Computer Graphics

5 - Vertex Processing 1

Yoonsang Lee
Hanyang University

Spring 2023

Outline

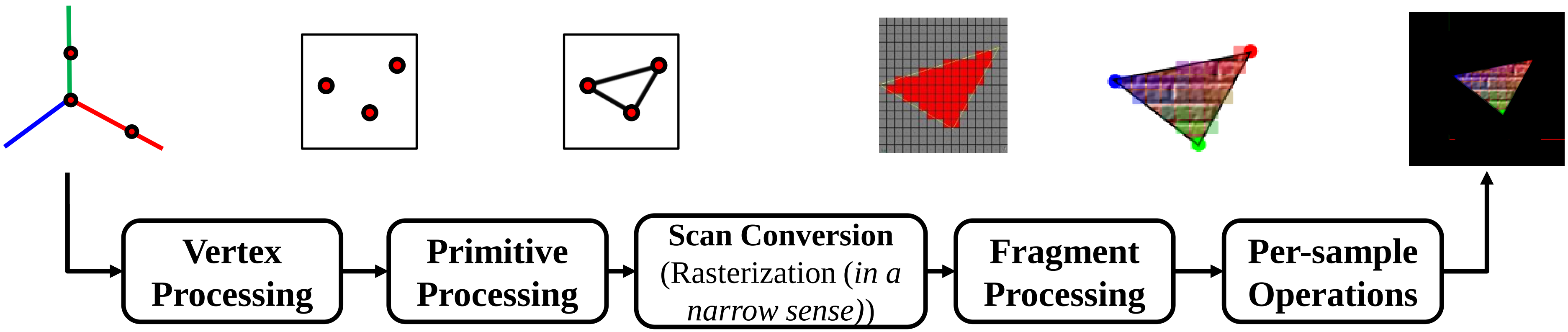
- Rasterization Pipeline & Vertex Processing

- Modeling Transformation

- Viewing Transformation

Rasterization Pipeline & Vertex Processing

Recall: Rasterization Pipeline



Vertex Processing: transforms vertices to screen space

Primitive Processing: assembles polygons

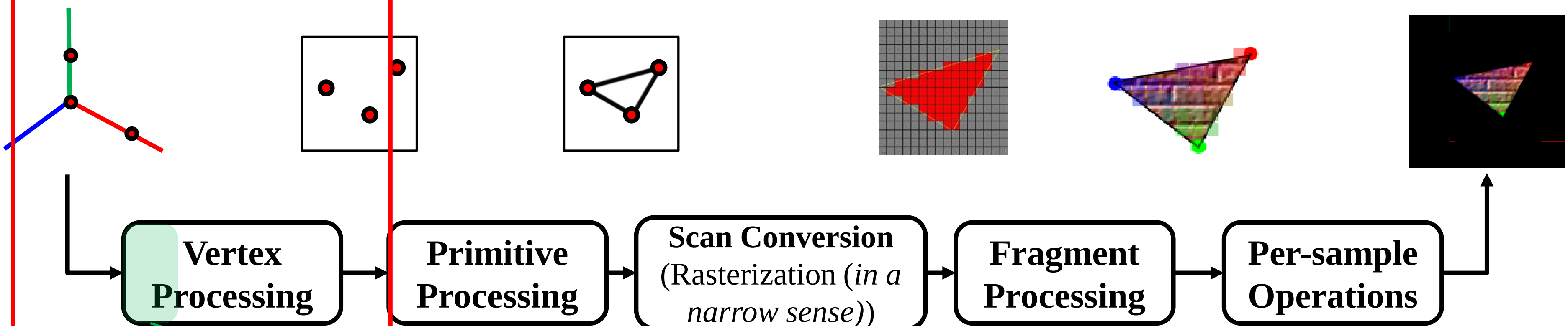
Scan Conversion: converts each polygon into a set of fragments

Fragment Processing: determines color of each fragment with light & texture

Per-sample Operations: depth test, alpha blending, …

* The image is from https://graphicscompendium.com/intro/01-graphics-pipeline

Recall: Rasterization Pipeline



: transform vertices to screen space

→ performs a sequence of vertex transformations

What we've been done so far

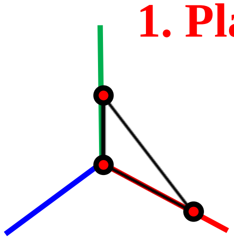
→ What we'll see today & next lecture

Vertex Processing

Vertex positions in each object's body frame

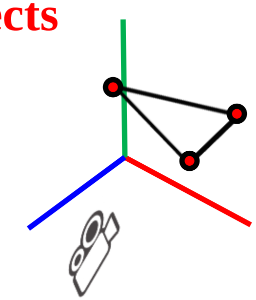
Vertex positions in world frame

Vertex positions in 2D viewport

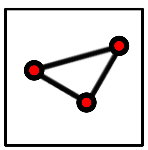


1. Placing objects

M →



?



$\mathbf{p}_1$
$\mathbf{p}_2$
$\mathbf{p}_3$

$\mathbf{Mp}_1$
$\mathbf{Mp}_2$
$\mathbf{Mp}_3$

We need to introduce the concept of a "camera" looking at the "scene".

Then the next steps are:

2. Placing a "camera"
3. Selecting its "lens"
4. Displaying on a "cinema screen"

In Terms of CG Transformation,

- 1. Placing objects

→ **Modeling transformation**

- 2. Placing a "camera"

→ **Viewing transformation**

- 3. Selecting its "lens"

→ **Projection transformation**

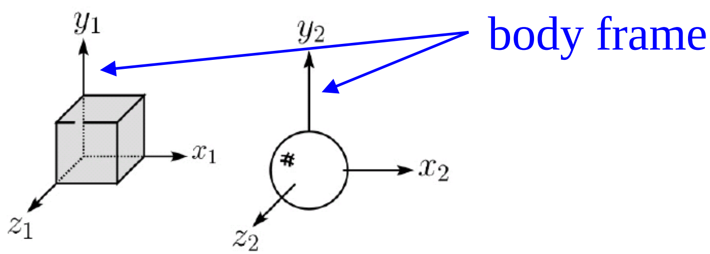
- 4. Displaying on a "cinema screen"

→ **Viewport transformation**

- All these transformations just work by **matrix multiplications**!

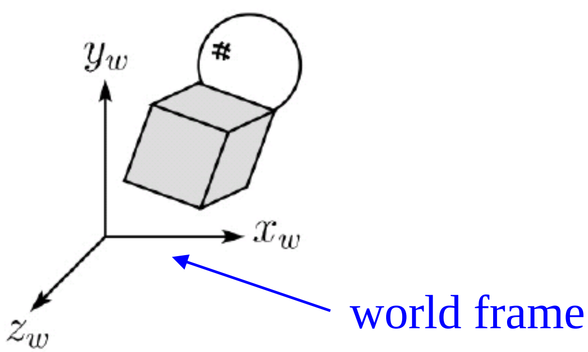
Vertex Processing (Transformation Pipeline)

Object space



Translate, scale, rotate, ... any affine transformations

(What we've covered in prev. lectures)

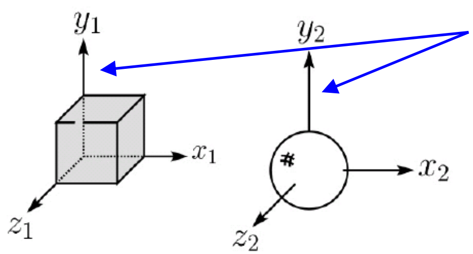


World space

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

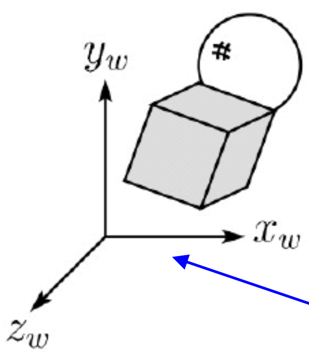
Vertex Processing (Transformation Pipeline)

Object space



body frame

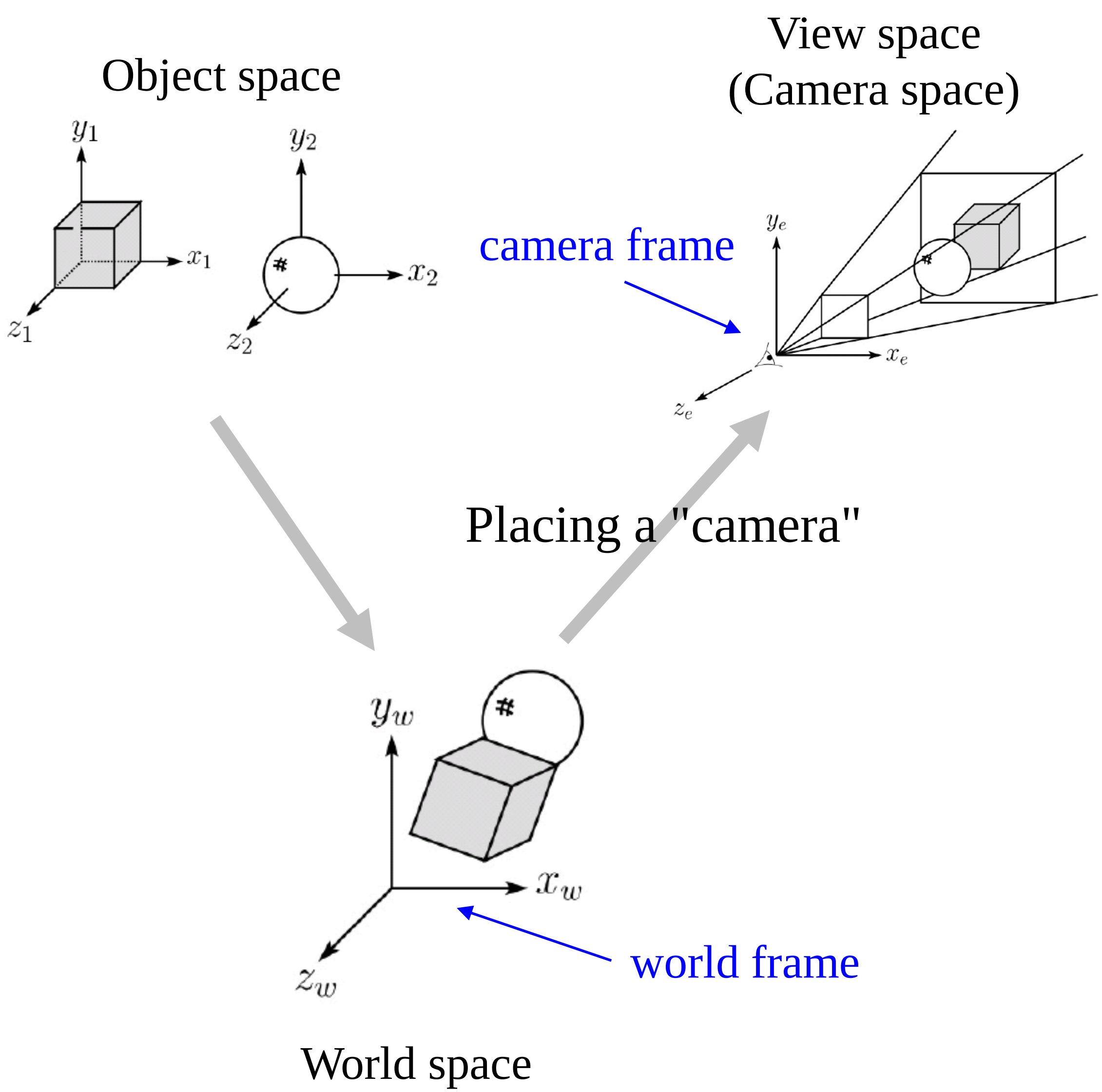
Modeling transformation



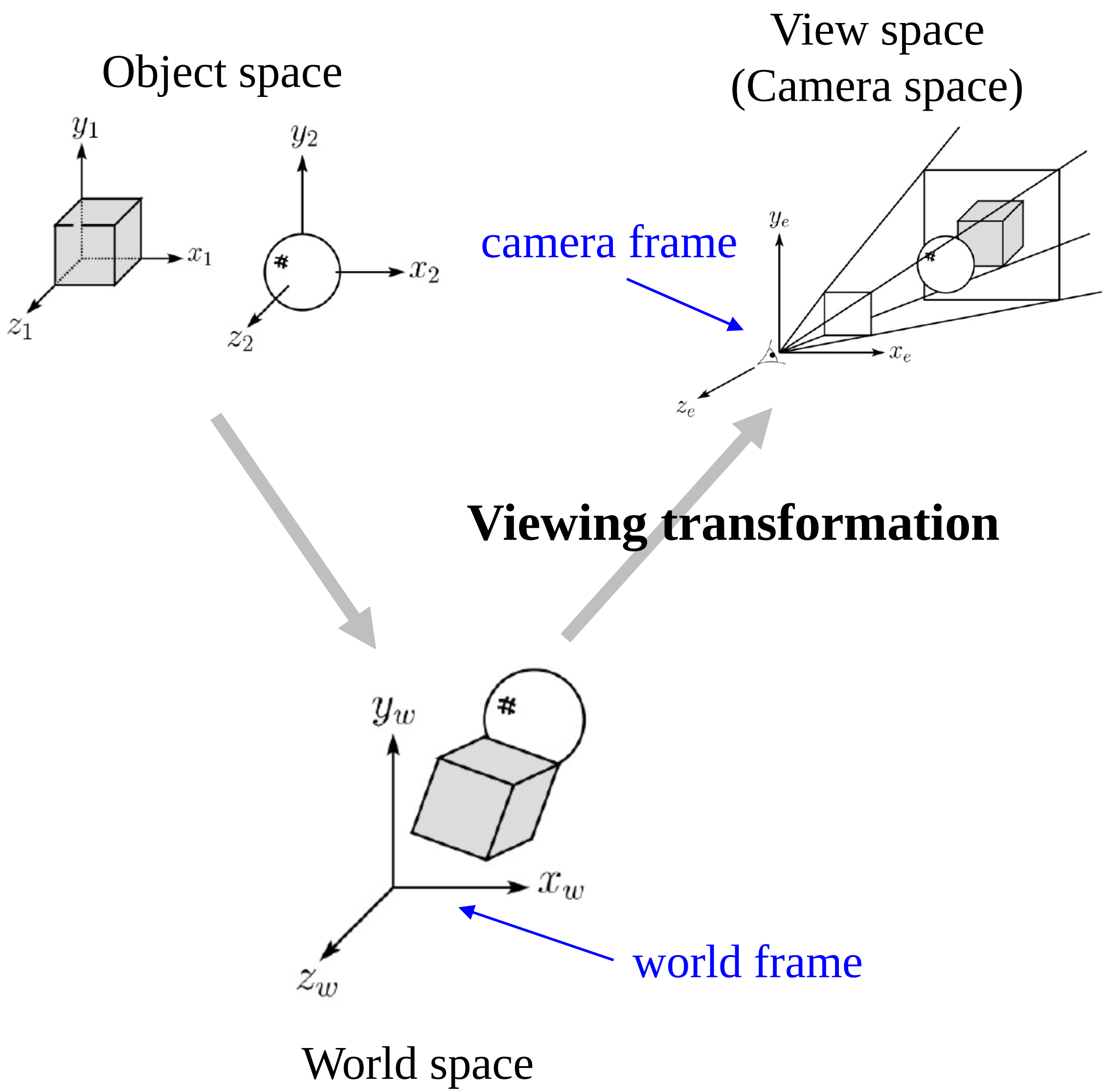
world frame

World space

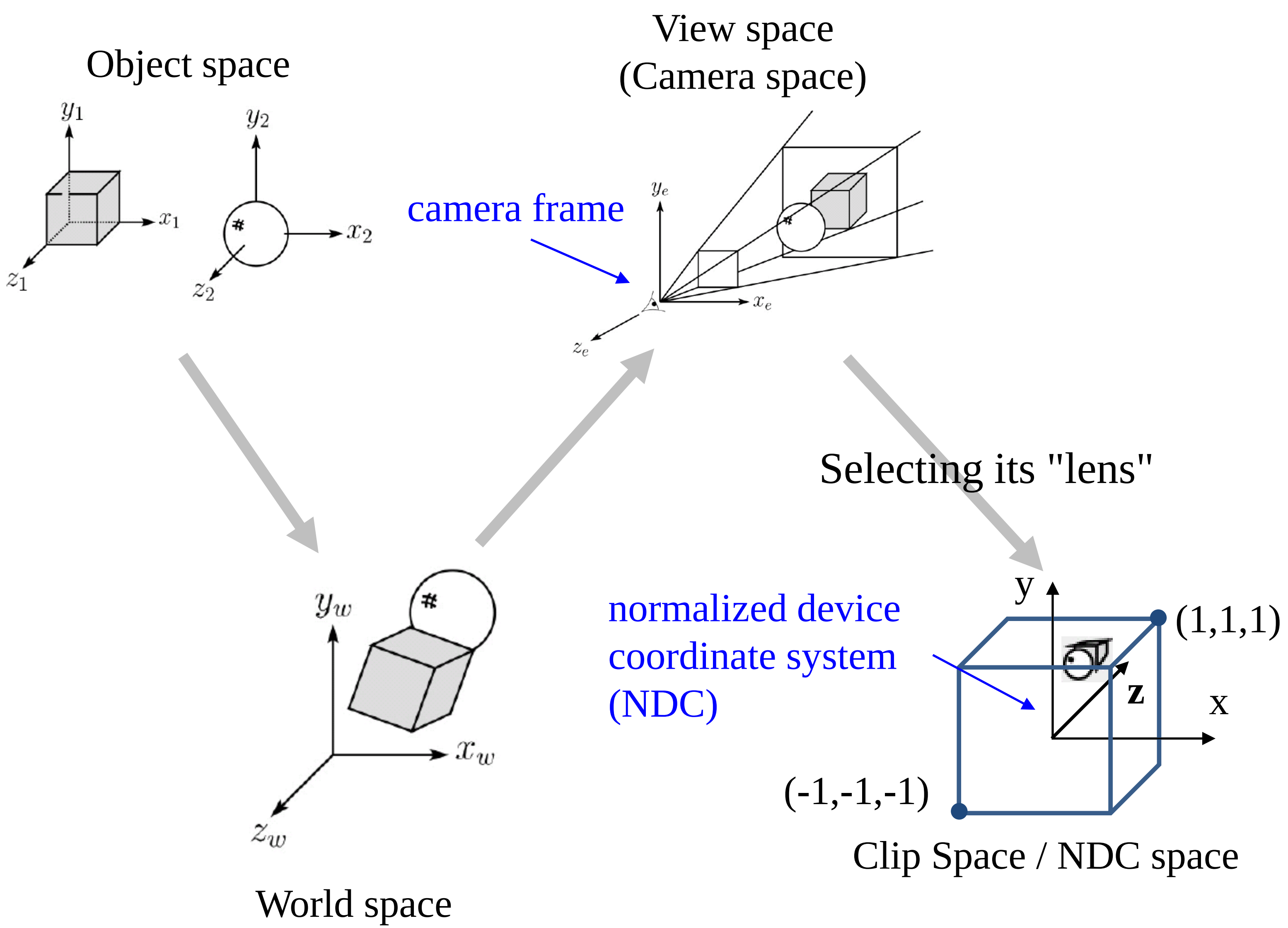
Vertex Processing (Transformation Pipeline)



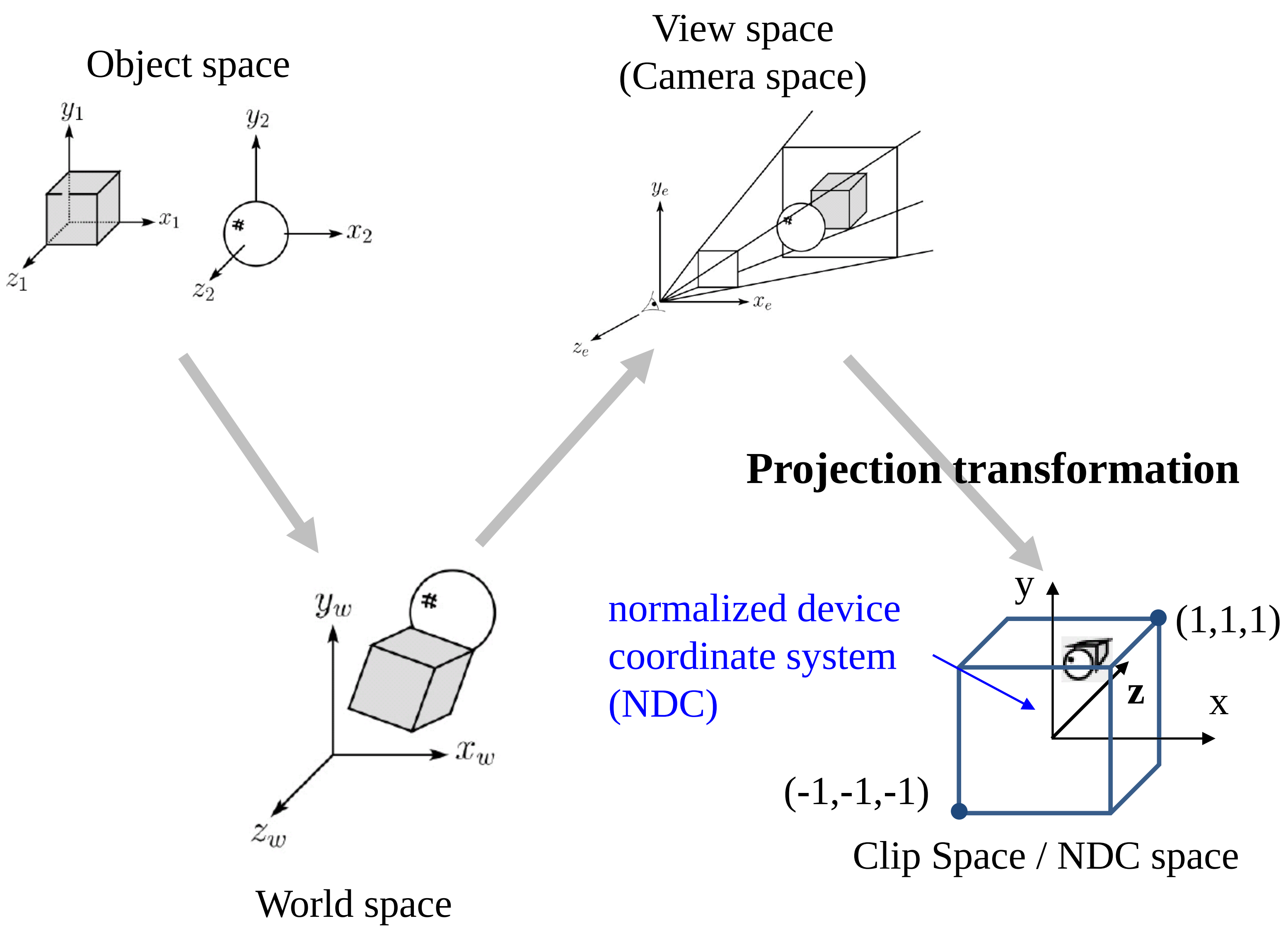
Vertex Processing (Transformation Pipeline)



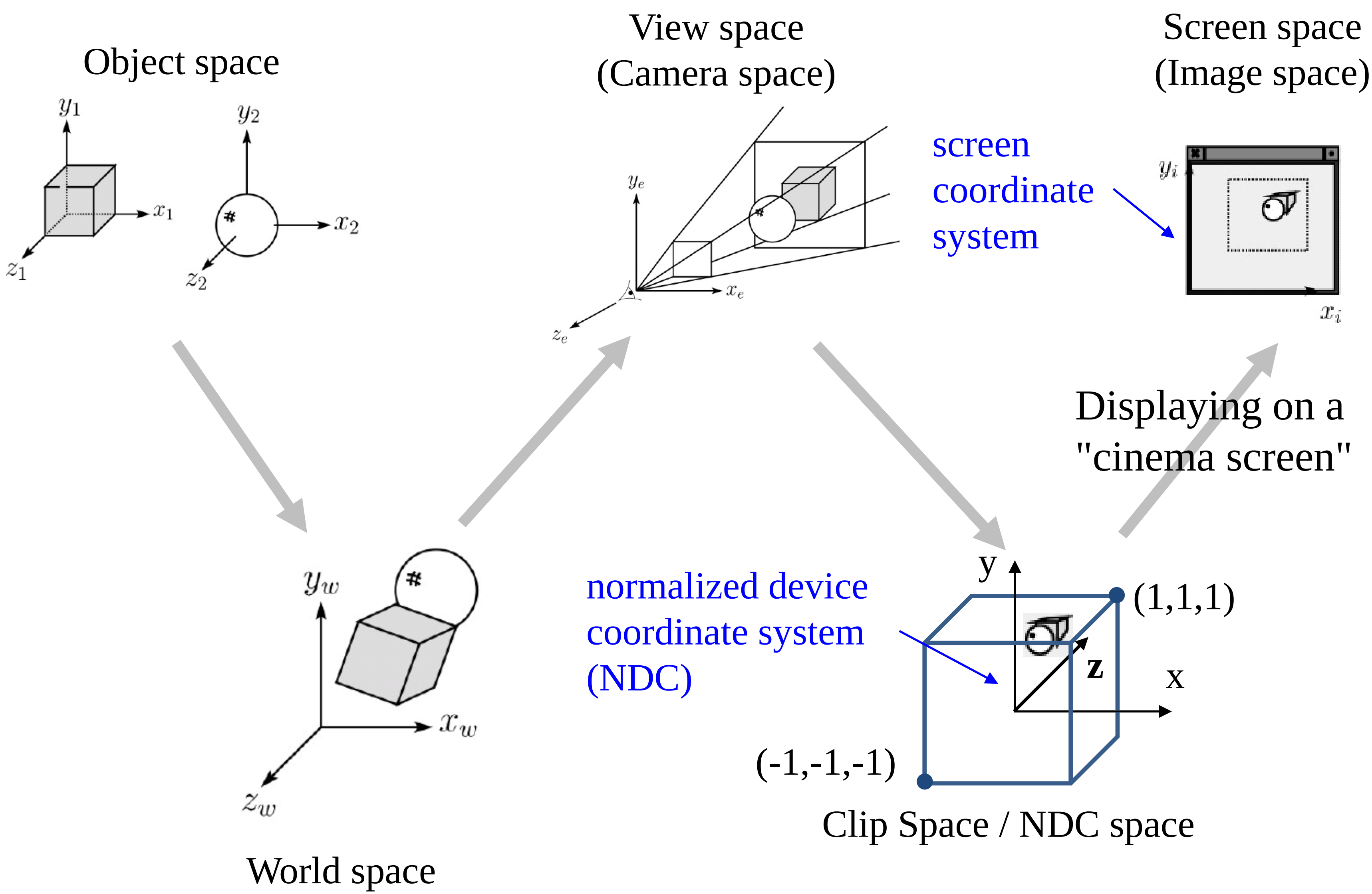
Vertex Processing (Transformation Pipeline)



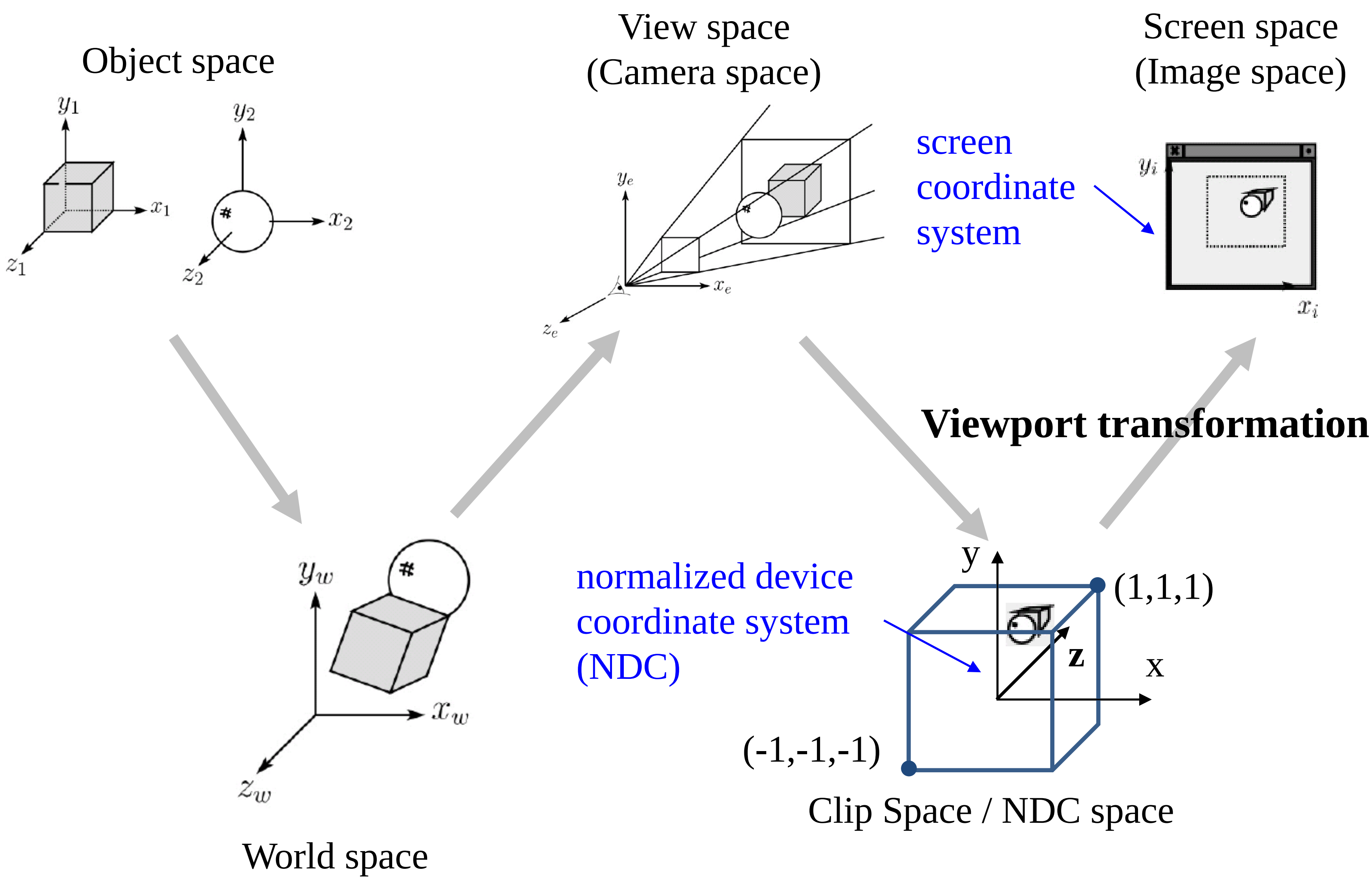
Vertex Processing (Transformation Pipeline)



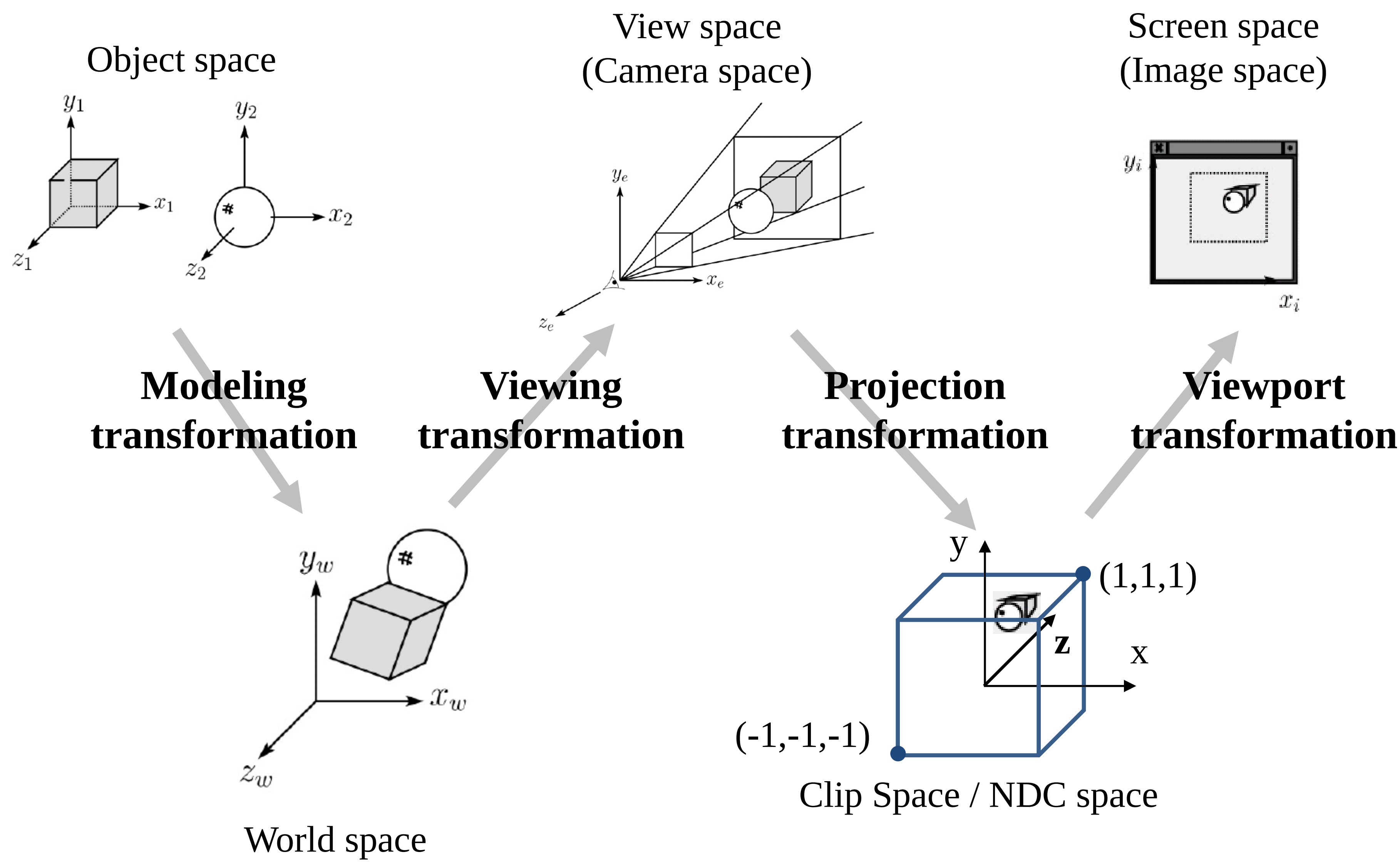
Vertex Processing (Transformation Pipeline)



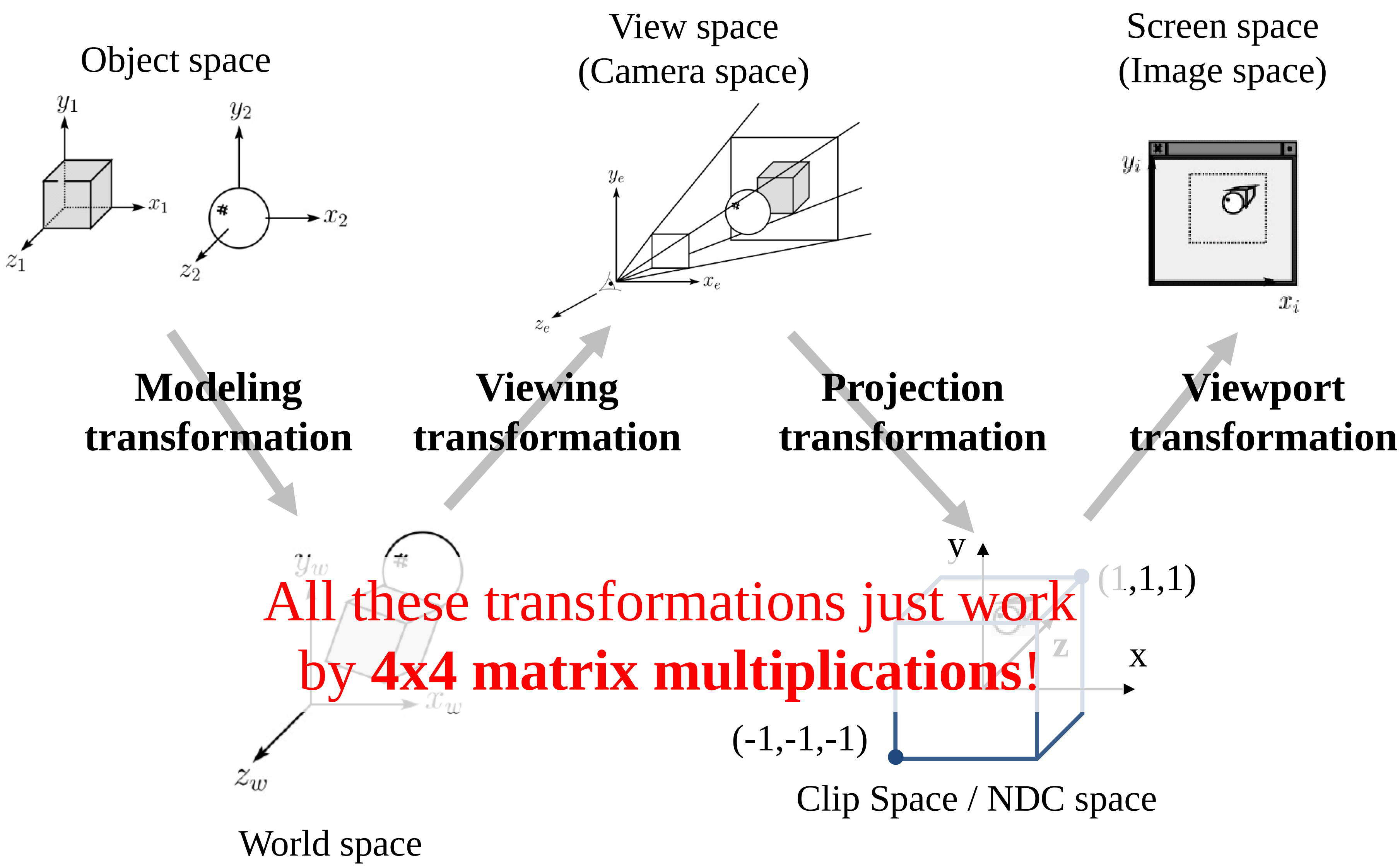
Vertex Processing (Transformation Pipeline)



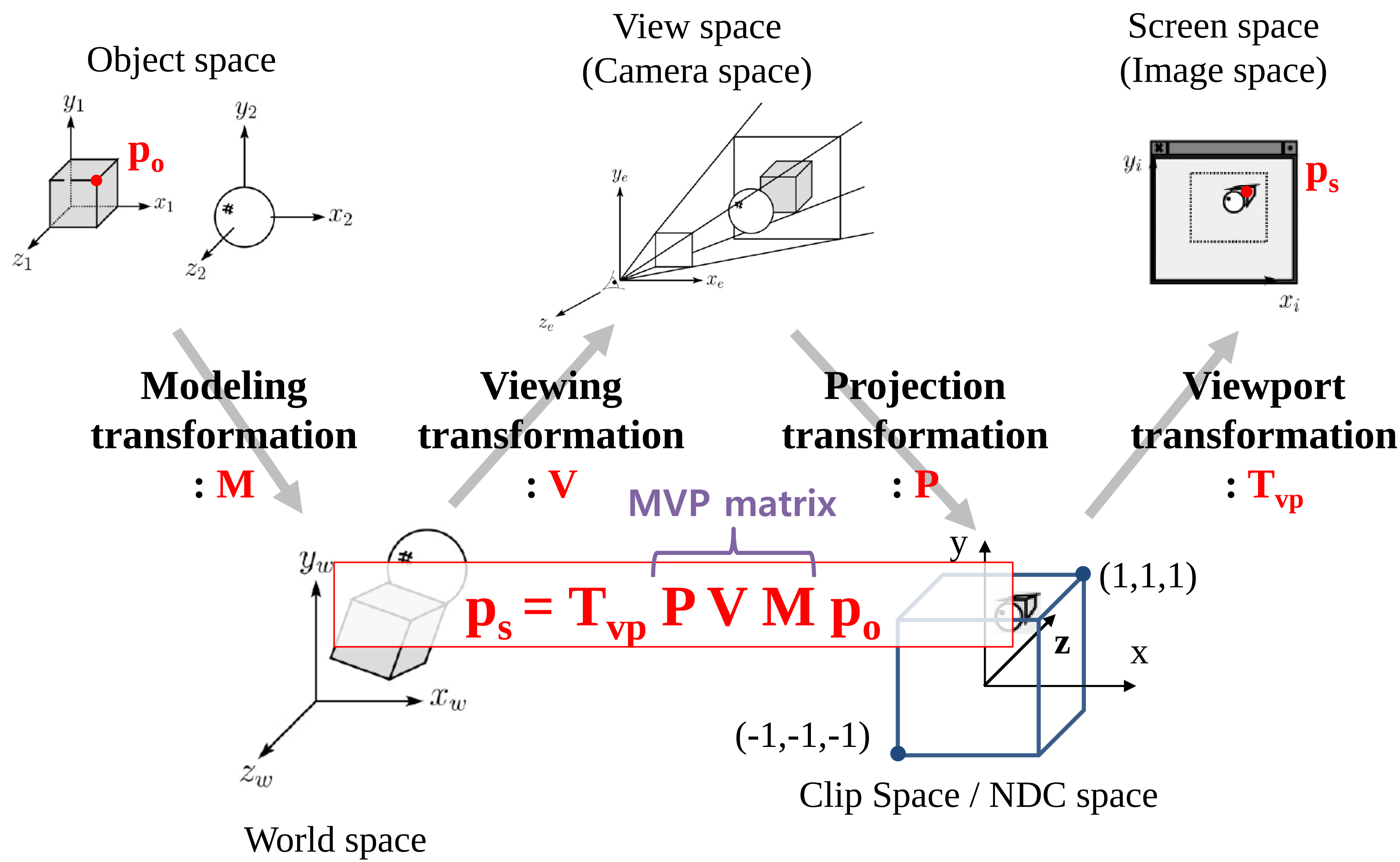
Vertex Processing (Transformation Pipeline)



Vertex Processing (Transformation Pipeline)

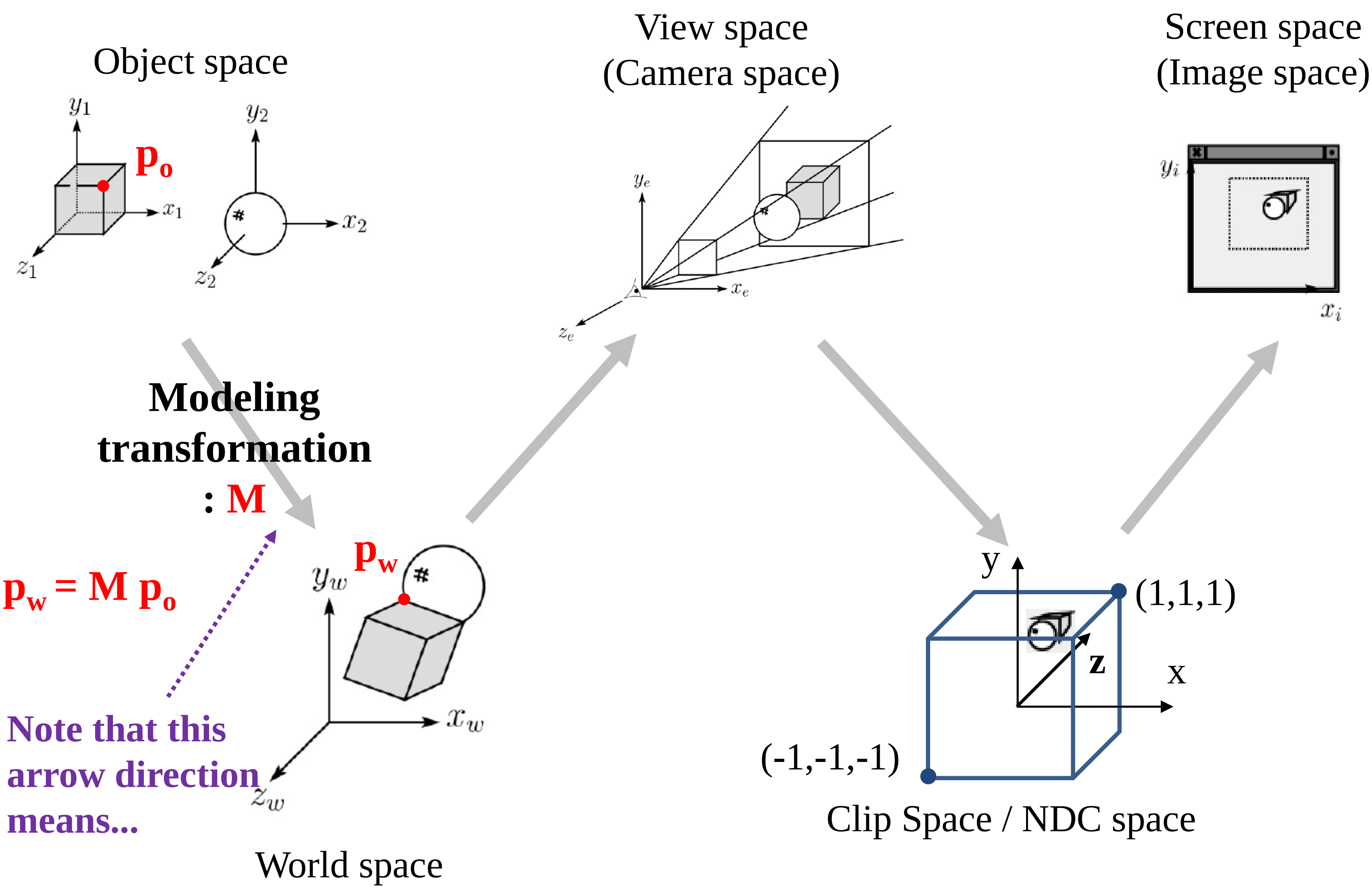


Vertex Processing (Transformation Pipeline)

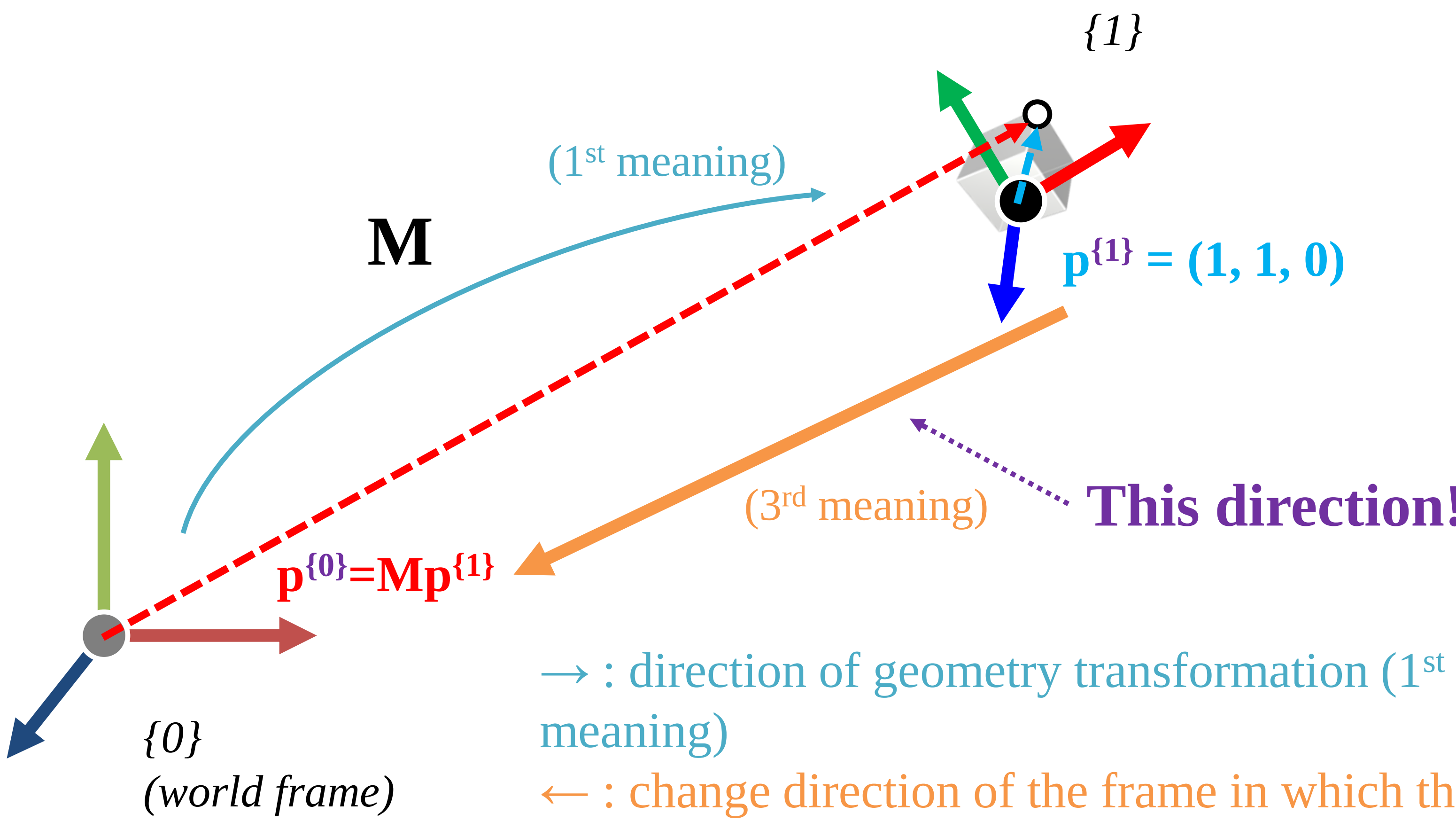


Modeling Transformation

Modeling Transformation



Recall: Directions of the "arrow"

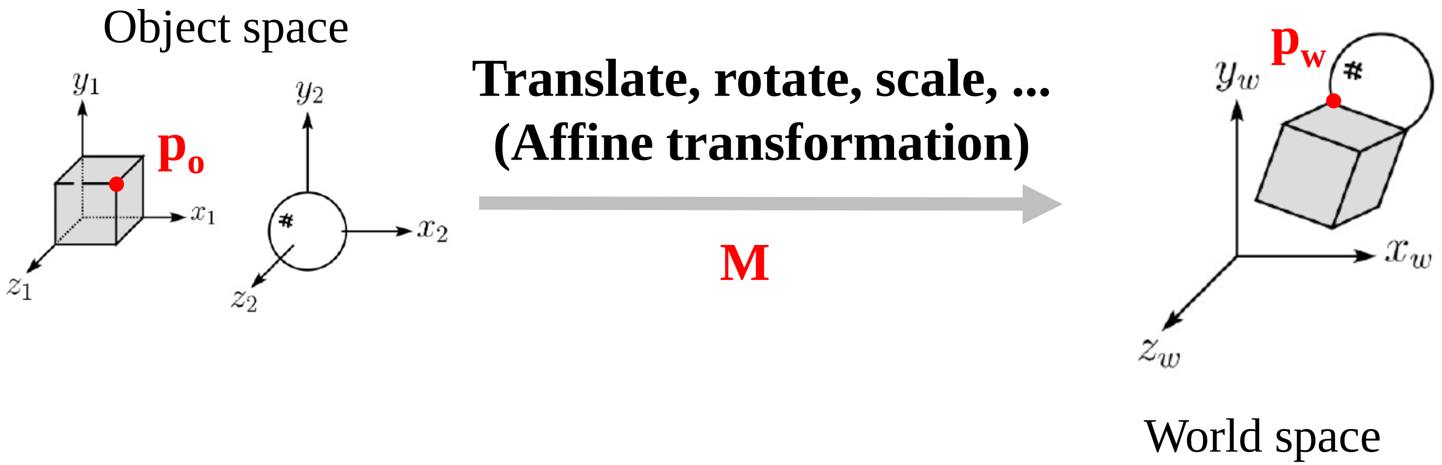


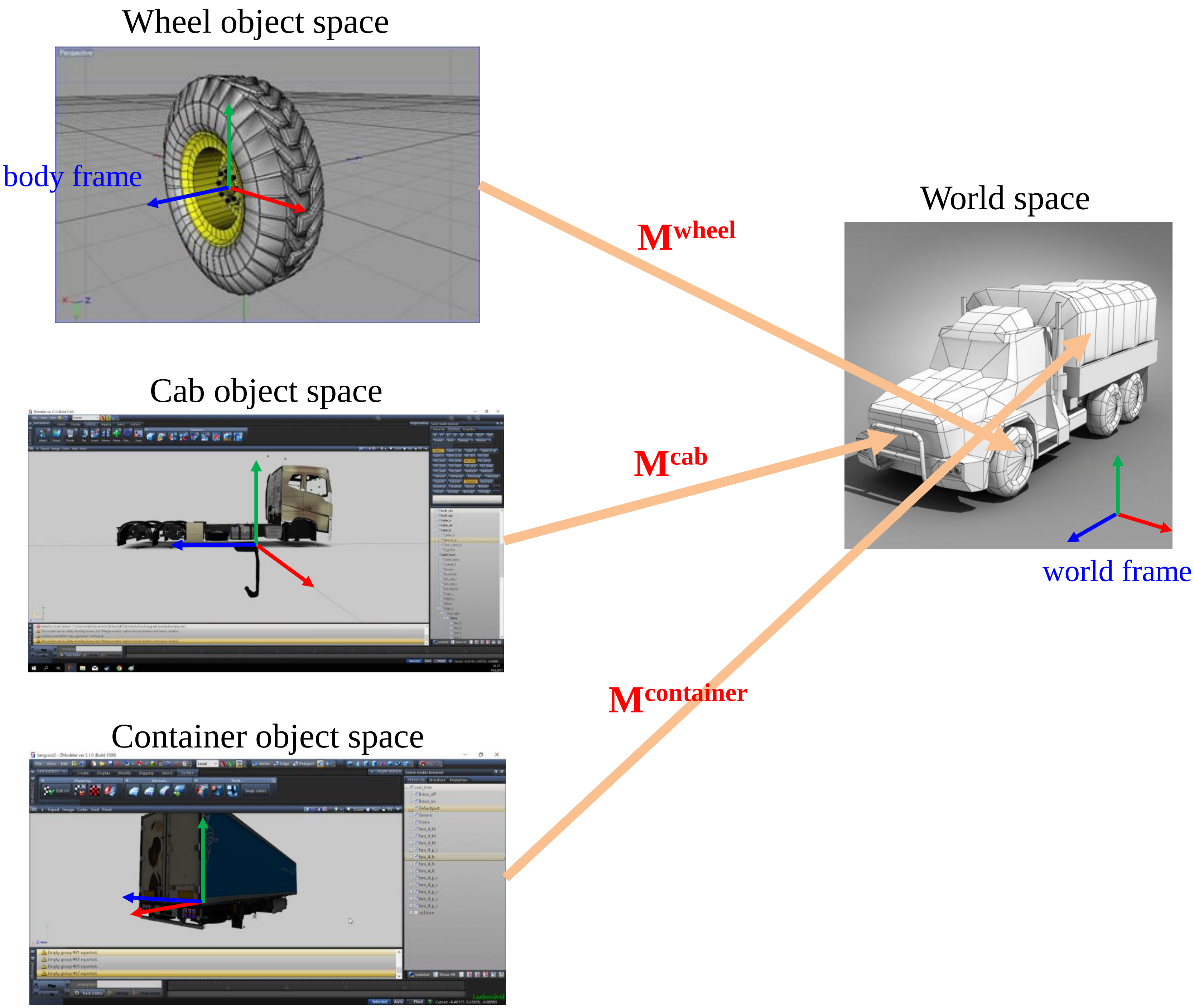
→ : direction of geometry transformation (1st meaning)

← : change direction of the frame in which the point is described (3rd meaning) - {1} → {0}

Modeling Transformation

- An object would originally have been in the **object's body frame**.
- Transformation from object space to world space is called the *Modeling matrix*, **M.**
- This is a composite affine transformations that we've covered so far.




Wheel object space
body frame
World space
Mwheel
Cab object space
Mcab
world frame
Mcontainer
Container object space

Quiz 1

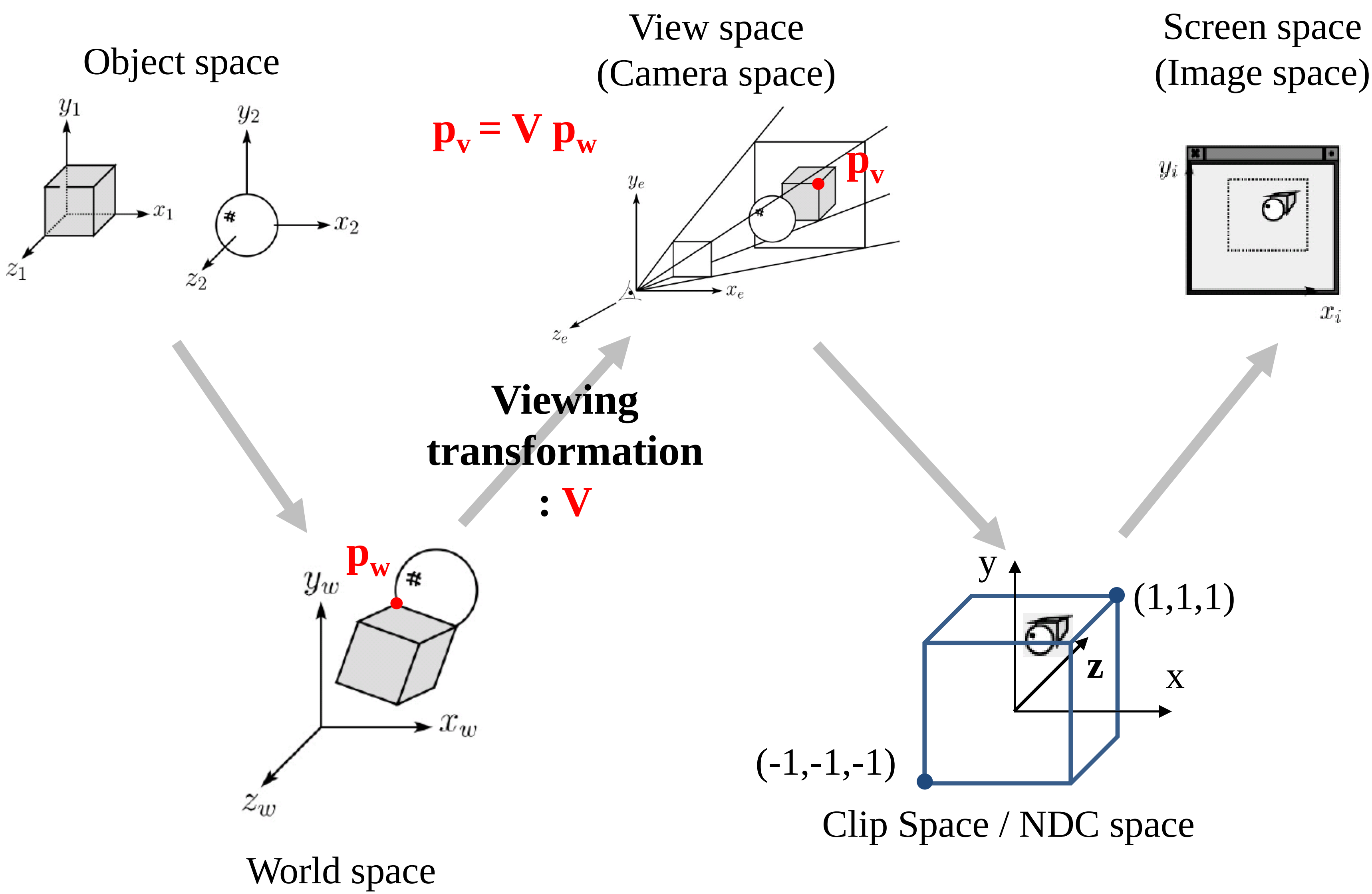
- Go to https://www.slido.com/
- Join #**cg-ys**
- Click "Polls"

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2021123456: 4.0**

- Note that your quiz answer must be submitted **in the above format** to receive a quiz score!

Viewing Transformation

Viewing Transformation



Recall that...

- 1. Placing objects

→ **Modeling transformation**

- 2. Placing a "camera"

→ **Viewing transformation**

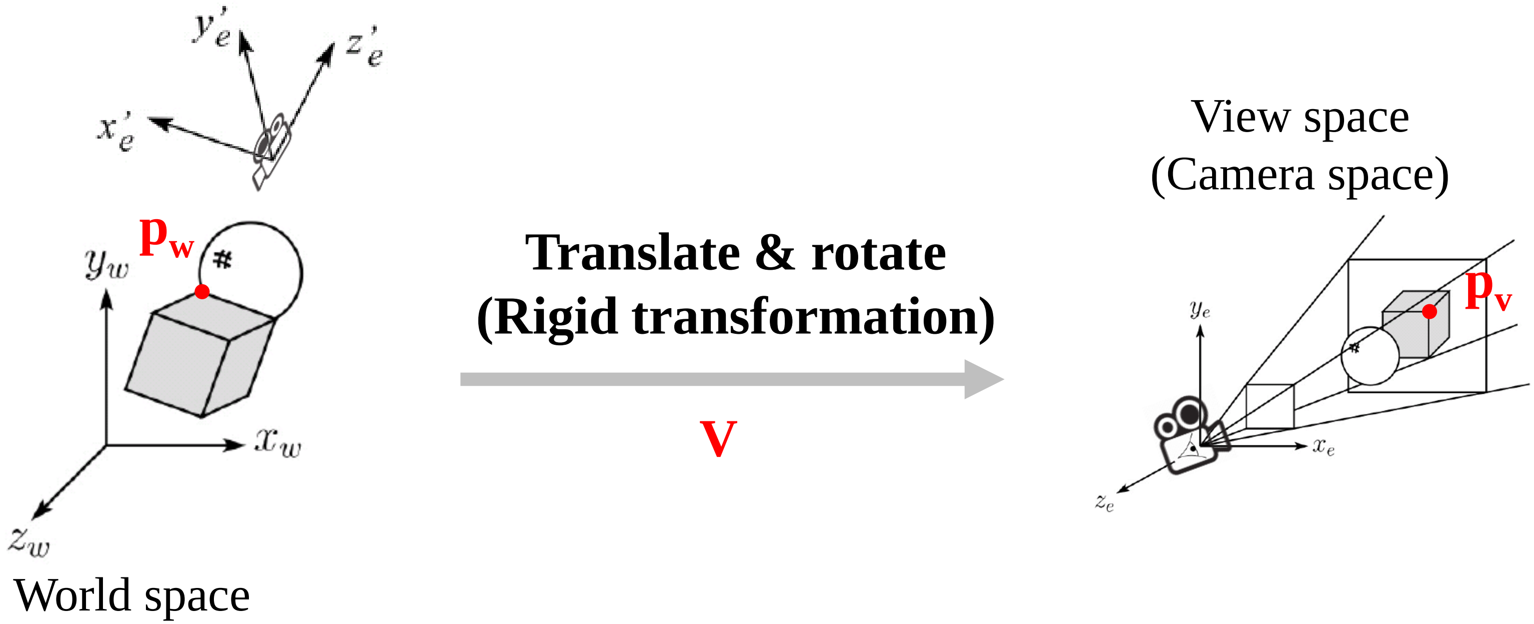
- 3. Selecting its "lens"

→ **Projection transformation**

- 4. Displaying on a "cinema screen"

→ **Viewport transformation**

Viewing Transformation

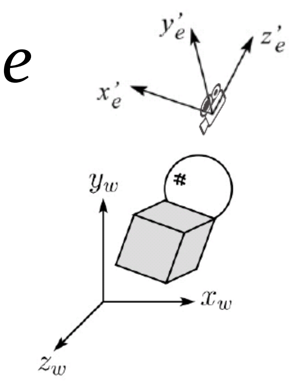


- Transformation from world space to view space is traditionally called the *viewing matrix*, **V**.

Viewing Transformation

- Goal: Expressing all object vertices in the camera's coordinate system (*camera frame*).

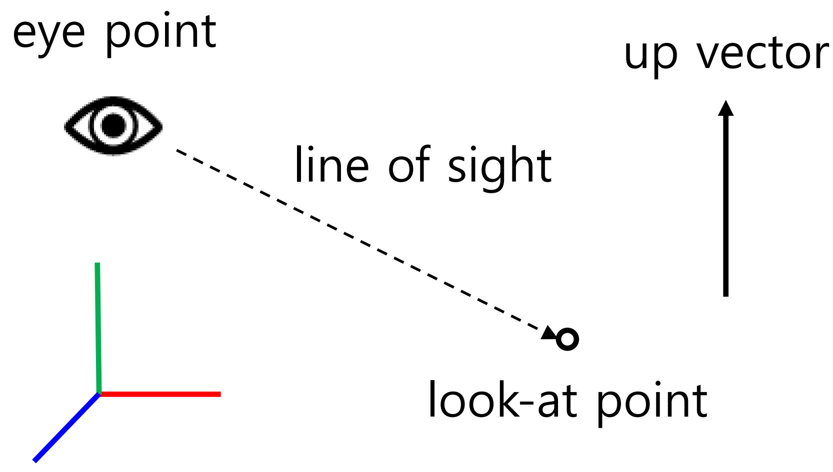
- For that goal, we have to define the *camera frame* (w.r.t. world frame).



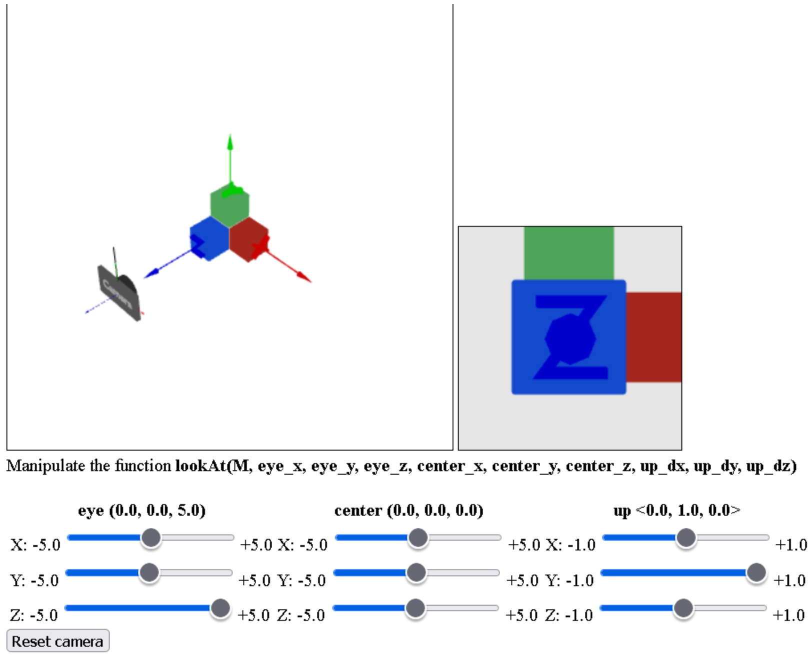
- Defining the *camera frame* is the same as determining *the position and orientation of the camera*.

Defining Camera Frame 1 - "LookAt"

- Many ways to specify the camera's position & orientation.
- I'd like to introduce an intuitive way, which is called "*lookat" function*, that uses:
- **Eye point**
  - Camera position
- **Look-at point**
  - Camera target position
- **Up vector**
  - Roughly defines which direction is *up*



[Demo] LookAt Function



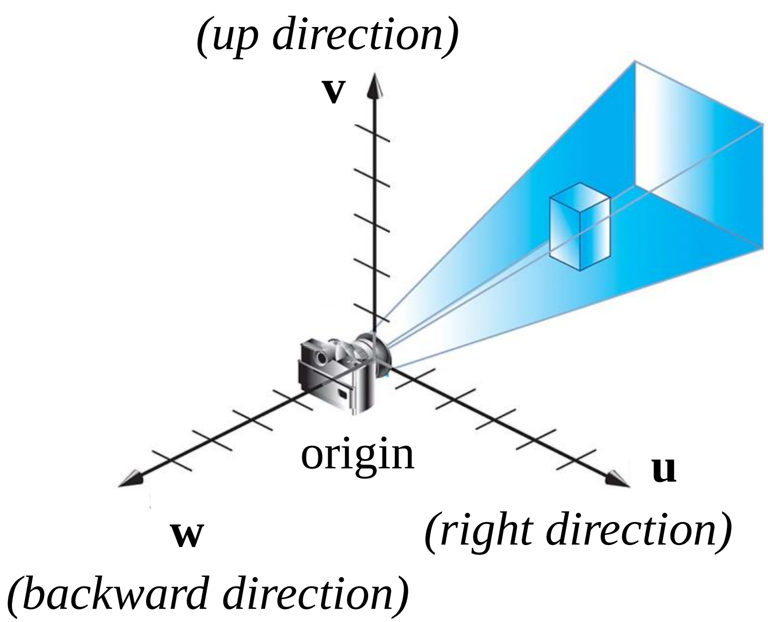
http://learnwebgl.brown37.net/07_cameras/camera_lookat/camera_lookat.html

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

Defining Camera Frame 1 - "LookAt"

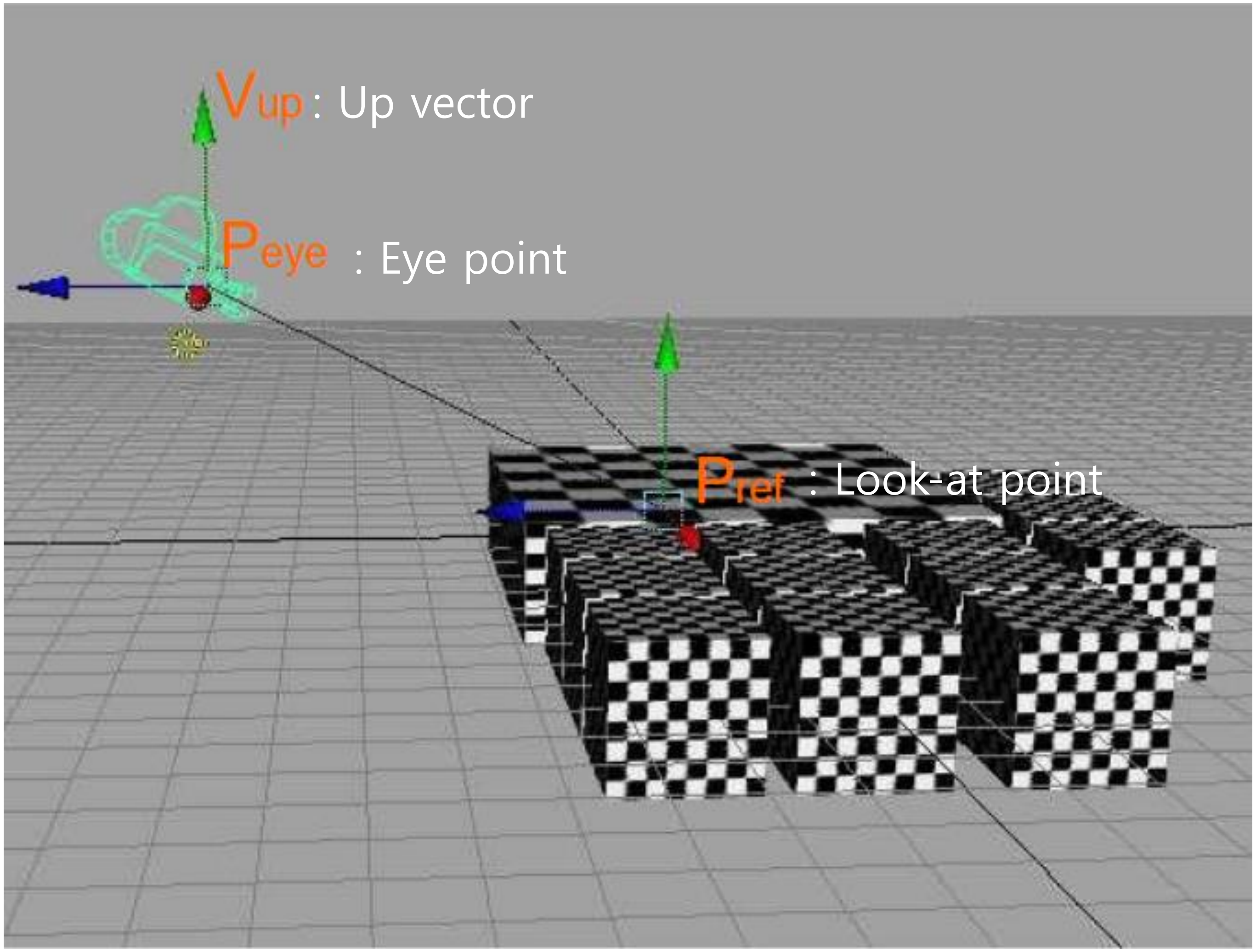
- From the given **eye point**, **look-at point**, **up vector**, we can compute the **camera frame**.

- **u, v, w** are commonly used for camera coordinates axes instead of x, y, z.



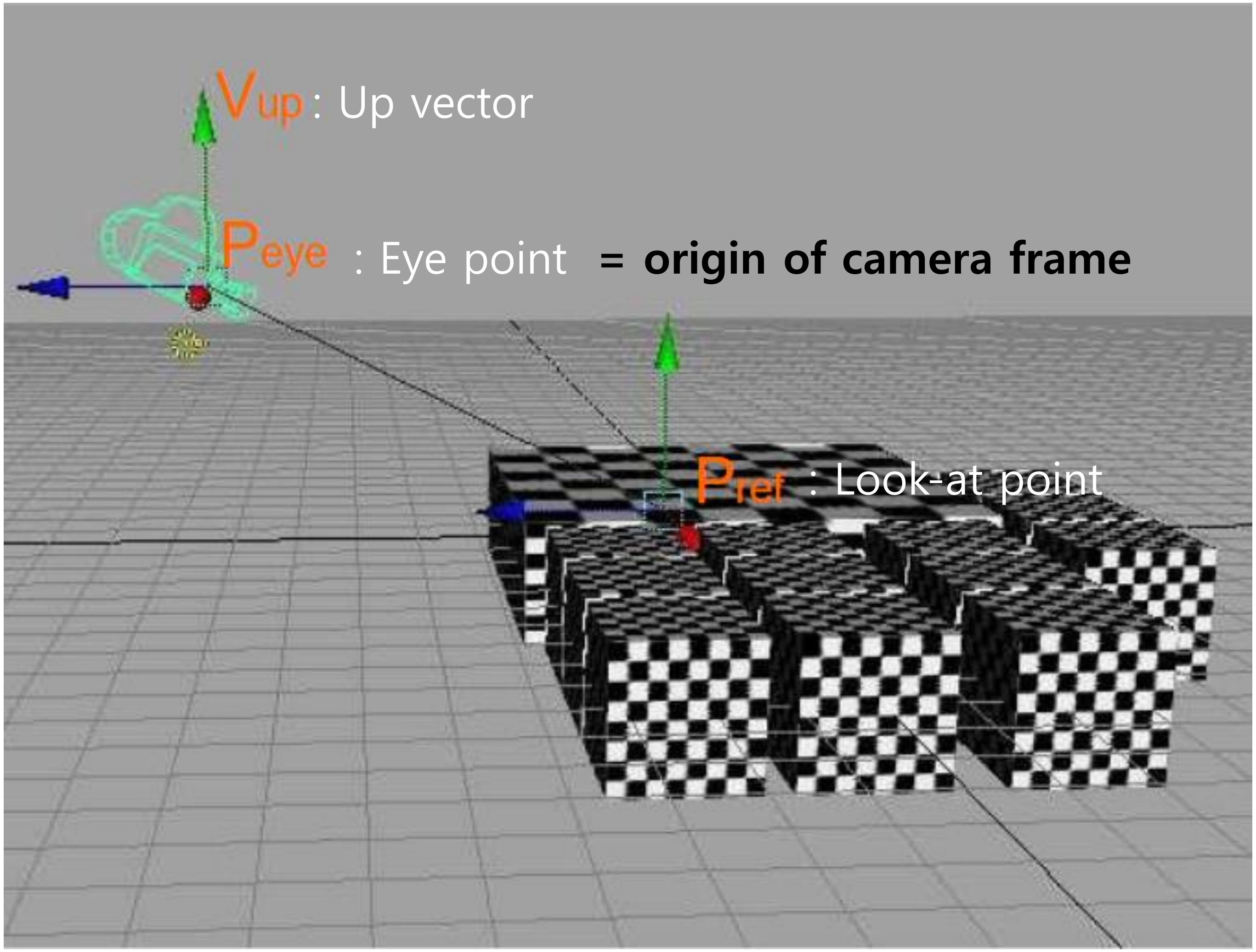
- To define the camera frame, we need to find:
- **u, v, w** vectors
- **origin** point

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

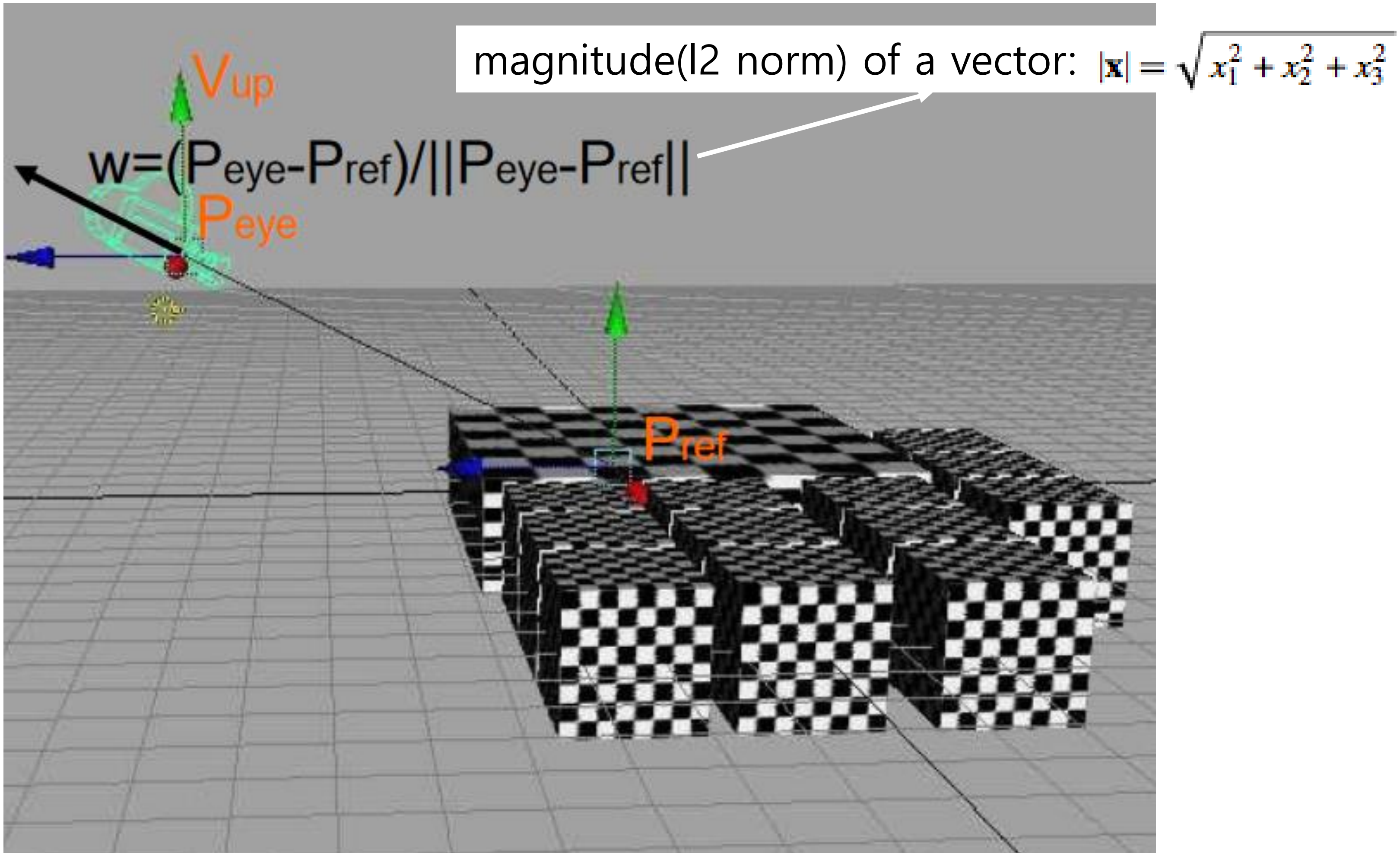


* The base images are from the slides of Prof. Karan Singh's (University of Toronto): http://www.dgp.toronto.edu/~karan/courses/418/

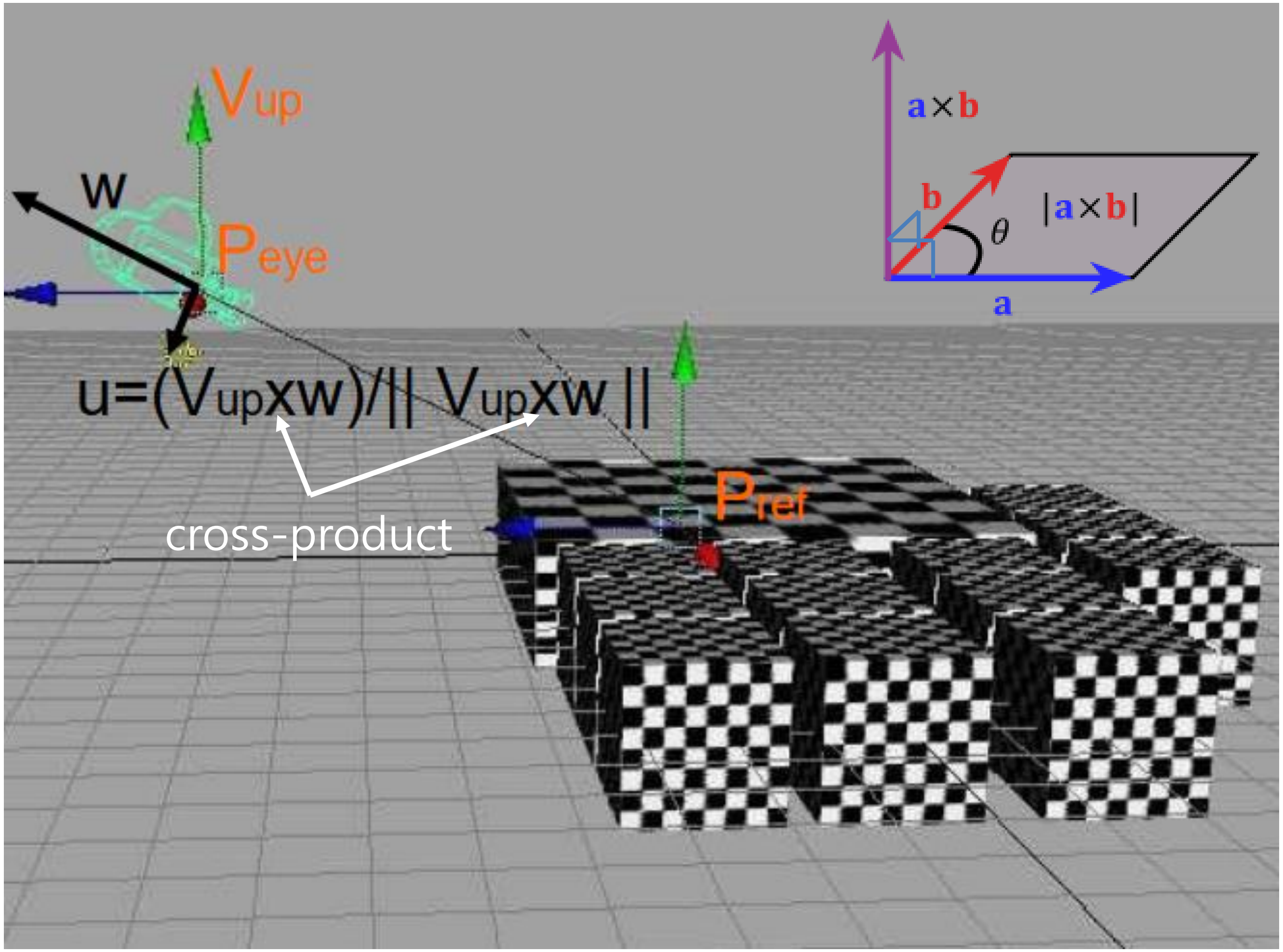
Getting origin point



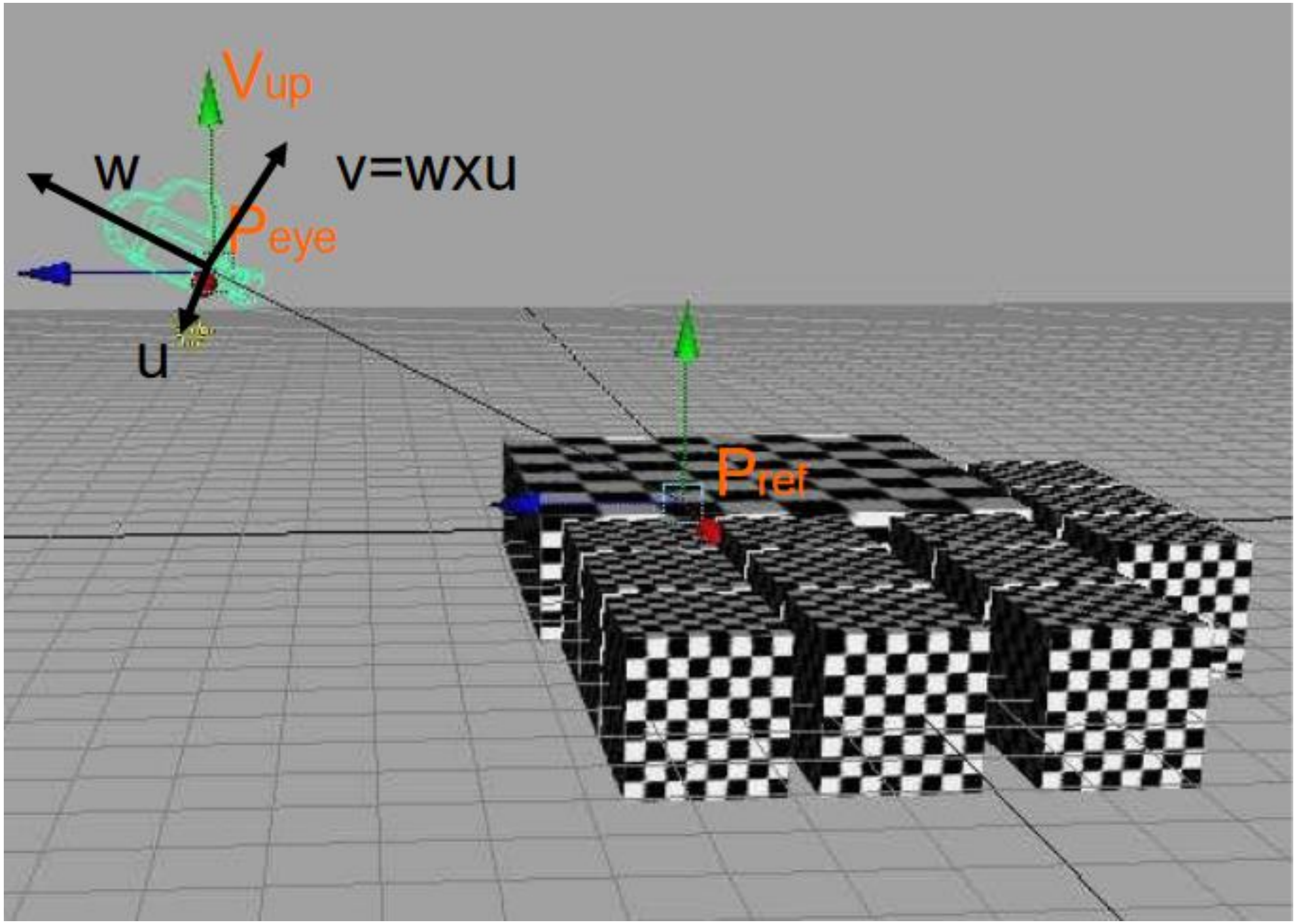
Getting “w” axis vector



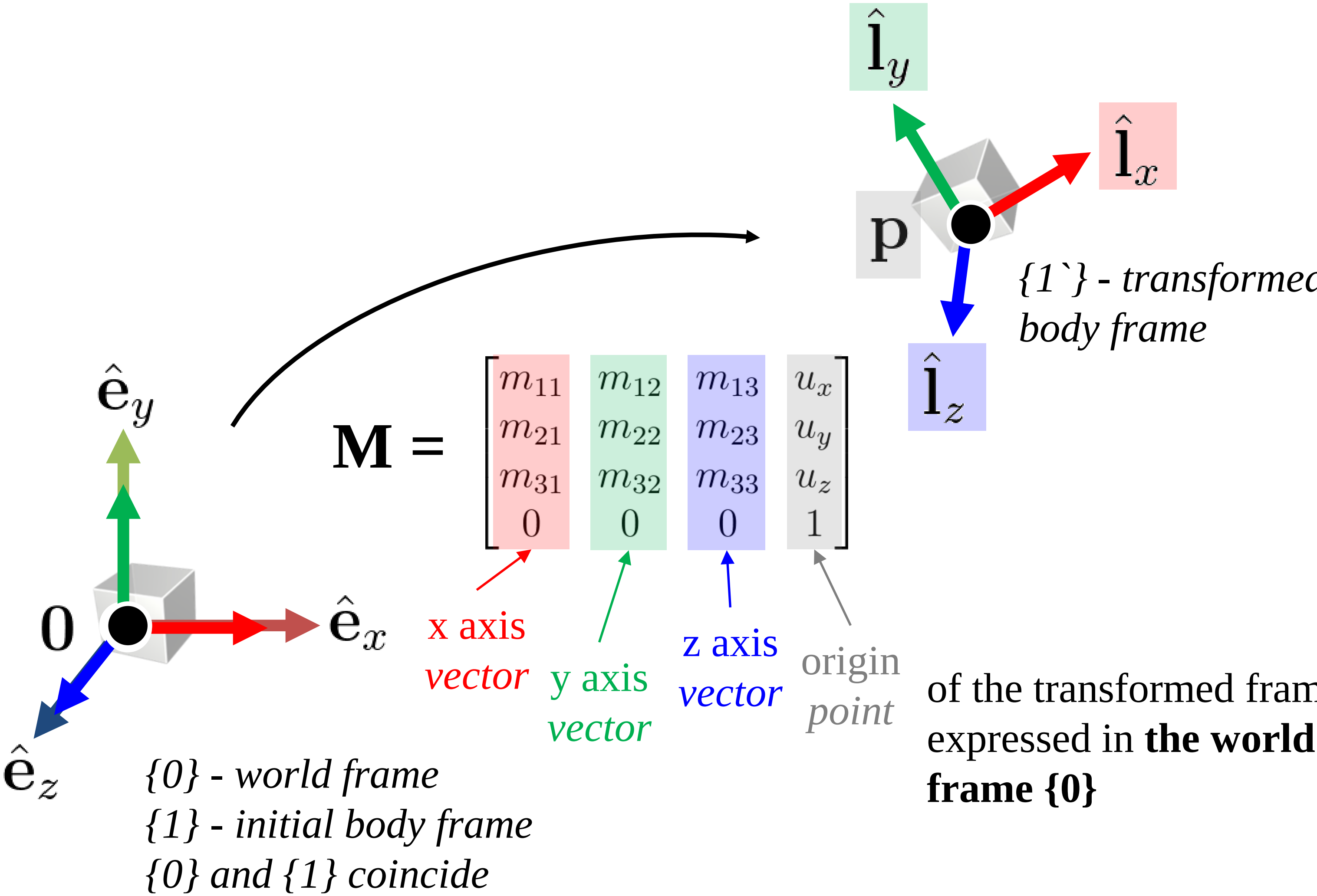
Getting "u" axis vector



Getting "v" axis vector



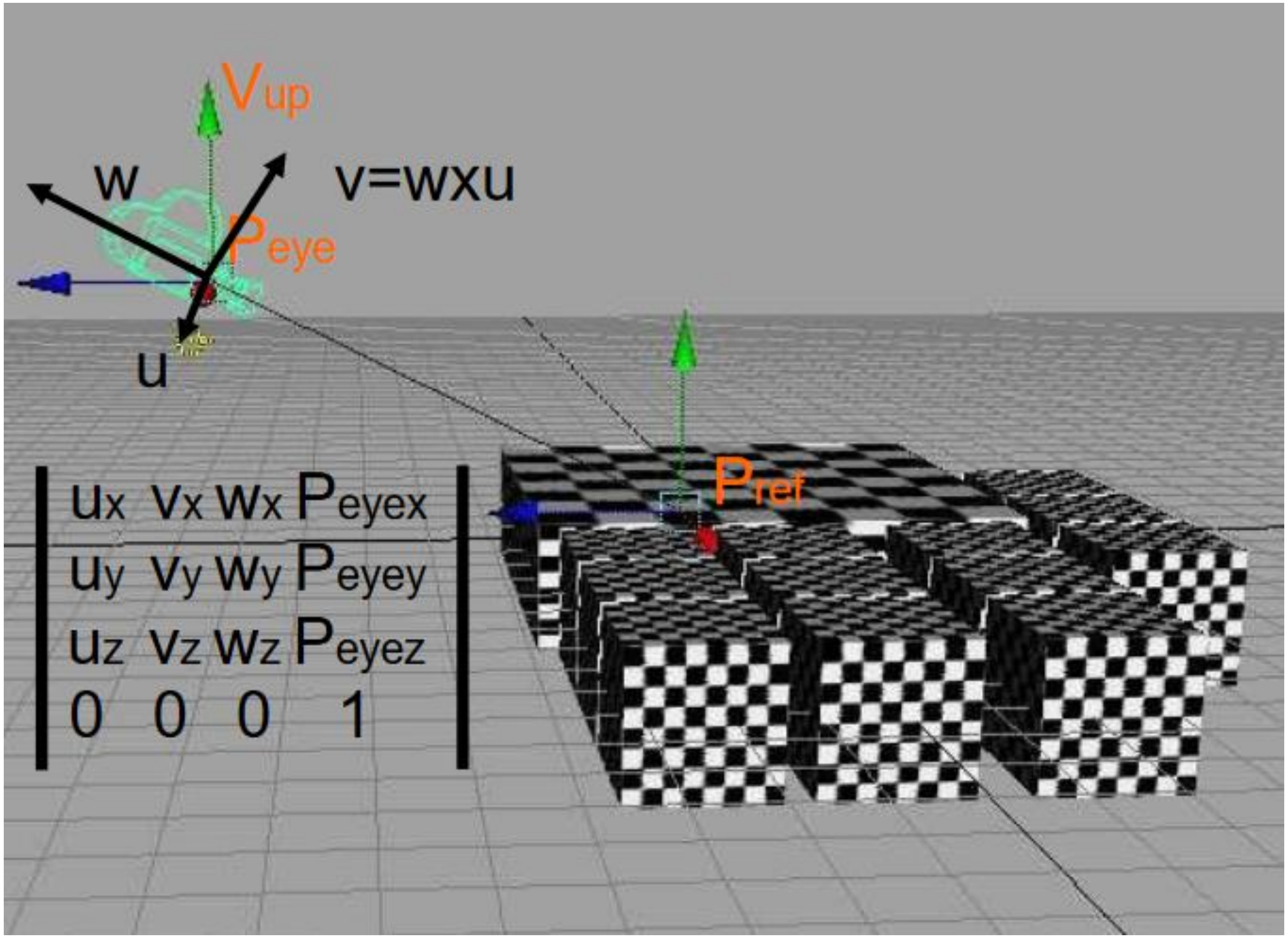
Recall: 2) Affine Transformation Matrix **defines an Affine Frame** w.r.t. World Frame



$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

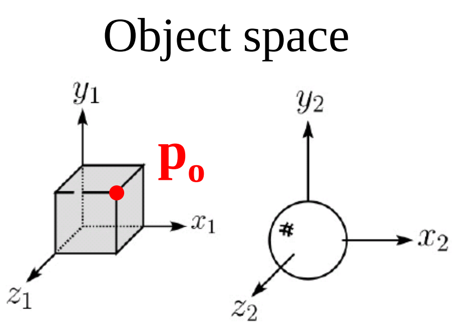
of the transformed frame {1'} expressed in **the world frame {0}**

Thus, the Camera Frame is defined by



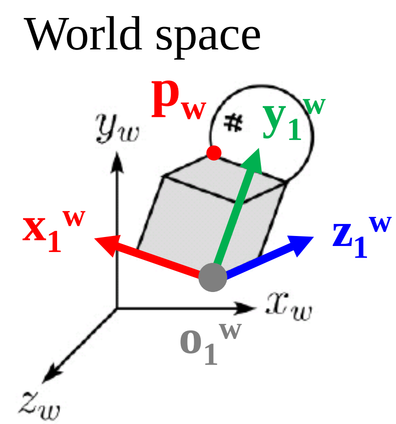
How can we get viewing matrix V from the camera frame?

- Recall the modeling transformation:



$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{12} & m_{13} & u_y \\ m_{31} & m_{12} & m_{13} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

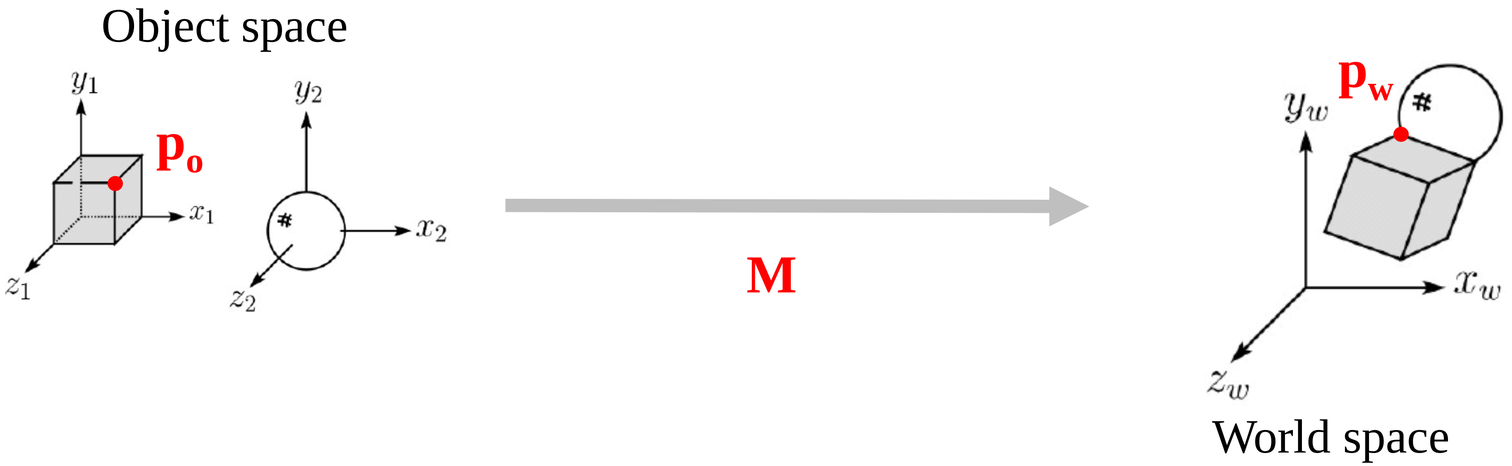
$\mathbf{x_1}^w$ *vector* | $\mathbf{y_1}^w$ *vector* | $\mathbf{z_1}^w$ *vector* | $\mathbf{o_1}^w$ *point*



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

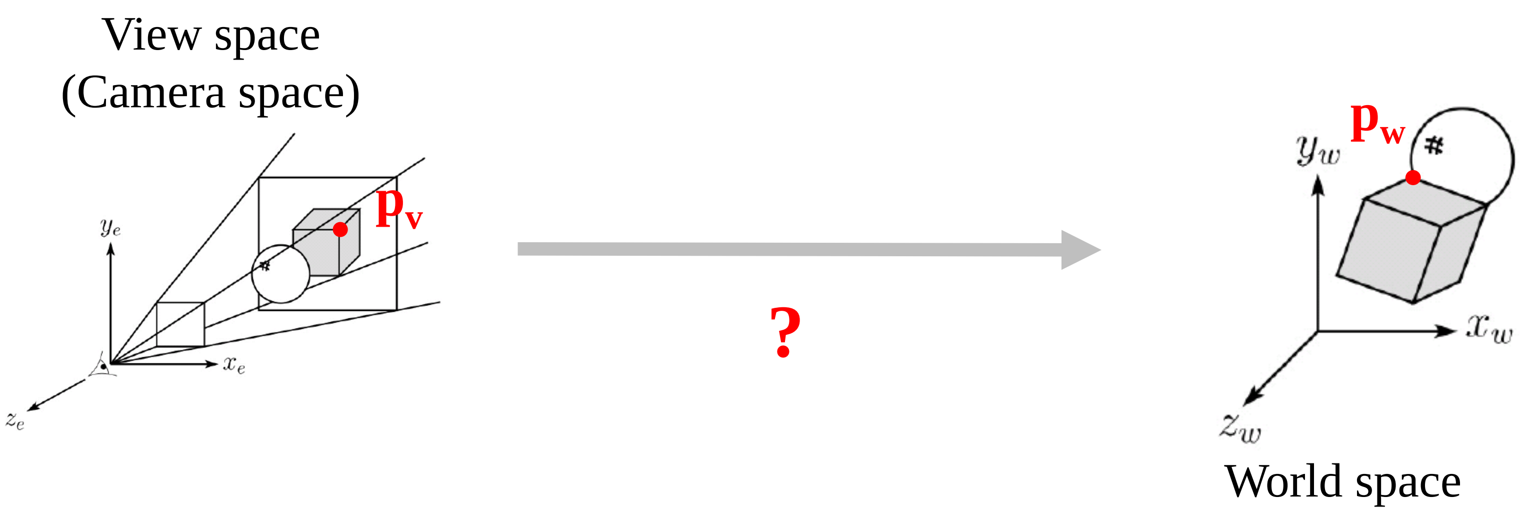
How can we get viewing matrix V from the camera frame?

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



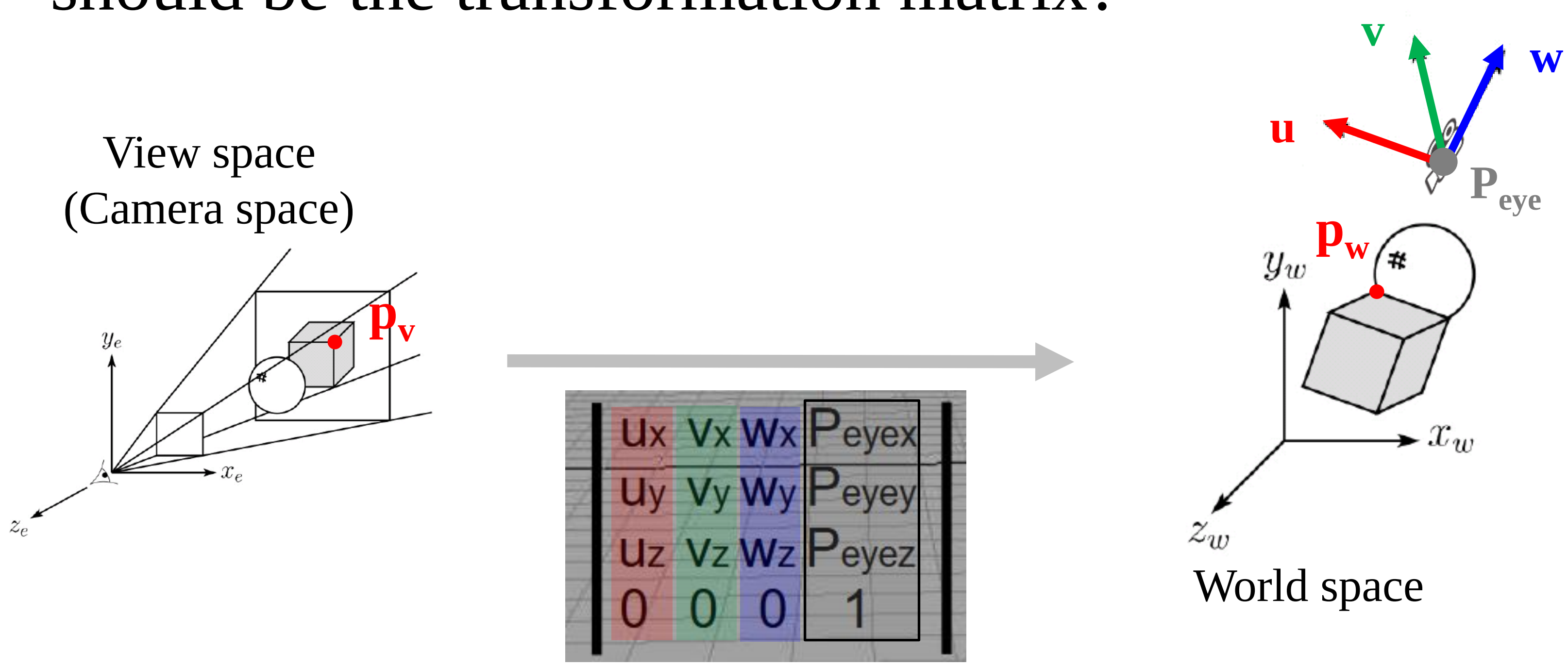
How can we get viewing matrix V from the camera frame?

- 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?

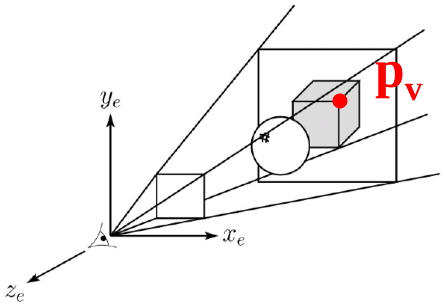
- If we replace *object space* to *camera space*, what should be the transformation matrix?

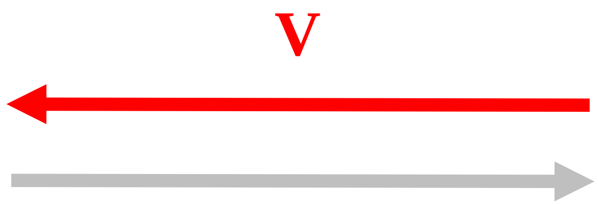


: The axis vectors and origin point of the **camera frame represented in the world frame**

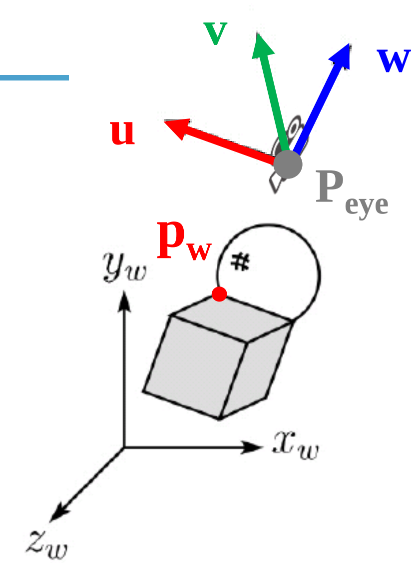
Viewing Transformation is the Opposite Direction

View space
(Camera space)





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



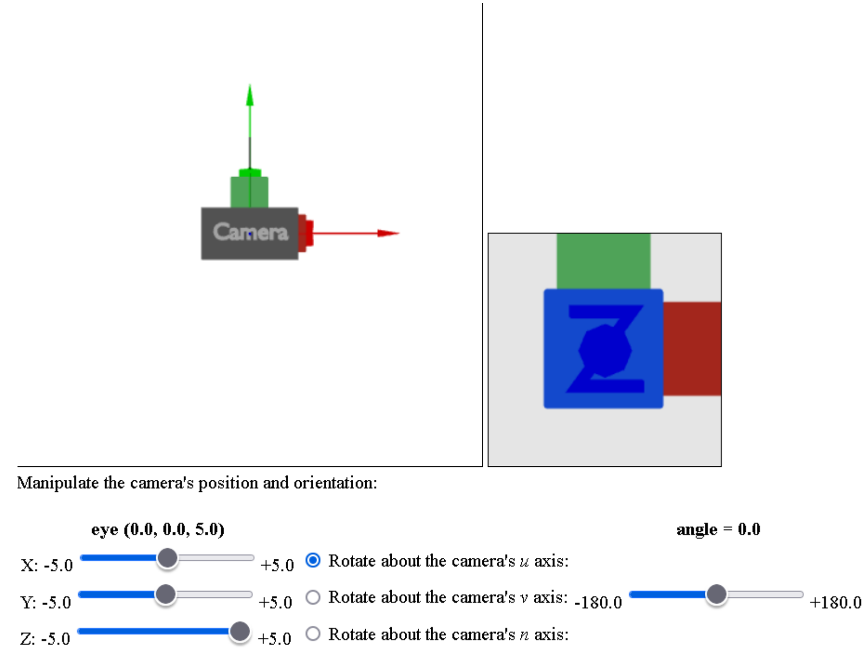
World space

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

Defining Camera Frame 2 - Translate & Rotate

- "LookAt" is not the only way to specify the camera's position and orientation.
- You can just "translate" and "rotate" the camera instead.
- The camera frame is then just defined by these rigid transformation matrices.

[Demo] Translate & Rotate Camera

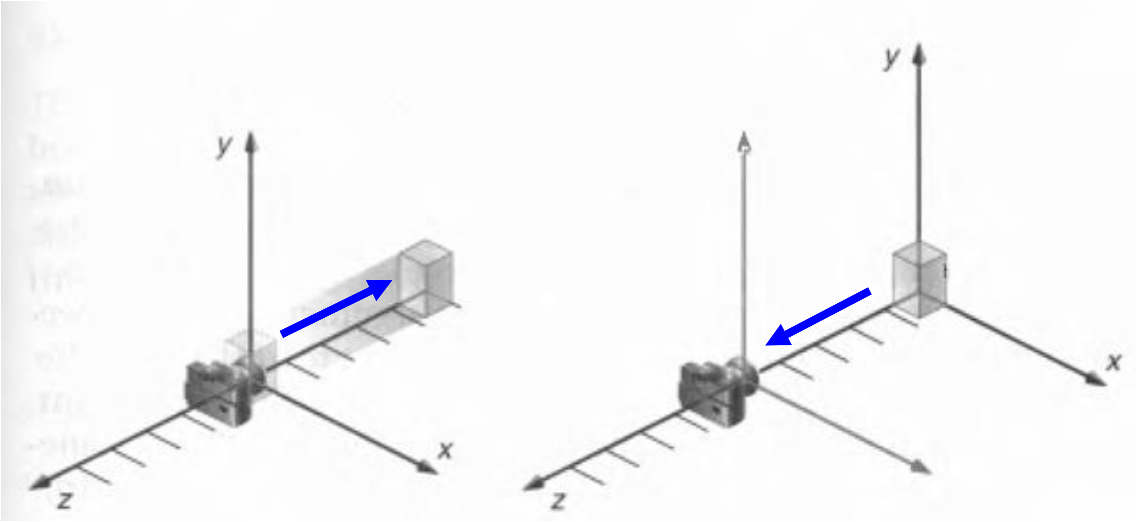


http://learnwebgl.brown37.net/07_cameras/camera_trunk_axes/camera_trunk_axes.html

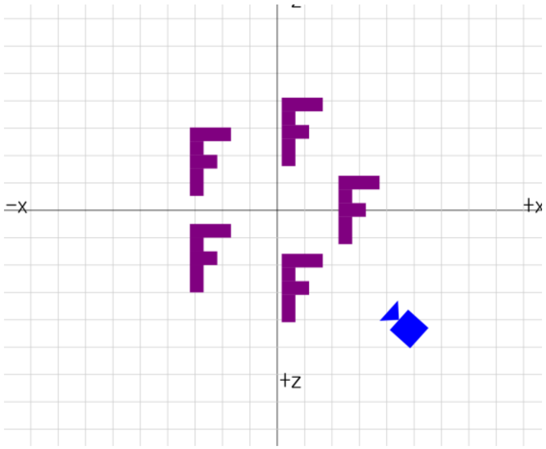
- Observe the 3D scene (left) and rendered view (right) while translating the camera with "eye" sliders and rotating it with axis selection and "angle" slider.

Moving Camera vs. Moving World

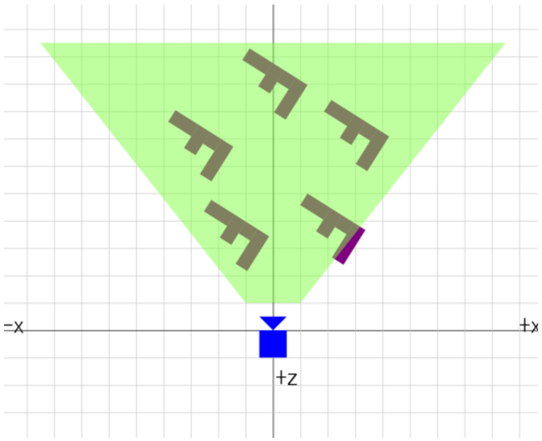
- Actually, these are two **equivalent operations**
- Translate camera by (1, 0, 2) == Translate world by (-1, 0, -2)
- Rotate camera by 60° about y == Rotate world by -60° about y



[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
- (Center) Moving world

Lab Session

- Now, let's start the lab today.