3D viewing under the hood

- Topics of Interest:
  - Viewing transformation
  - Projection transformation

Viewing Transformation

- Transform the object from world to eye space
  - Construct eye coordinate frame
  - Construct matrix to perform coordinate transformation
  - Flexible Camera Control
**Viewing Transformation**

- Recall OpenGL way to set camera:
  - `gluLookAt (Ex, Ey, Ez, cx, cy, cz, Up_x, Up_y, Up_z)`
  - The view up vector is usually (0,1,0)
  - Remember to set the OpenGL matrix mode to GL_MODELVIEW first
- Modelview matrix:
  - combination of modeling matrix $M$ and Camera transforms $V$
  - `gluLookAt` fills $V$ part of modelview matrix
- What does `gluLookAt` do with parameters (eye, COI, up vector) you provide?

**Eye Coordinate Frame**

- Known: eye position, center of interest, view-up vector
- To find out: new origin and three basis vectors

Assumption: direction of view is orthogonal to view plane (plane that objects will be projected onto)

Eye Coordinate Frame

- Origin: eye position (that was easy)
- Three basis vectors:
  - one is the normal vector ($n$) of the viewing plane,
  - other two ($u$ and $v$) span the viewing plane

Center of interest (COI)

- we can get $u$ first -
  - $u$ is a vector that is perp to the plane spanned by $N$ and view up vector ($V_{up}$)

\[ U = V_{up} \times N \]
\[ u = U / |U| \]
Eye Coordinate Frame

• How about \( v \)?

Knowing \( n \) and \( u \), getting \( v \) is easy

\[
\mathbf{v} = \mathbf{n} \times \mathbf{u}
\]

\( v \) is already normalized

Put it all together

Eye space origin:

\( \text{Eye.x, Eye.y, Eye.z} \)

Basis vectors:

\[
\begin{align*}
\mathbf{n} &= (\text{eye} - \text{COI}) / |\text{eye} - \text{COI}| \\
\mathbf{u} &= (\mathbf{V}_{\text{up}} \times \mathbf{n}) / |\mathbf{V}_{\text{up}} \times \mathbf{n}| \\
\mathbf{v} &= \mathbf{n} \times \mathbf{u}
\end{align*}
\]

World to Eye Transformation

• Transformation matrix \((M_{w2e})\):

\[
P' = M_{w2e} \times P
\]

1. Come up with the transformation sequence to move eye coordinate frame to the world
2. And then apply this sequence to the point \( P \) in a reverse order

rotation:

\[
\begin{bmatrix}
\mathbf{v} & \mathbf{w} & \mathbf{z} & 0 \\
\mathbf{x} & \mathbf{y} & \mathbf{z} & 0 \\
x & y & z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

translation:

\[
\begin{bmatrix}
1 & 0 & 0 & -\mathbf{ex} \\
0 & 1 & 0 & -\mathbf{ey} \\
0 & 0 & 1 & -\mathbf{ez} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
World to Eye Transformation

- Transformation order: apply the transformation to the object in a reverse order - translation first, and then rotate

\[
M_{w2e} = \begin{pmatrix}
ux & uy & ux & 0 \\
vx & vy & vz & 0 \\
xn & ny & nz & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 0 & -ex \\
0 & 1 & 0 & -ey \\
0 & 0 & 1 & -ez \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

Head tilt: Rotate your head by \( \delta \)

- Just rotate the object about the eye space z axis - \( \delta \)

\[
M_{w2e} = \begin{pmatrix}
\cos(-\delta) & -\sin(-\delta) & 0 & 0 \\
\sin(-\delta) & \cos(-\delta) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

Why - \( \delta \)?
When you rotate your head by \( \delta \), it is like rotate the object by -\( \delta \).