3D Viewing?

- Objects **inside** view volume show up on screen
- Objects outside view volume **clipped**!
Different View Volume Shapes

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

- Objects/scene initially defined in **world frame**
- Objects positioned, transformations (translate, scale, rotate) applied to objects in **world frame**
Camera Frame

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

- **Viewing**: After user sets camera (eye) position, represent objects in **camera frame** (origin at eye position)

- **Viewing transformation**: Changes object positions from world frame to positions in camera frame using **model-view matrix**
Default OpenGL Camera

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

![Diagram of default view volume with objects and projection plane]
Moving Camera Frame

default frames

Translate objects +5 away from camera

Translate camera -5 away from objects

Same relative distance after
Same result/look
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`
  - Can concatenate with modeling transformations

```cpp
void display( ){
    ........
    mat4 mv = LookAt(vec4 eye, vec4 at, vec4 up);
    ........
}
```
LookAt

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

Nate Robbins LookAt Demo

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

- Programmer defines eye, lookAt and Up
- **LookAt method:**
  - Form 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 $\text{LookAt}(\text{eye, at, up})$
- If eye, lookAt point changes $\Rightarrow u,v,n$ changes
Viewing Transformation Steps

1. Form camera (u,v,n) frame
2. Transform objects from world frame (Composes matrix for coordinate transformation)

- 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

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

\[
\mathbf{N} = \text{eye} - \text{Lookat Point}
\]

\[
\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|}
\]

Remember \( \mathbf{u}, \mathbf{v}, \mathbf{n} \) should be all unit vectors.
Eye Coordinate Frame

- How about u and v?

- 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

$$v = n \times u$$

$v$ is already normalized
Eye Coordinate Frame

- Put it all together

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

Basis vectors:

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

- Next, use $u$, $v$, $n$ to compose LookAt matrix
- Transformation matrix ($M_{w2e}$) ?

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

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:

<table>
<thead>
<tr>
<th></th>
<th>ux</th>
<th>uy</th>
<th>uz</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx</td>
<td>vy</td>
<td>vz</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>nx</td>
<td>ny</td>
<td>nz</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Translation:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>-ex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-ey</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-ez</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
World to Eye Transformation

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

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

Rotation Translation

Note: \( e.u = ex.ux + ey.uy + ez.uz \)
lookAt Implementation (from mat.h)

Eye space origin: (Eye.x, Eye.y, Eye.z)

Basis vectors:

\[
\begin{align*}
\mathbf{\text{n}} &= (\text{eye} - \text{Lookat}) / | \text{eye} - \text{Lookat}| \\
\mathbf{\text{u}} &= (\text{V_up} \times \mathbf{n}) / | \text{V_up} \times \mathbf{n} | \\
\mathbf{\text{v}} &= \mathbf{n} \times \mathbf{u}
\end{align*}
\]

\[
\begin{bmatrix}
\text{ux} & \text{uy} & \text{uz} & -\text{e} \cdot \mathbf{u} \\
\text{vx} & \text{vy} & \text{vz} & -\text{e} \cdot \mathbf{v} \\
\text{nx} & \text{ny} & \text{nz} & -\text{e} \cdot \mathbf{n} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

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