컴퓨터 그래픽스 (목) PI 2차시

3D Transformation
&
OpenGL
(2)

5:05PM에 시작됩니다
입장 후 채팅창에 학번/이름 작성 부탁드립니다.
감사합니다.

3D Transformation

- 이론
- 수식 & 계산
- 시험

예) 정렬 알고리즘에는

버블정렬, 퀵정렬, 병합정렬… 등이 있고,
구현 아이디어는 현재 인덱스의 값과 다음인덱스의 값을
비교해서…

시간복잡도는 …

OpenGL

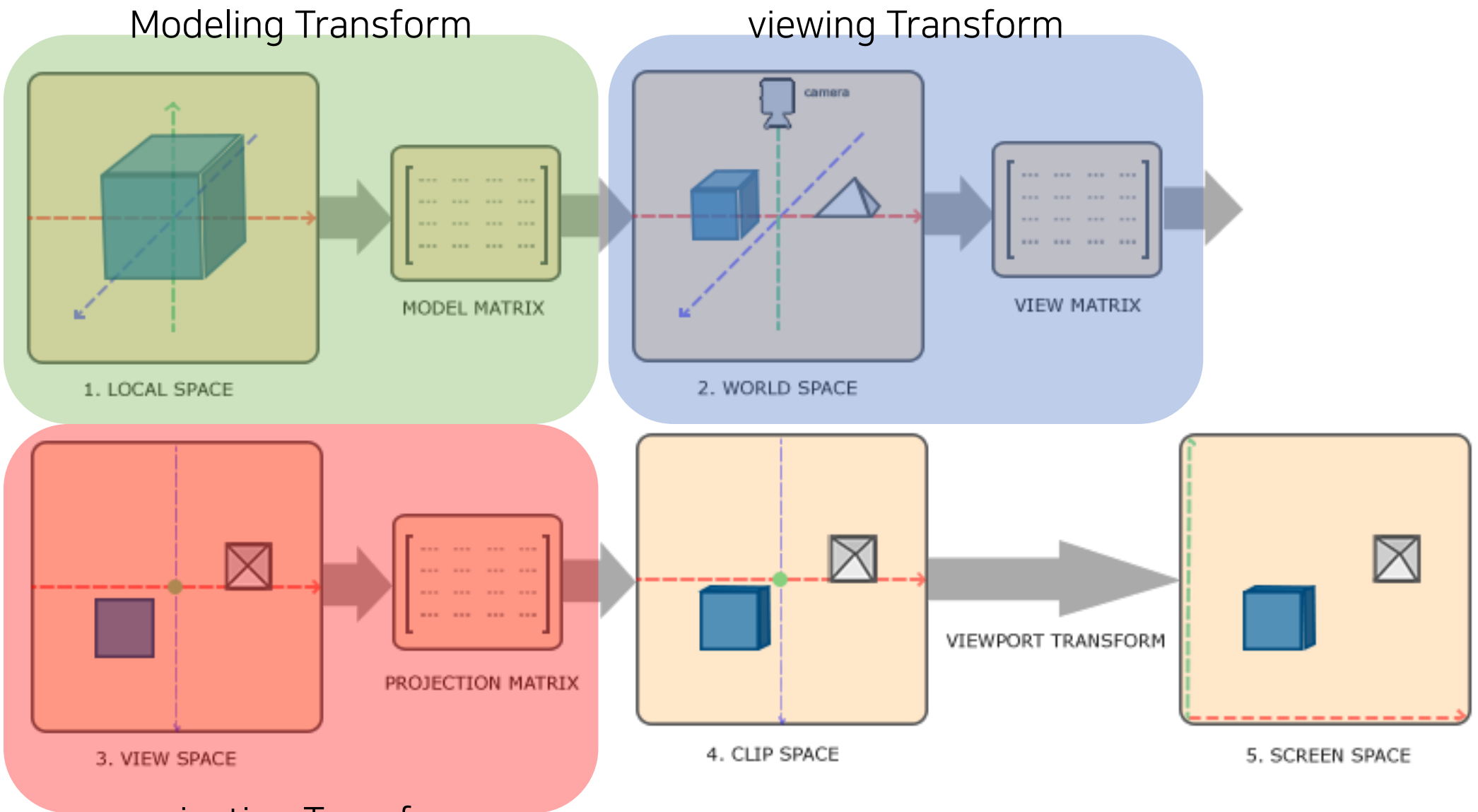
- 구현 (함수 사용)
- 과제

예) 해당 언어 라이브러리에서 제공해주는 sort함수 사용

```
sort(v.begin(), v.end(), cmp) //퀵정렬
```

목차

	이론 3D Transformation	코딩 OpenGL
Modeling Transformation	X	O
Viewing Transformation	O	O
Projection Transformation	O	O


Modeling Transform
viewing Transform
1. LOCAL SPACE
MODEL MATRIX
camera
2. WORLD SPACE
VIEW MATRIX
3. VIEW SPACE
PROJECTION MATRIX
4. CLIP SPACE
VIEWPORT TRANSFORM
5. SCREEN SPACE
projection Transform

Modeling Transformation

모델을 회전, 이동, 크기변경등의 transform을 수행하는 것

translate
rotate
scale

Matrix 선언

3차원 공간(3x3)인데, 차원을 확장했었음

4x4 Identity matrix

```
glm :: mat4 model = glm::mat4(1.0f)
```

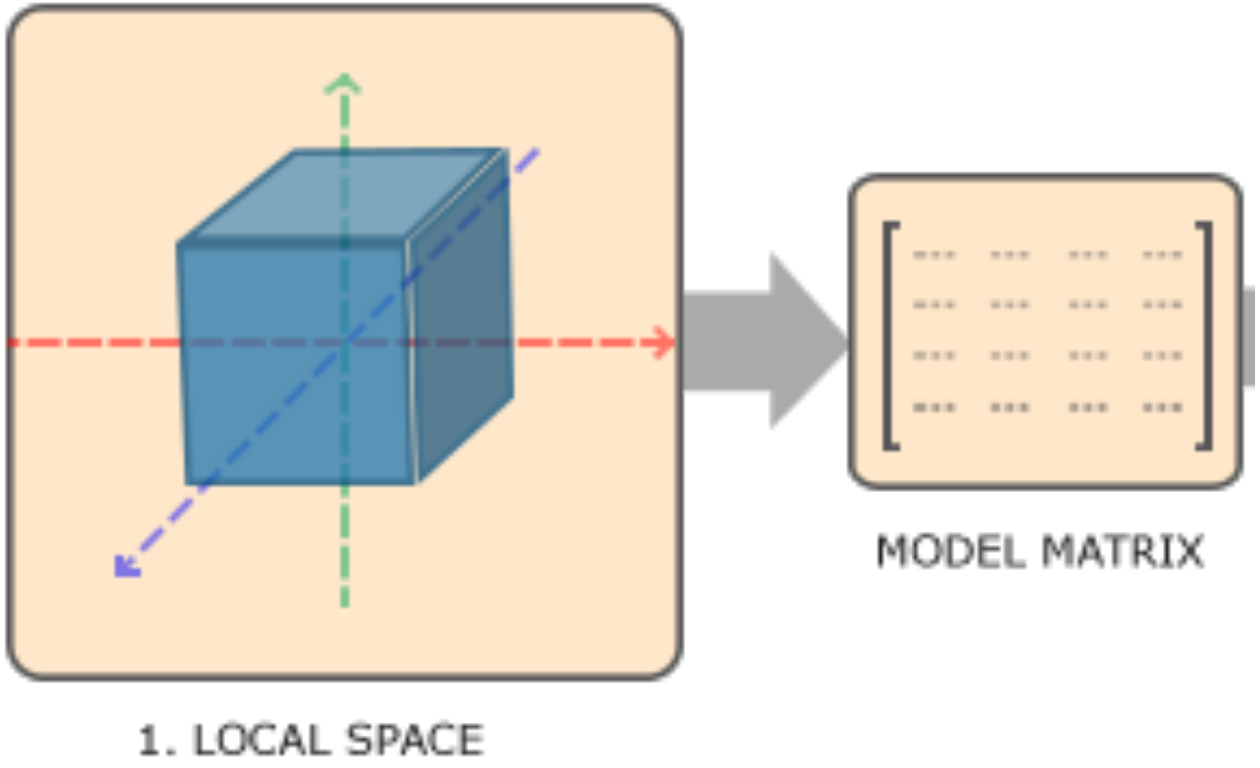
Modeling Transformation

모델을 회전, 이동, 크기변경등의 transform을 수행하는 것

각 모델이 보유한 Modeling Coordinate System이라는 좌표계에서

World Coordinate System이라는 공통된 좌표계로 전부 좌표를 재 계산하는 과정

→ 그 과정에서 모든 모델은 각각 위치이동, 회전하는 것을 반영




```
glm :: translate
glm :: rotate
glm :: scale
```

Modeling Transformation – Translation

```
model = glm :: translate(model,
                         glm::vec3(x, y , z))
```

적용할 모델의 matrix

각 축으로 얼마나 이동할 것인지

ex)

```
glm::vec3( 0.0f, 0.0f , -3.0f)
```

-z축 방향으로 3만큼 이동

Modeling Transformation – Rotation

```
model = glm :: rotate(
                    model,

                    회전 angle,
```

회전각, glm::radians를 쓰는 것을 추천

```
                    glm::vec3(x, y , z))
```

점(x, y, z) 와 원점(0, 0, 0)을 잇는 축이 회전축

Modeling Transformation – Scaling

```
model = glm :: scale(model,
                     glm::vec3(x, y , z))
```

줄일 비율

ex)

```
glm::vec3( 0.1f, 0.1f , 0.1f)
```

각각을 1/10씩 줄임

* 중심은 같고 크기만 변화

여기서 중요한 것은 " 순서 "

Transformation 연산

여러 transformation을 수행하는 경우, 가장 last로 작성된 코드가
가장 뒤 쪽에 추가되어 가장 먼저 적용됨

① rotation → translation

same

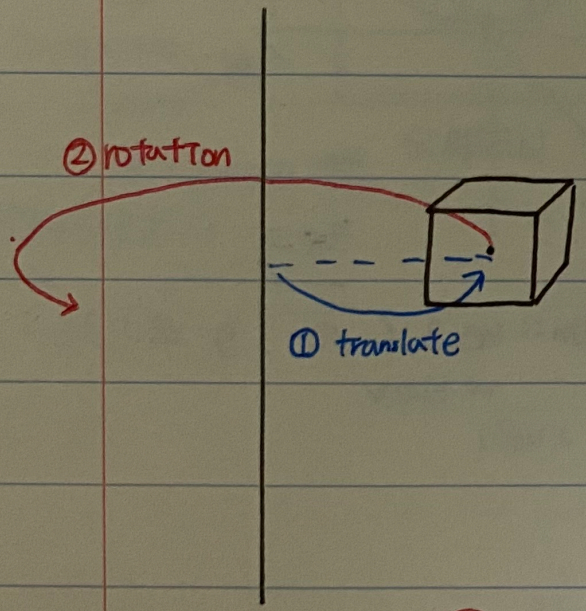
model = glm::rotate(model, —)

model = glm::translate(model, —)

model = model * rotate * translate

② ① 순서

translation이 먼저 발생! → 그 후 rotate



느낄 점: 공전!

② translation → rotation

same

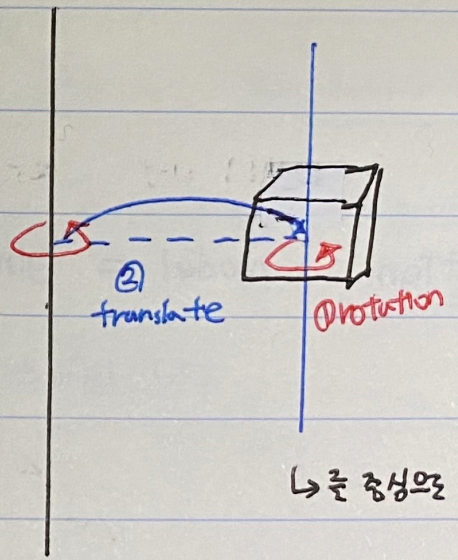
model = glm::translate(model, —)

model = glm::rotate(model, —)

model = model * translate * rotate

① ② 순서

rotation이 먼저발생 → 그 후 translate



느낄 점: 자전!

Viewing Transformation

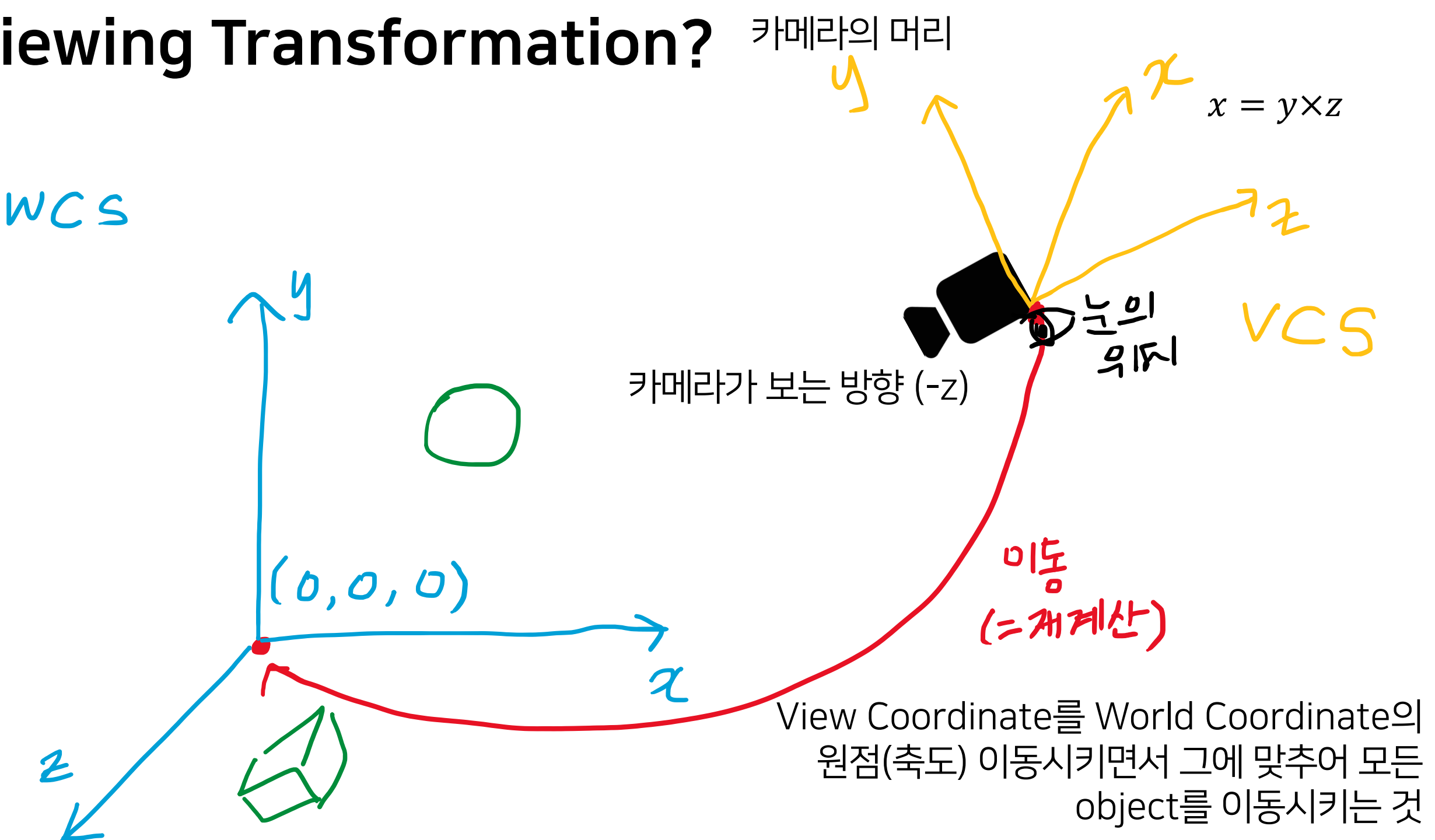
Viewing Coordinate System이라는 카메라 기준의 좌표계에 맞추어
WCS기준으로 되어있는 좌표를 VCS 기준에 맞추어 변경(재계산)하는 작업

1. Viewing Transformation 이란?
2. Viewing Coordinate($\mathbf{x_{view}}, \mathbf{y_{view}}, \mathbf{z_{view}}$) 구하기
3. Viewing Transformation 수행

Viewing Transformation

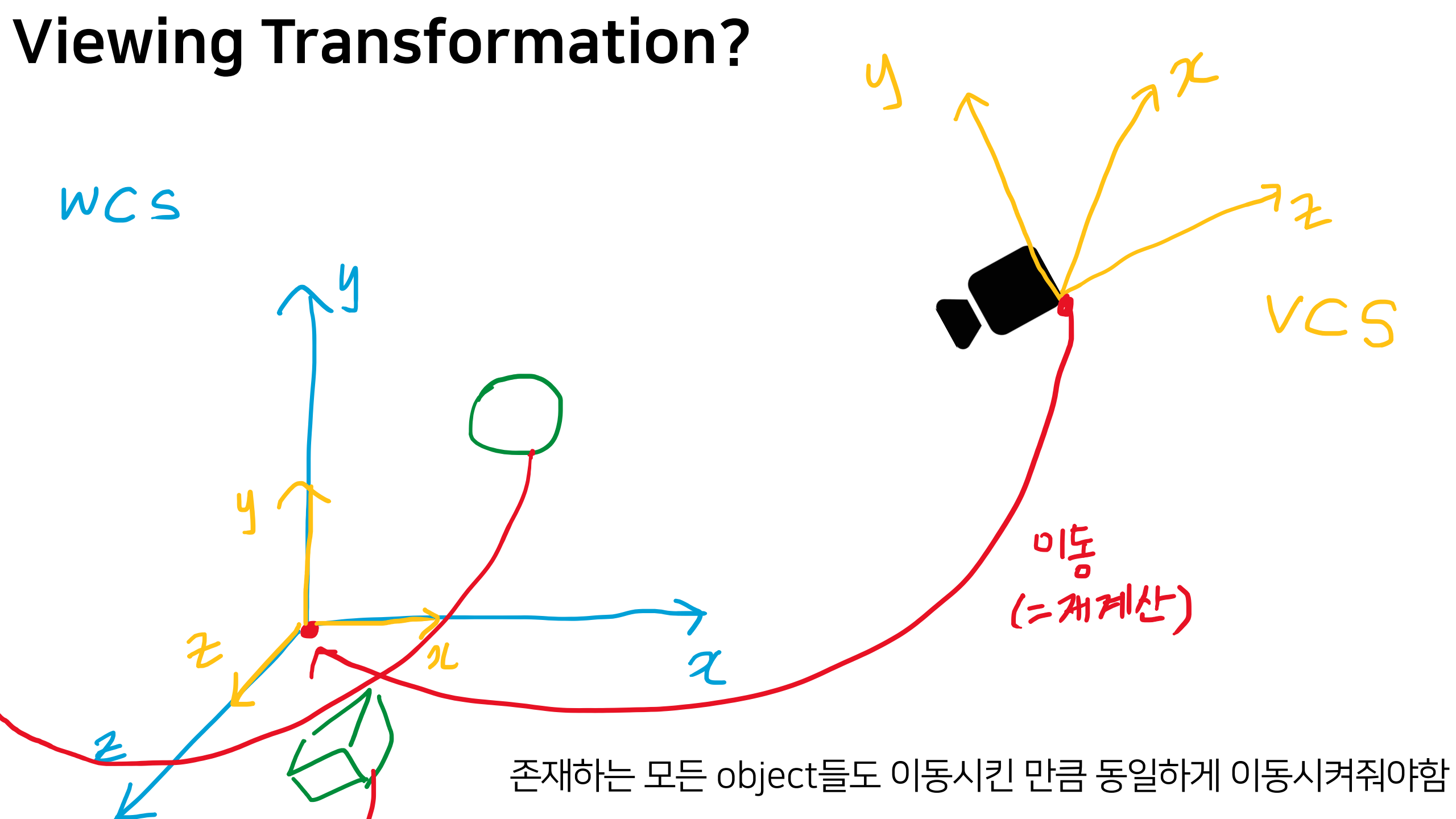
1. Viewing Transformation 이란?
2. Viewing Coordinate($x_{view}, y_{view}, z_{view}$) 구하기
3. Viewing Transformation 수행

Viewing Transformation?



View Coordinate를 World Coordinate의 원점(축도) 이동시키면서 그에 맞추어 모든 object를 이동시키는 것

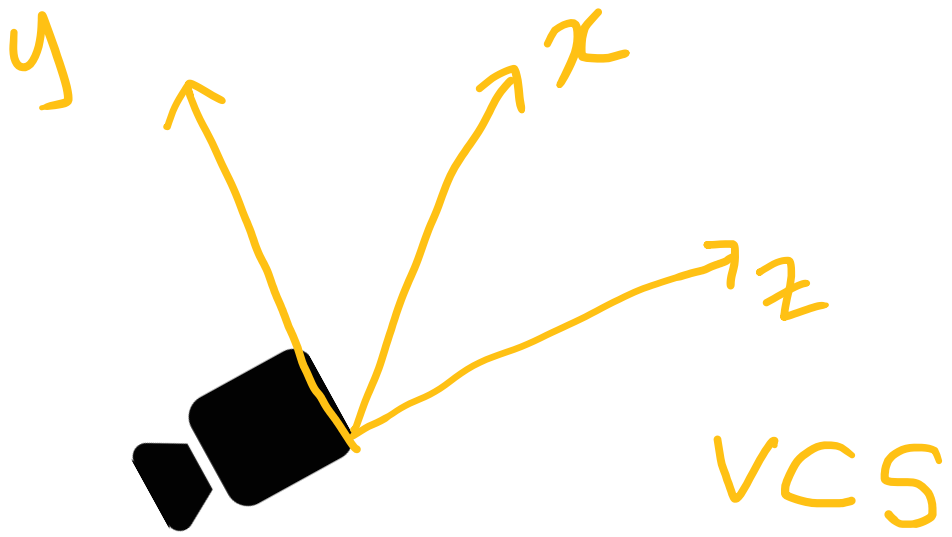
Viewing Transformation?



존재하는 모든 object들도 이동시킨 만큼 동일하게 이동시켜줘야함

Viewing Coordinate 구하기

view coordinate 의 x, y, z는 어떻게 정해지는가?

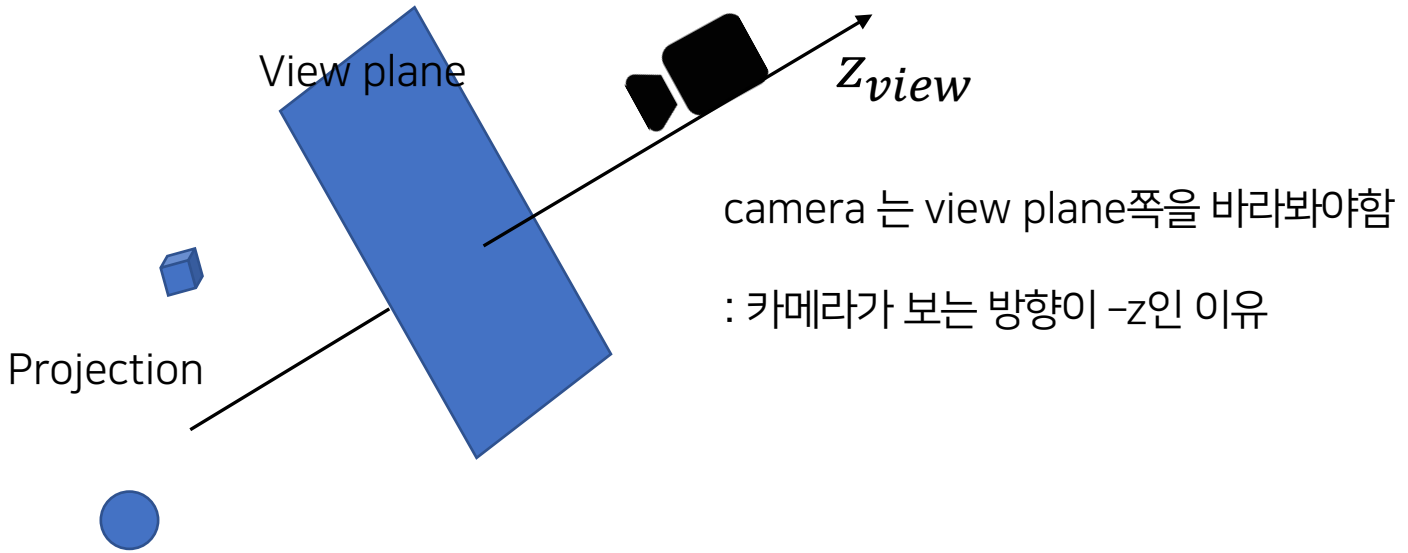


Viewing Coordinate 구하기

z_{view} : View Plane Normal

최종적으로 object들이 projection될 view plane 수직방향으로의 vector

그 벡터의 단위 벡터를 n = z_{view}



camera 는 view plane쪽을 바라봐야함

: 카메라가 보는 방향이 –z인 이유

Viewing Coordinate 구하기

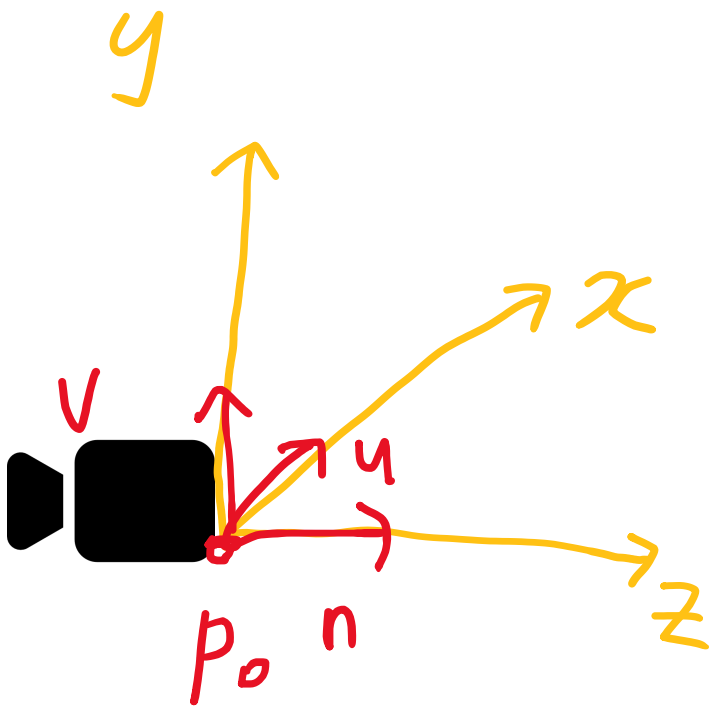
y_{view} : View – up이라고 불리는 카메라의 머리 방향에 해당 -> 단위 벡터는 v

x_{view} : x_{view} 방향으로의 단위 벡터 u

$$u = v \times n$$

fleming's right hand rule사용 (u, v 가 정해지면 자동으로 정해짐)

Viewing Coordinate 정리



viewing Coordinate

v = (0, 1, 0)
u = (1, 0, 0)
n = (0, 0, 1)
P0 = (0, 0, 0) 으로 만들면서,

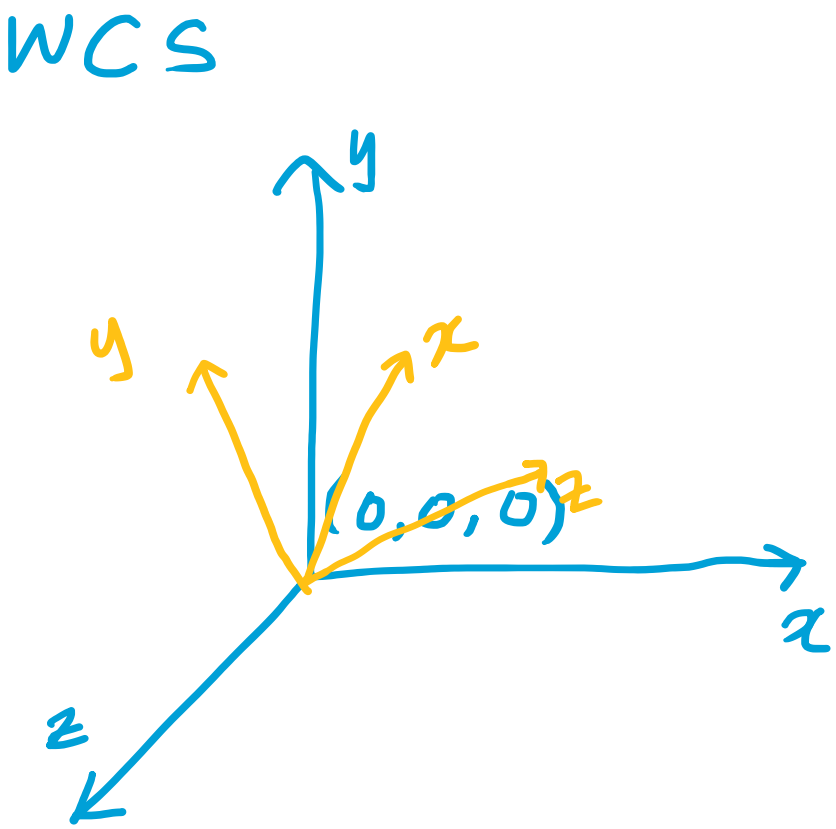
이들을 이렇게 만들어 줄 수 있는 행렬을 찾아

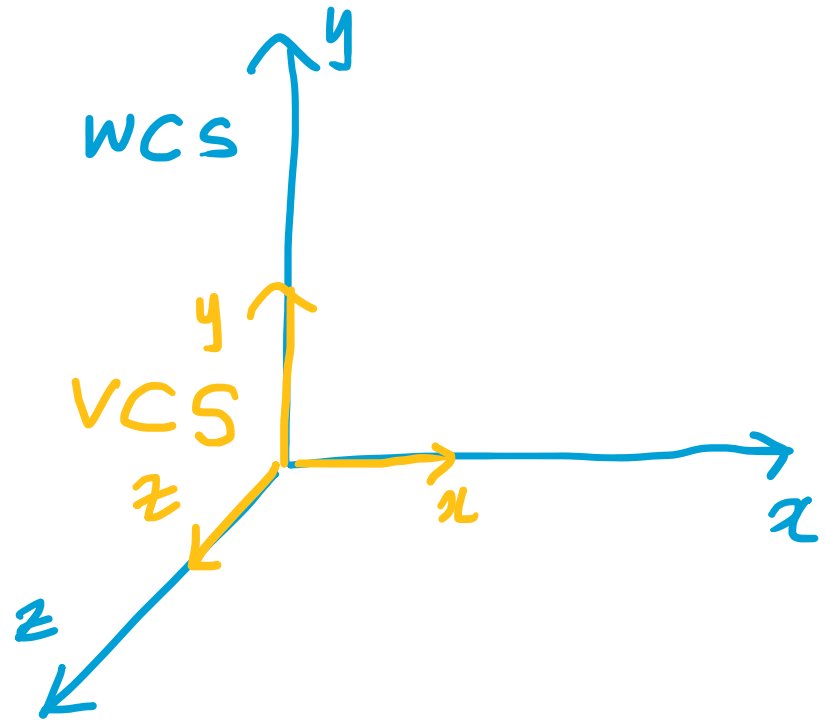
다른 모든 object에도 똑같이 적용해 주기

=> 해당 행렬을 구해보자

Viewing Transformation

1 . View Coordinate를 World Coordinate의 (0,0,0)의 기본 좌표계로 맞춰지게 하는 행렬을 구하자





1) Translate

VC origin을 WC origin으로 이동시키기

2) Rotate

u,v, n 이 x,y,z(wc) 와 일치하도록 회전시켜서 둘을 같게 만듬

Viewing Transformation

1) Translate

VC origin을 WC origin으로 이동시키기

$P_0(x_0, y_0, z_0)$이면, $O(0,0,0)$로 이동시키기 위한 행렬 T는

$$\begin{matrix} 0 \\ 0 \\ 0 \\ 1 \end{matrix} \quad \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{matrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{matrix}$$

O = T · P_0

Viewing Transformation

2) Rotate

u, v, n 이 x(1,0,0), y(0,1,0), z(0,0,1)(WC) 와 일치하도록 회전시켜서 둘을 같게 만듦

$$u=\begin{bmatrix}u_x\\u_y\\u_z\end{bmatrix}\quad v=\begin{bmatrix}v_x\\v_y\\v_z\end{bmatrix}\quad n=\begin{bmatrix}n_x\\n_y\\n_z\end{bmatrix}\quad \text{into}\quad x=\begin{bmatrix}1\\0\\0\end{bmatrix}\quad y=\begin{bmatrix}0\\1\\0\end{bmatrix}\quad z=\begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}=R\cdot\begin{bmatrix}u_x&v_x&n_x&0\\u_y&v_y&n_y&0\\u_z&v_z&n_z&0\\0&0&0&1\end{bmatrix}$$

Viewing Transformation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = R \cdot \begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

참

I 행렬 이므로, 두 행렬의 곱이 I matrix이면,

$$A \cdot B = I \; then,$$

$$A = B^{-1} = B^{T}$$

$$\begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \xrightarrow{\text{Transpose}} \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = R$$

Viewing Transformation

1 Translate , 2 Rotate를 차례대로 수행하면, $\mathrm{R} \cdot T = M$이다.
즉, M이라는 matrix 를 WC상의 모든 object에 곱하면 VC기준으로 좌표계가 변경되므로, viewing Transformation이 완료된다.

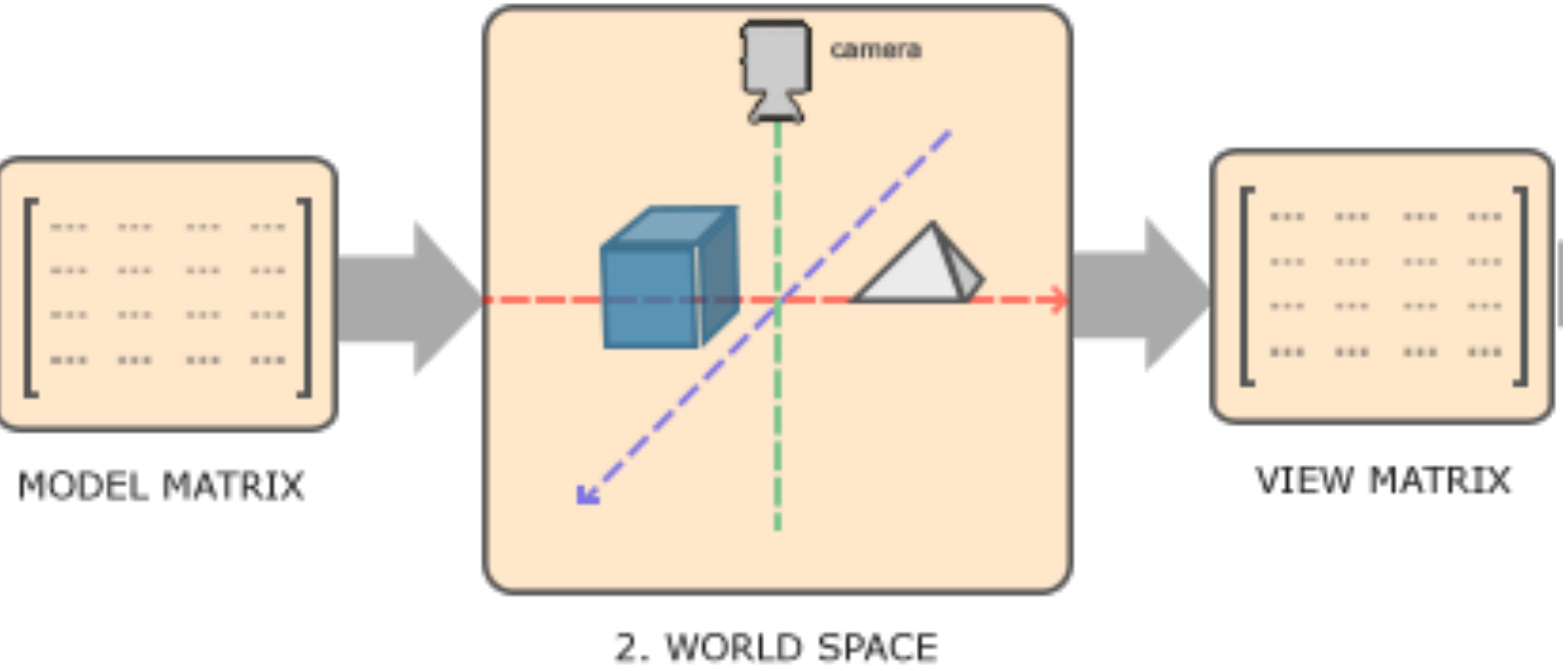
이때, 행렬 M을 $M_{wc,vc}$**라고 한다** .

$$\mathbf{M}_{WC,\,VC} = \mathbf{R} \cdot \mathbf{T}$$

$$= \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{P}_0 \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{P}_0 \\ n_x & n_y & n_z & -\mathbf{n} \cdot \mathbf{P}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Transformation

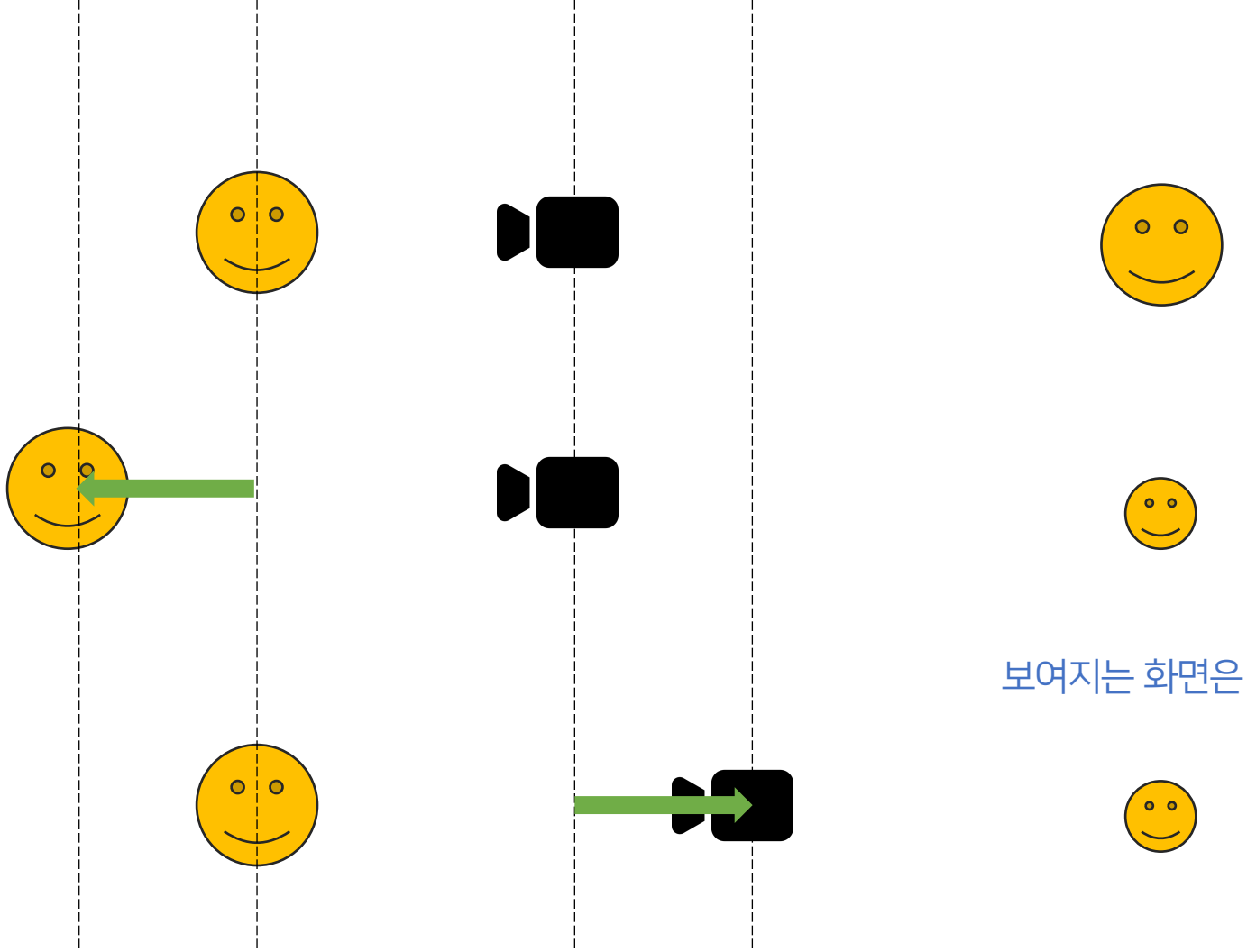
Viewing Coordinate System이라는 카메라 기준의 좌표계에 맞추어
WCS기준으로 되어있는 좌표를 VCS 기준에 맞추어 변경(재계산)하는 작업




```
glm :: lookAt
```

참고

사용자가 봤을 때 물체가 이동된 것 처럼 보이게 하는 방법



보여지는 화면은 같다

1. “물체”를 이동시키는 방법

   modeling transform을 이용한 편법

2. “카메라”를 이동시키는 방법

   진정한 viewing transform

3. <camera.h>를 이용하는 방법

learnopengl 작성자가 만든 기능 (opengl_gettingstarted_1 29p참고)

Matrix 선언

3차원 공간(3x3)인데, 차원을 확장했었음

4x4 Identity matrix

```
glm :: mat4 view = glm::mat4(1.0f)
```

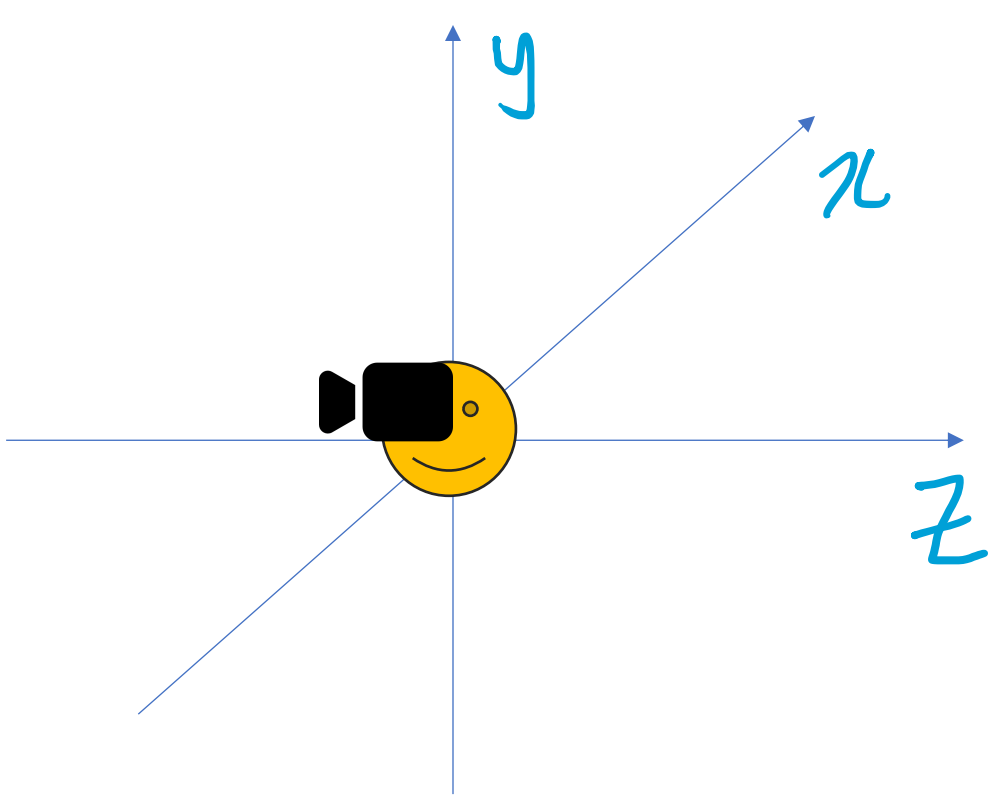
1. 물체를 이동시키는 방법

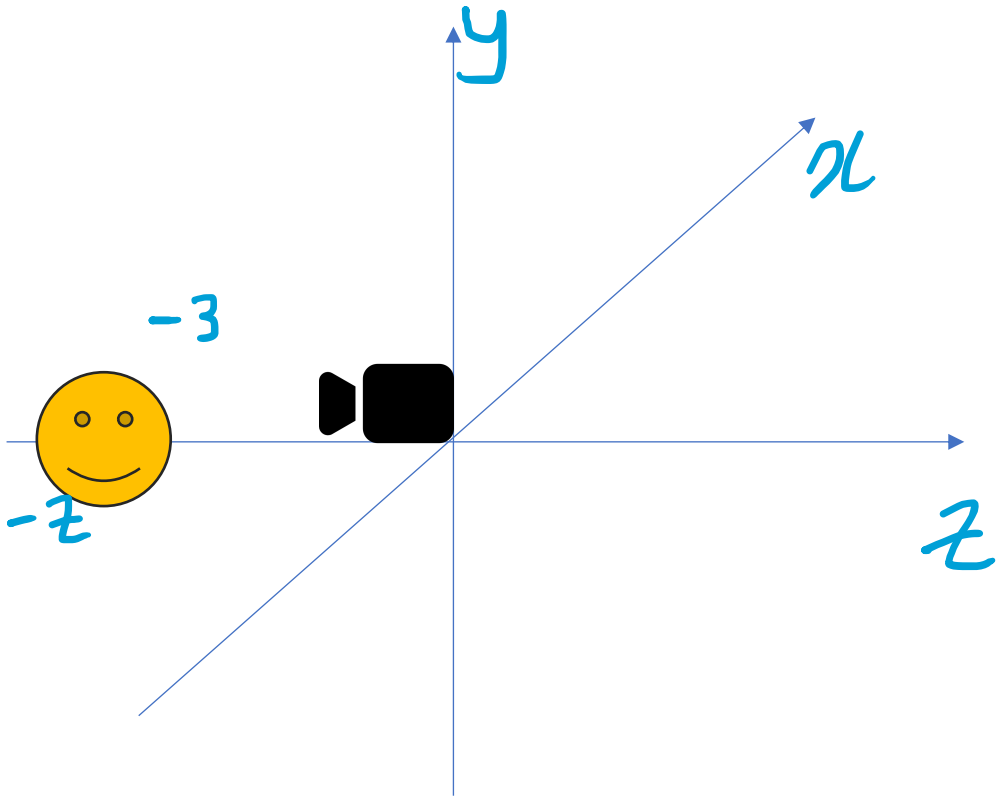
```
view = glm :: translate(view, glm::vec3(x, y , z))
```

적용할 모델의 matrix

각 축으로 얼마나 이동할 것인지

ex) `glm::vec3( 0.0f, 0.0f , -3.0f)`

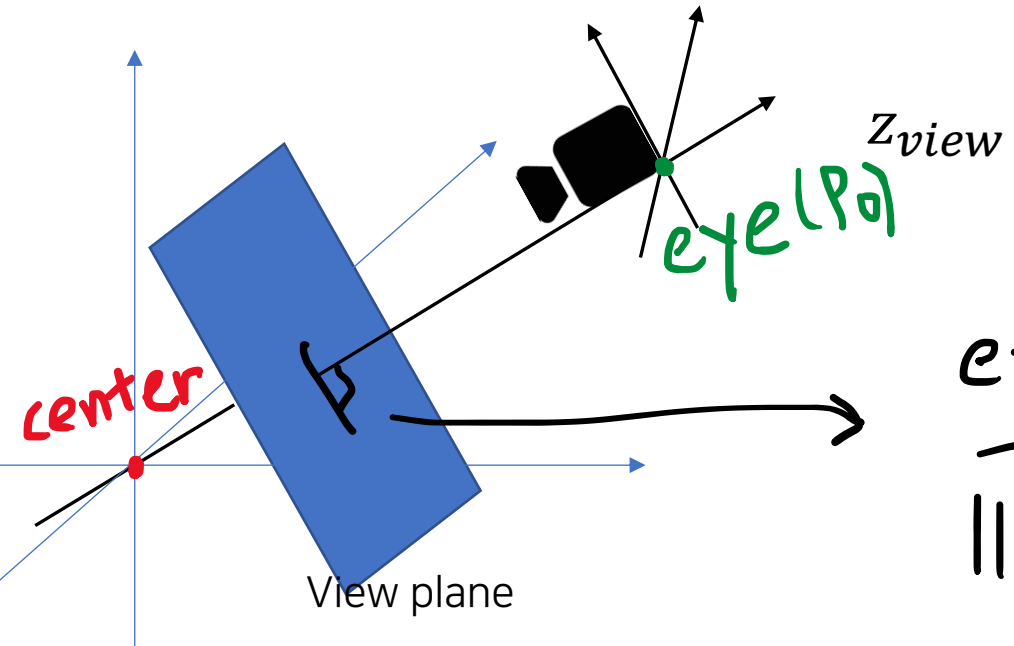




2. 카메라를 이동시키는 방법

```
view = glm :: lookAt(
            glm::vec3(eye.x, eye.y , eye.z),

            glm::vec3(center.x, center.y, center.z),
                                  카메라가 쳐다보는 점
            glm::vec3(up.x, up.y, up.z)
                    )
                                  up vector(카메라의 머리 방향 == v )
```



$$\frac{eye - center}{\|eye - center\|}$$

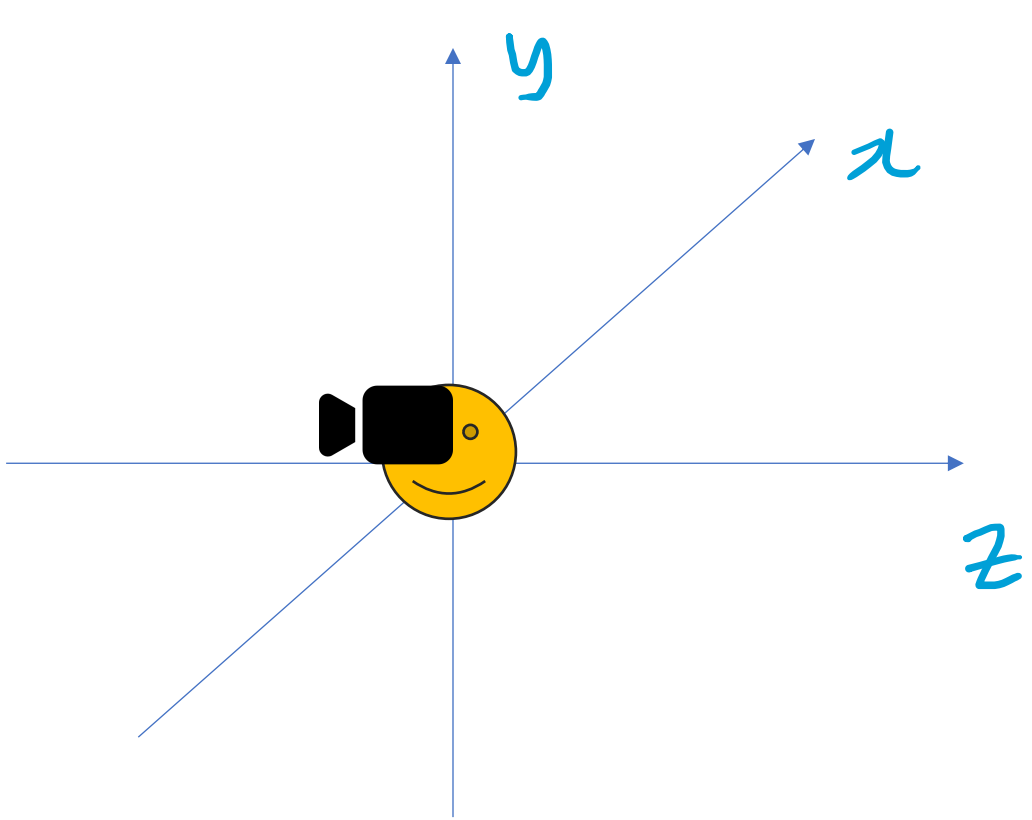
center에서 eye쪽으로 향하는 단위 벡터

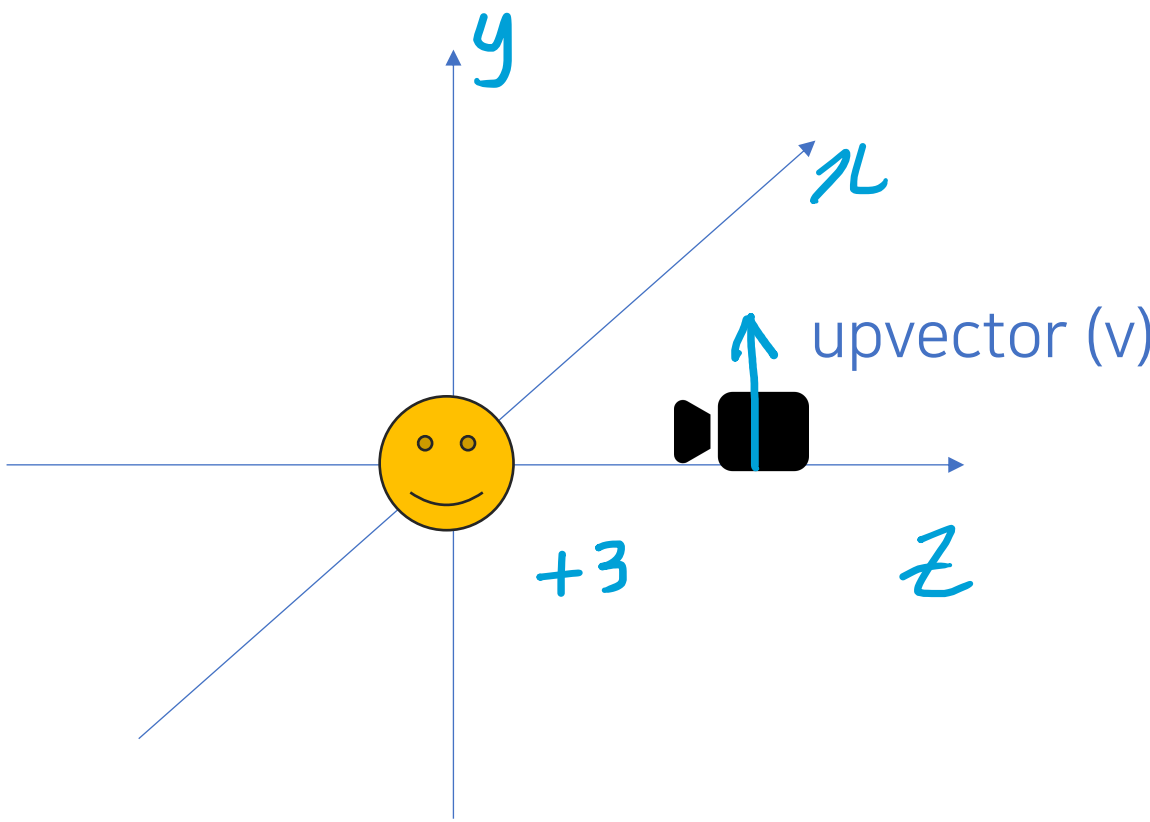
(view plane에 수직임! 즉, n)

2. 카메라를 이동시키는 방법

```
view = glm :: lookAt( glm::vec3(0.0f, 0.0f , 3.0f),
                      glm::vec3(0.0f, 0.0f, 0.0f),
                      glm::vec3(0.0f, 1.0f, 0.0f)
                      )
```

ex)





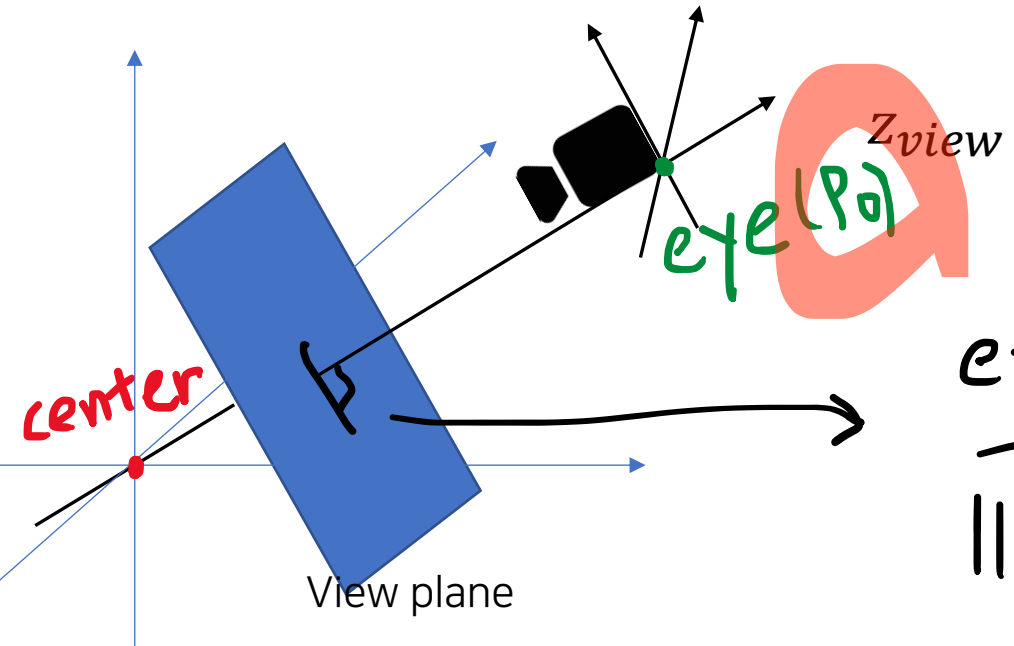
2. 카메라를 이동시키는 방법

```
view = glm :: lookAt(
                glm::vec3(eye.x, eye.y , eye.z),

                glm::vec3(center.x, center.y, center.z),

                glm::vec3(up.x, up.y, up.z)
                    )
```

up vector(카메라의 머리 방향 == v)



$$\frac{eye - center}{\|eye - center\|}$$

center에서 eye쪽으로 향하는 단위 벡터

(view plane에 수직임! 즉, n)

2. 카메라를 이동시키는 방법

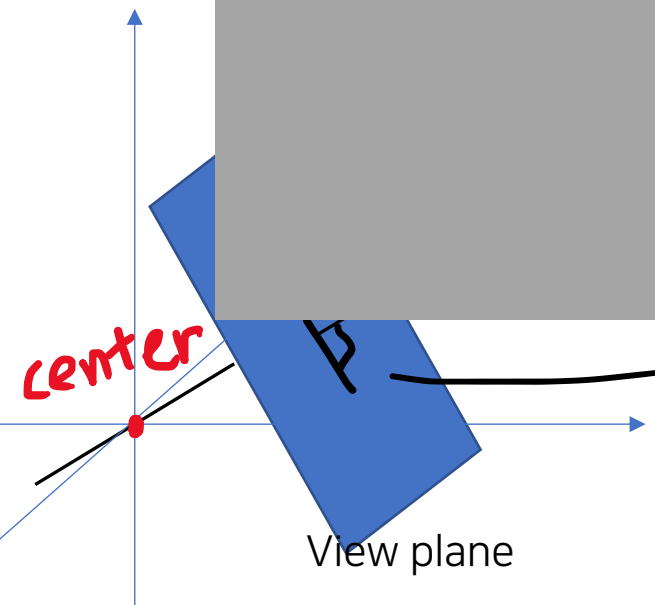
```
view = glm :: lookAt(
                glm::vec3(eye.x, eye.y , eye.z),
```

.z),

glm:: lookAt함수를 통해 P0, n , v 를 구할 수 있음
(u는 n,v를 이용하면 구할 수 있음)

= > Mwc,vc를 구할 수 있음!

= v)



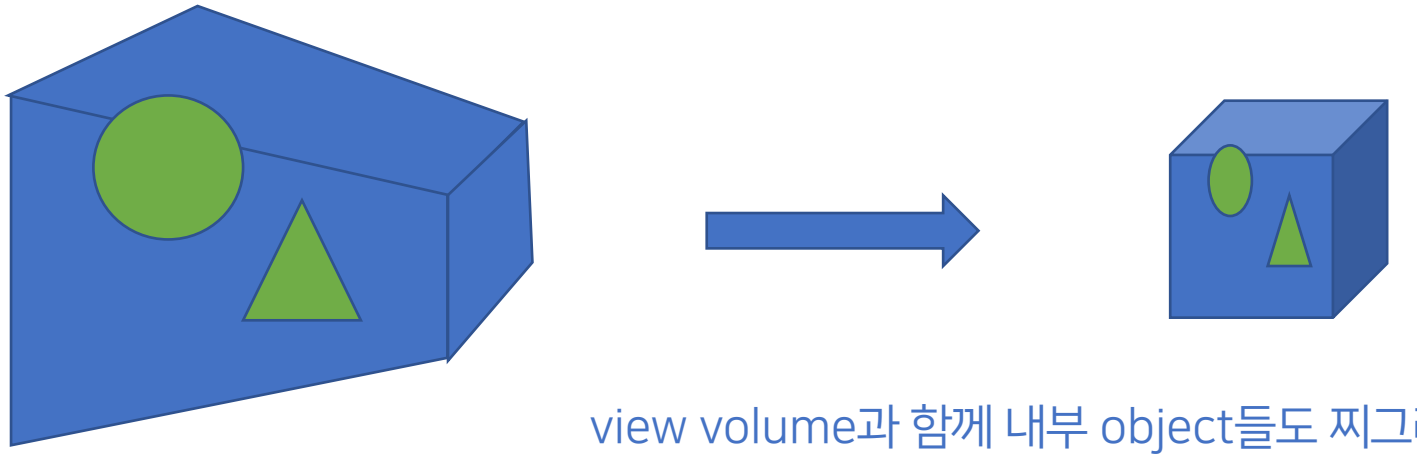
$$\frac{\text{eye-center}}{\|\text{eye-center}\|}$$

center에서 eye쪽으로 향하는 단위 벡터
(view plane에 수직임! 즉, n)

Projection Transformation

담기지 x부분은 clipping

VCS로 계산된 view volume에 담긴 object들을 2D(view plane)상에 투사 시키기 위해서 view volume을 조작하면서 내부의 object들도 같이 조작되는 과정



view volume과 함께 내부 object들도 찌그러짐 -> 좌표 재계산이 필요

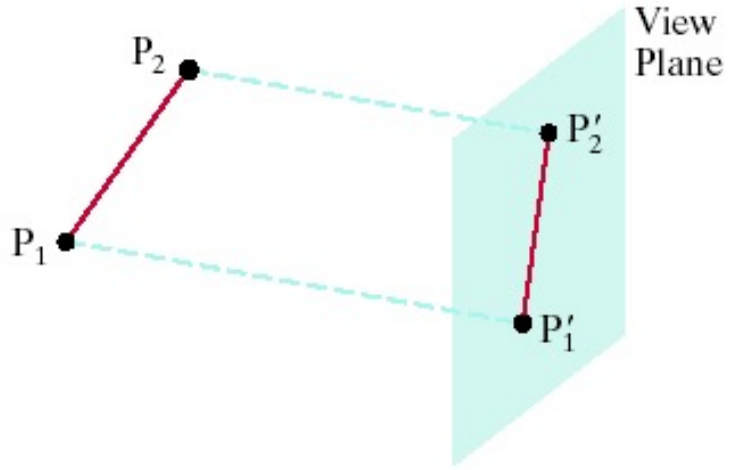
종류

Project의 방향 차이

1. Parallel Projection

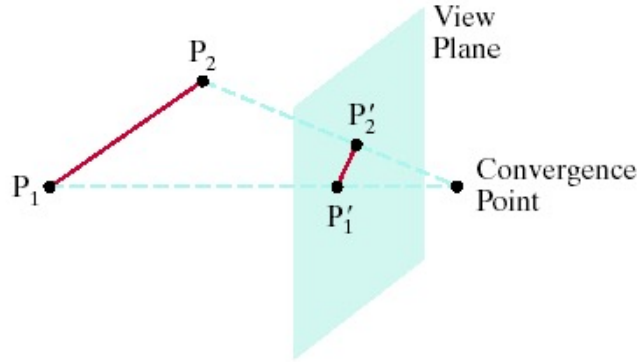
   - 평행한 line 보존 (Affine Transform)
   - orthographic (VPN과 평행)

   기계적 성질이 중요한 computer design, modeling등에 사용



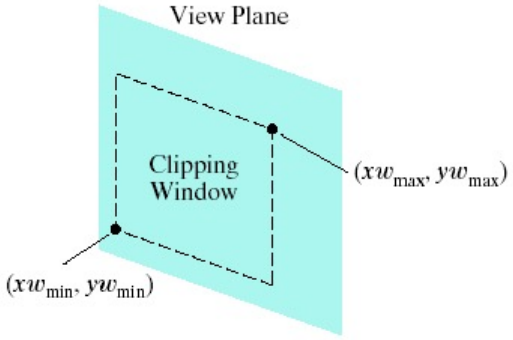
2. Perspective Projection

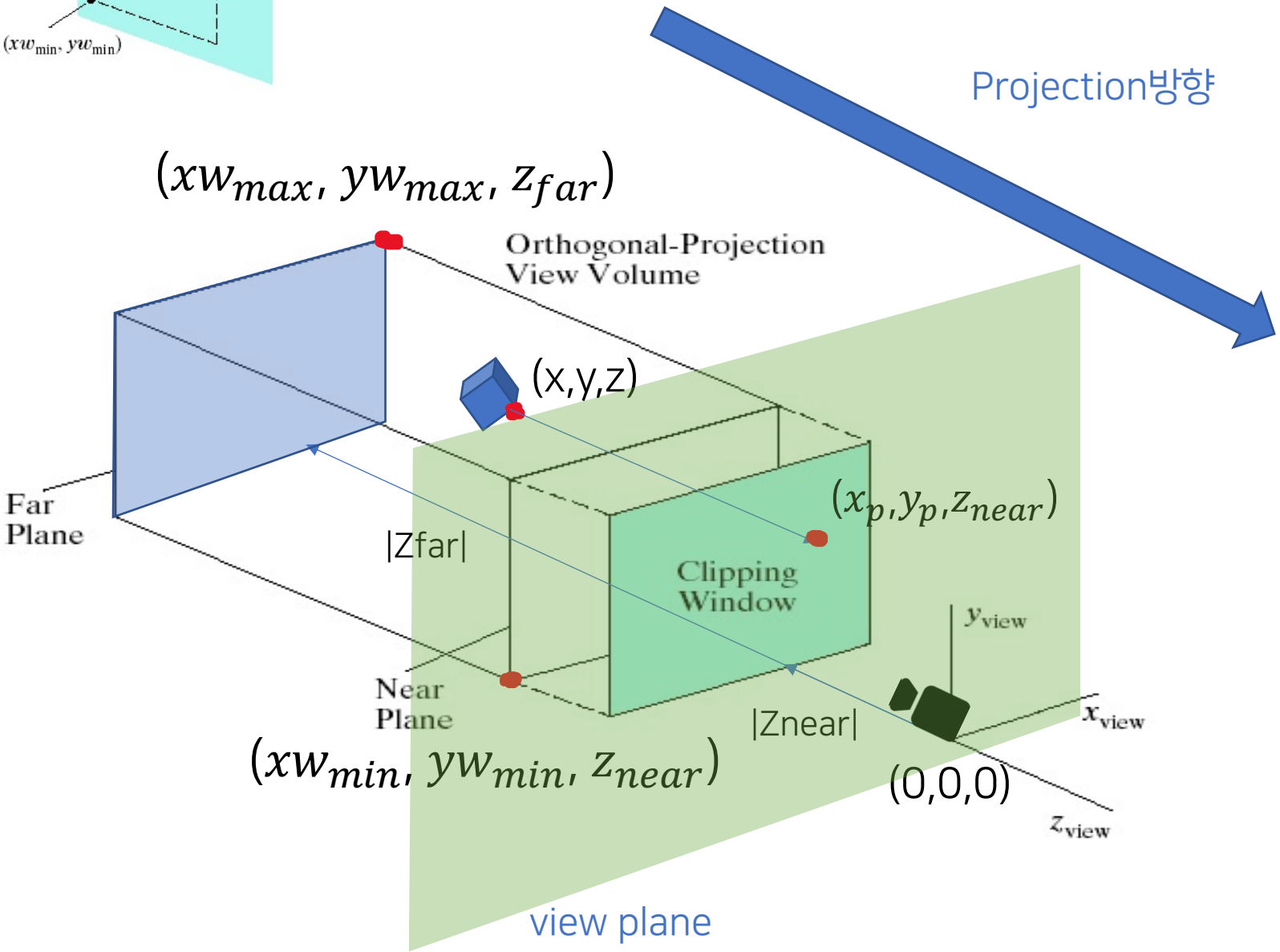
   - 평행한 line 보존 x
   - 가까운 것은 크게, 먼 것은 작게 보임 = Perspectivity 유지
     realistic rendering



Orthographic Projection

모든 projection 방향이 평행하며, VPN(view plane의 수직 방향)과도 평행한 경우





최종적으로 구해야 할 것은

2D(view plane)상의 x, y 점

$$x = x_p$$

$$y = y_p$$

Projection Transformation

Can you detect all the dfferences between the two scenes? There are 5 differences. See if you can find them all.

Projection Transformation

Can you detect all the dfferences between the two scenes? There are 5 differences. See if you can find them all.

Projection Transformation

P P'

" 현재 상태" 와 "목표 상태"는

$P' = M \cdot P$

1. 무엇이 다른가?

2. 각각 어떻게 바꾸어야 하는가?

M

Projection Transformation

목표 상태

현재 상태

무엇이 다른가?

- 신문 글자
- 머리카락 색
- 넥타이 색
- 시계 색
- 손수건 유무

Projection Transformation

P'
목표 상태

P
현재 상태

M
각각 어떻게 바꾸어야 하는가?

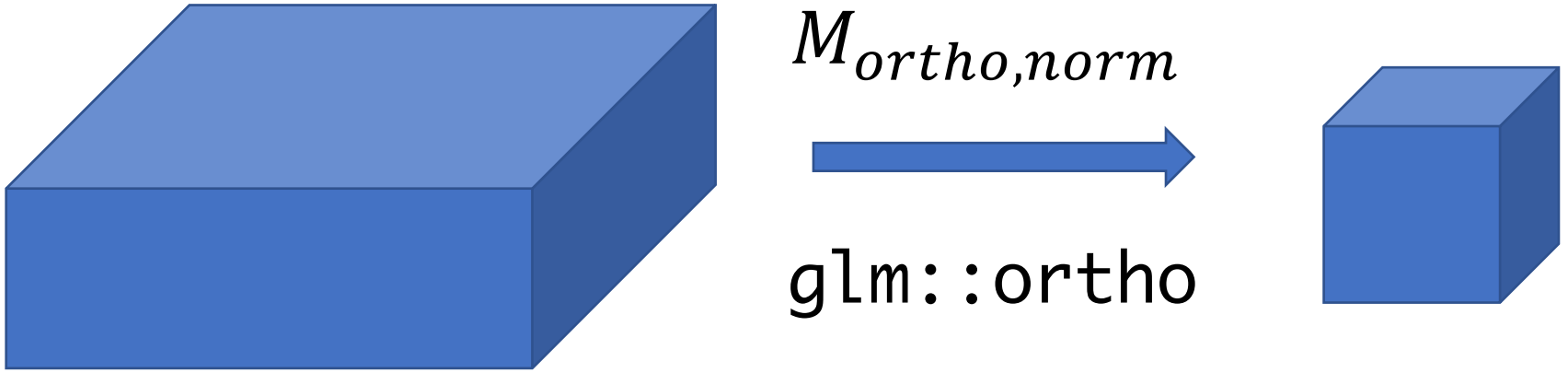
- 신문 글자
  : E와 M사이 "R" 추가
- 머리카락 색
  : 붉은색 -> 갈색

...

$P' = M \cdot P$

Parallel Projection

= Orthographic Projection

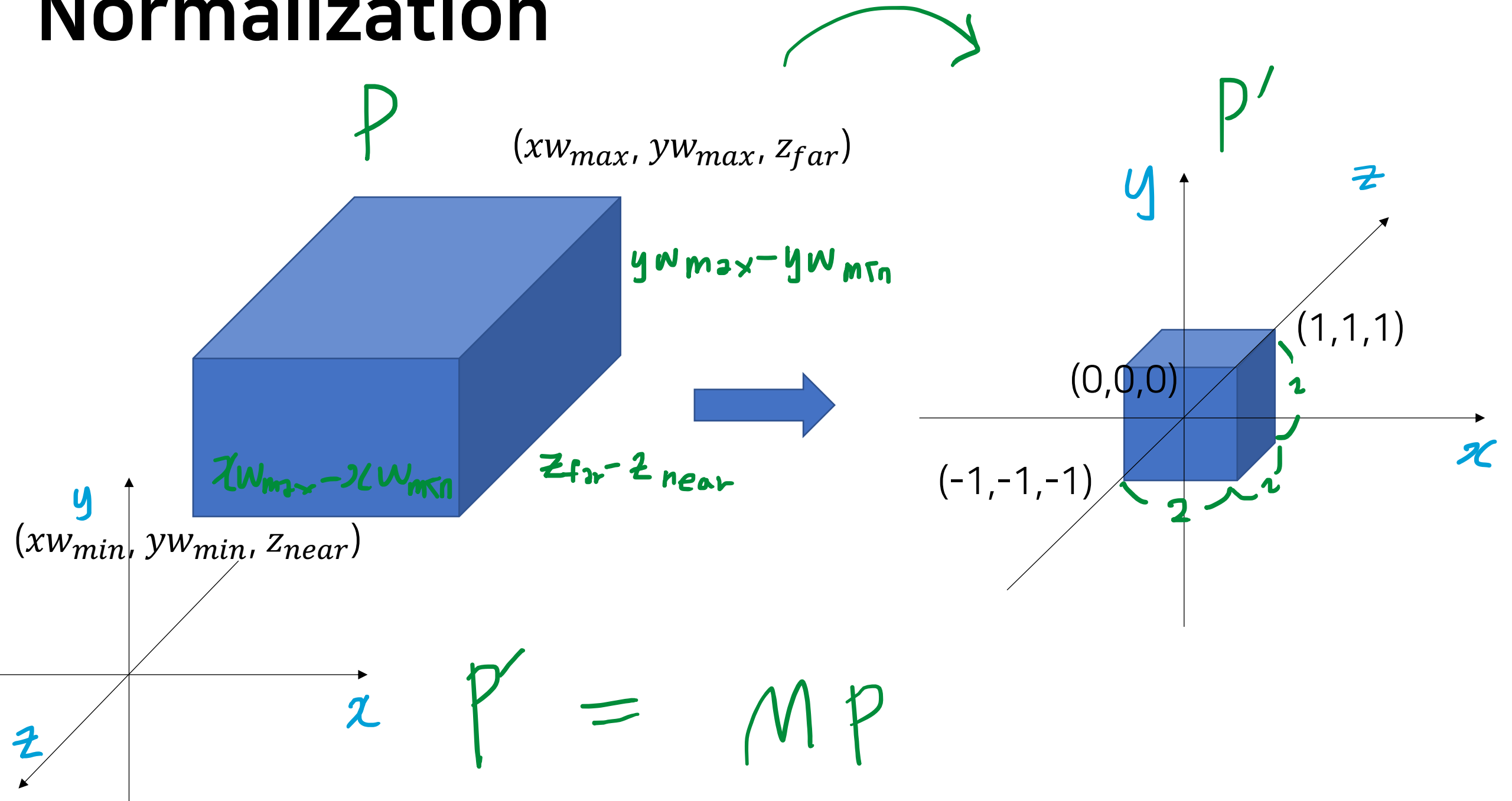


직육면체 view volume

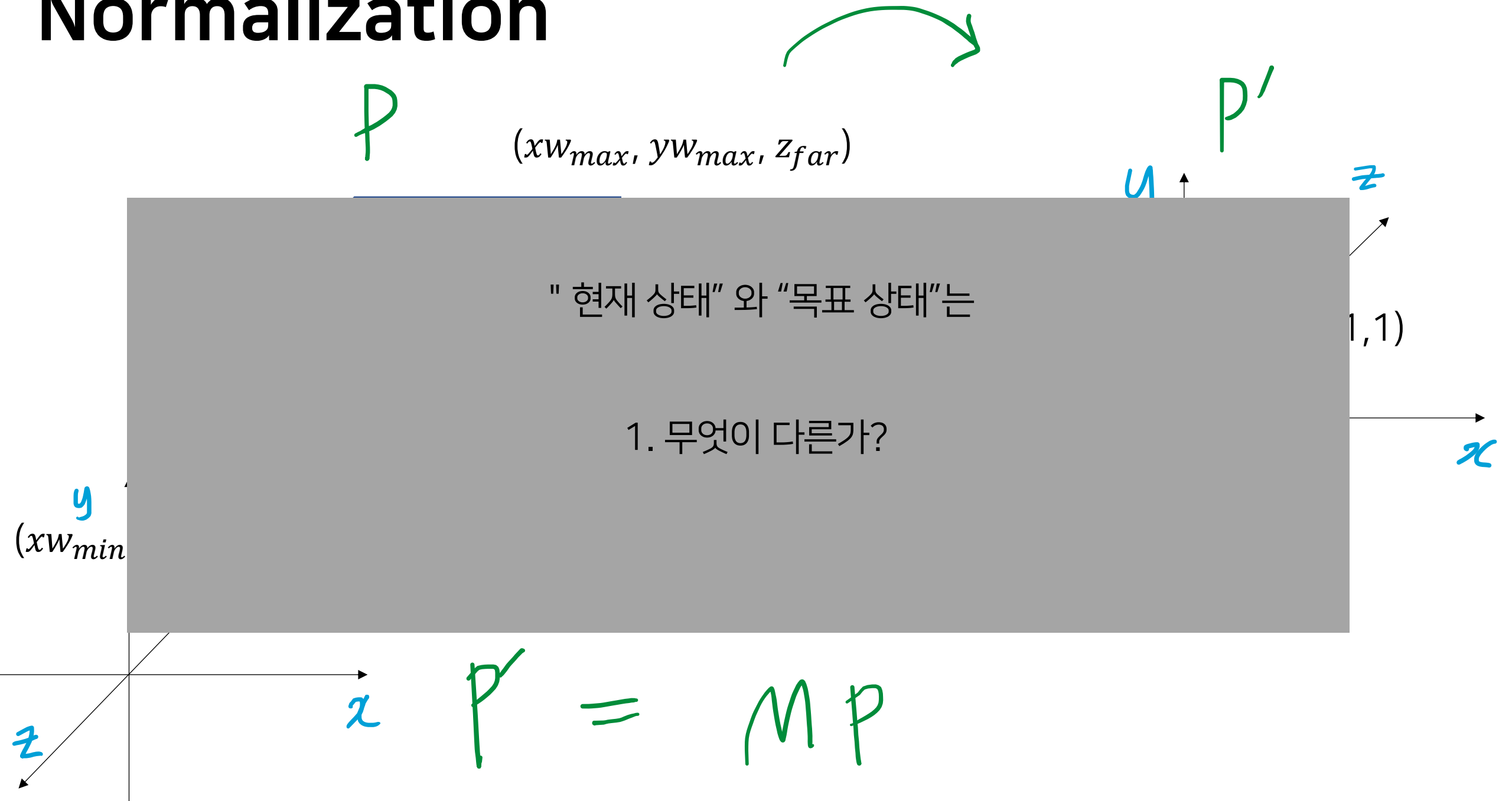
정육면체 view volume

=> Normalization

Normalization



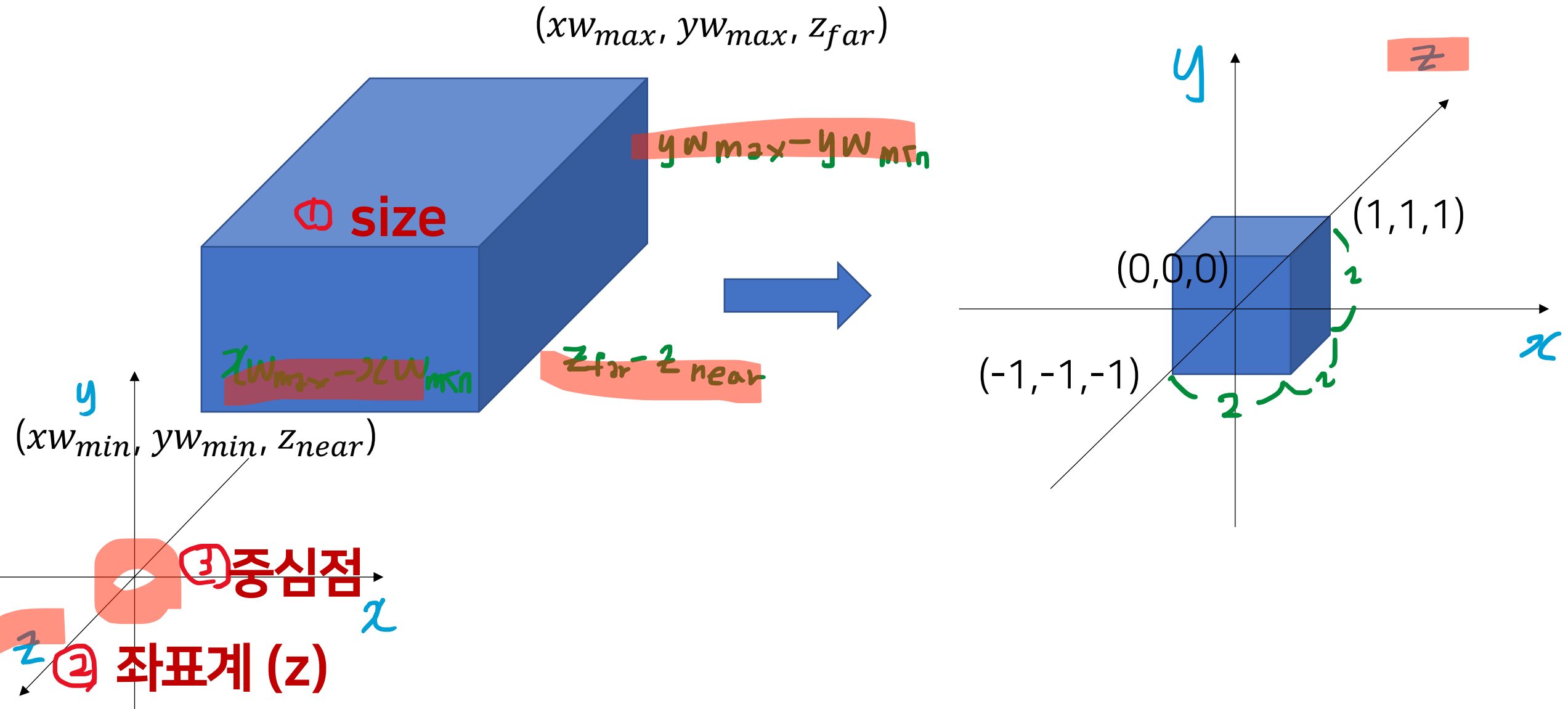
Normalization



" 현재 상태" 와 "목표 상태"는

1. 무엇이 다른가?

Normalization

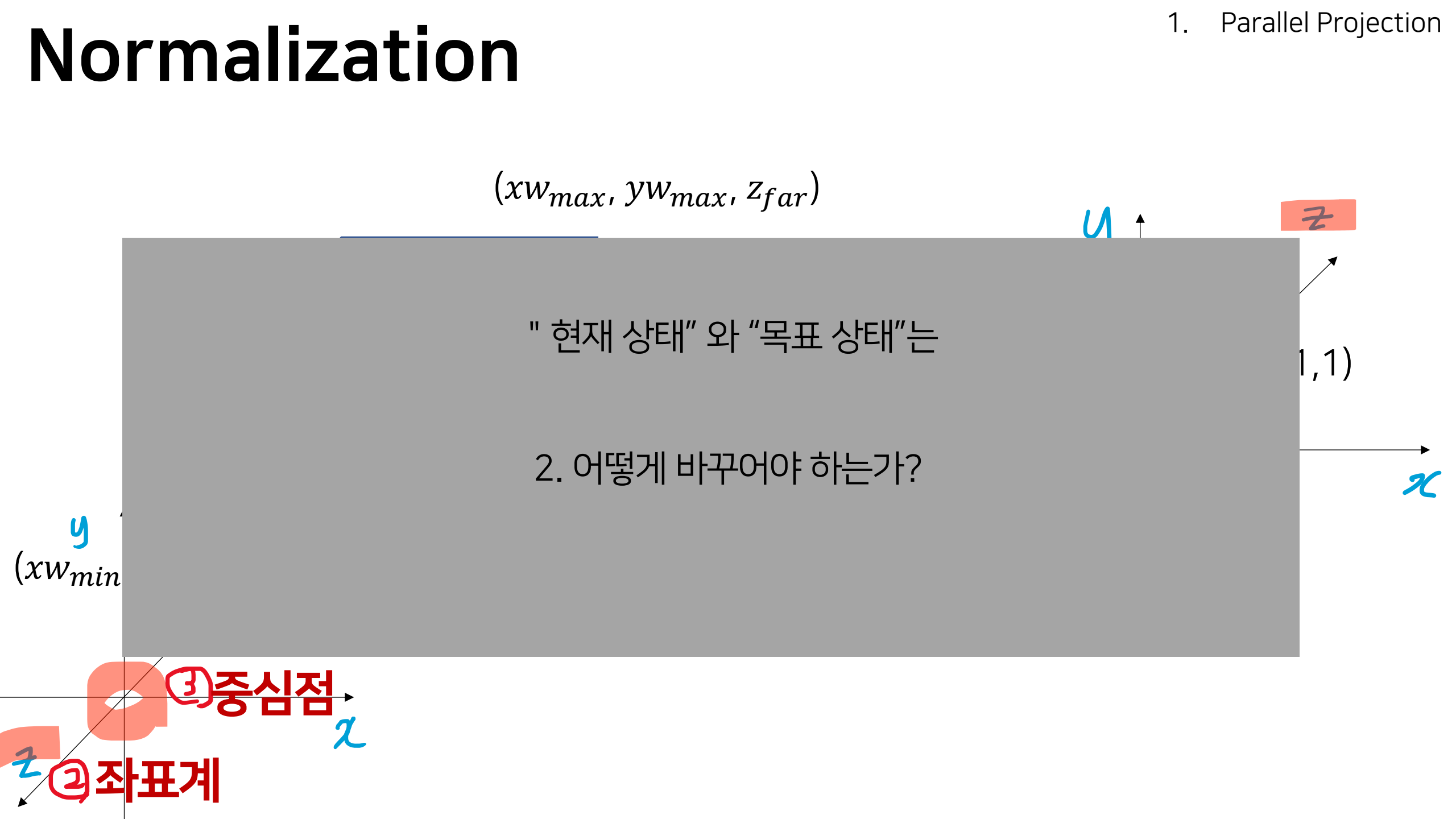


Normalization

$(xw_{max}, yw_{max}, z_{far})$

" 현재 상태" 와 "목표 상태"는

2. 어떻게 바꾸어야 하는가?



Normalization의 과제

1. View volume을 w, d, h -> 2,2,2

   normalized된 정육면체 형태로 만들면 clipping이 효율적이다

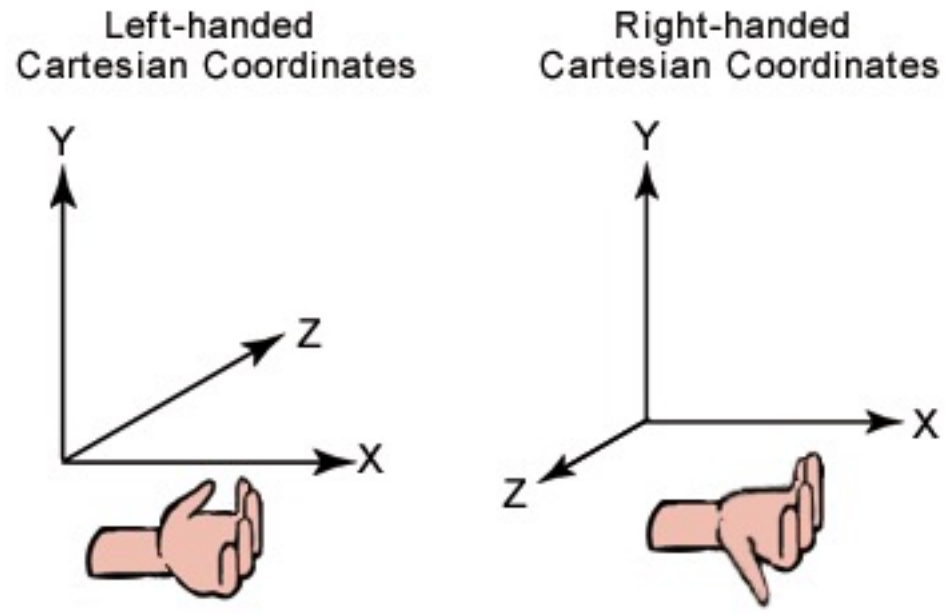
   *참고 : view volume에 따라 clipping algorithm이 달라짐

2. RHS into LHS

   RHS의 경우 눈의 방향이 -z방향,

   Znear, Zfar등이 음수이고, 그러면 distance(양수)

   값으로 사용하기가 어려워짐



3. 중심점 (0,0,0)으로

Normalization

$M_{ortho,norm}$

$P' = MP$

③ 중심점(0,0,0으로)

① size(2,2,2로)

$$\mathbf{M}_{\text{ortho,norm}} = \begin{bmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & -\frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & 0 & -\frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & \frac{-2}{z_{\text{near}} - z_{\text{far}}} & \frac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

② 좌표계

다음주

Perspective Projection
&
Clipping
&
(Line Rasterization)