3D Viewing?

- Specify a view volume
- Objects inside view volume drawn to viewport (screen)
- Objects outside view volume clipped (not drawn)!
Different View Volume Shapes

- Different view volume shape => different look
- **Foreshortening?** Near objects bigger
  - Perspective projection has **foreshortening**
  - Orthogonal projection: no foreshortening
The World Frame

- Object positions initially defined in **world frame**
- **World Frame origin** at (0,0,0)
- Objects positioned, oriented (translate, scale, rotate transformations) applied to objects in **world frame**
Camera Frame

- More natural to describe object positions **relative to camera (eye)**
- Why?
  - Our view of the world
  - First person shooter games
**Camera Frame**

- **Viewing**: After user chooses camera (eye) position, represent objects in **camera frame** (origin at eye position)
- **Viewing transformation**: Converts object \((x, y, z)\) positions in world frame to positions in camera frame

Objects initially specified in world frame

More natural to view Objects in camera frame

World frame (Origin at 0,0,0)

Camera frame (Origin at camera)
Default OpenGL Camera

- Initially Camera at origin: object and camera frames same
- Points in negative z direction
- Default view volume is cube with sides of length 2
Moving Camera Frame

default frames

Translate objects -5 away from camera
Same relative distance after
Same result/look

Translate camera +5 away from objects
Moving the Camera

- We can move camera using sequence of rotations and translations
- Example: side view
  - Rotate the camera
  - Move it away from origin
  - Model-view matrix $C = TR$

```c
// Using mat.h

mat4 t = Translate (0.0, 0.0, -d);
mat4 ry = RotateY(90.0);
mat4 m = t*ry;
```
Moving the Camera Frame

- Object distances relative to camera determined by the model-view matrix
  - Transforms (scale, translate, rotate) go into modelview matrix
  - Camera transforms also go in modelview matrix (CTM)
The LookAt Function

- Previously, command `gluLookAt` to position camera
- `gluLookAt` deprecated!!
- Homegrown mat4 method LookAt() in mat.h
  - Sets camera position, transforms object distances to camera frame

```cpp
void display() {
    ........

    mat4 mv = LookAt(vec4 eye, vec4 at, vec4 up);
    ........
}
```

Builds 4x4 matrix for positioning, orienting Camera and puts it into variable `mv`
The LookAt Function

LookAt(eye, at, up)

Programmer defines:
• eye position
• LookAt point \((at)\) and
• Up vector \((Up)\) direction usually \((0,1,0)\)

But Why do we set Up direction?
Nate Robbins LookAt Demo

```c
GLfloat pos[4] = { 1.50, 1.00, 1.00, 0.00 };  // <- eye
    gluLookAt( 0.00, 0.00, 2.00,         // <- center
              0.00, 0.00, 0.00,
              0.00, 1.00, 0.00 ); // <- up

    glLightfv(GL_LIGHT0, GL_POSITION, pos);
```

Click on the arguments and move the mouse to modify values.

Click on the arguments and move the mouse to modify values.
What does LookAt do?

- Programmer defines eye, lookAt and Up
- **LookAt method:**
  - Forms new axes \((u, v, n)\) at camera
  - Transform objects from world to eye camera frame
Camera with Arbitrary Orientation and Position

- Define new axes \((u, v, n)\) at eye
  - \(v\) points vertically upward,
  - \(n\) away from the view volume,
  - \(u\) at right angles to both \(n\) and \(v\).
  - The camera looks toward \(-n\).
  - All vectors are normalized.
LookAt: Effect of Changing Eye Position or LookAt Point

- Programmer sets `LookAt(eye, at, up)`
- If `eye`, `lookAt` point changes => `u,v,n` changes
Viewing Transformation Steps

1. Form camera \((u,v,n)\) frame
2. Transform objects from world frame (Composes matrix to transform coordinates)

- Next, let’s form camera \((u,v,n)\) frame
Constructing U,V,N Camera Frame

- **Lookat arguments**: `LookAt(eye, at, up)`
- **Known**: eye position, LookAt Point, up vector
- **Derive**: new origin and three basis (u,v,n) vectors
Eye Coordinate Frame

- **New Origin**: *eye position* (that was easy)
- 3 basis vectors:
  - one is the normal vector ($\mathbf{n}$) of the viewing plane,
  - other two ($\mathbf{u}$ and $\mathbf{v}$) span the viewing plane

$\mathbf{n}$ is pointing away from the world because we use left hand coordinate system

$\mathbf{N} = \text{eye} - \text{Lookat Point}$

$n = \frac{\mathbf{N}}{|\mathbf{N}|}$

(u,v,n should all be orthogonal)

Remember $\mathbf{u}, \mathbf{v}, \mathbf{n}$ should be all unit vectors
So... Normalize vectors!!!!!
Eye Coordinate Frame

- How about \( u \) and \( v \)?

- Derive \( u \) first -
  - \( u \) is a vector that is perpendicular 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?

To derive v from n and u

\[ v = n \times u \]

v is already normalized
Eye Coordinate Frame

- Put it all together

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

Basis vectors:

\[
\begin{align*}
\mathbf{n} &= \frac{\text{eye} - \text{Lookat}}{||\text{eye} - \text{Lookat}||} \\
\mathbf{u} &= \frac{\text{V_up} \times \mathbf{n}}{||\text{V_up} \times \mathbf{n}||} \\
\mathbf{v} &= \mathbf{n} \times \mathbf{u}
\end{align*}
\]
Step 2: World to Eye Transformation

- Next, use $u$, $v$, $n$ to compose LookAt matrix
- Transformation matrix ($M_{w2e}$)?
  - Matrix that transforms a point $P$ in world frame to $P'$ in eye frame

$$P' = M_{w2e} \times P$$

1. Come up with transformation sequence that lines up eye frame with world frame
2. Apply this transform sequence to point $P$ in reverse order
World to Eye Transformation

1. Rotate eye frame to “align” it with world frame
2. Translate \((-ex, -ey, -ez)\) to align origin with eye

Rotation: 
\[
\begin{bmatrix}
ux & uy & uz & 0 \\
vx & vy & vz & 0 \\
x & ny & nz & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

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

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

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

Note: \( e.u = ex.ux + ey.uy + ez.uz \)

\( e.v = ex.vx + ey.vy + ez.vz \)

\( e.n = ex.nx + ey.ny + ez.nz \)
lookAt Implementation (from mat.h)

Eye space **origin**: \((\text{Eye}.x, \text{Eye}.y, \text{Eye}.z)\)

Basis vectors:

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

```cpp
mat4 LookAt( const vec4& eye, const vec4& at, const vec4& up )
{
    vec4 n = normalize(eye - at);
    vec4 u = normalize(cross(up,n));
    vec4 v = normalize(cross(n,u));
    vec4 t = vec4(0.0, 0.0, 0.0, 1.0);
    mat4 c = mat4(u, v, n, t);
    return c * Translate( -eye );
}
```
References

- Interactive Computer Graphics, Angel and Shreiner, Chapter 4