Computer Graphics

3. Viewing in 3D

I-Chen Lin

National Yang Ming Chiao Tung University

Textbook: E.Angel, D. Shreiner Interactive Computer Graphics, 6th Ed., Pearson
Ref: D.D. Hearn, M. P. Baker, W. Carithers, Computer Graphics with OpenGL, 4th Ed., Pearson

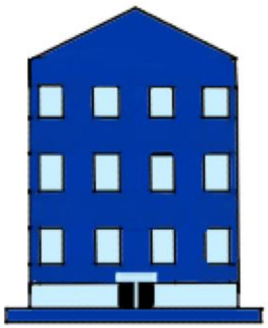
Intended Learning Outcomes

- On completion of this chapter, a student will be able to:
  - Outline the stages of the graphics pipeline.
  - Describe the sequence of transformations for viewing 3D objects (with graphics pipeline).
  - Identify and apply the transformations through OpenGL API.

Outline

- Classical views
- Computer viewing
- Projection matrices

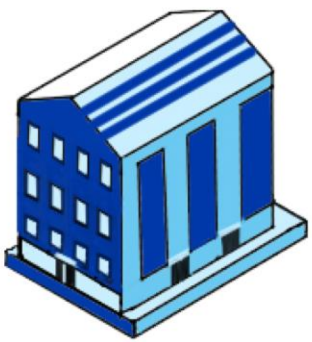
Classical Projections



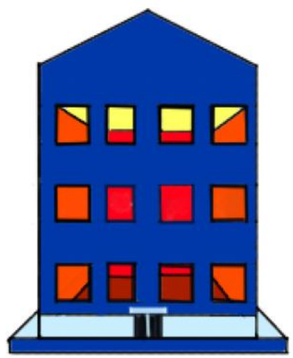
Front elevation

Elevation oblique

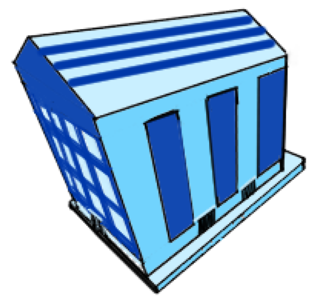
Plan oblique



Isometric

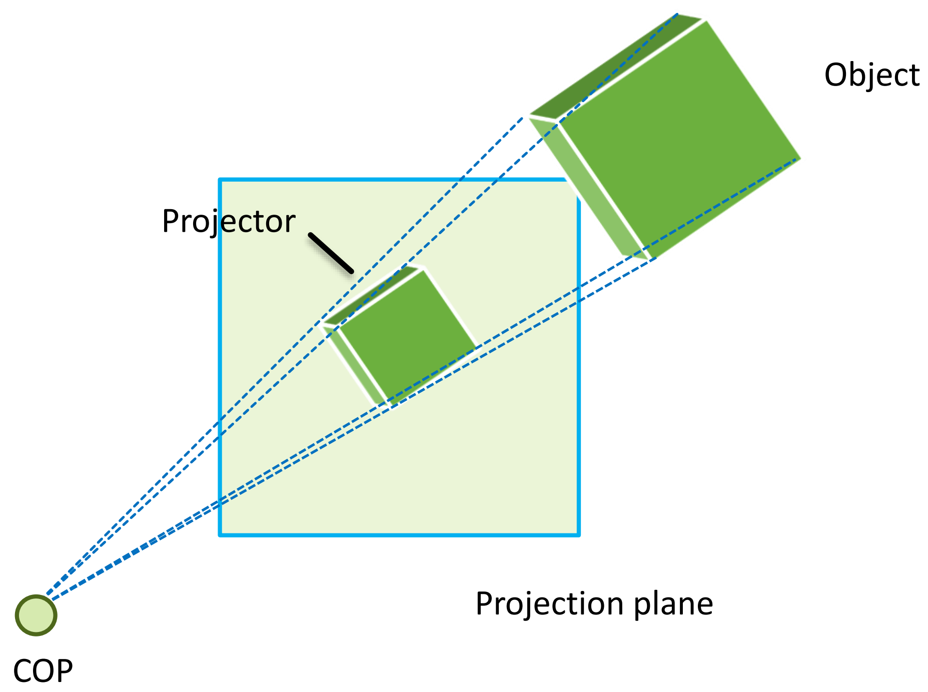


One-point perspective

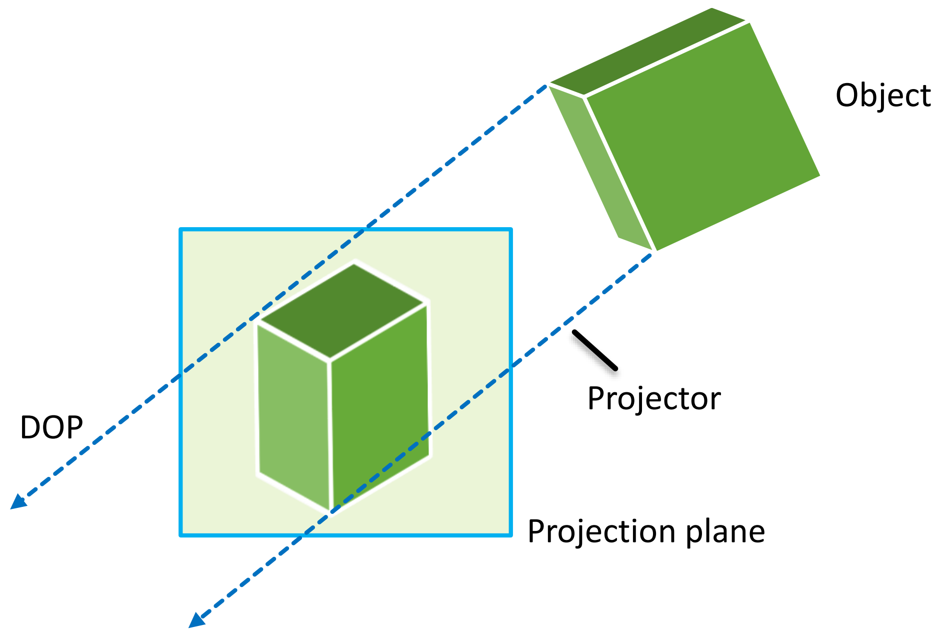


Three-point perspective

Perspective Projection

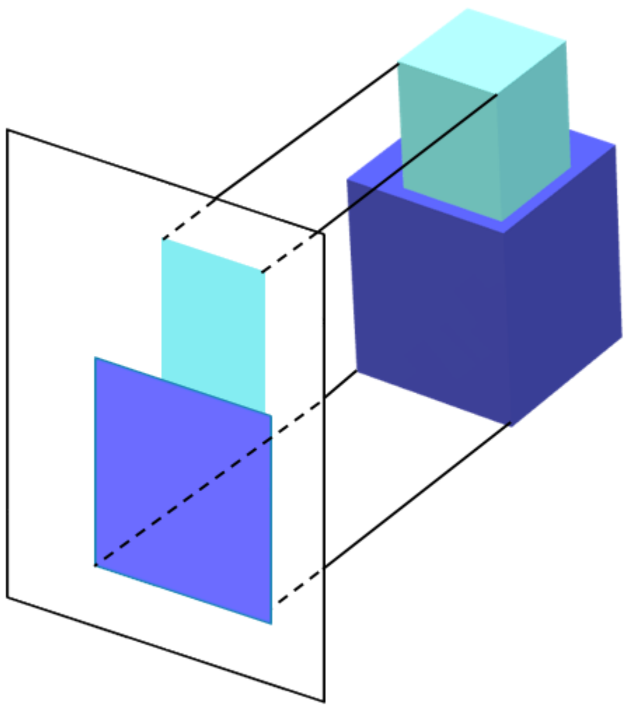


Parallel Projection



Orthographic Projection

- Projectors are orthogonal to projection surface

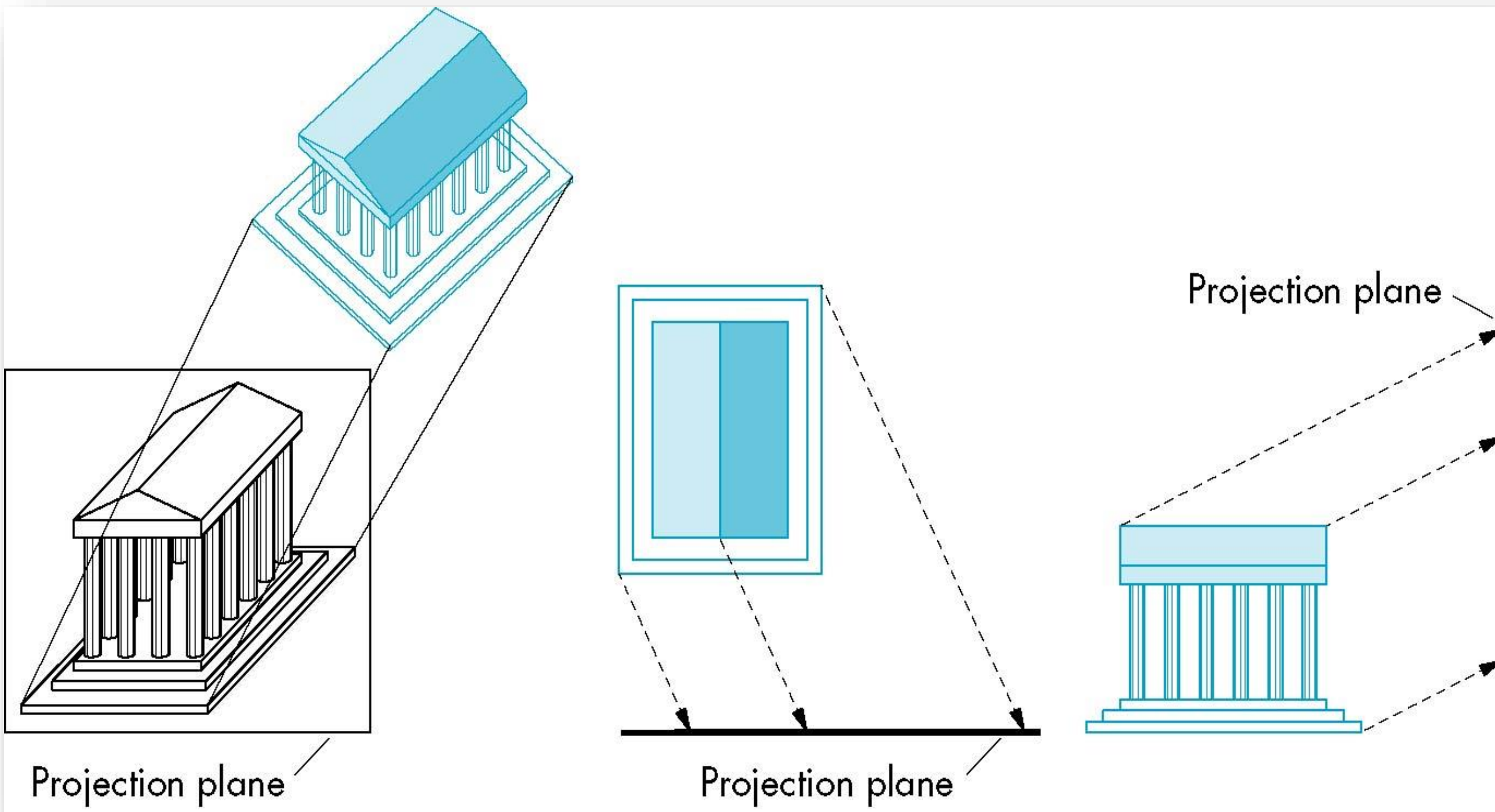


Advantages and Disadvantages

- Preserves both distances and angles
  - Shapes preserved
  - Can be used for measurements
    - Building plans
    - Manuals
- Cannot see what object really looks like because many surfaces hidden from view
  - Often we add the isometric

Oblique Projection

- Arbitrary relationship between projectors and projection plane



Advantages and Disadvantages

- Can pick the angles to emphasize a particular face
  - Architecture: plan oblique, elevation oblique

- Angles in faces parallel to the projection plane are preserved while we can still see “around” side

Computer Viewing

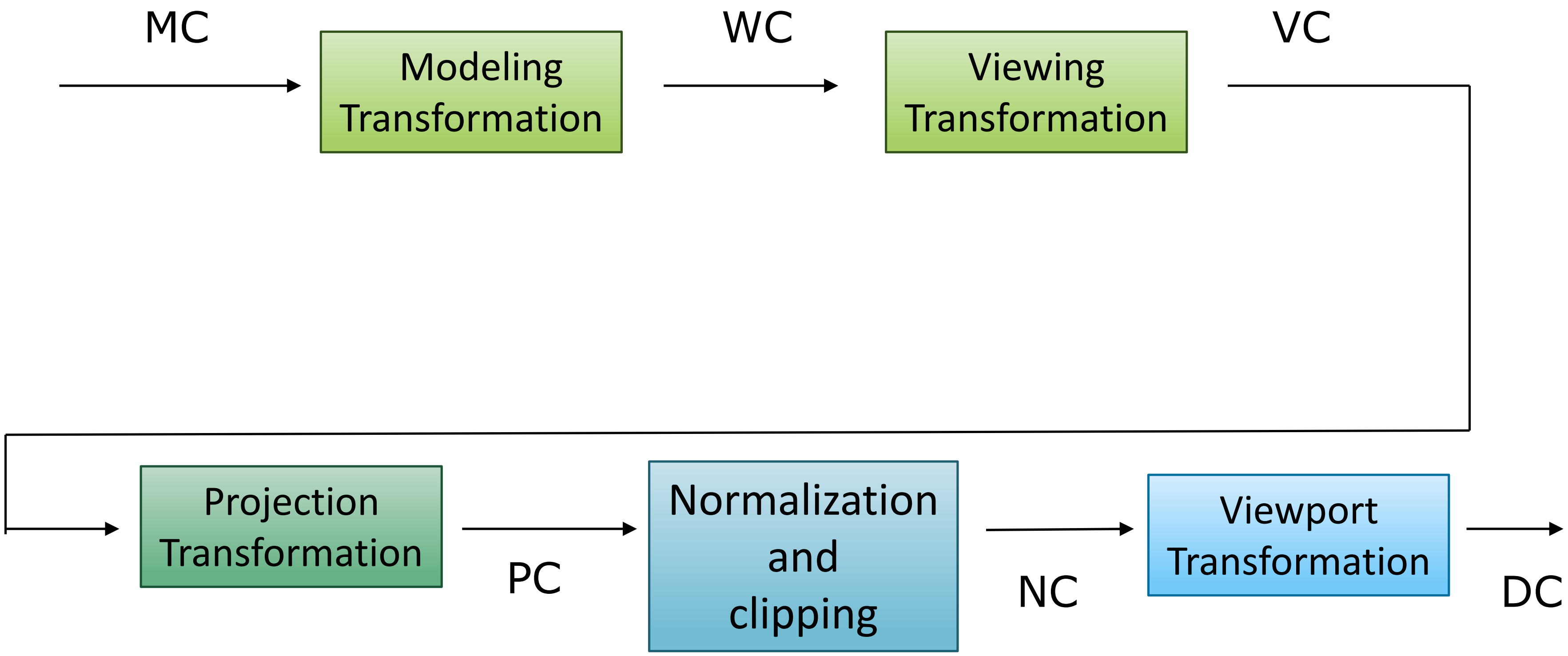
Computer Viewing

- Three aspects of the viewing process implemented in the pipeline:
  - Positioning the camera
    - Setting the *model-view matrix*
  - Selecting a lens
    - Setting the *projection matrix*
  - Clipping
    - Setting the *view volume*

The OpenGL Camera

- In OpenGL, initially the object and camera frames are the same
  - Default model-view matrix is an identity
- The camera is located at origin and points in the negative z direction
- OpenGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
  - Default projection matrix is an identity

Graphics Pipeline and Transformations



Let’s skip the clipping details temporarily !

Moving the Camera Frame

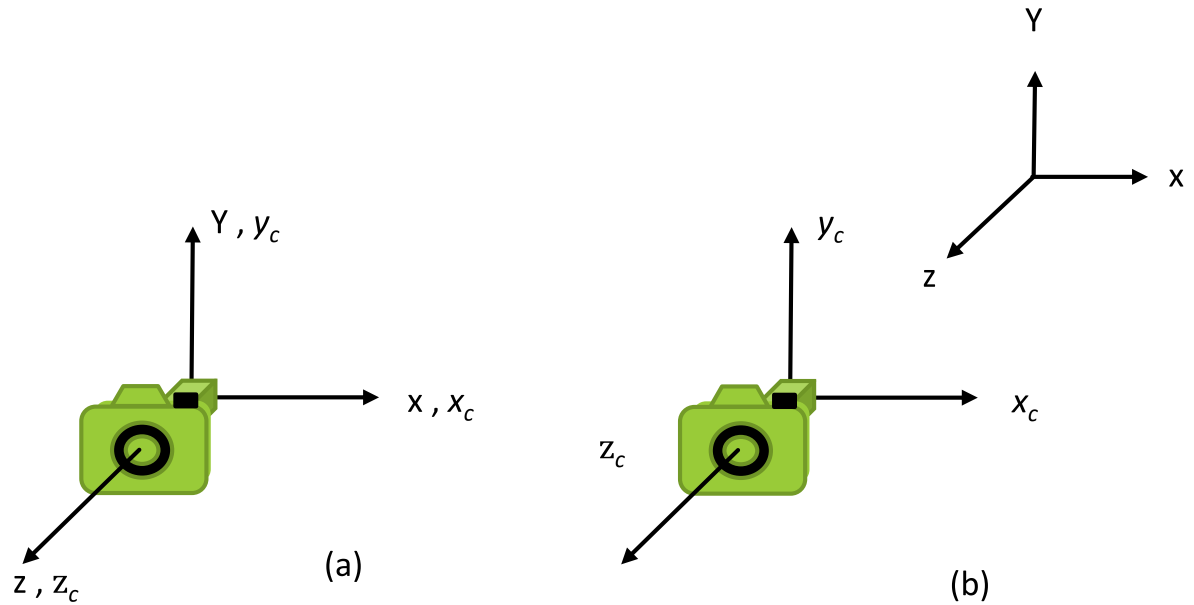
- If we want to visualize object with both positive and negative z values we can either
  - Move the camera in the positive z direction
    - Translate the camera frame
  - Move the objects in the negative z direction
    - Translate the world frame

- Both of these views are equivalent and are determined by the model-view matrix
  - Want a translation (`glTranslatef(0.0,0.0,-d);`)
  - `d > 0`

Moving Camera back from Origin

default frames

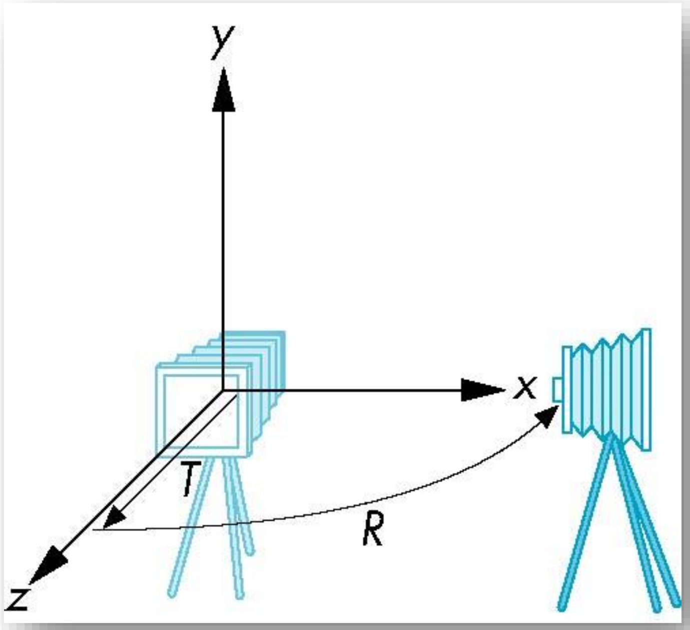
frames after translation by –d
d > 0



Moving the Camera

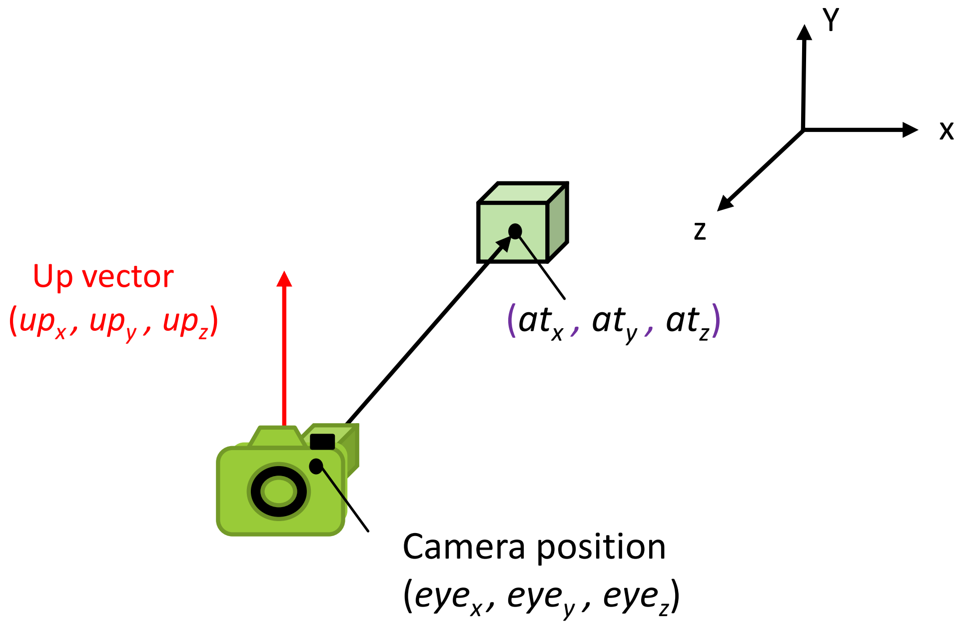
- We can move the camera to any desired position by a sequence of rotations and translations

- Example: side view
  - Move it away from origin
  - Rotate the camera
  - Apply C = T'R' to model-view matrix

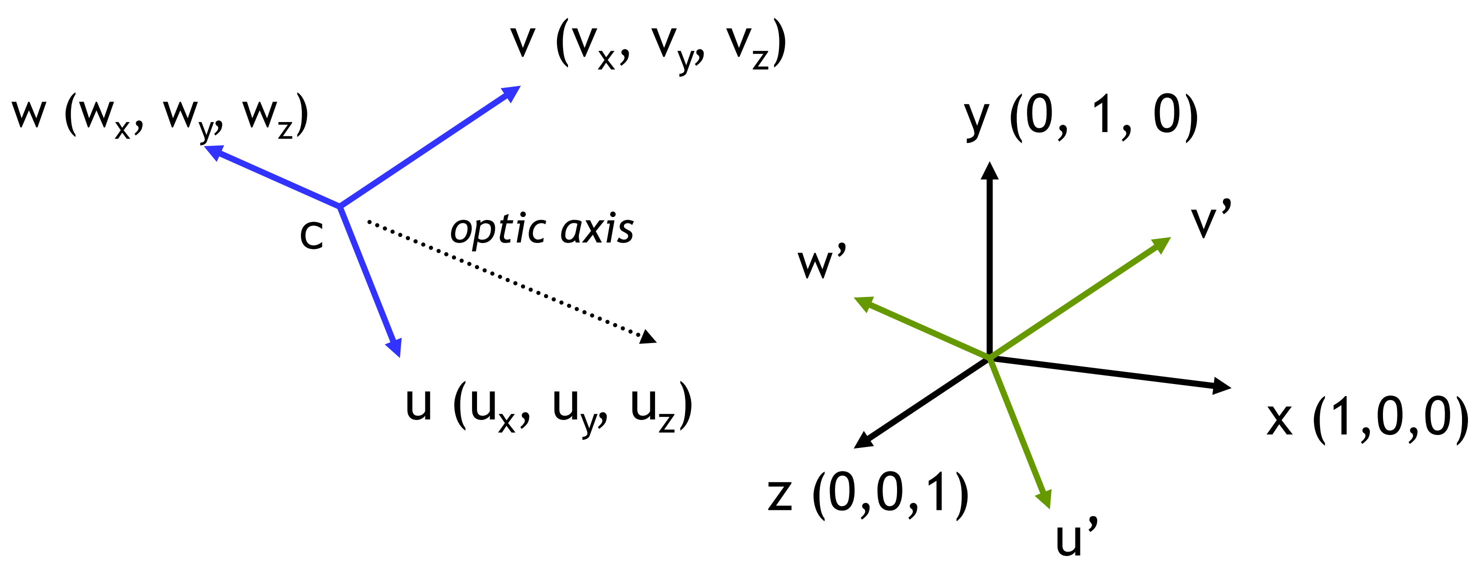


gluLookAt

- gluLookAt(eye_x, eye_y, eye_z, at_x, at_y, at_z, up_x, up_y, up_z)



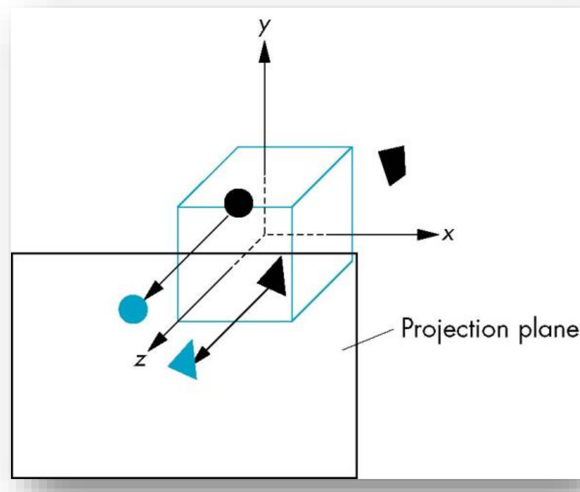
By Coordinate Transformations



$$\begin{bmatrix} x_{wc} \\ y_{wc} \\ z_{wc} \\ 1 \end{bmatrix} = \begin{bmatrix} u'_x & v'_x & w'_x & 0 \\ u'_y & v'_y & w'_y & 0 \\ u'_z & v'_z & w'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x'_{vc} \\ y'_{vc} \\ z'_{vc} \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x'_{vc} \\ y'_{vc} \\ z'_{vc} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{vc} \\ y_{vc} \\ z_{vc} \\ 1 \end{bmatrix}$$

Projections and Normalization

- The default projection in the eye (camera) frame is orthogonal
- For points within the default view volume
  - $x_p = x$
  - $y_p = y$
  - $z_p = 0$



- Most graphics systems use view normalization
  - All other views are converted to the default view by transformations that determine the projection matrix
  - Allows use of the same pipeline for all views

Homogeneous Coordinate Representation

default orthographic projection

- $x_p = x$
- $y_p = y$
- $z_p = 0$
- $w_p = 1$

$$\boldsymbol{p}_p = \boldsymbol{M}\boldsymbol{p}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In practice, we can let **M** = **I** and set the *z* term to zero later

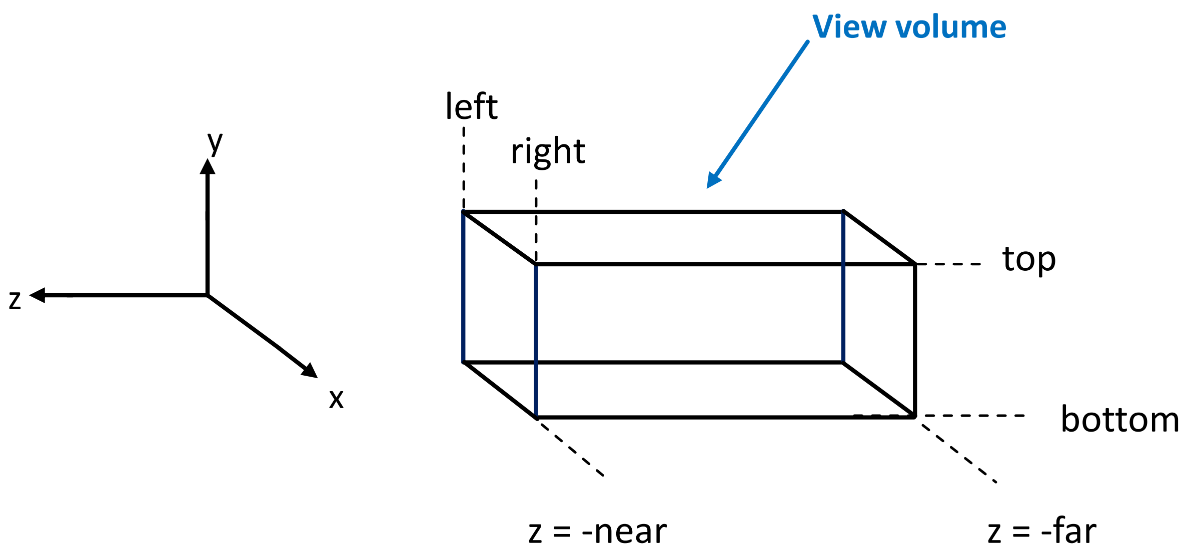
Taking Clipping into Account

- After the view transformation, a simple projection and viewport transformation can generate screen coordinate.
- However, projecting all vertices are usually unnecessary.
- Clipping with 3D volume.
- Associating projection with clipping and normalization.

Why do we use normalization ?

Orthogonal Viewing Volume

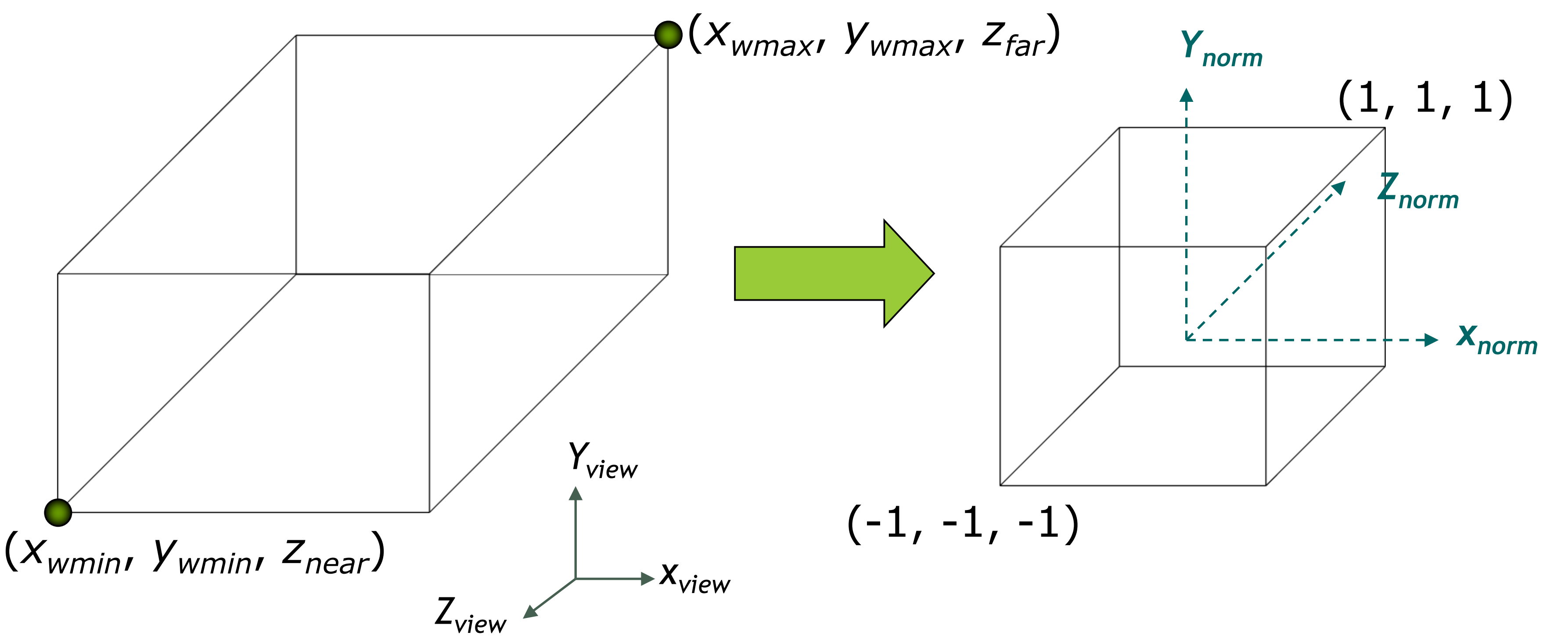
```
Ortho(left,right,bottom,top,near,far)
```



Orthogonal Normalization

`glOrtho(left,right,bottom,top,near,far)`

normalization $\Rightarrow$ find transformation to convert specified clipping volume to default



Orthogonal Matrix

- Two steps
  - T: Move the volume center to origin
  - S: Scale to have sides of length 2

$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} \dfrac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & -\dfrac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \dfrac{2}{yw_{\max} - yw_{\min}} & 0 & -\dfrac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & \dfrac{2}{z_{near} - z_{far}} & \dfrac{z_{near} + z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix maps the near clipping plane, $z = -near = Z_{near}$, to the plane $z = -1$ and the far clipping plane, $z = -far = Z_{far}$, to the plane $z = 1$.

Final Projection

- Set $z = 0$
- Equivalent to the homogeneous coordinate transformation

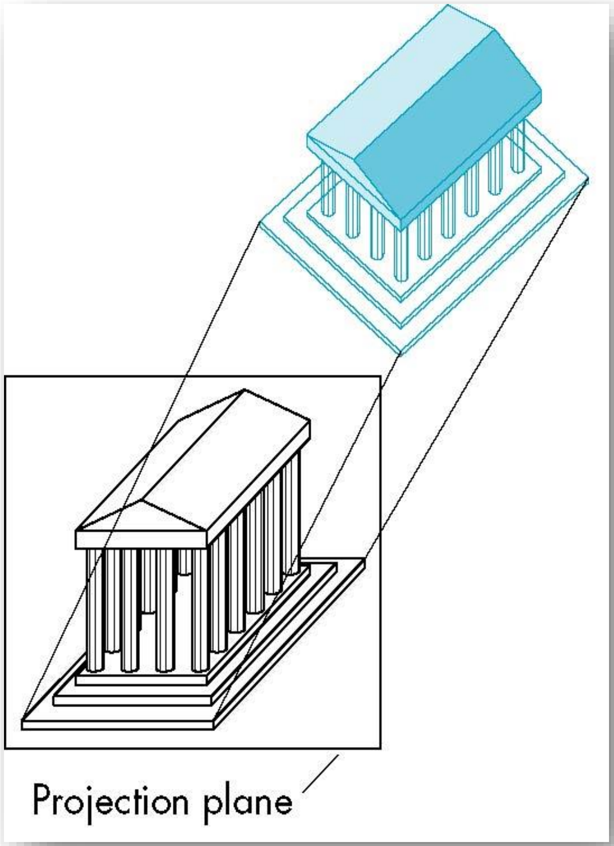
$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Hence, general orthogonal projection in 4D is

$$\mathbf{P} = \mathbf{M}_{\text{orth}}\mathbf{S}\mathbf{T}$$

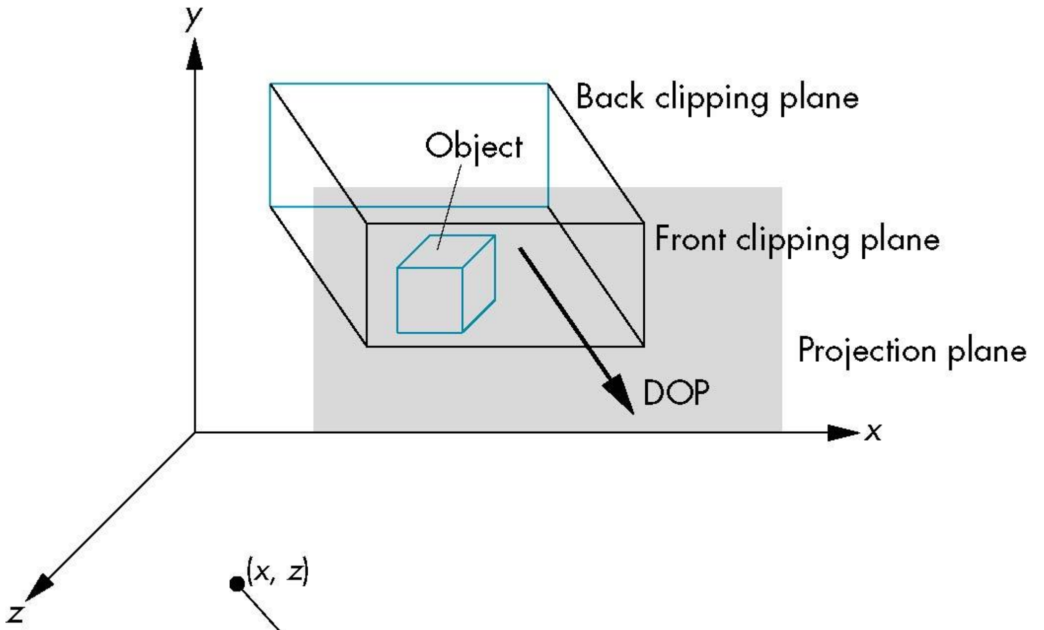
Oblique Projection

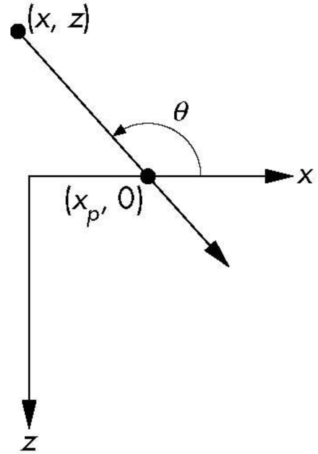
- The OpenGL projection functions cannot produce general parallel projections such as



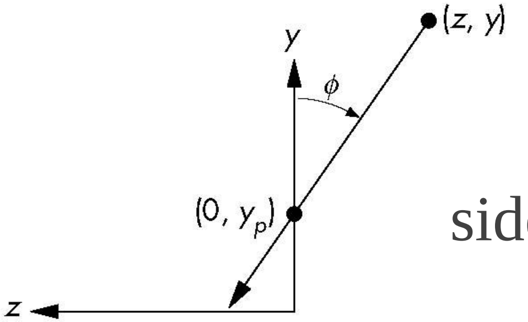
- How to efficiently produce such views?

Shear parallel to the x and y axes





top view



side view

Applying Shear Matrix

xy shear (*z* values unchanged)

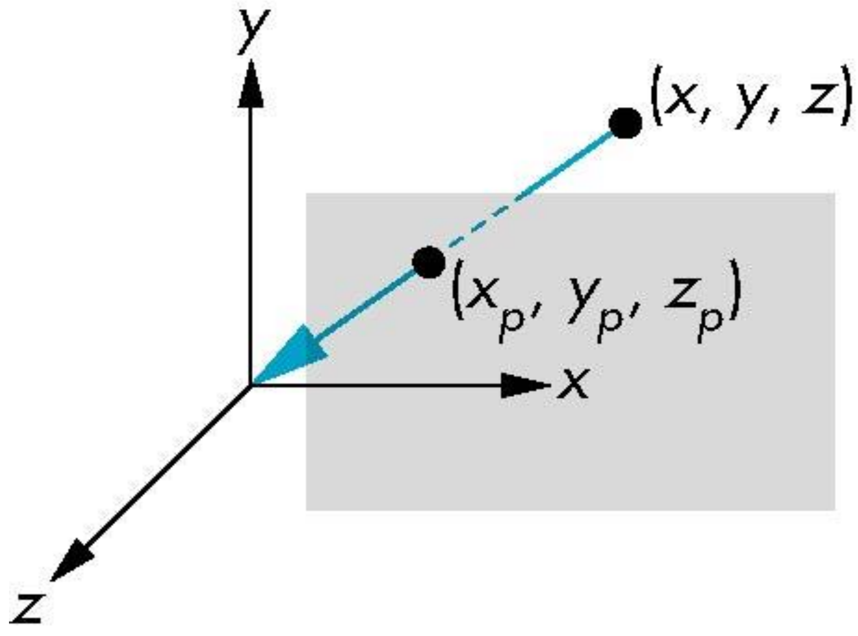
$$\mathrm{H}(\theta, \varphi) = \begin{bmatrix} 1 & 0 & -\cot\theta & 0 \\ 0 & 1 & -\cot\varphi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection matrix $\quad \mathbf{P} = \mathbf{M}_{\mathrm{orth}}\, \mathbf{H}(\theta,\phi)$

General case: $\quad \mathbf{P} = \mathbf{M}_{\mathrm{orth}}\, \mathbf{STH}(\theta,\phi)$

Simple Perspective

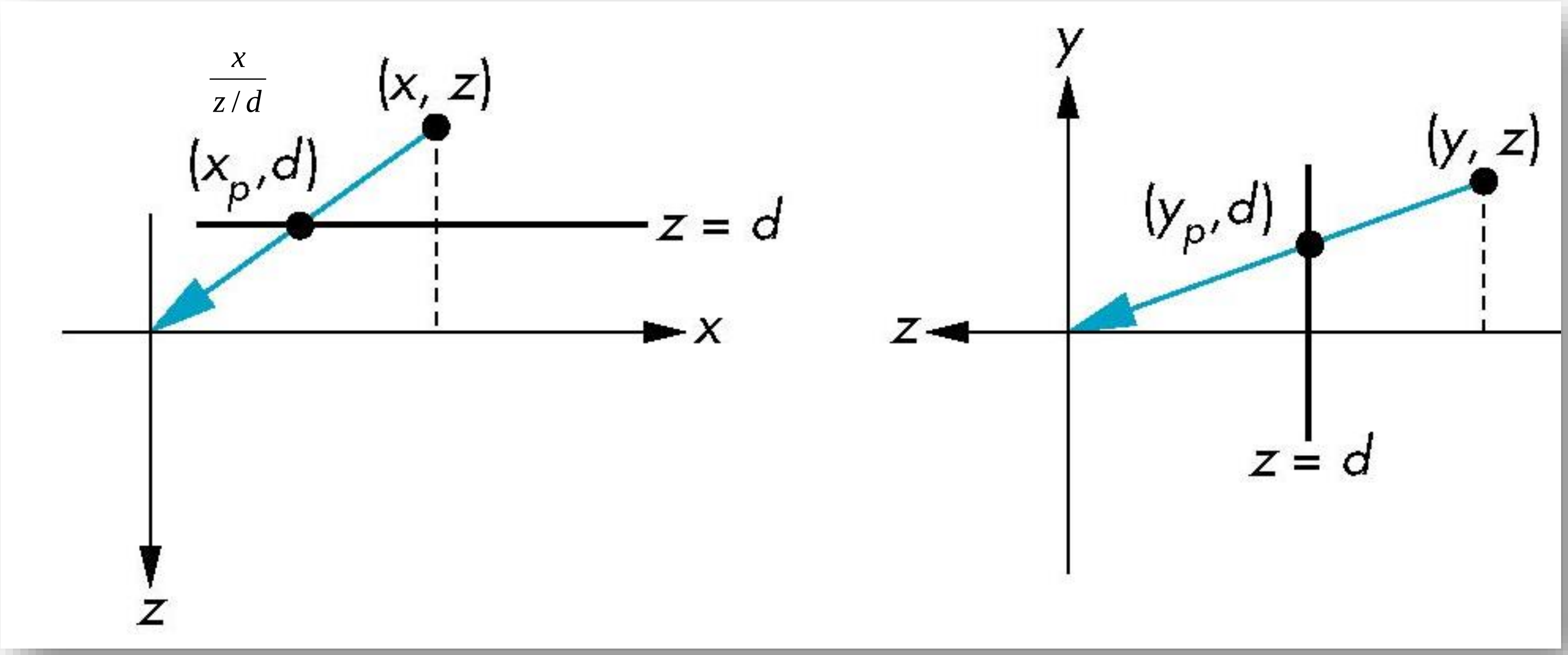
- Center of projection at the origin
- Projection plane $z = d$, $d < 0$



Perspective Equations

Top view

Side view



$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

Homogeneous Coordinate Form

consider **q** = **Mp** where

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

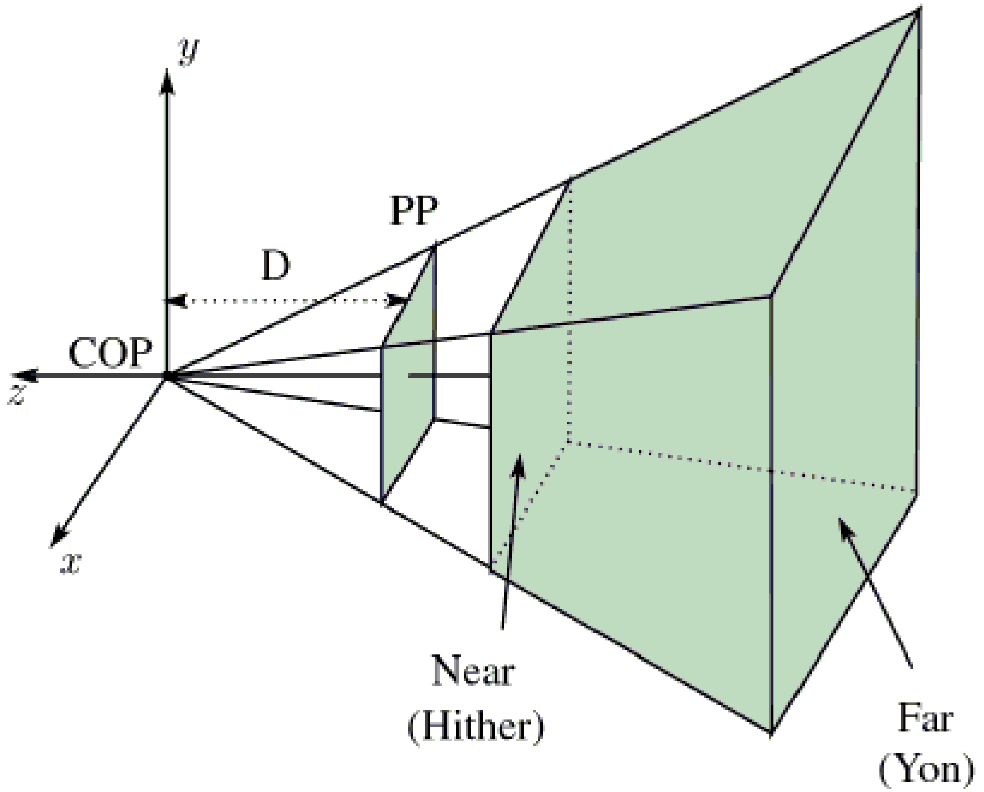
Perspective Division

- However $w \neq 1$, so we must divide by w to return from homogeneous coordinates
- This *perspective division* yields

$$x_\mathrm{p} = \frac{x}{z/d} \qquad y_\mathrm{p} = \frac{y}{z/d} \qquad z_\mathrm{p} = d$$

the desired perspective equations

Perspective Viewing Volume



$z = -near = Z_{near}$ $z = -far = Z_{far}$

Normalization

- Rather than derive a different projection matrix for each type of projection, we can *convert all projections to orthogonal projections* with the *default view volume.*

- This strategy allows us to use *standard transformations* in the pipeline and makes for *efficient clipping.*

Normalization

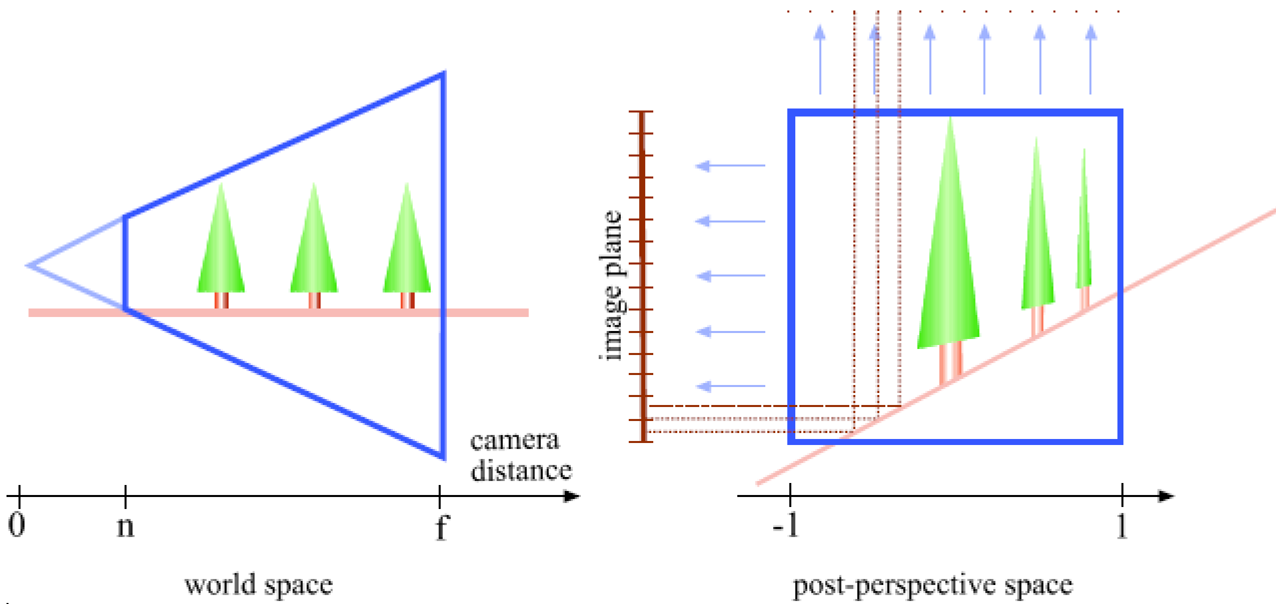


Fig. from: M. Stamminger, G. Drettakis, Perspective Shadow Maps, Proc. ACM SIGGRAPH 2002.

Perspective-Projection Trans.

$$M_{pers} = \begin{bmatrix} -z_{near} & 0 & 0 & 0 \\ 0 & -z_{near} & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

After perspective division, the point (x,y,z,1) goes to

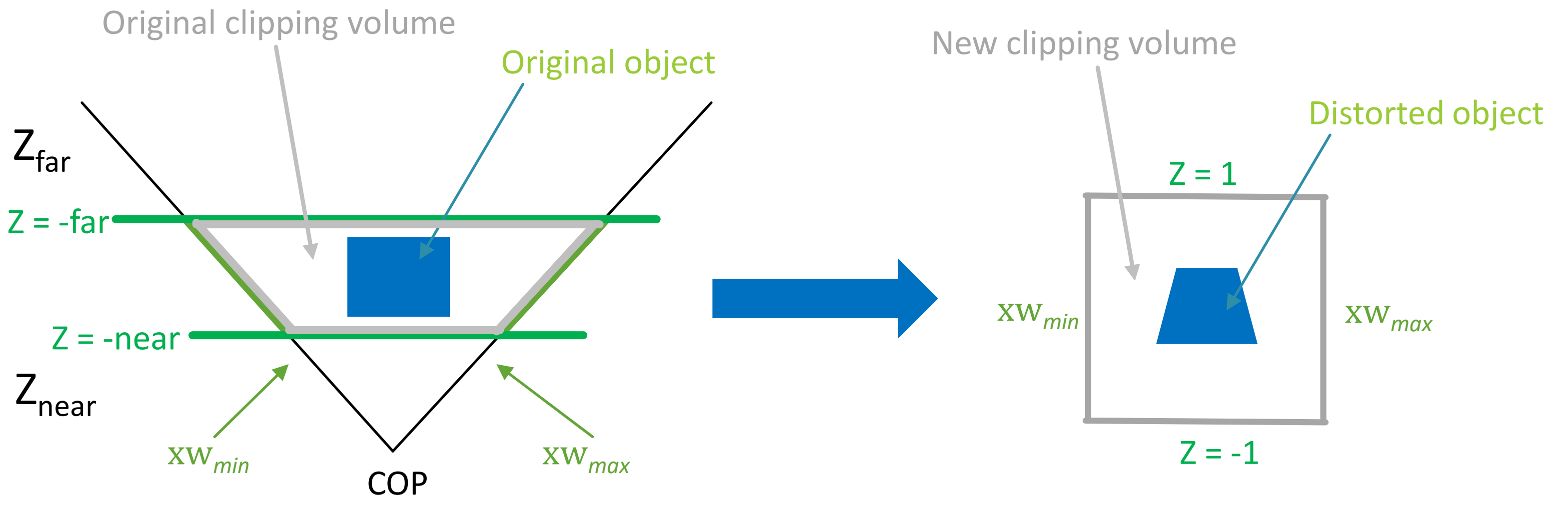
Find s_z, t_z To make $-1 \le z_p \le 1$

$$x_p = x\left(\frac{-z_{near}}{-z}\right)$$

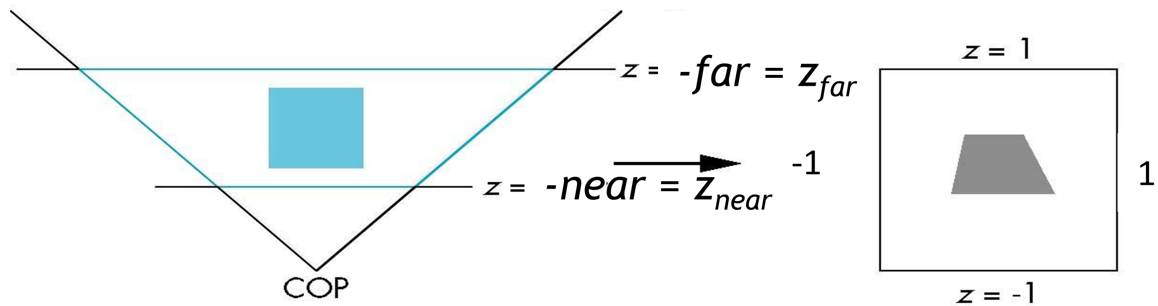
$$y_p = y\left(\frac{-z_{near}}{-z}\right)$$

$$z_p = \frac{s_z z + t_z}{-z} = -\left(s_z + \frac{t_z}{z}\right)$$

Perspective-Projection Trans.



Further Normalization



original object

$$M_{pers} = \begin{bmatrix} -z_{near} & 0 & 0 & 0 \\ 0 & -z_{near} & 0 & 0 \\ 0 & 0 & \dfrac{z_{near}+z_{far}}{z_{near}-z_{far}} & \dfrac{-2z_{near}z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Normalizing the x and y scales.

$$M_{normpers} = \begin{bmatrix} -z_{near}\dfrac{2}{xw_{\max}-xw_{\min}} & 0 & 0 & 0 \\ 0 & -z_{near}\dfrac{2}{yw_{\max}-yw_{\min}} & 0 & 0 \\ 0 & 0 & \dfrac{z_{near}+z_{far}}{z_{near}-z_{far}} & \dfrac{-2z_{near}z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Notes

- Normalization lets us clip against a simple cube regardless of type of projection
- Delay final “projection” until end
  - Important for *hidden-surface removal* to retain depth information as long as possible

Normalization and Hidden-Surface Removal

- if $z_1 > z_2$ in the original clipping volume then the for the transformed points $z_1' < z_2'$

- Hidden surface removal works if we first apply the normalization transformation

- However, the formula $z'' = -(s_z+t_z/z)$ implies that the distances are distorted by the normalization which can cause numerical problems especially if the near distance is small

Why do we do it this way?

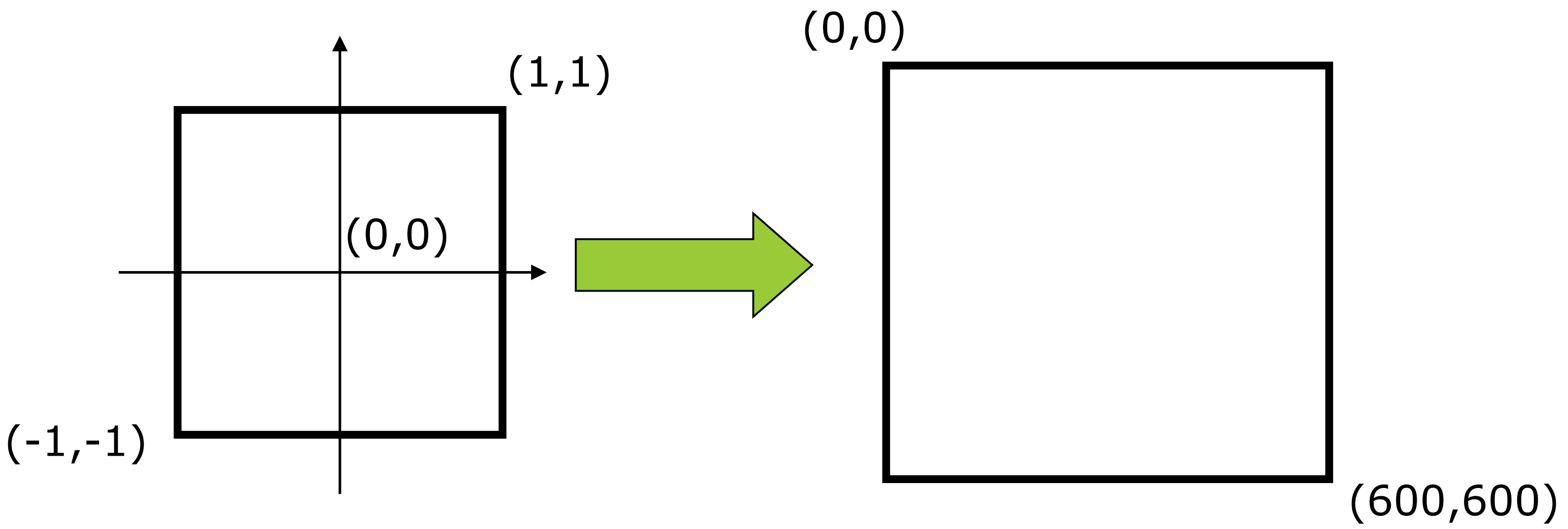
- Normalization allows for *a single pipeline* for both perspective and orthogonal viewing

- We stay in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed for hidden-surface removal and shading

- *Clipping is now "easier".*

Viewport Transformation

- From the working coordinate to the coordinate of display device.

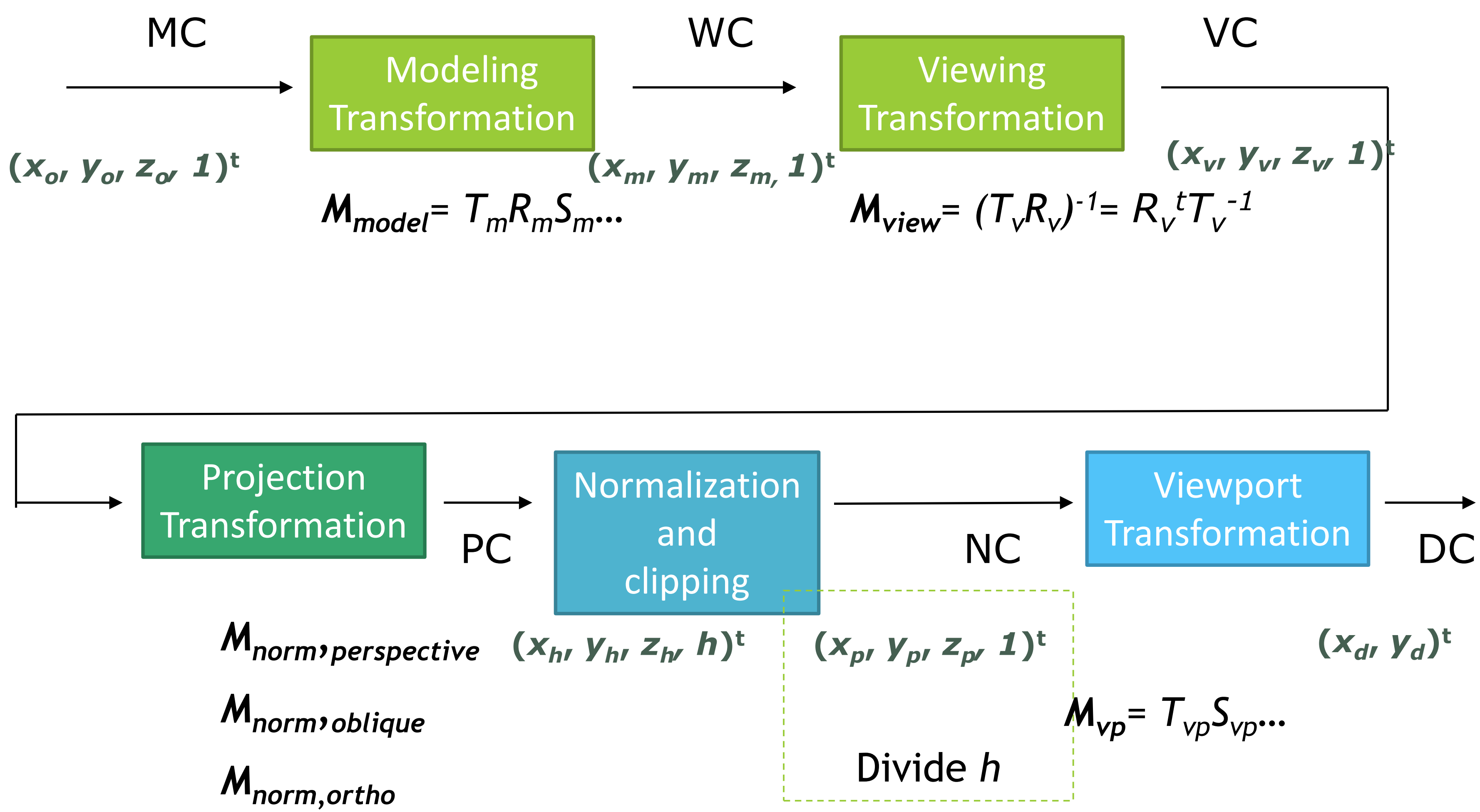


By 2D scaling and translation

Viewing in 3D

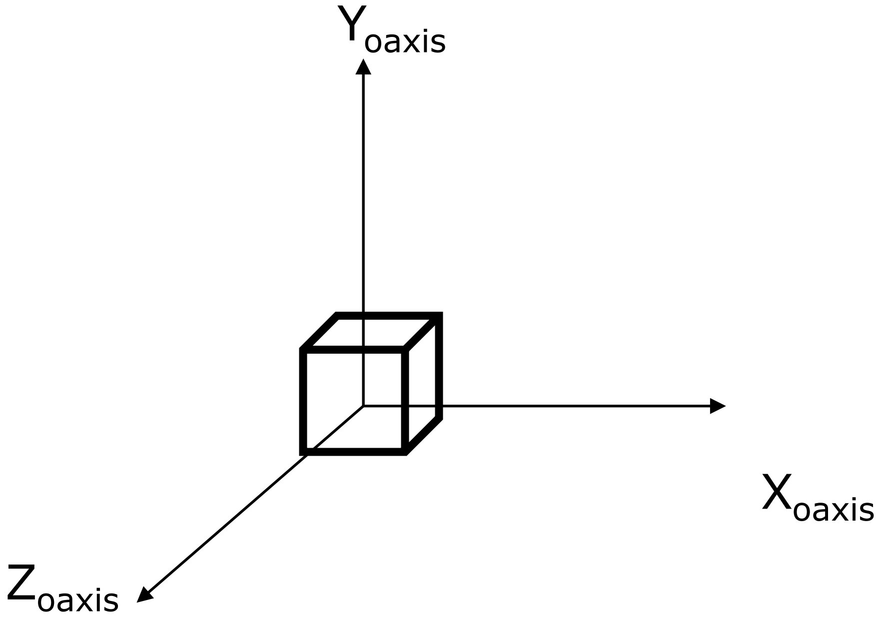
(Summary and Example)

Pipeline and Transformations



Loading an Object

$(x_o, y_o, z_o, 1)^t$

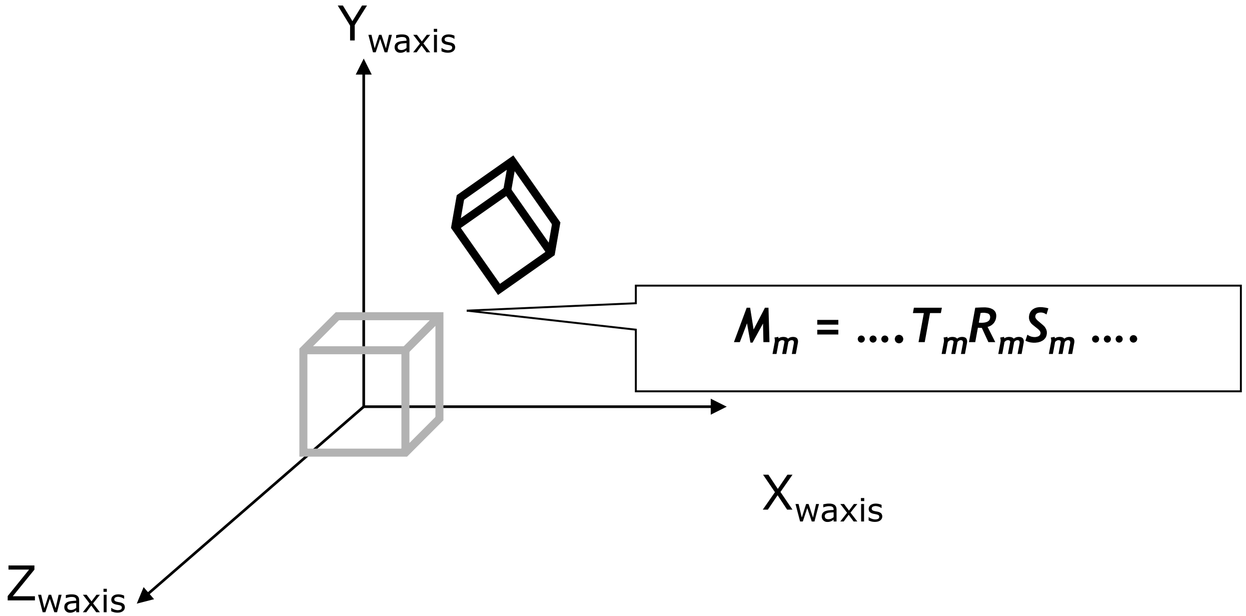


Object coord.

Modeling Transformation

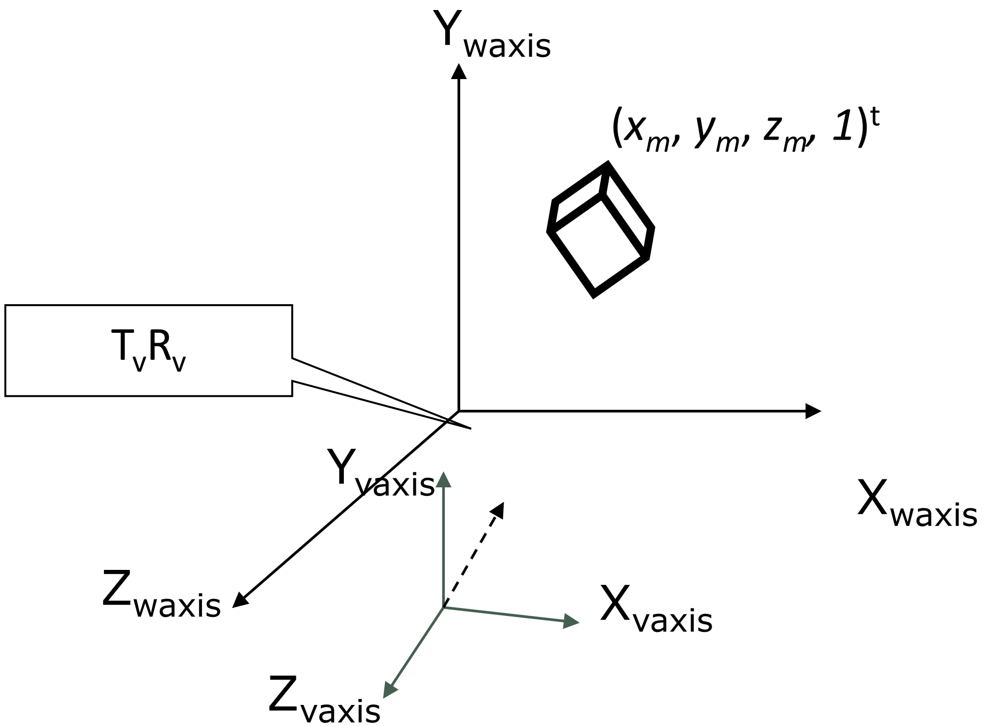
- $(x_m, y_m, z_m, 1)^t = M_m(x_o, y_o, z_o, 1)^t$

  where $M_m = \ldots.T_m R_m S_m \ldots.$



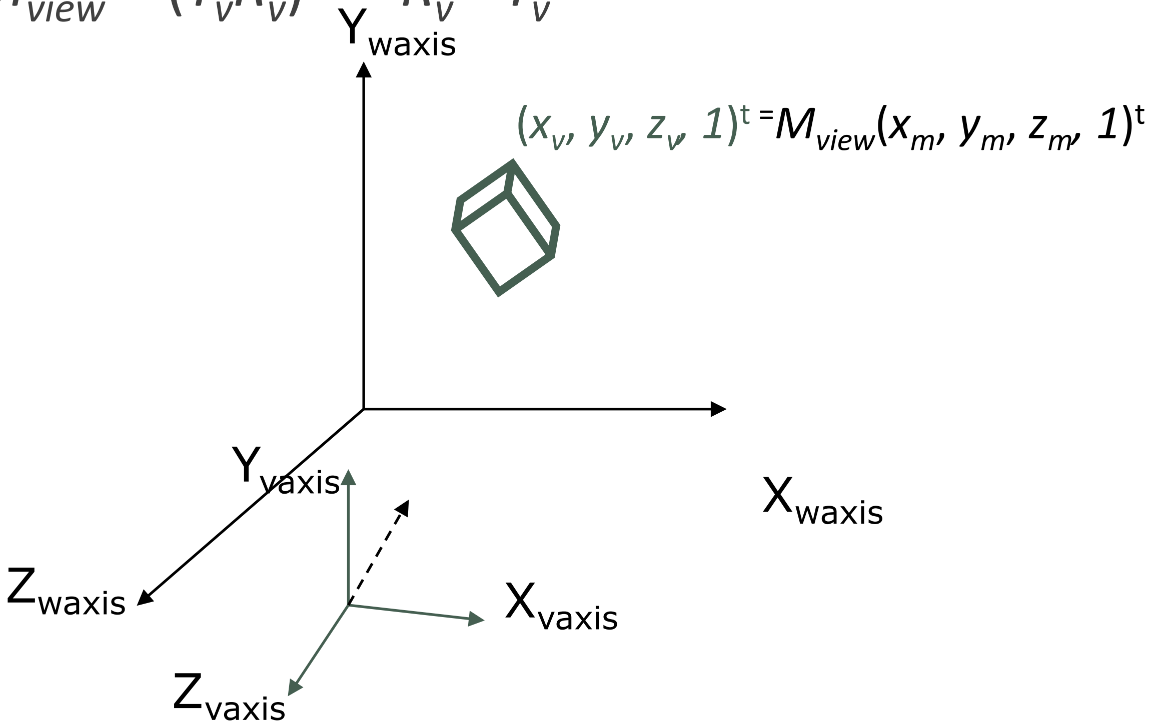
Put a Virtual Camera

- Move a camera from the origin (by $T_v R_v$)

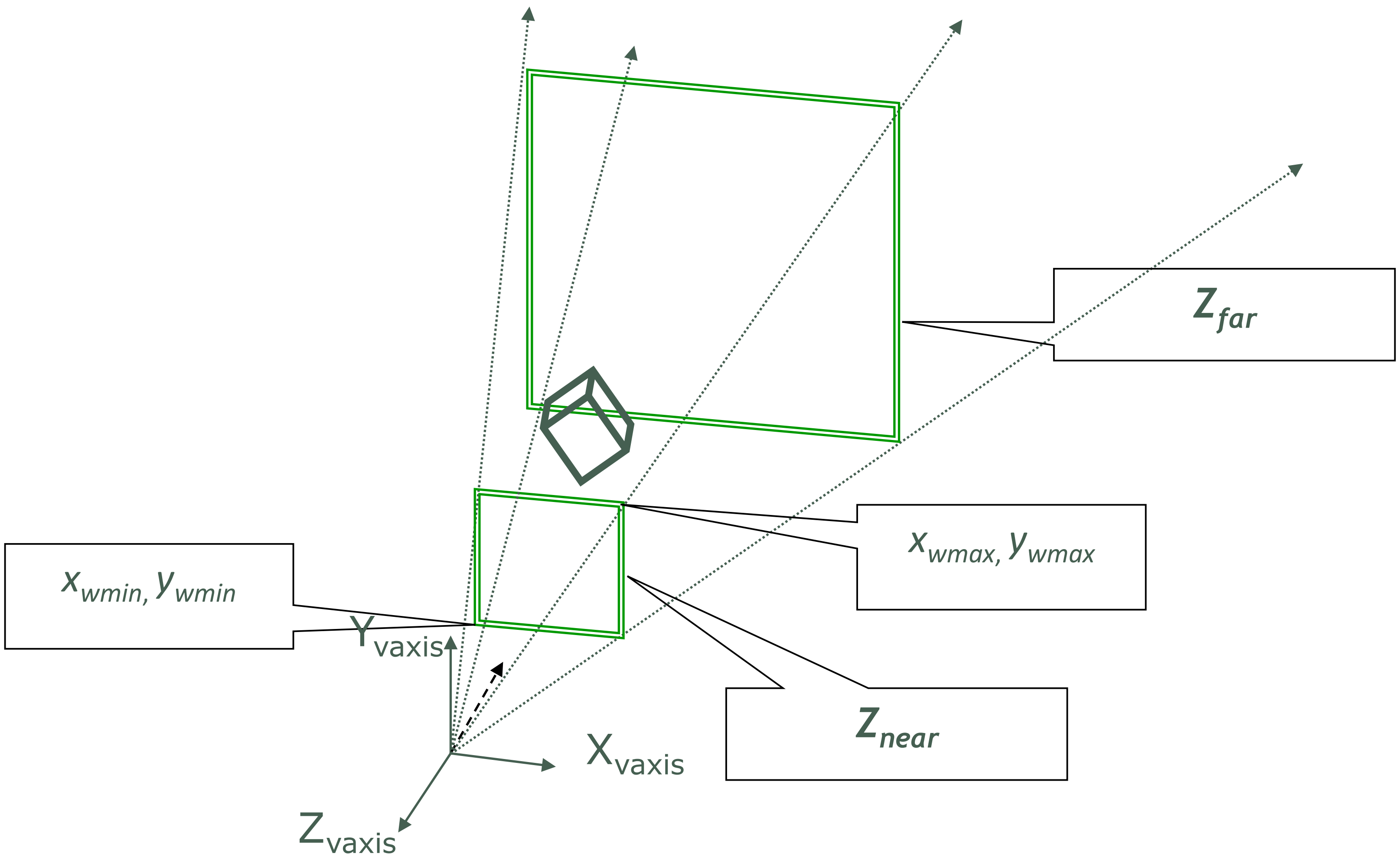


Virtual Camera's Coordinate

- Change the object's coordinate
- $(x_v, y_v, z_v, 1)^t = M_{view}\,(x_m, y_m, z_m, 1)^t$
- $M_{view} = (T_v R_v)^{-1} = R_v^{-1}\,T_v^{-1}$



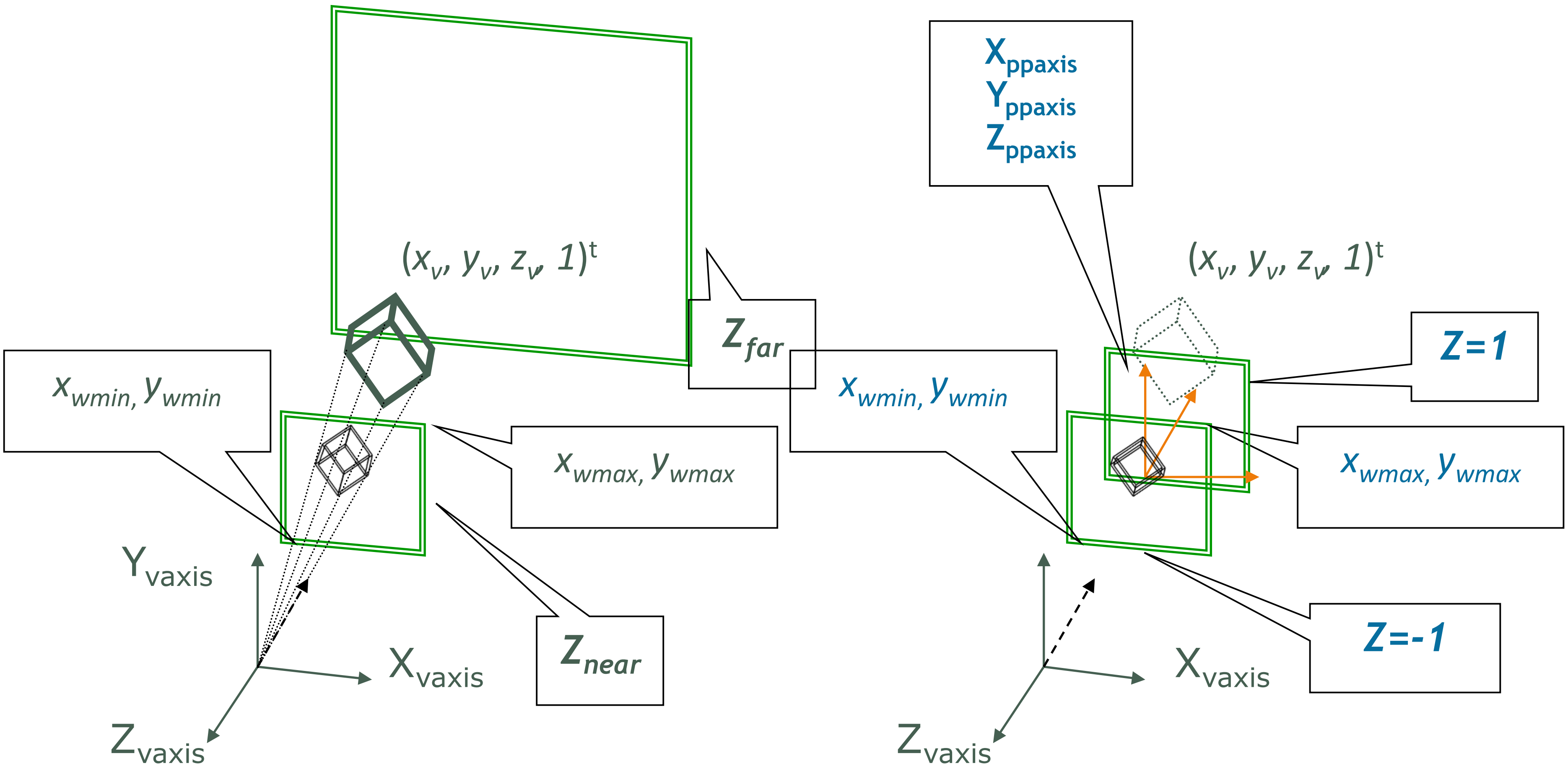
Virtual Camera's Coordinate



Perspective Proj.
(for derivation)

$$M_{pers} = \begin{bmatrix} -z_{near} & 0 & 0 & 0 \\ 0 & -z_{near} & 0 & 0 \\ 0 & 0 & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} & \frac{-2z_{near}z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

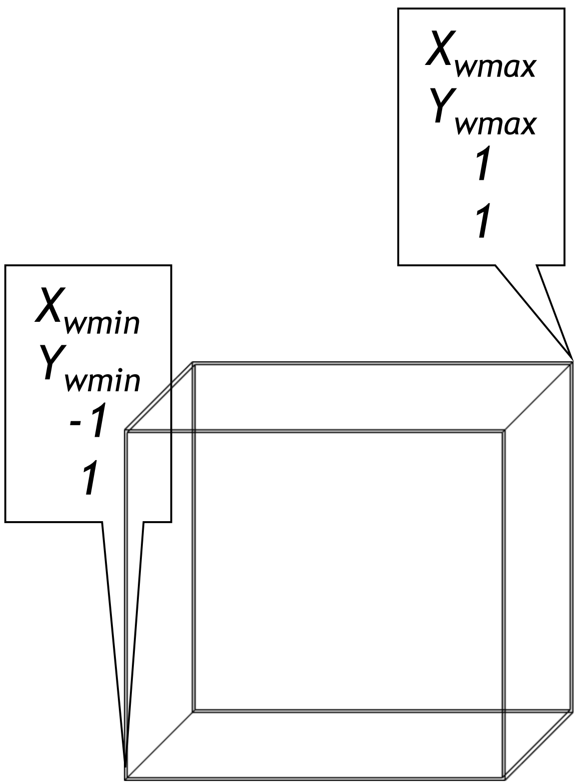
This matrix is usually combined with the normalization matrix.

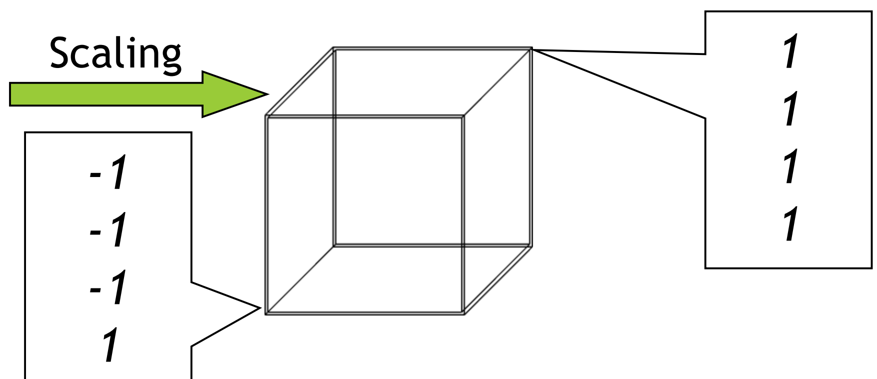


Projection + Normalization
(for derivation)

$$M_{normpers} = \begin{bmatrix} -z_{near}\dfrac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & 0 \\ 0 & -z_{near}\dfrac{2}{yw_{\max} - yw_{\min}} & 0 & 0 \\ 0 & 0 & \dfrac{z_{near} + z_{far}}{z_{near} - z_{far}} & \dfrac{-2z_{near}z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} \dfrac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & 0 \\ 0 & \dfrac{2}{yw_{\max} - yw_{\min}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} M_{pers}$$

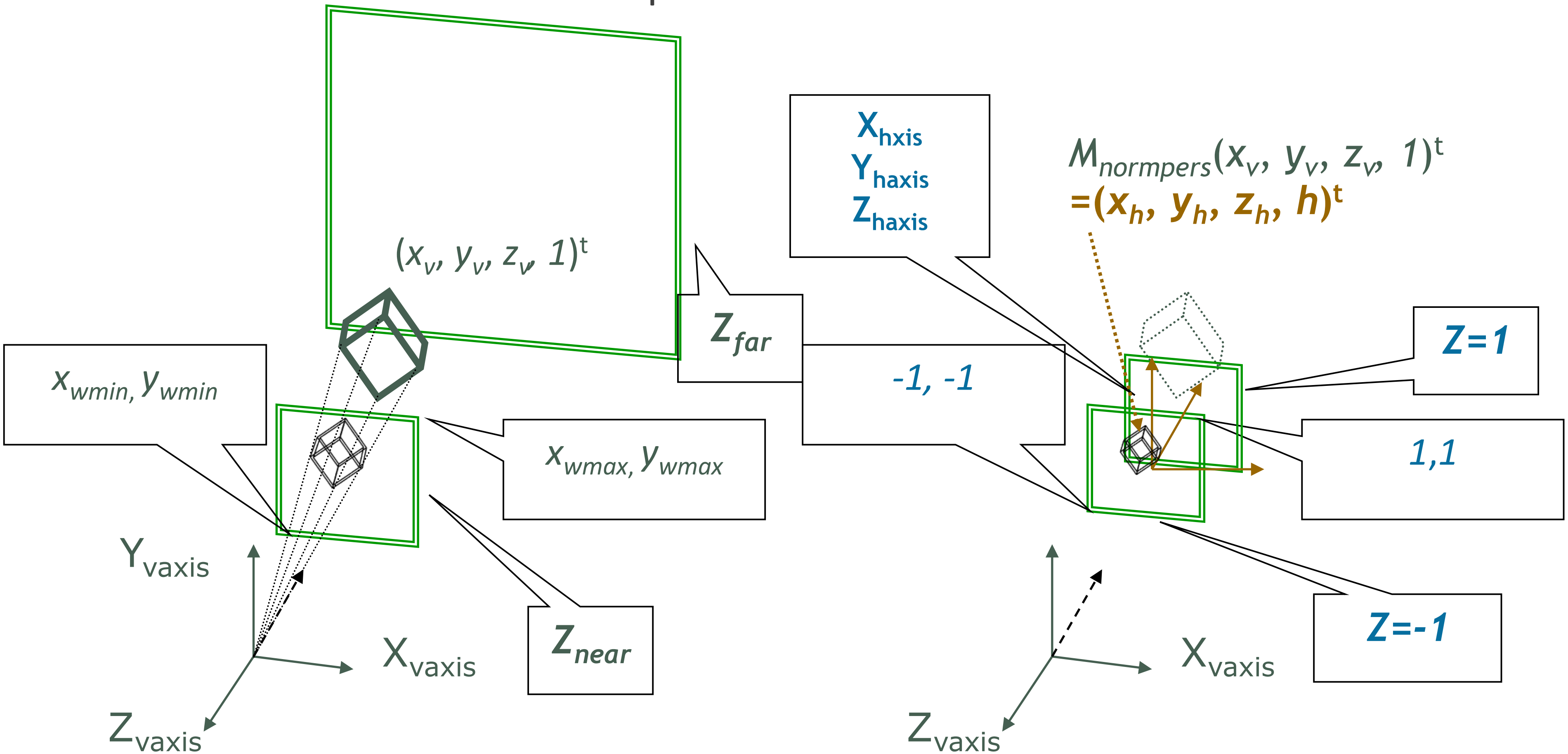




Proj.+Norm.

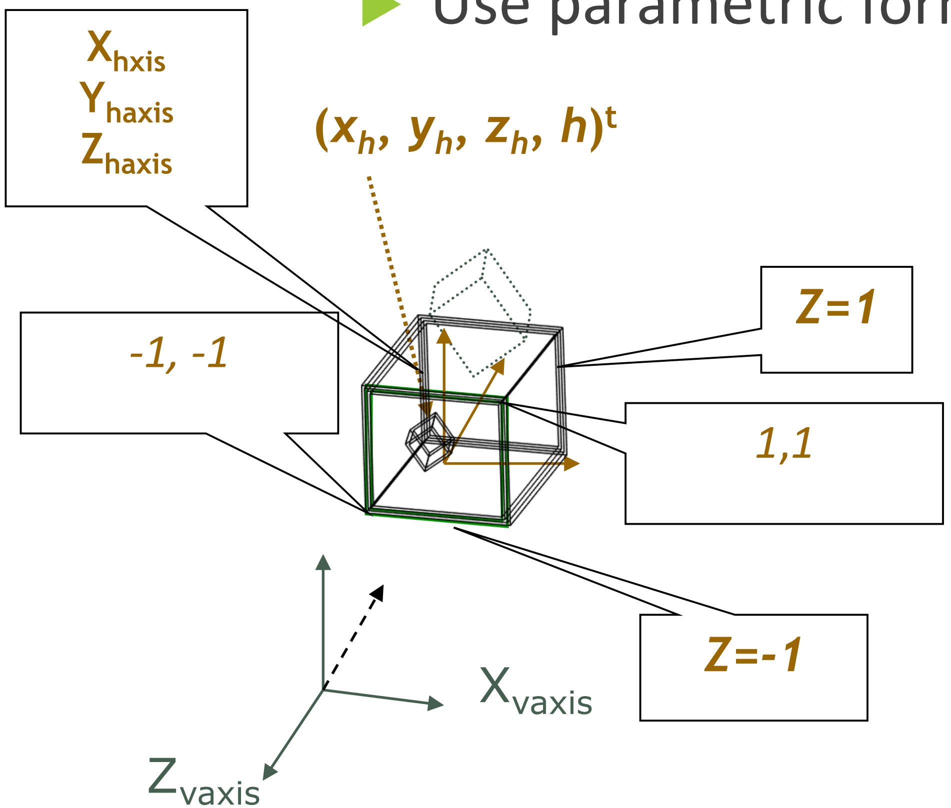
$$M_{normpers} = \begin{bmatrix} -z_{near}\dfrac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & 0 \\ 0 & -z_{near}\dfrac{2}{yw_{\max} - yw_{\min}} & 0 & 0 \\ 0 & 0 & \dfrac{z_{near} + z_{far}}{z_{near} - z_{far}} & \dfrac{-2z_{near}z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- $(x_h, y_h, z_h, h)^t = M_{normpers}(x_v, y_v, z_v, 1)^t$
- Don't divide h at this step.



Clipping

- Perform clipping with $(x_h, y_h, z_h, h)^t$
- Avoid unnecessary division $-h \le x_h \le h,\ -h \le y_h \le h,\ -h \le z_h \le h$
- Use parametric forms for intersection



$$x_h = x_{ha} + (x_{hb} - x_{ha})u$$

$$y_h = y_{ha} + (y_{hb} - y_{ha})u$$

$$z_h = z_{ha} + (z_{hb} - z_{ha})u$$

$$h = h_a + (h_b - h_a)u$$

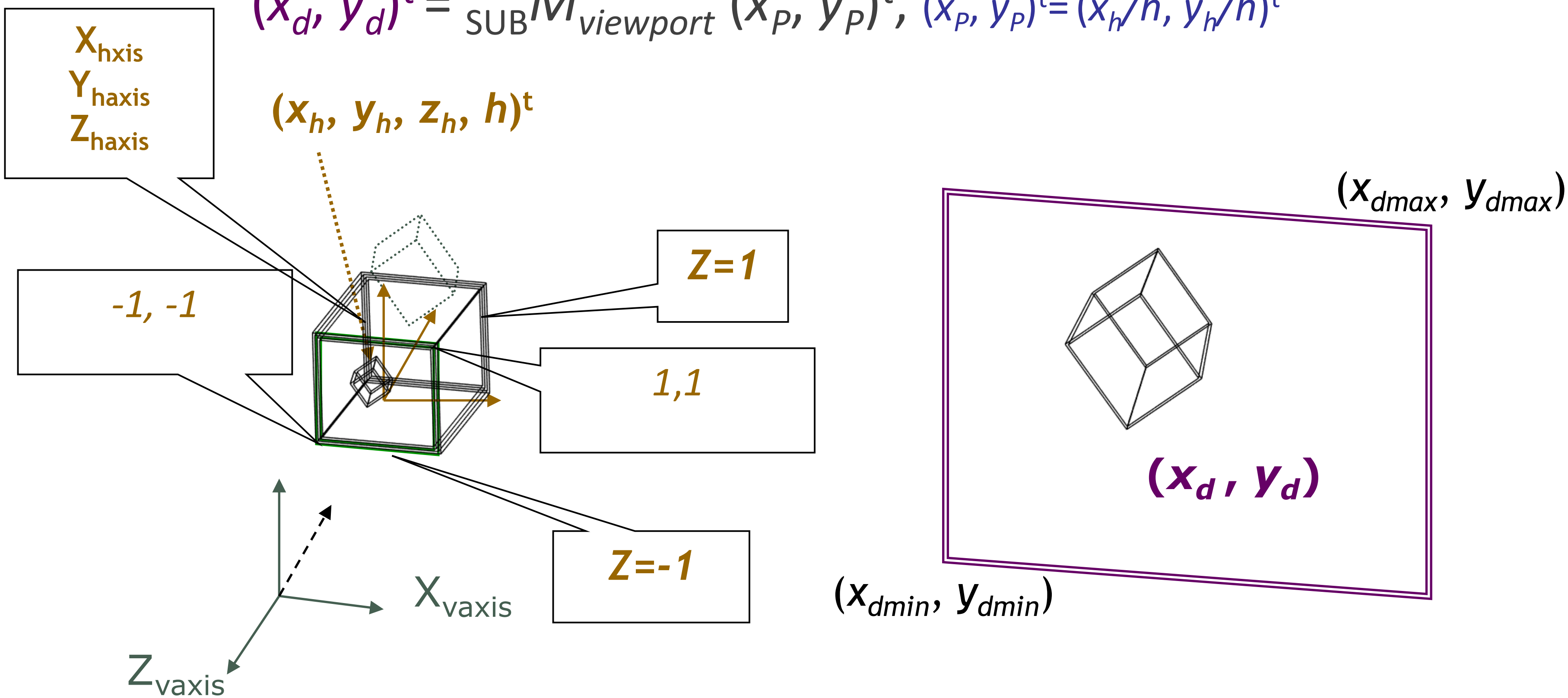
Viewport Transformation

$$M_{viewport} = \begin{bmatrix} \frac{x_{d\max} - x_{d\min}}{2} & 0 & 0 & \frac{x_{d\max} + x_{d\min}}{2} \\ 0 & \frac{y_{d\max} - y_{d\min}}{2} & 0 & \frac{y_{d\max} + y_{d\min}}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$(x_d, y_d, z_d, 1)^t = M_{viewport}\,(x_h, y_h, z_h, h)^t$$

OR

$$(x_d, y_d)^t = {}_{SUB}M_{viewport}\,(x_P, y_P)^t, \quad (x_P, y_P)^t = (x_h/h, y_h/h)^t$$



Rasterization

- Line drawing or polygon filling with

  $(x_d, y_d, z_d, 1)^t$ or $(x_d, y_d)^t$ and z_h