Introduction to Computer Graphics

3. Viewing in 3D

I-Chen Lin

National 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

Outline

- Classical views
- Computer viewing
- Projection matrices

Classical Viewing

- Viewing requires three basic elements
  - One or more objects
  - A viewer with a projection surface
  - Projectors that go from the object(s) to the projection surface
- Each object is assumed to constructed from flat *principal faces*
  - Buildings, polyhedra, manufactured objects

多面體

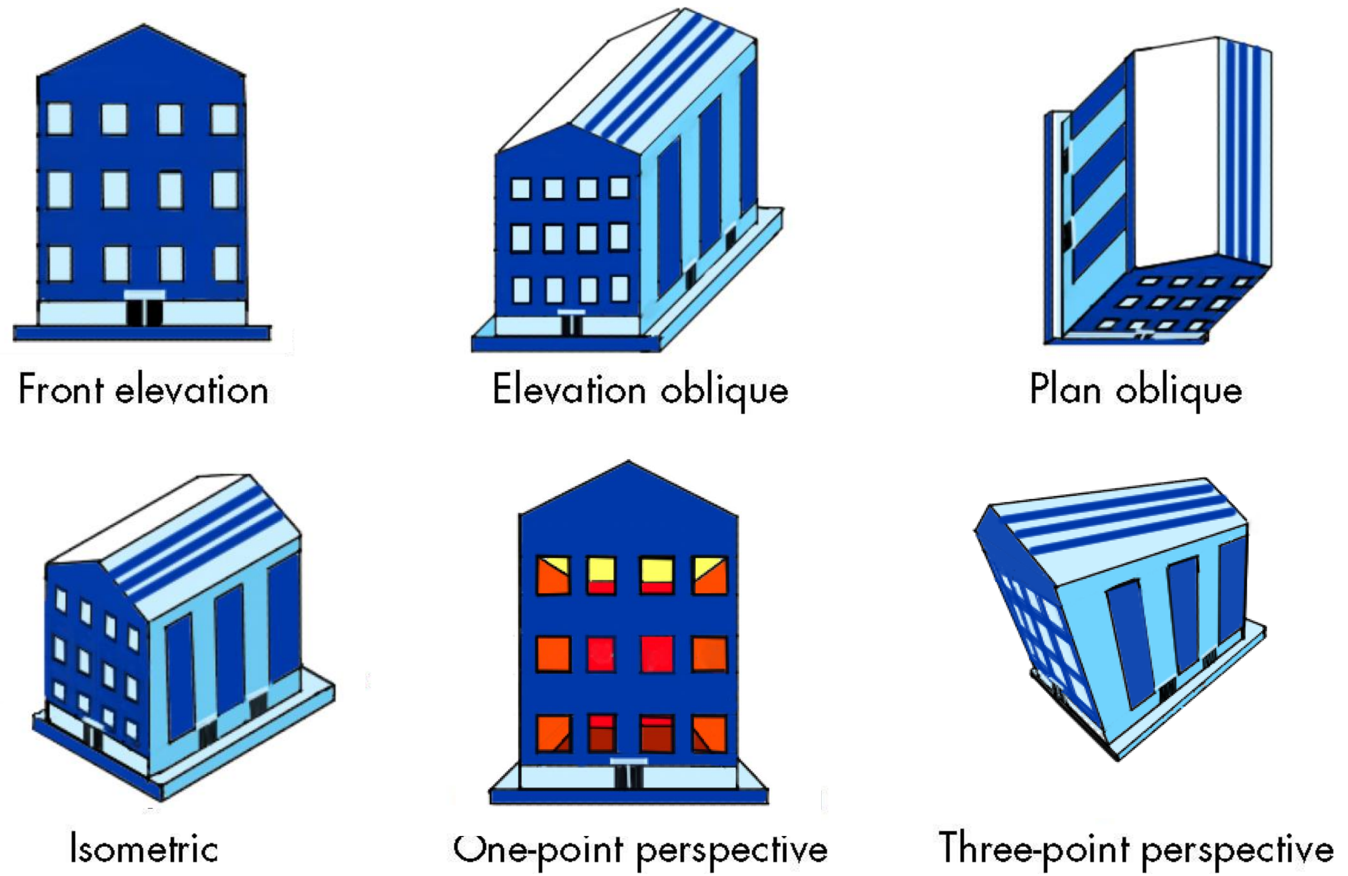
平面的

Planar Geometric Projections

- Standard projections project onto a plane.
- Projectors are lines that either
  - converge at a center of projection
  - are parallel
- Such projections preserve lines
  - but not necessarily angles
- When do we need non-planar projections?

Classical Projections

正投影



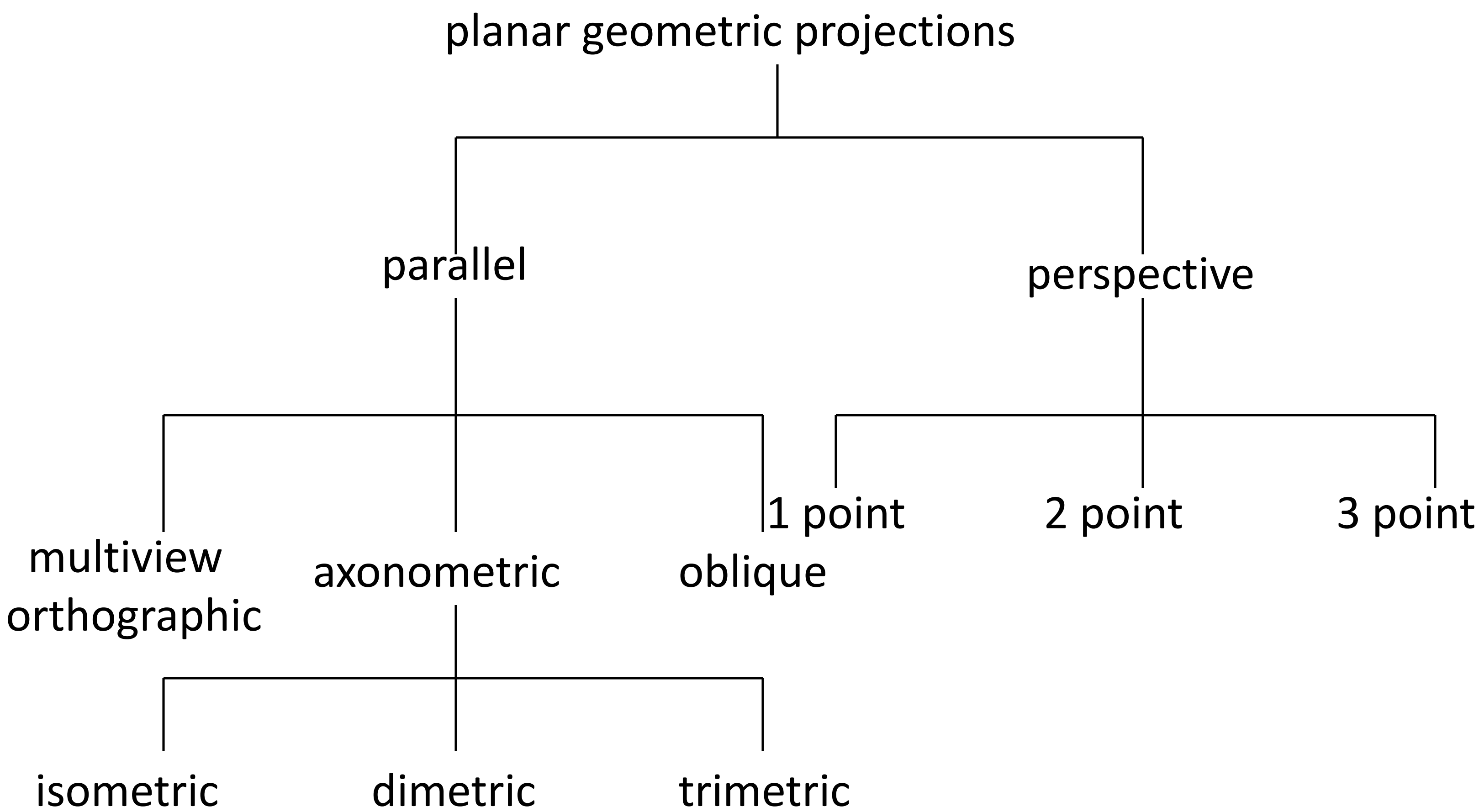
斜投影

近大遠小　　　　　　方便計算長度

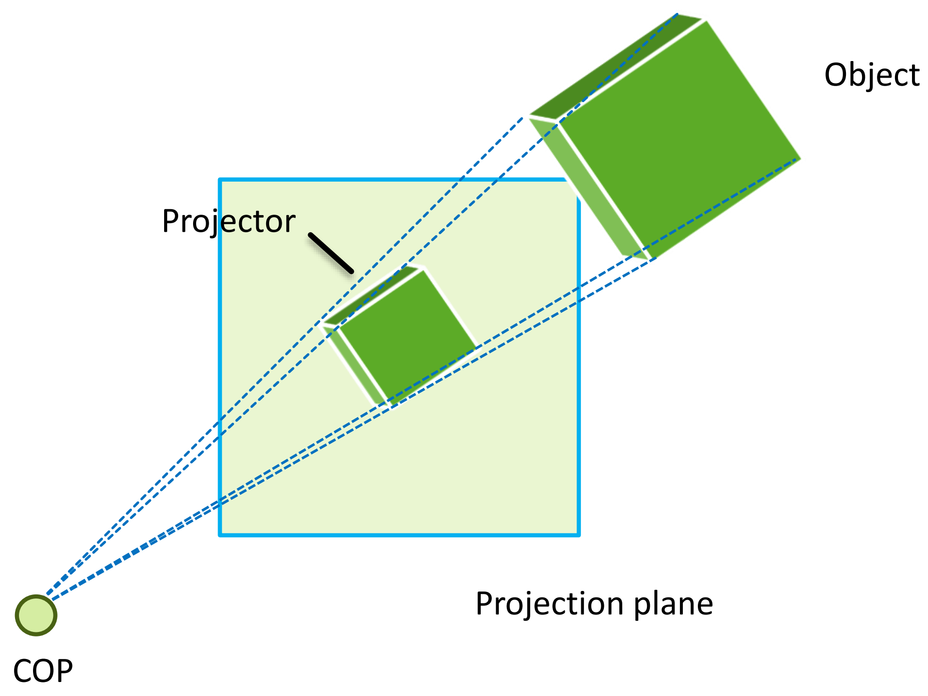
Perspective vs. Parallel

- Classical viewing developed different techniques for drawing each type of projection
- Mathematically parallel viewing is the limit of perspective viewing
- Computer graphics treats all projections the same and implements them with a single pipeline

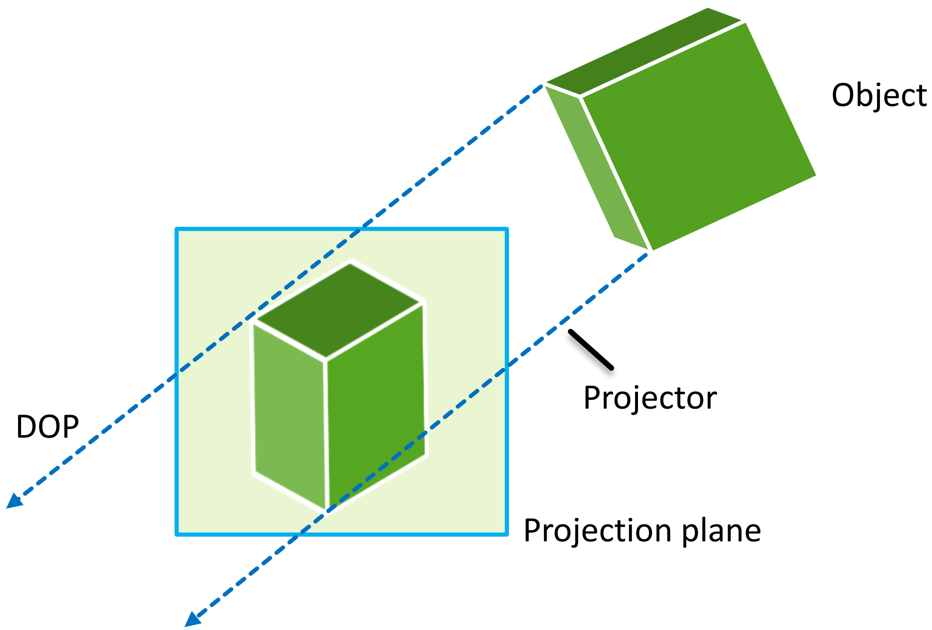
Taxonomy of Planar Geometric Projections



Perspective Projection

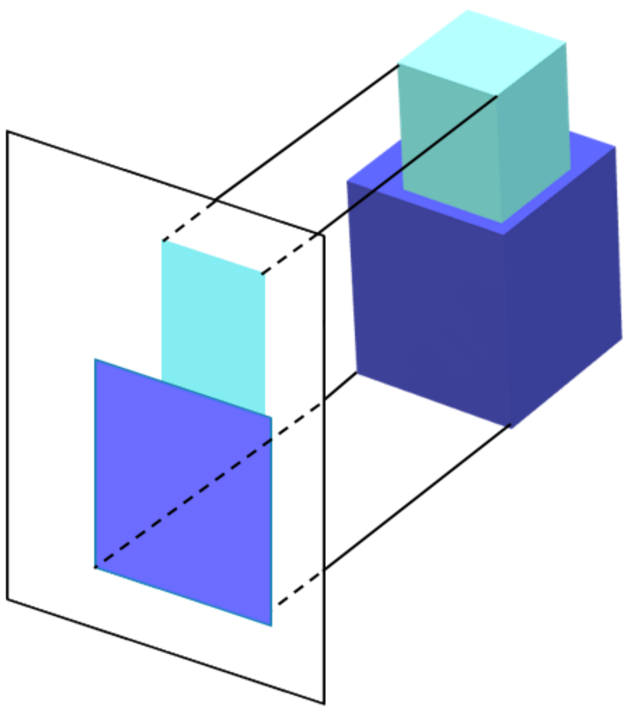


Parallel Projection



Orthographic Projection 正交投影法(直接xy投影)

- Projectors are orthogonal to projection surface

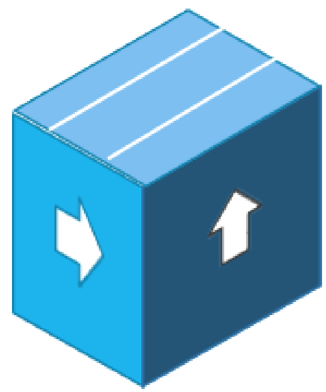


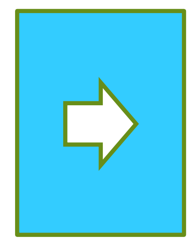
Multi-view Orthographic Projection

- Projection plane parallel to principal faces
- Usually form front, top, side views

等距

isometric (not multiview orthographic view)

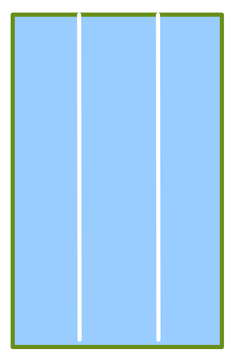


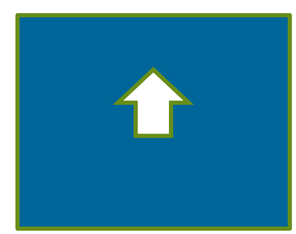


Front

in CAD and architecture, we often display three multiviews plus isometric

Top





Side

Advantages and Disadvantages

- Preserves both distances and angles
  - Shapes preserved
  - Can be used for measurements
    - Building plans
    - Manuals

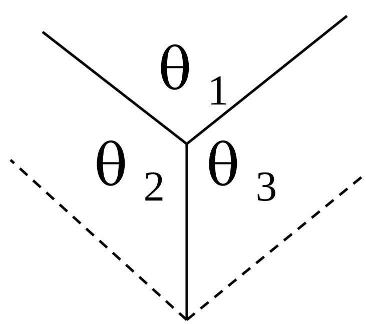
- Cannot see what object really looks like because many surfaces hidden from view
  - Often we add the isometric

軸測圖

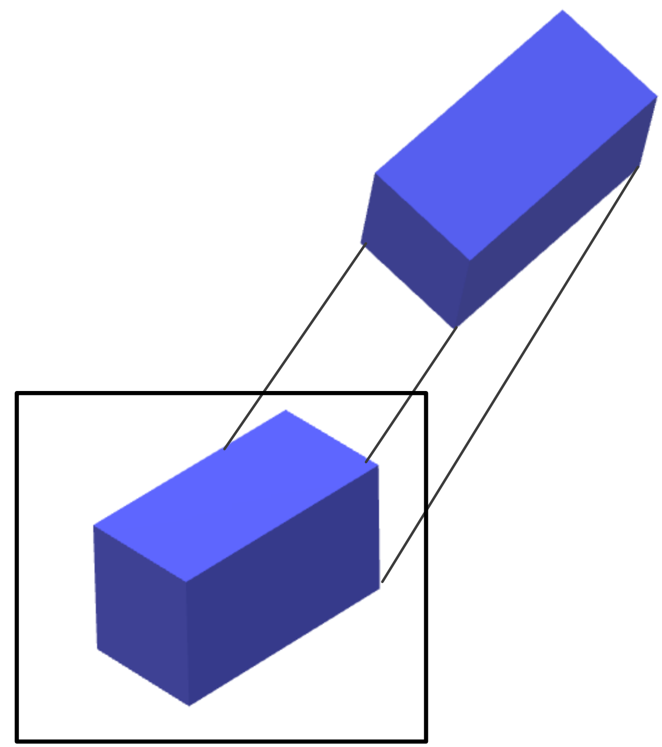
Axonometric Projections

- Allow projection plane to move relative to an object

classify by how many angles of a corner of a projected cube are the same

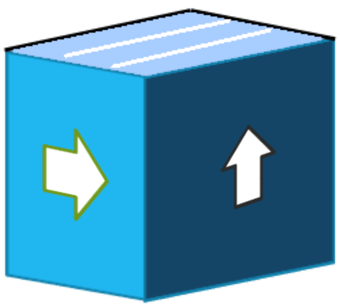


none: trimetric
two: dimetric
three: isometric
做出等角

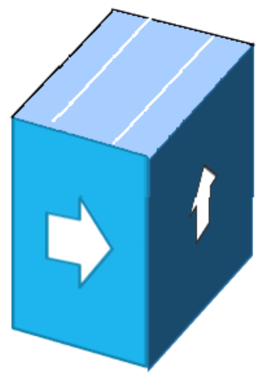


Projection Plane

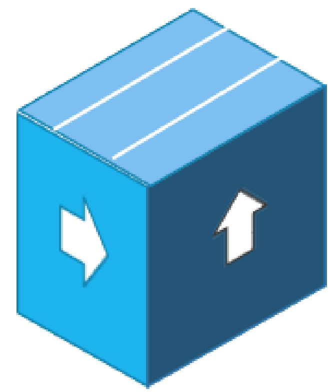
Types of Axonometric Projections



Dimetric



Trimetric



Isometric

Advantages and Disadvantages

- Lines are scaled (*foreshortened*) but <u>can find scaling factors</u>

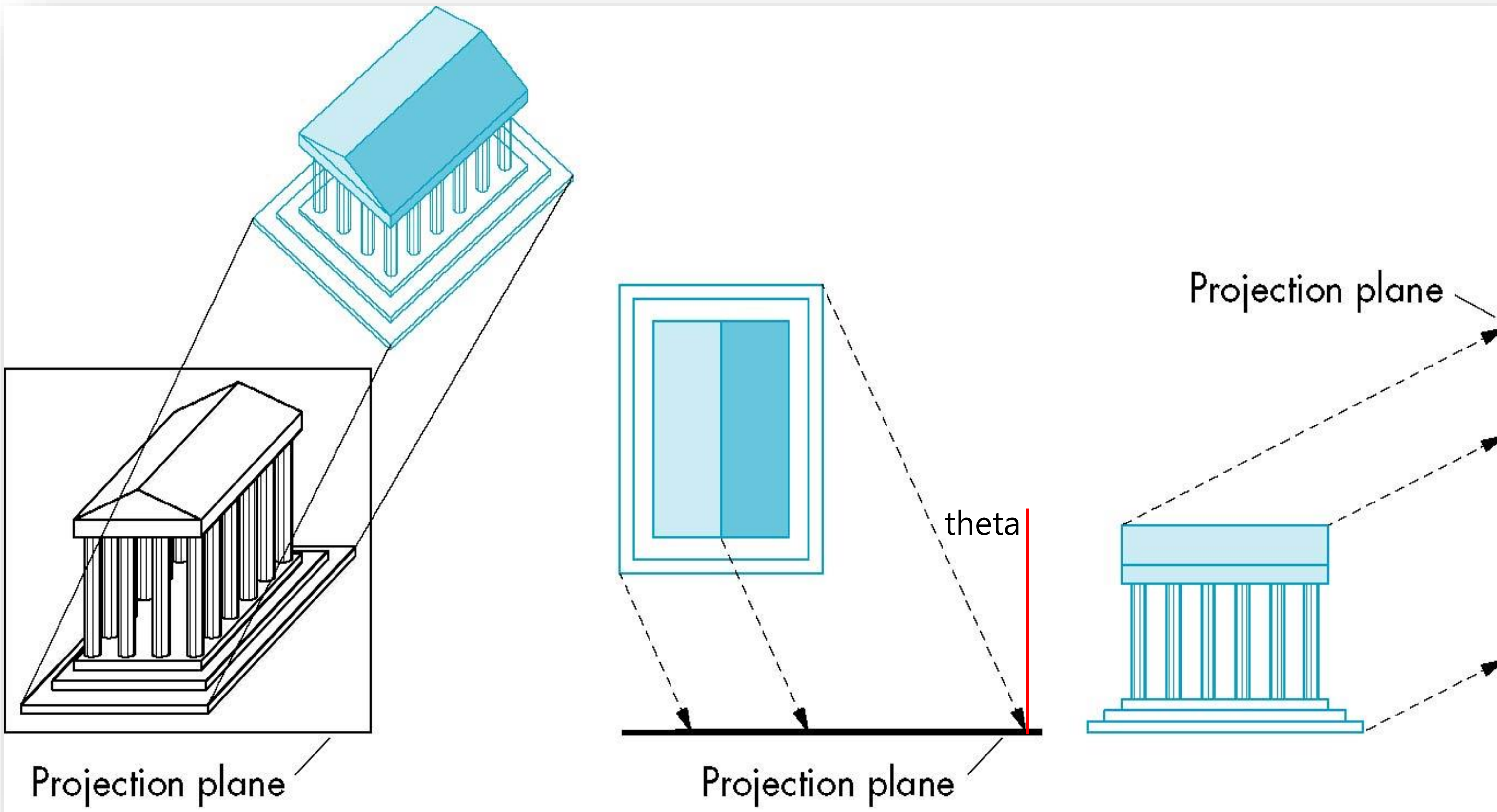
- <u>Lines preserved but angles are not</u>
  - Projection of a circle in a plane not parallel to the projection plane is an ellipse

- Does not look real because far objects are scaled the same as near objects

- Used in CAD applications

Oblique Projection 斜投影

- Arbitrary relationship between projectors and projection plane

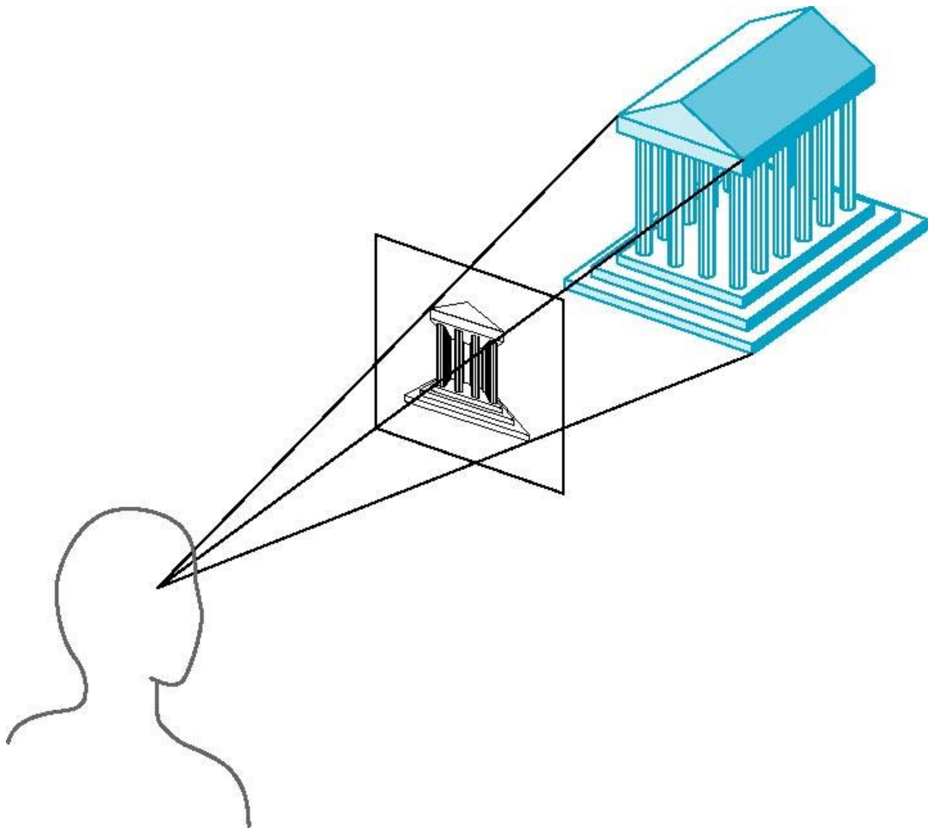


Advantages and Disadvantages

- Can pick the angles to emphasize a particular face
  - Architecture: plan oblique, elevation oblique 不是real world會出現的
- Angles in faces parallel to the projection plane are preserved while we can still see “around” side

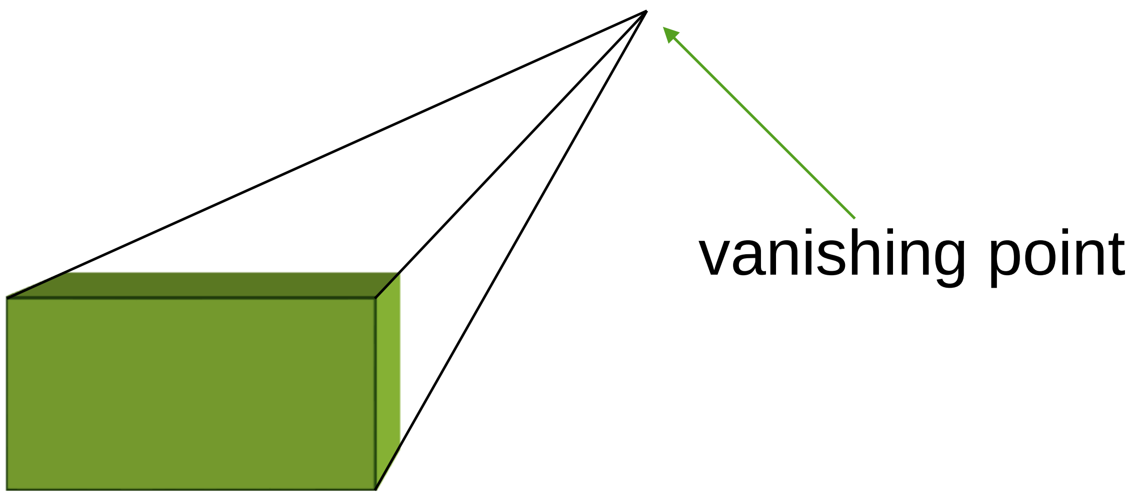
Perspective Projection

- Projectors' coverage at the center of projection



Vanishing Points

- Parallel lines (not parallel to the projection plan):
  - converge at a single point in the projection (the *vanishing point*)
- Drawing simple perspectives by hand uses these vanishing point(s)



Three-Point Perspective

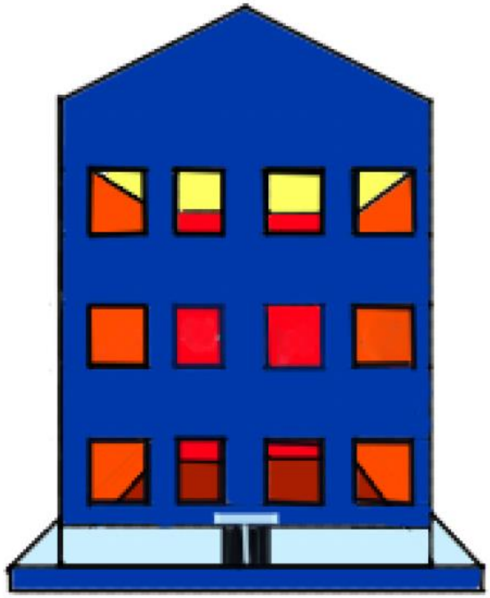
- No principal face parallel to projection plane
- Three vanishing points for a cube

Two-Point Perspective

- On principal direction parallel to projection plane
- Two vanishing points for a cube

One-Point Perspective

- One principal face parallel to projection plane
- One vanishing point for a cube



Advantages and Disadvantages

- Diminution:
  - Objects further from viewer are projected smaller (Looks realistic)
- Nonuniform foreshortening:
  - Equal distances along a line are not projected into equal distances
- Angles preserved only in planes parallel to the projection plane
- More difficult to construct by hand than parallel projections

Computer Viewing

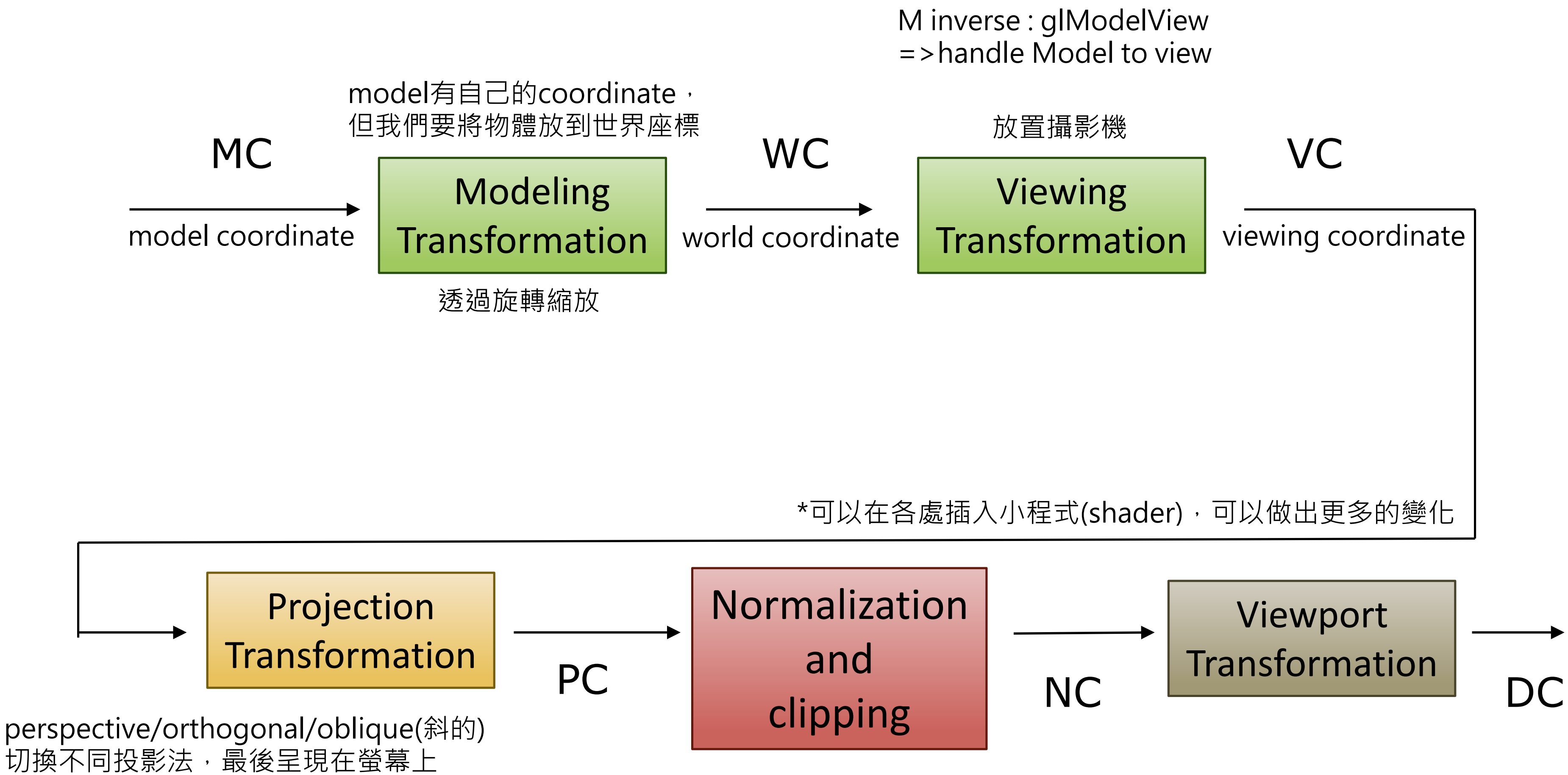
Computer Viewing

- 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* 僅保存視野內的模型

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

Pipeline View



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

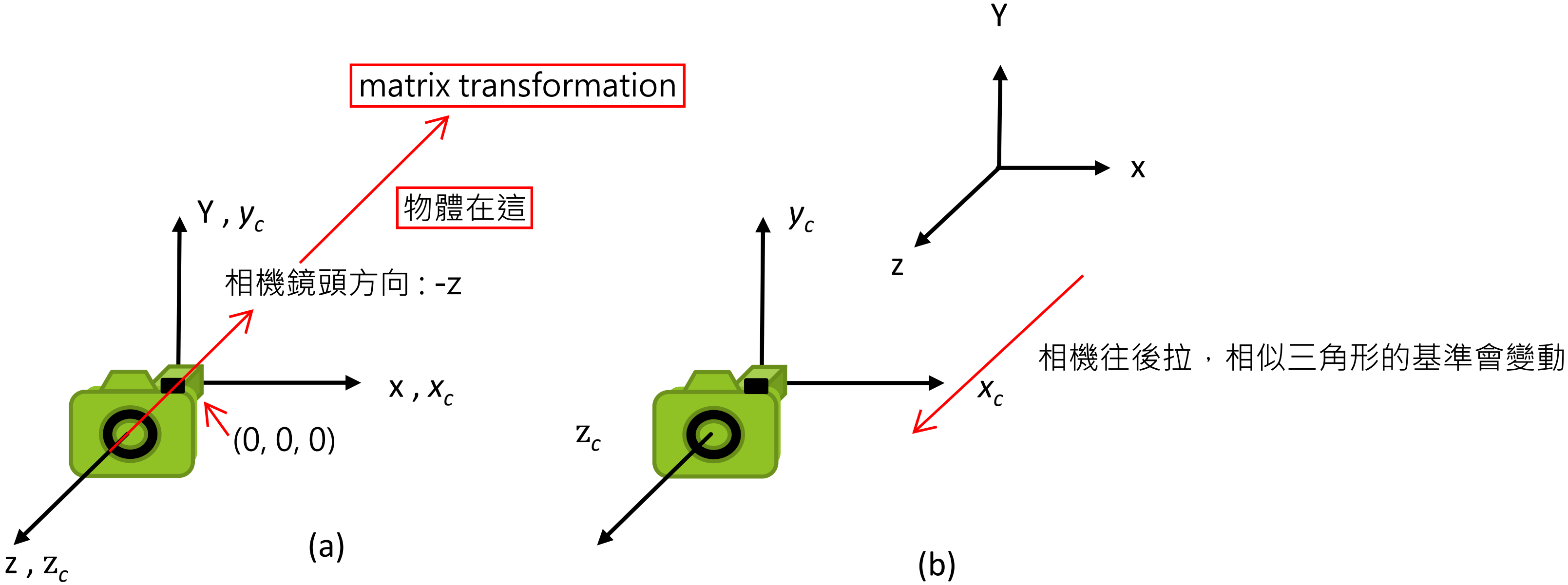
- Both of these views are equivalent and are determined by the model-view matrix
  - Want a translation (`glTranslatef(0.0,0.0,-d);`)
  - `d > 0`

Moving Camera back from Origin

default frames

frames after translation by –d

d > 0



相機也屬於物體，也可以有rotation和translation
相機與世界的動作方向相反(就像自己看鏡子裡的自己那樣)
EX : 若相機 : Ry(45度)Td(3m) => 則物體 : Td(-3m)Ry(-45度)

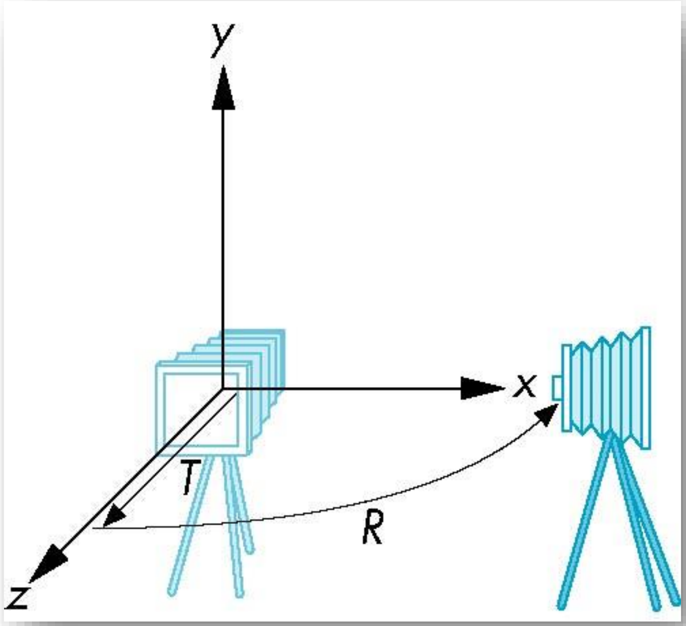
Td(0, 0, 3)

$\boxed{\text{!=Ry(-45度)Td(-3m)}}$

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
  - Model-view matrix C = T’R’

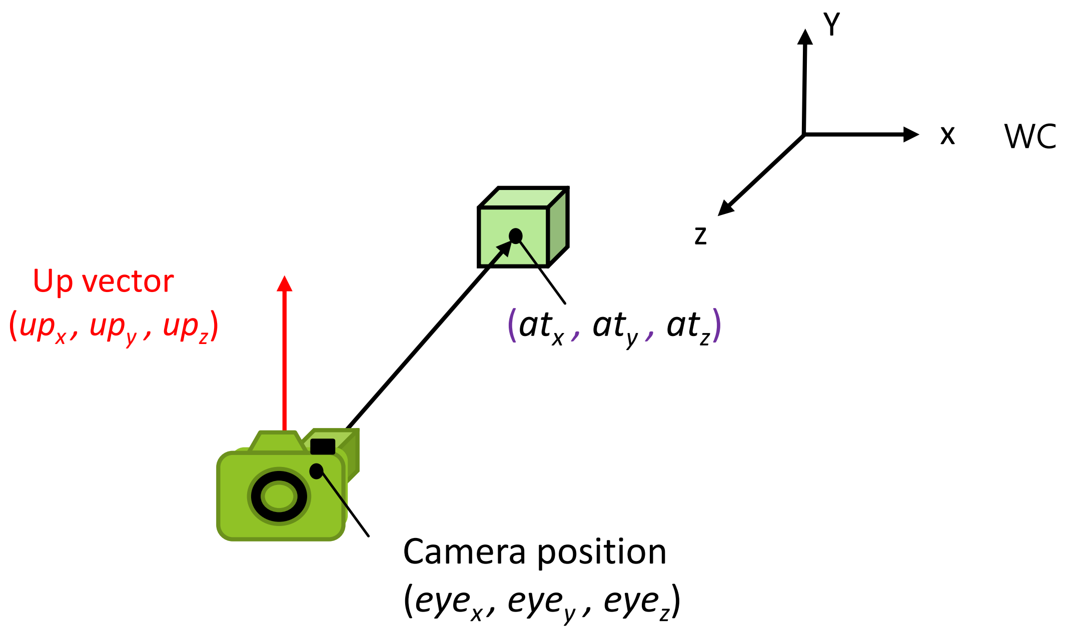


gluLookAt

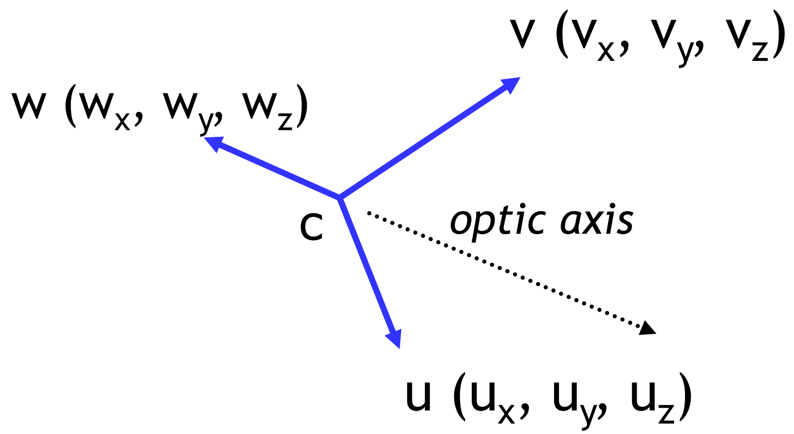
- glLookAt(eye$_x$, eye$_y$, eye$_z$, at$_x$, at$_y$, at$_z$, up$_x$, up$_y$, up$_z$)

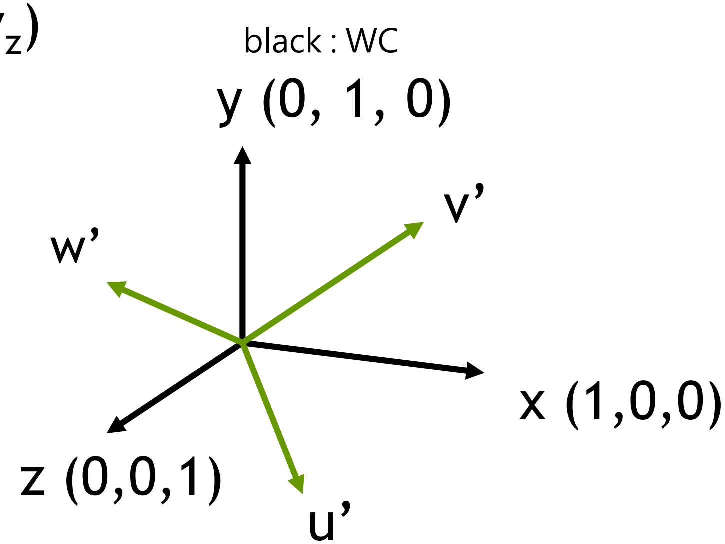
camera的z軸　camera的y軸

設定相機位置(相對WC)　相機看的方向



By Coordinate Transformations glLookAt





從自己的座標軸轉回世界座標軸

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

從世界座標軸轉回自己的座標軸

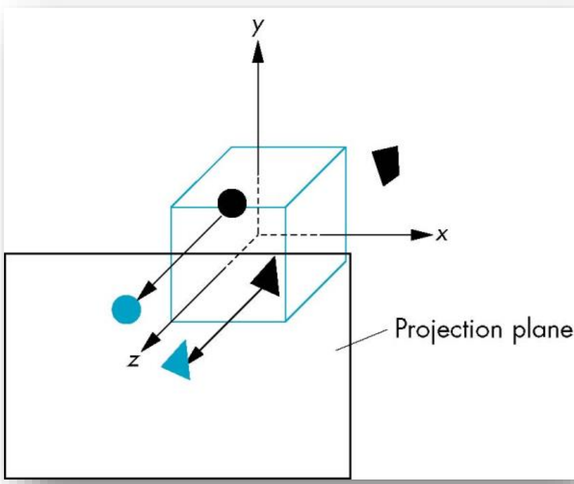
$$\begin{bmatrix} x'_{vc} \\ y'_{vc} \\ z'_{vc} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{vc} \\ y_{vc} \\ z_{vc} \\ 1 \end{bmatrix}$$

〖(RT)〗^(−1)=T^(−1) R^(−1)
相機座標會在世界座標的某處

Projections and Normalization

將其他運算的東西放到一個cube裡

- The default projection in the eye (camera) frame is orthogonal 從相機投影到畫面=>正交投影(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

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

$$\boxed{\boldsymbol{p}_p} = \boldsymbol{Mp}$$

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

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.

- <u>Clipping with 3D volume</u>.

  設立視野 : 不去計算看不到或太遠的model

- Associating projection with clipping and normalization.

Why do we use normalization ?

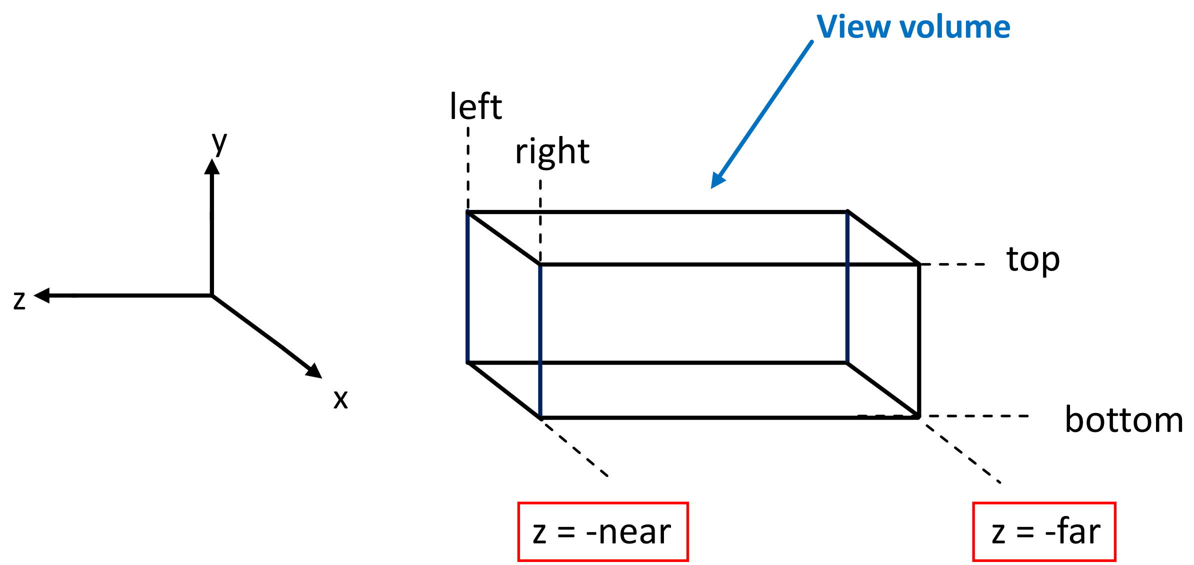
Orthogonal Viewing Volume

不在乎場景以外或太遠的物件，
在view volume外的東西都砍掉

distance

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

最近或最遠的物體突然跑出來或不見。
基本上near = 0，避免相機後方的東西突然跑出來



near : distance
Znear : z方向

實際看出去的空間是負的
(相機面向負z軸)

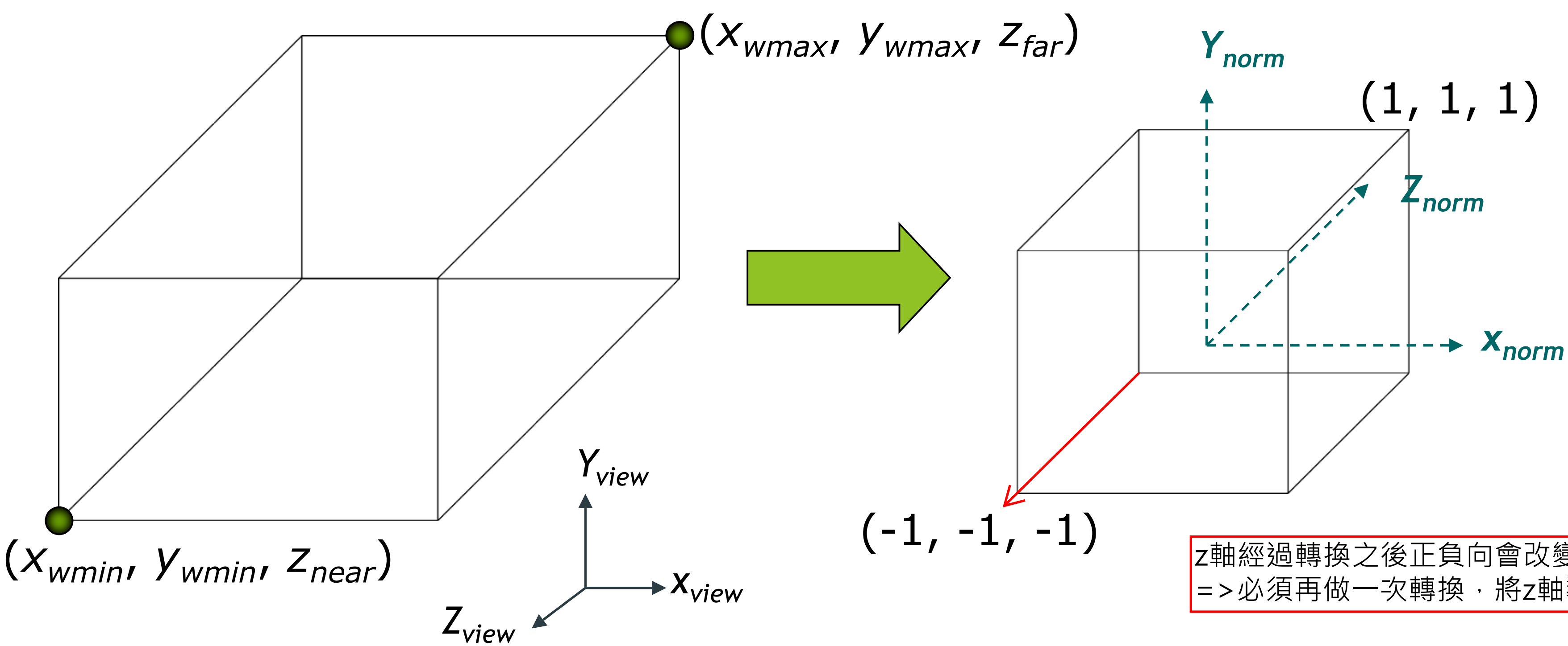
Orthogonal Normalization

glViewport : 投影回螢幕

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

將相機的座標系統乘上一個matrix，直接壓到我們要的平面上，基本上壓在-1~+1之間

normalization ⇒ find transformation to convert specified clipping volume to default



z軸經過轉換之後正負向會改變(慣例)
=>必須再做一次轉換，將z軸導正

Orthogonal Matrix

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

將中心壓回圓心

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

正負號變換問題

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

cube = MpMvMm*(物體)

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

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

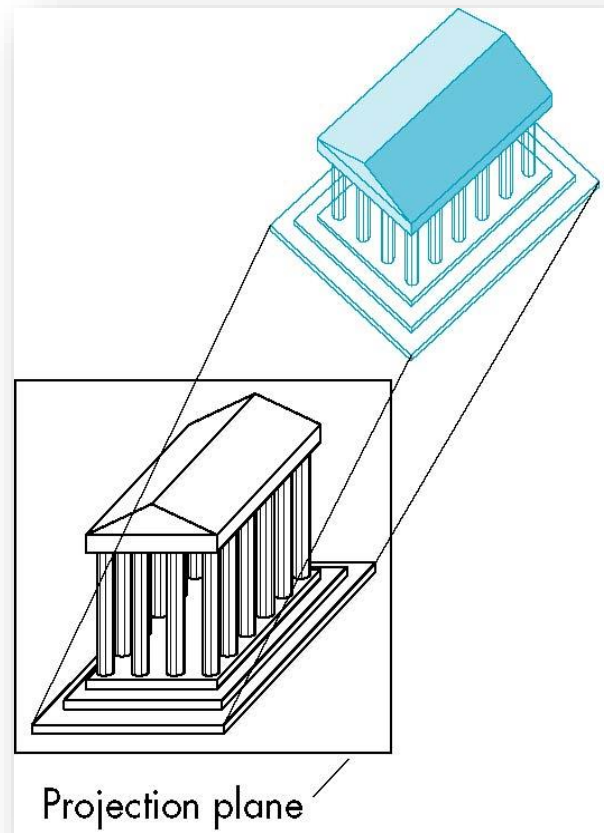
- Hence, general orthogonal projection in 4D is

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

Oblique Projection

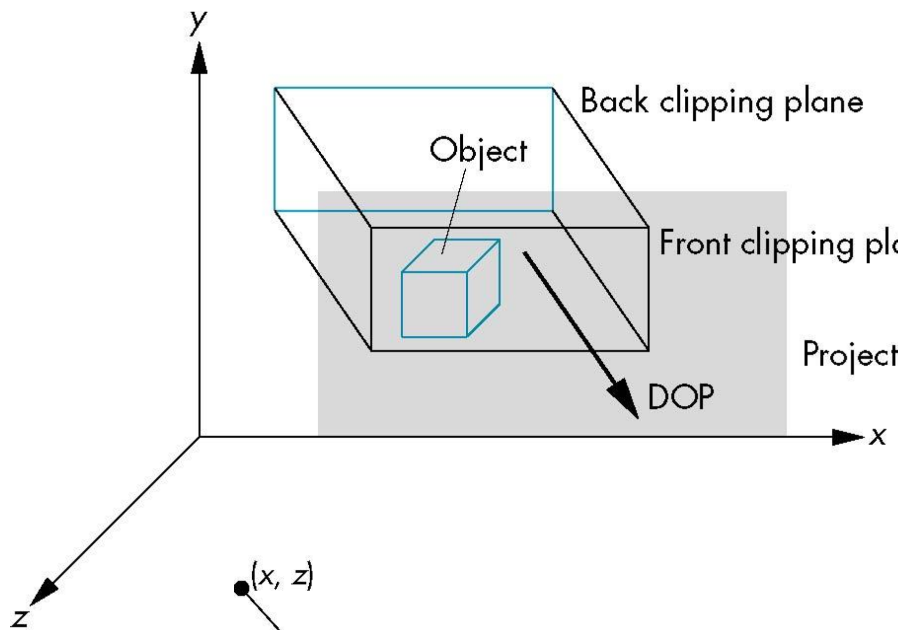
傾斜

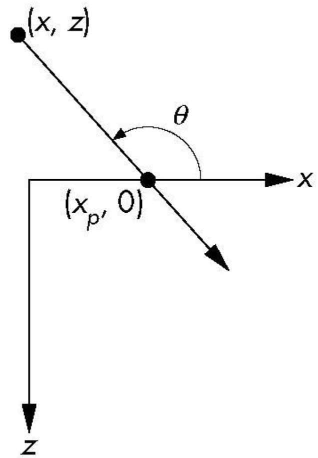
- The OpenGL projection functions cannot produce general parallel projections such as



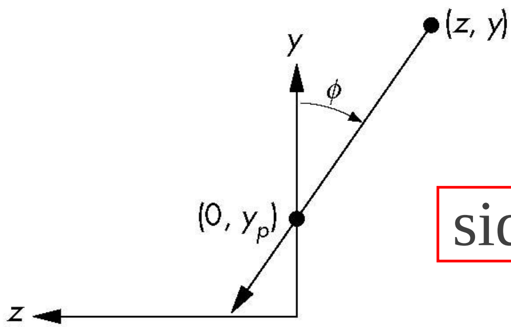
- How to efficiently produce such views?

Shear parallel to the x and y axes





top view



side view

Applying Shear Matrix

xy shear (*z* values unchanged)

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

把原來直投影的東西，
先將空間扭曲(shear)

Projection matrix

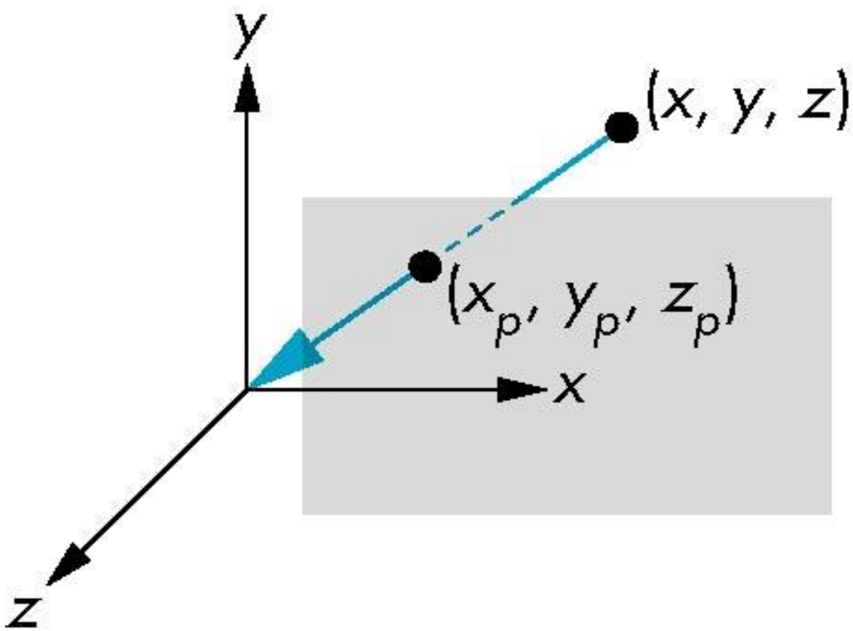
$$\mathbf{P} = \overset{\text{fixed}}{\mathbf{M}_{\text{orth}}} \boxed{\mathbf{H}(\theta,\phi)}$$

General case:

$$\mathbf{P} = \mathbf{M}_{\text{orth}}\,\mathbf{STH}(\theta,\phi)$$

Simple Perspective

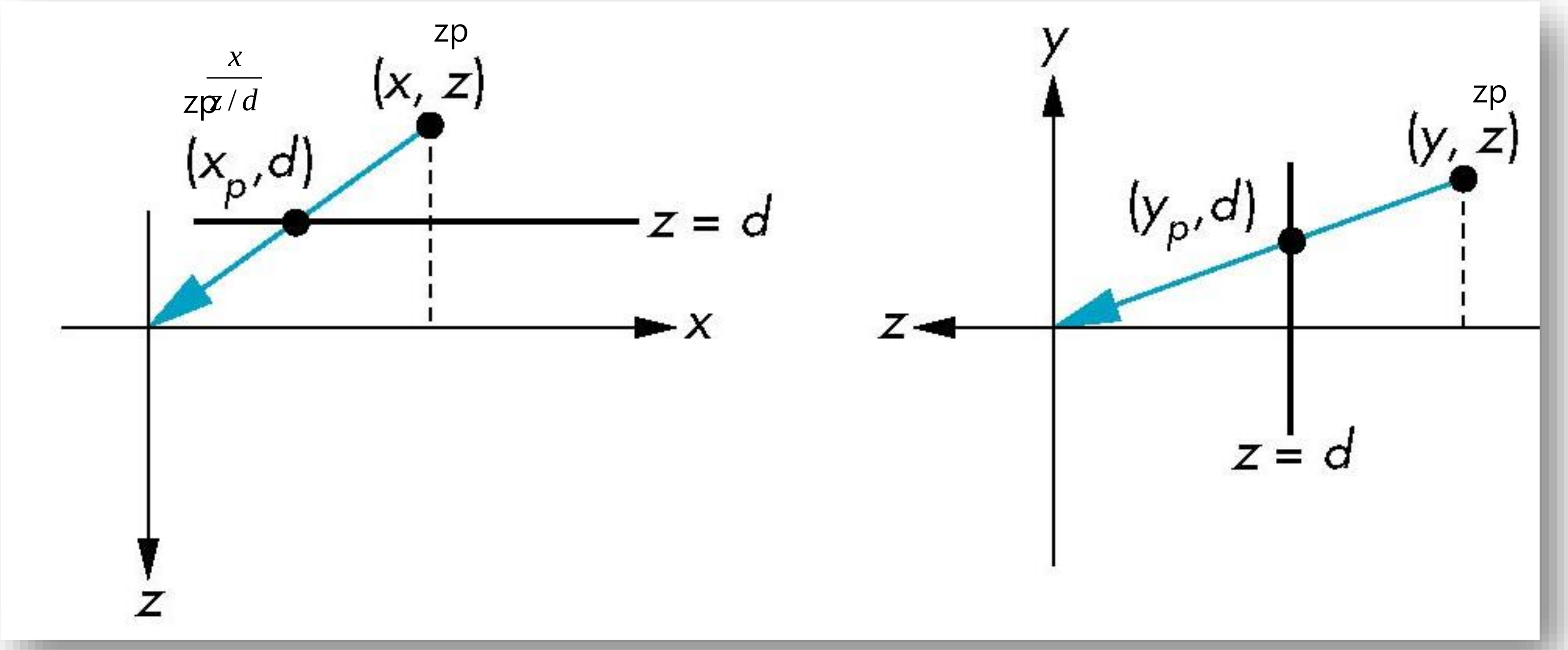
- Center of projection at the origin
- Projection plane $z = d, d < 0$



Perspective Equations 相似三角形

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

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

goal : matrix

dx/z

dy/z

d

1

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

投影時整條射線等價

Perspective Division

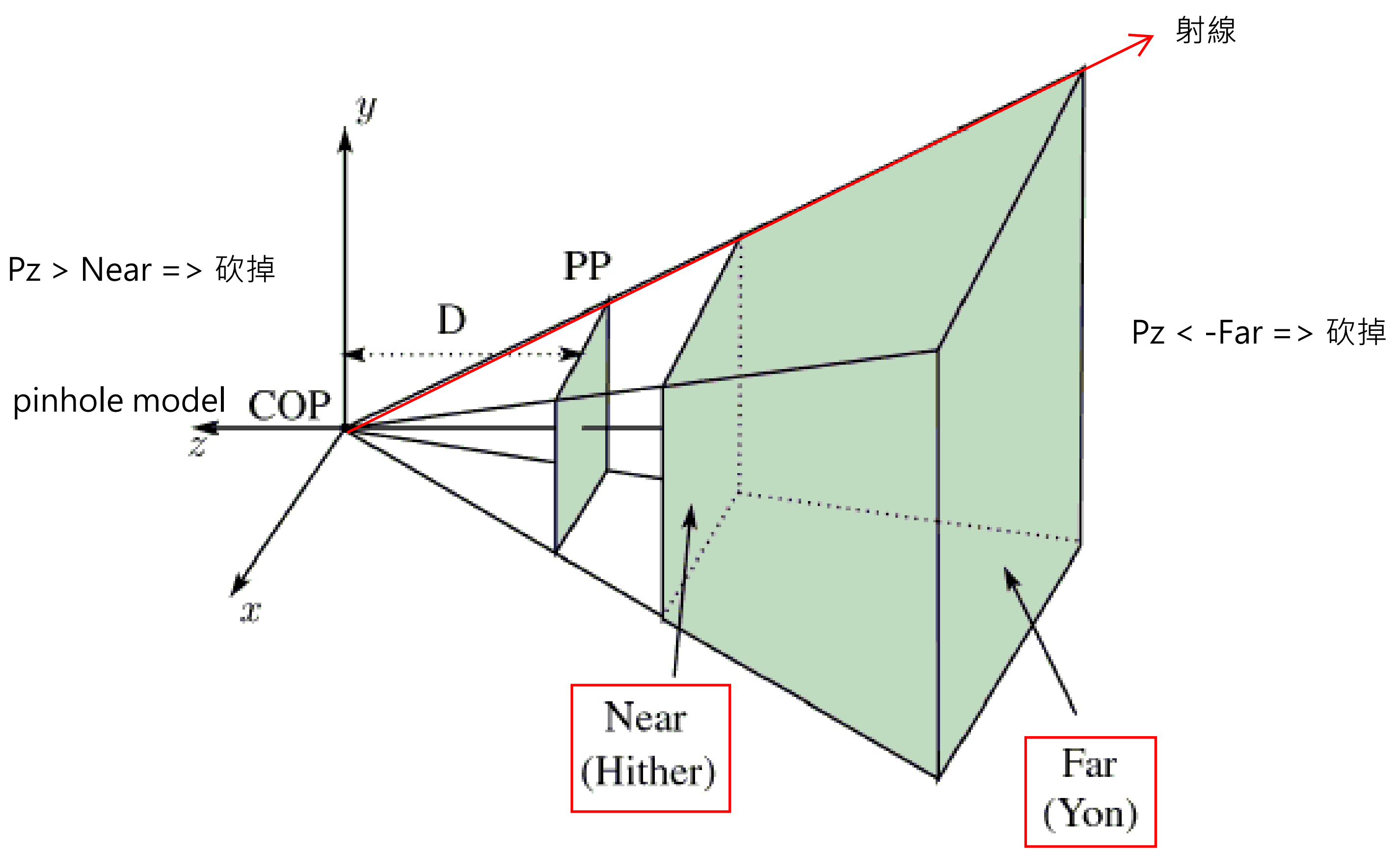
- However $w \neq 1$, so we must divide by w to return from homogeneous coordinates
- This *perspective division* yields

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

the desired perspective equations

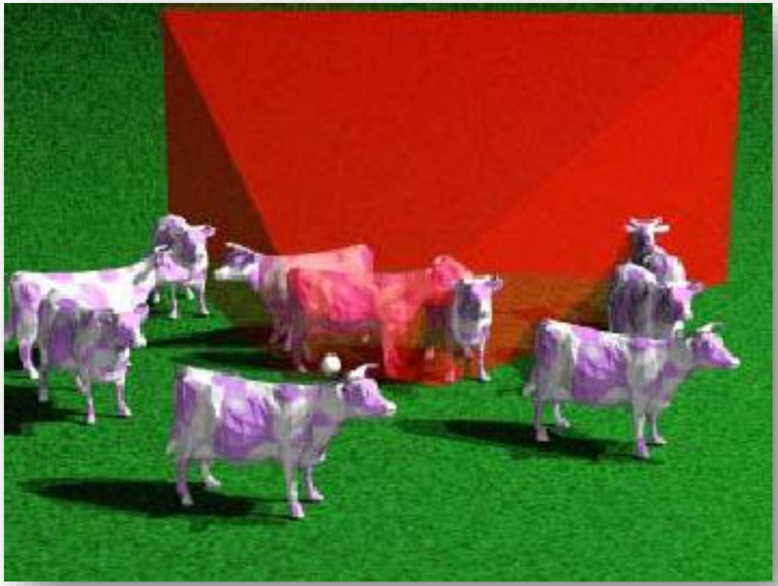
Perspective Viewing Volume

frustum : 截頭錐形體



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

Clipping for Perspective Views



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.*

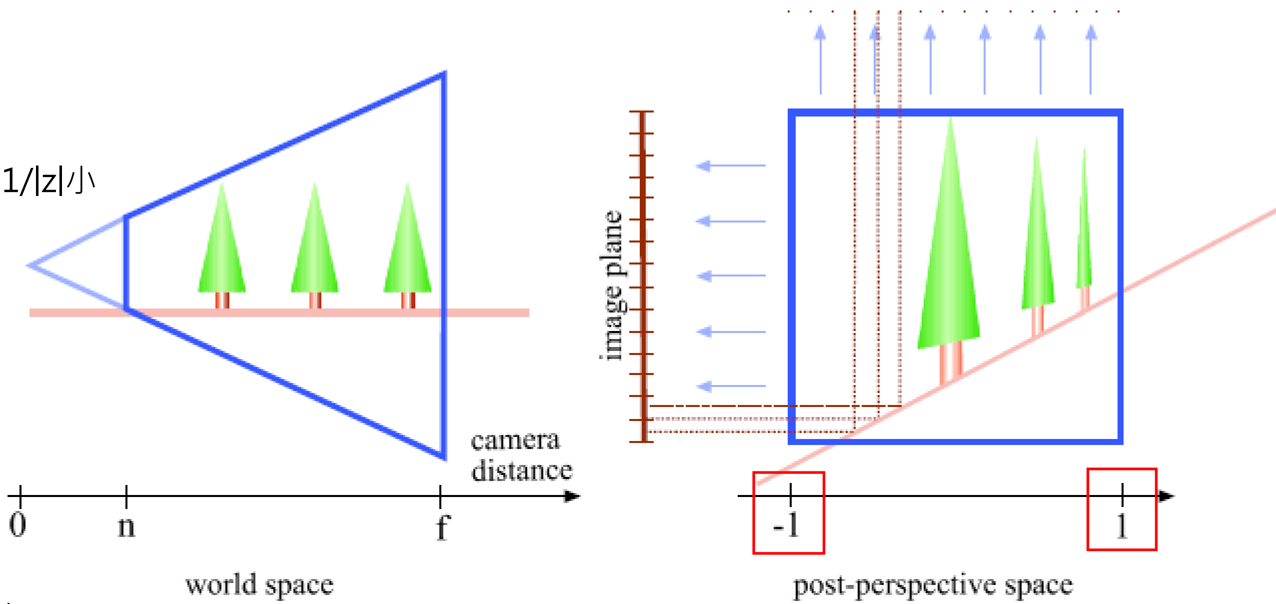
- This strategy allows us to use *standard transformations* in the pipeline and makes for *efficient clipping.*

Normalization

OpenGL處理p.47周圍四個平面：將空間扭曲=>使用orthogonal做投影

為了保持光線直線前進，
物體要在同樣的射線上、
物體前後順序也要相同

|z|大 => 1/|z|小



因為是非等比例轉換：
物體之間的距離變小

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

原本的射線轉換後變成水平線
前後關係不變，所以遮蔽演算法一樣能work

Perspective-Projection Trans.

$$
\begin{array}{cc}
-Znear/-z & -ZnearX \\
-ZnearY/z & -ZnearY \\
(SzZ+Tz)/z & SzZ+Tz \\
1 & -Z
\end{array}
=
\boxed{M_{pers} = \begin{bmatrix} -z_{near} & 0 & 0 & 0 \\ 0 & -z_{near} & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}}
\begin{array}{c} x \\ y \\ z \\ 1 \end{array}
$$

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

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

變數

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

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

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

Zp_near = -(Sz + Tz/Znear)

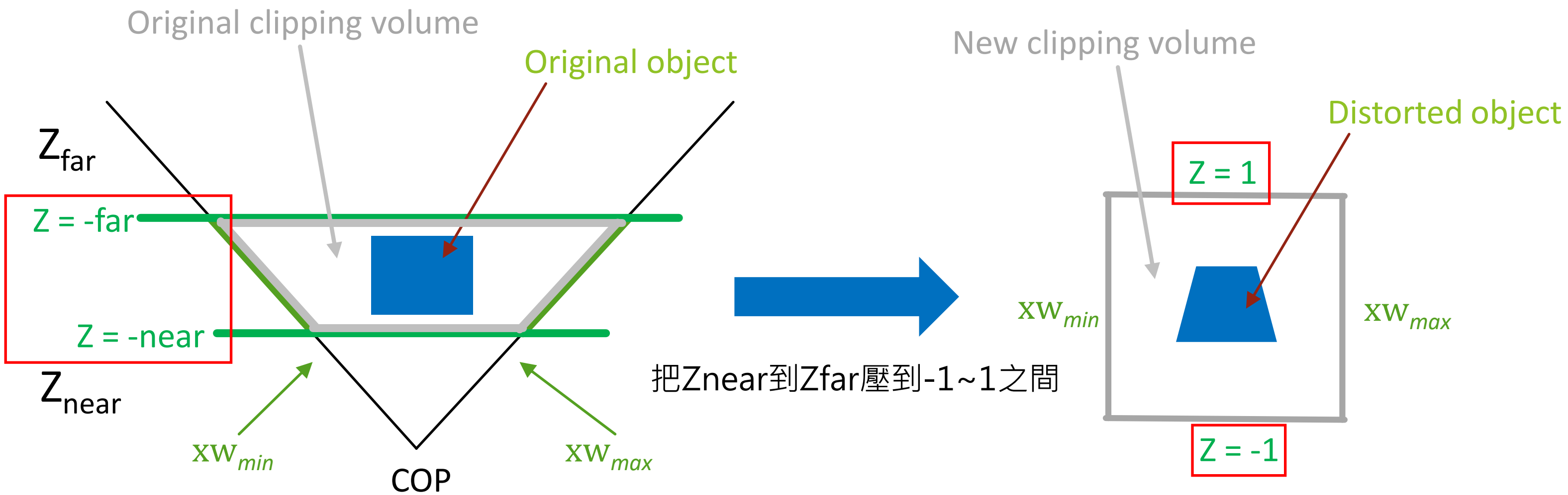
Zp_far = -(Sz+Tz/Zfar)

=>

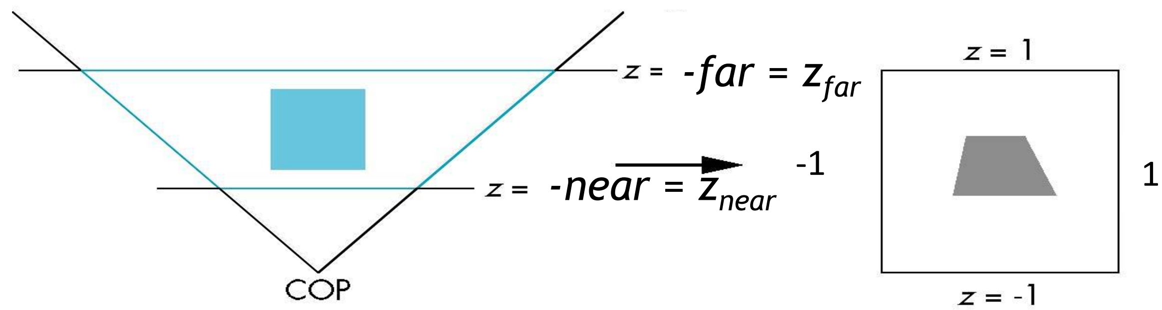
Sz = (Znear + Zfar)/(Znear - Zfar)

Tz = (-2ZnearZfar)/(Znear - Zfar)

Perspective-Projection Trans.



Further Normalization



original object

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

scaling vector (normalization)

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

Notes

- Normalization let 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

    保持順序性原因=>要把後方物件remove掉

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

浮點數誤差

Why do we do it this way?

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

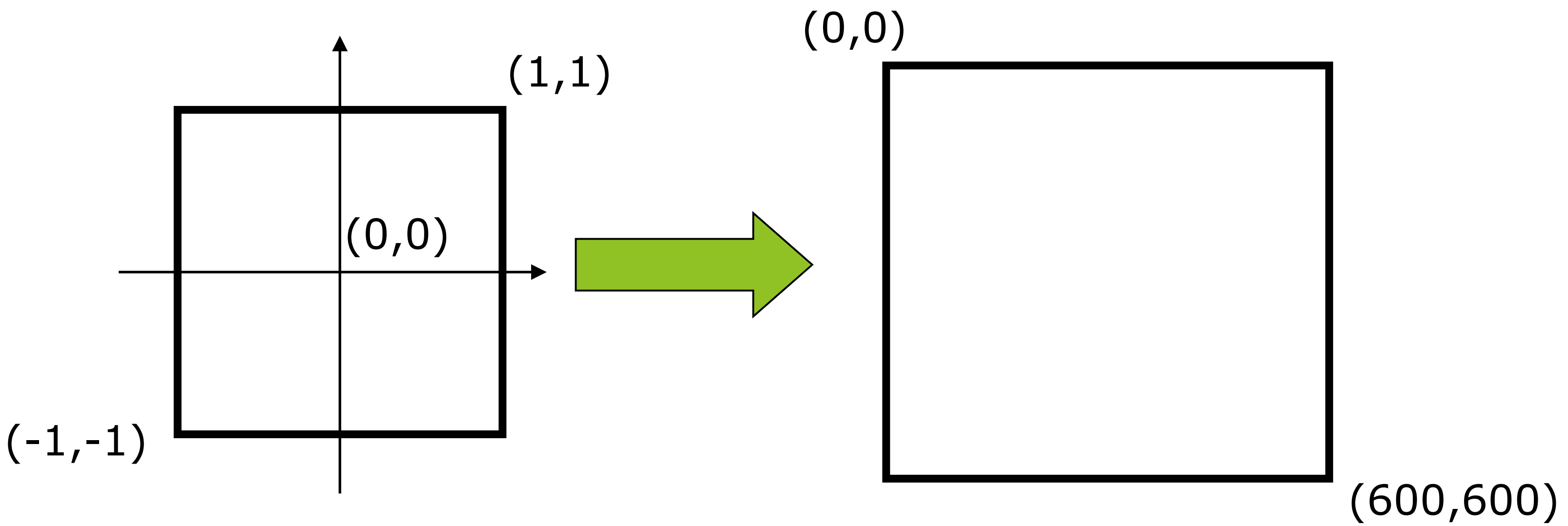
- We stay in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed for hidden-surface removal and shading

- *Clipping is now "easiler".*

Viewport Transformation

glViewport只要比例不要設錯就可以變回原來的圖

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

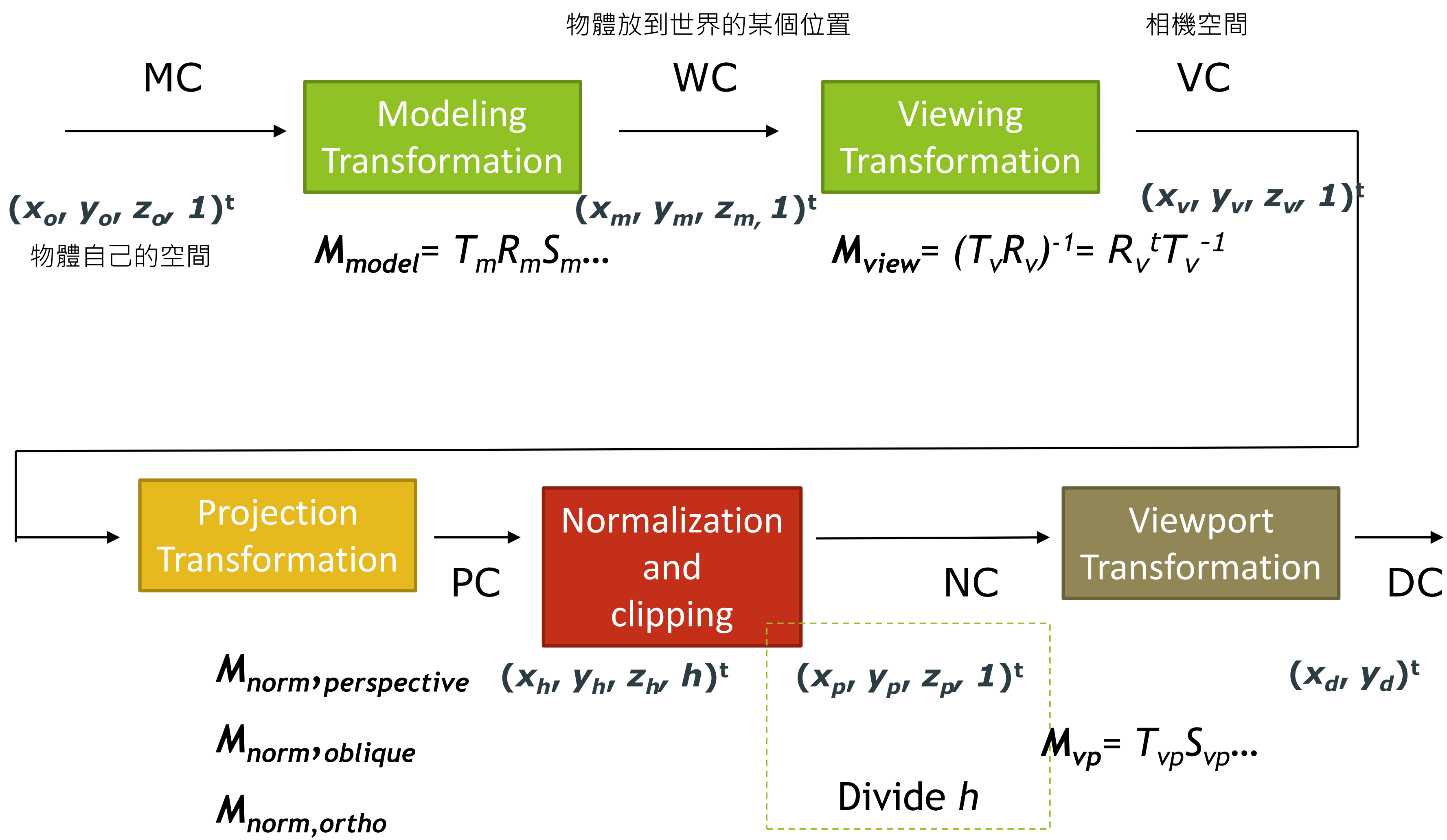


By 2D scaling and translation

Viewing in 3D

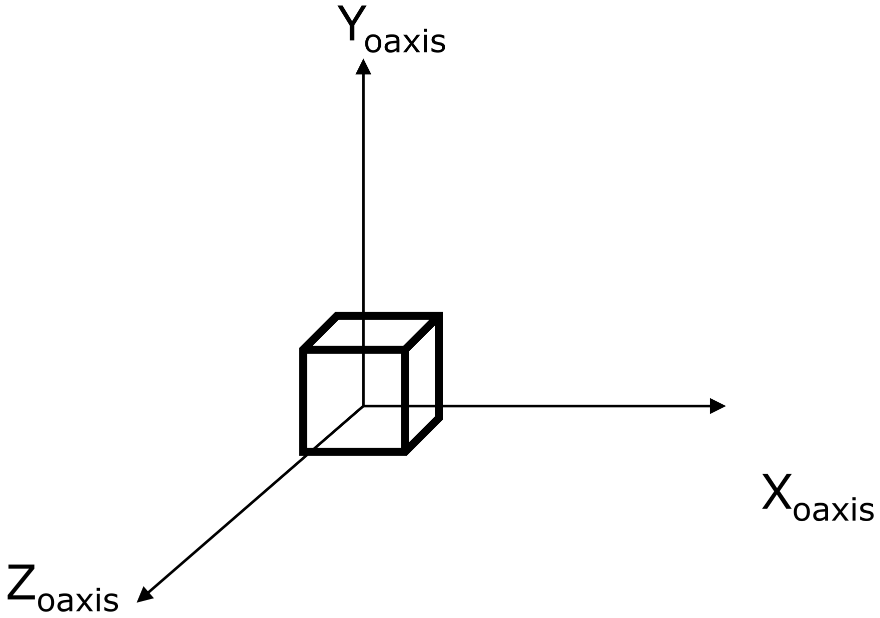
(Summary and Example)

Pipeline View



Loading an Object

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

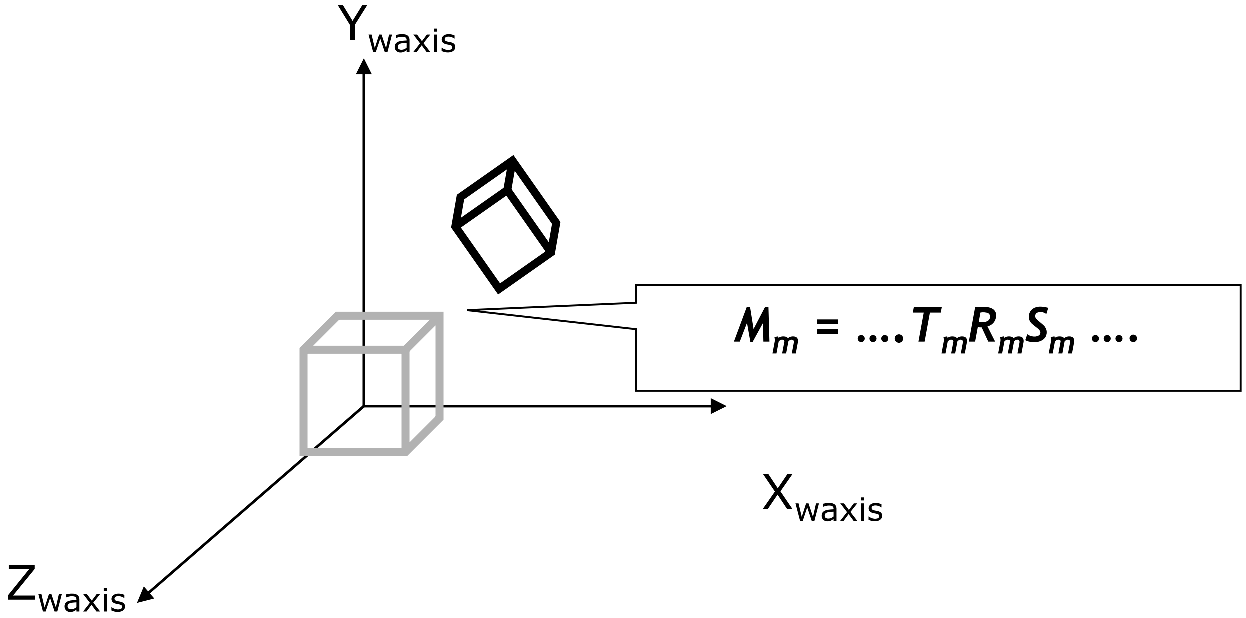


Object coord.

Modeling Transformation

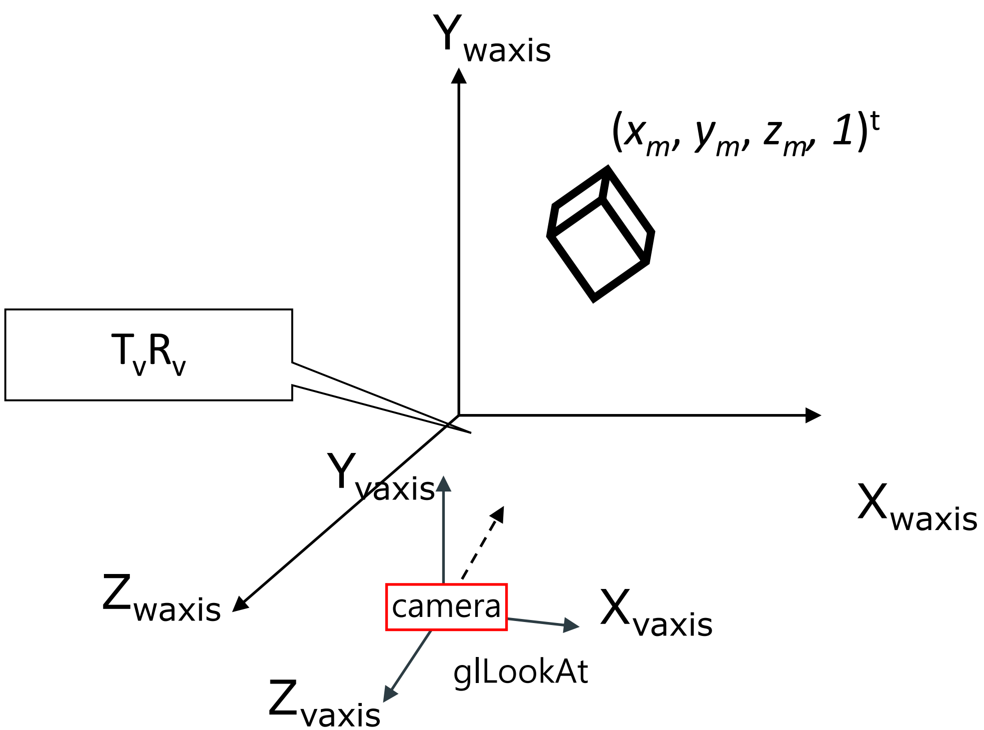
- $(x_m, y_m, z_m, 1)^t = M_m(x_o, y_o, z_o, 1)^t$

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



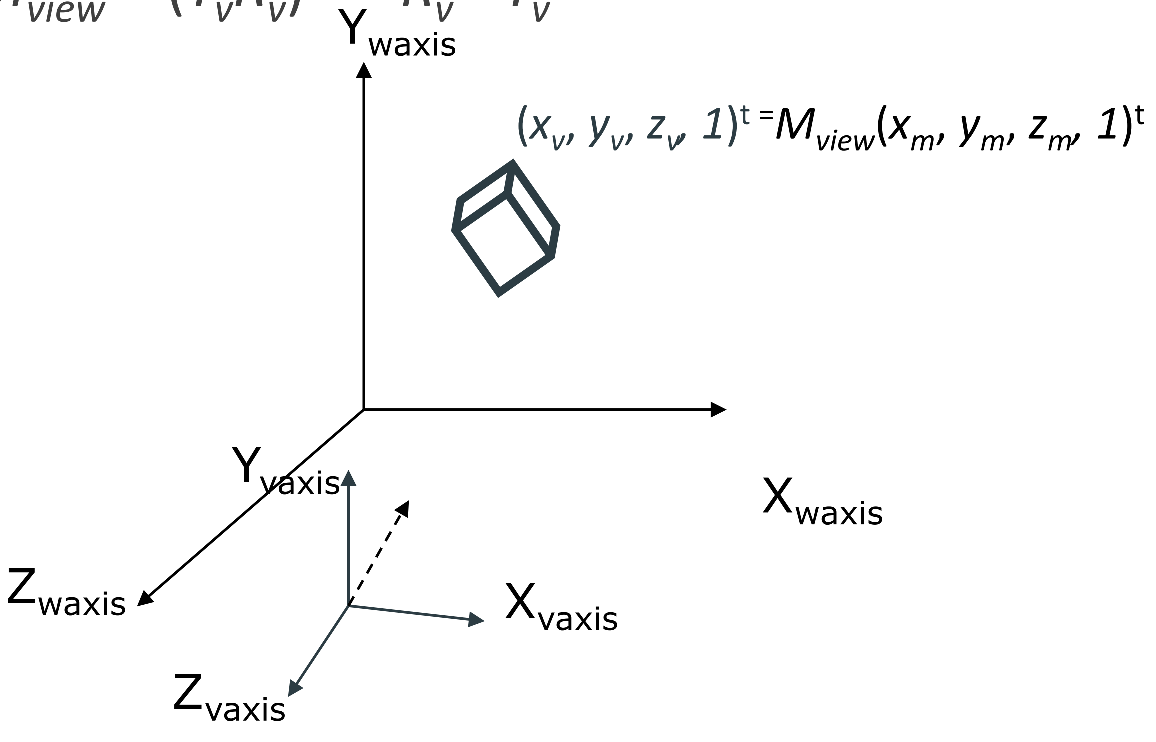
Put a Virtual Camera

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

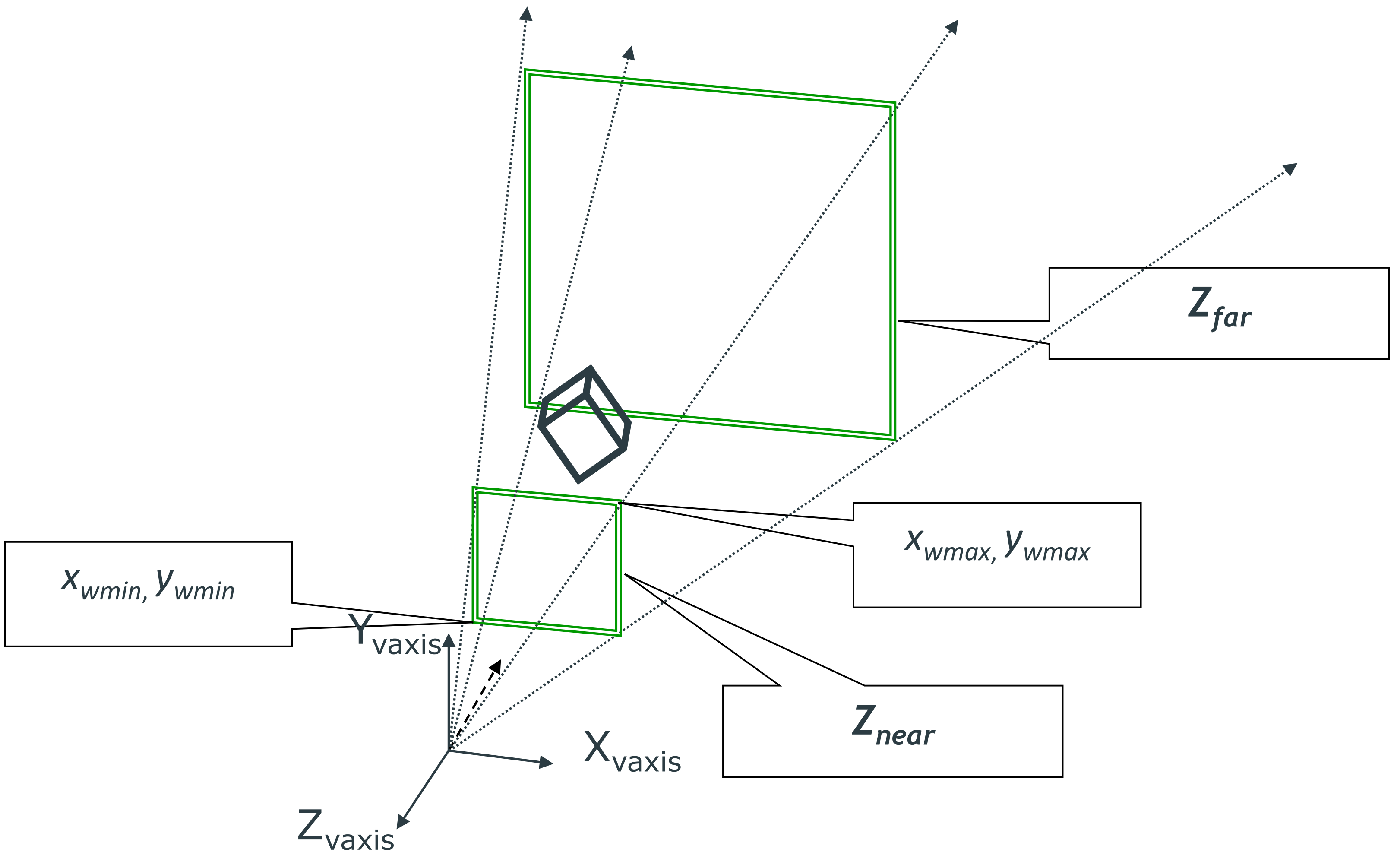


Virtual Camera's Coordinate

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



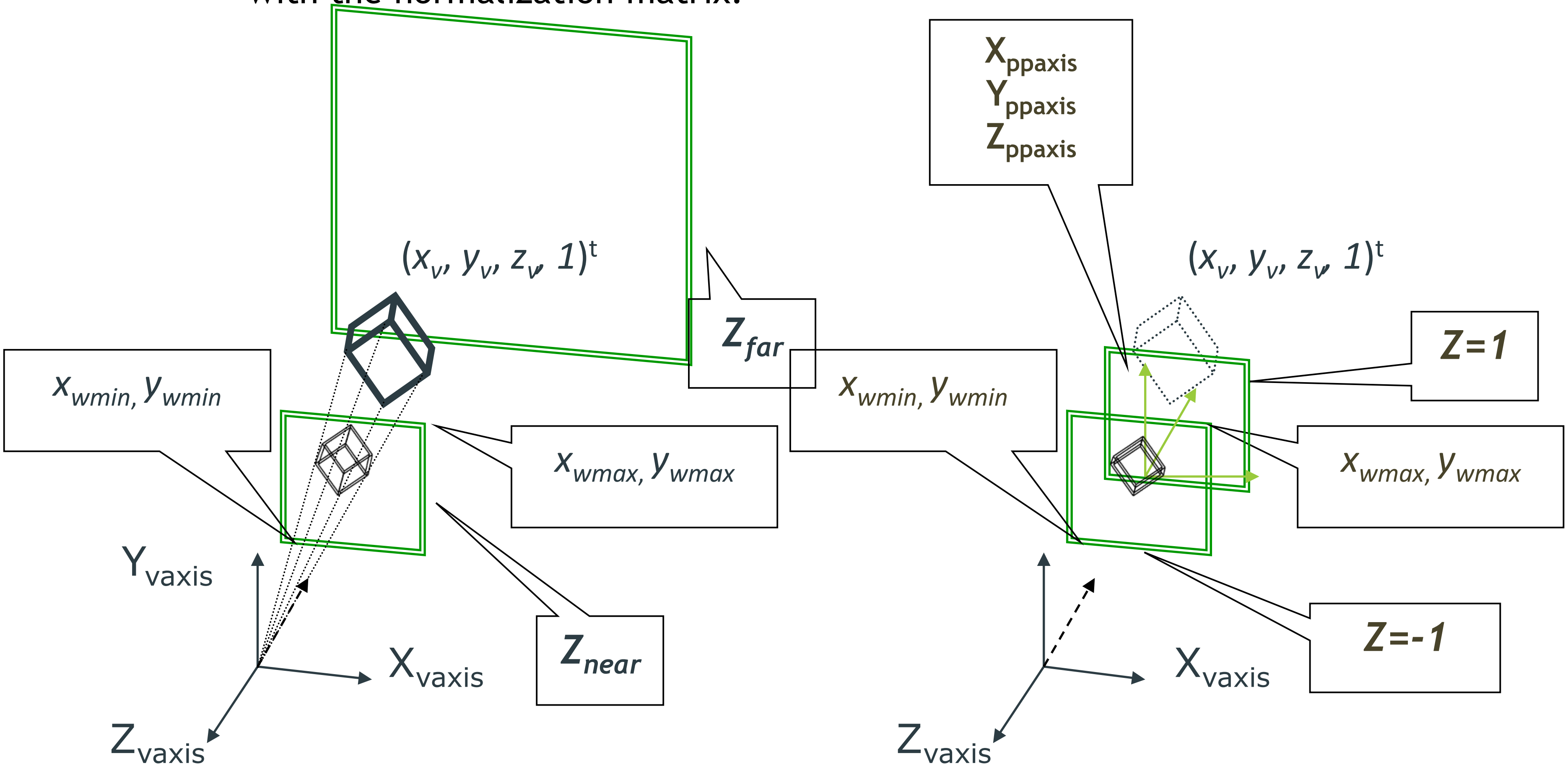
Virtual Camera's Coordinate



Perspective Proj.

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

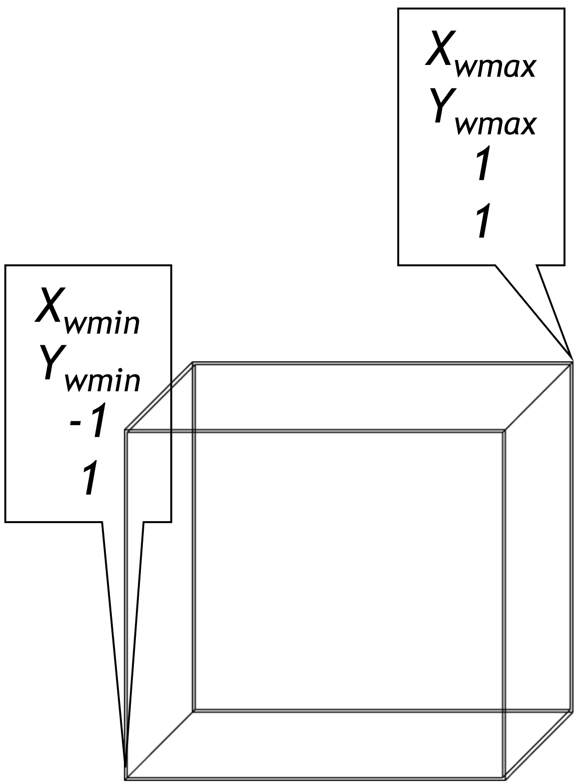
This matrix is usually combined with the normalization matrix.

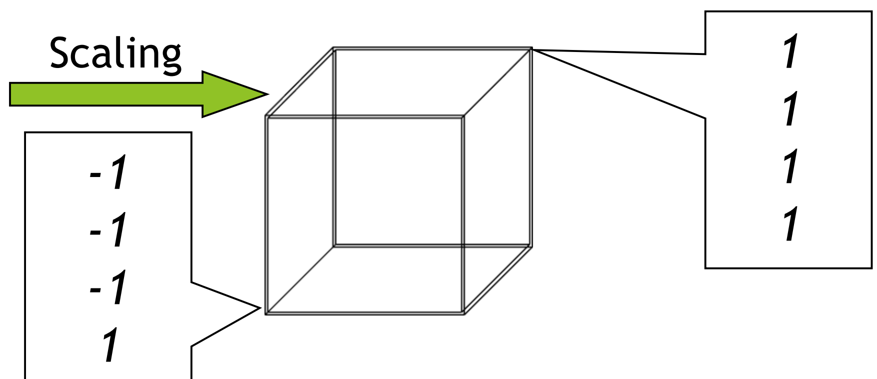


Projection + Normalization

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

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



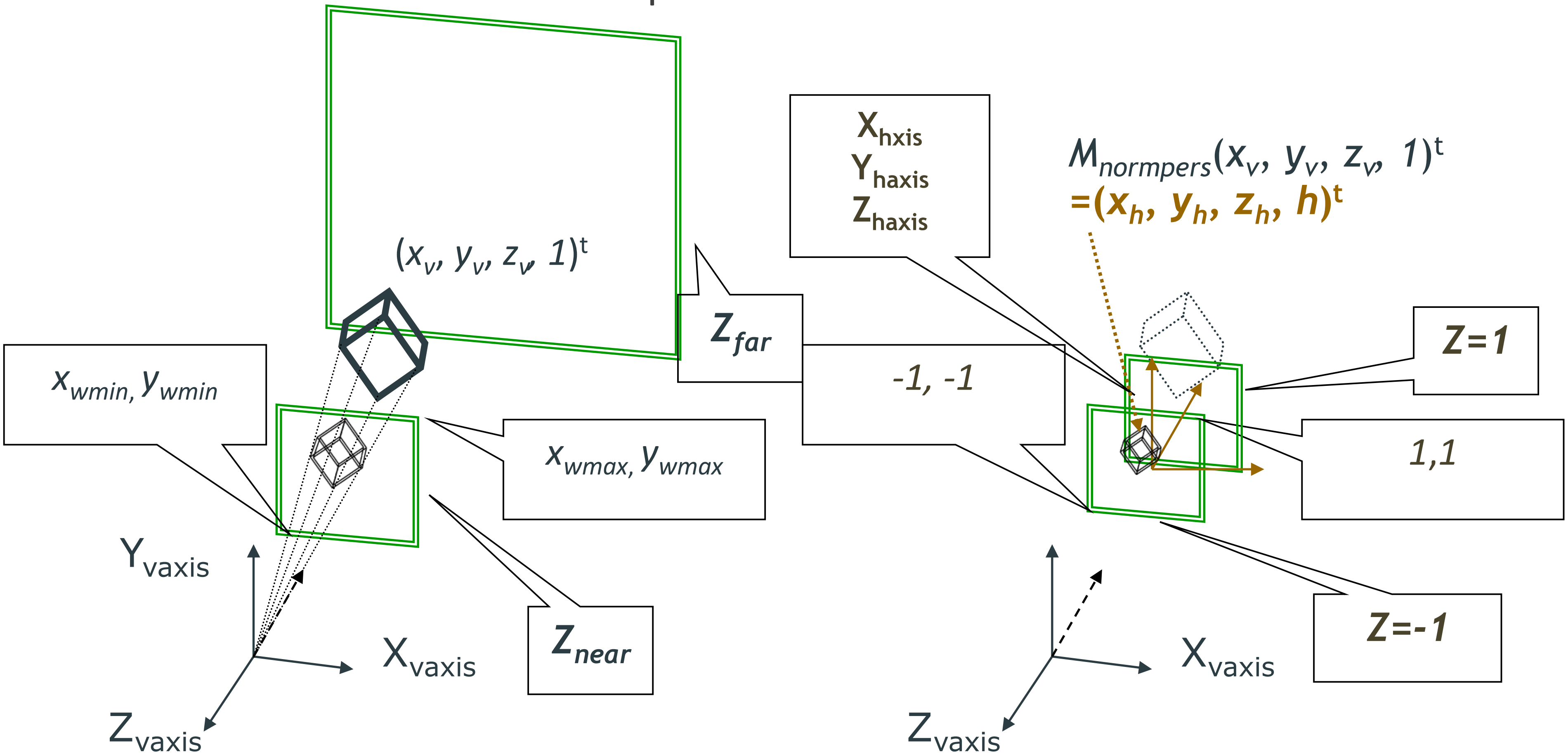


Projection+ Normalization

真正在使用的

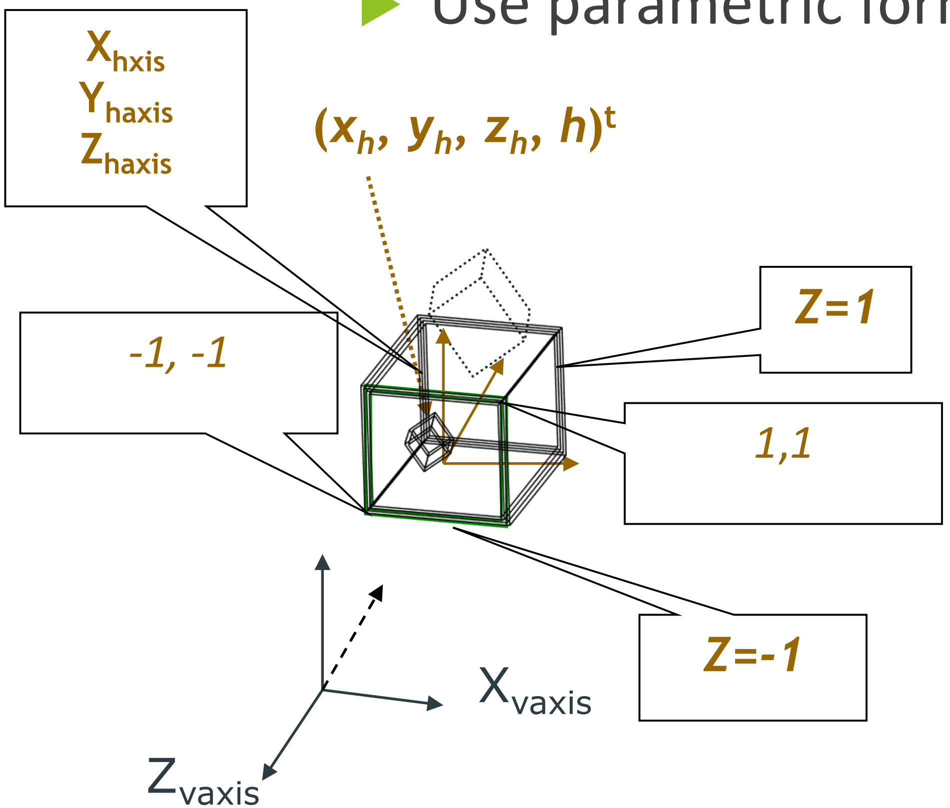
$$M_{normpers} = \begin{bmatrix} -z_{near}\dfrac{2}{xw_{max}-xw_{min}} & 0 & 0 & 0 \\ 0 & -z_{near}\dfrac{2}{yw_{max}-yw_{min}} & 0 & 0 \\ 0 & 0 & \dfrac{z_{near}+z_{far}}{z_{near}-z_{far}} & \dfrac{-2z_{near}z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- $(x_h, y_h, z_h, h)^t = M_{normpers}(x_v, y_v, z_v, 1)^t$
- 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



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

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

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

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

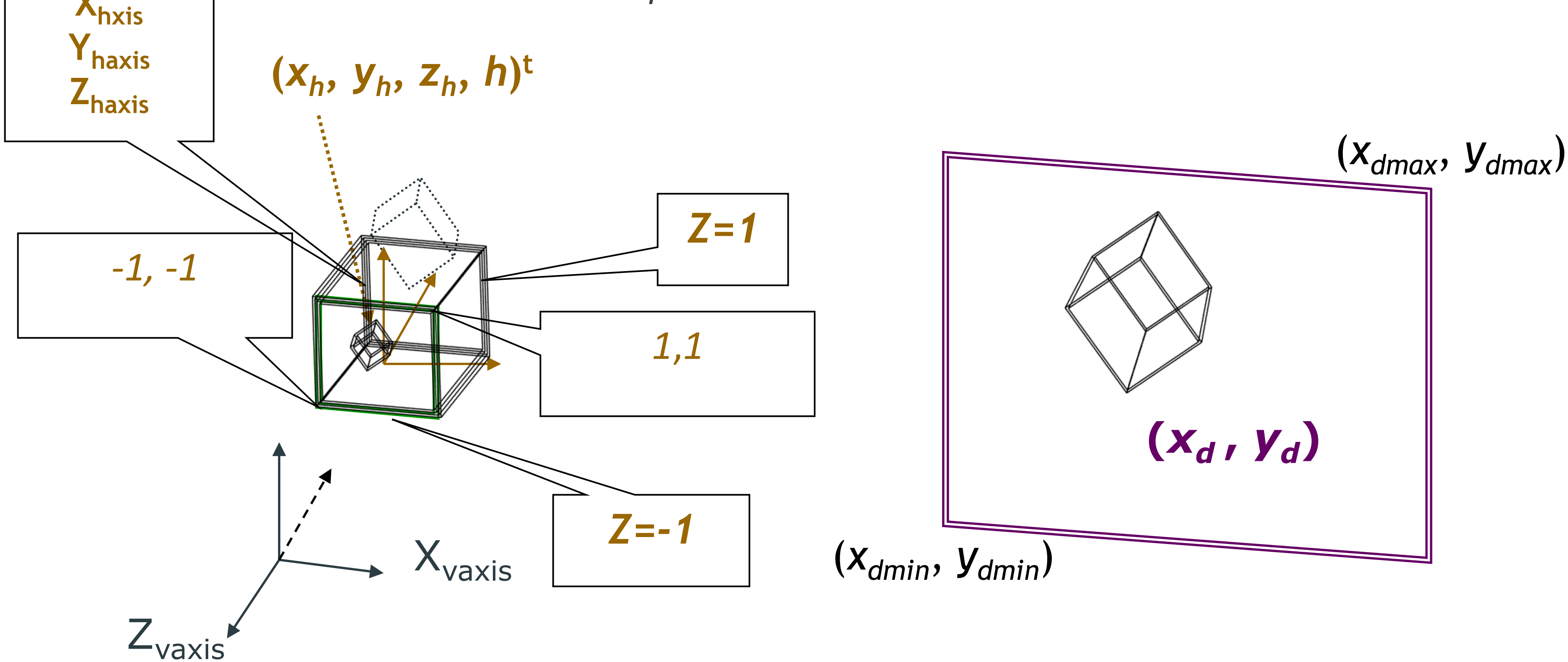
Viewport Transformation

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

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

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