Week 8: 3D Geometry II
Part 3: More about rotations

COMP270: Mathematics for 3D Worlds and Simulations

Objectives

- **Compare** different ways of representing rotations
- **Consider** some quirks of rotations in 3D

Recap: 3D rotation matrices

Anticlockwise rotation in a right-handed coordinate system about:

- The x-axis:

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

- The y-axis:

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

- The z-axis:

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

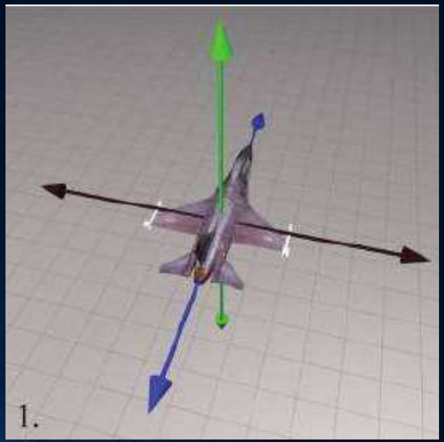
Pros and cons of matrices

Pros	**Cons**
Explicit/“brute force” representation: can be applied directly to vectors	Take up more memory than is really needed for the information stored
Commonly used by graphics APIs	Not intuitive for humans to use
Concatenation of multiple transforms in a single matrix	Can easily be ill-formed (sine & cosine are small): • Scale, skew, reflection or projection matrices aren’t orthogonal • Bad input data, e.g. from mocap • Floating point errors (from successive changes): *matrix creep* requires re-orthogonalisation
The opposite transform is given by the inverse, which is relatively straightforward to compute	

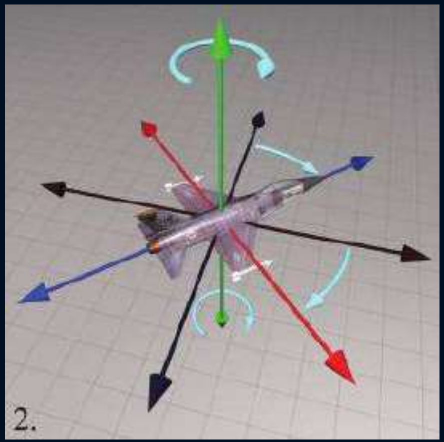
Alternative: Euler angles

- Define an angular displacement as a sequence of three rotations about three mutually perpendicular axes (usually x, y, z).
- Can be applied in any order – must be specified.
- Many variations on conventions/nomenclature, e.g. *yaw-pitch-roll*, or *heading-pitch-bank*
- Rotations occur about the body (local space) axes, which change after each rotation…
- Equivalent to a fixed-axes system provided that the rotations are performed in the opposite order.
- Original (symmetric) system: first and last rotations are about the same axis
- Common order:
  - First about the vertical axis (y)
  - Second about the body lateral axis (x)
  - Third about the body longitudinal axis (z)

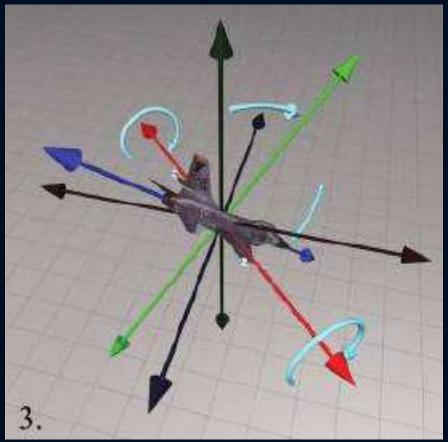
Euler angles example



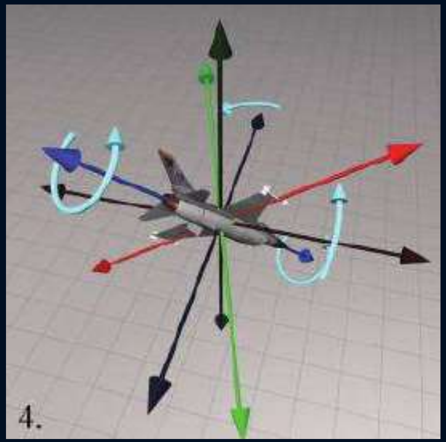
Initial orientation



Heading rotation
(vertical / y-axis)



Pitch rotation
(lateral / x-axis)



Bank rotation
(longitudinal / z-axis)

"3D Math Primer for Graphics and Game Development" (2nd Ed), figures 8.4-8.7

- Interactive demonstration: https://demonstrations.wolfram.com/EulerAngles/

Euler angles and aliasing

- **Problem**: different angles can give the same result
  - Adding a multiple of 360° changes the numbers but not the rotation
  - The angles are not completely independent of each other
    - e.g. pitching down 135° = heading 180°, pitching down 45°, banking 180°

Gimbal lock

Canonical Euler angles

- (Partial) **solution:** restrict range of angles to a canonical set, e.g. (-180°, +180°] for heading/bank and (-90°, +90°] for pitch
  - But still: 45° right then 90° down = down 90° then bank/twist 45°
  - Generally: an angle of ±90° for the second rotation causes the first and third rotations to be about the same axis
- **Additional restriction:** assign all rotation about the vertical axis to the first (heading) rotation, leaving the last (bank) at zero.

Interpolating Euler angles

- Standard linear interpolation (LERP):

$$\Delta\theta = \theta_1 - \theta_0$$
$$\theta_t = \theta_0 + t\Delta\theta$$

- Tends to choose the "long way round", even within canonical ranges, e.g. between -170° and +170°
  - Solution: wrap to find the shortest arc by adding/subtracting the appropriate multiple of 360°
- Gimbal lock causes sudden changes of orientation (angular velocity is not constant)
  - Cannot be eliminated, but can work around by choosing appropriate rotation orders for each scenario
  - More info: https://www.youtube.com/watch?v=zc8b2Jo7mno

Pros and cons of Euler angles

Pros	Cons
More intuitive to visualise (?)	The representation for a given orientation is not unique • Angles are cyclical; not mutually independent
Smallest possible representation – no wasted space	Interpolation is problematic • Gimbal lock
Can be compressed if necessary: angle values are larger than the sine/cosine values stored in matrices, so require less precision	Need to be converted to matrices to use
Any set of three numbers is valid (will produce a valid rotation)	

Another alternative: axis + angle

- **Euler's rotation theorem**: any 3D angular displacement can be accomplished by a single rotation through an angle θ about a carefully chosen axis $\hat{\mathbf{n}}$
- Axis-angle form uses these values directly
  - Encodes a 3D rotation in 4 values (x, y, z, θ)
  - To apply to a vector $\mathbf{v}$, either:
    - Convert to a matrix, or
    - Use Rodrigues' formula:

$$\mathbf{v}' = \mathbf{v}\cos\theta + (\hat{\mathbf{n}} \times \mathbf{v})\sin\theta + (1 - \cos\theta)(\hat{\mathbf{n}} \cdot \mathbf{v})\hat{\mathbf{n}}$$

Pros and cons of axis-angle

Pros	**Cons**
Most intuitive to visualise	Discontinuities at 0° and 180° - axis jumps suddenly for small change in input
Relatively small representation (4 values)	Cannot apply directly to a vector
Can be compressed if necessary: angle values are larger than the sine/cosine values stored in matrices, so require less precision	Cannot combine two rotations directly
Any axis/angle combination is valid (will produce a valid rotation)	

Another alternative: quaternions

- Encode the axis and angle of rotation as a scalar component w and a 3D vector component $\mathbf{v}$:

$$[w \quad \mathbf{v}] = \left[\cos\frac{\theta}{2} \quad \sin\frac{\theta}{2}\hat{\mathbf{n}}\right]$$
$$\mathbf{v} = [x \quad y \quad z]$$

  - Not to be confused with axis-angle, or homogeneous coordinates!
  - Row and column formats are interchangeable – they don't interact with matrices.

Pros and cons of quaternions

Pros	Cons
Relatively small representation (4 values)	One more value than Euler angles • Component values do not interpolate smoothly, so harder to compress
Only representation that provides smooth interpolation	Can become invalid (from bad input or rounding errors)
Fast concatenation and inversion	Least intuitive representation
Fast conversion to and from matrix form	