
Computer Graphics

5 - Vertex Processing 1

Yoonsang Lee
Hanyang University

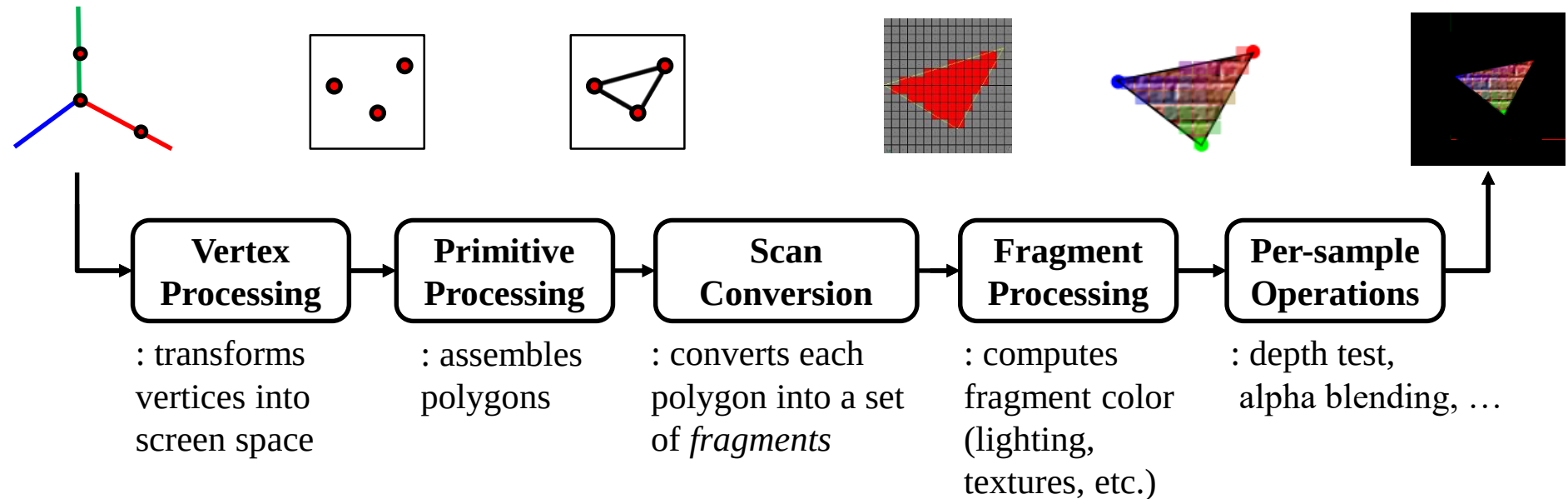
Spring 2026

Outline

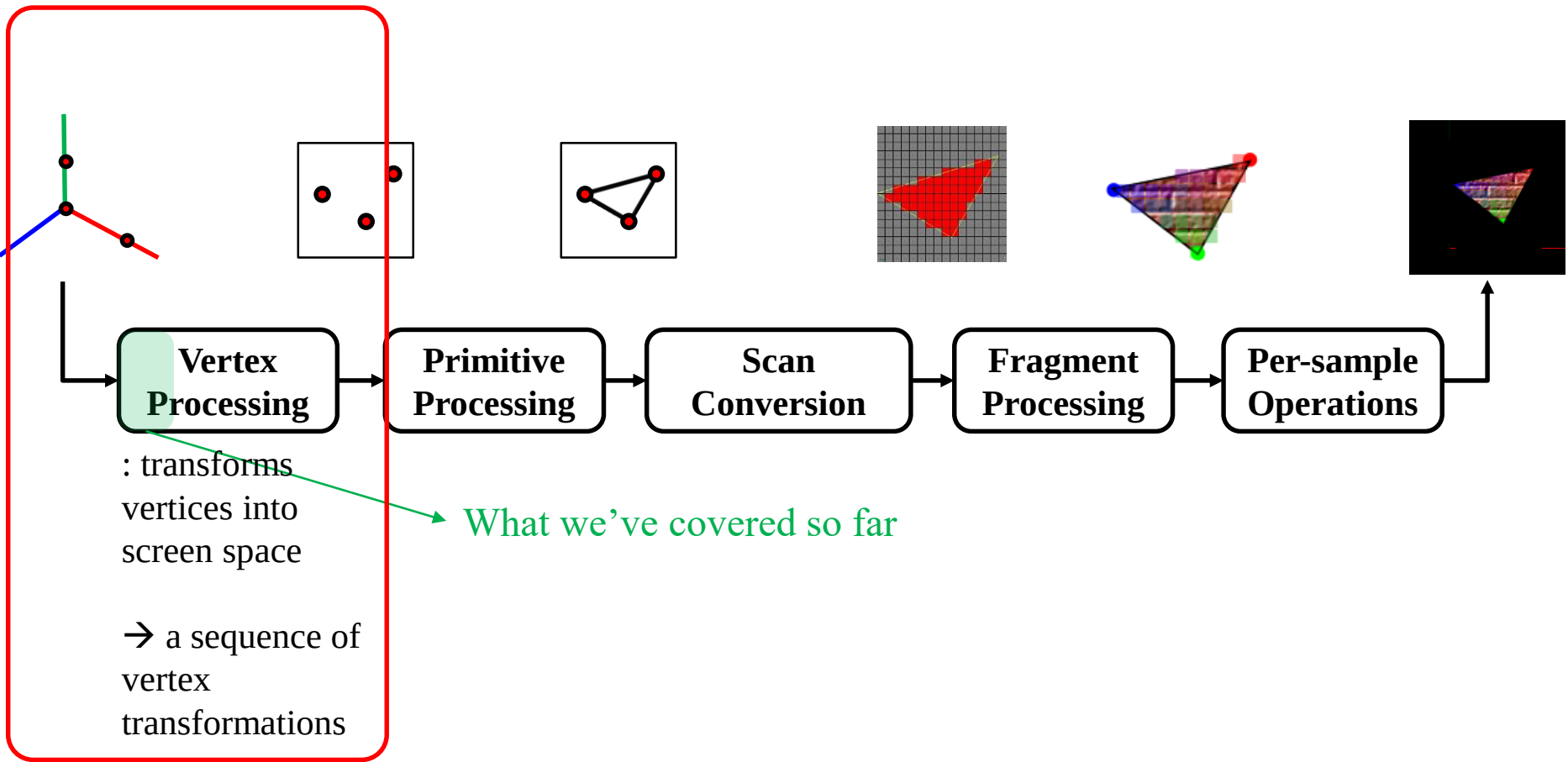
- Rasterization Pipeline & Vertex Processing
- Modeling Transformation
- Viewing Transformation

Rasterization Pipeline & Vertex Processing

Recall: Rasterization Pipeline



Recall: Rasterization Pipeline



→ What we'll cover
today and next time

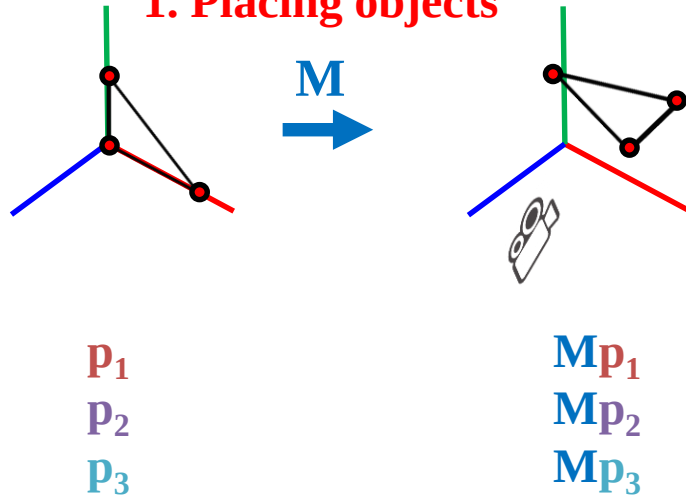
Vertex Processing

Vertex positions
in each object's
body frame

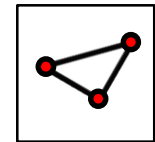
Vertex positions
in world frame

Vertex positions in
2D viewport

1. Placing objects



To do this, we need to
introduce the concept
of a ‘camera’ viewing
the scene.



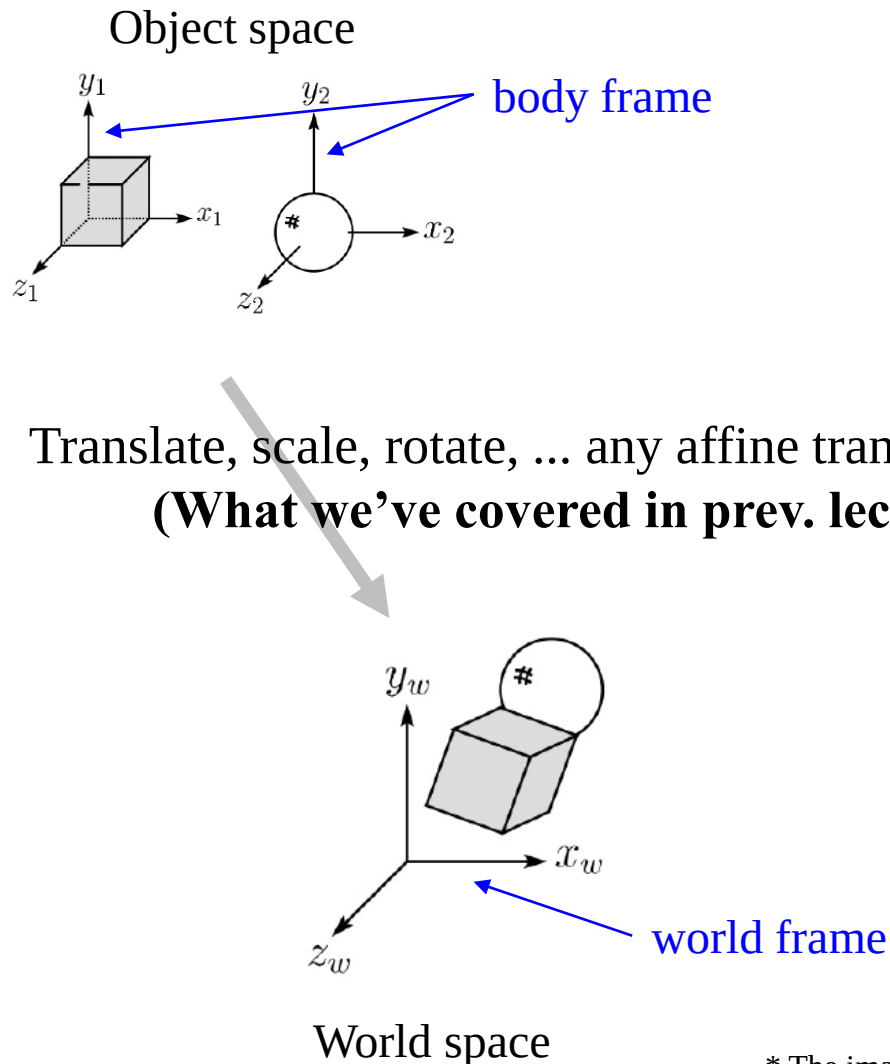
Then the next steps are:

- 2. Positioning the camera**
- 3. Choosing the lens**
- 4. Displaying on the screen**

From the Perspective of CG Transformations

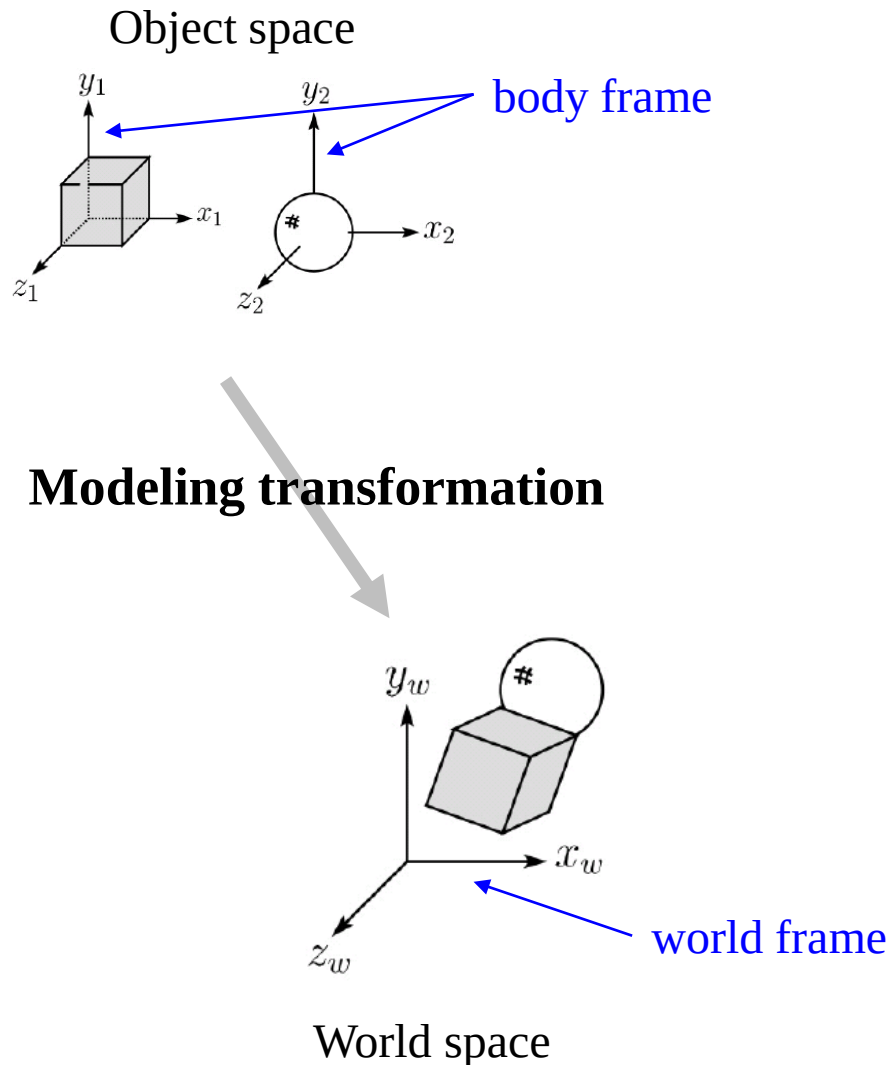
- 1. Placing objects
→ **Modeling transformation**
- 2. Positioning the camera
→ **Viewing transformation**
- 3. Choosing the lens
→ **Projection transformation**
- 4. Displaying on the screen
→ **Viewport transformation**
- All these transformations just work by **matrix multiplications!**

Vertex Processing (Transformation Pipeline)

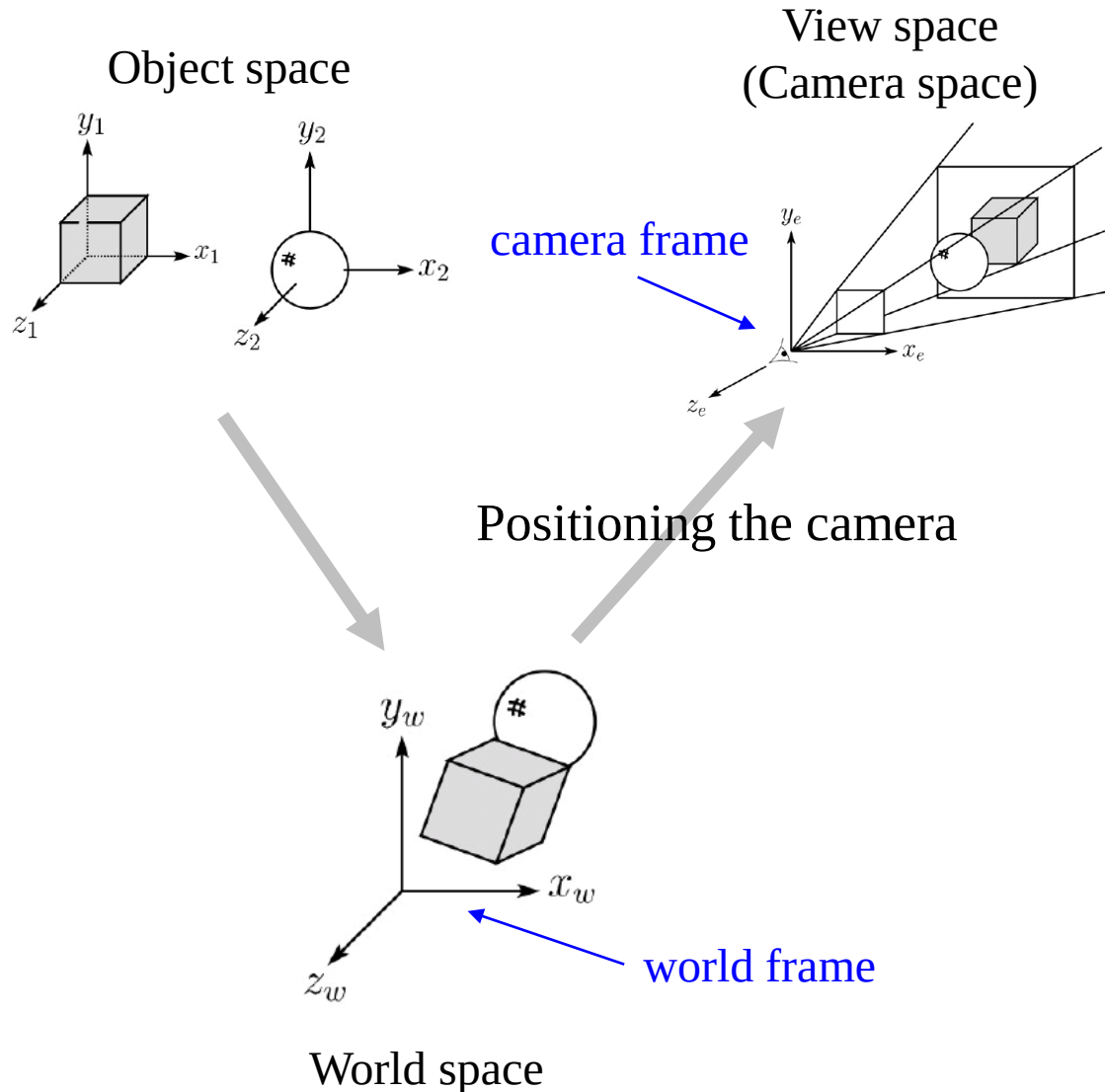


* The images are from the slides of Prof. Jinxiang Chai (Texas A&M University):
http://faculty.cs.tamu.edu/jchai/csce441_2016spring/lectures.html

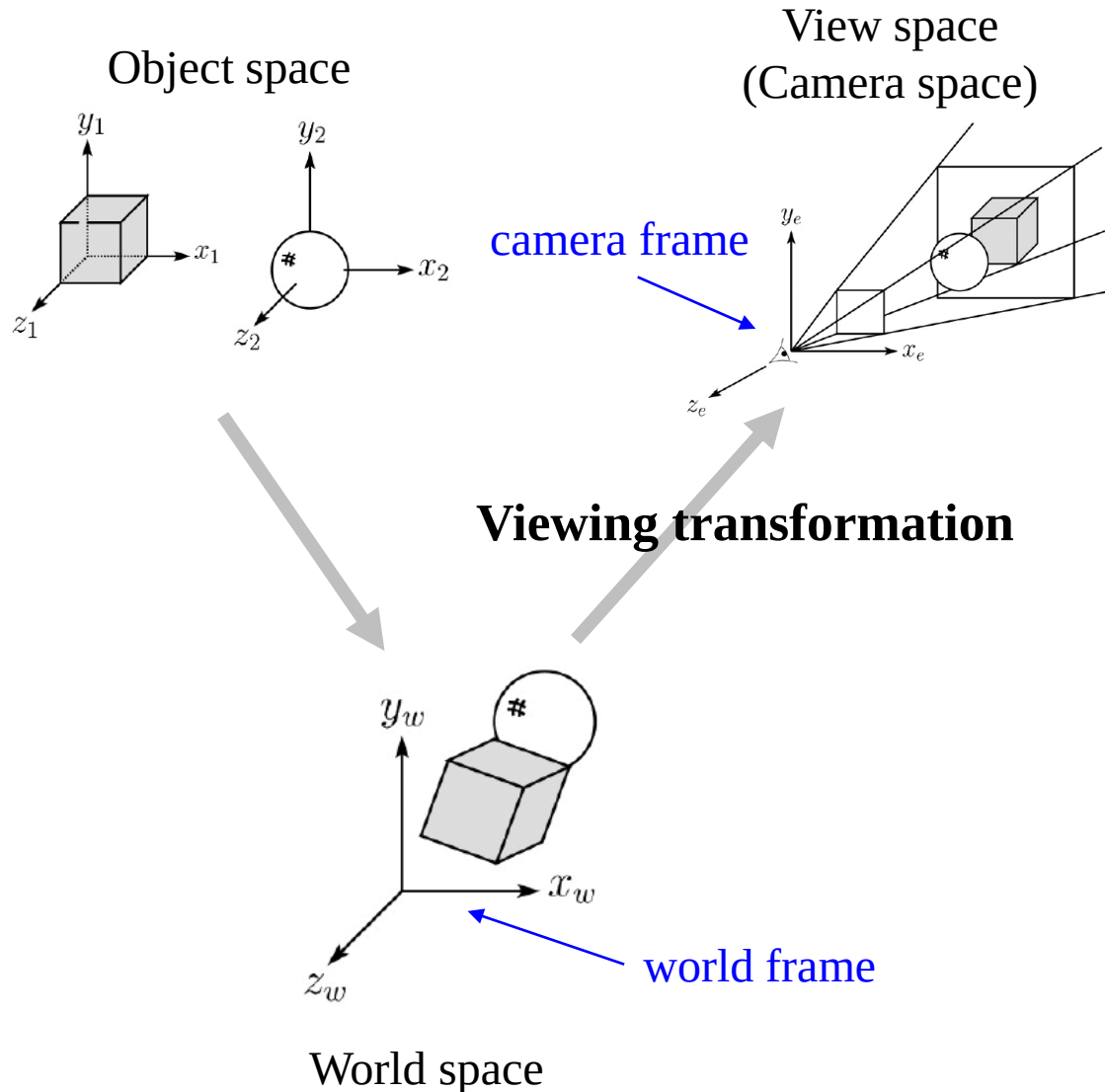
Vertex Processing (Transformation Pipeline)



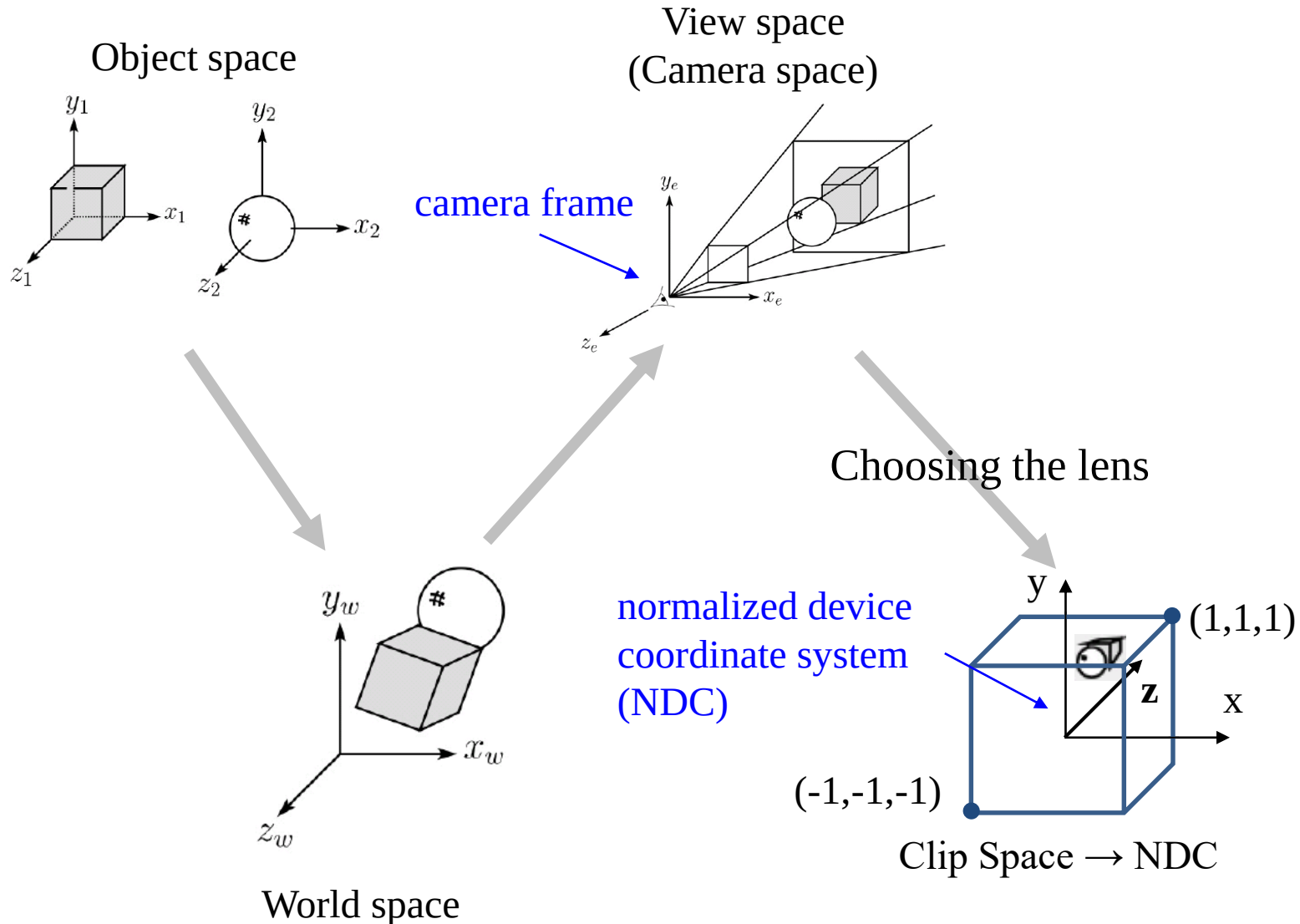
Vertex Processing (Transformation Pipeline)



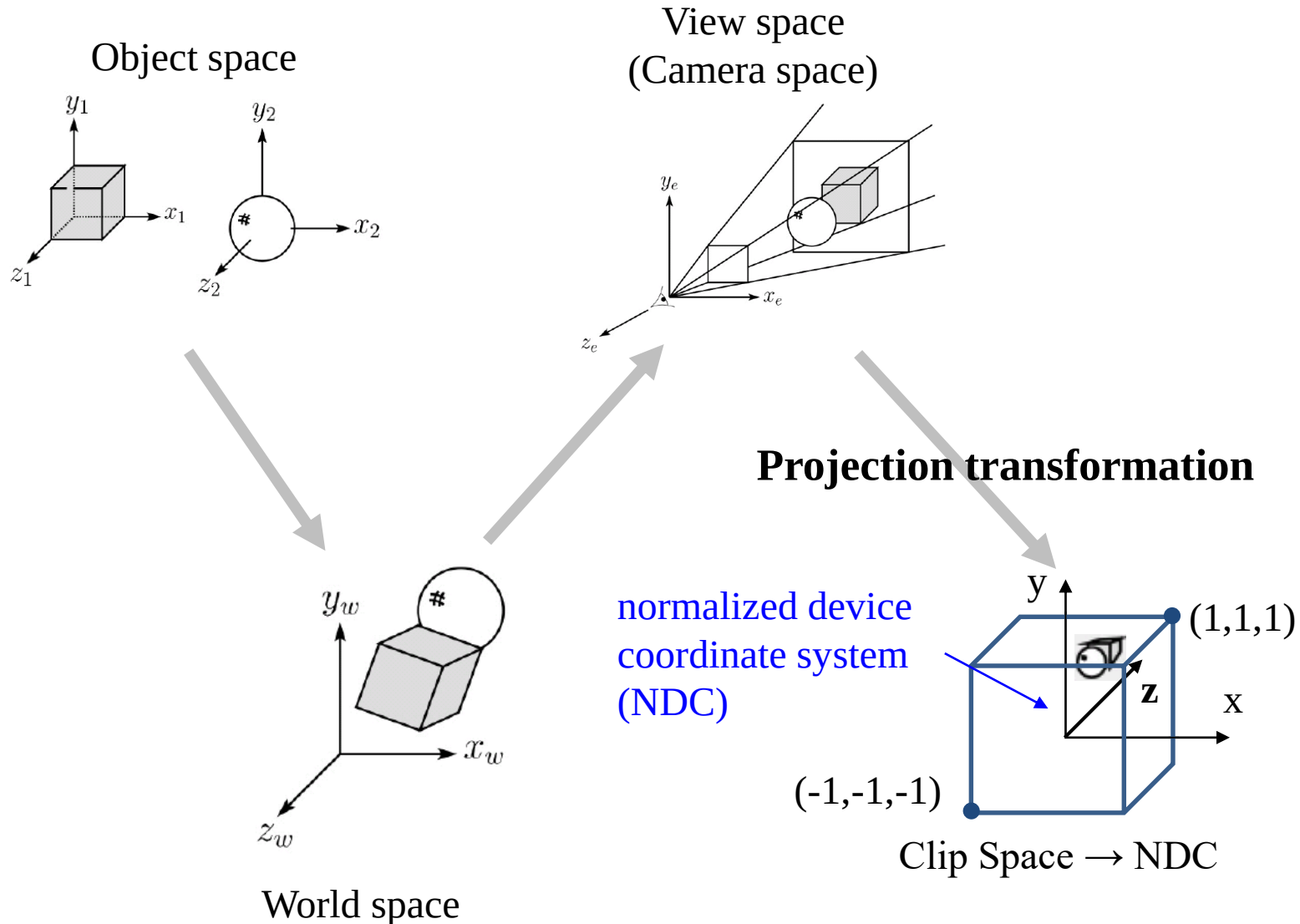
Vertex Processing (Transformation Pipeline)



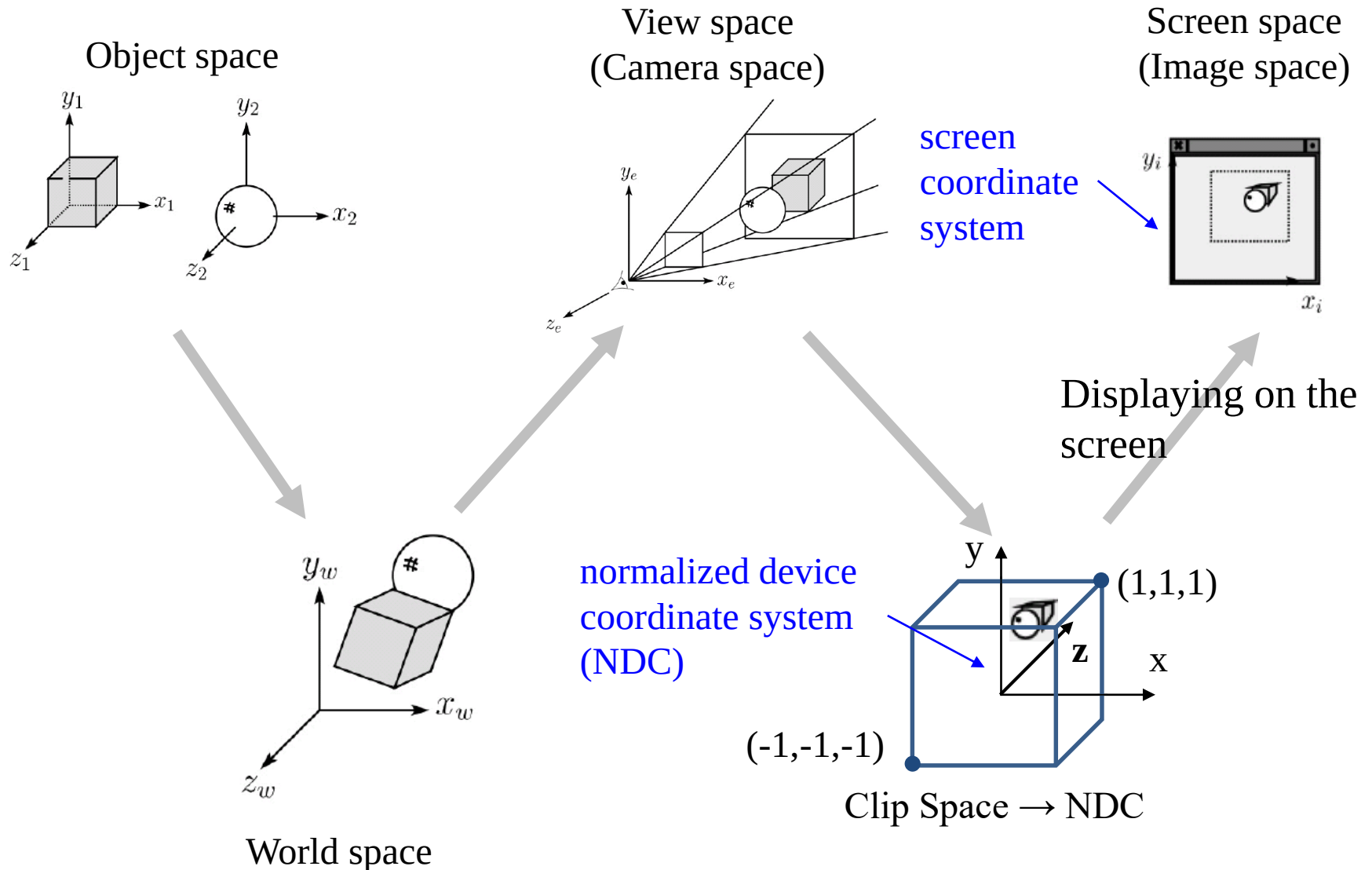
Vertex Processing (Transformation Pipeline)



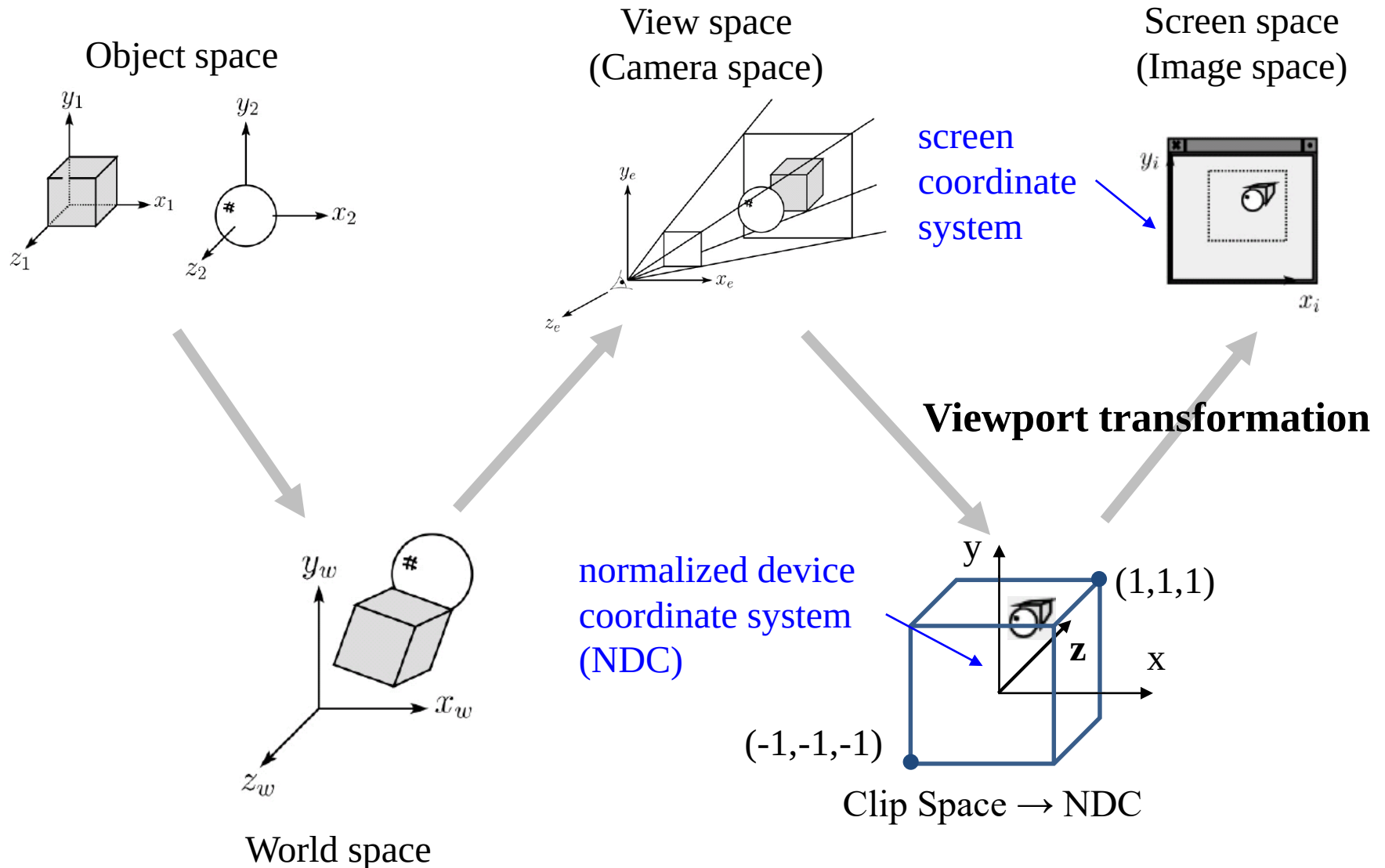
Vertex Processing (Transformation Pipeline)



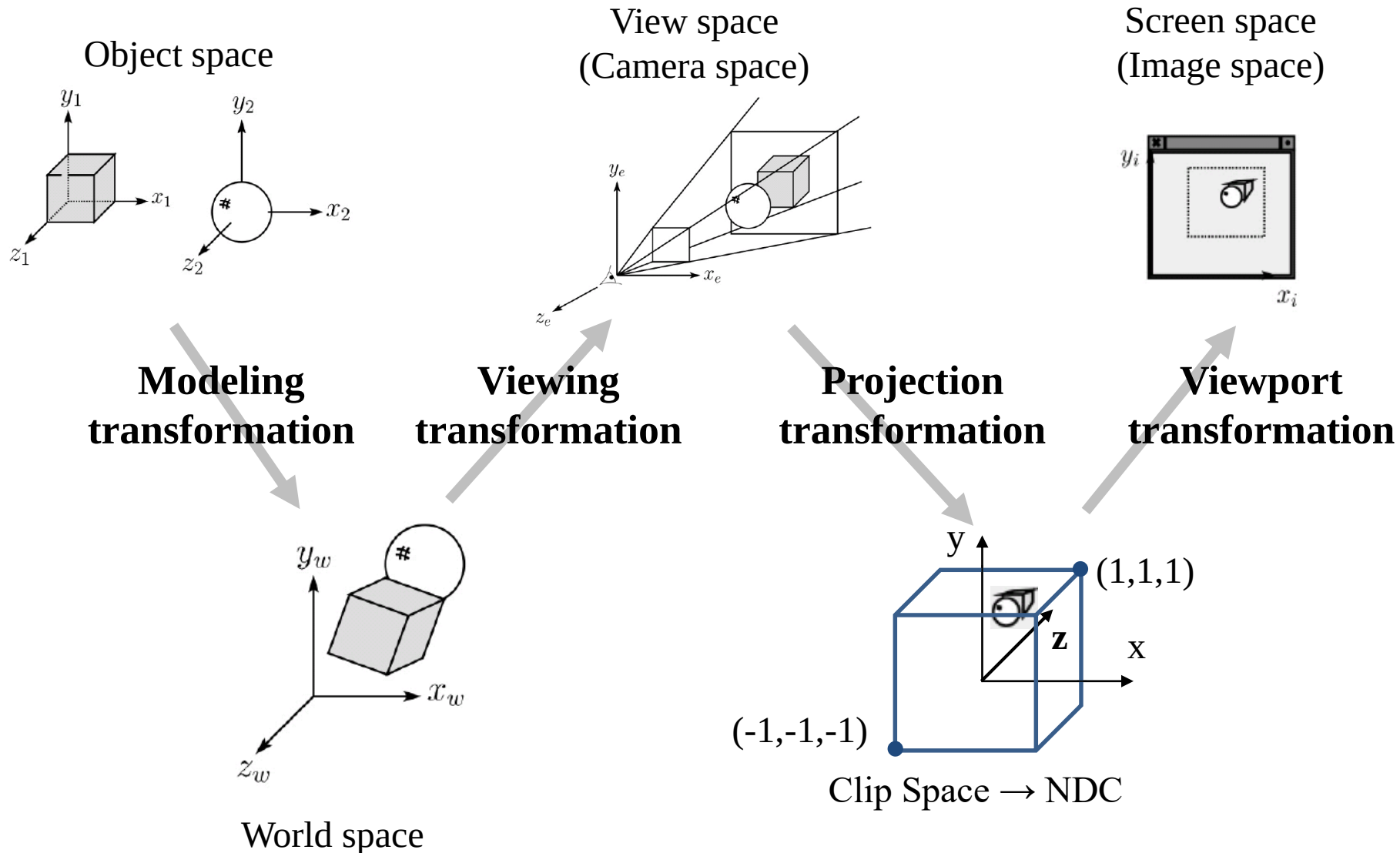
Vertex Processing (Transformation Pipeline)



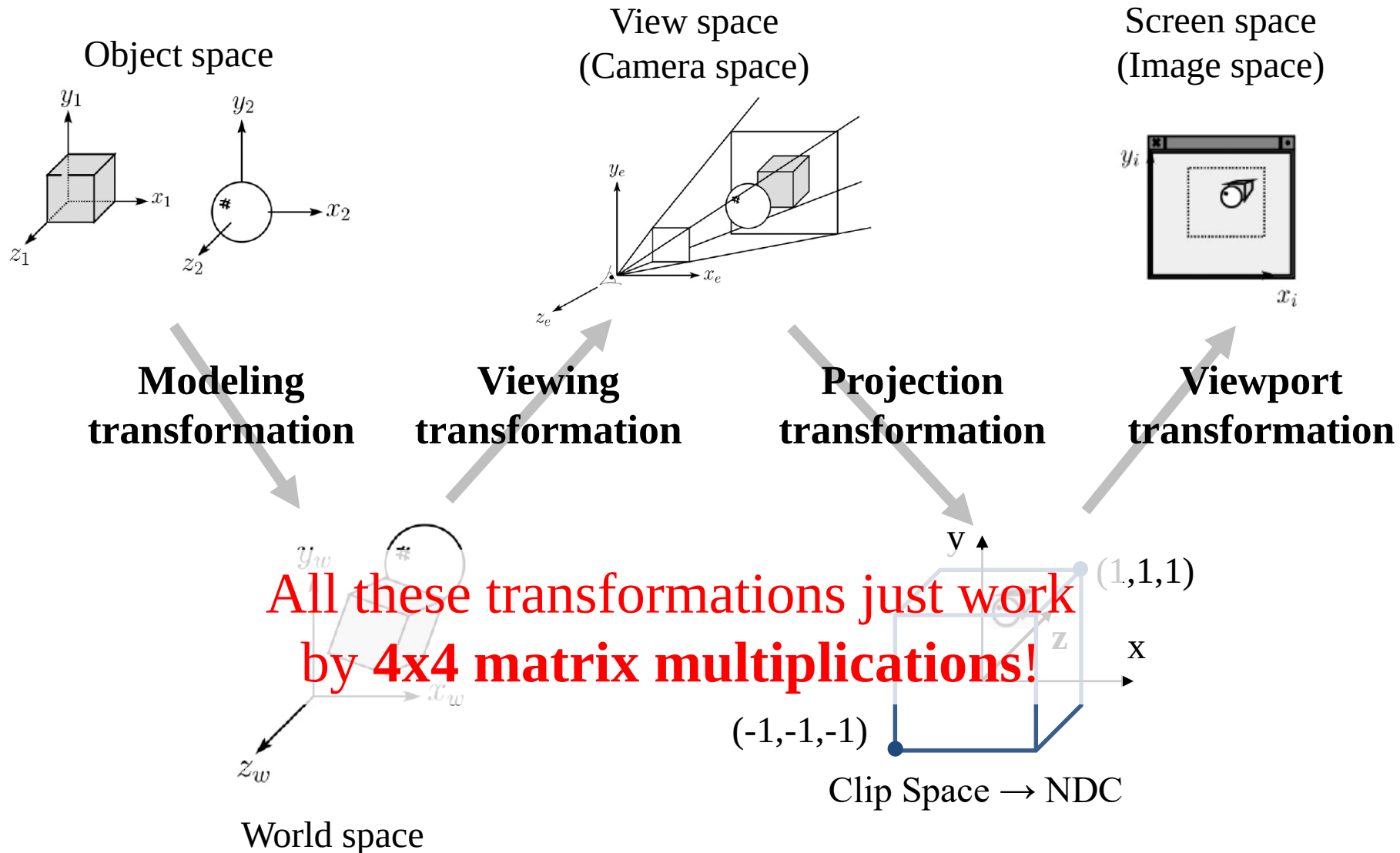
Vertex Processing (Transformation Pipeline)



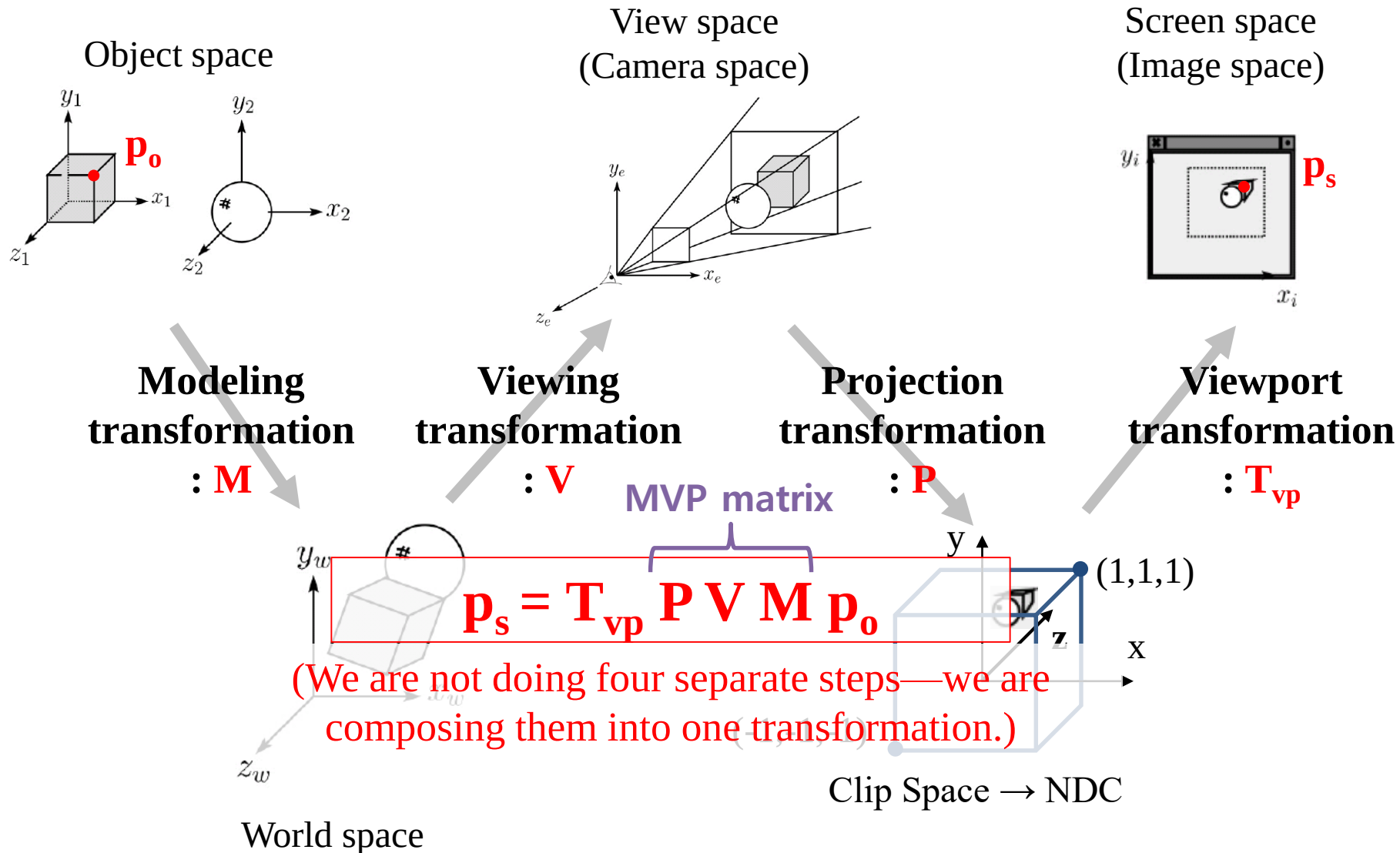
Vertex Processing (Transformation Pipeline)



Vertex Processing (Transformation Pipeline)



Vertex Processing (Transformation Pipeline)



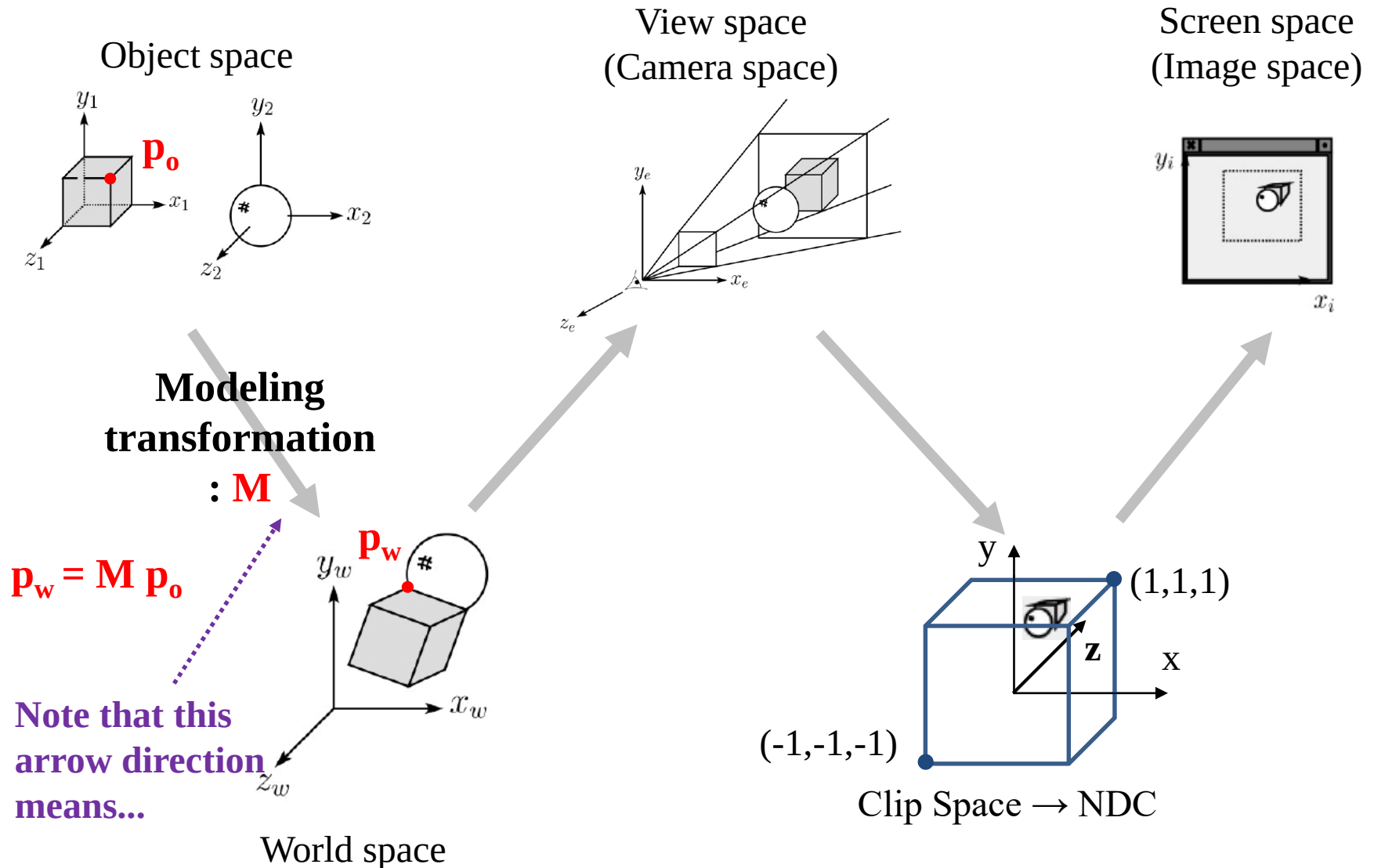
Two Interpretations of Matrix Composition

$$\mathbf{p}_s = \mathbf{T}_{vp} \mathbf{P} \mathbf{V} \mathbf{M} \mathbf{p}_o$$

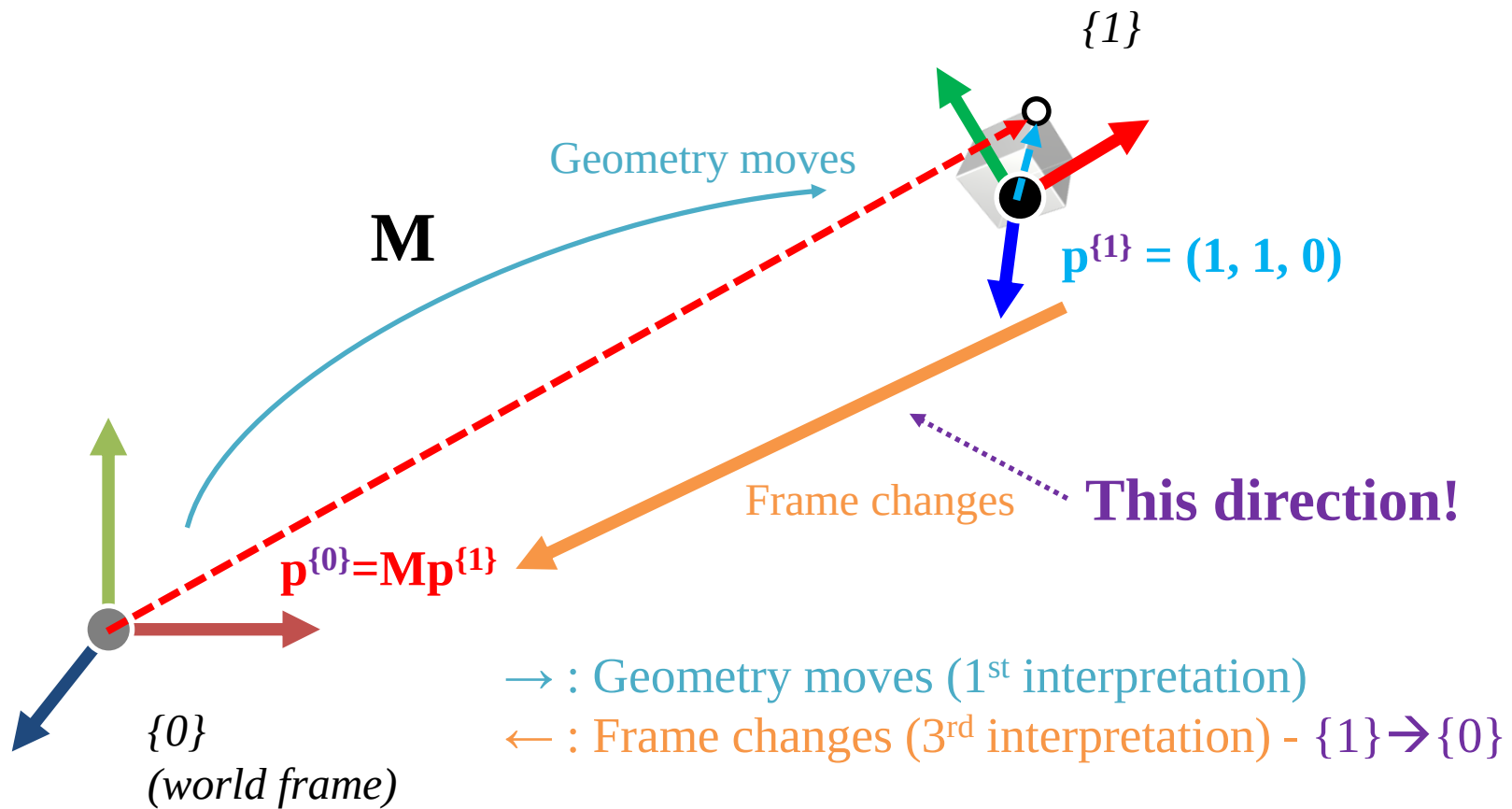
- L-to-R (frame interpretation) – less common in this context
 - Each step defines a new frame
- **R-to-L** (coordinate value transformation)
 - Points are transformed through different frames
- Same mathematics, different interpretations

Modeling Transformation

Modeling Transformation

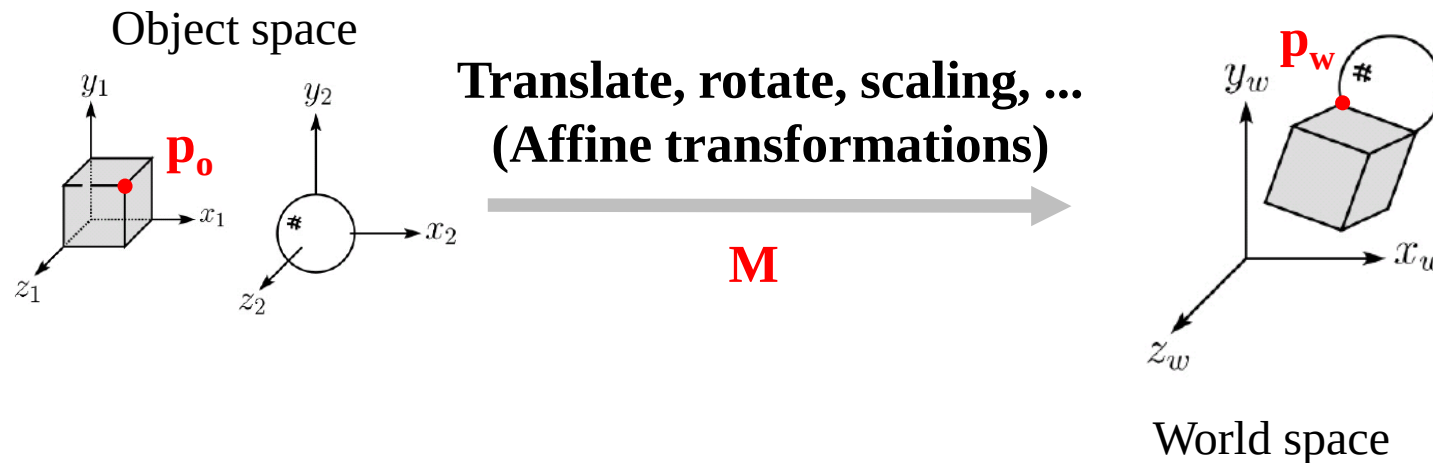


Recall: Direction Depends on Interpretation

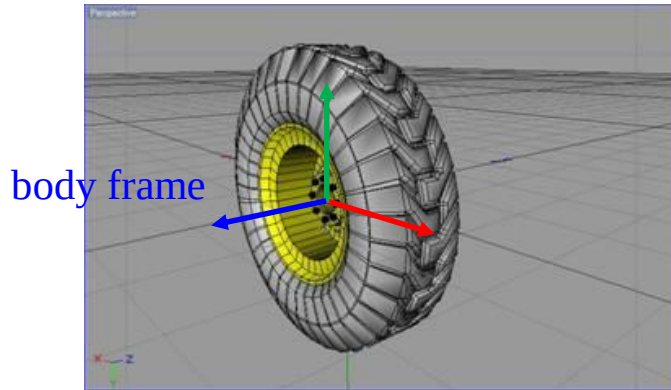


Modeling Transformation

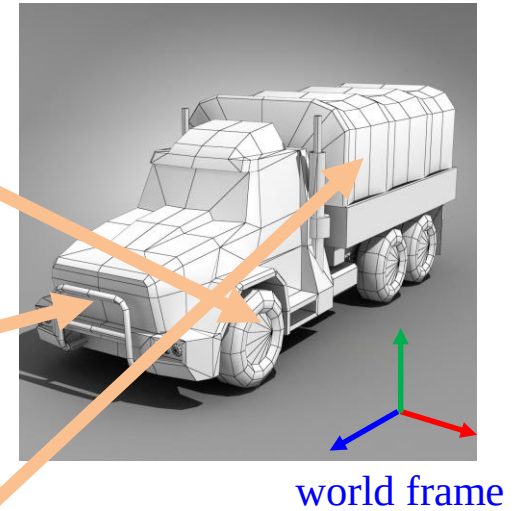
- An object is defined in the **object's body frame**.
- The modeling transformation (from object space to world space) is represented by the *modeling matrix*, **M**.
- This **M** is a composition of affine transformations we've covered so far.



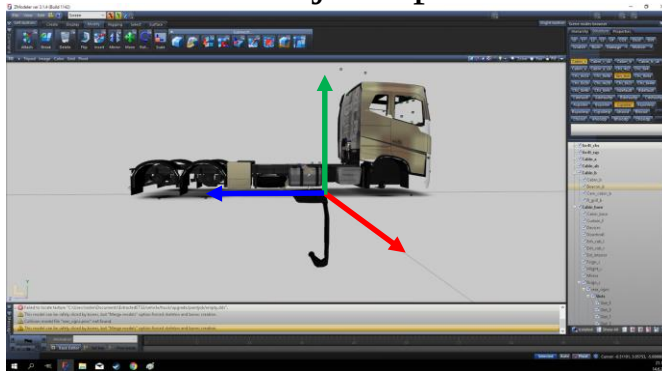
Wheel object space



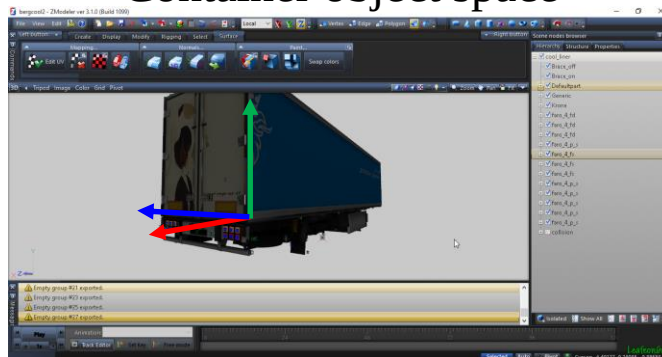
World space



Cab object space



Container object space



M^{wheel}

M^{cab}

$M^{\text{container}}$

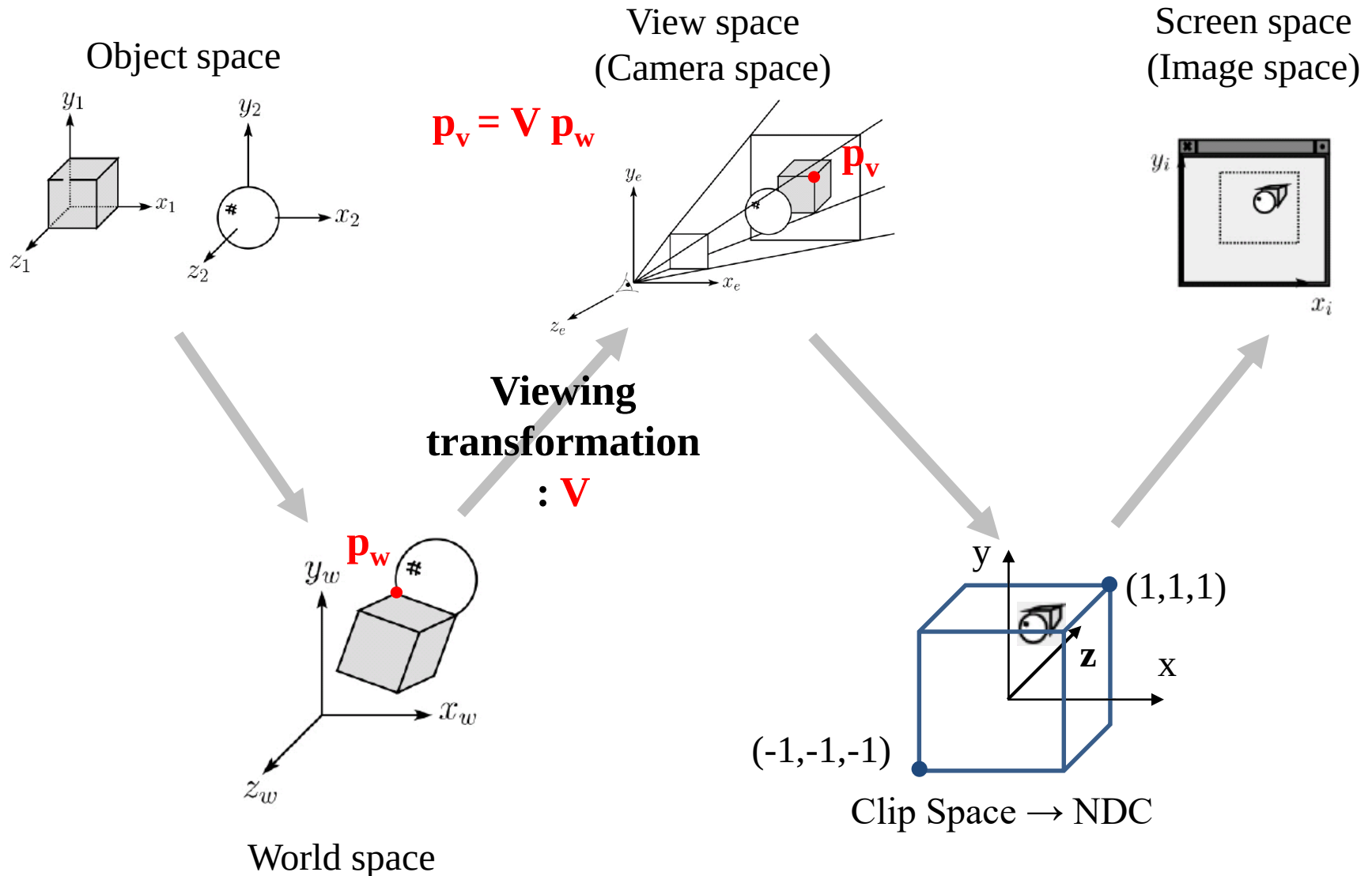
(Each object has its own modeling matrix)

Quiz 1

- Go to <https://www.slido.com/>
- Join #cg-ys
- Click "Polls"
- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2024123456: 4.0**
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

Viewing Transformation

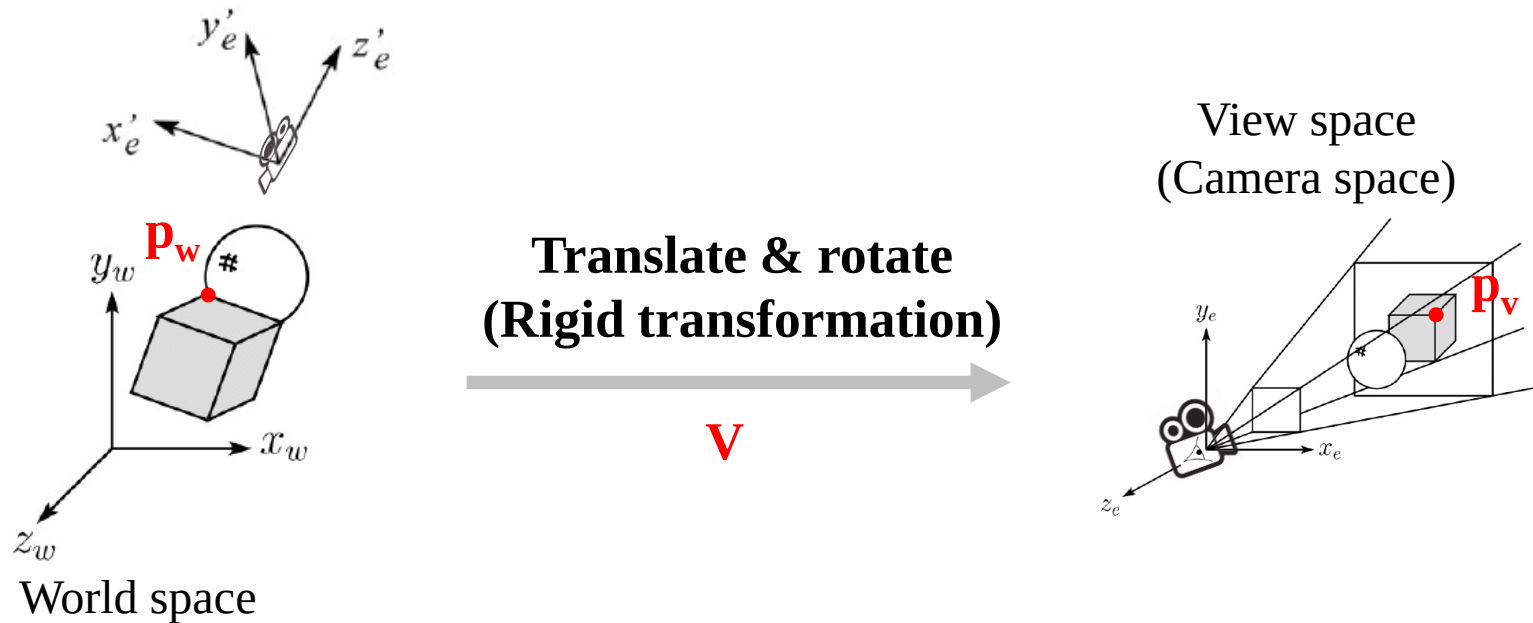
Viewing Transformation



Recall that...

- 1. Placing objects
→ **Modeling transformation**
- 2. Positioning the camera
→ **Viewing transformation**
- 3. Choosing the lens
→ **Projection transformation**
- 4. Displaying on the screen
→ **Viewport transformation**

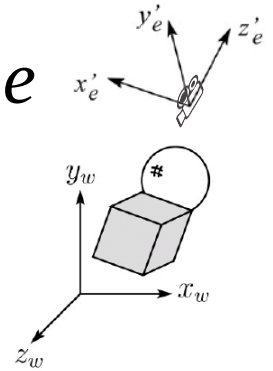
Viewing Transformation



- The viewing transformation (from world space to view space) is represented by the *viewing matrix*, V .
- Since the camera only translates and rotates, this is a rigid transformation.

Viewing Transformation

- Goal: Expressing all vertices in the camera's coordinate system (*camera frame*).
- To do that, we need to define the *camera frame* (w.r.t. world frame).



- In other words, defining the *camera frame* means specifying the *camera's position and orientation*.

Defining Camera Frame 1 - "LookAt"

- Many ways to specify the camera's position & orientation.
- One intuitive method is the “*Lookat*” function that uses:

- **Eye point**

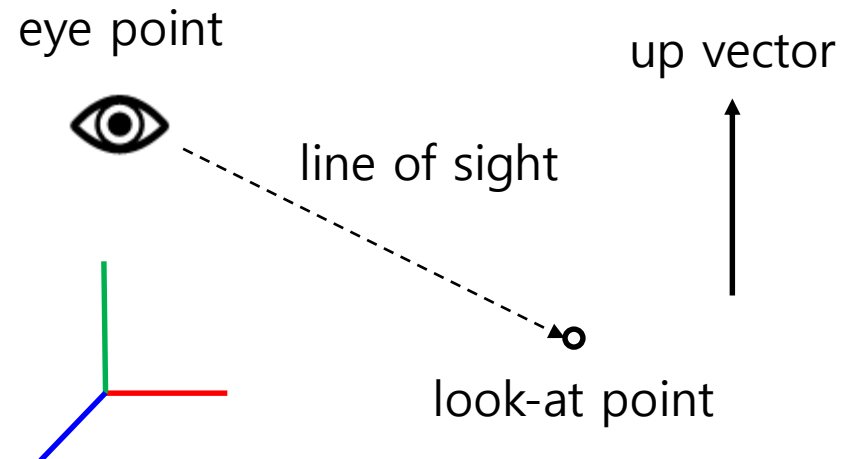
- Camera position

- **Look-at point**

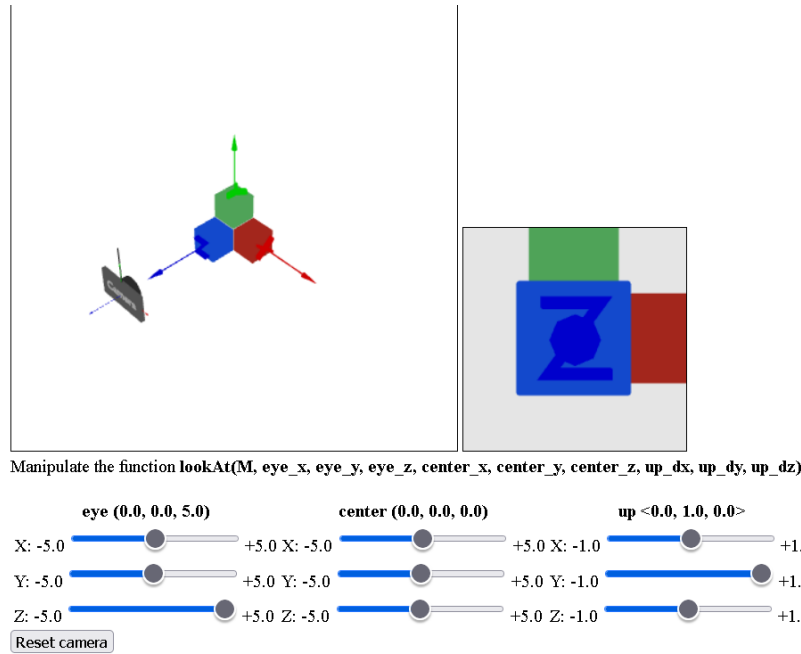
- Camera target position

- **Up vector**

- Reference direction for the camera’s ‘up’



[Demo] LookAt Function

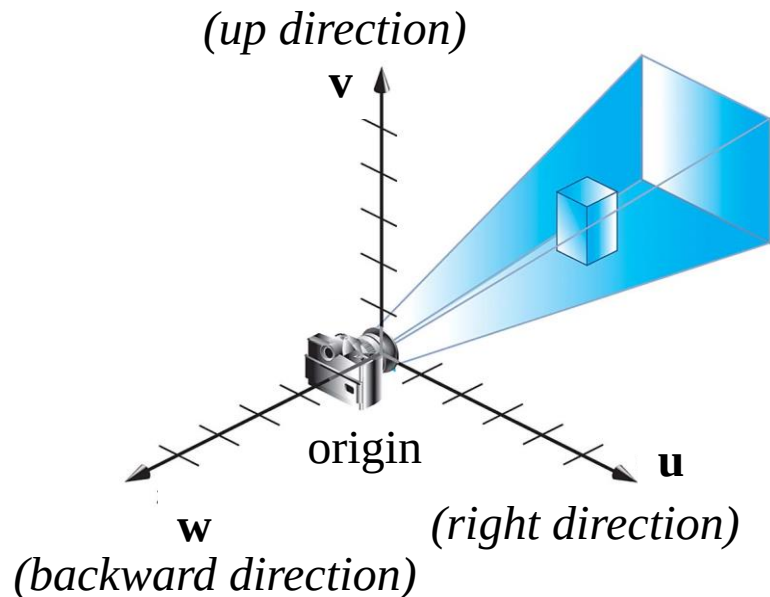


http://learnwebgl.brown37.net/07_cameras/camera_lookat/camera_lookat.html

- Observe the 3D scene (left) and rendered view (right) while changing eye, center, up by dragging sliders.

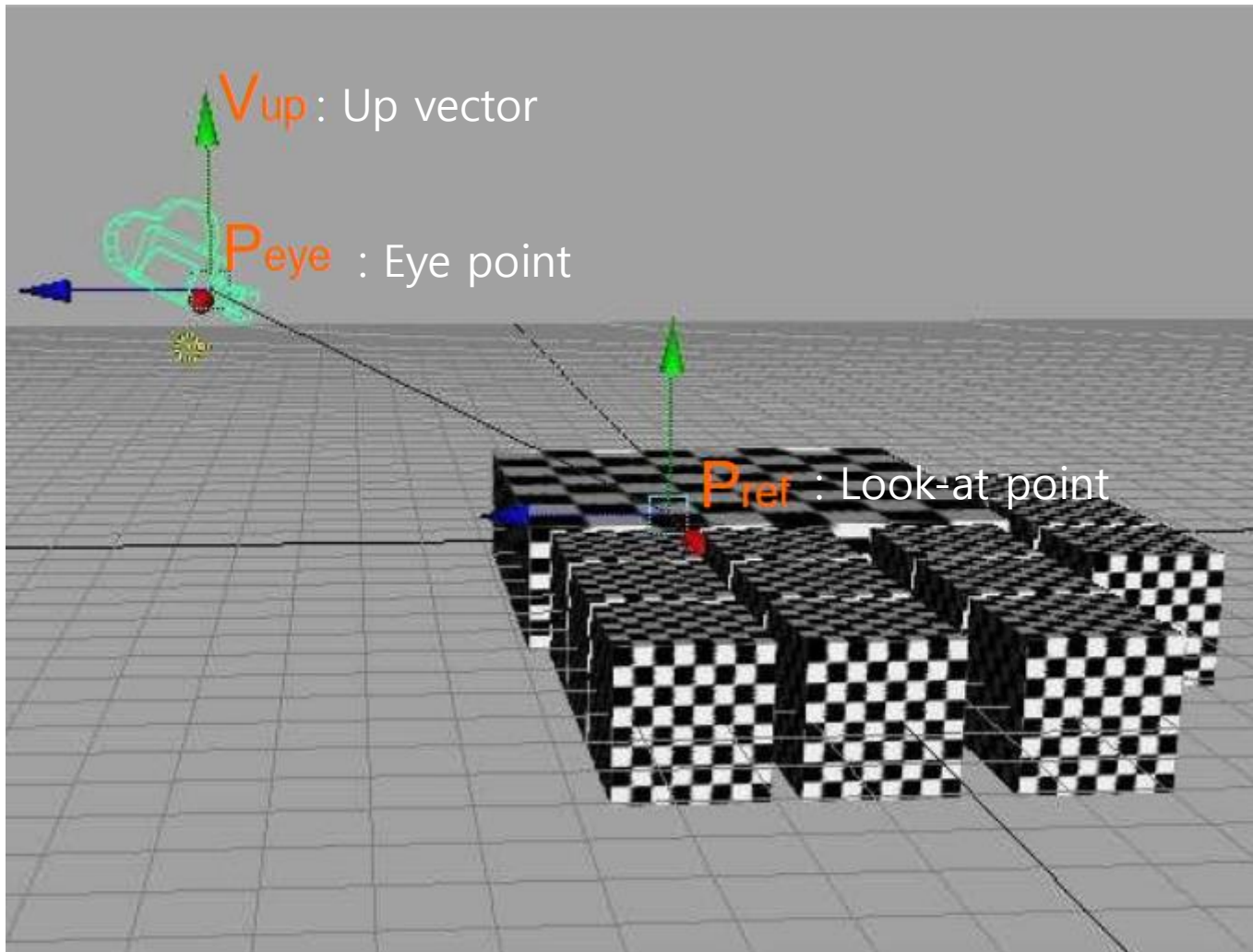
Defining Camera Frame 1 - "LookAt"

- Given **eye point**, **look-at point**, **up vector**, we can compute the **camera frame**.
- Note that **$\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$** are commonly used for camera coordinates axes instead of x , y , z .
 - These correspond to the camera's right, up, and backward directions.



- To define the camera frame, we need to find:
- **$\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$** vectors
- **origin** point

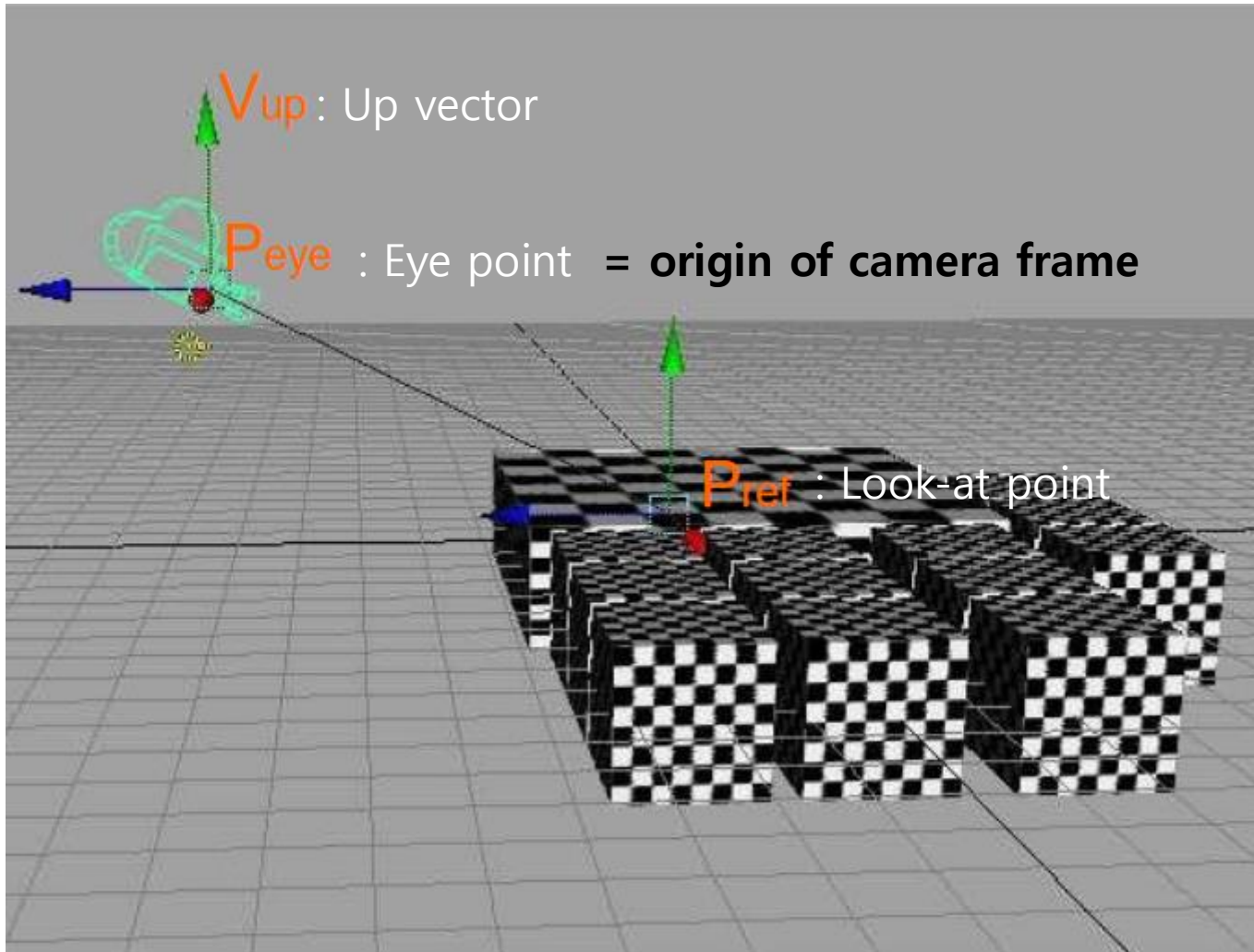
Given Eye point, Look-at point, Up vector,



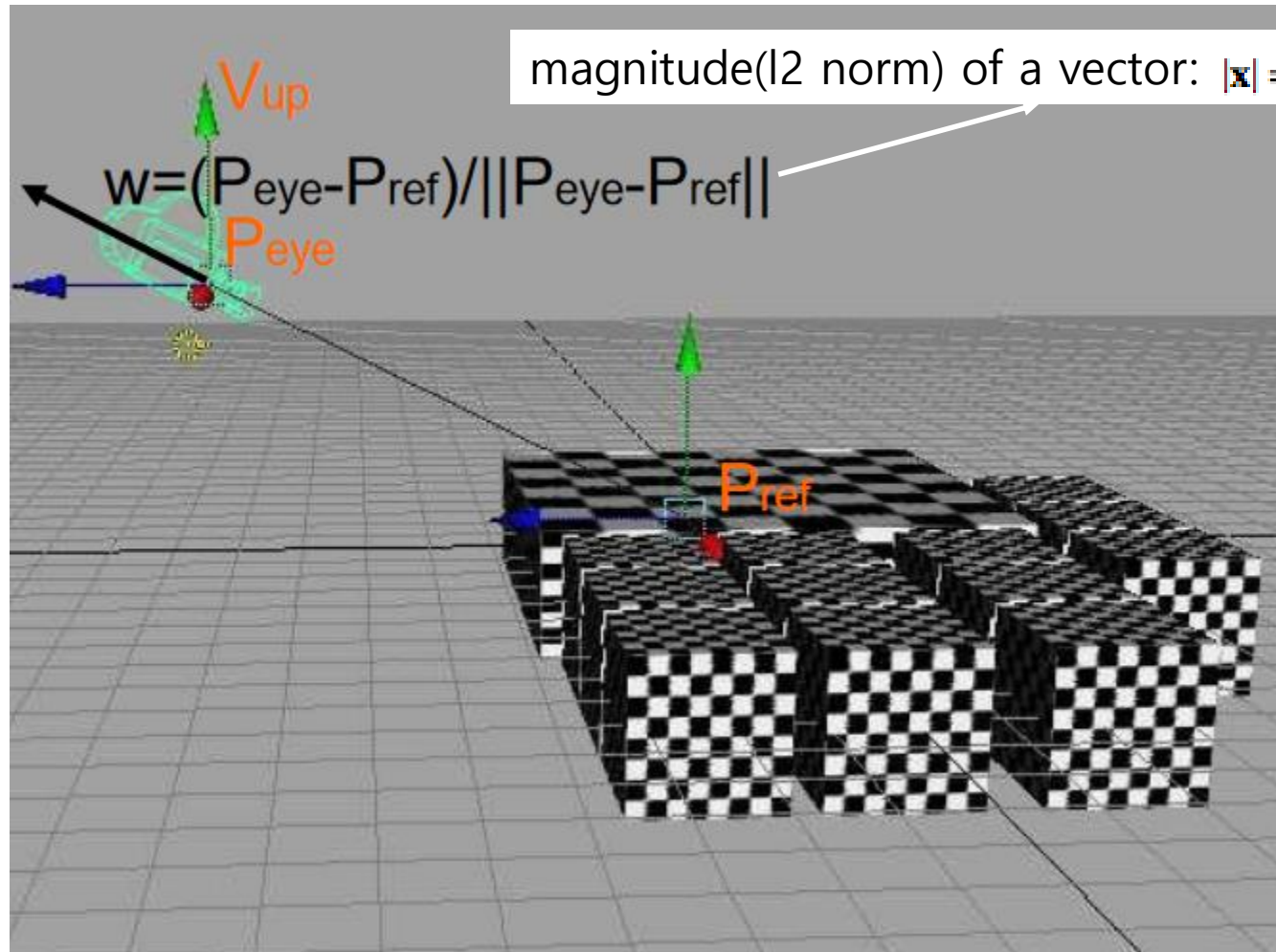
* The base images are from the slides of Prof. Karan Singh's (University of Toronto):

<http://www.dgp.toronto.edu/~karan/courses/418/>

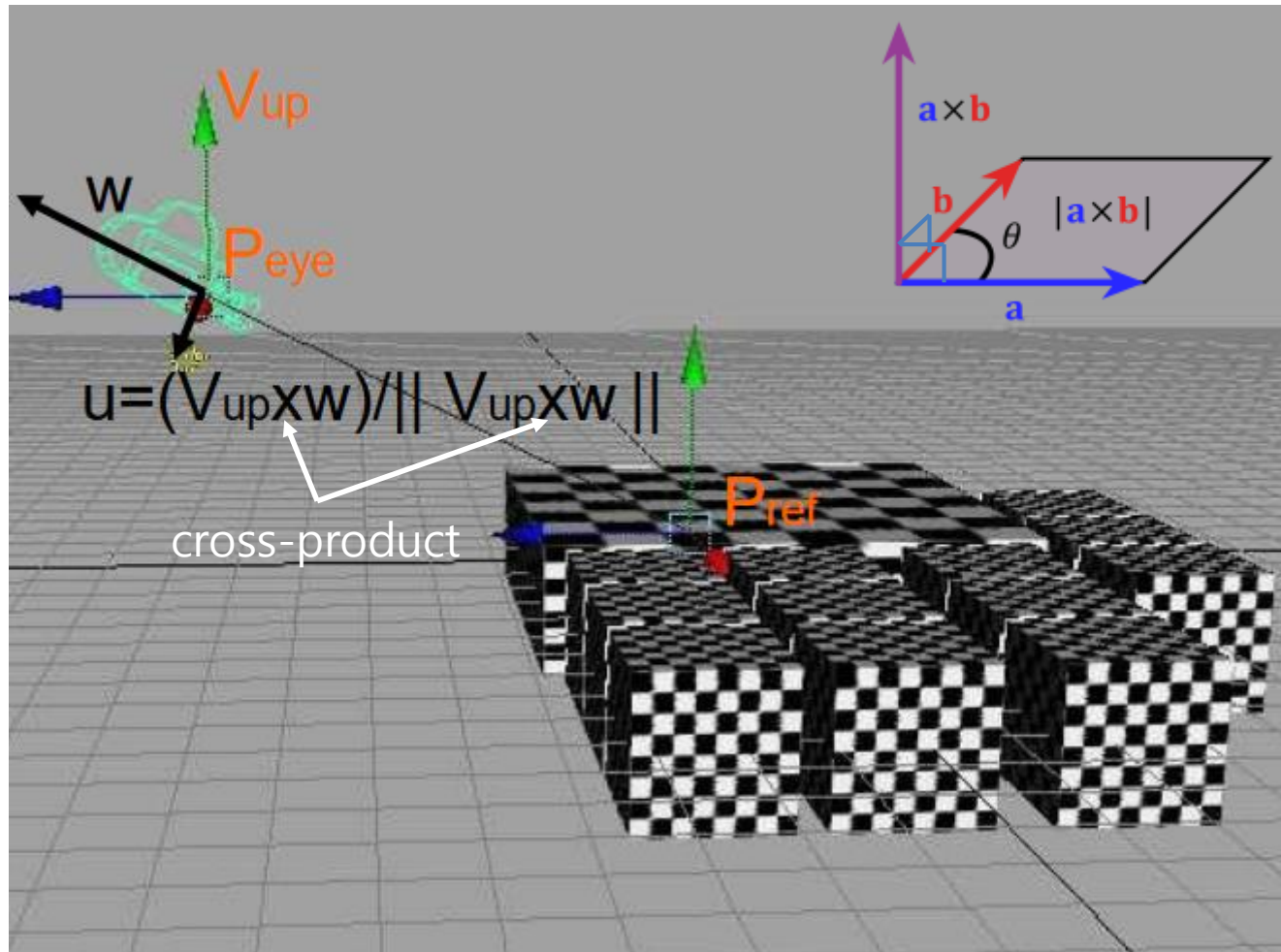
Getting origin point



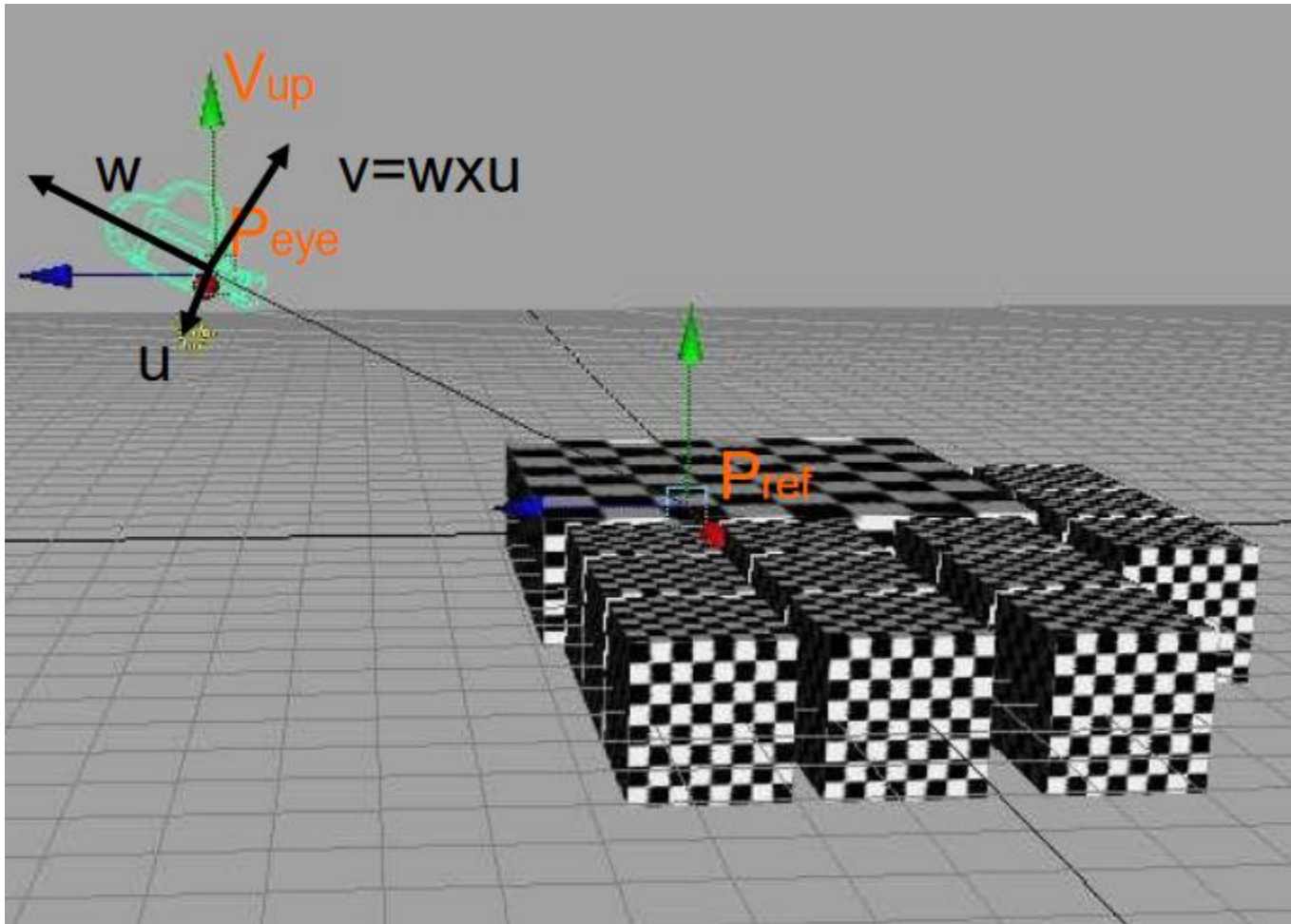
Getting “w” axis vector



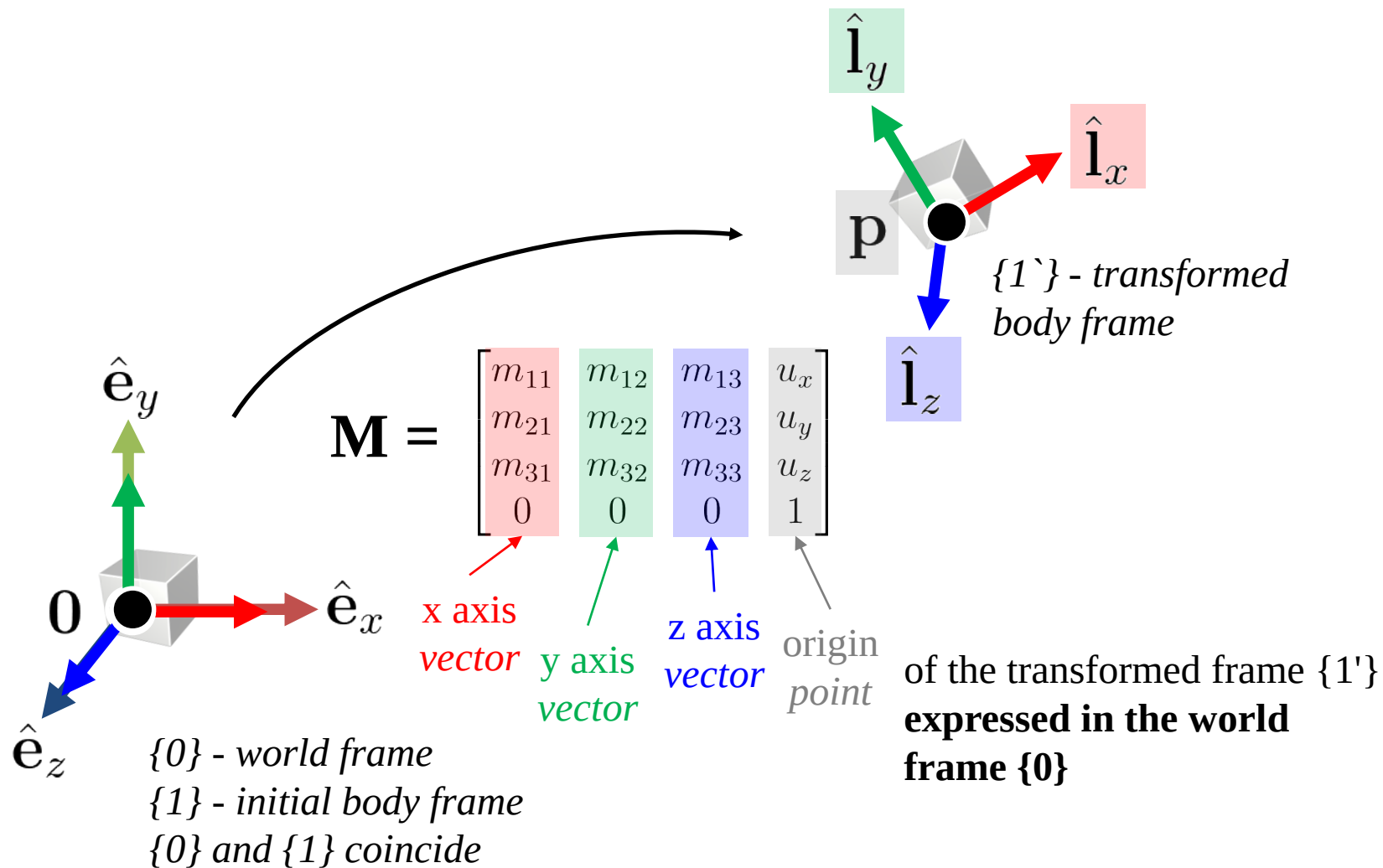
Getting “u” axis vector



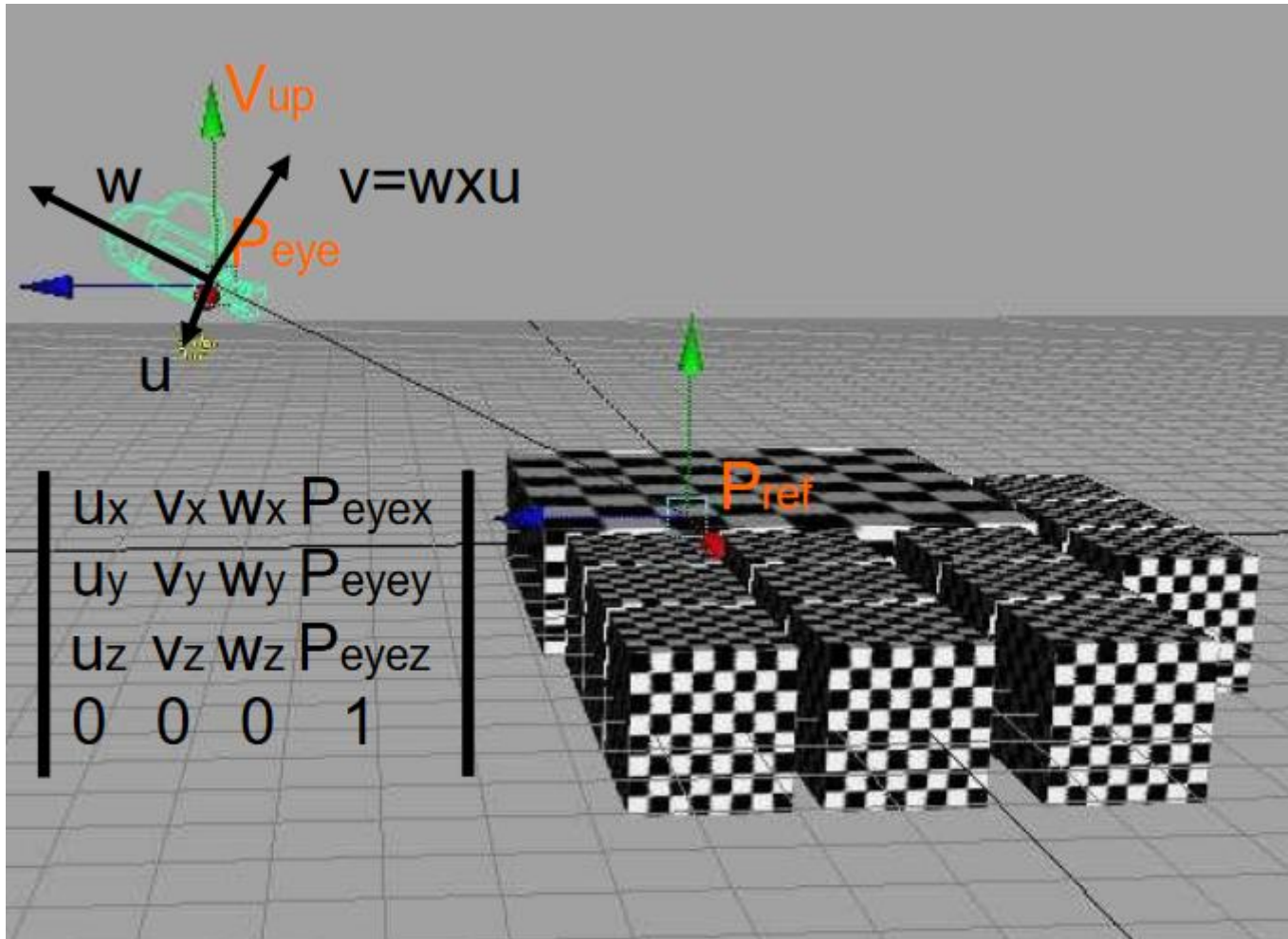
Getting “v” axis vector



Recall: 2) **Defining a New Frame** (w.r.t. World Frame)

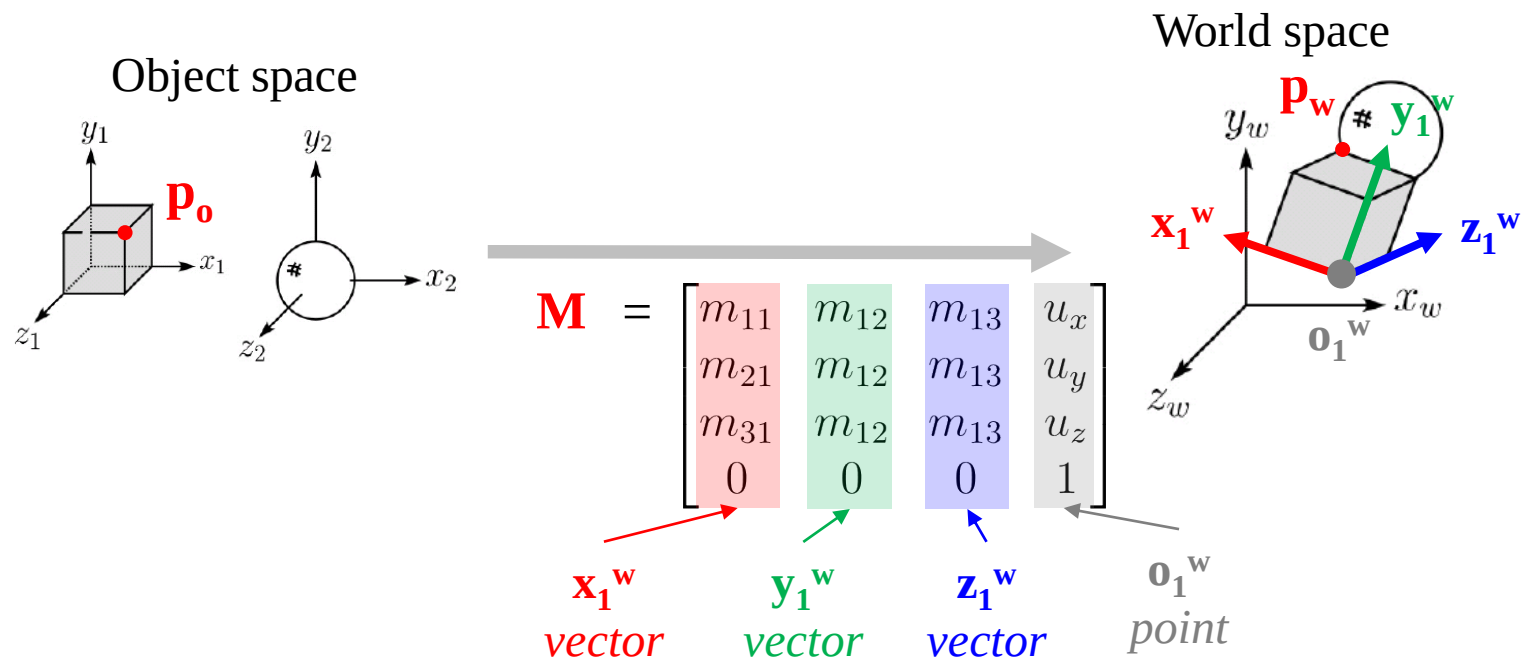


Thus, the Camera Frame is defined by



How can we get viewing matrix V from the camera frame?

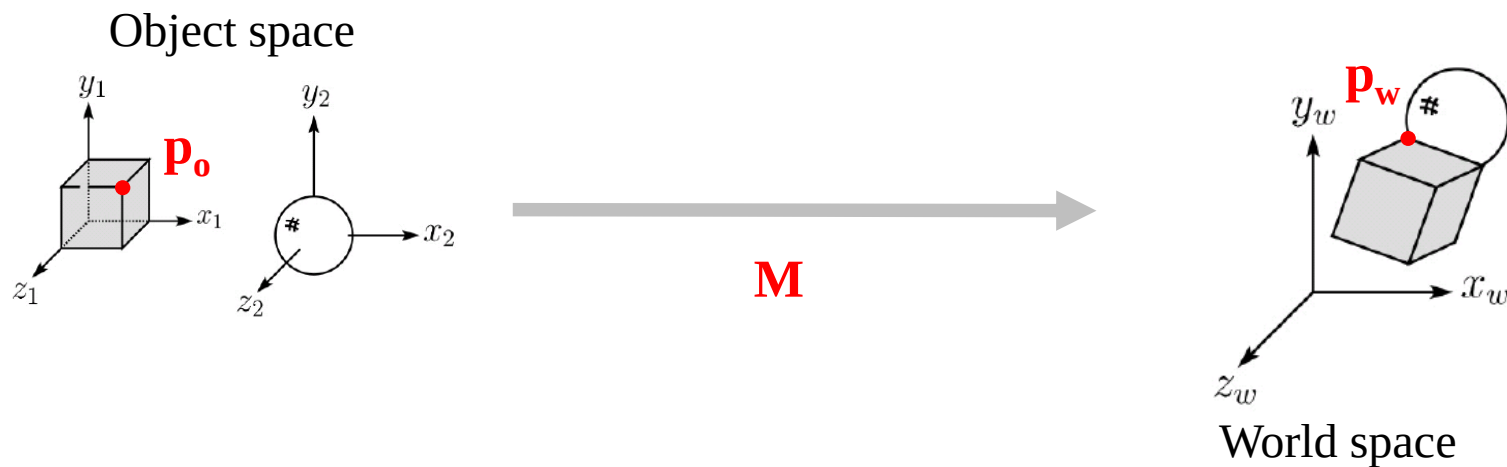
- Recall the modeling transformation:



: The axis vectors and origin point of the **object's body frame represented in the world frame**

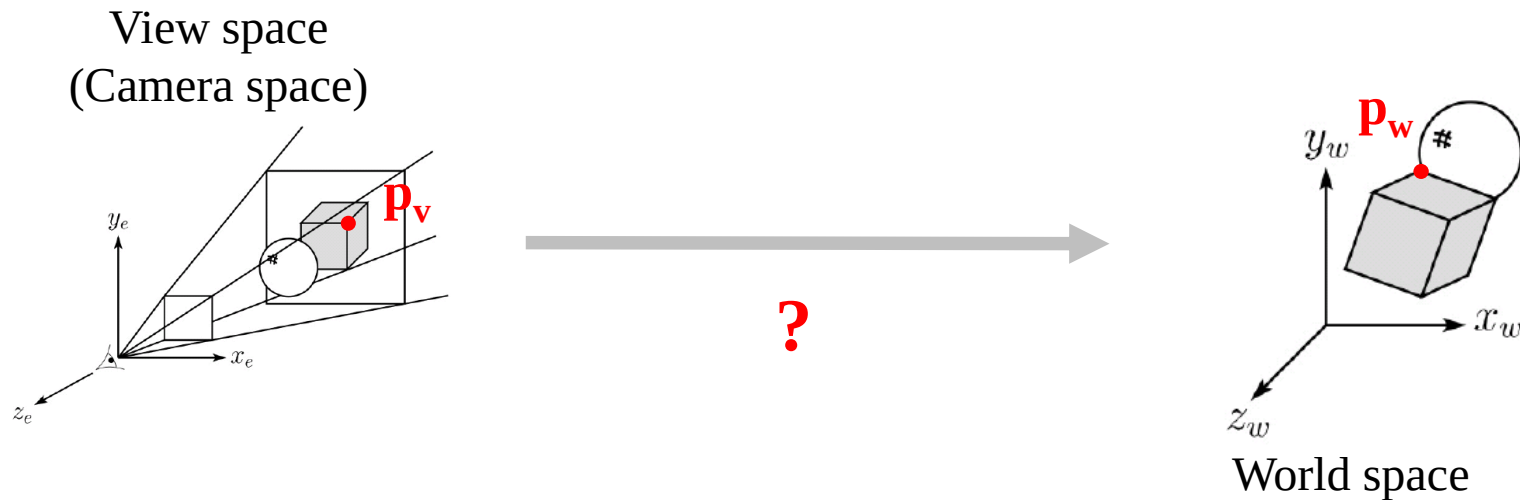
How can we get viewing matrix V from the camera frame?

- If we replace *object space* to *camera space*, what should be the transformation matrix?



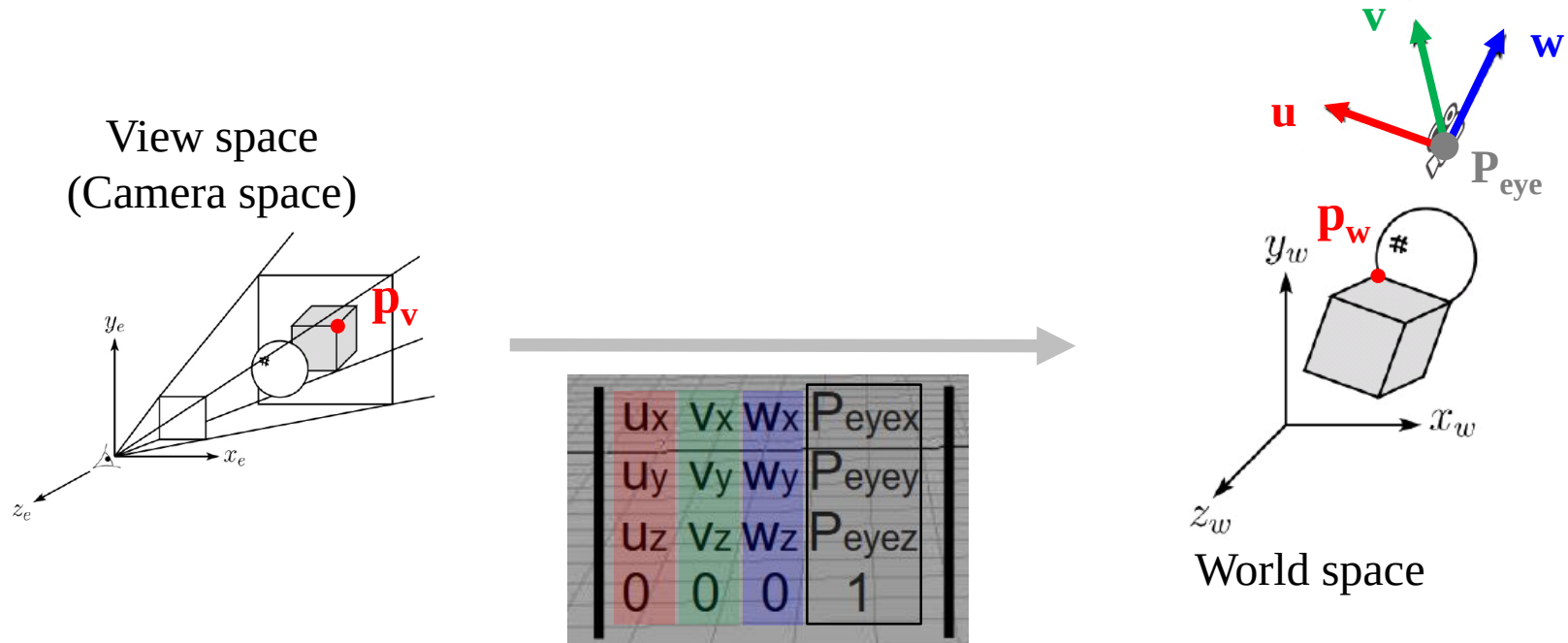
How can we get viewing matrix V from the camera frame?

- If we replace *object space* to *camera space*, what should be the transformation matrix?



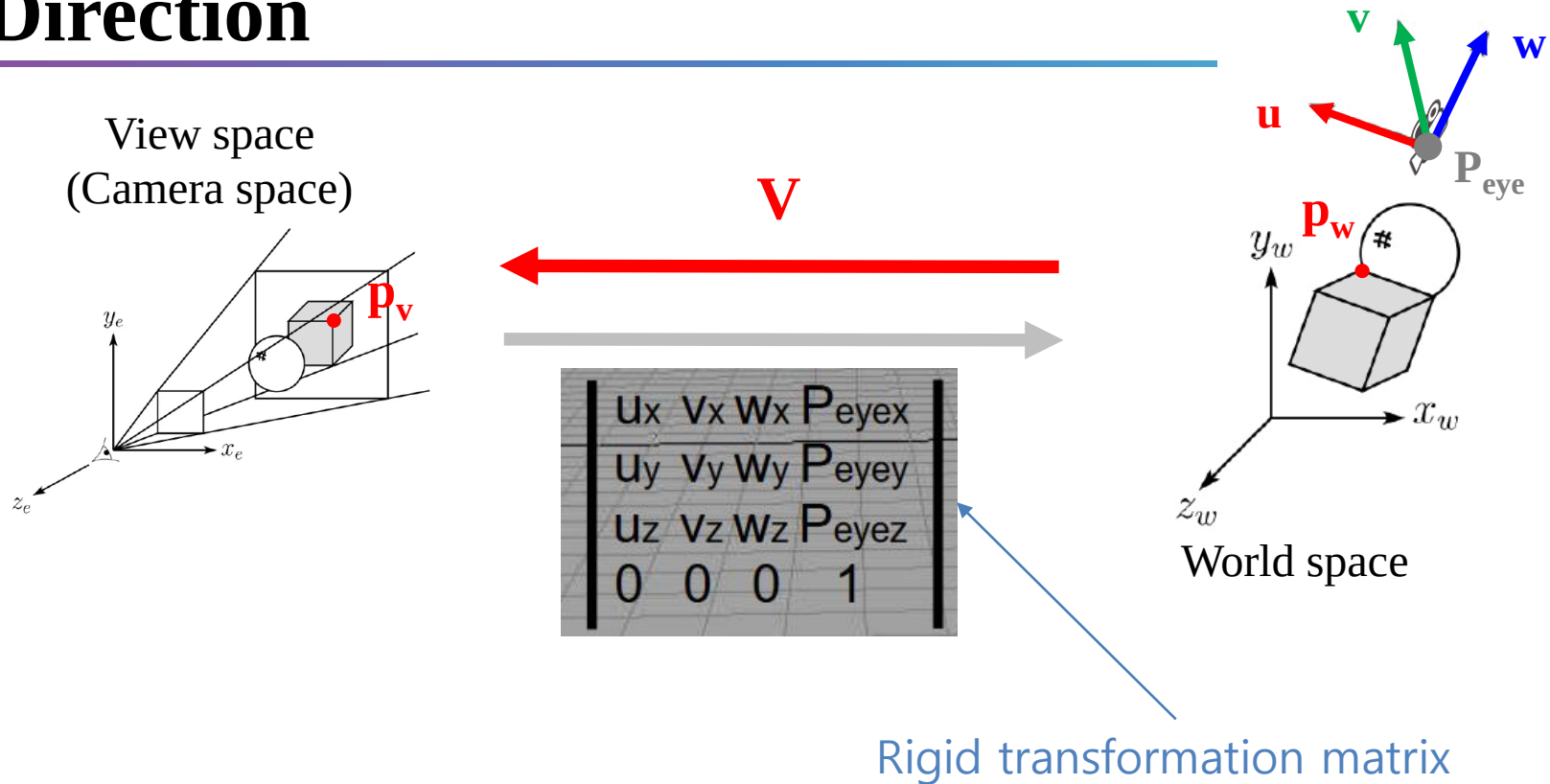
How can we get viewing matrix V from the camera frame?

- If we replace *object space* to *camera space*, what should be the transformation matrix?



: The axis vectors and origin point of the **camera frame** represented in the world frame

Viewing Transformation is the Opposite Direction



$$V = \begin{bmatrix} u_x & v_x & w_x & P_{eyex} \\ u_y & v_y & w_y & P_{eyey} \\ u_z & v_z & w_z & P_{eyez} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

Inverting Rigid Transformation Matrix

(R : 3×3 rotation matrix, t : 3×1 translation vector)

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \Rightarrow T^{-1} = \begin{bmatrix} R^T & -R^T t \\ 0 & 1 \end{bmatrix}$$

- For our camera frame matrix,

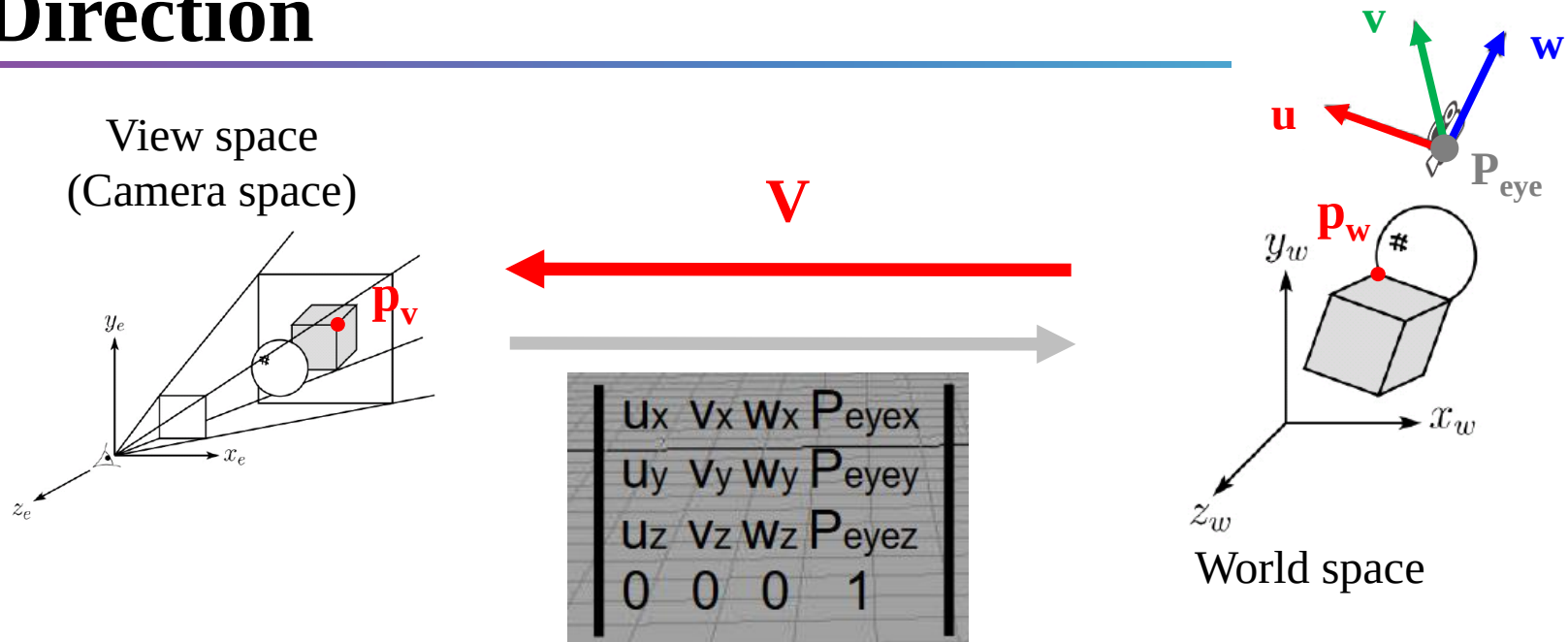
$$R = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} P_{eyex} \\ P_{eyey} \\ P_{eyez} \end{bmatrix}$$

$$\begin{bmatrix} u_x & v_x & w_x & P_{eyex} \\ u_y & v_y & w_y & P_{eyey} \\ u_z & v_z & w_z & P_{eyez} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow R^T = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix}$$

$$-R^T \cdot \mathbf{t} = - \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix} \begin{bmatrix} P_{eyex} \\ P_{eyey} \\ P_{eyez} \end{bmatrix} = \begin{bmatrix} -(\mathbf{u} \cdot \mathbf{t}) \\ -(\mathbf{v} \cdot \mathbf{t}) \\ -(\mathbf{w} \cdot \mathbf{t}) \end{bmatrix}$$

Viewing Transformation is the Opposite Direction

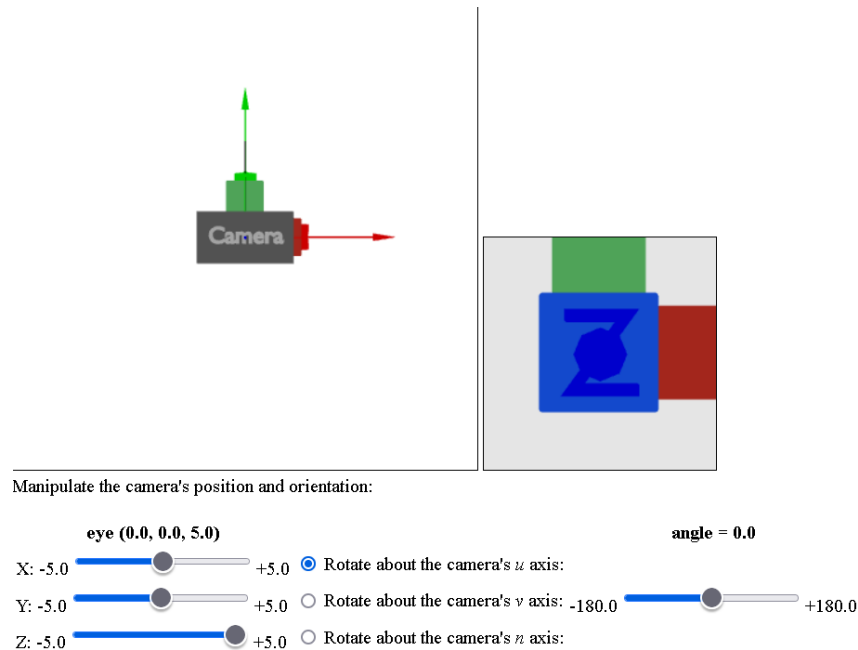


$$V = \begin{bmatrix} u_x & v_x & w_x & P_{eyex} \\ u_y & v_y & w_y & P_{eyey} \\ u_z & v_z & w_z & P_{eyez} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{p}_{eye} \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{p}_{eye} \\ w_x & w_y & w_z & -\mathbf{w} \cdot \mathbf{p}_{eye} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Defining Camera Frame 2 - Translate & Rotate

- The *LookAt* method is just one way to define a camera.
- Alternatively, we can define the camera using **translation and rotation**.
 - In other words, the camera can be described as a **rigid transformation**.
- Instead of computing the camera frame step by step like in *LookAt*, we can directly define it using rigid transformation matrices.

[Demo] Translate & Rotate Camera

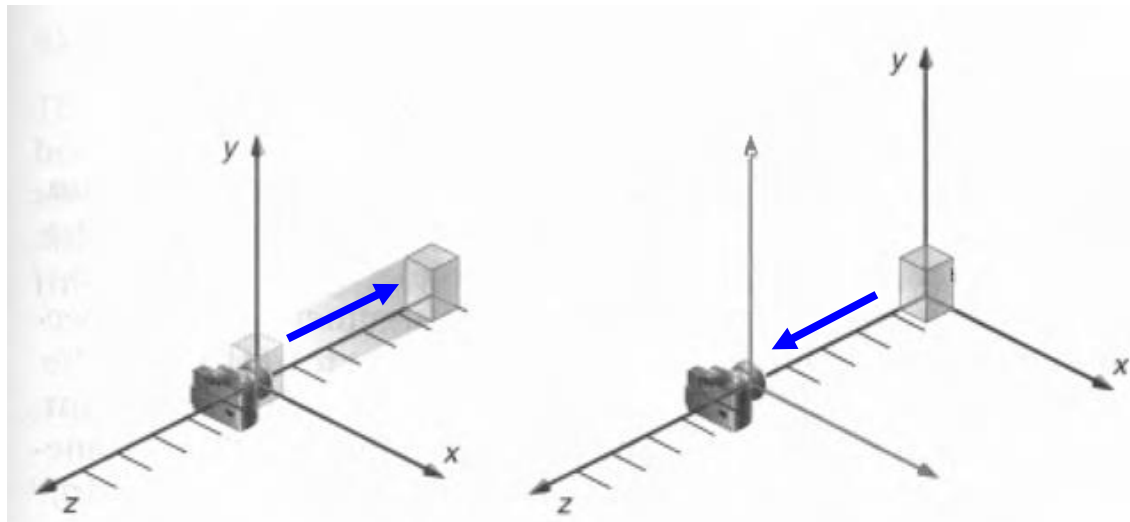


http://learnwebgl.brown37.net/07_cameras/camera_trunk_axes/camera_trunk_axes.html

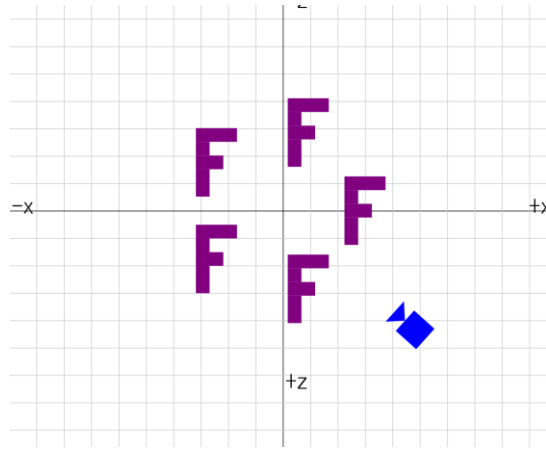
- Observe the 3D scene (left) and rendered view (right) while translating the camera with "eye" sliders and rotating it with axis selection and "angle" slider.

Moving Camera vs. Moving World

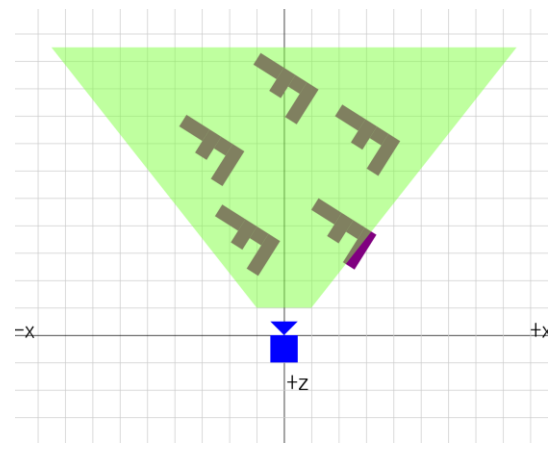
- Moving the camera is equivalent to **moving the world in the opposite way**.
- Translate camera by $(1, 0, 2)$ == Translate world by $(-1, 0, -2)$
- Rotate camera by 60° about y == Rotate world by -60° about y



[Demo] Moving Camera vs. Moving World



<https://webglfundamentals.org/webgl/lessons/resources/camera-move-camera.html?mode=0>



<https://webglfundamentals.org/webgl/lessons/resources/camera-move-camera.html?mode=2>

- (Left) Moving camera
- (Right) Moving world

Lab Session

- Now, let's start the lab today.