LEC18: Computer Viewing-part2

Ku-Jin Kim

School of Computer Science & Engineering

Kyungpook National University

Contents

View Matrix Construction

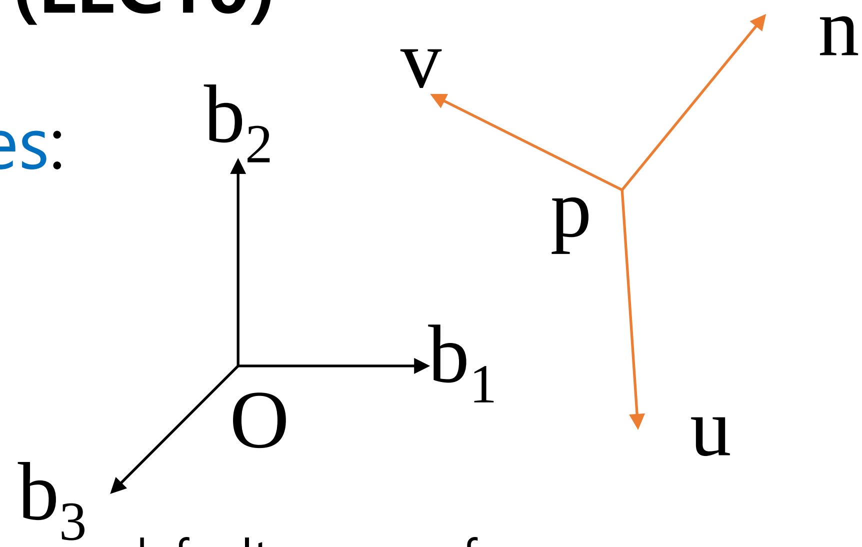
- For computing view matrix we can approach
  - Step1) Compute the transformation matrix (C) for the camera to be with the expected orientation (M_1) and location (M_2)
    - $C = M_2 M_1$
  - Step2) Compute the inverse matrix of C to apply it to objects
    - $C^{-1} = M_1^{-1} M_2^{-1}$

Change of Frames (LEC10)

Consider two frames:

$(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{O})$

$(\mathbf{u}, \mathbf{v}, \mathbf{n}, \mathbf{p})$



- We can represent $(\mathbf{u}, \mathbf{v}, \mathbf{n}, \mathbf{p})$ in terms of $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{O})$
  - $\mathbf{u} = [u_x, u_y, u_z, 0]^T$
  - $\mathbf{v} = [v_x, v_y, v_z, 0]^T$
  - $\mathbf{n} = [n_x, n_y, n_z, 0]^T$
  - $\mathbf{p} = [p_x, p_y, p_z, 1]^T$

  Left hand-rule

  - $\mathbf{b}_1 = [1, 0, 0, 0]^T$
  - $\mathbf{b}_2 = [0, 1, 0, 0]^T$
  - $\mathbf{b}_3 = [0, 0, 1, 0]^T$
  - $\mathbf{O} = [0, 0, 0, 1]^T$

  Right hand-rule

Matrix for constructing transformed camera frame (1)

- Constructing matrix M_1
  - $\mathbf{b}_1 = [1, 0, 0, 0]^T$ to be **u**
  - $\mathbf{b}_2 = [0, 1, 0, 0]^T$ to be **v**
  - $\mathbf{b}_3 = [0, 0, 1, 0]^T$ to be **n**

$$M_1 = \begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Constructing matrix M_2
  - Translate the origin $\mathbf{O}=[0, 0, 0, 1]^T$ to point $\mathbf{p} = [p_x, p_y, p_z, 1]^T$

$$M_2 = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix for constructing transformed camera frame (2)

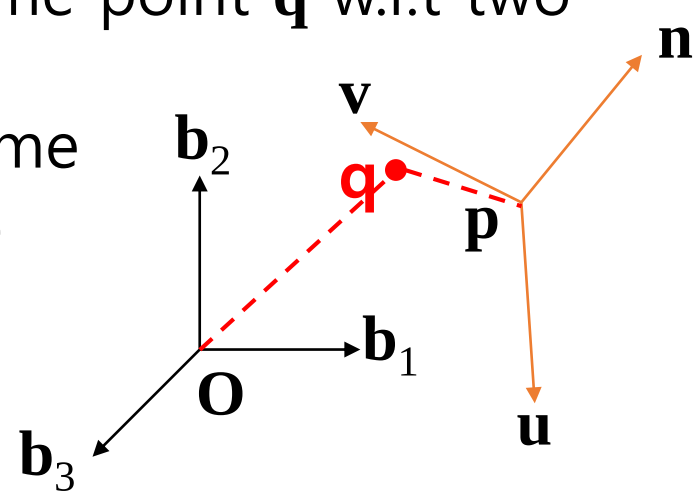
- $\mathbf{b}_1$-$\mathbf{b}_2$-$\mathbf{b}_3$-$\mathbf{O}$ frame is transformed to **u**-**v**-**n**-**p** frame by applying M_2M_1

$$\underset{\textbf{u-v-n-p frame}}{\begin{bmatrix} u_x & v_x & n_x & p_x \\ u_y & v_y & n_y & p_y \\ u_z & v_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}} = M_2 M_1 \underset{\mathbf{b}_1\text{-}\mathbf{b}_2\text{-}\mathbf{b}_3\text{-}\mathbf{O}\text{ frame}}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

$$\mathrm{M}_2 = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathrm{M}_1 = \begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How can we construct view matrix? (1)

- View matrix: represents objects in camera frame, where objects are defined in the world frame
- Consider two representations of the same point $\mathbf{q}$ w.r.t two different bases
  - $\mathbf{a} = [\alpha_1 \ \ \alpha_2 \ \ \alpha_3 \ \ 1]^T$ w.r.t. $\mathbf{b}_1$-$\mathbf{b}_2$-$\mathbf{b}_3$-$\mathbf{O}$ frame
  - $\mathbf{b} = [\beta_1 \ \ \beta_2 \ \ \beta_3 \ \ 1]^T$ w.r.t. $\mathbf{u}$-$\mathbf{v}$-$\mathbf{n}$-$\mathbf{p}$ frame



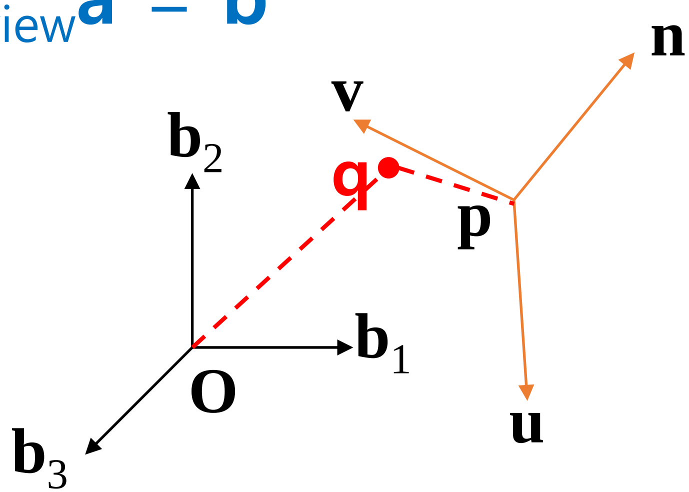
$$\beta_1\mathbf{u} + \beta_2\mathbf{v} + \beta_3\mathbf{n} + \mathbf{p} = \alpha_1\mathbf{b}_1 + \alpha_2\mathbf{b}_2 + \alpha_3\mathbf{b}_3 + \mathbf{O}$$

$$\begin{bmatrix} u_x & v_x & n_x & p_x \\ u_y & v_y & n_y & p_y \\ u_z & v_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix} = M_2 M_1 \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 1 \end{bmatrix}$$

$$\mathbf{a} = \mathrm{M}_2\mathrm{M}_1\mathbf{b}$$

How can we construct view matrix? (2)

- $\mathbf{a} = M_2 M_1 \mathbf{b}$
- Given point **a** w.r.t. $\mathbf{b}_1$-$\mathbf{b}_2$-$\mathbf{b}_3$-**O** (world frame), we need its representation w.r.t. **u**-**v**-**n**-**p** frame
- When **a** is transformed to **b** by M_{view},
  - M_{view} is a viewing matrix !!!
  - $M_{view}\mathbf{a} = \mathbf{b}$



Computing M_{view} (1)

- $\mathbf{a} = M_2 M_1 \mathbf{b}$

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 1 \end{bmatrix} \qquad (1)$$

- $M_2^{-1}\mathbf{a} = M_1\mathbf{b}$

$$\begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & -px \\ 0 & 1 & 0 & -py \\ 0 & 0 & 1 & -pz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 1 \end{bmatrix} \qquad (2)$$

Computing M_{view} (2)

$$\begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix} = \begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 1 \end{bmatrix} \quad (3)$$

Since $||\mathbf{u}|| = ||\mathbf{v}|| = ||\mathbf{n}|| = 1$

$\rightarrow u_x u_x + u_y u_y + u_z u_z = v_x v_x + v_y v_y + v_z v_z = n_x n_x + n_y n_y + n_z n_z = 1$

Since **u**, **v**, **n** are perpendicular each other

$\rightarrow \mathbf{u} \cdot \mathbf{v} = u_x v_x + u_y v_y + u_z v_z = 0$, Similarly, $\mathbf{u} \cdot \mathbf{n} = \mathbf{v} \cdot \mathbf{n} = 0$

$$M_1^T M_1 = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(4)

Computing M_{view} (3)

- We apply the inverse of M_1 to Equation (3)

$$\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 1 \end{bmatrix} \quad (5)$$

Computing M_{view} (4)

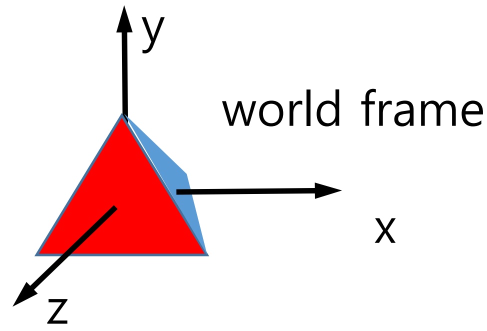
$$\mathrm{M}_{\mathrm{view}} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -px \\ 0 & 1 & 0 & -py \\ 0 & 0 & 1 & -pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

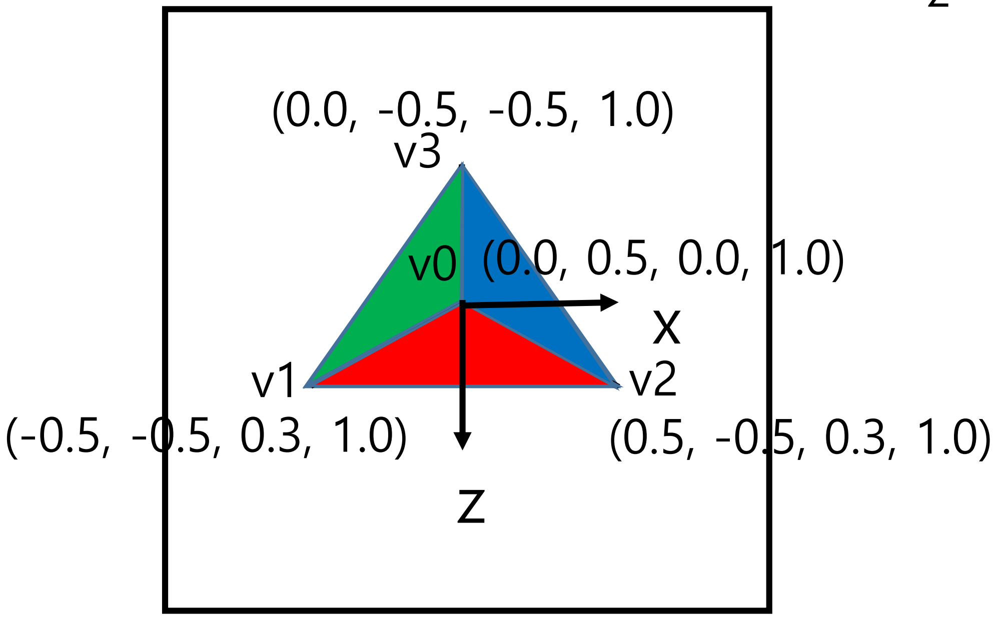
$$= \begin{bmatrix} u_x & u_y & u_z & -\mathbf{p}\cdot\mathbf{u} \\ v_x & v_y & v_z & -\mathbf{p}\cdot\mathbf{v} \\ n_x & n_y & n_z & -\mathbf{p}\cdot\mathbf{n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

model-view, projection matrices in GLSL

```
uniform mat4 mat_model;       // model matrix
uniform mat4 mat_view;        // view matrix
uniform mat4 mat_proj;        // projection matrix
in vec4 aPosition;
void main() {
    gl_Position = mat_proj * mat_view * mat_model * aPosition;
}
```

For a tetrahedron



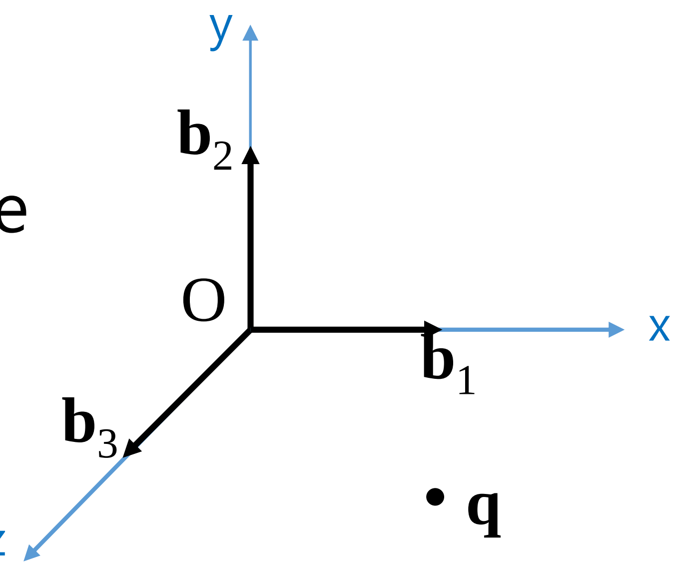


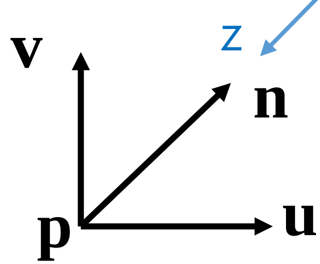
View Matrix Example (1)

- Ex) View matrix for camera frame
  - VRP: (0, 0, 0.5)
  - VPN: (0, 0, -1)
  - VUP: $\mathbf{v}_{\mathbf{up}}$ =(0, 1, 0)

  - $\mathbf{p}$ = (0, 0, 0.5)
  - normalized $\mathbf{n}$ = (0, 0, -1)
  - $\mathbf{v} = -(\mathbf{v}_{\mathrm{up}} \cdot \mathbf{n})\, \mathbf{n} + \mathbf{v}_{\mathrm{up}}$

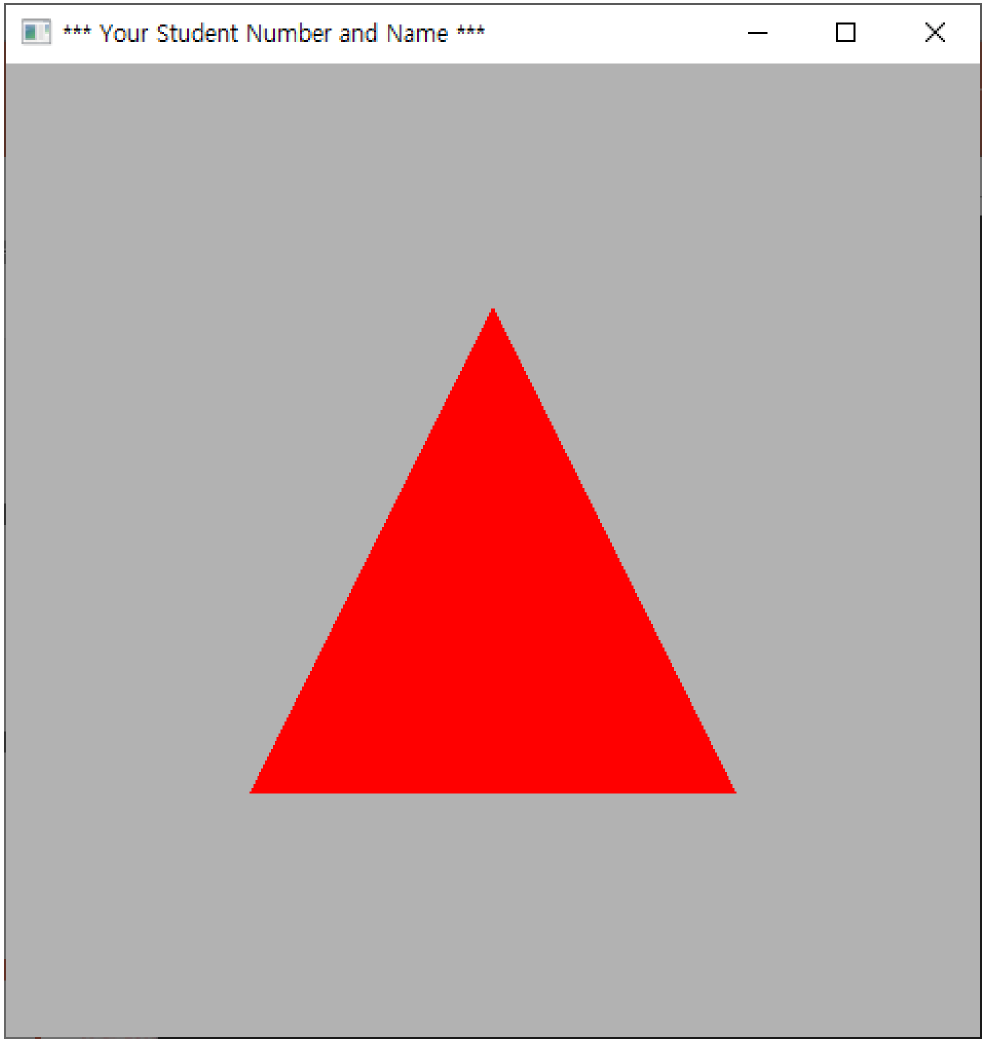
    normalized $\mathbf{v}$ = (0, 1, 0)
  - normalized $\mathbf{u}$ = (1, 0, 0)





$$\mathrm{M}_{\mathrm{view}} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{p}\cdot\mathbf{u} \\ v_x & v_y & v_z & -\mathbf{p}\cdot\mathbf{v} \\ n_x & n_y & n_z & -\mathbf{p}\cdot\mathbf{n} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

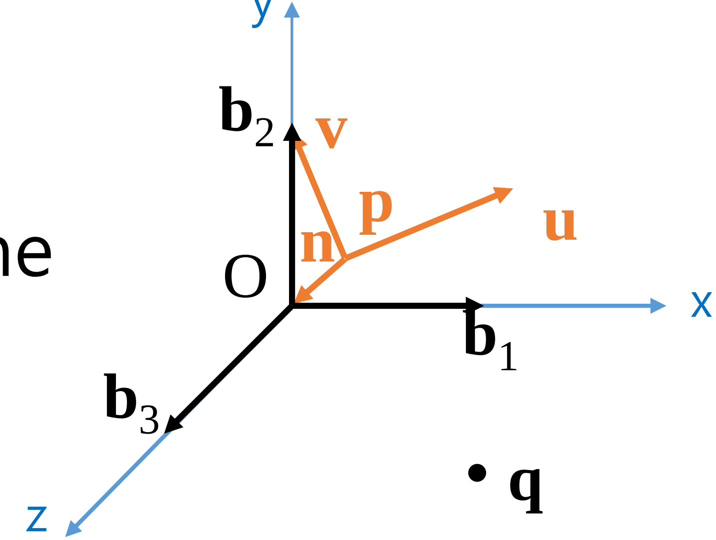
LEC18.1_simple_view_matrix.c



View Matrix Example (2)

- Ex) View matrix for camera frame
  - VRP: $\mathbf{p}$ = (0.1, 0.1, 0.1)
  - VPN: (-1, -1, -1)
  - VUP: $\mathbf{v}_{\text{up}}$ = (0, 1, 0)

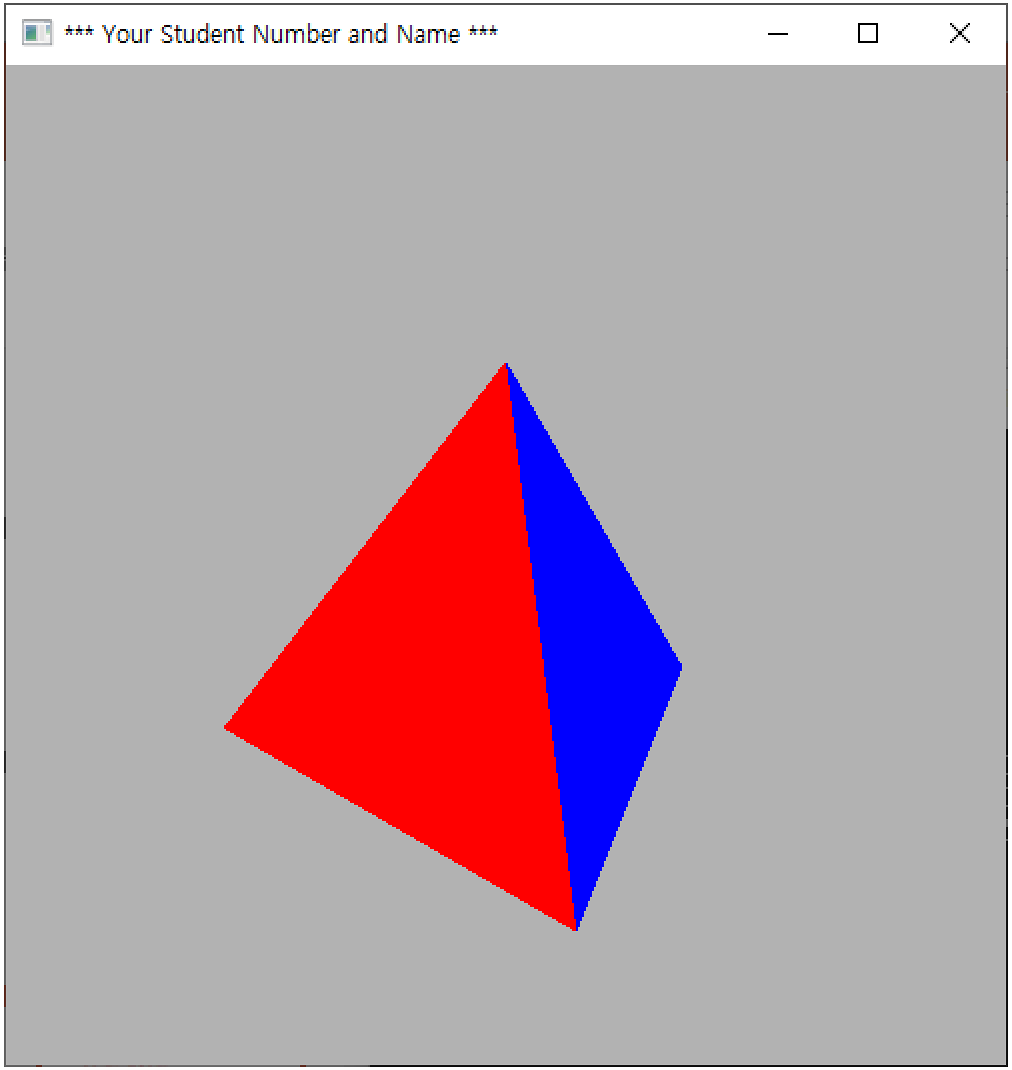
$$\mathbf{v} = -(\mathbf{v}_{\text{up}} \cdot \mathbf{n})\,\mathbf{n} + \mathbf{v}_{\text{up}}$$



$$\mathrm{M}_{\text{view}} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{p}\cdot\mathbf{u} \\ v_x & v_y & v_z & -\mathbf{p}\cdot\mathbf{v} \\ n_x & n_y & n_z & -\mathbf{p}\cdot\mathbf{n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.707107 & 0.000000 & -0.707107 & 0 \\ -0.408248 & 0.816497 & -0.408248 & 0 \\ -0.577350 & -0.577350 & -0.577350 & 0.173205 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

LEC18.1_simple_view_matrix.c

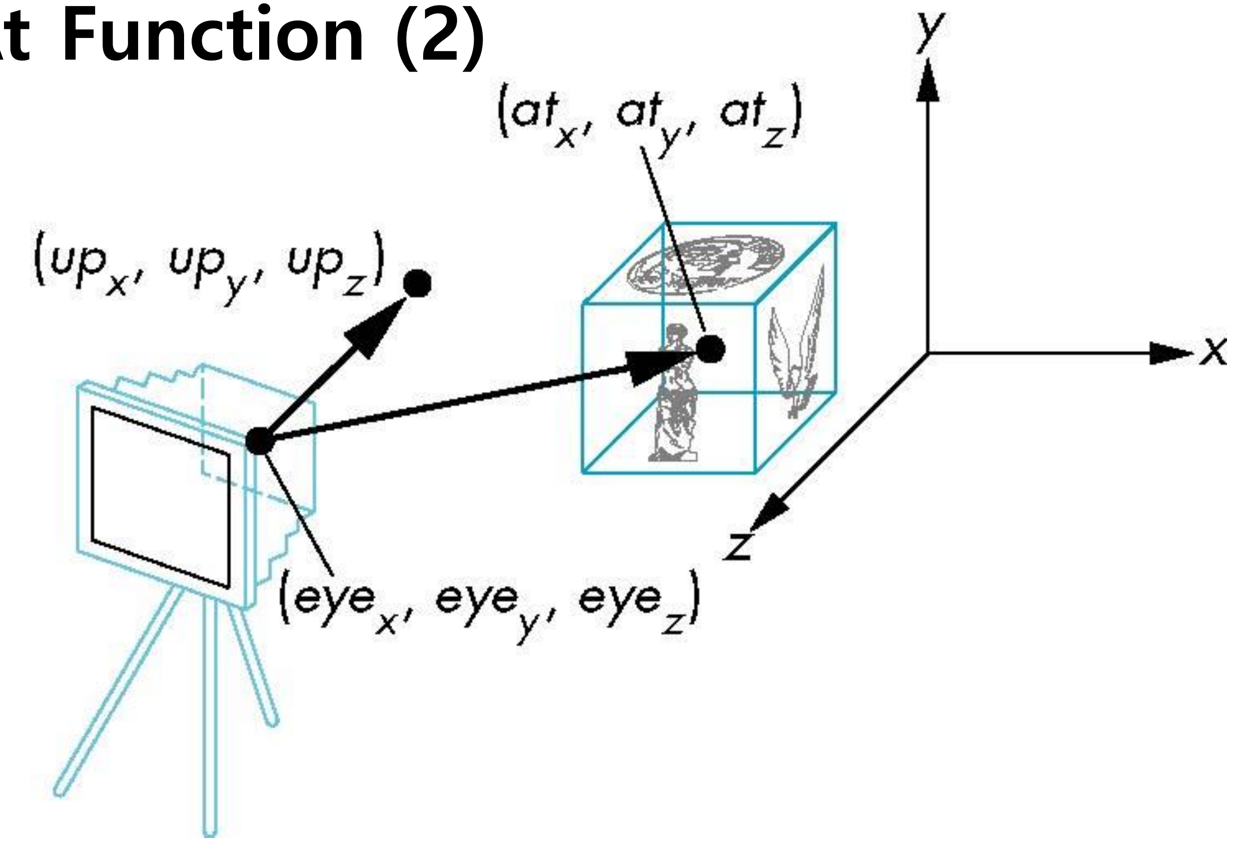


The LookAt Function (1)

- The GLU library contained the function gluLookAt to form the required modelview matrix through a simple interface
- But, gluLookAt is deprecated
- We can implement LookAt function in GLSL similar to gluLookAt function

```
mat4 mat_view = LookAt(vec4 eye, vec4 at, vec4 up);
```

The LookAt Function (2)



VRP : $\mathbf{p} = [p_x \ \ p_y \ \ p_z \ \ 1]^T$ = eye

VPN : $\mathbf{n} = [n_x \ \ n_y \ \ n_z \ \ 0]^T$ = (at – eye)/||at – eye||

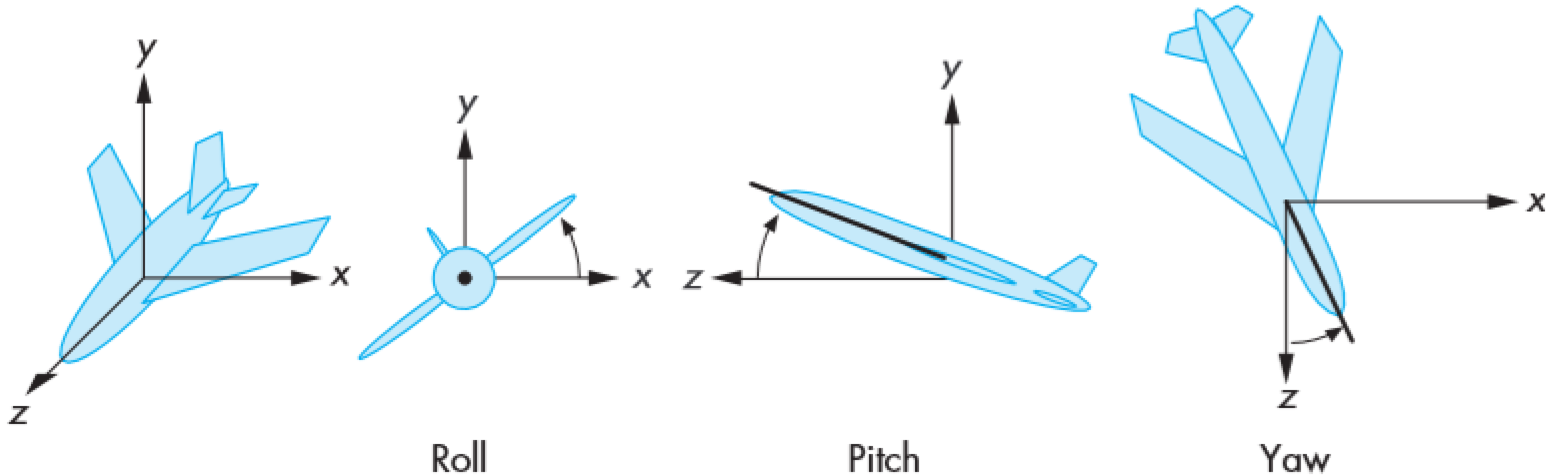
VUP : $\mathbf{v}_{up} = [v_{upx} \ v_{upy} \ v_{upz} \ 0]^T$ = up

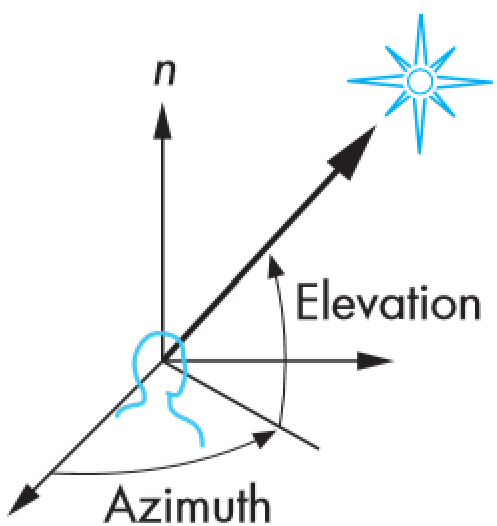
We can compute view matrix by using **u-v-n-p** frame

Other Viewing APIs (1)

- In OpenGL the LookAt function is only one possible API for positioning the camera
- Others include
  - View reference point, view plane normal, view up (PHIGS, GKS-3D)
  - roll, pitch, yaw
    - ex) flight simulation
  - Elevation, azimuth, twist

Other Viewing APIs (2)





HW#18 Fill the LookAt function (20points)

- Due date: This Friday 6:00pm
- Given the file ‘HW18.c’ in Lecture Note board in LMS, fill the function ‘LookAt’ to compute the view matrix by changing eye-at-up to u-v-n-p.
- With the given object and eye-at-up, you have to generate this image.
- Other parts except ‘LookAt’ function must not be modified.