3D Transformations

CS 4620 Lecture 3

Translation

\[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & 0 & t_x \\
    0 & 1 & 0 & t_y \\
    0 & 0 & 1 & t_z \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

Scaling

\[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    s_x & 0 & 0 \\
    0 & s_y & 0 \\
    0 & 0 & s_z \\
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z
\end{bmatrix}
\]

Rotation about z axis

\[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    \cos \theta & -\sin \theta & 0 & 0 \\
    \sin \theta & \cos \theta & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z
\end{bmatrix}
\]
Transformations in OpenGL

- Stack-based manipulation of model-view transformation, \( M \)
- \( \text{glMatrixMode} \) \( (\text{GL}_\text{MODELVIEW}) \) specifies model-view matrix
- \( \text{glLoadIdentity}() \) \( M \leftarrow 4x4 \) identity
- \( \text{glTranslatef} \) \((\text{float} \ ux, \ \text{float} \ uy, \ \text{float} \ uz) \) \( M \leftarrow MT \)
- \( \text{glRotatef} \) \((\text{float} \ \theta, \ \text{float} \ ux, \ \text{float} \ uy, \ \text{float} \ uz) \) \( M \leftarrow MR \)
- \( \text{glScalef} \) \((\text{float} \ sx, \ \text{float} \ sy, \ \text{float} \ sz) \) \( M \leftarrow MS \)
- \( \text{glLoadMatrixf} \) \((\text{float}[] \ A) \) \( M \leftarrow A \) (Note: column major)
- \( \text{glMultMatrixf} \) \((\text{float}[] \ A) \) \( M \leftarrow MA \) (Note: column major)
- Manipulate matrix stack using:
  - \( \text{glPushMatrix}() \)
  - \( \text{glPopMatrix}() \)

\[
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\]

Rotation about \( x \) axis

\[
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\]

Rotation about \( y \) axis
Transformations in OpenGL

– Tutors demo

General Rotation Matrices

• A rotation in 2D is around a point
• A rotation in 3D is around an axis
  – so 3D rotation is w.r.t a line, not just a point
  – there are many more 3D rotations than 2D
    • a 3D space around a given point, not just 1D

Properties of Rotation Matrices

• Columns of R are mutually orthonormal: \( RR^T = R^TR = I \)
• Right-handed coordinate systems: \( \text{det}(R) = 1 \)
  – Recall definition of \( \text{det}(R) = r_1^T(r_2 \times r_3) \)
• Such 3x3 rotation matrices belong to group, \( SO(3) \)
  – Special orthogonal
  – Special --> \( \text{det}(R) = 1 \)

Specifying rotations

• In 2D, a rotation just has an angle
  – if it’s about a particular center, it’s a point and angle
• In 3D, specifying a rotation is more complex
  – basic rotation about origin: unit vector (axis) and angle
    • convention: positive rotation is CCW when vector is pointing at you
  – about different center: point (center), unit vector, and angle
    • this is redundant: think of a second point on the same axis...
• Alternative: Euler angles
  – stack up three coord axis rotations
    • ZYX case: \( R_z(\alpha) R_y(\beta) R_x(\gamma) \)
    – degeneracies exist for some angles
      – E.g., gimbal lock
      – Black board

Unlocked
Gimbal lock
Coming up with the matrix

- Showed matrices for coordinate axis rotations
  - but what if we want rotation about some random axis?
- Can compute by composing elementary transforms
  - transform rotation axis to align with x axis
  - apply rotation
  - inverse transform back into position
- Just as in 2D this can be interpreted as a similarity transform

Building general rotations

- Using elementary transforms you need three
  - translate axis to pass through origin
  - rotate about y to get into x-y plane
  - rotate about z to align with x axis
- Alternative: construct frame and change coordinates
  - choose p, u, v, w to be orthonormal frame with p and u matching the rotation axis
  - apply similarity transform $T = F R_q(\theta) F^{-1}$

Orthonormal frames in 3D

- Useful tools for constructing transformations
- Recall rigid motions
  - affine transforms with pure rotation
  - columns (and rows) form right-handed ONB
    - that is, an orthonormal basis

$$F = \begin{bmatrix} u & v & w & p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Building 3D frames

- Given a vector a and a secondary vector b
  - The u axis should be parallel to a; the u-v plane should contain b
    - $u = a / ||a||$
    - $w = u \times b; w = w / ||w||$
    - $v = w \times u$
- Given just a vector a
  - The u axis should be parallel to a; don’t care about orientation about that axis
    - Same process but choose arbitrary b first
    - Good choice is not near a: e.g., set smallest entry to 1
Building general rotations

- Alternative: construct frame and change coordinates
  - choose \( p, u, v, w \) to be orthonormal frame with \( p \) and \( u \) matching the rotation axis
  - apply similarity transform \( T = F R_x(\theta) F^{-1} \)
  - interpretation: move to \( x \) axis, rotate, move back
  - interpretation: rewrite \( u \)-axis rotation in new coordinates
  - (each is equally valid)

- Or just derive the formula once, and reuse it (more later)

Derivation of General Rotation Matrix

- General 3x3 3D rotation matrix
- General 4x4 rotation about an arbitrary point

Building transforms from points

- Recall: 2D affine transformation has 6 degrees of freedom (DOFs)
  - this is the number of “knobs” we have to set to define one
- Therefore 6 constraints suffice to define the transformation
  - handy kind of constraint: point \( p \) maps to point \( q \) (2 constraints at once)
  - three point constraints add up to constrain all 6 DOFs
  (i.e. can map any triangle to any other triangle)
- 3D affine transformation has 12 degrees of freedom
  - count them by looking at the matrix entries we’re allowed to change
- Therefore 12 constraints suffice to define the transformation
  - in 3D, this is 4 point constraints
  (i.e. can map any tetrahedron to any other tetrahedron)

Transforming normal vectors

- Transforming surface normals
  - differences of points (and therefore tangents) transform OK
  - normals do not → use inverse transpose matrix

have: \( t \cdot n = t^T n = 0 \)
want: \( M t \cdot X n = t^T M^T X n = 0 \)
so set \( X = (M^T)^{-1} \)
then: \( M t \cdot X n = t^T M^T (M^T)^{-1} n = t^T n = 0 \)