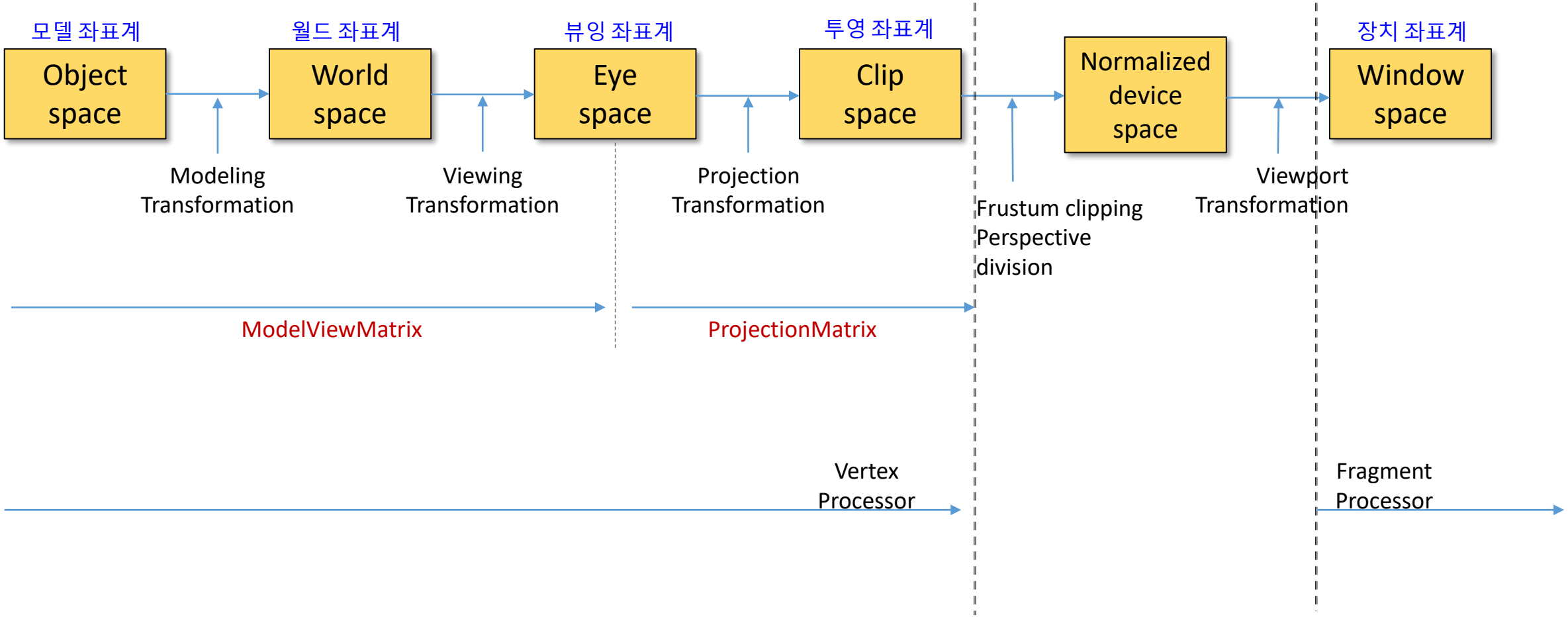
openGL 좌표계 변환 2

2025년 2학기

3차원 좌표계

- 좌표계 변환
  - 물체는 좌표계에 따라 새로운 좌표값으로 바뀌어 최종적으로 화면에 그려진다.



2) 뷰잉 변환

- 뷰잉 변환: 월드 좌표계를 유저의 시점인 view space (뷰 공간)로 변환
  - View space: 월드 좌표를 유저의 시점 앞에 있는 좌표로 변환한 결과
    - 즉, 카메라의 관점에서 바라보는 공간
    - 카메라 초기값: 좌표계의 원점에 위치

  - 카메라 위치 지정 방법
    1) 모델링 변환 함수들을 사용하여 카메라의 위치를 바꿀 수 있다
       - 카메라의 위치 변환은 객체들의 위치 변환과 반대로 변환하는 것과 같다.
         - 카메라를 오른쪽으로 이동 = 객체를 왼쪽으로 이동
       - 연속된 회전과 이동으로 카메라 위치를 지정할 수 있다.

    2) 카메라 설정 함수로 카메라 위치를 바꿀 수 있다.

2) 뷰잉 변환

- 뷰잉 변환 적용하기
  - 뷰잉 변환 행렬 M_{view} **행렬** 값을 설정하여 버텍스 세이더에 적용한다.
  - 변환 함수를 적용하여 카메라의 위치를 바꾼다.

```
//--- 응용 프로그램
void drawScene ()
{
	glUseProgram (shaderProgram);
	glm::mat4 view = glm::mat4(1.0f);
	view = glm::translate (view, glm::vec3(0.0f, 0.0f, -3.0f));
	unsigned int viewLocation = glGetUniformLocation (shaderProgram, "viewTransform");
	glUniformMatrix4fv (viewLocation, 1, GL_FALSE, &view[0][0]);

	glBindVertexArray (VAO);
	glDrawArrays (GL_TRIANGLES, 0, 3);
	glutSwapBuffers ();
}
```

//--- z축으로 -3만큼 이동 ➔ 카메라를 z축으로 3만큼 이동한 효과

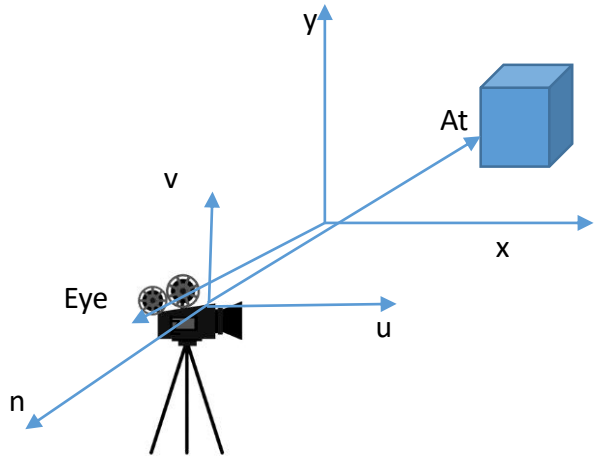
//--- 버텍스 세이더에서 viewTransform 변수위치

//--- viewTransform 변수에 변환값 적용하기

```
//--- 버텍스 세이더
#version 330 core
layout (location = 0) in vec3 vPos;

uniform mat4 modelTransform;
uniform mat4 viewTransform;

void main()
{
		gl_Position = viewTransform * modelTransform * vec4(vPos, 1.0);
}
```

2) 뷰잉 변환

- 카메라 공간
  - 다음의 세 가지 파라미터를 가진다.
    - Eye: 월드 공간에서의 카메라의 위치 → (t_x, t_y, t_z)
    - At: 월드 공간에서 카메라가 바라보는 기준점 → (n_x, n_y, n_z)
    - Up: 카메라의 상단이 가리키는 방향 → (v_x, v_y, v_z)
      - 대부분의 경우 Up은 월드 공간의 y축



  - 카메라 공간 설정을 위한 3차원 뷰잉 변환 행렬은

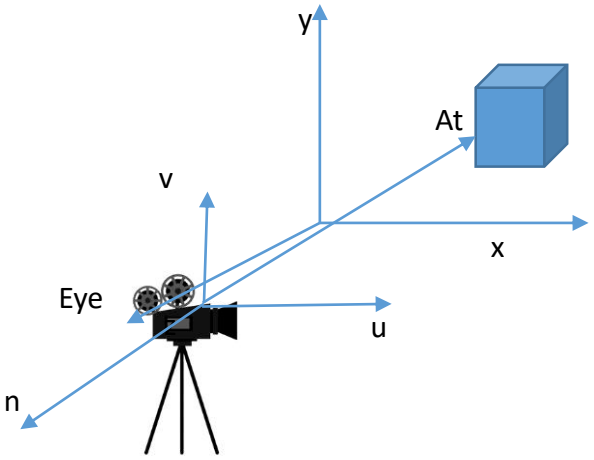
- $T = \begin{bmatrix} 1 & 0 & 0 & -tx \\ 0 & 1 & 0 & -ty \\ 0 & 0 & 1 & -tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- $M_{view} = R \cdot T = \begin{bmatrix} u_x & u_y & u_z & -t \cdot u \\ v_x & v_y & v_z & -t \cdot v \\ n_x & n_y & n_z & -t \cdot n \\ 0 & 0 & 0 & 1 \end{bmatrix}$

u: 오른쪽 벡터
v: 위쪽 벡터
n: 방향 벡터
t: 카메라의 위치 벡터

2) 뷰잉 변환

- 카메라 공간
  - 다음의 세 가지 파라미터를 가진다.
    - Eye: 월드 공간에서의 카메라의 위치
    - At: 월드 공간에서 카메라가 바라보는 기준점
    - Up: 카메라의 상단이 가리키는 방향,
      - 대부분의 경우 Up은 월드 공간의 y축



  - 단위 벡터, 외적 등을 이용하여 카메라의 파라미터를 설정한다.
    - 카메라 객체 좌표계 설정
      - 카메라 위치:
        - glm::vec3 **cameraPos** = glm::vec3 (0.0f, 0.0f, 5.0f); //--- (0.0, 0.0, 5.0)위치에 카메라를 놓음
      - 카메라 방향: 카메라가 바라보는 방향 (n)
        - glm::vec3 cameraTarget = glm::vec3 (0.0f, 0.0f, 0.0f); //--- 카메라가 바라보는 곳
        - glm::vec3 **cameraDirection** = glm::normalize (cameraPos-cameraTarget); //--- 카메라 방향 벡터: z축 양의 방향
      - 카메라의 오른쪽 축 (u)
        - 위쪽 벡터와 카메라 방향 벡터와의 외적
        - glm::vec3 up = glm::vec3 (0.0f, 1.0f, 0.0f);
        - glm::vec3 cameraRight = glm::normalize (glm::cross (up, cameraDirection));
      - 카메라의 위쪽 축 (v)
        - glm::vec3 **cameraUp** = glm::cross (cameraDirection, cameraRight); //--- 카메라의 위쪽 방향

2) 뷰잉 변환

- 카메라 설정 함수 프로토타입
  - glm::mat4 glm::**lookAt** (vec3 const &cameraPos, vec3 const &cameraDirection, vec3 const &cameraUp);
    - cameraPos: 카메라의 위치
    - cameraDirection: 카메라가 바라보는 기준점
    - cameraUp: 카메라의 상단이 가리키는 방향

  - 카메라 변환 위해서는
    - 위의 세 인자값 변경
    - 변환 함수를 적용하여 카메라 위치 및 방향 변경 가능

  - 사용 예)
    - 카메라를 (0, 0, 3)위치에 두고 z축의 음의 방향으로 카메라를 놓는다.
    - 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));

2) 뷰잉 변환

- 카메라의 위치 설정 예)

```
//--- 응용 프로그램
void drawScene ()
{
    glUseProgram (shaderProgram);
    glm::vec3 cameraPos   = glm::vec3(0.0f, 0.0f,  5.0f);          //--- 카메라 위치
    glm::vec3 cameraDirection = glm::vec3(0.0f, 0.0f, 0.0f);       //--- 카메라 바라보는 방향
    glm::vec3 cameraUp    = glm::vec3(0.0f, 1.0f,  0.0f);          //--- 카메라 위쪽 방향
    glm::mat4 view = glm::mat4 (1.0f);

    view = glm::lookAt (cameraPos, cameraDirection, cameraUp);
    unsigned int viewLocation = glGetUniformLocation (shaderProgram, "viewTransform");   //--- 뷰잉 변환 설정
    glUniformMatrix4fv (viewLocation, 1, GL_FALSE, &view[0][0]);

    glBindVertexArray (VAO);
    glDrawArrays (GL_TRIANGLES, 0, 3);                             //--- 도형 그리기
    glutSwapBuffers ();
}


//--- 버텍스 세이더
#version 330 core
layout (location = 0) in vec3 vPos;

uniform mat4 modelTransform;
uniform mat4 viewTransform;

void main()
{
        gl_Position = viewTransform * modelTransform * vec4(vPos, 1.0);
}
```

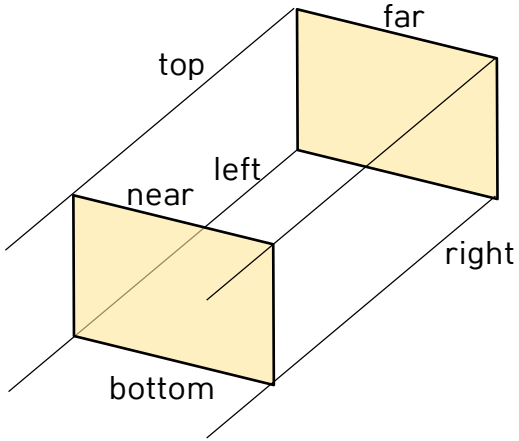
2) 뷰잉 변환

- 카메라 위치 바꾸기 예) z/Z 키를 누르면 카메라가 조금씩 뒤로/앞으로 이동

```
//--- 응용 프로그램
void drawScene ()
{
    glUseProgram (shaderProgram);
    glm::vec3 cameraPos   = glm::vec3(0.0f, 0.0f,  5.0f);          //--- 카메라 위치: 전역변수로 선언하여 사용할 수 있다.
    glm::vec3 cameraDirection = glm::vec3(0.0f, 0.0f, 0.0f);       //--- 카메라 바라보는 방향 : 전역변수로 선언하여 사용할 수 있다.
    glm::vec3 cameraUp    = glm::vec3(0.0f, 1.0f,  0.0f);          //--- 카메라 위쪽 방향 : 전역변수로 선언하여 사용할 수 있다.
    glm::mat4 view = glm::mat4 (1.0f);

    view = glm::lookAt (cameraPos, cameraDirection, cameraUp);
    unsigned int viewLocation = glGetUniformLocation (shaderProgram, "viewTransform");   //--- 뷰잉 변환 설정
    glUniformMatrix4fv (viewLocation, 1, GL_FALSE, &view[0][0]);

    glBindVertexArray (VAO);
    glDrawArrays (GL_TRIANGLES, 0, 3);                             //--- 도형 그리기
    glutSwapBuffers ();
}

void Keyboard(unsigned char key, int x, int y)
{
        if (key == 'z')                cameraPos.z += 0.1;         //--- 더하는 값은 변경될 수 있음
        else if ( key == 'Z')          cameraPos.z -= 0.1;
        …
        glutPostRedisplay ();
}
```

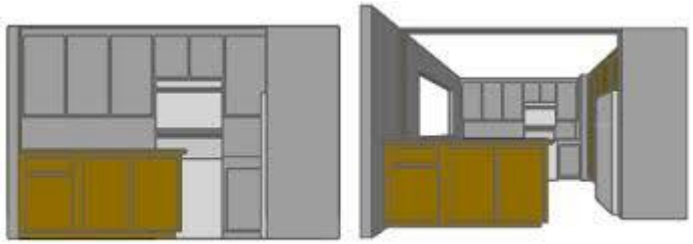
3) 투영 변환

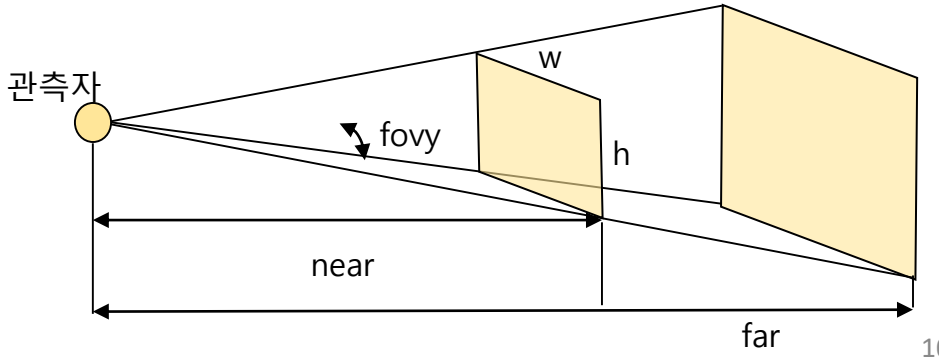
- 투영 변환: 객체가 놓이는 공간 설정
  - $M_{projection}$ 행렬: 투영 공간을 설정하는 변환 행렬
    - 절두체 (frustum)의 투영 공간: 직육면체 또는 피라미드 형태의 절두체
    - 공간 설정은 대개 바뀌지 않으므로 초기화 함수에서 한 번 적용하는 것이 좋음



  - 직각 투영
    - 투영 공간이 직육면체: orthographic projection 행렬이 적용된다.
    - 공간 외에 있는 객체들은 clip된다.
    - 2D 평면에 똑바로 매핑 되고 원근감이 없다.

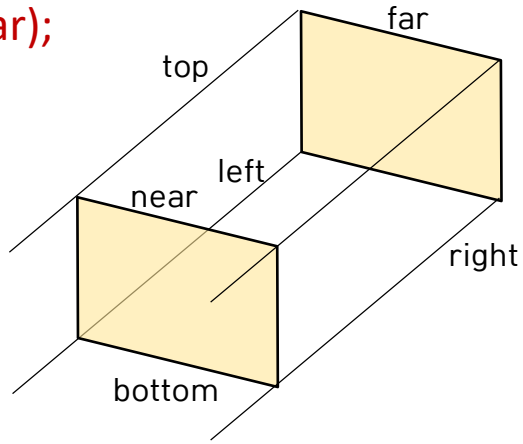
  - 원근 투영
    - 투영 공간이 피라미드 형태: perspective projection 행렬이 적용된다.
    - 원근감을 가진다.





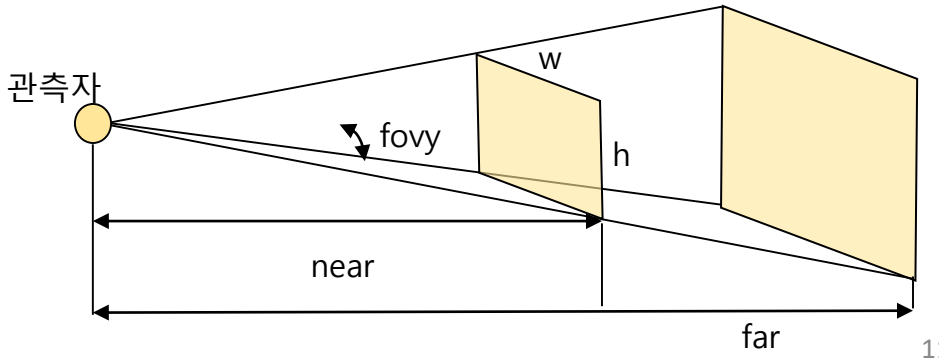
3) 투영 변환

- 직각 투영 볼륨 함수 프로토타입
  - glm::mat4 glm::ortho (float left, float right, float bottom, float top, float near, float far);
    - left, right: 절두체의 왼쪽, 오른쪽 좌표
    - bottom, top: 절두체의 맨 밑과 맨 위의 좌표
    - near, far: 가까운 평면과 먼 평면



- 원근 투영 볼륨 함수 프로토타입
  - glm:: mat4 glm::perspective (float fovy, float aspect, float near, float far);
    - fovy: 뷰잉 각도(라디언), 뷰잉 공간이 얼마나 큰지를 설정
    - aspect: 종횡비 (앞쪽의 클리핑 평면의 폭(w)을 높이(h)로 나눈 값)
      - 종횡비: 화면의 가로방향에 대한 단위 길이를 나타내는 픽셀수에 대한 세로방향의 단위길이를 나타내는 픽셀 수의 비율.

      예) 종횡비가 0.5 → 가로:세로 = 0.5:1 → 가로 길이가 세로 길이의 절반 가로 길이의 두 픽셀이 세로 길이의 한 픽셀에 대응한다.
    - near: 관측자에서부터 가까운 클리핑 평면까지의 거리 (항상 양의 값)
    - far: 관측자에서 먼 클리핑 평면까지의 거리 (항상 양의 값)



3) 투영 변환

- 직각 투영 변환 예)

```
//--- 응용 프로그램
void drawScene ()
{
	glUseProgram (shaderProgram);
	glm::mat4 projection = glm::mat4(1.0f);
	projection = glm::ortho (-100.0f, 100.0f, -100.0f, 100.0f, -100.0f, 100.0f);		//--- 투영 공간 설정: [-100.0, 100.0]

	unsigned int projectionLocation = glGetUniformLocation (shaderProgram, "projectionTransform");		//--- 투영 변환 값 설정
	glUniformMatrix4fv (projectionLocation, 1, GL_FALSE, &projection[0][0]);

	glBindVertexArray (VAO);
	glDrawArrays (GL_TRIANGLES, 0, 3);		//--- 도형 그리기
	glutSwapBuffers ();
}

//--- 버텍스 세이더
#version 330 core
layout (location = 0) in vec3 vPos;

uniform mat4 modelTransform;
uniform mat4 viewTransform;
uniform mat4 projectionTransform;

void main()
{
	gl_Position = projectionTransform * viewTransform * modelTransform * vec4(vPos, 1.0);
}
```

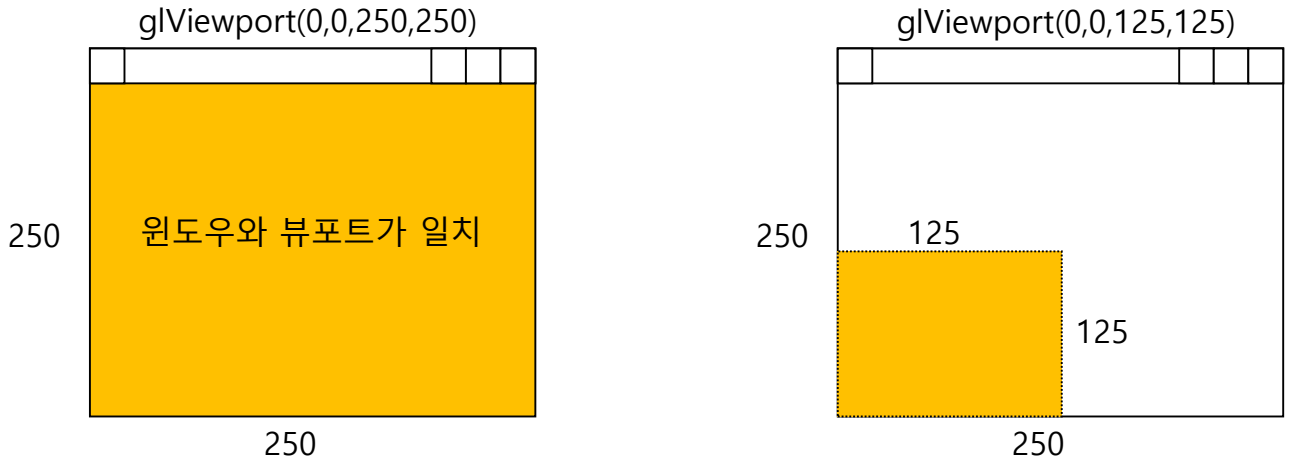
3) 투영 변환

- 원근 투영 변환 예)

```
//--- 응용 프로그램
void drawScene ()
{
	glUseProgram (shaderProgram);
	glm::mat4 projection = glm::mat4(1.0f);
	projection = glm::perspective (glm::radians(45.0f), 1.0f, 0.1f, 100.0f);	//--- 투영 공간 설정: fovy, aspect, near, far
	projection = glm::translate (projection, glm::vec3 (0.0, 0.0, -2.0);	//--- 공간을 z축 이동

	unsigned int projectionLocation = glGetUniformLocation (shaderProgram, "projectionTransform");	//--- 투영 변환 값 설정
	glUniformMatrix4fv (projectionLocation, 1, GL_FALSE, &projection[0][0]);

	glBindVertexArray (VAO);
	glDrawArrays (GL_TRIANGLES, 0, 3);	//--- 도형 그리기
	glutSwapBuffers ();
}

//--- 버텍스 세이더
#version 330 core
layout (location = 0) in vec3 vPos;

uniform mat4 modelTransform;
uniform mat4 viewTransform;
uniform mat4 projectionTransform;

void main()
{
	gl_Position = projectionTransform * viewTransform * modelTransform * vec4(vPos, 1.0);
}
```

4) 뷰포트 변환

- 화면의 최종적으로 출력될 영역 설정
- void **glViewport** (GLint x, GLint y, GLsizei width, GLsizei height);
  - 윈도우의 영역을 설정한다.
    - x, y: 뷰포트 사각형의 왼쪽 아래 좌표
    - width, height: 뷰포트의 너비와 높이



- Reshape 함수가 있을 때, 대개 그 함수에 포함시킨다.
  - 한 개 이상의 뷰포트를 설정해야 할 때는 그리기 함수에서 그리기 전에 뷰포트를 설정한다.

변환 적용하기

- 버텍스 세이더에 변환 행렬을 적용
  - 응용 프로그램에서 변환 행렬 설정
    - $\mathbf{M}_{\mathbf{model}}$ 행렬

```
glm::mat4 model;
model = glm::rotate(model, glm::radians(30.0f), glm::vec3(1.0f, 0.0f, 0.0f));          //--- x축에 대하여 30도 회전
```

    - $\mathbf{M}_{\mathbf{view}}$ 행렬

```
glm::mat4 view;
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));          //--- 카메라를 z축으로 3만큼 이동, 카메라와 객체 변환은 반대
```

    - $\mathbf{M}_{\mathbf{projection}}$ 행렬

```
glm::mat4 projection;
projection = glm::perspective(glm::radians(45.0f), screenWidth / screenHeight, 0.1f, 100.0f);    //--- 원근 투영 적용
```

변환 적용하기

- 버텍스 세이더에서 변환 행렬 적용

```
#version 330 core
layout (location = 0) in vec3 vPos;                 //--- 객체의 원래 버텍스 좌표값
layout (location = 1) in vec3 vColor;               //--- 객체의 색상값

uniform mat4 model;                                 //--- 모델링 변환값: 응용 프로그램에서 전달 → uniform 변수로 선언: 변수 이름“model”로 받아옴
uniform mat4 view;                                  //--- 뷰잉 변환값: 응용 프로그램에서 전달 → uniform 변수로 선언: 변수 이름“view”로 받아옴
uniform mat4 projection;                            //--- 투영 변환값: 응용 프로그램에서 전달 → uniform 변수로 선언: 변수 이름“projection”로 받아옴

out vec3 passColor;

void main()
{
    gl_Position = projection * view * model * vec4(vPos, 1.0);      //--- 변환은 ← 방향으로 적용
                                                                    //--- vPos: 객체의 원래 좌표값
    passColor = vColor;
}
```

변환 적용하기

- 응용 프로그램의 그리기 함수에서 버텍스 세이더에 uniform으로 선언되어 있는 변환 행렬 값을 전달

```
void drawScene ()
{
	glUseProgram (shaderProgram);
	glBindVertexArray (VAO);

	int modelLoc = glGetUniformLocation (shaderProgram,  "model");		//--- 버텍스 세이더에서 모델링 변환 행렬 변수값을 받아온다.
	int viewLoc = glGetUniformLocation (shaderProgram, "view");		//--- 버텍스 세이더에서 뷰잉 변환 행렬 변수값을 받아온다.
	int projLoc = glGetUniformLocation (shaderProgram,  "projection");	//--- 버텍스 세이더에서 투영 변환 행렬 변수값을 받아온다.

	//--- 모델링 변환, 뷰잉 변환, 투영 변환 행렬을 설정한 후, 버텍스 세이더에 저장한다.
	glm::mat4 mTransform = glm::mat4 (1.0f);
	transform = glm::rotate (mTransform, glm::radians(yAngle), glm::vec3(0.0f, 1.0, 0.0f));
	glUniformMatrix4fv (modelLoc, 1, GL_FALSE, & mTransform[0[0]);

	glm::mat4 vTransform = glm::mat4 (1.0f);
	 vTransform = glm::lookAt (cameraPos, cameraDirection, cameraUp);
	glUniformMatrix4fv (viewLoc, 1, GL_FALSE, & vTransform[0][0]);

	glm::mat4 pTransform = glm::mat4 (1.0f);
	 pTransform = glm::perspective (glm::radians(60.0f), (float)window_w / (float)window_h, 0.1f, 200.0f);
	glUniformMatrix4fv (projLoc, 1, GL_FALSE, & pTransform[0][0]);

	//--- 객체를 그린다.
	glBindVertexArray (VAO);
	glDrawArrays (GL_TRIANGLES, 0, 12);
}
```

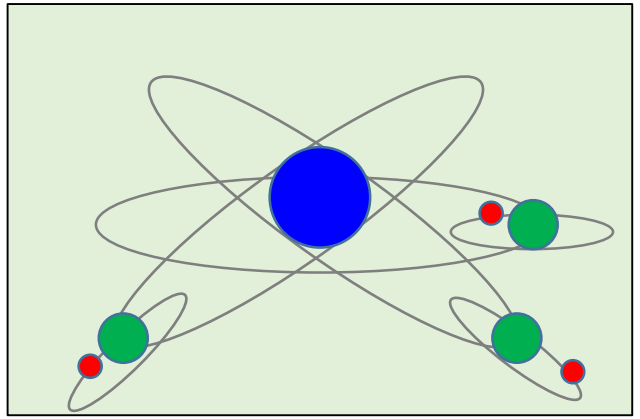
변환 적용하기

- 뷰포트 변환 적용

```
void Reshape (int w, int h)
{
	//  w : window width   h : window height
	g_window_w = w;
	g_window_h = h;
	glViewport (0, 0, w, h);	//--- 전체 윈도우를 사용해서 출력
}
```

실습 19

- 중심의 구를 중심으로 3개의 구가 다른 방향의 경로를 따라 회전하는 애니메이션 만들기
  - 은면제거, 투영을 적용한다.
    - 모든 구는 다른 색으로 설정한다.
  - 세개의 행성이 중심의 구를 중심으로 다른 속도, 다른 경로로 공전한다.
    - 경로 1: xz 평면
    - 경로 2: xz 평면이 반시계방향으로 45도 기울어져 있다.
    - 경로 3: xz 평면이 시계방향으로 45도 기울어져 있다.
  - 3개의 행성에는 각각 공전하는 달을 가지고 있다.
    - 달은 부모 행성 공전 궤도와 같은 기울기의 궤도로 행성 주위를 공전한다.
  - 모든 공전 경로를 화면에 그린다.



  - 키보드 명령:
    - p/P: 직각 투영/원근 투영
    - m/M: 솔리드 모델/와이어 모델
    - w/a/s/d: 위의 도형들을 좌/우/상/하로 이동 (x축과 y축 값 이동 – 객체 이동)
    - +/-: 위의 도형들을 앞/뒤로 이동 (z축 값 이동 – 객체 이동)
    - y/Y: 모든 궤도들의 반지름이 커진다/작아진다
    - z/Z: 중심의 구를 제외하고 행성, 달, 궤도가 z축에 대하여 양/음 방향으로 일제히 회전
    - q: 프로그램 종료