기초 컴퓨터 그래픽스 HW4

OpenGL API 함수를 사용한 3D 뷰잉 연습

20170123 정윤진

2020.05.24

[문제] 다음은 3차원 모델링 변환에 관한 문제이다.

(1) 아래 그림 A에서와 같이 세상 좌표계의 원점 주변에 도시한 소 모델을 적절한 모델링 변환을 통하여 세상에 배치해주는 프로그램의 일부가 주어져 있다. 이 코드가 올바르게 작동하기 위하여 빈칸에 들어가야 할 값을 기술하라. 그림 A에는 (0, 2, 0)을 중심으로 반지름이 2인 원을 따라 x – y 평면상에서 30도씩 회전하고 있는 검정색 소가 그려져 있다. x값이 가장 큰 소가 제일 먼저 그려진다.

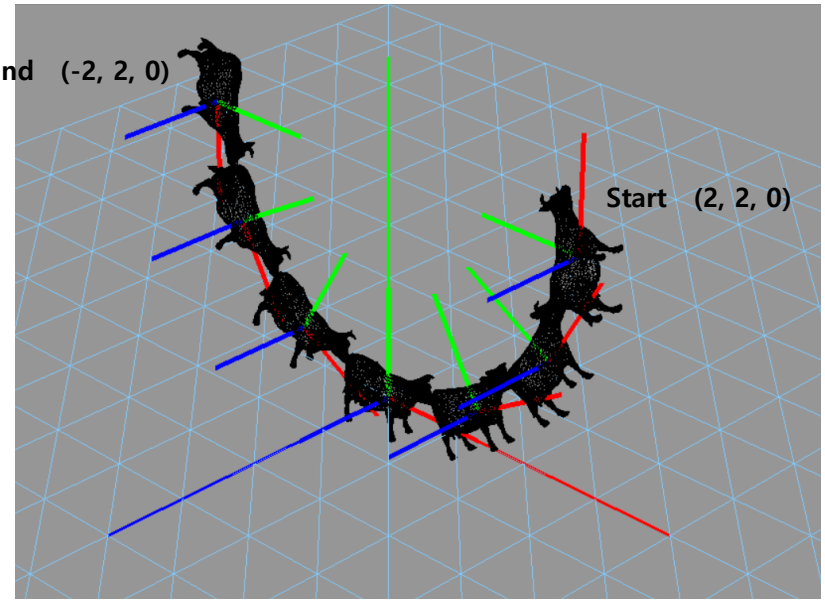
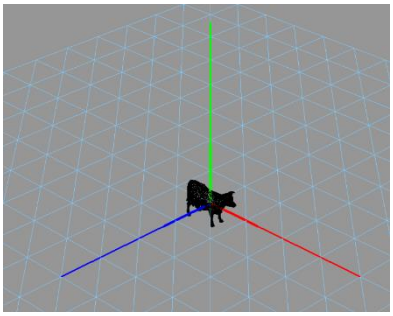


그림 A



원점의 소

(가, 다, 라, 바 는 vec3 type이다.)

```
for (int i = 0; i <= 180; i+=30) {
   float angle = (float)i;
   //Line M
   ModelViewProjectionMatrix = glm::translate(ViewProjectionMatrix, glm::vec3(   가   )); //Line K
   ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix,   나   * TO_RADIAN,
        glm::vec3(   다   ));
   ModelViewProjectionMatrix = glm::translate(ModelViewProjectionMatrix, glm::vec3(   라   ));
   ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix,   마   * TO_RADIAN,
        glm::vec3(   바   ));

   glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix[0][0]);
        glLineWidth(5.0f);
   draw_axes();
   glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix[0][0]);
   glLineWidth(1.0f);
   draw_object(OBJECT_COW, 0.0f, 0.0f, 0.0f); //cow color
}
```

(2) 아래의 그림 B는 그림 A의 소의 모습을 x-z 평면에 반사한 듯한 모습을 도시하고 있다. 문제 (1) 코드의 **Line K**의 ViewProjectionMatrix를 ModelViewProjectionMatrix로 바꾸고 그 윗줄(**Line M 위치**)에 아래의 코드 중 하나를 추가하면 각각 B.1과 B.2의 결과를 얻을 수 있다. 두 코드의 소의 위치는 동일하지만 축의 방향이 서로 다르다. 두 가지 방법 각각의 빈칸을 채우시오.

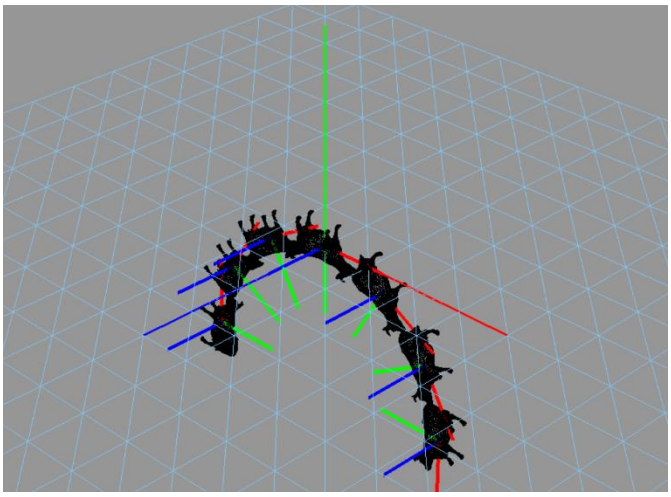


그림 B.1

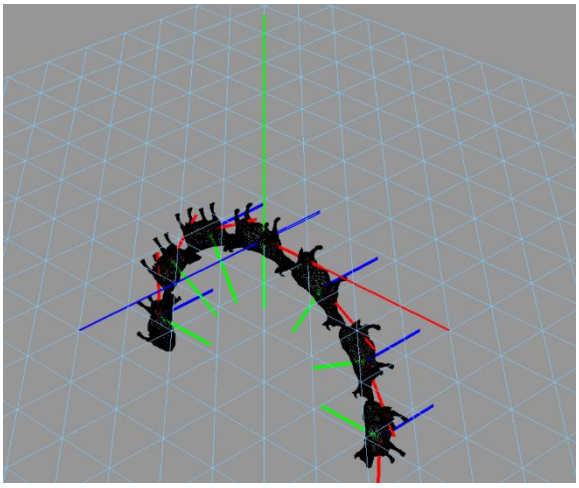


그림 B.2

첫 번째 방법: 반사 => 결과: 그림 B.1

```
ModelViewProjectionMatrix=glm::scale(ViewProjectionMatrix, glm::vec3(   가   ));
```

두 번째 방법: rotate => 결과: 그림 B.2

```
ModelViewProjectionMatrix=glm::rotate(ViewProjectionMatrix,   나   *TO_RADIAN,
    glm::vec3(   다   ));
```

(3) 위의 문제에 대해 다음 함수 호출이 리턴해주는 4행 4열 행렬을 정확히 기술하라.

```
glm::scale(glm::mat4(1.0f), glm::vec3(   가   ))
```

[답안 및 해설]

(1)번 문제 답안

(가) 0.0f, 2.0f,0.0f (나) -angle (다) 0.0f, 0.0f, 1.0f (라) 2.0f, 0.0f, 0.0f (마) 90 (바) 0.0f, 0.0f, 1.0f

(1)번 문제 해설

원점 소를 이용하여 그림 A와 같은 모델링 변환을 만들기 위해서는 아래와 같은 과정이 필요하다.
1) Start의 소가 i=0일 때 가장 먼저 그려지므로 소가 바라보는 방향을 맞추어주기 위하여 원점의 소를 Z축 둘레로 90도 회전한다.
2) 반원을 그리며 회전하기 위해 소를 X축으로 2.0만큼 이동시킨다.
3) i가 30씩 증가함에 따라 소를 Z축 둘레로 -1 * (float) i 만큼 회전시킨다. 이러면 원점을 중심으로 반지름이 2이고 y값이 음수인 반원을 따라 소가 x-y 평면에 그려진다.
4) 그림 A와 같이 만들어주기 위해 3)의 반원을 y축으로 2.0 이동한다.

따라서 T(0.0f, 2.0f, 0.0f) * R(-angle, 0.0f, 0.0f, 1.0f) * T(2.0f, 0.0f, 0.0f) * R(90, 0.0f, 0.0f, 1.0f) 의 모델링 변환을 만들 수 있고 이를 코드로 옮기면 아래와 같다.

```
for (int i = 0; i <= 180; i+=30) {
        float angle = (float)i;
        ModelViewProjectionMatrix = glm::translate(ViewProjectionMatrix, glm::vec3(0.0f,
          2.0f, 0.0f));
        ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, -angle *
          TO_RADIAN, glm::vec3(0.0f, 0.0f, 1.0f));
        ModelViewProjectionMatrix = glm::translate(ModelViewProjectionMatrix,
          glm::vec3(2.0f, 0.0f, 0.0f));
        ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, 90 * TO_RADIAN,
          glm::vec3(0.0f, 0.0f, 1.0f));
        glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE,
          &ModelViewProjectionMatrix[0][0]);
        glLineWidth(5.0f);
        draw_axes();
        glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE,
          &ModelViewProjectionMatrix[0][0]);
        glLineWidth(0.1f);
        draw_object(OBJECT_COW, 0.0f, 0.0f, 0.0f);//Cow color = (0.0, 0.0, 0.0)
}
```

(2)번 문제 답안

(가) 1.0f, -1.0f, 1.0f (나) 180 (다) 1.0f, 0.0f, 0.0f

(2)번 문제 해설

그림 B는 그림 A를 y축을 기준으로 x-z 평면에 대해 반사하거나 x축 둘레로 180도 회전시킨 것이다. 두 방법의 x, y, z 축 방향은 상이하다. 두 가지 방법을 코드로 나타내면 아래와 같다.

(1) 반사는 크기 변환으로 구현할 수 있다. y값이 양수에서 음수로 반사되므로 y축으로 -1배, x와 z축으로 1배 scaling 하여 x-z평면 반사를 할 수 있다.

```
for (int i = 0; i <= 180; i+=30) {
        float angle = (float)i;
        ModelViewProjectionMatrix = glm::scale(ViewProjectionMatrix, glm::vec3(1.0f, -1.0f,
```

```
          1.0f));
        ModelViewProjectionMatrix = glm::translate(ModelViewProjectionMatrix,
          glm::vec3(0.0f, 2.0f, 0.0f));
        ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, -angle *
          TO_RADIAN, glm::vec3(0.0f, 0.0f, 1.0f));
        ModelViewProjectionMatrix = glm::translate(ModelViewProjectionMatrix,
          glm::vec3(2.0f, 0.0f, 0.0f));
        ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, 90 * TO_RADIAN,
          glm::vec3(0.0f, 0.0f, 1.0f));
        glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE,
          &ModelViewProjectionMatrix[0][0]);
        glLineWidth(5.0f);
        draw_axes();
        glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE,
          &ModelViewProjectionMatrix[0][0]);
        glLineWidth(0.1f);
        draw_object(OBJECT_COW, 0.0f, 0.0f, 0.0f);//Cow color = (0.0, 0.0, 0.0)
}
```

(2) X축 둘레로 180도 회전

```
for (int i = 0; i <= 180; i+=30) {
        float angle = (float)i;
        ModelViewProjectionMatrix = glm::rotate(ViewProjectionMatrix, 180*TO_RADIAN,
          glm::vec3(1.0f, 0.0f, 0.0f));
        ModelViewProjectionMatrix = glm::translate(ModelViewProjectionMatrix,
          glm::vec3(0.0f, 2.0f, 0.0f));
        ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, -angle *
          TO_RADIAN, glm::vec3(0.0f, 0.0f, 1.0f));
        ModelViewProjectionMatrix = glm::translate(ModelViewProjectionMatrix,
          glm::vec3(2.0f, 0.0f, 0.0f));
        ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, 90 * TO_RADIAN,
          glm::vec3(0.0f, 0.0f, 1.0f));
        glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE,
          &ModelViewProjectionMatrix[0][0]);
        glLineWidth(5.0f);
        draw_axes();
        glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE,
          &ModelViewProjectionMatrix[0][0]);
        glLineWidth(0.1f);
        draw_object(OBJECT_COW, 0.0f, 0.0f, 0.0f);//Cow color = (0.0, 0.0, 0.0)
}
```

(3)번 문제 답안

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

(3)번 문제 해설

Scaling (크기 변환) 행렬은 아래와 같이 계산할 수 있다.

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

문제의 크기 변환 행렬은 x축으로 1배 (=Sx), y축으로 -1배(=Sy), z축으로 1배(=Sx) 이므로 아래와 같은 크기 변환 행렬을 구할 수 있다.

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