Transformations, continued

3D Rotation

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} (r_{11}, r_{12}, r_{13}) \bullet (x, y, z) \\ (r_{21}, r_{22}, r_{23}) \bullet (x, y, z) \\ (r_{31}, r_{32}, r_{33}) \bullet (x, y, z) \end{pmatrix}$$

So if the rows of R are orthogonal unit vectors (orthonormal), they are the axes of a new coordinate system, and matrix multiplication rewrites (x,y,z) in that coordinate system.

This also means that $RR^T = I$

This means that R^T is a rotation matrix that undoes *R.*

Alternately, …

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} r_{11} \\ r_{21} \\ r_{31} \end{pmatrix}$$

So R takes the x axis to be a vector equivalent to the first column of R.

Similarly, the y and z axes are transformed to be the second and third columns of R.

If R is a rotation, then the transformed axes should still be orthogonal unit vectors. So the columns of R should be orthonormal.

Simple 3D Rotation

$$\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & . & . & . & x_n \\ y_1 & y_2 & & & & y_n \\ z_1 & z_2 & & & & z_n \end{pmatrix}$$

Rotation about z axis.

Rotates x,y coordinates. Leaves z coordinates fixed.

Full 3D Rotation

$$R = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{pmatrix}$$

- Any rotation can be expressed as combination of three rotations about three axes.

$$RR^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Rows (and columns) of *R* are orthonormal vectors.
- R has determinant 1 (not -1).

- Intuitively, it makes sense that 3D rotations can be expressed as 3 separate rotations about fixed axes. Rotations have 3 degrees of freedom; two describe an axis of rotation, and one the amount.
- Rotations preserve the length of a vector, and the angle between two vectors. Therefore, (1,0,0), (0,1,0), (0,0,1) must be orthonormal after rotation. After rotation, they are the three columns of R. So these columns must be orthonormal vectors for R to be a rotation. Similarly, if they are orthonormal vectors (with determinant 1) R will have the effect of rotating (1,0,0), (0,1,0), (0,0,1). Same reasoning as 2D tells us all other points rotate too.
    - Note if R has determinant -1, then R is a rotation plus a reflection.

3D Rotation + Translation

- Just like 2D case

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation about a known axis

- Suppose we want to rotate about u.
- Find R so that u will be the new z axis.
  - u is third row of R.
  - Second row is anything orthogonal to u.
  - Third row is cross-product of first two.
  - Make sure matrix has determinant 1.
- Then rotate about (new) z axis.
- Then apply inverse of first rotation.

Let's look at an example of this. Suppose we want to rotate about the direction (1,1,1). A unit vector in this direction is:

$\frac{(1,1,1)}{\sqrt{3}}$. So we create a matrix like: $\begin{pmatrix} & & \\ & & \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix}$. Next, we need the first column to be a unit vector orthogonal to this. We can use $\left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0\right)$. I got this by guessing, but it's easy to verify. This gives us: $\begin{pmatrix} & & \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix}$. Taking the cross-product, we get the final row:

$$\begin{pmatrix} -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix}$$

Let's call that matrix R1. We apply R1, then apply a matrix that rotates about the z axis. Then the inverse of R1, to go back. This could look like:

$$\begin{pmatrix} -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{6}} & 0 & \frac{1}{\sqrt{3}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix}$$

This should rotate everything by 45 degrees about the axis in the direction (1,1,1). To verify this, check what happens when we apply this matrix to (2,2,2). It stays fixed. How else can we check this does the right thing?

Transformation of lines/normals

- 2D. Line is set of points (x,y) for which $(a,b,c).(x,y,1)^T=0$. Suppose we rotate points by R. Notice that:

$(a,b,c)R^T R(x,y,1)^T=0$

So, $(a,b,c)R^T$ is the rotation of the line (a,b,c).

Surface normals are similar, except they are defined by $(a,b,c).(x,y,z)^T = 0$

OpenGL

- Basically, OpenGL let's you multiply all objects by a matrix as they are drawn.
- Routines allow you to manage multiple matrices (pushing and popping).
- Routines allow you to combine many matrices (multiplied together in postfix order).
- Routines create matrices for you (translation, rotation about an axis, viewing).

Hierarchical Transformations in OpenGL

- Stacks for Modelview and Projection matrices
- ***glPushMatrix( )***
  - push-down all the matrices in the active stack one level
  - the top-most matrix is copied (the top and the second-from-top matrices are initially the same).
- ***glPopMatrix( )***
  - pop-off and discard the top matrix in the active stack
- Stacks used during recursive traversal of the hierarchy.
- Typical depths of matrix stacks:
  - Modelview stack = 32 (aggregating several transformations)
  - Projection Stack = 2 (eg: 3D graphics and 2D help-menu)

OpenGL Transformation Support

- Three matrices
  - GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE
  - *glMatrixMode**(mode)*** specifies the active matrix
- ***glLoadIdentity( )***
  - Set the active matrix to identity
- ***glLoadMatrix{fd}(TYPE *m)***
  - Set the 16 values of the current matrix to those specified by ***m***
- ***glMultMatrix{fd}(TYPE *m)***
  - Multiplies the current active matrix by ***m***

$$m = \begin{pmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{pmatrix}$$

Transformation Example

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity( );          // matrix = I
glMultMatrix(N);            // matrix = N
glMultmatrix(M);            // matrix = NM
glMultMatrix(L);            // matrix = NML
glBegin(GL_POINTS);
glVertex3f(v);      // v will be transformed:
   NMLv
glEnd( );
```

OpenGL Transformations

- ***glTranslate{fd}(TYPE x, TYPE y, TYPE z)***
  – Multiply the current matrix by the translation matrix
- ***glRotate{fd}(TYPE angle, TYPE x, TYPE y, TYPE z)***
  – Multiply the current matrix by the rotation matrix that rotates an object about the axis from (0,0,0) to (x, y, z)
- ***glScale{fd}(TYPE x, TYPE y, TYPE z)***
  – Multiply the current matrix by the scale matrix

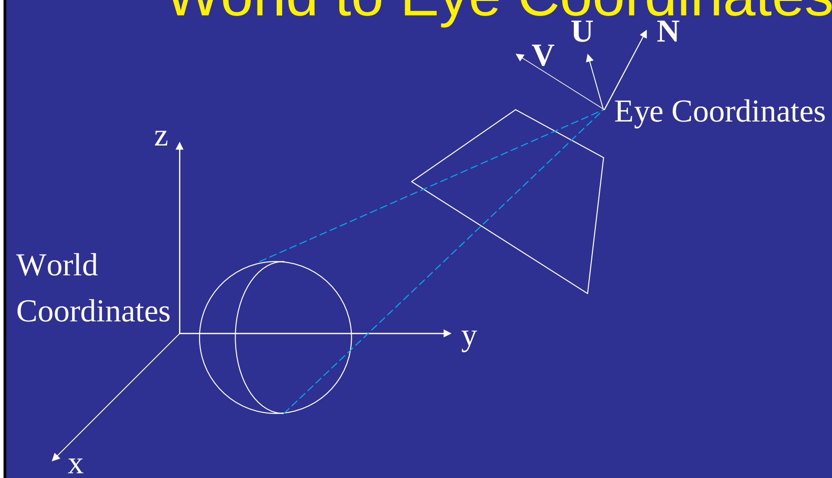
Examples

```
glMatrixMode(GL_MODELVIEW);
glRecti(50,100,200,150);
glTranslatef(-200.0, -50.0, 0.0);
glRecti(50,100,200,150);
glLoadIdentity();
glRotatef(90.0, 0.0, 0.0, 1.0);
glRecti(50,100,200,150);
glLoadIdentity();
glscalef(-.5, 1.0, 1.0)
glRecti(50,100,200,150);
```

Viewing in 3D

- World (3D) $\rightarrow$ Screen (2D)
- Orienting Eye coordinate system in World coordinate system
  - View Orientation Matrix
- Specifying viewing volume and projection parameters for $\Re^{n} \rightarrow \Re^{d}$ (d < n)
  - View Mapping Matrix

World to Eye Coordinates



World to Eye Coordinates

- We need to transform from the world coordinates to the eye coordinates
- The eye coordinate system is specified by:
  - View reference point (VRP)
    - (VRP_x, VRP_y, VRP_z)
  - Direction of the axes: eye coordinate system
    - $\mathbf{U} = (u_x, u_y, u_z)$
    - $\mathbf{V} = (v_x, v_y, v_z)$
    - $\mathbf{N} = (n_x, n_y, n_z)$

World to Eye Coordinates

- There are two steps in the transformation (in order)
  - Translation
  - Rotation

World to Eye Coordinates

- Translate World Origin to VRP

$$\begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 & -VRP_x \\ 0 & 1 & 0 & -VRP_y \\ 0 & 0 & 1 & -VRP_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

World to Eye Coordinates

- Rotate World X, Y, Z to the Eye coordinate system *u, v, n*, also known as the View Reference Coordinate system

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix}$$

Let's take an example. Suppose we have a bird's eye view of the world. We are looking from above down on the world. What is a possible view reference point? How about (0,50,0). What is a possible viewing direction (n)? (0, -1, 0). What would be a reasonable up vector (v)? How about (0,0,1)? How does our image change as we pick a different one? So what is the translation matrix we get:

1 0 0 0

0 1 0 -50

0 0 1 0

0 0 0 1

And what is our rotation matrix:

1 0 0 0

0 0 1 0

0 -1 0 0

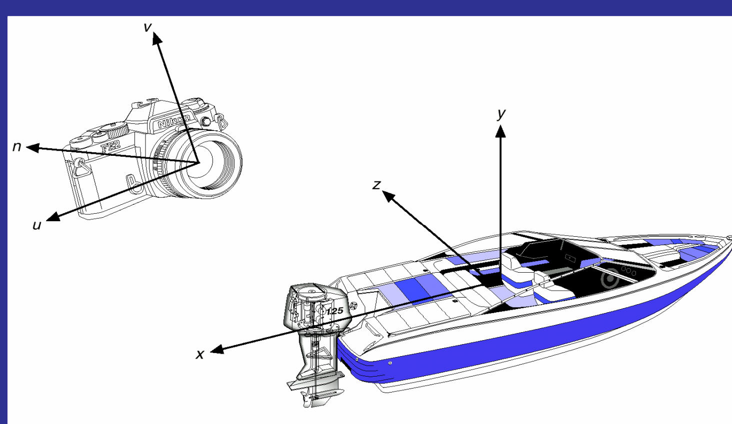
0 0 0 1

Does this make sense? What are the coordinates of a point on the ground. For example, the point (0 0 0)? Multiply the translation matrix and we get (0 -50 0 1). Multiply this by rotation matrix and we get:

(0 0 50 1). This point seems to have a distance of 50 in front of us, and to otherwise be at the origin.

What about a point at (0 0 10)? Where should this appear? Since (0,0,1) is the up vector, this should appear to be distant, and above. Translating we get (0 -50 10 1). Rotating we get: (0 10 50 1). 50 units in front of us, and up by 10.

Camera Analogy



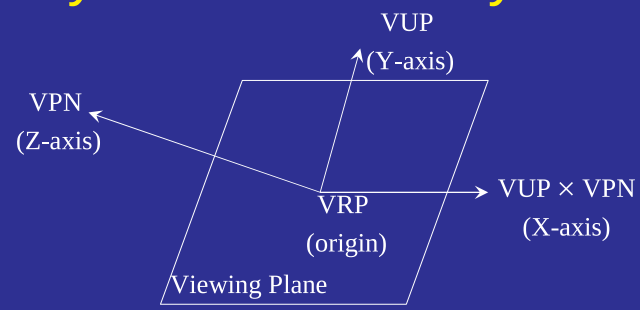
Specifying 3D View
(Camera Analogy)

- Center of camera (x, y, z) : 3 parameters
- Direction of pointing (θ,φ) : 2 parameters
- Camera tilt (ω) : 1 parameter
- Area of film (w, h) : 2 parameters
- Focus (f) : 1 parameter

Specifying 3D View

- Center of camera (x, y, z) : View Reference Point (VRP)
- Direction of pointing (θ,φ) : View Plane Normal (VPN)
- Camera tilt (ω) : View Up (VUP)
- Area of film (w, h) : Aspect Ratio (w/h), Field of view (fov)
- Focus (f) : Will consider later

Eye Coordinate System



- View Reference Point (VRP)
- View Plane Normal (VPN)
- View Up (VUP)

World to Eye Coordinates

- Translate World Origin to VRP
- Rotate World X, Y, Z to the Eye coordinate system, also known as the View Reference Coordinate system, VRC = (VUP × VPN, VUP, VPN), respectively:

$$\begin{pmatrix} (\ \mathrm{VUP} \times \mathrm{VPN}\ ) & & & 0 \\ (\ \ \mathrm{VUP}\ \ ) & & & 0 \\ (\ \ \mathrm{VPN}\ \ ) & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Eye Coordinate System
(OpenGL/GLU library)

- ***gluLookAt*** *(eye_x , eye_y , eye_z , $lookat_x$, $lookat_y$, $lookat_z$, up_x , up_y , up_z);*
- In our terminology:

  eye = VRP

  lookat = VRP + VPN

  up = VUP
- ***gluLookAt*** also works even if:
  – *lookat* is any point along the *VPN*
  – *VUP* is not perpendicular to *VPN*

gluLookAt()

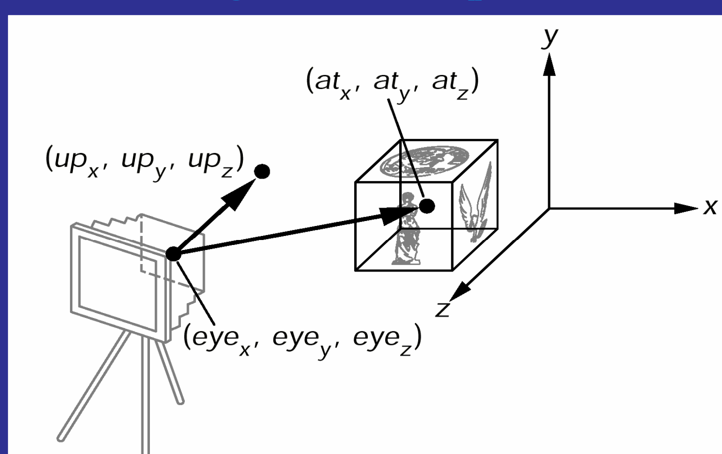


Image from: Interactive Computer Graphics by Ed Angel

Eye Coordinate System
(OpenGL/GLU library)

- This how the ***gluLookAt*** parameters are used to generate the eye coordinate system parameters:

  VRP = *eye*

  VPN = (*lookat* - *eye*) / $\|$ (*lookat* - *eye*) $\|_2$

  VUP = VPN $\times$ (*up* $\times$ VPN)
- The eye coordinate system parameters are then used in translation T(VRP) and rotation R(XYZ $\rightarrow$VRC) to get the view-orientation matrix