LEC17: Computer Viewing-part1

Ku-Jin Kim

School of Computer Science & Engineering

Kyungpook National University

Contents

- OpenGL Viewing Functions
- Viewing APIs

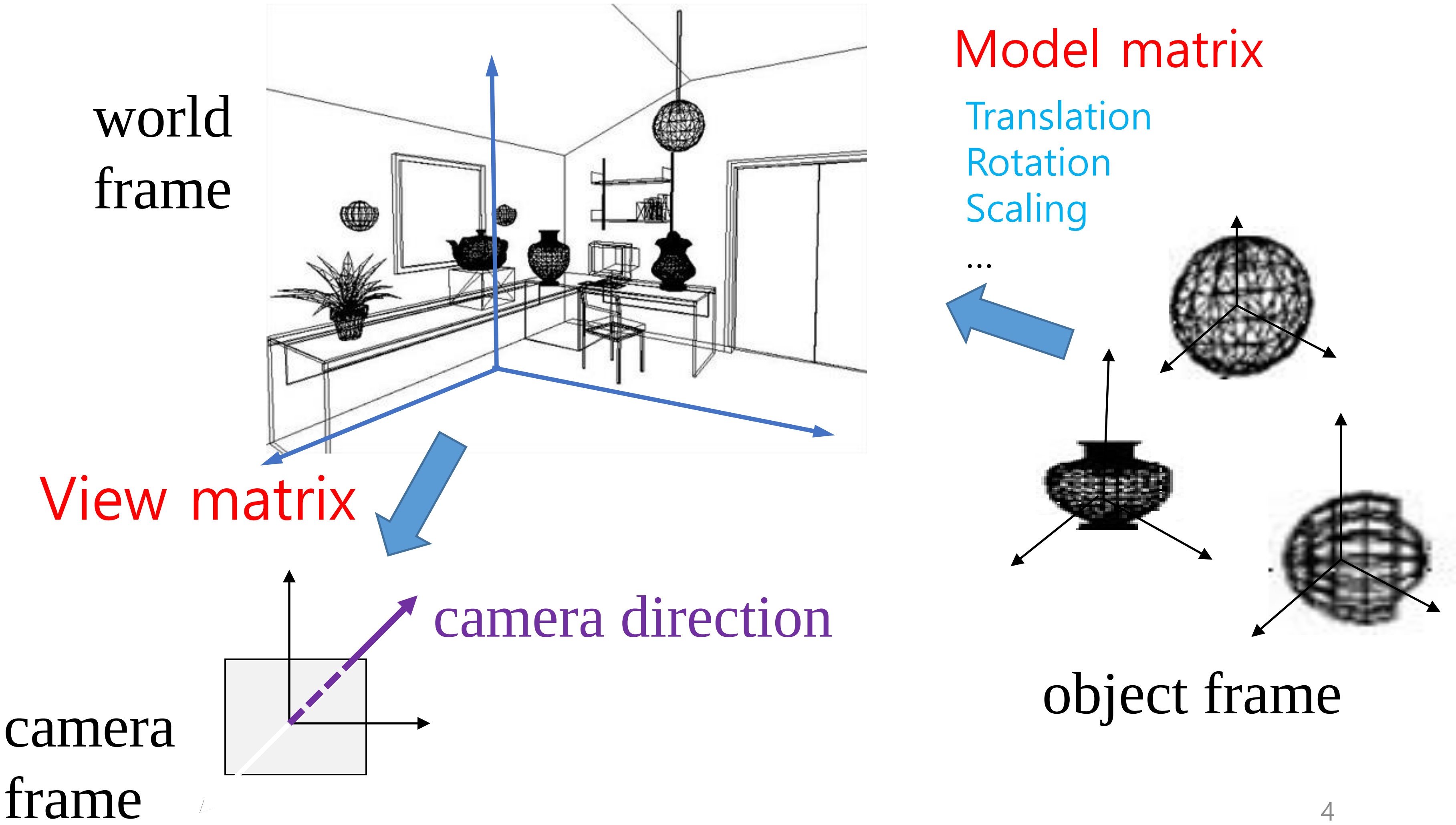
Computer Viewing

- Three steps of the viewing process
  - Positioning the camera
    - Setting the modeling & viewing matrices
    - Model-view matrix
  - Selecting a lens
    - Setting the projection matrix
    - Parallel (orthogonal) projection
    - Perspective projection
  - Clipping
    - Setting the view volume
- These steps are implemented in the pipeline

Model-View Matrix

object frame ⇒ world frame

world frame ⇒ camera frame

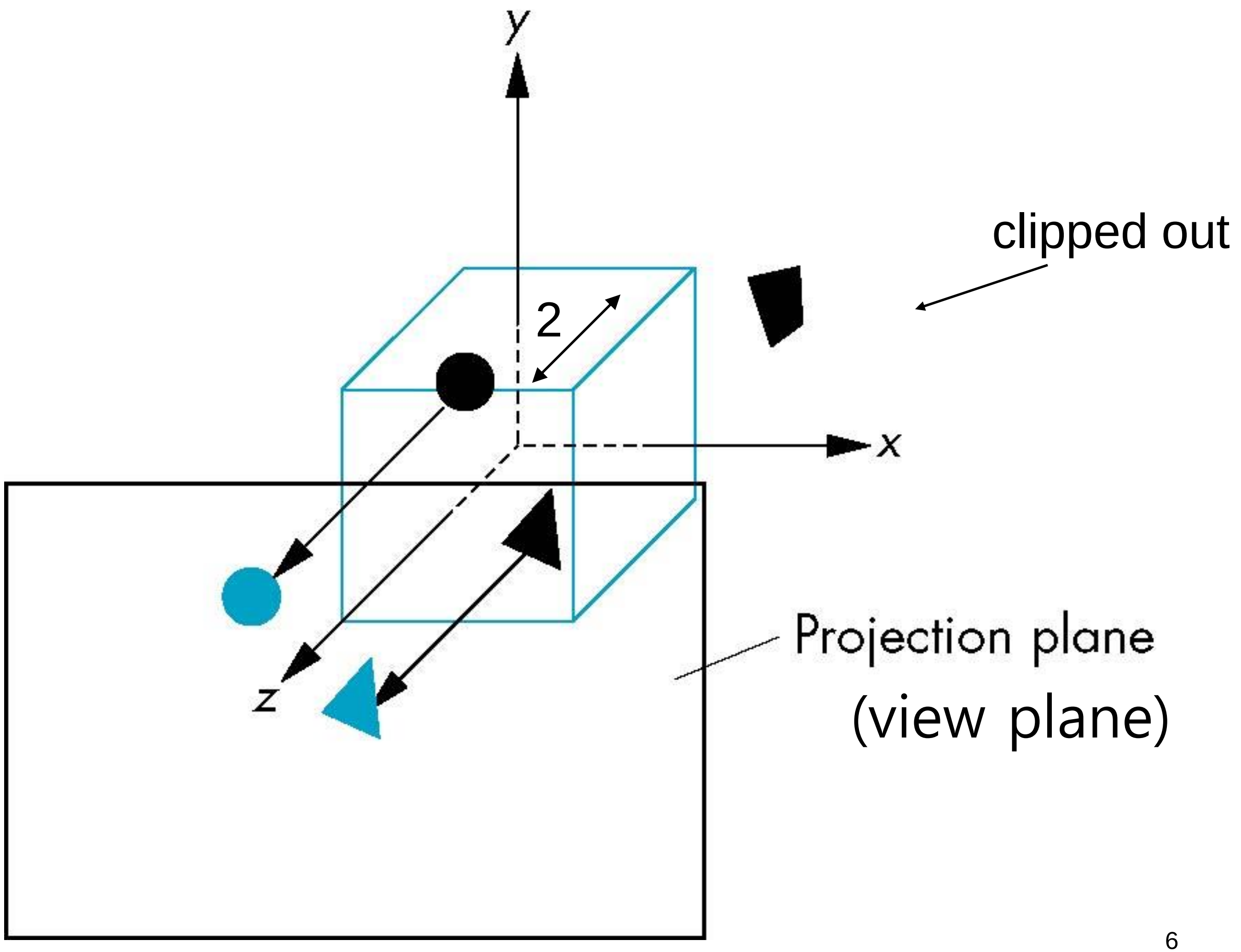


The Default OpenGL Camera

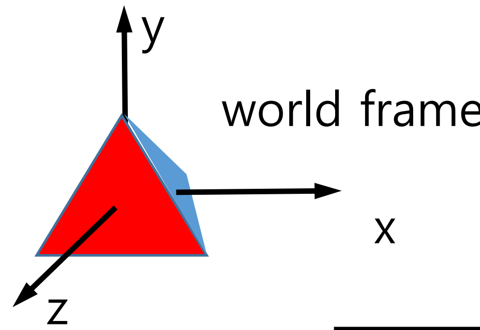
- In OpenGL, initially the object/world and camera frames are at the same origin & orientation
  - Default model matrix: identity matrix
  - Default view matrix: identity matrix
- OpenGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
  - $x = y = z = \pm 1$

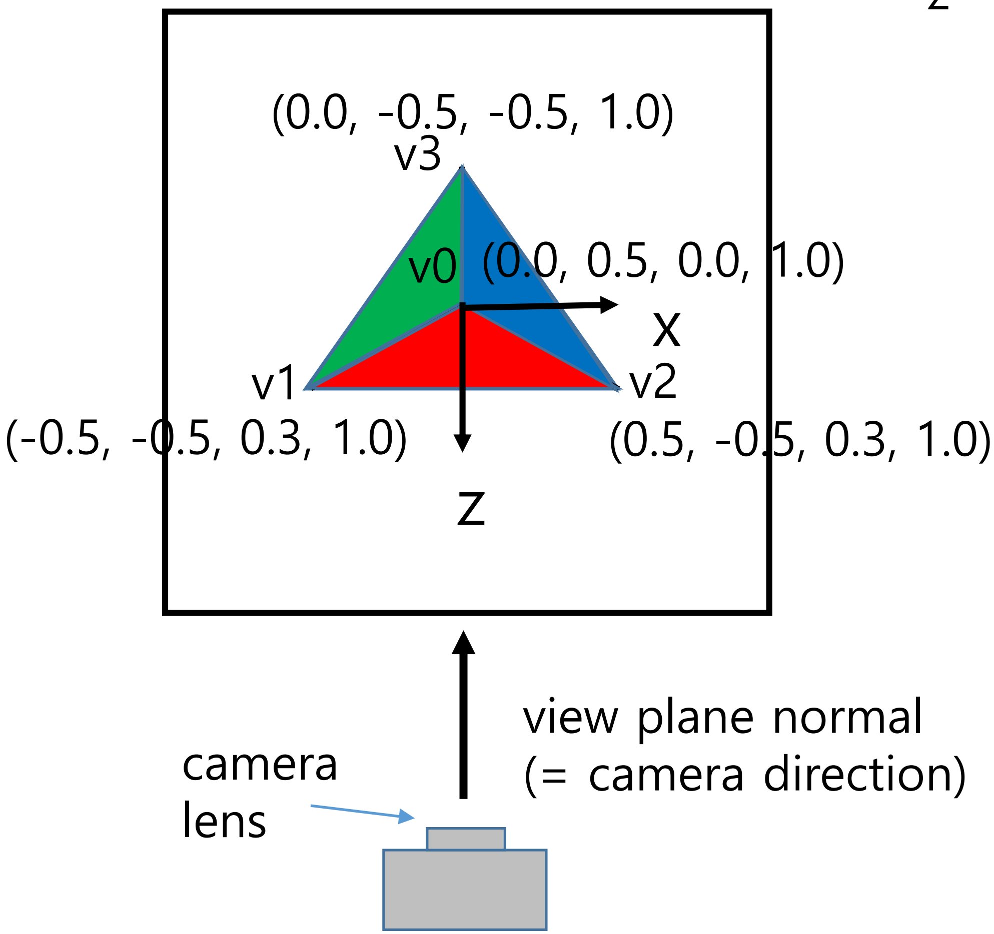
Default Projection

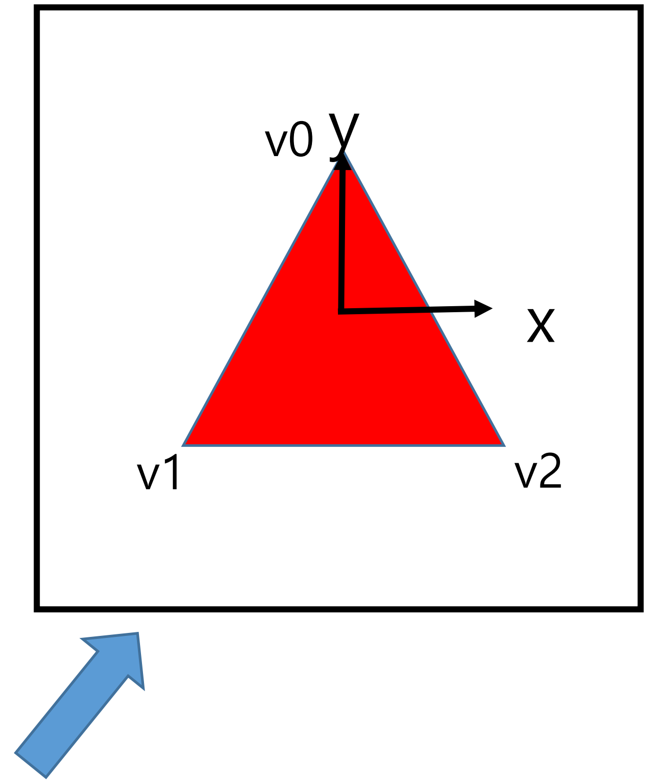
Default : orthogonal (parallel) projection



For a tetrahedron

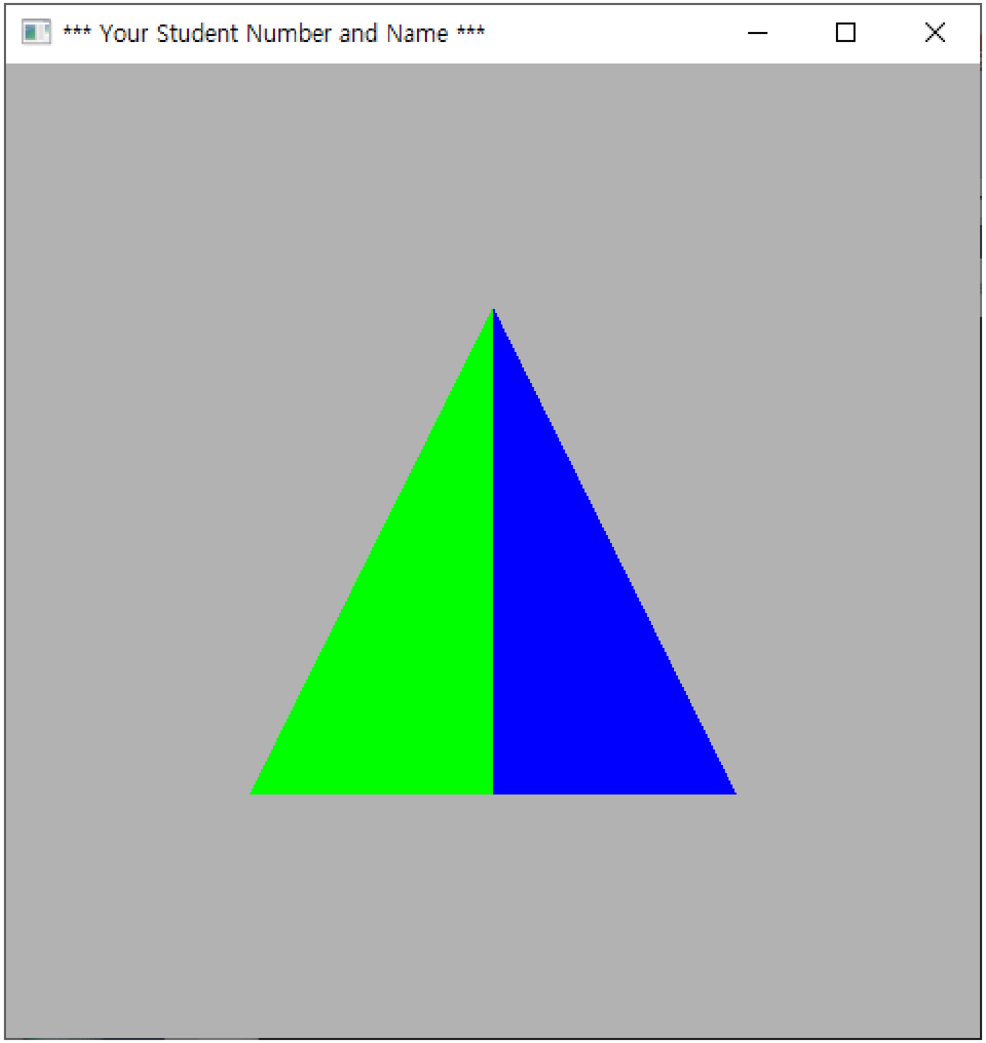




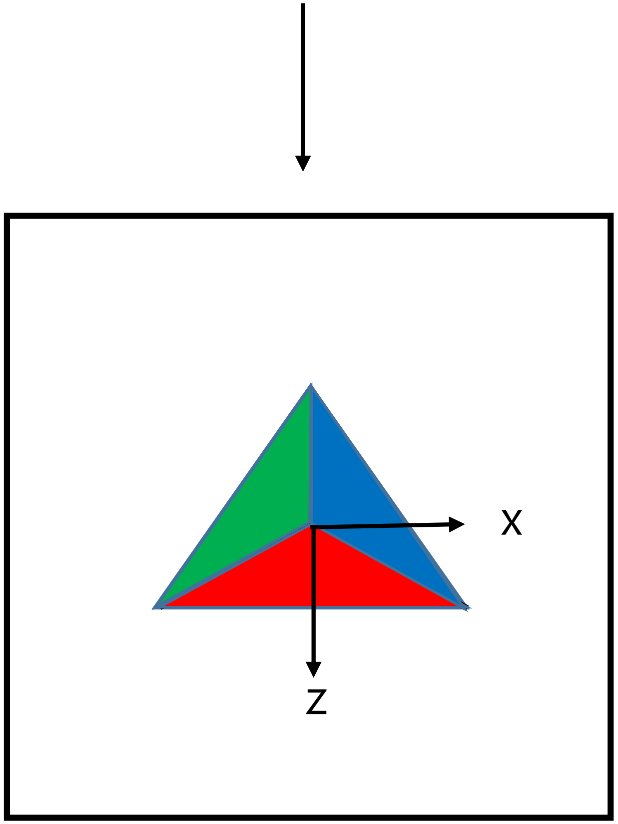


If view plane normal is
(0, 0, -1), the model will look

Actually we get is...



View plane normal : (0, 0, +1)



Moving the Camera

- If we want to move camera, we can do it either by
  - Moving the camera to **p** (Translate the camera frame)

  OR

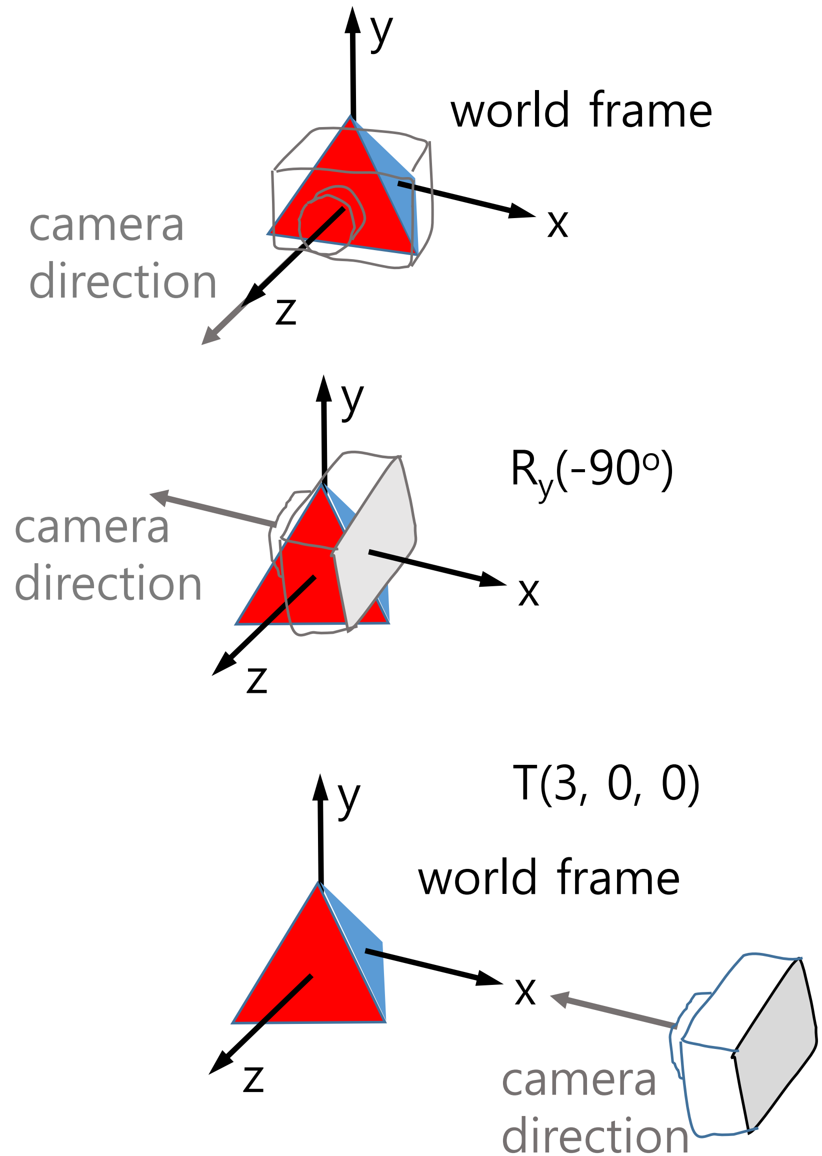
  - Moving the objects to –**p** (Translate the world frame)
- Both of these views are equivalent and are determined by the model-view matrix
  - Translation matrix: $T(-p_x, -p_y, -p_z)$ for the object

Model-View Matrix

- Both modeling matrix and viewing matrix are applied to objects with object transformation
- Concatenation of modeling and viewing matrices are called as model-view matrix

Moving the Camera (1)

- We can move the camera to render the side view of the object
- Example: side view
  - Rotate the camera: $R_y(-90^\circ)$
  - Move it away from origin: $T(3, 0, 0)$
- When we think the camera as an object:
  - Transformation matrix: $T(3, 0, 0)R_y(-90^\circ)$

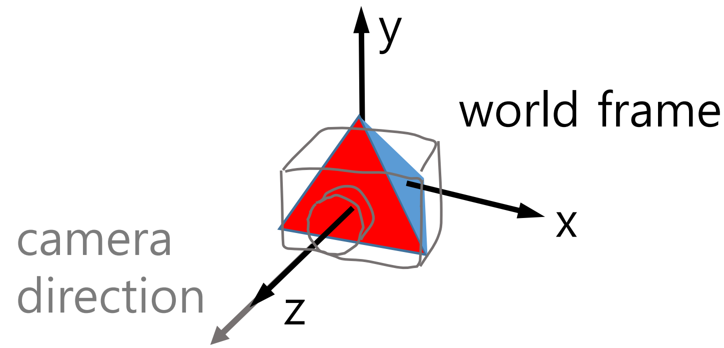


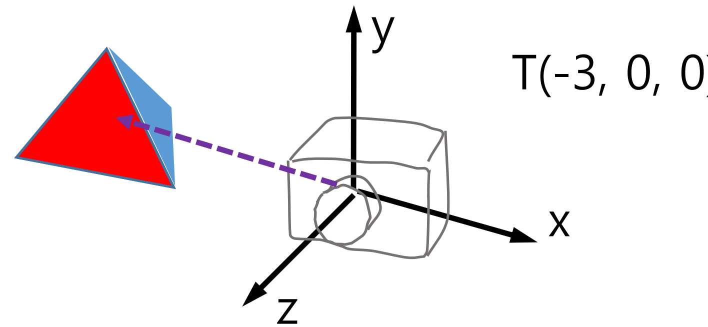
Moving the Camera (2)

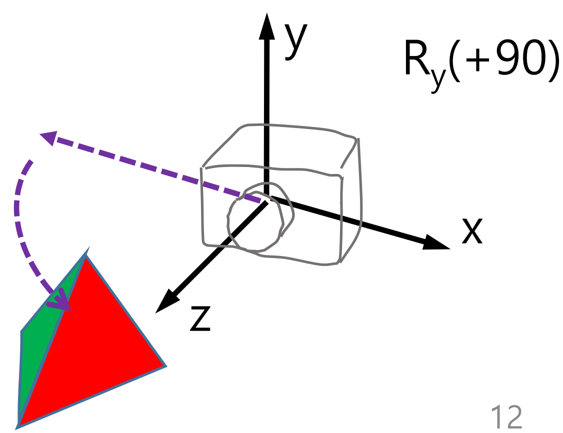
- If we want to move the object rather than camera for rendering the same result
  - Transformation matrix of camera:

    $C = T(3, 0, 0)R_y(-90^{\circ})$
  - Inverse of C must be applied to the object
  - Transformation matrix of object

    $C^{-1} = R_y(+90^{\circ})T(-3, 0, 0)$





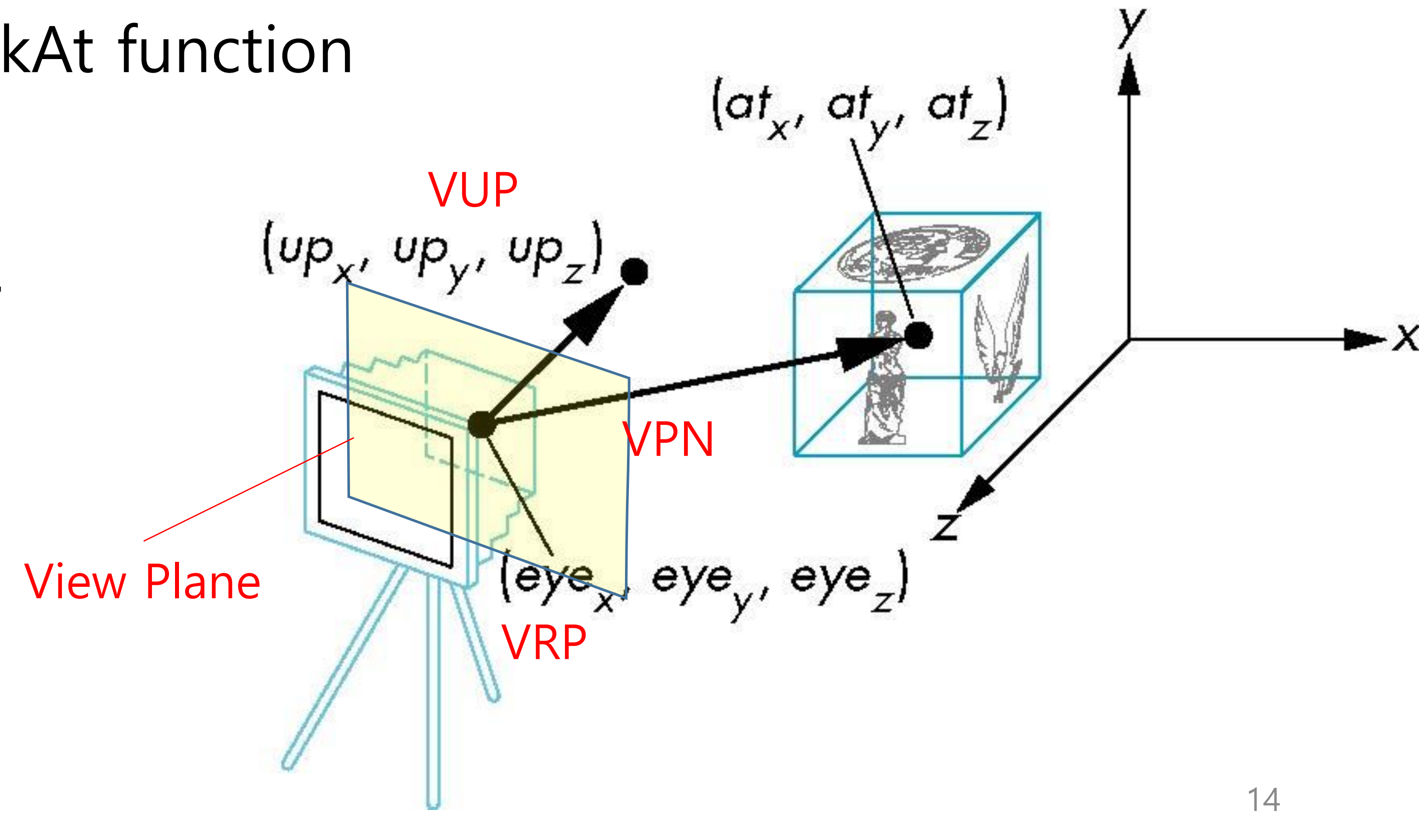


View Matrix Construction

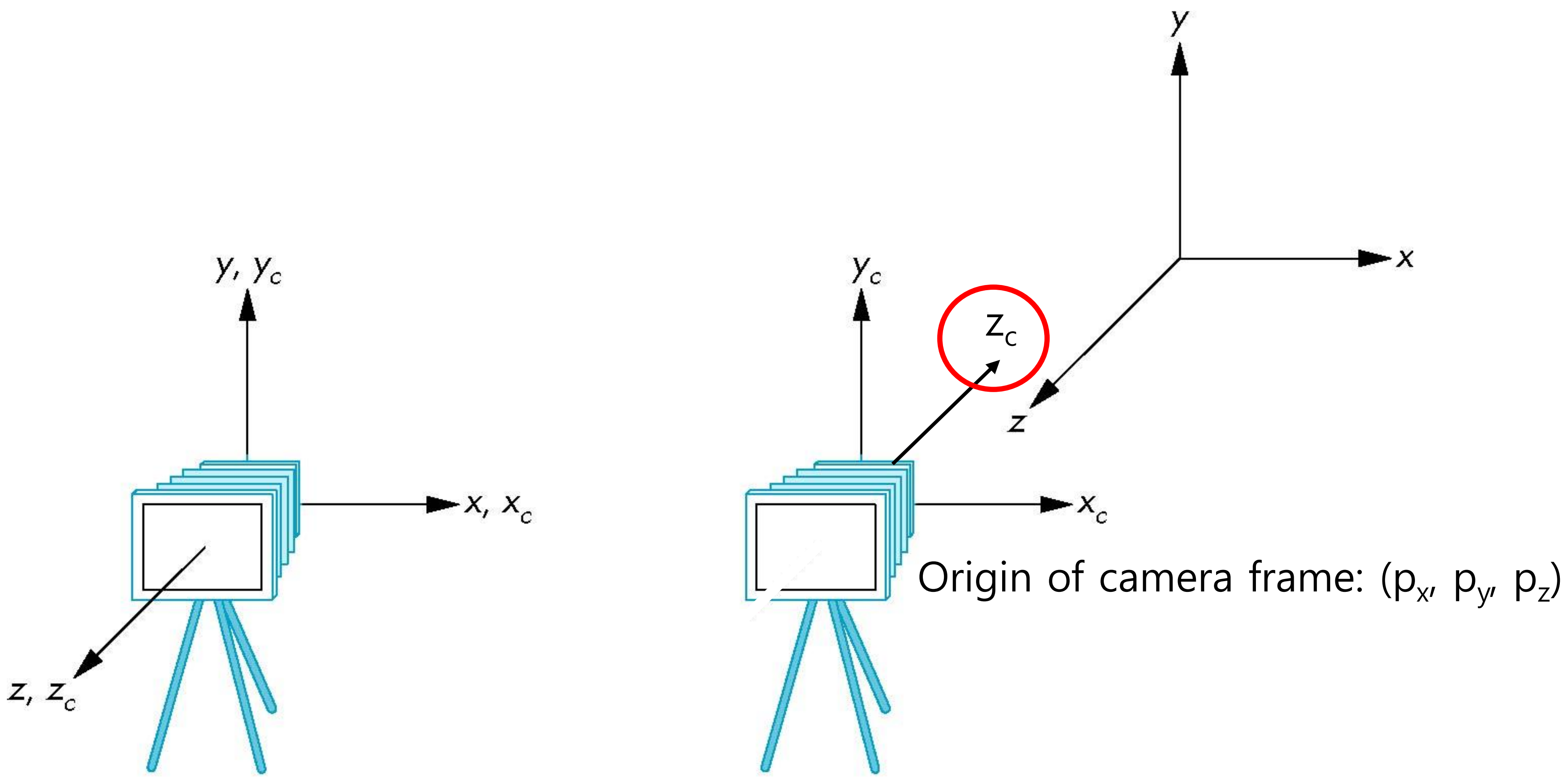
- For computing view matrix we can approach
  - Step1) Compute the transformation matrix (C) for the camera to be with the expected orientation (M_1) and location (M_2)
    - $C = M_2 M_1$
  - Step2) Compute the inverse matrix of C to apply it to objects
    - $C^{-1} = M_1^{-1} M_2^{-1}$

Deciding Camera Frame

- Deciding location and orientation of camera
  1) Using VRP, VUP, VPN
     - VRP (view reference point)
     - VPN (view plane normal)
     - VUP (view up vector)
  2) Using LookAt function
     - eye point
     - at point
     - up vector



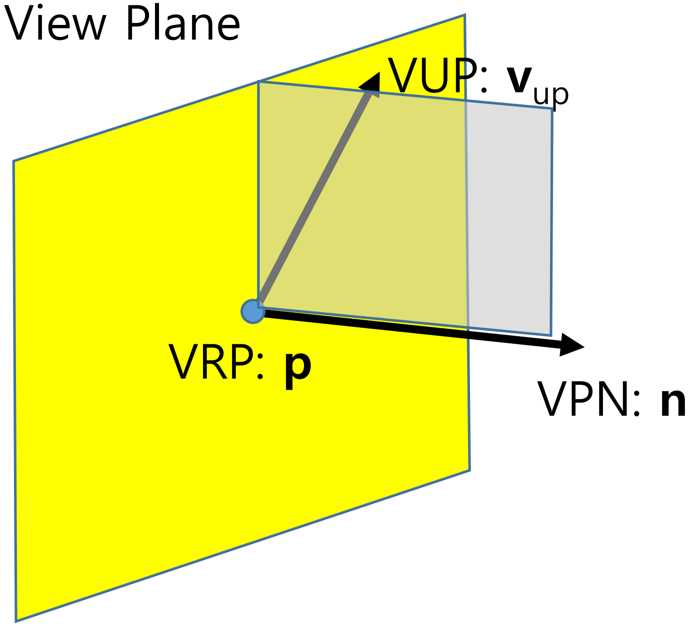
Moving Camera for Viewing Objects



default frames

frames after changing camera frame

Deciding Camera Frame by VRP, VPN, VUP (1)



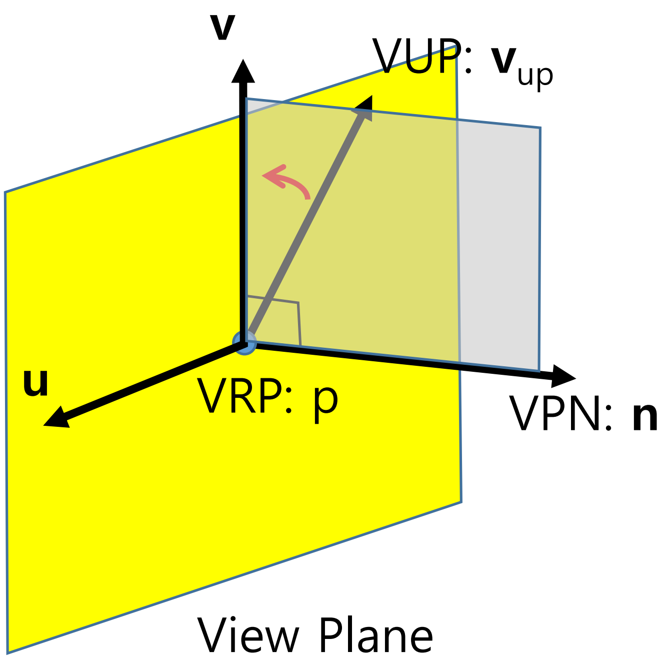
VRP : $\mathbf{p} = [p_x \;\; p_y \;\; p_z \;\; 1]^T$
VPN : $\mathbf{n} = [n_x \;\; n_y \;\; n_z \;\; 0]^T$
VUP : $\mathbf{v}_{up} = [v_{upx} \;\; v_{upy} \;\; v_{upz} \;\; 0]^T$

- Vector normalization
  for a vector $\mathbf{v} = (v_x, v_y, v_z)$:
  $\mathbf{v}/||\mathbf{v}||$,
  where $||\mathbf{v}|| = \mathrm{sqrt}(v_x^2 + v_y^2 + v_z^2)$

- $\mathbf{n}$, $\mathbf{v}$ are normalized
  : $\mathbf{n} = \mathbf{n} / ||\mathbf{n}||$
  $\mathbf{v}_{up} = \mathbf{v}_{up} / ||\mathbf{v}_{up}||$

→ length is 1

Deciding Camera Frame by VRP, VPN, VUP (2)



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

normalized VPN : $\mathbf{n} = [n_x \; n_y \; n_z \; 1]^T$

normalized VUP : $\mathbf{v}_{up}=[v_{upx} \; v_{upy} \; v_{upz} \;\; 1]^T$

Constructing camera frame (**u**-**v**-**n**-**p**):

1) v: Projection of VUP to view plane

Let's compute **v**:

$\mathbf{v} = \alpha\, \mathbf{n} + \beta\, \mathbf{v}_{up},$

If β is assumed to be 1, then

$(\alpha\, \mathbf{n} + \mathbf{v}_{up}) \cdot \mathbf{n} = 0,$

since **v** and **n** are perpendicular.

$\alpha = -(\mathbf{v}_{up} \cdot \mathbf{n})/(\mathbf{n} \cdot \mathbf{n})$, where $\mathbf{n} \cdot \mathbf{n} = 1$

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

2) Compute **u** as $\mathbf{u} = \mathbf{n} \times \mathbf{v}$

3) Normalize **u** and **v**

Default Camera Frame in OpenGL

- Camera position: p = (0, 0, 0)
- View plane normal: n = (0, 0, 1) // z_c in page15
- View up vector: v = (0, 1, 0) // y_c
- u = (1, 0, 0) //x_c
  - This follows right hand rule, when u, v, n correspond to x, y, z, respectively
  - But, we want camera frame with left hand rule, when we change it !!!

HW#17 Show Green Face (5points)

- Due date:This Friday 6:00pm
- Modify the program "LEC17.2_side_view.c" to show the green face of given object.
- Don't change vertices, colors, and indices arrays, and the result must be seen as the right figure.
- Submit .c file through LMS.