Outline
• Define 3D coordinates
• Homogeneous coordinate
• 3D transformations
  • Translation
  • Scaling
  • Rotation
  • Shear
  • Euclidean
  • Affine

3D Transformation

Outline
• Define 3D coordinates
• Homogeneous coordinate
• 3D transformations
  • Translation
  • Scaling
  • Rotation
  • Shear
  • Euclidean
  • Affine

3D Coordinates
• Left-handed
  • Default in RenderMan
• Right-handed
  • Default in OpenGL
  Negate any single axis

Homogeneous Coordinate
• Represent 3D point in 4D space
  Coordinates: \((x, y, z, w)\)
  Normalized coordinates: \((x/w, y/w, z/w, 1)\)
  Plane at infinity is identified with the set of points with \(w=0\)
• Represent 3D transformation using 4×4 matrices

Translation
• Move an object to a new location

Scaling
• Adjust the size of the object
Rotation about Z Axis

- Z coordinate unchanged
- X & Y coordinates follow 2D rotation

\[
R_z(\theta) = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

Rotation about X Axis

- X coordinate unchanged
- Y & Z coordinates follow 2D rotation

\[
R_x(\theta) = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \theta & -\sin \theta \\
0 & \sin \theta & \cos \theta
\end{bmatrix}
\]

Rotation about Y Axis

- Y coordinate unchanged
- X & Z coordinates follow 2D rotation
  - Sign of \( \sin(\theta) \) is negated since Z axis points downward

\[
R_y(\theta) = \begin{bmatrix}
\cos \theta & 0 & \sin \theta \\
0 & 1 & 0 \\
-\sin \theta & 0 & \cos \theta
\end{bmatrix}
\]

3D Rotation is Not Commutative

- Rotating about X axis by \( \alpha \) then about Y axis by \( \beta \) is not the same as rotating about Y axis by \( \beta \) then about X axis by \( \alpha \)

\[
R_x(\alpha)R_y(\beta) = \begin{bmatrix}
\cos \phi & \sin \phi & 0 \\
\sin \phi & \cos \phi & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

Rotate about Arbitrary Direction

- Rotating about an arbitrary direction \((x, y, z)\) by \( \theta \) can be achieved by combining multiple rotations
- Rotate about X & Y axis to align \((x, y, z)\) with Z axis
- Rotate \( \theta \) about Z axis
- Rotate about X & Y axis to restore \((x, y, z)\) direction

Let \( c = \cos(\theta) \), \( s = \sin(\theta) \), \( t = 1 - c \)

\[
R(\theta, x, y, z) = \begin{bmatrix}
x't + c & xy't - zs & xz't + ys \\
y't + cz & y'y't + c^2 & yz't - xs \\
xz't - ys & yz't + xs & z'^2 + c
\end{bmatrix}
\]

Representing Arbitrary Rotation

- Using 3x3 matrix:
  - Requires 9 numbers
  - \( R(\theta, x, y, z) \) only has 3 degrees of freedom (DOF)
  - 2 DOF for unit vector \((x, y, z)\)
  - 1 DOF for angle \( \theta \)
  - Problem:
    - Too much redundancy
    - Not all matrices are valid rotations
  - Use Euler angles:
    - Decompose \( R(\theta, x, y, z) \) into 3 elemental rotations (rotations around a single axis)
    - Store the angles \((\alpha, \beta, \gamma)\) used for these rotations
    - Require 3 numbers
  - Problem:
    - Hard to interpolate between 2 rotations
Quaternion Representation

- Use a unit quaternion \((\cos(\frac{\theta}{2}),\sin(\frac{\theta}{2})V)\) to denote rotating about a unit vector \(V\) by \(\theta^\circ\)
- Require 4 numbers
- Advantages:
  - Concatenating 2 rotations can be done by quaternion multiplication
  - Faster & numerically more stable.
  - Extracting the angle & axis of rotation is simple
  - Interpolation between 2 rotations is more straightforward
  - Use spherical linear interpolation (slerp)

Quaternion to Matrix Representation

- The quaternion representation can be easily converted into matrix representation
- Converting back requires solving quadratic equations

Composition of 3D Transformations

- How to rotate about edge \(PQ\) by \(\theta^\circ\)?
  - Translate \(P\) to origin
  - Rotate about direction \(PQ\) by \(\theta^\circ\)
  - Translate the origin back to point \(P\)
- Net transformation:
  \[ T(P_x,P_y,P_z) \cdot R(\theta,Q-P) \cdot T(-P_x,-P_y,-P_z) \]

Euclidean (Rigid Body) Transformation

- The product of any translation & rotation matrices
  \[ \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \]
  - Submatrix \(R\) gives the aggregated rotation
    - 3 DOF
  - Vector \(T\) gives the aggregated translation
    - 3 DOF
  - Preserve lengths, angles, & parallelism
  \[ P' = MP = RP + T \]

Affine Transformation

- The product of any translation, scaling, rotation, & shear matrices
  \[ \begin{pmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \]
  - Submatrix \(R\) gives the aggregated rotation, scaling & shearing
    - 9 DOF
  - Vector \(T\) gives the aggregated translation
    - 3 DOF
  - Preserve parallelism
  \[ P' = MP = AP + T \]