Computer Graphics

Prof. Jibum Kim

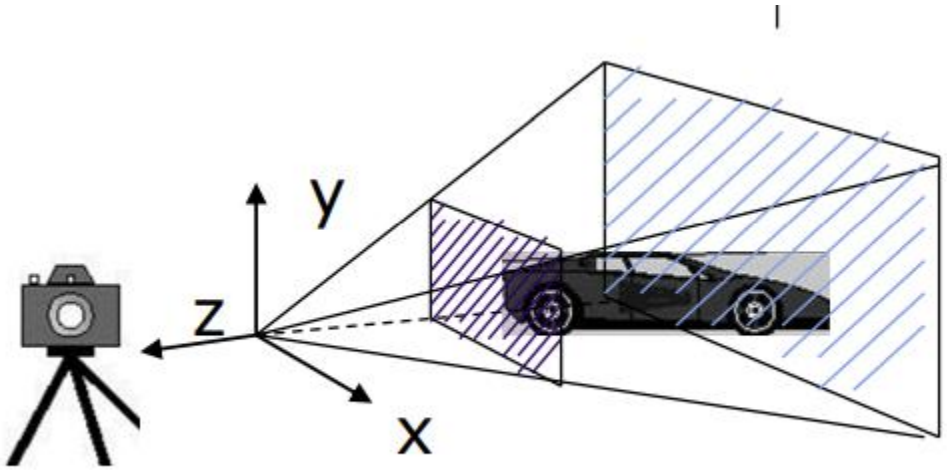
Department of Computer Science & Engineering

Incheon National University

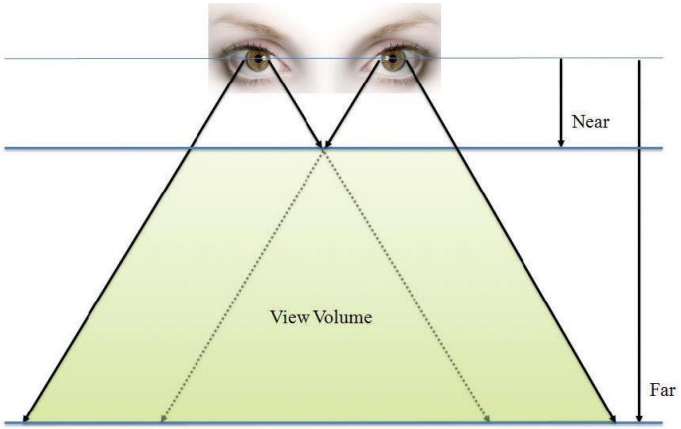
- **Perspective projection (원근 투영)**

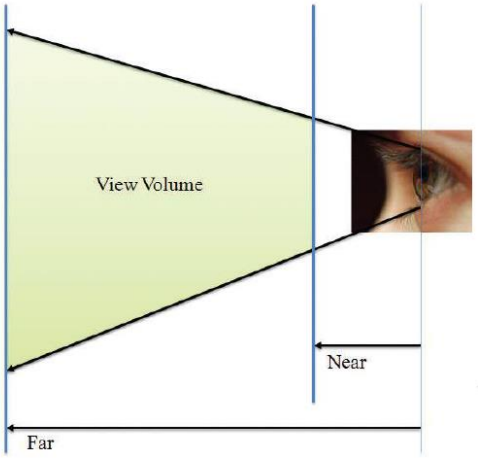
- 원근 투영의 예: 르네상스 그림
- 카메라로부터 멀리 있는 물체는 작게 보이고
  카메라로부터 가까이 있는 물체는 크게 보인다

- Recall projection
- Map the object from 3D space to 2D screen
- Perspective projection (원근 투영)
- Similar to real world
- Objects appear larger if they are closer to camera



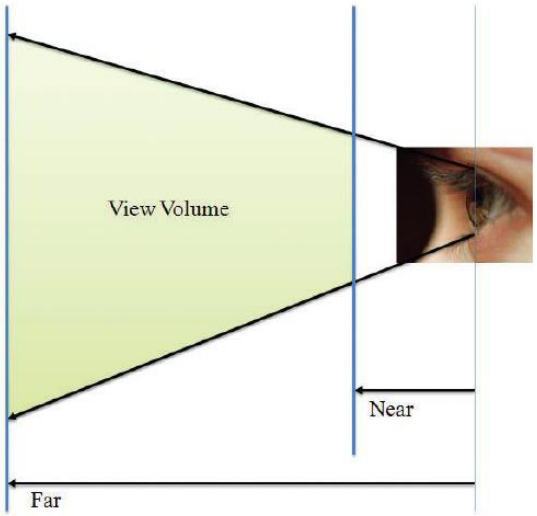
- OpenGL의 Perspective projection은 실제 사람의 눈 (카메라)과 비슷하게 가상으로 만들어져 있다
- 실제 사람의 눈은 좌, 우 가 있고 시야 각이 있다
- 어떤 물체가 카메라와 너무 가까이 있으면 상이 맺히지 않아서 어느 정도 거리가 필요하다. 두 개의 시야 각이 처음으로 만나는 위치에서부터 물체를 제대로 볼 수 있다 (눈으로부터 near만큼 거리)
- 또한, 사람의 눈 (카메라)은 무한히 멀리 있는 물체까지 볼 수 있는 것이 아니므로 최대 볼 수 있는 한계를 정해놓는다 (눈으로부터 far 만큼 거리)
- 즉, only objects between near and far planes are drawn



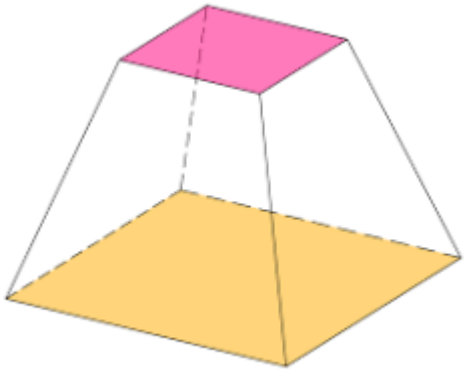


- Perspective projection에서는 이와 같이 눈으로 볼 수 있는 가상의 영역을 viewing frustum이라 한다(why frustum ? truncated pyramid)
- Orthogonal projection의 viewing box와 모양이 다르다
- 단, viewing box와 마찬가지로 물체가 viewing frustum안에 있어야 물체가 보인다고 가정한다

Viewing frustum

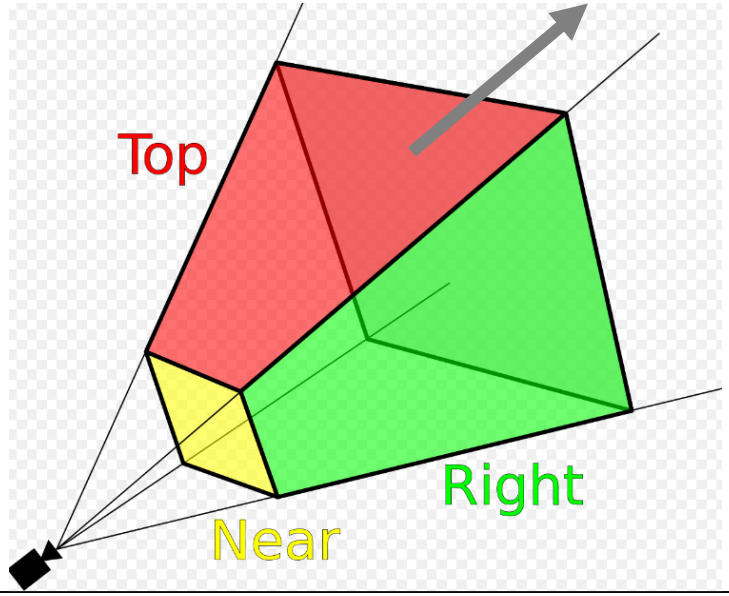


- **Viewing frustum의 형태: 사람 시야의 좌, 우, 위 아래의 시야 각을 모두 고려하면 아래와 같은 frustum (잘라진 피라미드) 형태의 viewing frustum이 만들어 진다**
- **원근 투영 시에 물체는 이 viewing frustum안에 있어야만 보인다고 가정한다**

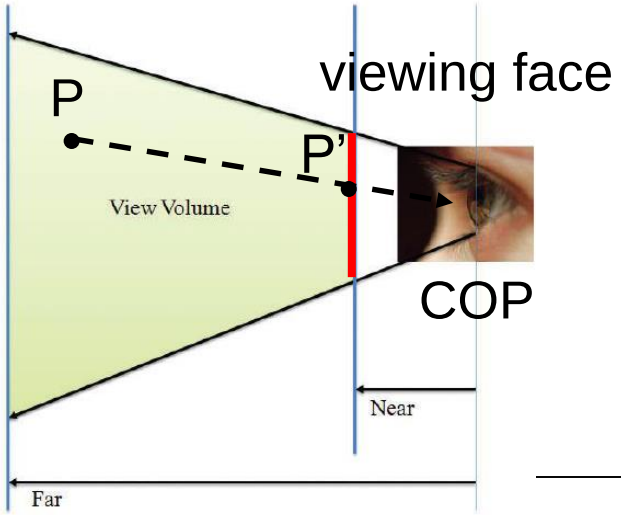


frustum

viewing frustum

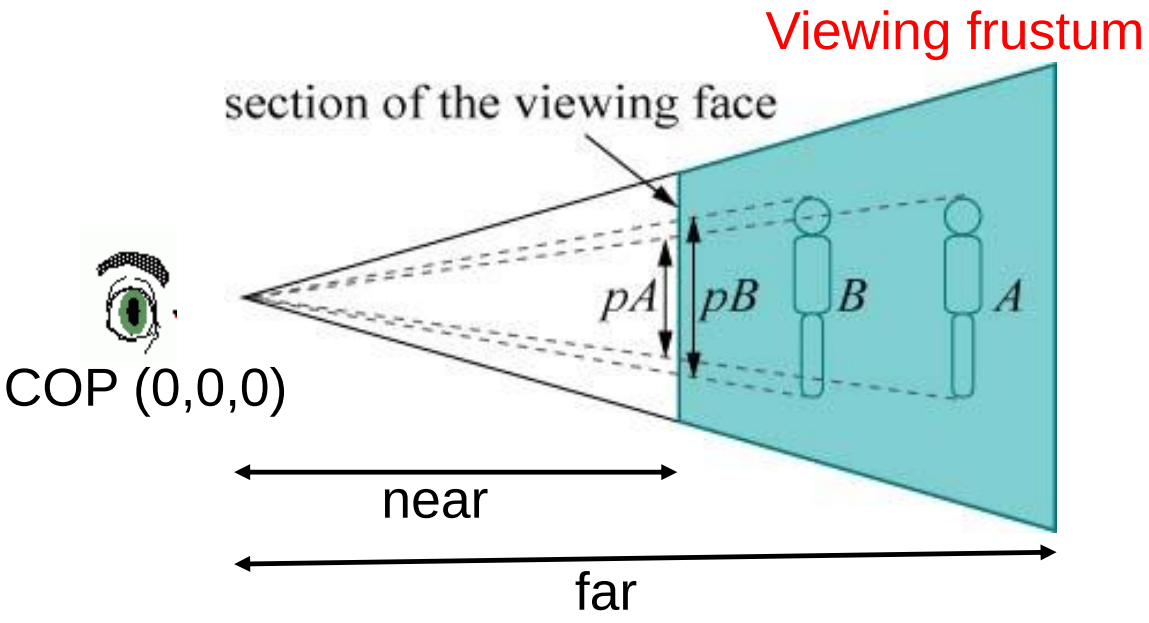


- Camera (eye)의 위치는 : (0, 0, 0) 이고 이를 center of projection (COP)라고 한다. 카메라는 –z축 방향을 바라본다고 가정한다
- 물체는 두 눈의 시야 각이 최초로 만나는 (z=-near)위치에 투영 (projection)된다고 가정한다
- 이러한 원근 투영 (perspective projection)시 viewing frustum 안의 물체는 물체와 눈의 위치(apex)를 연결한 선이 z=-near와 만나는 위치 (viewing face)에 투영된다. 아래 그림에서 점 P는 P'로 투영된다



Viewing frustum 안에 있는 물체는 카메라와 그 물체를 연결한 선이 viewing face에 닿는 곳에 투영된다

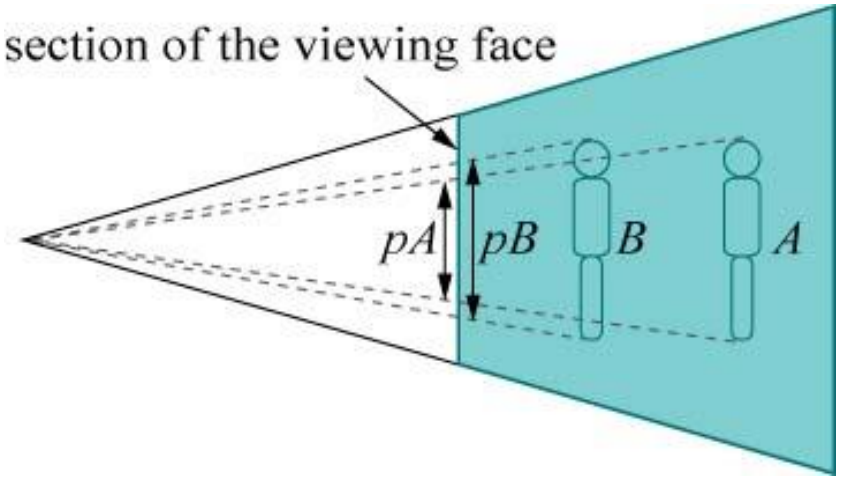
Q) 똑같은 크기 (모양)의 2개의 물체가 frustum안에 있을 때 viewer와 가까이 있는 물체는 크게 보이고 멀리 있는 물체는 작게 보일까?

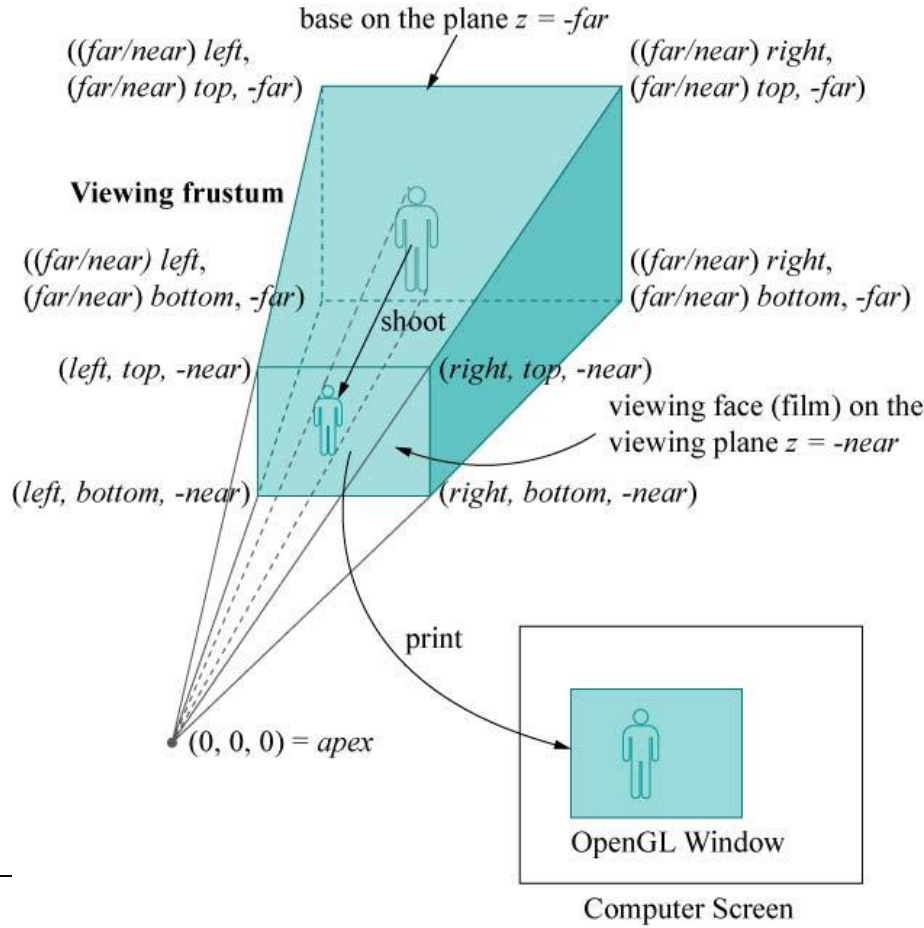


- **OpenGL에서 viewing frustum을 사용하는 방법**

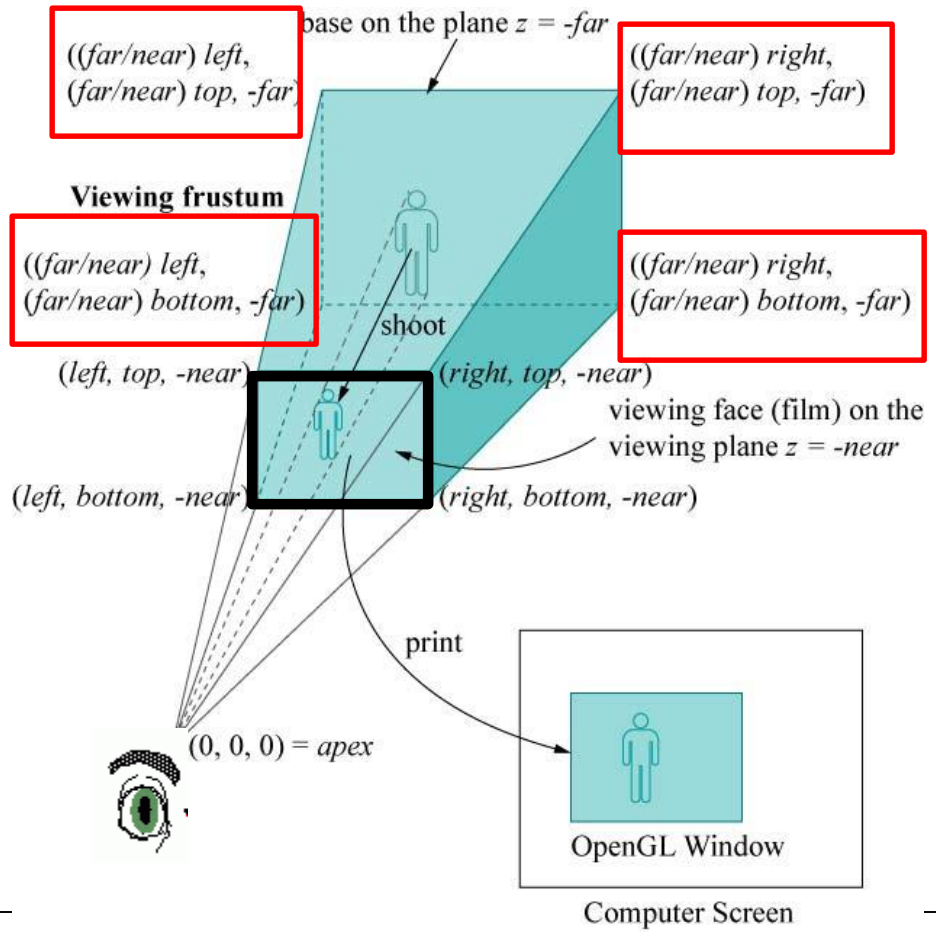
- OpenGL에서 perspective projection은 synthetic camera가 (0,0,0)에 위치하고 있으며 –z축을 바라보고 있다
- OpenGL에서 perspective projection을 사용하려면
- glFrustum(left, right, bottom, top, near, far)
- glm 라이브러리 사용
- glm::frustum(left, right, bottom, top, near, far)

- **glFrustum(left, right, bottom, top, near, far)**





- **glFrustum(left, right, bottom, top, near, far)**



glOrtho vs glFrustum

차이: 뒷면에 있는

네 corner 점의 (x,y)위치가

(far/near)배 만큼

커짐

- 단, glFrustum에서 near와 far값은 모두 양수여야 하고 near<far 값이어야 한다. Why 양수?

- 대부분의 경우 z축을 기준으로 대칭으로 frustum을 만든다 (Why?)

- Left와 right는 크기는 같고 부호만 반대 (right는 양수. Why?)
- Top과 bottom은 크기는 같고 부호만 반대 (Top은 양수. Why?)

- Now, let's draw a viewing frustum of (z축을 기준으로 대칭)
- glFrustum(-15.0, 15.0, -10.0, 10.0, 5.0, 50.0)

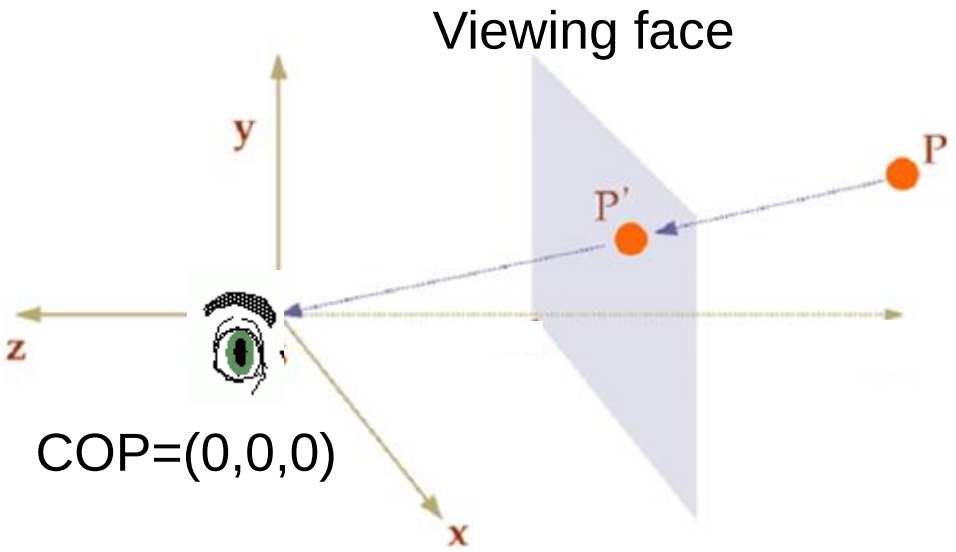
- **원근 투영시 투영되는 위치**

- 그렇다면 viewing frustum안의 어떤 점 (P)이 viewing face의 어디 (P’)에 투영되는지 생각해보자

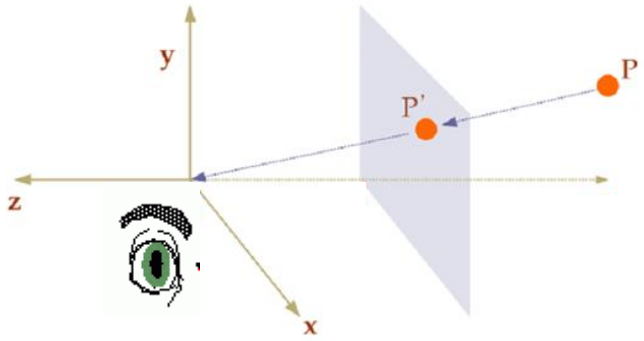
- 앞에서 봤지만 원근투영에서는 viewing face에서 멀어질수록 물체가 축소되어 보인다. 그렇다면 물체가 얼마나 축소되어 보이는지 생각해보자

Viewing Frustum안의 어떤 점 P=(x, y, z) 이 viewing face에 있는 P'=(x', y', z')로 투영 된다고 할 때 P'의 좌표를 구해보자. 아래 그림과 같이 Viewing face의 z 좌표가 z=d 라면 z'=d.

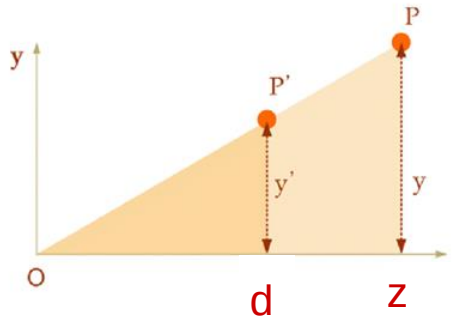
- P'=(x', y', d) 이므로 x'와 y' 만 구하면 된다



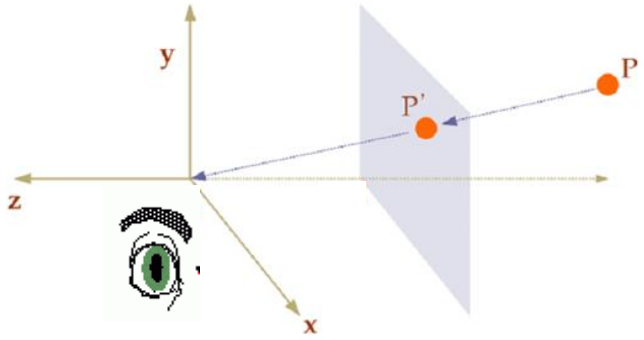
- P=(x, y, z) , P'=(x', y', z')=(x', y', d) 이것을 옆에서 바라보면



- 삼각형의 닮은꼴 공식 사용: z:y = d : y' => y'=(d/z)y=$\frac{y}{z/d}$

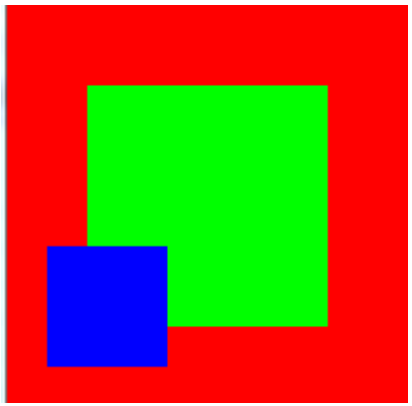


- 같은 방법으로 계산 해보면
- x'=$\frac{x}{z/d}$
- P =(x, y, z) 가 viewing face에 투영되면 P'=(x', y', z')=($\frac{x}{z/d}, \frac{y}{z/d}, d$)
- 즉, 먼 곳에 있는 물체, 즉, z 값이 큰 물체일 수록 투상의 결과가 작게 보인다 (perspective projection의 원리



- 예) glFrustum(0, 100, 0, 100, 5, 50) 로 주어져 있을 때
- P1: (20, 80, -10), P2: (20, 80, -20)
- 이 두 점이 viewing face에 투영된 좌표를 구해보자
- P1’=?
- P2’=?
-

square.cpp 코드를 변형하여
glFrustum(0.0, 100.0, 0.0, 100.0, 5.0, 50.0);
을 사용하고 z=-10에 polygon, z=-20에 polygon을 생성하여
아래와 같은 결과를 확인해보자
왜 아래와 같이 보일까?



```
 glBegin(GL_POLYGON);
 glVertex3f(20.0, 20.0, -10.0);
 glVertex3f(80.0, 20.0, -10.0);
 glVertex3f(80.0, 80.0, -10.0);
 glVertex3f(20.0, 80.0, -10.0);
glEnd();
```

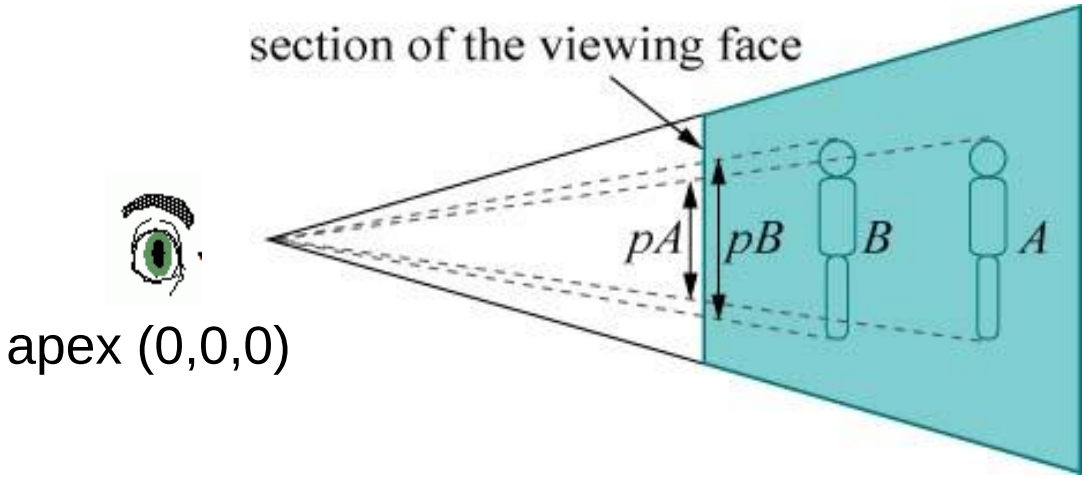
```
   glBegin(GL_POLYGON);
 glVertex3f(20.0, 20.0, -20.0);
 glVertex3f(80.0, 20.0, -20.0);
 glVertex3f(80.0, 80.0, -20.0);
 glVertex3f(20.0, 80.0, -20.0);
glEnd();
```

Q) glFrustum을 사용시 near 와 far 값은 양수를 사용하였다. 그 이유는?

glFrustum(0, 100.0, 0, 100.0, 5.0, 50.0);

glFrustum(-15.0, 15.0, -10.0, 10.0, 5.0, 50.0)

(Synthetic) Camera model에서의 default camera 위치 및 방향



- **ExperimenterSource/Chapter2/Helix.cpp**

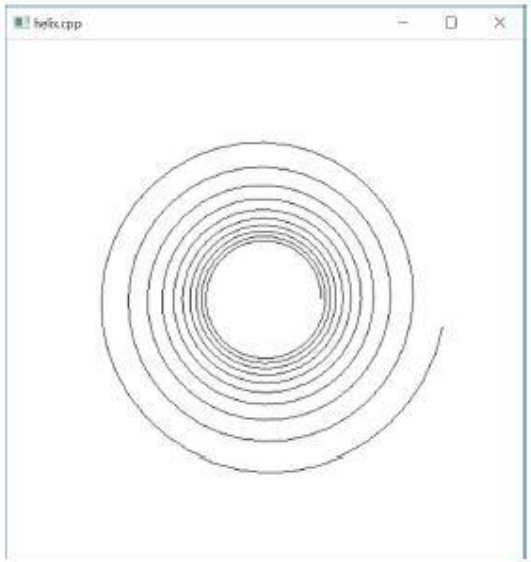


Figure 2.52: Screenshot of `helix.cpp` using perspective projection with the helix coiling up the z-axis.

- **동차 좌표 (homogeneous coordinate)**

- 정의: Point (점) is a location in space
- 특징: Points have position, but neither length nor direction
- 정의: A point $P=[x_1\ x_2\ x_3 \ldots x_m]^T$ in R^m is represented in homogeneous coordinates (동차 좌표) by any m+1 tuple of the form $[cx_1\ cx_2\ cx_3\ \ldots\ cx_m\ c]^T$, where c is a non-zero scalar
- 전치 행렬? Tuple의 의미?
- 예: 점 $P = \begin{bmatrix} 3 \\ 7 \end{bmatrix},\ P \in R^2$
- P 의 가능한 동차 좌표 예 (차원을 하나 올림)
- $\begin{bmatrix} 3 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 6 \\ 12 \\ 2 \end{bmatrix}, \ldots.$

- 동차 좌표의 물리적 의미
- http://darkpgmr.tistory.com/78

- Point에 대해서는 c=1로 고정하고 사용 (vector는 c=0)
- $P=[x_1\ x_2\ x_3 \ldots x_m]^T$
- 동차좌표
- $[x_1\ x_2\ x_3\ \ldots\ x_m\ 1]^T$

- 예: 점 $P=\begin{bmatrix}3\\7\end{bmatrix}$ 의 동차 좌표 $\begin{bmatrix}3\\7\\1\end{bmatrix}$

- 왜 컴퓨터 그래픽스에서는 동차좌표를 사용할까?
- 1. 기본적으로 다음에 배울 변환 (transformation)시에 동차좌표를 사용하면 **변환을 행렬 곱, 형태로 표현 가능**하다
- 2. 여러 개의 변환이 연속으로 적용되는 복합 변환의 경우에는 이러한 **행렬 곱을 하나의 행렬 곱 형태**로 바꿀 수 있다 (이것은 계산 속도 면에서도 유리하다)
- OpenGL에서도 동차좌표를 사용함
- 3. 또한, Point와 vector를 구별할 수 있다

$v = [a_1, a_2, a_3]$ $\quad v = [a_1, a_2, a_3, 0]^T$

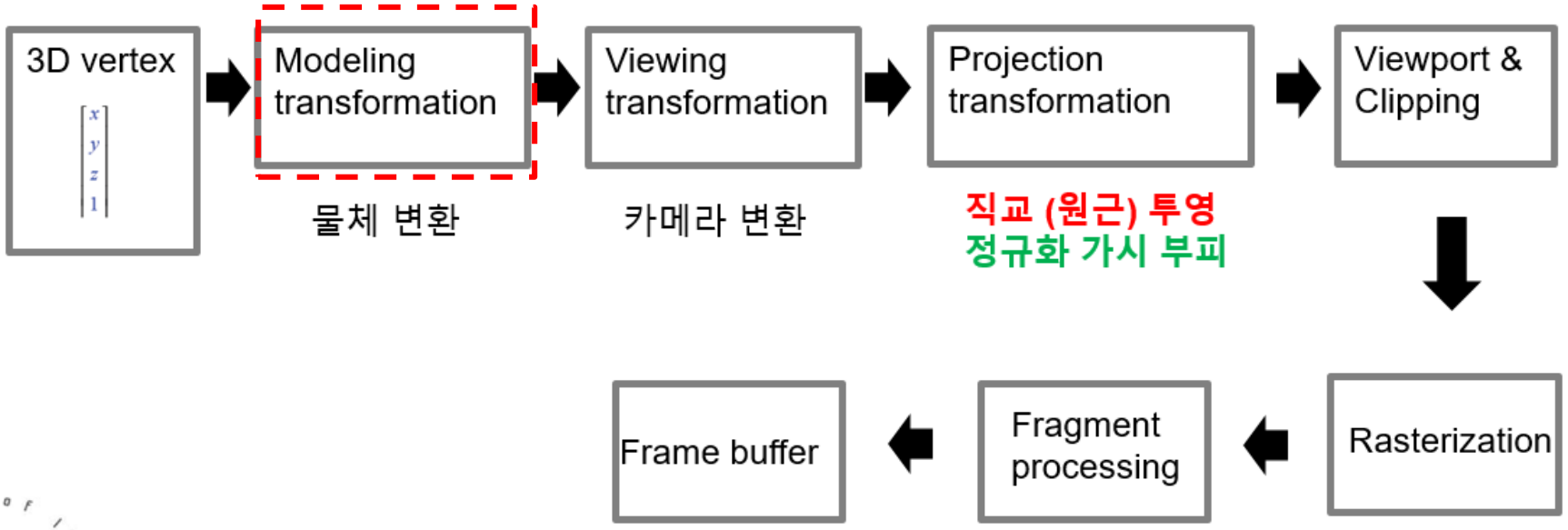
$p = [b_1, b_2, b_3]$ $\quad p = [b_1, b_2, b_3, 1]^T$

- 동차 좌표를 이용 perspective projection을 행렬 곱 형태로 표현 가능
- P'=(x', y', z')=$(\frac{x}{z/d}, \frac{y}{z/d}, d)$
- 이를 동차 좌표로 표현 시에 P'=(x', y', z', 1)=$(\frac{x}{z/d}, \frac{y}{z/d}, d, 1)$
- 동차 좌표는 모든 요소에 같은 값을 곱해주어도 변화가 없다
- (z/d)를 곱해주면
- P'=(x, y, z, z/d). 즉,
- $P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$,

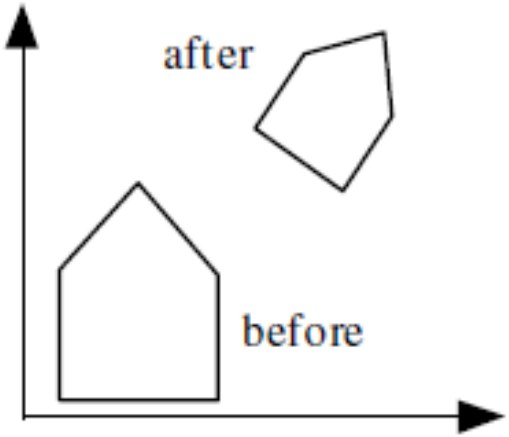
이를 4x4 homogeneous perspective projection matrix 라고 한다

- **Object transformations (물체의 변환)**

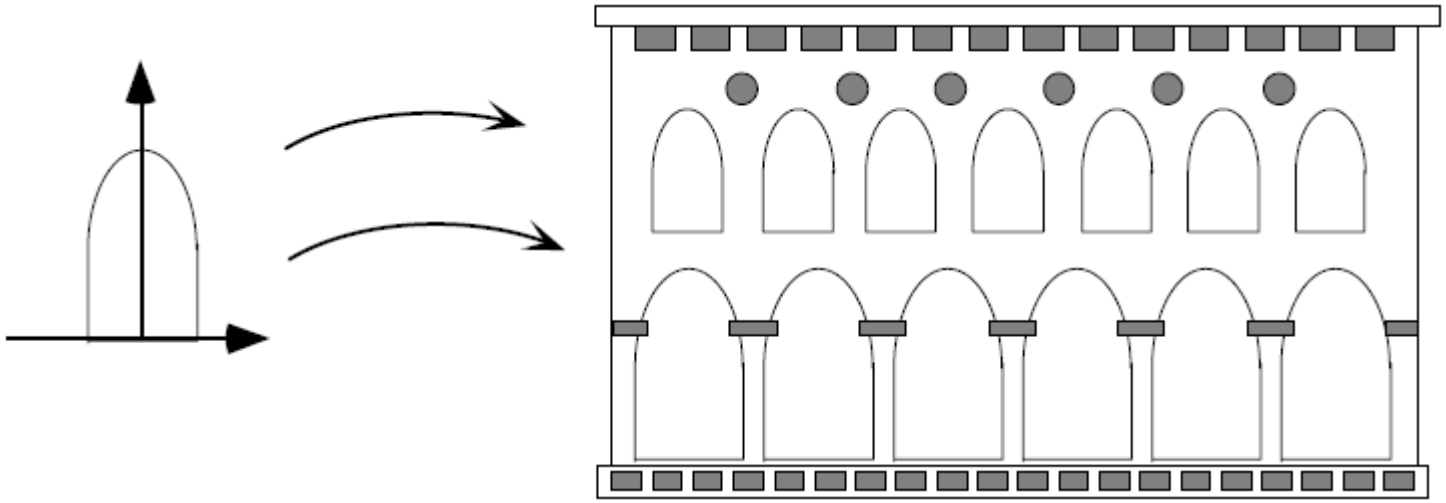
Graphics pipeline (Pre-shader OpenGL)



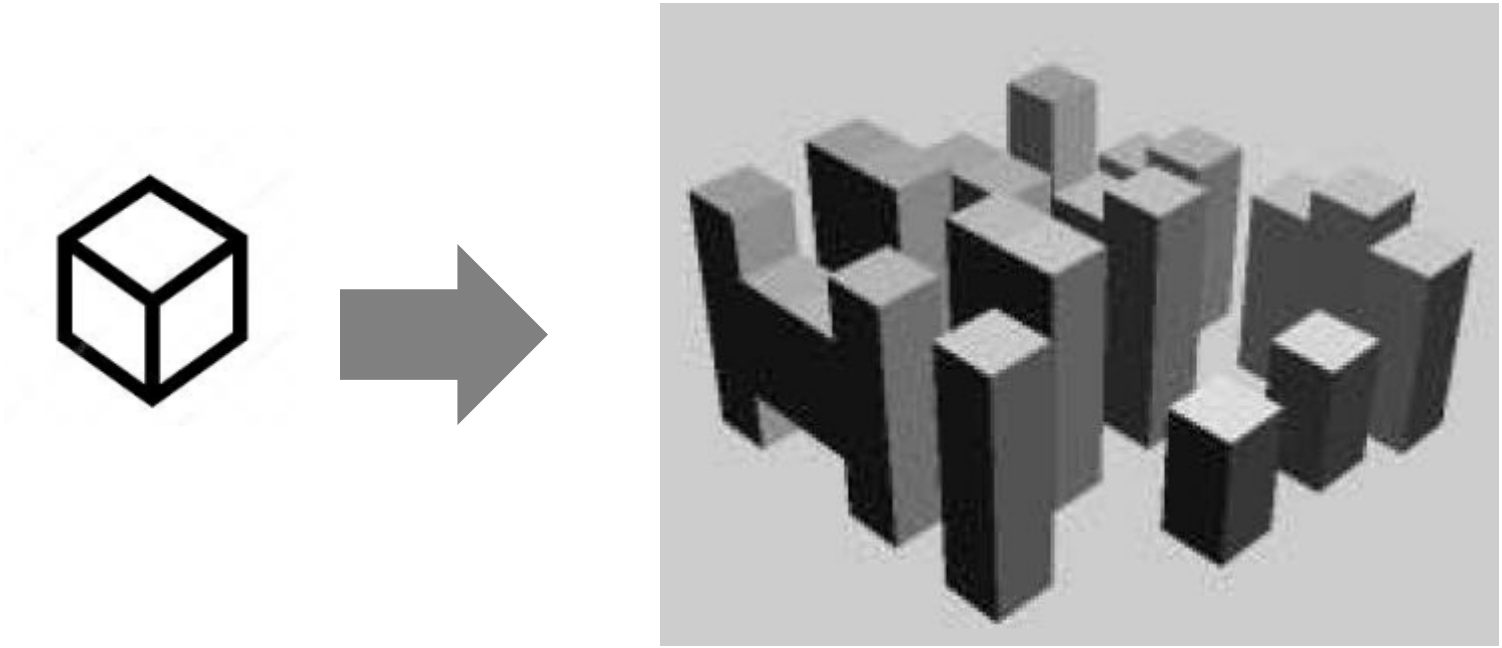
- A transformation is a function that takes a point (or a vector) and maps it into another point (or vector)
- 물체의 변환 (transformation of objects)이란 간단히 얘기하면 물체의 크기가 변화되거나 물체의 위치가 이동하는 것들을 말한다.
- 물체의 좌표 P(x, y, z)가 어떤 함수를 통하여 P'(x', y', z')로 mapping 된 것을 의미 한다. P'=f(P)
- 먼저 물체의 변환의 필요성에 대해서 살펴보자



- 물체의 변환의 필요성
- 1) 2D 예: 기본적인 형태의 물체를 만든 후 이를 변환 (이동, 크기 변환 등..)시켜서 다양한 형태를 손쉽게 구성할 수 있다



- 2) 3D 예: 위치 이동 및 크기 변환



- 3) 눈송이 예: 아래의 눈송이 (snowflake)와 같은 몇몇 물체들은 기본 형태를 회전, 이동, 반사와 같은 여러 개의 변환을 반복 적용한 결과이다.