3-D Geometric Transformations

Move objects in a 3-D scene
Extension of 2-D Affine Transformations
Three important ones:
- Translation
- Scaling
- Rotations

Representing 3-D Points

Homogeneous coordinates
P (x,y,z) → P’ (x’,y’,z’)

Translations

Given 3-D translation vector T=(tx, ty, tz)
Component equations
x’ = x + tx
y’ = y + ty
z’ = z + tz
Represent translation as matrix equation

P’ = T * P
T is a 4 X 4 Homogeneous Matrix

Homogeneous Translation Matrix

| 1 0 0 tx |
T = | 0 1 0 ty |
| 0 0 1 tz |
| 0 0 0 1 |

Notice obvious extension from 2-D to 3-D
Scaling with respect to origin
- Given three scaling factors sx, sy, sz
  \[ P' = S \times P \]
- S is the following 4 x 4 scaling matrix:
  \[
  \begin{vmatrix}
  sx & 0 & 0 & 0 \\
  0 & sy & 0 & 0 \\
  0 & 0 & sz & 0 \\
  0 & 0 & 0 & 1
  \end{vmatrix}
  \]
- Again obvious extension from 2D

Rotations
- Need to specify angle of rotation
- And axis about which the rotation is to be performed
- Infinite number of possible rotation axes
  - Rotation about any axis: linear combinations of rotations about x-axis, y-axis, z-axis

Rotations about z-axis
- Consider rotation of point P=(x,y,z) by angle theta about the z-axis giving rotated point P'=(x',y',z')
  - Same x,y equations as in the 2-D case
  - z will not change

Z-Axis Rotation Component Equations
- \[ x' = x\cos(\theta) - y\sin(\theta) \]
- \[ y' = x\sin(\theta) + y\cos(\theta) \]
- \[ z' = z \]
- Represented as homogeneous matrix equation:
  \[ P' = R_z \times P \]

Z-Axis Rotation Matrix
\[
R_z = \begin{vmatrix}
\cos(\theta) & -\sin(\theta) & 0 & 0 \\
\sin(\theta) & \cos(\theta) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{vmatrix}
\]

Rx Matrix for rotations about x-axis
- Symmetry argument
- Make replacements:
  - x --> y
  - y --> z
  - z --> x

Rx Matrix
\[
\begin{vmatrix}
y & z \\
x & y \\
0 & 0 \\
0 & 0 & 1
\end{vmatrix}
\]

about z

about x
Resulting equations:

\[ y' = y\cos(\theta) - z\sin(\theta) \]
\[ z' = y\sin(\theta) + z\cos(\theta) \]
\[ x' = x \]

Represented as matrix equation:

\[ P' = Rx \cdot P \]

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

Ry Rotation Matrix

Symmetry:

\[ \begin{array}{ccc}
y & x & \text{Replacements:} \\
\hline
x & z & x \rightarrow z \\
y & x & y \rightarrow x \end{array} \]

\[ \begin{array}{ccc}
z & y & z \rightarrow y \\
\end{array} \]

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

Rotation Sense

Positive sense

Defined as counter clockwise as we look down the rotation axis toward the origin

Composite 3-D Geometric Transformations

Series of consecutive transformations

- Represented by homogeneous transformation matrices \( T_1, T_2, ... , T_n \)

Equivalent to a single transformation

- Represented by composite transformation matrix \( T \)

- \( T \) is given by the matrix product:

\[ T = T_n \cdots T_2 \cdot T_1 \]

Just like in 2-D, except matrices are 4 \( \times \) 4
Example

- Rotate the line segment:
  - $(0,2,0) \rightarrow (0,4,3)$
- by 90 degrees about an axis parallel to the z-axis and passing through its left endpoint.
- 1. Translate to origin $(0,-2,0)$
- 2. Rotate by 90 degrees about z-axis
- 3. Translate back $(0,2,0)$

Library of 3-D Transformation Functions

- 3-D Transformation Package
- Straightforward Extension of 2-D
- Enables setting up and transforming points & polygons
- 4 X 4 Matrices have 12 non-trivial matrix elements
- Package Might contain the following functions:

3-D Transformation Functions

- void settranslate3d(a[12], tx, ty, tz);
- void setscale3d(a[12], sx, sy, sz);
- void setrotatex3d(a[12], theta);
- void setrotatey3d(a[12], theta);
- void setrotatez3d(a[12], theta);
- void combine3d(c[12], a[12], b[12]);  // C = A * B
- void xformcoord3d(c[12], vi, * vo );    // vo = C * vi
- void xformpoly3d(inpoly[], outpoly[], float c[12]);

a, b, and c are arrays
- Contain 12 non-trivial matrix elements of a 4 X4 homogeneous transformation matrix
- vi and vo are 3-D point structures; inpoly and outpoly are polygons

Rotation about an Arbitrary Axis

- Rotate point P by angle $\theta$ about a line
- Given: endpoints $P_1=(x_1,y_1,z_1)$ & $P_2=(x_2,y_2,z_2)$
- Convert problem into rotation about x-axis
- 1. Translate so that $P_1$ is at origin: $T_1 = T(-x_1,-y_1,-z_1)$
- 2. Compute spherical coordinates of the other endpoint:
  - $\rho = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2}$
  - $\phi = \arccos((z_2-z_1)/\rho)$
  - $\theta = \arctan((y_2-y_1)/(x_2-x_1))$
- 3. Rotate about z-axis by $-\theta$ so line lies in x-z plane: $T_2 = R_z(-\theta)$
- 4. Rotate about y-axis by $(90-\phi)$ to make line coincide with x-axis $T_3 = R_y(90-\phi)$
- 5. Rotate about x-axis by given angle $\phi$ $T_4 = R_x(\phi)$
- 6. Rotate back to undo step 4: $T_5 = R_y(\phi)$
- 7. Rotate back to undo step 3: $T_6 = R_z(\phi)$
- 8. Translate back to undo step 1: $T_7 = T(x_1,y_1,z_1)$
- Composite transformation then will be: $T = T_7 * T_6 * T_5 * T_4 * T_3 * T_2 * T_1$
3-D Coordinate System
Transformations

There’s a symmetrical relationship between 3-D geometric transformations
– (moving the object)
and 3-D coordinate system transformations
– (moving the coordinate system)

For translations, relationship is:
\[ T_{\text{coord}}(x,y,z) = T_{\text{geom}}(-x,-y,-z) \]

For each principal-axis, rotation relationship is:
\[ R_{\text{coord}}(\theta) = R_{\text{geom}}(-\theta) \]

Useful in deriving 3-D viewing transformation

3D Viewing and Projection

See CS-460/560 notes on 3-D Viewing and Projection Transformations
http://www.cs.binghamton.edu/~reckert/460/3dview.htm

3D Viewing/Projection Transformations

3-D points in model must be transformed to viewing coordinate system
– the Viewing Transformation
Then projected onto a projection plane
– Projection Transformation

3-D Viewing Transformation

Converts world coordinates \((x_w,y_w,z_w)\) of a point to viewing coordinates \((x_v,y_v,z_v)\) of the point
– As seen by a “camera” that is going to "photograph" the scene
\[ (x_w,y_w,z_w) \rightarrow (x_v,y_v,z_v) \]
Viewing transformation

Projection Transformation

Converts viewing coordinates \((x_v,y_v,z_v)\) of a point to 2-D coordinates \((x_p,y_p)\) of point's projection onto a projection plane

Think of projection plane as containing screen upon which image is to be displayed
\[ (x_v,y_v,z_v) \rightarrow (x_p,y_p) \]
Projection transformation
Viewing Setups

- Specify position/orientation of coordinate systems & projection plane
- Many possible viewing setups
- We’ll use a simple, 4-parameter viewing setup
  - Somewhat restricted
  - But adequate for most common situations

4-Parameter Viewing Setup

Parameters

- Position of viewpoint (camera location)
  - Position of origin of Viewing Coordinate System (VCS)
  - Specify in spherical coordinates
    - distance \( \theta \) from world coordinate system (WCS) origin
    - azimuthal angle \( \phi \)
    - polar angle \( \theta \)
- Distance \( d \) of projection plane (PP) from viewpoint

Viewing Setup Properties

VCS’s zv-axis points toward WCS origin
- So objects we want to be visible must be placed close to WCS origin
PP is perpendicular to the zv-axis at a distance \( d \) from VCS origin
  - So \( ? \) must be greater than \( d \)
Center of projection coincides with VCS origin

VCS’s yv-axis is parallel to projection of WCS’s zw-axis
- So WCS zw-axis defines “screen up” direction
VCS’s xv-axis is chosen so that xv-yv-zv axes form a left-handed coordinate system
  - objects far from the VCS’s origin have large zv
2-D Projection Plane coordinate system’s origin is at intersection of \( ? \) and PP
  - Its xp-yp-axes are projections of xv-yv axes onto PP
    - i.e., xv-yv translated a distance \( d \) along zv axis

3-D Viewing Transformation

- Must convert xw-yw-zw to xv-yv-zv
- A coordinate system transformation
- Perform the following steps:
  1. Translate origin by distance \( ? \) in direction \((?, ?)\)
  2. Rotate by \(-90-\) degrees about z-axis to bring new y-axis into plane of zw and \( ? \)
  3. Rotate by \(180-\) about x-axis to point transformed z-axis toward origin of world coordinate system
  4. Invert x-axis
1. Translate by ?

- Homogeneous transformation matrix for translation by (x,y,z):

\[
\begin{bmatrix}
1 & 0 & 0 & x \\
0 & 1 & 0 & y \\
0 & 0 & 1 & z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

- Use relationship between coordinate system transformations & geometric transformations:

\[
\text{Tcoord}(x,y,z) = \text{Tgeom}(-x,-y,-z)
\]

2. Rotate by -(90-?) about z

3. Rotate by (180-?) about x

4. Invert x-axis

So first transformation matrix, T1:

\[
\begin{bmatrix}
1 & 0 & 0 & -x \\
0 & 1 & 0 & -y \\
0 & 0 & 1 & -z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Express x, y, z in terms of ?, ?, ?

\[
x = ? \sin(?) \cos(?)
\]

\[
y = ? \sin(?) \sin(?)
\]

\[
z = ? \cos(?)
\]
2. Rotate by -(90°) about z

- Use relationship between coordinate system rotations & geometric rotations: $T_{\text{coord}}(\alpha) = T_{\text{geom}}(-\alpha)$
- So transformation is $T_2 = R_z(90°)$:

$$
\begin{bmatrix}
\cos(90°) & -\sin(90°) & 0 & 0 \\
\sin(90°) & \cos(90°) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

3. Rotate by (180°) about x

- Again use relationship between geometric & coordinate system rotations:
- So $T_3 = Rx(-180°)$:

$$
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos(-180°) & -\sin(-180°) & 0 \\
0 & \sin(-180°) & \cos(-180°) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

4. Invert x-axis

- Result of step 3: x-axis points opposite from direction it should
  - Because WCS is right-handed, while VCS is left-handed
  - So need to reflect across y*-z* plane
    - Will convert x to -x

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

Composite Viewing Transformation Matrix

- $T_v = T_4 T_3 T_2 T_1$
- Result (after simplification):

$$
\begin{bmatrix}
-\sin(\alpha) & \cos(\alpha) & 0 & 0 \\
-\cos(\alpha)\cos(\beta) & -\cos(\alpha)\sin(\beta) & \sin(\alpha) & 0 \\
-\sin(\alpha)\cos(\beta) & -\sin(\alpha)\sin(\beta) & -\cos(\alpha) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

Projection Transformation

- Look down xv axis at viewing setup:
  - Triangles OAP & OBP are similar
  - So set up proportion:
    $$
    \frac{y_p}{d} = \frac{y_v}{z_v}
    $$
  - Solve for $y_p$:
    $$
    y_p = \frac{y_v}{z_v}d
    $$
  - Look down yv axis for $x_p$:
    Result: $x_p = (x_v'd)/z_v$

Plotting Points on Screen

- Get screen coordinates $(x_s, y_s)$ from Projection Plane coordinates $(x_p, y_p)$
- Final Transformation:
  - 2D Window-to Viewport Transformation:
    $$(x_s, y_s) \leftarrow (x_p, y_p)$$
- See earlier notes
  - Replace $x_v, y_v$ with $x_s, y_s$
  - Replace $x_w, y_w$ with $x_p, y_p$