
Computer Graphics

6 - Vertex Processing 2

Yoonsang Lee
Hanyang University

Spring 2026

Midterm Exam Announcement

- Date & time: **May 4 (Mon), 10:00 - 11:00 AM**
- Place: IT.BT 508
- Scope: Lecture, Lab, Lab Check 2~7
- **You cannot leave until 30 minutes** after the start of the exam even if you finish the exam earlier.
- That means, **you cannot enter the room after 30 minutes** from the start of the exam (do not be late, never too late!).
 - This policy is to prevent early finishers from sharing exam information with late arrivals.
- Bring your **student ID card** to the exam.

Midterm Exam Announcement

- **No digital or electronic devices** are allowed during the exam.
- This includes (but is not limited to): smartphones, smartwatches, tablets, laptops, wireless earphones, and smart glasses.
- All devices must be **powered off and kept in your bag**.
- **Do not wear any electronic devices** during the exam.
- Use of a device will be treated as **academic misconduct**.
- If you are unsure whether a device is allowed, **do not bring it**.

Inverting Rigid Transformation Matrix

- 4x4 rigid trans. matrix: $T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix}}_{\text{translation}} \underbrace{\begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix}}_{\text{rotation}}$

$$(AB)^{-1} = B^{-1}A^{-1}$$

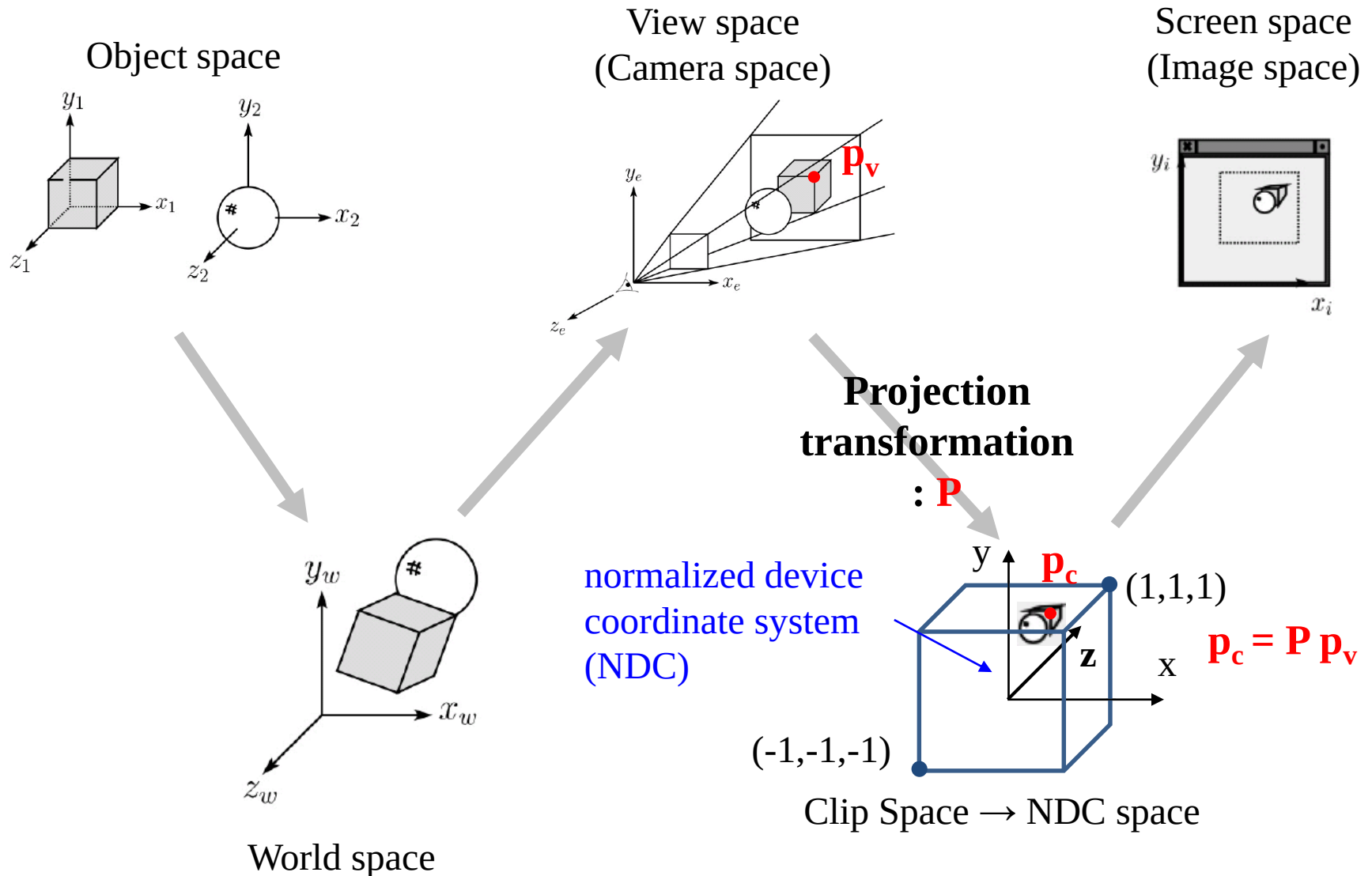
$$\begin{aligned} T^{-1} &= \left(\begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \right)^{-1} = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix}^{-1} \\ &= \begin{bmatrix} R^T & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I & -t \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} R^T & -R^T t \\ 0 & 1 \end{bmatrix} \end{aligned}$$

Outline

- Projection Transformation
 - Orthographic Projection
 - Perspective Projection
- Viewport Transformation

Projection Transformation

Projection Transformation

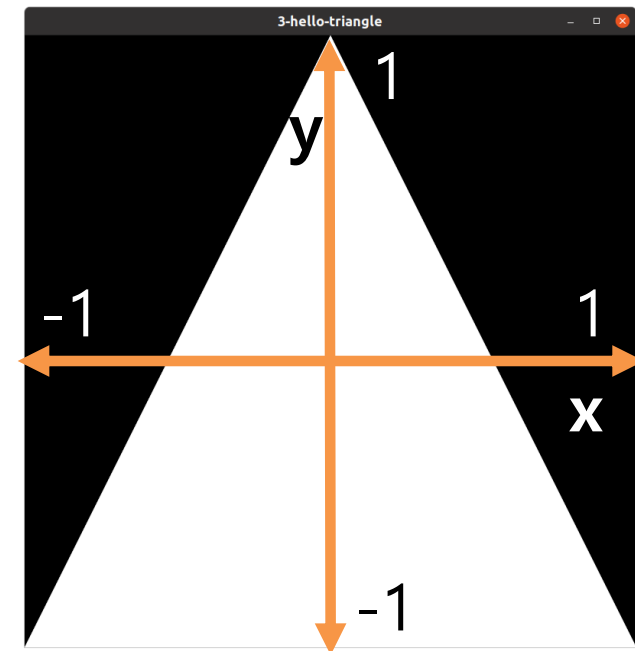
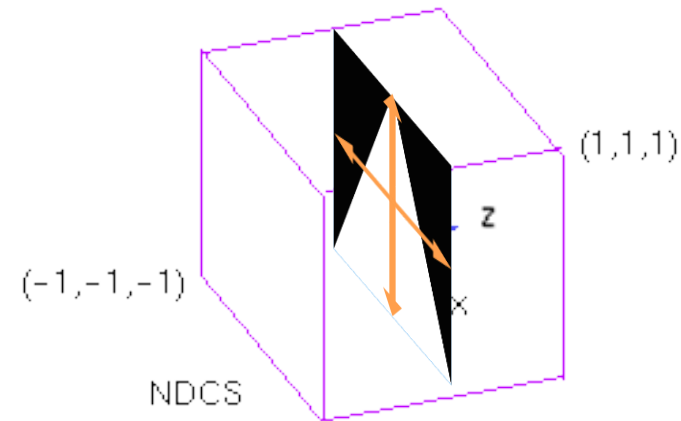


Recall that...

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

Recall: OpenGL Clip Space / NDC

- Vertices are mapped into a cube where x, y, z range from -1 to 1 .
→ Clip Space / NDC
- Its xy plane corresponds to the 2D screen.
- Its coordinate system is *normalized device coordinate (NDC)*.

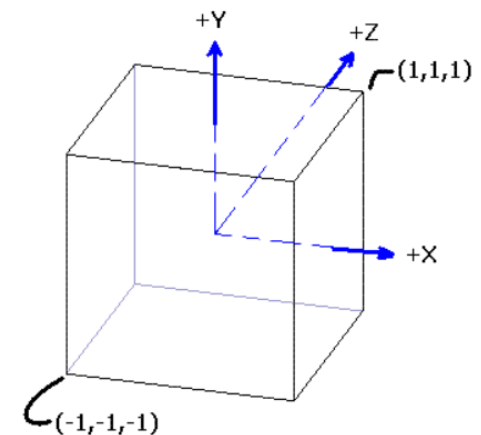
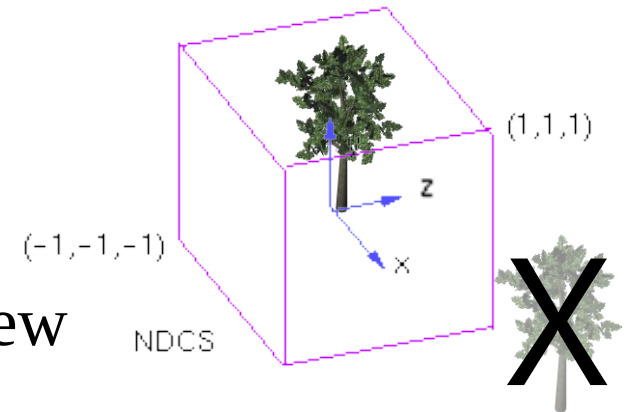


Normalized Device Coordinates (NDC)

- *Normalized device coordinates (NDC)* is a device independent display coordinate system.
 - Display resolutions (in pixels) vary across devices.
 - NDC provides a consistent coordinate system regardless of screen size.
- (Note)
 - Clip space and NDC are related but slightly different.
 - This will be explained in today's lecture.

Canonical View Volume

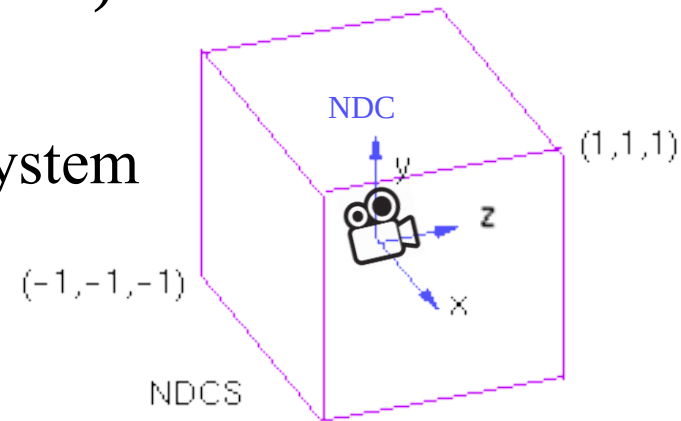
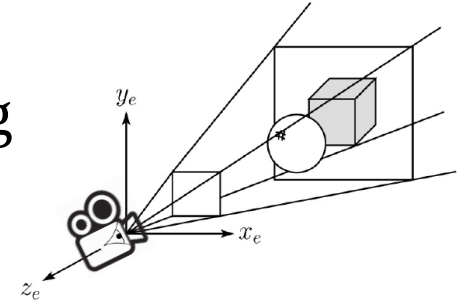
- The *canonical view volume* is the visible region in NDC space.
- Only objects **inside** the canonical view volume are rendered.
 - Objects too near, too far, or outside the view are excluded.
- (Note)
 - The exact definition of this volume may vary depending on the graphics system. (e.g., OpenGL vs Direct3D)



OpenGL canonical view volume

Canonical View Volume

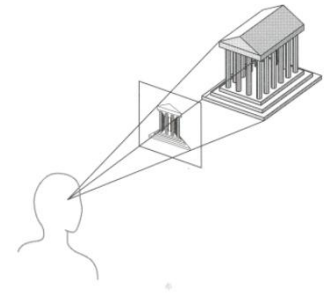
- View space (in OpenGL): camera looks along $-z$ (right-handed)
- NDC: camera looks along $+z$ (left-handed)
- $\rightarrow$ Projection changes the coordinate system handedness.
- (Note)
 - These conventions may differ across systems.
(e.g., OpenGL vs Direct3D)



View Volume

- In view space, objects are not restricted to the range $[-1, 1]$.
- You define a **view volume** (e.g., a cuboid or frustum) and place objects inside it.
- This **view volume** in view space is mapped (projected) into the **canonical view volume** in NDC space.
- → **Projection transformation**

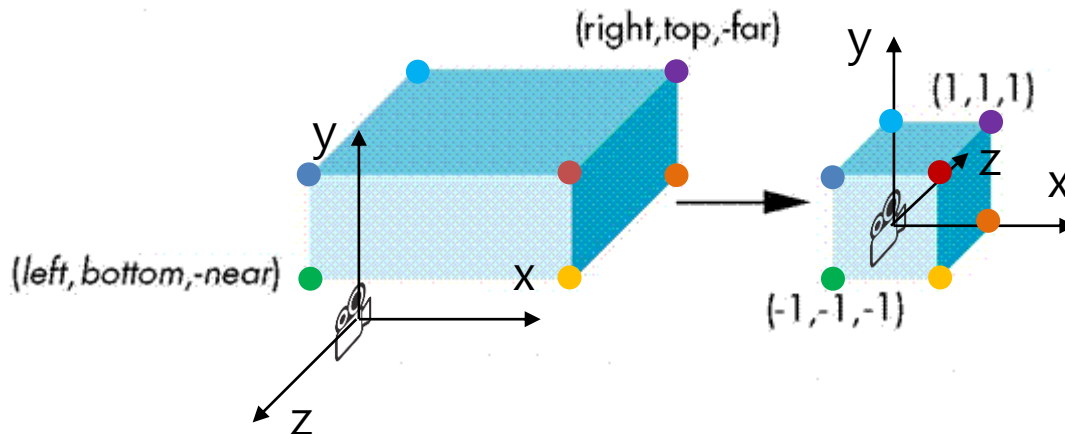
Projection Transformation



- Projection maps 3D coordinates to 2D screen coordinates.
- Projection transformation: Map a **view volume** to the **canonical view volume**
- Conceptually, we interpret the result as a 2D projection onto the xy plane, while keeping the z-value for depth testing.
- Two common projection transformation methods:
 - Orthographic projection
 - Perspective projection

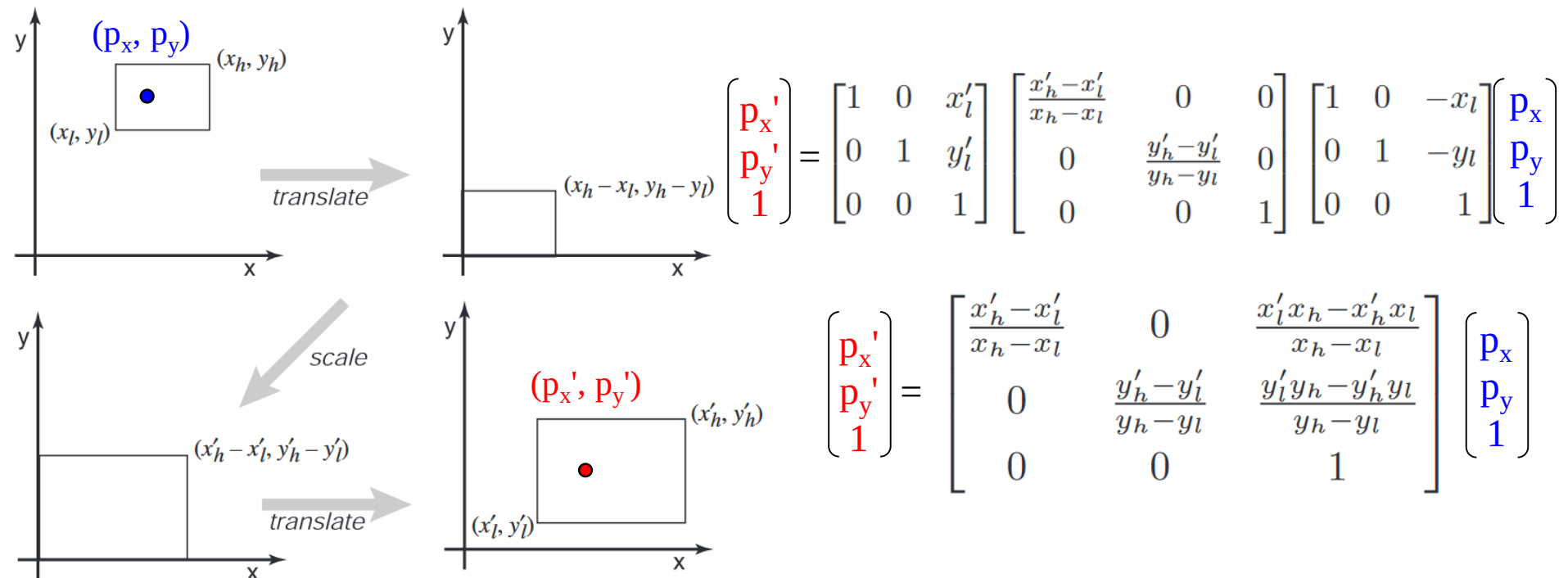
Orthographic (Orthogonal) Projection

- View volume: Cuboid (직육면체)
- Orthographic projection:
cuboid view volume → **canonical view volume**
- No perspective effect (size does not change with distance)
- Combination of scaling & translation: "Windowing" transformation



Windowing Transformation

- Map one rectangle to another rectangle
 (p_x, p_y) in $(x_l, y_l) \times (x_h, y_h)$
 $\rightarrow (p'_x, p'_y)$ in $(x'_l, y'_l) \times (x'_h, y'_h)$
- Achieved by: translate $\rightarrow$ scale $\rightarrow$ translate

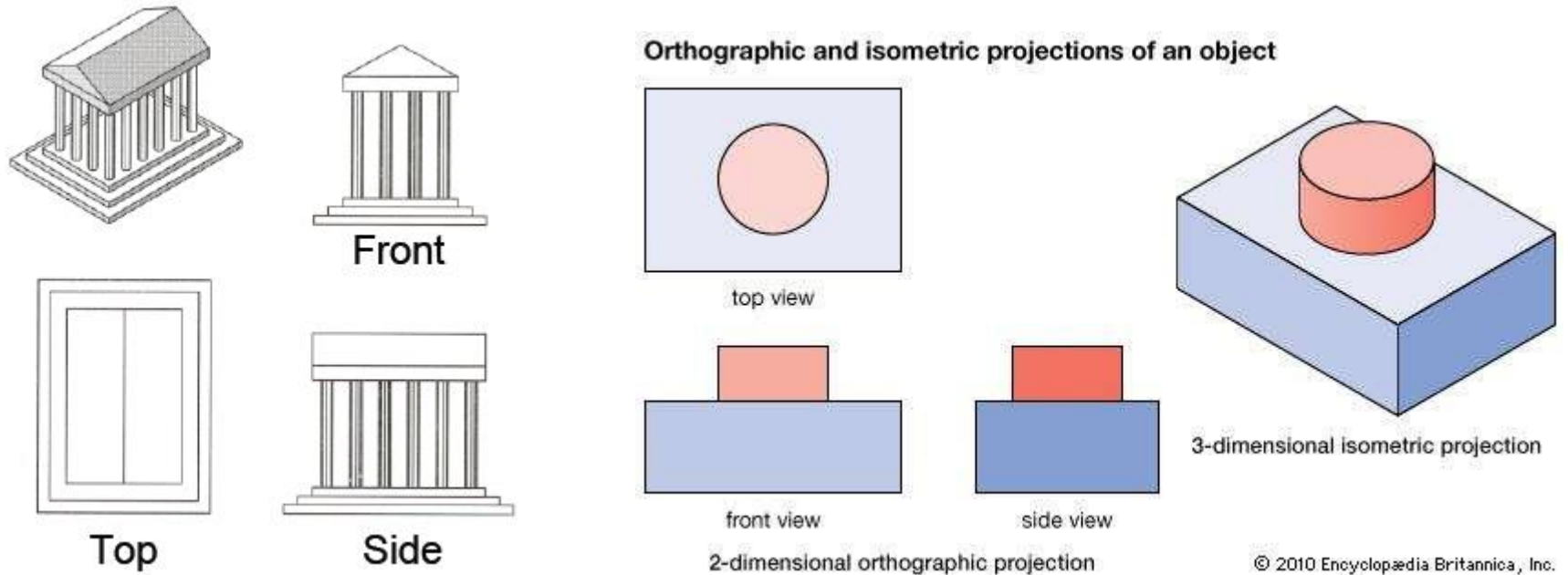


Orthographic Projection Matrix

- Map a cuboid view volume $[\text{left}, \text{right}] \times [\text{bottom}, \text{top}] \times [\text{near}, \text{far}]$ to the canonical view volume $[-1, 1]^3$.
- By extending the 2D windowing transformation to 3D and substituting:
 - $x_h = \text{right}, x_l = \text{left}, x_h' = 1, x_l' = -1$
 - $y_h = \text{top}, y_l = \text{bottom}, y_h' = 1, y_l' = -1$
 - $z_h = -\text{far}, z_l = -\text{near}, z_h' = 1, z_l' = -1$
 - (Note: In view space, near and far correspond to $z = -\text{near}$ and $z = -\text{far}$.)

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

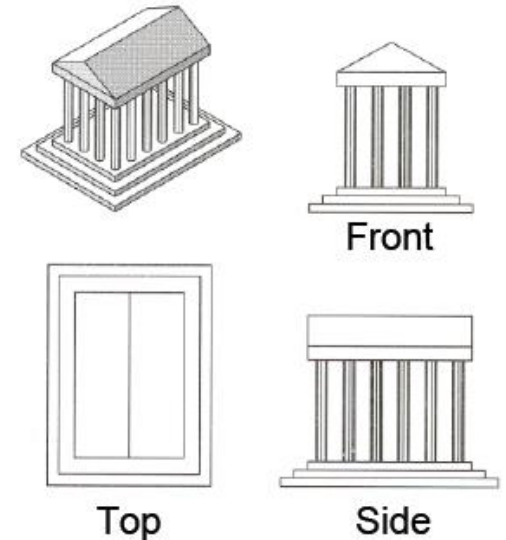
Examples of Orthographic Projection



- No perspective effect: objects do not get smaller with distance.

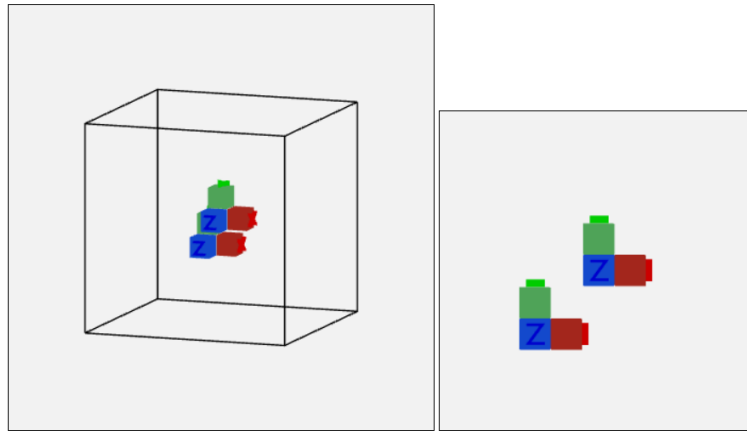
Properties of Orthographic Projection

- Less realistic (no perspective effect)
- No size change with distance
→ suitable for accurate measurement
- Parallel lines remain parallel
- Widely used in CAD and architectural drawings
- Combination of scaling and translation
→ Affine transformation (no perspective distortion)

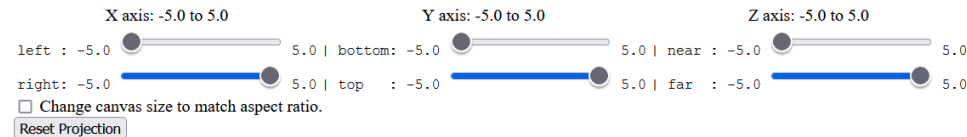


[Demo] Orthographic Projection

An orthographic projection demo.



Manipulate the parameters of the `createOrthographic(left, right, bottom, top, near, far)` function:



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

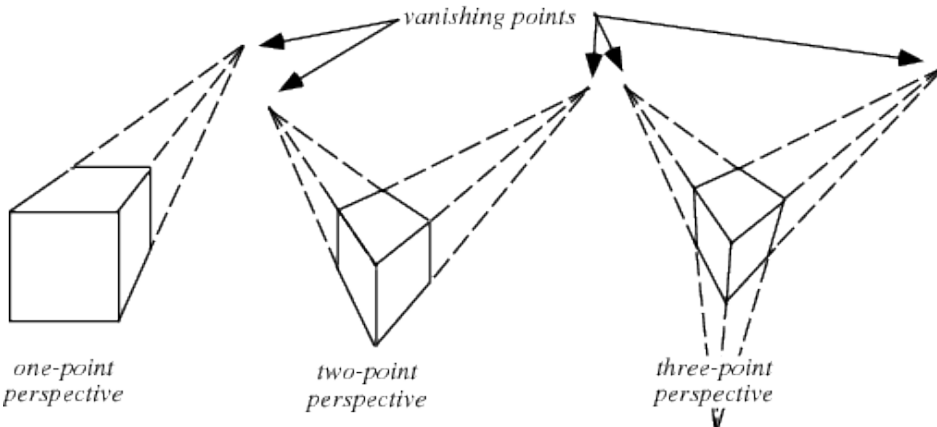
- Observe the view volume (left) and rendered view (right) while dragging left, right, bottom, top, near, far sliders.

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. **2024123456: 4.0**
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

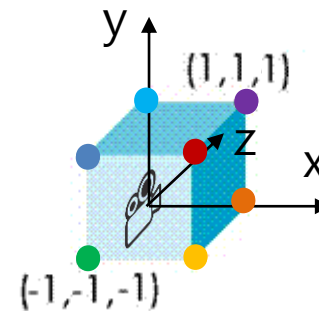
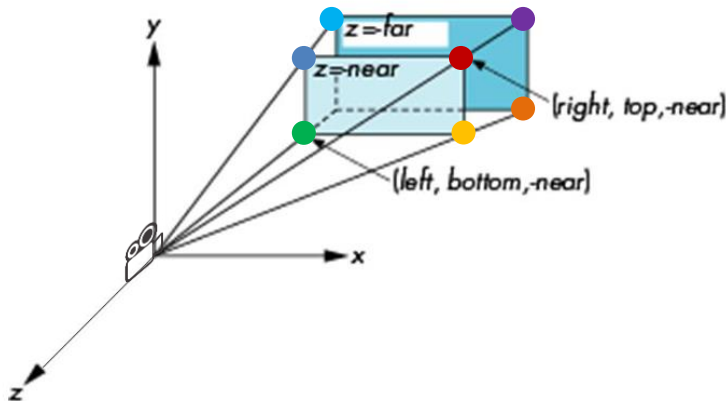
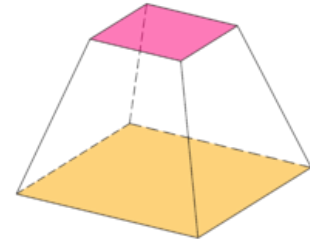
Perspective Effects

- Unlike orthographic projection, **distant objects appear smaller.**
- Parallel lines appear to converge to a **vanishing point.**



Perspective Projection

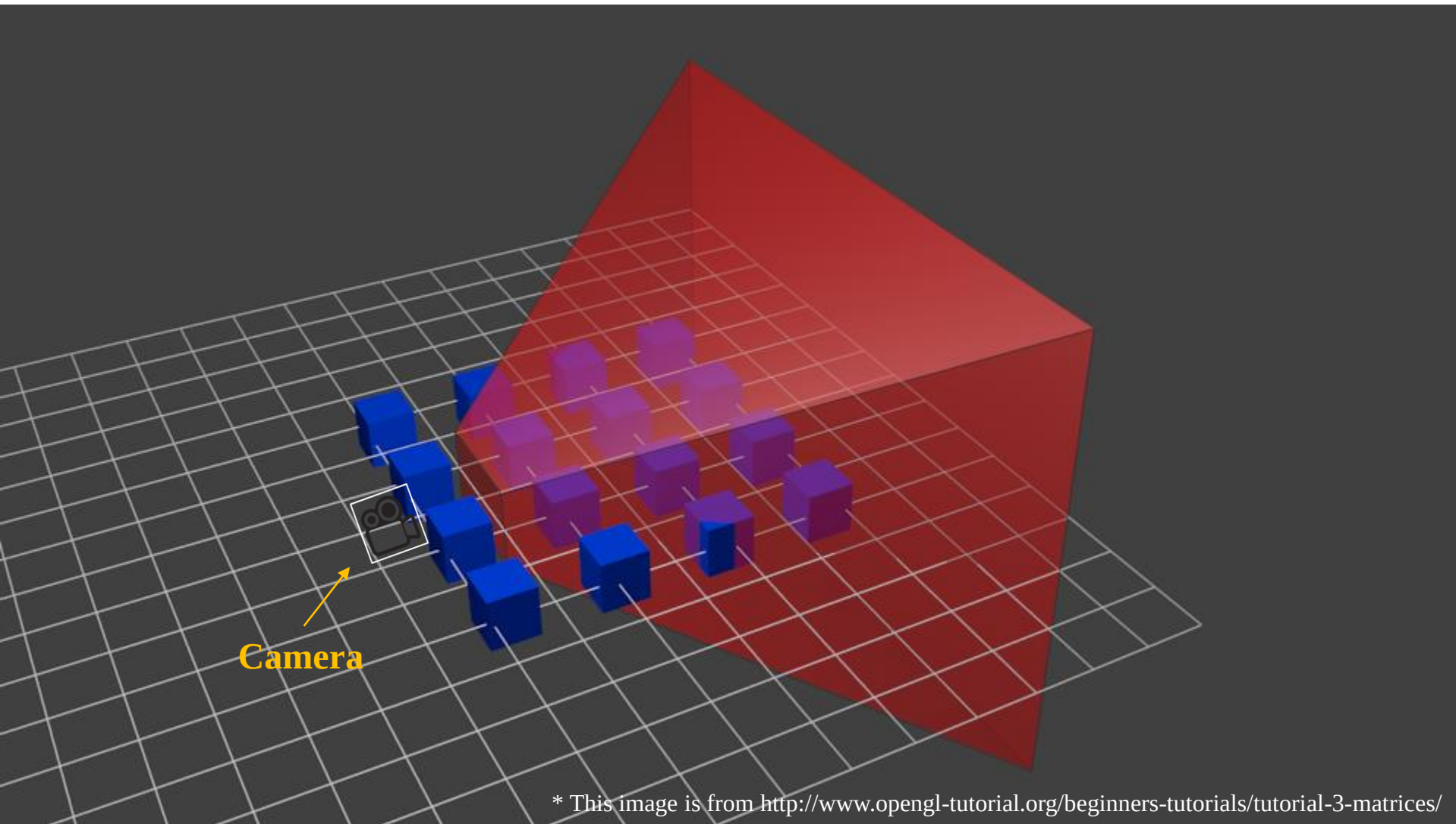
- View volume: Frustum (절두체)
→ “Viewing frustum”
- Perspective projection:
viewing frustum → **canonical view volume**
→ followed by *perspective division* (divide by w)
- Projection lines converge at the eye (camera)



Why does this mapping generate a perspective effect?

Original 3D scene

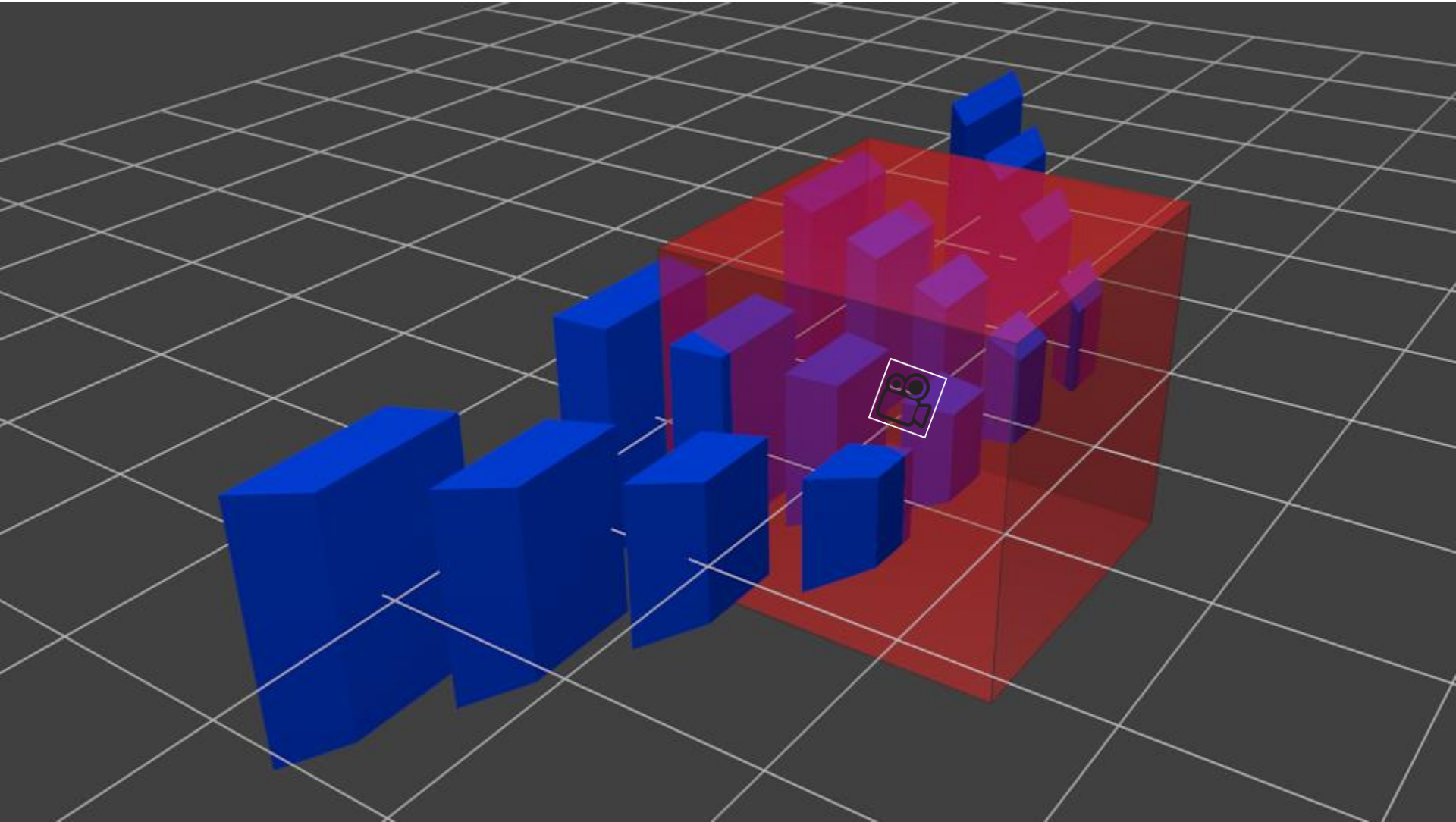
Red: viewing frustum, Blue: objects



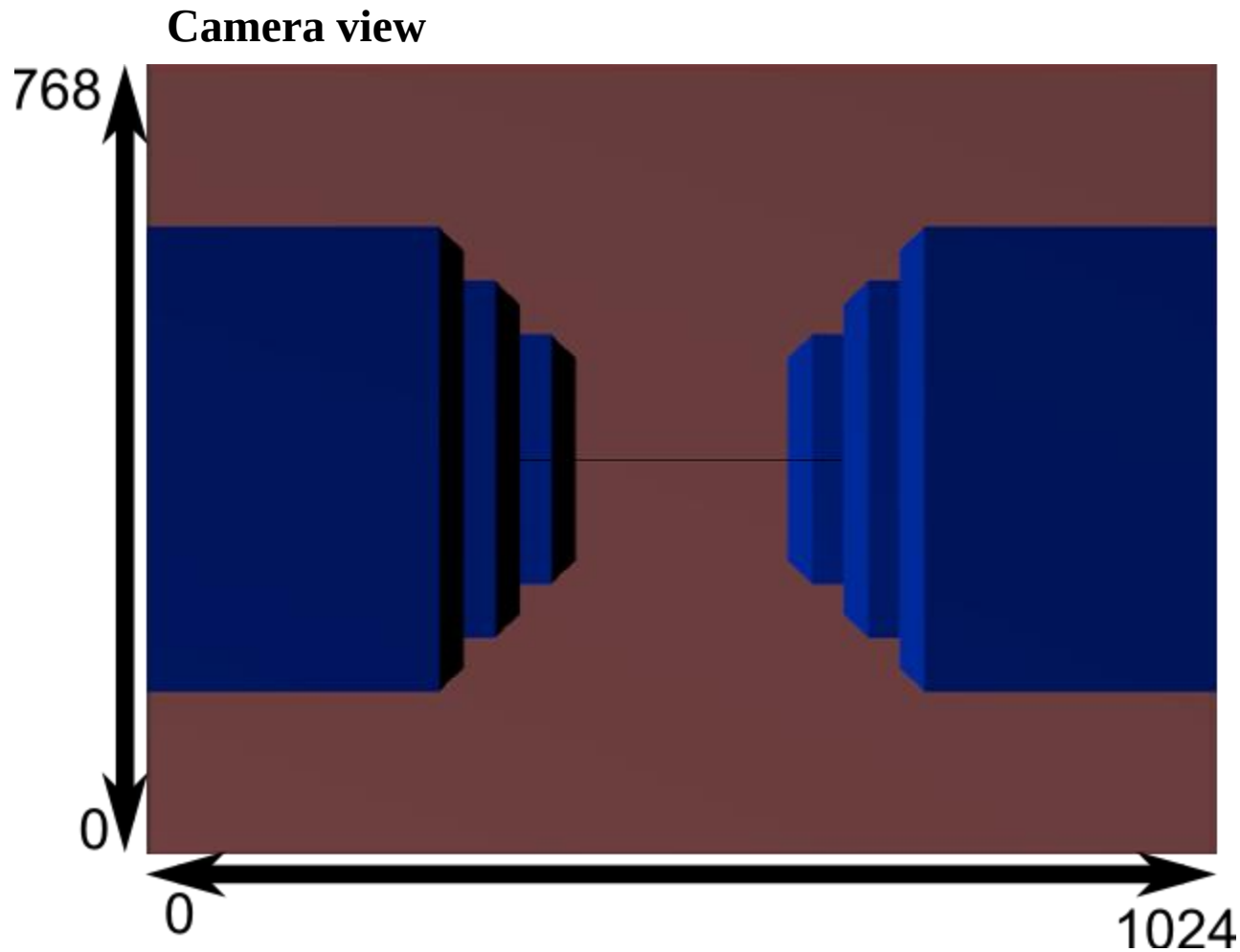
* This image is from <http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/>

An Example of Perspective Projection

After perspective projection

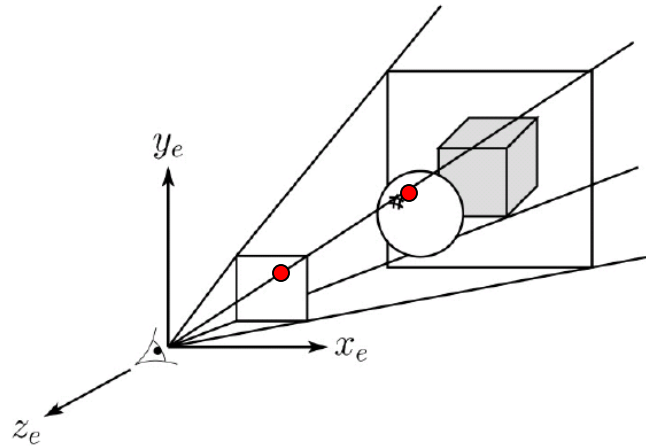


An Example of Perspective Projection

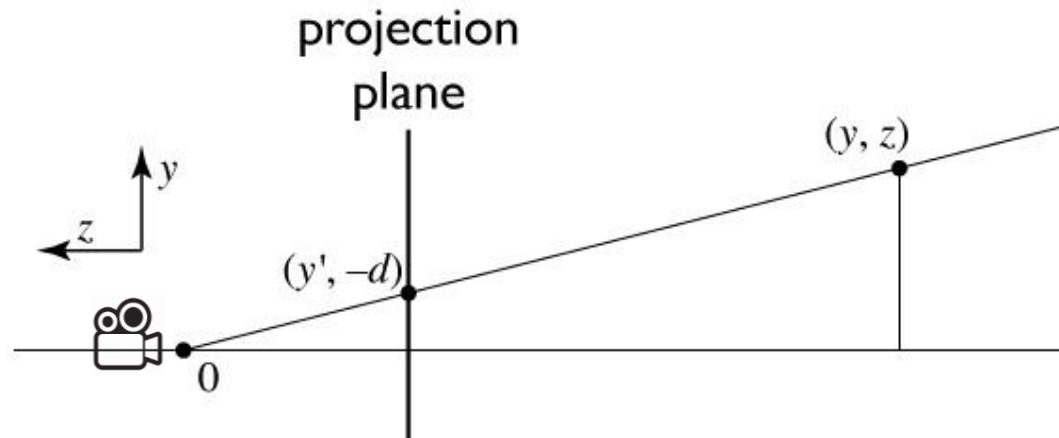


Let's first consider 3D View Frustum→2D Projection Plane

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



Perspective Projection



From similar triangles:

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

$$y' = -\underline{dy/z}$$

- Objects farther away (larger $|z|$) appear smaller

$$\rightarrow \text{Size} \propto 1 / |z|$$

- Projection involves **division by z**

Perspective is Not Affine

- Perspective projection involves division (x/z , y/z)
→ NOT an affine transformation
- In affine transformations:
 - Parallel lines remain parallel
 - No vanishing points
- Therefore:
Perspective projection is fundamentally different

Why is this a Problem?

- Graphics pipelines rely on matrix transformations
- If projection is not affine:
 - cannot combine transformations using matrices (model, view, projection)
 - interpolate then project $\neq$ project then interpolate
 - attribute interpolation becomes incorrect
 - clipping becomes difficult
- This breaks the pipeline structure

Solution: Homogeneous Coordinates

- Idea: Convert division into matrix multiplication using homogeneous coordinates
- This allows:
 - projection as matrix multiplication
 - delay division until the final step
- Therefore, the pipeline remains matrix-based

Homogeneous Coordinates Revisited

- Represent a 3D point in homogeneous coordinates:

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

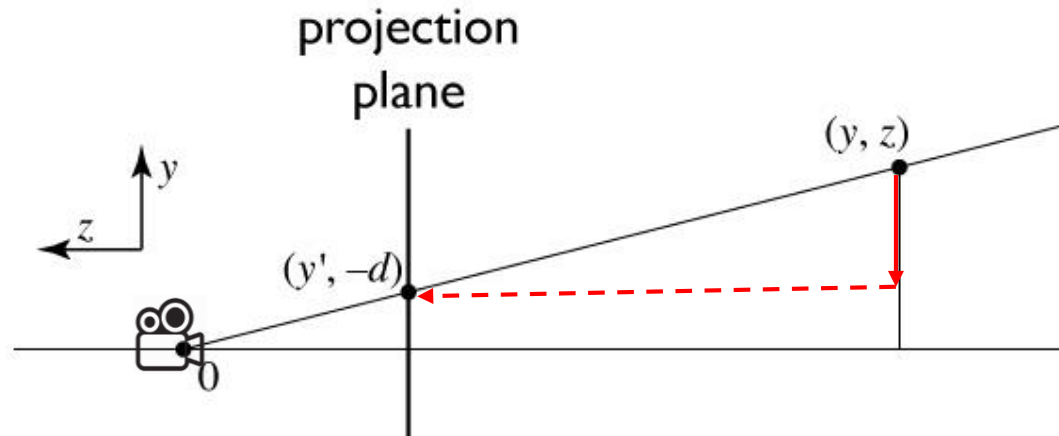
- Allow arbitrary scale w (homogeneous equivalence):
 - $\sim$ means equal up to a scale w
 - All scalar multiples represent the same point
 - After dividing by w , they map to the same 3D point

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

Homogeneous Coordinates Revisited

- Allow arbitrary scale w (homogeneous equivalence):
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$
- Key idea:
 - We can **move division into the w coordinate**
 - i.e., divide by w at the final step
- This enables:
Perspective projection via division by w

Perspective Projection using Homogeneous Coordinates



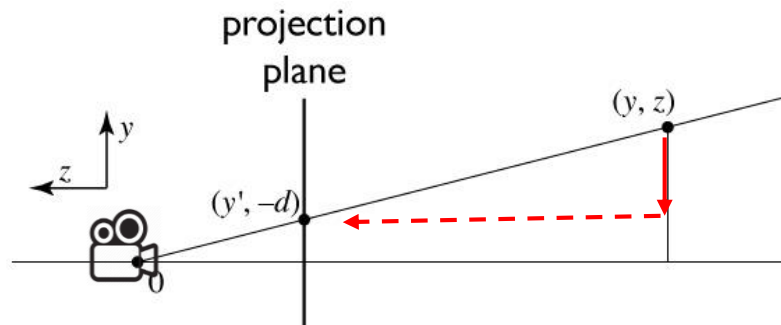
To implement perspective, encode depth (z) into w and divide later:

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

→ Perspective division: divide by w

So far, 3D $\rightarrow$ 2D

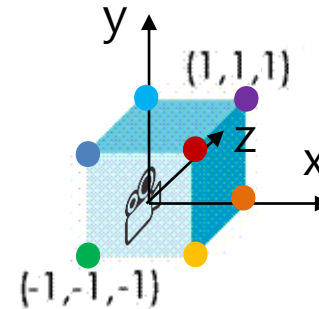
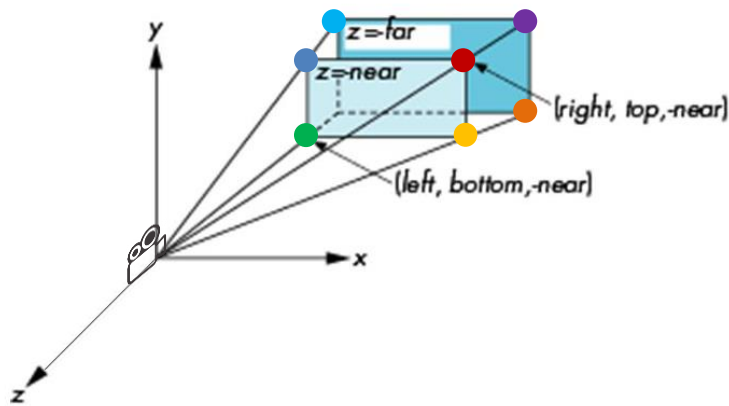
- What we've seen so far:
3D View Frustum $\rightarrow$ 2D Projection Plane



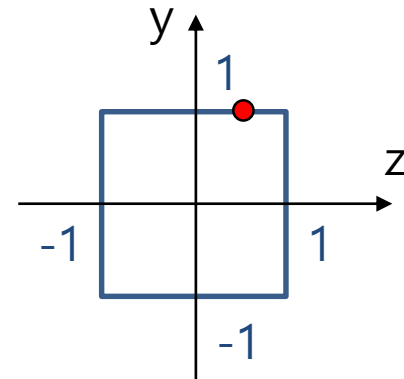
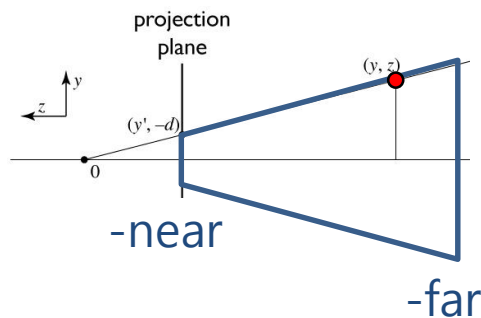
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ 1 \end{bmatrix} \sim \begin{bmatrix} dx \\ dy \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Now, 3D $\rightarrow$ 3D

- But in the graphics pipeline, we actually do:
3D View Frustum $\rightarrow$ 3D Canonical View Volume

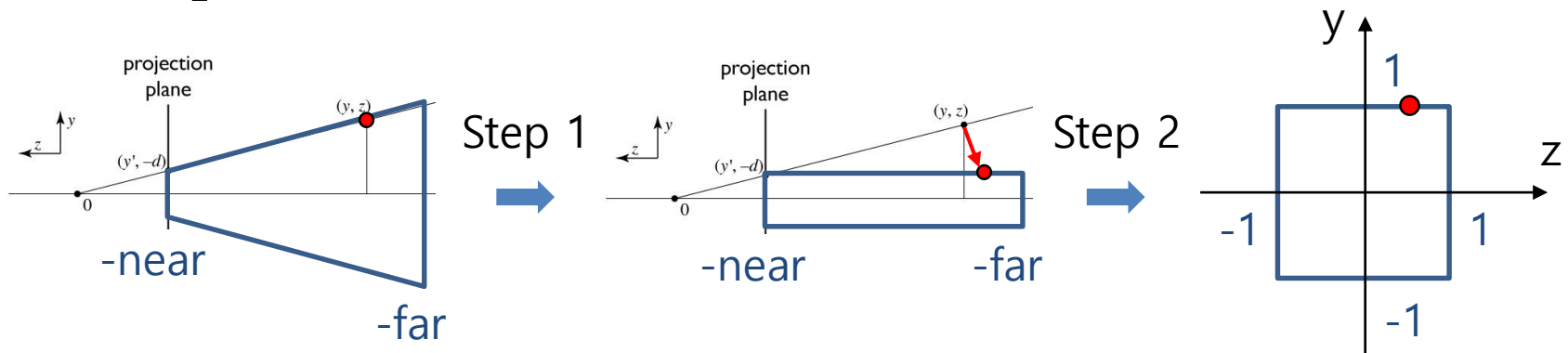


(side view)



Big Picture: Perspective Projection Pipeline

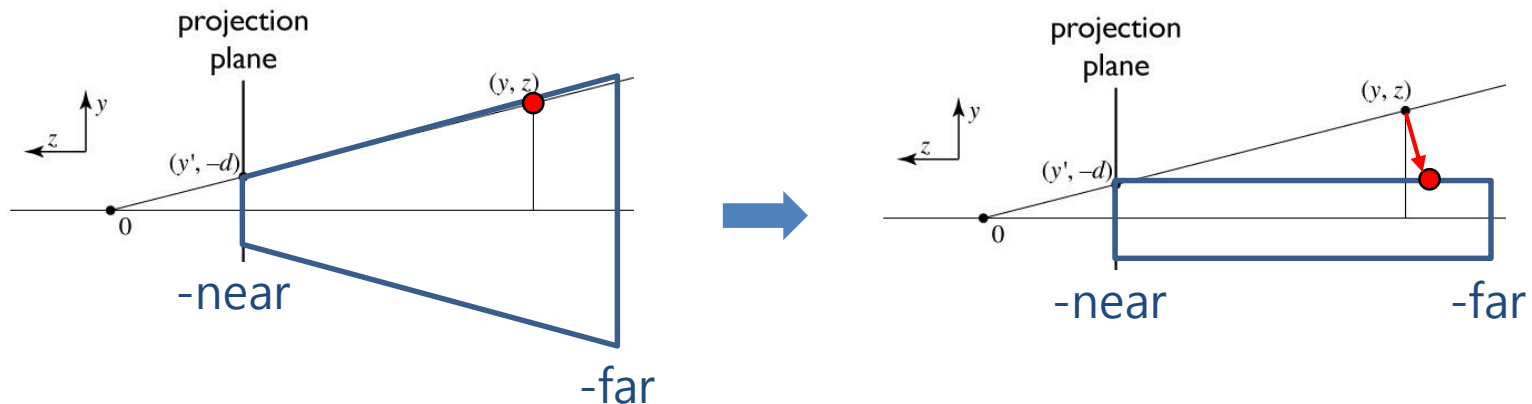
- We map the viewing frustum to a canonical cube in two steps:
- Step 1: Frustum $\rightarrow$ Cuboid
- Step 2: Cuboid $\rightarrow$ Canonical view volume



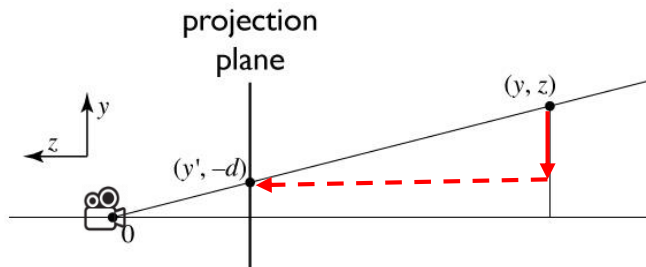
- Perspective projection can now be implemented using matrices

Step 1: Frustum → Cuboid (perspective effect)

- We transform the frustum into a cuboid while preserving near and far planes:



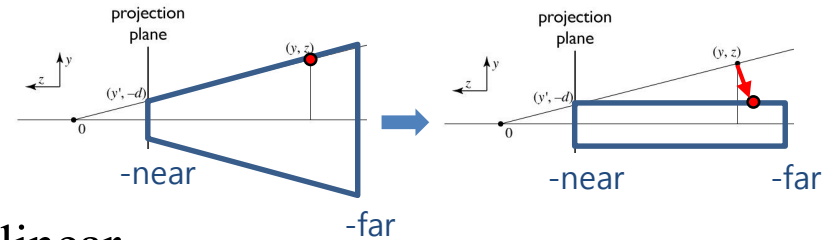
- For x , y coordinates (perspective effect):
 - $x' = -dx / z$, $y' = -dy / z$ → depends on z (nonlinear)



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

Step 1: Frustum $\rightarrow$ Cuboid (handling depth)

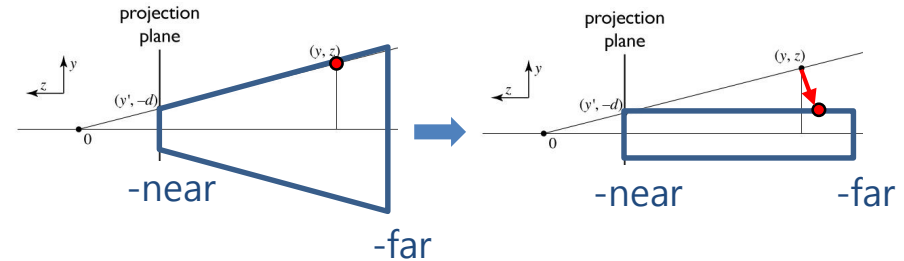
- The problem is depth (z):
 - After perspective projection:
 z' depends on division by $z \rightarrow$ nonlinear
 - Depth is distorted (not linear)
- To handle this, we remap z so that:
 - $z = -\text{near} \rightarrow -\text{near}$
 - $z = -\text{far} \rightarrow -\text{far}$
- $\rightarrow$ Cannot preserve depth exactly (nonlinear $1/z$),
so we match it at the boundaries (near/far)
- $\rightarrow$ We encode this mapping into the projection matrix



Step 1: Frustum → Cuboid (handling depth)

- We define z' as a function of z :

- $z' = f(z)$



- Since we use a matrix form, assume:

- Let $X = az + b$ (z' is linear in z , independent of x, y)

- $z' = -X / z$

- This leads to the projection matrix:

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

- We will determine a, b to match near and far planes

Step 1: Frustum → Cuboid (handling depth)

- Since we use a matrix form, assume:

- $X = az + b$ (z' is independent of x, y)

- $z' = -X / z$

- $$z'(z) = -X / z = -(az + b)/z \quad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ -X/z \\ 1 \end{bmatrix} \sim \begin{bmatrix} dx \\ dy \\ X \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- We want to match near and far:

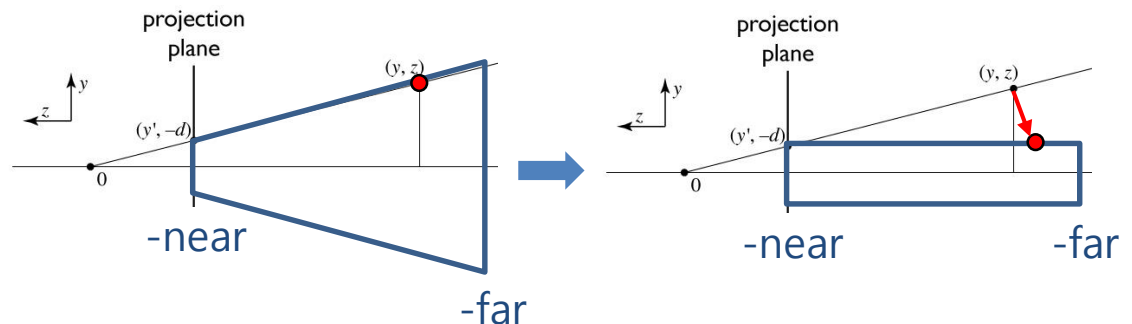
- $z'(-n) = -n$

- $z'(-f) = -f$

- Solving for a, b :

- $a = f + n$

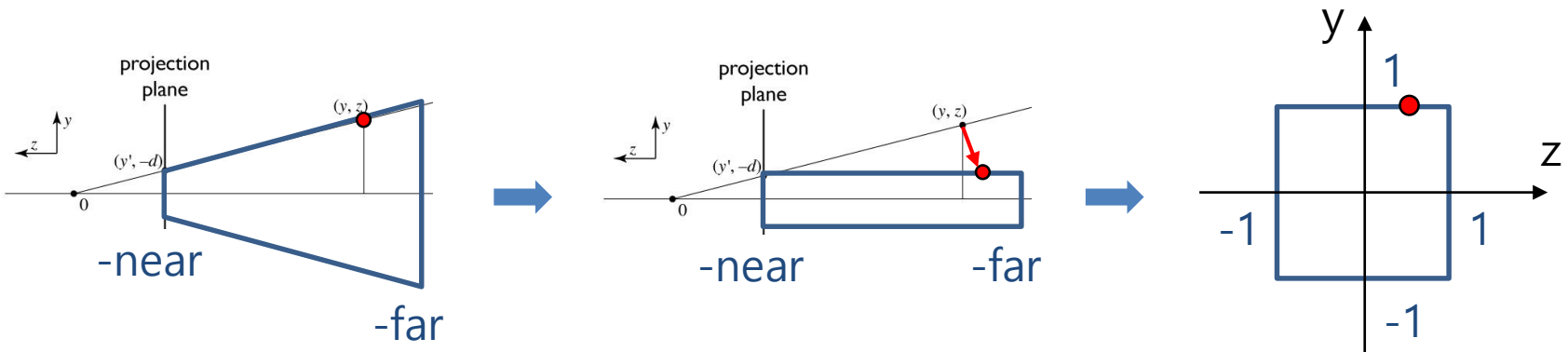
- $b = fn$



Step 1 + Step 2:

Frustum $\rightarrow$ Cuboid $\rightarrow$ Canonical view volume

- By setting $d = n$ (near plane distance), $P_{f2c} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & f + n & fn \\ 0 & 0 & -1 & 0 \end{bmatrix}$
- Remaining step: cuboid $\rightarrow$ canonical view volume (P_{orth})
- Final mapping:: $P_{pers} = P_{orth} P_{f2c}$



Perspective Projection Matrix

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

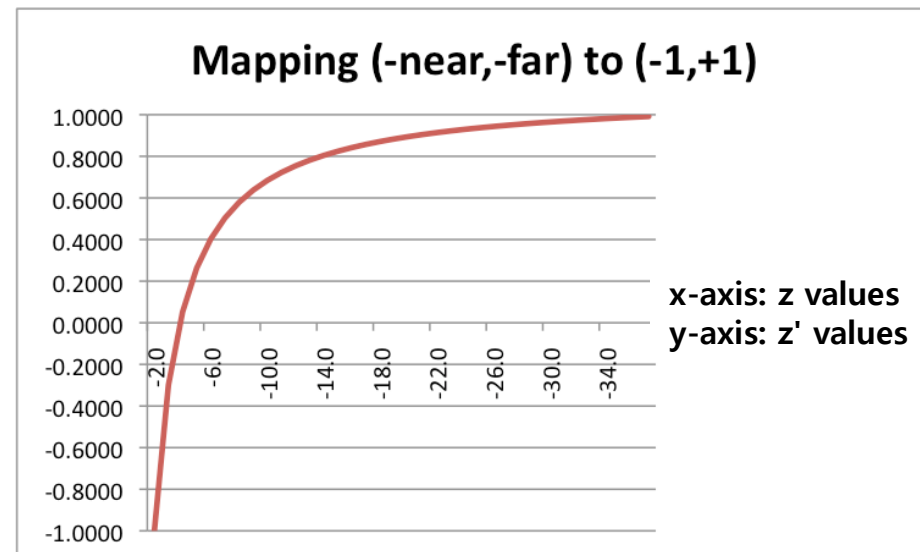
$$= \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & f+n & fn \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{pmatrix} \frac{2n}{r-l}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

Note on Mapped Depth (Z' value)

- Perspective projection produces a **non-linear mapping** of depth ($z \rightarrow z'$), where $z' \in (-1, +1)$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -A/z \\ -B/z \\ -C/z \\ 1 \end{bmatrix} \sim \begin{bmatrix} A \\ B \\ C \\ -z \end{bmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Key: $z' \propto 1/z$
- As a result:
 - More precision near the camera
 - Less precision farther away

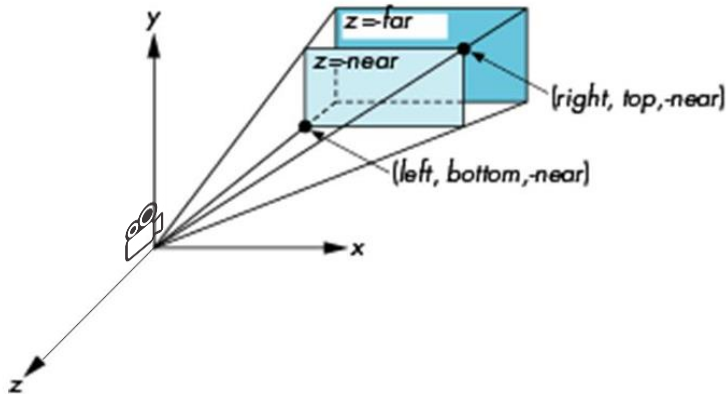


Perspective Division, Clip / NDC Space

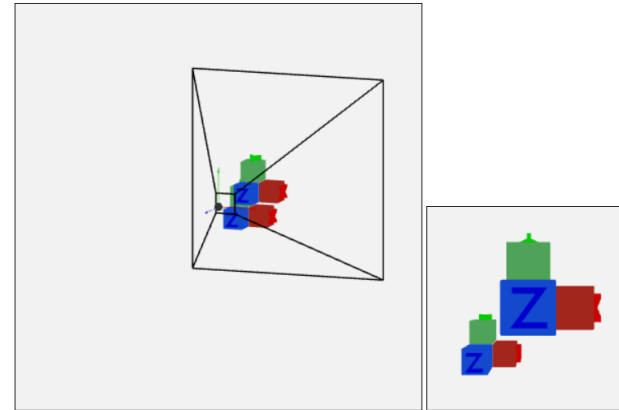
$$\begin{array}{ccc}
 \begin{array}{c} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -A/z \\ -B/z \\ -C/z \\ 1 \end{bmatrix} \\ \text{in NDC space} \end{array} & \begin{array}{c} \xleftarrow{\text{perspective division}} \\ \text{perspective division} \end{array} & \begin{array}{c} \begin{bmatrix} A \\ B \\ C \\ -z \end{bmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \\ \text{in clip space} & & \text{in view space} \end{array}
 \end{array}$$

- Clip space (4D):
 - Homogeneous coordinates (x, y, z, w)
 - Output of vertex shader
- NDC space (3D):
 - After perspective division
- Clip space and NDC represent the same visible region, which maps to the range [-1, 1] in NDC
 - NDC = clip space after dividing by w

[Demo] Perspective Projection - *frustum*



A perspective projection demo.



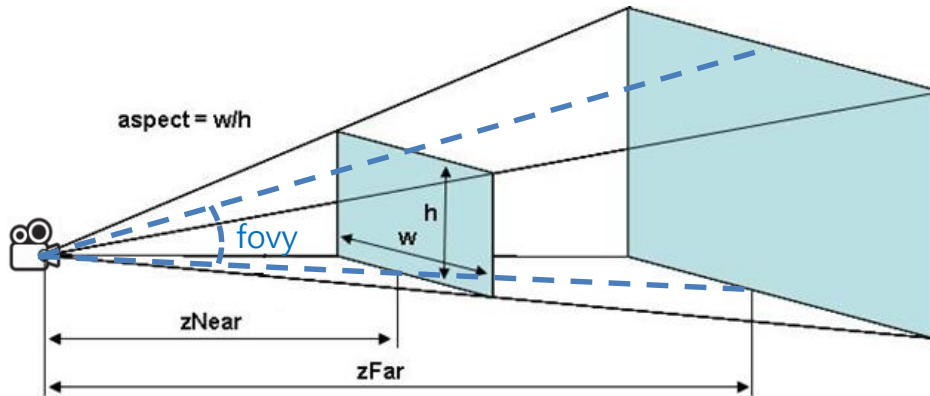
Manipulate the parameters of the `createFrustum(left, right, bottom, top, near, far)` function:

X axis: -5.0 to 5.0		Y axis: -5.0 to 5.0		Z axis: 1.0 to 10.0	
left :	<input type="range"/>	bottom :	<input type="range"/>	near :	<input type="range"/>
-5.0	5.0	-5.0	5.0	0.1	10.0
right:	<input type="range"/>	top:	<input type="range"/>	far:	<input type="range"/>
-5.0	5.0	-5.0	5.0	2.0	20.0
☐ Change canvas size to match aspect ratio.					
Reset Projection					

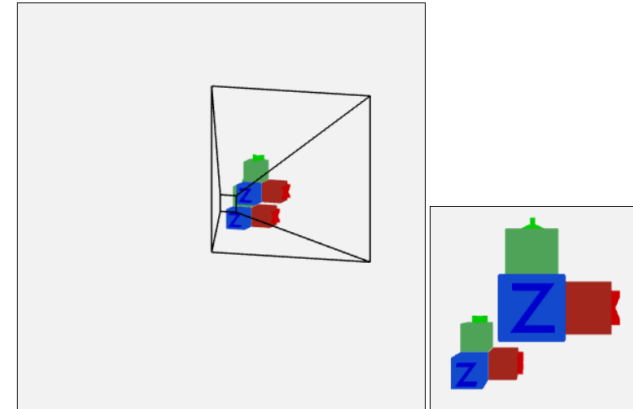
[http://learnwebgl.brown37.net/08_projections/create frustum/create frustum.html](http://learnwebgl.brown37.net/08_projections/create%20frustum/create%20frustum.html)

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

[Demo] Perspective Projection - *perspective*



A perspective projection demo.



Manipulate the parameters of the `createPerspective(fovy, aspect, near, far)` function:

fovy : 5.0 179 Field-of-view (y axis) = 45 degrees
aspect : 0.1 5.0 aspect = 1.00 (width/height)
near : 0.1 10.0 near = 1.0
far : 2.0 20.0 far = 10.0

☐ Change canvas size to match aspect ratio.

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

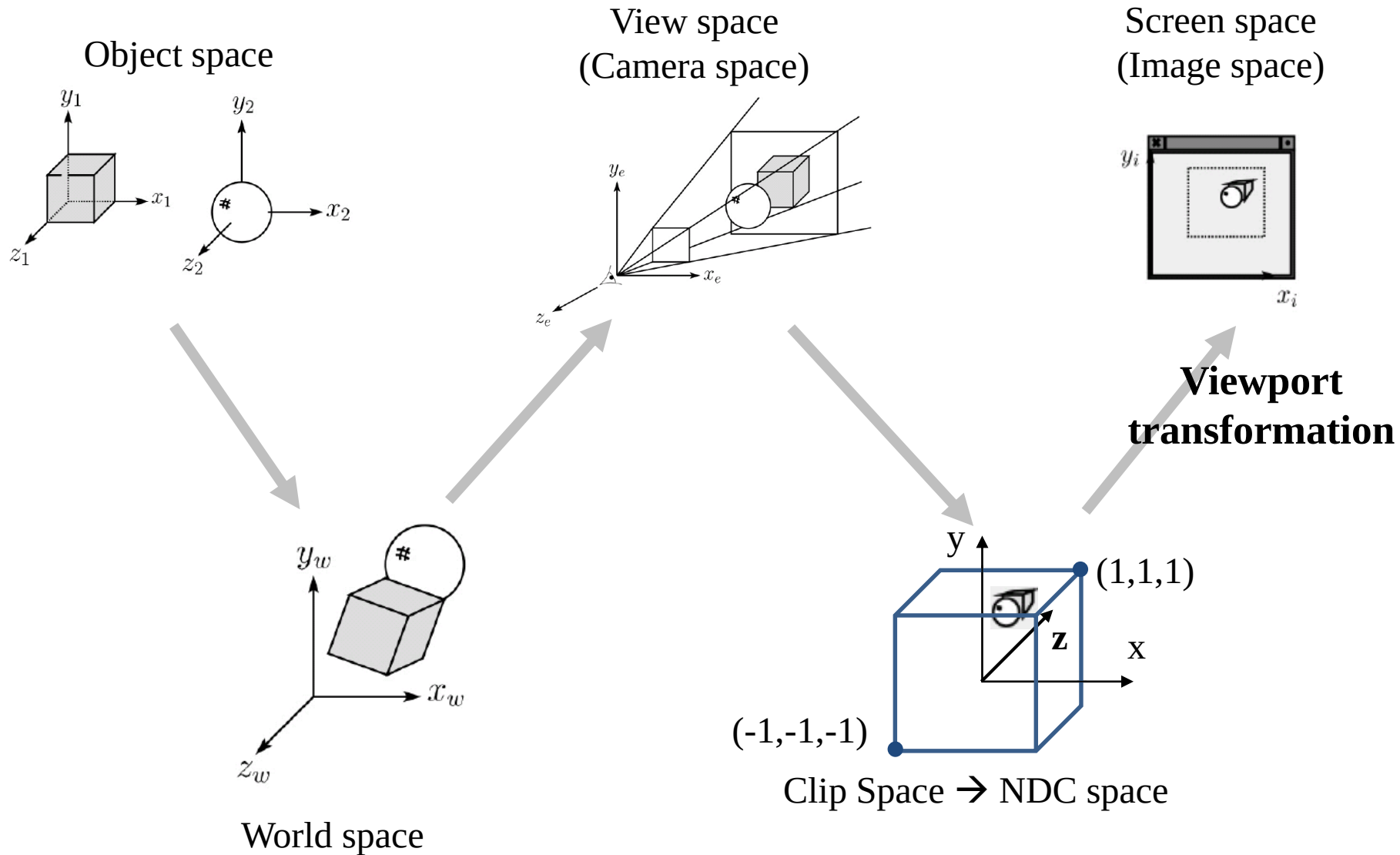
- Observe the view volume (left) and rendered view (right) while dragging *fovy*, *aspect*, *near*, *far* sliders.
- Which one is more convenient, *frustum* or *perspective*?

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. **2024123456: 4.0**
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

Viewport Transformation

Viewport Transformation

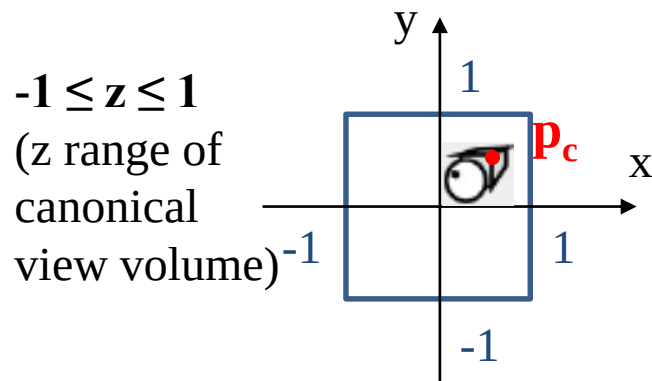


Recall that...

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

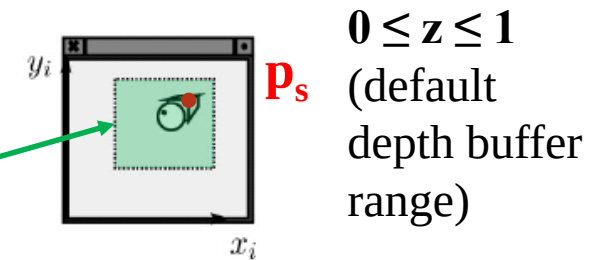
Viewport Transformation

Canonical view volume
(looking down +z direction)



Viewport transformation
: T_{vp}

Screen space
(Image space)

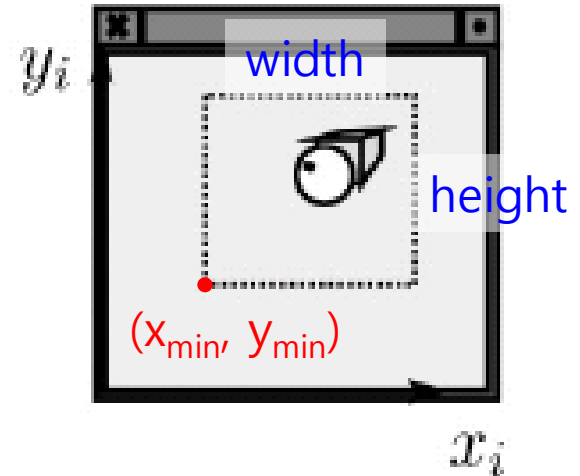


- Viewport: a rectangular viewing region of screen
- So, viewport transformation is also a kind of windowing transformation.

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.