Problem #2

Practice with Viewing Transformation,

or,

Whose coordinate system is it, anyway?

COMPUTER GRAPHICS

15-462

13 Feb. 2001

A Popular View Specification Approach

- **Focal length, image size/shape and clipping planes are in the perspective transformation**
- **In addition:**
  - ***lookfrom:*** **where the focal point (camera) is**
  - ***lookat:*** **the world point to be centered in the image**
- **Also specify camera orientation about the *lookat-lookfrom* axis**
  - ***vup:*** **a vector that is pointing straight up in the image -- usually want world up dir.**

The Notorious Problem #2

» I wish to use OpenGL to render a set of X, Y, and Z axes. The axes will be drawn in world coordinates, with each axis 10 units long and pointing in the X, Y, and Z directions respectively. They come together at the origin of world coordinates, and they make a right-handed coordinate system. In other words, the axes are set up just as you would expect.

» I want to view these axes from a point on the X axis 20 units in the positive X direction from the origin. Thus my "camera" is looking in the negative X direction, so my view plane in world coordinates is the Y-Z plane. I want the Z axis to point toward the top of my image, and I want to set up a 60 degree field of view. I want the near clipping distance to be at X=10, half way between my "camera" and the origin. I want the far clipping distance to be at X=-10, 10 units beyond the origin. The final output image should be 640 by 480 pixels.

Setting Up OpenGL

What are the parameters to glRotatef(), glTranslatef(), and gluPerspective() that I will need to set this up?

- **From the OpenGL man pages:**

```
void gluPerspective( GLdouble fovy,
                     GLdouble aspect,
                     GLdouble zNear,
                     GLdouble zFar )
```

Fovy= 60 degrees
aspect= 640/480
zNear= 10
zFar= 30

•Given these inputs, the transformation matrix is on the man page.
•Perspective trans happens *last*, and so gluPerspective gets called *first*.
•Normalizes the Viewing Frustrum

Implementation

Implementing the *lookat/lookfrom/vup* viewing scheme

(1) Translate by -*lookfrom*, bring focal point to origin

(2) Rotate *lookat-lookfrom* to the *z*-axis with matrix R:

- » v = (*lookat-lookfrom*) (normalized) and z = [0,0,1]
- » rotation axis: a = (v×z)/|v×z|
- » rotation angle: cosθ = v•z and sinθ = |v×z|

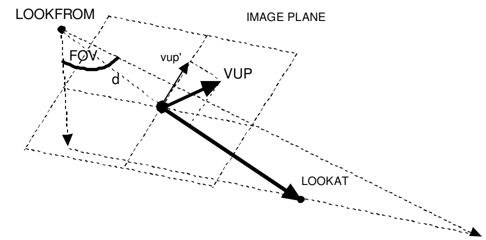
$$\mathbf{R} = \mathbf{a}\mathbf{a}^T + (\mathbf{v}\cdot\mathbf{z})(\mathbf{I} - \mathbf{a}\mathbf{a}^T) + |\mathbf{v}\times\mathbf{z}|\mathbf{a}^* \quad \text{where} \quad \mathbf{a}^* = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

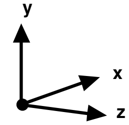
or: glRotate(θ, a_x, a_y, a_z)

(3) Rotate about *z*-axis to get *vup* parallel to the y-axis

- » watch out if *vup* is along the *z*-axis

The Whole Picture

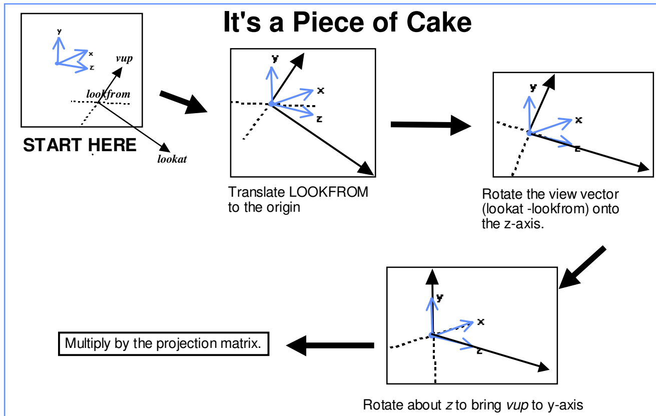




World Coordinates

LOOKFROM:	Where the camera is
LOOKAT:	A point that should be centered in the image
VUP:	A vector that should be pointing straight up in the image
FOV:	Field-of-view angle.
d:	focal length

It's a Piece of Cake



Setting Up OpenGL 2

- Follow this recipe. We'll demonstrate that the results are correct shortly.
- Translate lookfrom (at x=20, y=z=0) to the origin:

  glTranslatef(-20,0,0)
- Rotate view vector (lookfrom-lookat) onto Z axis.
- This means rotate X axis onto Z axis, a rotation by -90 degrees about Y.

  glRotatef(-90,0,1,0)
- Finally, rotate the view up direction onto the Y axis.
  - In world coords, view up was +Z
  - First rotation moved it to -X
  - Need a rotation by -90 about the rotated Z axis (which used to be X) to bring it into line with Y

  glRotatef(-90,0,0,1)

Setting Up OpenGL 3

- The first transformation that happens to a vertex is the translation.
- That means it is the *last* OpenGL call.
- Conversely, the perspective transformation happens last, and so it is called first.

gluPerspective(60,A,10,30)
glRotatef(-90,0,0,1)
glRotatef(-90,0,1,0)
glTranslatef(-20,0,0)

where A=640/480

Lining Up the Matrices

If I have a point P = (x,y,z,w)^T in homogeneous coordinates, write the matrix equation for P' = (x',y',z',w')^T in terms of P and the matrices for the transformations from (1).

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \underbrace{\begin{pmatrix} B/A & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & -2 & -30 \\ 0 & 0 & -1 & 0 \end{pmatrix}}_{\textbf{gluPerspective}} \underbrace{\begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{glRotatef}} \underbrace{\begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{glRotatef}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & -20 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{glTranslatef}} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Where B=cot(30) and A=640/480

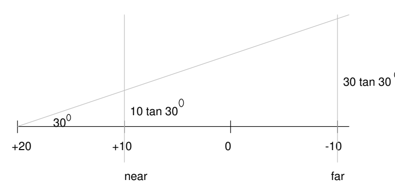
Test It Out

- Just to get confident, let's apply the translation and rotations to a few test points.
  - (these should be column vectors, obviously, but then they wouldn't fit on the slide…)
- (0,0,0,1) -> (0,0,-20,1) so the origin goes where we'd expect
- (20,0,0,1) -> (0,0,0,1) so the lookfrom point goes to the origin
- (0,0,1,1) -> (0,1,-20,1) so a point one unit up the Z axis (in the view up direction) ends up displaced in the camera up direction
- (0,1,0,1) -> (1,0,-20,1) so a point to the right of the origin in world coordinates ends up to the right in our image

View Frustum, in World Coordinates

What are the corners of the view frustum in world coordinates?

Let's look at the X-Z plane first:



(and symmetrically below the axis)

Note that tan 30° is 1/B

View Frustum in World Coordinates 2

- In the Y-Z plane, we know the aspect ratio.
- If the Z coordinate of a corner of the clip boundary is 10/B ...
- and the aspect ratio is A=640/480 …
- then the Y coordinate of that point must be 10A/B.

- Hence the world coordinate clip boundaries are:

near: (10, + or - 10A/B, + or - 10/B, 1)

far: (-10, + or - 30A/B, + or - 30/B, 1)

Normalizing the View Frustum

Use the matrix equation from (2) to show that two opposite corners of the world coordinate frustum are mapped to the corresponding corners of the normalized view frustum.

•If we just visualize the view frustum in <u>camera coordinates</u>, we can see that the frustum corners will be:

near: (+ or -10A/B, + or - 10/B, -10, 1)

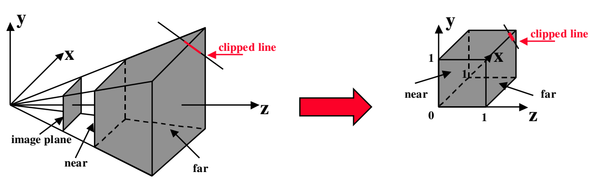
far: (+ or - 30A/B, + or - 30/B, -30, 1)

•We can carry out the trans and rots and prove this.

•LLF is - signs in near eqn; TRB is + signs in far.

Normalizing the Viewing Frustum

- Solution: transform frustum to a cube before clipping



- Converts perspective frustum to *orthographic* frustum
- This is a homogeneous transformation!

The Normalized Frustum

- Foley uses -1<=x<=1, -1<=y<=1, 0<=z<=1
- OpenGL uses -1<=x<=1, -1<=y<=1, -1<=z<=1
  - (try applying the transformation in glFrustum to the boundary points of the clipping volume and see)
- But *who cares!* They only differ by a homogeneous transformation; we can clip against either version.
  - Or, we can translate normalizing transformations by applying the appropriate trans.
- Must clip in homogeneous coordinates:

  w>0: -w<=x<=w, -w<=y<=w, -w<=z<=w (OpenGL)

  w<0: -w>=x>=w, -w>=y>=w, -w>=z>=w (ditto)

Normalize the Frustum 2

- **We just apply the perspective transformation.**

Lower left front

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} B/A & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & -2 & -30 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} -10A/B \\ -10/B \\ -10 \\ 1 \end{pmatrix} = \begin{pmatrix} -10 \\ -10 \\ -10 \\ 10 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$$

Top right back

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} B/A & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & -2 & -30 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 30A/B \\ 30/B \\ -30 \\ 1 \end{pmatrix} = \begin{pmatrix} 30 \\ 30 \\ 30 \\ 30 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Voila!