컴퓨터그래픽스

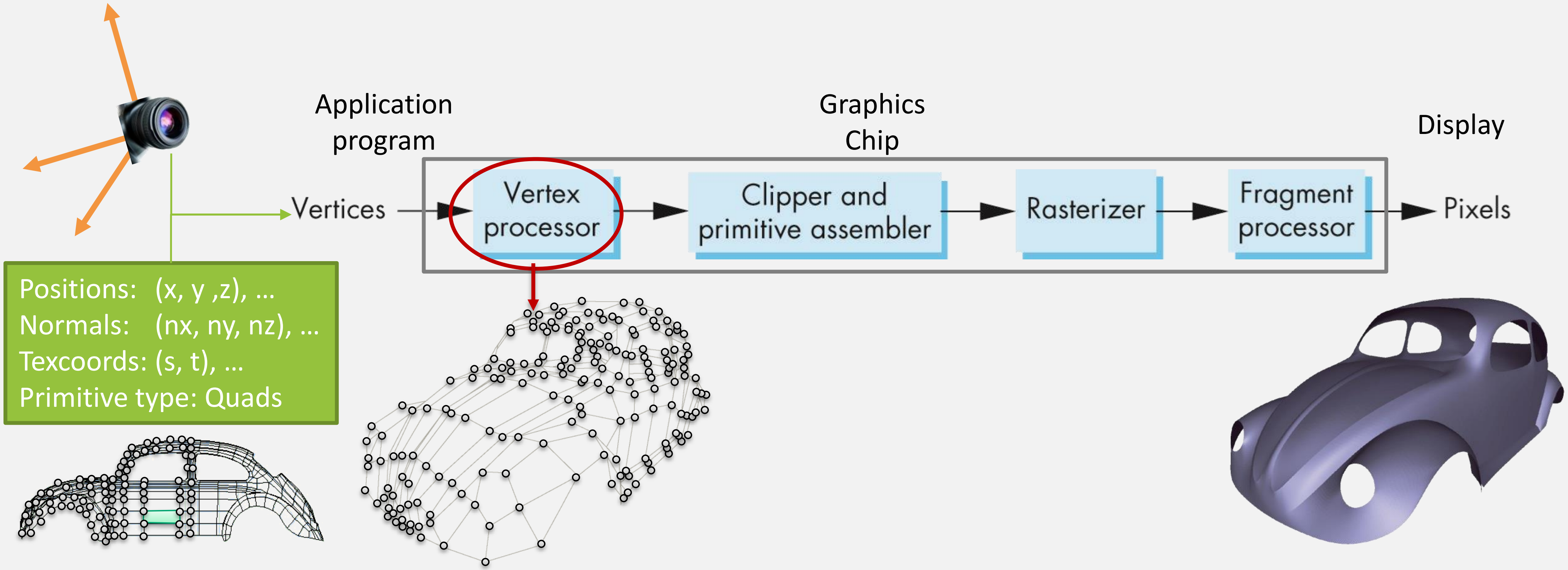
김준호
Visual Computing Lab.
국민대학교 소프트웨어학부

- Overview of Vertex Processor
- Coordinate System & Coordiante Values
- ModelView matrix
- Projection matrix
- Viewport

Vertex Processor

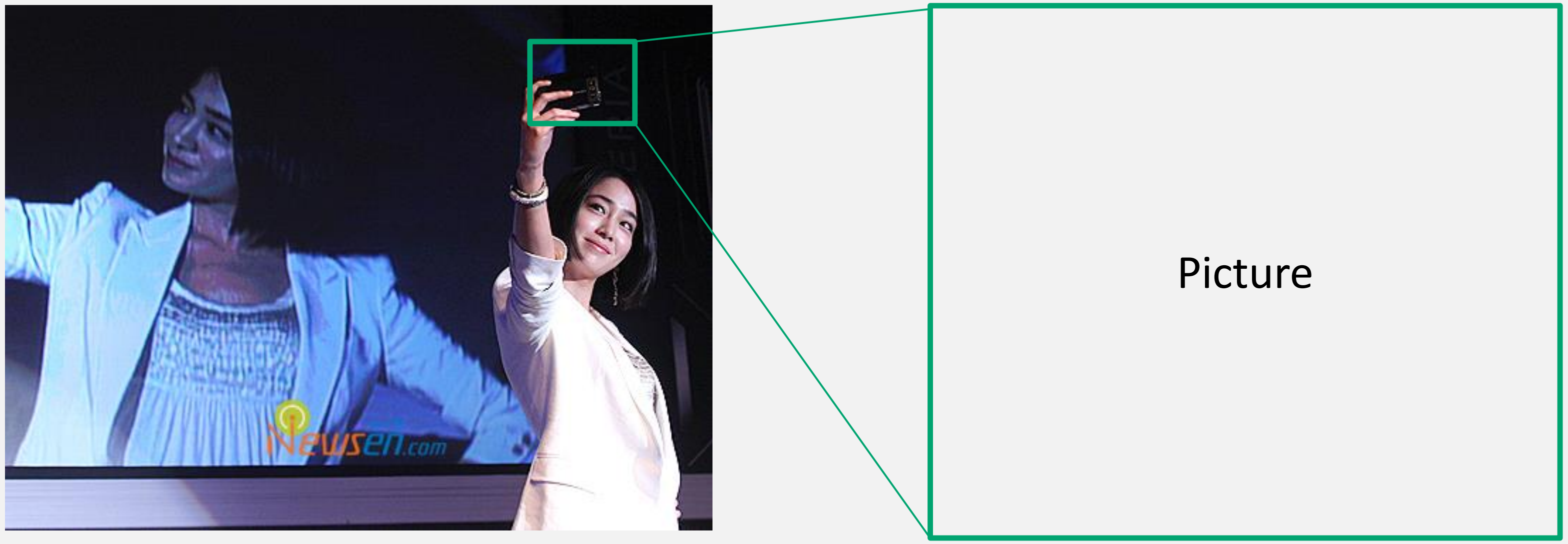
Overview of Vertex Processor

- Vertex processor
  - Converting object representations from one coordinate system to another
    - Object coordinates → Camera coordinates → Screen coordinates



Objectives

- We are interested in an image captured from the camera
  - First of all, we should know the coordinate of a 3D point, from camera's viewpoint
  - It means, we have to understand the change of coordinates
    - Coordinate values in object space → Coordinate values in camera space

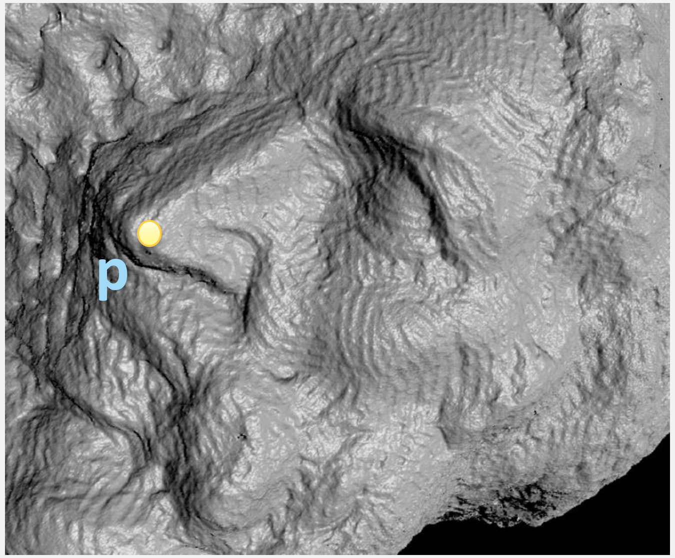


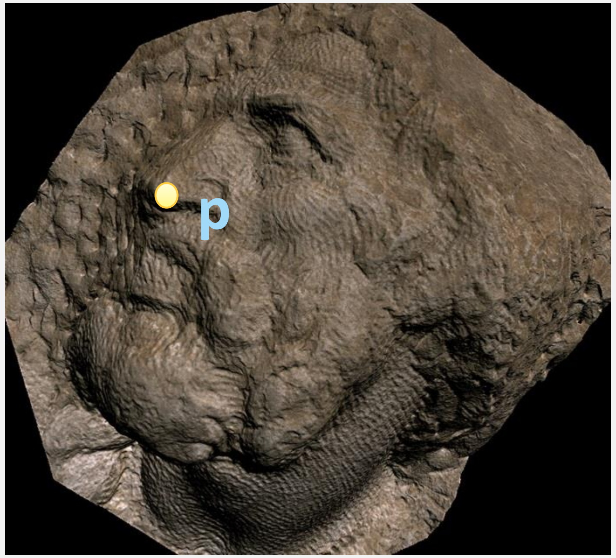
- Coordinate system & Coordinate Values
- MovelView matrix

Coordinate System and Coordinate Values

Coordinate Value – Representation of a Point

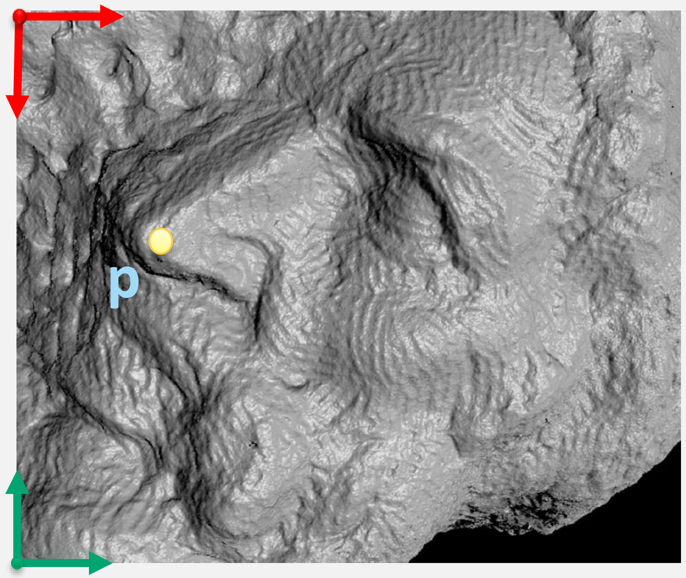
- Where is a point **p**?
  - For the same point, we can represent it with different coordinates

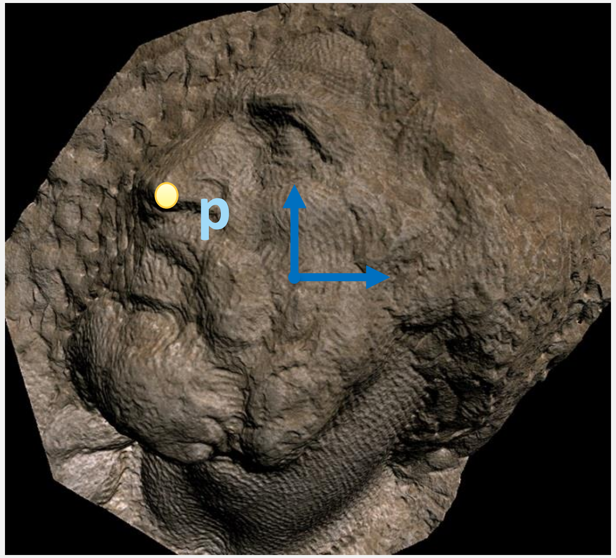




Coordinate Value – Representation of a Point

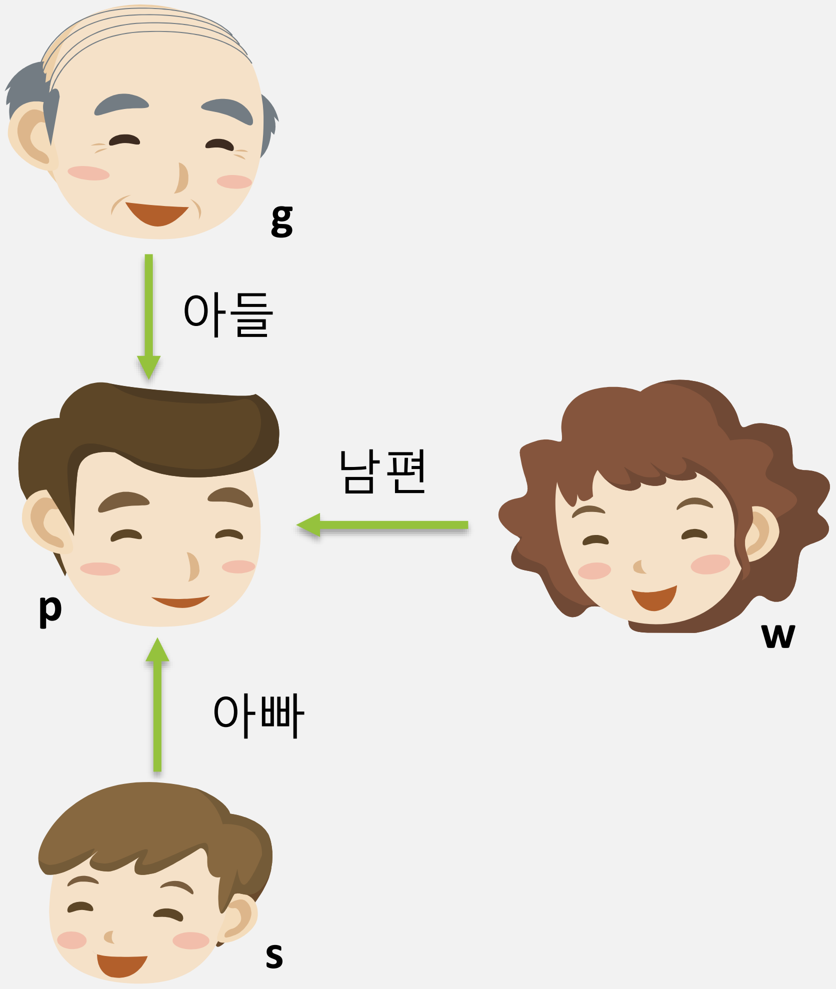
- The coordinate value of a point is meaningful, only when we specify a coordinate system
  - The same point can be reprsented with different coordinate values!
    - **p** = (1.5, 3) = (1.5, 2.5) = (-1.2, 1)





Coordinate Value – Representation of a Point

- Analogy in real-world
  - A point **p**
    - 존재
  - Coordinate system (or Frame)
    - 관점
  - Coordinate value of **p**
    - 특정 관점에서 해당 존재를 부르는 호칭 (representation)
    - 동일한 존재는 여러가지 호칭으로 불릴 수 있음
    - $\mathbf{p}_{[g]}$ = 아들
    - $\mathbf{p}_{[w]}$ = 남편
    - $\mathbf{p}_{[s]}$ = 아빠



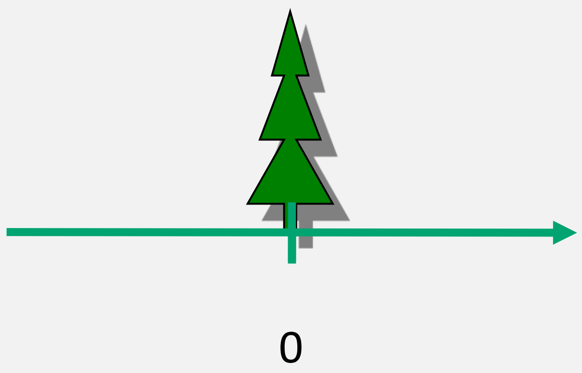
ModelView matrix

What is ModelView Matrix?

- The composition of a model matrix and a view matrix
  - OpenGL manages the model matrix and the view matrix together
    - c.f.) Direct3D seperates the model matrix and the view matrix
  - Model matrix
    - 3D transformation of an object (or model) in the world coordinate system
  - View matrix
    - 3D transformation of a camera in the world coordinate system
    - This is the extrinsic parameters of the camera!
- We can obtain the camera coordinates by multiplying the ModelView matrix to the object coordinates
  - $\mathbf{x}_{view} = \mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj}$
  - $\mathbf{V}^{-1}\mathbf{M}$ is called the modelview matrix

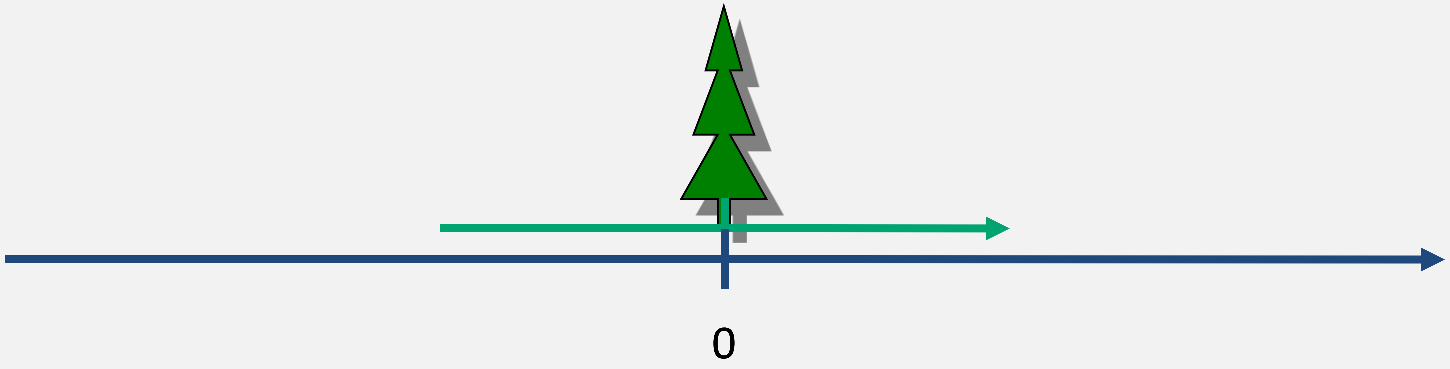
1D Case

- You model an object in the object-space coordinate system
  - Every point is represented with object-space coordinates $\mathbf{x}_{obj}$
  - 3D positions specified in glVertexAttribPointer() are in the objects-space coordinate system



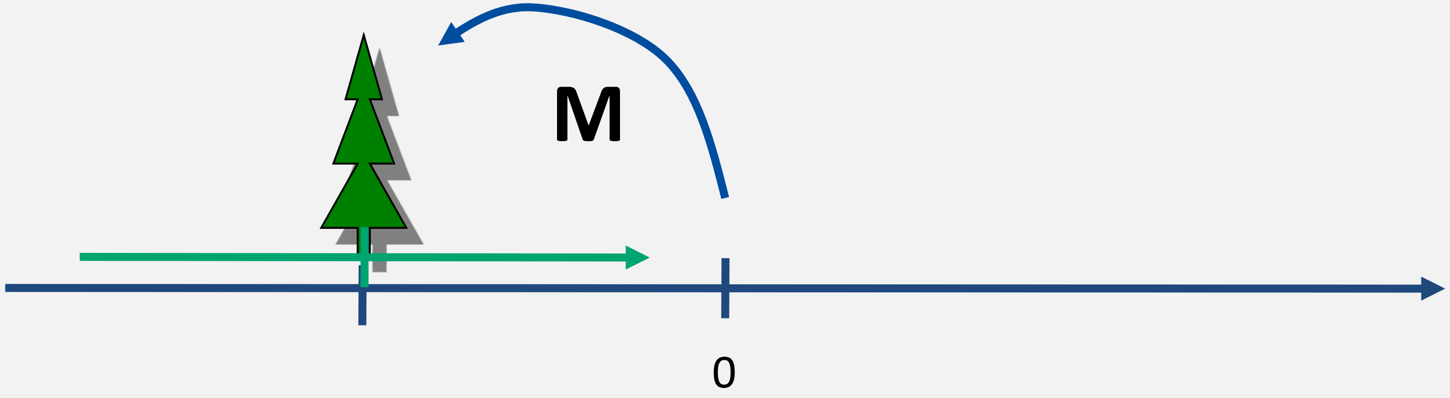
1D Case

- You may place the object in the origin of the world
  - $\mathbf{x}_{world} = \mathbf{x}_{obj}$



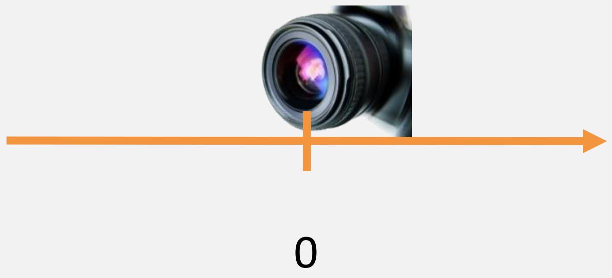
1D Case

- You move the object to somewhere in the world
  - **M**: model matrix (or world-transform matrix)
    - You set **M** by using the composition of ~~glTranslate~~(), ~~glRotate~~(), ~~glScale~~()
  - $\mathbf{x}_{world} = \mathbf{M}\mathbf{x}_{obj}$



1D Case

- Now, let's consider a camera

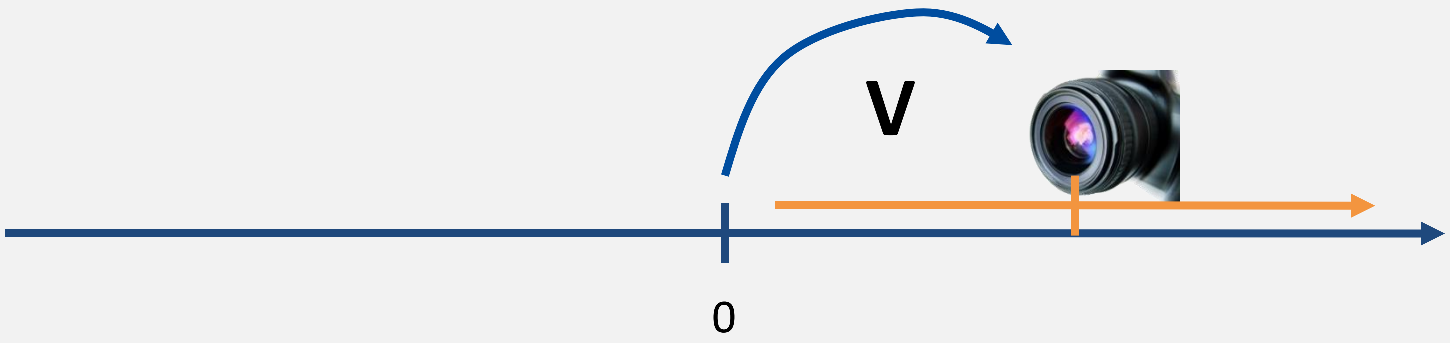


1D Case

- You may place the camera in the origin of the world
  - $\mathbf{x}_{world} = \mathbf{x}_{view}$

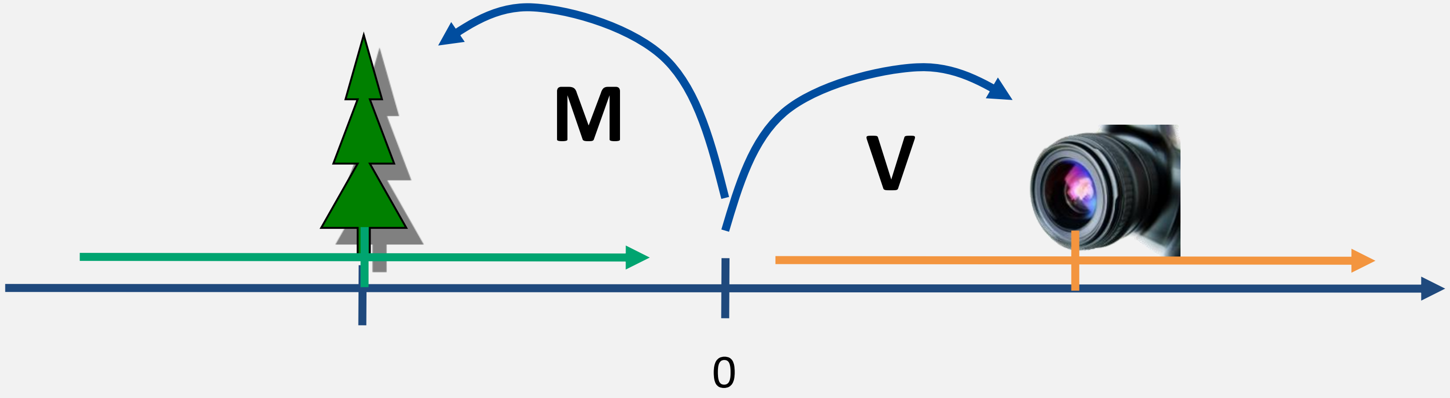
1D Case

- You move the camera to somewhere in the world
  - **V**: view-transform matrix
    - You set $\mathbf{V}^{-1}$ by using ~~gluLookAt~~()
  - $\mathbf{x}_{world} = \mathbf{V}\mathbf{x}_{view}$



1D Case

- Let's consider both of camera & object
  - $\mathbf{x}_{world} = \mathbf{M}\mathbf{x}_{obj}$
  - $\mathbf{x}_{world} = \mathbf{V}\mathbf{x}_{view}$

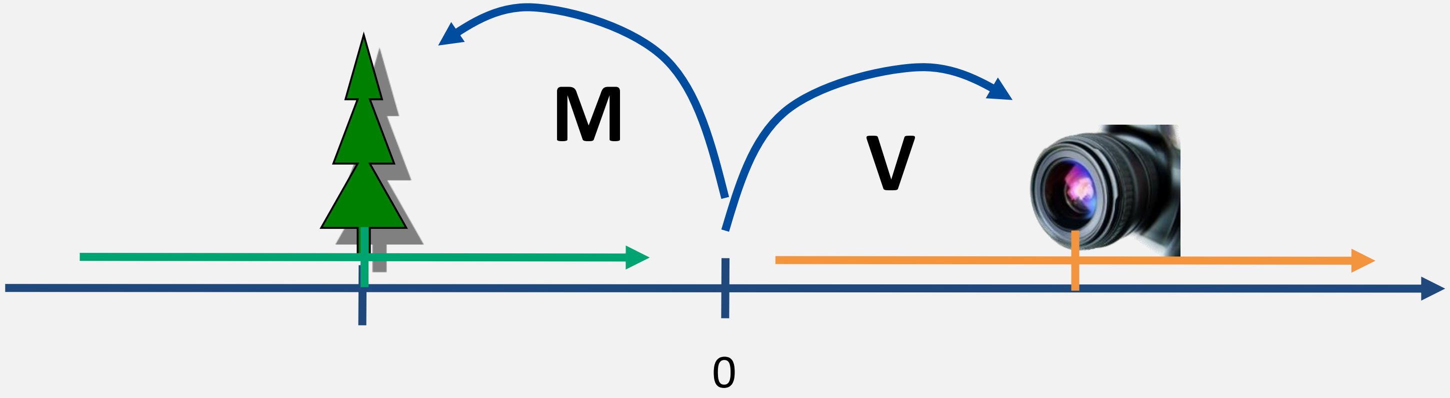


1D Case

- How can we obtain the camera-space coordinates of the object?
  - $\mathbf{x}_{world} = \mathbf{M}\mathbf{x}_{obj}$
  - $\mathbf{x}_{world} = \mathbf{V}\mathbf{x}_{view}$

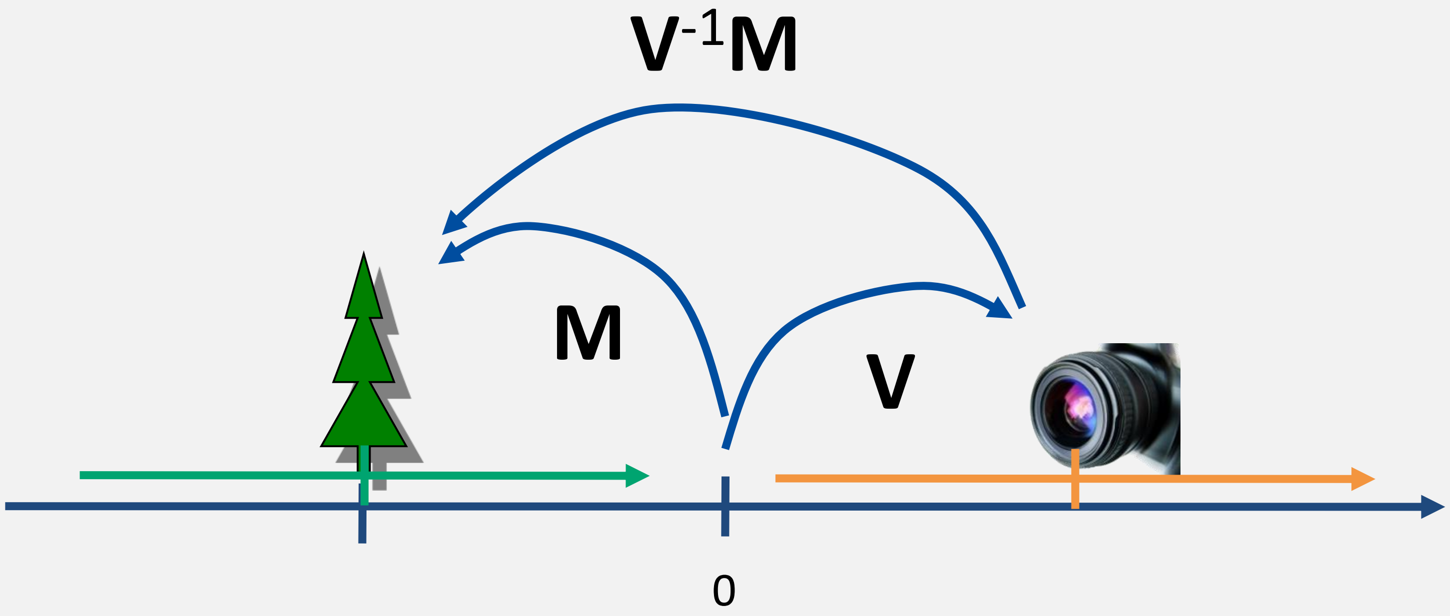
$$\mathbf{M}\mathbf{x}_{obj} = \mathbf{V}\mathbf{x}_{view}$$

$$\boxed{\mathbf{x}_{view} = \mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj}}$$



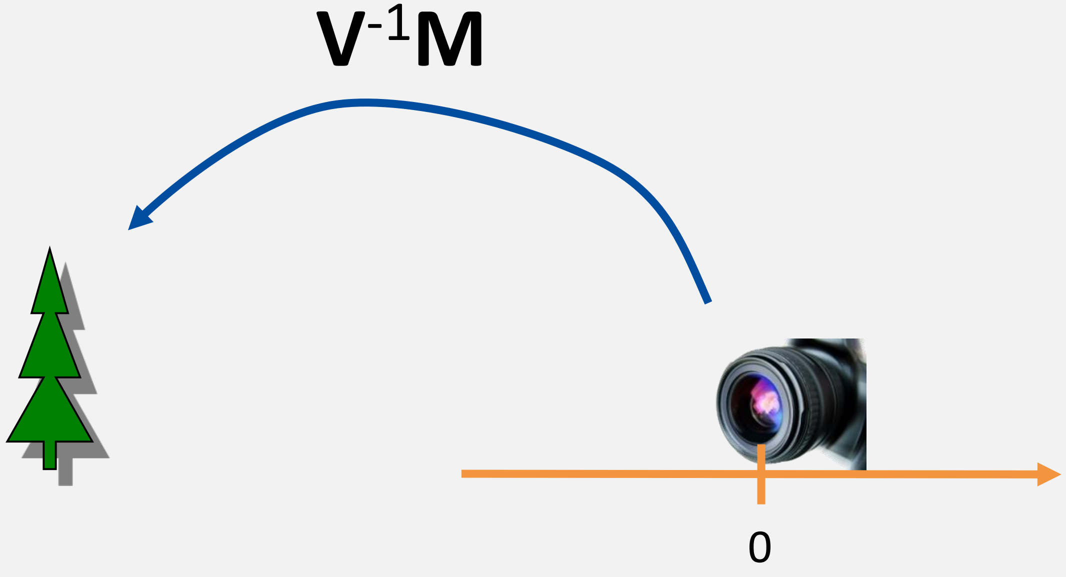
1D Case

- What does "$\mathbf{x}_{view} = \mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj}$" mean?



1D Case

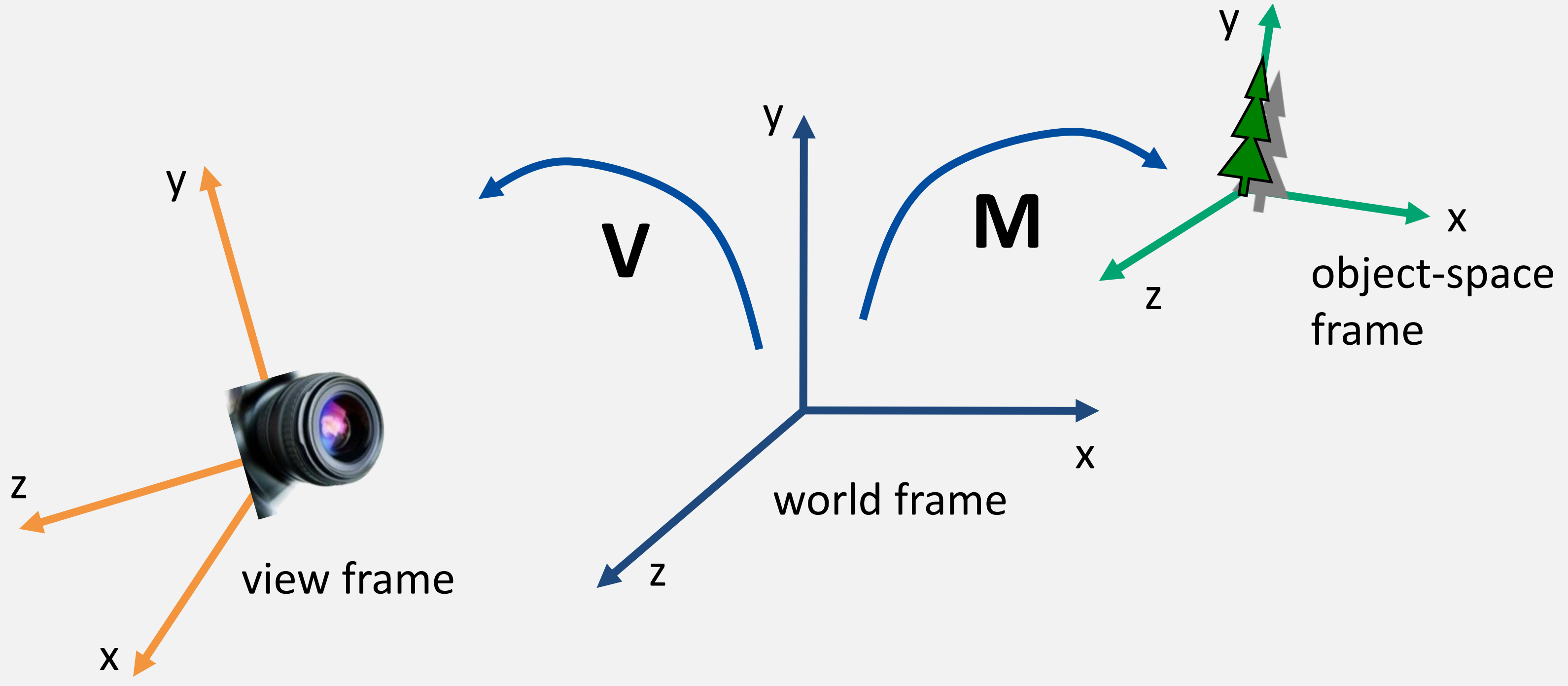
- What does "$\mathbf{x}_{view} = \mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj}$" mean?
  - 3D Position of **x**, measured from the coordinate system of the camera
  - World-frame-independent representation
  - Now, you may think the world frame as an illusion.



3D Case

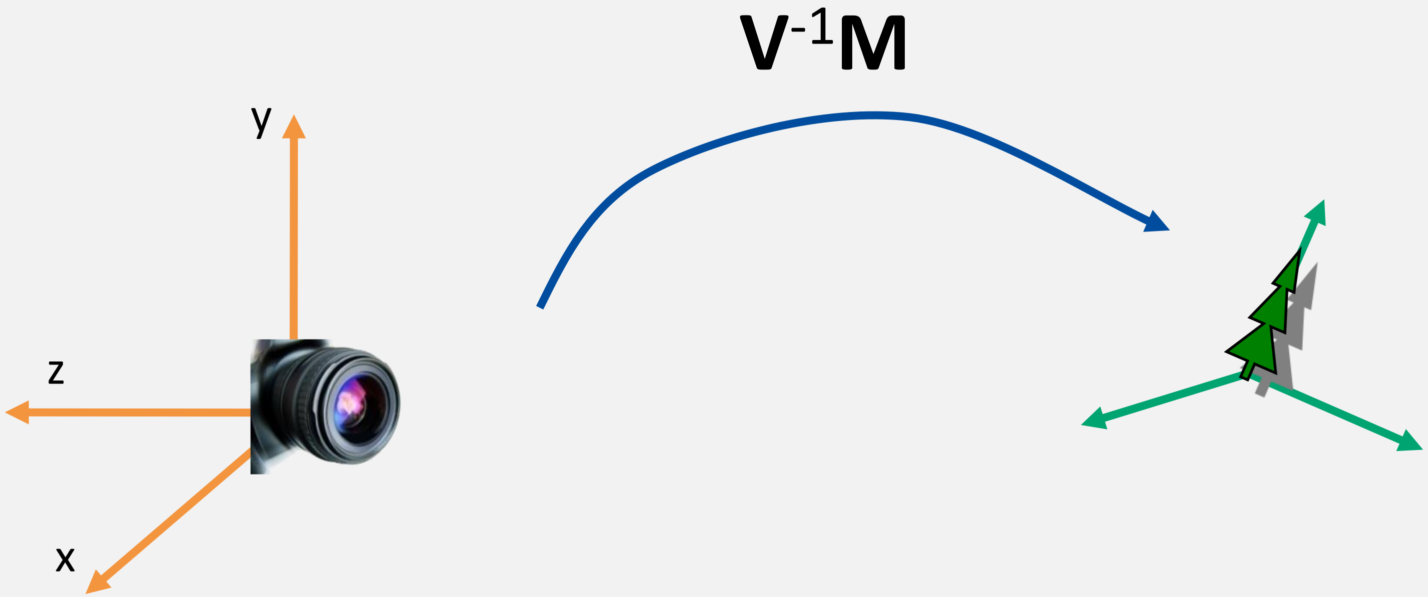
- Exactly same!
  - $\mathbf{x}_{world} = \mathbf{M}\mathbf{x}_{obj}$
  - $\mathbf{x}_{world} = \mathbf{V}\mathbf{x}_{view}$

$$\boxed{\mathbf{x}_{view} = \mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj}}$$



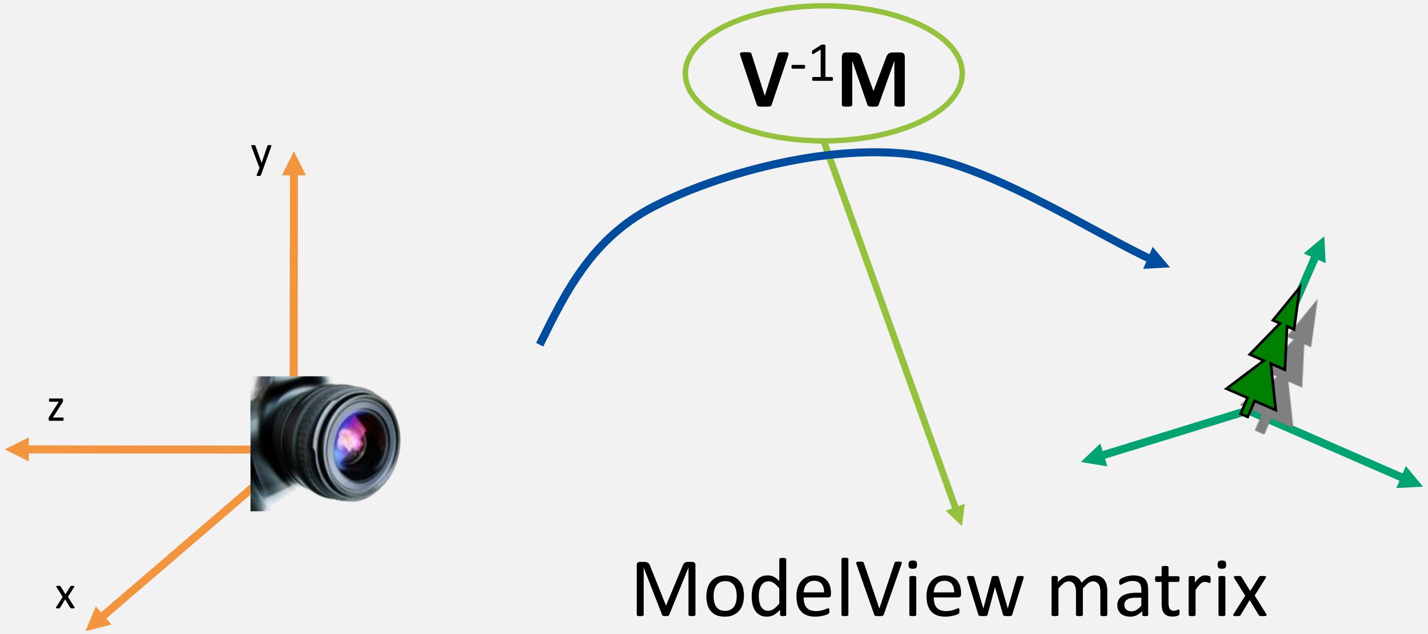
3D Case

- $\mathbf{x}_{view} = \mathbf{V}^{-1}\mathbf{M}\ \mathbf{x}_{obj}$



3D Case

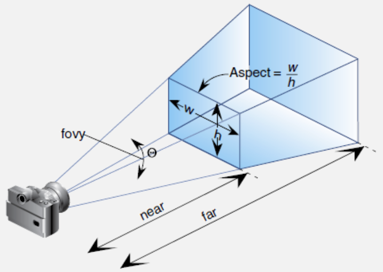
- In OpenGL, $\mathbf{V}^{-1}\mathbf{M}$ is called as the ModelView matrix
  - GL_MODELVIEW_MATRIX



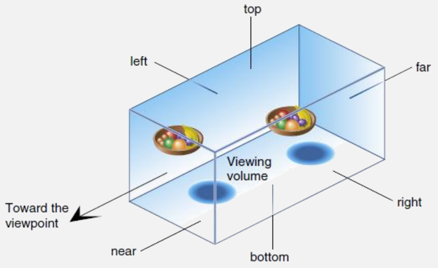
Projection matrix

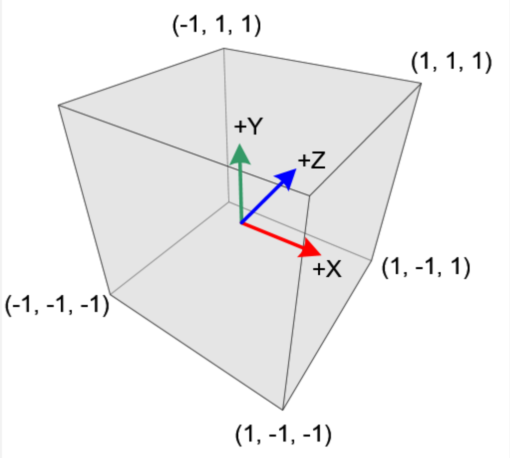
What is Projection Matrix?

- Projection matrix **P** transforms camera coordinates into clip coordinates
  - $\mathbf{x}_{clip} = \mathbf{P}\mathbf{x}_{view}$
    $= \mathbf{P}\mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj}$
- The canonical view volume is defined in the clip space



or

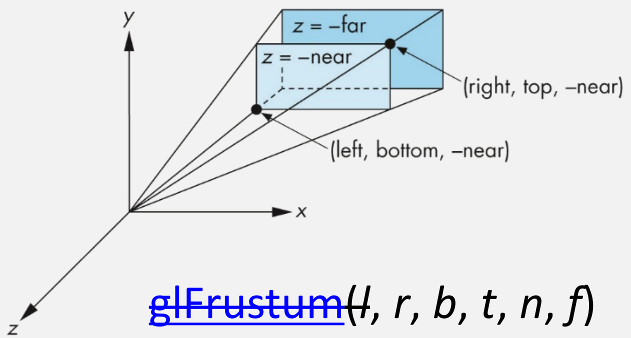




canonical view volume
in the clip space

Projection Matrix

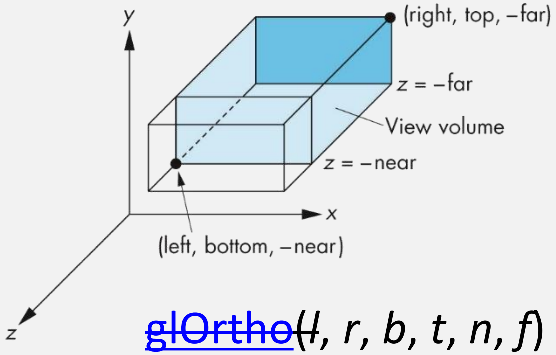
- Perspective projection



~~glFrustum~~(l, r, b, t, n, f)

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

- Orthographic projection

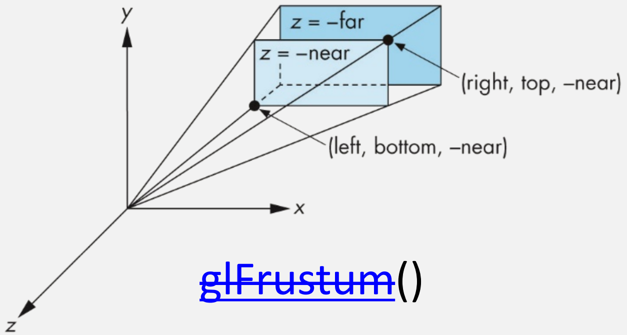


~~glOrtho~~(l, r, b, t, n, f)

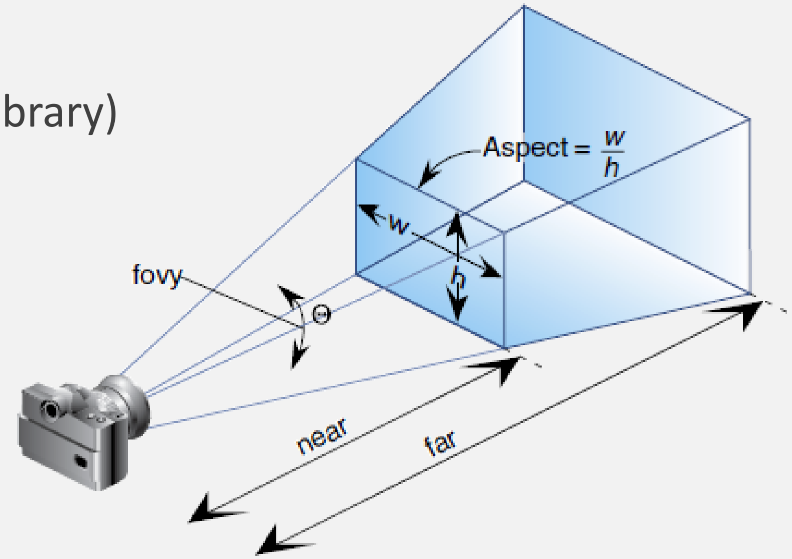
$$\boldsymbol{P} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection

- Focal length
  - In OpenGL, there is no physical meaning
- Field of view (FOV)
  - In OpenGL, zoom-in/-out is handled by changing the field of view
    - Perspective projection: ~~glFrustum~~() (or ~~gluPerspective~~() in GLU library)



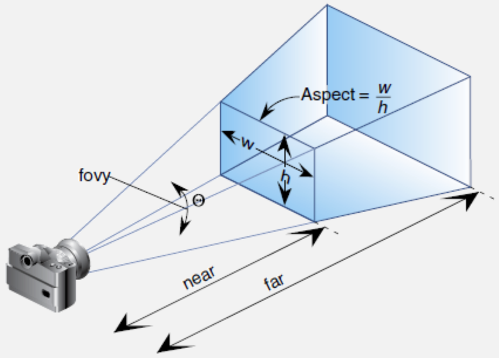
~~glFrustum~~()



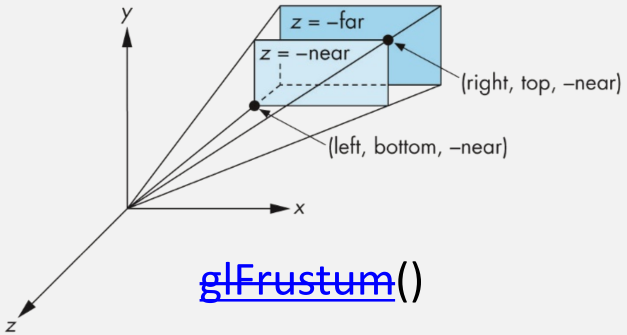
~~gluPerspective~~()

Perspective Projection: gluPerspective() → glFrustum()

3D case



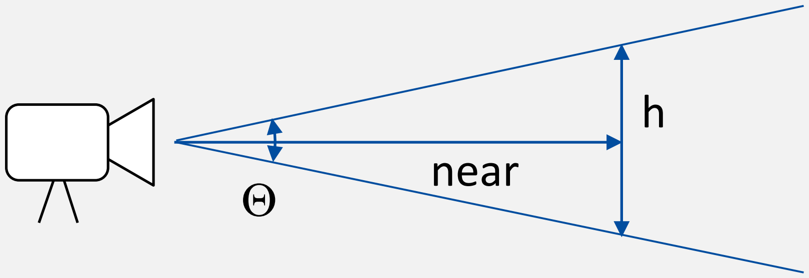
~~gluPerspective~~()



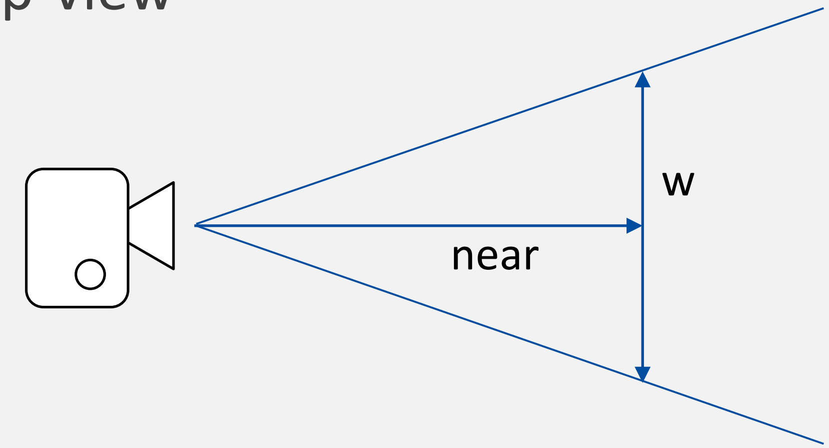
~~glFrustum~~()

Side-/Top-view of gluPerspective()

- Side-view



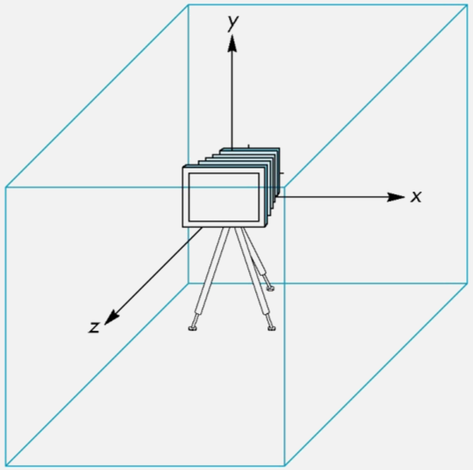
- Top-view

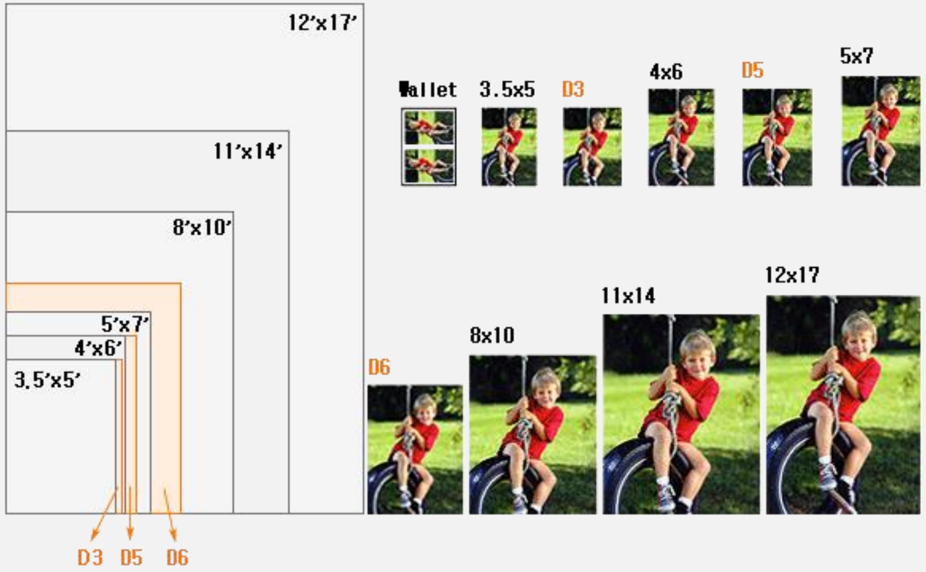


Viewport

Camera Specification – Viewport

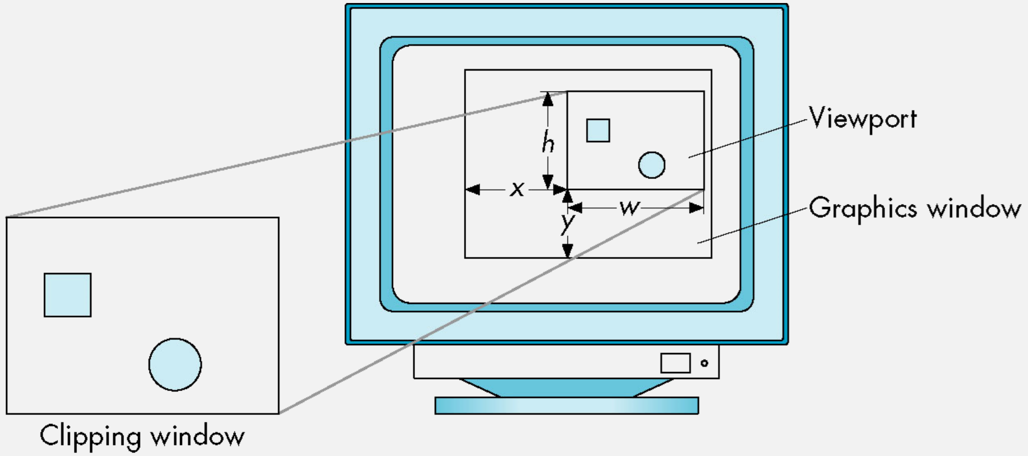
- Viewport
  - Similar to the size of photo printing
    - A film → Photos of different sizes
  - A rectangular area of the display window





Camera Specification – Viewport

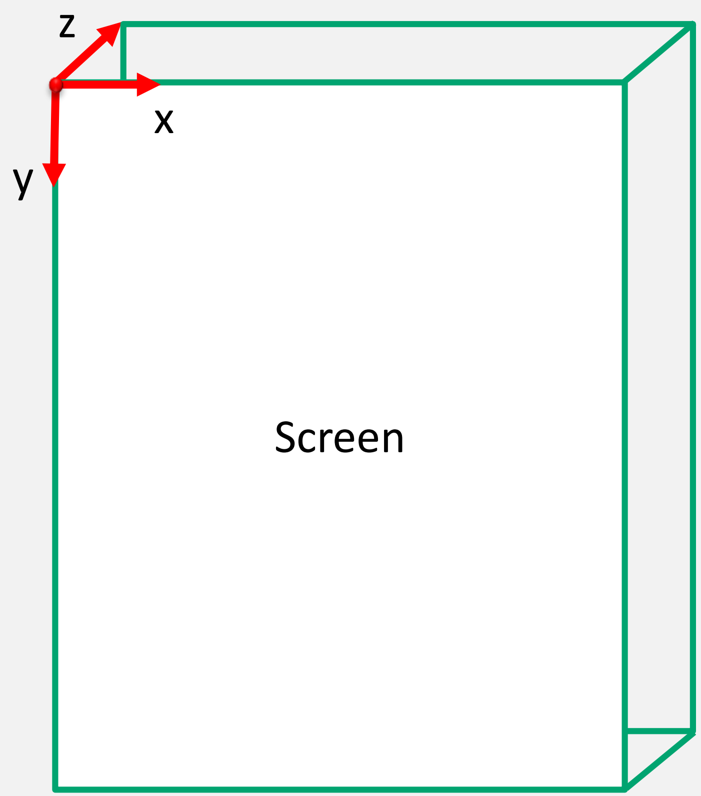
- Viewport
  - Similar to the size of photo printing
    - A film → Photos of different sizes
  - A rectangular area of the display window: x, y, w, h
    - (x, y): the lower-left corner of the viewport
    - w, h: the width and height of the viewport



A mapping to the viewport

What is Viewport?

- Viewport matrix **W** transforms clip coordinates into screen-space coordinates
  - $\mathbf{x}_{screen} = \mathbf{W}\mathbf{x}_{clip}$
    $= \mathbf{W}\mathbf{P}\mathbf{x}_{view}$
    $= \mathbf{W}\mathbf{P}\mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj}$ $= \begin{bmatrix} win_x \\ win_y \\ win_z \end{bmatrix}$



- (win_x, win_y) are screen-space coordinates
  - (win_x, win_y) units are in pixel (with fractions)
- win_z is depth coordinate
  - win_z is in range of 0.0 to 1.0, or depth range
    - See details in [glDepthRange]()

Relationship between Aspect Ratio & Viewport

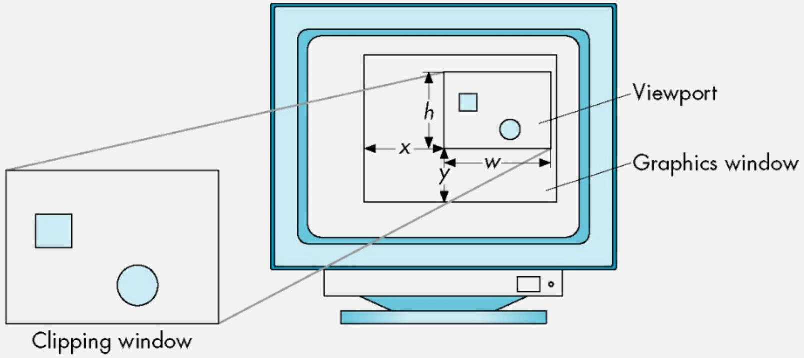
What’s wrong with Aspect Ratio? ([link]())

Wrong Aspect Ratio

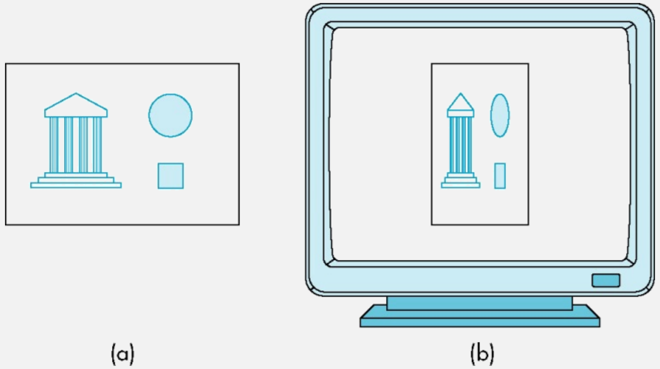
Correct Aspect Ratio

Camera Specification – Aspect ratio

- Aspect ratio
  - width / height
    - For aspect ratio, absolute sizes of width & height are meaningless
    - Aspect ratio of display window (e.g., device screen) is important



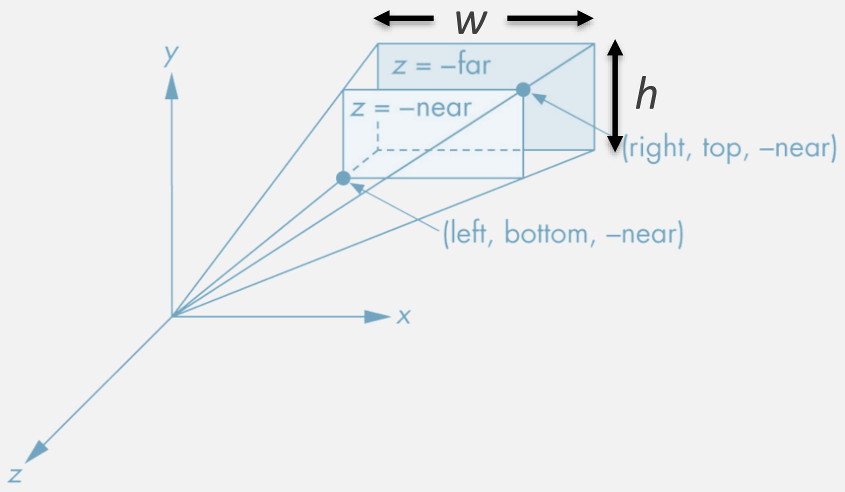
A mapping to the viewport

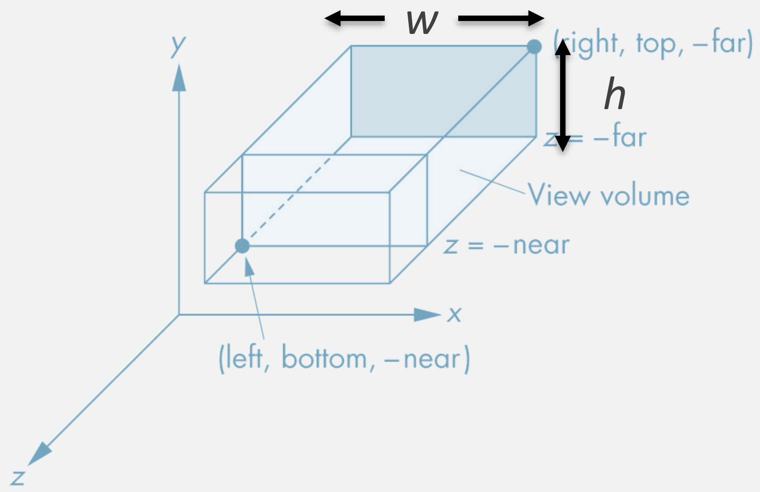


Aspect-ratio mismatch.
(a) viewing rectangle, (b) display window

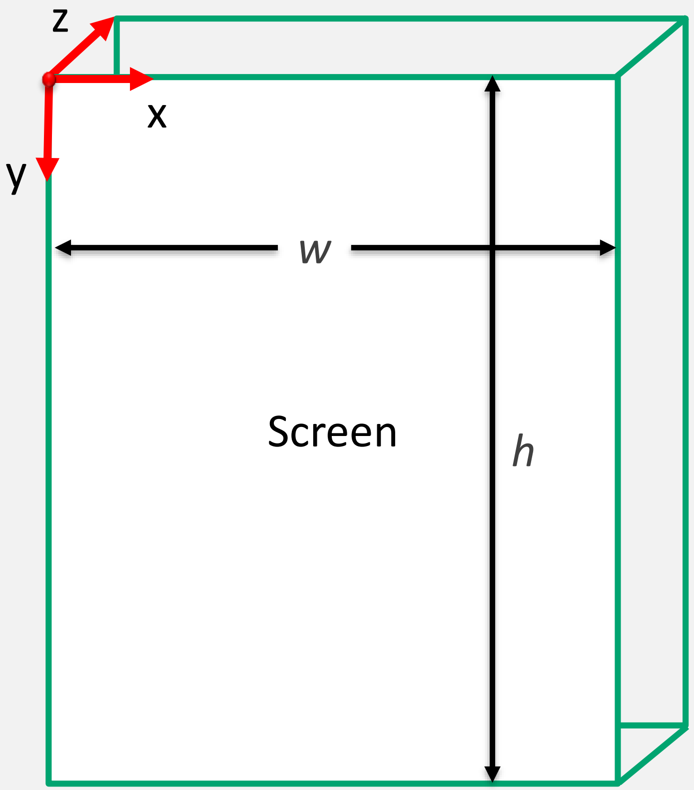
Camera Specification – Aspect ratio

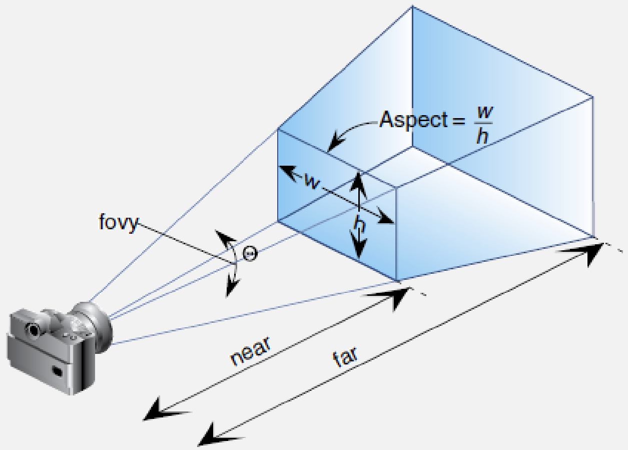
- Aspect ratio
  - width / height
    - For aspect ratio, absolute sizes of width & height are meaningless
    - Aspect ratio of display window (i.e., device screen) is important



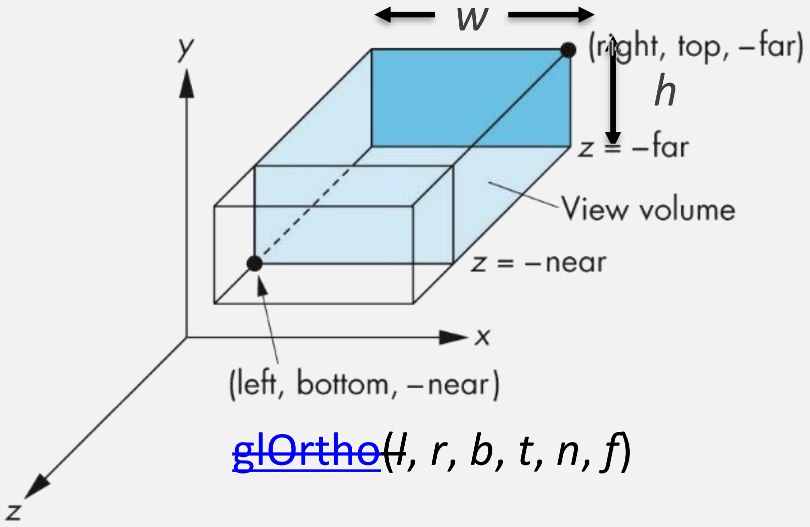


Aspect Ratio (= w/h)





~~gluPerspective~~()



~~glOrtho~~(l, r, b, t, n, f)

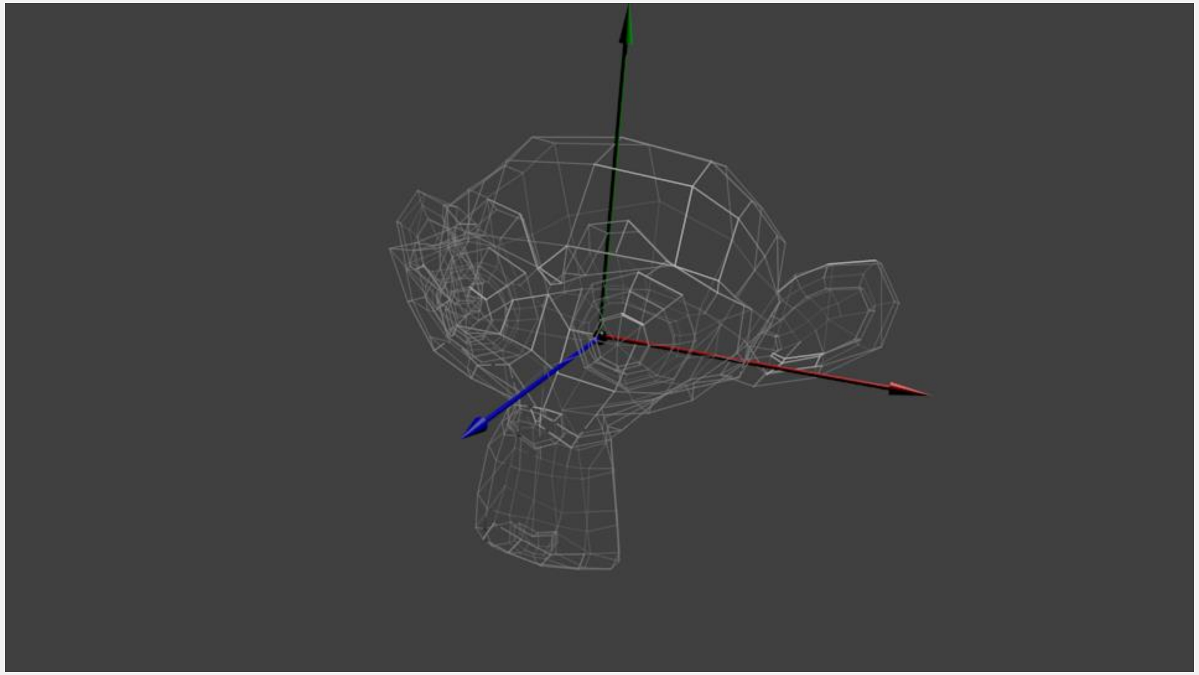
- References: opengl-tutorial
  - Tutorial 3: Matrices (link)

Recap:
Model, View, Projection, Viewport Transformations

Model Transformation

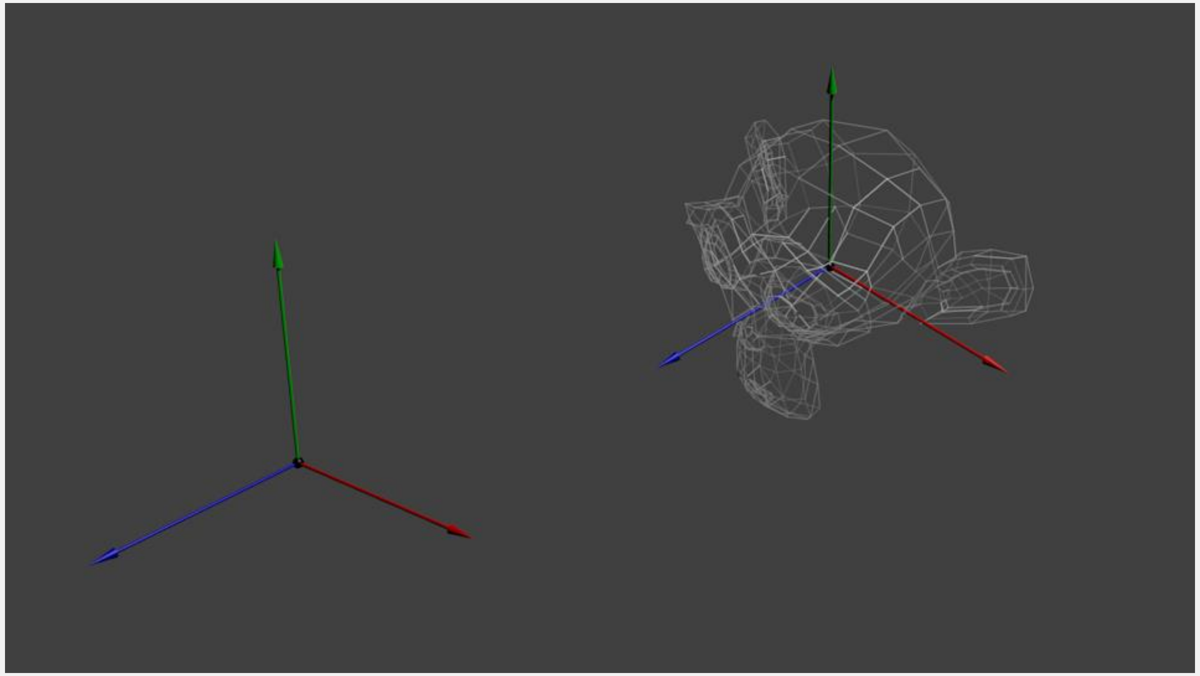
[Model space]

$\mathbf{x}_{obj}$



[World space ← Model space]

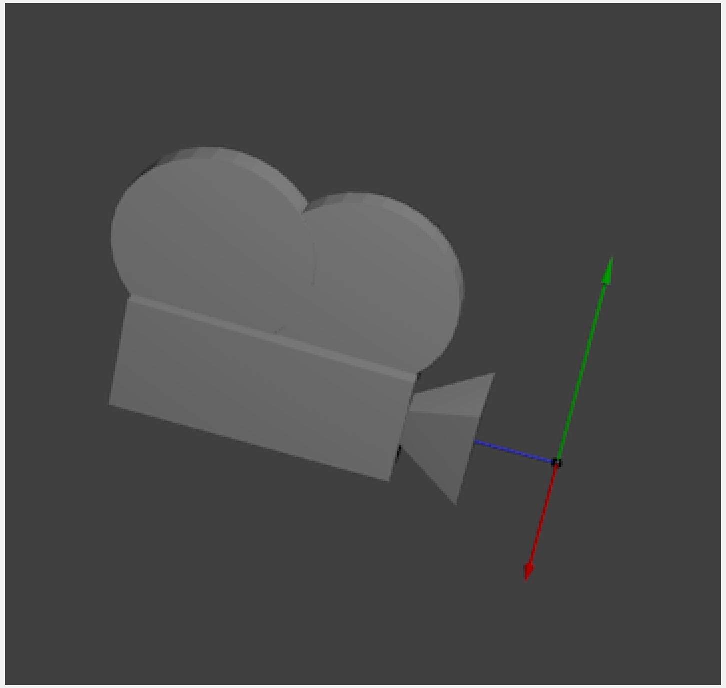
$\mathbf{M}\mathbf{x}_{obj}$



View Transformation

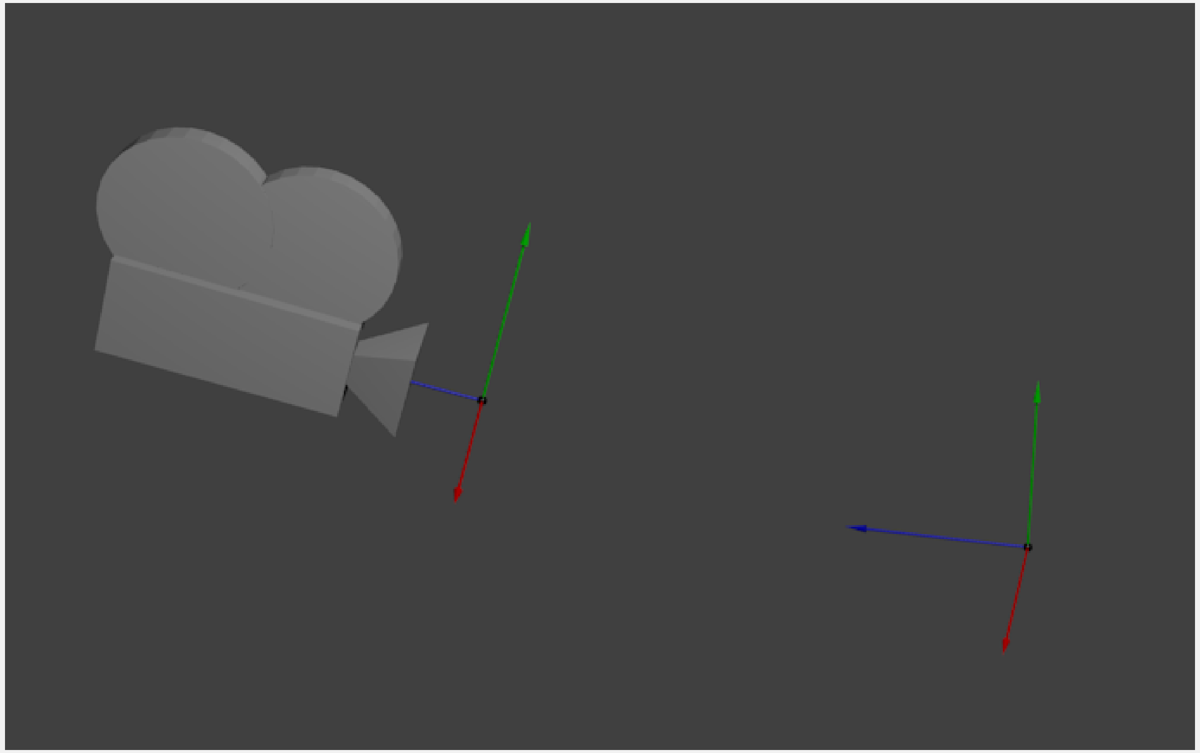
[Camera space]

$\mathbf{x}_{view}$



[World space ← Camera space]

$\mathbf{Vx}_{view}$



ModelView Transformation

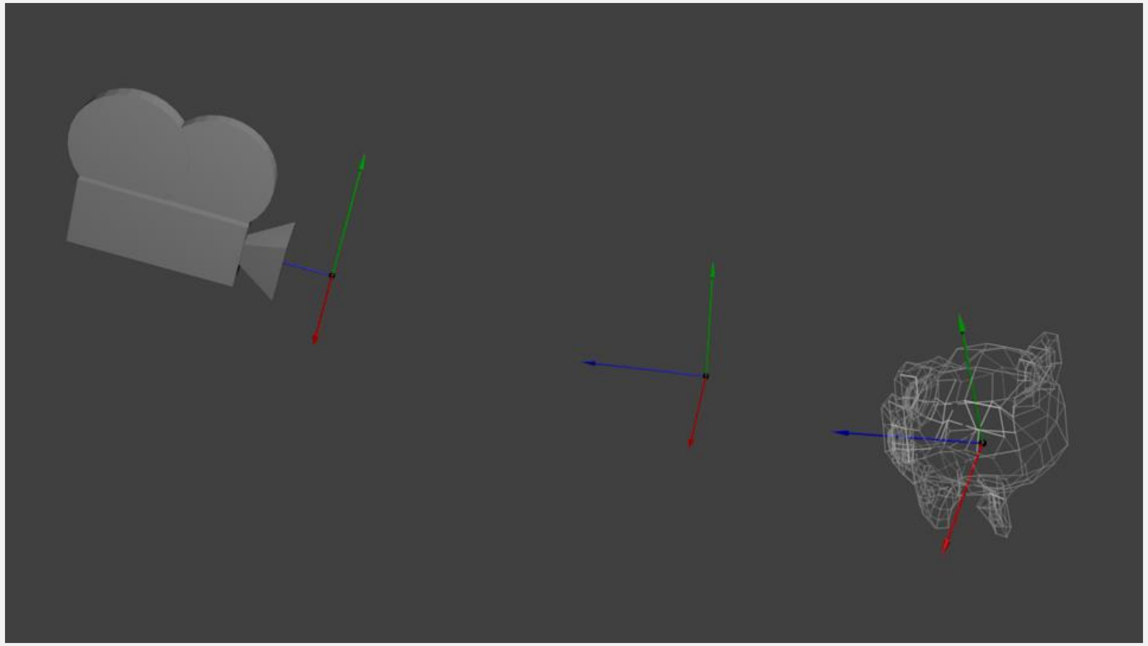
[Camera space ← Model space]

~~gluLookAt~~()

mat_Model = T*R*S

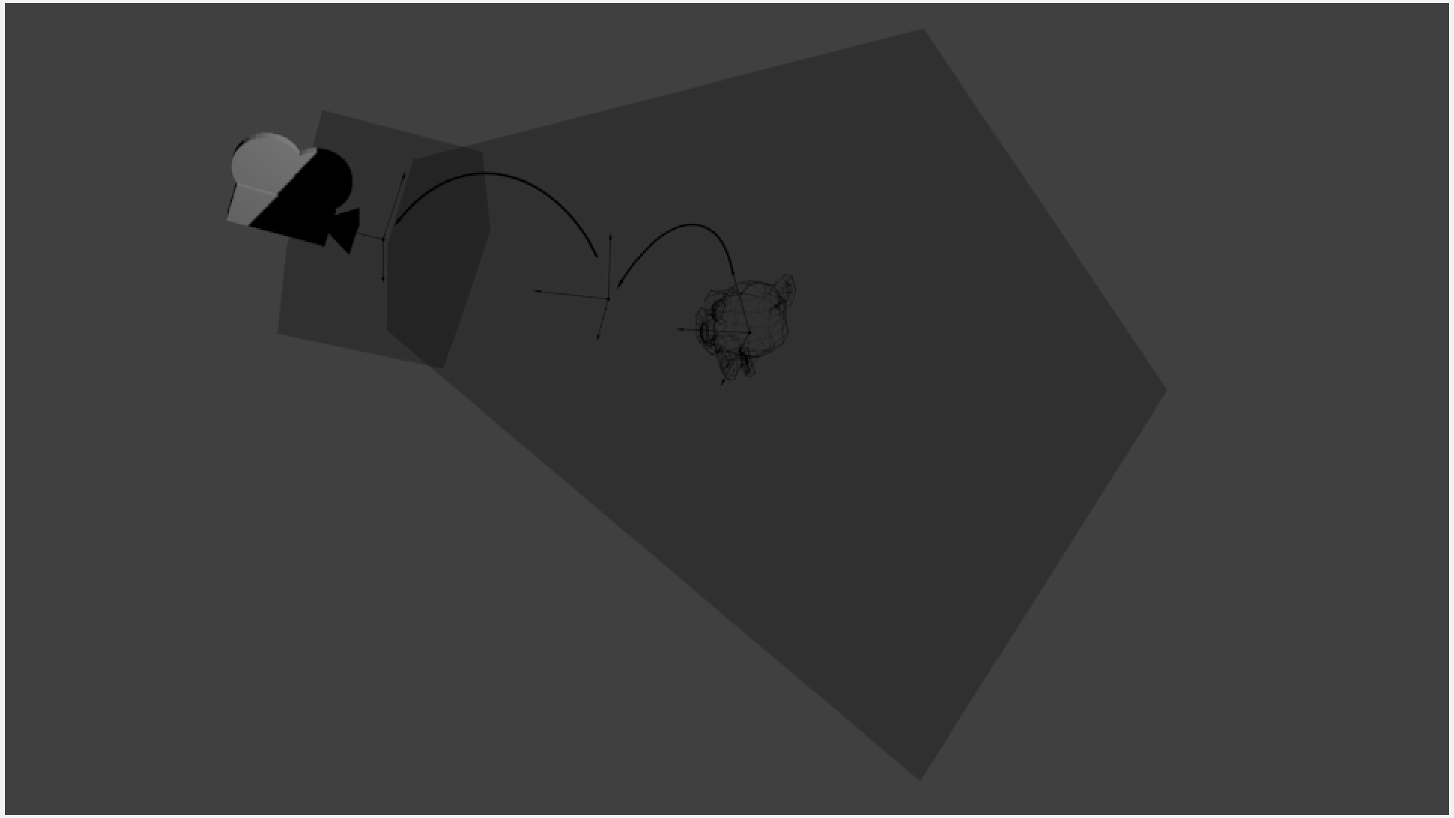
$$\mathbf{x}_{view} = \mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj}$$

(Camera space) (Model space)



Projection Matrix

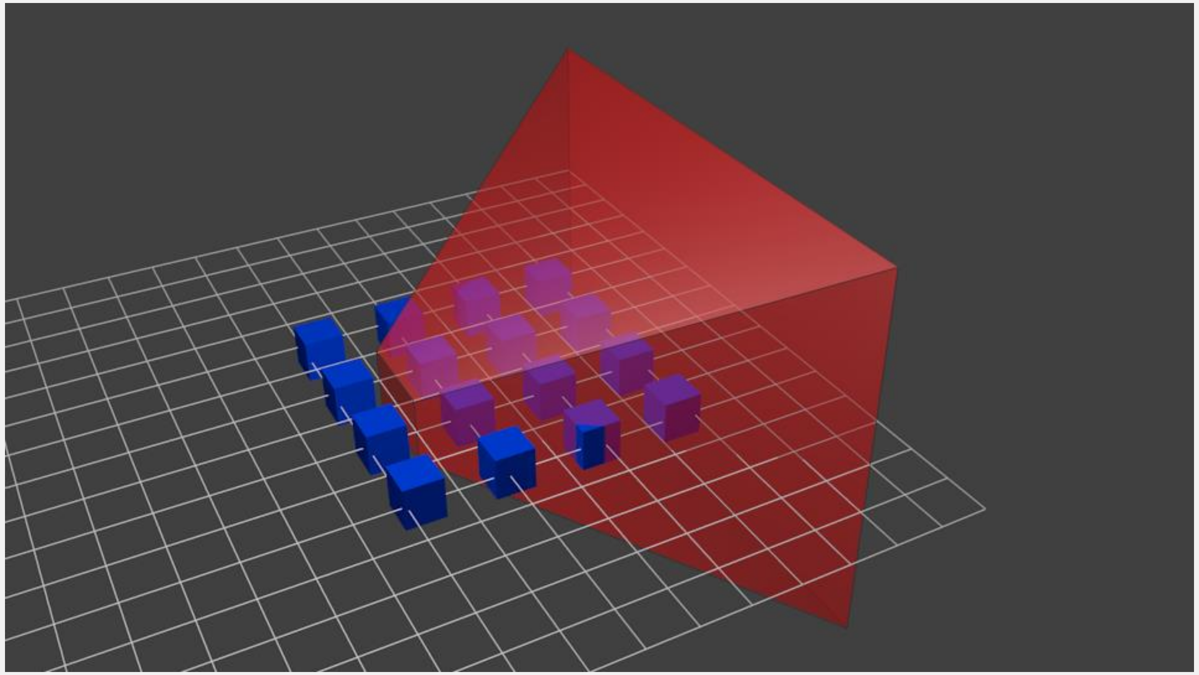
- Now, we consider a view **frustum**

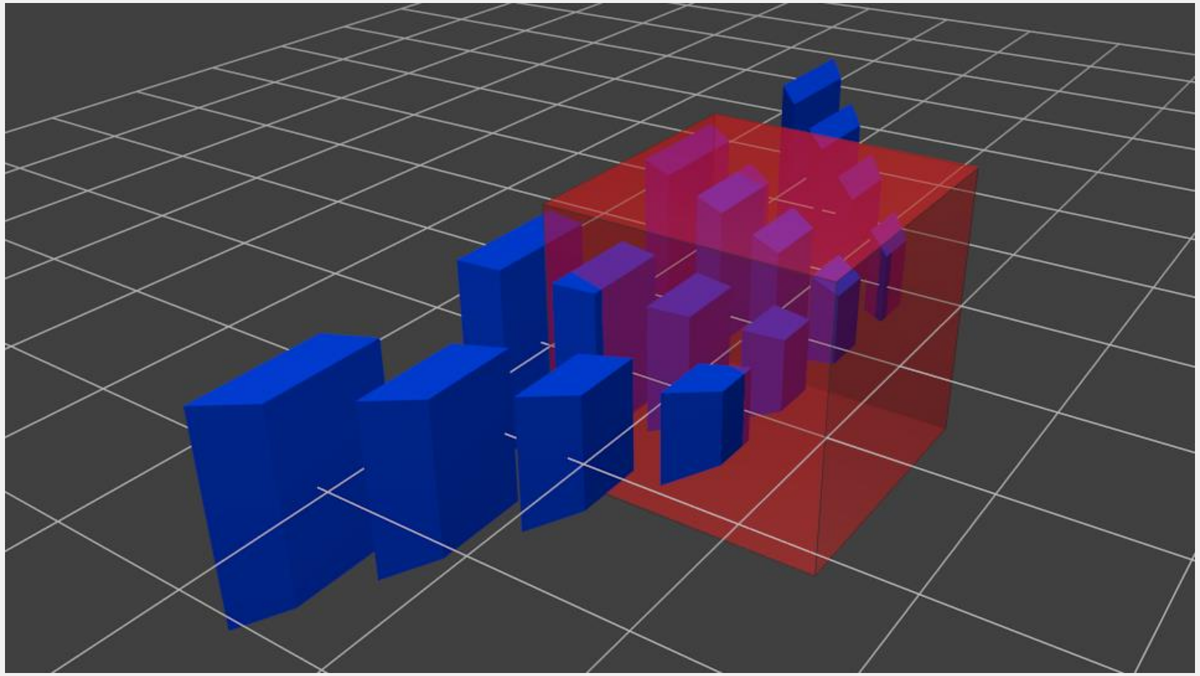


Projection Matrix

[Camera space]
$\mathbf{x}_{view}$ **(=** $\mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj}$**)**

[Clipping space]
$\mathbf{x}_{clip}$ **(=** $\mathbf{P}\mathbf{x}_{view} = \mathbf{P}\mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj}$**)**

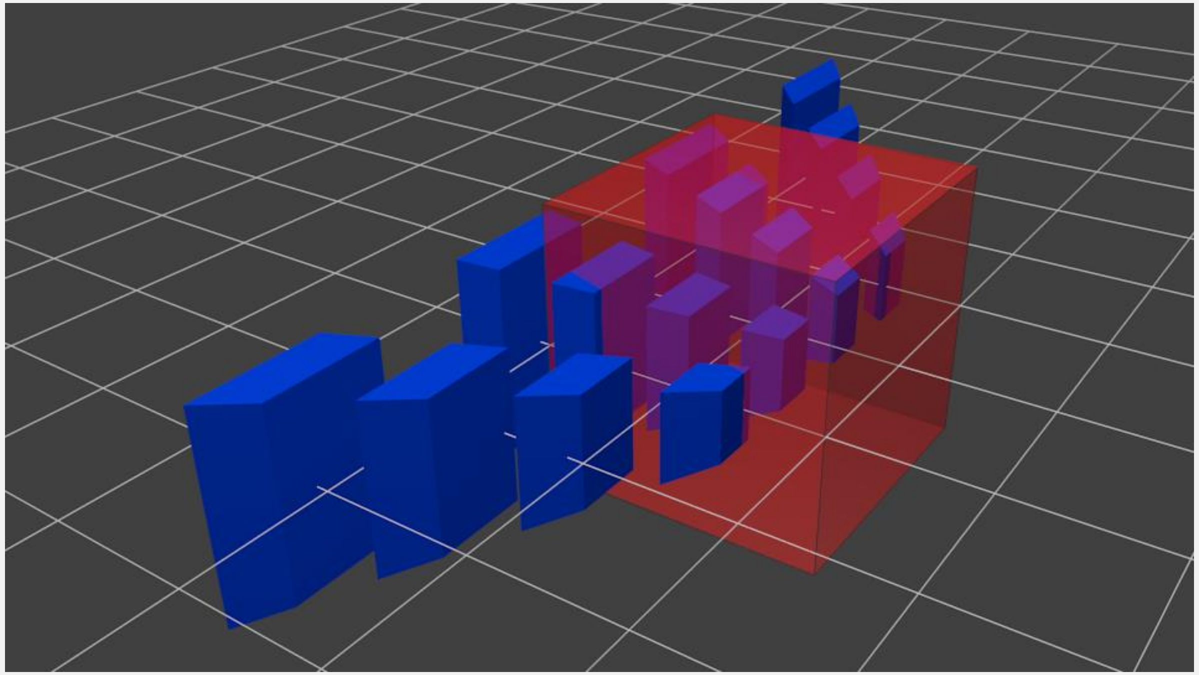




Viewport

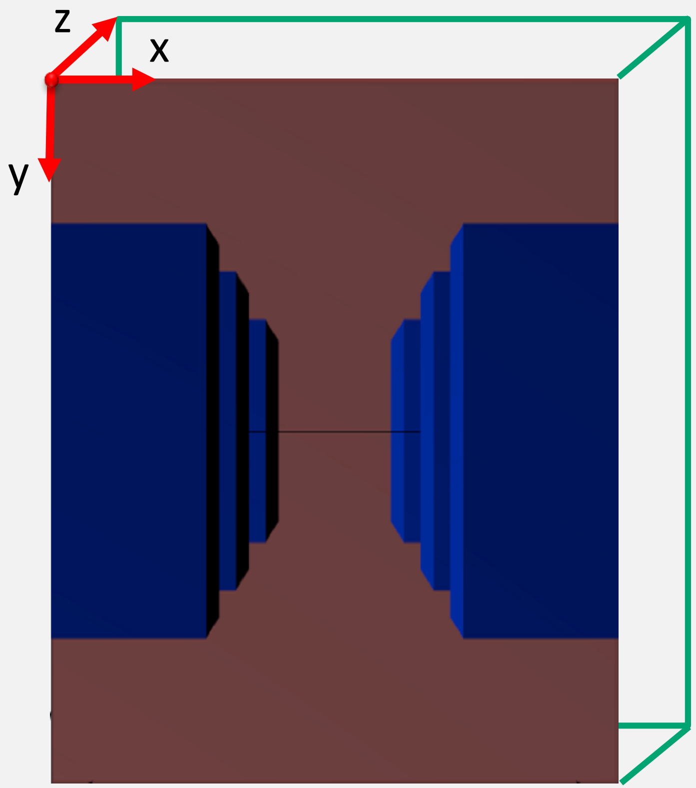
[Clipping space]

$\mathbf{x}_{clip}$ $(= \mathbf{P}\mathbf{x}_{view} = \mathbf{P}\mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj})$



[Screen space]

$\mathbf{x}_{screen}$ $(=\mathbf{W}\mathbf{x}_{clip} = \mathbf{W}\mathbf{P}\mathbf{x}_{view} = \mathbf{W}\mathbf{P}\mathbf{V}^{-1}\mathbf{M}\mathbf{x}_{obj})$



감사합니다

Contacts:

- Prof. Junho Kim junho@kookmin.ac.kr