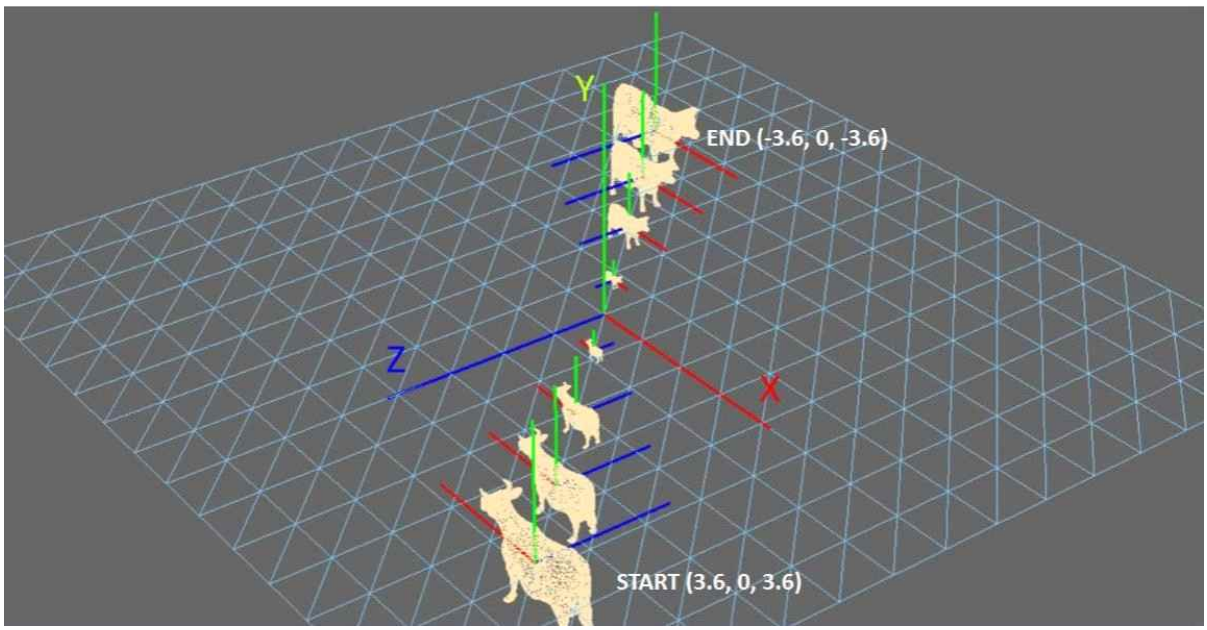
기초 컴퓨터 그래픽스 - HW4

OpenGL API 함수를 사용한 3D 뷰잉 연습

20181618 김하늘

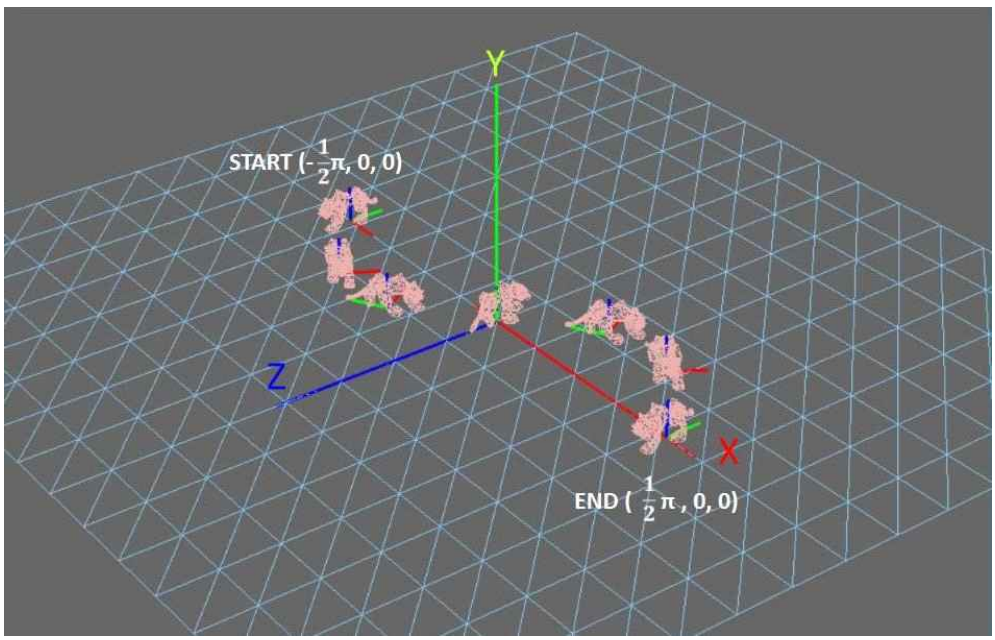
(A)



(A)는 그림과 같이 대각선 경로를 따라 이동하며, 동시에 크기 및 방향도 변화한다.

원점에 가까워질수록 소의 크기는 작아지며, 원점에 다다랐을 때 없어졌다가 방향이 바뀌고 다시 멀어지면서 원래의 크기로 돌아온다. 소의 최대 크기는 원본 모델링의 1.8배이다.

(B)



(B)는 그림과 같이 y = sin(x)꼴의 경로를 따라 움직이며, 호랑이의 머리 방향은 진행하는 방향과 일치한다.

(1) 개념 이해

문제

두 모델링 코드에 대한 공통적인 reshape함수이다.
윈도우의 크기를 바꿀 때 윈도우에 나타나는 화면의 범위 및 모델링의 비율 변화를 설명하시오.

```
void reshape(int width, int height) {
	float aspect_ratio;
	glViewport(0, 0, width, height);

	aspect_ratio = (float)width / height;
	ProjectionMatrix = glm::perspective(15.0f*TO_RADIAN, aspect_ratio, 1.0f, 1000.0f);

	ViewProjectionMatrix = ProjectionMatrix * ViewMatrix;
	glutPostRedisplay();
}
```

답 & 해설

glm::perspective(15.0f*TO_RADIAN, aspect_ratio, 1.0f, 1000.0f) 에서 투영변환을 사용한다는 것을 알 수 있다.
'15.0f*TO_RADIAN'에서 각이 위, 아래 15도로 고정되어있기 때문에 카메라의 시점은 동일하며 최대로 보이는 위, 아래의 시야는 변하지 않는다.
'aspect ratio'는 width/height로 윈도우의 크기가 바뀔 때마다 함께 조정되기 때문에 가로방향의 시야는 바뀔 수 있지만 모델링의 비율은 바뀌지 않는다.

(2) 빈칸 채우기

문제

두 모델링에 공통적으로 사용되는 timer 함수와 소, 호랑이의 변환에 대한 코드이다. 빈 칸에 들어갈 적절한 코드를 rotation_time 변수를 이용하여 작성하시오. 단, ②, ③, ④의 크기는 같으며, 이중 일부는 절댓값을 취해야 한다.

```
void timer_scene(int timestamp_scene) {
	rotation_time = (float)(timestamp_scene % 360);
	glutPostRedisplay();

	glutTimerFunc(100, timer_scene, (timestamp_scene + 2) % INT_MAX);
}
```

```
ModelViewProjectionMatrix = glm::translate(ViewProjectionMatrix, glm::vec3( ① , 0.0f, ① );
ModelViewProjectionMatrix = glm::scale(ModelViewProjectionMatrix, glm::vec3(②, ③, ④);
glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix[0][0]);
glLineWidth(3.0f);
draw_axes();
glLineWidth(1.0f);
draw_object(OBJECT_COW, 255 / 255.0f, 229 / 255.0f, 180 / 255.0f); // Peach
```

```
ModelViewProjectionMatrix = glm::translate(ViewProjectionMatrix,glm::vec3( ⑤ -180* TO_RADIAN, 0.0f, ⑥));
ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, ⑦ ), glm::vec3(0.0f, 1.0f, 0.0f));
ModelViewProjectionMatrix = glm::rotate(ModelViewProjectionMatrix, -90.0f * TO_RADIAN,
        glm::vec3(1.0f, 0.0f, 0.0f));
ModelViewProjectionMatrix = glm::scale(ModelViewProjectionMatrix, glm::vec3(0.005f, 0.005f, 0.005f));
glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix[0][0]);
draw_object(OBJECT_TIGER, 232 / 255.0f, 173 / 255.0f, 170 / 255.0f); // Rose
ModelViewProjectionMatrix = glm::scale(ModelViewProjectionMatrix, glm::vec3(100.0f, 100.0f, 100.0f));
glUniformMatrix4fv(loc_ModelViewProjectionMatrix, 1, GL_FALSE, &ModelViewProjectionMatrix[0][0]);
glLineWidth(3.0f);
draw_axes();
glLineWidth(1.0f);
```

답

① : -rotation_time/50+3.6

② : rotation_time/100 - 1.8

③ : fabs(rotation_time/100 - 1.8)

④ : rotation_time/100 - 1.8

⑤ : (rotation_time * TO_RADIAN)

⑥ : sin(rotation_time * TO_RADIAN)

⑦ : -fabs((rotation_time * TO_RADIAN)+(180 * TO_RADIAN)) + (180.0f * TO_RADIAN)

해설

① : -rotation_time/50+3.6

소는 (3.6, 0, 3.6)에서 (-3.6, 0, -3.6) 까지 이동한다.

rotation_time의 범위는 [0, 359]이므로 -rotation_time/50을 통해 0~-7.2의 범위로 값을 정하고 +3.6을 하면 <3.6 ~ 0 ~ -3.6> 으로 설정할 수 있다.

② : rotation_time/100 - 1.8

③ : fabs(rotation_time/100 - 1.8)

④ : rotation_time/100 - 1.8

소의 크기는 기존 모델링의 1.8배로 출발해서 0이 되었다가 다시 1.8배로 돌아온다.

따라서 ①과 비슷하게 rotation_time/100을 통해 0~3.6의 범위로 값을 정하고 -1.8을 해서 <-1.8 ~ 0 ~ 1.8> 으로 범위를 설정한다.

단, 그림을 보면 원점을 기준으로 회전되어있다. rotate 변환은 없으므로 scale을 통해 방향을 설정해야한다. 그림을 관찰해보면 원점을 지나기 전에는 x, z축이 반대로 설정되어있으므로, y값에만 절댓값을 취해주어야 한다.

⑤ : (rotation_time * TO_RADIAN)

⑥ : sin(rotation_time * TO_RADIAN)

rotation의 범위는 0~359이므로 그림 (B)와 같이 이동하기 sin함수 꼴로 위해선 translate 변환은 위와 같이 설정하여야한다.

각도는 0도부터 시작하므로 x값에 180*TO_RADIAN을 빼서 원점에 맞춰주었다.

⑦ : -fabs((rotation_time * TO_RADIAN)+(180 * TO_RADIAN)) + (180.0f * TO_RADIAN)
시작위치의 각도는 x축을 중심으로 -90도로 회전되어 있는 상태이다.
그 다음 y축을 중심으로 회전하는데, 진행 방향과 일치해야 하므로 rotation_time이 0~180일때는 +방향으로 회전해야하고, 181~359까지는 다시 -방향으로 회전해야한다.
즉 <0 ~ 180 ~ 0> 이어야 하기때문에 -fabs((rotation_time * TO_RADIAN)+(180 * TO_RADIAN)) 에서 -180 ~ 0 ~ 180으로 맞추고 180*TO_RADIAN을 더해준다.

(3) 행렬

문제

(2) 에서 rotation_time = 90일 때, (a), (b) 변환의 결과로 반환되는 행렬을 각각 작성하시오.

(a) ModelViewProjectionMatrix = glm::scale(ModelViewProjectionMatrix, glm::vec3(②, ③, ④);

(b) ModelViewProjectionMatrix = glm::translate(ViewProjectionMatrix,glm::vec3(⑤-180*TO_RADIAN, 0.0f, ⑥));

답

(a) $\begin{bmatrix} -0.9 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 \\ 0 & 0 & -0.9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

(b) $\begin{bmatrix} 1 & 0 & 0 & -\frac{1}{2}\pi \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

해설

(a) ModelViewProjectionMatrix = glm::scale(ModelViewProjectionMatrix,
glm::vec3(rotation_time/100 - 1.8, fabs(rotation_time/100 - 1.8), rotation_time/100 - 1.8);
rotation_time = 90일 때 S(-0.9, 0.9, -0.9)이므로 반환되는 행렬은 다음과 같다.

$\begin{bmatrix} -0.9 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 \\ 0 & 0 & -0.9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

(b) ModelViewProjectionMatrix = glm::translate(ViewProjectionMatrix,
glm::vec3((rotation_time * TO_RADIAN)-180*TO_RADIAN, 0.0f, sin(rotation_time * TO_RADIAN)));
rotation_time = 90일 때 T(-$\frac{1}{2}\pi$, 0, sin($\frac{1}{2}\pi$)) = T(-$\frac{1}{2}\pi$, 0, 1) 이므로 반환되는 행렬은 다음과 같다.

$\begin{bmatrix} 1 & 0 & 0 & -\frac{1}{2}\pi \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$