Modeling and Viewing

- Modeling
  - Use modeling (local) coordinates and geometric transformations to build hierarchically more complex objects and scenes. The final scene is in world frame
- Viewing
  - Model the camera: position , orientation, camera (view) reference frame, projection
  - Viewing transformations: from world to camera coordinates
  - Clipping/hidden surface removal: clip out from consideration parts outside of view volume
  - Projection transformations and hidden surface removal: from 3D viewing coord. to 2D projection coord. and normalized device coordinates
  - Viewport transformations: from normalized device coordinates to screen (device) coordinates

Viewing and Projection

- Transforming
  - Modeling transformations (affine), 3D to 3D
  - Viewing transformations (affine), 3D to 3D
    - Linear in homogeneous coordinates
    - Images of parallel lines stay parallel
    - Transformation matrix in homogeneous coordinates has last row, 0 0 0 1
  - Projection transformations (not affine), 3D to 2D
    - Linear in homogeneous coordinates
    - Images of parallel lines may intersect at infinity
    - Transformation matrix most general
  - Viewport transformations (affine), maps viewing window to viewport, 2D to 2D

Viewing Terminology

- Viewing volume: the region in 3D that can contain objects that are visible by the camera
- Projection: math transformations that maps from 3D to 2D (or 4D to 3D, in homogeneous)
- Projection plane: the plain containing the 2D image
- Viewing window: the rectangle in the image plane that will be mapped to the screen eventually
- Viewport: 2D rectangle within the display window on the screen that shows the viewing window
- Clipping: cutting off from consideration parts outside the view volume (done easier if the view volume is mapped to a canonical view volume which is a cube)

Graphics functions

- Graphics systems support viewing by
  - Providing a viewing model whose parameters specify the camera
  - Providing functions for viewing, projection and viewport
  - Implementing viewing and projection transformations as matrix multiplications in homogeneous coordinates

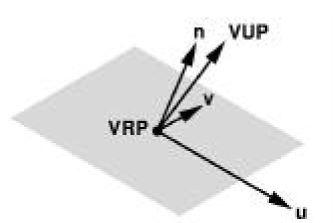
Viewing APIs

- The the position of the eye or camera is called the view reference point (VRP)
- A unit view plane normal (VPN), is in the viewing direction, it is perpendicular to the image plane. In open GL VPN is in direction opposite to the one in which camera is looking
- Another vector called the view-up vector is a vector specifying which is the approximate “up” direction for the camera

Camera Model: viewing

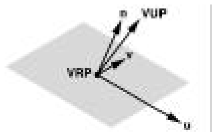
VRP: Camera(eye) position,
(u,v,n): camera frame.

Viewing: specify VRP,n,VUP.



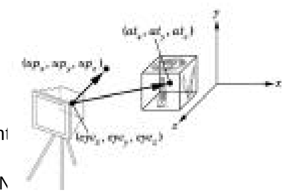
Viewing Coordinate System: (u,v,n)

- right handed
- **v**, the y-axis of the view frame, is the perpendicular projection of VUP on the projection plane
- **n** is the z-axis of the view frame
- **u** = **v** × **n**



Setting up the camera

- Construct a scene and then look at it from a point of view, **eye:**
- **eye,** the eyepoint, is the VRP specified in the world coordinates
- Camera is pointed at a point **at**, the at point
- These points determine VPN
- **vpn** = **eye** – **at**



Two Points of View

- Hold camera frame fixed, move objects in front of the camera

```
glLoadIdentity();
glTranslatef(0,0,-d);
```

- Keep objects stationary and move the camera away from the objects

```
glLoadIdentity();
glLookAt(0,0,d,0,0,0,0,1,0);
```


gluLookAt Utility Routine in OpenGL

- Defines a viewing transformation matrix M, and M postmultiplies CTM, i.e. CTM=CTM*M
- Eye point: *eyex*, *eyey*, and *eyez*.
- At point: *atx*, *aty*,and *atz.*
- VUP: *upx*, *upy*, and *upz*

gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz);

Viewing Transformations

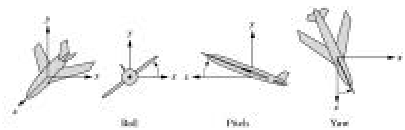
- Given 3D model of a scene/object in 4D homogeneous coordinates P, with respect to the world coordinate system
- Given camera coordinate system (position, VRP, and camera frame (u,v,n))
- Viewing transformation M converts coordinates of objects from world to camera coordinates
- How?

Viewing Transformations

- Let W=(O,ex,ey,ez) be the world coord. system
- Let V=(VRP,u,v,n) be the camera coord. System
- Let M be the change of frame matrix, mapping V to W, M = R.T(-VRP), where
  - T is a translation mapping VRP to O
  - R is a rotation aligning (u,v,n) with (ex,ey,ez), in 3D affine coordinates it represented by a matrix with rows u',v',n'
- Then for a point P with modeling coordinates **w**=(xw,yw,zw,1)' the viewing coordinates are **v** =(vx,vy,vz,1)', where

$$\mathbf{w} = (\mathbf{M}')^{-1}\mathbf{v}, \quad \mathbf{v} = \mathbf{M}'\mathbf{w}$$

Custom Utility Routine



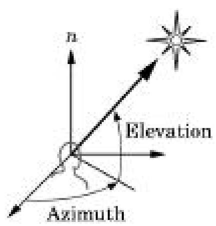
- You might need to define your own transformation routine
- Flight simulator: Display the world from the pilot's point of view
- Pilot see the world in terms of *roll*, *pitch*, and *heading*

Custom Utility Routine

The following routine could serve as the viewing transformation:

```
void pilotView{GLdouble planex, GLdouble planey,
               GLdouble planez, Gldouble roll,
               GLdouble pitch, GLdouble heading)
{
     glRotated(roll, 0.0, 0.0, 1.0);
    glRotated(pitch, 0.0, 1.0, 0.0);
    glRotated(heading, 1.0, 0.0, 0.0);
    glTranslated(-planex, -planey, -planez);
}
```


Custom Utility Routine



- Orbiting the camera around an object that's centered at the origin
- Use polar coordinates.
- Let the *distance* variable define the radius of the orbit
- The *azimuth* is the angle of rotation of the camera about the object in the *x-y* plane
- *elevation* is the angle of rotation of the camera in the *y-z* plane