Binghamton University

EngiNet™

State University of New York

Thomas J. Watson

School of Engineering
and Applied Science

EngiNet™

WARNING

CS 460/560

Computer Graphics

Professor Richard Eckert

Lecture # 20

April 18, 2001

Lecture 20

- 3-D Graphics
  - 3-D Geometric Transformations
  - 3-D Viewing Transformation
  - Projection Transformation
  - Hidden Surface Removal

Review of 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')

$$\begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix} \rightarrow \begin{vmatrix} x' \\ y' \\ z' \\ 1 \end{vmatrix}$$

Translations

- Given 3-D translation vector T=(tx, ty, tz)
- Represent translation as matrix equation

$$P' = T * P$$

$$T = \begin{vmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Scaling with respect to origin

- Given three scaling factors sx, sy, sz

$$P' = S * P$$

- S is the following 4 X 4 scaling matrix:

$$S = \begin{vmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

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

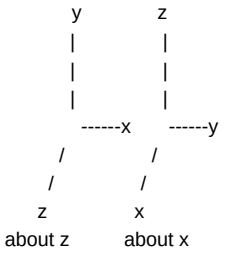
P' = Rz * P

- **Z-Axis Rotation Matrix, Rz**

$$Rz = \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Rx Matrix for rotations about x-axis

- Symmetry argument



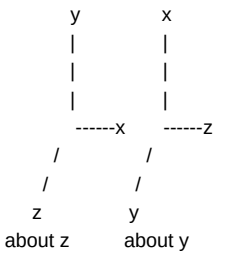
Make replacements:
x --> y
y --> z
z --> x

- P' = Rx * P

$$Rx = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Ry Rotation Matrix

- Symmetry:



Replacements:
x --> z
y --> x
z --> y

P' = Ry * P

$$Ry = \begin{vmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

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 T1, T2, ..., Tn
- Equivalent to a single transformation
  - Represented by composite transformation matrix T
  - T is given by the matrix product:
    T = Tn*...*T2*T1
- Just like in 2-D, except matrices are 4 X 4

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

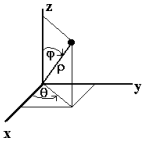
Rotation about an Arbitrary Axis

- Rotate point P by angle α about a line
- Given: endpoints P1=(x1,y1,z1) & P2=(x2,y2,z2)
- Convert problem into rotation about x-axis
  1. Translate so that P1 is at origin: T1 = T(-x1,-y1,-z1)
  2. Compute spherical coordinates of the other endpoint:

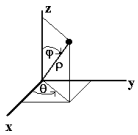
$\rho = \text{sqrt}((x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2)$

$\phi = \arccos((z2-z1)/\text{rho})$

$\theta = \arctan((y2-y1)/(x2-x1))$



  - 3. Rotate about z-axis by $-\theta$ so line lies in x-z plane: T2 = Rz($-\theta$)
  - 4. Rotate about y-axis by $(90-\phi)$ to make line coincide with x-axis: T3 = Ry$(90-\phi)$



  - 5. Rotate about x-axis by given angle α: T4 = Rx(α)
  - 6. Rotate back to undo step 4: T5 = Ry(ϕ-90)
  - 7. Rotate back to undo step 3: T6 = Rz(θ)
  - 8. Translate back to undo step 1: T7 = T(x1,y1,z1)
- Composite transformation then will be:
  T = T7*T6*T5*T4*T3*T2*T1

3-D Coord. 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:
  Tcoord(x,y,z) = Tgeom(-x,-y,-z)
- For each principal-axis, rotation relationship is:
  Rcoord(θ) = Rgeom(-θ)
- Useful in deriving 3-D viewing transformation

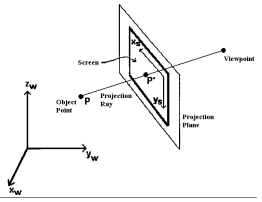
3D Viewing and Projection

- 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

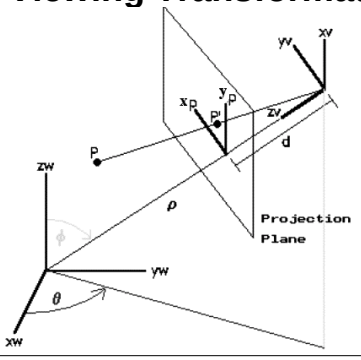


3-D Viewing Transformation

- Converts world coordinates (xw,yw,zy) of a point to viewing coordinates (xv,yv,zv) of the point
  - As seen by a "camera" that is going to "photograph" the scene

(xw,yw,zw) ----------> (xv,yv,zv)
Viewing transformation

3-D Viewing Transformation



Projection Transformation

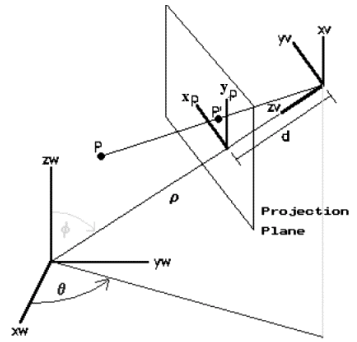
- Converts viewing coordinates (xv,yv,zv) of a point to 2-D coordinates (xp,yp) of point's projection onto a projection plane
- Think of projection plane as containing screen upon where image is to be displayed

(xv,yv,zv) ----------> (xp,yp)
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



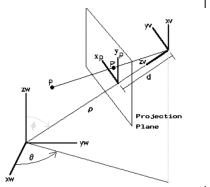
Parameters

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

Viewing Setup Properties

- VCS 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 ρ must be greater than d
- Center of projection coincides with VCS origin



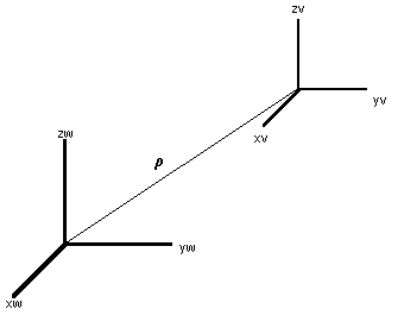
- VCS's yv-axis is parallel to projection of WCS's zw-axis
  - So WCS zw-axis defines "screen up" direction
- VCS's xv-axis is chosen so that xv-yv-zv axes form a left-handed coordinate system

  objects far from the VCS's origin have large zv
- 2-D PP coordinate system's origin is at intersection of ρ 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

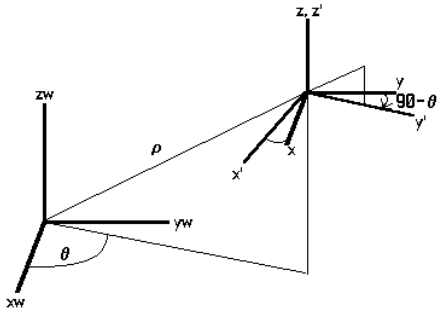
3-D Viewing Transformation

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

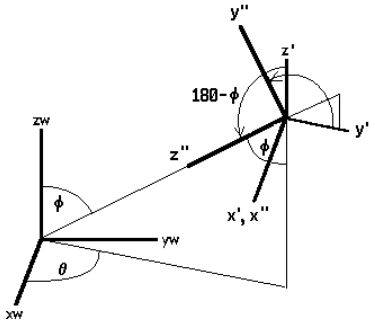
Viewing Xform: 1. Translate by ρ



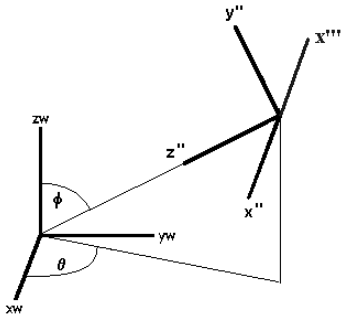
2. Rotate by -(90-θ) about z



3. Rotate by (180-ϕ) about x



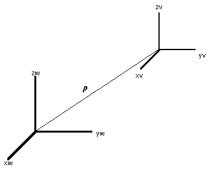
4. Invert x-axis



1. Translate by ρ

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

$$T_{geom} = \begin{vmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$



- Use relationship between coordinate system transformations & geometric transformations:

  Tcoord(x,y,z) = Tgeom(-x,-y,-z)

- So first transformation matrix, T1:

$$T1 = \begin{vmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- Express x, y, z in terms of ρ, θ, ϕ

  $x = \rho*\sin(\phi)*\cos(\theta)$

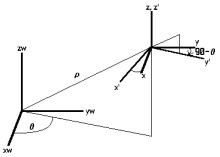
  $y = \rho*\sin(\phi)*\sin(\theta)$

  $z = \rho*\cos(\phi)$

2. Rotate by -(90-θ) about z

- Use relationship between coordinate system rotations & geometric rotations:
  Tcoord(alpha) = Tgeom(-alpha)
- So transformation is T2 = Rz(90-θ):

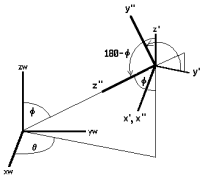
$$T2 = \begin{vmatrix} \cos(90-\theta) & -\sin(90-\theta) & 0 & 0 \\ \sin(90-\theta) & \cos(90-\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$



3. Rotate by (180-ϕ) about x

- Again use relationship between geometric & coordinate system rotations:
  So T3 = Rx(ϕ -180):

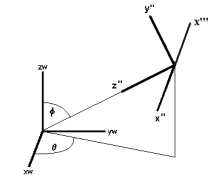
$$T3 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi-80) & -\sin(\phi-80) & 0 \\ 0 & \sin(\phi-80) & \cos(\phi-80) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$



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

$$T4 = \begin{vmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$



Composite Viewing Transformation Matrix

- Tv = T4*T3*T2*T1
- Result:

$$Tv = \begin{vmatrix} -\sin(\theta) & \cos(\theta) & 0 & 0 \\ -\cos(\phi)*\cos(\theta) & -\cos(\phi)*\sin(\theta) & \sin(\phi) & 0 \\ -\sin(\phi)*\cos(\theta) & -\sin(\phi)*\sin(\theta) & -\cos(\phi) & \rho \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Projection Transformation

- Look down xv axis at viewing setup:

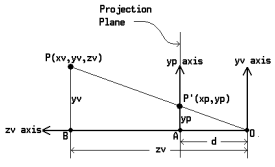
Triangles OAP' & OBP are similar

So set up proportion:

$$\frac{yp}{yv} = \frac{d}{zv}$$

Solve for yp:

$yp = (yv*d)/zv$



Look down yv axis for xp:

Result: $xp = (xv*d)/zv$

Plotting Points on Screen

- Get screen coordinates (xs,ys) from Projection Plane coordinates (xp,yp)
- Final Transformation:
  2D Window-to Viewport Transformation
  (xs,ys) <--- (xp,yp)
  See earlier notes
  - Replace xv,yv with xs,ys
  - Replace xw,yw with xp,yp

Simple Hidden Surface Removal (Back-Face Culling)

- For complex objects there are lots of polygons
- Many polygons not visible
  - Because they face away from observer
- More realistic, less complex image is produced if only visible polygons are drawn
  - So draw only those facing toward observer
- Technique of back-face culling determines if polygon is visible or not

Back-Face Culling

- Define one side of each polygon to be the visible side
  - That side is the outward-facing side
- Defining each polygon in the polygons array:
  - Systematically number vertices in counter-clockwise fashion as seen from outside of the object

First: Review of Vector Products

- Dot (Scalar) Product

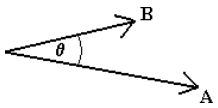
  $s = A \cdot B$

  $s = |A| * |B| * \cos(\theta)$

  θ is the angle between vectors A and B

  In terms of components (RH coord system):

  $s = Ax*Bx + Ay*By + Az*Bz$



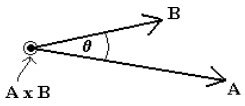
Cross (Vector) Product

- V = A x B, a vector
- Magnitude: $|V| = |A| * |B| * \sin(\theta)$

  θ is angle between vectors A and B
- Direction: Given by right-hand rule
  - 1. Align fingers of right hand with first vector
  - 2. Rotate toward second
  - 3. Thumb points in direction of V

In the following diagram:

V = A x B would point out of the screen toward the observer



In terms of components (RH coordinate system):

$$V = \begin{vmatrix} i & j & k \\ Ax & Ay & Az \\ Bx & By & Bz \end{vmatrix}$$

i, j, k are unit vectors along x,y,z axes

Triple Product

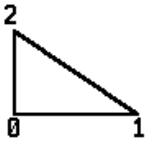
$A \cdot (B \times C)$

$$A \cdot (B \times C) = \begin{vmatrix} Ax & Ay & Az \\ Bx & By & Bz \\ Cx & Cy & Cz \end{vmatrix}$$

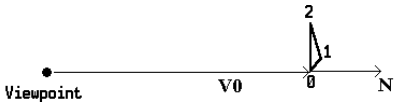
(Components in terms of RH coord system)

Back to Back-Face Culling

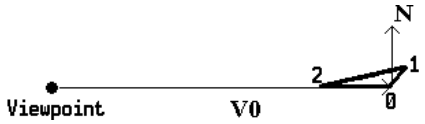
- Consider triangle with vertices 0, 1, 2
- Visible side of the triangle: 0,1,2
  - Vertices numbered in counter-clockwise order
  - Invisible side is: 0,2,1
    - (clockwise vertex ordering)



- Define vector N
  - Outward normal to triangle
- DefineVectorV0,
  - Vector from observer to vertex 0
- Some Cases:
  - N and V0 nearly parallel (V0 . N = 1)
  - Visible side of triangle 012 invisible to viewer



- Rotate triangle about side 01 by 90 degrees
  - Now N and V0 are perpendicular (V0 . N = 0)
  - Triangle is about to become visible
  - At all other points between these two orientations:
    - V0 . N is positive
    - And triangle is invisible to viewer



- Continue rotation about side 01
- Triangle becomes visible to the viewer
- 90 degrees more, N and V0 are antiparallel

  V0 . N = -1

  Triangle facing toward viewer and is visible
  - At all intermediate orientations:
    - Triangle is visible
    - And V0 . N is negative