2023-2 Computer Graphics PI 2차시

3D Transformation & OpenGL

AM11:00에 시작됩니다.

입장 후 채팅창에 학번/이름 작성 부탁드립니다.

3D Transformation

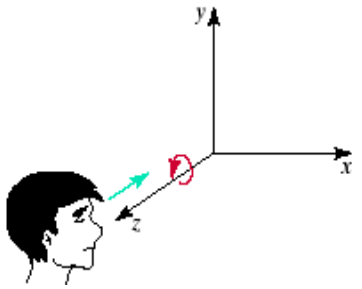
3D Transformation

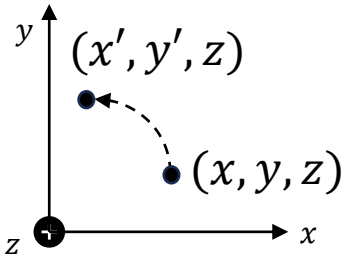
- x축, y축, z축 기준 회전
- 임의의 축 기준 회전
  - 방법1) Rodrigue's formula
  - 방법2) Quaternion
- Other Transformations – Scaling, Reflection
- Affine Transformation

x, y, z 축 기준 회전

Z-axis rotation

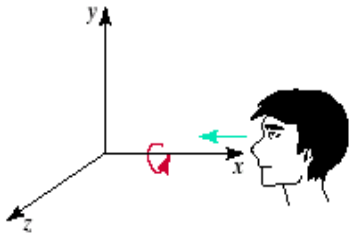
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

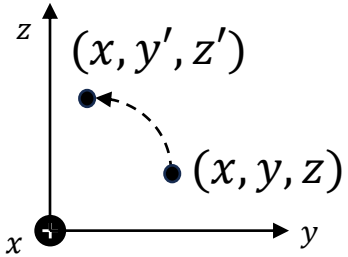




X-axis rotation

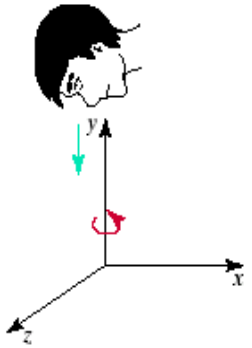
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

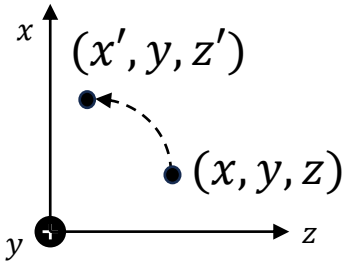




Y-axis rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$





임의의 축 회전 – (방법1) Rodrigue's Formula

- (의의) 임의의 회전축에 대한 rotation matrix R 을 공식으로 구할 수 있다.
- (의의) **u vector** $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ 를 알 때 **R matrix** 를 구할 수 있다.

$$\begin{bmatrix} & & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

- (전제조건) 임의의 회전축을 **u** 라 하자.
  - 이때 **u** 는 (1) 원점을 지나고 (2) unit vector 이다.
  - 만약 **u** 가 주어지지 않고
    지나는 두 점 $P'_1 = (x_1, y_1, z_1)$ $P'_2 = (x_2, y_2, z_2)$ 만 주어졌다면

$$\boldsymbol{u} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \frac{1}{\|P'_2 - P'_1\|} \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix}$$

로 구하면 된다.

u 가 주어졌는데 원점을 지나지 않는다면?
: 지난 시간에 배운 방법으로 해결
(1) u 가 원점 지나게 translation
(2) 원점 지나는 u′ 에 대해 rotation
(3) 다시 원래대로 translation

임의의 축 회전 – (방법1) Rodrigue's Formula

- (정의)

회전축 $\boldsymbol{u} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ 가 (1) 원점 지나고 (2) 단위 벡터 이고

회전각이 $\boldsymbol{\theta}$ 일 때

$$\hat{\mathbf{u}} = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix}$$ 이고

$\mathbf{R} = \mathbf{I} + \sin\theta\hat{\mathbf{u}} + (1 - \cos\theta)\hat{\mathbf{u}}^2$ 이다.

Skew Symmetric Matrix, $\hat{\mathbf{u}}$

- (정의) $\hat{\mathbf{u}}^{\mathrm{T}} + \hat{\mathbf{u}} = \mathbf{0}$

- (기하학적 의미) $\hat{\mathbf{u}}\mathbf{v} = \mathbf{u} \times \mathbf{v}$

$\mathbf{v} \in \mathbb{R}^3$ 인 vector $\mathbf{v}$ 에 대해,

$\mathbf{u} \times \mathbf{v}$ 외적 연산을 determinant 등을 이용해 계산하지 않고

$\mathbf{u}$ 로 $\hat{\mathbf{u}}$ 를 구하고 $\mathbf{v}$ 와 행렬곱 연산을 통해 구할 수 있다.

즉, 외적을 계산할 수 있는 또 다른 방법

임의의 축 회전 – (방법1) Rodrigue's Formula

(문제 풀이 방법)

- STEP1. 원점 지나고, 단위 vector 인 $\boldsymbol{u}$ 를 구하고 ($\boldsymbol{u}$ 는 문제 설명에 주어지는 경우 많음)
- STEP2. $\hat{\mathbf{u}}$ 를 구하고
- STEP3. $\hat{\mathbf{u}}^2$ 를 계산해서
- STEP4. Rodrigue's Formula 적용 $\mathbf{R} = \mathbf{I} + \sin\theta\hat{\mathbf{u}} + (1 - \cos\theta)\hat{\mathbf{u}}^2$

(문제 풀이 예시) z축 회전을 Rodrigue's Formula 로 풀어보자

STEP1. $\mathbf{u} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

STEP2. $\hat{\mathbf{u}} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

STEP3. $\hat{\mathbf{u}}^2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

STEP4. $\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \sin\theta \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + (1 - \cos\theta) \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

임의의 축 회전 – (방법1) Rodrigue's Formula

(증명) 간단하게만 보자. 자세한 계산은 직접해보고 질문

- (1) u가 회전축이 맞는지 증명
  - 회전축은 회전하지 않으니, 회전해도 변화 없음을 보이면 됨.
  - 즉, **Ru=u**임을 보이자.

- (2) **Rv** 가 회전축 **u** 를 중심으로 **v** 를 θ 만큼 rotate 한게 맞는지 증명 (**u** 와 v 는 수직)
  - 1) **v** 가 **Rv** 가 되어도 길이 변화 없음을 보이기
  - 2) **v** 와 **Rv** 두 벡터의 끼인 각이 임을 θ 보이기

  - **Rv**

$$= (\mathbf{I} + \sin\theta\hat{\mathbf{u}} + (1-\cos\theta)\,\hat{\mathbf{u}}^2)\mathbf{v}$$
$$= \mathbf{v} + \sin\theta(\mathbf{u}\times\mathbf{v}) + (1-\cos\theta)\mathbf{u}\times(\mathbf{u}\times\mathbf{v})$$
$$= \mathbf{v} + \sin\theta(\mathbf{u}\times\mathbf{v}) + (1-\cos\theta)(-\mathbf{v})$$
$$= \sin\theta(\mathbf{u}\times\mathbf{v}) + \cos\theta\mathbf{v}$$

$$\mathbf{a}\times(\mathbf{b}\times\mathbf{c}) = \mathbf{b}(\mathbf{c}\cdot\mathbf{a}) - \mathbf{c}(\mathbf{a}\cdot\mathbf{b})$$

$$\|\sin\theta(\mathbf{u}\times\mathbf{v}) + \cos\theta\mathbf{v}\|^2 = (\sin\theta(\mathbf{u}\times\mathbf{v}) + \cos\theta\mathbf{v})\cdot(\sin\theta(\mathbf{u}\times\mathbf{v}) + \cos\theta\mathbf{v}) = \sin^2\theta\|\mathbf{u}\times\mathbf{v}\|^2 + \cos^2\theta\|\mathbf{v}\|^2$$
$$= \sin^2\theta\|\mathbf{v}\|^2 + \cos^2\theta\|\mathbf{v}\|^2 = \|\mathbf{v}\|^2$$ 따라서 1) 이 증명됨

$$(\sin\theta(\mathbf{u}\times\mathbf{v}) + \cos\theta\mathbf{v})\cdot\mathbf{v} = \cos\theta\|\mathbf{v}\|^2 = \cos\theta\|\sin\theta(\mathbf{u}\times\mathbf{v}) + \cos\theta\mathbf{v}\|\|\mathbf{v}\|$$ 따라서 2) 가 증명됨

임의의 축 회전 – (방법2) Quaternion

(참고) 2차원에서도 rotation 을 표현하는 방법이 여러가지.
- 1) 2x2 행렬
- 2) 복소수 i 이용

마찬가지로, 3D rotation 도 quaternion 을 이용해서 표현 가능.

(Quaternion 정의)

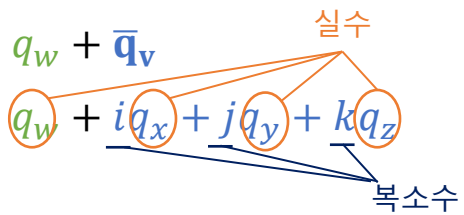
$\mathbf{q}_v \in \mathbb{R}^3, q_w \in \mathbb{R}$ 일때

$\mathbf{q} \in \mathbb{H}$

$= (q_w, \mathbf{q}_v)$ → (q_x, q_y, q_z) 는 3차원 벡터

$= q_w + \overline{\mathbf{q}}_\mathbf{v}$

$= q_w + iq_x + jq_y + kq_z$



이때 $i^2 = j^2 = k^2 = ijk = -1, \quad jk = -kj = i, \quad ki = -ik = j, \quad ij = -ji = k$

임의의 축 회전 – (방법2) Quaternion

(Quaternion Algebra) multiplication 곱, addition 합, conjugate 켤레 복소수, norm 크기, identity, inverse

$\mathbf{q} = q_w + iq_x + jq_y + kq_z = (q_w, \mathbf{q_v})$ $\mathbf{r} = r_w + ir_x + jr_y + kr_z = (r_w, \mathbf{r_v})$	
Multiplication	$\mathbf{qr} = (q_w r_w - \mathbf{q_v} \cdot \mathbf{r_v}, \mathbf{q_v} \times \mathbf{r_v} + r_w \mathbf{q_v} + q_w \mathbf{r_v})$ $\mathbf{qr} = \mathbf{rq}$ iff $\mathbf{q_v} \times \mathbf{r_v} = 0$ (i.e. $\mathbf{q_v}, \mathbf{r_v}$ are parallel)
Addition	$\mathbf{q} + \mathbf{r} = (q_w + r_w, \mathbf{q_v} + \mathbf{r_v})$
Conjugate	$\mathbf{q}^* = (q_w, -\mathbf{q_v})$
Norm	$\|\mathbf{q}\| = \sqrt{\mathbf{q}\mathbf{q}^*} = \sqrt{q_x^2 + q_y^2 + q_z^2 + q_w^2}$
Identity	$id = (1, \mathbf{0}) = 1$
Inverse	$\mathbf{q}^{-1} = \frac{1}{\|\mathbf{q}\|^2}\mathbf{q}^*$ $(\mathbf{q}^{-1} = \mathbf{q}^*$ if $\mathbf{q}$ is a unit quaternion)

자세한 설명은 생략.
Quaternion 의 연산 방법이니 외워주면 됨.

임의의 축 회전 – (방법2) Quaternion

Quaternion Rotation

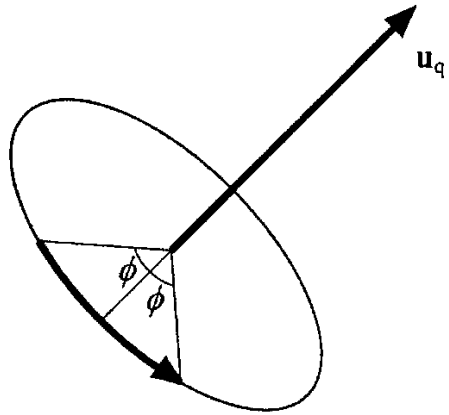
- 앞서 배운 quaternion 으로 rotation 을 표현해보자.

- Rotation 을 표현하는 방법 중에 matrix 도 있는데, 왜 quaternion 을 이용할까?

  (= quaternion rotation 의 장점)
  - 저장 공간이 덜 필요하다
  - 산술 연산이 덜 필요하다
  - 회전을 쉽게 interpolate 할 수 있다. 따라서 더 자연스러운 애니메이션을 만들 수 있다.

임의의 축 회전 – (방법2) Quaternion

Quaternion Rotation



- (unit quaternion)

$\mathbf{u_q}$: 회전축, $||\mathbf{u_q}|| = 1$

$\boldsymbol{\varphi}$: 회전각

이걸로 unit quaternion $\mathbf{q}$ 를 만들면,

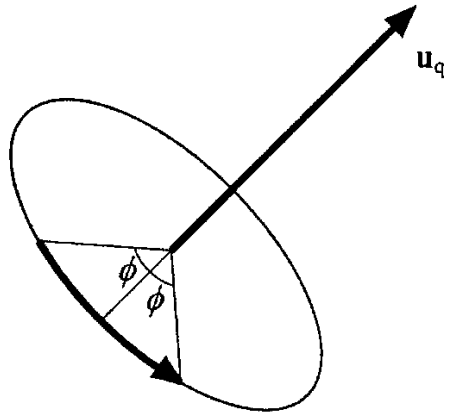
$$\mathbf{q} = (\cos\frac{\varphi}{2}, \sin\frac{\varphi}{2}\mathbf{u_q}) = \cos\frac{\varphi}{2} + \sin\frac{\varphi}{2}\,\bar{\mathbf{u}}_\mathbf{q} \in \mathbb{S}^3$$

이때 $||\mathbf{q}|| = 1, (\bar{\mathbf{u}}_\mathbf{q})^2 = -1$

임의의 축 회전 – (방법2) Quaternion

Quaternion Rotation

- Rotation 을 어떻게 quaternion 으로 표현하는가?



p= $(p_x,\ p_y,\ p_z) \in \mathbb{R}^3$를 회전축 $\mathbf{u_q}$, 회전각 $\boldsymbol{\varphi}$ 에 대해 rotate 해보자

STEP1. 주어진 점 p= $(p_x,\ p_y,\ p_z)$ 를 quaternion 으로 바꾸기.

$$\mathbf{p} = (0, p_x,\ p_y,\ p_z)$$

STEP2. 주어진 회전축을 이용해 unit quaternion 인 $\mathbf{q}$ 를 구하기.

$$\mathbf{q} = (\cos\frac{\varphi}{2}, \sin\frac{\varphi}{2}\mathbf{u_q}) = \cos\frac{\varphi}{2} + \sin\frac{\varphi}{2}\bar{\mathbf{u}}_\mathbf{q}$$

STEP3. 회전을 나타내는 quaternion $\mathbf{q}$ 에 대해

$$\mathbf{qpq^{-1}} = \mathbf{qpq}^*$$

를 계산하면

$$(0, p'_x,\ p'_y,\ p'_z)$$

i, j, k 의 계수가 $p'_x,\ p'_y,\ p'_z$ 이고 $(p'_x,\ p'_y,\ p'_z)$ 가 회전된 점의 좌표가 된다.

임의의 축 회전 – (방법2) Quaternion

Quaternion Rotation

- (예시) Rotate $\mathbf{p}$=(1, 0, 0) $\in \mathbb{R}^3$ by 90 degrees around z-axis

STEP1. 주어진 점 p= $(p_x,\ p_y,\ p_z)$ 를 quaternion 으로 바꾸기. $\mathbf{p} = (0, p_x,\ p_y,\ p_z)$

$$\mathbf{p} = (0, 1, 0, 0)$$

STEP2. 주어진 회전축을 이용해 unit quaternion 인 $\mathbf{q}$ 를 구하기. $\mathbf{q} = (\cos\frac{\varphi}{2}, \sin\frac{\varphi}{2}\mathbf{u_q}) = \cos\frac{\varphi}{2} + \sin\frac{\varphi}{2}\bar{\mathbf{u}}_\mathbf{q}$

$$\mathbf{q} = \cos\frac{\frac{\pi}{2}}{2} + \left(\sin\frac{\frac{\pi}{2}}{2}\right)(0i + 0j + 1k) = \frac{1}{\sqrt{2}}(1 + k)$$

STEP3. 회전을 나타내는 quaternion $\mathbf{q}$ 에 대해 $\mathbf{qpq^{-1}} = \mathbf{qpq}^*$ 계산하기.

$$\mathbf{p}' = \mathbf{qpq}^* = \frac{1}{\sqrt{2}}(1 + k)(1i + 0j + 0k)\frac{1}{\sqrt{2}}(1 - k)$$

$$= \tfrac{1}{2}(i + j)(1 - k) = \tfrac{1}{2}(i - ik + j - jk) = \tfrac{1}{2}(i + j + j - i) = j = 0i + 1j + 0k$$

따라서 (0, 1, 0) 이 답이 된다.

임의의 축 회전 – (방법2) Quaternion

Quaternion, Matrix 간의 변환

1) Quaternion -> Matrix

Unit quaternion $\mathbf{q} = w + xi + yj + zk$ 를 rotation matrix **R**로 바꾸면,

$$\mathbf{R} = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix}$$

2) Matrix -> Quaternion

Rotation matrix $\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ 을 $\mathbf{q} = w + xi + yj + zk$ 로 바꾸면, w, x, y, z 를 아래와 같이 구할 수 있다

$\mathrm{tr}(\mathbf{R}) := r_{11} + r_{22} + r_{33}$ 이라 하자. (trace 라 부르며, 대각선을 다 더하면 된다)

$w = \frac{\sqrt{\mathrm{tr}(\mathbf{R})+1}}{2}$

$x = (r_{32} - r_{23})/(4w)$

$y = (r_{13} - r_{31})/(4w)$

$z = (r_{21} - r_{12})/(4w)$

Other Transformations - Scaling, Reflection

Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Reflection

$$R_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_{yz} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_{zx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Affine Transformation

- (정의) transformed coordinate 가 input coordinate 에 대한 linear function 일 때

그 transformation 을 affine transformation 이라 한다.

$$x' = a_{xx}x + a_{xy}y + a_{xz}z + b_x$$
$$y' = a_{yx}x + a_{yy}y + a_{yz}z + b_y$$
$$z' = a_{zx}x + a_{zy}y + a_{zz}z + b_z$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{xx} & a_{xy} & a_{xz} & b_x \\ a_{yx} & a_{yy} & a_{yz} & b_y \\ a_{zx} & a_{zy} & a_{zz} & b_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

=> translation, rotation, reflection, scaling, shear : 앞서 본 모든 transformation 은 affine transformation 에 해당된다.

- (성질1) **collinearity** : line 을 affine transformation 해도 line 이다.
  (성질2) **parallelism** : 두 개의 평행한 line 이 있다면, 그 두 line 을 affine transformation 해도 여전히 평행하다.

- Rotation, Translation : rigid body transformation 에 해당
- Rotation, Reflection : Orthogonal matrix. $AA^T = I$.

=> **Rotation, Reflection 의 역행렬을 구하고 싶으면 transpose 를 구하면 된다.**

QUIZ – 3D Transformations

What is the 4×4 homogeneous matrix that rotates in 3D about the x-axis in 90 degrees?

하나를 선택하세요.

a.
$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b.
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c.
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

d.
$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

답이 맞습니다.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 90 & -\sin 90 & 0 \\ 0 & \sin 90 & \cos 90 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

정답 : $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotate (1,1, 1) using the result from question 1.

하나를 선택하세요.

a. (1, -1, -1)

b. (-1, -1, 1)

c. (1, -1, 1)

d. (1, 1, -1)

답이 맞습니다.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

정답 : (1, -1, 1)

QUIZ – Quaternion and Affine Transformation

문제 1

정답

총 1.00 점에서 1.00 점 할당

문제 표시

What is the unit quaternion $\mathbf{q}$ that rotates around the x-axis in 90 degrees, and its conjugate?

하나를 선택하세요.

a. $\frac{\sqrt{2}}{2}(-1+i), \frac{\sqrt{2}}{2}(-1-i)$

b. $\frac{\sqrt{2}}{2}(-1-i), \frac{\sqrt{2}}{2}(-1+i)$

c. $\frac{\sqrt{2}}{2}(1-i), \frac{\sqrt{2}}{2}(1+i)$

d. $\frac{\sqrt{2}}{2}(1+i), \frac{\sqrt{2}}{2}(1-i)$

✔

답이 맞습니다.

정답 : $\frac{\sqrt{2}}{2}(1+i), \frac{\sqrt{2}}{2}(1-i)$

QUIZ – Quaternion and Affine Transformation

문제 2

정답

총 1.00 점에서 1.00 점 할당

문제 표시

Rotate the quaternion $\mathbf{p} = i + j + k$ (i.e. (1, 1, 1) in 3D Euclidean coordinate) using the result of question 1.

하나를 선택하세요.

a.
$\mathbf{p}' = -i + j + k$

b.
$\mathbf{p}' = i - j + k$

✔

c.
$\mathbf{p}' = i + j - k$

d.
$\mathbf{p}' = i - j - k$

답이 맞습니다.

$\frac{\sqrt{2}}{2}(1+i)(i+j+k)\frac{\sqrt{2}}{2}(1-i) = \frac{1}{2}(i+2k-1)(1-i) = i - j + k$

정답 : $\mathbf{p}' = i - j + k$

QUIZ – Quaternion and Affine Transformation

문제 3

정답

총 1.00 점에서 1.00 점 할당

문제 표시

Convert the unit quaternion $\mathbf{q} = \frac{\sqrt{2}}{2}(1 + j)$ to a 3×3 rotational matrix.

하나를 선택하세요.

a.
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

b.
$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

c.
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

✔

d.
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

답이 맞습니다.

$w = y = \frac{\sqrt{2}}{2}, x = z = 0$

정답 : $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$

QUIZ – Quaternion and Affine Transformation

문제 4

정답

총 1.00 점에서 1.00 점 할당

문제 표시

Which of the following is not an affine transformation?

하나를 선택하세요.

- a. Orthogonal matrix
- b. Rigid body transformation
- c. A linear function of input coordinates
- d. Projective transformation ✔

답이 맞습니다.

Projective transformation generally does not preserve parallelism due to perspectivity.

정답 : Projective transformation

Open GL

OpenGL Basics

OpenGL 3.3 Basics

Basic OpenGL Pipeline

GLSL Basics

Viewing Pipeline

OpenGL Basics

1. OpenGL : 여러사람들이 공통으로 만든 graphic API에 대한 규약
- OpenGL은 c 언어 기반이지만 sample code는 c++

2. 강의에서는 3.3 버전 core을 사용
- 현재는 4.6 버전까지 나옴
- 3.3 버전은 backward compatibilit가 있어 3.3 버전을 배워도 뒷버전 쓰는데 문제 없음

3. OpenGL extensions
- 개발한 graphic feature 을 ARB(Architecture Review Board) 승인을 받기 전 사용할 수 있도록 제공하는 방법
- 문제점) OpenGL Core에 없어서 하우스 키핑 과정이 복잡함 -> function/low level call을 여러 번 해야함

4. State machine (OpenGL context)
- 한번 state를 저장하면 바꾸기 전까지 뒷 코드에도 state가 영향을 미침

OS-specific Support Libs

- OpenGL은 어떤 운영체제에서도 사용 가능하기 때문에 운영체제 의존적인 일을 해줄 방법이 필요함
- Window, canvas 만들어야 그림을 그릴 수 있는데 이것은 opengl이 해주지 않음
- 과거에는 이것들을 직접 코딩했지만 지금은 support libray를 사용하면 됨 -> GLFW, GLAD 라이브러리

GLFW

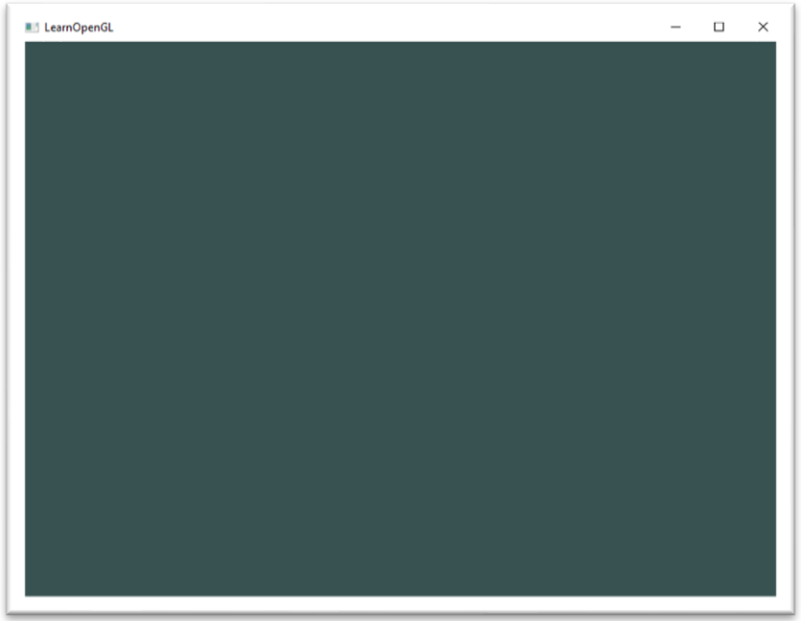
1. Window를 만드는 기능
2. User interface message handling 기능

GLAD

Extension을 사용할 때 필요한 Low level 작업

Hello Window

- Window를 만들어보자!

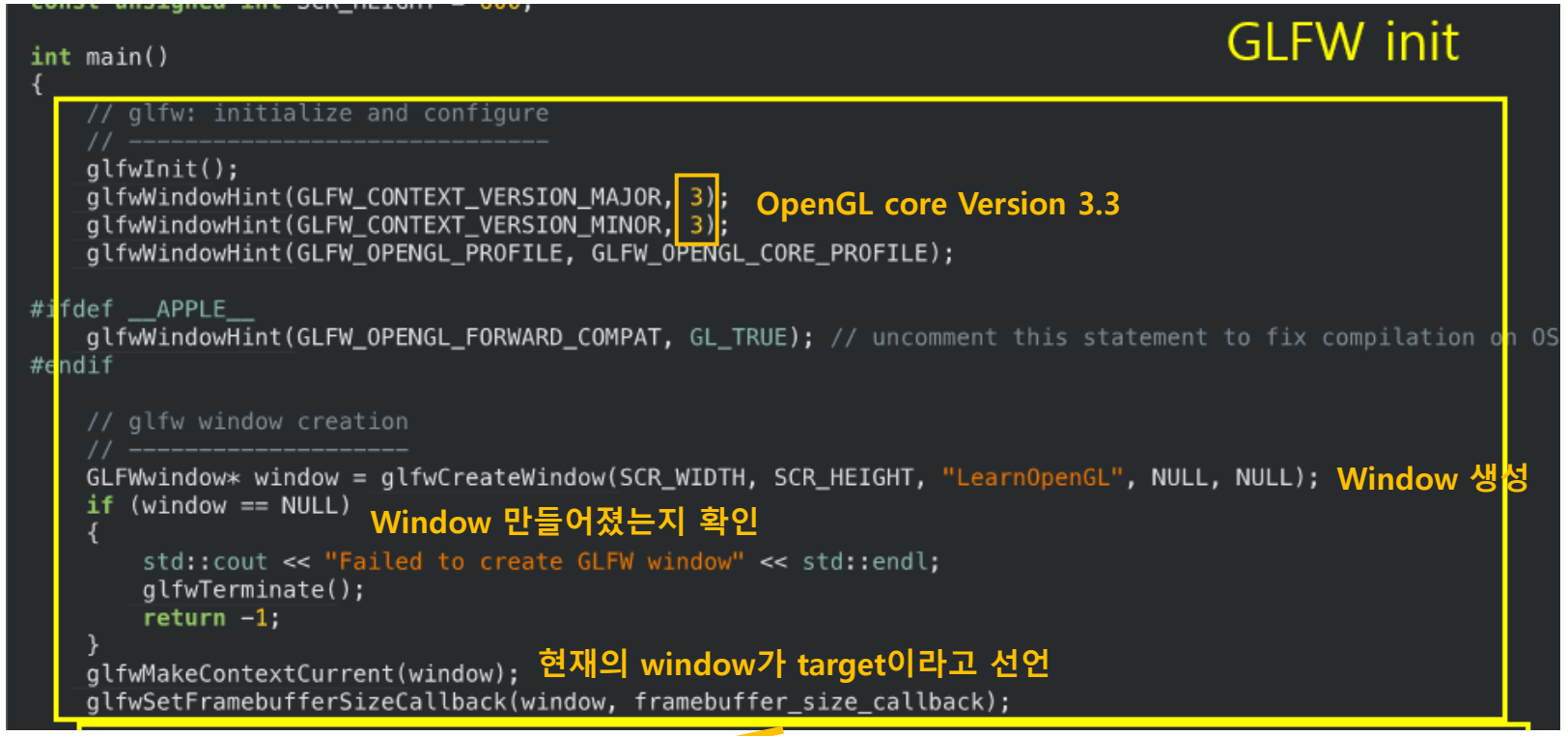


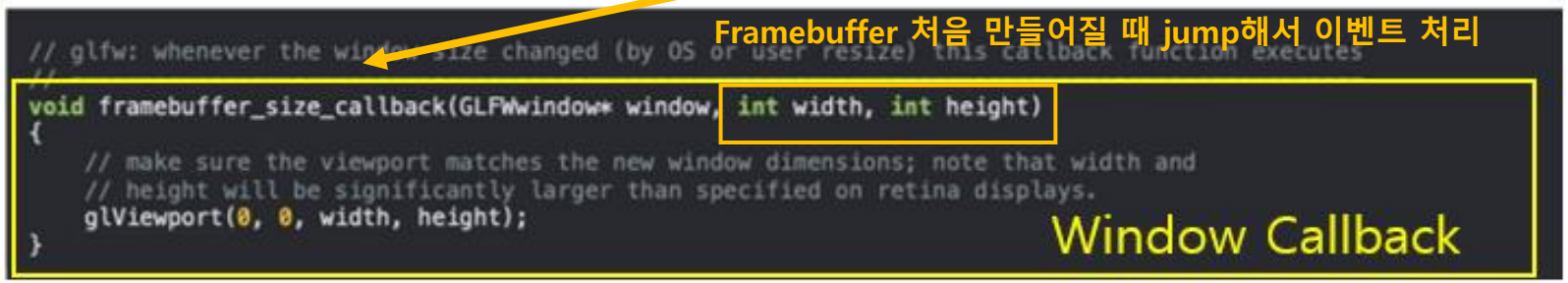
1. Header(GLFW/GLAD)
- GLFW, GLAD 헤더를 포함

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>

#include <iostream>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow *window);

// settings
const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;
```

Header (GLFW/GLAD)

2. GLFW init





GLFW init

1. **Window 생성**
2. **이벤트 메시지 처리**

Viewport 생성

- **Viewport의 width, height는 frame buffer 사이즈와 동일**

3. GLAD init + 4. Render Loop

```
// glad: load all OpenGL function pointers
// ---------------------------------------
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    std::cout << "Failed to initialize GLAD" << std::endl;
    return -1;
}
```

GLAD init

```
// render loop
// -----------
while (!glfwWindowShouldClose(window))
{
    // input
    // -----
    processInput(window);

    // render
    // ------
    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    // glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)
    // -------------------------------------------------------------------------------
    glfwSwapBuffers(window);
    glfwPollEvents();
}
```

Render Loop

Window close 될 때까지 무한 루프

1. 키보드, 마우스 같은 Interface input 처리

R g b a

2. Rendering
0~1사이 normalized 된 color 값으로 clear

3. Double buffering (swapping)

다음 이벤트 처리

```
// glfw: terminate, clearing all previously allocated GLFW resources.
// ------------------------------------------------------------------
glfwTerminate();
return 0;
}
```

Window terminate 하며 끝

```
// process all input: query GLFW whether relevant keys are pressed/released this frame and react accordingly
// ---------------------------------------------------------------------------------------------------------
void processInput(GLFWwindow *window)
{
    if(glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}
```

Escape 키 눌렀는지 확인 -> true : close (loop 정지)

GLAD init

- 여러 function pointer들 불러옴

Render Loop

- 무엇을 렌더링할지 정하는 부분으로 우리가 중점적으로 신경써야 하는 부분

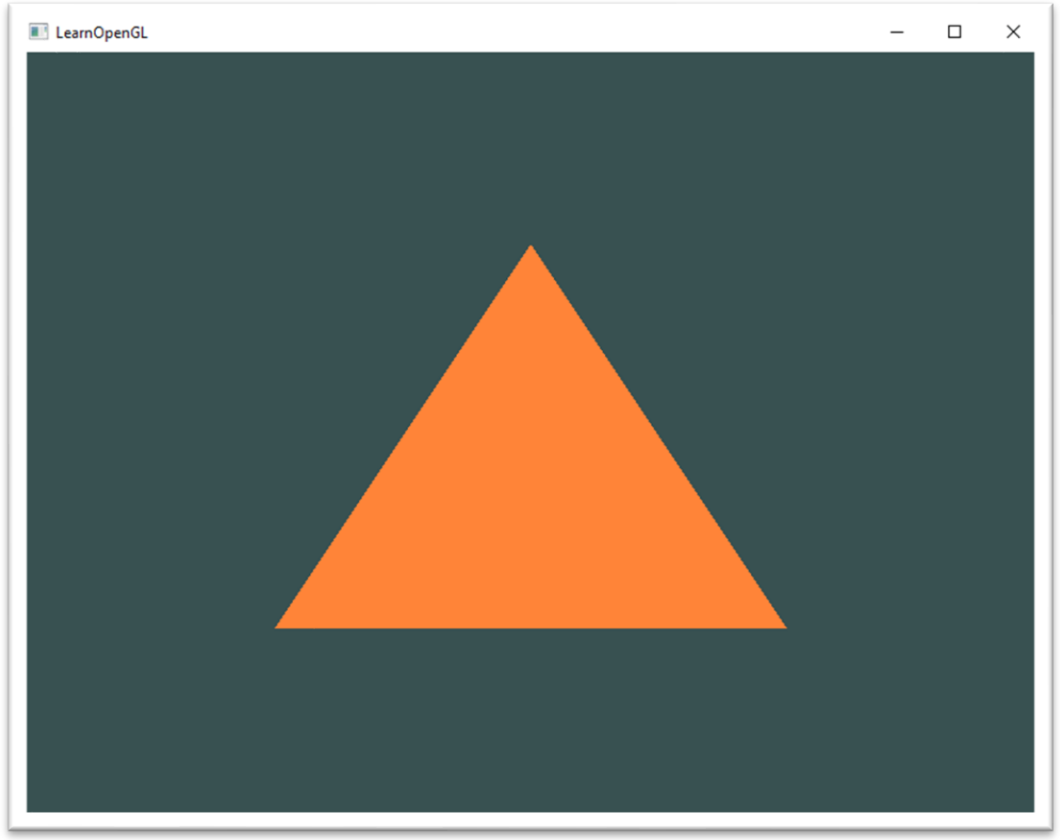
과정

1. 사용자 input 처리
2. Rendering
3. Double buffering

* 지금까지의 코드들은 대부분 기계적으로 들어가야 하는 내용으로 바꿀일이 별로 없음

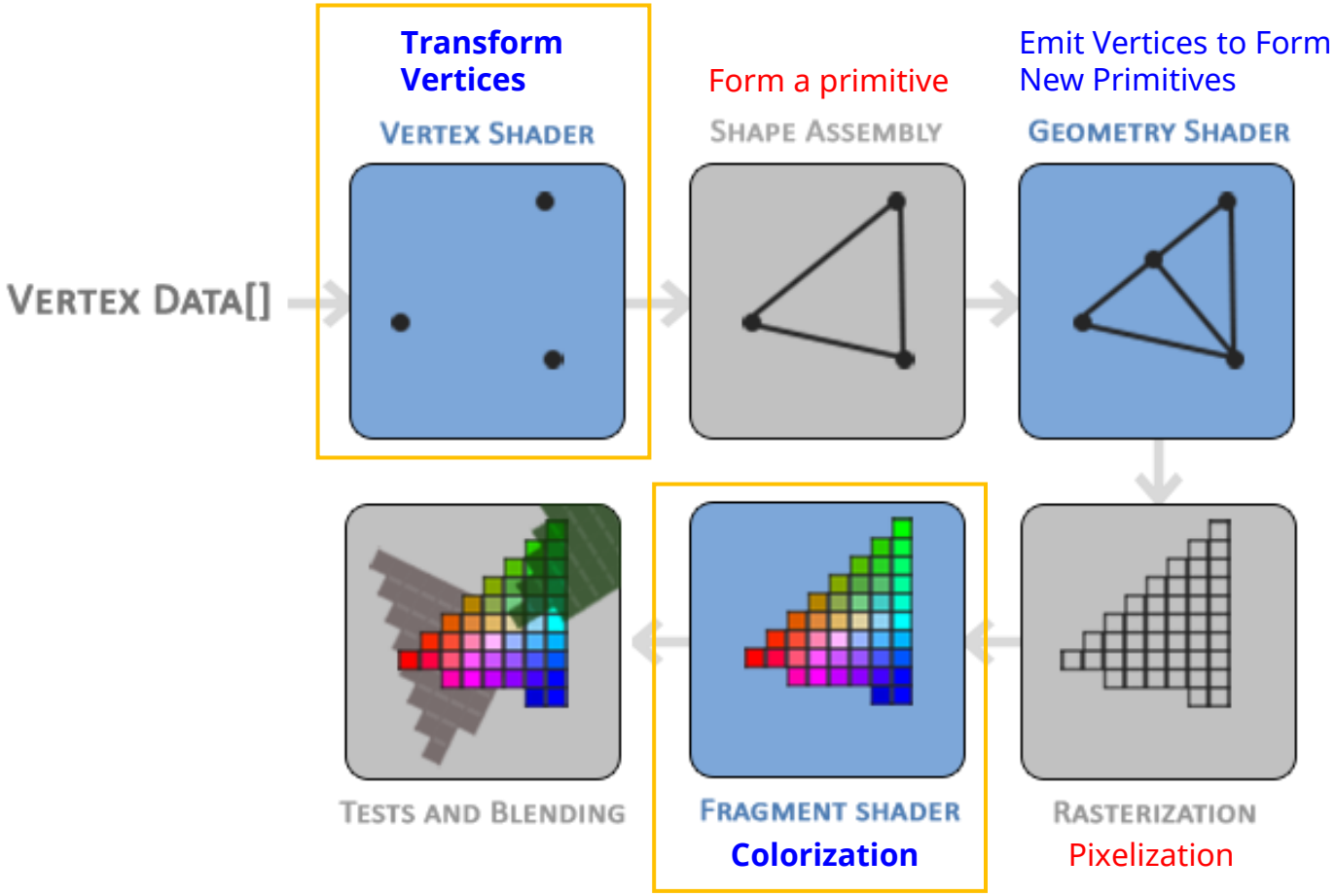
Hello Triangle

- Triangle을 그려보자 -> OpenGL graphics pipeline에 대해 알아야함

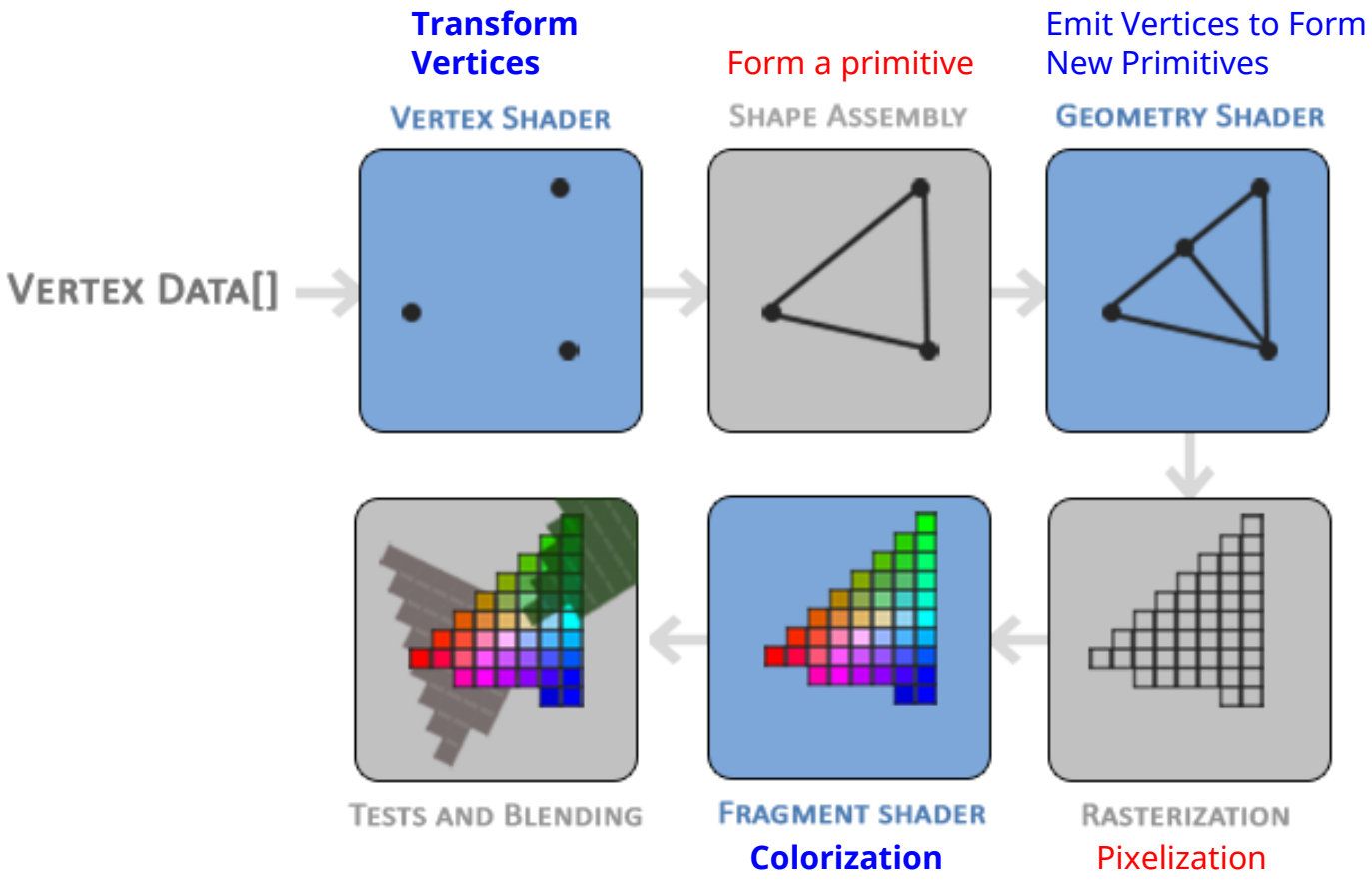


OpenGL Graphics Pipeline

- Basic concept에서 배운 pipeline보다 세분화 되어 있음
- Input은 3차원 꼭짓점 데이터로 position외의 color, texture data 도 들어갈 수 있음
- 파란색 부분은 shader(GLSL)를 사용해서 구현



OpenGL Graphics Pipeline



1. Vertex shader
- Transform vertices
- Modeling, viewing, projection, viewport transformation이 발생
- 각 vertice에 개별적, 병렬적으로 적용
- 점 정보로는 여러 해석이 가능하기 때문에 shape assembly 단계가 필요

2. Shape Assembly
- 점으로 어떤 shape를 만드는지 결정

3. Geometry Shader
- 수업시간에는 다루지 않음
- Primitive를 만드는 과정

4. Rasterization
- 렌더링하기 위해 pixelization

5. Fragment Shader
- Colorization 즉, Pixel의 색을 정함

6. Tests and Blending
- 물체가 여러 개일 경우 뒤덮을 때 어디가 가려지는지, blending되는지 결정

Rendering Loop

삼각형 그리는 절차

1. 2개의 shader(vertex, fragment) 중 어느 shader를 사용할 것인지 결정
2. Vertex data를 넘겨줌
3. 그리기

```
// render loop
// -----------
while (!glfwWindowShouldClose(window))
{
    // input
    // -----
    processInput(window);

    // render
    // ------
    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    // draw our first triangle
    glUseProgram(shaderProgram);
    glBindVertexArray(VAO); // seeing as we only have a single VAO there's no need to bind it every time, but we'll
    glDrawArrays(GL_TRIANGLES, 0, 3);
    // glBindVertexArray(0); // no need to unbind it every time

    // glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)
    // -------------------------------------------------------------------------------
    glfwSwapBuffers(window);
    glfwPollEvents();
}
```

1. Specify shaders to use
2. Provide vertex data
3. Draw

Vertex/fragment shader 정의 + shader Load & compile

Character string

```
const char *vertexShaderSource = "#version 330 core\n"
    "layout (location = 0) in vec3 aPos;\n"
    "void main()\n"
    "{\n"
    "   gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"
    "}\0";
const char *fragmentShaderSource = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = vec4(1.0f, 0.5f, 0.2f, 1.0f);\n"
    "}\n\0";
```

개별 파일

```
#version 330 core
layout (location = 0) in vec3 aPos;

void main()
{
    gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);
}
```

Vertex shader
- 이 코드에선 Vertex input을 그대로 출력
- 즉, transformation이 없는 상태

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0f, 0.5f, 0.2f, 1.0f);
}
```

Fragment shader
- R=1.0, G=0.5, B=0.2 인 색으로 통일

2. vertexShaderSource 불러오기

```
// build and compile our shader program
// ------------------------------------
// vertex shader
int vertexShader = glCreateShader(GL_VERTEX_SHADER);    1. Vertex shader 만들기
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);    3. compile
// check for shader compile errors
int success;
char infoLog[512];
glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
if (!success)
{
    glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
    std::cout << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n" << infoLog << std::endl;
}
// fragment shader
int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);        Fragmentshader에서도 동일한 과정 거침
glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
glCompileShader(fragmentShader);
// check for shader compile errors
glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);
if (!success)
{
    glGetShaderInfoLog(fragmentShader, 512, NULL, infoLog);
    std::cout << "ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n" << infoLog << std::endl;
}
```

Shader Load & Compile

Vertex/fragment shader 정의
- 보통은 개별 파일로 저장하지만 이 코드에서는 character string 형태로 정의

Shader Load&Compile
- Rendering loop 들어가기 전에 shader를 load, compile 해주어야 함

Link Shader

```
int shaderProgram = glCreateProgram(); 전체 shader 프로그램 만들기
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram); link
// check for linking errors
glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);
if (!success) {
    glGetProgramInfoLog(shaderProgram, 512, NULL, infoLog);
    std::cout << "ERROR::SHADER::PROGRAM::LINKING_FAILED\n" << infoLog << std::endl;
}
glDeleteShader(vertexShader);
glDeleteShader(fragmentShader);
```

Link Shader

전 페이지에서 정의한 vertexShader, fragmentShader 사용

```
glUseProgram(shaderProgram);
glBindVertexArray(VAO); // VAO 쓰겠다고 선언
glDrawArrays(GL_TRIANGLES, 0, 3); Rendering
// glBindVertexArray(0); // no need to unbind it every time
```

1. Specify shaders to use
2. Provide vertex data
3. Draw

Link Shader

- 어떤 shader를 link 하냐에 따라 서로 다른 shader를 사용할 수 있음

Main rendering loop 내부

- 위에서 정의한 ShaderProgram 쓸 것을 선언
- Rendering 할 준비 완료
- 2. Provide vertex data가 필요
  - 다음 페이지에서 구체적 설명
- VAO 쓰겠다고 선언
- Rendering
  - GL_TRIANGLES : 삼각형을 그릴 것을 shape assembly

Prepare Vertex Data

```
// set up vertex data (and buffer(s)) and configure vertex attributes
// ------------------------------------------------------------------
float vertices[] = {
    -0.5f, -0.5f, 0.0f, // left
     0.5f, -0.5f, 0.0f, // right
     0.0f,  0.5f, 0.0f  // top
};

unsigned int VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
// bind the Vertex Array Object first, then bind and set vertex buffer(s), and then configure vertex attributes(s).
glBindVertexArray(VAO);

glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);

// note that this is allowed, the call to glVertexAttribPointer registered VBO as the vertex attribute's bound vert
glBindBuffer(GL_ARRAY_BUFFER, 0);

// You can unbind the VAO afterwards so other VAO calls won't accidentally modify this VAO, but this rarely happens.
// VAOs requires a call to glBindVertexArray anyways so we generally don't unbind VAOs (nor VBOs) when it's not dire
glBindVertexArray(0);
```

Prepare Vertex Data

좌표

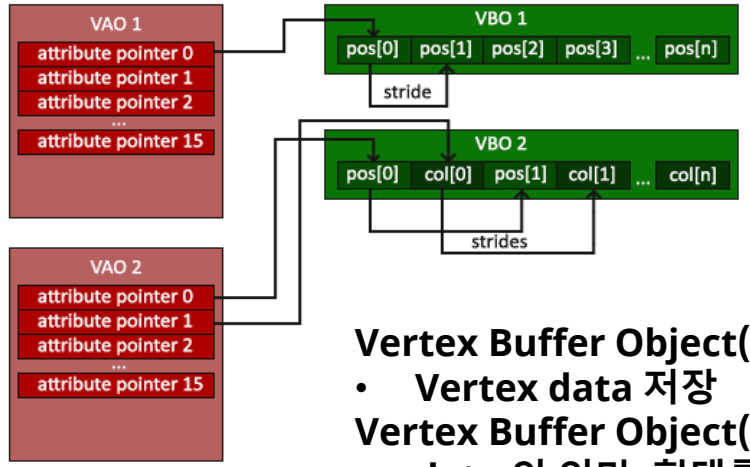
1. Bind VAO

2. Bind VBO, vertices 데이터를 VBO에 복사

3. VAO가 VBO의 데이터 형태를 선언 (vertex attribute pointers)

Prepare Vertex Data

- Vertex data 전달 해주어야 함
- VBO, VAO 사용
- Rendering을 할 때는 실제로 VAO 사용



Vertex Buffer Object(VBO)

- Vertex data 저장

Vertex Buffer Object(VAO)

- data 의 의미, 형태를 저장

Sequence

1. Bind VAO
2. Bind VBO
3. Configure vertex attributes in VBO using VAO
4. Drawing using VAO

Prepare Vertex Data

```
// set up vertex data (and buffer(s)) and configure vertex attributes
// ------------------------------------------------------------------
float vertices[] = {
    -0.5f, -0.5f, 0.0f, // left
     0.5f, -0.5f, 0.0f, // right
     0.0f,  0.5f, 0.0f  // top
};

unsigned int VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
// bind the Vertex Array Object first, then bind and set vertex buffer(s), and then configure vertex attributes(s).
glBindVertexArray(VAO);

glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);

// note that this is allowed, the call to glVertexAttribPointer registered VBO as the vertex attribute's bound vertex
glBindBuffer(GL_ARRAY_BUFFER, 0);

// You can unbind the VAO afterwards so other VAO calls won't accidentally modify this VAO, but this rarely happens.
// VAOs requires a call to glBindVertexArray anyways so we generally don't unbind VAOs (nor VBOs) when it's not dire
glBindVertexArray(0);
```

좌표

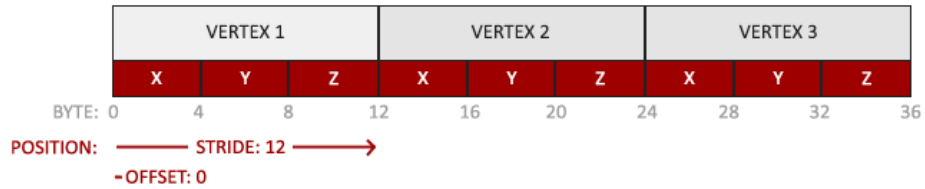
Prepare Vertex Data

1. Bind VAO

2. Bind VBO, vertices 데이터를 VBO에 복사

3. VAO가 VBO의 데이터 형태를 선언 (vertex attribute pointers)

Attribute 1개, component 3개(x,y,z), 데이터 타입, normalize 여부, stride, offset



Vertex는 이러한 형태로 VBO에 들어감

Stride: 첫번째 vertex의 시작으로부터 두번째 vertex 시작까지 건너 뛰어야 하는 양
offset: 첫번째 vertex가 시작하는 위치

Sequence

1. **Bind VAO**
2. **Bind VBO**
3. **Configure vertex attributes in VBO using VAO**
4. **Drawing using VAO**

Attribute가 여러 개인 경우

```
GLfloat data[] = {
  // Position  // Color          // TexCoords
  1.0f, 0.0f,  0.5f, 0.5f, 0.5f, 0.0f, 0.5f,
  0.0f, 1.0f,  0.2f, 0.8f, 0.0f, 0.0f, 1.0f
};
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE,
        7 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
        7 * sizeof(GLfloat), (GLvoid*)(2 * sizeof(GLfloat)));
glEnableVertexAttribArray(2);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE,
        7 * sizeof(GLfloat), (GLvoid*)(5 * sizeof(GLfloat)));
```

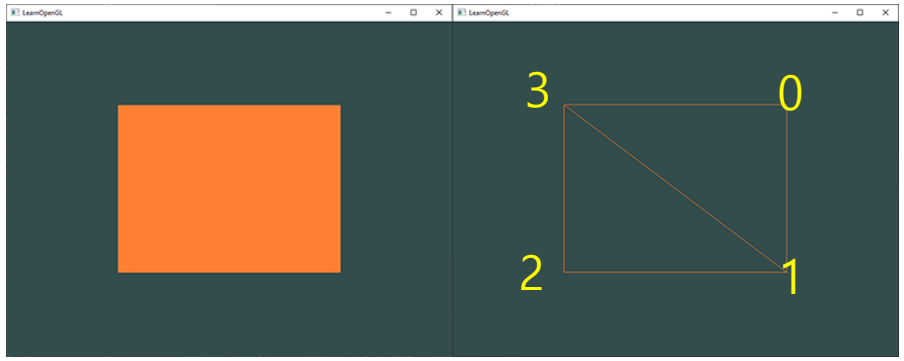
position

color

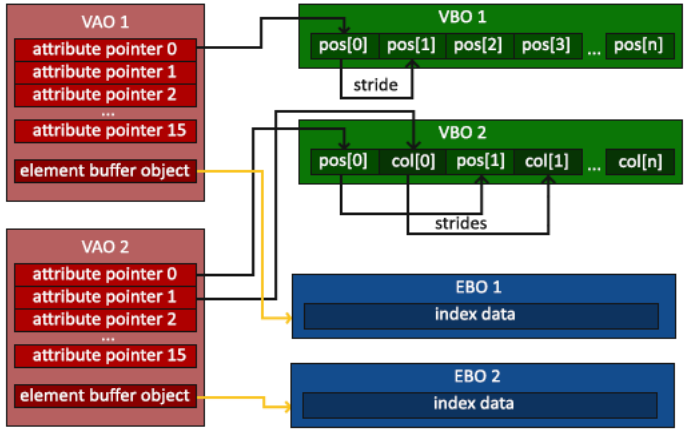
texcoords

Indexed Face-set Rendering

- 렌더링 할 때 direct vertex position으로 렌더링 하지 않고 index를 사용
- 사각형을 그려보자



- 사각형은 삼각형 2개로 나눠서 그림
- 이 경우 중복되는 꼭짓점이 생기고 이 vertex들에 대해 여러 번 데이터를 보내야하는 비효율적인 상황 발생
  -> index를 사용
- 앞의 코드에서는 VBO에 vertex position을 저장했고 이 코드에서는 추가로 EBO에 index를 저장함
- VAO가 VBO,EBO 모두를 point 함



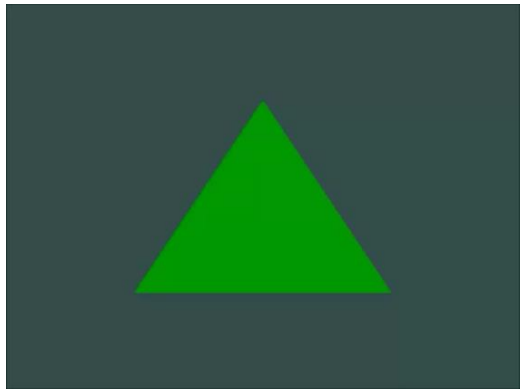

```
float vertices[] = {
     0.5f,  0.5f, 0.0f,  // top right
     0.5f, -0.5f, 0.0f,  // bottom right
    -0.5f, -0.5f, 0.0f,  // bottom left
    -0.5f,  0.5f, 0.0f   // top left
};
unsigned int indices[] = {  // note that we start from 0!
    0, 1, 3,   // first triangle
    1, 2, 3    // second triangle
};
```

```
// ..:: Initialization code :: ..
// 1. bind Vertex Array Object
glBindVertexArray(VAO);
// 2. copy our vertices array in a vertex buffer for OpenGL to use
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
// 3. copy our index array in a element buffer for OpenGL to use
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
// 4. then set the vertex attributes pointers
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);

[...]

// ..:: Drawing code (in render loop) :: ..
glUseProgram(shaderProgram);
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0)
glBindVertexArray(0);
```

Index copy

OpenGL Shading Language Basics

- Vertex shader, fragment shader : 실질적으로 프로그래밍을 많이 할 곳
- 시간에 따라 계속 색이 변하는 삼각형을 그려보자!



Typical stucture

```
#version version_number
in type in_variable_name;
in type in_variable_name;

out type out_variable_name;

uniform type uniform_name;

void main()
{
  // process input(s) and do some weird graphics stuff
  ...
  // output processed stuff to output variable
  out_variable_name = weird_stuff_we_processed;
}
```

1. **버전 선언**
2. **변수 선언**
- in : shader의 input
- out : shader의 output
- uniform : shader 뿐만 아니라 opengl host도 접근 가능한 글로벌 변수
3. **메인**
- Shader function

Vector Type

```
vec2 someVec;
vec4 differentVec = someVec.xyxx;  x,y -> xyxx, 2차원->4차원      //swizzling
vec3 anotherVec = differentVec.zyw;   Xyzw -> zyw , 4차원->3차원
vec4 otherVec = someVec.xxxx + anotherVec.yxzy;
```

```
vec2 vect = vec2(0.5, 0.7);
vec4 result = vec4(vect, 0.0, 0.0);
vec4 otherResult = vec4(result.xyz, 1.0);
```

- Swizzling operation: dimension 쉽게 늘리고 줄일 수 있게 해주는 연산

- vecn: the default vector of n floats
- n에 따라 component가 몇 개인지 결정
- component 1-x, 2-y, 3-z, 4-w
- bvecn: a vector of n booleans
- ivecn: a vector of n integers
- uvecn: a vector of n unsigned integers
- dvecn: a vector of n double precisions

Matrix Type

- matn: $n \times n$ matrix of floats
- dmatn: $n \times n$ matrix of doubles

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoord;

out vec2 TexCoord;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
        gl_Position = projection * view * model * vec4(aPos, 1.0f);
        TexCoord = vec2(aTexCoord.x, aTexCoord.y);
}
```

In and Out Variables

- 실제 코딩할 때는 shader 파일 (.vs, .fs) 파일을 따로 만든다

Vertex shader

```
#version 330 core  버전을 선언
layout (location = 0) in vec3 aPos; // the position variable has attribute position 0

out vec4 vertexColor; // specify a color output to the fragment shader

void main()
{
    gl_Position = vec4(aPos, 1.0); // see how we directly give a vec3 to vec4's constructor
    vertexColor = vec4(0.5, 0.0, 0.0, 1.0); // set the output variable to a dark-red color
}
```

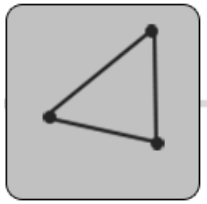
Fragment shader

```
#version 330 core
out vec4 FragColor;

in vec4 vertexColor; // the input variable from the vertex shader (same name and same type)

void main()
{
    FragColor = vertexColor;
}
```

Vertex, pixel이 1:1 대응이 되지 않는 내부의 점들은 interpolation

Vertex shader
- location
- 몇번째 데이터를 사용하는가
-location=0 첫번째 데이터 사용 (position x,y,z)
- location=1 두번째 데이터 사용 (color r,g,b)
- gl_Position
- 무조건 출력해야 하는 output
- modeling, viewing, projection 한 결과

Fragment shader
- vertexColor
- Vertex shader의 output vertex color를 input으로 받음



Uniform Variable

Fragment shader 코드

```
#version 330 core
out vec4 FragColor;

uniform vec4 ourColor; // we set this variable in the OpenGL code.

void main()
{
    FragColor = ourColor;
}
```

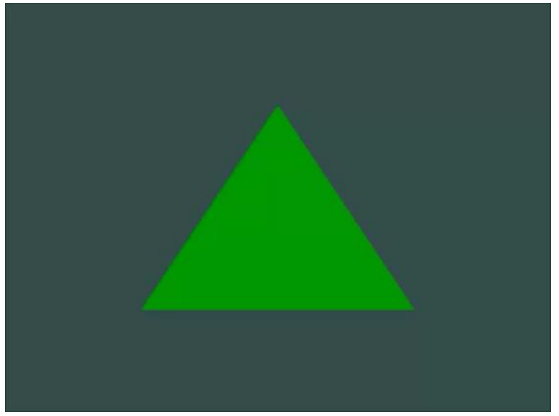
Uniform 변수 ourColor

OpenGL 코드

```
float timeValue = glfwGetTime();
float greenValue = (sin(timeValue) / 2.0f) + 0.5f;
int vertexColorLocation = glGetUniformLocation(shaderProgram, "ourColor");
glUseProgram(shaderProgram);
glUniform4f(vertexColorLocation, 0.0f, greenValue, 0.0f, 1.0f);
```

ourColor에 접근

ourColor에 setting

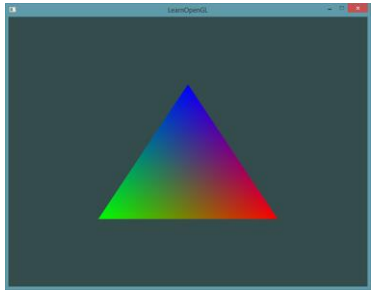


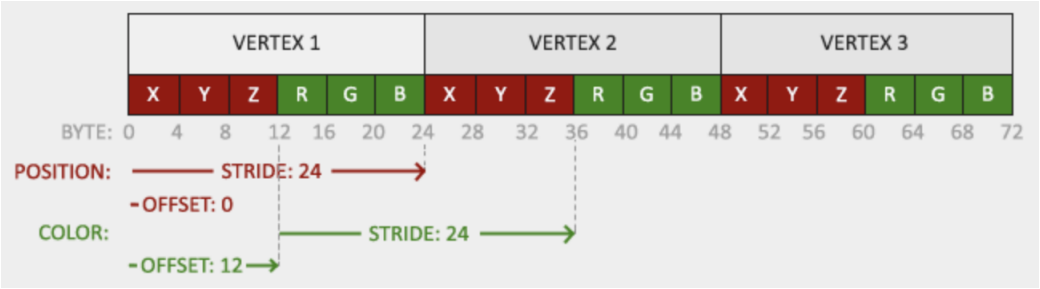
R=0.0, g=greenValue로 계속 바꿔줌, B=0.0, a=1.0
-> 현재 시간에 따라 색이 바뀌는 삼각형 완성

More : 삼각형을 무지개 색으로 바꿔주자!

```
float vertices[] = { 꼭짓점마다 다른 color의 vertex set up
    // positions         // colors
     0.5f, -0.5f, 0.0f,  1.0f, 0.0f, 0.0f,   // bottom right
    -0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f,   // bottom left
     0.0f,  0.5f, 0.0f,  0.0f, 0.0f, 1.0f    // top
};
```

Vertex data in host with positions/colors





VBO memory layout

```
// position attribute                    Color 추가로 stride=6
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
// color attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3* sizeof(float)));
glEnableVertexAttribArray(1);
```

Configure VAO

```
#version 330 core                  모든 vertex에 대해 병렬적으로 진행
layout (location = 0) in vec3 aPos;   // the position variable has attribute position 0
layout (location = 1) in vec3 aColor; // the color variable has attribute position 1

out vec3 ourColor; // output a color to the fragment shader

void main()
{
    gl_Position = vec4(aPos, 1.0);
    ourColor = aColor; // set ourColor to the input color we got from the vertex data
}
```

Vertex Shader

```
#version 330 core                  내부 값은 color interpolation
out vec4 FragColor;
in vec3 ourColor;

void main()
{
    FragColor = vec4(ourColor, 1.0);
}
```

Fragment Shader

shader.h

- shader 파일 (.vs, .fs) 은 따로 만드는게 보편적
- File open/load 과정이 필요하고 shader.h 헤더는 이 과정을 도와줌

```
#include <learnopengl/shader_s.h>
```

Include

```
// build and compile our shader program
// ------------------------------------
Shader ourShader("3.3.shader.vs", "3.3.shader.fs"); // you can name your shader files however you like
```

Load and compile

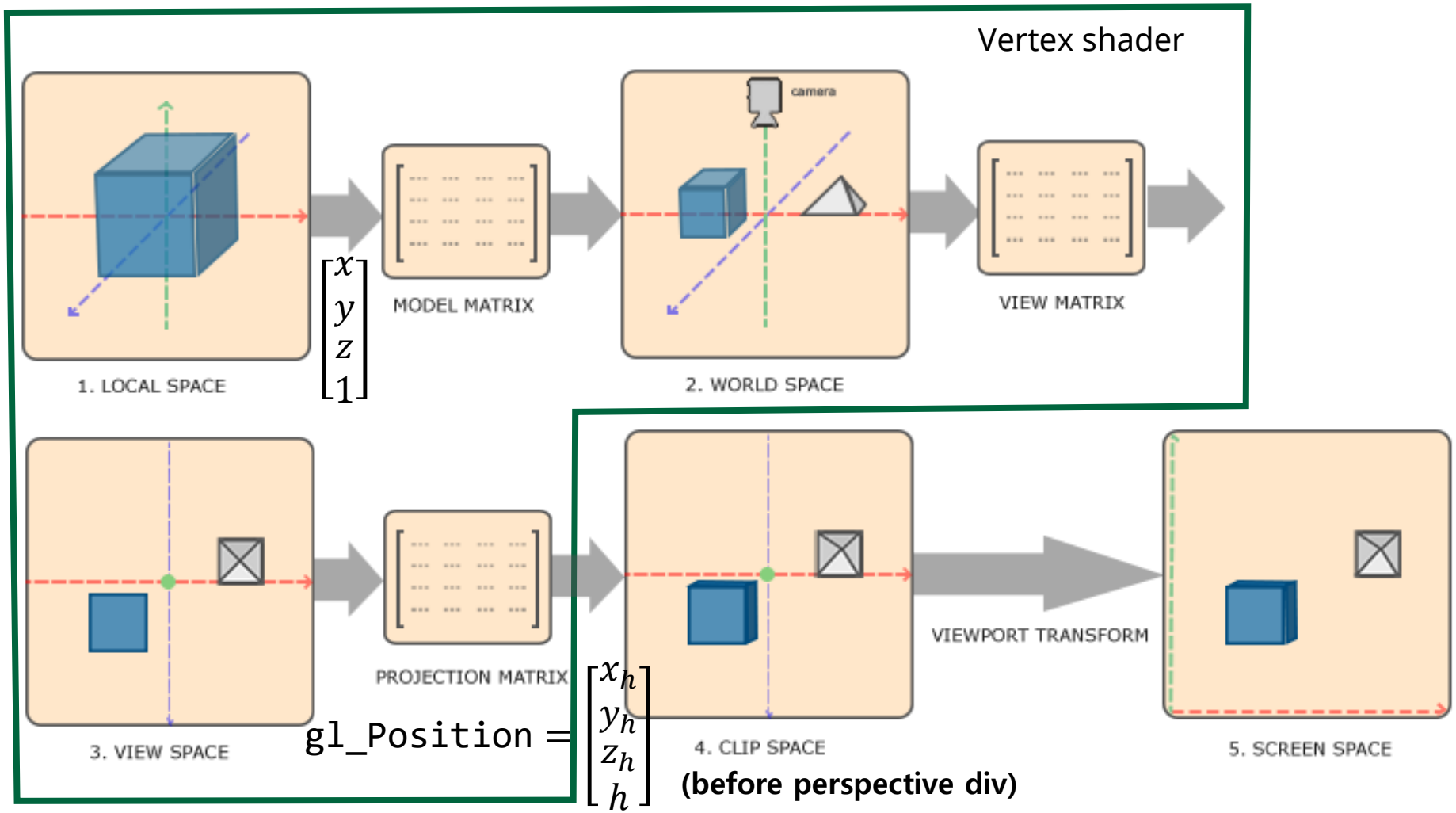
Vertex/ fragment shader 중 무엇인지 class를 선언해야함

```
// render the triangle
ourShader.use();
glBindVertexArray(VAO);
```

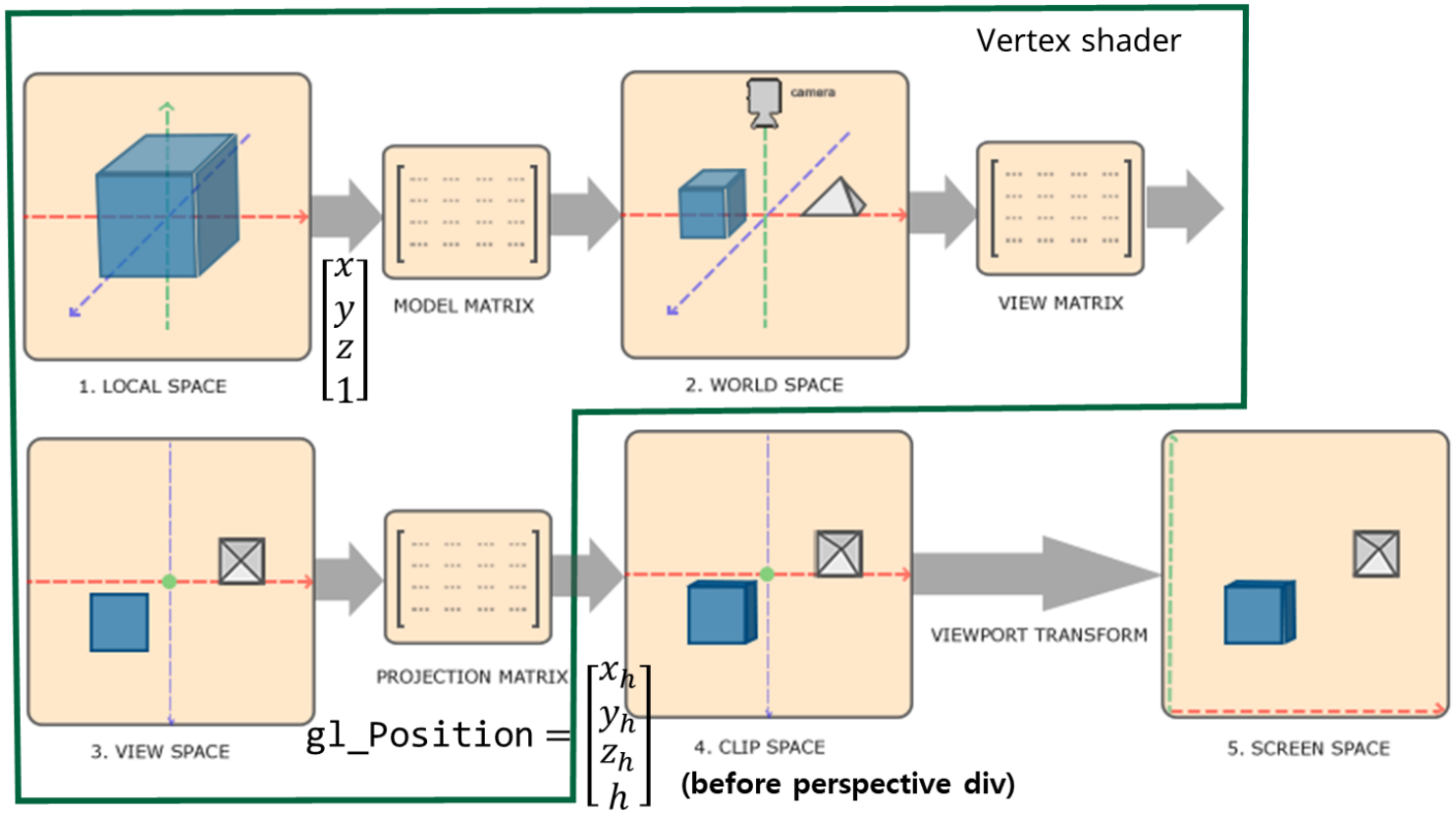
Use

Viewing pipeline

- Basic concept 시간에 본 pipeline과 동일
- Projection까지가 vertex shader에서 하는 일
- input, output은 둘 다 homogeneous coordinate
- 각 matrix들을 곱해가면서 계산



Viewing pipeline



Model Matrix
- Rotation, translation, scaling 등의 행렬

View matrix
- Camera 기준으로 물체 위치 결정

Projection matrix
- 원근감 projection
- h는 보통 1이 아니고 h로 나누기 전의 space를 clip space라고 함
- h로 나누는 건 perspective division이라고 함

회전하는 3D 육면체를 만들어보자!

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
```

GLM include

```
float vertices[] = {
    -0.5f, -0.5f, -0.5f,  0.0f, 0.0f,
     0.5f, -0.5f, -0.5f,  1.0f, 0.0f,
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
    -0.5f,  0.5f, -0.5f,  0.0f, 1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, 0.0f,

    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
     0.5f, -0.5f,  0.5f,  1.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 1.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 1.0f,
    -0.5f,  0.5f,  0.5f,  0.0f, 1.0f,
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,

    -0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
    -0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
    -0.5f,  0.5f,  0.5f,  1.0f, 0.0f,

     0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
     0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
     0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
     0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 0.0f,

    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
     0.5f, -0.5f, -0.5f,  1.0f, 1.0f,
     0.5f, -0.5f,  0.5f,  1.0f, 0.0f,
     0.5f, -0.5f,  0.5f,  1.0f, 0.0f,
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,

    -0.5f,  0.5f, -0.5f,  0.0f, 1.0f,
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
    -0.5f,  0.5f,  0.5f,  0.0f, 0.0f,
    -0.5f,  0.5f, -0.5f,  0.0f, 1.0f
};
unsigned int VBO, VAO;
```

36 vertices (6F*2Tri*3Verts) defined in modeling coordinate

GLM include

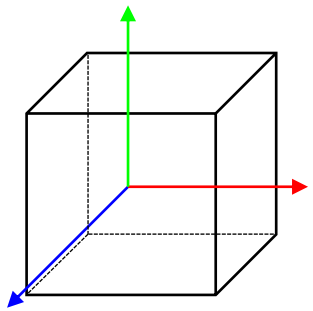
- GLM 라이브러리(OpenGL Mathematics library)를 사용
- Linear algebra 등 수학 포함

Data

- 육면체의 꼭짓점 데이터를 input으로 받아야 함
- Index face-set 대신 vertex position을 넣어주는 방식 사용
- Modeling coordinate 기준 좌표
- 6개의 면 – 하나의 면은 2개의 삼각형

-> 6x2x3=36 개의 좌표 필요

- 이 좌표들을 vertex shader에 넘겨줌

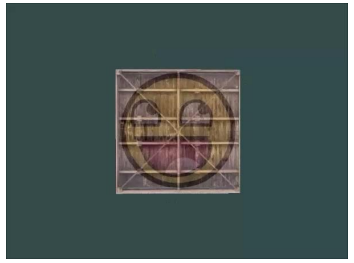


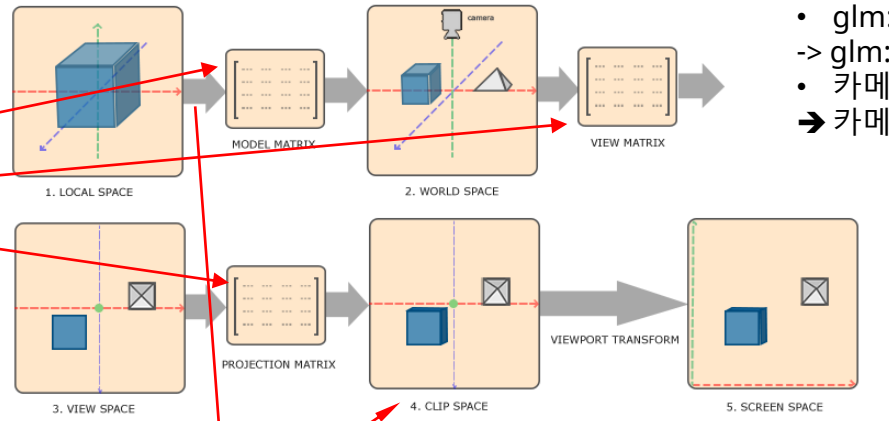
```
// render loop
// -----------
while (!glfwWindowShouldClose(window))
{
    // input
    // -----
    processInput(window);

    // render
    // ------
    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // also clear the depth buffer now!

    // bind textures on corresponding texture units
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture1);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, texture2);

    // activate shader
    ourShader.use();

    // create transformations 단위 행렬로 initialize
    glm::mat4 model         = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
    glm::mat4 view          = glm::mat4(1.0f); 회전각도
    glm::mat4 projection    = glm::mat4(1.0f);
    model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(0.5f, 1.0f, 0.0f)); 회전축
    view  = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
    projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);
    // retrieve the matrix uniform locations
    unsigned int modelLoc = glGetUniformLocation(ourShader.ID, "model");
    unsigned int viewLoc  = glGetUniformLocation(ourShader.ID, "view");
    // pass them to the shaders (3 different ways)
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &view[0][0]);
    // note: currently we set the projection matrix each frame, but since the projection matrix rarely changes it's of
    ourShader.setMat4("projection", projection);

    // render box
    glBindVertexArray(VAO);
    glDrawArrays(GL_TRIANGLES, 0, 36);


    // glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)
    // -------------------------------------------------------------------------------
    glfwSwapBuffers(window);
    glfwPollEvents();
}
```

model=modelxR

FOV : field of view angle
Aspect ratio
0.1f: 카메라-near plane 거리
100.f0f: far plane 까지의 거리

OpenGL 코드에서 Uniform 변수
model, view, projection 넘겨줌





Model
- glm::rotate 함수 사용
- 회전각도, 회전축 정보를 담고 있음
- model=modelxR

View
- 카메라의 위치 결정
- glm::translate 함수 사용

-> glm::lookAt 함수를 사용 가능
- 카메라의 default position = 원점

➔ 카메라의 위치 z 방향으로 3 이동

Vertex shader

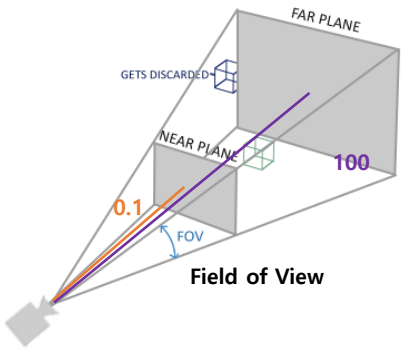
```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoord;

out vec2 TexCoord;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
        gl_Position = projection * view * model * vec4(aPos, 1.0f);
        TexCoord = vec2(aTexCoord.x, aTexCoord.y);
}
```

Projection
- Perspective, orthographic 2가지가 있으나 이 코드에선 glm::perspective 사용
- FOV, aspect ratio, near plane/far plane 까지 거리 정보를 담고 있음
- View volume 밖은 clipping

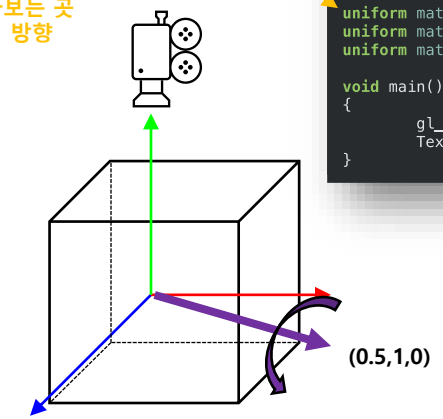
This is equivalent to:

```
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));
```

카메라 위치
카메라가 쳐다보는 곳
카메라의 머리 방향



Field of View



***코드에서 model transform 2번 이상일때**
- 코드 순서가 rotate – translate면 실제 순서는 translate - rotate

= 공전 효과
- 코드 순서가 translate – rotate면 실제 순서는 rotate - translate

= 자전 효과