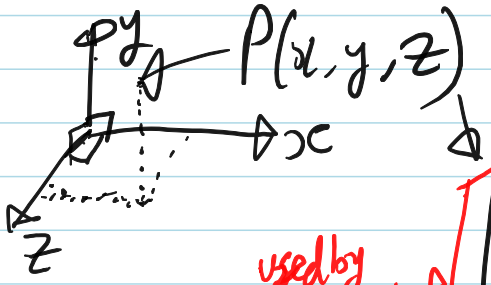
3D Transformations

- Models, relationships, pathfinding (with avatars), shading, collisions
- Transformations are essential to comp. graphics
- CRUCIAL: 3D Cartesian coordinates



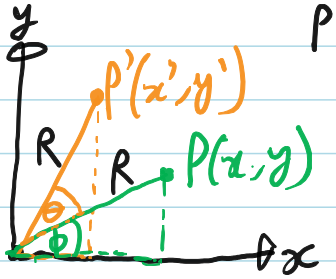
- VECTOR: direction in space

used by OpenGL → $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ OR $\begin{bmatrix} x & y & z \end{bmatrix}$ → Matlab

- 3D Translation: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = x \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Scaling: $\begin{bmatrix} x \\ y \\ z \end{bmatrix} \cdot b \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$

2D Rotation: How to find P' coordinates given P rotated by θ radians?



$x = R\cos\phi, \quad y = R\sin\phi \quad (1)$

$x' = R\cos(\phi+\theta)$
$= R\cos\phi\cos\theta - R\sin\phi\sin\theta \quad (2)$

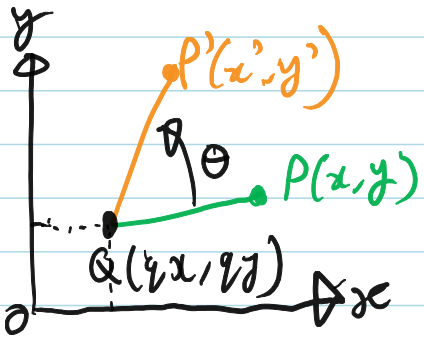
$y' = R\sin(\phi+\theta)$
$= R\sin\phi\cos\theta + R\cos\phi\sin\theta \quad (3)$

∴ subbing (1) into (2) and (3)

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

} new coordinates for P', when P is rotated by θ rad.

- 2D Rotation NOT about origin: 3 steps



1) Translate Q back to origin by $\begin{bmatrix} -qx \\ -qy \end{bmatrix}$, same with P'
2) Rotate P to P' THERE (we know how to do this already)
3) Translate all points forward by $\begin{bmatrix} qx \\ qy \end{bmatrix}$ again ☺

- The above rotations are 3D rotations with NO change in z direction

- Scaling using matricies: $\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 7 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ 4y \\ 7z \end{bmatrix}$

3D rotation with matricies: $\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ z \end{bmatrix}$

(!) CAN'T perform translations with simple 3x3 matrix
- unless we add 4th ⟨row / column⟩ like so:

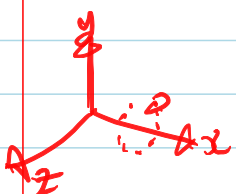
$$\begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix}$$

w always set to (1) for 3D stuff

- Flip that PP around: column vectors always read R ← (row) . . . . . P → R

- CRUCIAL: 3D Rotation matricies about

- X axis:



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \begin{matrix} \rightarrow x \\ \rightarrow y\cos\theta - z\sin\theta \\ \rightarrow y\sin\theta + z\cos\theta \\ \rightarrow 1 \end{matrix}$$

- Y axis:

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

- Z axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \begin{matrix} \rightarrow x\cos\theta - y\sin\theta \\ \rightarrow x\sin\theta + y\cos\theta \\ \rightarrow z \\ \rightarrow 1 \end{matrix}$$

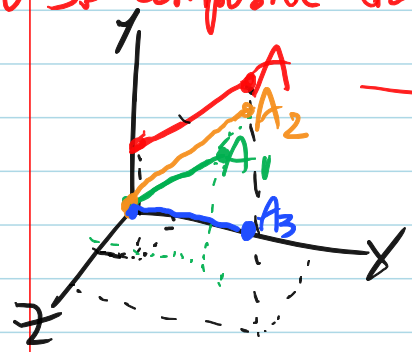
- Composing transformations: OpenGL handles ALL inter-matrix transformations
  - ☺ You only have to multiply one matrix to a vector at any time
- ALL graphics systems are just matrix transformations
- ☹ Can only SOMETIMES tell what a composite matrixes parts are
- As you know, with matricies (M_1, M_2), $M_1 \cdot M_2 \neq M_2 \cdot M_1$ in general
- Composite transformations: 2D example
  - Scaling about a non-origin point:
    1) Translate image to origin → M_1
    2) Scale HERE → M_2
    3) Translate back to original point → M_3

∴ NEW = $M_3 \cdot M_2 \cdot M_1 \cdot old$

- **Inverse:** M_3 and M_1 were INVERSES of each other
  - $M_3 \cdot M_1 = I$ ← identity matrix

- However, not all matricies are invertible
  - e.g singular transformations, which THROW AWAY all information about certain values
    ∴ can't undo it

- 3D composite transformations about random vectors



- How do we rotate something about vector A?

1) TRANSLATE A until passes thru origin (M_1)
   - new vector, A_1
2) ROTATE A_1 into XY plane (M_2)
   - new vector, A_2
3) ROTATE A_2 about Z axis until co-incident with X axis (M_3)
   - new vector, A_3
4) PERFORM actual rotation about A_3 (M_4)
5) do $(M_3)^{-1}$
6) do $(M_2)^{-1}$
7) do $(M_1)^{-1}$

— to get shape BACK to where we started

∴ new = $(M_1)^{-1} \cdot (M_2)^{-1} \cdot (M_3)^{-1} \cdot M_4 \cdot M_3 \cdot M_2 \cdot M_1 \cdot$ old