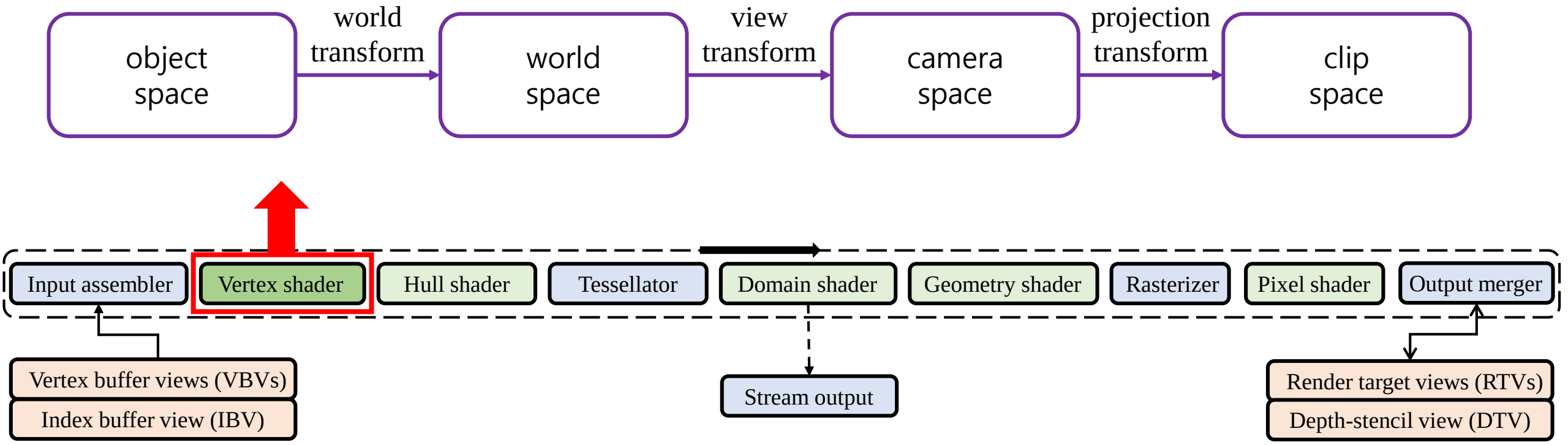
5. Input Assembler & Vertex Processing (2)

Prof. HyeongYeop Kang
siamiz@khu.ac.kr
YouTube: HKang IIIXR LAB
IIIXR LAB

Vertex Shader

The vertex specifications are taken as the input for Vertex Shader in each frame.

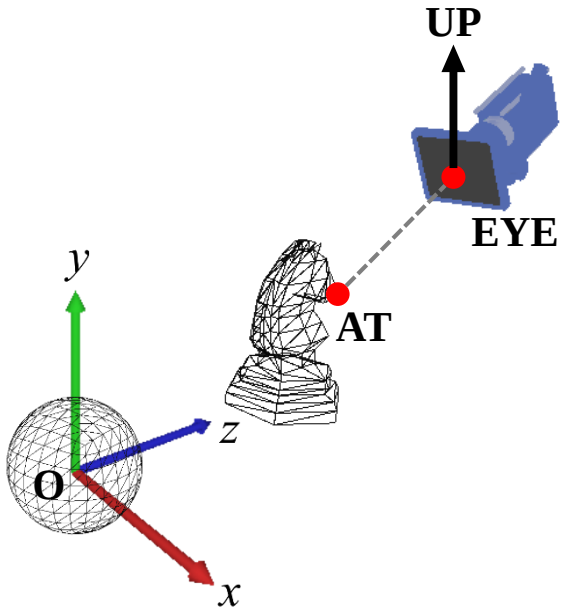
- Vertex shaders are mostly used for computing final positions of input vertices.



View Transform

In the previous lecture, we studied the view matrix.

- M_{view} is applied to all objects in the world space to transform them into the camera space.
- In this lecture, we learn about the projection matrix that transforms objects into the clip space.



$$M_{view} = TR$$

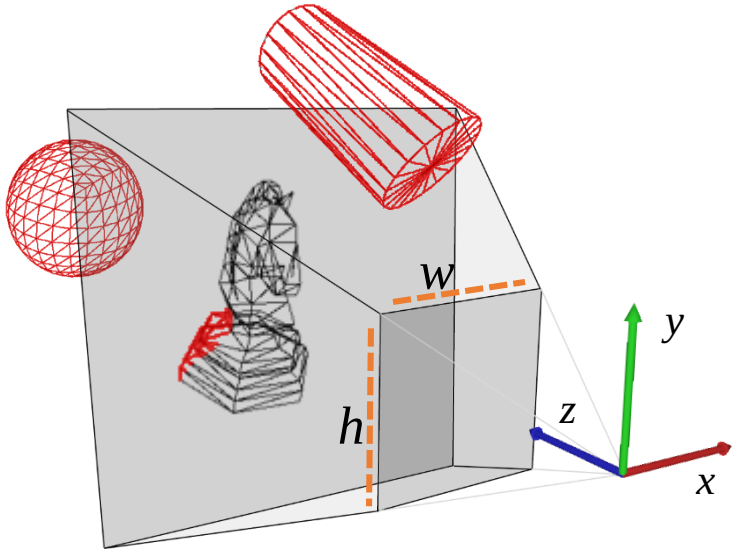
$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\mathbf{EYE}_x & -\mathbf{EYE}_y & -\mathbf{EYE}_z & 1 \end{pmatrix} \begin{pmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

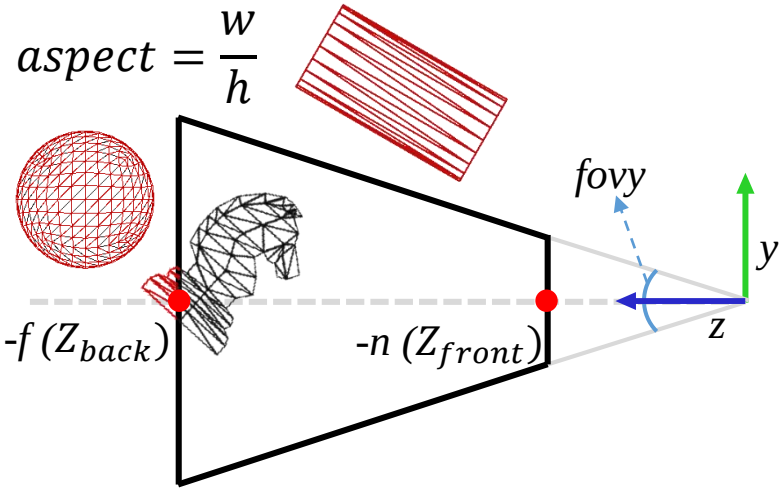
$$= \begin{pmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ -u \cdot \mathbf{EYE} & -v \cdot \mathbf{EYE} & -n \cdot \mathbf{EYE} & 1 \end{pmatrix}$$

View Frustum

View volume

- Camera cannot capture all objects in the scene because its *field of view* is limited.
- The visible region in a scene is called the view frustum (or view volume).





- View frustum parameters, *fovy*, *aspect*, *n*, and *f*, define a truncated pyramid.
- *fovy* specifies the field of view with respect to the y-axis.
- *aspect* denotes the aspect ratio of the view frustum.

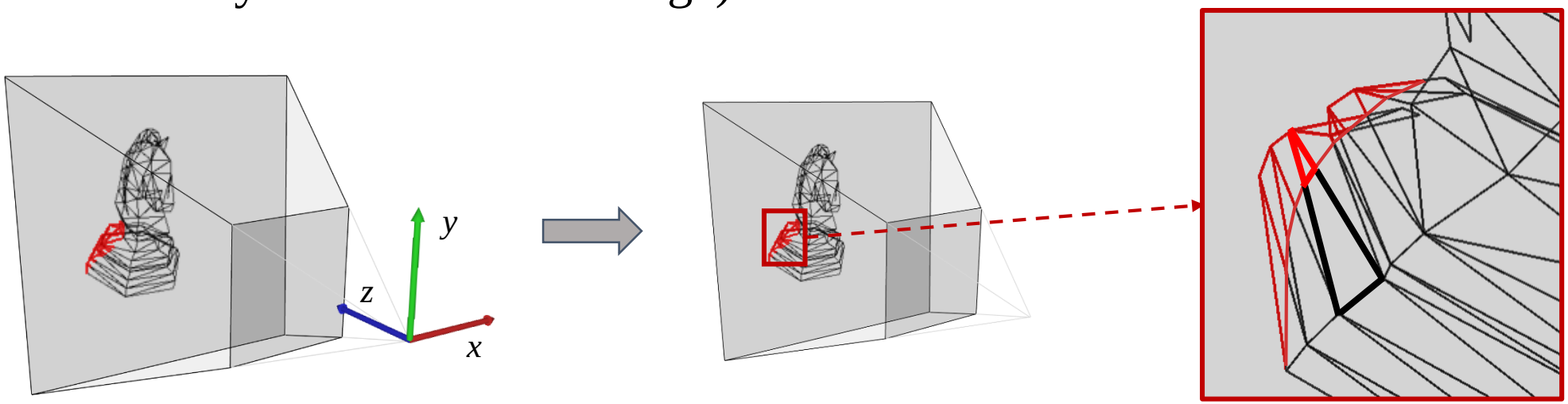
View Frustum

View frustum culling

- The out-of-frustum objects do not contribute to the final image and usually discarded before entering the GPU pipeline.
- This eliminating process saves computational power of GPU.

Clipping

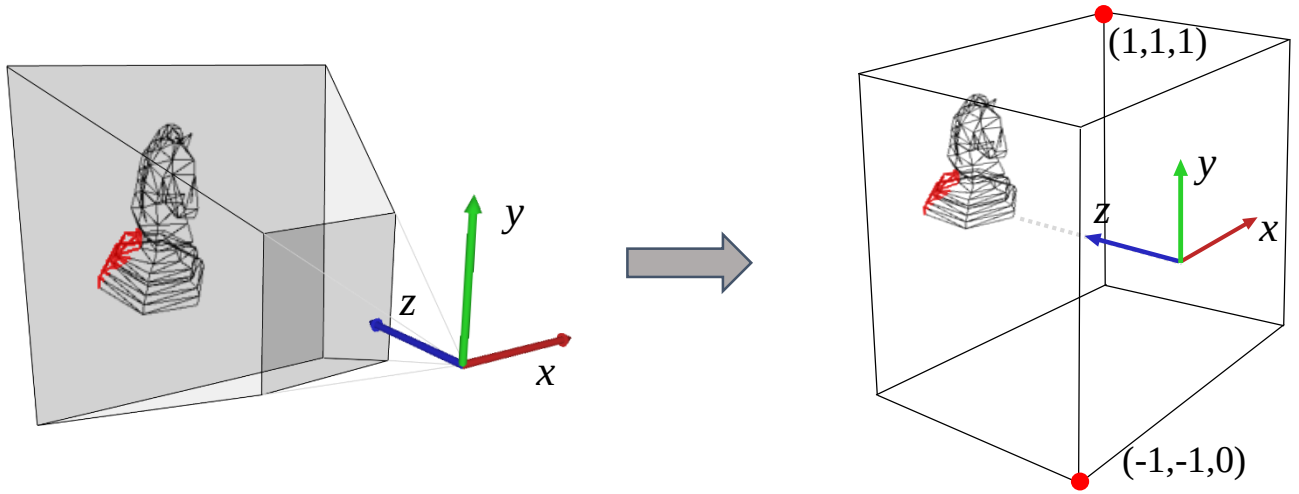
- If a polygon intersects the boundary of the view frustum, it is clipped with respect to the boundary, and only the portion inside the view frustum is processed for display. (This is done by the rasterization stage)

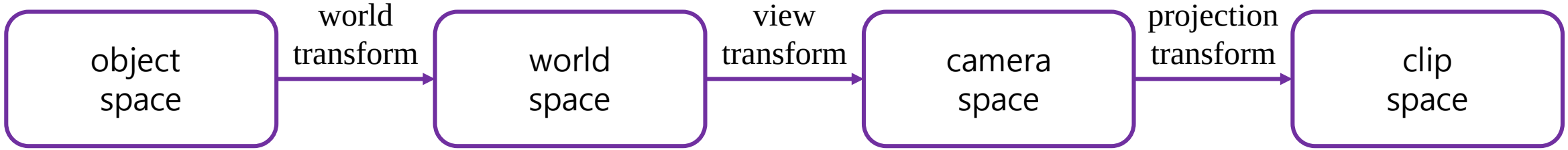


Projection Transform

Projection transform and clip space

- The pyramidal view frustum is not used for clipping.
- Instead, a transform is used that deforms the view frustum into the axis-aligned 2×2×1-sized cube centered at the origin. It is called *projection transform*.
- The camera-space objects are projection-transformed and then clipped against the cube.
- The projection-transformed objects are said to be in the clip space.

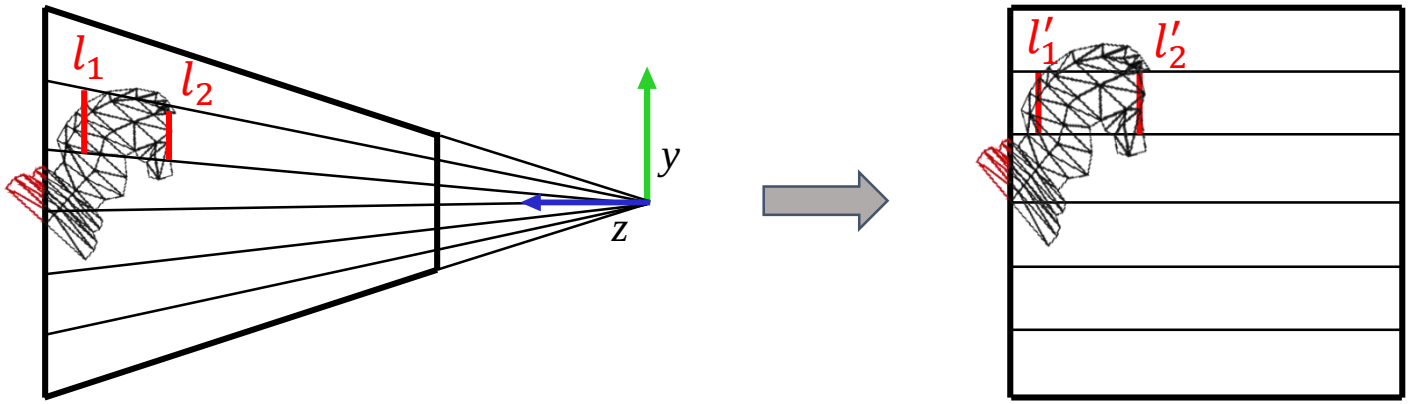




Projection Transform

View frustum to cube

- The view frustum can be taken as a convergent pencil of projection lines. The lines converge on the origin, where the camera (**EYE**) is located.
- All 3D points on a projection line are mapped onto a single 2D point in the projected image. It brings the effect of *perspective projection*, where objects farther away look smaller (See l_1 and l_2).
- The projection transform ensures that the projection lines become parallel to the z-axis. Now viewing is done along the universal projection line. The projection transform brings the effect of perspective projection "within a 3D space."



Projection Transform

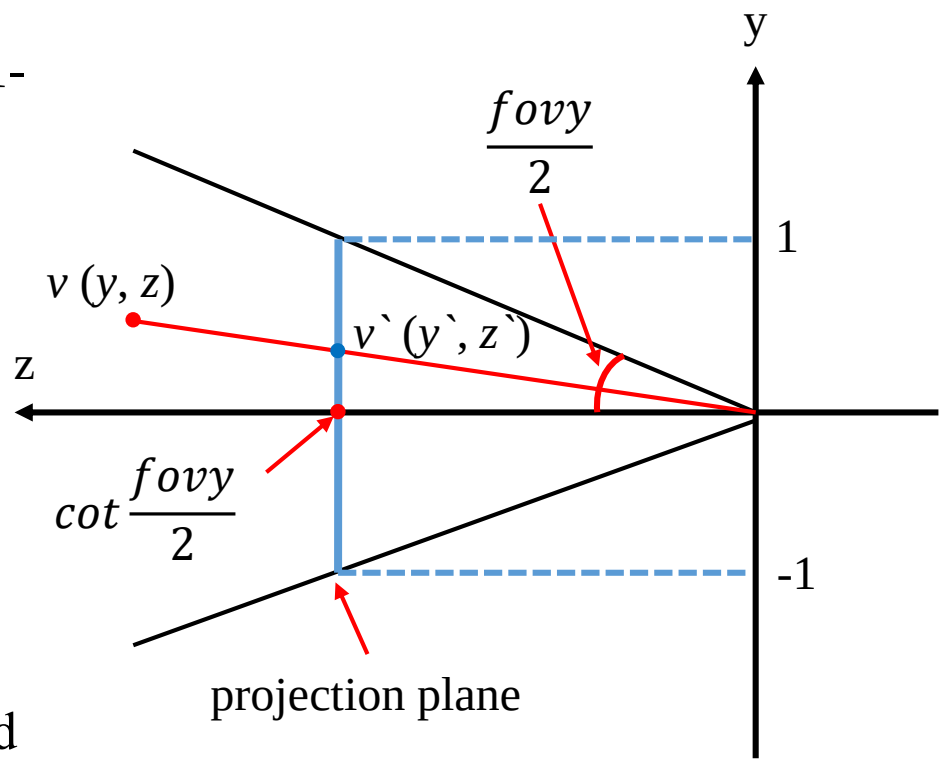
Derivation of projection transform matrix (1)

- The projection matrix converts the view frustum into the 2×2×1-sized cube.
- The coordinates of a point v are represented in 2D: (y, z).
- Suppose a projection plane orthogonal to the z-axis.
- Conceptually, v will be projected to $v`$ on the plane, ($y`$, $z`$).
- If the projection plane is at the z-coordinate of $cot\frac{fovy}{2}$, y` is confined to the range of [-1, 1].
- Now, we can compute y` using the principle of similar triangles:

$$y` = cot\frac{fovy}{2}\frac{y}{z}$$

- If *fovx* were given as the 'horizontal' field of view, we could find

$$x` = cot\frac{fovx}{2}\frac{x}{z}$$



Projection Transform

Derivation of projection transform matrix (2)

- Figure on the right shows that

$$aspect = \frac{W}{H} = \frac{tan\frac{fovx}{2}}{tan\frac{fovy}{2}} = \frac{cot\frac{fovy}{2}}{cot\frac{fovx}{2}}$$

- It is rewritten as follows:

$$cot\frac{fovx}{2} = \frac{cot\frac{fovy}{2}}{aspect}$$

- Therefore, we can define $x`$ in terms of *fovy* and *aspect*:

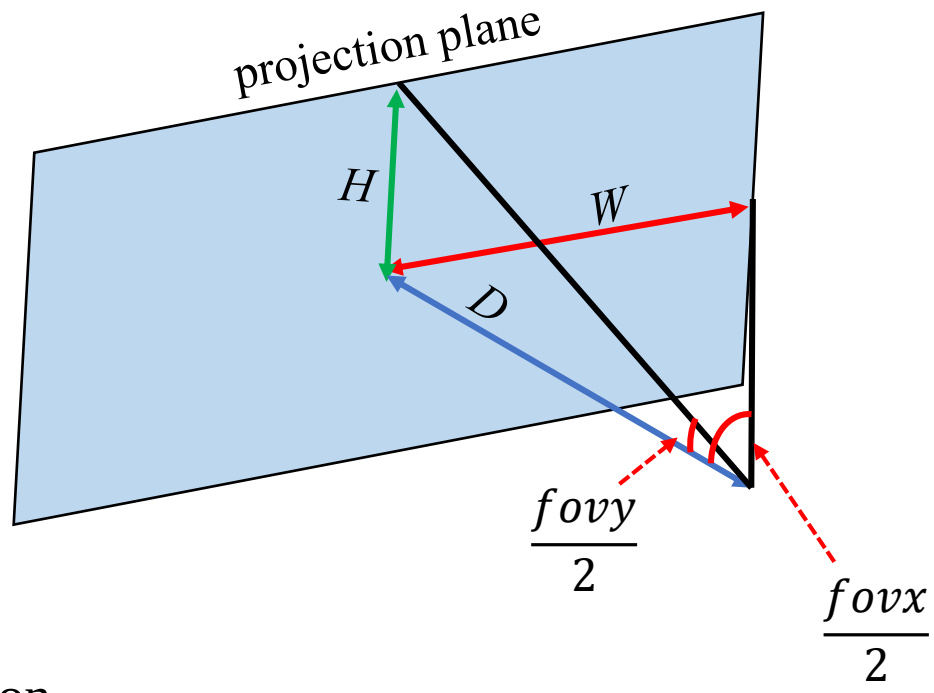
$$x` = cot\frac{fovx}{2}\frac{x}{z} = \frac{cot\frac{fovy}{2}}{aspect}\frac{x}{z}$$

- Let's abbreviate $cot\frac{fovy}{2}$ to *D* and *aspect* to *A*. Then the projection-transformed point v` is represented in homogeneous coordinates:

$$v` = (x`, y`, z`, 1) = (\frac{D}{A}\frac{x}{z}, D\frac{y}{z}, z`, 1)$$

- Subsequently, let's multiply all coordinates by *z*:

$$(\frac{D}{A}\frac{x}{z}, D\frac{y}{z}, z`, 1) = (\frac{D}{A}x, Dy, zz`, z)$$

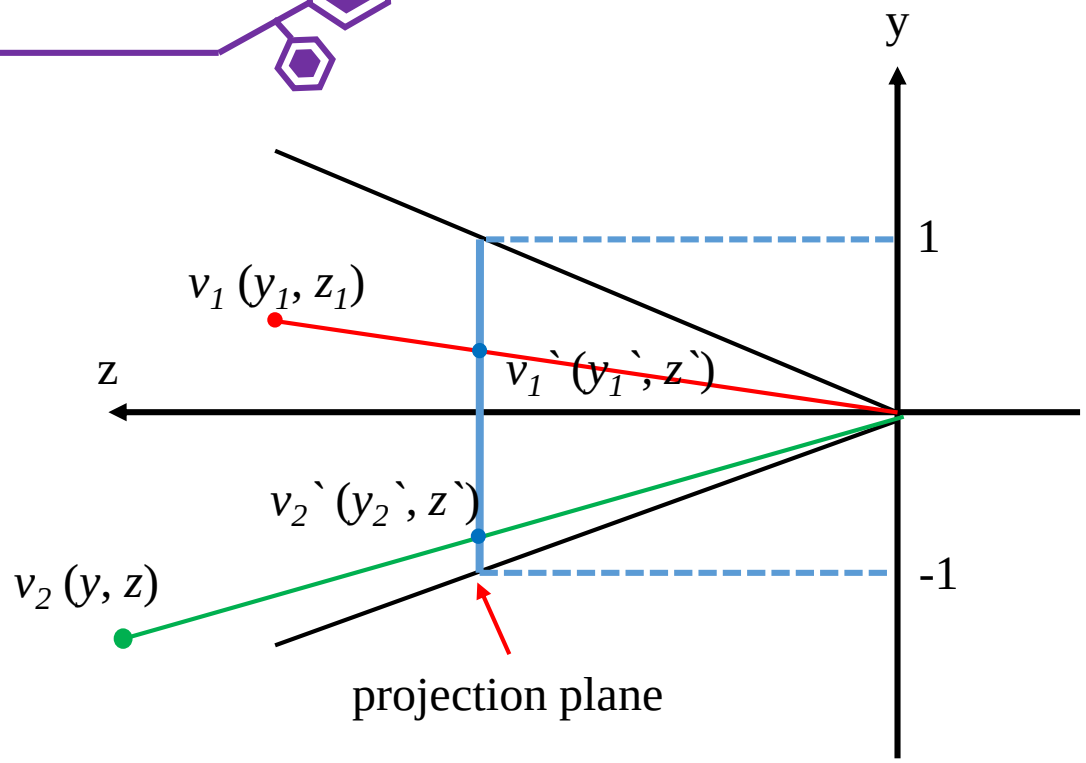


Projection Transform

Derivation of projection transform matrix (3)

- Let's abbreviate zz` to z``.

$$\left(\frac{D}{A}x \quad Dy \quad z'' \quad z\right) = (x \quad y \quad z \quad 1)\begin{pmatrix} \frac{D}{A} & 0 & m_1 & 0 \\ 0 & D & m_2 & 0 \\ 0 & 0 & m_3 & 1 \\ 0 & 0 & m_4 & 0 \end{pmatrix}$$



- Note that *z*' is independent of the *x*- and *y*- coordinates of *v*. Therefore, the third row of the projection matrix is simplified to (0 0 m_3 m_4), and the equation can be written as follows:

$$\left(\frac{D}{A}x \quad Dy \quad z'' \quad z\right) = (x \quad y \quad z \quad 1)\begin{pmatrix} \frac{D}{A} & 0 & 0 & 0 \\ 0 & D & 0 & 0 \\ 0 & 0 & m_3 & 1 \\ 0 & 0 & m_4 & 0 \end{pmatrix}$$

Projection Transform

Derivation of projection transform matrix (3)

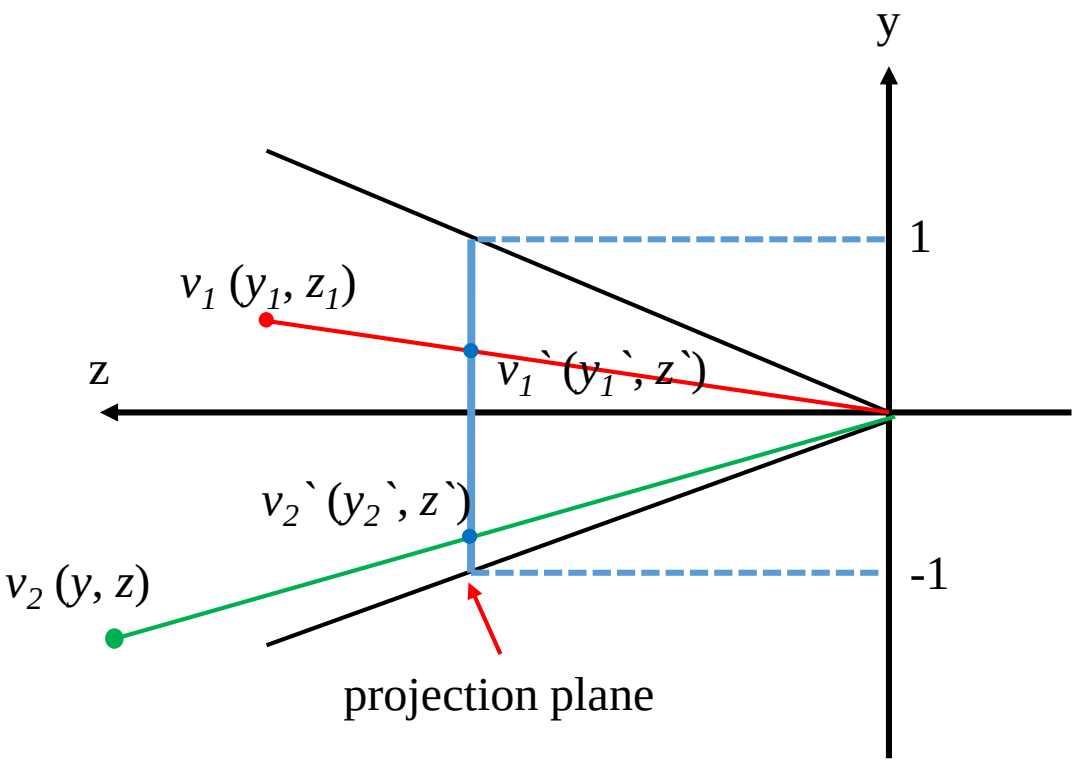
- Then we can obtain:

$$\left(\frac{D}{A}x \quad Dy \quad z'' \quad z\right) = (x \quad y \quad z \quad 1)\begin{pmatrix} \frac{D}{A} & 0 & 0 & 0 \\ 0 & D & 0 & 0 \\ 0 & 0 & m_3 & 1 \\ 0 & 0 & m_4 & 0 \end{pmatrix}$$

$$\left(\frac{D}{A}x \quad Dy \quad m_3 z + m_4 \quad z\right) = \left(\frac{D}{Az}x \quad D\frac{y}{z} \quad m_3 + \frac{m_4}{z} \quad 1\right)$$

- We found that

$$z`` = m_3 + \frac{m_4}{z}$$



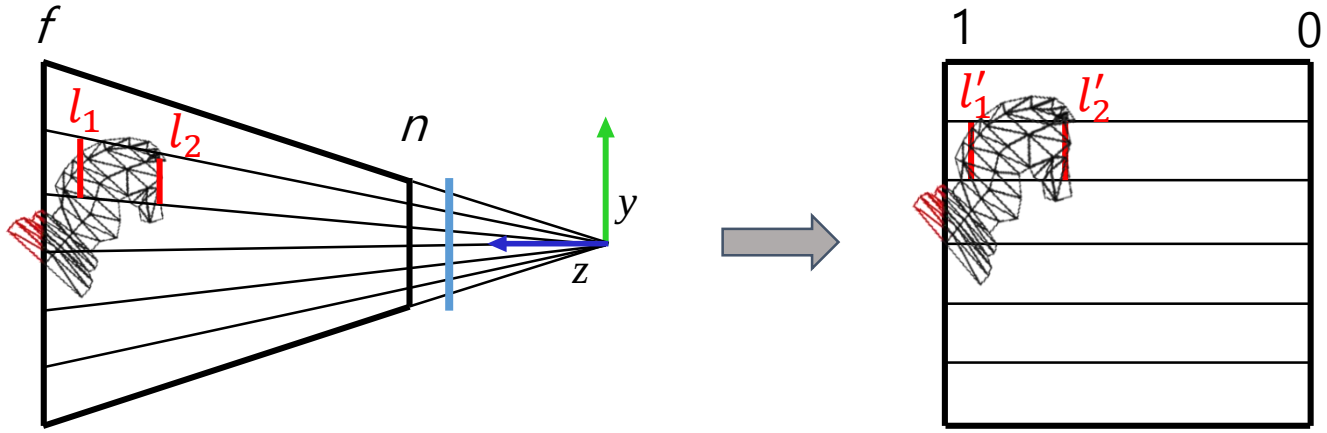
Projection Transform

Derivation of projection transform matrix (4)

- As shown in the figure on the right, the *z*-coordinates *f* and *n* are mapped to 1 and 0, respectively, by the projection transform. Therefore, we obtain the following:
  - $1 = m_3 + \frac{m_4}{f}$
  - $0 = m_3 + \frac{m_4}{n}$

- Solving the equation for m_3 and m_4 gives
  - $m_3 = \frac{f}{f-n}$
  - $m_4 = \frac{-fn}{f-n}$

- Now the third row of the matrix is determined.



$$M_{\text{proj}} = \begin{pmatrix} \frac{\cot\frac{fovy}{2}}{aspect} & 0 & 0 & 0 \\ 0 & \cot\frac{fovy}{2} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 1 \\ 0 & 0 & \frac{-fn}{f-n} & 0 \end{pmatrix}$$

Right-hand System vs. Left-hand System

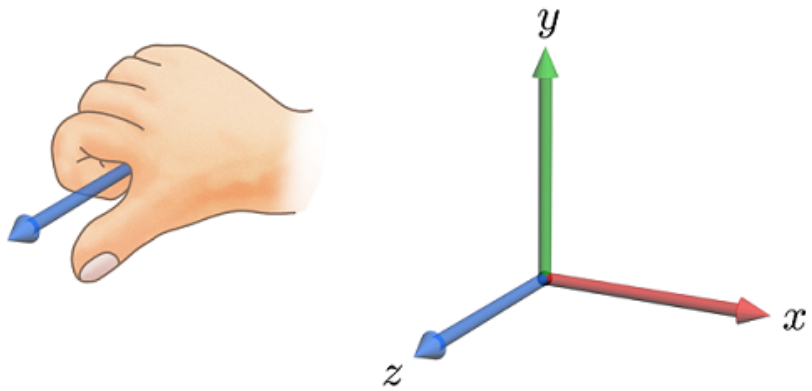
A 3D Cartesian coordinate system is either *right-handed* or *left-handed*.

- In the *right-hand system* (RHS), the thumb of the right hand points toward the positive end of the *z*-axis when the other four fingers curl from the *x*-axis to the *y*-axis.

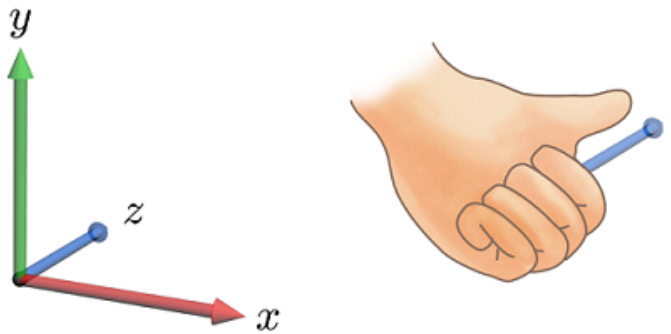
- In the *left-hand system* (LHS), the thumb of the left hand points toward the positive end of the *z*-axis when the other four fingers curl from the *x*-axis to the *y*-axis.

- OpenGL uses the RHS, but Direct3D uses the LHS.

right-hand system (RHS)



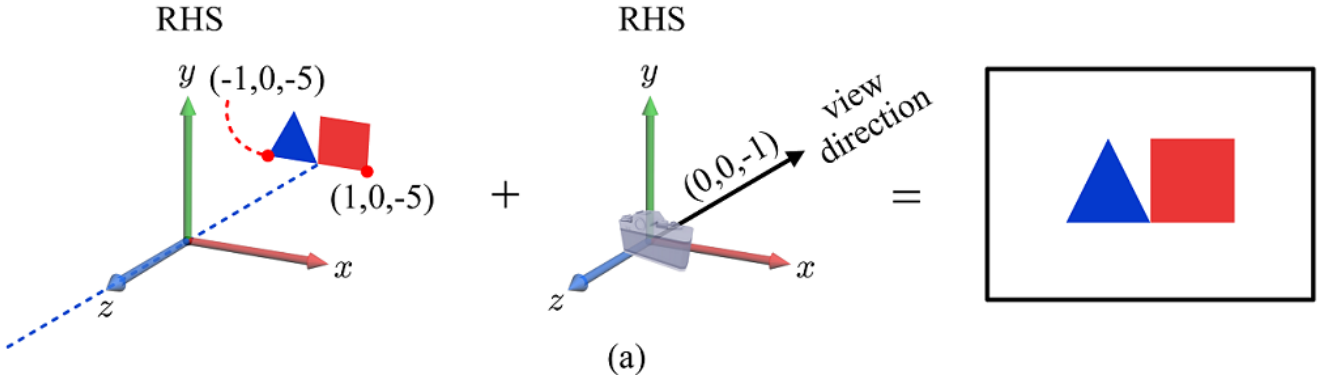
left-hand system (LHS)



Right-hand System vs. Left-hand System

RHS example

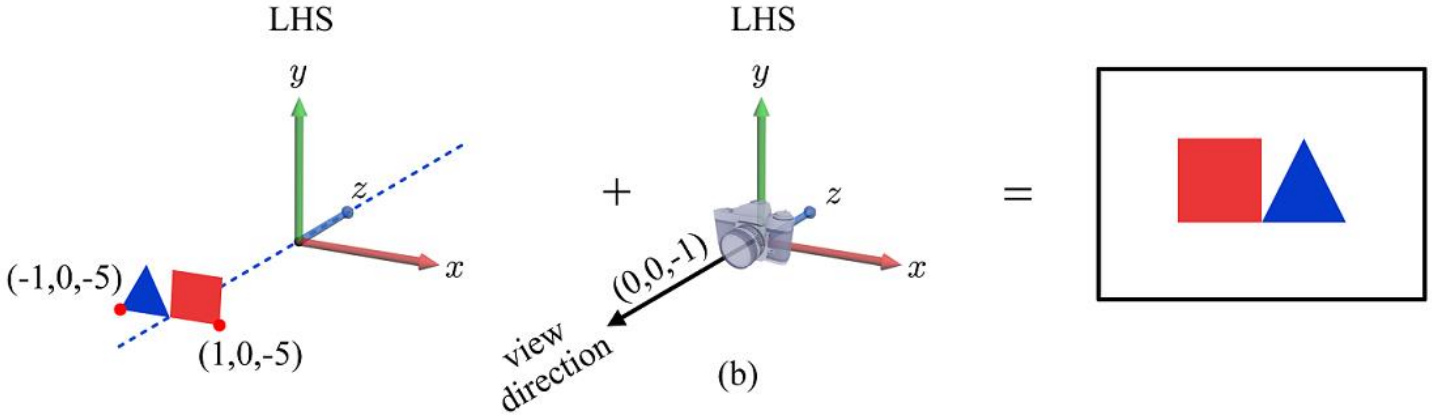
- Assume that the **EYE** = (0, 0, 0) and **AT** is (0, 0, -1)
- Then the camera captures the image.



(a)

LHS example

- Let's port the object and camera parameters into the LHS.
- Observe that the coordinates of the vertices and view direction are not changed, but the image is flipped.

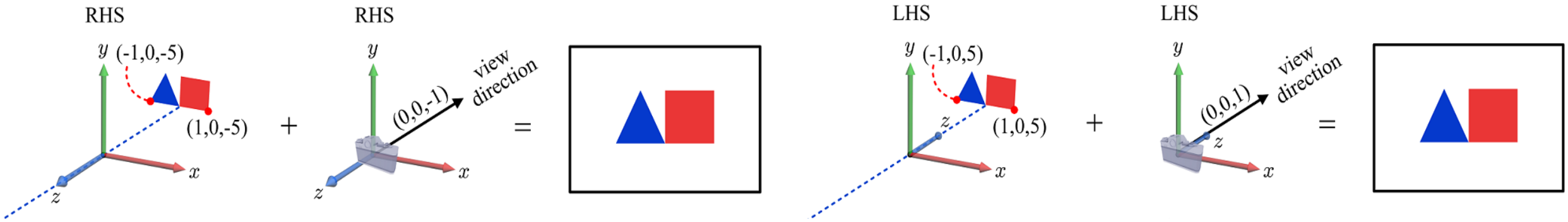


(b)

Right-hand System vs. Left-hand System

RHS ⇔ LHS

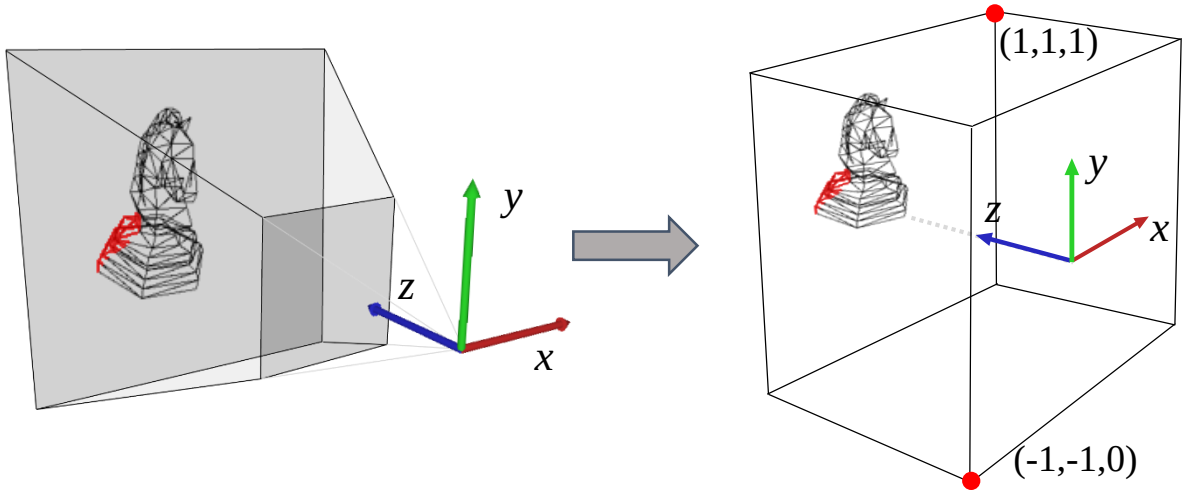
- In order to avoid the reflected image shown in the previous page, the *z*-coordinates of both the objects and view parameters should be negated.
- Note that *z*-negation is conceptually equivalent to the *z*-axis inversion.



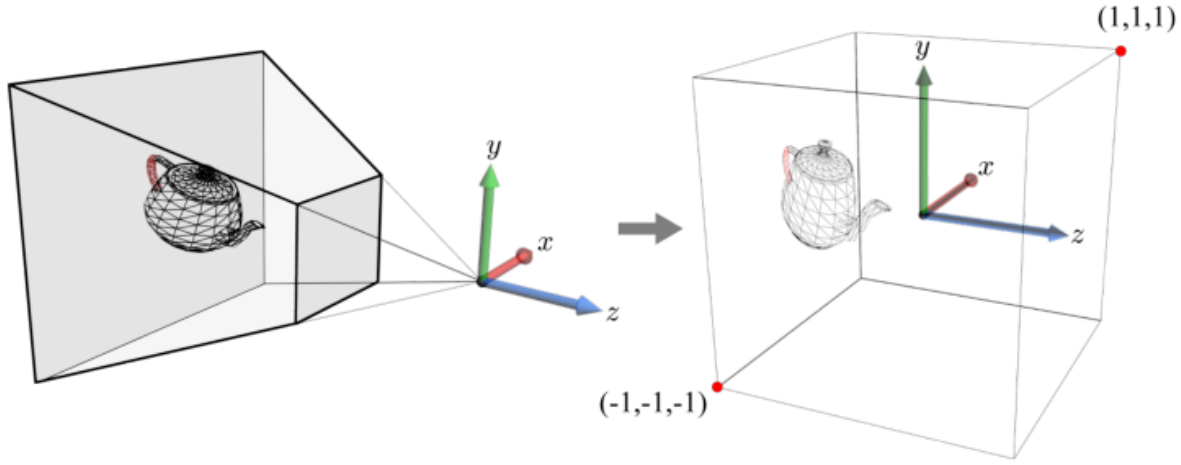
Projection Transform (OpenGL)

Projection transform and clip space

- The *z*-axis is inverted.
- Projection transform deforms the view frustum into the axis-aligned 2×2×2-sized cube instead of 2×2×1-sized cube.



DirectX



OpenGL

Projection Transform (OpenGL)

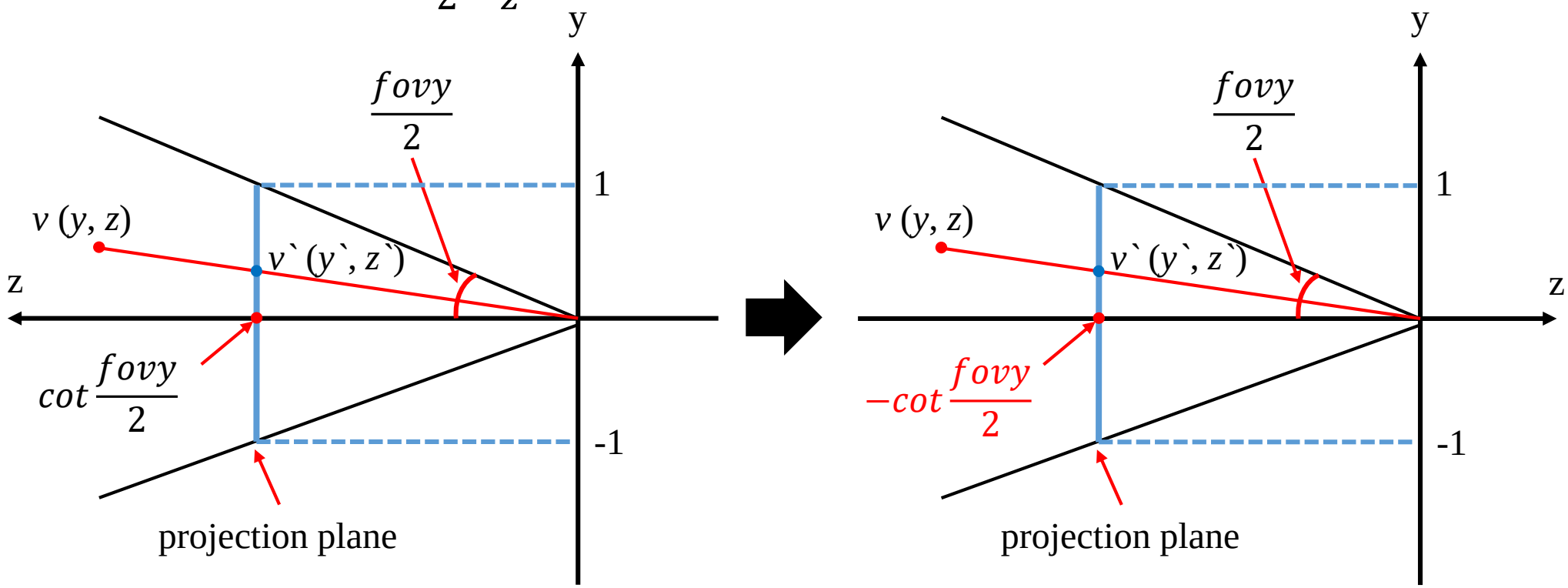
Derivation of projection transform matrix (1)

- We can compute y` using the principle of similar triangles:

$$y` = -cot\frac{fovy}{2}\frac{y}{z}$$

- If *fovx* were given as the 'horizontal' field of view, we could find

$$x` = -cot\frac{fovx}{2}\frac{x}{z}$$



Projection Transform (OpenGL)

Derivation of projection transform matrix (2)

- Let's abbreviate $cot\frac{fovy}{2}$ to *D* and *aspect* to *A*. Then the projection-transformed point v` is represented in homogeneous coordinates:

$$v` = (x`, y`, z`, 1) = (-\frac{D}{A}\frac{x}{z}, -D\frac{y}{z}, z`, 1)$$

- Subsequently, let's multiply all coordinates by -*z*:

$$(-\frac{D}{A}\frac{x}{z}, -D\frac{y}{z}, z`, 1) = (\frac{D}{A}x, Dy, -zz`, -z)$$

- Let's abbreviate –*zz`* to *z``*.

$$\begin{pmatrix} \frac{D}{A}x \\ Dy \\ z`` \\ -z \end{pmatrix} = \begin{pmatrix} \frac{D}{A} & 0 & 0 & 0 \\ 0 & D & 0 & 0 \\ m_1 & m_2 & m_3 & m_4 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- The third row of the projection matrix is simplified to (0 0 m_3 m_4):

$$\begin{pmatrix} \frac{D}{A}x \\ Dy \\ z`` \\ -z \end{pmatrix} = \begin{pmatrix} \frac{D}{A} & 0 & 0 & 0 \\ 0 & D & 0 & 0 \\ 0 & 0 & m_3 & m_4 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} \frac{D}{A}x \\ Dy \\ m_3z+m_4 \\ -z \end{pmatrix} = \begin{pmatrix} -\frac{Dx}{Az} \\ -D\frac{y}{z} \\ -m_3-\frac{m_4}{z} \\ 1 \end{pmatrix}$$

- We found that $z` = -m_3 - \frac{m_4}{z}$

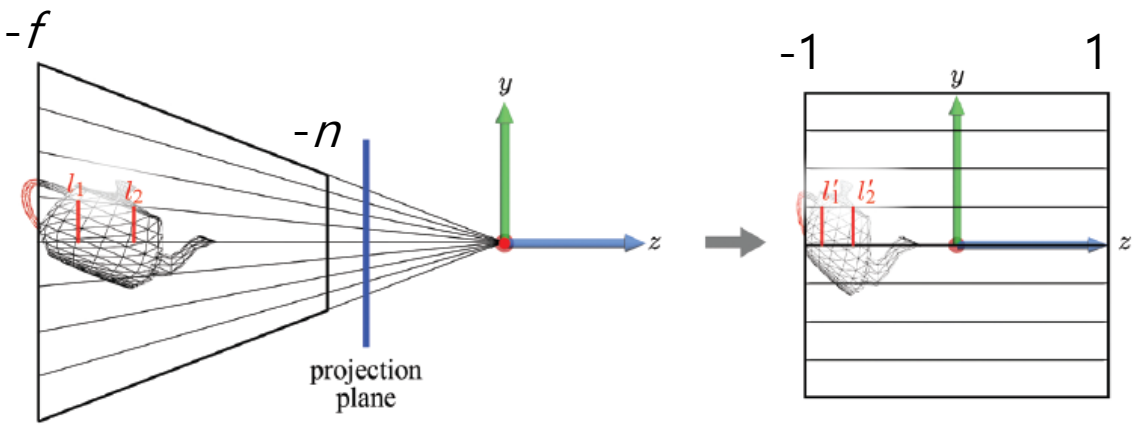
Projection Transform (OpenGL)

Derivation of projection transform matrix (3)

- As shown in the figure on the right, the *z*-coordinates $-f$ and $-n$ are mapped to -1 and 1, respectively, by the projection transform. Therefore, we obtain the following:

$$-1 = -m_3 + \frac{m_4}{f}$$

$$1 = -m_3 + \frac{m_4}{n}$$



- Solving the equation for m_3 and m_4 gives

$$m_3 = \frac{f+n}{f-n}$$

$$m_4 = \frac{2nf}{f-n}$$

- Now the third row of the matrix is determined.

$$M_{\text{proj}} = \begin{pmatrix} \frac{\cot\frac{fovy}{2}}{aspect} & 0 & 0 & 0 \\ 0 & \cot\frac{fovy}{2} & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$