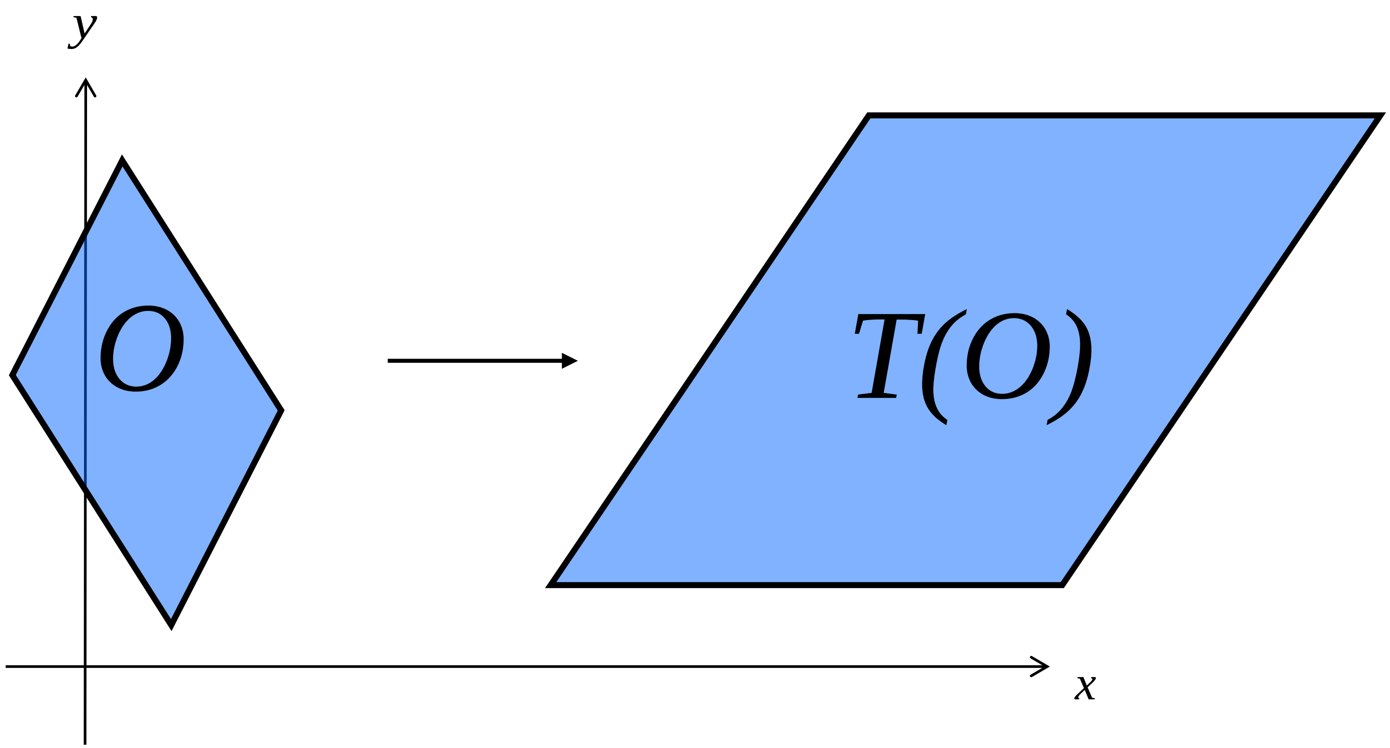


$$S_2 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

=> reflections can be achieved with negative scaling factors: try -1 instead of 2 in the example

Transformation of "Objects"

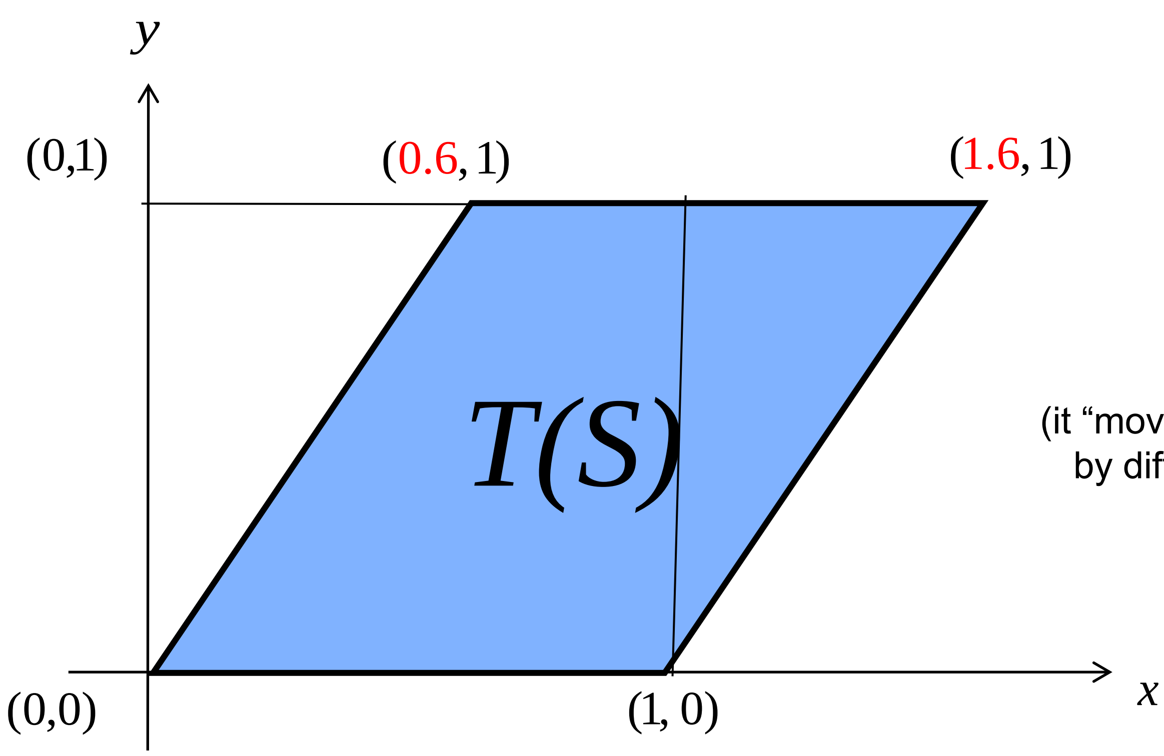
- Just transform every vertex of the object!
  - In 2D our objects are polygons described by a sequence of 2D points
  - Ex:



Shearing

- Shearing transformation $Sh_{x,y} = \begin{pmatrix} 1 & x \\ y & 1 \end{pmatrix}$

- Ex: shearing in x by 0.6 $Sh_{x,y} = \begin{pmatrix} 1 & 0.6 \\ 0 & 1 \end{pmatrix}$



(it "moves the coordinates by different amounts")

Types of transformations

- Linear
  - Identity
  - Rotations
  - Scalings
  - Reflections (reflections are “scalings by -1”)
  - Shears
  - Combinations of linear transformations are also linear
  - All invertible (except scaling of 0)

- Affine = Linear plus translations
  - An affine transformation preserves collinearity and ratios of distances (ex: a midpoint remains a midpoint)

- But, we also need a way to encode translations!

Homogeneous Coordinates

- Allow us to perform affine mappings $\mathbf{y}=A\mathbf{x}+\mathbf{p}$
  – We need to encode both linear transformations and translations

- Homogeneous coordinates
  – Allows to include translations in matrix representation
  – Mathematical interpretation: Projective Geometry

Affine Maps

- Affine maps can be encoded with homogeneous coordinates
  - From Cartesian to homogeneous coordinates:

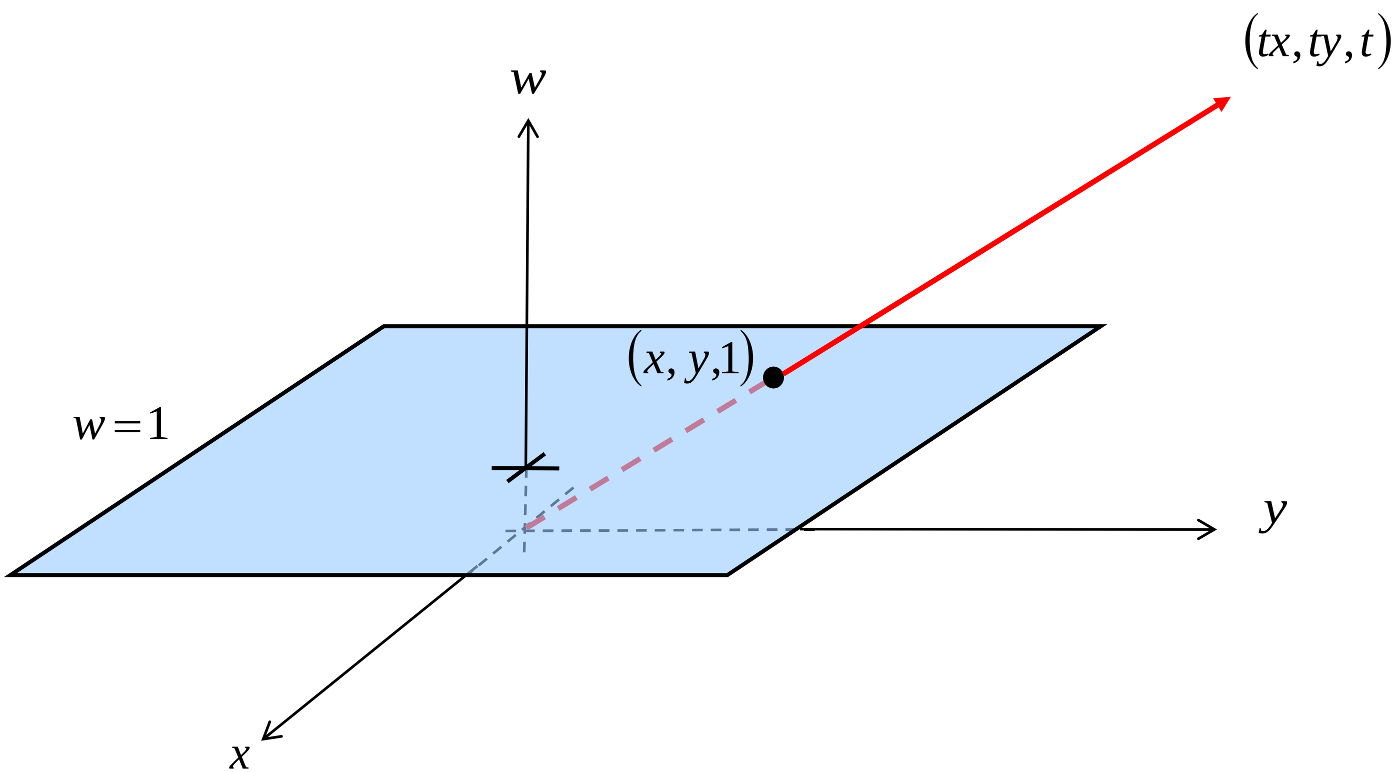
$$\mathbf{v} = \begin{pmatrix} x \\ y \end{pmatrix} \quad \Rightarrow \quad \underline{\mathbf{v}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

  - From homogeneous to Cartesian coordinates:

$$\underline{\mathbf{v}} = \begin{pmatrix} x \\ y \\ w \end{pmatrix} \quad \Rightarrow \quad \mathbf{v} = \begin{pmatrix} x/w \\ y/w \end{pmatrix}$$

Homogeneous Coordinates

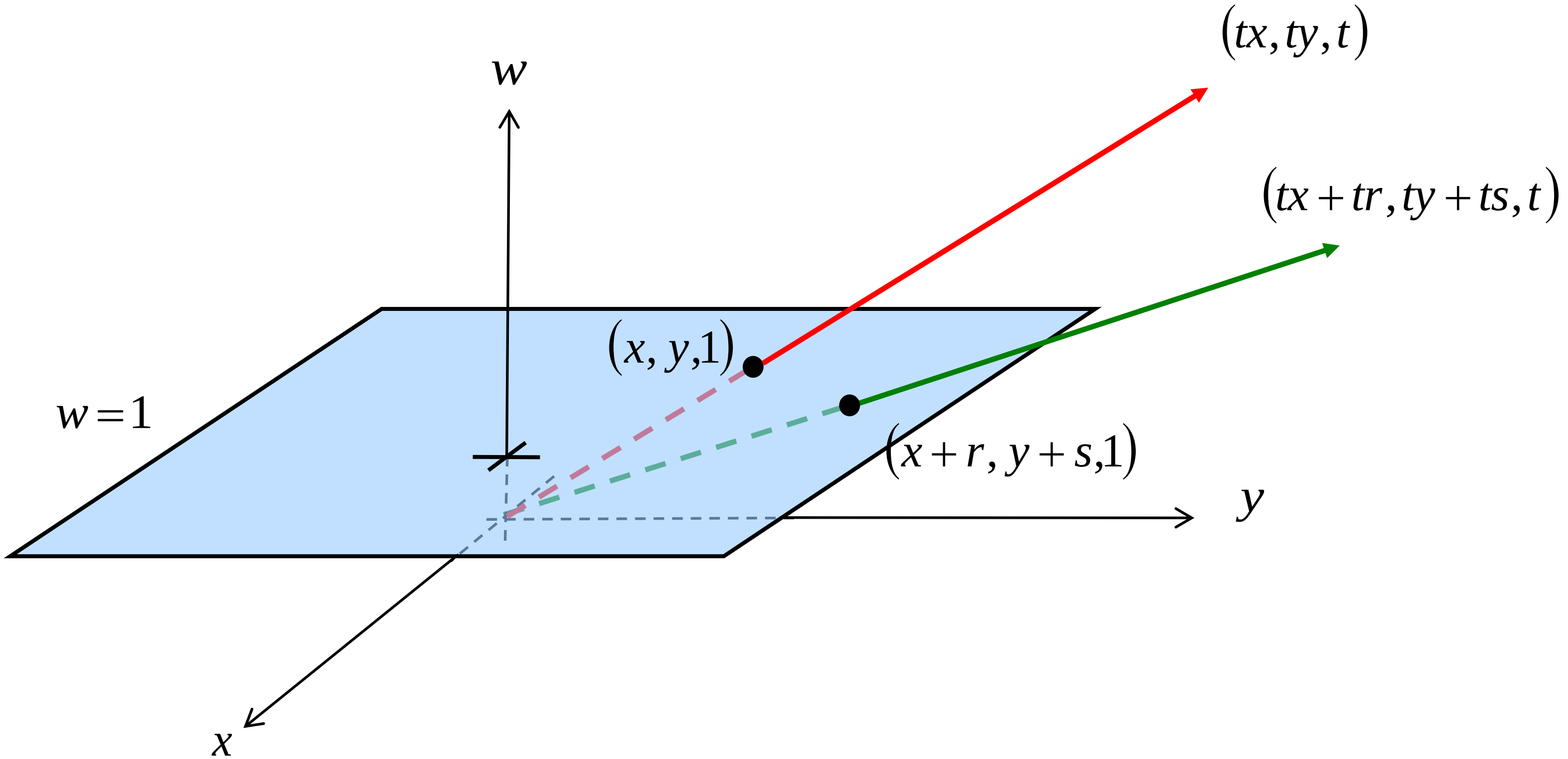
- An infinite number of points correspond to $(x,y,1)$
  - All points are in the line (tx,ty,t), $t \in \Re$



Homogeneous Coordinates

- Let's now apply a shear transformation in 3D:

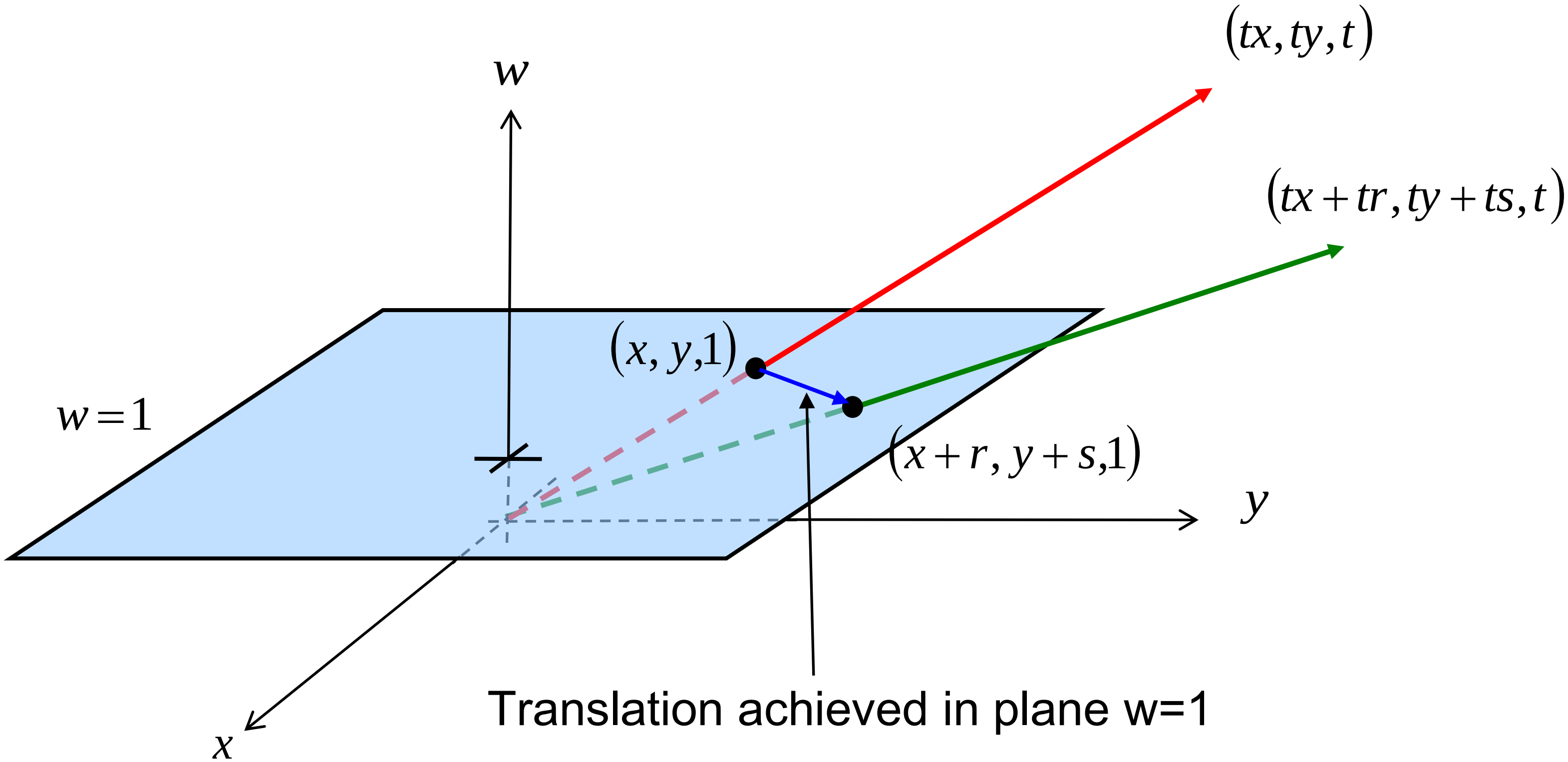
$$\begin{pmatrix} 1 & 0 & r \\ 0 & 1 & s \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+r \\ y+s \\ 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & r \\ 0 & 1 & s \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} tx \\ ty \\ t \end{pmatrix} = \begin{pmatrix} tx+tr \\ ty+ts \\ t \end{pmatrix}$$



Homogeneous Coordinates

- Let's now apply a shear transformation in 3D:

$$\begin{pmatrix} 1 & 0 & r \\ 0 & 1 & s \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+r \\ y+s \\ 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & r \\ 0 & 1 & s \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} tx \\ ty \\ t \end{pmatrix} = \begin{pmatrix} tx+tr \\ ty+ts \\ t \end{pmatrix}$$



Homogeneous Transformation

- Summary
  - Shear in 3D results in a translation in the 2D plane w=1
  - We used a linear transformation in 3D to achieve a homogeneous transformation in 2D
  - This can be generalized to any dimension
    - The last column in the homogeneous matrix encodes the translation:

$$\mathbf{T}_{r,s} = \begin{pmatrix} 1 & 0 & r \\ 0 & 1 & s \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & r \\ 0 & 1 & s \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+r \\ y+s \\ 1 \end{pmatrix}$$

Homogeneous Transformation

- Using Homogeneous Transformations:

1. Convert to homogeneous coordinate

$$\mathbf{v} = \begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \underline{\mathbf{v}} = \begin{pmatrix} x \\ y \\ \mathbf{1} \end{pmatrix}$$

2. Transform

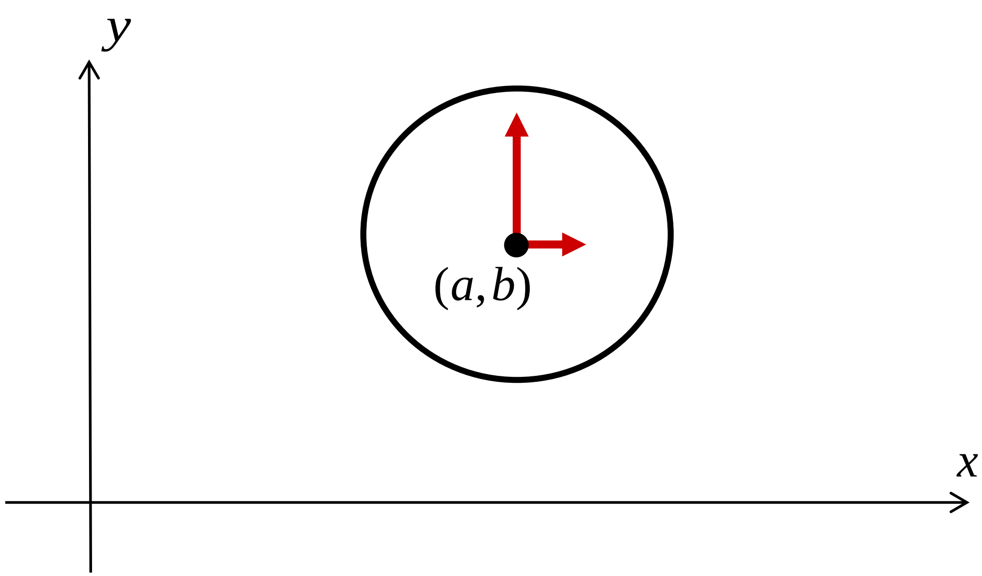
$$\underline{\mathbf{v}}' = \begin{pmatrix} 1 & 0 & r \\ 0 & 1 & s \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+r \\ y+s \\ 1 \end{pmatrix}$$

3. Convert back to original dimension

$$\underline{\mathbf{v}}' = \begin{pmatrix} x+r \\ y+s \\ 1 \end{pmatrix} \Rightarrow \mathbf{v}' = \begin{pmatrix} \frac{x+r}{1} \\ \frac{y+s}{1} \end{pmatrix} = \begin{pmatrix} x+r \\ y+s \end{pmatrix}$$

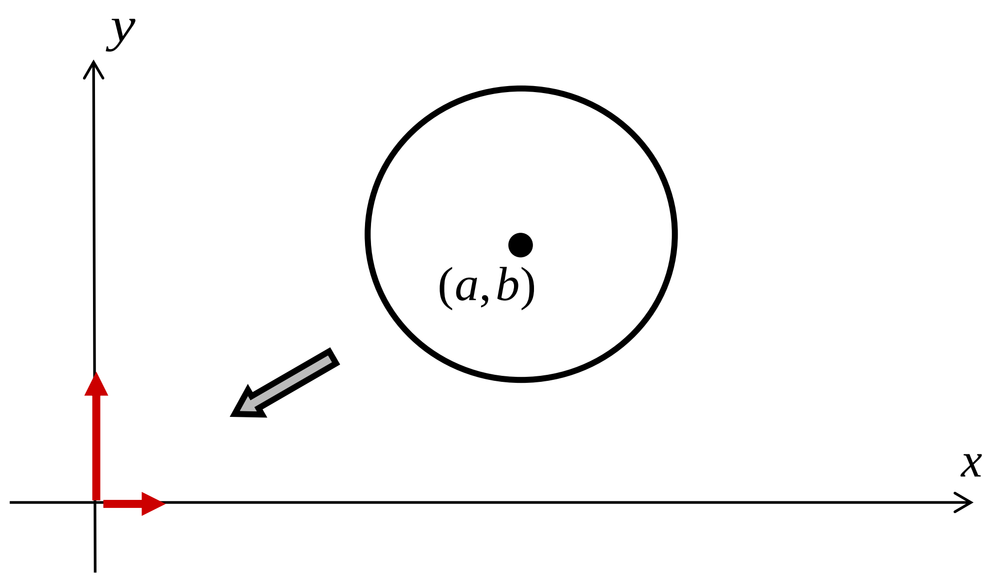
Composition of Transformations

- Example: advance clock hands
  - Which transformation matrix would you use to rotate the clock hands?



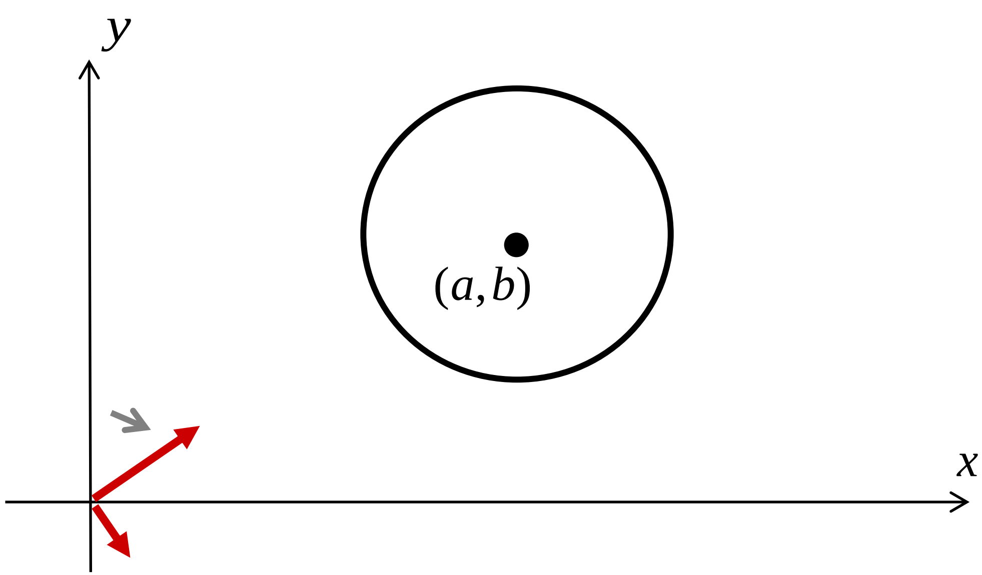
Composition of Transformations

- Composing transformations
  - First, translate to origin



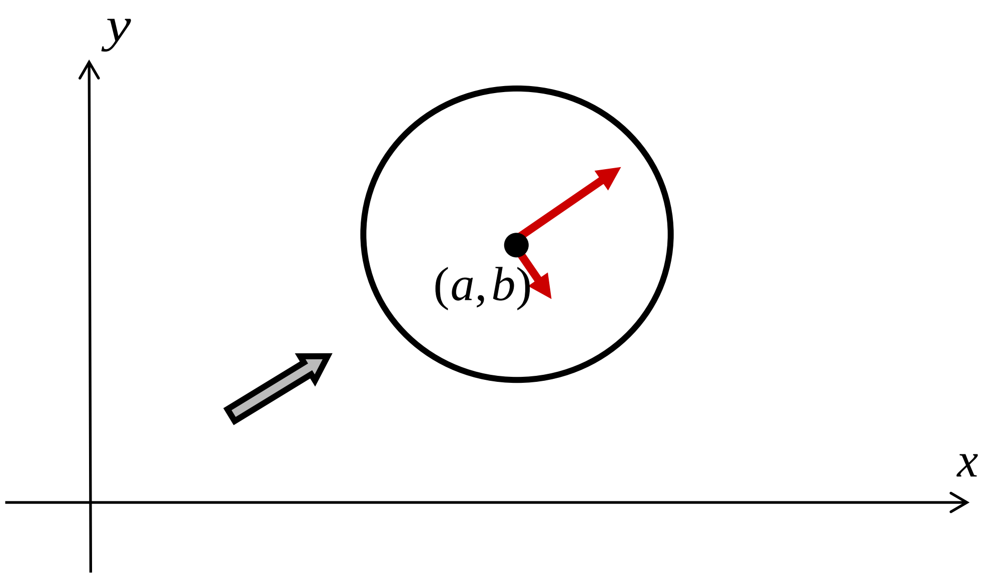
Composition of Transformations

- Composing transformations
  - Now rotate



Composition of Transformations

- Composing transformations
  - Finally translate back



Composition of Transformations

- The final transformation is obtained with the composition of the three transformations:

$$\underbrace{\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix}}_{\text{translation back}} \underbrace{\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{rotation around origin}} \underbrace{\begin{pmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{pmatrix}}_{\text{translation to origin}}$$

- Applying the transformation:

$$\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$$

3D Transformation Matrices

3D Transformations

- Homogeneous Coordinates
  - in 2D, we used 3 x 3 matrices
  - in 3D, we use 4 x 4 matrices

- Again, in homogeneous coordinates each 3D point has an extra value, *w*

$$\begin{pmatrix} e_{11} & e_{12} & e_{13} & e_{14} \\ e_{21} & e_{22} & e_{23} & e_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

Homogeneous Coordinates

- Affine transformations
  - If we multiply a homogeneous coordinate by an *affine matrix*, w is unchanged

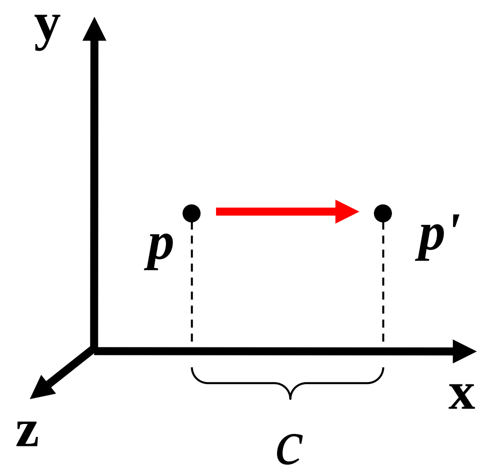
$$\begin{pmatrix} e_{11} & e_{12} & e_{13} & e_{14} \\ e_{21} & e_{22} & e_{23} & e_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w \end{pmatrix}$$

Translation

- Translation of (a,b,c)

$$\begin{pmatrix} 1 & 0 & 0 & \boxed{a} \\ 0 & 1 & 0 & \boxed{b} \\ 0 & 0 & 1 & \boxed{c} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x+a \\ y+b \\ z+c \\ 1 \end{pmatrix}$$

Translation(c,0,0)

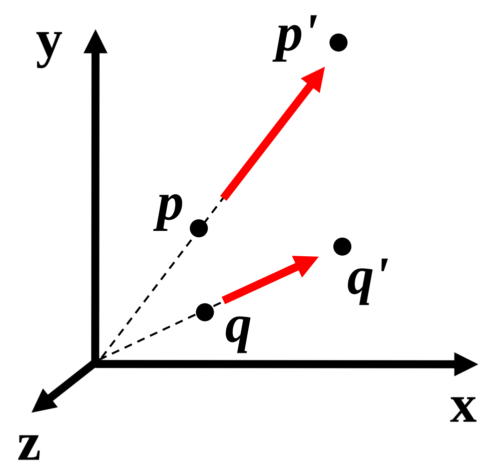


Scaling

- Scaling of (a,b,c)

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} ax \\ by \\ cz \\ 1 \end{pmatrix}$$

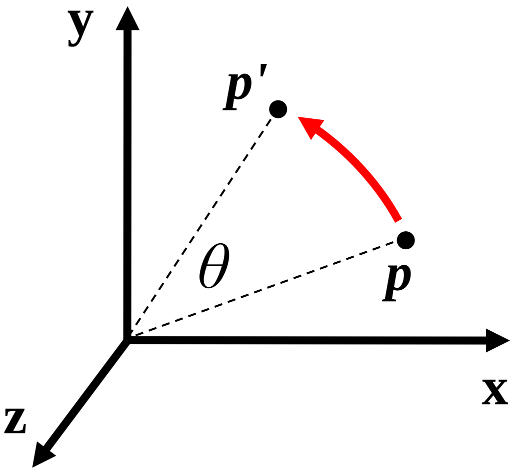
Scaling:



Rotation

- Around z axis

$$R_z(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

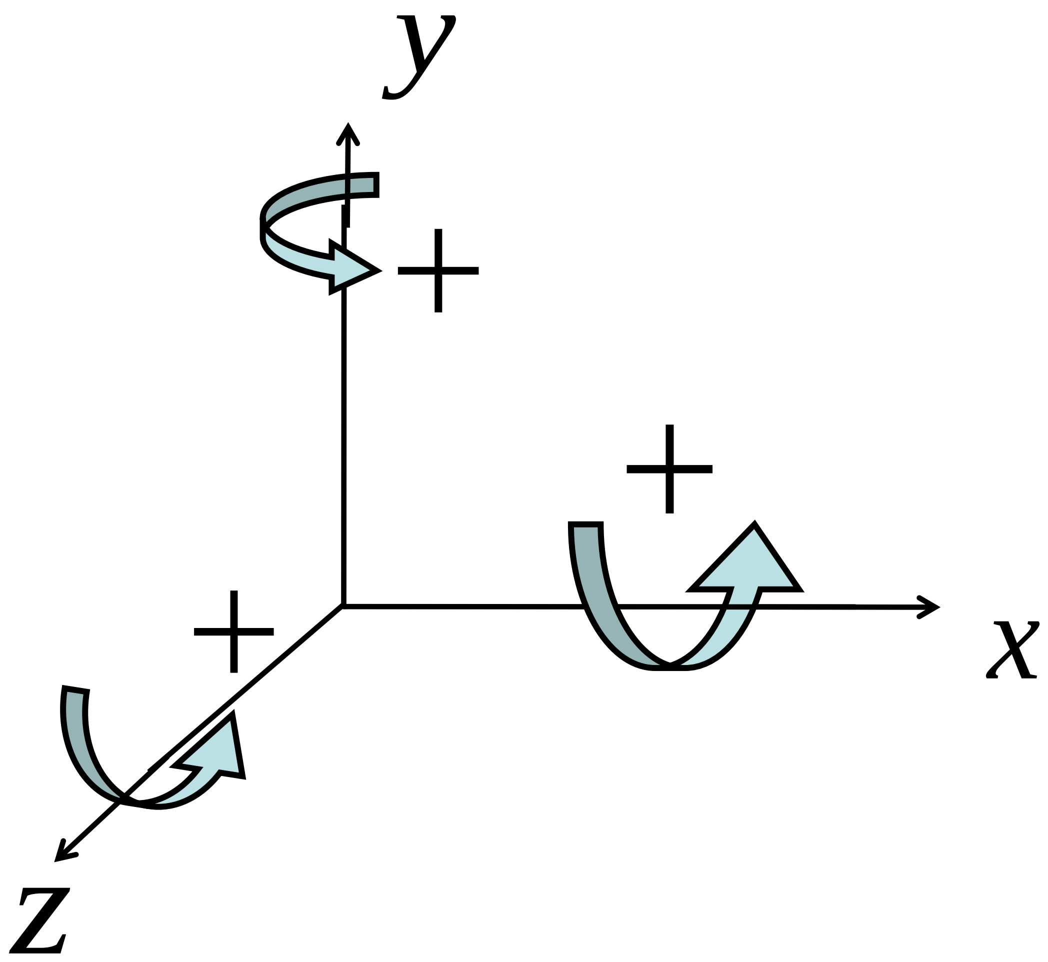


Rotation

- Around x axis: $$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

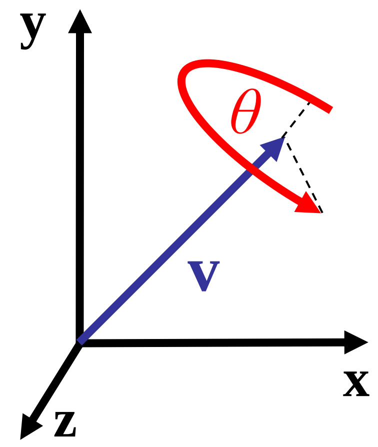
- Around y axis: $$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D Positive Rotations



Rotation around arbitrary axis

- About $\mathbf{v}=(x,y,z)$, a unit vector on an arbitrary axis (Rodrigues Formula):



$$\begin{pmatrix} \cos\theta+(1-\cos\theta)x^2 & (1-\cos\theta)xy-(\sin\theta)z & (1-\cos\theta)xz+(\sin\theta)y & 0 \\ (1-\cos\theta)yx+(\sin\theta)z & \cos\theta+(1-\cos\theta)y^2 & (1-\cos\theta)yz-(\sin\theta)x & 0 \\ (1-\cos\theta)zx-(\sin\theta)y & (1-\cos\theta)zy+(\sin\theta)x & \cos\theta+(1-\cos\theta)z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - How can you derive the above matrix?
    - Decompose the desired transformation in simple transformation matrices, then multiply all of them together. The idea is to use the transformations mapping the generic axis *v* to one frame axis, rotate your points around that frame axis, and then transform everything back.

OpenGL matrices

- OpenGL stores matrices in 16-value arrays with column-major format
  - The elements of each column are contiguous in memory:

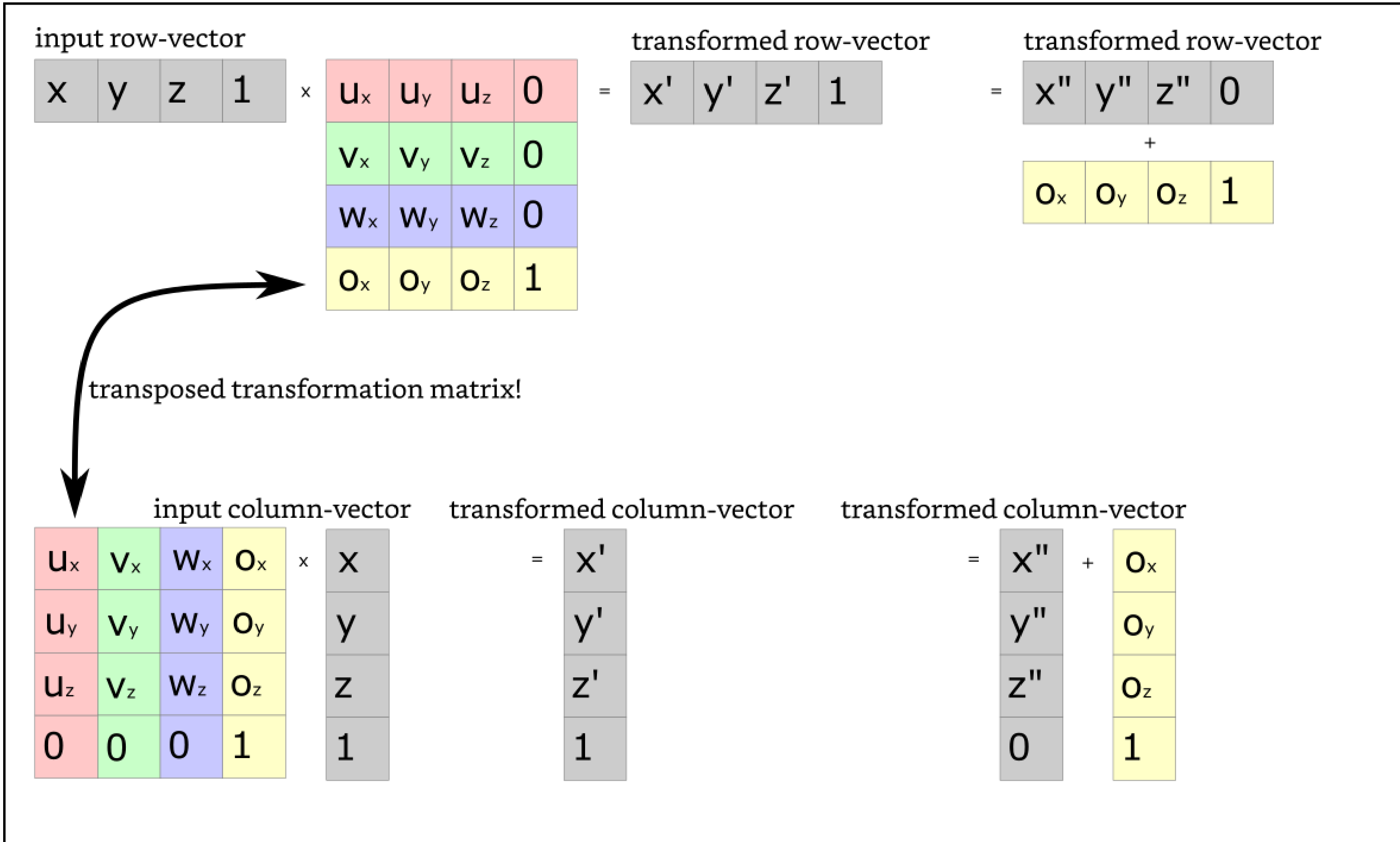
```
mat4 (  vec4,   // first column
        vec4,   // second column
        vec4,   // third column
        vec4 ); // fourth column
```

  - This follows all the notation we have seen

OpenGL matrices

- If you build your matrices according to the notation we've seen here, you should be fine with post-multiplication
- When values in line-major format are stored in column-major format:
  - The resulting matrix is transposed!
  - Simple change: vectors should then multiply the matrices from the left
- Either works, your program just has to be consistent in which it uses!

OpenGL matrices



https://al-radkov.com/blog/2020/03/10/matrix-multiplication-and-the-GPU/