Computer Graphics

6 - Vertex Processing 2

Yoonsang Lee
Hanyang University

Spring 2023

Midterm Exam Announcement

- Date & time: **May 1**, **7:30 - 8:30 PM**
- Place: TBA
- Scope: Lecture 2~7, Lab 2~7
  - Lecture & Lab 8 is included in the final exam scope.

- More details will be announced later.

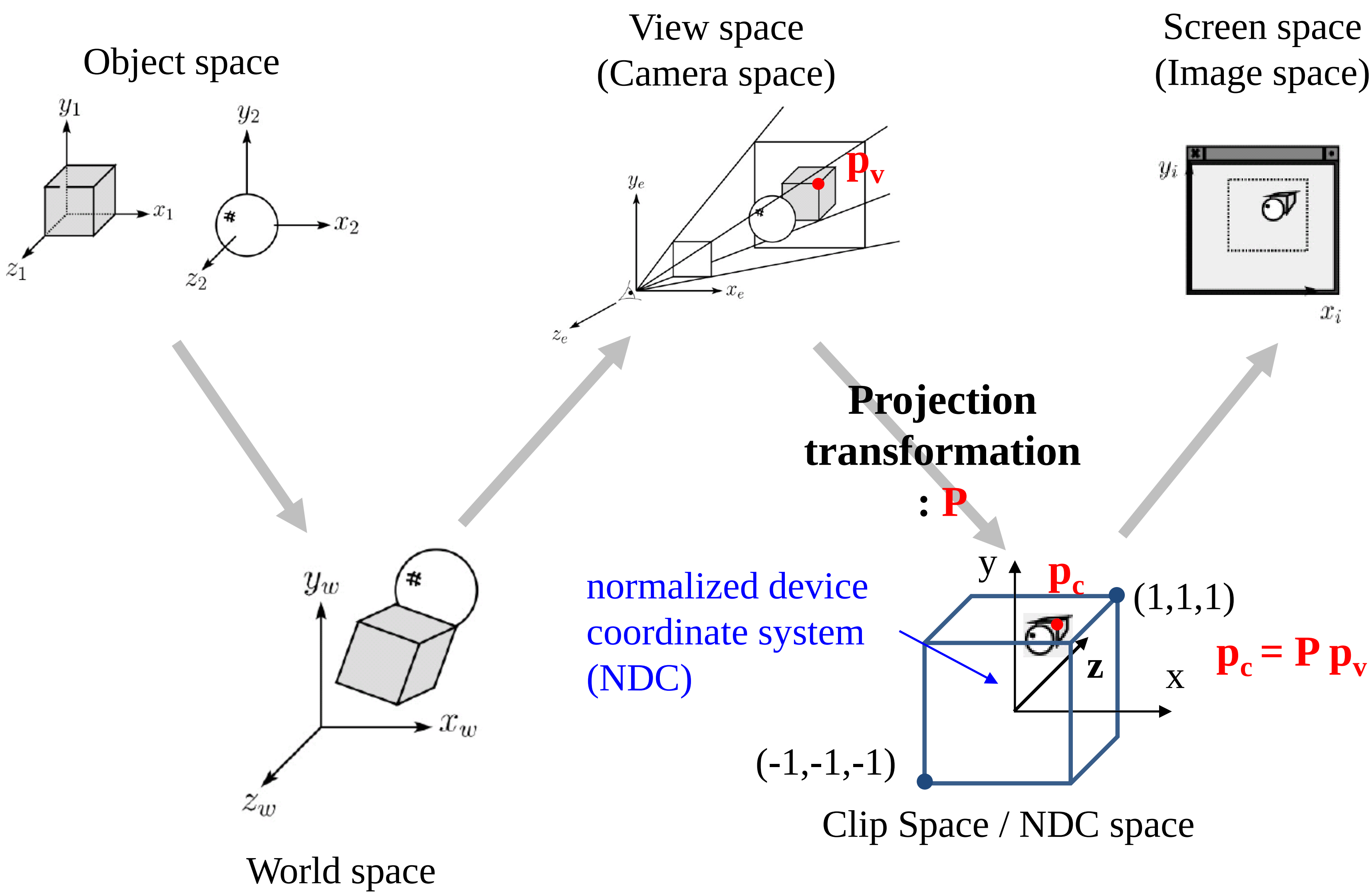
Outline

- Projection Transformation
  - Orthographic Projection
  - Perspective Projection

- Viewport Transformation

Projection Transformation

Projection Transformation



Recall that...

- 1. Placing objects

→ **Modeling transformation**

- 2. Placing a "camera"

→ **Viewing transformation (covered in the last class)**

- 3. Selecting its "lens"

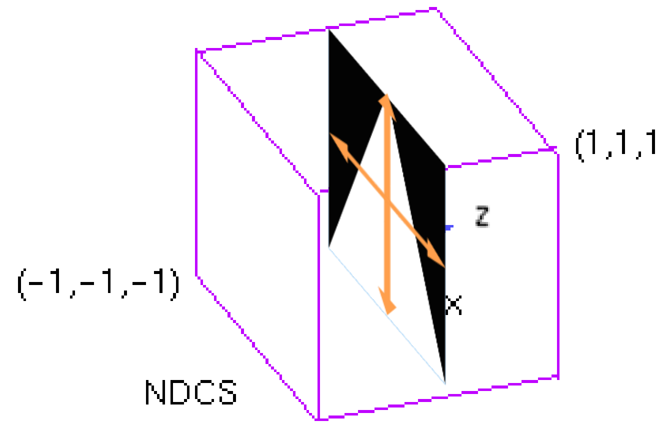
→ **Projection transformation**

- 4. Displaying on a "cinema screen"

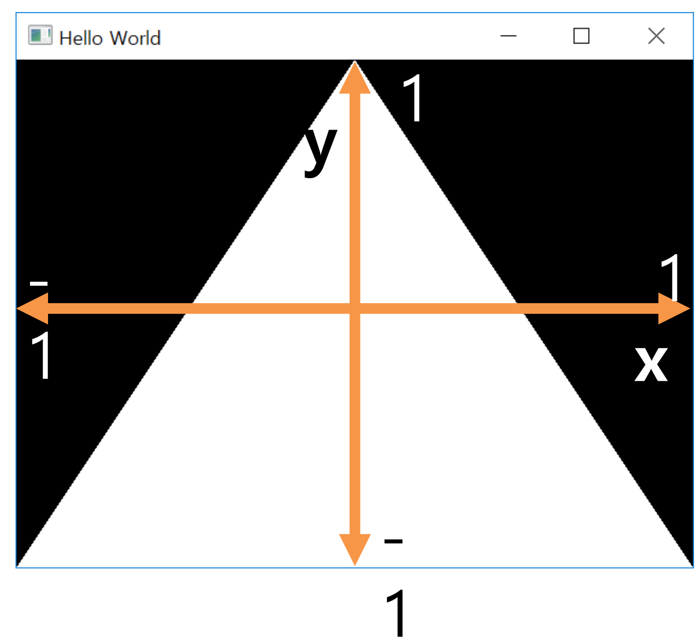
→ **Viewport transformation**

Recall: OpenGL Clip Space

- By default (with no transformation), you can draw an object anywhere in cube space with x, y, z coordinates ranging from -1 to 1.



- This space is called *clip space* (or *NDC space*).
  - Its **xy** plane is a 2D “viewport”.
  - Its coordinate system is *normalized device coordinate (NDC).*



- We will take a closer look at their meaning in later lectures.

Normalized Device Coordinates (NDC)

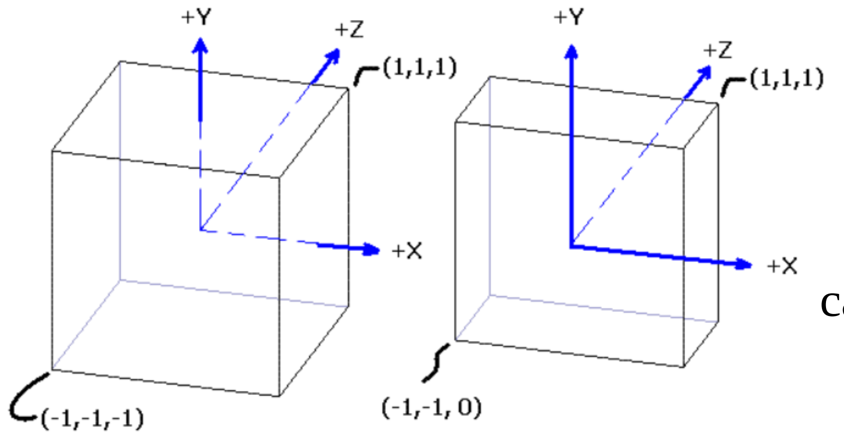
- *Normalized device coordinates (NDC)* is a device independent display coordinate system.
  - Various display devices' size, measured in pixels, can vary.
  - So, it's important to specify coordinates using units other than pixels to make the programs device independent.

- The space expressed by NDC: *clip space* (or *NDC space*)
  - However, clip / NDC space are slightly different, which is covered in today's lecture.

Canonical View Volume

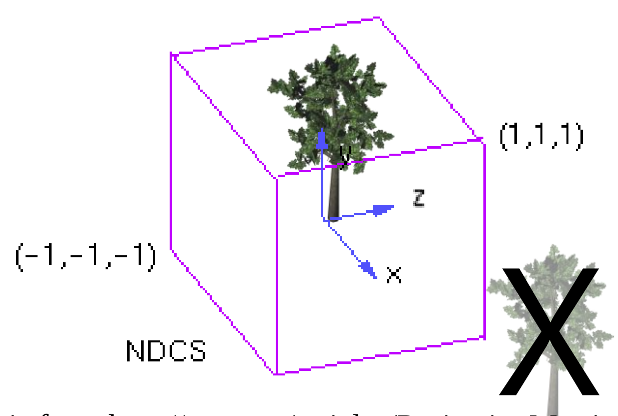
- *Canonical view volume* is a 3D volume in clip / NDC space that defines the visible region of the scene.

OpenGL
canonical view volume



Direct3D
canonical view volume

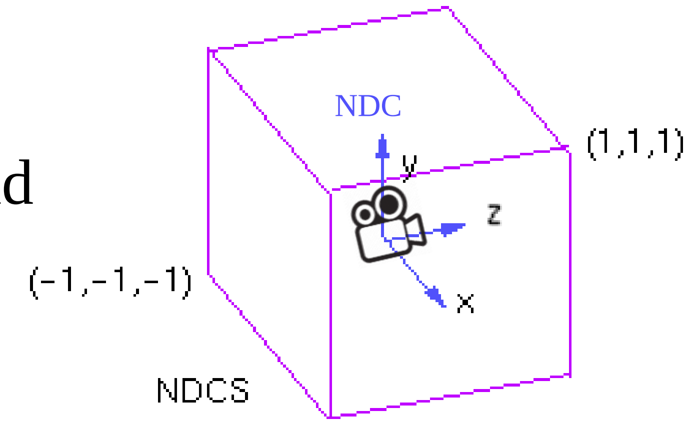
- Only objects **inside** the canonical view volume are rendered.
  - To draw objects only in the camera's view
  - Not to draw objects too near or too far from the camera



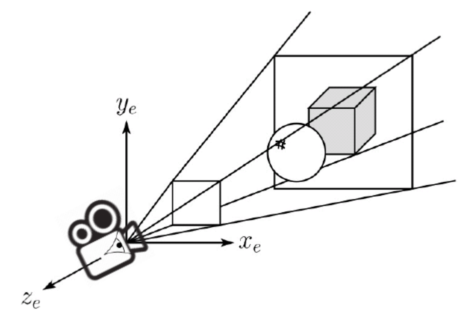
* This image is from http://perry.cz/articles/ProjectionMatrix.xhtml

Canonical View Volume

- Conventionally, NDC is a left-handed coordinate system (both in OpenGL and Direct3D).
  - Viewing direction in NDC : +z direction



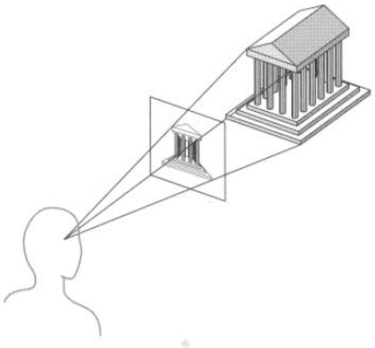
- In OpenGL, projection functions change the hand-ness by default – Thus view, world, model spaces use a right-handed coordinate system.
  - Viewing direction in view space : -z direction
  - (Direct3D use a left-handed system by default; it does not change hand-ness.)



View Volume

- However, you don't have to try to place all objects in the range -1 to +1 in x, y, z coordinates.
- Instead, you can set up a cuboid or frustum volume of any size and draw objects inside it.
- Then this view volume (and everything inside it) is mapped (projected) into the canonical view volume.
- → **Projection transformation**

Projection Transformation



- To map 3D coordinates to 2D screen coordinates,

- An arbitrary view volume is mapped to the canonical view volume.

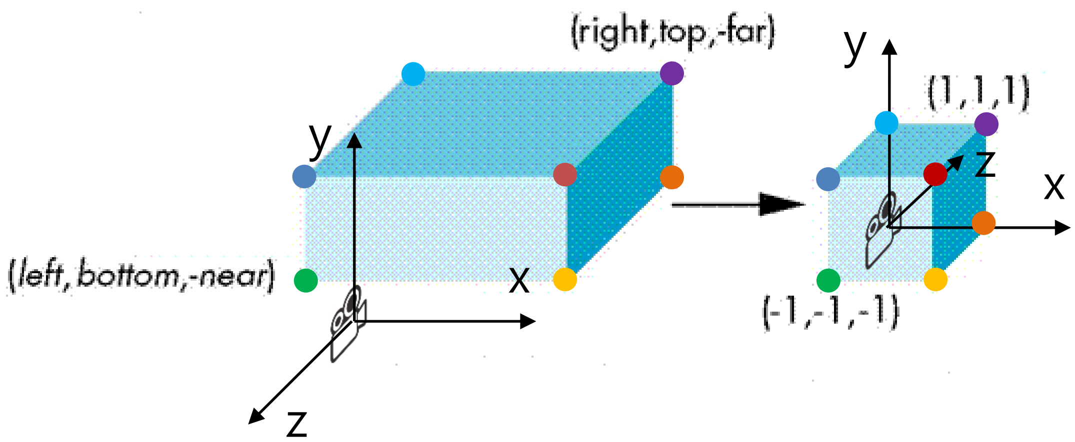
  – Mapping the 3D points of the canonical view volume onto the xy plane doesn't actually happen, because we still need the z-value of the point for "depth testing".

- Two common projection methods:
  – Orthographic projection
  – Perspective projection

Orthographic (Orthogonal) Projection

- View volume : Cuboid (직육면체)
- Orthographic projection : Mapping from a cuboid view volume to a canonical view volume
  - Combination of scaling & translation

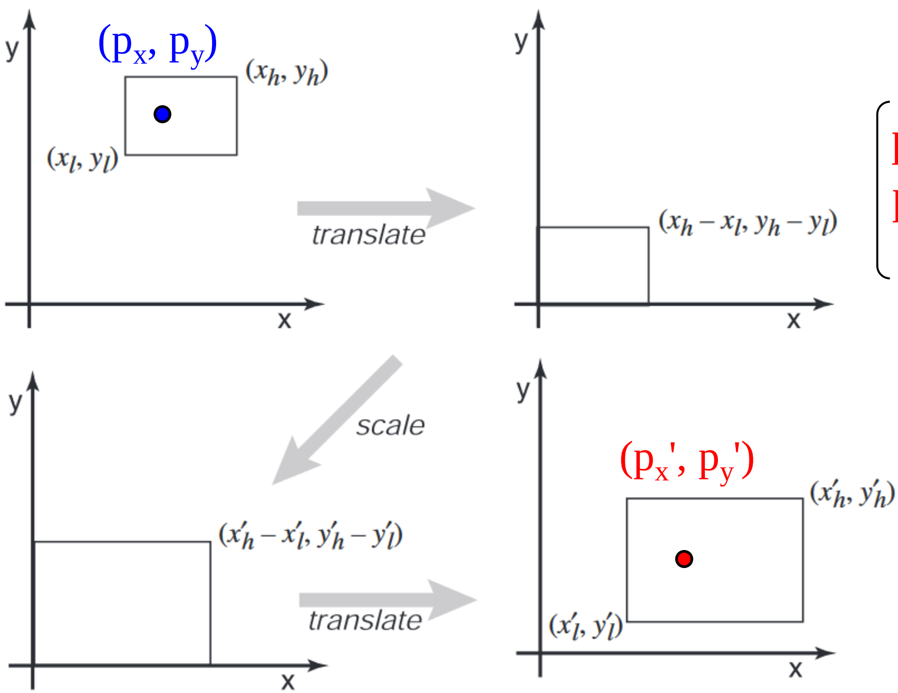
  → "Windowing" transformation



* The base image is from https://jcsites.juniata.edu/faculty/rhodes/graphics/projectionmat.htm

Windowing Transformation

- Transformation that maps a point (p_x, p_y) in a rectangular space from (x_l, y_l) to (x_h, y_h) to a point (p_x', p_y') in a rectangular space from (x_l', y_l') to (x_h', y_h')



$$\begin{pmatrix} p_x' \\ p_y' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & x'_l \\ 0 & 1 & y'_l \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_l \\ 0 & 1 & -y_l \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix}$$

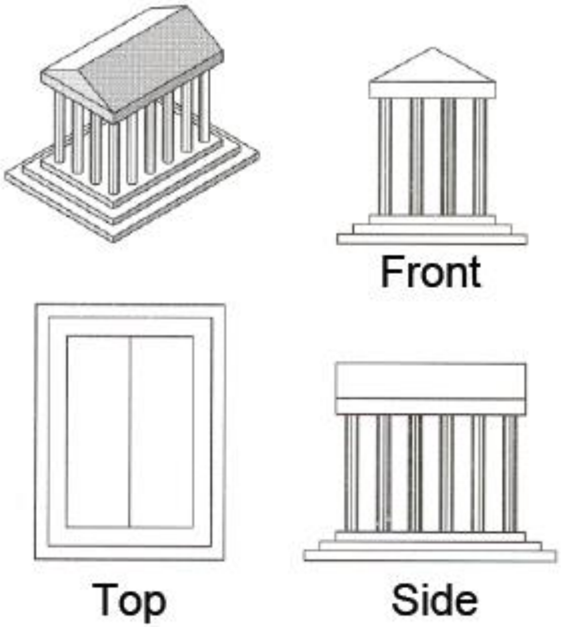
$$\begin{pmatrix} p_x' \\ p_y' \\ 1 \end{pmatrix} = \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix}$$

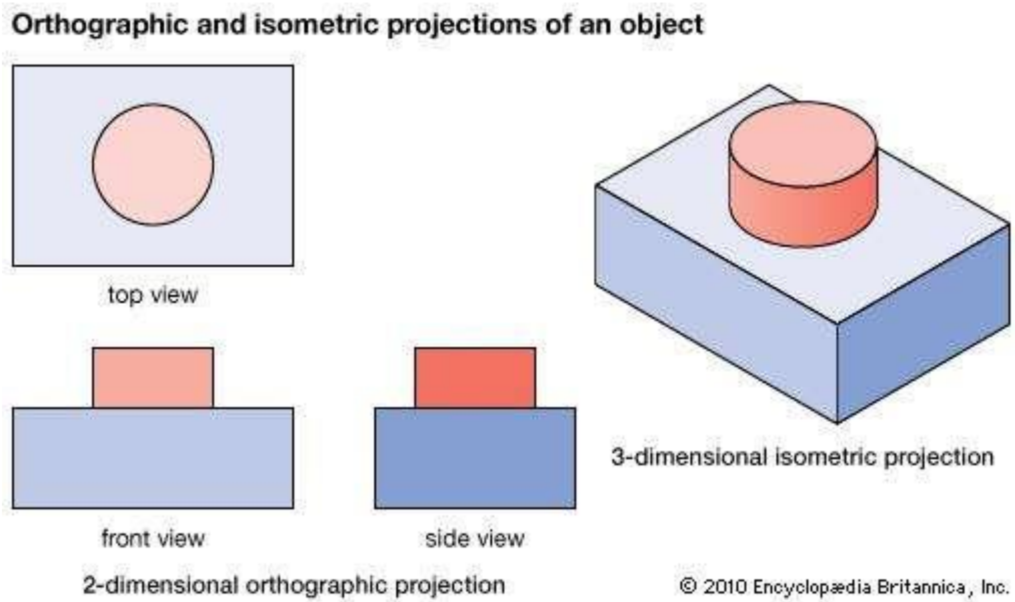
Orthographic Projection Matrix

- By extending the matrix to 3D and substituting
  - x_h=right, x_l=left, x_h'=1, x_l'=-1
  - y_h=top, y_l=bottom, y_h'=1, y_l'=-1
  - z_h=-far, z_l=-near, z_h'=1, z_l'=-1

$$P_{orth} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Examples of Orthographic Projection

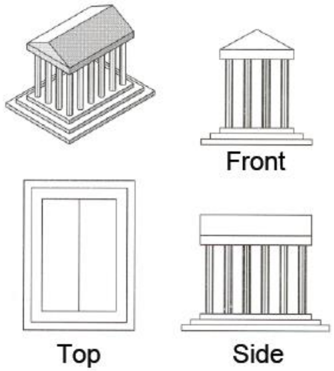




An object always stay the same size, no matter its distance from the viewer.

 * The image is from https://www.britannica.com/technology/orthographic-projection-engineering

Properties of Orthographic Projection



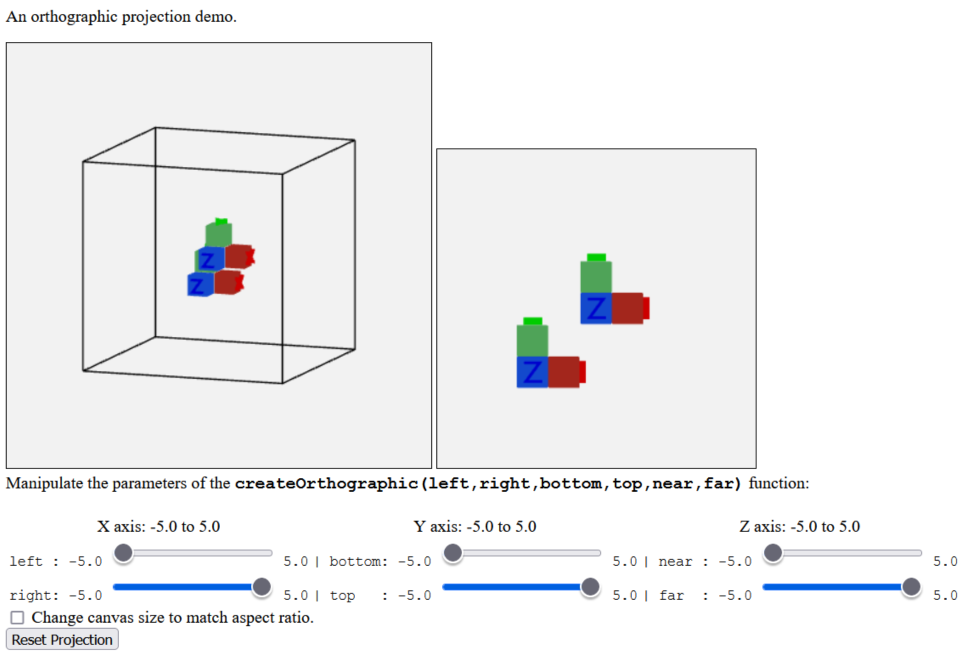
- Not realistic looking

- Good for exact measurement

- Most often used in CAD, architectural drawings, etc. where taking exact measurement is important.

- Combination of scaling and translation
  → Affine transformation

[Demo] Orthographic Projection



http://learnwebgl.brown37.net/08_projections/create_ortho/create_ortho.html

- Observe the view volume (left) and rendered view (right) while dragging left, right, bottom, top, near, far sliders.

Quiz 1

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

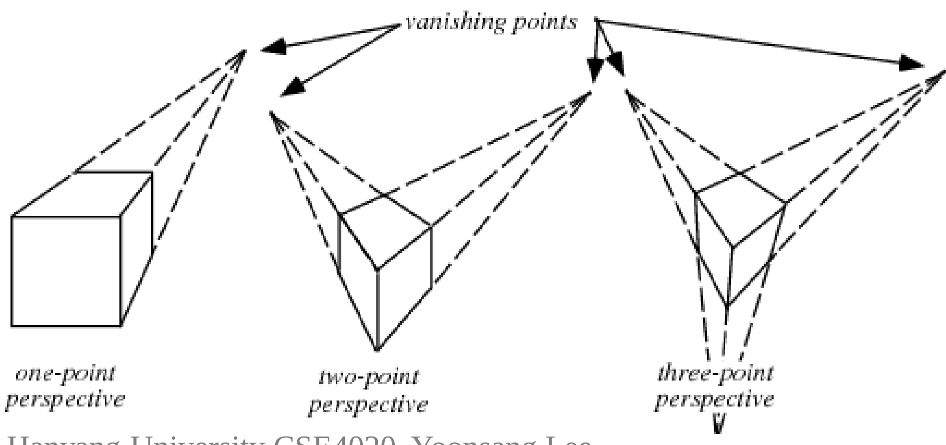
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2021123456: 4.0**

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

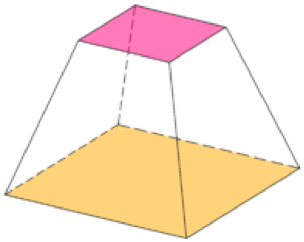
Perspective Effects

- Distant objects become small.

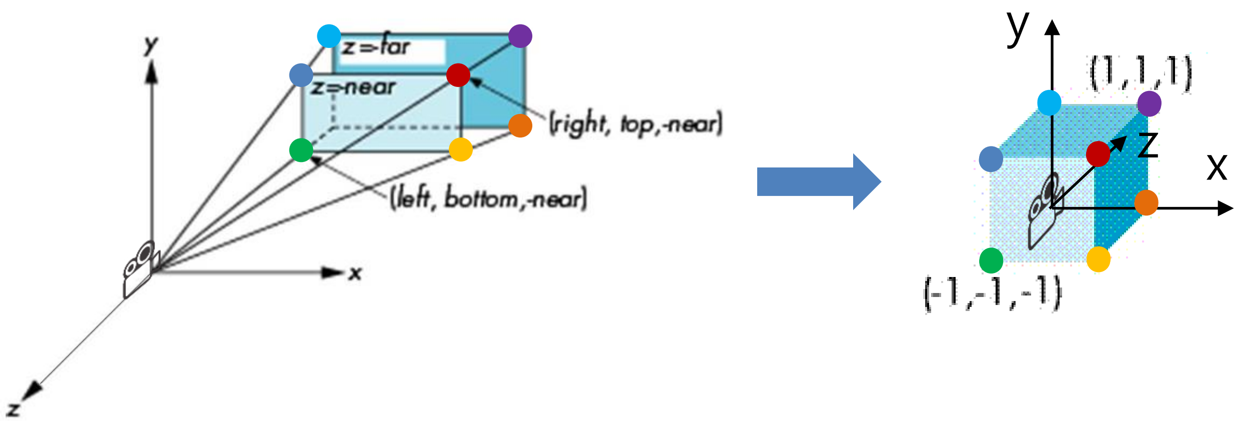
Vanishing point: The point or points to which the extensions of parallel lines appear to converge in a perspective drawing



Perspective Projection



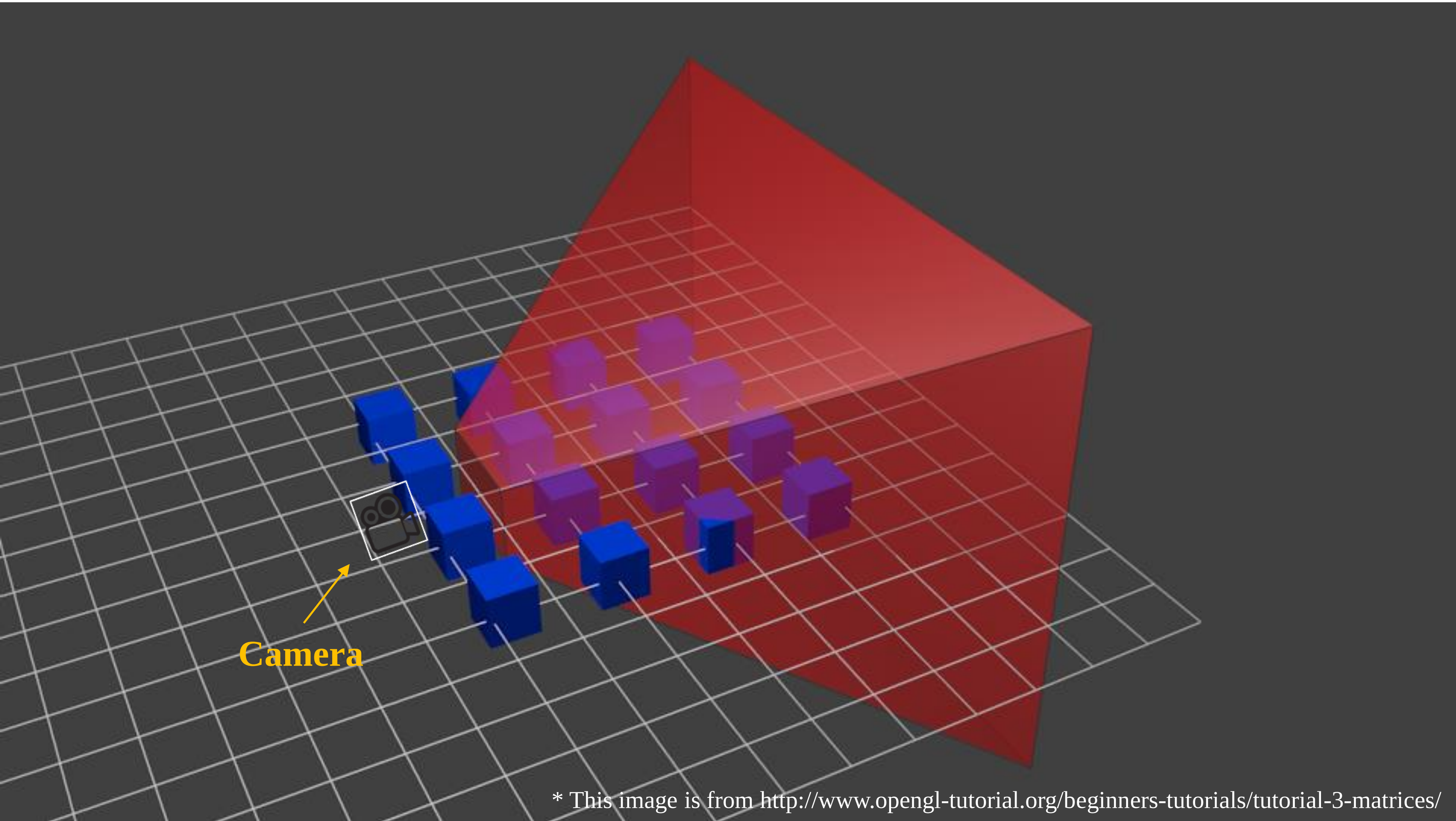
- View volume : Frustum (절두체)
- → "Viewing frustum"
- Perspective projection : Mapping from a viewing frustum to a canonical view volume



Why does this mapping generate a perspective effect?

Original 3D scene

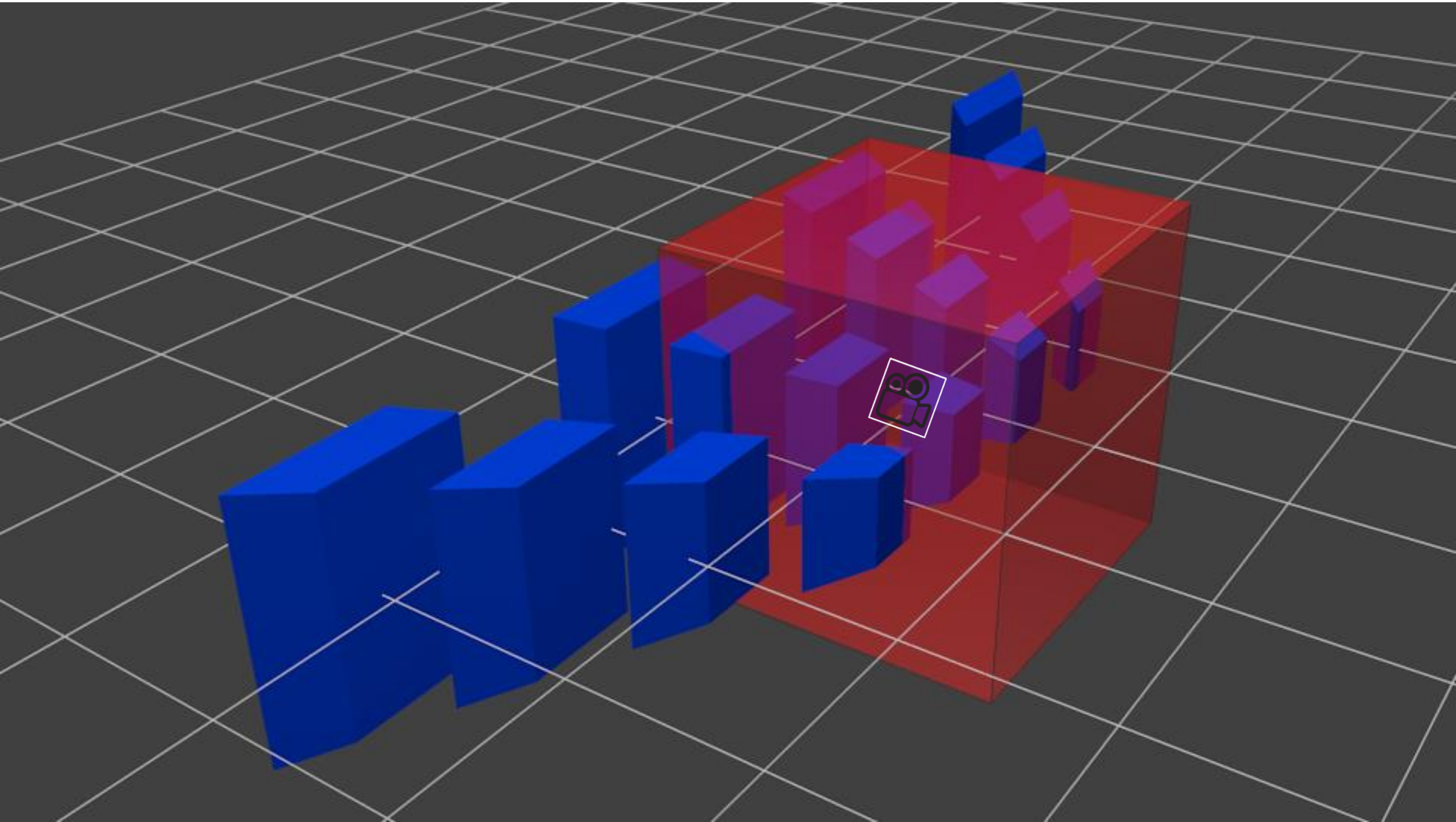
Red: viewing frustum, Blue: objects



* This image is from http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

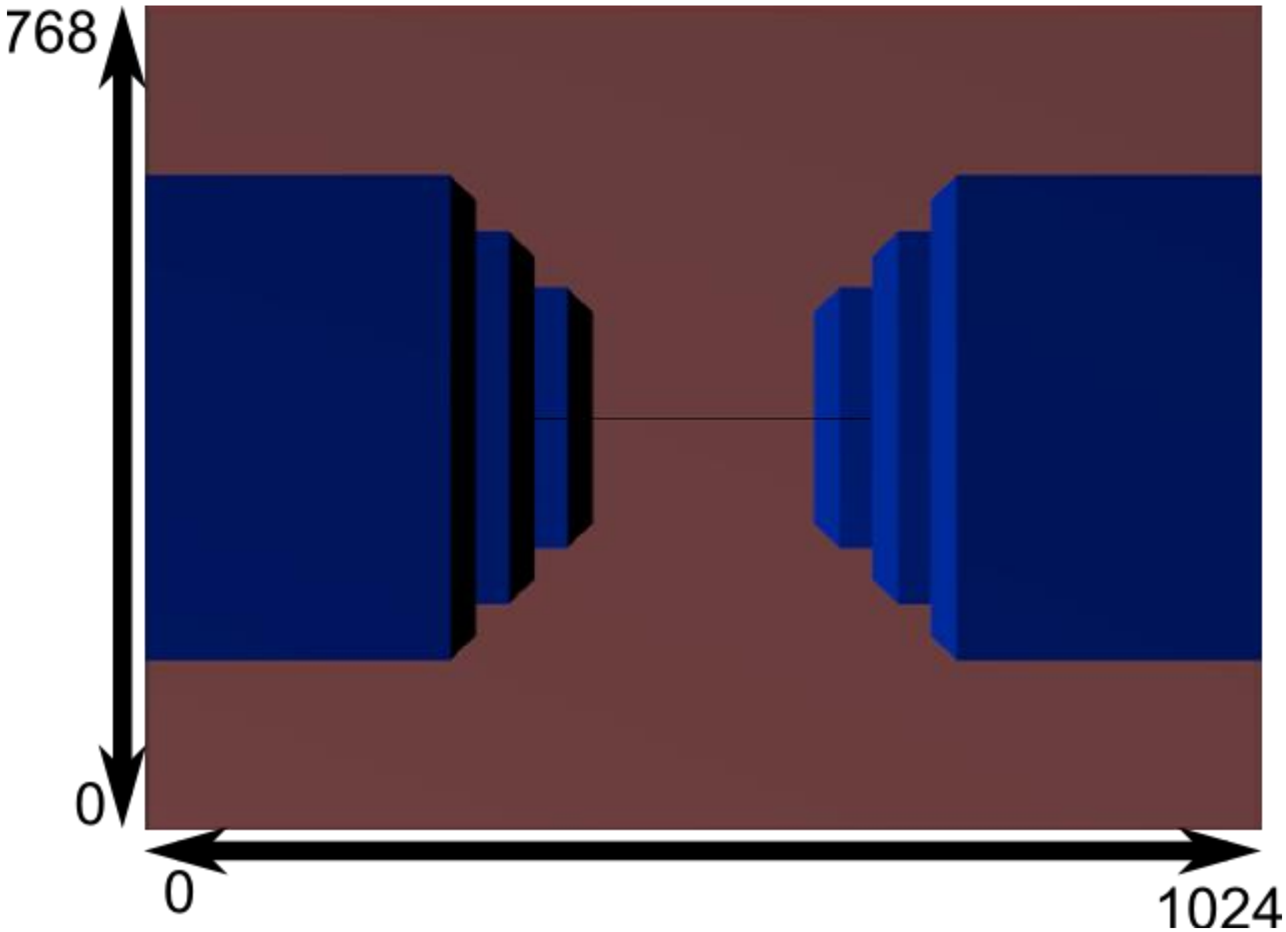
An Example of Perspective Projection

After perspective projection



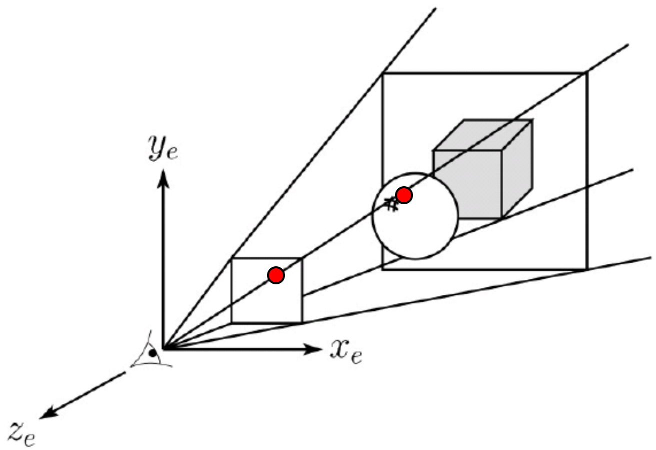
An Example of Perspective Projection

Camera view



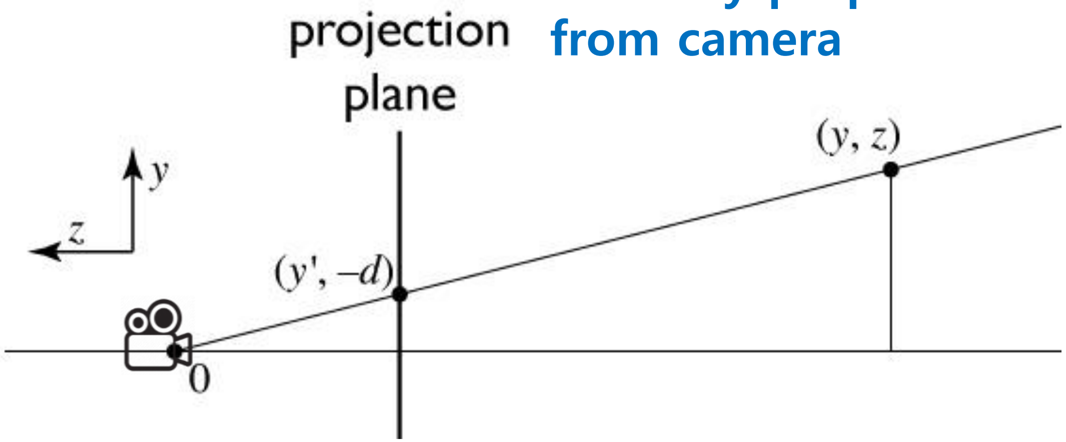
Let's first consider 3D View Frustum→2D Projection Plane

- Consider the projection of a 3D point on the camera plane



Perspective projection

The size of an object on the screen is inversely proportional to its distance from camera



similar triangles:

$$\frac{y'}{d} = \frac{y}{-z}$$

$$y' = -dy/z$$

Homogeneous coordinates revisited

- Perspective requires division
  - that is **not** part of affine transformations
  - in affine, parallel lines stay parallel
    - therefore not vanishing point
    - therefore no rays converging on viewpoint
- “True” purpose of homogeneous coords: projection

Homogeneous coordinates revisited

- Introduced *w* = 1 coordinate as a placeholder

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

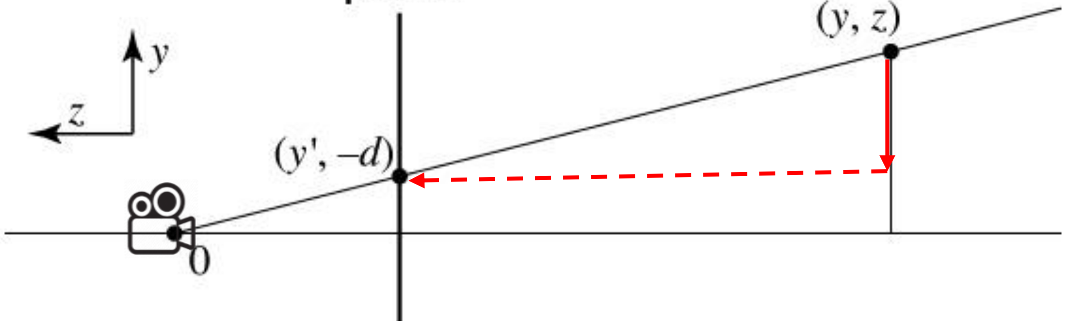
  - used as a convenience for unifying translation with linear transformation

- Can also allow arbitrary *w*

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

All scalar multiples of a 4-vector are equivalent

Perspective projection

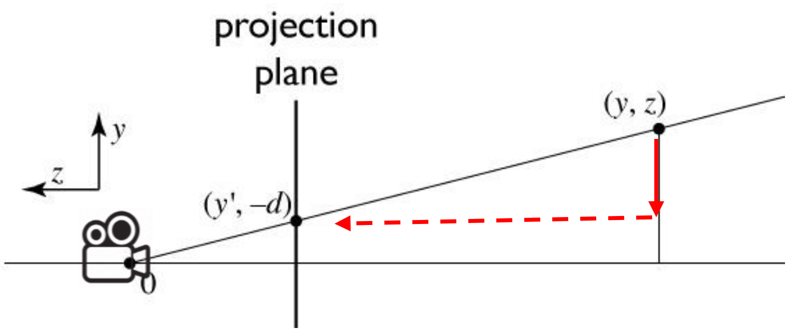


to implement perspective, just move z to w:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ 1 \end{bmatrix} \sim \begin{bmatrix} dx \\ dy \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

So far, 3D → 2D

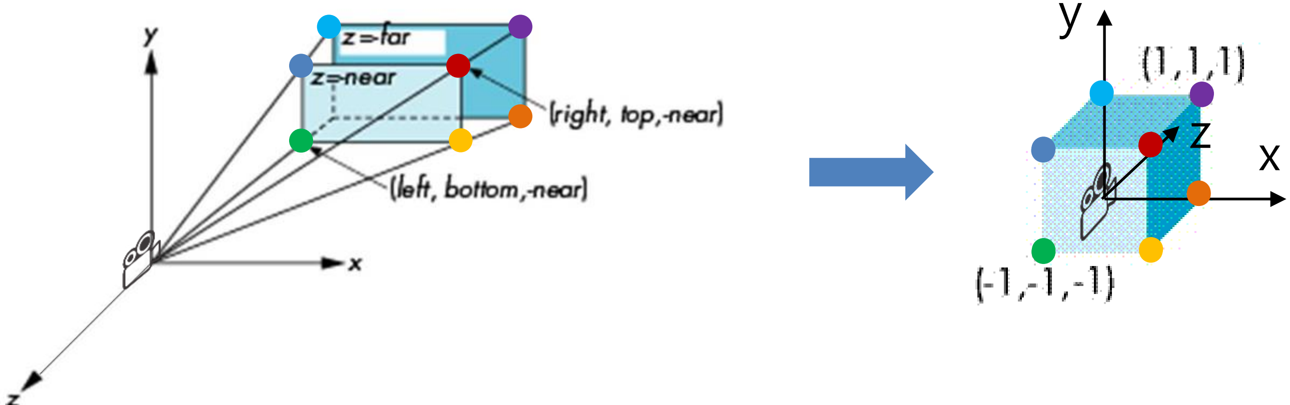
- What we've just seen is a story of
  3D View Frustum → 2D Projection Plane:



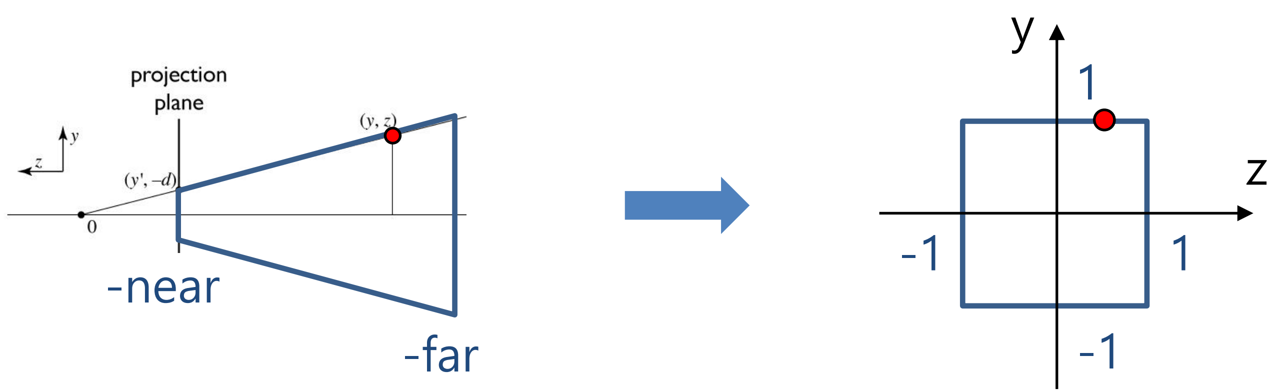
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ 1 \end{bmatrix} \sim \begin{bmatrix} dx \\ dy \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Now, 3D → 3D

- However, what we really have to do is
  3D View Frustum → 3D Canonical View Volume:

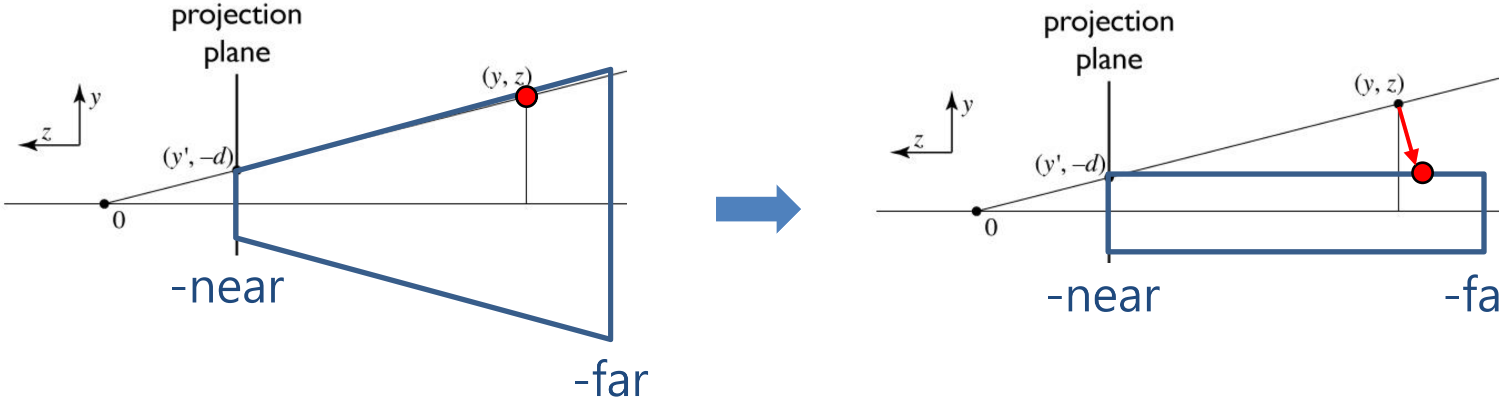


(side view)



First, 3D View Frustum → 3D Cuboid

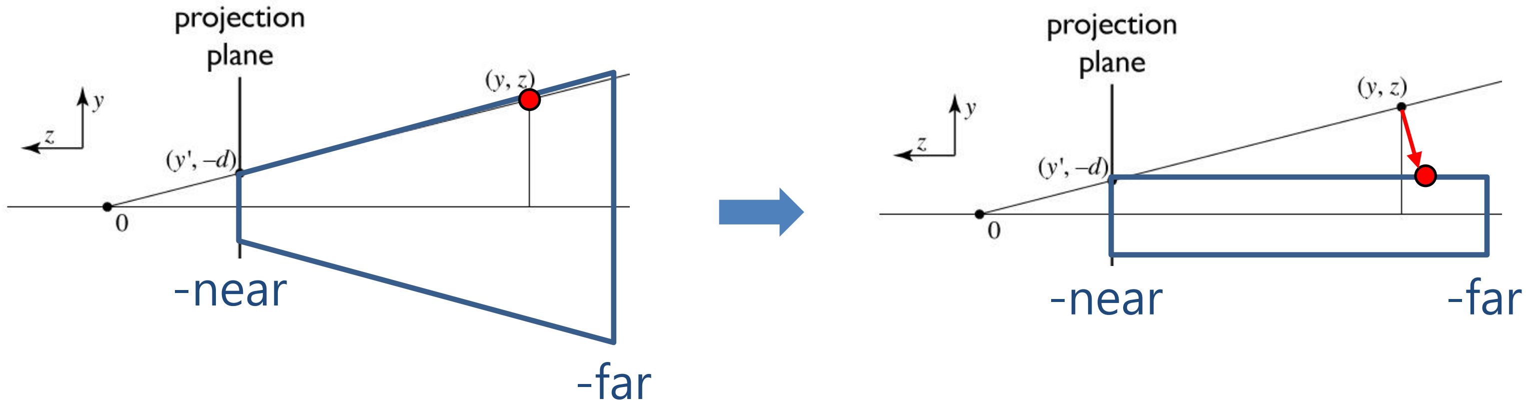
- Let’s first consider a viewing frustum → a cuboid with the same near and far offset (not a canonical view volume)



- Perspective has a varying denominator—can’t preserve depth z!
- Instead, let's map z values in such a way that the mapped z-values preserve depth in the near and far planes.

3D View Frustum → 3D Cuboid

- Let’s first consider a viewing frustum → a cuboid with the same near and far offset (not a canonical view volume)

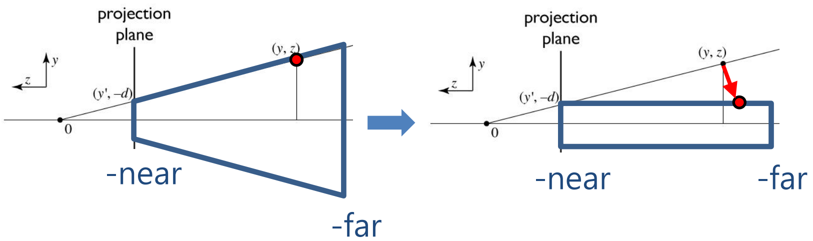


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ -\ X\ /z \\ 1 \end{bmatrix} \sim \begin{bmatrix} dx \\ dy \\ X \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ c_0 & c_1 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective has a varying denominator—can’t preserve depth z!

3D View Frustum → 3D Cuboid

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ -X/z \\ 1 \end{bmatrix} \sim \begin{bmatrix} dx \\ dy \\ X \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ c_0 & c_1 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



- Note that z' is independent of x and y, therefore $c_0 = c_1 = 0$

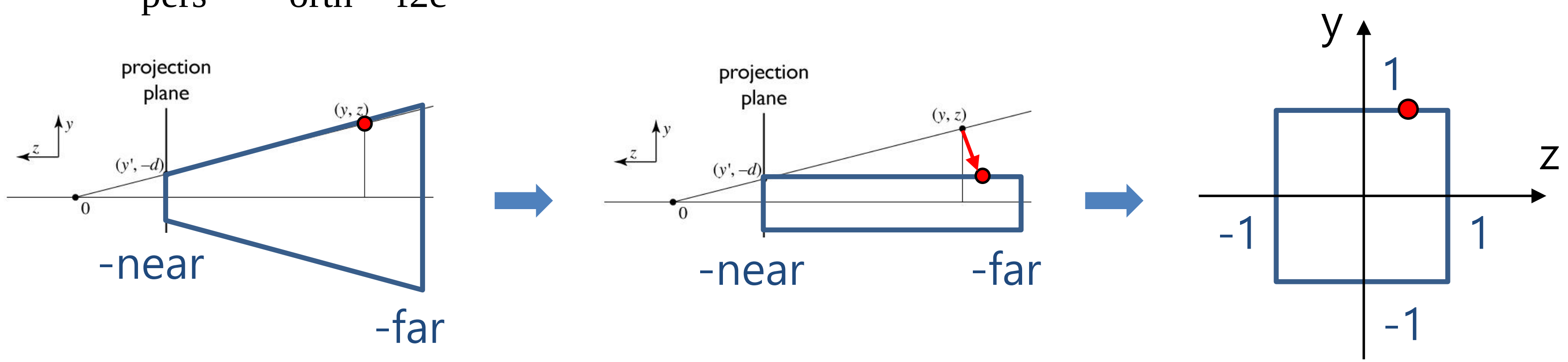
- We want z depth -near → -near, -far → -far.
- This means $z'(-\mathrm{n}) = -\mathrm{n}, \mathrm{z}'(-\mathrm{f}) = -\mathrm{f}$, where $\mathrm{z}'(\cdot)$ is defined as:

$$X = az + b \qquad z'(z) = -X/z = -\frac{az+b}{z}$$

- There are 2 unknowns a, b and 2 equations $z'(-\mathrm{n}) = -\mathrm{n}, \mathrm{z}'(-\mathrm{f}) = -\mathrm{f}$, so we can solve it:
- → a = f+n, b = fn (try it)

Final: 3D View Frustum → 3D Canonical View Volume

- By substituting d with n, $P_{f2c} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & f+n & fn \\ 0 & 0 & -1 & 0 \end{bmatrix}$
- Now the remaining work is **mapping the cuboid to a canonical view volume: P_{orth}**
- Viewing frustum → cuboid → canonical view volume: $P_{pers} = P_{orth} P_{f2c}$



Perspective Projection Matrix

- $P_{pers} = P_{orth}\, P_{f2c}$

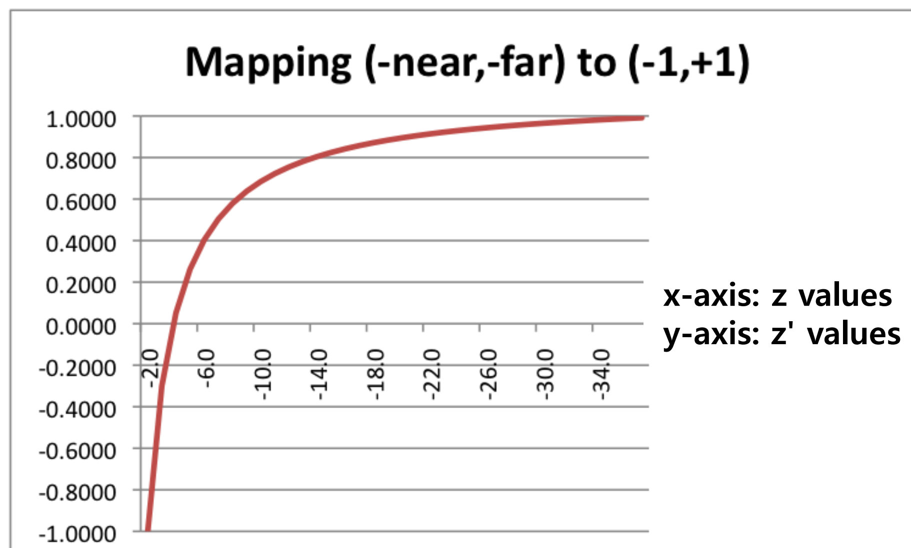
$$= \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & f+n & fn \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Note on Mapped Depth (Z' value)

- This perspective projection results in **non-linear mapping** of the original depth values (z values) to the range (-1, +1).

$$\begin{bmatrix} x' \\ y' \\ \boxed{z'} \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ \underline{-\ X/z} \\ 1 \end{bmatrix} \sim \begin{bmatrix} dx \\ dy \\ X \\ -z \end{bmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- This makes more precision for z values close to the camera and less precision for z values farther from the camera.



Perspective Division, Clip / NDC Space

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ -X/z \\ 1 \end{bmatrix} \xleftarrow{\text{perspective division}} \begin{bmatrix} dx \\ dy \\ X \\ -z \end{bmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

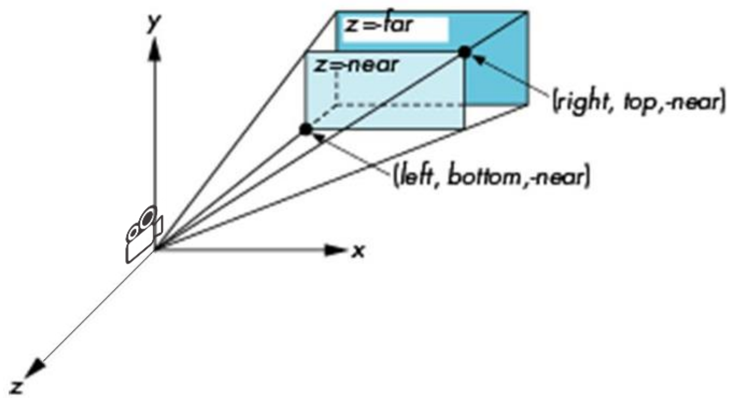
in **NDC space** | **perspective division** | in **clip space** | in **camera space**

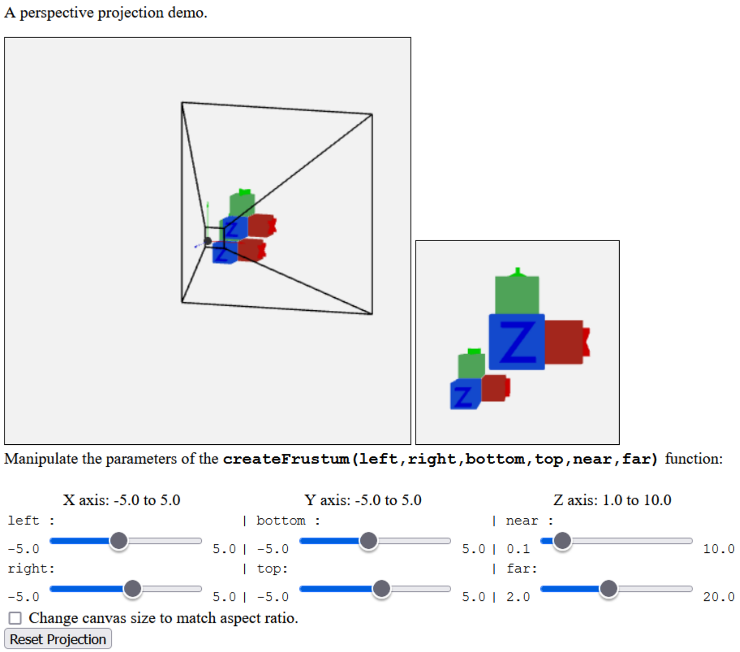
- **Clip space** is a **4D homogeneous coordinate system space** that represents the scene <u>after being transformed by the **vertex shader**</u>.

- **NDC space** is a **3D coordinate system space** that represents the scene <u>after being transformed from clip space by the **perspective division**</u>.

- Actually, <u>both spaces represent the same "space"</u>(visible region), a cube with a range of [-1,1], but in NDC space the fourth dimension (w) has been removed.

[Demo] Perspective Projection - *frustum*

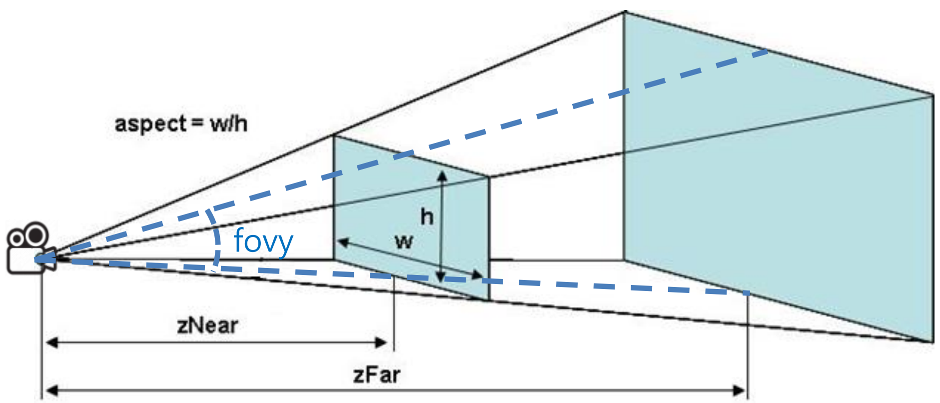


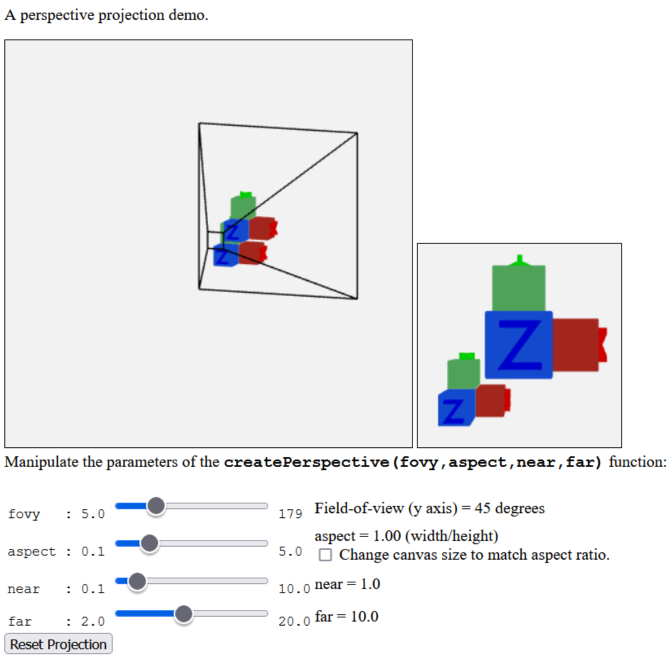


http://learnwebgl.brown37.net/08_projections/create_frustum/create_frustum.html

- Observe the view volume (left) and rendered view (right) while dragging left, right, bottom, top, near, far sliders.

[Demo] Perspective Projection - *perspective*





http://learnwebgl.brown37.net/08_projections/create_perspective/create_perspective.html

- Observe the view volume (left) and rendered view (right) while dragging fovy, aspect, near, far sliders.
- Which one is more convenient, *frustum* or *perspective*?

Quiz 2

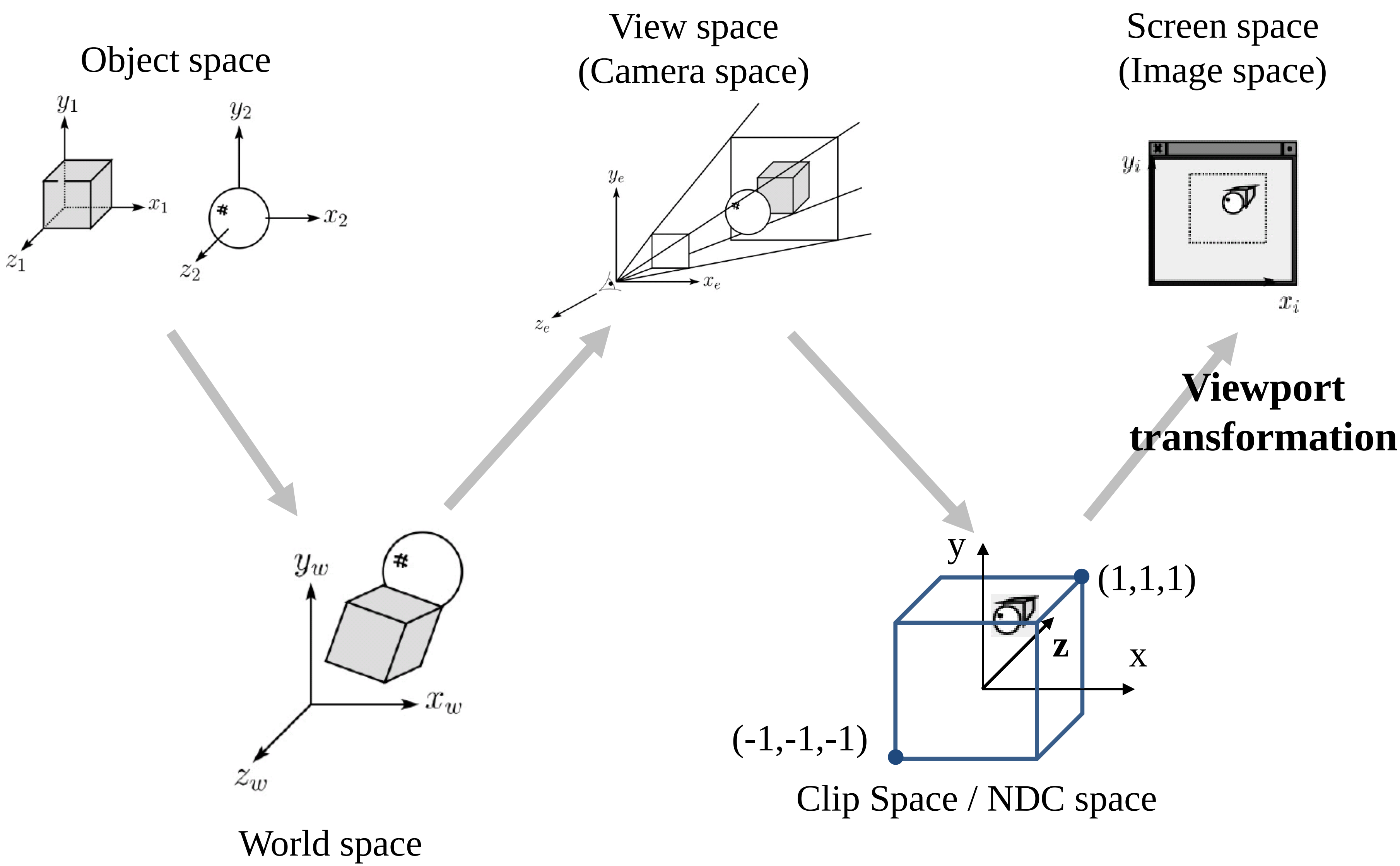
- Go to https://www.slido.com/
- Join #**cg-ys**
- Click "Polls"

- 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!

Viewport Transformation

Viewport Transformation



Recall that...

- 1. Placing objects

→ **Modeling transformation**

- 2. Placing a "camera"

→ **Viewing transformation**

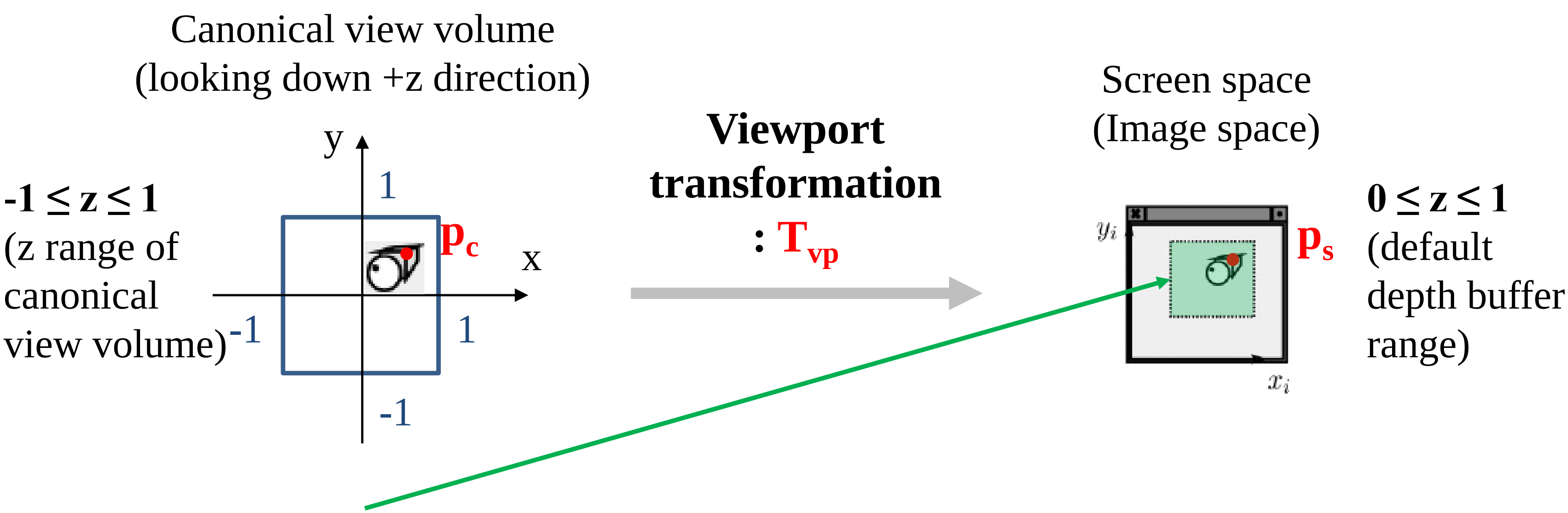
- 3. Selecting its "lens"

→ **Projection transformation**

- 4. Displaying on a "cinema screen"

→ **Viewport transformation**

Viewport Transformation



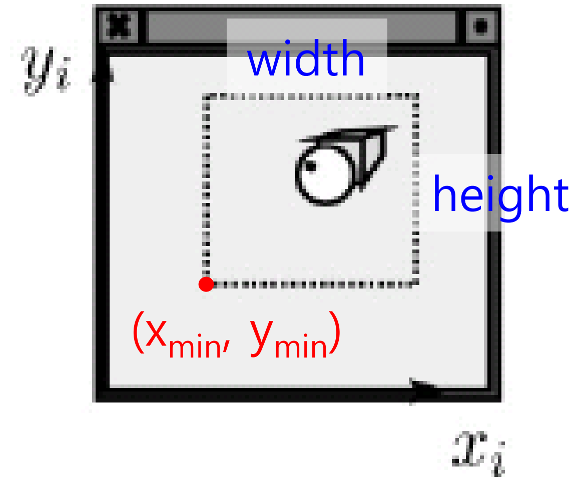
- Viewport: a rectangular viewing region of screen

- So, viewport transformation is also a kind of windowing transformation.

Viewport Transformation Matrix

- In the windowing transformation matrix,
- By substituting x_h, x_l, x_h', ... with corresponding variables in viewport transformation,

$$T_{vp} = \begin{bmatrix} \frac{width}{2} & 0 & 0 & \frac{width}{2} + x_{min} \\ 0 & \frac{height}{2} & 0 & \frac{height}{2} + y_{min} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Lab Session

- Now, let's start the lab today.