Colorado School of Mines

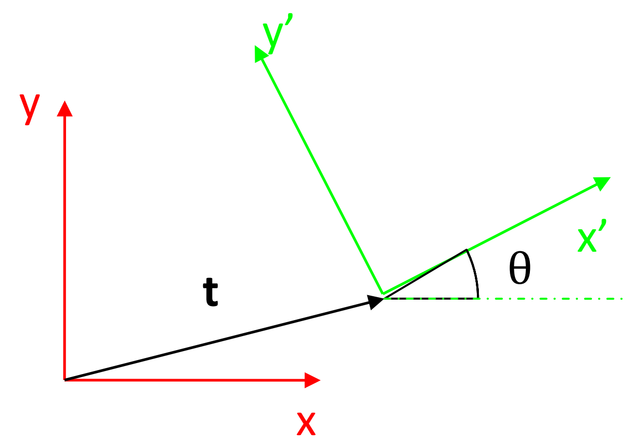
Computer Vision

Coordinate Transformations

2D-2D Coordinate Transforms

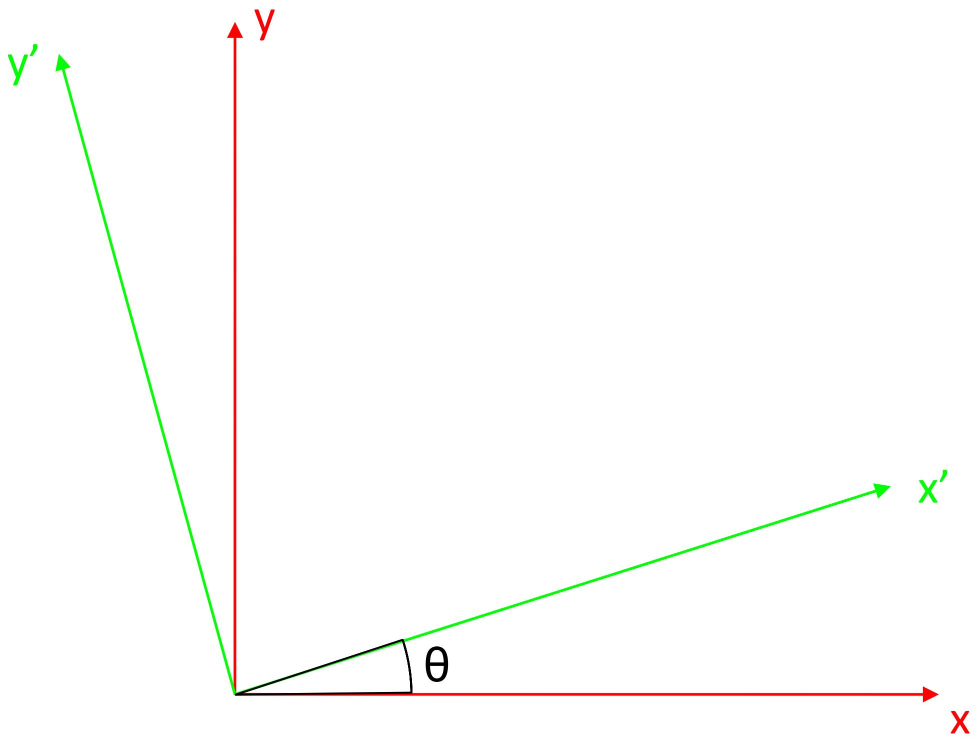
2D Rigid Frame Transformations

- The pose of one 2D frame with respect to another is described by
  - Translation vector **t**=(Δx,Δy)^T
  - Rotation angle θ
    - Rotation can also be represented as a 2x2 matrix **R**
- Object shape and size is preserved

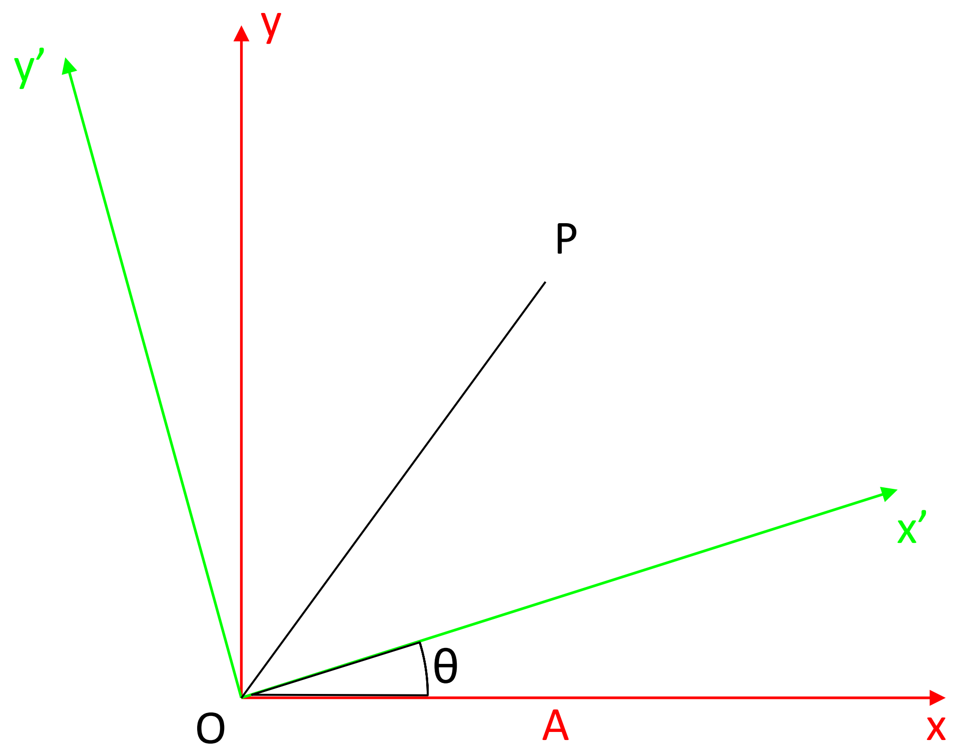


Rotations in 2D

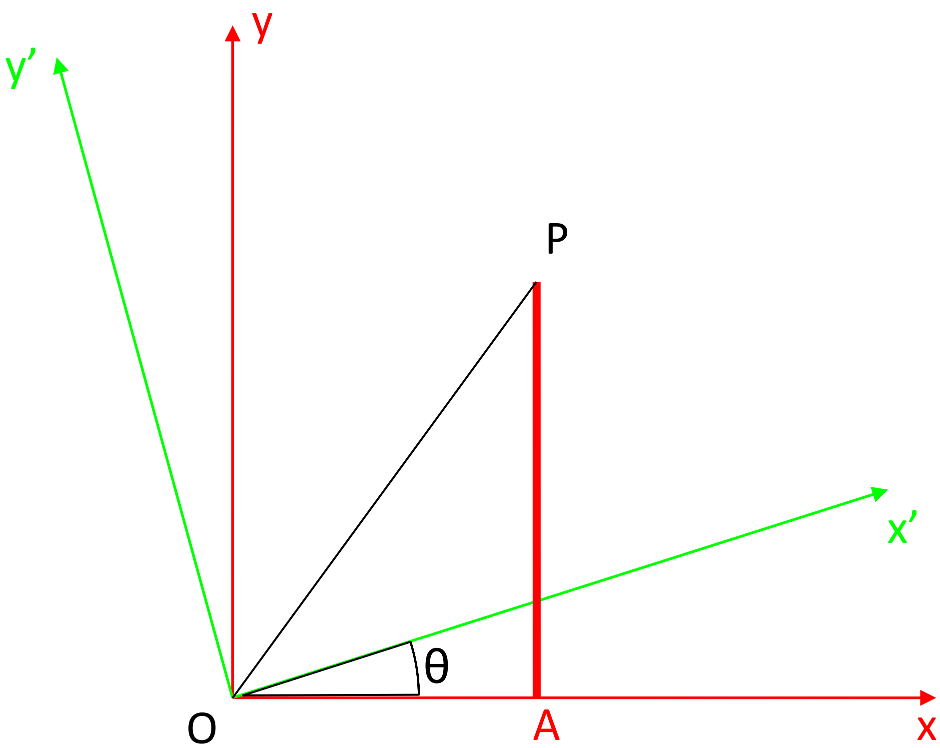
- Let's derive the formula for a 2D rotation



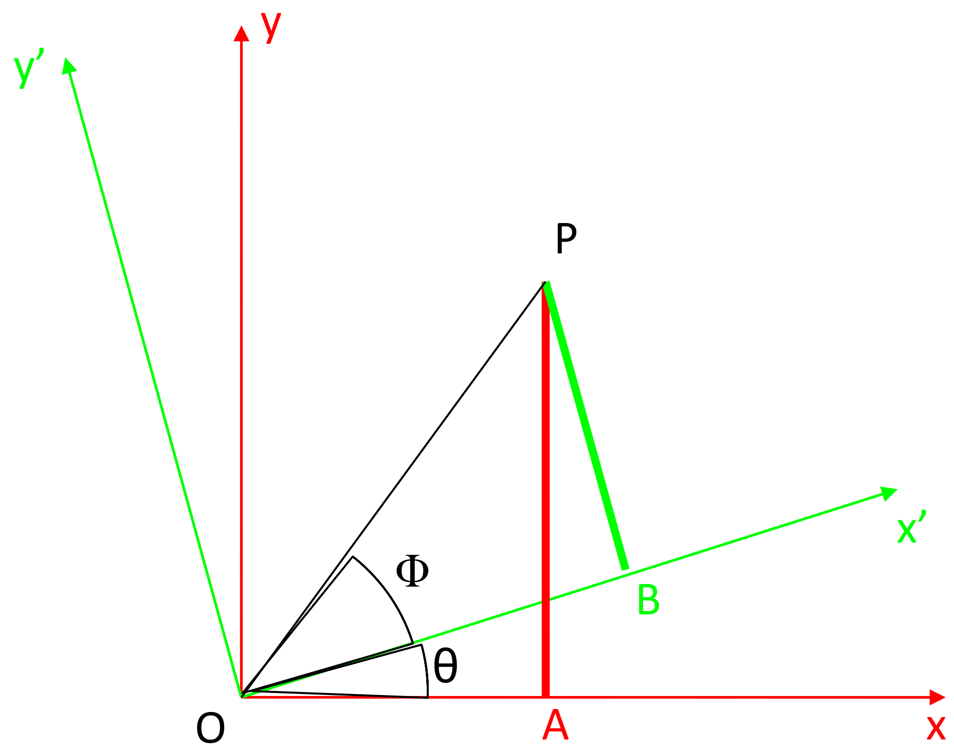
Rotations in 2D



Rotations in 2D

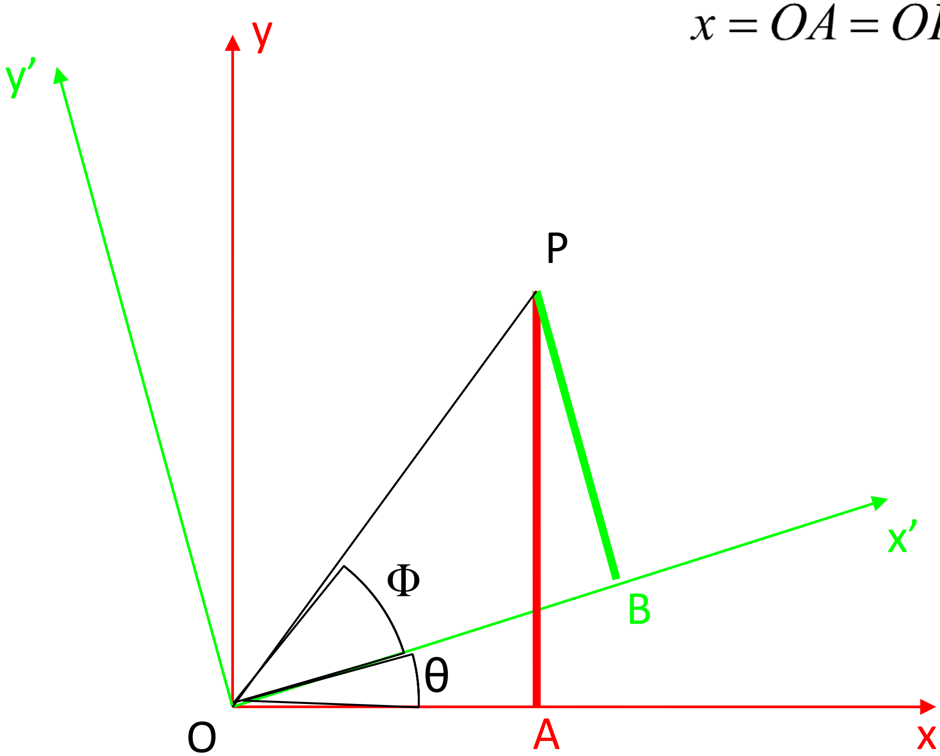


Rotations in 2D



Rotations in 2D

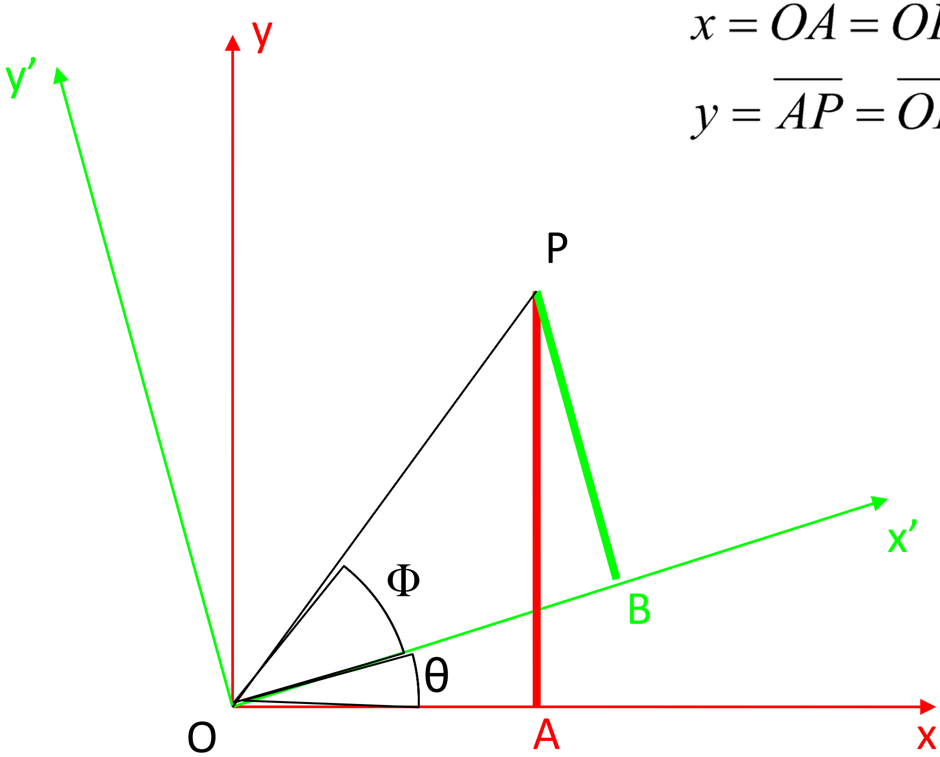
$$x = \overline{OA} = \overline{OP}\cos(\theta + \phi)$$



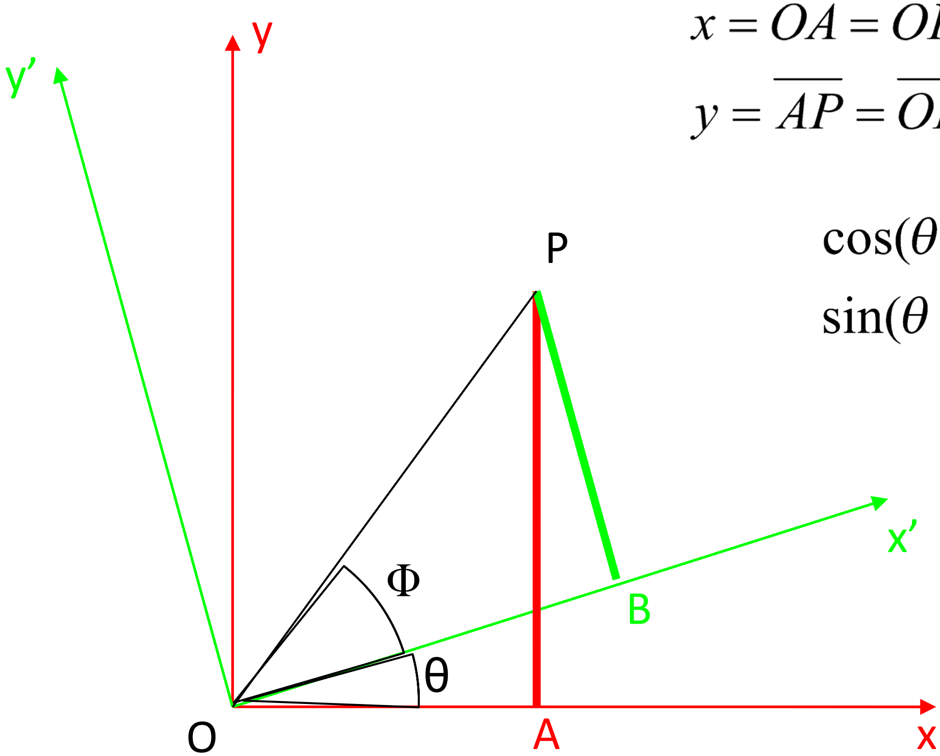
Rotations in 2D

$$x = \overline{OA} = \overline{OP}\cos(\theta + \phi)$$

$$y = \overline{AP} = \overline{OP}\sin(\theta + \phi)$$



Rotations in 2D



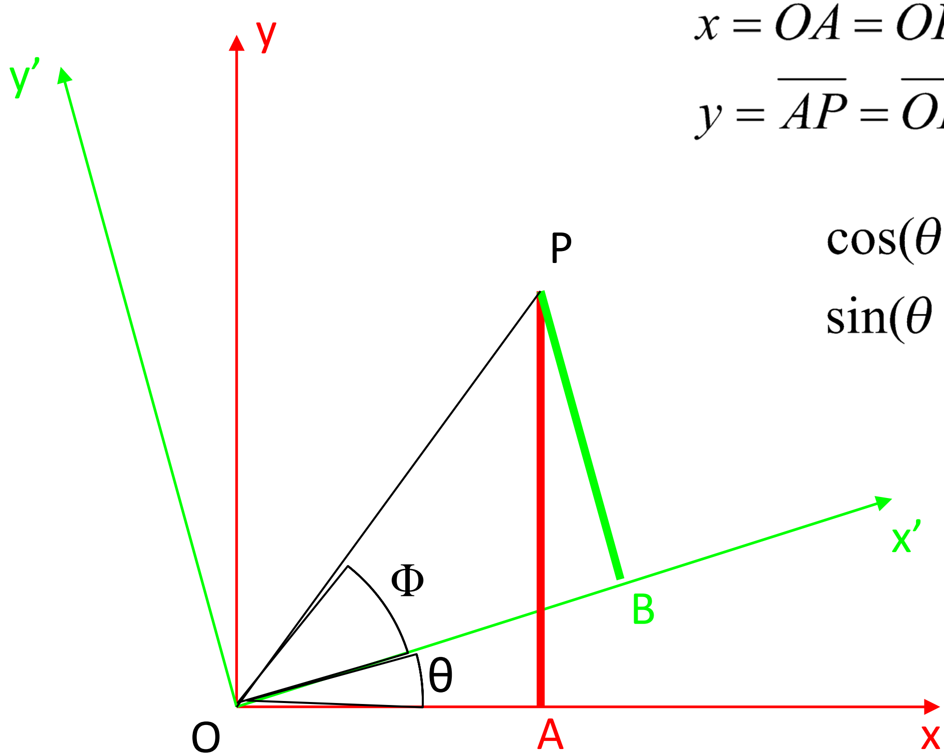
$$x = \overline{OA} = \overline{OP}\cos(\theta + \phi)$$

$$y = \overline{AP} = \overline{OP}\sin(\theta + \phi)$$

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

$$\sin(\theta + \phi) = \cos\theta\sin\phi + \sin\theta\cos\phi$$

Rotations in 2D



$$x = \overline{OA} = \overline{OP}\cos(\theta + \phi)$$

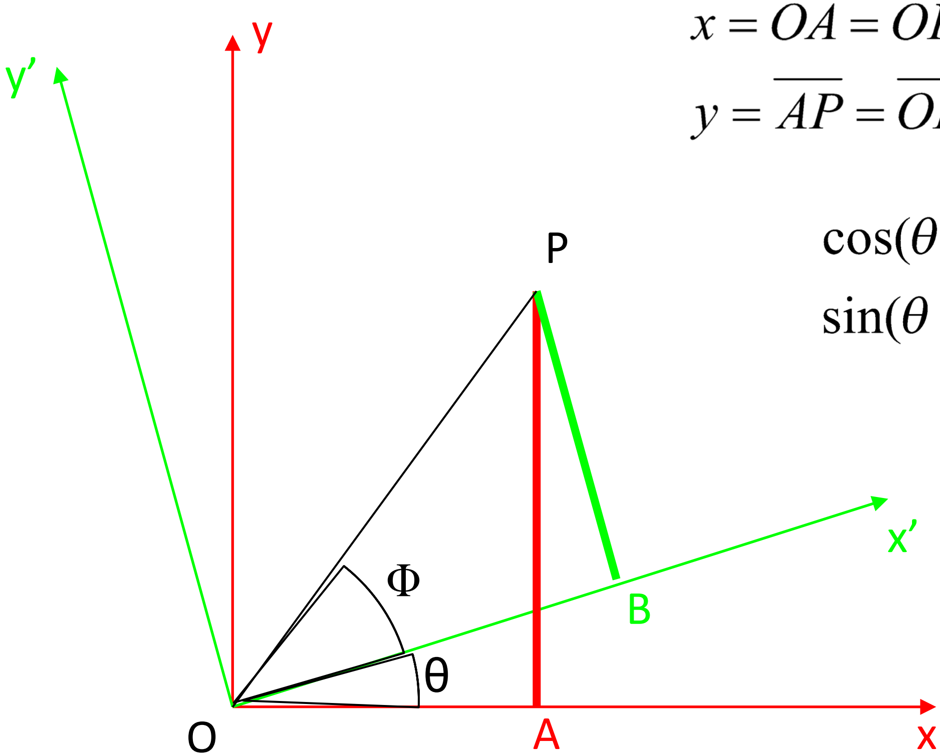
$$y = \overline{AP} = \overline{OP}\sin(\theta + \phi)$$

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

$$\sin(\theta + \phi) = \cos\theta\sin\phi + \sin\theta\cos\phi$$

$$x = \overline{OP}\cos\phi\cos\theta - \overline{OP}\sin\phi\sin\theta$$

Rotations in 2D



$$x = \overline{OA} = \overline{OP}\cos(\theta + \phi)$$

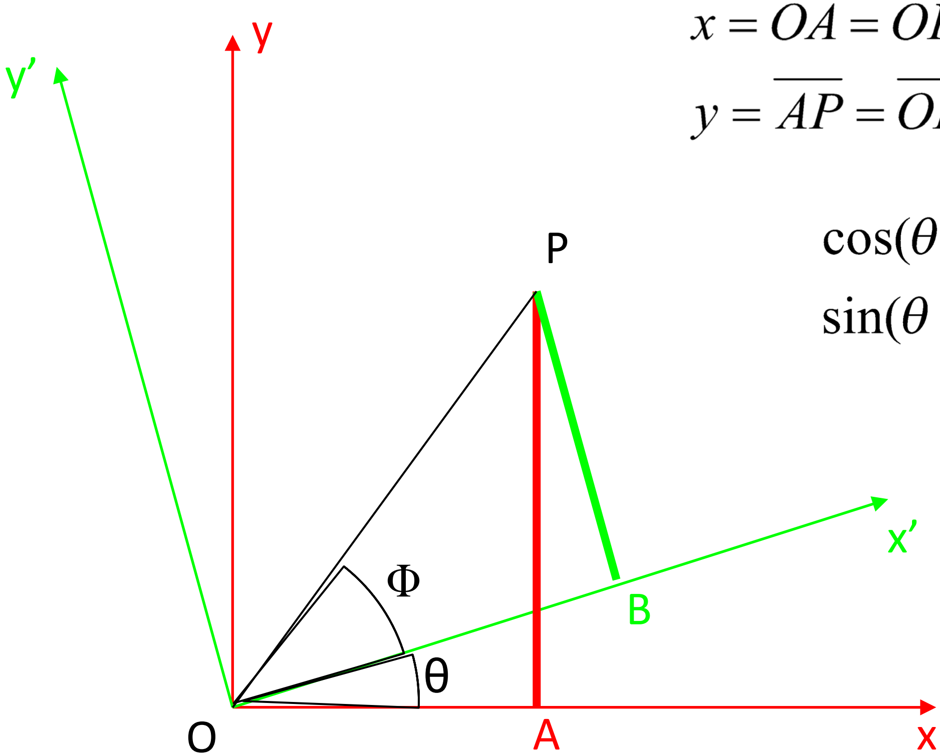
$$y = \overline{AP} = \overline{OP}\sin(\theta + \phi)$$

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

$$\sin(\theta + \phi) = \cos\theta\sin\phi + \sin\theta\cos\phi$$

$$x = \underbrace{\overline{OP}\cos\phi}_{x'}\cos\theta - \underbrace{\overline{OP}\sin\phi}_{y'}\sin\theta$$

Rotations in 2D

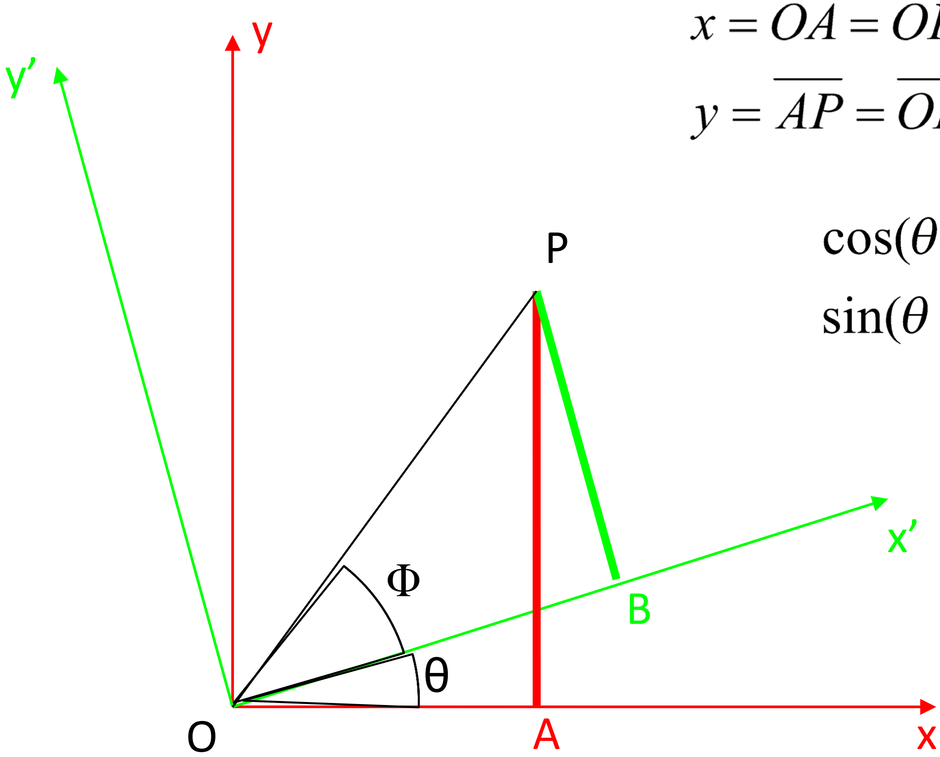


$$x = \overline{OA} = \overline{OP}\cos(\theta + \phi)$$
$$y = \overline{AP} = \overline{OP}\sin(\theta + \phi)$$

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

$$x = \underbrace{\overline{OP}\cos\phi}_{x'}\cos\theta - \underbrace{\overline{OP}\sin\phi}_{y'}\sin\theta$$
$$= x'\cos\theta - y'\sin\theta$$

Rotations in 2D



$$x = \overline{OA} = \overline{OP}\cos(\theta + \phi)$$

$$y = \overline{AP} = \overline{OP}\sin(\theta + \phi)$$

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

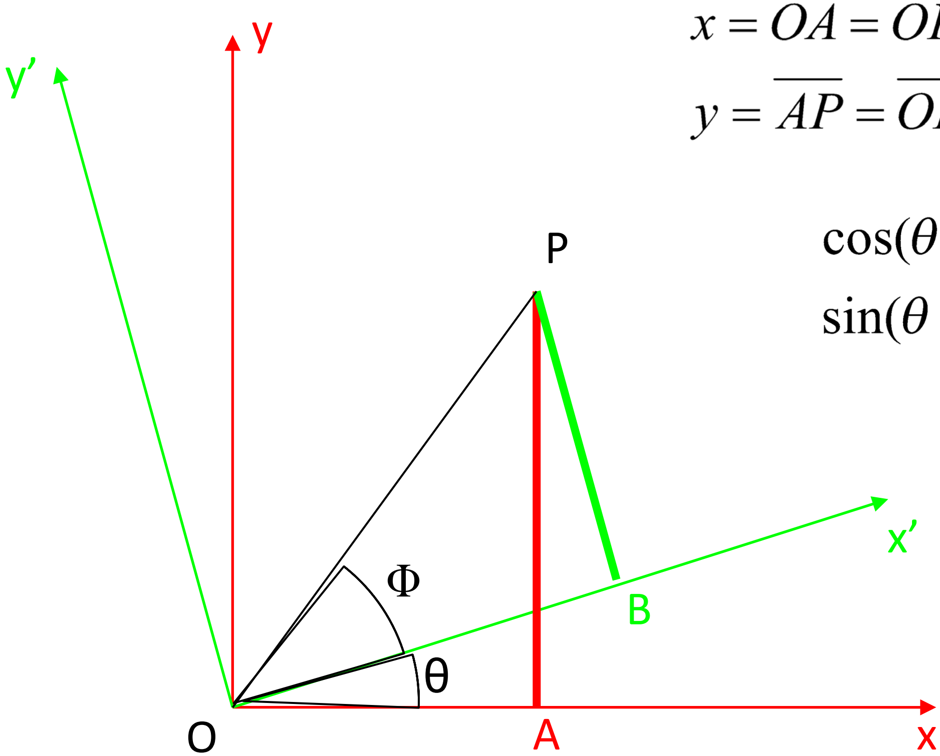
$$\sin(\theta + \phi) = \cos\theta\sin\phi + \sin\theta\cos\phi$$

$$x = \underbrace{\overline{OP}\cos\phi}_{x'}\cos\theta - \underbrace{\overline{OP}\sin\phi}_{y'}\sin\theta$$

$$= x'\cos\theta - y'\sin\theta$$

Similarly, $y = x'\sin\theta + y'\cos\theta$

Rotations in 2D



$$x = \overline{OA} = \overline{OP}\cos(\theta + \phi)$$
$$y = \overline{AP} = \overline{OP}\sin(\theta + \phi)$$

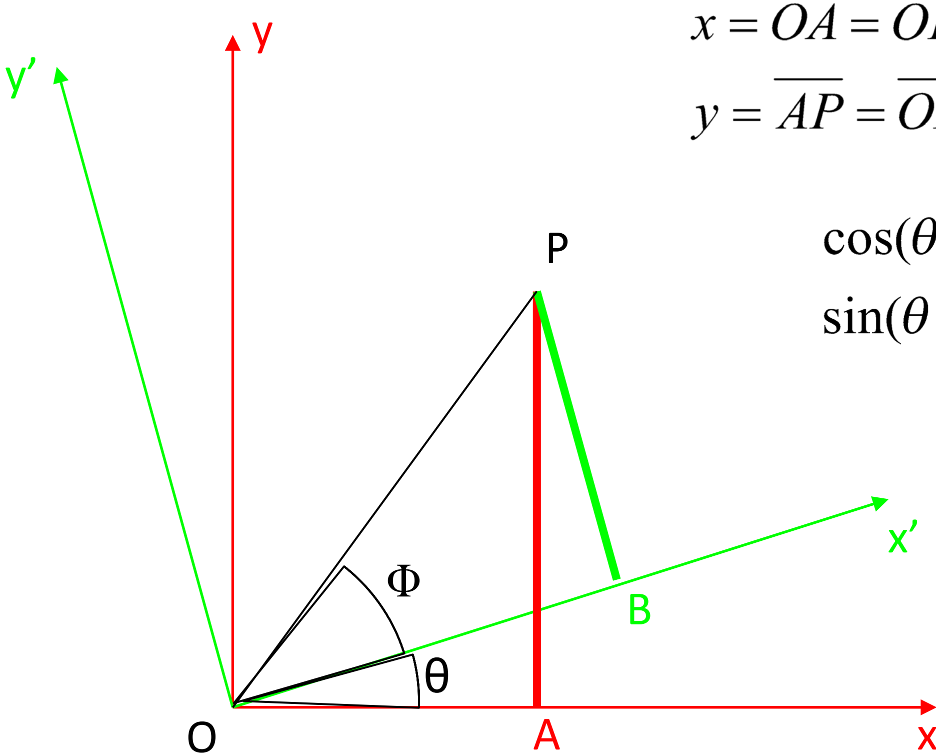
$$\cos(\theta + \phi) = \cos\theta\cos\phi - \sin\theta\sin\phi$$
$$\sin(\theta + \phi) = \cos\theta\sin\phi + \sin\theta\cos\phi$$

$$x = \underbrace{\overline{OP}\cos\phi}_{x'}\cos\theta - \underbrace{\overline{OP}\sin\phi}_{y'}\sin\theta$$
$$= x'\cos\theta - y'\sin\theta$$

Similarly, $y = x'\sin\theta + y'\cos\theta$

So $$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

Rotations in 2D



$$x = \overline{OA} = \overline{OP}\cos(\theta + \phi)$$

$$y = \overline{AP} = \overline{OP}\sin(\theta + \phi)$$

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

$$\sin(\theta + \phi) = \cos\theta\sin\phi + \sin\theta\cos\phi$$

$$x = \underbrace{\overline{OP}\cos\phi}_{x'}\cos\theta - \underbrace{\overline{OP}\sin\phi}_{y'}\sin\theta$$

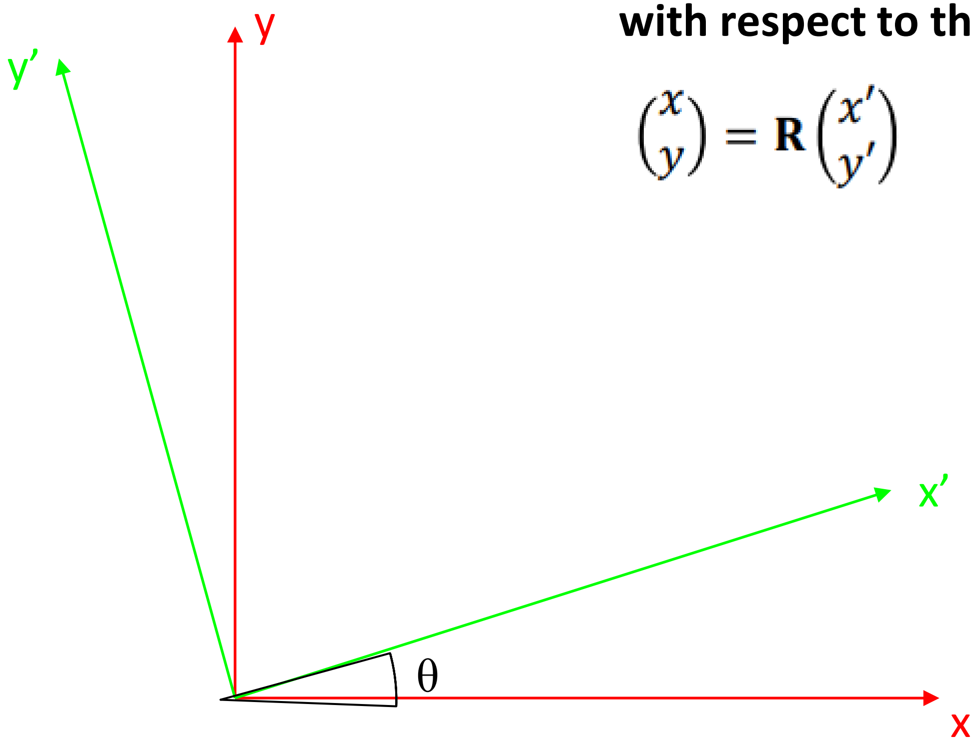
$$= x'\cos\theta - y'\sin\theta$$

Similarly, $y = x'\sin\theta + y'\cos\theta$

So $$\begin{pmatrix} x \\ y \end{pmatrix} = \underbrace{\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}}_{\mathbf{R}} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

$$\boxed{\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}}$$

Rotations in 2D

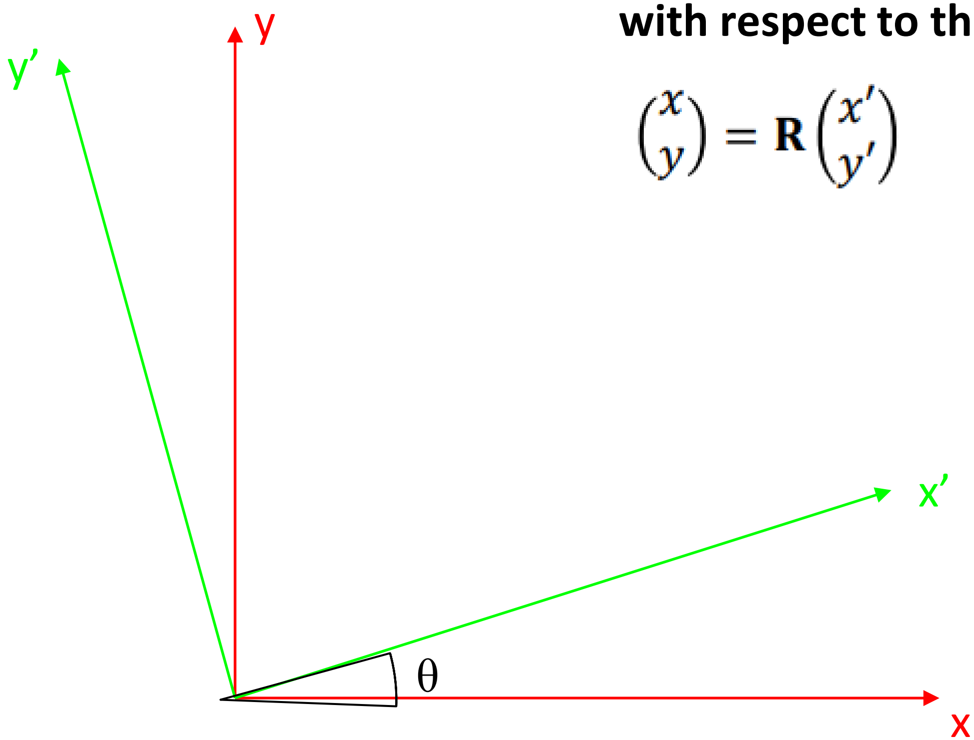


R describes the orientation of the primed frame with respect to the unprimed frame.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{R} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

- **R** is orthonormal
  - Rows, columns are orthogonal ($\mathbf{r}_1 \cdot \mathbf{r}_2 = 0$, $\mathbf{c}_1 \cdot \mathbf{c}_2 = 0$)
  - (in both directions you get $\cos\theta\sin\theta - \cos\theta\sin\theta$)
  - Transpose is the inverse; $\mathbf{R}\mathbf{R}^T = \mathbf{I}$
  - Determinant is $|\mathbf{R}| = 1$

Rotations in 2D



R describes the orientation of the primed frame with respect to the unprimed frame.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{R} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

- **R** is orthonormal
  - Rows, columns are orthogonal ($\mathbf{r}_1 \cdot \mathbf{r}_2 = 0$, $\mathbf{c}_1 \cdot \mathbf{c}_2 = 0$)
  - (in both directions you get $\cos\theta\sin\theta - \cos\theta\sin\theta$)
  - Transpose is the inverse; $\mathbf{R}\mathbf{R}^T = \mathbf{I}$
  - Determinant is $|\mathbf{R}| = 1$

WE CARE BECAUSE THIS MEANS THAT $\begin{pmatrix} x' \\ y' \end{pmatrix} = \mathbf{R}^{T} \begin{pmatrix} x \\ y \end{pmatrix}$

Homogeneous Coordinates

- Points can be represented using homogeneous coordinates
  - This simply means to append a 1 as an extra element
  - If the 3rd element becomes ≠ 1, we divide through by it

$$\tilde{\mathbf{x}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} sx \\ sy \\ s \end{pmatrix}$$

Homogeneous Coordinates

- Points can be represented using homogeneous coordinates
  - This simply means to append a 1 as an extra element
  - If the 3rd element becomes ≠ 1, we divide through by it

$$\tilde{\mathbf{x}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} sx \\ sy \\ s \end{pmatrix}$$

- Effectively, vectors that differ only by scale are considered to be equivalent

Homogeneous Coordinates

- Points can be represented using homogeneous coordinates
  - This simply means to append a 1 as an extra element
  - If the 3rd element becomes ≠ 1, we divide through by it

$$\tilde{\mathbf{x}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} sx \\ sy \\ s \end{pmatrix}$$

- Effectively, vectors that differ only by scale are considered to be equivalent
- This simplifies transform equations; instead of

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \mathbf{R} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \qquad \mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t}$$

Homogeneous Coordinates

- Points can be represented using homogeneous coordinates
  - This simply means to append a 1 as an extra element
  - If the 3rd element becomes ≠ 1, we divide through by it

$$\tilde{\mathbf{x}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} sx \\ sy \\ s \end{pmatrix}$$

- Effectively, vectors that differ only by scale are considered to be equivalent
- This simplifies transform equations; instead of

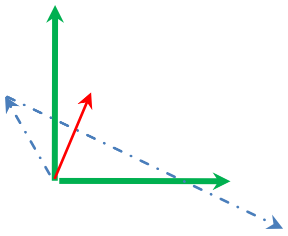
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \mathbf{R} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \qquad \mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t}$$

- we have

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & & \mathbf{t} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \qquad \widetilde{\mathbf{x}'} = \mathbf{H}\tilde{\mathbf{x}}$$

Example

- Transform the 2D point $x = (10, 20)^T$ using a rotation of 45 degrees and translation of (+40, -30).

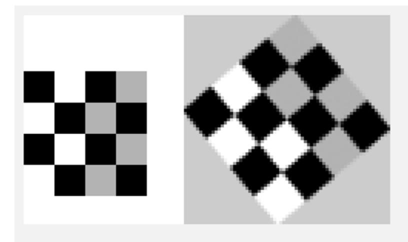


1) Point in Homogeneous Coordinates?
2) Rotation Matrix R?
3) Translation Matrix T?
4) Full Transformation Matrix?
5) How do we apply this transformation to the point?

Other 2D-2D Transforms

- Scaled (similarity) transform
  - preserves angles but not distances

$$\begin{pmatrix} x_B \\ y_B \\ 1 \end{pmatrix} = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_A \\ y_A \\ 1 \end{pmatrix}$$



- Affine transform
  - Models rotation, translation, scaling, shearing, and reflection

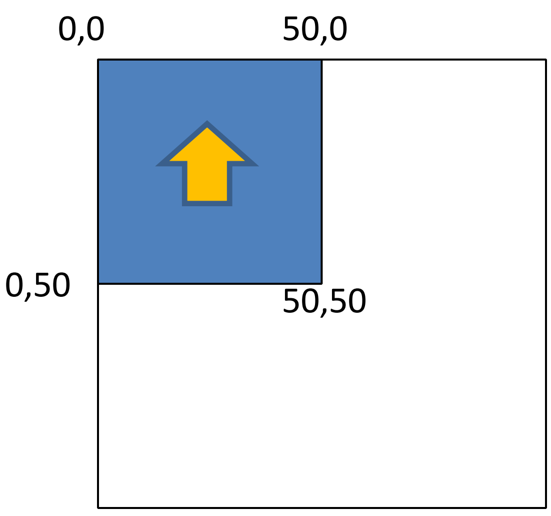
$$\begin{pmatrix} x_B \\ y_B \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_A \\ y_A \\ 1 \end{pmatrix}$$

  - How many degrees of freedom?
    How many pairs of corresponding points needed to calculate transformation?

Example

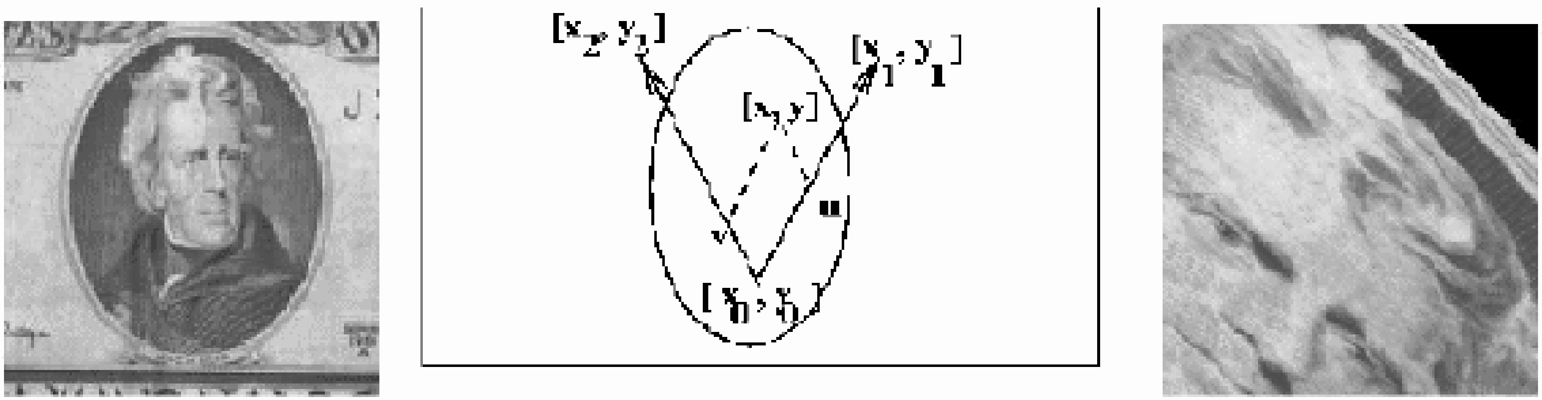
- Image "A" is modified by the affine transform below. Sketch image "B"

$$\begin{pmatrix} x_B \\ y_B \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0.25 & 1.5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_A \\ y_A \\ 1 \end{pmatrix}$$



A

Example Affine Warp



Distorted face of Andrew Jackson extracted from a $20 bill by defining an affine mapping with shear.

from http://www.cse.msu.edu/~stockman/CV

Projective Transform (Homography)

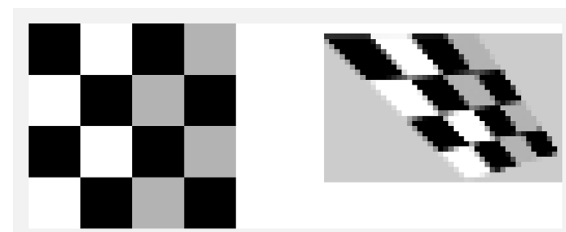
- Most general type of linear 2D-2D transform
- H is an arbitrary 3x3 matrix

$$\tilde{\mathbf{x}}' = \mathbf{H}\,\tilde{\mathbf{x}}$$

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- We still need to divide by the 3rd element

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 / x_3 \\ x_2 / x_3 \\ 1 \end{pmatrix}$$



As we will see later, a homography maps points from the projection of one plane to the projection of another plane

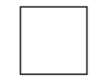
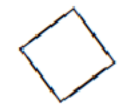
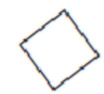
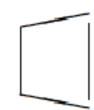
Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

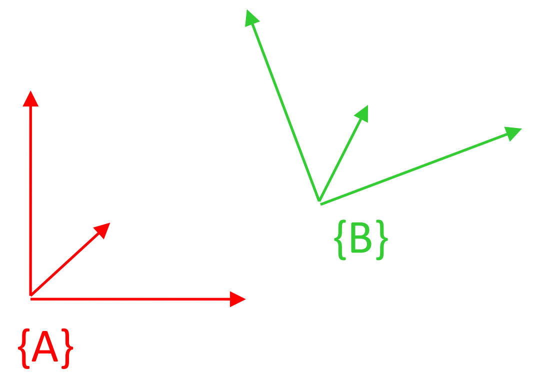
Table 2.1 Hierarchy of 2D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The 2×3 matrices are extended with a third $[\mathbf{0}^T\ 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

From Szeliski, Computer Vision: Algorithms and Applications

3D-3D Coordinate Transforms

3D Coordinate Systems

- Coordinate frames
  - Denote as {A}, {B}, etc
  - Examples: camera, world

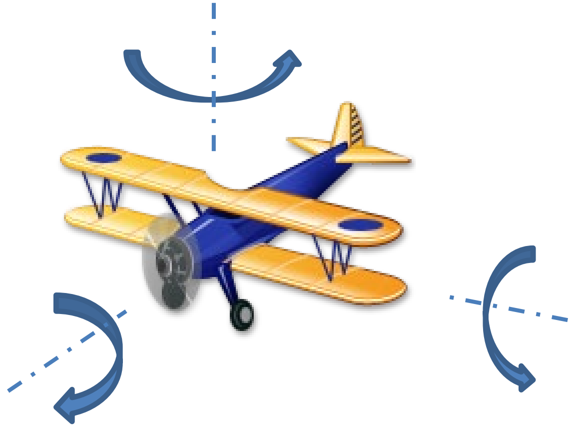


- The pose of {B} with respect to {A} is described by
  - Translation vector **t**
  - Rotation matrix **R**
- Spoilers: Rotation is a 3x3 matrix
  - It represents 3 angles… with 9 numbers – why so many???

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

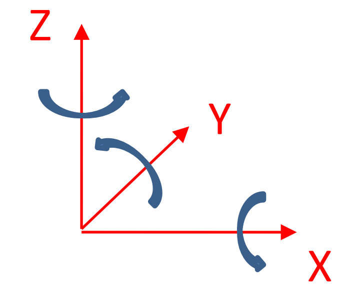
Rotations in 3D

- A 3D rotation has only 3 degrees of freedom
  - I.e., it takes 3 numbers to describe the orientation of an object in the world
  - Think of “roll”, “pitch”, “yaw” for an airplane



XYZ angles to represent rotations

- One way to represent a 3D rotation is by doing successive rotations about the X,Y, and Z axes
- BUT...



XYZ angles to represent rotations

The result depends on the order in which the transforms are applied; i.e., XYZ or ZYX

Easy demo: Hold your phone in front of you facing you.

Rotate around Z 90* then around X 90*
Reset
Rotate around X 90* then around Z 90*

XYZ angles to represent rotations

Some orientations can be represented by multiple XYZ angles

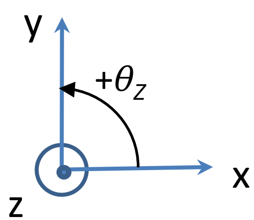
(X=0, Y=90, Z=0) vs (X=90, Y=90, Z=90)

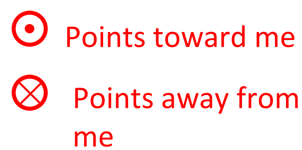
Alternative

Instead of representing orientation as three 2D rotation **angles**, we'll describe an orientation as a matrix composed from three **3x3** 2D rotation **matrices**

XYZ Angles

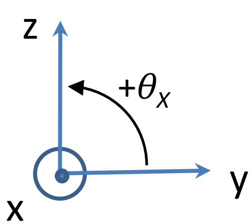
- Rotation about the Z axis





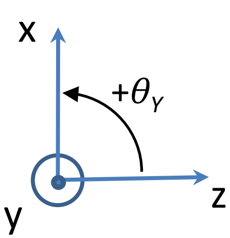
$$\begin{pmatrix} {}^Bx \\ {}^By \\ {}^Bz \end{pmatrix} = \begin{pmatrix} \cos\theta_Z & -\sin\theta_Z & 0 \\ \sin\theta_Z & \cos\theta_Z & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^Ax \\ {}^Ay \\ {}^Az \end{pmatrix}$$

- Rotation about the X axis



$$\begin{pmatrix} {}^Bx \\ {}^By \\ {}^Bz \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_X & -\sin\theta_X \\ 0 & \sin\theta_X & \cos\theta_X \end{pmatrix} \begin{pmatrix} {}^Ax \\ {}^Ay \\ {}^Az \end{pmatrix}$$

- Rotation about the Y axis



$$\begin{pmatrix} {}^Bx \\ {}^By \\ {}^Bz \end{pmatrix} = \begin{pmatrix} \cos\theta_Y & 0 & \sin\theta_Y \\ 0 & 1 & 0 \\ -\sin\theta_Y & 0 & \cos\theta_Y \end{pmatrix} \begin{pmatrix} {}^Ax \\ {}^Ay \\ {}^Az \end{pmatrix}$$

Note signs are different than in the other cases

3D Rotation Matrix

- We can concatenate the 3 rotations to yield a single 3x3 rotation matrix; e.g.,

$$
{}^{A}_{B}R_{XYZ}\left(\theta_X, \theta_Y, \theta_Z\right) = R_Z(\theta_Z)\; R_Y(\theta_Y)\; R_X(\theta_X)
$$

$$
= \begin{pmatrix} cz & -sz & 0 \\ sz & cz & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cy & 0 & sy \\ 0 & 1 & 0 \\ -sy & 0 & cy \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & cx & -sx \\ 0 & sx & cx \end{pmatrix}
$$

where

$$
cx = \cos(\theta_X),\ sy = \sin(\theta_Y), \text{etc}
$$

Result: A unique matrix for each orientation!

- Note: we use the convention that to rotate a vector, we pre-multiply it; i.e., **v’ = R v**
  – This means that if **R** = $\mathbf{R}_Z\,\mathbf{R}_Y\,\mathbf{R}_X$, we actually apply the X rotation first, then the Y rotation, then the Z rotation

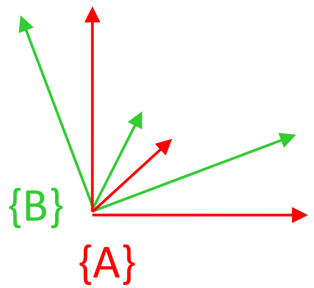
3D Rotation Matrix

- **R** can represent a rotational transformation of one frame to another

$$ {}^{B}_{A}\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} $$

- We can rotate a vector represented in frame *A* to obtain its representation in frame *B*

$$ {}^{B}\mathbf{v} = {}^{B}_{A}\mathbf{R} \; {}^{A}\mathbf{v} $$



- Note: as in 2D, rotation matrices are orthonormal so the inverse of a rotation matrix is just its transpose

$$ \left({}^{B}_{A}\mathbf{R}\right)^{-1} = \left({}^{B}_{A}\mathbf{R}\right)^{T} = {}^{A}_{B}\mathbf{R} $$

Notation

- For vectors, such as this, the leading superscript represents the coordinate frame that the vector is expressed in

$$ {}^{A}\mathbf{v} = \begin{pmatrix} {}^{A}x \\ {}^{A}y \\ {}^{A}z \end{pmatrix} $$

- For transforms, such as this, this matrix represents a rotational transformation of frame *A* to frame *B*
  - The leading subscript indicates “from”
  - The leading superscript indicates “to”

$$ {}^{B}_{A}\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} $$

3D Rotation Matrix

- The elements of **R** are direction cosines (the projections of unit vectors from one frame onto the unit vectors of the other frame)

$$
{}^B_A\mathbf{R} = \begin{pmatrix} \hat{\mathbf{x}}_A \cdot \hat{\mathbf{x}}_B & \hat{\mathbf{y}}_A \cdot \hat{\mathbf{x}}_B & \hat{\mathbf{z}}_A \cdot \hat{\mathbf{x}}_B \\ \hat{\mathbf{x}}_A \cdot \hat{\mathbf{y}}_B & \hat{\mathbf{y}}_A \cdot \hat{\mathbf{y}}_B & \hat{\mathbf{z}}_A \cdot \hat{\mathbf{y}}_B \\ \hat{\mathbf{x}}_A \cdot \hat{\mathbf{z}}_B & \hat{\mathbf{y}}_A \cdot \hat{\mathbf{z}}_B & \hat{\mathbf{z}}_A \cdot \hat{\mathbf{z}}_B \end{pmatrix}
$$

- The columns of **R** are the unit vectors of A, expressed in the B frame

$$
{}^B_A\mathbf{R} = \begin{pmatrix} \begin{pmatrix} {}^B\hat{\mathbf{x}}_A \end{pmatrix} & \begin{pmatrix} {}^B\hat{\mathbf{y}}_A \end{pmatrix} & \begin{pmatrix} {}^B\hat{\mathbf{z}}_A \end{pmatrix} \end{pmatrix}
$$

- The rows of **R** are the unit vectors of {B} expressed in {A}

$$
{}^B_A\mathbf{R} = \begin{pmatrix} \begin{pmatrix} {}^A\hat{\mathbf{x}}_B{}^T \end{pmatrix} \\ \begin{pmatrix} {}^A\hat{\mathbf{y}}_B{}^T \end{pmatrix} \\ \begin{pmatrix} {}^A\hat{\mathbf{z}}_B{}^T \end{pmatrix} \end{pmatrix}
$$

$$
{}^B_A R\, {}^A\hat{x}_A = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} r_{11} \\ r_{21} \\ r_{31} \end{pmatrix} = {}^B\hat{x}_A
$$

Python: Creating a Rotation Matrix

```
import numpy as np

ax, ay, az = 0.1, -0.2, 0.3 # radians

sx, sy, sz = np.sin(ax), np.sin(ay), np.sin(az)
cx, cy, cz = np.cos(ax), np.cos(ax), np.cos(az)

Rx = np.array(((1, 0, 0), (0, cx, -sx), (0, sx, cx)))
Ry = np.array(((cy, 0, sy), (0, 1, 0), (-sy, 0, cy)))
Rz = np.array(((cz, -sz, 0), (sz, cz, 0), (0, 0, 1)))

# Apply X rotation first, then Y, then Z
R = Rz @ Ry @ Rx     # Use @ for matrix mult
print(R)



# Apply Z rotation first, then Y, then X
R = Rx @ Ry @ Rz
print(R)
```

Python: Creating a Rotation Matrix

```
import numpy as np

ax, ay, az = 0.1, -0.2, 0.3 # radians

sx, sy, sz = np.sin(ax), np.sin(ay), np.sin(az)
cx, cy, cz = np.cos(ax), np.cos(ax), np.cos(az)

Rx = np.array(((1, 0, 0), (0, cx, -sx), (0, sx, cx)))
Ry = np.array(((cy, 0, sy), (0, 1, 0), (-sy, 0, cy)))
Rz = np.array(((cz, -sz, 0), (sz, cz, 0), (0, 0, 1)))

# Apply X rotation first, then Y, then Z
R = Rz @ Ry @ Rx     # Use @ for matrix mult
print(R)
```

```
[[ 0.95056379 -0.31299183 -0.15934508]
 [ 0.29404384  0.94470249 -0.153792  ]
 [ 0.19866933  0.09933467  0.99003329]]
```

```
# Apply Z rotation first, then Y, then X
R = Rx @ Ry @ Rz
print(R)
```

Python: Creating a Rotation Matrix

```python
import numpy as np

ax, ay, az = 0.1, -0.2, 0.3 # radians

sx, sy, sz = np.sin(ax), np.sin(ay), np.sin(az)
cx, cy, cz = np.cos(ax), np.cos(ax), np.cos(az)

Rx = np.array(((1, 0, 0), (0, cx, -sx), (0, sx, cx)))
Ry = np.array(((cy, 0, sy), (0, 1, 0), (-sy, 0, cy)))
Rz = np.array(((cz, -sz, 0), (sz, cz, 0), (0, 0, 1)))

# Apply X rotation first, then Y, then Z
R = Rz @ Ry @ Rx    # Use @ for matrix mult
print(R)
```

```
[[ 0.95056379 -0.31299183 -0.15934508]
 [ 0.29404384  0.94470249 -0.153792  ]
 [ 0.19866933  0.09933467  0.99003329]]
```

```python
# Apply Z rotation first, then Y, then X
R = Rx @ Ry @ Rz
print(R)
```

```
[[ 0.95056379 -0.29404384 -0.19866933]
 [ 0.27509585  0.95642509 -0.09933467]
 [ 0.21835066  0.03695701  0.99003329]]
```

Transforming a Point

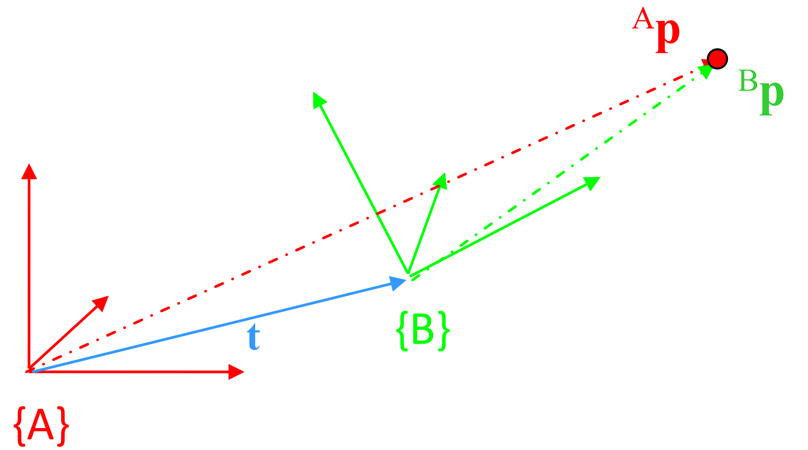
- We can use **R,t** to transform a point from coordinate frame {B} to frame {A}

$$ {}^{A}\mathbf{p} = {}^{A}_{B}\mathbf{R}\ {}^{B}\mathbf{p} + \mathbf{t} $$

- Where
  - ${}^{A}\mathbf{p}$ is the representation of **p** in frame {A}
  - ${}^{B}\mathbf{p}$ is the representation of **p** in frame {B}

- Note

$\mathbf{t}$ is the translation of B's origin in the A frame, ${}^{A}\mathbf{t}_{Borg}$



Homogeneous Coordinates

- We can represent the transformation with a single matrix multiplication if we write **p** in homogeneous coordinates
  - This simply means to append a 1 as a 4th element
  - If the 4th element ever becomes ≠ 1, we divide through by it

- Then

The leading superscript indicates what coordinate frame the point is represented in

$$ {}^{A}\mathbf{p} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} sx \\ sy \\ sz \\ s \end{pmatrix} $$

$$ {}^{A}\mathbf{p} = {}^{A}_{B}\mathbf{H} \; {}^{B}\mathbf{p} \quad \text{where} \quad {}^{A}_{B}\mathbf{H} = \begin{bmatrix} & {}^{A}_{B}\mathbf{R} & & {}^{A}\mathbf{t}_{Borg} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Notation Note: Cancel leading subscript with trailing superscript

Example

- In coordinate frame A, point **p** is (-1,0,1)
- Frame *A* is located at (1,2,4) with respect to B, and is rotated 90 degrees about the x axis with respect to frame *B*
- What is point **p** in frame *B*?

Example

- In coordinate frame A, point **p** is (-1,0,1)
- Frame *A* is located at (1,2,4) with respect to B, and is rotated 90 degrees about the x axis with respect to frame *B*
- What is point **p** in frame *B*?

We want to do

$$ {}^{B}\mathbf{p} = {}^{B}_{A}\mathbf{H} \; {}^{A}\mathbf{p} $$

Example

- In coordinate frame A, point **p** is (-1,0,1)
- Frame *A* is located at (1,2,4) with respect to B, and is rotated 90 degrees about the x axis with respect to frame *B*
- What is point **p** in frame *B*?

We want to do

$${}^{B}\mathbf{p} = {}^{B}_{A}\mathbf{H}\ {}^{A}\mathbf{p}$$

where

$${}^{B}_{A}\mathbf{H} = \begin{bmatrix} {}^{B}_{A}\mathbf{R} & {}^{B}\mathbf{t}_{Aorg} \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

Example

- In coordinate frame A, point **p** is (-1,0,1)
- Frame *A* is located at (1,2,4) with respect to B, and is rotated 90 degrees about the x axis with respect to frame *B*
- What is point **p** in frame *B*?

We want to do

$$ {}^{B}\mathbf{p} = {}^{B}_{A}\mathbf{H} \ {}^{A}\mathbf{p} $$

where

$$ {}^{B}_{A}\mathbf{H} = \begin{bmatrix} {}^{B}_{A}\mathbf{R} & {}^{B}\mathbf{t}_{Aorg} \\ 0\ 0\ 0 & 1 \end{bmatrix} $$

$$ {}^{B}_{A}\mathbf{R} = \mathbf{R}_x(90) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(90) & -\sin(90) \\ 0 & \sin(90) & \cos(90) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} $$

Python and Numpy: Transforming a point

```
# Construct 4x4 transformation matrix to transform A to B

# Rotation matrix of A with respect to B.
R_A_B = np.array(((1,0,0),(0,0,-1),(0,1,0)))    # Get 3x3 matrix

# The translation is the origin of A in B.
tAorg_B = np.array([[1,2,4]]).T   # Get as a 3x1 matrix

# H_A_B means transform A to B.
H_A_B = np.block([[R_A_B, tAorg_B], [0,0,0,1]])  # Get 4x4 matrix

# Define a point in the A frame, as [x,y,z,1].
P_A = np.array([[-1,0,1,1]]).T    # Get as a 4x1 matrix

# Convert point to B frame.
P_B = H_A_B @ P_A
```

Inverse Transformations

- The **matrix inverse** is the inverse transformation

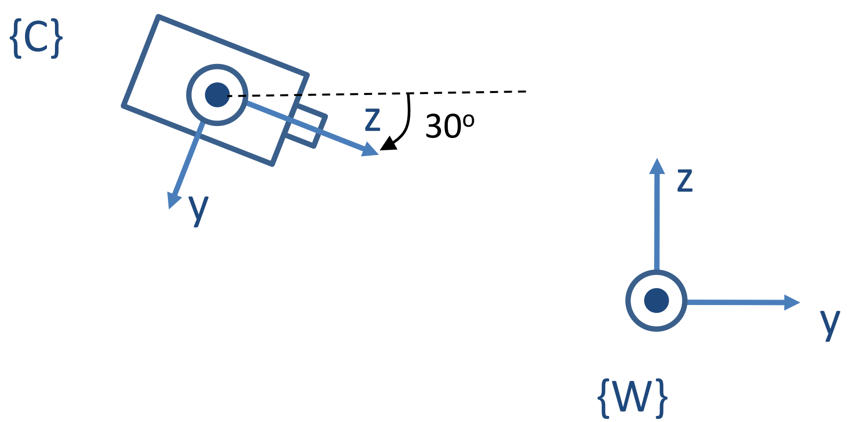
$$ {}^{A}_{B}\mathbf{H} = \left({}^{B}_{A}\mathbf{H} \right)^{-1} $$

- Note – unlike rotation matrices, the inverse of a full 4x4 homogeneous transformation matrix is **not the transpose**

$$ {}^{A}_{B}\mathbf{H} \neq \left({}^{B}_{A}\mathbf{H} \right)^{T} $$

Example

- A camera is located at point (0,-5,3) with respect to the world. The camera is tilted down by 30 degrees from the horizontal. Find the transformation from {W} to {C}. (Note that in "the world" Z is up (X-Y ground plane) but in "the camera", Z is out (X-Y image plane)!)



Summary

- 3D rigid body transformations (i.e., a rotation and translation) can be represented by a single 4x4 homogeneous transformation matrix
- A 3D rotation is represented uniquely by a 3x3 rotation matrix
- 3D rotations can also be represented by
  – XYZ angles (the order matters, easy to understand, but not computationally stable)
  – Axis, angle (minimal representation, computationally stable, but axis is not intuitively obvious)

3D-2D Coordinate Transforms

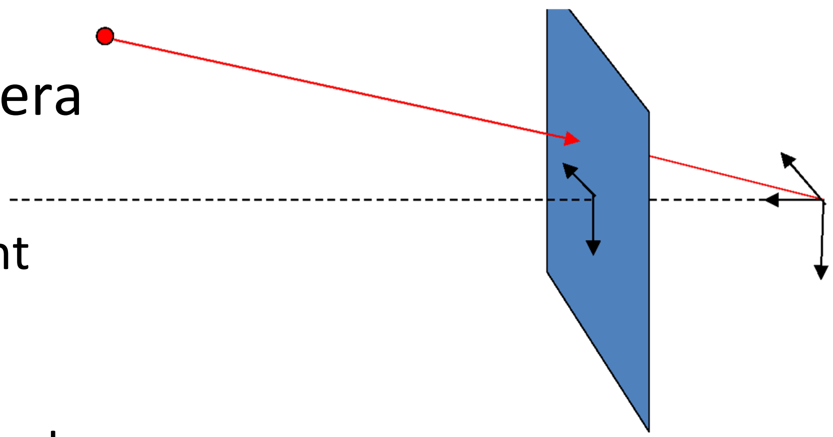
3D to 2D Projections

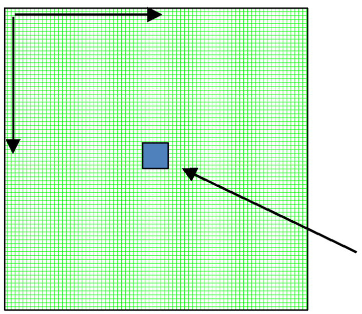
- We have already seen how to project 3D points onto a 2D image
  - We used the pinhole camera model
  - Also the geometry of similar triangles
- Now we will look at how to model this using matrix multiplication

- This will help us better understand and model:
  - Perspective projection
  - Other projection types, such as weak perspective projection
  - Special cases such as the projection of a planar surface

Intrinsic Camera Parameters

- Recall the intrinsic camera parameters, for a pinhole camera model
  - Focal length *f* and sensor element sizes *sx,sy*
    - Or, just focal lengths in pixels *fx,fy*
  - Optical center of the image at pixel location *cx, cy*





- We can capture all the intrinsic camera parameters in a matrix **K**

$$\mathbf{K} = \begin{pmatrix} f/s_x & 0 & c_x \\ 0 & f/s_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad \text{or} \quad \mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

3D to 2D Perspective Transformation

- We can project 3D points onto 2D with a matrix multiplication

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

- We treat the result as a 2D point in homogeneous coordinates. So we divide through by the last element.

$$\tilde{\mathbf{x}} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X/Z \\ Y/Z \\ 1 \end{pmatrix}$$

Complete Perspective Projection

- To project 3D points *represented in the coordinate system attached to the camera*, to the 2D image plane:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \mathbf{K} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^{\mathrm{C}}\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \qquad \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 / x_3 \\ x_2 / x_3 \\ 1 \end{pmatrix} \qquad \mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

- To see this:

$$\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^{\mathrm{C}}\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} f_x X + c_x Z \\ f_y Y + c_y Z \\ Z \end{pmatrix} \sim \begin{pmatrix} f_x X / Z + c_x \\ f_y Y / Z + c_y \\ 1 \end{pmatrix}$$

Extrinsic Camera Matrix

- If 3D points are in world coordinates, we first need to transform them to camera coordinates

$$ {}^{C}\mathbf{P} = {}^{C}_{W}\mathbf{H}\,{}^{W}\mathbf{P} = \begin{pmatrix} {}^{C}_{W}\mathbf{R} & {}^{C}\mathbf{t}_{Worg} \\ \mathbf{0} & 1 \end{pmatrix} {}^{W}\mathbf{P} $$

- We can write this as an extrinsic camera matrix, that does the rotation and translation, then a projection from 3D to 2D

$$ \mathbf{M}_{ext} = \begin{pmatrix} {}^{C}_{W}\mathbf{R} & {}^{C}\mathbf{t}_{Worg} \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix} $$

Complete Perspective Projection

- Projection of a 3D point ${}^{W}\mathbf{P}$ in the world to a point in the pixel image (x_{im},y_{im})

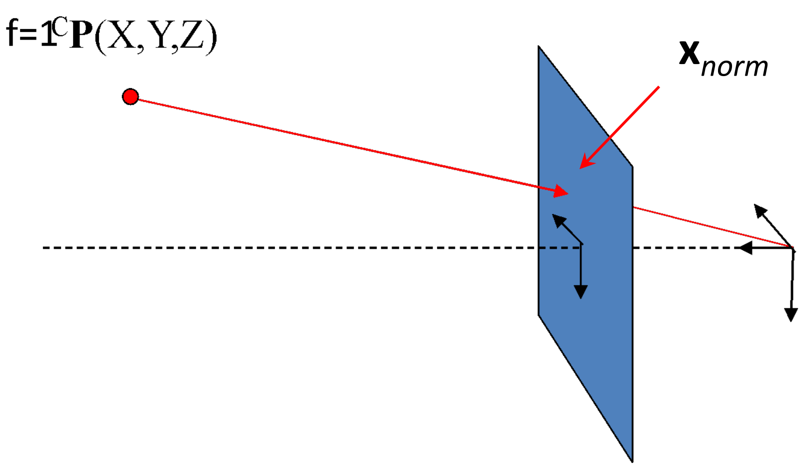
$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \mathbf{K}\,\mathbf{M}_{ext}\; {}^{W}\!\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \qquad x_{im} = x_1 / x_3,\;\; y_{im} = x_2 / x_3$$

  - where **K** is the intrinsic camera parameter matrix
  - and $\mathbf{M}_{ext}$ is the 3x4 matrix given by

$$\mathbf{M}_{ext} = \begin{pmatrix} {}^{C}_{W}\mathbf{R} & {}^{C}\mathbf{t}_{Worg} \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix}$$

Back Projection

- If you have an image point, you can “back project” that point into the scene
- However, the resulting 3D point is not uniquely defined
  - It is actually a ray emanating from the camera center, out through the image point, to infinity
  - Any 3D point along that ray could have projected to the image point



$\mathbf{r} = s\mathbf{x}_{norm}$ is a ray emanating outward to infinity

- Assume that the cameraman image[1] was taken using a camera with focal length = 600 pixels, with cx,cy in middle of image.
  - Find the unit vector in the direction of the man’s eye

[1]A popular test image, can download from many places, such as https://github.com/antimatter15/cameraman/blob/master/cameraman.png

- Assume that the cameraman image[1] was taken using a camera with focal length = 600 pixels, with cx,cy in middle of image.
  – Find the unit vector in the direction of the man's eye
- Solution:
  – The eye is at pixel (ximg=126, yimg=61)
  – Then pimg = K*pn, or pn = Kinv*pimg
  – Let u=unit vector to the eye = pn/norm(pn)

```
img = cv2.imread("cameraman.png")

xeye = 126
yeye = 61
cv2.drawMarker(img, (xeye,yeye), color=(0,0,255),
   markerType=cv2.MARKER_DIAMOND, thickness=3)
cv2.imshow("image", img)
cv2.waitKey(0)

K = np.array([
   [600, 0, 128],
   [0, 600, 128],
   [0, 0, 1]])

pn = np.linalg.inv(K) @ np.array([xeye, yeye, 1])
u = pn / np.linalg.norm(pn)
print(u)
```

u = -0.003313   -0.110976   0.993818

[1]A popular test image, can download from many places, such as https://github.com/antimatter15/cameraman/blob/master/cameraman.png

Special Case

- Small planar patch
  - Often we want to track a small patch on an object
  - We want to know how the image of that patch transforms as the object rotates
- Assume
  - Size of patch small compared to distance -> *weak perspective*
  - Rotation is small -> *small angle approximation*
  - Patch is planar
- It can be shown that the patch undergoes affine transformation

$$\begin{pmatrix} x_B \\ y_B \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_A \\ y_A \\ 1 \end{pmatrix}$$