CS770/870 Spring 2017
3D Transformations

Related material

Any text on Computer Graphics

the Web

- 3D Transformations
  - Homogeneous coordinates
  - Matrices, vectors, points

3D Transformations

- 3D is (mostly) straightforward extension of 2D
- Homogeneous vector is 4x1
- Homogeneous matrix is 4x4
- Intuitive analogy is that a shear by w in 4D, results in a 3D translation in the w=1 plane

3D Homogeneous Scale Matrix

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} S_x * x \\ S_y * y \\ S_z * z \\ 1 \end{bmatrix}$$

3D Homogeneous Translation Matrix

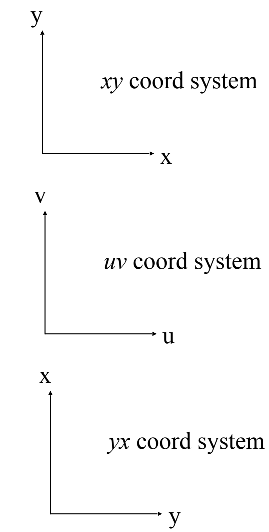
$$\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x + d_x \\ y + d_y \\ z + d_z \\ 1 \end{bmatrix}$$

3D Homogeneous Rotation

- 3D Rotation is more complicated
  - Rotation occurs about an axis, not a point
  - Need an unambiguous way to define the axis and to define what is a positive angle of rotation
  - This requires a careful definition of the coordinate system being used

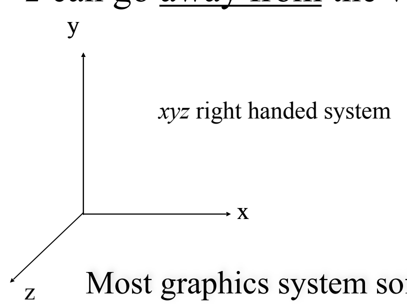
2D Axis Ordering Conventions

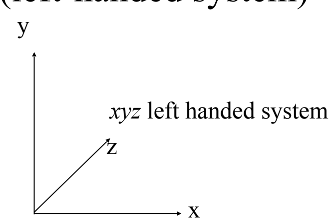
- In 2D
  - If we say the axes are labeled *xy,* we assume that *x* is horizontal, increasing to the right, and *y* is vertical increasing upward
  - It's pure convention!



3D Axis Ordering Conventions

- Given *xyz* coordinate system, *xy* should match the 2D case with $z \perp xy$ plane
  - *z* can come <u>toward</u> the viewer (right-handed system)
    - thumb is *x,* index finger is *y,* big finger is *z*
  - *z* can go <u>away from</u> the viewer (left-handed system)





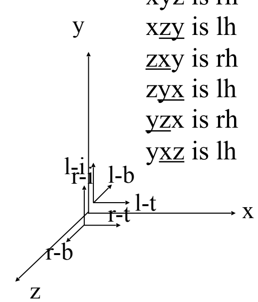
Most graphics system software defaults to right handed.

It's a <u>labeling</u> convention!

- The <u>same</u> axis labeling can be listed in any order
  - Circular rotation of the same letters has no effect

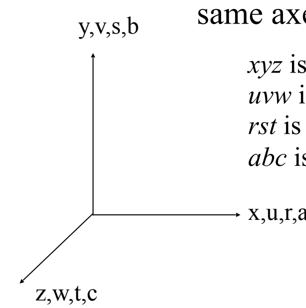
    xyz, zxy, yzx are all rh
  - Swapping 2 adjacent letters changes handedness

xyz is rh
x<u>zy</u> is lh
<u>zx</u>y is rh
z<u>yx</u> is lh
<u>yz</u>x is rh
y<u>xz</u> is lh



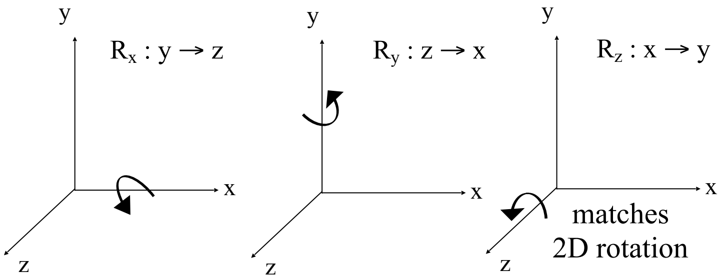
Any labeling can be assigned to the same axes.

xyz is rh
uvw is rh
rst is rh
abc is rh



3D Rotation Conventions

- Rotation is defined about an *axis*
- Elementary rotations are defined about each of the principal coordinate axes
  - A positive angle is *CCW* as seen by looking at the origin from +∞ along the axis of rotation



mnemonic:
given xyz coord
$x \rightarrow y \rightarrow z \rightarrow x$
R_z R_x R_y

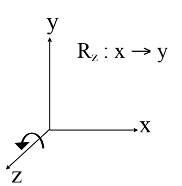
3D Homogeneous Rotation Matrices

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_y = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

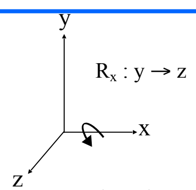
Map Rotations to Same View

Canonical representation:
xyz axes: rotate about z

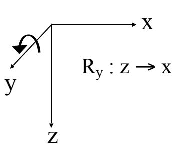


Can re-orient the axes to show that all RH rotations are really same:

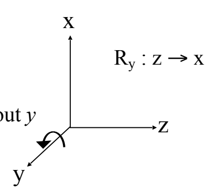
1. Map to rotation about y
2. Map to rotation about x



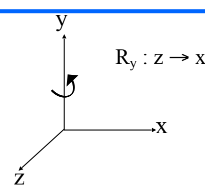
1a. Rotate view about x, get y out



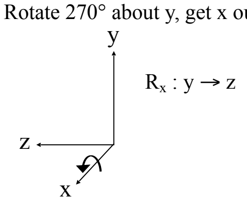
1b. Rotate view about y, get x up



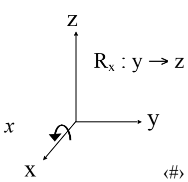
1c. *zxy* axes:
rotate about *y*



2a. Rotate 270° about y, get x out



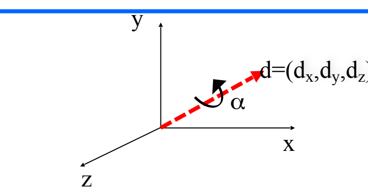
2b. Rotate 270° about x, get z up

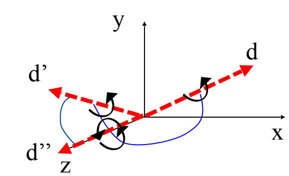


2c. *yzx* axes:
rotate about *x*

Rotation about Arbitrary Direction

- Rotate α degrees about a *direction,* $d=(d_x,d_y,d_z)$ using axis through (0,0,0)
  1. R_y: rotate world about y-axis until transformed d is in YZ
  2. R_x: rotate world about x-axis until d’ along z
  3. **Rotate α about z**
  4. “Undo” R_x
  5. “Undo” R_y





Next: do this in more detail

Rotation about Arbitrary Axis

- Rotate α degrees about an axis defined by a point (c_x,c_y,c_z) and a *direction,* $d=(d_x,d_y,d_z)$
  - Translate (c_x,c_y,c_z) to origin: $T=T(-c_x,-c_y,-c_z)$
  - Rotate about x so that (d_x,d_y,d_z) lies in xy plane
  - Rotate about z so that d' (transformed d) lies along y axis
  - Rotate α degrees about y axis: $R_y(\alpha)$
  - Apply inverse of z rotation
  - Apply inverse of x rotation
  - Apply inverse of translation: $T^{-1}=T\ (c_x,c_y,c_z)$

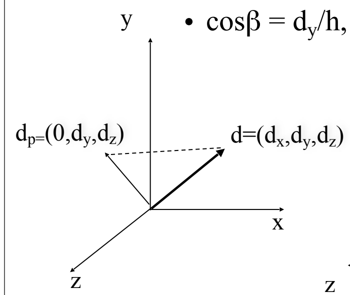
Key steps

Composite transformation: $R = T^{-1}R_x^{-1}R_z^{-1}R_y(\alpha)R_zR_xT$

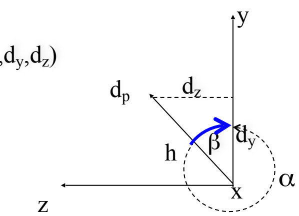
R_x: map d to xy plane

R_x to get $d=(d_x,d_y,d_z)$ into xy plane

- Project d onto yz plane: $d_p=(0,d_y,d_z)$
- Rotate about x to map d_p to y
- Since rotating z to y, $\beta < 0$, need $\alpha = 360-\beta$
- Don't need β, but cos β and sin β
- $\cos\beta = d_y/h$, $\sin\beta= d_z/h$ where $h = \mathrm{sqrt}(d_y*d_y + d_z*d_z)$



3D sketch



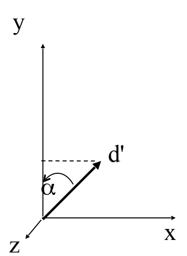
2D sketch

- $\alpha=360-\beta$
- $\cos(360-\beta) = \cos\beta$
- $\sin(360-\beta) = -\sin\beta$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & d_y/h & d_z/h & 0 \\ 0 & -d_z/h & d_y/h & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

R_z: Map d' to y axis

- $d' = R_x*d = (d'_x,d'_y,0)$
- $L = \mathrm{sqrt}(d_x*d_x + d_y*d_y + d_z*d_z)=\mathrm{length}(d)=\mathrm{length}(d')$
- $\cos\alpha = d'_y/L$
- $\sin\alpha= d'_x/L$



$$R_z = \begin{bmatrix} d'_y/L & -d'_x/L & 0 & 0 \\ d'_x/L & d'_y/L & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Note: if d is a *unit* vector (or has been converted to one), L = 1.

Composite Rotation Matrix

- The composite rotation matrix is:

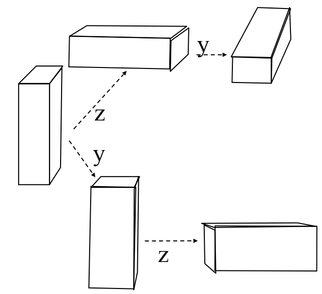
  $R = T(c_x,c_y,c_z)R_x^{-1}R_z^{-1}R_y(\alpha)R_zR_xT(-c_x,-c_y,-c_z)$
  - but what are the inverse rotations?
  - $R^{-1}(\alpha) = R(-\alpha)=R^T(\alpha)$ = transpose of $R(\alpha)$
  - And
    - $\cos\alpha = \cos(-\alpha)$
    - $\sin\alpha = -\sin(-\alpha)$
  - So, just change the location of the -sin term
- A lot of computation, but composite is computed once and applied to hundreds or thousands of points.

Alternate Rotation Formulation

- We arbitrarily chose to use R_x to map to xy plane, then R_z to align with y-axis
- Could do R_x, then R_z to align with x-axis
- Could do R_x to map to xz plane then R_z to align with x-axis, then R_y to align with either x- or z-axis
- Do some!

Are 3D Rotations Symmetric?

- Given two affine transforms, M_1 and M_2
  - Does $M_1 * M_2 = M_2 * M_1$?
  - try 90° rotation about y and z.

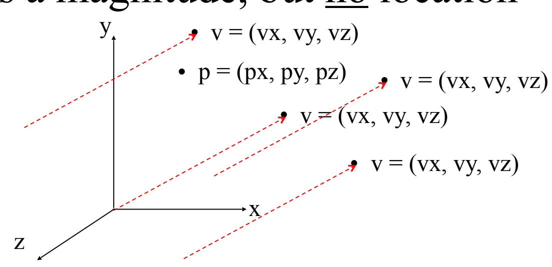


Rotation Matrices

- Pure rotation matrices are <u>orthogonal</u>
  - All columns are unit length
  - All columns are mutually orthogonal
    - their dot products are 0
- All the basic rotations are orthogonal
- Any composition of orthogonal matrices is orthogonal
- $R^{-1}=R^T$ for any orthogonal matrix R

Points and Vectors

- 3D points and vectors both need 3 components.
  - Point: represents a <u>location</u> in a coordinate system; it has <u>no</u> magnitude and <u>no</u> direction
  - Vector: represents a <u>direction</u> in a coordinate system; it has a magnitude, but <u>no</u> location



Homogeneous Coordinate Vector

- Homogenous coordinates provide a convenient opportunity to disambiguate the Vector/Point representation:
  - 3D point: [x y z 1]
  - 3D vector [x y z 0]
    - Matrix * vector operation "ignores" translation part of the Matrix -- since those parts are multiplied by 0
    - But, that is appropriate: vectors have no location (or they are at <u>all</u> locations), so translation is meaningless

Coordinate Systems

- A 3D <u>coordinate system</u> requires specification of an *origin* plus 3 orthogonal *vectors* representing the coordinate system *axes*
  - but, what coordinate system are those defined in?
  - usually a coordinate system's origin is *implicit* and treated as (0,0,0) and its vectors are (1,0,0), (0,1,0) and (0,0,1) and we just accept it as given
- What happens when we need to deal with multiple coordinate systems simultaneously?
  - need to be more rigorous!

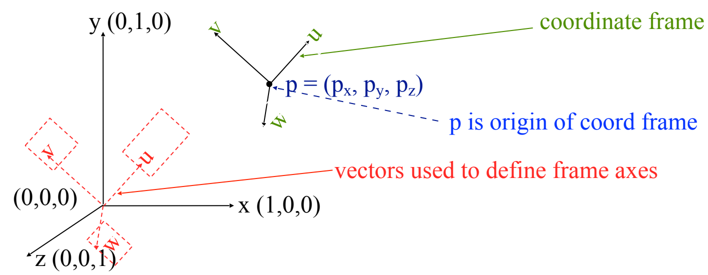
Coordinate Frame

- A <u>coordinate *frame*</u> explicitly describes a coordinate system <u>in another coordinate system</u> via 3 orthogonal vectors and a point:
  F={u,v,w,p}
- Can compactly represent a coordinate frame as a 4x4 matrix:

$$F = \begin{bmatrix} u_x & v_x & w_x & p_x \\ u_y & v_y & w_y & p_y \\ u_z & v_z & w_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Coordinate Frame Example

- Normally, specify coordinate frames in a *world* coordinate system, whose origin is (0,0,0) and whose axes are defined by (1,0,0), (0,1,0), and (0,0,1)



Coordinate Frame Transformation

- Consider the Frame matrix as a transformation
  - What does it do to the frame's u-axis vector ([1 0 0 0])?

$$F * \begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}u_x & v_x & w_x & p_x\\ u_y & v_y & w_y & p_y\\ u_z & v_z & w_z & p_z\\ 0 & 0 & 0 & 1\end{bmatrix} * \begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}u_x\\u_y\\u_z\\0\end{bmatrix}$$

  - It transforms it to the world's representation
  - Similarly, the v-axis [0 1 0 0]→[v_x v_y v_z 0]
    and the w-axis [0 0 1 0]→[w_x w_y w_z 0]
  - Finally, the frame origin [0 0 0 1]→[p_x p_y p_z 1]
  - This works for <u>any</u> point represented in the frame

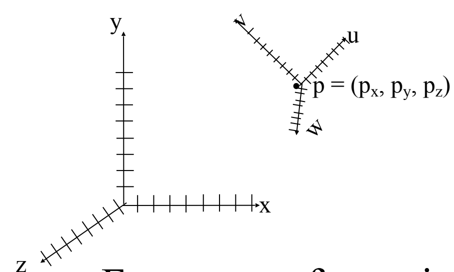
Compositing Frame Transformations

- Let F_1 be a frame transform specified in the *world.*
- Let F_2 be a frame transform specified in F_1 coords.
- How do we map coordinates in F_2 to world?
  - F_2 matrix maps from F_2 coords to F_1 coords
  - F_1 matrix maps from F_1 coords to world coords
  - So F_1*F_2 will map any coordinate from F_2 to world
- Key: successive frame transformations are composed by *post-multiplying* the new transform times the previous.
  - This composition is opposite to the ordering of successive transformations of the points, but the semantics is consistent.

Inverse Transformation

- Sometimes we have points in the world that need to be represented in a coordinate frame.
  - Can get these by finding the inverse of the coordinate frame transformation
  - General inverse matrix calculations are expensive
  - Composite affine transforms built from simple components are cheap:
    - $T^{-1}(d_x, d_y, d_z) = T(-d_x, -d_y, -d_z)$
    - $S^{-1}(s_x, s_y, s_z) = S(1/s_x, 1/s_y, 1/s_z)$
    - $R^{-1}(\alpha) = R(-\alpha)=R^T(\alpha)$ -- transpose matrix!
      - In fact, the inverse of any composition of rotation matrices is the transpose of the composite!

Inverse Transformation Example



uvw frame specification:
- Composite rotation, R
- Scale, S(s_x, s_y, s_z)
- Translate(p_x, p_y, p_z)

- Frame transformation, F = TRS
- $F^{-1} = (TRS)^{-1} = S^{-1}R^{-1}T^{-1}$
  $= S(1/s_x,1/s_y,1/s_z)\ R^T\ T(-p_x,-p_y,-p_z)$

Frame Transformation Revisited

- F defined by vectors defining the new axes
- These vectors need not be unit length!
- How are scale and rotate embedded in the upper 3x3 submatrix?

$$F = \begin{bmatrix} u_x & v_x & w_x & p_x \\ u_y & v_y & w_y & p_y \\ u_z & v_z & w_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let $A = U/\|U\|$, $B = V/\|V\|$, $C = W/\|W\|$ (Unit length vectors)

$$\text{Let } R = \begin{bmatrix} a_x & b_x & c_x & 0 \\ a_y & b_y & c_y & 0 \\ a_z & b_z & c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } S = \begin{bmatrix} \|U\| & 0 & 0 & 0 \\ 0 & \|V\| & 0 & 0 \\ 0 & 0 & \|W\| & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It's easy to show that $F = TRS$

Hierarchical Object Definition

- A *Shape* class object definition should define an object in a convenient "local" coordinate system
- Instances of the *Shape* class should be placed into another coordinate system subject to an appropriate transformation.
  - could be the "world" or another *Shape's* local coordinate system.
- Results in *nested* hierarchical shape definition

Review

- Coordinate systems
- 2D Transformations
- Homogeneous coordinates
- Matrices, vectors, points
- 3D Transformations
- Coordinate Frames

Next

- 3D Viewing
- Synthetic camera model
- Modeling transformations
- Viewing transformations