Getting Started (4)

GPU Programming

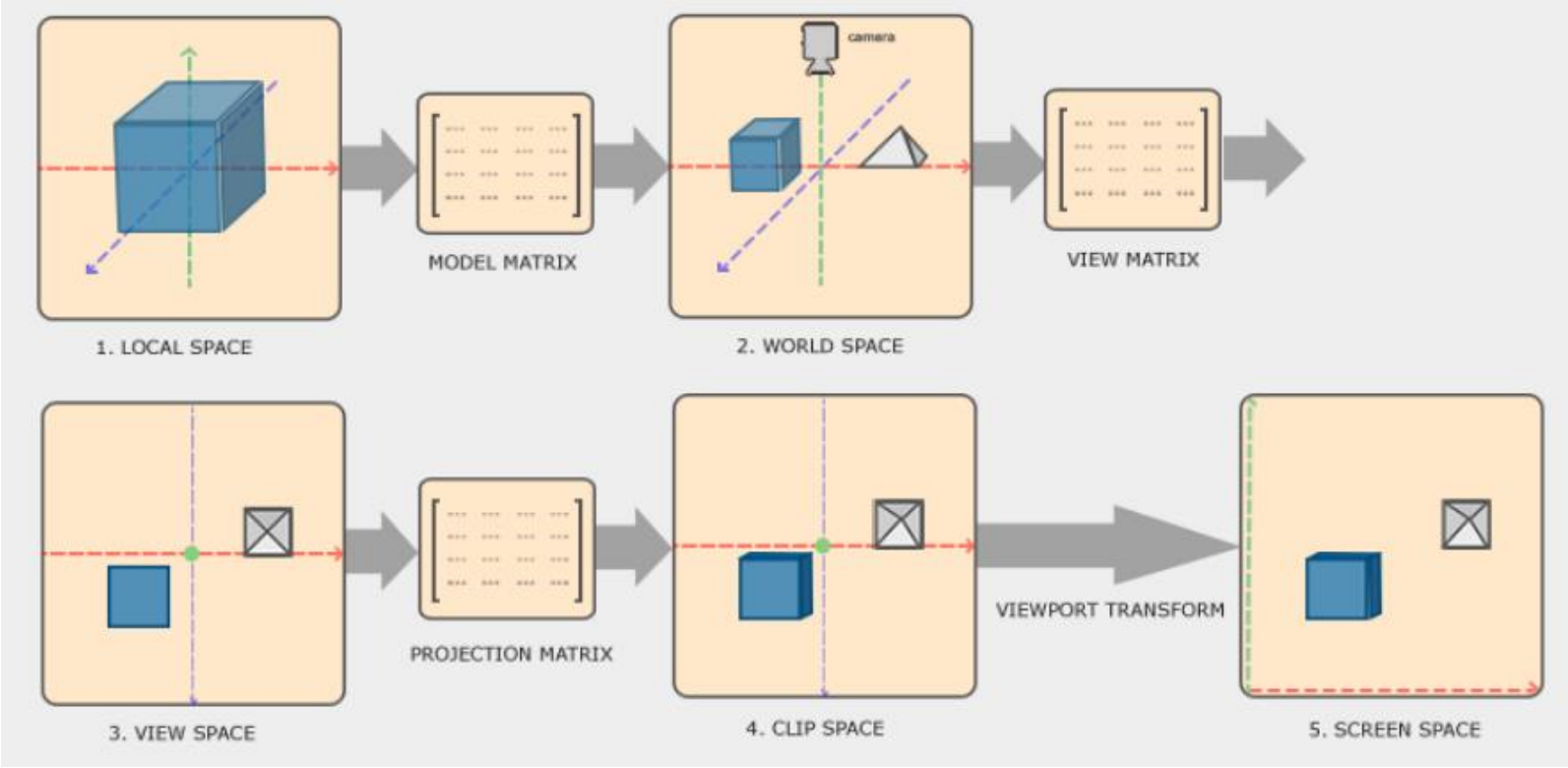
2022학년도
2학기

ⓒ nvidia

Coordinate Systems

The Global Picture

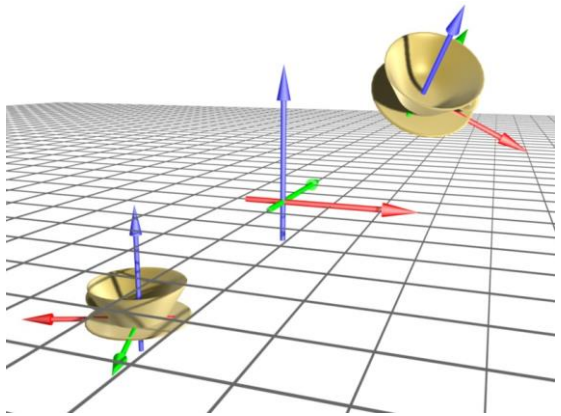
- 지난 시간까지는 [-1, 1] 범위의 NDC(normalized device coordinates) 상의 삼각형을 그려 봤습니다.
- 그렇다면 좀더 복잡한, 실제로 화면을 그리는데 쓰이는 객체(object)는?
  - MVP (model-view-projection) 행렬을 이용하여 clip space 상의 NDC로 변환한 후,
  - Viewport transform을 이용하여 이를 최종적으로 screen space로 변환



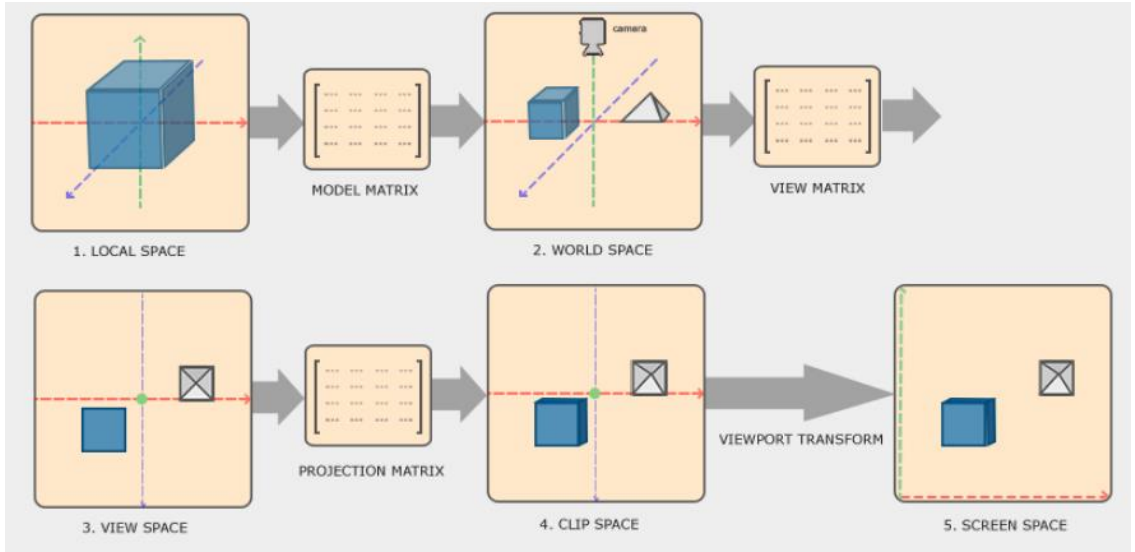
Local Space

- 객체에 대한 좌표 공간
  - 객체의 원점을 (0, 0, 0)으로 놓은 공간
  - 모델의 모든 vertex는 이 local space를 기준으로 함
- 지난주 사용한 나무 컨테이너 객체의 vertex들도 local space에 존재

```
float vertices[] = {
    // positions          // colors           // texture coords
     0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f,   // top right
     0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,   // bottom right
    -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,   // bottom left
    -0.5f,  0.5f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f    // top left
};
```

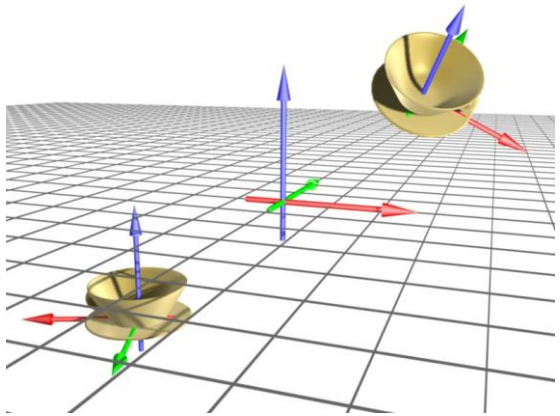


Blender 3D: Noob to Pro/Coordinate Spaces in Blender - Wikibooks, open books for an open world

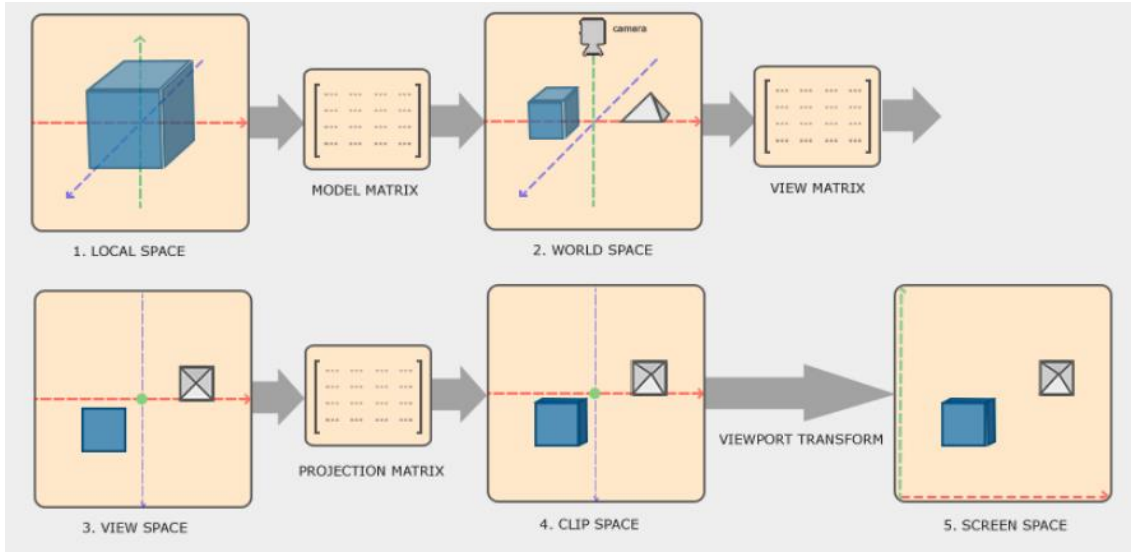


World Space

- Local space 상의 객체를, 실제 world space 상에서 이 객체가 위치하는 곳으로 배치
  - 지난 시간에 배운 이동, 스케일, 회전 정보가 담긴 model 행렬을 이용
  - 이 행렬을 수정하면, 매 프레임마다 객체의 위치를 변화시키거나 확대/축소시킬 수 있음



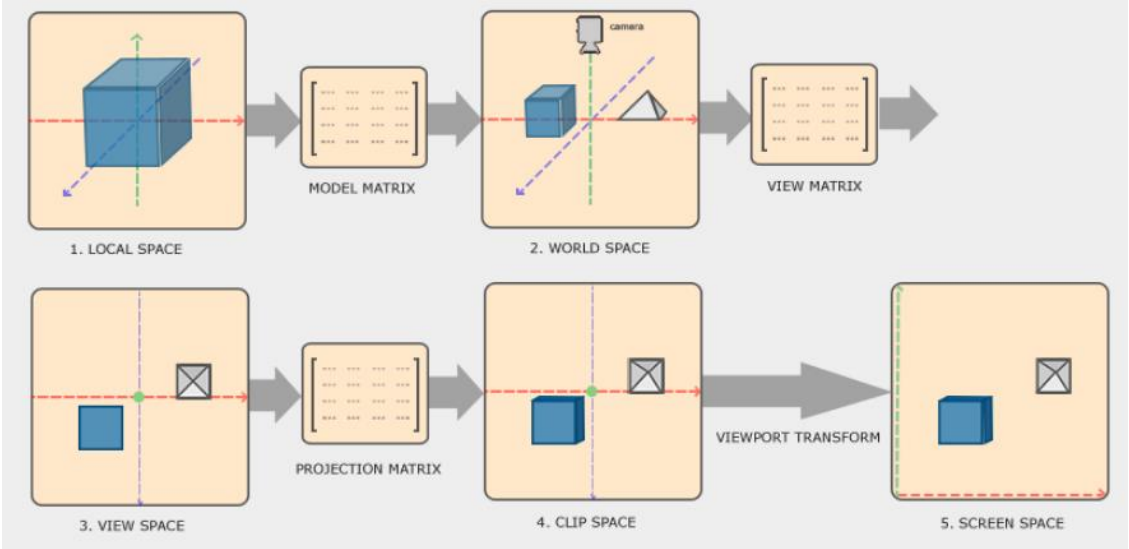
Blender 3D: Noob to Pro/Coordinate Spaces in Blender - Wikibooks, open books for an open world



View Space

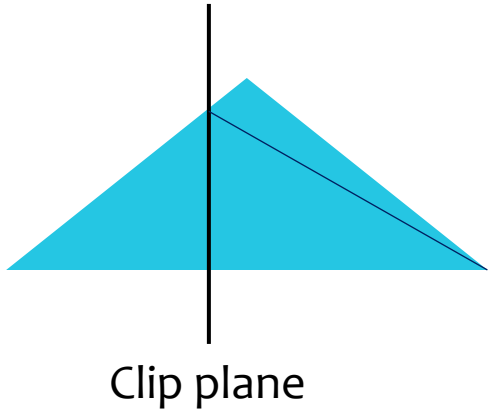
- 사용자의 시점, 즉 카메라의 관점에서 바라보는 공간
  - Camera space 또는 eye space라고도 함

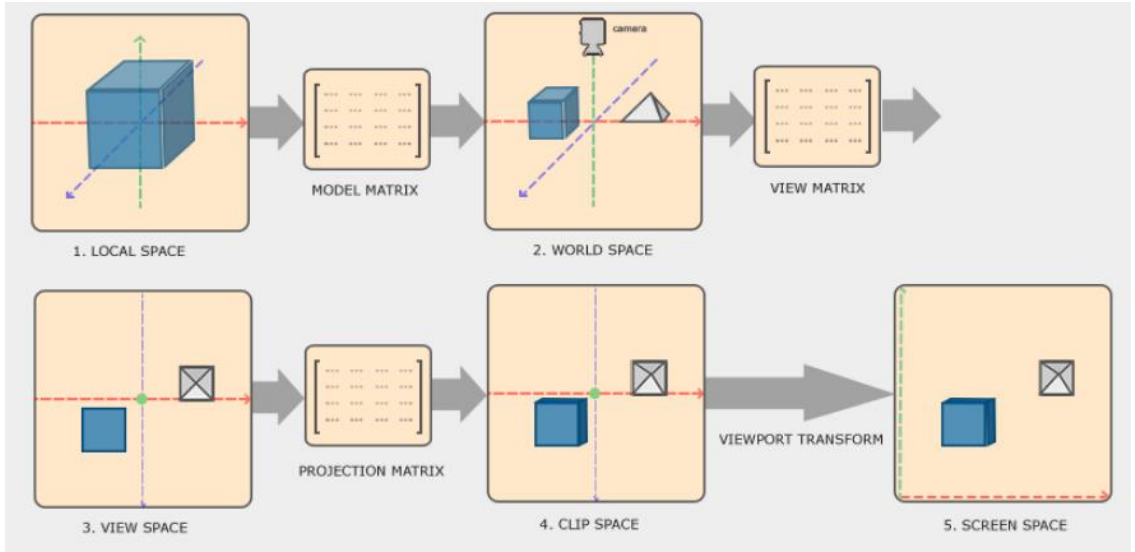
- View 행렬을 이용하여 객체들을 view space로 변환
  - View 행렬에는 객체들의 이동 및 회전 정보가 포함됨
  - 즉, OpenGL과 같은 rasterization 기반 시스템은 카메라의 위치가 바뀌면 이에 따라 객체의 좌표가 바뀜
  - 참고로, ray tracing에서는 객체는 그대로 world space 상에 있지만, camera ray의 원점 및 방향이 달라짐



Clip Space

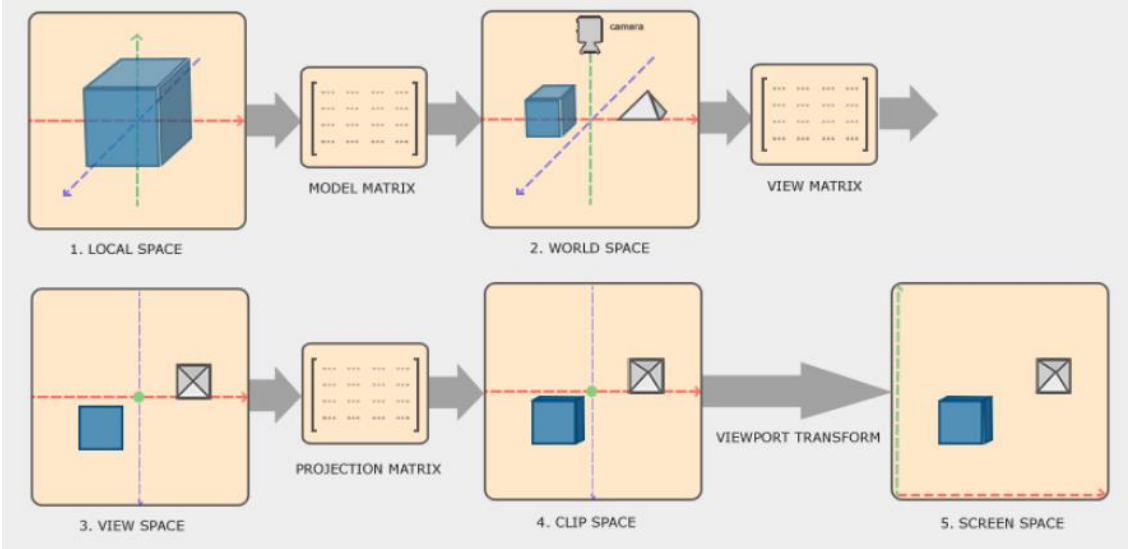
- Projection(투영) 행렬을 이용하여 view space를 clip space로 변환
  - 각 축에 대해 범위를 지정
  - 지정된 범위에 있는 좌표는 NDC [-1.0, 1.0]로 변환
  - 지정된 범위에서 벗어난 좌표는 clipping함 (잘라냄)
- 예를 들어 각 축의 투영 범위가 [-1000, 1000] 이라면?
  - 좌표 (1250, 500, 750)는 화면 밖에 있는 것으로 판단되어 걸러짐
- 만약 삼각형과 같은 primitive의 일부만 화면에서 벗어났을 경우
  - Clipping 범위에 맞게 primitive를 재구성
    (추가 삼각형이 생성될 수 있음)





Clip Space

- Frustum (절두체)
  - Projection 행렬이 만드는 view box를 frustum이라고 함
  - 투영 방식에 따라 직교(orthographic) 투영 행렬과 원근(perspective) 투영 행렬 중 하나 선택
  - 전자의 경우 frustum이 직육면체 모양으로, 후자의 경우 사각뿔 윗면을 자른 모양이 됨
- Perspective division
  - 모든 vertex들이 clip space로 변환된 후 수행되는 마지막 단계
  - 위치 벡터의 x, y, z 요소들을 벡터의 w 요소로 나눔 → 원근 투영 행렬 사용시 원근감 생성
  - 4D clip space 좌표가 3D NDC로 최종 변환

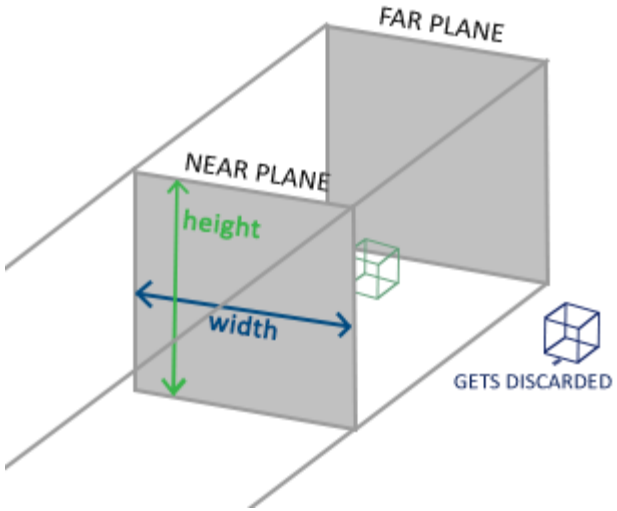


Orthographic Projection (직교 투영)

- 원근감을 고려하지 않는 투영
  - 멀리 있는 물체와 가까이 있는 물체가 1:1로 투영
  - 2D/3D 도면이나 top-view 스타일의 2D 게임에서 사용

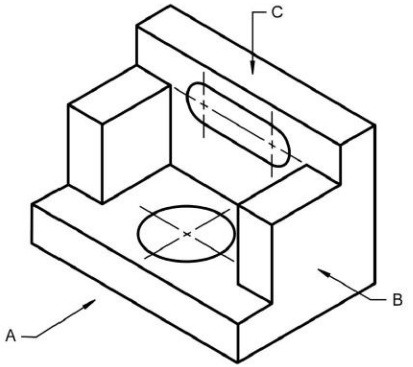
- 직교 투영 행렬

$$\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



- glm의 ortho() 함수를 통한 행렬 생성 방법
  - 각 파라미터: left, right, bottom, top, near, far

```
glm::ortho(0.0f, 800.0f, 0.0f, 600.0f, 0.1f, 100.0f);
```

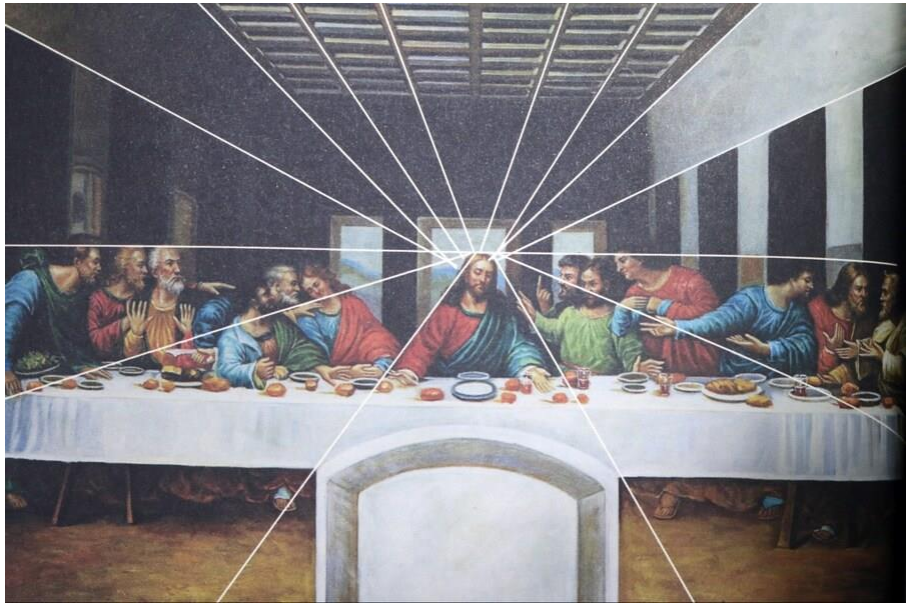


실습 과제 03- 중급 직교 투영(정투상법) : 네이버 블로그 (naver.com)

Fantastic Top Down Games Everyone Should Play (thegamer.com)

Perspective Projection (원근 투영)

- 원근감을 고려하는 투영
  - 가까이 있는 물체일수록 크게 보이는 방법
  - 대부분의 3D 앱에서 사용

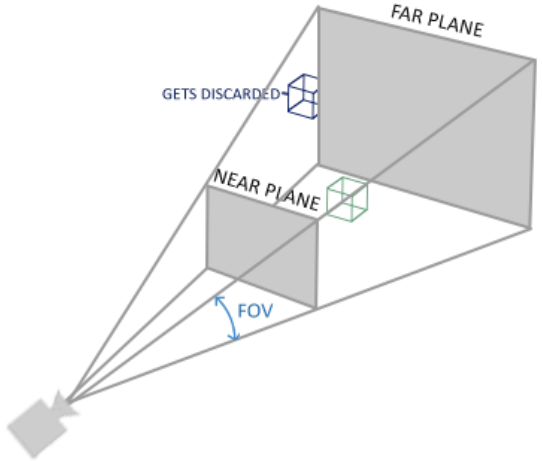


원근법이 적용된 레오다르도 다빈치의 <최후의 만찬>(1490)

[알라딘서재]레오나르도 다빈치의 창조성이 그를 천재로 만들었다! (aladin.co.kr)

- 원근 투영 행렬

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

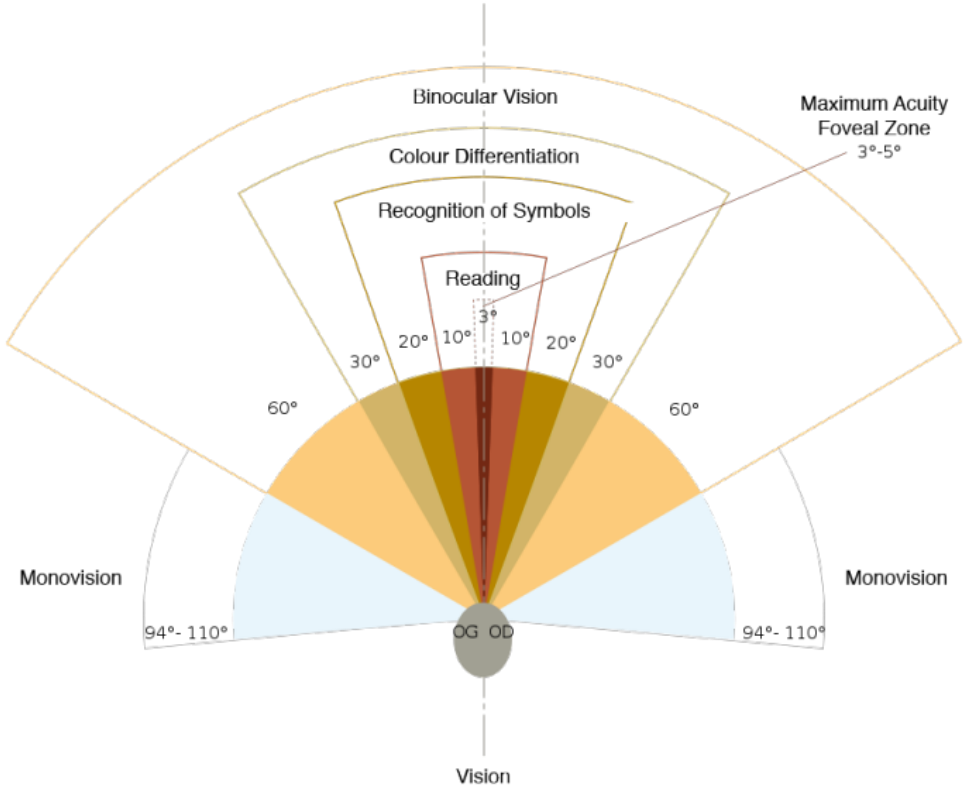


- glm의 perspective() 함수를 통한 생성 방법
  - 각 파라미터: fovy(y축 방향의 field of view), aspect ratio, near, far

```
glm::mat4 proj = glm::perspective(glm::radians(45.0f), (float)width/(float)height, 0.1f, 100.0f);
```

Field of View (화각/시야각)

- FOV가 클수록 더 넓은 영역을 표현 가능하나, 원근 왜곡도 발생 가능
  - 스마트폰의 광각 카메라를 생각해 보세요
- 참고로 사람 눈의 시야각은 아래와 같음

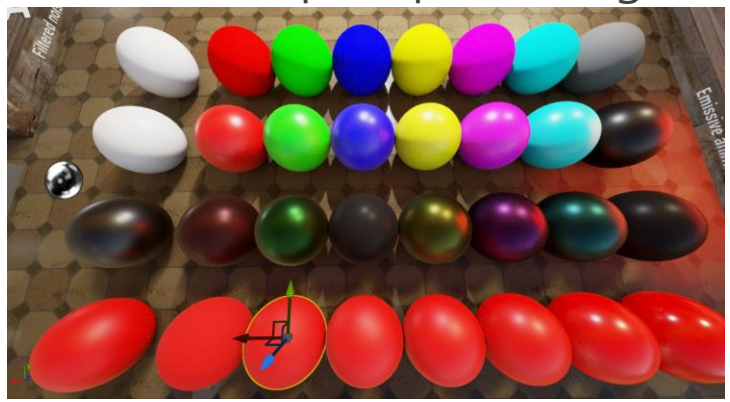


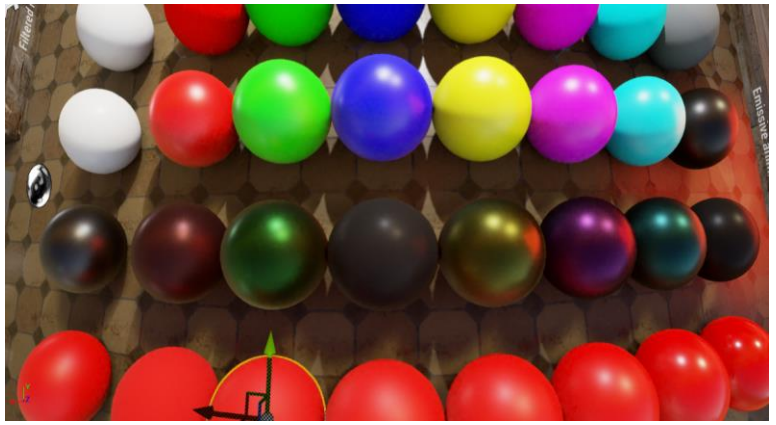
[KoSyoU] 사람 눈의 시야각을 렌즈 기준.. : 네이버블로그 (naver.com)

콜 오브 듀티: 월드 워 2 75종 그래픽카드 성능 분석 : 네이버 포스트 (naver.com)

Field of View (화각/시야각)

- 최신 유니티/언리얼에서는 넓은 화각에서의 원근 왜곡을 방지하기 위한 Pannini Projection 지원
  - panini.pdf (tksharpless.net)
  - 별도 후처리(post-processing)를 통해 구현되므로, 연산량은 그만큼 증가

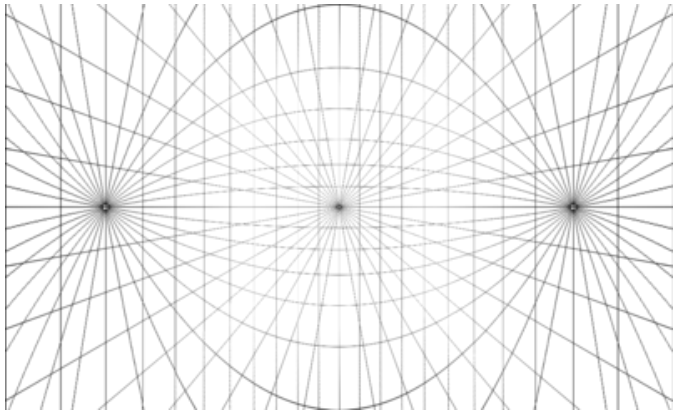




FOV 125도

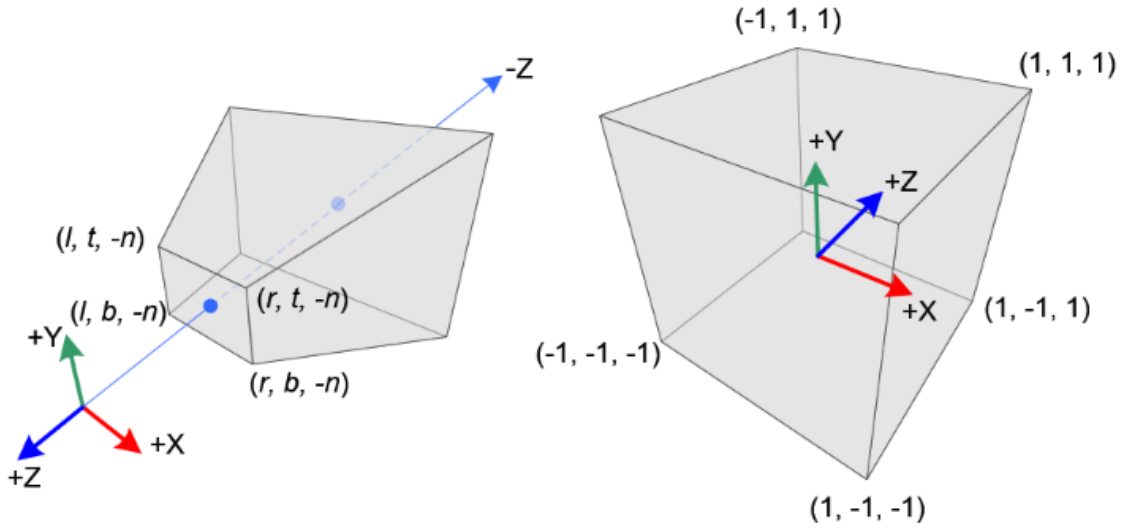
파니니 투영 | 언리얼 엔진 문서 (unrealengine.com)

Gian Paolo Pannini, Interior San Pietro (1754)



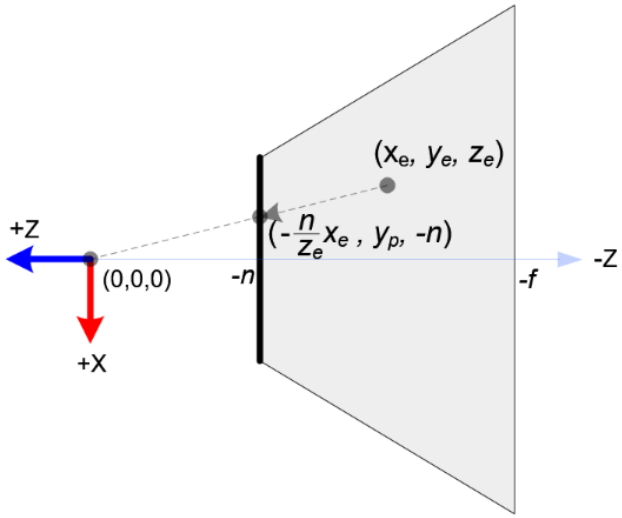
The General Panini Projection - PanoTools.org Wiki

Perspective Projection Matrix

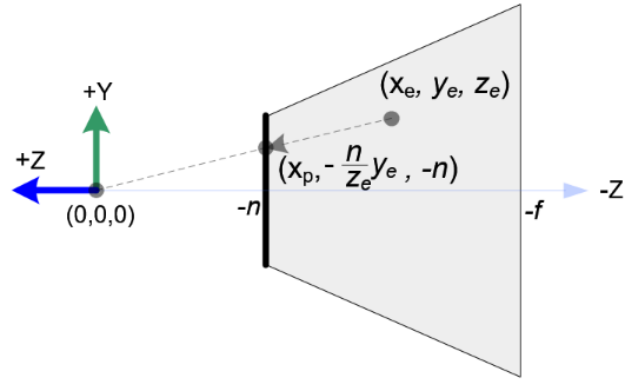


Perspective Frustum and Normalized Device Coordinates (NDC)

- Eye space 상의 점 (x_e, y_e, z_e)를 *near* space 상의 점 (x_p, y_p, z_p)으로 투영



Top View of Frustum



Side View of Frustum

$$\frac{x_p}{x_e} = \frac{-n}{z_e}$$

$$x_p = \frac{-n \cdot x_e}{z_e} = \frac{n \cdot x_e}{-z_e}$$

$$\frac{y_p}{y_e} = \frac{-n}{z_e}$$

$$y_p = \frac{-n \cdot y_e}{z_e} = \frac{n \cdot y_e}{-z_e}$$

OpenGL Projection Matrix (songho.ca)

Perspective Projection Matrix

- 이 결과를 이용해, 투영된 이후 clip space 상의 w_c 의 값을 구할 수 있음

$$\frac{x_p}{x_e} = \frac{-n}{z_e} \qquad \frac{y_p}{y_e} = \frac{-n}{z_e}$$

$$x_p = \frac{-n \cdot x_e}{z_e} = \frac{n \cdot x_e}{-z_e} \qquad y_p = \frac{-n \cdot y_e}{z_e} = \frac{n \cdot y_e}{-z_e}$$

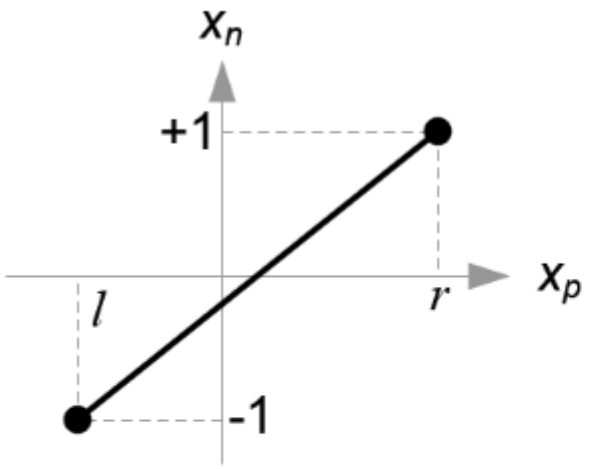
$$\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = M_{projection} \cdot \begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix}, \quad \begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{pmatrix}$$

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}, \qquad \therefore w_c = -z_e$$

OpenGL Projection Matrix (songho.ca)

Perspective Projection Matrix

- 다음으로, 투영된 x_p를 NDC 상의 x_n으로 mapping(사상) 가능 (y축도 마찬가지)



Mapping from x_p to x_n

$$x_n = \frac{1-(-1)}{r-l} \cdot x_p + \beta$$

$$1 = \frac{2r}{r-l} + \beta \qquad (\text{substitute } (r,1) \text{ for } (x_p, x_n))$$

$$\beta = 1 - \frac{2r}{r-l} = \frac{r-l}{r-l} - \frac{2r}{r-l}$$

$$= \frac{r-l-2r}{r-l} = \frac{-r-l}{r-l} = -\frac{r+l}{r-l}$$

$$\therefore x_n = \frac{2x_p}{r-l} - \frac{r+l}{r-l}$$

OpenGL Projection Matrix (songho.ca)

Perspective Projection Matrix

- x_n식의 x_p를 기존에 구한
  eye space 상의 값으로 대입 (y축도 마찬가지)

$$
\begin{aligned}
x_n &= \frac{2x_p}{r-l} - \frac{r+l}{r-l} \qquad \left(x_p = \frac{nx_e}{-z_e}\right) \\
&= \frac{2 \cdot \dfrac{n \cdot x_e}{-z_e}}{r-l} - \frac{r+l}{r-l} \\
&= \frac{2n \cdot x_e}{(r-l)(-z_e)} - \frac{r+l}{r-l} \\
&= \frac{\dfrac{2n}{r-l} \cdot x_e}{-z_e} - \frac{r+l}{r-l} \\
&= \frac{\dfrac{2n}{r-l} \cdot x_e}{-z_e} + \frac{\dfrac{r+l}{r-l} \cdot z_e}{-z_e} \\
&= \Bigg(\underbrace{\frac{2n}{r-l} \cdot x_e + \frac{r+l}{r-l} \cdot z_e}_{x_c} \Bigg) \Big/ -z_e
\end{aligned}
$$

OpenGL Projection Matrix (songho.ca)

Perspective Projection Matrix

- 앞서 구한 x_c, y_c값을 투영 행렬에 반영

$$x_n = \underbrace{\left(\frac{2n}{r-l} \cdot x_e + \frac{r+l}{r-l} \cdot z_e \right)}_{x_c} \Big/ -z_e$$

$$y_n = \underbrace{\left(\frac{2n}{t-b} \cdot y_e + \frac{t+b}{t-b} \cdot z_e \right)}_{y_c} \Big/ -z_e$$

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

OpenGL Projection Matrix (songho.ca)

Perspective Projection Matrix

- 투영 행렬의 세번째 행의 값을 구하기 위해, 미지수 A와 B를 설정

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix},$$

$$z_n = z_c / w_c = \frac{Az_e + Bw_e}{-z_e}$$

$$z_n = \frac{Az_e + B}{-z_e} \quad (\because w_e = 1)$$

OpenGL Projection Matrix (songho.ca)

Perspective Projection Matrix

- A와 B를 구하기 위해, (z_e, z_n) 관계를 사용. (-n, -1) 및 (-f, 1)

$$\begin{cases} \dfrac{-An+B}{n} = -1 \\ \dfrac{-Af+B}{f} = 1 \end{cases} \rightarrow \begin{cases} -An+B = -n & (1) \\ -Af+B = f & (2) \end{cases}$$

$$z_n = \frac{Az_e + B}{-z_e}$$

$$= \frac{-\frac{f+n}{f-n}z_e - \frac{2fn}{f-n}}{-z_e} \qquad (3)$$

To solve the equations for *A* and *B*, rewrite eq.(1) for B;

$$B = An - n \qquad (1')$$

Substitute eq.(1') to *B* in eq.(2), then solve for A;

$$-Af + (An - n) = f \qquad (2)$$

$$-(f-n)A = f + n$$

$$A = -\frac{f+n}{f-n}$$

$$B = -n - \left(\frac{f+n}{f-n}\right)n = -\left(1 + \frac{f+n}{f-n}\right)n = -\left(\frac{f-n+f+n}{f-n}\right)n$$

$$= -\frac{2fn}{f-n}$$

OpenGL Projection Matrix (songho.ca)

Perspective Projection Matrix

- 최종 원근 투영 행렬

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

$$z_n = \frac{-\frac{f+n}{f-n} z_e - \frac{2fn}{f-n}}{-z_e} \qquad (3)$$

- 직교 투영 행렬의 유도 과정도 이와 유사하나 더 간단함 (아래 링크 참고)

OpenGL Projection Matrix (songho.ca)

원근 / 직교 투영 비교

- 유니티 카메라 모드 비교체험(Camera Projection Perspective/Orthographic) - YouTube

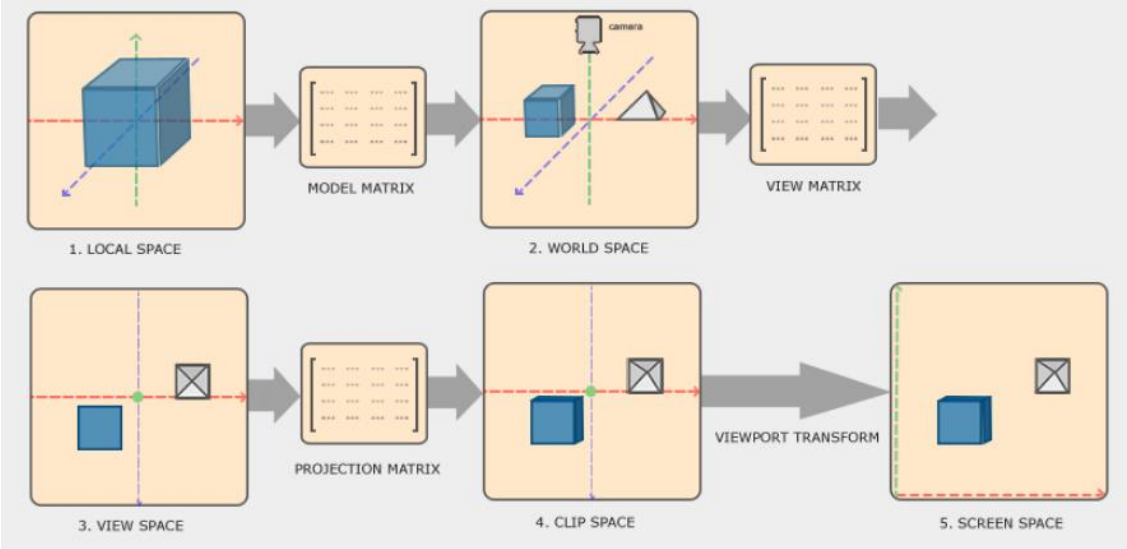
Orthographic 모드

Putting It All Together

- Local space 상의 vertex 좌표는 MVP 행렬이 곱해지면 clip space로 변환
- 이 값은 VS 상의 `gl_Position` 에 저장 가능

$$V_{clip} = M_{projection} \cdot M_{view} \cdot M_{model} \cdot V_{local}$$

- 이후 OpenGL은 perspective division 및 clipping 수행하여 NDC를 얻음
- 마지막으로, glViewport() 함수의 파라미터를 사용하여 뷰포트 변환 (viewport transform) 수행



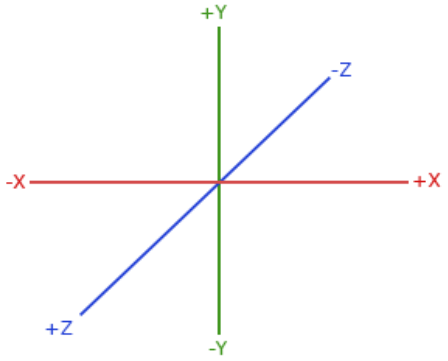
Going 3D

- 코드 상에서 3D 객체를 만드는 방법 소개
- 먼저 x축으로 -55도만큼 회전한 model 행렬을 생성 (약간 누운 듯하게 변환)

```
glm::mat4 model = glm::mat4(1.0f);
model = glm::rotate(model, glm::radians(-55.0f), glm::vec3(1.0f, 0.0f, 0.0f));
```

- 다음으로 view 행렬 생성
  - 카메라를 약간 뒤로 이동 (오른손 좌표계이므로 z값은 뒤로 갈수록 작아짐)

```
glm::mat4 view = glm::mat4(1.0f);
// note that we're translating the scene in the reverse direction of where we want to move
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
```



- 4:3 비율로 45도 화각을 가지는 원근 projection 행렬 생성

```
glm::mat4 projection;
projection = glm::perspective(glm::radians(45.0f), 800.0f / 600.0f, 0.1f, 100.0f);
```

Going 3D

- VS에서 uniform으로 세 행렬을 입력 받은 다음 vertex에 이를 곱하도록 함

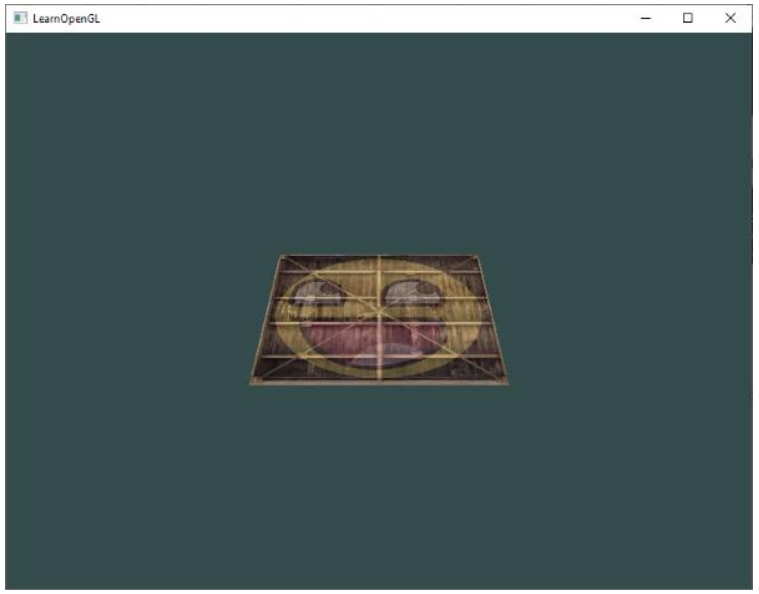
```
#version 330 core
layout (location = 0) in vec3 aPos;
...
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    // note that we read the multiplication from right to left
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    ...
}
```

- CPU쪽 코드에서 glm을 통해 계산한 model 행렬을 VS의 model 변수로 넘겨줌

```
int modelLoc = glGetUniformLocation(ourShader.ID, "model");
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
... // same for View Matrix and Projection Matrix
```

Going 3D

- 결과 화면



- 더 최적화하는 방법 (CPU 단에서 MVP를 미리 구함)

```
layout (location = 0) in vec3 position;
uniform mat4 mvp;

void main() {
    gl_Position = mvp * vec4(position, 1.0);
}
```

c++ - Setting up a MVP Matrix in OpenGL - Stack Overflow

More 3D

- 좀 더 3D답게 평면이 아닌 정육면체를 렌더링
- 총 36개(6개 면 * 2개 삼각형 * 3개 vertex)의 vertex 정의 필요
  - Code Viewer. Source code: getting-started/cube_vertices (learnopengl.com)
- 정육면체를 회전

```
model = glm::rotate(model, (float)glfwGetTime() * glm::radians(50.0f), glm::vec3(0.5f, 1.0f, 0.0f));
```

- 36개 vertex를 이용해 삼각형을 그리도록 명령

```
glDrawArrays(GL_TRIANGLES, 0, 36);
```

- 결과 화면

```
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
};
```

Z-buffer (Depth Buffer)

- OpenGL은 Z-buffer에 모든 깊이 정보 저장
  - 깊이 버퍼는 GLFW에서 자동으로 생성. GLFW 미사용시 FBO(framebuffer object)로 수동 생성해야 함
- Depth test
  - 현재 fragment의 깊이 값과 z-buffer의 깊이 값을 비교하여 현재 fragment의 유지 및 폐기 여부 결정
  - 기본적으로는 다른 fragment 뒤에 존재하는 fragment를 폐기하여 hidden surface 제거
  - Depth test를 활성화하기 위해서는 glEnable() 함수 사용 `glEnable(GL_DEPTH_TEST);`
  - 렌더 루프가 돌때마다 컬러 버퍼와 함께 깊이 버퍼도 비워줄 필요가 있음

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

- 결과 화면

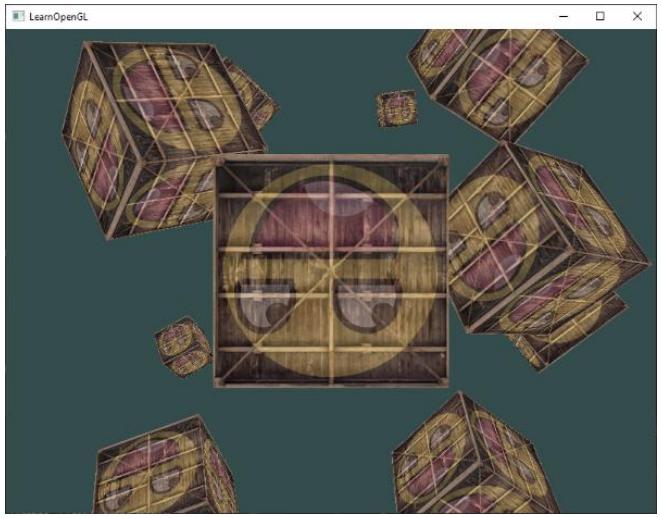
More Cubes!

- World space 상에서 10개 정육면체가 어디에 위치할지 지정하는 10개의 벡터 정의
- For문을 돌면서 각 정육면체를 렌더링
  - 각 정육면체별로 회전도 다르게 되도록 추가

```
glBindVertexArray(VAO);
for(unsigned int i = 0; i < 10; i++)
{
    glm::mat4 model = glm::mat4(1.0f);
    model = glm::translate(model, cubePositions[i]);
    float angle = 20.0f * i;
    model = glm::rotate(model, glm::radians(angle), glm::vec3(1.0f, 0.3f, 0.5f));
    ourShader.setMat4("model", model);

    glDrawArrays(GL_TRIANGLES, 0, 36);
}
```

```
glm::vec3 cubePositions[] = {
    glm::vec3( 0.0f,  0.0f,  0.0f),
    glm::vec3( 2.0f,  5.0f, -15.0f),
    glm::vec3(-1.5f, -2.2f, -2.5f),
    glm::vec3(-3.8f, -2.0f, -12.3f),
    glm::vec3( 2.4f, -0.4f, -3.5f),
    glm::vec3(-1.7f,  3.0f, -7.5f),
    glm::vec3( 1.3f, -2.0f, -2.5f),
    glm::vec3( 1.5f,  2.0f, -2.5f),
    glm::vec3( 1.5f,  0.2f, -1.5f),
    glm::vec3(-1.3f,  1.0f, -1.5f)
};
```

- 결과 화면



ⓒ nvidia

Camera

Camera/View space

- 카메라의 정의
  - 위치
  - 방향 vector
  - right vector
  - up vector

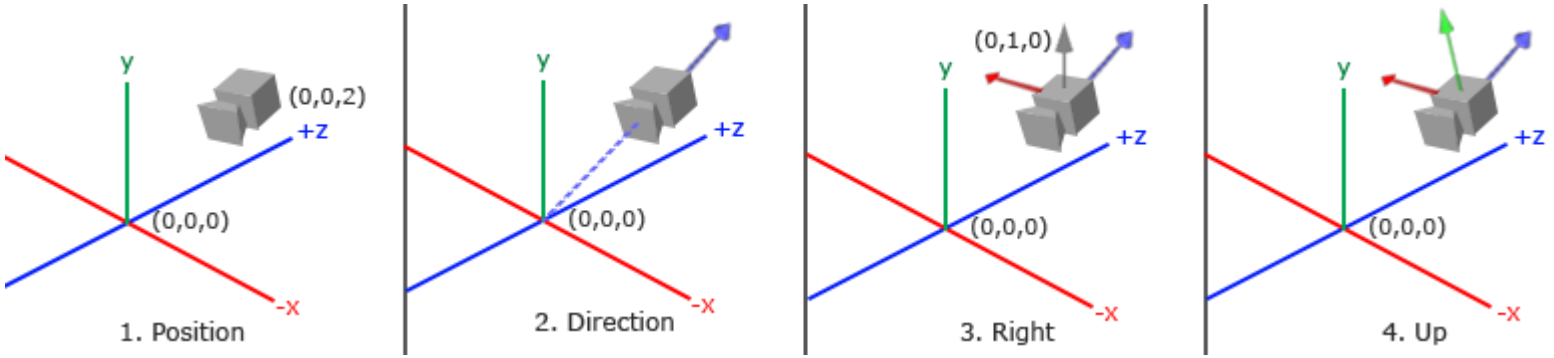
```
glm::vec3 cameraPos = glm::vec3(0.0f, 0.0f, 3.0f);
```

```
glm::vec3 cameraTarget = glm::vec3(0.0f, 0.0f, 0.0f);
glm::vec3 cameraDirection = glm::normalize(cameraPos - cameraTarget);
```

```
glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f);
glm::vec3 cameraRight = glm::normalize(glm::cross(up, cameraDirection));
```

```
glm::vec3 cameraUp = glm::cross(cameraDirection, cameraRight);
```

- 방향, right, up 벡터는 카메라를 중심으로 한 local 좌표계가 됨
  - right, up 벡터로 카메라의 회전 표현 가능



Look At

- LookAt 행렬

$$LookAt = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - OpenGL에서는 카메라와 반대 방향으로 world space의 모든 오브젝트들을 변환(W)하여 시점 조정 $W = T_W R_W$
  - 따라서 W를 역변환하면 viewing 변환 V를 얻을 수 있음 $V = W^{-1} = (T_W R_W)^{-1} = R_W{}^{-1} T_W{}^{-1}$

- 위 행렬을 편리하게 만들어 주는 lookAt() 함수
  - 1~3번째 파라미터는 각각
    앞 장의 cameraPos, cameraTarget, up

```
glm::mat4 view;
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));
```

- 이를 활용하여 카메라를 회전시키는 예제

```
const float radius = 10.0f;
float camX = sin(glfwGetTime()) * radius;
float camZ = cos(glfwGetTime()) * radius;
glm::mat4 view;
view = glm::lookAt(glm::vec3(camX, 0.0, camZ), glm::vec3(0.0, 0.0, 0.0), glm::vec3(0.0, 1.0, 0.0));
```

Walk Around

- WSAD 키 입력에 따라 카메라 이동해 보기
- 관련 변수 초기화

```
glm::vec3 cameraPos   = glm::vec3(0.0f, 0.0f,  3.0f);
glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);
glm::vec3 cameraUp    = glm::vec3(0.0f, 1.0f,  0.0f);
```

```
view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
```

- 지난 번에 정의한 ProcessInput() 함수 수정
  - 좌우 이동을 위해서는 cameraFront와 cameraUp의 외적 및 normalization을 통해 right 벡터 생성 (normalization을 하지 않으면 방향에 따라 이동속도가 달라짐)

```
void processInput(GLFWwindow *window)
{
    ...
    const float cameraSpeed = 0.05f; // adjust accordingly
    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
        cameraPos += cameraSpeed * cameraFront;
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        cameraPos -= cameraSpeed * cameraFront;
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;
}
```

Walk Around

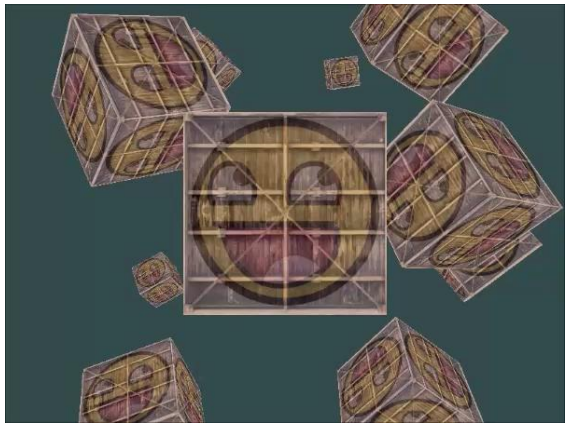
- 이동 속도 설정
  - Frame rate와 관계 없이 균일한 속도로 이동되도록, frame time(deltaTime)을 구해 이를 cameraSpeed에 반영

```
float deltaTime = 0.0f; // Time between current frame and last frame
float lastFrame = 0.0f; // Time of last frame
```

```
float currentFrame = glfwGetTime();
deltaTime = currentFrame - lastFrame;
lastFrame = currentFrame;
```

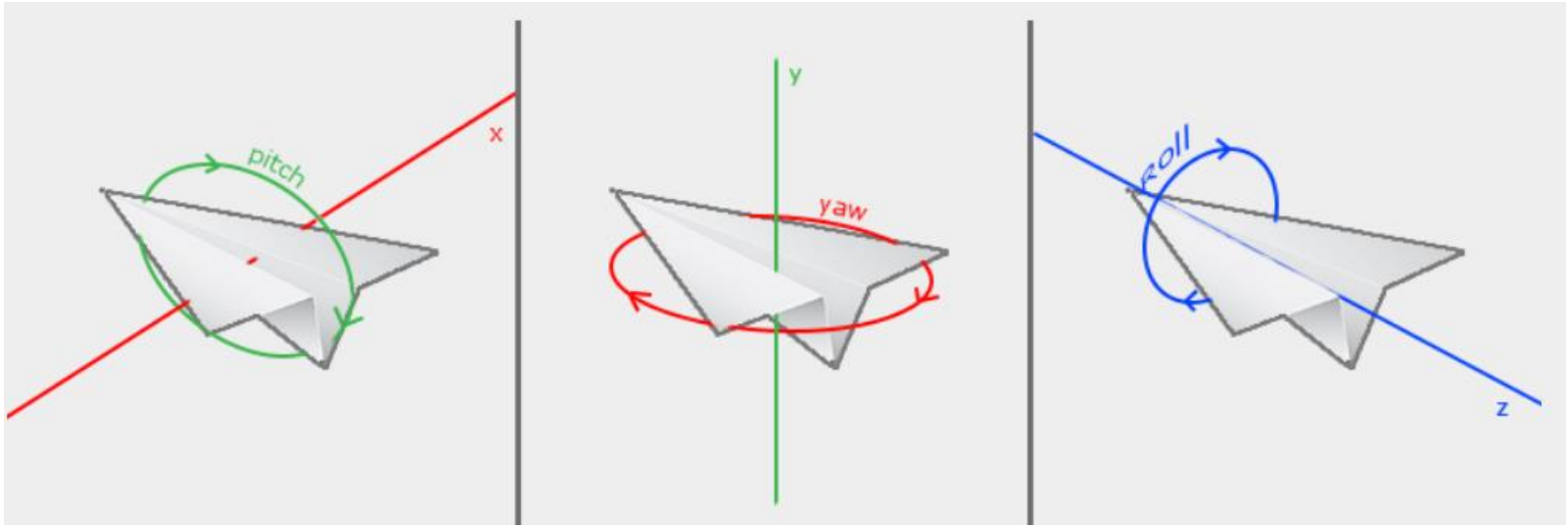
```
void processInput(GLFWwindow *window)
{
    float cameraSpeed = 2.5f * deltaTime;
    [...]
}
```

- 실행 화면



Look Around - Euler Angles

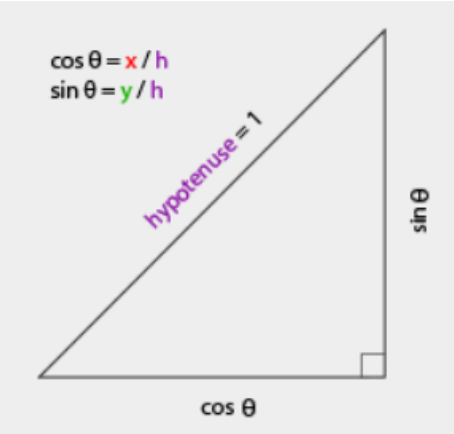
- 마우스 위치에 따라 회전을 하기 위해, 오일러 각 (Euler angles) 사용

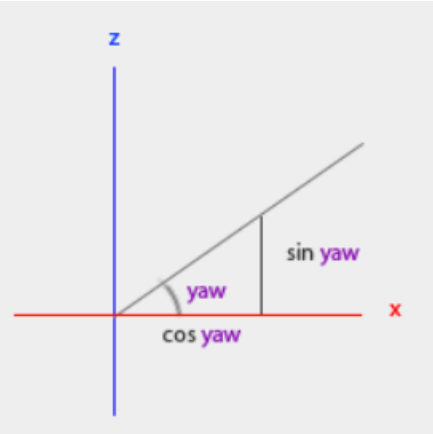


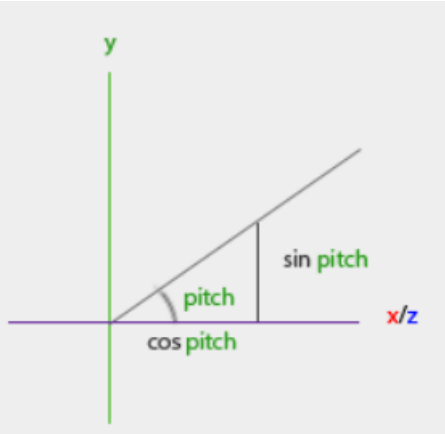
  - 👍👍👍DJI 미니2 시네마틱 촬영을 위한 짐벌(Yaw)설정하는 방법과 롤(Roll),피치(Pitch),요(Yaw) 확실히 구분하는 방법을 알려드립니다. – YouTube
  - 본 예제에서는 roll은 고정되었다고 가정

Look Around - Euler Angles

- 오일러 각에 따른 방향벡터 및 yaw 초기값 설정

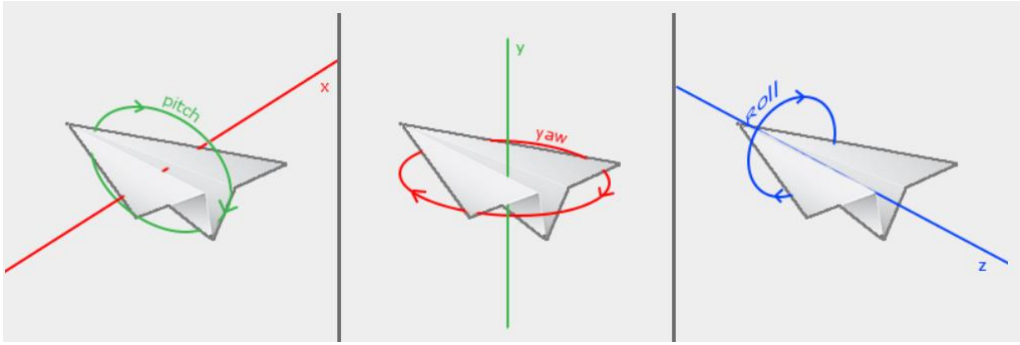







```
direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
direction.y = sin(glm::radians(pitch));
direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
```

```
yaw = -90.0f;
```



Look Around – Mouse Input

- 마우스의 수평 및 수직 움직임이 각각 yaw 및 pitch에 영향을 끼치도록 설정

- 먼저 glfw에서 마우스 커서를 숨기도록 설정

```
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
```

- 마우스의 이동 위치 및 스크롤 offset을 입력받는 콜백 함수들 선언

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos);
```

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset);
```

- 이 glfw에 이 콜백 함수들을 등록하면, 마우스를 움직이거나 휠을 굴릴 때마다 각 콜백이 호출

```
glfwSetCursorPosCallback(window, mouse_callback);
```

```
glfwSetScrollCallback(window, scroll_callback);
```

Look Around – Mouse Input

- 마우스 이동(시점 회전) 콜백 함수 구현
  1. 마우스가 움직인 offset 계산
     (단, 맨 처음 프레임에서는 이 offset을 0으로 설정)
  2. offset에 민감도(sensitivity)를 곱함
  3. 카메라의 yaw & pitch 값에 offset 값을 더함
  4. pitch의 min/max 설정
  5. 방향 벡터 계산

- 마우스 스크롤 (시점 zoom) 콜백 함수 구현
  - yoffset에 따라 fov 조정
  - 단, fov의 범위는 1도~45도

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
    fov -= (float)yoffset;
    if (fov < 1.0f)
        fov = 1.0f;
    if (fov > 45.0f)
        fov = 45.0f;
}
```

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    float xoffset = xpos - lastX;
    float yoffset = lastY - ypos;
    lastX = xpos;
    lastY = ypos;

    float sensitivity = 0.1f;
    xoffset *= sensitivity;
    yoffset *= sensitivity;

    yaw   += xoffset;
    pitch += yoffset;

    if(pitch > 89.0f)
        pitch = 89.0f;
    if(pitch < -89.0f)
        pitch = -89.0f;

    glm::vec3 direction;
    direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
    direction.y = sin(glm::radians(pitch));
    direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
    cameraFront = glm::normalize(direction);
}
```

Look Around – Result

- 실행 결과

Camera Class

- LearnOpenGL에서는 카메라 세부사항을 추상화하여 카메라 객체를 만들 수 있게 해 주는 클래스 제공
  - Code Viewer. Source code: includes/learnopengl/camera.h

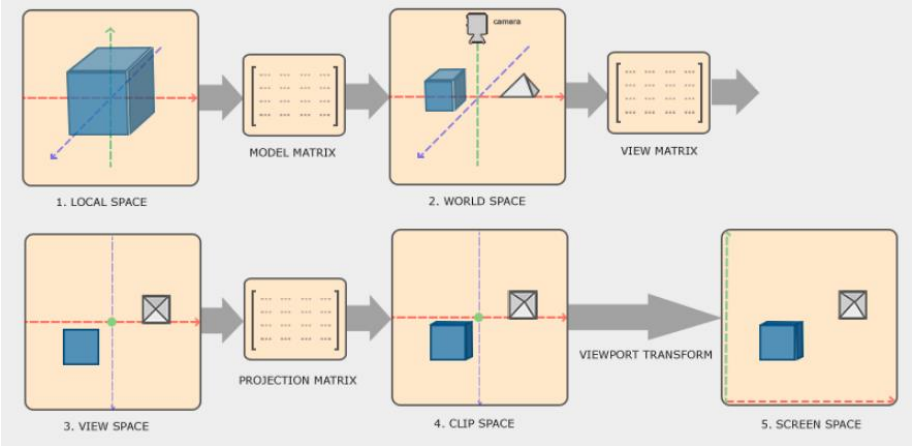
- 제공 함수
  - 생성자에서는 초기 position, up, yaw, pitch를 설정 (vector 또는 scalar 형식)
  - GetViewMatrix() – 계산된 view matrix 반환
  - ProcessKeyboard() – 키보드 입력에 따른 시점 변경
  - ProcessMouseMovement() – 마우스 이동에 따른 시점 변경
  - ProcessMouseScroll() – 마우스 휠 스크롤에 따른 시점 변경

ⓒ nvidia

마무리

마무리

- 이번 시간에는 아래와 같은 내용을 살펴보았습니다.
  - OpenGL에서 사용되는 좌표계
  - 투영 방법
  - 보이지 않는 면을 제거하기 위한 깊이 검사
  - 카메라 설정 방법 및 오일러 각
  - 키보드와 마우스를 이용한 시점 변환



- 실습 문제
  - 아래 기반 코드를 실행하여 시점 변환 확인
    LearnOpenGL/src/1.getting_started/7.4.camera_class at master · JoeyDeVries/LearnOpenGL (github.com)
    LearnOpenGL/camera.h at master · JoeyDeVries/LearnOpenGL (github.com)
  - 여러가지 파라미터를 조절하여 시점 변환에 미치는 영향 파악
  - 아래 코드를 조합하여 각 객체를 회전하도록 변경
    LearnOpenGL/src/1.getting_started/6.3.coordinate_systems_multiple at master · JoeyDeVries/LearnOpenGL (github.com)
  - 위 기반 코드가 포함된 VS 압축 파일은 실습 전 업로드 예정