CS100433
2D and 3D Transformation

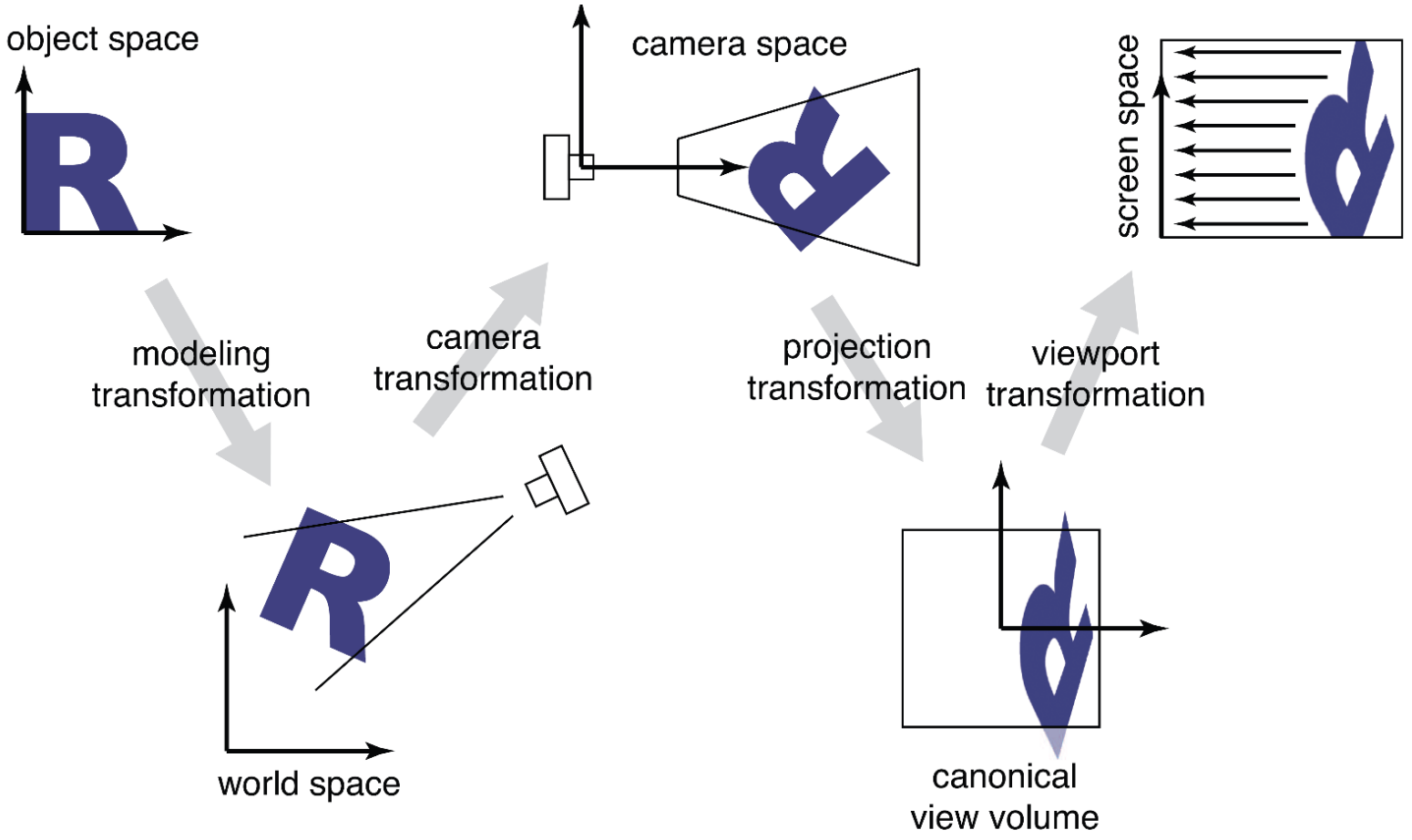
Junqiao Zhao 赵君峤

Department of Computer Science and Technology

College of Electronics and Information Engineering

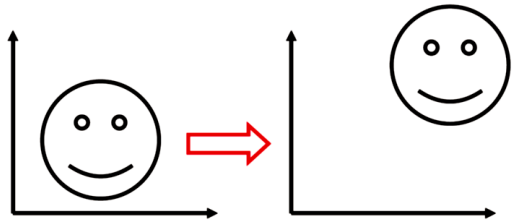
Tongji University

Why transformation?

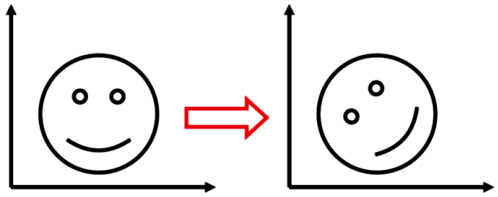


(2013 Steve Marschner)

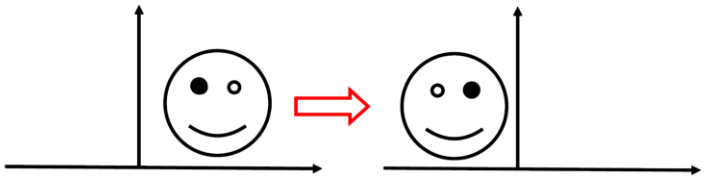
Transformations



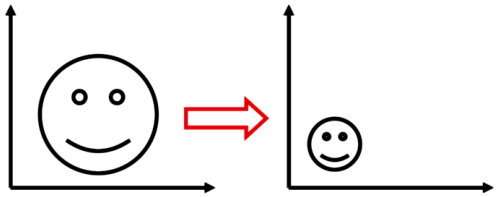
translation



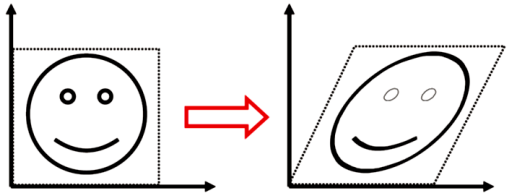
rotation



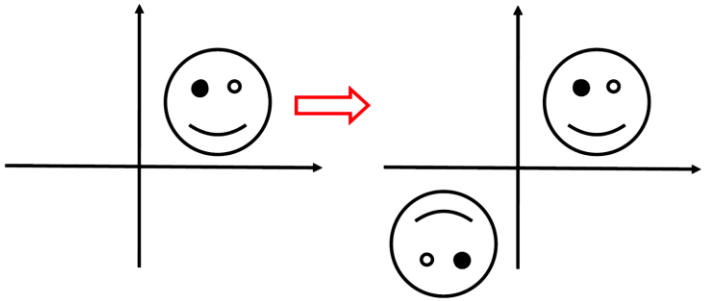
reflection with respect to the y-axis



scaling



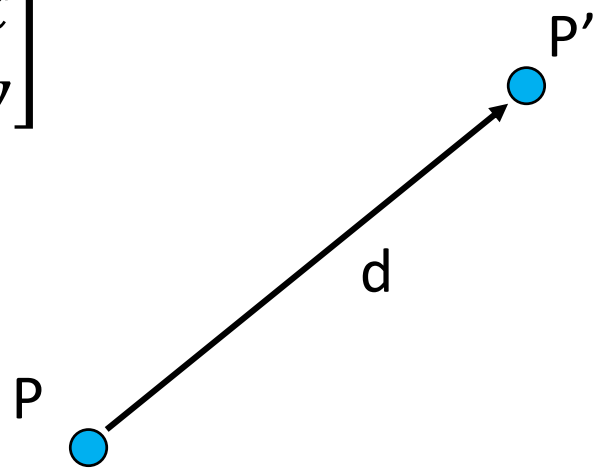
shear



reflection with respect to the origin

Translation

- Move (translate, displace) a point to a new location
- Displacement determined by a vector d
  - $P' = P + d$
  - $P = \begin{bmatrix} x \\ y \end{bmatrix},\ P' = \begin{bmatrix} x' \\ y' \end{bmatrix},\ d = \begin{bmatrix} dx \\ dy \end{bmatrix}$
- Simplest transformation:
  - $T(v) = v + u$
  - Inverse: $T^{-1}(v) = v - u$



Linear Transformations

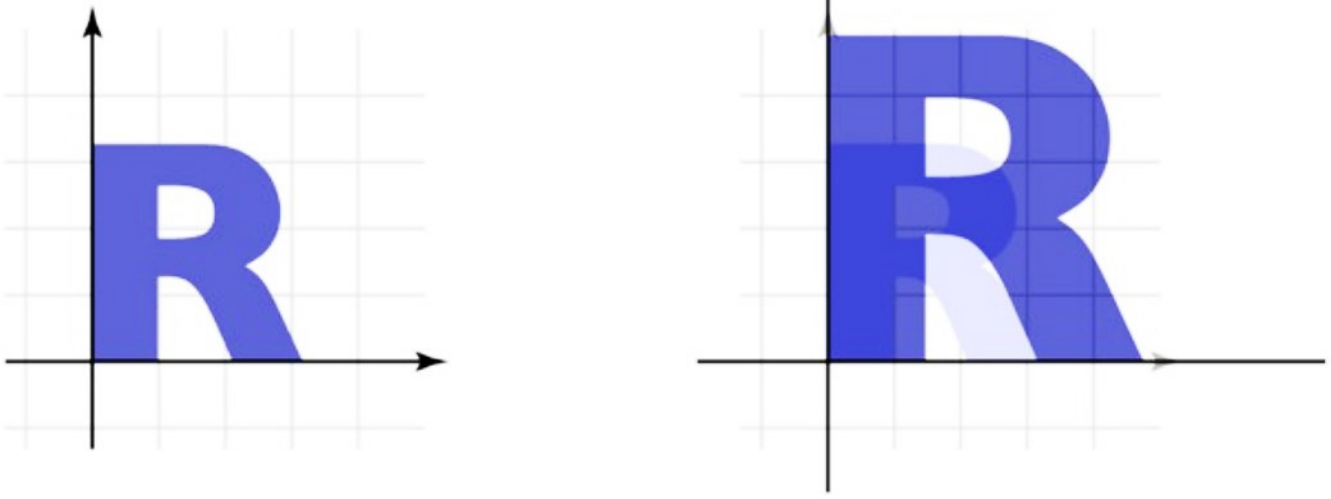
- Such transformations are *linear*:
  - $T(v+u) = T(v) + T(u)$
  - $T(\alpha v) = \alpha T(v)$
  - $T(\alpha v + \beta u) = \alpha T(v) + \beta T(u)$
- Linear Transformations can be represented in *matrix* form
- Rotation, scale, shear, reflection are all *linear*
- Is translation *linear*?

2D Linear Transformations

- $T(v) = Mv = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_v$
- *M* is *a* 2x2 transformation matrix
  - Uniform scale
  - Non-uniform scale
  - Rotation
  - Shear
  - Reflection

Uniform Scale

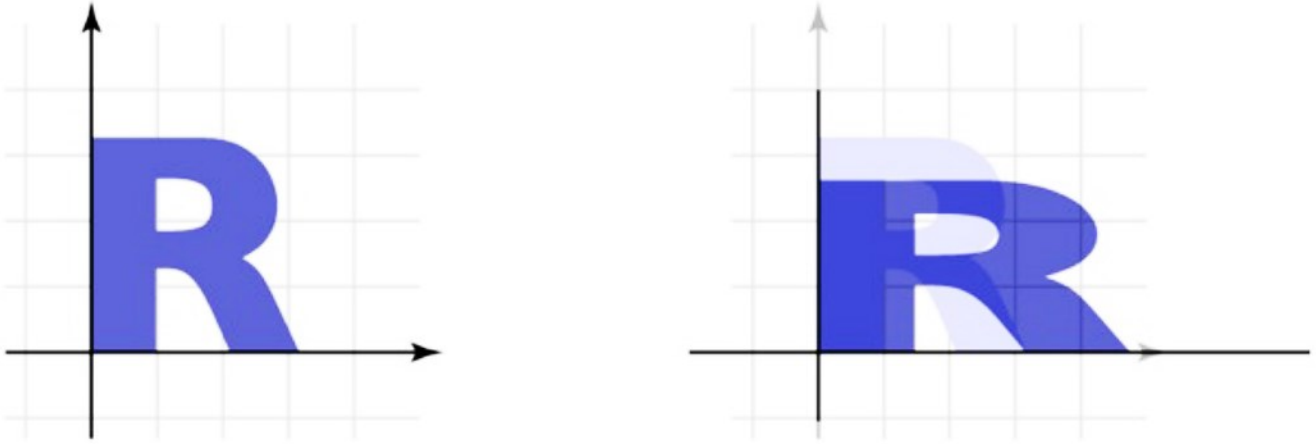
- $\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ xy \end{bmatrix}$



Non-uniform Scale

- $\begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \times x \\ sy \times y \end{bmatrix}$

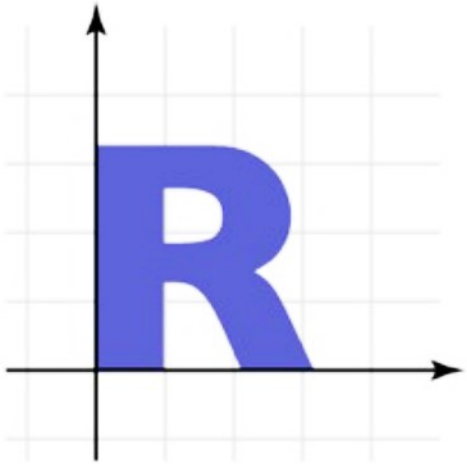
$$\begin{bmatrix} 1.5 & 0 \\ 0 & 0.8 \end{bmatrix}$$

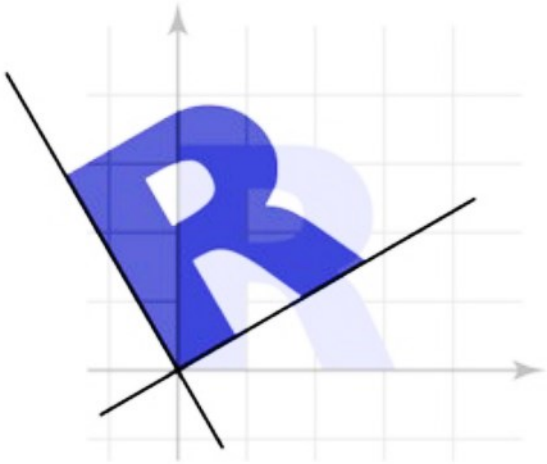


Rotation

- $\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} xcos\theta - ysin\theta \\ xsin\theta + ycos\theta \end{bmatrix}$

$$\begin{bmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{bmatrix}$$



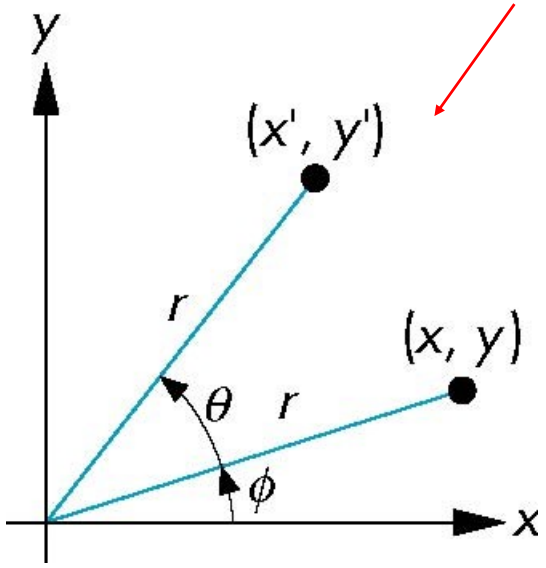


Rotation about the origin

Consider rotation about the origin by θ degrees

- radius stays the same, angle increases by θ

$$x' = r \cos(\phi + \theta)$$
$$y' = r \sin(\phi + \theta)$$



$$x = r \cos \phi$$
$$y = r \sin \phi$$

2D Rotation about the origin

$$x' = r.\cos(\theta + \phi) = r.\cos\phi.\cos\theta - r.\sin\phi.\sin\theta$$

$$y' = r.\sin(\theta + \phi) = r.\cos\phi.\sin\theta + r.\sin\phi.\cos\theta$$

Substituting for r :

$$x = r.\cos\phi$$

$$y = r.\sin\phi$$

Gives us :

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

$$y' = x.\sin\theta + y.\cos\theta$$

Rewriting in Matrix form

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

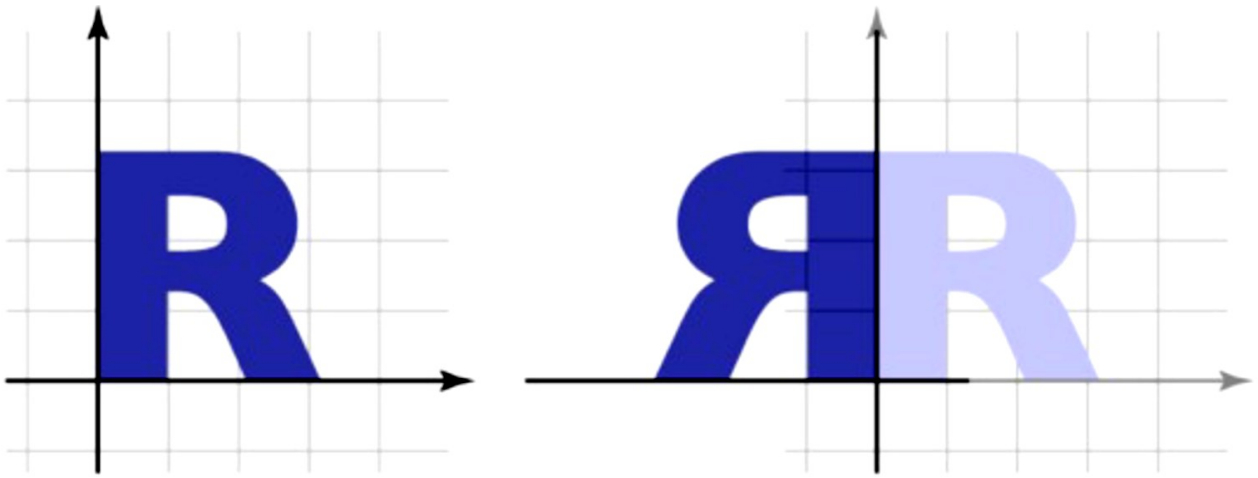
$$y' = x.\sin\theta + y.\cos\theta$$

Rewriting in matrix form gives us :

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

Reflection

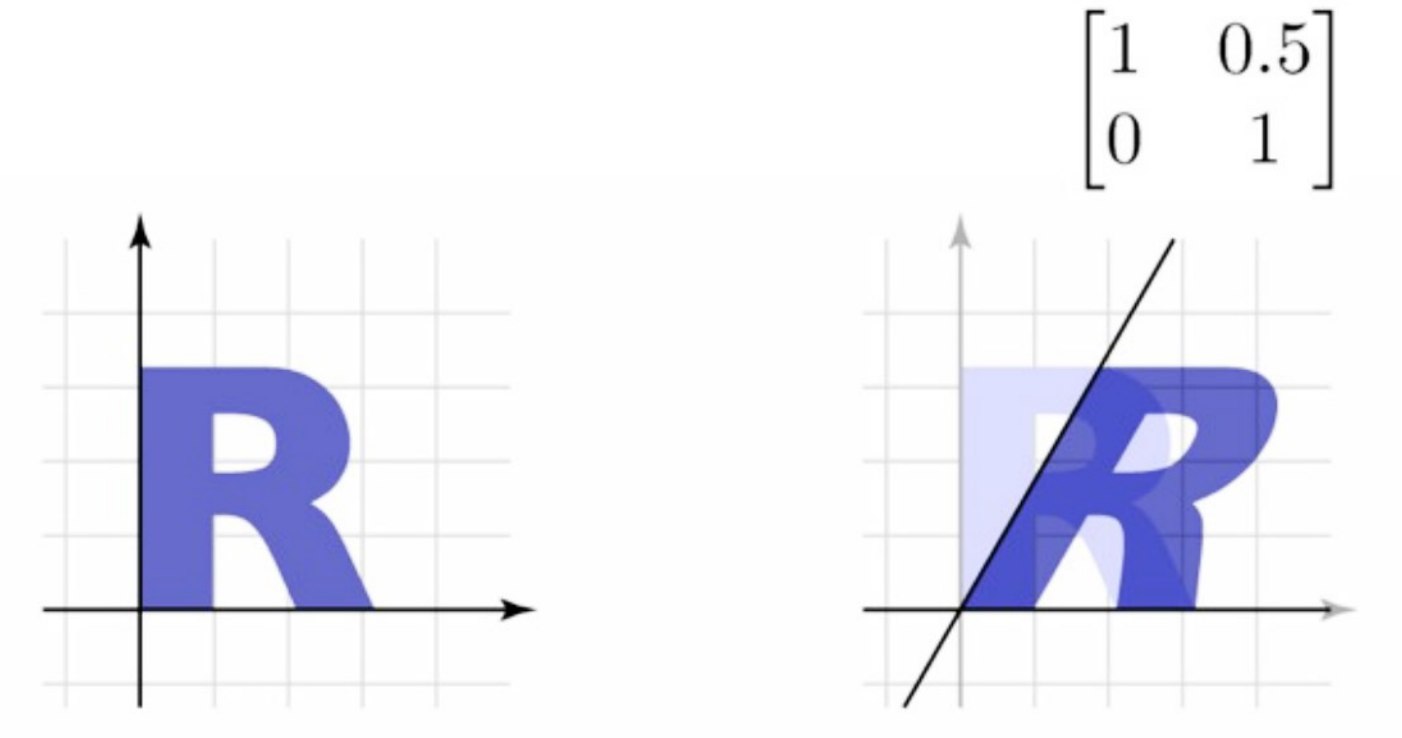
- Can be considered as a special case of non-uniform scale
- $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$



Shear

- $\begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + \alpha y \\ y \end{bmatrix}$

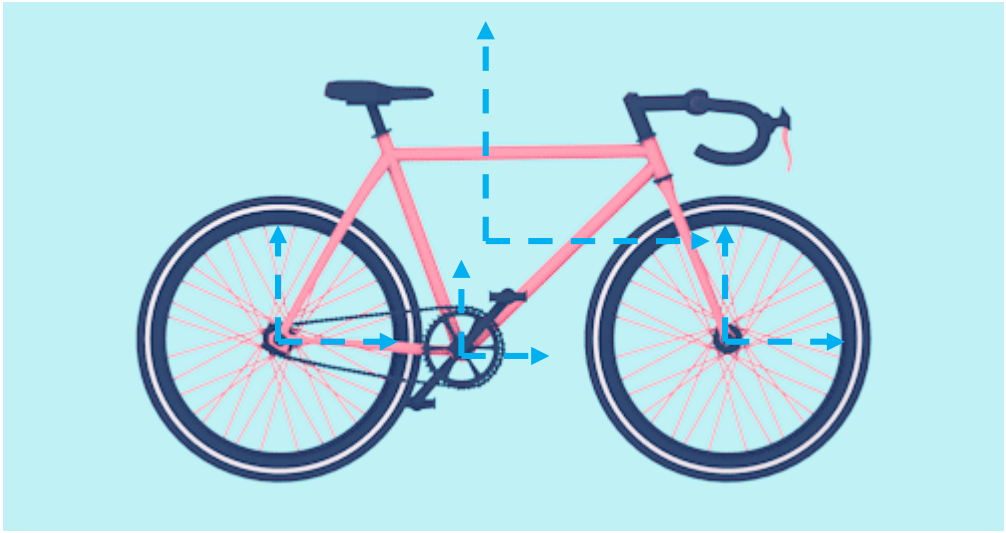
$$\begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}$$



- Questions?

How to **animate** a bicycle?

How to **animate** a bicycle?



Composing transformations

- We “move” an object twice or more
  - $P \rightarrow T(P) = P'$
  - $P' \rightarrow S(P') = P''$
  - $P'' = S(P') = S\big(T(P)\big) = (S \circ T)(P)$
- $S \circ T$ is S compose T
- (**which one is firstly applied?**)
- Translations
  - $T(P) = P + d_T$
  - $S(P) = P + d_S$
  - $(S \circ T)(P) = P + (d_T + d_S)$
  - Therefore, translation by d_T and d_S is translation by $d_T + d_S$
  - Commutative!

Composing transformations

- Linear Transformations
  - $P \rightarrow T(P) = M_T P = P'$
  - $P' \rightarrow S(P') = M_S P' = P''$
  - $(S \circ T)(P) = M_S M_T P$
  - (**which one is firstly applied?**)
- *Be careful!*
  - *Only some are commutative*

Composing transformations

- Translation together with linear transformations - Affine transformation
- Applications require a single framework
  - $T(P) = M_T P + d_T$
  - $S(P) = M_S P + d_S$
  - $(S \circ T)(P) = M_S(M_T P + d_T) + d_S = M_S M_T P + M_S d_T + d_S$
  - Not elegant!

Homogeneous coordinates

- Extra component *w*
  - $V = [x, y, w]^T$
- If $w \neq 0$, divide by it to get Cartesian coordinates of point (*x*/*w*, *y*/*w*, 1)
  - $P = [x, y, 1]^T$
  - $P' = [x', y', 1]^T$
  - $d = [dx, dy, 0]^T$
- Hence
  - $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ 0 \end{bmatrix}$

note that this expression is in Three dimensions and expresses

point = point + vector

Point – point = ?

Can we translate vectors?

Translation Matrix

We can now express translation using a

3 x 3 matrix $\mathbf{T_d}$ in homogeneous coordinates

$\mathbf{p'} = \mathbf{T_d p}$, where

$$\mathrm{T_d} = \mathrm{T_d}(dx, dy) = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + dx \\ y + dy \\ 1 \end{bmatrix}$$

Properties of translations

- $T(0,0) = I$
- $T(s_x, s_y) \cdot T(t_x, t_y) = T(s_x + t_x, s_y + t_y)$
- $T(s_x, s_y) \cdot T(t_x, t_y) = T(t_x, t_y) \cdot T(s_x, s_y)$
- $T^{-1}(s_x, s_y) = T(-s_x, -s_y)$

- Note : translation matrices are *commutative.*

Rotation Matrix

- With Cartesian coordinate frame
  - $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$
- With homogenous coordinate frame

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

$$\begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} xcos\theta - ysin\theta \\ xsin\theta + ycos\theta \\ 1 \end{bmatrix}$$

Properties of rotation

- Rotation matrices are orthogonal, i.e.:
  - $R^{-1}(\theta) = R^{T}(\theta)$
- $R^{-1}(\theta) = R(-\theta)$
- $R(0) = I$
- $\mathrm{Det(R)} = 1$
- $R(\theta) \cdot R(\phi) = R(\theta + \phi)$
- And
- $R(\theta) \cdot R(\phi) = R(\phi) \cdot R(\theta)$
  - But this is only valid when the axis of rotation is the same
  - Not the case in 3D!

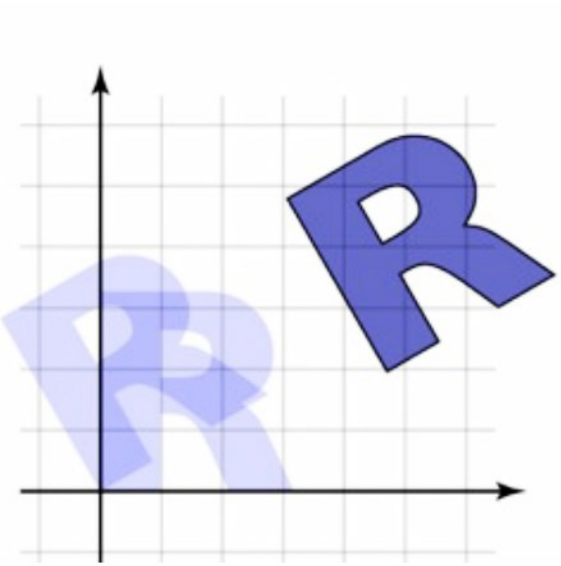
Composing transformations

- Compose translation and linear transformations using homogenous coordinates
- $$\begin{bmatrix} M_S & d_S \\ 0 & 1 \end{bmatrix} \begin{bmatrix} M_T & d_T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P \\ 1 \end{bmatrix} = \begin{bmatrix} M_S M_T P + M_S d_T + d_S \\ 1 \end{bmatrix}$$
- This form is better for implementation because all **Affine Transformations** can be expressed this way and multiple transformations can be concatenated together

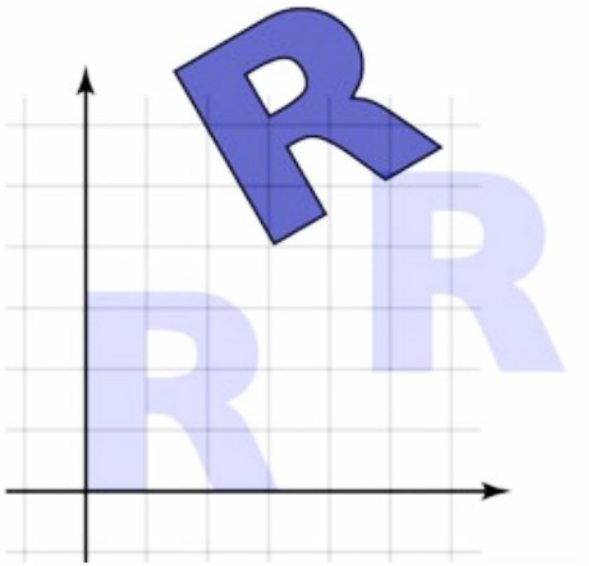
- Questions?

Composite Transformations

- In general NOT commutative

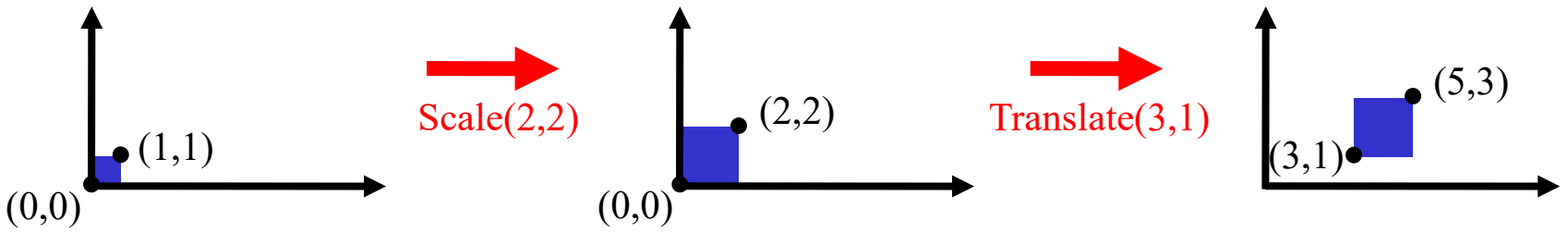


rotate, then translate

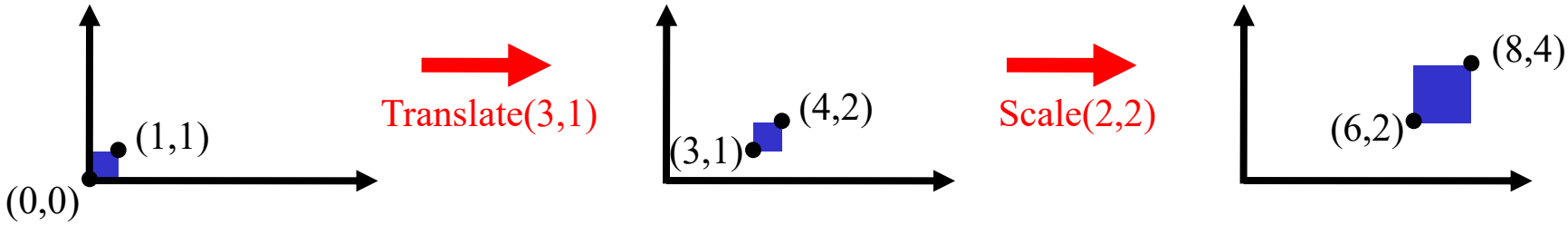


translate, then rotate

Scale then Translate: $p' = T(S\,p) = TS\,p$



Translate then Scale: $p' = S(T\,p) = ST\,p$



Scale then Translate: $p' = T(S\,p) = TS\,p$

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Translate then Scale: $p' = S(T\,p) = ST\,p$

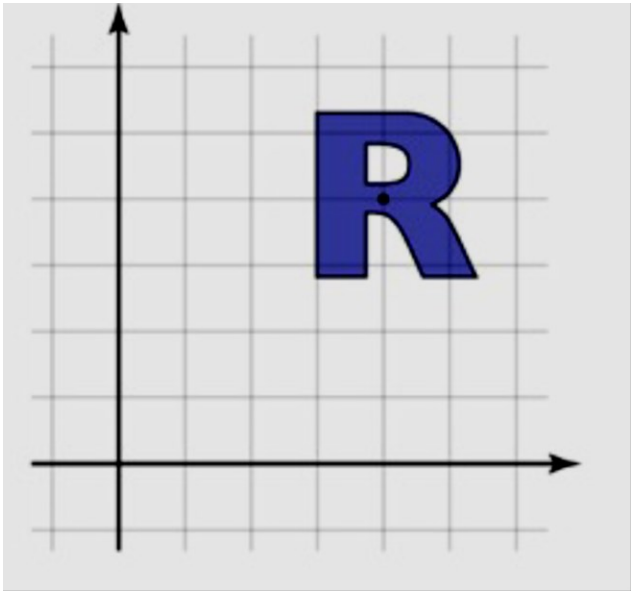
$$ST = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Properties of Affine transformations

- Rotation and translation
  - Angles and distances are preserved
  - Unit cube is always unit cube
  - ***Rigid-Body* transformations or *rigid motion***
- Rotation, translation and scale.
  - Angles & distances not preserved.
  - But parallel lines are.

Want to rotate about a particular point

- We know how to rotate about the origin
  - So translate that point to the origin

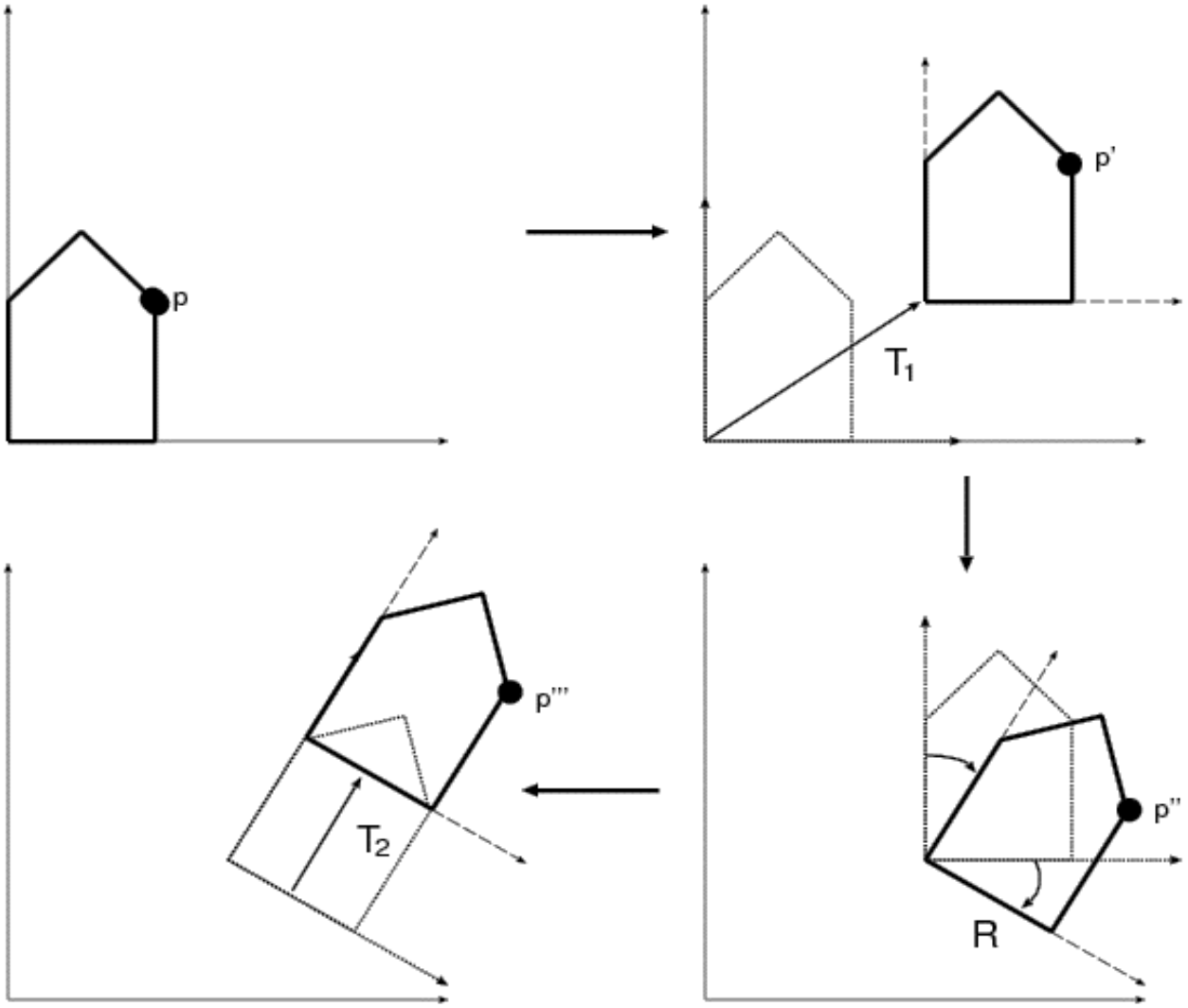


$$M = T^{-1}RT$$

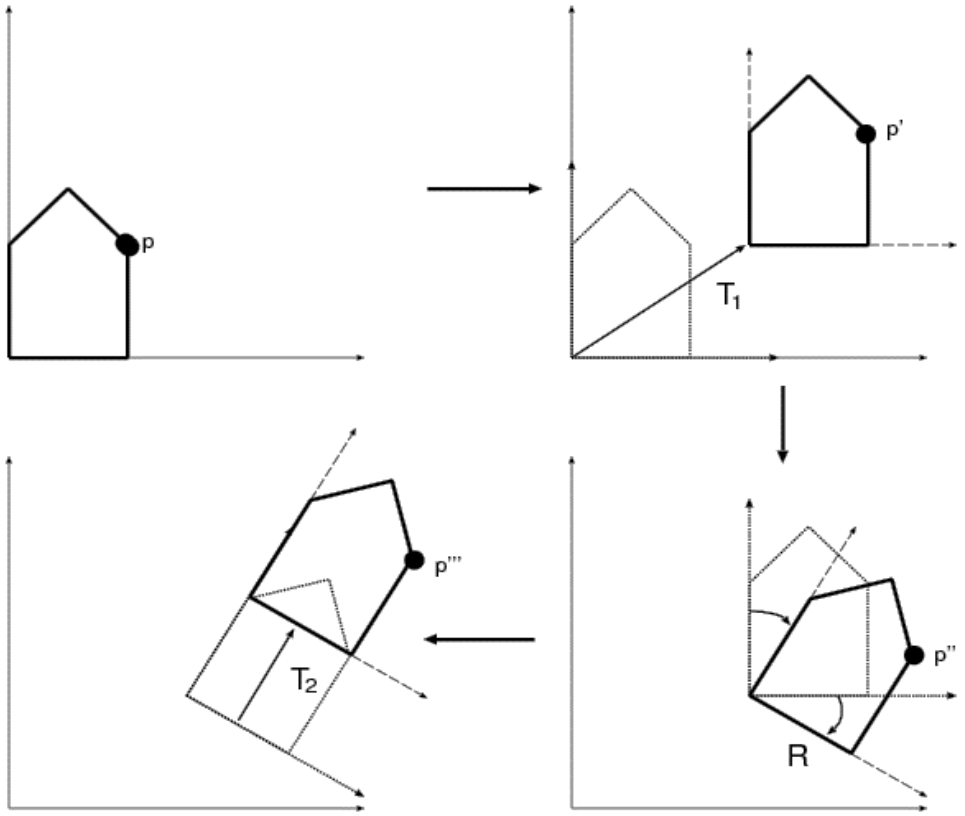
Transformations of local coordinate systems

- Have been discussing transformations as transforming points
- Always need to think the transformation in the world coordinate system
- Sometimes this might not be convenient
  - i.e. rotating objects at its location

Transformations of local coordinate systems - Example



Transformations of local coordinate systems - Example



$p' = T_1\, p$

$p'' = T_1\, R\, p$

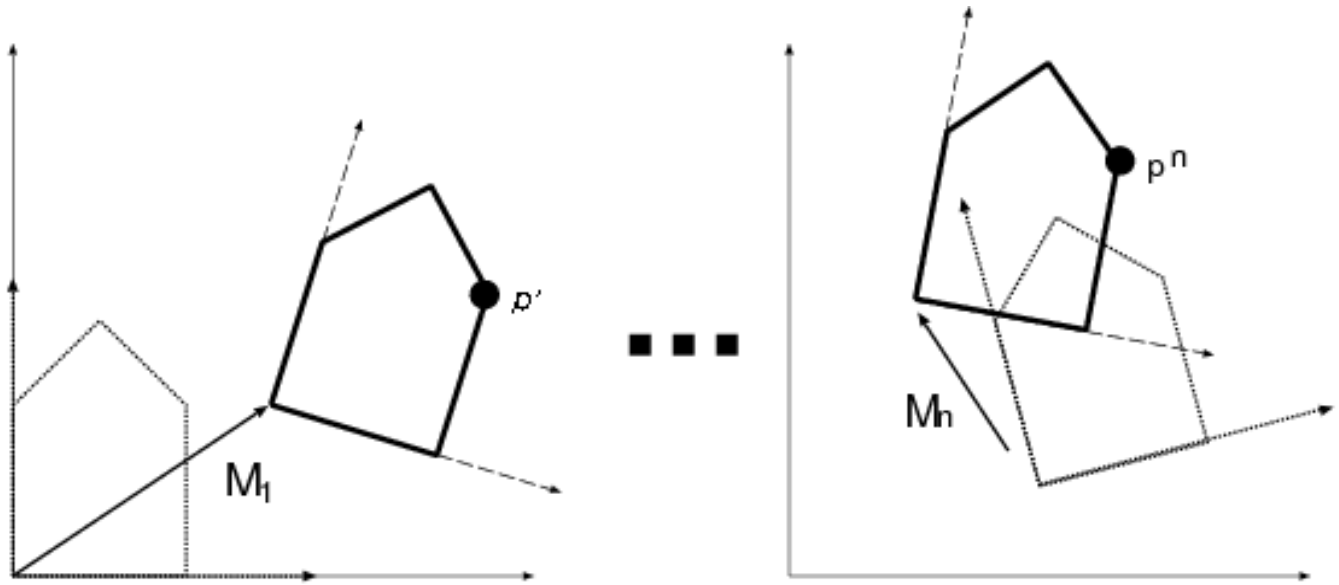
$p''' = T_1\, R\, T_2\, p$

- **Concatenate local transformation matrices from left to right**
- Can obtain the local – world transformation matrix
- p', p'', p''' are the world coordinates of p after each transformation

Transformations of local coordinate systems - example

- p^n is the world coordinate of point p after n transformations

$$p^n = M_1 M_2 \cdots M_n p$$

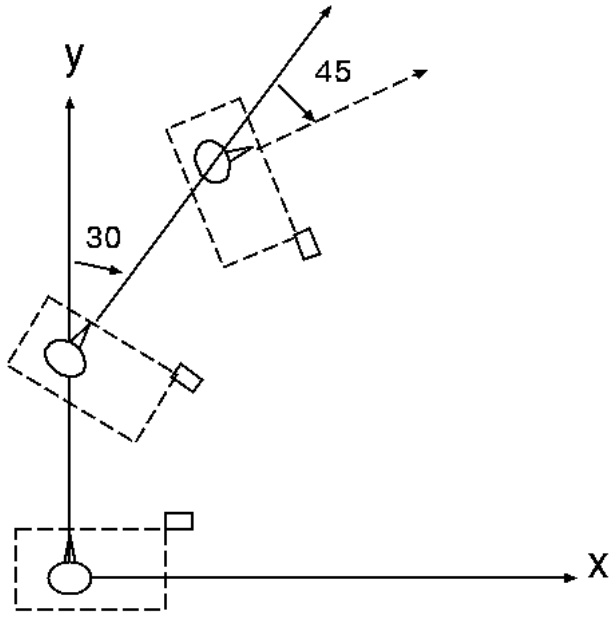


Quiz

- I sat in the car, and find the side mirror is 0.4m on my right and 0.3m in my front
- I started my car and drove 5m forward, turned 30 degrees to right, moved 5m forward again, and turned 45 degrees to the right, and stopped
- What is the position of the side mirror now, relative to where I was sitting in the beginning?

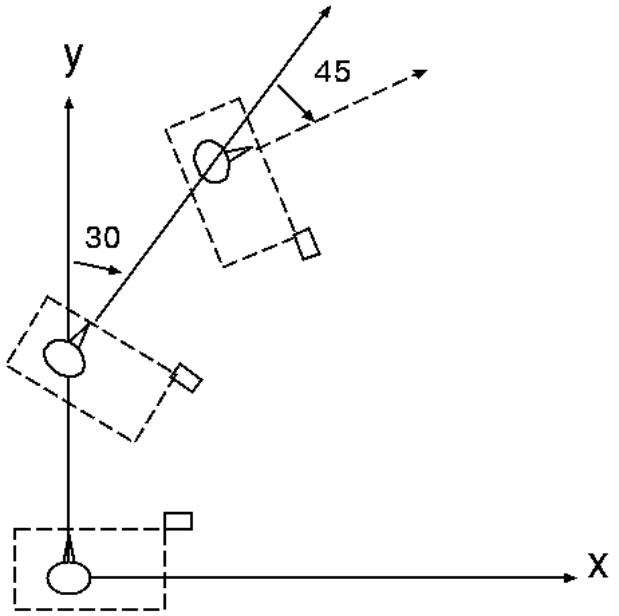
Solution

- The side mirror position is locally (0.4, 0.3)
- The matrix of first driving forward 5m is
- $T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix}$



Solution

- The matrix to turn to the right 30 and 45 degrees (rotating -30 and -45 degrees around the origin) are
- $R_1 = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- $R_2 = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$

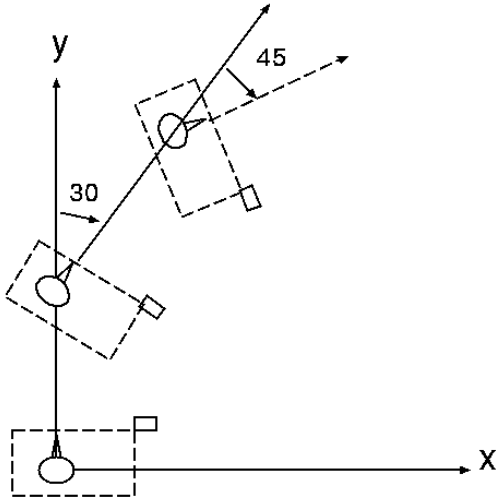


Solution

- The local-to-global transformation matrix at the last configuration of the car is

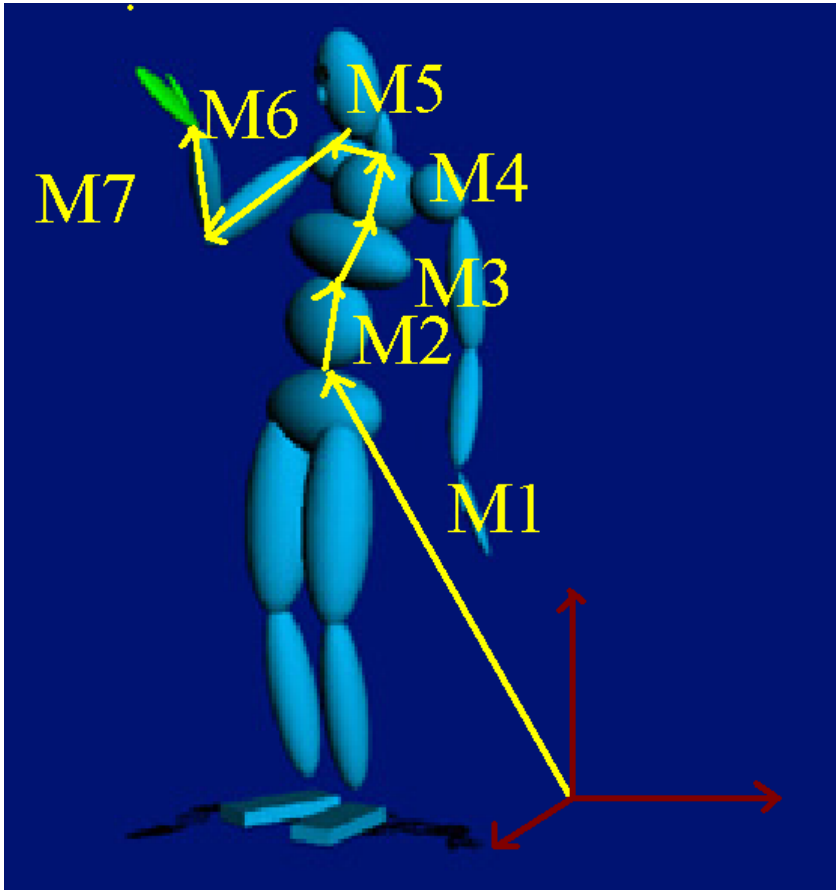
- $$TR_1TR_2P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.4 \\ 0.3 \\ 1 \end{bmatrix}$$

The final position of the side mirror can be computed by TR_1TR_2P which is around (2.89331, 9.0214)



This is convenient for character animation / robotics

- In robotics / animation, we often want to know what is the current 3D location of the end effectors (like the hand)
- Can concatenate matrices from the origin of the body towards the end effecter



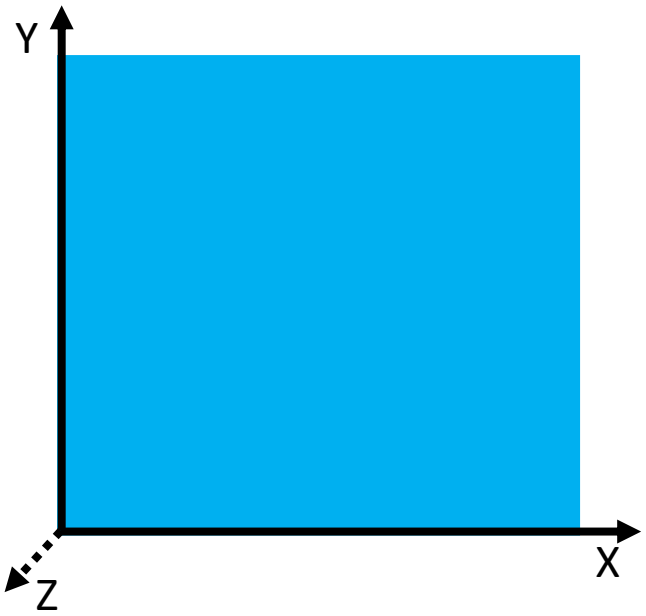
How to implement?

- Queue?
- Stack?
- Tree?

- Questions?

3D Transformations

- Use homogeneous coordinates, just as in 2D case
- Transformations are now 4x4 matrices
- We will use a right-handed (world) coordinate system - (z out of the page)



Translation

- $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Scale

- $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Reflection

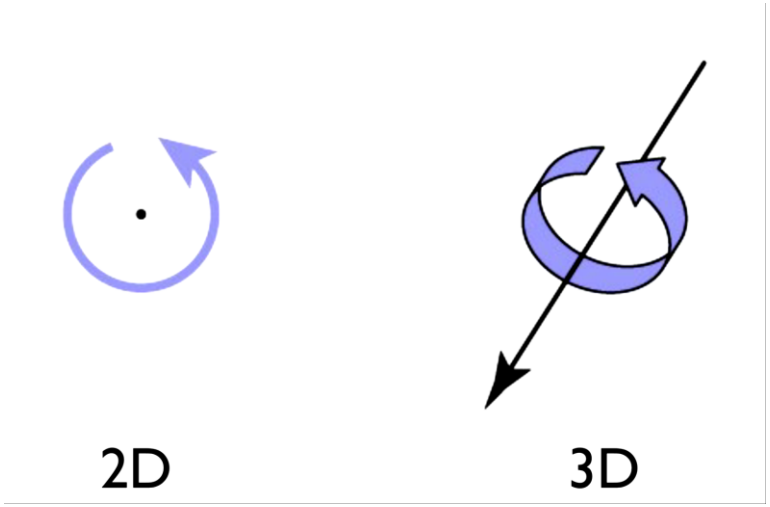
- $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Shear

- $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a_1 & a_2 & 0 \\ a_3 & 1 & a_4 & 0 \\ a_5 & a_6 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

General Rotation Matrices

- Rotation in 2D?
  - Around an arbitrary point
- Rotation in 3D?
  - Around an arbitrary axis
  - There are many more 3D rotations than 2D

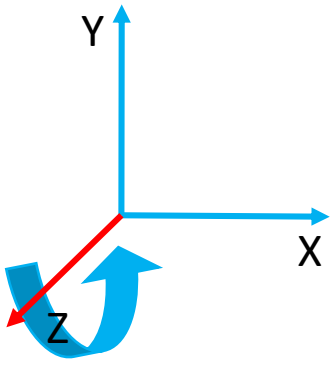


Specifying Rotations

- In 2D, just an angle θ
- In 3D, is more complex
  - Basic rotation about origin: Axis and angle
  - Convention: positive rotation is CCW (right handed)
  - Many ways
    - Directly through
      - Euler angles: 3 angles about 3 axes
    - Indirectly through frame transformations
    - Quaternions

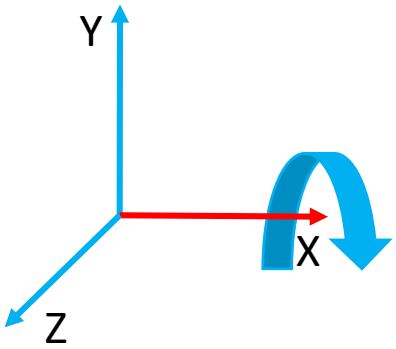
Rotation

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



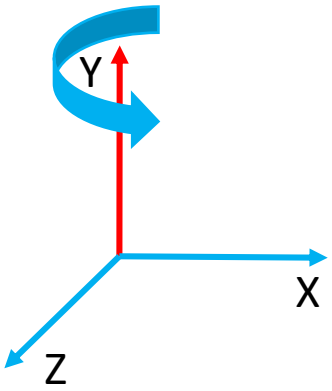
Rotation

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



Rotation

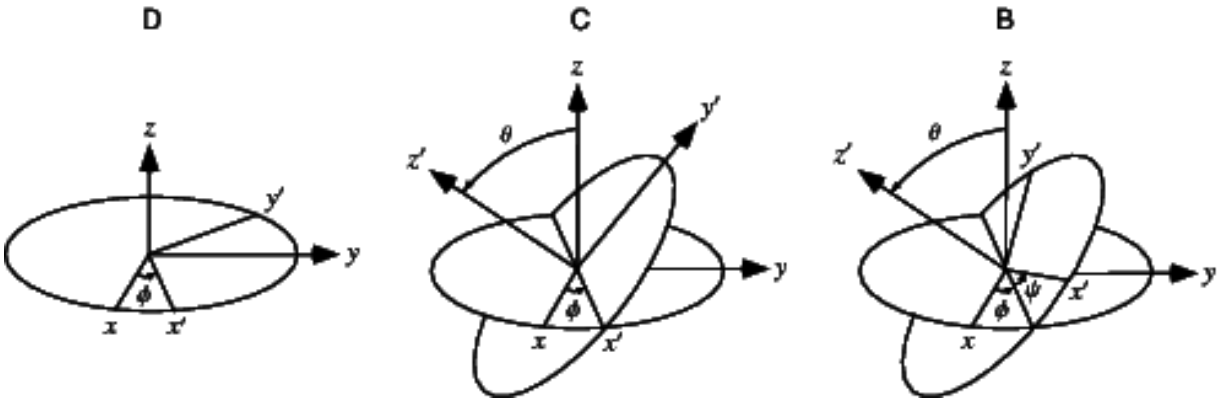
- $R_y(\theta) = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$



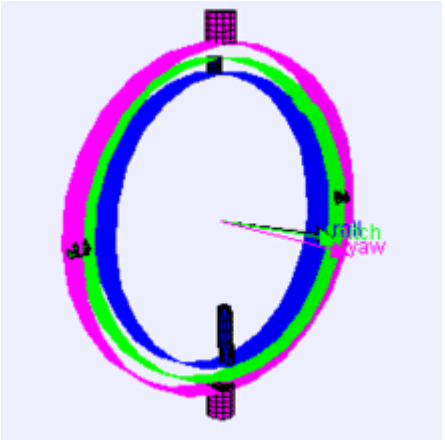
Euler Angles

- Any rotation may be described using three angles. If the rotations are written in terms of rotation matrices D, C, and B, then a general rotation A can be written as:

$$A = BCD$$

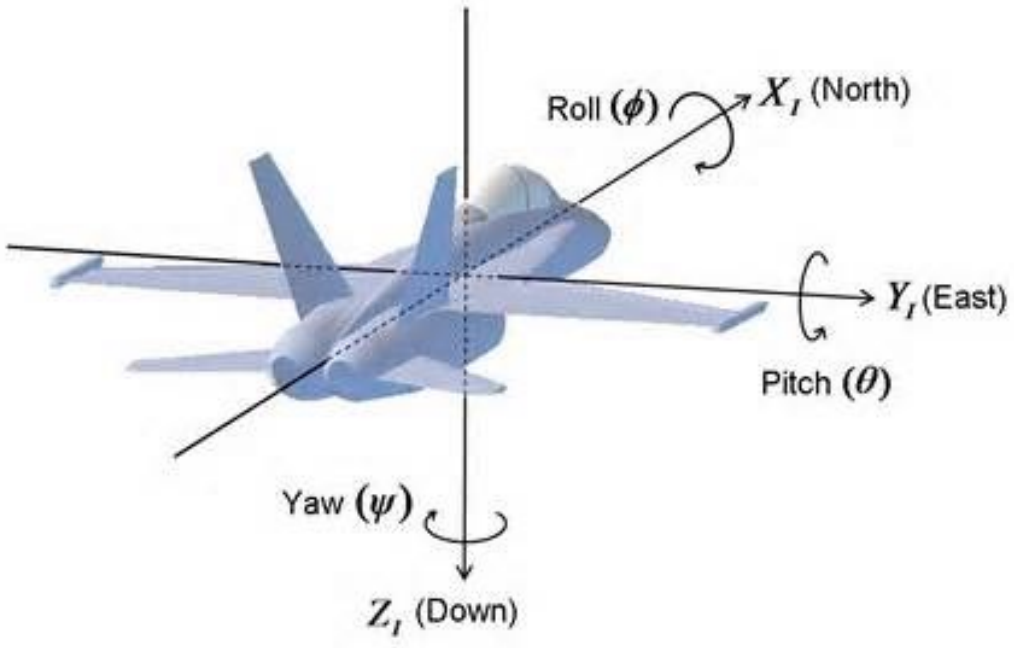


Gimbal



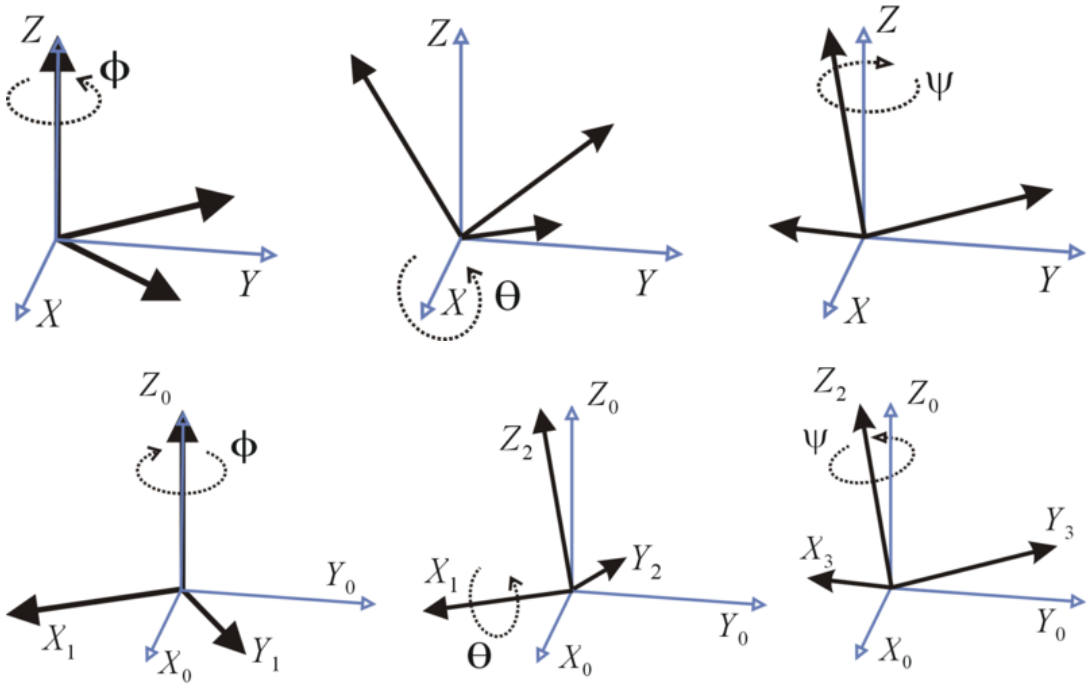
Euler Angles

- Roll(Φ) around X
- Pitch(Θ) around Y
- Yaw(Ψ) around Z



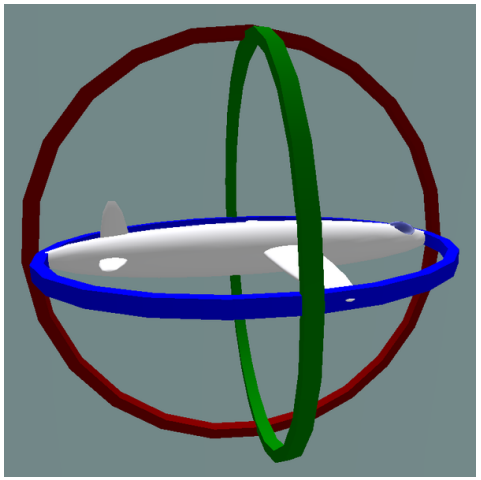
Extrinsic vs Intrinsic Rotations

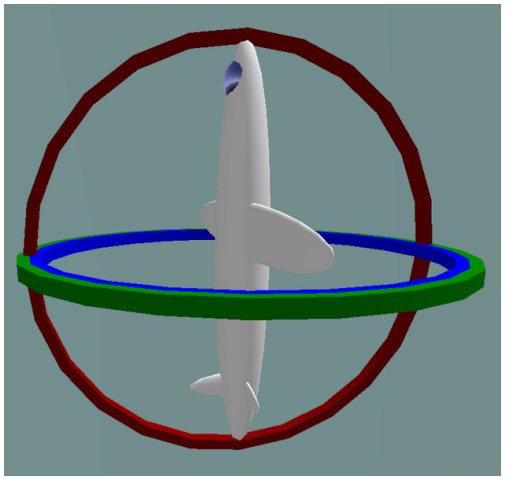
- Any extrinsic rotation is equivalent to an intrinsic rotation by the same angles but with inverted order of elemental rotations

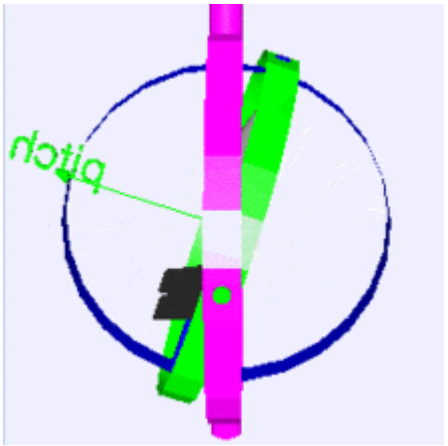


Gimbal Lock

- Euler basic motions are never expressed in a frame, but a **mixed axes of rotation**
- $R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix} \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- Set $\beta = \pi/2$?





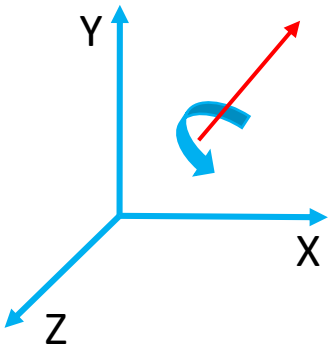


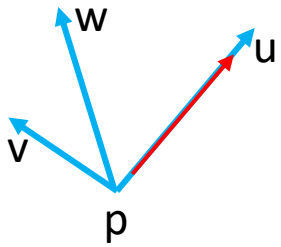
Matrices for Axis-angle Rotations

- Ruler angles are for coordinates axes
  - What if we want rotation about some random axis?
- Compute by composing elementary transforms
  - Transform rotation axis to align with x axis
  - Apply rotation
  - Inverse transform back into position

Building General Rotations

- Using elementary transforms
  - Translate axis to pass through origin
  - Rotate about y to get into x-y plane
  - Rotate about z to get into x axis
- Alternative: construct a frame and change coordinates
  - Choose p, u, v, w
  - Apply transform $T = FR_x(\theta)F^{-1}$
  - $F = \begin{bmatrix} u & v & w & p \\ 0 & 0 & 0 & 1 \end{bmatrix}$





Quaternions

- Compare to Euler angles and Rotation matrices
  - Can avoid the gimbal lock
  - Can smoothly interpolate over a sphere
- Representation
  - A quaternion is composed of one real element and three complex elements

$$q = q_0 + q_1 i + q_2 j + q_3 k$$

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ ij = k, ji = -k \\ jk = i, kj = -i \\ ki = j, ik = -j \end{cases}$$

Quaternions

- A rotation through an angle of θ around the axis defined by a unit vector $n = \left[n_{x,}\, n_y, n_z\right]^T$ can be represented by a quaternion:

$$q = \left[cos\frac{\theta}{2}, n_x\, sin\frac{\theta}{2}, n_y\, sin\frac{\theta}{2}, n_z\, sin\frac{\theta}{2}\right]^T$$

- Rotation： $p' = qpq^{-1}$
- Rotation matrix, Rotation Axis, Euler Angle and Quaternions are exchangeable.

Rodriguez formula

$$R = cos(\theta)\ \cdot I + \big(1 - cos(\theta)\big)\phi\phi^T + sin(\theta)\phi\times$$

- *I* identity matrix, ϕ axis-angle rotation
- x is skew-symmetric matrix of a vector:

$$(a\times) = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix}$$

Transformations with GLM

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
…
glm::vec4 vec(1.0f, 0.0f, 0.0f, 1.0f);
glm::mat4 trans = glm::mat4(1.0f);
trans = glm::translate(trans, glm::vec3(1.0f, 1.0f, 0.0f));
trans = glm::rotate(trans, glm::radians(90.0f),
glm::vec3(0.0, 0.0, 1.0));
vec = trans * vec;
```

- Questions?

References

- Ed Angel, CS/EECE 433 Computer Graphics, University of New Mexico
- Durand and Cuter, MIT EECS 6.837
- Steve Marschner, CS4620/5620 Computer Graphics, Cornell
- Tom Thorne, COMPUTER GRAPHICS, The University of Edinburgh
- Elif Tosun, Computer Graphics, The University of New York
- Lin Zhang, Computer Graphics, Tongji Unviersity