# 기초컴퓨터그래픽스 HW#4

학번 : 20141547
이름 : 오새암

## <문제>

1.
그림 1은 중앙의 소(A)를 기본으로 하여 회전변환 및 이동변환하는 소의 모습을 구현한 이미지이다. 두 개의 움직임을 구현하였고, 각각 (B), (C) 경로를 따라 이동하고 각각 움직임에 대해 회전 각도는 일정하다, (다음 위치로 갈 때 회전하는 각도는 다 (B), (C)는 각각 일정하다.) (B)의 소는 y축에 대하여 360° 를 돌았고, (C)의 소는 z축에 대하여 180° 를 돌았다. (B)의 소는 (-3, 0, 2)에서 (-3, 0, 2)로 이동했고, (C)의 소는 (-3, 0, 2)에서 (3, 0, 2)로 이동했다. 다음 질문에 답하여라.((A)의 소는 원점에 있고, (A)를 중심으로 한 축들에 대해 빨간색은 x축, 초록색은 y축, 파랑색은 z축이다.)

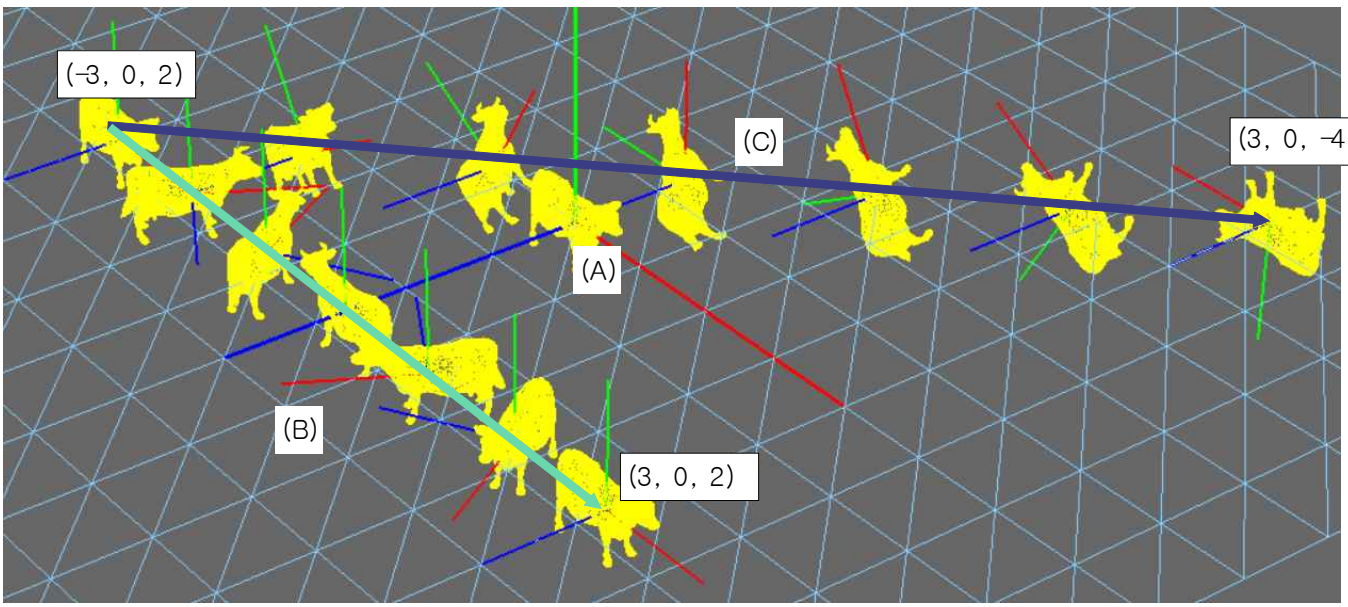


그림 1

(a) (B)의 경로로 이동하는 소를 구현한 코드이다. 빈 칸에 알맞은 값을 넣으시오.

```
for (int i = 0; i <= (가); i += (나)) {
                    float angle = (float)i;
                    ModelViewProjectionMatrix = glm::translate(ViewProjectionMatrix, glm::vec3((다), 0.0f, (라));
                    ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, angle * TO_RADIAN,
                            glm::vec3((마), (바), (사)));
                    glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE,
                            &ModelViewProjectionMatrix[0][0]);
                    glLineWidth(2.0f);
                    draw_axes();
                    glLineWidth(1.0f);
                    draw_object(OBJECT_COW, 255 / 255.0f, 255 / 255.0f, 0 / 255.0f); //yellow
```

```
}
```

(b) (B)의 경로로 이동하는 소를 구현한 코드이다. 빈 칸에 알맞은 값을 넣으시오.

```
for (int i = 0; i <= (가); i += (나)) {
        float angle = (float)i;
        ModelViewProjectionMatrix = glm::translate(ViewProjectionMatrix, glm::vec3((다), 0.0f, (라)));
        ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, angle * TO_RADIAN,
                glm::vec3((마), (바), (사)));
        glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE,
                &ModelViewProjectionMatrix[0][0]);
        glLineWidth(2.0f);
        draw_axes();
        glLineWidth(1.0f);
        draw_object(OBJECT_COW, 255 / 255.0f, 255 / 255.0f, 0 / 255.0f); //yellow
}
```

(c) ((a)에서 구한 값에 대하여)  대하여) 다음 함수 호출이 리턴해주는 4행 4열 행렬을 정확히 기술하라.

---

glm::rotate(glm::mat4(1.0f), angle * TO RADIAN, glm::vec3((마), (바), (사)))

---

2.

다음 [그림 2]는 소가 x = sin(-z), y = 0를 따라서 이동하고 y축을 중심으로 z만큼 회전하는 변환을 구현한 이미지이다. [그림 3]은 z을 중심으로 길이 3의 반경으로 일정한 속도로 움직이는 원운동을 하는 이동변환을 하고, 회전은 이동으로 한바퀴를 돌면 역시 소도 y축을 중심으로 한바퀴를 돈다. 다음 질문에 답하시오.

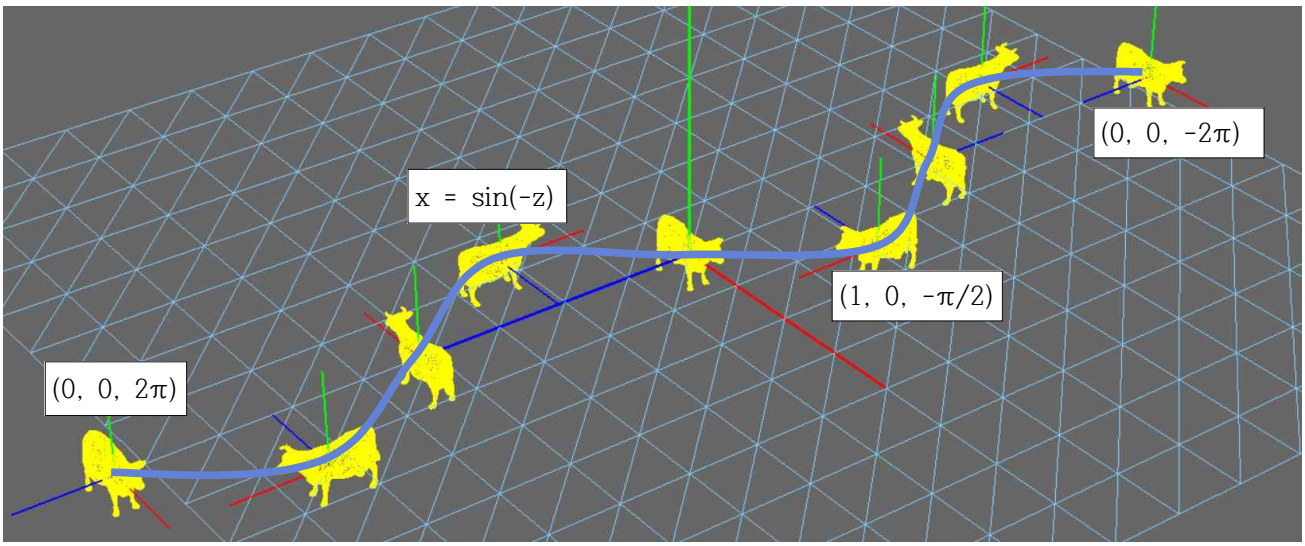


그림 2

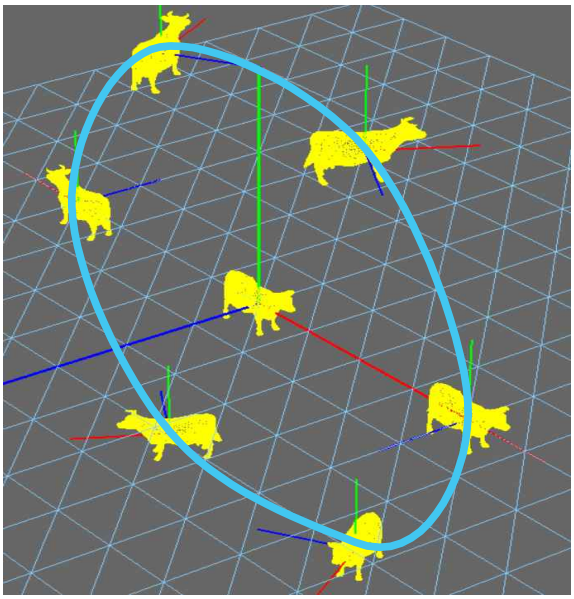

그림 3

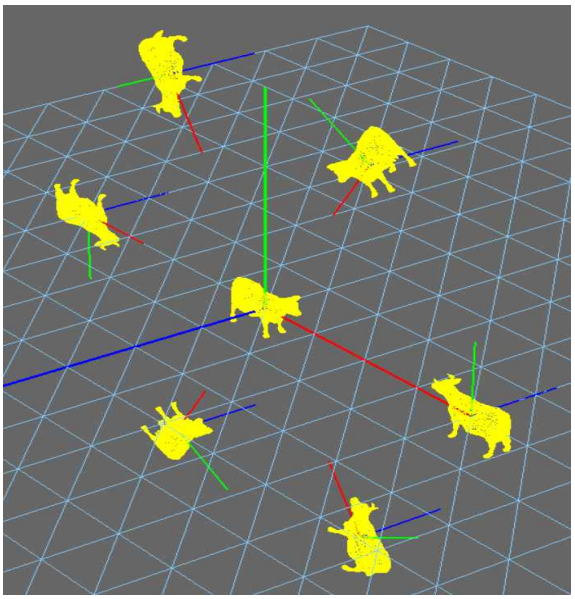

그림 4

(a) [그림 2]의 경로로 이동하는 소를 구현한 코드이다. 빈 칸에 알맞은 값을 넣으시오. ((0, 0, -2π)에서 시작 한다고 가정한다. 그리고 종료는 (0, 0 , 2π)에서 이루어진다. z좌표 변화량은 일정하다.) 또한 sin(x)는 x의 sin값을 return하는 것을 활용하자.)

```
for (int i = (가); i <= (나); i += (다)) {
                float angle = (float)i;
                ModelViewProjectionMatrix = glm::translate(ViewProjectionMatrix, glm::vec3((라), 0.0f, (마));
```

```
                ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, angle * TO_RADIAN,
                        glm::vec3((바), (사), (아)));
                glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE,
                        &ModelViewProjectionMatrix[0][0]);
                glLineWidth(2.0f);
                draw_axes();
                glLineWidth(1.0f);
                draw_object(OBJECT_COW, 255 / 255.0f, 255 / 255.0f, 0 / 255.0f); //yellow
        }
```

(b) [그림 3]의 경로로 이동하는 소를 구현한 코드이다. 빈 칸에 알맞은 값을 넣으시오.(원 중심의 소를 그리는 코드는 생략함.)

```
for (int i = 0; i < (가); i += (나)) {
                float angle = (float)i;
                ModelViewProjectionMatrix = glm::translate(ViewProjectionMatrix,
                        glm::vec3(3 * cos(angle * TO_RADIAN), (다), 0.0f));
                ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, (라),
                glm::vec3((마), (바), (사)); ---(1)
                glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE,
                        &ModelViewProjectionMatrix[0][0]);
                glLineWidth(2.0f);
                draw_axes();
                glLineWidth(1.0f);
                draw_object(OBJECT_COW, 255 / 255.0f, 255 / 255.0f, 0 / 255.0f); //yellow
        }
```

(c) [그림 4]는 (b)의 코드를 일부 수정하여서 회전하는 소들이 모두 중심을 향해 바라보게 만들었다. 궤도는 (b)와 일치하다. (1) 기준벡터를 일부 수정하고, 추가로 rotate하는 함수를 한 줄 추가하여 [그림 4]처럼 나오게 추가 부분을 작성하시오. ((1) 수정한 코드 + 추가 코드 2줄 작성하면 됩니다.)

## <답>

1.

(a)

(가) : 360

(나) : 60

(다) : -3.0f + (angle / 60.0f)

(라) : 2.0f

(마) : 0.0f

(바) : 1.0f

(사) : 0.0f

(b)

(가) : 180

(나) : 30

(다) : -3.0f + (angle / 30.0f)
(라) : 2.0f - (angle / 30.0f)
(마) : 0.0f
(바) : 0.0f
(사) : 1.0f

(c)
문제에서 제시한 함수를 호출 시 소는 세상 좌표계가 아닌 모델 좌표계로 다시 이동하게 된다. 모델링 한 소를 중심으로  y축을 중심으로 angle만큼 회전 후 본 소의 모습이 보이게 된다.(화면 중앙에 있게 된다.)

$$\begin{bmatrix}\cos(a) & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}-\sin(a) & 0\\ 0 & 0\end{bmatrix}$$
$$\begin{bmatrix}\sin(a) & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}\cos(a) & 0\\ 0 & 1\end{bmatrix}$$
다음과 같은 (a가 angle이다.) 4행 4열의 행렬이 나오고, 이를 직접 출력해 보면 z축에서 바라본 소의 모습들을 볼 수 있다.

2.
(a)
(가) : -360
(나) : 360
(다) : 90
(라) : sin(-angle * TO_RADIAN)
(마) : angle * TO_RADIAN
(바) : 0.0f
(사) : 1.0f
(아) : 0.0f

(b)
(가) : 360
(나) : 60
(다) : 3 * sin(angle * TO_RADIAN)
(라) : angle * TO_RADIAN
(마) : 0.0f
(바) : 1.0f
(사) : 0.0f

(c)
(1)번 수정코드 :
```
ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, angle * TO_RADIAN, glm::vec3(0.0f, 0.0f, 1.0f));
```
추가 코드 :
```
ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, 180 * TO_RADIAN, glm::vec3(0.0f, 1.0f, 0.0f));
```

# <풀이>

1.

(a)

코드의 흐름을 보면, angle(라디안)만큼 회전하는 코드가 적용되는 것을 볼 수 있다. (B)는 360도 회전을 하므로 (가)는 360이고 처음 시작을 제외하고 6번의 변환이 있음을 그림으로 확인할 수 있다. 따라서 360/6 = 60도만큼 회전을 하는 것을 알 수 있다. 따라서 (나)는 60임을 알 수 있다. 이동변환 관점만 보면 y = 0, z = 2의 직선 위를 이동하는 것을 알 수 있다. (-3. 0, 2)에서 (3, 0, 2)까지의 거리는 6이고, x축의 양의 방향으로 움직인 것으로 알 수 있다. 따라서 원점에서 소는 angle도 만큼 회전하고, angle만큼 회전하면 x축으로는 angle/60만큼 이동하는 것을 알 수 있다. z축으로는 항상 2만큼 이동하므로, (다)는 -3 + (angle/60)이고 (라)는 2이다. 마지막으로, 소는 y축을 중심으로 회전하였으므로, y축 방향의 단위벡터는 (0, 1, 0)이므로, (마), (바), (사)는 알맞게 값을 넣으면 0.0f, 1.0f, 0,0f이다.

(b)

코드의 흐름을 보면, angle(라디안)만큼 회전하는 코드가 적용되는 것을 볼 수 있다. (C)는 180도 회전을 하므로 (가)는 180이고 처음 시작을 제외하고 6번의 변환이 있음을 그림으로 확인할 수 있다. 따라서 180/6 = 30도만큼 회전을 하는 것을 알 수 있다. 따라서 (나)는 30임을 알 수 있다. 이동변환 관점만 보면 y = 0, x + z = -1의 직선 위를 이동하는 것을 알 수 있다. (-3. 0, 2)에서 (3, 0, -4)까지의 거리는 $6\sqrt{2}$ 이고, 한 번 변환할 때마다 x축과 y축으로 1씩 이동하는 것을 알 수 있다. 따라서 원점에서 소는 angle도 만큼 회전하고, angle만큼 회전하면 x축으로는 angle/30, y축으로 angle/30만큼 이동하는 것을 알 수 있다, (다)는 -3 + (angle/30)이고 (라)는 2 + (angle/30)이다. 마지막으로, 소는 z축을 중심으로 회전하였으므로, z축 방향의 단위벡터는 (0, 0, 1)이므로, (마), (바), (사)는 알맞게 값을 넣으면 0.0f, 0.0f, 1,0f이다.

(c)

<답>부분이 모범 답안임.

2.

(a)

코드의 흐름을 보면, angle(라디안)만큼 회전하는 코드가 적용되는 것을 볼 수 있다. 중간에 TO_RADIAN이 있는 것으로 보아 i는 각도법을 사용한 즉, °(도)로 것임을 알 수 있다. translate는 문제 설명에 이동 변환은 x = sin(-z), y = 0을 따라 움직인다고 하였다. 시작은 (0, 0, -360°)에서 시작한다고 하였고, 종료는 (0, 0, 360°)이므로 (가)는 -360이고, (나)는 360임을 알 수 있다. 시작점을 제외하고 소는 총 8마리 있으므로 (360-(-360))/8 = 90으로 연속된 소와 소의 z좌표는 90°도 차이나고 따라서 (다)는 90임을 알 수 있다. 이동변환 관점에서 x = sin(-z), y = 0을 따라 이동한다고 했으므로 x좌표는 sin(-z)이다. 따라서 z는 angle * TO_RADIAN임을 코드에서 알 수 있다. 밑의 줄의 rotate를 보면 angle * TO_RADIAN만큼 회전하였는데, 문제에서 회전에서 y축에 대해 z만큼 회전하였다고 기술했는데 여기서 angle * TO_RADIAN이 z임을 알 수 있다. 따라서 (라)는 sin(- angle * TO_RADIAN), (마)는 angle * TO_RADIAN임을 알 수 있다. 마지막으로, 소는 y축을 중심으로 회전하였으므로, y축 방향의 단위벡터는 (0, 1, 0)이므로, (바), (사), (아)는 알맞게 값을 넣으면 0.0f, 1.0f, 0,0f이다.

(b)

코드의 흐름을 보면, angle(라디안)만큼 회전하는 코드가 적용되는 것을 볼 수 있다. [그림 3]는 이동 경로는 360도 회전을 하고, 6번의 변환이 있음을 그림으로 확인할 수 있다. 시작은 0°임을 코드에서 확인할 수 있고, 한 번의 이동변환마다 360/60 = 60°만큼 중심을 기준으로 이동하고, 또한 회전 변환도 소가 원의 한 바퀴를 따라 이동할 때 역시 y축에 대해 한바퀴 360°를 돈다고 하였으므로 회전 역시 소와 소 사이는 60°만

큼 회전하는 것을 알 수 있다. (가)는 360, (나)는 60임을 알 수 있다. z축을 중심으로 반경 3인 원을 그리고 있었다고 문제에서 기술했는데, 코드를 보면 translate의 x는 3 * cos(angle * TO_RADIAN)으로 둔 것에서 힌트를 얻을 수 있다. 따라서 (다)는 3 * sin(angle * TO_RADIAN)으로 두면 위의 [그림 3]을 만족할 수 있다, (라)는 소의 회전각도 원의 궤도 이동각도와 같으므로 angle * TO_RADIAN이다. 마지막으로, 소는 y축을 중심으로 회전하였으므로, y축 방향의 단위벡터는 (0, 1, 0)이므로, (마), (바), (사)는 알맞게 값을 넣으면 0.0f, 0.0f, 1,0f이다.

(c)
(b)의 코드를 일부 변경하여서 [그림 4]와 같은 결과를 만드는 것이 목표인데, (1)부분은 rotate 기준 벡터 부분(3번째 parameter)만 변경하라고 하였으므로, 회전 각도는 똑같은 것을 알 수 있다. angle * TO_RADIAN만큼 회전하는데 그림을 보면 z축을 중심으로 angle * TO_RADIAN만큼 회전하면 아직은 완벽하지 않지만, 한 번의 회전 변환을 하면 [그림 4]와 같은 비슷한 결과가 나온다. 따라서 (1)번 부분은 기준 벡터를 glm::vec3(0.0f, 0.0f, 1.0f)로 바꾸면 된다. 추가로는 y축을 중심으로 반 바퀴를 돌리면 모든 소들이 중심을 볼 수 있기 때문에

```
ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, 180 * TO_RADIAN, glm::vec3(0.0f, 1.0f, 0.0f));
```

를 추가하면 [그림 4]같은 코드를 얻을 수 있다.