CS4533 Lecture 4 Slides/Notes

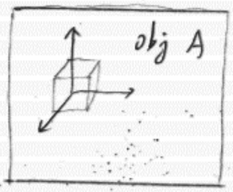
Viewing and Projection
(Ch 5, 10, 11, 12.1, Notes)

By Prof. Yi-Jen Chiang
CSE Dept., Tandon School of Engineering
New York University

1

Viewing & Projection

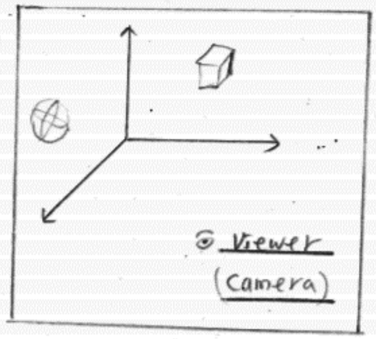
* Coordinate Systems & Transformations



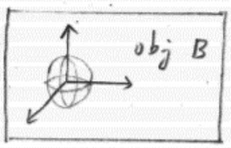
obj A's transformation

① (Modeling) transformation

2. (World) Coordinate system



1. (Object) Coordinate Systems



obj B's transf.

Modeling transf.

(X_o, Y_o, Z_o)

② (Viewing) Transformation

⊗ $(X, Y, Z)_{world}$ $[= (X_{world}, Y_{world}, Z_{world})]$

5. (Normalized Device) coord system

Perspective Division

(if Perspective projection)

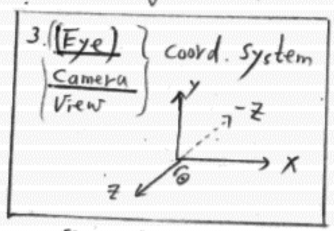
4. (clip) coord. system

~~3R~~ for clipping

(X_c, Y_c, Z_c)

③ (Projection) transf.

3. (Eye) / Camera / View coord. system



(X_e, Y_e, Z_e)

3D

Z: perpendicular to the paper

Note: Mention left-handed frame historically.
But we will use (right-handed frame) thru-out for consistency

(X_n, Y_n, Z_n)

(Viewport) Transf.

6. (Window) coord System

(2D/3D) (X_w, Y_w, Z_w)

2D, but still retain Z-values for hidden-surface removal. (Z-buffering)

Rasterization → fragments/Pixels

Z-buffering, fragment processing etc → Pixels

---

cf. World coord: $(X, Y, Z)_{world}$ $[= (X_{world}, Y_{world}, Z_{world})]$

Window ": (X_w, Y_w, Z_w)

Note:
1. Modeling Transf: Rotation, Translation, Scaling
   Viewing ": Coord. system change. (Using Translation & Rotation)
   } Same types ⇒ Combined into (model-view matrix)
2. Projection: Different type ⇒ Use a separate (Projection matrix)

Viewing & Projection Process : 2 Parts

1. Describe the position & orientation of Camera.

(Viewing) use model-view matrix to transform objects, world frame → {(eye) / (camera)} frame.

2. Decide type of Projection : { parallel / perspective

(Projection) what part to image. clipping / view volume } ⇒ (projection) matrix

* Concatenate projection matrix (P) with model-view Matrix (M)

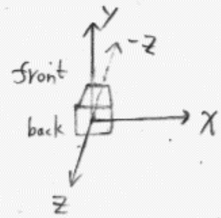
vertex → $\boxed{\text{model-view}}$ → $\boxed{\text{projection}}$

$q = (PM)p$

Viewing (Part 1): Specifying the camera position & Orientation.

then transform World frame $\longrightarrow$ Camera/Eye frame

Here: camera frame is a (right-handed) frame, with the camera/eye at the origin looking at the $(-z)$ direction.



① Specify VRP — view reference point

i.e. camera/eye position

VPN — View Plane normal

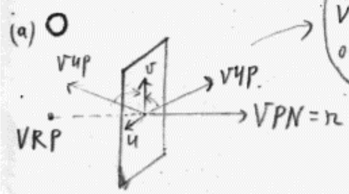
i.e. normal vector to the (view / projection) plane

vector of line of sight

VUP — view up vector (camera top direction)

for

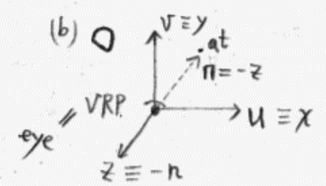
in the world frame

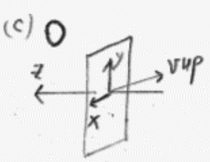


(VUP is not necessarily on the view plane)

② Construct the camera frame:

- * Project VUP onto the projection plane to get v
- * $n \times v = u$ to get u
- * VRP as origin





③ Transform world frame $\longrightarrow$ camera frame

* Typically, use function LookAt(eye, at, vup) (eye, at, vup in (world frame)) to obtain a 4×4 matrix M to transform world frame $\longrightarrow$ camera frame (right-handed)

eye $\cdot \longrightarrow \cdot$ at

line of sight $= VPN = n = -z$

* Derivation of matrix M for LookAt( ):

(1) $\underline{z} = -n = \text{normalize}(eye - at) = (z_1, z_2, z_3)$ (from fig (b))

$\underline{x} = u = \text{normalize}(VUP \times z) = (x_1, x_2, x_3)$ ($\because$ fig (c))

$\underline{y} = v = \text{normalize}(z \times x) = (y_1, y_2, y_3)$ ($\because$ fig (c))

origin $= eye = (eye_1, eye_2, eye_3)$

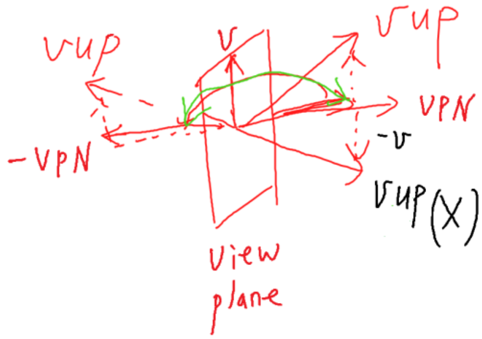
* We need to (normalize) since we need $x \cdot x = y \cdot y = z \cdot z = 1$

(2) ① $\text{Translate}(-eye) = \begin{bmatrix} 1 & 0 & 0 & -eye_1 \\ 0 & 1 & 0 & -eye_2 \\ 0 & 0 & 1 & -eye_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

② Rotate:

$$\begin{bmatrix} [x] & 0 \\ [y] & 0 \\ [z] & 0 \\ 0\ 0\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & 0 \\ y_1 & y_2 & y_3 & 0 \\ z_1 & z_2 & z_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = R$$

$M = R \cdot \text{Translate}(-eye)$ ※



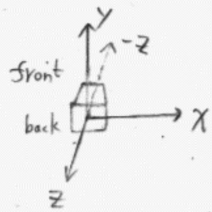
* **Note**: vup can be anywhere in the direction range (vup ← → vup) since in this range vup projecting onto the view plane results in the +v direction, which indicates the top of the camera

Any direction below (-VPN ← → VPN) eg. ↘ vup, is wrong, since its projection onto the view plane is in the -v direction.

Viewing (Part I): Specifying the camera position & Orientation.

then transform World frame → Camera/Eye frame

Here: camera frame is a (right-handed) frame, with the camera/eye at the origin looking at the (−z) direction

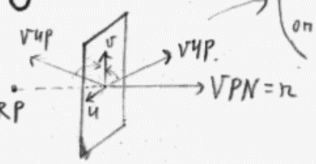


① Specify VRP — view reference point
ie camera/eye position

VPN — View Plane normal
ie normal vector to the {view / projection} plane
vector of line of sight

VUP — view up vector (camera top direction) for
in the world frame

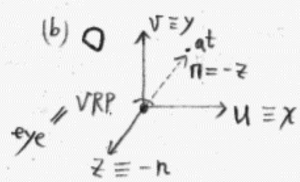
(a) O



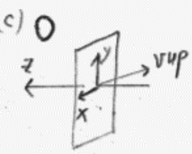
(VUP is not necessarily on the view plane)

② Construct the camera frame:
* Project VUP onto the projection plane to get v
* n × v = u to get u
* VRP as origin

(b) O

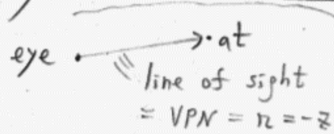


(c) O



③ Transform world frame → camera frame

* Typically: use function LookAt(eye, at, vup) (eye, at, vup in (world frame)) to obtain a 4×4 matrix M to transform world frame → camera frame (right-handed)



* Derivation of matrix M for LookAt( ):

(1) $z = -n = \text{normalize}(eye - at) = (z_1, z_2, z_3)$ (from fig (b))

$x = u = \text{normalize}(vup \times z) = (x_1, x_2, x_3)$ (: fig (c))

$y = v = \text{normalize}(z \times x) = (y_1, y_2, y_3)$ (: fig (c))

origin $= eye = (eye_1, eye_2, eye_3)$

* We need to (normalize) since we need $x \cdot x = y \cdot y = z \cdot z = 1$

(2)

① Translate(−eye) $= \begin{bmatrix} 1 & 0 & 0 & -eye_1 \\ 0 & 1 & 0 & -eye_2 \\ 0 & 0 & 1 & -eye_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

② Rotate:

$\begin{bmatrix} [x] & 0 \\ [y] & 0 \\ [z] & 0 \\ 0\ 0\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & 0 \\ y_1 & y_2 & y_3 & 0 \\ z_1 & z_2 & z_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = R$

$M = R \cdot \text{Translate}(-eye)$ *

Verification for M [eye]$_{world}$ = (origin)$_{eye}$?

$$M = R \cdot \mathrm{Translate}(-eye) = R \cdot \begin{bmatrix} 1 & 0 & 0 & -eye_1 \\ 0 & 1 & 0 & -eye_2 \\ 0 & 0 & 1 & -eye_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$M \cdot (eye)_{world}$

$$= M \cdot \begin{bmatrix} eye_1 \\ eye_2 \\ eye_3 \\ 1 \end{bmatrix} = R \cdot \begin{bmatrix} 1 & 0 & 0 & -eye_1 \\ 0 & 1 & 0 & -eye_2 \\ 0 & 0 & 1 & -eye_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} eye_1 \\ eye_2 \\ eye_3 \\ 1 \end{bmatrix} = R \cdot \begin{bmatrix} \cancel{eye_1} + 0 + 0 + \cancel{(-eye_1)} \cdot 1 \\ 0 + \underline{eye_2} + 0 - \underline{eye_2} \\ 0 + 0 + \underline{eye_3} - \underline{eye_3} \\ 0 + 0 + 0 + 1 \end{bmatrix}$$

$$= R \cdot \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{\text{origin}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{\text{origin}} = (\text{origin})_{eye}. \checkmark$$

* origin is the fixed pt of rotation.

So $R \cdot (\text{origin})$ does NOT change the origin.

✱ Verification :

① $M \cdot \begin{bmatrix} eye \end{bmatrix}_{world} = (origin)_{eye}$? $\quad M \cdot \begin{bmatrix} eye_1 \\ eye_2 \\ eye_3 \\ 1 \end{bmatrix} = R \cdot \begin{bmatrix} eye_1 + 0 + 0 - eye_1 \\ 0 + eye_2 + 0 - eye_2 \\ 0 + 0 + eye_3 - eye_3 \\ 1 \end{bmatrix} = R \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

$= origin$ ✓

② $M \cdot \begin{bmatrix} x \end{bmatrix}_{world} = (e_1)_{eye}$? $\quad M \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 0 \end{bmatrix} = \begin{bmatrix} [x] & 0 \\ [y] & 0 \\ [z] & 0 \\ 0\ 0\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 0 \end{bmatrix}$ (∵ Translate(−eye) has no effect on vector x)

$$= \begin{bmatrix} x \cdot x + 0 \\ y \cdot x + 0 \\ z \cdot x + 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = e_1 \ ✓$$

Similarly. $M \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ 0 \end{bmatrix}_{world} = \begin{bmatrix} x \cdot y + 0 \\ y \cdot y + 0 \\ z \cdot y + 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = (e_2)_{eye}$ ✓ $\quad M \cdot \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ 0 \end{bmatrix}_{world} = \begin{bmatrix} x \cdot z + 0 \\ y \cdot z + 0 \\ z \cdot z + 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = (e_3)_{eye}$ ✓

Note✱: $M = (M^{-1})^{-1}$

$M^{-1} = \left[R \cdot Translate(-eye)\right]^{-1} = \left(Translate(-eye)\right)^{-1} \cdot R^{-1} = Translate(eye) \cdot R^{t} = Translate(eye) \begin{bmatrix} [x] & [y] & [z] & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$$= \begin{bmatrix} 1 & 0 & 0 & eye_1 \\ 0 & 1 & 0 & eye_2 \\ 0 & 0 & 1 & eye_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & y_1 & z_1 & 0 \\ x_2 & y_2 & z_2 & 0 \\ x_3 & y_3 & z_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & z_1 & eye_1 \\ x_2 & y_2 & z_2 & eye_2 \\ x_3 & y_3 & z_3 & eye_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} [x] & [y] & [z] & [eye] \\ 0 & 0 & 0 & 1 \end{bmatrix} = A$$

$M = A^{-1}$ (This is the method of Textbook Sec 5.2.3 Lookat. Note that the matrix given there is just A. But final M is $M = A^{-1}$)

✱ Verification

① $M^{-1} (origin)_{eye} = M^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} eye_1 \\ eye_2 \\ eye_3 \\ 1 \end{bmatrix} = (eye)_{world}$ ✓

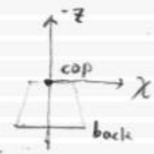
② $M^{-1} \cdot (e_1)_{eye} = M^{-1} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 0 \end{bmatrix} = (\text{vector } x)_{world}$ ✓

③ $M^{-1} \cdot (e_2)_{eye} = M^{-1} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ 0 \end{bmatrix} = (\text{vector } y)_{world}$ ✓

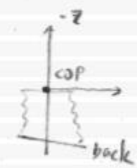
④ $M^{-1} \cdot (e_3)_{eye} = M^{-1} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ 0 \end{bmatrix} = (\text{vector } z)_{world}$ ✓

Projection (Part 2)

- perspective
- parallel



back of camera $\perp$ z axis

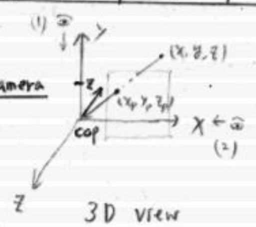


general case. (oblique)

Overview of Plan:
(1) Derive resulting pt q of orig. pt p by def. of projection
(2) Derive projection matrix M: $q = Mp$

Simple perspective projection (back $\perp$ z axis)

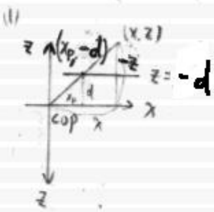
(0)



3D view

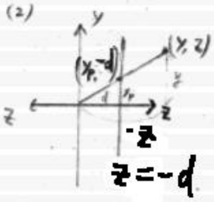
Use camera frame but with camera default direction (facing -z)

(1)



top view

(2)



side view

$z_p = -d$, $z < 0$ (d > 0)

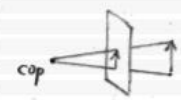
$$\frac{x}{-z} = \frac{x_p}{d} \quad \Rightarrow \quad x_p = \frac{x \cdot d}{-z}$$

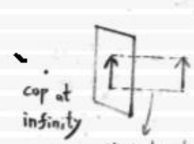
$$\frac{y}{-z} = \frac{y_p}{d} \quad \Rightarrow \quad y_p = \frac{y \cdot d}{-z}$$

$$p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad M = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \Rightarrow \quad q = Mp = \begin{bmatrix} xd \\ yd \\ zd \\ -z \end{bmatrix}$$

projection matrix for simple perspective projection

Def: perspective — parallel/orthogonal





projection lines are // to each other. All are $\perp$ to the view plane

$$(x_p, y_p, z_p) = \left(\frac{x \cdot d}{-z}, \frac{y \cdot d}{-z}, -d\right)$$

$$\underbrace{\begin{bmatrix} \frac{x \cdot d}{-z} \\ \frac{y \cdot d}{-z} \\ -d \\ 1 \end{bmatrix}}_{= q'} \xleftarrow{\text{perspective division}} \underbrace{\begin{bmatrix} xd \\ yd \\ zd \\ -z \end{bmatrix}}_{= q} = \underbrace{\begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}_{p}$$

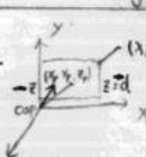
$$q' = \begin{bmatrix} \frac{xd}{-z} \\ \frac{yd}{-z} \\ -d \\ 1 \end{bmatrix} = \cancel{\begin{bmatrix} x_p \\ \\ -d \\ 1 \end{bmatrix}} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

perspective division.

Remark: Homogeneous coord. system. If $w \neq 0$: $\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \equiv \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}$ equivalent representation but different ~~comp~~ computation cost → perspective division.

Vertices → [Model-View] → [Projection] → [Perspective division] →

Simple Orthogonal Projection (back $\perp$ z axis)

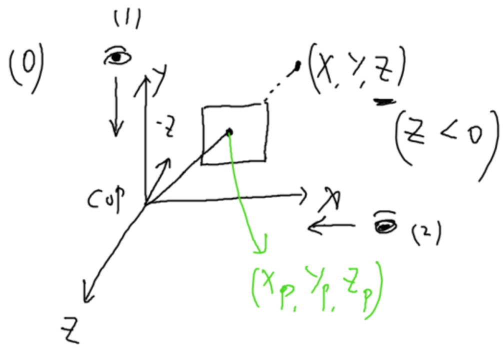


Use camera frame but with ~~???~~ camera default dir (facing -z)

$\left(\begin{matrix} d \geq 0 \\ z < 0 \end{matrix} \right)$

$$\begin{cases} x_p = x \\ y_p = y \\ z_p = -d \end{cases} \quad \begin{bmatrix} x \\ y \\ -d \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}}_{q} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}_{p}$$

(value of d does not matter as long as $d \geq 0$)

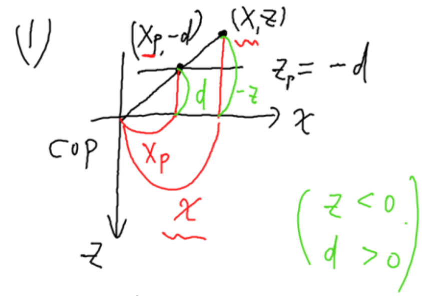


3D View

View plane: $z_p = -d$ $(d > 0)$

$z < 0$.

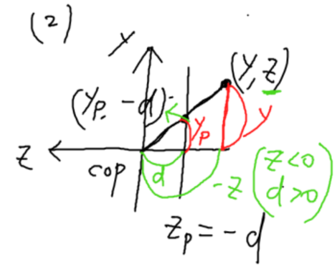
$\therefore (x_p, y_p, z_p) = \left(\frac{x\cdot d}{-z}, \frac{y\cdot d}{-z}, -d\right)$ ※



Top View

$$\frac{x_p}{x} = \frac{d}{-z}$$

$$\Rightarrow x_p = \frac{x\cdot d}{-z}$$



Side View

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

$$\Rightarrow y_p = \frac{y\cdot d}{-z}$$

Now Express Perspective Projection by Matrix Multiplication:

$$q = \begin{bmatrix} \frac{x\cdot d}{-z} \\ \frac{y\cdot d}{-z} \\ -d \\ 1 \end{bmatrix} = \overset{M}{\begin{bmatrix} A & B & C & D \\ & & & \\ & & & \\ & & & \end{bmatrix}} \overset{p}{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}$$

$$\frac{x\cdot d}{-z} = Ax + By + Cz + D$$

☆ We want: A, B, C, D be constants, independent of x, y, z.

$\Rightarrow$ NOT possible!!

☆ Key Idea: Use Homogeneous Coord. System.

If $w \neq 0$: $\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \overset{\text{Perspective Division}}{\equiv} \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}$

$w \neq 1$: pt. with coord. $(x/w, y/w, z/w)$

$$\begin{bmatrix} \frac{x\cdot d}{-z} \\ \frac{y\cdot d}{-z} \\ -d \\ 1\end{bmatrix}_{q} \equiv \begin{bmatrix} \frac{x\cdot d}{(-z)}\cdot(-z) \\ \frac{y\cdot d}{(-z)}\cdot(-z) \\ -d\cdot(-z) \\ 1\cdot(-z)\end{bmatrix} = \begin{bmatrix} x\cdot d \\ y\cdot d \\ z\cdot d \\ -z\end{bmatrix} = \underbrace{\begin{bmatrix} d&0&0&0\\0&d&0&0\\0&0&d&0\\0&0&-1&0\end{bmatrix}}_{M} \underbrace{\begin{bmatrix}x\\y\\z\\1\end{bmatrix}}_{p}$$

Perspective Division

Desired Matrix

* For any point p:

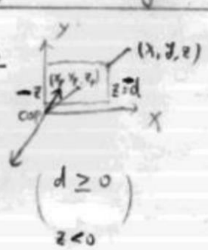
① $M\cdot p$

② Perform Perspective Division to get the projection point q.

Next, we look at Orthogonal Projection:

Simple Orthogonal Projection (look ⊥ z axis).

Use camera frame but with camera default dir (facing $-z$)



$$\begin{cases} x_p = x \\ y_p = y \\ z_p = -d \end{cases}$$

$$\begin{bmatrix} x \\ y \\ -d \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}}_{q} = \underbrace{\begin{bmatrix} 1&0&0&0\\0&1&0&0\\0&0&0&-d\\0&0&0&1\end{bmatrix}}_{M} \underbrace{\begin{bmatrix}x\\y\\z\\1\end{bmatrix}}_{p}$$

(value of d does not matter as long as $d \geq 0$)

Note: There is no division involved.

(Therefore, we call the division in the perspective projection perspective division.)

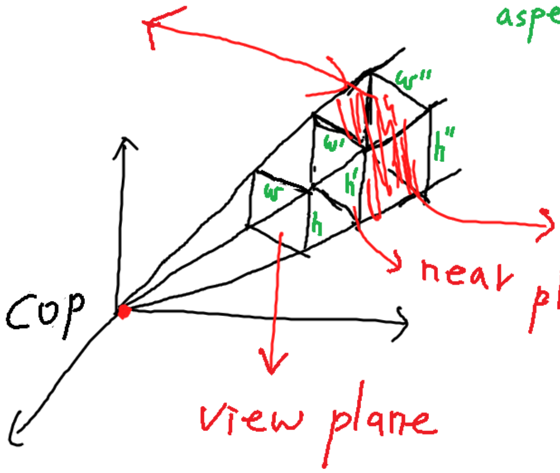
In Perspective Projection:

View { Volume / Frustum

clipping op:

* objects outside the view volume are clipped out.

aspect ratio $\frac{w}{h} = \frac{w'}{h'} = \frac{w''}{h''}$



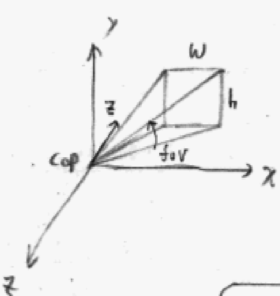
→ Perspective (;) camera frame

② gluPerspective(fovy, aspect, near, far)

Easier to use than glFrustum( )

field of view degree $\in [0°, 180°)$

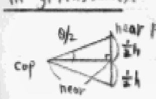
again must be > 0 (view from cop)



aspect = $\frac{w}{h}$

Note: cop is in the center of graphics window & view volume is symmetric

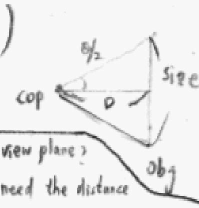
Recover the same info as in glFrustum():



$\tan\frac{\theta}{2} = \frac{h/2}{near} \Rightarrow h$

aspect = $\frac{w}{h} \Rightarrow w$

field of view: (side view)



$\tan\frac{\theta}{2} = \frac{size/2}{D}$

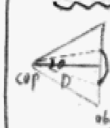
θ = fovy.

Q: Where is the view plane?

A: In theory, we need the distance from cop to the view plane i.e. $z_p = d > 0$ on P.3 (In the projection matrix M, d is used in M)
But in OpenGL, d is implemented as a default value and we don't/can't change it (actually no need to change it)

→ In OpenGL, view plane = near clipping plane.

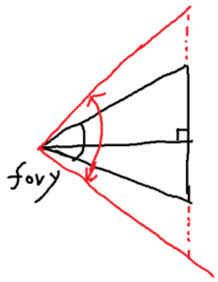
After specifying obj and cop in the world, D is fixed (Also, window is fixed)



⇒ θ↑ more objects are in the scene, objects appear small i.e. zoom out.

θ↓ : zoom in

Perspective (fovy, aspect, near, far)



When fovy increases, more objects enter the fixed-sized graphics window.

⇒ objects appear smaller

⇒ zoom out effect.

Similarly, when fovy ⇓ ⇒ objects appear bigger

⇒ zoom in effect.

Perspective Normalization: Transform (distort) the entire scene so that the view volume is (axis-parallel) and the "same" perspective projection result is still kept

(except that the aspect ratio of the view plane becomes (1)
(It will be (restored) later in (viewport) transformation.))

⇒ (clipping) becomes very simple